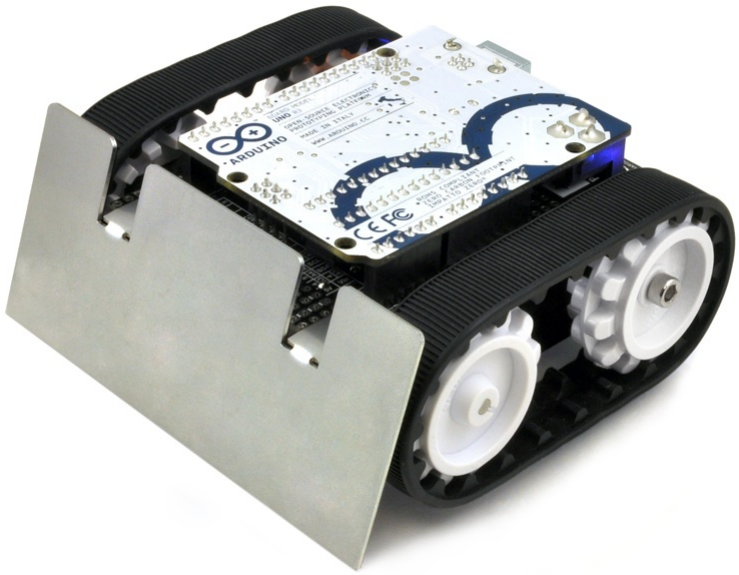


Guide utilisateur du Robot Zumo Pololu

Un guide complet et abordable pour assembler, utiliser et exploiter rapidement votre Robot Zumo.

(version 0.1)

Traduit par MicroControleur Hobby (shop.mchobby.be)
Compilé depuis la traduction maintenue sur https://wiki.mchobby.be/index.php?title=Pololu-Zumo-Shield-Arduino
Les hyperliens sont disponibles sur la version en ligne du document.
Translated by MicroControleur Hobby (shop.mchobby.be)
Compiled from online translation available at https://wiki.mchobby.be/index.php?title=Pololu-Zumo-Shield-Arduino
Hyperlinks are available on the online version of this document.

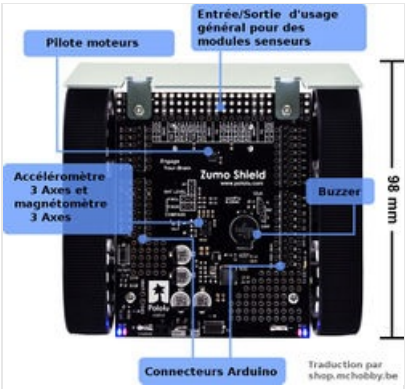
Aperçu

Le Zumo Shield offre une interface pratique entre le châssis Zumo http://shop.mchobby.be/product.php?id_product=447 lien pololu https://www.pololu.com/product/1418 et un A-Star 32U4 Prime lien pololu https://www.pololu.com/category/165/a-star-32u4-prime , Arduino Uno http://shop.mchobby.be/product.php?id_product=10 lien pololu https://www.pololu.com/product/2191 ou un Arduino Leonardo http://shop.mchobby.be/product.php?id_product=96 lien pololu https://www.pololu.com/product/2192 (il n'est pas compatible avec un Arduino Mega ou Due mais il peut être utilisé avec de plus vieux Arduino qui disposent du même Form Factor qu'un Arduino Uno (comme le Duemilanove par exemple). Le shield est monté directement sur le châssis, connecté directement sur les moteurs, les connecteurs du bloc pile et l'Arduino vient se connecter sur les connecteurs mâles du shield Zumo (face vers le bas). Le shield offre toute l'électronique nécessaire pour alimenter et commander les moteurs ainsi que quelques composants additionnels permettant de réaliser un robot plus intéressant (comme un buzzer pour faire du son, un senseur inertiel incluant un accéléromètre et un gyroscope).

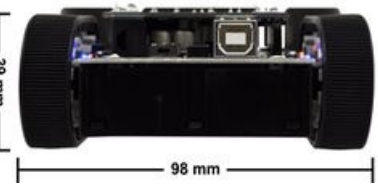
Shied Zumo pour Arduino, v1.2, tel que livré (avec composant CMS déjà assemblé).
Cliquer l'image pour l'agrandir

Robot Zumo Arduino assemblé avec un Arduino Uno (et roues blanche d'origine).
Cliquer l'image pour l'agrandir

Un châssis Zumo, Shield Zumo et Arduino (ou carte compatible) peuvent être combinés pour réaliser un robot traqueur bas-profile, contrôlé par Arduino. Il fait moins de 10 cm de chaque côté (assez petit pour être qualifié dans les compétitions Mini-Sumo).



Fonctionnalités principales du Shield Zumo pour Arduino, v1.2.
Cliquer l'image pour l'agrandir



Robot Zumo assemblé avec un Shield Zumo et Arduino Uno, vue arrière.
Cliquer l'image pour l'agrandir

La dernière révision du Shield Zumo est la version **1.2**. Cette version ajoute un gyroscope 3 axes L3GD20H lien pololu https://www.pololu.com/product/2129 et fait un upgrade du composant accéléromètre et magnétomètre vers le nouveau LSM303D lien pololu https://www.pololu.com/product/2127 . Il est disponible sous forme de kit ou sous forme d'un robot complètement assemblé:

- Shield Zumo, v1.2 lien pololu https://www.pololu.com/product/2508
- Kit robot Zumo, v1.2 lien pololu https://www.pololu.com/product/2509 avec un châssis Zumo http://shop.mchobby.be/product.php?id_product=447 lien pololu https://www.pololu.com/product/1418 et la lame Zumo en inox lien pololu https://www.pololu.com/product/1410
- Robot Zumo pour Arduino, v1.2 http://shop.mchobby.be/product.php?id_product=448 lien pololu https://www.pololu.com/product/2510 , complètement assemblé avec des moteurs 75:1 HP http://shop.mchobby.be/product.php?id_product=751 lien pololu https://www.pololu.com/product/2361 et un réseau de capteur infrarouge (suiveur de ligne) lien pololu https://www.pololu.com/product/1419 déjà monté.

Les information dans ce guide utilisateur conviennent également pour le Zumo Shield original qui lui, n'inclus pas de gyroscope et le accéléromètre+magnétomètre LSM303DLHC lien pololu https://www.pololu.com/product/2124 :

- Shield Zumo lien pololu https://www.pololu.com/product/2504
- Kit Zumo Robot pour Arduino lien pololu https://www.pololu.com/product/2505
- Zumo robot pour Arduino http://shop.mchobby.be/product.php?id_product=448 lien pololu https://www.pololu.com/product/2506

A Propos de MCHobby
Crédit de traduction

Sommaire

A propose de .: MC Hobby :.

MCHobby investi du temps et de l'argent dans la réalisation de traduction et/ou documentation. C'est un travail long et fastidieux réalisé dans l'esprit Open-Source... donc gratuit et librement accessible.
SI vous aimez nos traductions et documentations ALORS aidez nous à en produire plus en achetant vos produits chez MCHobby http://shop.mchobby.be .

MC Hobby SPRL, basé en Belgique, est le traducteur de ce manuel.

MC Hobby cherche à promouvoir, en français, la plateforme Open-Source Arduino, ses extensions et exemples de programmation afin de la rendre accessible au plus grand nombre.

License

Crédit de traduction

Toute référence, mention ou extrait de cette traduction doit être explicitement accompagné du texte de crédit de traduction.

Ce texte est repris en "pied de document" sur toutes les pages/documents, merci d'en prendre connaissance.

Limite de traduction

Anglicisme

Du fait que de nombreux anglicismes soient utilisés au jour le jour dans les milieux techniques, je me suis permis d'en préserver quelques-uns qui me semblaient avoir plus de sens dans la langue de Shakespeare que traduit dans la langue de Voltaire. Par ailleurs, ces anglicismes pourraient se révéler fort utiles lors de vos prochaines recherches sur Internet.

Ainsi, vous retrouverez les termes suivants :

- Pin : fait référence à une « broche » de raccordement d'un composant (Anglicisme fort répandu).
- Datasheet : fait référence à la « fiche technique » d'un composant.

- Pin Header : malheureusement intraduisible sans en perdre le sens mais vous aurez vite identifié les « Pinheaders ». La traduction courant est *connecteur*, ce terme restant beaucoup trop vague.

• LED : ce sont les diodes électroluminescentes (aussi appelées DEL). Cependant, le terme LED est si répandu qu'il a été préservé. • RGB : Fait référence aux 3 couleurs fondamentales Rouge Vert Bleu (Red Green Blue). Si le terme RVB est d'usage fréquent en Belgique et en France, l'acronyme RGB reste encore le plus répandu.

Code source

Nous avons traduit les commentaires dans les programmes afin de les rendre plus intelligibles.

Par contre, nous n'avons pas modifié les codes sources (nom des variables, ...) qui, eux, restent ceux fournit par le concepteur et le fabricant du kit.

Autorisations

Le présent manuel a été traduit avec l'autorisation de Pololu (www.pololu.com).

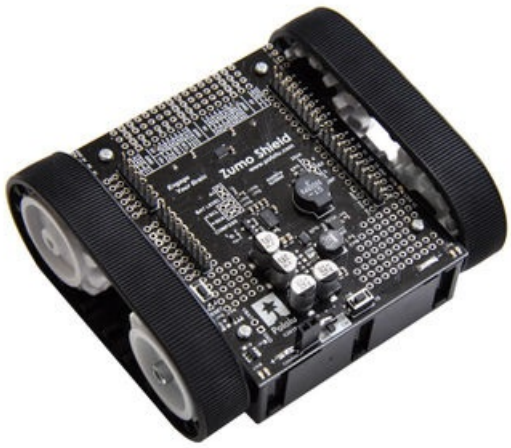
Signaler une erreur

Malgré tout le soin apporté à la réalisation de cette traduction, il n'est pas impossible qu'une erreur se soit malgré tout glissée dans ce document. N'hésitez pas à nous envoyez un e-mail si vous en constatiez une. Pour adresser vos remarques et commentaires relatifs à la traduction, ou pour demander une traduction complémentaire, contactez nous par e-mail à

- support (arobase) mchobby.be.

N'oubliez pas de mentionner le manuel/page/lien en mentionnant l'erreur ;-)

Contacter Pololu



Fully assembled Zumo chassis with assembled Zumo Shield (v1.0).

Cliquer l'image pour l'agrandir

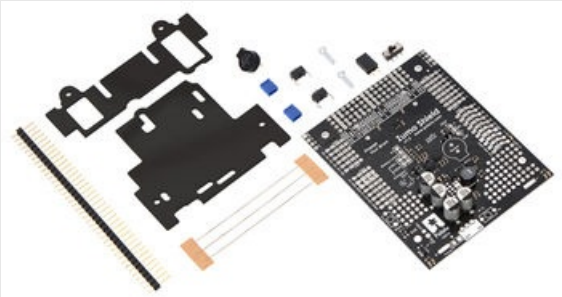
Pololu serait ravit d'en apprendre plus sur vos expériences avec le Shield Zumo pour Arduino lien pololu https://www.pololu.com/product/2508 , le kit Robot Zumo pour Arduino lien pololu https://www.pololu.com/product/2509 ou le robot Zumo pour Arduino http://shop.mchobby.be/product.php?id_product=448 lien pololu https://www.pololu.com/product/2510 . Si vous avez besoin de support technique ou envie de partager un retour d'expérience, vous pouvez contacter directement Pololu https://www.pololu.com/contact (*anglais*) ou ouvrir un billet sur le forum de pololu http://forum.pololu.com/viewforum.php?f=29 (*anglais*). N'hésitez pas à dire ce qui va bien, ce qui pourrait être amélioré, ce que vous voudriez dans le futur ou tout ce que vous désireriez partager à propos de la plateforme!

Composants inclus

Le shield Zumo est disponible:

- comme shield lien pololu https://www.pololu.com/product/2508 ;
- comme partie du Kit Robot Zumo pour Arduino lien pololu https://www.pololu.com/product/2509 qui inclus également un Châssis Zumo http://shop.mchobby.be/product.php?id_product=447 lien pololu https://www.pololu.com/product/1418 et la lame Zumo en inox lien pololu https://www.pololu.com/product/1410 ; ou
- Comme Robot Zumo pour Arduino http://shop.mchobby.be/product.php?id_product=448 lien pololu https://www.pololu.com/product/2510 **complètement assemblé** avec des moteurs 75:1 HP lien pololu https://www.pololu.com/product/2361 et un réseau de capteur infrarouge (suiveur de ligne) lien pololu https://www.pololu.com/product/1419 .

Le shield Zumo



Shield Zumo pour Arduino, v1.2, présenté avec le matériel inclus

Cliquer l'image pour l'agrandir

Le Shield -seul- est livré avec les composants suivants:

- Un interrupteur a glissière (à angle droit)
- Deux boutons poussoirs lien pololu https://www.pololu.com/product/1400
- Un buzzer
- connecteur 2-broches pour la recharge des piles http://shop.mchobby.be/product.php?id_product=610 lien pololu https://www.pololu.com/product/1012
- Trois morceaux de fils (pour souder les moteurs sur le shield)
- Deux sections de connecteurs 25-broches empattement de 2.54mm connecteurs mâles/pinHeader http://shop.mchobby.be/product.php?id_product=76 lien pololu https://www.pololu.com/product/965
- Cavaliers bleu lien pololu https://www.pololu.com/product/968
- Des visses 5/16" #2-56 (à utiliser à la place des visses 1/4" inclues avec le kit du châssis included, utilisées si vous attachez la lame Zumo)
- Entretoise de 1/16" en acrylique noir (deux pièces)

Le kit Robot Zumo Arduino

Kit Châssis Zumo de Pololu (composants inclus).

Cliquer l'image pour l'agrandir

An plus du shield et du matériel déjà inclus, le kit Zumo robot pour Arduino inclus également ces composants:

- Zumo châssis kit http://shop.mchobby.be/product.php?id_product=447 lien pololu https://www.pololu.com/product/1418 qui inclus:
  - Corps du Zumo châssis
  - Plaque de montage 1/16" en acrylique noir (non utilisé avec le shield Zumo)
  - Deux roues dentées d'entraînement
  - Deux roues dentées en roue libre
  - Deux chenilles 22-tooth silicone tracks
  - Two shoulder bolts with washers and M3 nuts
  - Quatres vis 1/4" #2-56 + écrous
  - Contacts du bloc pile
- Lame sumo pour châssis Zumo lien pololu https://www.pololu.com/product/1410

Lame sumo (modèle basique) pour le châssis Zumo.
Cliquer l'image pour l'agrandir

> Vous recevrez une entretoise en acrylique et plaques de montage avec un papier de protection masquant les deux côtés. Vous pouvez pelez cette protection pour exposer la surface acrylique ou laisser le films de protection en place pour accroître l'épaisseur de la plaque.

> Le shield et le kit châssis inclus des éléments complémentaires comme des fils, vis, écrous et entretoise. Par conséquent, ne vous inquiétez pas si vous avez encore des éléments en trop après avoir assemblé votre Zumo.

Robot Zumo pour Arduino

Robot Zumo pour Arduino, v1.2.
Cliquer l'image pour l'agrandir

Le robot Zumo pour Arduino en une plateforme robot complètement assemblée à partir des composants du Kit Robot Zumo, avec ces quelques éléments complémentaires:

- Deux micro moteurs 75:1 HP, engrenage métal http://shop.mchobby.be/product.php?id_product=431 lien pololu https://www.pololu.com/product/2361
- Un réseau de capteur infrarouge Zumo (suiveur de ligne) lien pololu https://www.pololu.com/product/1419

Assembler

Cette partie du tutoriel est composée des sections suivantes:

- Matériel nécessaire
- Assembler le Shield et le châssis
- Ajouter le détecteur de ligne Zumo (optionel)

Si vous avez un Kit Zumo pour Arduino lien pololu https://www.pololu.com/product/2509 --OU-- un shield Zumo lien pololu https://www.pololu.com/product/2508 plus un châssis Zumo http://shop.mchobby.be/product.php?id_product=447 lien pololu https://www.pololu.com/product/1418 alors cette section du guide vous guidera au travers des différentes étapes d'assemblage du robot.

Si vous avez acheté un Robot Zumo pour Arduino assemblé http://shop.mchobby.be/product.php?id_product=448 lien pololu https://www.pololu.com/product/2510 , les différentes étapes d'assemblage ont déjà été faites pour vous. Vous pourriez cependant avoir envie de configurer votre Zumo en ajoutant ou retirant des cavaliers de configuration. Sinon, vous pouvez simplement installer 4 piles AA et un Arduino (ou contrôleur compatible) et ignorer la Section 3 d'assemblage pour vous consacrer à l'utilisation du Zumo!

Matériel nécessaire

Le shield Zumo est conçu pour être monté sur le kit Zumo châssis http://shop.mchobby.be/product.php?id_product=447 lien pololu https://www.pololu.com/product/1418 qui est inclus, avec la lame Zumo lien pololu https://www.pololu.com/product/1410 , dans le Kit robot Zumo pour Arduino lien pololu https://www.pololu.com/product/2509 .

Vous aurez également besoin des éléments suivants pour faire fonctionner votre Robot Zumo avec un Arduino:

Composants additionnels nécessaires

- Deux micro moteurs à engrenage métal http://shop.mchobby.be/recherche?controller=search&orderby=position&orderway=desc&search_query=micro-mot&submit_search=Rechercher lien Pololu https://www.pololu.com/category/60/micro-metal-gearmotors (nous recommandons un rapport 100:1, 75:1 ou 50:1 avec moteurs HP https://www.pololu.com/category/173/6v-high-power-hp-micro-metal-gearmotors haute-puissances OU HPCB https://www.pololu.com/category/174/6v-high-power-carbon-brush-hpcb-micro-metal-gearmotors *Pololu.com, anglais*). La version pre-assemblée du robot Zumo http://shop.mchobby.be/product.php?id_product=448 lien pololu https://www.pololu.com/product/2510 inclus deux micros moteurs engrenage métal 75:1 HP http://shop.mchobby.be/product.php?id_product=431 lien pololu https://www.pololu.com/product/2361 .
- Un Arduino ou carte compatible (nous recommandons un A-Star 32U4 Primelien pololu https://www.pololu.com/product/165 , Arduino Uno R3 http://shop.mchobby.be/product.php?id_product=10 lien pololu https://www.pololu.com/product/2191 ou Arduino Leonardo http://shop.mchobby.be/product.php?id_product=96 lien pololu https://www.pololu.com/product/2192)
- Quatre piles AA (nous recommandons des piles rechargeables AA NiMH lien pololu https://www.pololu.com/product/1003)

Vous pouvez prendre connaissance de la description de la fiche produit du kit châssis http://shop.mchobby.be/product.php?id_product=447 pour plus d'informations et recommandations sur la sélection des composants.

Composants optionnels

- Un réseau de capteur infrarouge Zumo (suiveur de ligne) lien pololu https://www.pololu.com/product/1419
- Lame sumo (modèle basique) pour le châssis Zumo lien pololu https://www.pololu.com/product/1410
- Senseurs http://shop.mchobby.be/category.php?id_category=9 lien Pololu https://www.pololu.com/category/7/sensors , tels que les senseurs infrarouge QTR de pololu http://shop.mchobby.be/recherche?controller=search&orderby=position&orderway=desc&search_query=qtr&submit_search=Rechercher lien Pololu https://www.pololu.com/category/123/pololu-qtr-reflectance-sensors </small>
- Connecteurs http://shop.mchobby.be/category.php?id_category=42 lien Pololu https://www.pololu.com/category/19/connectors et jumper fils de prototypages http://shop.mchobby.be/category.php?id_category=78 pour connecter les senseurs et composants
- Un chargeur d'accu (comme le iMAX-B6AC lien pololu https://www.pololu.com/product/2588) si vous utilisez des accu rechargeables

Outillage

Voici une suggestion d'outils si vous réalisez l'assemblage d'un kit:

- Un fer à souder et de la soudure (nous recommandons l'utilisation d'un fer avec température réglable)
- Un dénudeur
- Petites tournevis à croix
- Clé 3 mm Allen (clé hexagonale)
- Une pince à bec (pour plier la languettes de fixations de la lame Zumo)

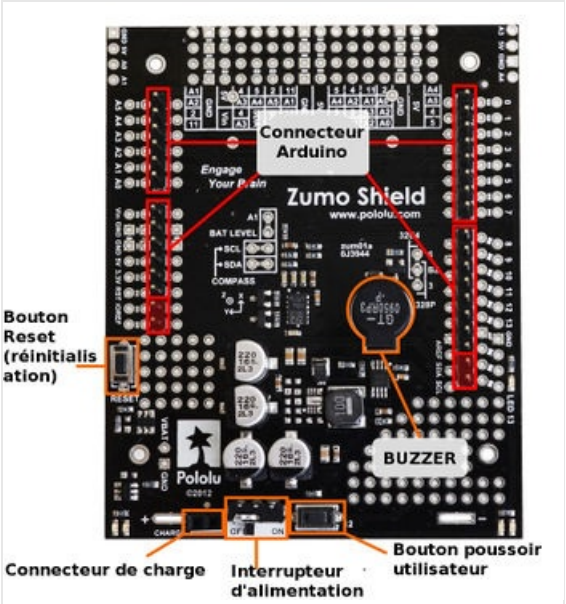
Assembler le Shield et le châssis

Sommaire

Introduction

Suivez soigneusement ces instructions pour assembler correctement le Shield Zumo et le châssis. (Ces images présente le shield Zumo original lien pololu https://www.pololu.com/product/2504 mais les étapes d'assemblage restent identique pour la dernière version v1.2 lien pololu https://www.pololu.com/product/2508)

Les composants traversants



Zumo Shield avec connecteur Arduino et composant tranversant

Cliquer l'image pour l'agrandir

1. Soudez les composants traversant sur le shield:
    1. Interrupteur d'alimentation (dit *power* en anglais)
    2. Le bouton poussoir "reset" (réinitialisation de la carte Arduino)
    3. Le bouton utilisateur (dit *user* en anglais)
    4. Le buzzer
    5. Le connecteur de charge (connecteur femelle 1×2-broches)
2. En dessous de la carte, coupez les broches qui dépassent de plus 1.5mm (l'épaisseur de la plaque entretoise) afin que les broches n'empêche pas le shield de prendre sa position BIEN A PLAT sur le châssis et la plaque entretoise.

Connecteurs Arduino

Brisez le connecteur mâle/pinHeader 1×40-broches en segments pour connecter votre Arduino et les souders sur le shield. Placez ces connecteurs sur votre carte Arduino puis venez placer le shield par dessus. la partie courte des connecteurs doivent dépasser par les trous du shield.

Faites bien attention à positionner le shield et l'Arduino comme indiqué sur l'image.

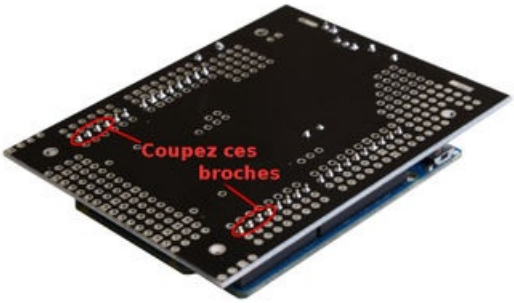
L' A-Star 32U4 Primes et le nouvel Arduino (Arduino R3 et Leonardo) utilisent un connecteur 1×10 broches, deux connecteur 1×8 broches et un connecteur 1×6 broches; les cartes Arduino plus anciennes utilisent deux connecteur 1×8 et deux connecteur 1×6 (les deux paires de broches mises en surbrillance ci-dessous ne doivent pas être peuplés si vous utilisez une ancienne carte Arduino qui ne supportait par encore les nouvelles broches de la révision R3). Faites attention à bien souder le nombre de broches correspondant à votre révision d'Arduino!

Une façon simple et efficace d'aligner les connecteur sur le shield est d'utiliser votre Arduino. Vous y enfoncez la partie longue des pinHeader/connecteur mâle dans les connecteurs de votre Arduino, puis vous placez le shield (face vers le bas) comme montré sur l'image. Faites attention à bien insérer les têtes des broches dans les bons trous du shield avant de commencer à souder. **Note:** si vous utilisez cette technique, assurez vous que votre fer à souder ne soit pas trop chaud (de façon excessive) et évitez de maintenir le fer en contact avec la broches plus de quelques secondes.... sinon vous risquez de fondre le connecteur femelle de votre Arduino.

Le Zumo Shield déposé sur les connecteurs Arduino avant la soudure

Cliquer l'image pour l'agrandir

En dessous de la carte: coupez les connecteurs Arduino au ras de la carte (aussi proche que possible du shield) de sorte qu'elles ne rentrent pas en contact avec la carcasse du moteur. Si vous craignez que la broche rentre malgré tout en contact avec la carcasse des moteurs, vous pouvez utilisez de la toile isolante comme isolant.



Le Zumo Shield soudé et connecteurs Arduino coupé

Cliquer l'image pour l'agrandir

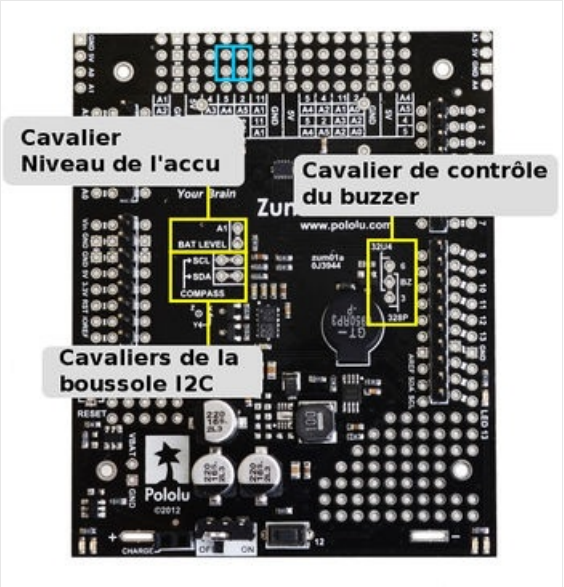
Cavaliers et connexions complémentaires

- **Optionnel:** SI vous désirez activer le buzzer, activer l'entrée du niveau de charge des piles, ou désactiver la boussole ALORS c'est le bon moment pour ajouter et/ou couper des connexions "cavaliers" pour configurer le shield comme vous le souhaitez. Cela peut également être réalisé plus tard mais souder les broches sera plus difficile une fois le robot assemblé (plus spécialement si vous décidez -plus tard- d'ajouter les connecteurs/pinHeader pour les utiliser avec les cavaliers; cela nécessitera une longue phase de désassemblage). L'utilisation des cavaliers est détaillé dans la section 3.c. Les cavaliers du buzzer et du niveau des piles peuvent être connectés en soudant un morceau de fil entre les deux trous, tandis que les connexions I2C de la boussole peuvent être rompus en coupant les traces sur le dessus de la carte (entre les trous).

Note: Si vous utilisez un Arduino through-hole (format DIP), vous devez savoir qu'il n'y aura pas assez de place pour utiliser un connecteur/pinHeader mâle pour le niveau des piles et les connecteurs I2C.

> Plutôt que faire une connection avec un fils, vous pouvez souder un connecteur/pinHeader 1×3 mâle dans les trous de configuration du buzzer afin de permettre l'utilisation d'un cavalier pour connecter le buzzer. Vous pouvez également utiliser des connecteurs mâles et cavaliers pour les cavaliers du *niveau des piles* et de la *bousolle* si vous avez un Arduino Uno avec des composants montés en surface

(CMS), Arduino Leonardo ou A-Star 32U4 Prime. Cependant, il n'y a pas assez de place pour utiliser un connecteur mâle sur le niveau des piles et boussole I2C si vous utilisez un Arduino through-hole/DIP.



Le Zumo Shield avec identification des cavaliers
Cliquer l'image pour l'agrandir

- **Optionnel:** A ce point, vous devriez considérer les opérations de soudures/raccordement des composants additionnels (comme les senseurs), connecteurs, ou fils pour les connecter sur le shield. Si vous le faites, vérifiez que le placement de vos éléments n'interfère pas avec la possibilité de placer correctement le shield sur le châssis ou l'Arduino sur le shield. Plus particulièrement, notez que seules les 3 premières lignes à l'avant de la carte peuvent s'étendre sous la carte (de même, seules les 4 lignes d'extension à l'avant du shield peuvent être utilisées pour réaliser des extensions au dessus de la carte) -ET- si vous soudez des composants traversants sur les zones de prototypage du shield, il faudra forer les trous correspondants dans la plaque entretoise pour que les pattes des composants puissent facilement y prendre place).

Les moteurs

- Coupez les deux fils inclus dans le kit, au milieu, pour réaliser 4 segments. Eliminez la protection adhésive à chaque bout des fils (l'adhésif peut interférer avec le bon contact électrique sur les moteurs). Ces segments de fils seront utilisés pour "étendre" les pattes des moteurs.
- Soudez une paire de fils sur chaque moteur. Courber une pointe de chaque fils pour l'accrocher plus facilement dans le trou de connexion du moteur (afin de faciliter les opérations de soudures). **Attention:** maintenir la pointe du fer plus de quelques secondes sur une broche du moteur commencera à endommager le balais du moteur. Par conséquent, essayez d'être raisonnablement rapide avec les opérations de soudure; si la première tentative ne marche pas bien, alors retirez le fer et laissez le moteur refroidir avant de refaire une nouvelle tentative.

Préparer les connexions du moteur pour les souder sur le shield Zumo
Cliquer l'image pour l'agrandir

La broche positive de chaque de chaque moteur est indiqué à l'aide d'un signe (+) sur le plastique noir du moteur, visible en bas de l'image ci-dessus. Les moteurs doivent être soudés sur le shield avec la broche positive proche de l'avant. Vous devriez donc raccorder les fils sur les moteurs pour qu'ils puissent être orientés de cette façon. Cependant, ne vous inquiétez pas si vous avez branchez un (ou deux) moteurs à l'envers, il est possible de compenser cette erreur de façon logiciel avec la bibliothèque ZumoMotors.)

Identification des broches positives des moteurs
Cliquer l'image pour l'agrandir

- Placez les moteurs dans la rainure à l'avant du châssis, alignez les boîtes de vitesse avec les cannelures. La plaque avant de la boîte de vitesse doit arriver à ras de du châssis.

Châssis et shield

> Pour assembler le châssis et le shield Zumo, vous devrez utiliser la plaque entretoise en acrylique (deux-pièces) qui est inclus avec le shield. Vous n'aurez pas besoin de la plaque de montage (en une piece) incluse avec le châssis Zumo.

- Placez un écrou M3 dans chacun des emplacements prévu à cet effet à l'arrière du châssis. Les emplacements sont dimensionnés pour que les écrous ne puissent pas tourner à l'intérieur. Plus tards, ces écrous seront utilisés pour monter les roues crantées libres.
- Si vous le voulez, peler le films de protection masquant les deux faces des éléments de l'entretoise acrylique (l'entretoise sur notre image est ce à quoi elle doit ressembler une fois le film de protection enlevé). Vous pouvez également laisser les films de protection en vue d'obtenir une épaisseur supplémentaire. Si vous laissez les films de protections, ces derniers seront presque invisibles une fois le robot totalement assemblé.
- Couvrez les le châssis et les moteurs avec les plaques entretoises puis pas le shield Zumo. Les trous dans la plaque entretoise doivent s'aligner avec les trous du shield qui sera déposé au dessus. Les broches des moteurs sont également placés de sorte qu'ils passent part les ouvertures dans les plaques entretoises (comme présenté dans sur l'image ci-dessous). Il y a une seule orientation possible pour ces plaques. L'entretoise est constituée de deux pièces séparées pour permettre permettre le désassemblage du Zumo sans dessouder les moteurs et les connecteur des piles.)


Entretoise en plexiglas positionné au dessus du châssis Zumo
Cliquer l'image pour l'agrandir

- Dans chacun des 4 trous de montage, insérez une vis #2-56 à travers le shield, l'entretoise et le chassis (voir image ci-dessous) et serrez le tout à l'aide d'un écrou sous le châssis. Il est généralement plus facile de maintenir l'écrou en

dessous et de le maintenir avec un doigt puis insérez la vis que vous visserez plus facilement. Notez que le kit inclus deux tailles différentes de vis #2-56: 1/4" (6.35mm) et 5/16″ (8mm). Les vis les plus longues sont destinées à être utilisés dans les trous avants (près des moteurs) si vous allez monter la lame sumo; sinon, vous pouvez utiliser les vis plus courtes (6.35mm) dans tous les trous de montage.

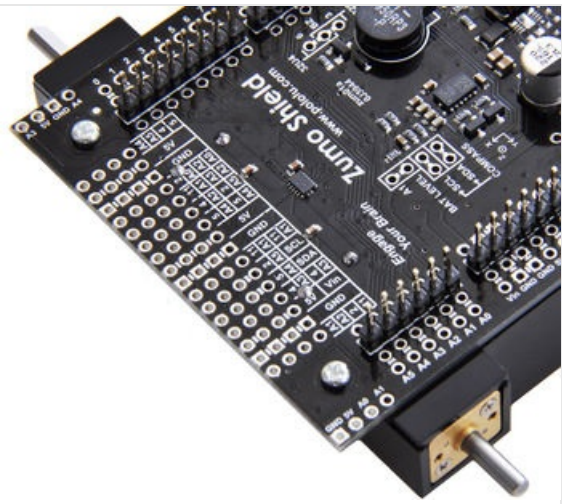
Si vous ajoutez la lame sumo (modèle basique), vous pouvez soit la monter maintenant ou l'ajouter plus tard après avoir fait les soudures des moteurs et des contacts des piles. (**Note:** Si vous comptez souder quoi que ce soit sur la zone d'extension à l'avant du shield zumo, comme le réseau de capteur infrarouge (suiveur de ligne), vous aurez plus de place pour travailler si vous le faites avant de placer la lame sumo.)

> Etant donné qu'il y a une possibilité que les pattes de montages de la lame sumo crée des court-circuits si le masque de soudure de la carte n'est pas assez épais. Nous recommandons d'ajouter un isolant électrique (toile isolante) entre la lame et le shield.

Pour installer la lame: commencez par plier les pattes de montages à l'angle approprié. Ensuite, placez les au dessus du shield de sorte que les trous s'alignent avec les trous à l'avant du châssis et insérez les deux vis #2-56 de 8mm (5/16") de long (incluses dans le shield). Les vis doivent traverser la lame, le shield, l'entretoise et le châssis. Faites attention en ajustant l'angle de la lame sumo alors qu'elle est montée sur le châssis, cette opération peut casser la plaque entretoise acrylique si vous appliquez une force excessive ou soudaine. **Nous recommandons de ne pas ajuster la lame lorsque cette dernière est montée sur le châssis.**

Shield placé au dessus du châssis Zumo
Cliquer l'image pour l'agrandir

- Soudez les fils de chaque moteur sur le shield, puis raccourcissez la longueur des fils.

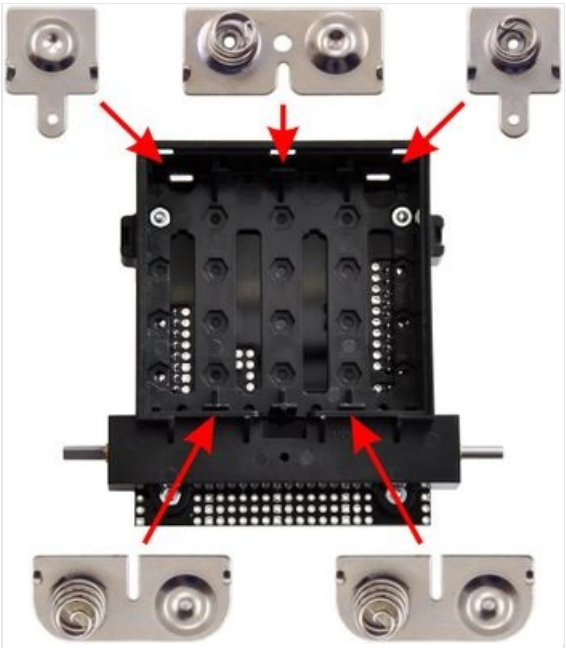


Moteurs soudés sur le Zumo Shield
Cliquer l'image pour l'agrandir

Contacts des piles

- Retournez le châssis et installez les connecteurs de piles comme présenté sur l'image ci-dessous. Les 3 doubles contacts doivent être fermement pressés dans leur emplacement jusqu'à ce qu'ils s'introduisent à l'intérieur de leur emplacement du bloc pile. Les deux contacts individuels doivent être insérés dans le compartiments du bloc pile de sorte

que leur excroissance (patte de contact pour le shield) passe à travers le trou du châssis; Vous pourriez avoir besoin d'utiliser un morceau de papier collant pour les maintenir temporairement en place jusqu'à ce que ces contacts soient soudés sur le shield.



Ajouter les connecteurs des piles sur le châssis et les piles

Cliquer l'image pour l'agrandir

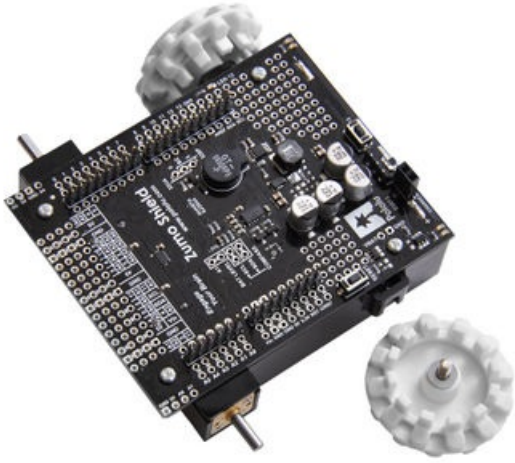
- Retournez à nouveau le châssis et soudez les pattes des contacts du bloc pile (point précédent) sur le shield. Note: Si vous utilisez une pile pour maintenir le contact en place alors cette pile agira comme un dissipateur de chaleur, ce qui rendre l'opération de soudure plus difficile (ou nécessitera une température plus élevée pour le fer à souder). Le trou par lequel la contact de batterie est soudée doit être complètement remplit de soudure comme présenté sur l'image ci-dessous.

Les connecteurs des piles soudés sur le shield

Cliquer l'image pour l'agrandir

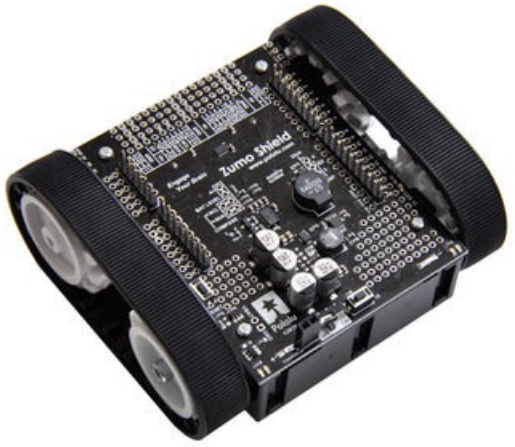
Roues dentées et les chenilles

- Placez les roues dentées libres (à pignon visser) sur chacune des vis à l'arrière du du Zumo. Placez une rondelle entre la roue et le chassis. Les "dents" de le roue dentée doivent être dirigées vers le chassis comme indiqué sur l'image.
- Insérez la vis du pignon dans l'écrou que nous avons placé dans le châssis (plus tôt dans les étapes d'assemblage). Utilisez ensuite une clé hexagonale de 3 mm (clé Allen) pour visser la vis du pignon jusqu'à ce que la rondelle ,ou la vis, touche le chassis. Veuillez à ne pas trop serrer la vis dans le pignon car cela pourrait serrer et bloquer la rondelle et donc la roue, la roue doit pouvoir tourner librement. **Note:** Faites attention si vous voulez utiliser un produit *frein filet* (comme ceux de Loctite) car ils peuvent corroder le chassis. Vous devriez d'abord tester un tel produit sur une partie plus éloignée du châssis (ou pièce non utilisée) pour vous assurer qu'il ne l'endommage pas. Note de MC Hobby: un *frein filet* est un produit prévu pour bloquer -plus ou moins fort en fonction du produit sélectionné- une vis dans le filet de l'écrou.



Ajouter les roues dentées sur le Zumo
Cliquer l'image pour l'agrandir

- Poussez les roues dentées sur les axes des moteurs, toujours avec les dents vers le chassis (et le moteur dans ce cas). La fin de l'axe du moteur doit venir à ras de l'autre côté de l'axe de la roue dentée. Truc et astuce: vous pouvez placer la roue sur la table et enfoncer l'axe moteur dans la roue jusqu'à ce que l'axe touche la table.
- C'est le moment d'ajouter les chenilles en silicone en les tendant entre les roues dentées sur le châssis. Votre Shield Zumo et châssis sont maintenant complets; ajoutez des piles et un Arduino pour faire bouger votre zumo!



Le Zumo assemblé
Cliquer l'image pour l'agrandir

Le Zumo assemblé avec un Arduino Uno
Cliquer l'image pour l'agrandir

Démonter

Si, plus tard, vous avez besoin de souder des composants sur le shield Zumo, il est possible de retirer le shield du châssis (avec quelques effort et de la prudence).

1. Retirez les chenilles et enlevez les roues dentées des moteurs avec précaution.
2. Elnevez le couvercle du bloc pile et enlevez les piles.
3. Dévisser les 4 jeux de vis qui maintiennent le shield sur le chassis.
4. Pressez le ressort (pile négatif) dans le bloc pile et faites pousser les deux connecteurs piles --avec précaution et soin-- au travers des trous du châssis. Les moteurs resterons attachés sur le shield (ils ne sont pas attachés sur le châssis mais glissé dedans).
5. Faites osciller les moteurs avec précautions pour permettre de dégager l'entretoise (plaque) avant.

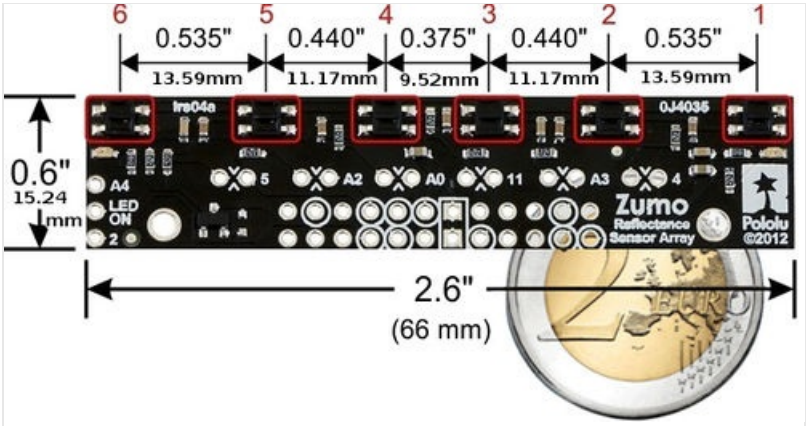
Par la suite, vous pouvez suivre la procédure de ré-assemblage dans le sens inverse. (Assurez vous de replacer les entretoises/plaques correctement.)

Ajouter le détecteur de ligne Zumo (optionel)

Sommaire

Présentation

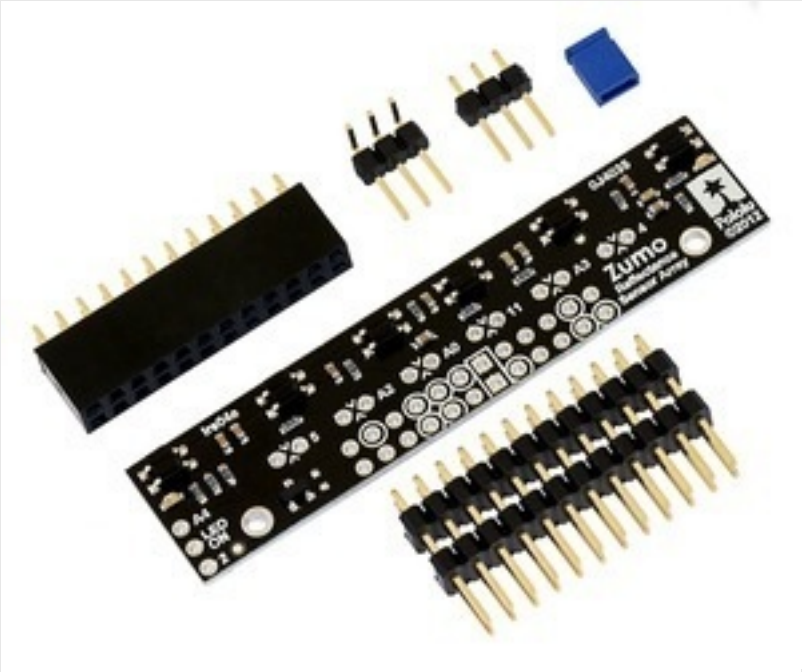
Le réseau de capteur infrarouge Zumo (suiveur de ligne) lien pololu https://www.pololu.com/product/1419 permet d'ajouter facilement la fonctionnalité de suiveur de ligne ou détection de bordure sur le robot Zumo. Il est conçu pour faciliter le montage sur la zone d'extension à l'avant du Zumo Shield et inclus tout le nécessaire pour réaliser l'installation. Notez que le réseau de capteur infrarouge n'est pas inclus avec le Shield Zumo ou le kit Robot Zumo. Il est inclus avec le Robot Zumo mais vous pouvez utiliser votre Robot Zumo sans ce senseur. Pour plus d'information concernant les fonctionnalités du réseau de capteur infrarouge, son fonctionnement et schéma, vous pouvez consulter sa fiche produit lien pololu https://www.pololu.com/product/1419 . Cette section est dévolue à l'assemblage du senseur et son utilisation sur le Shield Zumo.



Détecteur de ligne

Cliquer l'image pour l'agrandir

Assemblage du senseur



Détecteur de ligne - vue des éléments

Cliquer l'image pour l'agrandir

Le réseau de senseur infrarouge Zumo est livré avec tous les composants dont vous aurez besoin pour le connecter sur le shield Zumo:

- La carte avec le réseau de senseur monté en surface
- Connecteur mâle 2×12 - empattement de 2.54mm (sera soudé sur la carte de senseur)
- Connecteur femelle 2×12 - empattement 2.54mm (sera soudé sur le shield Zumo)
- Connecteur mâle/pinHeader 1×3 - empattement 2.54mm (optionnel, à souder sur la carte du senseur)
- Connecteur mâle 1×3 angle droit - empattement 2.54mm (optionnel, à souder sur la carte du senseur)
- Un cavalier bleu

Avant de souder le connecteur mâle principal, nous vous recommandons de souder l'un des deux connecteurs mâles 1×3 dans l'ensemble de 3 trous sur bord de la carte. Cette étape optionnelle est recommandée parce qu'elle permet un contrôle dynamique des LEDS infrarouges (et LEDs rouges). En contrôlant l'allumage/extinction de ces LEDs, vous pouvez économiser de l'énergie et rendre votre programme plus facile à déboguer. Si vous sautez cette étape, les émetteurs InfraRouge seront activer à chaque fois que le senseur sera branché sur le Zumo et que le Zumo est en marche. Nous recommandons l'usage d'un connecteur coudé monté comme indiqué sur l'image ci-dessous. Utiliser un connecteur mâle droit fonctionnera également si vous n'avez rien soudé sur l'avant du Shield Zumo (sur la zone d'extension à l'avant du shield). Si vous placez le connecteur droit, assurez-vous qu'il n'interférera pas avec les senseurs que vous placerez sur l'avant du shield Zumo (ex: en l'installant du mauvais côté ou en installant le connecteur angle-droit dans la mauvaise orientation)! Si vous comptez installer un connecteur 3 broches, il est plus facile de l'installer avec le grand connecteur 24-broches.

Pour activer le contrôle dynamique des émetteurs infrarouges, placez le connecteur 3 broches et utiliser le cavalier inclus pour connecter la broche LEDON sur la broche digitale appropriée. Si vous utilisez un **Arduino Uno** (ou plus ancien) alors vous devriez placer le cavalier pour connecter LEDON vers la **broche digital 2** (la position contre le bord de la carte); si vous utilisez un Arduino **Leonardo** ou un **A-Star 32U4 Prime**, vous devriez placer le cavalier pour connecter la broche LEDON sur la **broche analogique 4 (A4)**.

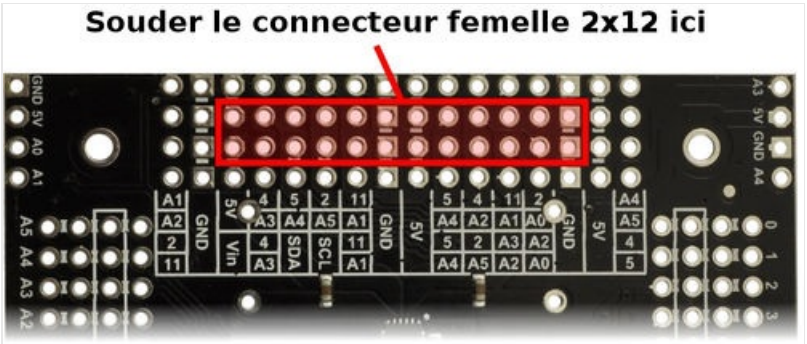
Détecteur de ligne - assemblé

Cliquer l'image pour l'agrandir

Le connecteur d'extension 2×12 mâle devrait être monté sur la carte du réseau de capteur (sur le côté opposé à celui des composants). Assurez-vous que la partie la plus courte des broches soient celui placé sur la carte d'extension (pas le côté le plus long!). Notez que seules 12 des 24 broches sont utilisées par le réseau des capteurs; ces broches sont identifiées par un **cercle sur la sérigraphie** sur le côté de la carte portant les composants CMS. Il est néanmoins recommandé de souder les 24 broches.

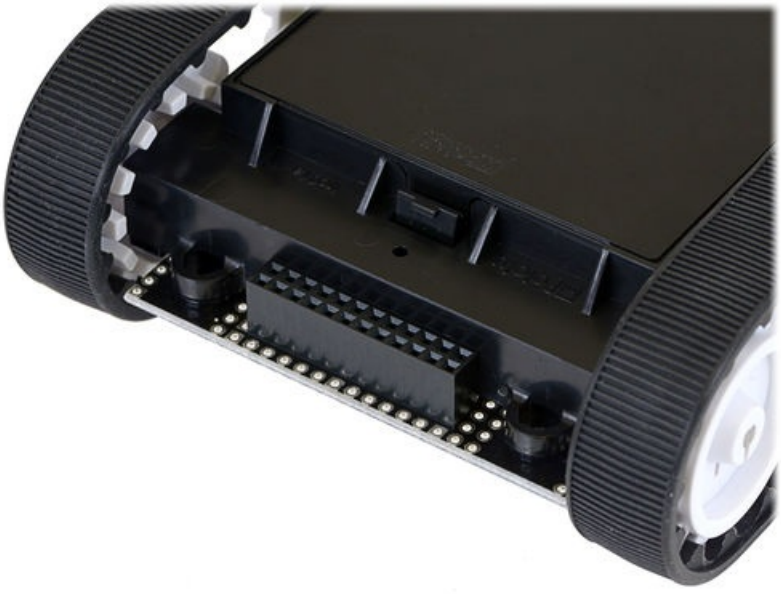
souder le connecter sur le Zumo Shield

Le connecteur femelle 2×12 inclus avec le réseau de capteur infrarouge devrait être souder sur la zone d'extension à l'avant du shield Zumo, centré sur la zone d'extension et prenant place contre le châssis zumo (les lignes 2 et 3). Même si nous recommandons de souder la totalité des 24 broches du shield, seules 12 broches sont nécessaires pour souder pour utiliser le senseur (voyez ci-dessous pour plus d'information sur les broches utilisée).



Soudez le connecteur 2x12 femelle sur le shield (vue par dessous) pour utiliser le détecteur de ligne sur le Zumo

Cliquer l'image pour l'agrandir



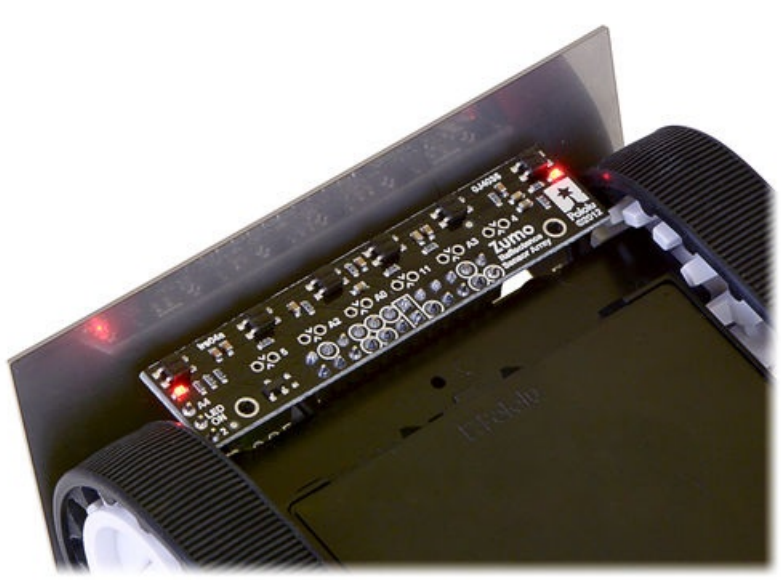
Connecteur femelle 2x12 soudé sur le Zumo Shield

Cliquer l'image pour l'agrandir

Avec le connecteur femelle en place, le réseau de senseur (assemblé) peut être branché directement sur le shield Zumo.

Placement du détecteur de ligne (sans lame sumo)

Cliquer l'image pour l'agrandir



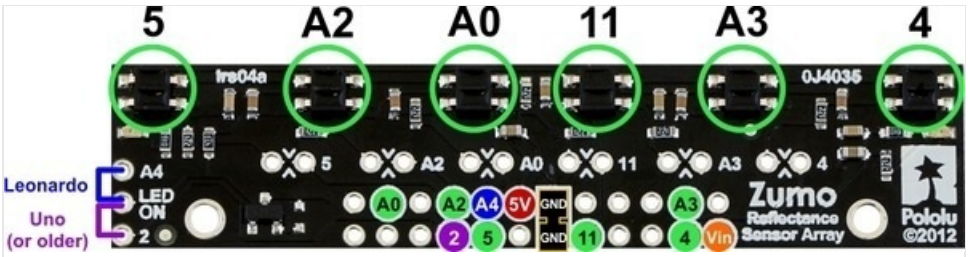
Placement du détecteur de ligne (avec lame sumo)

Cliquer l'image pour l'agrandir

Le réseau de senseur contient également deux LEDs rouge en lumière visible. Ces leds sont placées en série avec les LED infrarouges. Cela vous permet de savoir dans le réseau de capteur est activé (ou désactivé).

Broches du senseur

Le réseau de capteur infrarouge du Zumo utilise 12 broches du connecteur 2x12 pour obtenir l'alimentation et les connections I/O (*entrée/sortie*) nécessaires. Ces broches sont repérées sur la sérigraphique à l'aide d'un cercle:



Brochage du senseur

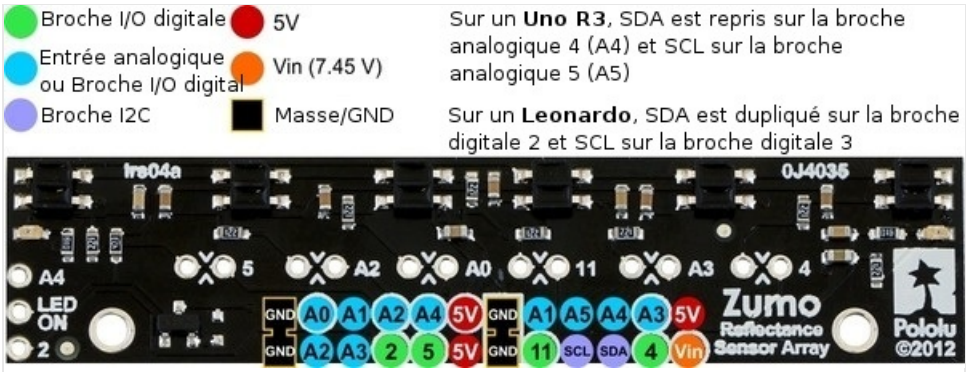
Cliquer l'image pour l'agrandir

Les entrées/sorties utilisés par défauts sur le réseau de senseur sont des broches qui ne sont pas utilisés par le shield Zumo. Le shield prévoit une broche digitale, une pour chaque senseur (5, A2, A0, 11, A3 et 4), et si vous utilisez le cavalier LEDON alors une entrée/sortie additionnelle est utilisée (soit A4 ou 2). Pour configurer la bibliothèque ZumoReflectanceSensorArray avec la configuration par défaut, appelez simplement la fonction `init` sans argument:

```
reflectanceSensors.init();
```

> Si vous ne désirez pas utiliser le cavalier de configuration LEDON alors vous devriez utiliser QTR_NO_EMITTER_PIN comme paramètre d'initialisation: `reflectanceSensors.init(QTR_NO_EMITTER_PIN)`. Dans le cas contraire, la bibliothèque essayera toujours de faire quelque-chose avec la broche d'émission (A4 ou 2, en fonction de la carte Arduino utilisée) ET cela vous empêchera d'utiliser cette broche à votre guise pour votre propre usage.

Lorsque vous souderez le connecteur mâle 2×12 sur le senseur, vous aurez uniquement besoin de souder ces broches entourées. Si vous soudez les 24 broches, le réseau de senseur sera connecté sur les broches additionnelles du shield Zumo (sur la zone d'extension à l'avant)... même si le réseau ne fait rien avec ces broches dans sa configuration par défaut:



Brochage du senseur

Cliquer l'image pour l'agrandir

Désactivé et remapper le senseur

De nombreuses applications ne requière pas l'utilisation des 6 capteurs infrarouges, ce qui vous permettrait de libérer des entrées/sorties pour votre projet (ex: détecteurs d'obstacle). Dans pareil cas,, vous pouvez désactiver la ligne du senseur et libérer la ligne d'entrée/sortie associée. La carte du réseau de senseur dispose de 6 paires de trous correspondant aux différents senseurs. L'ordre des paires correspond à l'ordre des senseurs. En regardant le côté de la carte avec les composants, le trou à droite de la paire est connecté vers Arduino et le trou gauche est connecté sur le senseur. Il y a une simple trace reliant les deux trous de chaque paire sur la face composant -ET- cette trace peut être coupée pour libérer la ligne d'entrée/sortie. L'emplacement correct pour sectionner la piste est indiqué à l'aide d'une croix sur la sérigraphie.

couper la trace pour remapper
Cliquer l'image pour l'agrandir

Par exemple, si vous désirez utiliser votre Zumo pour résoudre un labyrinthe à ligne (*line maze*), vous pouvez atteindre le résultat avec seulement 4 senseurs: vous pouvez utiliser les deux senseurs du milieu pour traquer la ligne et les deux senseurs extérieurs pour détecter les intersections. Cela permet de libérer les lignes des deux autres senseurs en faisant les modifications suivantes:

couper la trace pour remapper
Cliquer l'image pour l'agrandir

Voilà, vous avez maintenant un réseau de capteur infrarouge avec quatre-senseurs effectifs libérant les broches A2 et A3 sont disponibles pour d'autres utilisations. Pour configurer la bibliothèque ZumoReflectanceSensorArray avec cette nouvelle configuration, appelez la fonction `init` avec les paramètres suivants:

```
byte pins[] = {4, 11, A0, 5};
reflectanceSensors.init(pins, 4);
```

Aternativement, vous pouvez déclarer deux objets `ZumoReflectanceSensorArray`, un des objets pour les deux senseurs extérieurs et un autre pour les deux senseurs intérieurs, cela permet de rédiger un code plus propre mais avec l'inconvénient qu'il n'est alors plus possible de lire les quatres senseur en parallèle.

Si, plus tard, vous décidez de réactiver ces senseurs, vous pouvez connecter un fils pour remplacer la trace coupée -ou- remappée le senseur vers une broche différente. l'exemple suivant montre comment ré-activer le senseur A2 et remapper le senseur de A3 vers A5:

Remapper le senseur
Cliquer l'image pour l'agrandir

Pour configurer la bibliothèque ZumoReflectanceSensorArray avec ce nouveau mapping, appelez la fonction `init` avec les arguments suivants:

```
byte pins[] = {4, A5, 11, A0, A2, 5};
reflectanceSensors.init(pins, 6);
```

Ou, si vous ne désirez pas utiliser la ligne de contrôle des LEDs infrarouges:

```
byte pins[] = {4, A5, 11, A0, A2, 5};
```

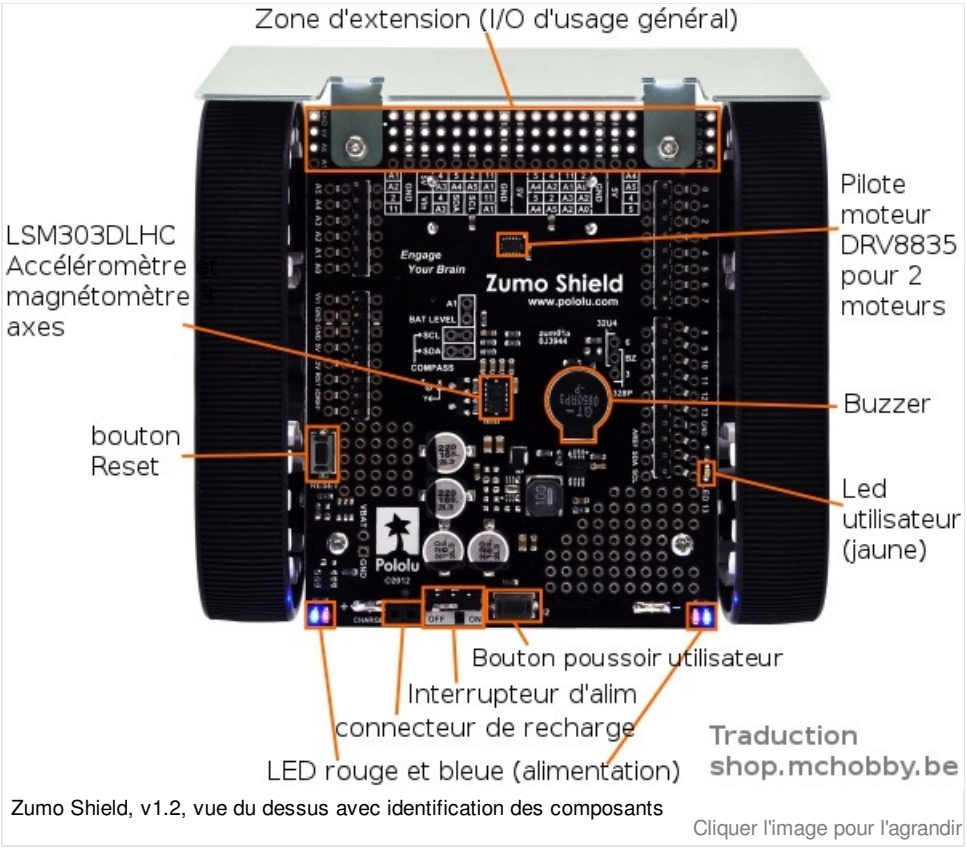
```
reflectanceSensors.init(pins, 6, 2000, QTR_NO_EMITTER_PIN);  // 2000 = timeout après 2 ms
```

Fonctionnalités et composants

Sommaire

Fonctionnalités principales

Les fonctionnalités de base du shield Zumo (v1.2) sont indiqués sur le diagramme suivant:



Zumo Shield, v1.2, vue du dessus avec identification des composants

Cliquer l'image pour l'agrandir

Pour le shield Zumo original, un diagramme correspond est disponible ici https://www.pololu.com/file/download/zumo-shield-v1.0-labeled-components.jpg?file_id=0J810 (206k, jpg) (la seule différence concerne le senseur intertiel placé sur la carte).

Alimentation

Le châssis Zumo dispose d'un comportement interne pour 4 piles AA. Pololu recommande l'utilisation de piles NiMH AA rechargeables, ce qui représente une tension nominale de 4.8 V (1.2 V par pile). Vous pouvez également utiliser des piles alcaline, ce qui produit une tension nominale de 6V.

La bordure arrière du shield Zumo reprend un connecteur femelle deux pôles connecté directement sur les connecteurs des piles. Ce connecteur peut être utilisé pour recharger les piles du Zumo sans retirer les piles du châssis. La broche positive du connecteur de recharge est sur la gauche et identifié par un signe (+). Un chargeur tel que le iMAX-B6AC lien pololu https://www.pololu.com/product/2588 , connecté sur des fils de prototypage permet de recgarger directement le Zumo.

Après la protection contre la polarisation inverse accidentelle, la tension des piles est connecté sur le reste du shield par l'intermédiaire de l'interrupteur d'alimentation. Cette tension d'alimentation (derrière l'interrupteur) est désigné comme VBAT et

fournit la puissance aux moteurs par l'intermédiaire du DRV8835. Un régulateur boost est également alimenté par VBAT et produit une tension de 7.45 V permettant d'alimenter votre Arduino par l'intermédiaire de la broche Vin. Par la suite, les régulateurs 5V et 3.3V de votre Arduino fournira la tension d'alimentation au contrôleur moteur (étage logique), au buzzer et à la boussole du Zumo Shield.

Attention: lorsque vous alimentez votre Arduino depuis le shield Zumo, vous ne devez JAMAIS connecter une source d'alimentation différente sur la broche VIN - ou- connecteur d'alimentation de votre Arduino. Si vous le faites cela créera un court-circuit entre l'alimentation du shield et l'alimentation de l'Arduino, ce qui pourrait endommager votre Arduino et le shield Zumo de façon permanente.

Lorsque votre Arduino est connecté sur votre ordinateur par l'intermédiaire du port USB, votre Arduino sera alimenté (et produira du 5V et 3.3V sur le shield) même si l'interrupteur du shield Zumo est coupé. Cette fonctionnalité est bien pratique si vous voulez tester votre programme Arduino sans permettre aux moteurs de fonctionner (puisque l'interrupteur ouvert déconnecte l'alimentation VBAT des moteurs.

LEDs

Il y a 5 LEDs sur le shield Zumo:

- Un enseble de LEDs d'alimentation (dite "*Power*" en anglais), l'une **bleue** et l'autre **rouge**, sont localisées sur chaque coin arrière du shield.
- Une LED utilisateur **jaune** est localisée sur le bord droit du shield. La LED est connectée sur la **broche digitale 13** d'Arduino (en parallèle de celle présente sur la carte Arduino).

Boutons poussoirs

Deux boutons poussoirs peuvent être soudés sur le shield Zumo:

- Le **bouton Reset** est localisé sur le bord gauche du shield. Il est connecté sur la broche RESET d'Arduino et peut être pressé pour réinitialiser votre Arduino.
- Le **bouton Utilisateur** (dit "*user*" en anglasi) est localisé sur la bordure arrière du shield. Le bouton est connecté sur la **broche digitale 12** de votre Arduino; Presser le bouton ramène le potentiel. Nous recommandons donc d'activer la résistance pull-up de la broche (pour maintenir la broche au niveau haut par défaut). La bibliothèque Pushbutton, inclus dans la bibliothèque du shield Zumo, permet de détecter plus facilement la pression du bouton (et applique un déparasitage logiciel).

Pilote moteur

Le Zumo shield intègre un contrôleur de moteur DRV8835 capable de piloter les deux micro-moteurs du shield Zumo. Quatres broches Arduino sont utilisé pour piloter le contrôleur DRV8835:

- **Broche digitale 7** contrôle le **sens de rotation du moteur droit** (LOW pour faire avancer le Zumo et HIGH pour le sens inverse).
- **Broche digitale 8** contrôle le **sens de rotation du moteur gauche**.
- **Broche digitale 9** contrôle la **vitesse du moteur droit** avec un signal PWM (pulse width modulation).
- **Broche digitale 10** contrôle la **vitesse du moteur gauche** avec un signal PWM.

La bibliothèque ZumoMotors offre des fonctions qui permet de contrôler facilement les moteurs (et peut également prendre en charge l'inversion du sens de rotation au cas où vous auriez accidentellement inversé les soudures du moteurs).

Buzzer

Le shield Zumo inclus un buzzer qui peut être utilisé pour générer de simples sons et de la musique (par exemple, vous pourriez l'utiliser pour produire un décompte audible avant le début d'un match sumo). La ligne contrôlant le buzzer est identifiée par le libellé BZ; Si vous pilotez cette broche avec un signal haut/bas pulsé à une fréquence donnée, le buzzer produira un son de cette même fréquence.

La bibliothèque ZumoBuzzer utilise le générateur PWM matériel pour jouer des notes sur le buzzer avec la **broche digitale 3** (OC2B) de votre Arduino **Uno** (ou ancien Arduino), ou avec la **broche digitale 6** (OC4D) d'un Arduino **Leonardo** ou **A-Star 32U4 Prime**. Un cavalier permet de connecter l'entrée BZ sur la broche Arduino appropriée comme cela est détaillé dans la Section 3.c.

Zone d'extension Frontale/Avant

Plusieurs connexions d'entrée/sortie, d'alimentation et masse sont placés dans la zone d'extension frontale du shield Zumo. Ces connexions permettent de monter des senseurs complémentaires ou autres composants. Le brochage de cette zone d'extension est détaillé dans la Section 3.b.

Senseurs Inertiels

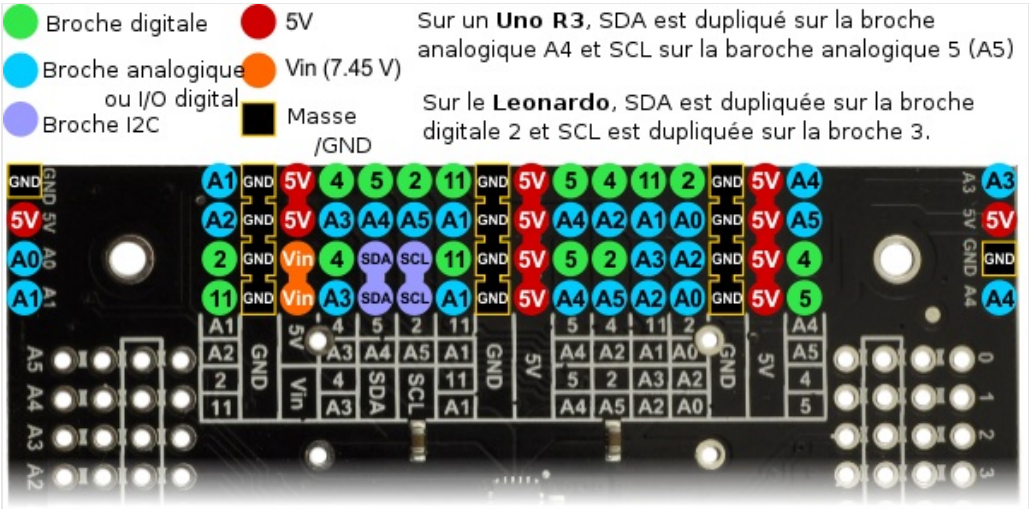
Le shield Zumo inclus des senseurs inertiels sur la carte. Ces senseurs peuvent être utilisés par des applications avancées pour détecter l'accélération et l'orientation:

- Le Shield Zumo **v1.2** embarque un accéléromètre et magnétomètre 3 axes **LSM303D** ainsi qu'un gyroscope 3-axes **L3GD20H**.
- Le Shield Zumo **original** embarque un accéléromètre et magnétomètre 3 axes **LSM303DLHC**.

Les senseurs inertiels sont détaillés dans la Section 3.d.

L'extension avant

Les broches du connecteur d'extension frontal/avant du shield Zumo est présenté sur le diagramme suivant:



Brochage de l'extension avant

Cliquer l'image pour l'agrandir

Ce diagramme est également disponible sous la d'un document PDF téléchargeable : Zumo Shield front expansion pinout https://www.pololu.com/file/download/zumo_shield_front_expansion_pinout.pdf?file_id=0J592 (*552k, pdf, pololu.com*).

Le connecteur d'extension frontal permet d'avoir accès au broches digitales 2, 4, 5 et 11 ainsi que les broches analogiques A0 à A5. Il offre également un accès aux deux broches du bus I2C (SDA et SCL). Notez cependant que les broches du bus I2C ne sont pas indépendante; elles sont respectivement dupliquées sur les broches A4 et A5 d'un Uno R3 -ET- les broches digitales 2 et 3 d'un Leonardo / A-Star 32U4 Prime. Typiquement, vous serez capable d'utiliser ces broches pour la communication I2C ou utilisation comme broche d'entrée/sortie (mais pas les deux en même temps).

De surcroît, la broche analogique A1 est utilisée pour surveiller la tension des piles (uniquement si vous avez placez le cavalier "niveau des piles".

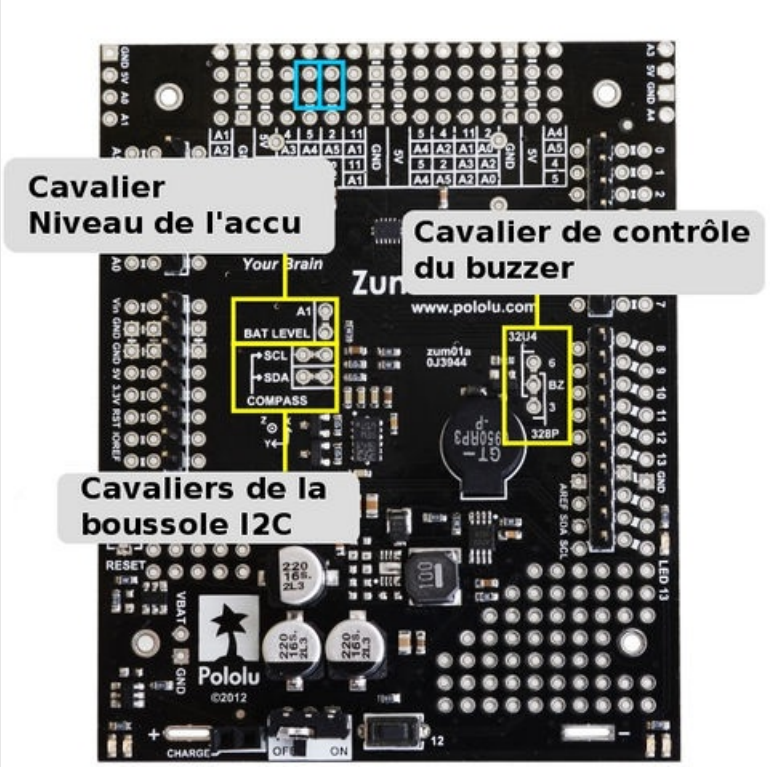
Notez que seul les composants et connecteurs sur les trois premières lignes de broches peuvent étendre des interfaces sous le shield; La quatrième ligne est au dessus du châssis et peut uniquement être utilisé pour monter des extensions au dessus du shield.

Si vous utilisez un Arduino Uno R2 (ou plus ancien), qui ne dispose pas de broche I2C séparée, les broches SDA et SCL du Shield Zumo seront connectés sur *rien*. Pour utiliser un périphérique I2C sur ces broches, vous pouvez connecter vous même SDA sur A4 --ET-- SCL sur A5 (en réalisant des connexions entre les deux ensembles de broches sur la zone frontale. La Section 3.c explique les lignes I2C et les cavaliers les connectans sur le module boussole.

En fonction du modèle d'Arduino, la broche digitale 3 ou 6 est utilisée pour contrôler le buzzer si vous avez installé le cavalier de contrôle du buzzer. Si vous utilisez un Uno, la broche 6 sera disponible pour une utilisation générale. Si vous utilisez un Leonardo ou un A-Star, la broche 3 sera disponible si vous n'utilisez pas le bus I2C. Ces broches ne sont pas accessibles via la zone d'extension frontale, mais sont disponibles depuis d'autres points sur le shield et permettre d'interfacer de l'électronique additionnelle si disponible. En plus, la broche digitale 12 peut être utilisée pour interfacer différent type de composant électronique, plus particulièrement si vous n'utilisez pas le bouton utilisateur. La broche 12 est complètement disponible lorsque le bouton est laissé dans son état par défaut -non pressé- et est placé à la masse par l'intermédiaire d'une résistance de 1k lorsque le bouton est pressé.

Cavaliers de configuration

Le shield Zumo a plusieurs cavaliers de configuration qui vous permet de modifier la façon dont votre Arduino est connecté (voir l'image ci-dessous).



Zumo shield, vue du dessus avec connexions identifiées
Cliquer l'image pour l'agrandir

- Le **cavalier 'niveau des piles'** connecte la broche analogique A1 de votre Arduino sur un pont diviseur de tension, ce qui permet de mesurer la tension des piles du Zumo. Ce cavalier est déconnecté par défaut. Il peut être connecté en soudant un morceau de fil entre les deux trous.

La diviseur produit une tension de sortie éqale au 2/3 de la tension des piles, ce qui permet de rester en dessous de la tension de sécurité (5V) des entrées analogiques. Par exemple, si la tension des piles est de 4.8V, la tension sur la broche analogique A1 sera de 3.2V (soit 4.8 * 2 / 3). En utilisant la fonction analogRead() d'Arduino, une tension de 5V retourne une valeur 1023. Une tension de 3.2 V retounera une valeur égale à 655. Pour convertir la valeur de analogRead() en tension (celle des piles), il faut multiplier cette valeur par 5000 mV×3/2 et la diviser par 1023:

```
unsigned int batteryVoltage = analogRead(1) * 5000L * 3/2 / 1023;
```

- La cavalier **contrôle buzzer** connecte une des sortie PWN d'Arduino sur le Buzzer du shield Zumo. Ce cavalier est déconnecté par défaut sur le robot Zumo assemblé et en kit; le cavalier doit être connecté pour activer le buzzer.

Si vous avez un Arduino **Uno** (ou plus ancien, avec un microcontrôleur ATmega328P ou ATmega168), vous devriez ponter la paire de trous BZ et **328P**, cela connecte la broche BZ (le buzzer) sur la broche digitale 3. Si vous avez un **A-Star 32U4 Prime** ou Arduino **Leonardo**, vous devriez ponter la paire de trous BZ et **32U4**, cela connecte la broche BZ sur la broche digitale 6. Ce sont les broches utilisées par la bibliothèque ZumoBuzzer (la bibliothèque sélectionne automatiquement la bonne broche en fonction du microcontrôleur détecté). Vous trouverez plus de détails sur le buzzer dans la Section 3.a.

- Le cavalier **boussole/gyro I2C** connecte la la ligne d'horloge I2C (SCL) et ligne de donnée I2C (SDA) du senseur inertiel du Zumo sur les broches SCL et SDA de votre Arduino. Ces cavaliers sont connectés par défaut mais peut être déconnecté en coupant les traces entre chaque paire de trous.

Sur un Arduino Uno R3, les signaux SCL et SDA sont dupliqués respectivement sur les broches analogiques 5 et 4. Sur un A-Star et Arduino Leonardo, les signaux SCL et SDA sont dupliqués respectivement sur les broches digitales 3 et 2. En utilisant les senseurs I2C du shield, vous ne pourrez pas utiliser ces broches pour d'autres fonction. Notez que les résistances pull-up I2C affecterons les lectures sur ces broches même si vous n'utilisez pas activement la boussole I2C (il est donc important de couper les traces pour déconnecter les senseurs inertiels --et ses résistances pull-ups-- si vous voulez réattribuer la fonction des broches SCL et SDA).

Notez que les broches SCL et SDA n'existent pas sur les Arduino dont la révision est inférieure à R3. Dans ce cas, vous devrez connecter manuellement le trou SCL sur la broche analogique A5 et le trou SDA sur la broche analogique A4 du Zumo Shield pour pouvoir utiliser la boussole avec un ancien Arduino. L'emplacement le plus approprié pour réaliser cette opération est la zone d'extension frontale, où toutes ces broches sont localisée ensemble, comme indiqué dans la section d'information ci-dessous.

Vous trouverez plus de détails sur les senseurs inertiels dans la Section 3.d.

A la place de faire des connexions filaires, vous pouvez souder un connecteur 1×3 mâle dans les trou du buzzer. Cela permet d'utiliser un cavalier pour connecter/déconnecter le buzzer (note: ce connecteur est déjà installé sur vous avez chatez une version assemblée du Zumo Robot, mais il faudra positionner le cavalier en fonction de votre modèle d'Arduino). Vous pouvez également utiliser des connecteurs mâle et des cavaliers pour la "niveau des piles" et le bus I2C de la boussole si vous avez un Arduino Uno avec un microcontrôleur CMS (composant monté en surface), Arduino Leonardo ou A-Star 32U4 Prime. Cependant, il n'y a pas assez d'espace disponible pour utiliser des connecteurs mâles sur le "niveau de batterie" et la boussole I2C avec un Arduino équipé d'un microcontrôleur DIP (through-hole).

Senseurs inertiels (accéléromètre, magnétomètre et gyro)

Introduction

Le shield Zumo inclus des senseurs inertiels sur la carte. Ces senseurs peuvent être utilisés pour des applications avancées comme détecter des collisions et déterminer sa propre orientation.

Tous les versin du shield Zumo comporte une boussole qui combine un accéléromètre 3 axes et un magnétomètre 3 axes dans une seule puce interfacée sur le bus I2C. Cette puce est le LSM303D lien pololu https://www.pololu.com/product/2127 sur un shield v1.2 ou un LSM303DLHC lien pololu https://www.pololu.com/product/2124 on the sur le shield Zumo original.

La **version v1.2** inclus également un gyroscope 3 axes L3GD20H lien pololu https://www.pololu.com/product/2129 également interfacé sur le bus I2C.

Pololu recommande la lecture de la Fiche technique LSM303D https://www.pololu.com/file/download/LSM303D.pdf?file_id=0J703 (*1MB pdf, pololu*), Fiche technique L3GD20H https://www.pololu.com/file/download/L3GD20H.pdf?file_id=0J731 (*3MB pdf, pololu*), et/ou fiche technique LSM303DLHC https://www.pololu.com/file/download/LSM303DLHC.pdf?file_id=0J564 (*629k pdf, pololu*) pour comprendre comment fonctionne les senseurs afin de savoir comment les utiliser.

Utiliser les senseurs

Le shield inclus également des convertisseurs de niveau logique (dit *level sshifter* en anglais) qui permettent de connecter les senseurs inertiels 3.3v sur un Arduino utilisant des niveau logiques en 5V. Les senseurs, convertisseurs de niveau logique (*level shifters*) et résistances pull-up du bus I2C sont connectés, par défaut, sur les broches SCL et SDA du shield Zumo. Il est possible de les déconnecter en coupant les traces entre les paires de trou SDA et SCL, ce qui permet d'attribuer une autre fonction à ces broches SDA et SCL de votre Arduino. Notez que dans le cas d'un ancien Arduino (inférieur à R3), il sera nécessaire de faire quelques connexions additionnels pour utiliser les senseurs inertiels du shield Zumo car les anciens Arduino ne disposent pas broches SCL et SDA séparées; Reportez vous à la Section 3.c pour plus d'information concernant les connexions de la boussole.

Pololu à écrit une bibliothèque LSM303 pour Arduino https://github.com/pololu/lsm303-arduino et une bibliothèque L3G pour Arduino https://github.com/pololu/l3g-arduino qui facilite l'interfaçage avec les senseurs depuis votre Arduino. Comme le démontre ce projet d'exemple, il est possible d'utiliser le magnétomètre pour aider le Zumo à évaluer l'amplitude de ses rotations.

De surcroît, la combinaison de l'accéléromètre, du magnétomètre et du gyroscope de la version v1.2 du shield Zumo est suffisant pour implémenter une unité de mesure inertielle (un IMU: inertial measurement unit); Les senseurs utilisés sont identiques à ceux utilisé sur le MinIMU-9 v3 de Pololu. Par conséquent, vos codes Arduino écrit pour le MinIMU-9 lien pololu https://www.pololu.com/product/2468 (tel que l'exemple AHRS de Pololu https://github.com/pololu/minimu-9-ahrs-arduino) peuvent être adaptés pour fonctionner le Zumo Robot équipé d'un Shield Zumo V1.2 (contrôlé par Arduino).

Notes sur le magnétomètre

Notez que le magnétomètre LSM303 est affecté par les courants circulant dans les moteur et le buzzer lorsque ces derniers fonctionnent. Les lectures sont facilement influencées par les distorsions magnétiques dans l'environnement du Zumo (comme les armatures des bétons armés). Il en résulte qu'il est très difficile de déterminer la direction du Zumo avec précision en se basant sur les données du magnétomètre. Cependant, durant les tests de Pololu, le magnétomètre se révèle être très utile pour détecter des changements d'orientation relatif; par exemple, une fois les lectures du champs magnétique compensée pour un environnement particulier, ces données peuvent être utilisées pour aider le Zumo à tourner à gauche ou à droite d'un angle donné (au lieu de faire tourner les moteurs pendant un temps donné pour atteindre approximativement l'angle souhaité).

Durant ces tests, Pololu à noté que que les piles, les moteur et le courant des moteurs affecte l'axe Z du magnétomètre plus sévèrement que les axes x et y. Par conséquent, vous souhaiterez probablement ignorer les lectures de l'axe Z. Il est généralement possible de déterminer la direction avec des résultats corrects à partir de la lecture des axes x et y du magnétomètre. Par ailleurs, vous pourriez avoir besoin d'ajuster la sensibilité du magnétomètre; si le magnétomètre retourne une valeur de **-4096**, cela indique une sensibilité trop faible pour votre environnement.

Diagrammes et schémas

Les schémas et diagrammes du shield Zumo sont téléchargeable aux format PDF:

- Diagrammes du shield Zumo v1.2 https://www.pololu.com/file/download/zumo-shield-v1_2-schematic.pdf?file_id=0J779 (**449k pdf, Pololu**)
- Diagrammes du shield Zumo original https://www.pololu.com/file/download/zumo_shield_schematic.pdf?file_id=0J591 (**121k pdf, Pololu**)

Table des broches Arduino

Broche digitale	Fonction du Shield Zumo	Notes/fonctions alternatives
0	Entrée/Sortie digitale	Ligne RX du port série pour programmation et communication sur un **Uno** (ou version antérieure)
1	Entrée/Sortie digitale	Ligne TX du port série pour programmation et communication sur un **Uno** (ou version antérieure)
2	Entrée/Sortie digitale (Extension avant)	Ligne **SDA** I2C sur un **Leonardo** et **A-Star 32U4 Prime**
3	Entrée/Sortie digitale	Cavalier optionnel pour le **signal de contrôle du buzzer** pour **UNO** et **A-Star 32U4 Prime** Ligne **SCL** I2C sur un **Leonardo** et **A-Star 32U4 Prime**
4	Entrée/Sortie digitale (Extension avant)	yyyy
5	Entrée/Sortie digitale (Extension avant)	yyyy
6	Entrée/Sortie digitale	Cavalier optionnel pour le **signal de contrôle du buzzer** pour **Leonardo** et **A-Star 32U4 Prime**
7	**Signal SENS ROTATION pour moteur droit**	
8	**Signal SENS ROTATION pour moteur gauche**	
9	**Signal PWM pour moteur droit**	
10	**Signal PWM pour moteur gauche**	
11	Entrée/Sortie digitale (Extension avant)	
12	Entrée/Sortie digitale	**Bouton utilisateur** (niveau bas lorsqu'il est pressé)
13	Entrée/Sortie digitale	**LED utilisateur JAUNE** (niveau haut pour alluler)

Broche analogique	Fonction Shield Zumo	Notes/fonctions alternatives
A0	Entrée analogique et entrée/sortie digitale (extension avant)	
A1	Entrée analogique et entrée/sortie digitale (extension avant)	optional jumper to **battery level voltage divider**
A2	Entrée analogique et entrée/sortie digitale (extension avant)	
A3	Entrée analogique et entrée/sortie digitale (extension avant)	
A4	Entrée analogique et entrée/sortie digitale (extension avant)	Ligne **SDA** I2C sur un **Uno** ou Arduino plus anciens
A5	Entrée analogique et entrée/sortie digitale (extension avant)	Ligne **SCL** I2C sur un **Uno** ou Arduino plus anciens

Bibliothèque Zumo Shield (Arduino)

Sommaire

Introduction

Le shield Zumo de Pololu permet d'écrire des croquis/sketchs de contrôle du Robot très facilement et très rapidement. Un lien pour télécharger la bibliothèque (et instruction d'installation) peut être trouvé sur la page GitHub de la bibliothèque https://github.com/pololu/zumo-shield .

Une fois installée, nous recommandons de tester les croquis/sketch d'exemples de chaque bibliothèque, exemple qui peuvent être trouvés sous le menu **Fichier > Exemples > (nom de la bibliothèque)**. Consulter les exemples permet d'avoir une meilleure compréhension des fonctions de la bibliothèque et de leur utilisation. Vous pouvez également trouver des exemples plus élaborés, et non spécifique à une bibliothèque particulière, sous le point de menu **Fichier > Exemples > ZumoExamples**.

La Section 7 détaille ces exemples.

La bibliothèque du shield Zumo inclus les éléments suivants:

- **ZumoMotors** - Contrôle des moteurs du Zumo
- **ZumoBuzzer** - Contrôle du Buzzer
- **PushButton** - Détection de la pression sur le bouton
- **ZumoReflectanceSensorArray** - Utilisation du senseur réflectif (senseur de ligne)
- **QTRSensors** - Concerne les senseurs infrarouge utilisés avec le senseur réflectif.

ZumoMotors

La bibliothèque ZumoMotors offre des fonctions de contrôle de vitesse (et direction) basées sur les signaux PWM. Elle permet de contrôler les deux moteurs du Zumo via le pilote de moteur DRV8835 inclus sur la carte. Sur un Arduino à base de microcontrôleur ATmega328P, ATmega168 et ATmega32U4 (inclus le A-Star 32U4 Prime, Arduino Leonardo, Arduino Uno et la plupart des aciens Arduinos), les fonctions de contrôle moteur utilisent les sorties PWM du Timer1 pour générer la modulation du signal en largeur d'impulsion (PWM: pulse width modulation) à la fréquence de 20 kHz. (Voyez la Section 3 "Le Shield Zumo en détails" pour plus de détails sur le pilote moteur et ses connexions.)

Si vous avez accidentellement soudé un moteur à l'envers (ex: orienté à l'opposé des instructions d'assemblage), vous pouvez simplement appeler `flipLeftMotor(true)` et/ou `flipRightMotor(true)` pour faire en sorte que le moteur agisse en concordance avec les instructions de direction dans votre code.

ZumoBuzzer

La bibliothèque ZumoBuzzer offre des fonctions permettant de produire différents sons qui sont reproduits sur le buzzer du shield Zumo. Cela va du simple "beep" à la mélodie complexe. (Voir la Section 3 pour plus de détails à propos du buzzer et la Section 3.c pour des explications concernant le cavalier de configuration du buzzer.)

La bibliothèque ZumoBuzzer est pleinement compatible avec les fonctions OrangutanBuzzer https://www.pololu.com/docs/0J18/3 de la bibliothèque AVR C/C++ de Pololu https://www.pololu.com/docs/0J20 . En conséquence, les mélodies écrites pour les fonctions OrangutanBuzzer fonctionnerons également avec les fonctions ZumoBuzzer.

PushButton

La bibliothèque Pushbutton offre un ensemble de fonctions utiles pour détecter (et déparasiter) la pression des bouton. L'utilisation la plus évidente de cette bibliothèque concerne le bouton poussoir du shield zumo (celui branché sur la broche digitale 12). Cette bibliothèque peut également être utilisée pour une utilisation plus générale, a savoir l'interfaçage d'autre

bouton poussoir ou interrupteur (même sans le shield Zumo).

ZumoReflectanceSensorArray

Cette bibliothèque offre un ensemble de fonctions lisant les valeurs des différents capteurs infrarouges du réseau de capteur du Zumo lien pololu https://www.pololu.com/product/1419 . Voyez la Section 2.c pour plus d'informations sur le réseau de senseur infrarouge du Zumo.

Cette bibliothèque est basées sur la bibliothèque QTRSensors. La classe ZumoReflectanceSensorArray est une sous classe de QTRSensorsRC. Les fonctions offertes par QTRSensorsRC peuvent également être utilisées sur la classe ZumoReflectanceSensorArray. Elles sont documentées dans le document de la bibliothèque Arduino pour les senseurs réflectifs QTR de Pololu (*QTR Reflectance Sensors*) https://www.pololu.com/docs/0J19 .

QTRSensors

This library, which can also be found in the qtr-sensors-arduino repository https://github.com/pololu/qtr-sensors-arduino , is a general library for interfacing with Pololu QTR reflectance sensors lien pololu https://www.pololu.com/category/123/pololu-qtr-reflectance-sensors . Since the Zumo reflectance sensor array lien pololu https://www.pololu.com/product/1419 has the same interface as the QTR RC reflectance sensors, the ZumoReflectanceSensorArray library uses QTRSensors to read the sensor array.

Projets d'exemples

- RC Zumo
- Détection des bords simplifiée
- Collision-detecting sumo robot
- Suiveur de ligne
- Résolution de labyrinthe
- Utiliser la boussole

Ces exemples montrent comment programmer un Zumo contrôlé par Arduino. Ces exemples couvrent des tâches intéressantes et complexes. Les fichiers sources des exemples sont inclus dans le téléchargement de la bibliothèque Zumo Shield Arduino. Une fois la bibliothèque installée, les exemples sont accessibles depuis l'environnement Arduino depuis le menu **Fichier > Exemples > ZumoExamples**.

RC Zumo

En connectant un récepteur RC (Radio Commande) et en exécutant ce programme d'exemple, vous pouvez transformer un Zumo en véhicule télécommandé. Si vous avez installé la bibliothèque Shield Zumo pour Arduino alors vous trouverez le croquis/sketch Arduino via le point de menu **Fichier > Exemples > ZumoExamples > RCControl**.

Un robot Zumo branché avec un récepteur RC pour créer un véhicule contrôlé à distance
Cliquer l'image pour l'agrandir

Pour connecter facilement un récepteur sur le Shield Zumo, vous pouvez souder deux connecteur/pinHeader mâle http://shop.mchobby.be/product.php?id_product=76 1×3 broches lien pololu https://www.pololu.com/product/966 comme indiqué sur le diagramme ci-dessous. Ensuite, vous pouvez connecter une paire de câble servo lien pololu https://www.pololu.com/category/112/servo-cables entre le récepteur et les shield Zumo. (Si votre récepteur RC dispose d'une source d'alimentation séparée alors vous devriez uniquement connecter la masse et le source du signal vers votre Zumo.)

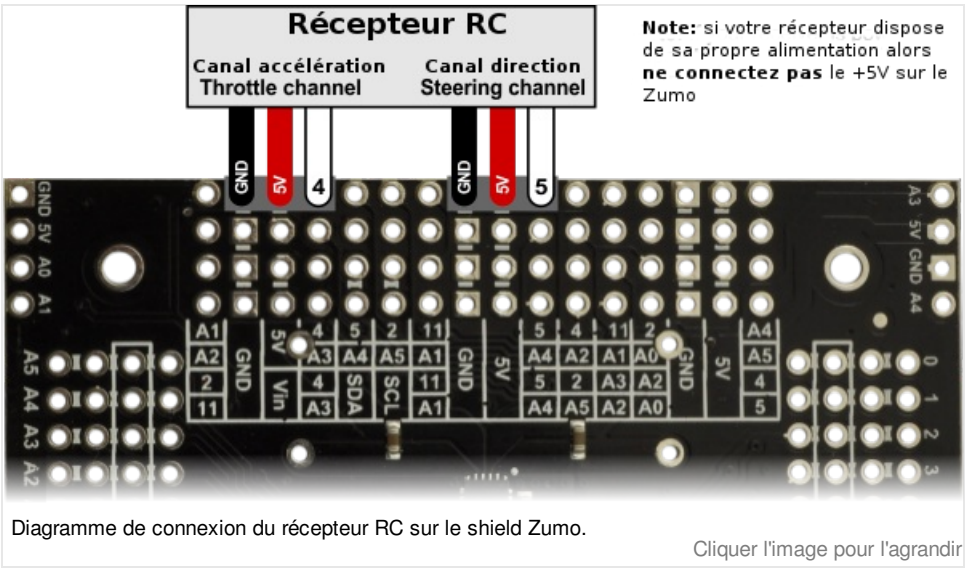


Diagramme de connexion du récepteur RC sur le shield Zumo.
Cliquer l'image pour l'agrandir

Ce programme utilise la bibliothèque PulseIn http://arduino.cc/en/Reference/PulseIn d'Arduino pour lire le signal en provenance du récepteur. Par défault, l'exemple par du principe que les canaux accélérateur (*throttle*) et direction (*steering*) sont connecté respectivement sur les broches 4 et 5 (comme sur le diagramme). Le signal des deux canaux est analysé pour déterminer la vitesse du moteur droit et moteur gauche (ce qui permet un contrôle plus intuitif).

Code

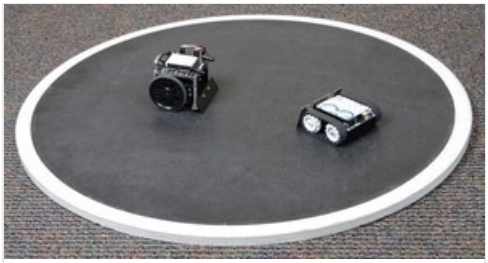
Voici une copie de l'exemple avec traduction des commentaires pour vous aider à mieux comprendre le fonctionnement du croquis/sketch

Nous recommandons de toujours charger l'exemple depuis les codes d'exemples de la bibliothèque Zumo.

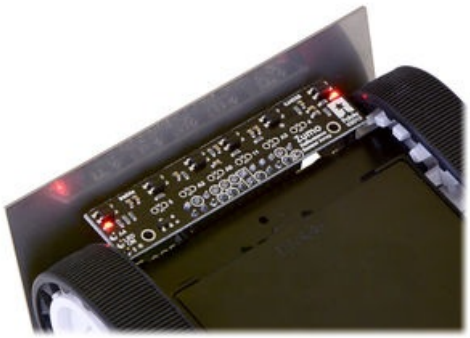
```
#include <ZumoMotors.h>

#define THROTTLE_PIN   4 // Canal accélération (throttle) du récepteur RC
#define STEERING_PIN   5 // Canal de direction (steering) du récepteur RC
#define LED_PIN       13 // broche de la LED utilisateur

#define MAX_SPEED             400 // Vitesse max moteur
#define PULSE_WIDTH_DEADBAND   25 // Différence de largeur d'impulsion depuis 1500 us (microseconds) à ignorer (pour compenser l'offset/décalage de la position centrale)
#define PULSE_WIDTH_RANGE     350 // Différence de largeur d'impulsion depuis 1500 us qui doit être considéré comme l'amplitude totale de l'entrée (valeur max)
                                  // Par exemple une valeur de 350 signifie l'utilisation d'une largeur d'impulsion (pulse width) <= 1150 us OU >= 1850 us
                                  // est considérée comme ayant atteint l'amplitude maximale de l'entrée (de la manette de contrôle)


void setup()
{
  pinMode(LED_PIN, OUTPUT);

  // Décommenter une ou deux ligne pour corriger la direction du moteur (si nécessaire)
  //motors.flipLeftMotor(true);
  //motors.flipRightMotor(true);
}

void loop()
{
  // accélération : Attend que l'entrée passe à HIGH puis démarre le compteur de temps.
  //    le comtpeur de temps est stoppé dès que le signal passe à LOW
  int throttle = pulseIn(THROTTLE_PIN, HIGH);
  // direction
  int steering = pulseIn(STEERING_PIN, HIGH);

  int left_speed, right_speed;

  if (throttle > 0 && steering > 0)
  {
    // Les deux signaux RC sont correctes; allumer la LED
    digitalWrite(LED_PIN, HIGH);

    // Le signal RC encode l'information sur un signal ayant une largeur d'impulsion "centrale" de 1500 us (microseconds);
    // Soustraire 1500 pour obtenir une valeur centrée sur 0
    throttle -= 1500;
    steering -= 1500;

    // Appliquer la zone morte (deadband) où toute variation du signal est ignoré
    if (abs(throttle) <= PULSE_WIDTH_DEADBAND)
      throttle = 0;
    if (abs(steering) <= PULSE_WIDTH_DEADBAND)
      steering = 0;

    // Combiner les valeurs d'accélération (throttle) et de direction (steering) pour
    // obtenir la vitesse du moteur gauche et du moteur droit
    left_speed = ((long)throttle * MAX_SPEED / PULSE_WIDTH_RANGE) - ((long)steering * MAX_SPEED / PULSE_WIDTH_RANGE);
    right_speed = ((long)throttle * MAX_SPEED / PULSE_WIDTH_RANGE) + ((long)steering * MAX_SPEED / PULSE_WIDTH_RANGE);

    // Limiter les vitesses (borner) à des vitesses max
    left_speed = min(max(left_speed, -MAX_SPEED), MAX_SPEED);
    right_speed = min(max(right_speed, -MAX_SPEED), MAX_SPEED);
  }
  else
  {
    // Au moins un des signaux RC n'est pas correct; Eteindre la la LED et stopper les moteurs
    digitalWrite(LED_PIN, LOW);

    left_speed = 0;
    right_speed = 0;
  }

  // Appliquer les vitesses sur les moteurs
  ZumoMotors::setSpeeds(left_speed, right_speed);
}
```

Détection simplifiée des bordures

Ajouter des senseurs au Zumo lui permet de sentir et réagir au monde qui l'entoure. Dans les compétitions sumo où deux robots essayent de se pousser l'un l'autre hors du ring (ring circulaire), il est important que que le robot puisse détecter le bord du ring afin de ne pas dépasser la frontière. Etant donné que les rings sumo sont noir avec une bordure blanche (tout le long de la frontière), un senseur infrarouge tel que les capteurs QTR lien pololu https://www.pololu.com/category/123/pololu-qtr-reflectance-sensors est particulièrement bien adapté. Le réseau de senseur infrarouge du Zumo lien pololu https://www.pololu.com/product/1419 est équipé de 6 capteurs QTR sur une carte conçue pour prendre place directement sur le connecteur d'extension situé à l'avant du shield Zumo (note: le zumo robot pre-assemblé http://shop.mchobby.be/product.php?id_product=448 lien pololu https://www.pololu.com/product/2510 est déjà équipé avec le réseau de senseur infrarouge).



Un robot Zumo se préparant à attaquer un SumoBot Parallax.
Cliquer l'image pour l'agrandir



Le réseau de senseur infrarouge du Robot Zumo, vue du dessous.
Cliquer l'image pour l'agrandir

Cet exemple montre comment programmer un Robot Zumo Arduino équipé du réseau de senseur infrarouge pour qu'il se balade dans sur un ring sans jamais en sortir. Notez qu'il utilise les deux senseurs situés sur chaque extrémité du réseau de capteur (ce qui est suffisant pour une détection de bordure). Après avoir installé les bibliothèques Arduino du shield Zumo, vous pouvez ouvrir le croquis/sketch en sélectionnant le point de menu **Fichier > Exemples > ZumoExamples > BorderDetect**.

Vous pourriez avoir besoin de modifier quelques éléments du script pour qu'il fonctionne bien avec votre Zumo:

- S'il l'un de vos moteurs est connecté à l'envers, dé-commentez les lignes 48 et/ou 49 pour corriger leur sens de rotation.
- Ajuster la vitesse et les temps de de pause (*duration*) dans lignes 13-17. D'une façon générale, des vitesses plus faibles et temps de pause plus courts fonctionnent mieux avec des moteurs rapides -tandis- que des vitesses plus élevées et temps de pause plus long seront plus adéquat avec des moteurs plus lents.
  Les valeurs par défaut fonctionnent bien avec un Zumo équipé de moteurs 75:1 HP lien pololu https://www.pololu.com/product/2361 .
- Pour finir, le seuil de lecture du senseur (*threshold* en anglais) utilisé pour différencier les surfaces noires et blanches, est définit à la ligne 10. Vous pourriez avoir besoin de changer cette valeur en fonction de votre environnement surface de votre ring). Attention: les surfaces noires lisses/en verre réfléchissent également la lumière infrarouge... il sera donc difficile, voire impossible, de distinguer la surface noire réfléchissante des surfaces blanches.

Téléversez le croquis/sketch sur l'Arduino monté sur le Zumo, Placez votre robot Zumo sur le ring (ou une grade surface noire entouré d'une bordure claire) et pressez le bouton utilisateur. Soyez prêt à rattraper votre Zumo au cas où il quitterait le ring! Si tout fonctionne comme prévu, votre Zumo devrait entamer un décompte sonot (sur le Buzzer) puis commence à se déplacer en ligne droite jusqu'à ce qu'il rencontre le bord du ring; alors, il doit reculer, tourner puis continuer son déplacement.
Essayez d'ajuster les paramètres mentionnés ci-dessus si cela ne fonctionne pas comme prévu.

Voici quelques trucs et astuces de dépannage:

- Si le Zumo dépasse de bord du ring, essayez de diminuer la valeur de FORWARD_SPEED (*vitesse en marche avant*, particulièrement indiqué s'il se déplace rapidement) ou réduire la valeur de QTR_THRESHOLD (*seuil de détection du senseur infrarouge*).
- Si le Zumo s'arrête sur le bord mais tourne trop (ou pas assez) avant de continuer alors ajustez la valeur de TURN_SPEED (*vitesse de rotation*) et/ou TURN_DURATION (*temps de rotation*).
- Si vous n'entendez pas de le son du buzzer, vérifiez que le cavalier de configuration du buzzer est configuré conformément à votre modèle de carte Arduino.

La possibilité de pouvoir errer à l'intérieur du ring sumo est suffisant pour que votre robot Zumo puisse participer à des compétitions de robot sumo. C'est une fonctionnalité rudimentaire, un croquis/sketch plus avancé permettrait la détection de l'opposant pour se diriger directement sur lui. Pour améliorer votre robot Zumo, vous pourriez considérer l'ajout de plusieurs senseurs, comme ceux permettant d'évaluer la distance lien pololu https://www.pololu.com/category/79/sharp-distance-sensors (permettant ainsi de détecter un opposant plutôt que de compter sur la chance de le percuter).

Code

Voici une copie de l'exemple avec traduction des commentaires pour vous aider à mieux comprendre le fonctionnement du croquis/sketch

Nous recommandons de toujours charger l'exemple depuis les codes d'exemples de la bibliothèque Zumo.

```
#include <ZumoBuzzer.h>
#include <ZumoMotors.h>
#include <Pushbutton.h>
#include <QTRSensors.h>
#include <ZumoReflectanceSensorArray.h>

#define LED 13

// Cette valeur aura peut être besoin d'être ajusté pour les
// différentes conditions d'éclairages, surfaces, etc.
#define QTR_THRESHOLD  1500 // microseconds

// A ajuster en en fonction des différents types de moteur
#define REVERSE_SPEED     200 // 0 est stoppé, 400 à pleine vitesse
#define TURN_SPEED        200
#define FORWARD_SPEED     200
#define REVERSE_DURATION  200 // ms
#define TURN_DURATION     300 // ms

ZumoBuzzer buzzer;
ZumoMotors motors;
Pushbutton button(ZUMO_BUTTON); // Bouton poussoir sur la broche 12

#define NUM_SENSORS 6
unsigned int sensor_values[NUM_SENSORS];

ZumoReflectanceSensorArray sensors(QTR_NO_EMITTER_PIN);

# Attendre après le bouton et faire le décompte
void waitForButtonAndCountDown()
{
  digitalWrite(LED, HIGH);
  button.waitForButton();
  digitalWrite(LED, LOW);

  // Jouer un décompte audio
  for (int i = 0; i < 3; i++)
  {
    delay(1000);
    buzzer.playNote(NOTE_G(3), 200, 15);
  }
  delay(1000);
  buzzer.playNote(NOTE_G(4), 500, 15);
  delay(1000);
}

void setup()
{
  // Dé-commenter pour corriger le sens de rotation des moteurs (si nécessaire)
  //motors.flipLeftMotor(true);
  //motors.flipRightMotor(true);

  pinMode(LED, HIGH);

  waitForButtonAndCountDown();
}

void loop()
{
  if (button.isPressed())
  {
    // Si le bouton est pressé, arrêter et attente une seconde
    // pression du bouton pour continuer
    motors.setSpeeds(0, 0);
    button.waitForRelease();
    waitForButtonAndCountDown();
  }


  sensors.read(sensor_values);

  if (sensor_values[0] < QTR_THRESHOLD)
  {
    // Si le senseur le plus à gauche détecte une ligne, reculer et tourner à droite
    motors.setSpeeds(-REVERSE_SPEED, -REVERSE_SPEED);
    delay(REVERSE_DURATION);
    motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
```

```
    delay(TURN_DURATION);
    motors.setSpeeds(FORWARD_SPEED, FORWARD_SPEED);
  }
  else if (sensor_values[5] < QTR_THRESHOLD)
  {
    // Si le senseur le plus à droit détecte une ligne, reculer et tourner à gauche
    motors.setSpeeds(-REVERSE_SPEED, -REVERSE_SPEED);
    delay(REVERSE_DURATION);
    motors.setSpeeds(-TURN_SPEED, TURN_SPEED);
    delay(TURN_DURATION);
    motors.setSpeeds(FORWARD_SPEED, FORWARD_SPEED);
  }
  else
  {
    // Sinon, aller tout droit
    motors.setSpeeds(FORWARD_SPEED, FORWARD_SPEED);
  }
}
```

Collision-detecting sumo robot

Cet exemple étend le projet de détection simplifiée des bordures du robot sumo tel que décrit dans la section précédente. Cet exemple utilise l'accéléromètre du shield Zumo (le LSM303, boussole 3 axes, décrit dans la Section 3.d) pour détecter les collisions. With the Zumo Shield Arduino Libraries installed, the sketch file can be opened in Arduino by selecting **Fichier > Exemples > ZumoExamples > SumoCollisionDetect**.

Cet exemple requière l'installation de la LSM303 https://github.com/pololu/lsm303-arduino%7Cbibliothèque .

Ce croquis/exemple utilise les composantes X et Y du vecteur d'accélération mesuré à l'aide du LSM303 afin détecter une collision avec un robot adversaire. Lorsqu'il détecte un contacte, le Zumo accélère, ce qui lui permet d'éjecter l'opposant hors du ring avec plus d'efficacité. Accélérer permet également d'échapper à l'opposant si c'est ce dernier vient percuter le Zumo sous un angle indésirable (ex: par l'arrière).

Pour en apprendre plus sur le fonctionnement du programme vous pouvez lire les commentaire du croquis SumoCollisionDetect.ino.

Code

Voici une copie de l'exemple avec traduction des commentaires pour vous aider à mieux comprendre le fonctionnement du croquis/sketch.

Nous recommandons de toujours charger l'exemple depuis les codes d'exemples de la bibliothèque Zumo.

```
#include <ZumoBuzzer.h>
#include <ZumoMotors.h>
#include <Pushbutton.h>
#include <QTRSensors.h>
#include <ZumoReflectanceSensorArray.h>
#include <avr/pgmspace.h>
#include <Wire.h>
#include <LSM303.h>

/* Cet exemple utilise l'accéléromètre du shield Zumo (LSM303DLHC) avec la bibliothèque pour
 * détecter les contact avec un robot adversaire sur un ring sumo. La bibliothèque LSM303
 * n'est pas incluse dans dans les bibliothèques du shield Zumo;
 * Elle peut être téléchargée séparément sur GitHub:
 *
 *    https://github.com/pololu/LSM303
 *
 * Cet exemple étend le code de BorderDetect, qui utilise le réseau de senseur du Zumo
 * et bibliothèque pour détecter la bordure du ring sumo ring. Il illustre également l'utilisation
 * des bibliothèques ZumoMotors, PushButton et ZumoBuzzer.
 *
 * Dans la loop(), le programme lit les composantes x et y de l'accélération (ignorant la composante z),
 * et détecte un contact quand la moyenne, sur 3 périodes, de la magnitude du vecteur x-y
 * excède la valeur du seuil XY_ACCELERATION_THRESHOLD (déterminé de façon empirique).
 * Lors de la détection d'une collision, la vitesse en marche avant est augmenté de la
 * valeur par défaut SEARCH_SPEED (vitesse de recherche) à à  FULL_SPEED (pleine vitesse).
 * Cela permet de simuler une réponse de type "fight or flight" (frapper ou s'échapper).
 *
 * Le programme essayer de détecter le contact uniquement lorsque le Zumo avance en ligne droite.
 * Lorsque le Zumo tourne après une détection de la bordure du ring, la fait de tourner génère une
 * accélération sur le plan x-y, ce qui rend l'interprétation des lectures difficiles (entre
 * rotation volontaire et détection de collision).
 * Etant donné que le Zumo accélère également lorsqu'il redémarre en ligne droit, les lectures
 * d'accélération sont également ignorés durant la période de MIN_DELAY_AFTER_TURN millisecondes
 * après avoir terminé sa rotation. (MIN_DELAY_AFTER_TURN : delais minimum après rotation)
 *
 * Pour éviter d'autres faux positifs, une valeur MIN_DELAY_BETWEEN_CONTACTS est également
 * spécifiée. (MIN_DELAY_BETWEEN_CONTACTS : délais minimum entre contacts)
 *
 * Cet exemple contient également les améliorations suivantes:
 *
 *  - Utilise la bibliothèque Zumo Buzzer pour jouer un effet sonore (Mélodie de "charge") au
 *    début de la compétition et lorsqu'un contact est fait l'adversaire.
 *
 *  - L'angle de rotation (lors de la détection des boards) est choisit au hasard. Ce qui permet
 *    au Zumo d'avoir un schéma de recherche plus efficace.
 *
 *  - Support de FULL_SPEED_DURATION_LIMIT (temps max à pleine vitesse), permettant au robot de
 *    passer à la vitesse SUSTAINED_SPEED (vitesse soutenue) après une courte période de
 *    mouvement à la vitesse FULL_SPEED (pleine vitesse). Dans cet exemple, les deux vitesses sont
 *    fixées au Maximum de 400, mais cette fonctionnalité peut être utile pour éviter de déraper
 *    au moment de tourner (sur le bord du ring) si la surface du ring est anormalement lisse.
 *
 *  - Log des données de l'accéléromètre vers le moniteur sérue lorsque LOG_SERIAL est #defined.
 *
```

```
 *  Cette exemple utilise également la bibliothèque RunningAverage (domaine publique) du site Arduino;
 *  Le code utile a été copié dans ce fichier .ino. Vous n'avez donc pas besoin de le
 *  télécharger séparément .
 */

// #define LOG_SERIAL // écrire la sortie du log vers le port série

#define LED 13
Pushbutton button(ZUMO_BUTTON); // bouton poussoir sur la broche 12

// Paramètre de l'accéléromètre
#define RA_SIZE 3  // Nombre de lectures à inclure dans la moyenne de accélération durant la phase de déplacement (running average of accelerometer readings)
#define XY_ACCELERATION_THRESHOLD 2400  // pour la détection de contact (~16000 = mangitude de l'accélération du à la gravité)

// Configuration du réseau senseur infrarouge (Reflectance Sensor)
#define NUM_SENSORS 6
unsigned int sensor_values[NUM_SENSORS];
// Ce paramètre pourrait avoir besoin d'être mis au point en fonction des conditions de luminosité, surface, etc.
#define QTR_THRESHOLD  1500 // microseconds
ZumoReflectanceSensorArray sensors(QTR_NO_EMITTER_PIN);

// Configuration moteur
ZumoMotors motors;

// Ces valeurs pourrait avoir besoin d'être mis-au-pnt en fonction des différents types de moteur
#define REVERSE_SPEED     200 // 0 = à l'arrêt, 400  = pleine vitesse
#define TURN_SPEED        200
#define SEARCH_SPEED      200
#define SUSTAINED_SPEED   400 // Vitesse soutenue SUSTAINED_SPEED après la vitesse FULL_SPEED appliquée durant FULL_SPEED_DURATION_LIMIT ms
#define FULL_SPEED        400
#define STOP_DURATION     100 // ms
#define REVERSE_DURATION  200 // ms
#define TURN_DURATION     300 // ms

#define RIGHT 1
#define LEFT -1

enum ForwardSpeed { SearchSpeed, SustainedSpeed, FullSpeed };
ForwardSpeed _forwardSpeed;  // Configuration de la vitesse actuelle
unsigned long full_speed_start_time;
#define FULL_SPEED_DURATION_LIMIT     250  // ms

// Effet sonore
ZumoBuzzer buzzer;
const char sound_effect[] PROGMEM = "O4 T100 V15 L4 MS g12>c12>e12>G6>E12 ML>G2"; // Mélodie de "charge"
 // Utiliser V0 pour supprimer le son; v15 pour volume max

 // Minutage
unsigned long loop_start_time;
unsigned long last_turn_time;
unsigned long contact_made_time;
#define MIN_DELAY_AFTER_TURN          400  // ms = délais minimum avant la détection d'un événement collision
#define MIN_DELAY_BETWEEN_CONTACTS   1000  // ms = délais minimum entre avant la détection d'un nouvel événement collision

// classe RunningAverage
// Basé sur la bibliothèque RunningAverage pour Arduino
// source:  http://playground.arduino.cc/Main/RunningAverage
template <typename T>
class RunningAverage
{
  public:
    RunningAverage(void);
    RunningAverage(int);
    ~RunningAverage();
    void clear();
    void addValue(T);
    T getAverage() const;
    void fillValue(T, int);
  protected:
    int _size;
    int _cnt;
    int _idx;
    T _sum;
    T * _ar;
    static T zero;
};

// Classe Accelerometer -- étend la bibliothèque LSM303 pour supporter la lecture et moyenne des vecteurs
// d'accélérations x-y provenant de l'accéléromètre/magnétomètre LSM303DLHC.
class Accelerometer : public LSM303
{
  typedef struct acc_data_xy
  {
    unsigned long timestamp;
    int x;
    int y;
    float dir;
```

```
  } acc_data_xy;

  public:
    Accelerometer() : ra_x(RA_SIZE), ra_y(RA_SIZE) {};
    ~Accelerometer() {};
    void enable(void);
    void getLogHeader(void);
    void readAcceleration(unsigned long timestamp);
    float len_xy() const;
    float dir_xy() const;
    int x_avg(void) const;
    int y_avg(void) const;
    long ss_xy_avg(void) const;
    float dir_xy_avg(void) const;
  private:
    acc_data_xy last;
    RunningAverage<int> ra_x;
    RunningAverage<int> ra_y;
};

Accelerometer lsm303;
boolean in_contact;  // activé lorsque l'accéléromètre détecte un contact avec le robot opposé

// pré-déclaration de setForwardSpeed - vitesse en marche avant
void setForwardSpeed(ForwardSpeed speed);

void setup()
{
  // Initialise la bibliothèque Wire et joindre le bus I2C comme maître
  Wire.begin();

  // Initialiser le LSM303
  lsm303.init();
  lsm303.enable();

#ifdef LOG_SERIAL
  Serial.begin(9600);
  lsm303.getLogHeader();
#endif

  randomSeed((unsigned int) millis());

  // Dé-commentez les ligne pour corriger le sens de rotation des moteurs (si nécessaire)
  //motors.flipLeftMotor(true);
  //motors.flipRightMotor(true);

  pinMode(LED, HIGH);
  buzzer.playMode(PLAY_AUTOMATIC);
  waitForButtonAndCountDown(false);
}

void waitForButtonAndCountDown(bool restarting)
{
#ifdef LOG_SERIAL
  Serial.print(restarting ? "Restarting Countdown" : "Starting Countdown");
  Serial.println();
#endif

  digitalWrite(LED, HIGH);
  button.waitForButton();
  digitalWrite(LED, LOW);

  // jouer le décompte audio
  for (int i = 0; i < 3; i++)
  {
    delay(1000);
    buzzer.playNote(NOTE_G(3), 50, 12);
  }
  delay(1000);
  buzzer.playFromProgramSpace(sound_effect);
  delay(1000);

  // Réinitialiser les variables de loop()
  in_contact = false;  // 1 si collision; 0 si pas de collision (ou perte de contact)
  contact_made_time = 0;
  last_turn_time = millis();  // Evite les fausses détections de collision durant l'accélération initiale
  _forwardSpeed = SearchSpeed;
  full_speed_start_time = 0;
}

void loop()
{
  if (button.isPressed())
  {
    // Si le bouton est pressé, arrêter et attente une autre pression sur le bouton
    motors.setSpeeds(0, 0);
    button.waitForRelease();
```

```
  waitForButtonAndCountDown(true);
 }

 loop_start_time = millis();
 lsm303.readAcceleration(loop_start_time);
 sensors.read(sensor_values);

 if ((_forwardSpeed == FullSpeed) && (loop_start_time - full_speed_start_time > FULL_SPEED_DURATION_LIMIT))
 {
  setForwardSpeed(SustainedSpeed);
 }

 if (sensor_values[0] < QTR_THRESHOLD)
 {
  // Si le senseur le plus à gauche détecte une ligne, marche arrière et tourner à droite
  turn(RIGHT, true);
 }
 else if (sensor_values[5] < QTR_THRESHOLD)
 {
  // Si le senseur le plus a droite détecte une ligne, marche arrière et tourner à gauche
  turn(LEFT, true);
 }
 else  // Sinon, aller tout droit
 {
  // vérifier si contact; signaler contact
  if (check_for_contact()) on_contact_made();
  int speed = getForwardSpeed();
  motors.setSpeeds(speed, speed);
 }
}

// Exécuter une rotation
// direction:  RIGHT (droite) ou LEFT (gauche)
// Angle aléatoire pour améliorer la recherche
void turn(char direction, bool randomize)
{
#ifdef LOG_SERIAL
 Serial.print("turning ...");
 Serial.println();
#endif

 // assume qu'il n'y a pas de contact (pas de collision).
 on_contact_lost();

 static unsigned int duration_increment = TURN_DURATION / 4;

 // motors.setSpeeds(0,0);
 // delay(STOP_DURATION);
 motors.setSpeeds(-REVERSE_SPEED, -REVERSE_SPEED);
 delay(REVERSE_DURATION);
 motors.setSpeeds(TURN_SPEED * direction, -TURN_SPEED * direction);
 delay(randomize ? TURN_DURATION + (random(8) - 2) * duration_increment : TURN_DURATION);
 int speed = getForwardSpeed();
 motors.setSpeeds(speed, speed);
 last_turn_time = millis();
}

void setForwardSpeed(ForwardSpeed speed)
{
 _forwardSpeed = speed;
 if (speed == FullSpeed) full_speed_start_time = loop_start_time;
}

int getForwardSpeed()
{
 int speed;
 switch (_forwardSpeed)
 {
  case FullSpeed:
   speed = FULL_SPEED;
   break;
  case SustainedSpeed:
   speed = SUSTAINED_SPEED;
   break;
  default:
   speed = SEARCH_SPEED;
   break;
 }
 return speed;
}

// Vérifie s'il y a une collision/contact, mais ignore les lectures immédiatement après une rotation ou une perte de contact
bool check_for_contact()
{
 static long threshold_squared = (long) XY_ACCELERATION_THRESHOLD * (long) XY_ACCELERATION_THRESHOLD;
 return (lsm303.ss_xy_avg() >  threshold_squared) && \
  (loop_start_time - last_turn_time > MIN_DELAY_AFTER_TURN) && \
```

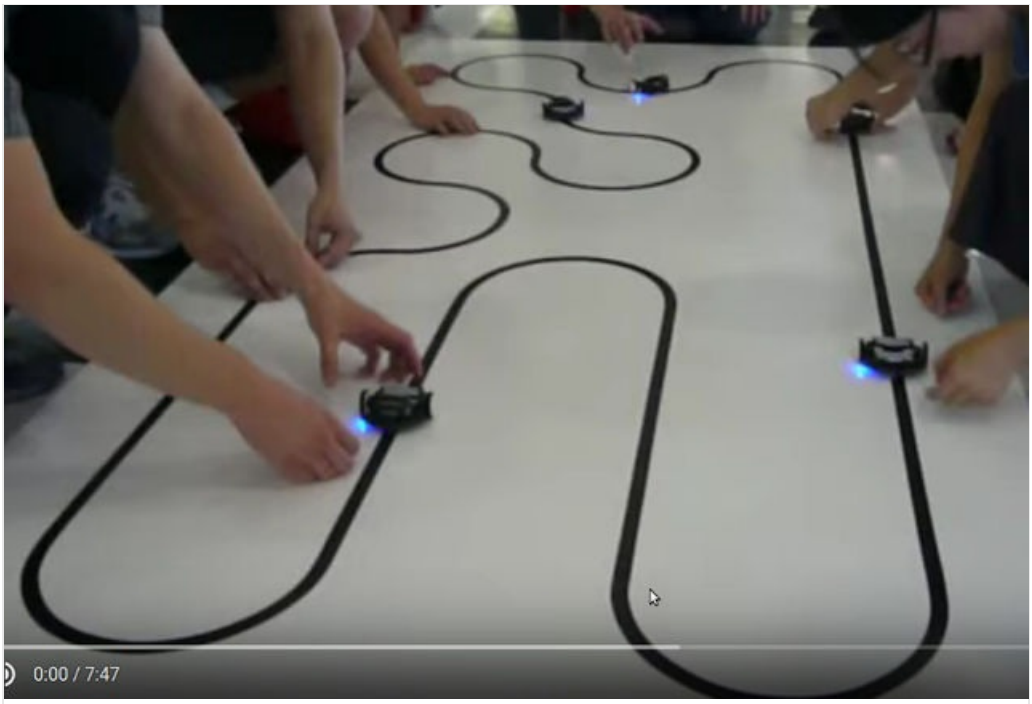
```
    (loop_start_time - contact_made_time > MIN_DELAY_BETWEEN_CONTACTS);
}

// Klaxon et accélération au contact/collision -- frapper ou s'échapper (fight or flight)
void on_contact_made()
{
#ifdef LOG_SERIAL
  Serial.print("contact made");
  Serial.println();
#endif
  in_contact = true;
  contact_made_time = loop_start_time;
  setForwardSpeed(FullSpeed);
  buzzer.playFromProgramSpace(sound_effect);
}

// Réinitialiser la vitesse (de la marche avant)
void on_contact_lost()
{
#ifdef LOG_SERIAL
  Serial.print("contact lost");
  Serial.println();
#endif
  in_contact = false;
  setForwardSpeed(SearchSpeed);
}

// classe Accelerometer -- définition des fonctions membres

// Activer uniquement l'accéléromètre
// Appeler enableDefault() pour activer l'accéléromètre et le magnétomètre.
void Accelerometer::enable(void)
{
  // Activer l'accéléromètre
  // 0x27 = 0b00100111
  // Mode d'alimentation normal, activer tous les axes
  writeAccReg(LSM303::CTRL_REG1_A, 0x27);

  if (getDeviceType() == LSM303::device_DLHC)
  writeAccReg(LSM303::CTRL_REG4_A, 0x08); // DLHC: activer le mode haute résolution
}

void Accelerometer::getLogHeader(void)
{
  Serial.print("millis    x      y     len     dir  | len_avg  dir_avg  |  avg_len");
  Serial.println();
}

void Accelerometer::readAcceleration(unsigned long timestamp)
{
  readAcc();
  if (a.x == last.x && a.y == last.y) return;

  last.timestamp = timestamp;
  last.x = a.x;
  last.y = a.y;

  ra_x.addValue(last.x);
  ra_y.addValue(last.y);

#ifdef LOG_SERIAL
  Serial.print(last.timestamp);
  Serial.print("  ");
  Serial.print(last.x);
  Serial.print("  ");
  Serial.print(last.y);
  Serial.print("  ");
  Serial.print(len_xy());
  Serial.print("  ");
  Serial.print(dir_xy());
  Serial.print("  |  ");
  Serial.print(sqrt(static_cast<float>(ss_xy_avg())));
  Serial.print("  ");
  Serial.print(dir_xy_avg());
  Serial.println();
#endif
}

float Accelerometer::len_xy() const
{
  return sqrt(last.x*a.x + last.y*a.y);
}

float Accelerometer::dir_xy() const
{
  return atan2(last.x, last.y) * 180.0 / M_PI;
}
```

```
int Accelerometer::x_avg(void) const
{
  return ra_x.getAverage();
}

int Accelerometer::y_avg(void) const
{
  return ra_y.getAverage();
}

long Accelerometer::ss_xy_avg(void) const
{
  long x_avg_long = static_cast<long>(x_avg());
  long y_avg_long = static_cast<long>(y_avg());
  return x_avg_long*x_avg_long + y_avg_long*y_avg_long;
}

float Accelerometer::dir_xy_avg(void) const
{
  return atan2(static_cast<float>(x_avg()), static_cast<float>(y_avg())) * 180.0 / M_PI;
}



// Classe RunningAverage
// basé sur la bibliothèque RunningAverage d'Arduino
// source:  http://playground.arduino.cc/Main/RunningAverage
// autheur: Rob.Tillart@gmail.com
// public domain

template <typename T>
T RunningAverage<T>::zero = static_cast<T>(0);

template <typename T>
RunningAverage<T>::RunningAverage(int n)
{
  _size = n;
  _ar = (T*) malloc(_size * sizeof(T));
  clear();
}

template <typename T>
RunningAverage<T>::~RunningAverage()
{
  free(_ar);
}

// Réinitialiser tous les compteurs
template <typename T>
void RunningAverage<T>::clear()
{
  _cnt = 0;
  _idx = 0;
  _sum = zero;
  for (int i = 0; i< _size; i++) _ar[i] = zero;  // Nécessaire pour maintenir addValue simple
}

// Ajouter une nouvelle valeur au data-set (ensemble des données)
template <typename T>
void RunningAverage<T>::addValue(T f)
{
  _sum -= _ar[_idx];
  _ar[_idx] = f;
  _sum += _ar[_idx];
  _idx++;
  if (_idx == _size) _idx = 0;  // plus rapide que %
  if (_cnt < _size) _cnt++;
}

// Retourne la moyenne des valeurs du data-set
template <typename T>
T RunningAverage<T>::getAverage() const
{
  if (_cnt == 0) return zero; // NaN ?  math.h
  return _sum / _cnt;
}

// Rempli la moyenne avec une valeur
// Le paramètre number détermine ne nombre de fois que la valeur est ajouté (poids)
// number devrait être entre 1 et size (la taille)
template <typename T>
void RunningAverage<T>::fillValue(T value, int number)
{
  clear();
  for (int i = 0; i < number; i++)
  {
```

```
    addValue(value);
  }
}
```

Suiveur de ligne

Ce croquis/sketch montre comment le programmer le Zumo et son réseau de senseur lien pololu https://www.pololu.com/product/1419 pour suivre des lignes et participer à des courses de suivit de ligne (également appelé "**Line Tracker**" en anglais). Une fois les bibliothèques du shield Zumo installées, vous pouvez ouvrir l'exemple depuis le point de menu **Fichier > Exemples > ZumoExamples > LineFollower**.



Exemple de circuit "suiveur de ligne"

Cliquer l'image pour l'agrandir

Vous pouvez également consulter cette **vidéo YouTube** de Pololu https://youtu.be/fl0CJhPiEfY utilisant le 3Pi de Pololu mais dont le principe est identique avec notre Zumo.

Cette implémentation du suiveur de ligne est très similaire à l'exemple de Pololu pour le robot 3pi lien pololu https://www.pololu.com/product/975 . Les concepts et stratégies mises en oeuvre sont expliquées en détails dans la section 7 https://www.pololu.com/docs/0J21/7 du utilisateur du robot 3pi https://www.pololu.com/docs/0J21%7Cguide (*pololu, anglais*).

Des exemples de compétitions

Si le sujet vous captive, vous trouverez de nombreux exemple sur YouTube en faisant une recherche sur "**line following race**".

Le code

Voici une copie de l'exemple avec traduction des commentaires pour vous aider à mieux comprendre le fonctionnement du croquis/sketch

Nous recommandons de toujours charger l'exemple depuis les codes d'exemples de la bibliothèque Zumo.

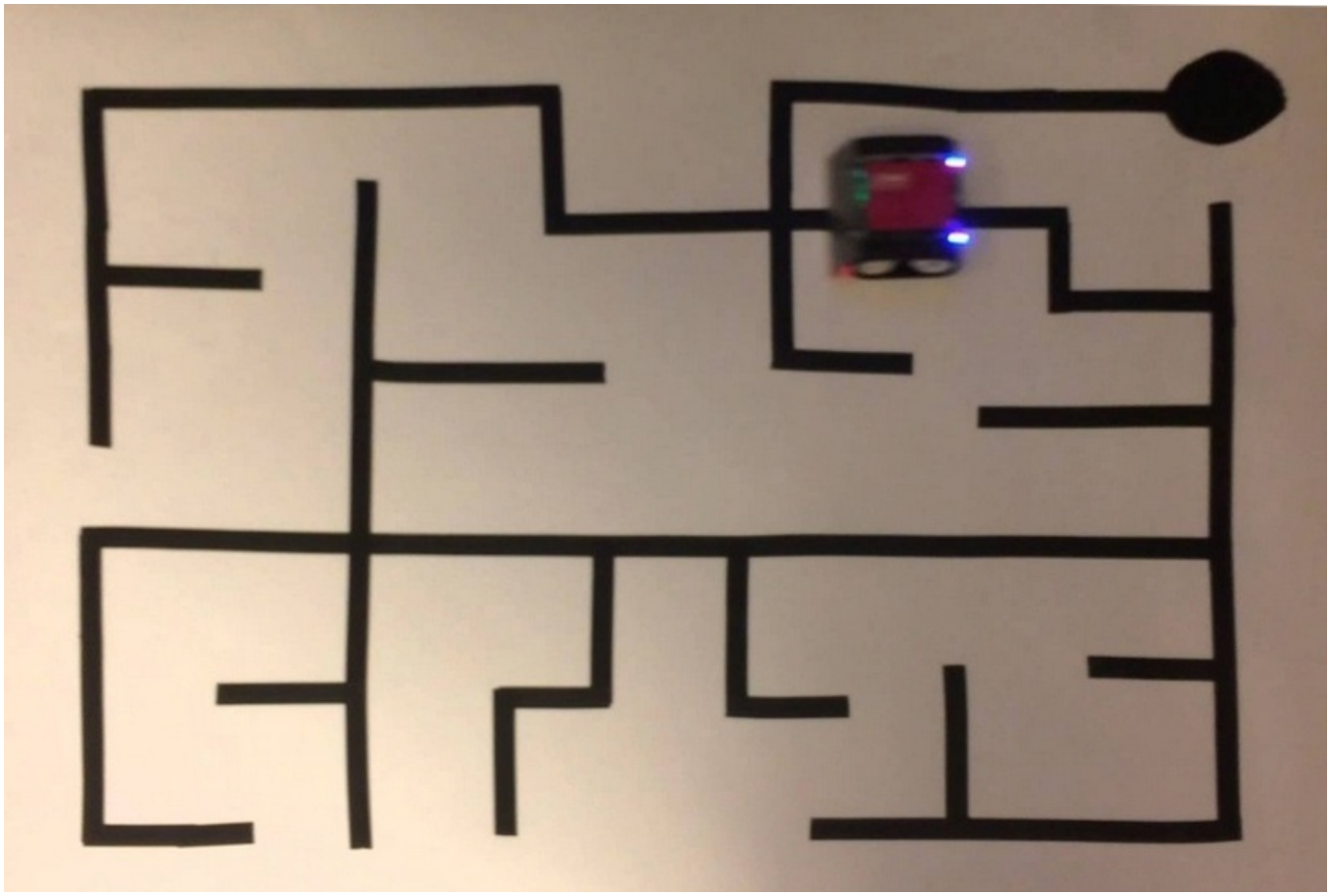
```
/*
 * Code de démo suiveur de ligne (line-following) pour le Robot Zumo de Polulo
 *
 * Ce code suivra une ligne noire sur un fond blan et utilise un
 * algorithme de type PID.  Il fonctionne correctement sur des
 * circuits avec des courbes ayant un rayon de 15 cm.
 * L'algorithme à été testé sur Zumo avec des moteurs 30:1 HP et
 * 75:1 HP.  Pourrait demander des modifications pour fonctionner
 * sur d'autres circuits ou avec d'autres moteurs.
 *
 * http://www.pololu.com/catalog/product/2506
 * http://www.pololu.com
 * http://forum.pololu.com
 *
 * Zumo également disponible chez MC Hobby (le traducteur du tutoriel)
 *
 * http://shop.mchobby.be/product.php?id_product=448
 */
```

```
#include <QTRSensors.h>
#include <ZumoReflectanceSensorArray.h>
#include <ZumoMotors.h>
#include <ZumoBuzzer.h>
#include <Pushbutton.h>


ZumoBuzzer buzzer;
ZumoReflectanceSensorArray reflectanceSensors;
ZumoMotors motors;
Pushbutton button(ZUMO_BUTTON);
int lastError = 0;

// Ceci est la vitesse de rotation maximale des moteurs.
// (400 permet au moteur d'aller a vitesse max; diminuer la valeur pour imposer une vitesse limite)
const int MAX_SPEED = 400;


void setup()
{
  // Jouer une petite chanson d'acceuil
  buzzer.play(">g32>>c32");

  // Initialise le réseau de senseur infrarouge
  reflectanceSensors.init();

  // Attendre que le bouton utilisateur soit pressé et relâché
  button.waitForButton();

  // Allumer la LED et pour indiquer que l'on est en mode calibration
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);

  // Attendre 1 seconde puis démarrer la calibration automatique
  // du senseur en faisant des rotations sur place pour faire passer
  // le senseur au dessus de la ligne
  delay(1000);
  int i;
  for(i = 0; i < 80; i++)
  {
    if ((i > 10 && i <= 30) || (i > 50 && i <= 70))
      motors.setSpeeds(-200, 200);
    else
      motors.setSpeeds(200, -200);
    reflectanceSensors.calibrate();

    // Puisque notre compteur va jusque 80, le délais total sera de
    // 80*20 = 1600 ms.
    delay(20);
  }
  motors.setSpeeds(0,0);

  // Eteindre la LED pour indiquer que nous avons terminé la calibration
  digitalWrite(13, LOW);
  buzzer.play(">g32>>c32");

  // Attendre que le bouton utilisateur soit pressé et relâché
  button.waitForButton();

  // Jouer la musique et attendre qu'elle soit finie pour
  // commencer le pilotage.
  buzzer.play("L16 cdegreg4");
  while(buzzer.isPlaying());
}

void loop()
{
  unsigned int sensors[6];

  // Obtenir la position de la ligne.  Notez qu'il FAUT fournit le senseur "sensors"
  // en argument à la fonction readLine(), même si nous ne sommes intéressé
  // par les lectures individuelles des différents senseurs.
  int position = reflectanceSensors.readLine(sensors);

  // L'erreur ("error") est la distance par rapport au centre de la ligne, qui
  // correspond à la position 2500.
  int error = position - 2500;

  // Calculer la différence de vitesse (speedDifference) entre les moteurs
  // en utilisant un les termes proportionnels et dérivés du régulateur PID.
  // (Le terme intégral n'est généralement pas très utile dans le
  // suivit de ligne).
  // Nous utiliserons 1/4 pour la constante proportionnelle et 6 pour la
  // constante dérivée 6, qui devrait fonctionner correctement avec de
  // nombreux choix de Zumo.
  // Vous aurez probablement besoin d'ajuster ces constantes par
  // essai/erreur pour votre zumo et/ou le circuit.
```

```
  int speedDifference = error / 4 + 6 * (error - lastError);

  lastError = error;

  // Calculer la vitesse de chaque moteur. Le signe de la différence (speedDifference)
  // determine si le moteur tourne à gauche ou a droite.
  int m1Speed = MAX_SPEED + speedDifference;
  int m2Speed = MAX_SPEED - speedDifference;

  // Nous allons contraindre la vitesse des moteurs entre 0 et MAX_SPEED.
  // D'une façon générale, un des moteurs est toujours à MAX_SPEED
  // et l'autre sera à MAX_SPEED-|speedDifference| si elle est positif,
  // sinon il sera en vitesse stationnaire. Pour certaines applications,
  // vous pourriez désirer une vitesse négative, ce qui permettrai de
  // tourner à l'envers.
  if (m1Speed < 0)
    m1Speed = 0;
  if (m2Speed < 0)
    m2Speed = 0;
  if (m1Speed > MAX_SPEED)
    m1Speed = MAX_SPEED;
  if (m2Speed > MAX_SPEED)
    m2Speed = MAX_SPEED;

  motors.setSpeeds(m1Speed, m2Speed);
}
```

Résolution de labyrinthe

Ce croquis/sketch de démonstration montre comment utiliser le réseau de senseur infrarouge lien pololu https://www.pololu.com/product/1419 pour résoudre un labyrinthe (labyrinthe constitué de lignes). Une fois les bibliothèques Arduino du shield Zumo installée, le croquis/sketch peut être ouvert en sélectionnant le point de menu **Fichier > Exemples > ZumoExamples > MazeSolver**.



Source: Pololu Zumo Robot Maze Solving par Soon Min Park, voyez également sa **vidéo sur YouTube** https://youtu.be/QuHL17jx_7g .

Cette implémentation de la résolution de labyrinthe (*maze solver* en anglais) est très similaire et à l'exemple du robot 3pi lien pololu https://www.pololu.com/product/975 de Pololu. Les concepts et stratégies mises en oeuvres sont expliqués en détail dans la section 8 https://www.pololu.com/docs/0J21/8 du guide utilisateur du robot 3pi https://www.pololu.com/docs/0J21 (*pololu, anglais*).

Utiliser la boussole

Ce croquis/sketch d'exemple démontre l'utilisation du magnétomètre du shield Zumo (le module boussole 3-axes LSM303, décrit dans la section Section 3.d) pour aider le Zumo à coordonner ses moteurs pour faire des rotations de 90° et se déplacer en carré. Une fois la bibliothèque Arduino du shield Zumo installée, le croquis/sketch Arduino peut être ouvert via le point de menu **Fichier > Exemples > ZumoExamples > Compass**. Cet exemple requière l'installation de la bibliothèque LSM303 https://github.com/pololu/lsm303-arduino .

Les piles, le moteur, les courants dans le moteurs affecte les mesures sur l'axe Z du magnétomètre (dans une proportion nettement plus importante que pour les axes X et Y). Pour cette raison, le programme calcule l'orientation du Zumo en utilisant uniquement les lectures sur l'axe X et Y du magnétomètre; le programme part du principe que le Zumo reste au même niveau. Avant chaque rotation, le programme fait un relevé magnétique (de la direction actuelle du Zumo) puis fait une rotation *relative* de 90°. Le relevé juste avant rotation permet de se prévenir de l'influence des sources externes, variations/perturbations locales du champs magnétique (ex: produit par les barres d'aciers dans les bétons armés).

Contrôler un servo

Cette section explique comment contrôler un servo moteur hobby RC servo http://shop.mchobby.be/category.php?id_category=74 lien pololu https://www.pololu.com/category/23/rc-servos depuis un Arduino Uno, Arduino Leonardo ou A-Star 32U4 Prime connecté sur le Zumo Shield.

L'environnement Arduino IDE inclus une bibliothèque Servo http://arduino.cc/en/Reference/Servo qui génère les impulsions nécessaires au contrôle d'un servo moteur. Cette bibliothèque est cependant en conflit avec la bibliothèque ZumoMotors étant donné que les deux bibliothèques utilisent le Timer 1. Il est donc nécessaire de faire quelque-chose de spécial pour faire fonctionner des servo moteurs.

Voyez la section section 8.a pour contrôler un servo moteur avec un Arduino Uno . La Section 8.b se penchera sur le contrôle d'un servo avec un Arduino Leonardo ou A-Star 32U4 Prime.

- Contrôler un servo avec Arduino Uno
- Contrôler un servo avec Arduino Leonardo ou A-Star 32U4 Prime

Contrôler un servo avec Arduino Uno

Le code d'exemple Arduino Uno ci-dessous montre comment contrôler un simple servo en utilisant le Timer 2. Nous utilisons le Timer 2 à la place du Timer 1 parce que le Timer 1 est utilisé par la bibliothèque ZumoMotors (ce qui interfère avec la bibliothèque servo d'Arduino). Il nous faut donc gérer notre servo moteur nous même avec un autre timer (le Timer 2).

Utiliser le Timer 2 interfère avec la bibliothèque ZumoBuzzer, il ne sera donc pas possible d'utiliser le Buzzer et le Timer 2 en même temps. Vous pouvez intégrer ce code avec d'autre codes pilotant des moteurs.

```
/** Exemple Servo Timer 2 sur Arduino Uno
Ce code d'exemple est pour un Arduino Uno et montre comment utiliser le
Timer 2 d'un ATmega328P pour controler un simple servo moteur.
Cela est tule pour les gens qui ne peuvent pas utiliser la
bibliotheque Servo d'Arduino IDE.
Par exemple, la bibliotheque ZumoMotors utilise le même timer que
la bibliotheque Servo (Timer 1). Il y a donc un conflit.

La macro SERVO_PIN ci-dessous spécifie la broche sur laquelle
est branché le servo moteur. Cette broche doit être raccordée
sur l'entrée signal du servo. La masse du servo doit être connecté
sur la masse de votre Arduino (masse commune). Pour finir
la masse et broche d'alimentation du servo doivent être connecté
sur une source d'alimentation adéquate.
*/

// Cette ligne indique quel est la broche utilisée pour envoyer
// le signal au servo moteur. Vous pouvez changer cette valeur.
#define SERVO_PIN 11

// Le temps depuis le dernier flan montage (rising edge) en unités de 0.5us.
uint16_t volatile servoTime = 0;

// Contient la largeur d impulsion désirée en unités de 0.5us.
uint16_t volatile servoHighTime = 3000;

// Vrai (true) si la broche servo est actuellement au niveau haut.
boolean volatile servoHigh = false;

void setup()
{
  servoInit();
}

void loop()
{
  servoSetPosition(1000);  // Envoi une impulsion de 1000 us.
  delay(1000);
  servoSetPosition(2000);  // Envoi une impulsion de 2000 us.
  delay(1000);
}

// Cette routine d'interruption (ISR) est appelée après que
// le Timer 2 ait atteind OCR2A et resets.
// Dans cet ISR, nous initialisons OCR2A pour déterminer
// quand la prochaine interruption est déclenchée.
// Généralement, nous fixons OCR2A à 255 pour avoir une
// interruption toutes les 128 us, mais les deux premiers
// intervales d'interruption après le flan montant (rising edge)
// seront plus petit ce qui nous empêche d'atteindre la
// largeur d'impulsion désirée.
ISR(TIMER2_COMPA_vect)
{
  // Le temps qui est passé depuis la dernière interruption est
  // égale à OCR2A + 1 parce que la valeur du timer sera égale
  // à OCR2A avant de revenir à 0.
  servoTime += OCR2A + 1;

  static uint16_t highTimeCopy = 3000;
  static uint8_t interruptCount = 0;

  if(servoHigh)
  {
    if(++interruptCount == 2)
    {
      OCR2A = 255;
    }

    // La broche du signal Servo est au niveau haut (high).
    // Vérifier s'il est temps d'avoir le flan descendant
    // du signal (falling edge).
    // Note: Nous pouvons utiliser == à la place de >=.
    if(servoTime >= highTimeCopy)
```

```
    {
      // La broche est au niveau haut depuis assez longtemps
      // donc on applique la flan descendant.
      digitalWrite(SERVO_PIN, LOW);
      servoHigh = false;
      interruptCount = 0;
    }
  }
  else
  {
    // La broche du servo est au niveau bas (low).

    if(servoTime >= 40000)
    {
      // Nous avons atteind la fin de la période (20 ms),
      // Donc on applique un flan montant (rising edge) sur
      // le signal.
      highTimeCopy = servoHighTime;
      digitalWrite(SERVO_PIN, HIGH);
      servoHigh = true;
      servoTime = 0;
      interruptCount = 0;
      OCR2A = ((highTimeCopy % 256) + 256)/2 - 1;
    }
  }
}

void servoInit()
{
  digitalWrite(SERVO_PIN, LOW);
  pinMode(SERVO_PIN, OUTPUT);

  // Passe en mode CTC. Le Timer 2 comptera jusqu'à OCR2A,
  // puis se réinitialise à 0 et provoque une interruption.
  TCCR2A = (1 << WGM21);
  // Initialiser le pre-scaler sur 1:8 . ce qui offre une
  // résolution de 0.5us.
  TCCR2B = (1 << CS21);

  // Mettre le timer dans le bon état par défaut.
  TCNT2 = 0;
  OCR2A = 255;

  TIMSK2 |= (1 << OCIE2A);  // Activer le "timer compare interrupt".
  sei();   // Activer les interruptions.
}

void servoSetPosition(uint16_t highTimeMicroseconds)
{
  TIMSK2 &= ~(1 << OCIE2A); // Désactiver le "timer compare interrupt"
  servoHighTime = highTimeMicroseconds * 2;
  TIMSK2 |= (1 << OCIE2A); // Activer le "timer compare interrupt"
}
```

Contrôler un servo avec Arduino Leonardo ou A-Star 32U4 Prime

Il est possible de modifier la bibliothèque Servo distribué avec Arduino IDE pour utiliser le Timer 3 à la place du Timer 1 sur Arduino Leonardo ou A-Star 32U4 Prime. La bibliothèque Servo modifiée n'interfère pas avec la bibliothèque ZumoMotors permettant de contrôler simultanément des servos et les moteurs.

Attention: les modifications décrites ici affecte tous les croquis/sketchs Arduino Leonardo ou A-Star utilisant la bibliothèque Servo.

1. Pour commencer, vous aurez besoin de localiser l'emplacement de votre bibliothèque Servo d'Arduino IDE PUIS localiser le fichier nommé ServoTimers.h. Dans Arduino IDE 1.6.x ce fichier peut être trouvé dans da le répertoire libraries/Servo/src/avr/ServoTimers.h . Si vous utiliser Mac OS X, il sera nécessaire de faire un clique droit sur l'icône Arduino IDE puis sélectionner "Show Package Contents" pour vois les fichiers à l'intérieur.

2. Ouvrez le fichier ServoTimers.h avec un éditeur de texte.

3. Cherchez les lignes suivantes dans le code de ServoTimers.h:

```
#elif defined(__AVR_ATmega32U4__)
#define _useTimer1
typedef enum { _timer1, _Nbr_16timers } timer16_Sequence_t ;
```

4. Les deux dernières lignes de code indique la bibliothèque devrait utiliser le Timer 1.
Pour utiliser le Timer 3 à la place, changer simplement le _useTimer1 vers _useTimer3 -ET- _timer1 vers _timer3.

5. Sauvez le fichier.

Arduino IDE intégrera automatiquement vos modifications dans la bibliothèque Servo. La prochaine fois que vous compilerez le croquis/sketch pour un Arduino Leonardo ou A-Star, la bibliothèque Servo utilisera le Timer 3 à la place du Timer 1.
