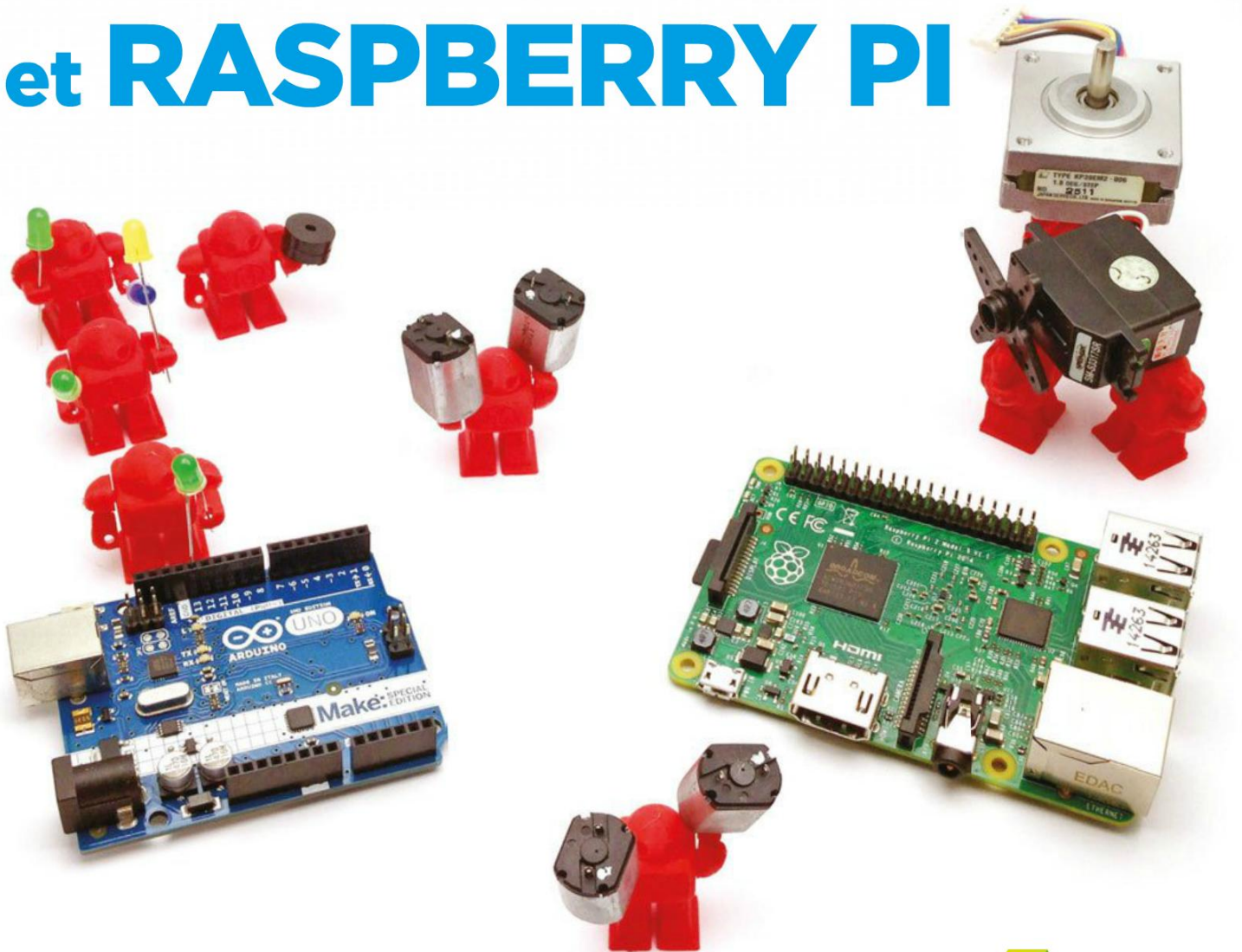
Simon Monk

Avec 30 projets ludiques

Mouvement, lumière et son avec ARDUINO et RASPBERRY PI



EYROLLES

SERIAL MAKERS

À l'action avec Arduino et Raspberry Pi !

Cet ouvrage à vocation pratique explique comme créer et contrôler des mouvements, de la lumière et du son à l'aide d'un Arduino et d'un Raspberry Pi. Avec à l'appui 30 projets ludiques à réaliser, il détaille comment utiliser ces deux plates-formes pour contrôler des LED, des moteurs de divers types, des bobines, des dispositifs à courant alternatif, des pompes, ou encore des systèmes d'affichage ou de production de sons. Il se clôt par des projets permettant de contrôler des mécanismes et des systèmes avec Internet, faisant ainsi pénétrer le lecteur dans le monde des objets connectés. Le maker, qui aura déjà eu l'occasion d'utiliser un Arduino ou un Raspberry Pi pour mesurer le monde réel à l'aide de capteurs, passera ici à l'action en découvrant les bases de l'automatisation.

À qui s'adresse ce livre ?

Aux makers, amateurs d'électronique, bricoleurs, bidouilleurs...

Dans ce livre, vous apprendrez notamment à :

- créer un système d'arrosage automatique de vos plantes avec Arduino
- mettre au point un rafraîchisseur de boissons
- fabriquer une marionnette qui danse en fonction de vos tweets
- concevoir un éclateur aléatoire de ballon

Sur www.serialmakers.com

Téléchargez le code source des programmes Arduino et Raspberry Pi présentés dans cet ouvrage.

Simon Monk est l'auteur de nombreux best-sellers d'électronique pour les makers. Il aide également sa femme, qui a lancé le site web MonkMakes.com, à concevoir et vendre des kits et d'autres articles liés à ses ouvrages. Vous pouvez le suivre sur Twitter et en savoir plus sur ses livres sur www.simonmonk.org.

Make:
makezine.com

SERIAL MAKERS
www.serialmakers.com

www.editions-eyrolles.com

Mouvement, lumière et son avec ARDUINO et RASPBERRY PI

CHEZ LE MÊME ÉDITEUR

Dans la collection « Serial Makers »

C. PLATT. – **L'électronique en pratique (2e édition).**
N°14425, 2016, 328 pages.

E. DE KEYSER. – **Filmer et photographier avec un drone.**
N°14241, 2016, 168 pages.

F. BOTTON. – **Les drones de loisir (2e édition).**
N°14436, 2016, 224 pages.

R. JOBARD. – **Les drones (2e édition).**
N°14189, 2016, 202 pages.

C. PLATT. – **L'électronique en pratique 2.**
N°14179, 2015, 336 pages.

E. BARTMANN. – **Le grand livre d'Arduino (2e édition).**
N°14117, 2015, 612 pages.

C. BOSQUE, O. NOOR et L. RICARD. – **FabLabs, etc.** *Les nouveaux lieux de fabrication numérique.*
N°13938, 2015, 216 pages.

M. RICHARDSON et S. WALLACE. – **À la découverte du Raspberry Pi.**
N°13747, 2013, 176 pages.

B. PETTIS, A. KAZIUNAS FRANCE et J. SHERGILL. – **Imprimer en 3D avec la MakerBot.**
N°13748, 2013, 226 pages.

J. BOYER. – **Réparez vous-même vos appareils électroniques.**
N°13936, 2014, 384 pages.

A. KAZIUNAS FRANCE *et al.* – **Pratique de l'impression 3D**.
N°13924, 2014, 228 pages.

Simon Monk

Mouvement, lumière et son avec ARDUINO et RASPBERRY PI

EYROLLES

SERIAL MAKERS

ÉDITIONS EYROLLES
61, bld Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Traduction autorisée de l'ouvrage en langue anglaise intitulé *Make: Action* de Simon Monk (ISBN : 978-1-457-18779-7), publié par Maker Media, Inc.

Adapté de l'anglais par Danielle Lafarge, relecture technique par Jean Boyer

Table des matières

Arduino vs Raspberry Pi

1

Il n'a jamais été aussi facile pour un amateur de faire ses premiers pas dans l'univers de l'électronique avec les plates-formes Arduino et Raspberry Pi. Vous pourrez tout aussi bien créer un système de domotique permettant de réguler l'éclairage et le chauffage de votre maison via le réseau Wi-Fi ou simplement piloter quelques moteurs.

Mais même si Arduino et Raspberry Pi sont tous les deux des circuits de la taille d'une carte de crédit, ils diffèrent par de nombreux aspects. Arduino est une carte à microcontrôleur rudimentaire qui n'est pas dotée d'un quelconque système d'exploitation, tandis que Raspberry Pi est un nano-ordinateur qui fonctionne sous Linux et qui peut être interfacé avec des composants électroniques externes.

Arduino

Il existe un vaste choix de cartes Arduino. Dans ce livre, nous nous concentrerons sur le modèle le plus répandu : Arduino Uno (figure 1-1). Une carte Arduino coûte un peu moins cher qu'un Raspberry Pi. Il vous faudra débourser 25 € environ pour une Arduino Uno.

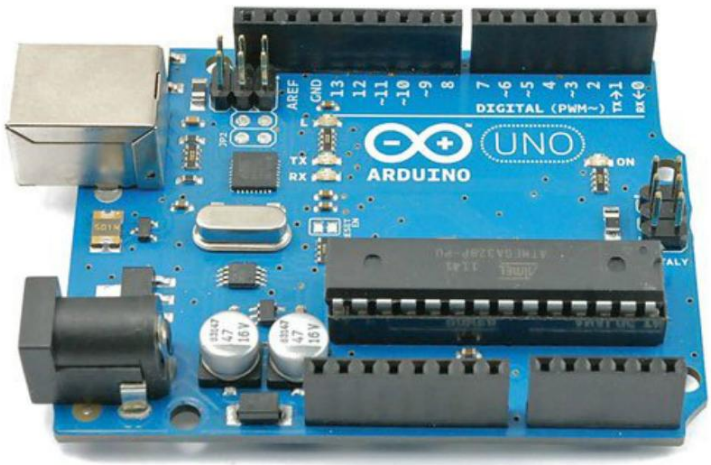


Figure 1-1. *Une platine Arduino Uno Revision 3*

Si vous avez l'habitude de travailler sur un PC ordinaire, les caractéristiques techniques d'Arduino vous paraîtront totalement insuffisantes. La carte ne dispose que de 34 Ko de mémoire (de différents types). Cela signifie que Raspberry Pi a environ 30 000 fois plus de mémoire, sans compter la mémoire flash de la carte SD du Pi ! En outre, Arduino Uno est équipé d'un processeur qui n'est cadencé qu'à 16 MHz. Il n'est pas possible de brancher un clavier, une souris ou un écran à la platine. Enfin, il n'y a pas non plus de système d'exploitation.

Peut-être vous interrogez-vous sur l'utilité de cette nanocarte. Le secret réside dans son extrême simplicité. Il n'y a pas de système d'exploitation à initialiser ou d'interfaces superflues qui ne font qu'accroître les coûts et consommer de l'énergie.

Alors que Raspberry Pi est un ordinateur polyvalent, Arduino ne joue qu'un seul rôle – la connexion et le pilotage de composants électroniques.

Pour programmer un Arduino, vous avez besoin d'un ordinateur ordinaire (ou d'un Raspberry Pi, si vous le souhaitez) sur lequel un IDE *(Integrated Development Environment)* a préalablement été installé. Cet environnement de programmation vous sert à écrire un programme qui sera ensuite transféré dans la mémoire flash d'Arduino.

L'Arduino ne peut exécuter qu'un seul programme à la fois. Une fois programmé, il le gardera en mémoire et l'exécutera automatiquement à chaque initialisation.

Arduino est conçu pour recevoir des shields, c'est-à-dire des cartes qui s'enfichent sur ses prises d'entrée-sorties afin de doter la platine de fonctionnalités matérielles supplémentaires, comme divers types d'écrans ou des adaptateurs Ethernet et Wi-Fi.

Un Arduino se programme à l'aide du langage de programmation C (vous en apprendrez plus sur la programmation et Arduino au chapitre 2).

Raspberry Pi

Si vous débutez dans le domaine de l'électronique, mais que vous savez utiliser un ordinateur, vous vous sentirez en terrain connu avec Raspberry Pi (figure 1-2). La nanocarte est une version réduite d'un ordinateur fonctionnant sous Linux. Elle est dotée de ports USB permettant de connecter un clavier et une souris, ainsi que d'une une sortie audio et d'une sortie vidéo HDMI pour le branchement d'un écran.

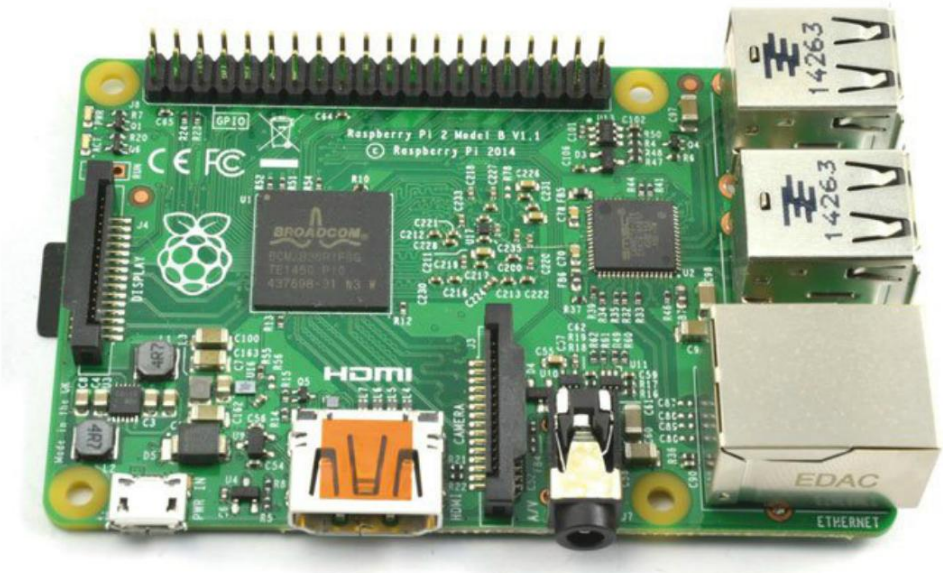


Figure 1-2. *Un Raspberry Pi 2*

Vous pouvez relier Raspberry Pi à un réseau via son port Ethernet ou par le biais d'adaptateurs Wi-Fi USB. La carte est alimentée par un port micro USB.

Il n'y a pas de disque dur ; une carte microSD est utilisée comme support de stockage. Elle contient le système d'exploitation, ainsi que tous vos documents et programmes.

À l'origine, Raspberry Pi a été créé au Royaume-Uni pour servir d'ordinateur bon marché destiné à l'enseignement des bases de l'informatique, et notamment de la programmation en Python, à des écoliers. D'ailleurs, « Pi » serait dérivé du « Py » de Python.

Qu'est-ce qui différencie le Raspberry Pi d'un ordinateur de bureau ou d'un ordinateur portable fonctionnant sous Linux ?

- Il coûte moins de 40 € (le modèle A+ qui est une version simplifiée du Raspberry Pi est encore meilleur marché et le modèle Zéro est quasiment donné).
- Il nécessite une alimentation de moins de 5 watts.
- Raspberry Pi est doté de deux rangées de broches GPIO (*General-Purpose Input/Output*, en haut à gauche sur la figure 1-1) auxquelles vous pouvez connecter directement des composants électroniques, tels que des LED, un écran, des moteurs, ainsi que les différents types de périphériques de sortie utilisés pour la réalisation des montages présentés dans ce livre.

Par ailleurs, Raspberry Pi peut être connecté à Internet via le Wi-Fi ou un câble LAN, ce qui permet de réaliser des projets dans le domaine de l'Internet des objets (voir le chapitre 16).

Raspberry Pi 2 (dernière et meilleure version au moment de l'écriture de ce livre) présente les caractéristiques suivantes :

- processeur ARM v7 900 MHz Quad-Core ;
- 1 Go de mémoire DDR2
- Ethernet 10.100 BaseT
- quatre ports USB 2.0
- sortie vidéo HDMI
- prise pour caméra
- connecteur GPIO 40 broches (qui fonctionnent toutes en 3,3 V)

Si vous découvrez Raspberry Pi, reportez-vous au chapitre 3 où nous présentons les composants matériels et le langage de programmation Python.

Quelle carte choisir ?

Ce livre explique comment connecter des composants électroniques à Arduino et Raspberry Pi parce que certains projets se prêtent mieux à l'un ou à l'autre. D'autres cartes qui se situent entre ces deux extrêmes partagent plus de points communs avec l'un ou l'autre et les explications fournies dans ce livre resteront valables.

Au moment de vous lancer dans un nouveau projet, je vous conseille d'opter en priorité pour une carte Arduino. Toutefois, si le projet présente l'une des exigences suivantes, il sera sans doute préférable de privilégier une carte Raspberry Pi :

- Internet ou connexion réseau
- grand écran
- clavier ou souris
- périphériques USB, tels qu'une webcam

En ne ménageant pas ses efforts et son porte-monnaie, il est possible d'étendre les fonctionnalités d'Arduino par des shields qui répondront à la plupart des exigences précédentes. Toutefois, les montages sont plus compliqués, car ce ne sont pas des fonctionnalités natives d'Arduino alors qu'elles le sont pour Pi.

Parmi les bonnes raisons d'utiliser un Arduino plutôt qu'un nano-ordinateur Raspberry Pi, il faut citer :

Coût

Un Arduino Uno est moins cher qu'un Raspberry Pi 2.

Temps de démarrage

Avec un Arduino, on n'a pas besoin d'attendre que le système d'exploitation démarre. Il y a un temps de latence d'une seconde environ pendant que le processeur vérifie si un nouveau programme est en cours de transfert. Ensuite, le module est prêt.

Fiabilité

Une platine Arduino est intrinsèquement beaucoup plus simple et résistante qu'un micro-ordinateur Raspberry Pi et elle n'est pas supervisée par un système d'exploitation.

Consommation d'énergie

Un Arduino consomme 1/10^{e} environ de l'énergie nécessaire au fonctionnement d'un Raspberry Pi. Si des piles ou l'énergie solaire sont utilisées comme source d'alimentation, mieux vaut opter pour une platine Arduino.

Courant de sortie GPIO

Les broches GPIO du Raspberry Pi doivent uniquement être utilisées pour fournir un courant maximum d'environ 16 mA. En revanche, les broches d'Arduino autorisent une sortie nominale de 40 mA. Dans certains cas, vous pouvez donc connecter directement un composant (comme une LED) à un Arduino alors que ce n'est pas possible avec un Raspberry Pi.

L'Arduino et le Raspberry Pi sont tous les deux d'excellentes cartes sur lesquelles baser vos projets. Dans une certaine mesure, le choix est aussi une question de préférence personnelle.

De manière générale, lorsque vous connectez des composants externes à un Raspberry Pi, souvenez-vous qu'il fonctionne en 3,3 V, alors qu'Arduino utilise une tension de 5 V. Si vous branchez un composant fonctionnant en 5 V à l'une des broches GPIO du Raspberry Pi, vous risquez d'endommager la broche, voire de détruire la carte.

Les alternatives

L'Arduino Uno et le Raspberry Pi se situent chacun à une extrémité de la palette des cartes permettant de piloter des éléments. Comme vous pouvez vous en douter, d'autres cartes intermédiaires ont été produites dans le but de pallier les faiblesses des unes et des autres.

De nouvelles platines voient régulièrement le jour. La nature open source d'Arduino permet de donner naissance à de nombreuses variations, ainsi qu'à des modules répondant à des besoins spécifiques, comme le contrôle de drones ou l'interfaçage avec des capteurs sans fils.

La figure 1-3 réunit quelques exemples de cartes les plus populaires dans ce domaine.

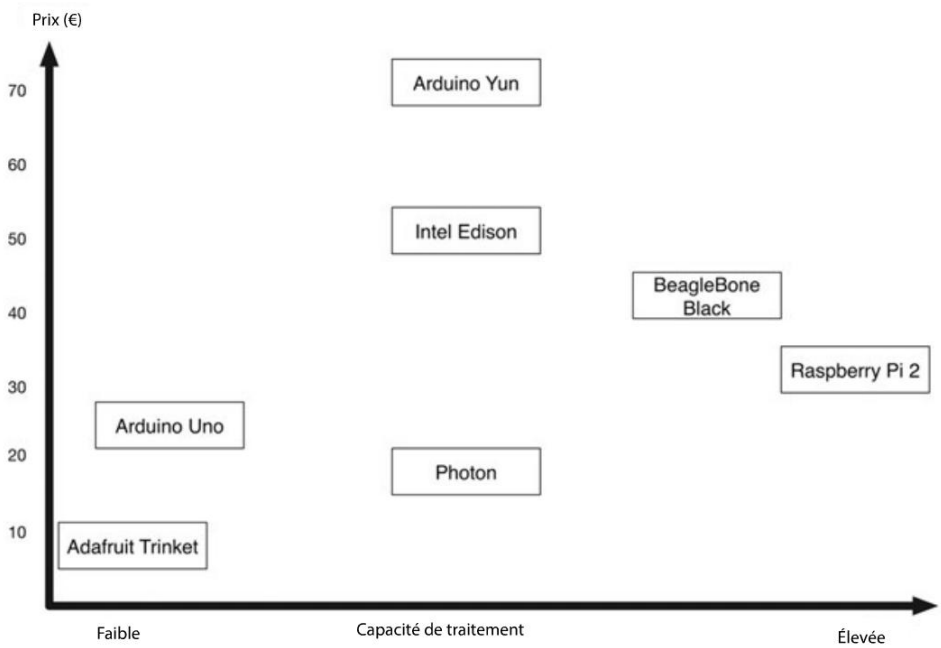


Figure 1-3. *Plates-formes intégrées*

Le modèle Adafruit Trinket se place avant l'Arduino Uno à la fois en termes de prix et de performance. Cette carte intéressante ne possède que quelques broches GPIO. Mais, elle est relativement compatible avec l'Arduino. Elle mérite donc d'être envisagée pour un projet ne nécessitant qu'une ou deux entrées ou sorties.

Il existe une catégorie de produits intermédiaires comprenant les cartes Arduino Yun, Intel Edison et Photon, qui intègrent toutes des fonctionnalités Wi-Fi et sont destinées à des projets dans le domaine de l'Internet des objets (IoT, *Internet of Things,* voir le chapitre 16). Parmi ces cartes, le modèle Photon présente probablement le meilleur rapport qualité-prix. Ces trois cartes se programment en Arduino C. Par conséquent, tout ce que vous aurez appris sur la carte Arduino s'appliquera aussi à ces cartes.

La carte BeagleBone Black est très proche du Raspberry Pi du point de vue de son concept. Il s'agit aussi d'un ordinateur monocarte et même si la version actuelle du BeagleBone Black arrive derrière le Raspberry Pi 2 en termes de puissance brute, elle présente néanmoins des avantages

sur ce dernier : elle compte davantage de broches GPIO, certaines pouvant même être utilisées comme des entrées analogiques, ce qui n'est pas possible sur le Raspberry Pi 2. En outre, la platine BeagleBone Black peut être programmée en Python, comme Raspberry Pi, ou en JavaScript.

Résumé

Dans ce chapitre, nous avons brièvement présenté Arduino et Raspberry Pi, avec leurs avantages et leurs inconvénients, et nous avons évoqué quelques alternatives. Dans les deux chapitres suivants, nous examinerons le fonctionnement et la programmation d'une carte Arduino, puis d'un nano-ordinateur Raspberry Pi.

Si vous avez déjà utilisé ces deux composants électroniques, vous pouvez aller directement au chapitre 4 où nous passerons à la pratique. Vous pourrez toujours revenir aux chapitres 2 et 3 si vous en éprouvez le besoin.

Arduino

Ce chapitre est une version modifiée de l'annexe de mon livre *The Maker's Guide to the Zombie Apocalypse*, paru chez NoStarch Press. Je l'ai repris ici avec leur aimable autorisation.

Si vous faites vos premiers pas avec Arduino, ce chapitre vous expliquera les bases de l'utilisation de très petite carte.

Qu'est-ce qu'un Arduino ?

Il existe différents types de cartes Arduino, mais la plus répandue est l'Arduino Uno (figure 2-1). C'est celle qui est utilisée pour tous les projets de ce livre.

L'Arduino Uno a connu des versions successives, ou révisions. Le modèle illustré à la figure 2-1 est une révision 3 (R3). C'est la plus récente à l'heure où j'écris ces lignes.

Nous commencerons notre présentation d'Arduino par la prise USB qui joue plusieurs rôles : elle peut servir à l'alimentation de la carte, à sa programmation depuis l'ordinateur et, enfin, comme liaison de communication.

Le petit bouton rouge qui se trouve à côté de la prise USB est le bouton de réinitialisation. Lorsque vous appuyez dessus, l'Arduino redémarre et exécute le programme qui y est installé.

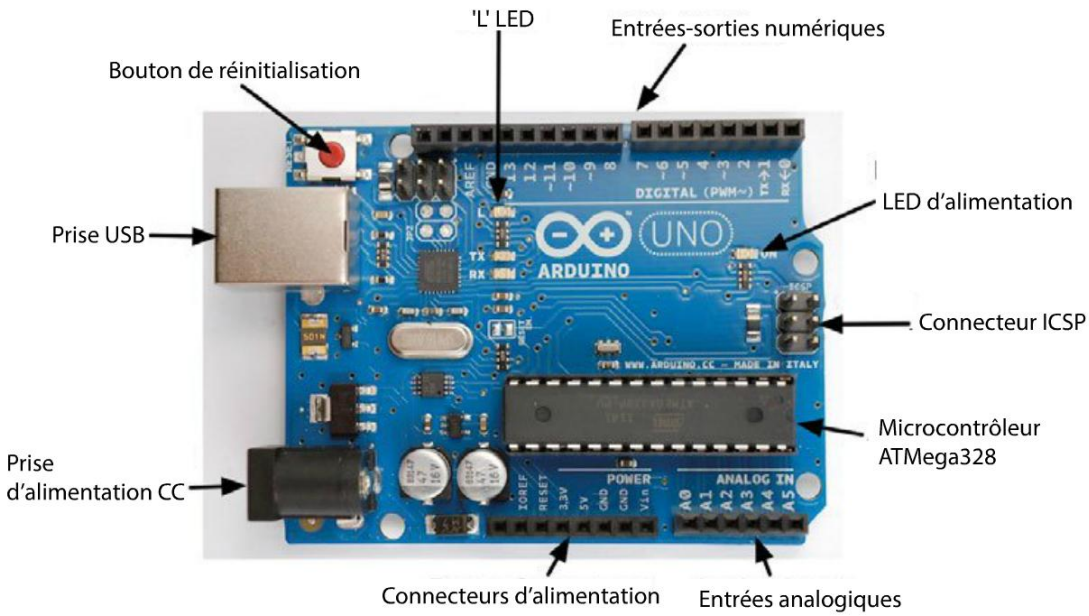


Figure 2-1. *Un Arduino Uno R3*

Le long des bords supérieurs et inférieurs de l'Arduino se trouvent des broches auxquelles il est possible de connecter des composants électroniques. Les entrées et sorties numériques, désignées par des chiffres compris entre 0 et 13, sont visibles le long du bord supérieur de la carte illustrée à la figure 2-1. Chaque broche peut être configurée dans vos programmes pour servir d'entrée ou de sortie. Si vous connectez un interrupteur à une entrée numérique, celle-ci indiquera s'il est enfoncé ou non. Ou bien, lorsque vous connectez une LED à une sortie numérique et que cette dernière passe de l'état haut à l'état bas, la LED s'allume. Une LED se trouve déjà sur la carte : il s'agit de la LED « L » qui est connectée à la broche numérique 13.

Au-dessous des broches d'E/S numériques, une LED d'alimentation indique simplement si la carte est sous tension. Le connecteur ICSP *(In-Circuit Serial Programming)* sert uniquement à la programmation avancée de carte Arduino qui ne passe pas par la connexion USB. La majorité des utilisateurs ne s'en servent jamais.

L'ATMega328 (le cerveau de l'Arduino) est un microcontrôleur à circuit intégré (IC, *Integrated Circuit*). Les 32 Ko de mémoire flash de la puce contiennent le programme nécessaire au fonctionnement de la carte.

Au-dessous de l'ATMega328, il y a une rangée de broches d'entrées analogiques étiquetées d'A0 à A5. Alors que les entrées numériques détectent seulement si un composant est allumé ou éteint, les entrées analogiques peuvent mesurer la tension reçue par la broche (à condition que cette tension soit comprise entre 0 et 5 V). Cette tension peut notamment provenir d'un capteur. Si vous n'avez pas assez d'entrées et sorties numériques, ces broches d'entrées analogiques peuvent aussi être utilisées comme des entrées numériques.

À côté de ces entrées se trouve une rangée de connecteurs d'alimentation qui peuvent être utilisés de façon alternative pour l'alimentation de l'Arduino. Ils peuvent également être utilisés pour alimenter d'autres composants électroniques contrôlés depuis la carte.

L'Arduino dispose également d'une prise d'alimentation en courant continu. Elle peut recevoir une tension comprise entre 7 et 12 V CC et utilise un régulateur de tension pour fournir la tension de 5 V nécessaire à l'Arduino. La platine acceptera automatiquement l'alimentation provenant de la prise USB ou du connecteur d'alimentation, selon le cas.

Installation de l'IDE Arduino

L'Arduino ne correspond pas tout à fait à l'image que l'on se fait d'un ordinateur. Il n'a pas de système d'exploitation et on n'y connecte pas de clavier, de moniteur ou de souris. Un seul programme peut y être exécuté à la fois et il faut préalablement le charger dans la mémoire flash de la carte à l'aide d'un ordinateur. L'Arduino peut être reprogrammé autant de fois que vous le souhaitez.

Pour pouvoir le programmer, vous devez installer l'IDE Arduino sur votre ordinateur. Ce logiciel peut fonctionner sur différentes plates-formes – Windows, Mac et Linux – et c'est l'une des raisons de la popularité de l'Arduino. De plus, l'IDE vous permet de programmer la carte via l'interface USB sans qu'il soit nécessaire d'utiliser de matériel particulier.

Pour installer l'IDE Arduino sur votre ordinateur, téléchargez le logiciel et suivez les instructions affichées sur le site web Arduino (*http://arduino.cc/en/Guide/HomePage*).

Sous Windows et Mac, il faut également installer des pilotes USB afin que l'IDE puisse communiquer avec la carte Arduino.

Une fois que tout est installé, démarrez l'IDE. La figure 2-2 présente la fenêtre de l'interface.

Comme son nom l'indique, le bouton Téléverser permet de transférer le sketch sur la carte Arduino. Avant que le sketch ne soit téléversé, l'IDE convertit le code de programmation textuel en code exécutable par l'Arduino. Si des erreurs s'y sont glissées, elles sont affichées dans la fenêtre de messagerie. Le bouton Vérifier remplit le même rôle, mais il ne lance pas le transfert du programme sur la carte.

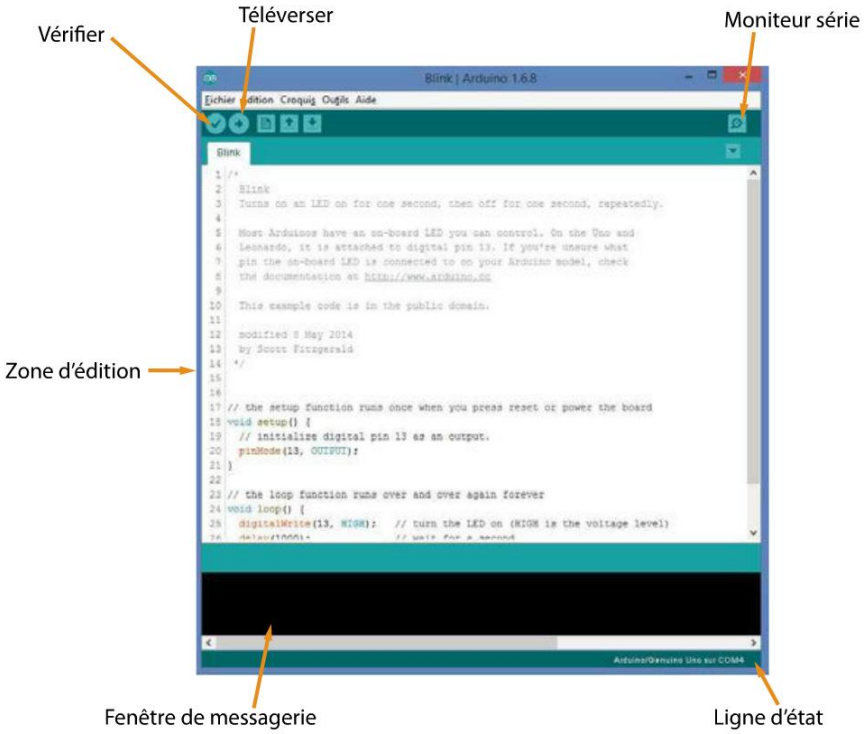


Figure 2-2. *L'IDE Arduino*

Le bouton Moniteur série ouvre la fenêtre du même nom qui sert à communiquer avec l'Arduino. Vous vous servirez de cette interface pour de nombreuses expériences proposées dans ce livre, car c'est un excellent moyen de transmettre des commandes à l'Arduino depuis votre ordinateur. Le moniteur série permet de communiquer dans les deux sens : vous pouvez envoyer des messages texte à l'Arduino et recevoir des réponses de la part de la carte.

La ligne d'état dans le bas de la fenêtre indique le type de carte Arduino et le port série utilisé pour sa programmation lorsque vous cliquez sur le bouton Téléverser. Le port illustré à la figure 2-2 (`/dev/cu.usbmodem411`) s'affiche généralement lorsque l'on utilise un ordinateur sous Mac ou Linux. Si vous utilisez un ordinateur Windows pour programmer l'Arduino, il s'agira du port COM4, ou des lettres COM suivies du numéro du port alloué par Windows à l'Arduino lorsque vous avez connecté la carte.

Enfin, la partie principale de l'IDE Arduino est la zone d'édition dans laquelle vous saisissez le code du programme que vous voulez téléverser sur l'Arduino.

Dans le monde de l'Arduino, les programmes se nomment des sketches (ou croquis dans la version française de l'IDE). Depuis le menu Fichier de l'IDE Arduino, vous pouvez Ouvrir et Enregistrer des sketches à la façon d'un document dans un traitement de texte. Le menu Fichier contient aussi un sous-menu Exemples à partir duquel vous pouvez charger des exemples de sketches fournis avec l'IDE.

Téléversement d'un sketch

Pour tester votre carte Arduino et pour vous assurer que l'IDE Arduino est correctement installé, commencez par ouvrir le sketch intitulé *Blink* qui se trouve dans Fichier>Exemples>01. Basics. (Le sketch *Blink* est affiché dans la zone d'édition illustrée à la figure 2-2.)

Utilisez un câble USB pour raccorder la carte à l'ordinateur qui servira à programmer l'Arduino. La LED d'alimentation de la carte s'allume et des LED clignotent.

Lorsque l'Arduino est connecté, vous devez indiquer à l'IDE quel type de carte est programmé (Arduino Uno) et le port série auquel elle est connectée. Définissez le type de carte à l'aide de la commande Outils>Type de carte>Arduino Uno.

Définissez le port série en sélectionnant la commande Outils>Port. Si vous utilisez un ordinateur sous Windows, le sous-menu ne devrait pas comporte beaucoup d'options. Il est même possible que vous n'y trouviez que l'option COM4. Sur un ordinateur fonctionnant sous Mac ou Linux, vous devriez avoir le choix entre divers ports USB et vous aurez du mal à savoir lequel correspond à celui auquel l'Arduino est connecté. La liste doit proposer une entrée commençant par `dev/tty.usbmodemNNN`, `NNN` étant un nombre. À la figure 2-3, la carte Arduino connectée à mon Mac a été sélectionnée.

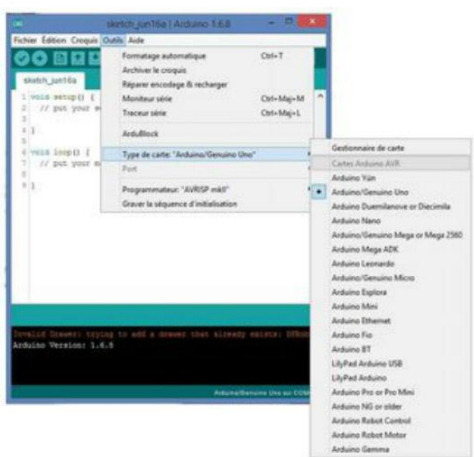


Figure 2-3. *Sélection du port série Arduino/Genuino Uno*

Si votre Arduino n'apparaît pas dans la liste, c'est généralement dû à un problème de pilote USB. Dans ce cas, essayez de réinstaller les pilotes.

Lorsque vous êtes prêt à transférer le sketch sur la carte, cliquez sur le bouton Téléverser. Des messages s'affichent dans la partie inférieure de la fenêtre. Puis, au bout de quelques secondes, les LED intitulées « TX » et « RX » se mettent à clignoter pendant que le programme est transmis à la carte.

Si tout se déroule comme prévu, un message ressemblant à celui illustré à la figure 2-4 devrait s'afficher lorsque la transmission est terminée.

Figure 2-4. *Transmission réussie*

Ce message indique que le sketch a été transmis et qu'il utilise 1 066 octets sur les 32 256 octets disponibles.

Une fois la transmission terminée, vous devriez voir clignoter la LED « L » intégrée à la carte. C'est le signe que le sketch Blink fonctionne correctement.

Le code du livre

Tous les programmes de ce livre – que ce soit les sketches Arduino ou les programmes Python pour le Raspberry Pi – sont disponibles en anglais sur la page GitHub du livre (*https://github.com/simonmonk/make_action*). Pour importer ces fichiers sur votre ordinateur sous Mac, Linux ou Windows, cliquez sur le bouton Clone or download de la page GitHub, puis sur Download ZIP. Les sketches Arduino en français sont disponibles sur *www.serialmakers.com*.

Enregistrez l'archive téléchargée sur le Bureau ou à un autre emplacement facile d'accès. Lorsque vous décompressez l'archive, vous obtenez un dossier intitulé `make_actionmaster`. Le code Arduino se trouve dans un dossier `arduino`. À l'intérieur, vous trouverez les deux sous-dossiers `experiments` et `projects`.

Chaque programme est rangé dans un dossier spécifique qui contient généralement un seul fichier qui correspond au programme proprement dit. Par exemple, à l'intérieur du dossier `experiments`, vous trouverez le dossier `ex_01_basic_motor_control` qui contient l'unique fichier `basic_motor_control.ino`. Si l'IDE Arduino est déjà installé sur votre ordinateur, vous pouvez ouvrir directement le fichier dans l'environnement de programmation.

Une autre façon d'accéder à ces sketches est de copier les dossiers d'expériences et de projets dans votre sketchbook Arduino. Il s'agit du dossier intitulé `Arduino`, qui se trouve dans votre dossier de documents habituel (`Mes documents` sous Windows, ou `Documents` sous Mac).

Si les fichiers sont copiés dans le dossier sketchbook, vous pourrez y accéder à l'aide de la commande Fichier>Carnet de croquis dans l'IDE Arduino.

Guide de programmation

Nous allons passer en revue les principales commandes pour vous aider à mieux comprendre les sketches utilisés dans le livre.

Setup et Loop

Tous les sketches Arduino doivent contenir une fonction `setup()` et une fonction `loop()` (les fonctions sont des blocs de code de programme qui réalisent une tâche). Pour comprendre le fonctionnement de `setup()` et `loop()`, nous allons disséquer l'exemple du sketch *Blink* que nous avions transmis à la carte Arduino :

```
int led = 13;
// la fonction d'initialisation est exécutée une fois après l'action sur le bouton de
réinitialisation :
void setup() {
  // initialise la broche numérique comme sortie.
  pinMode(led, OUTPUT);
}

// la fonction de boucle est exécutée en continu :
void loop() {
    digitalWrite(led, HIGH);   // allume la LED (le niveau de tension est HIGH)
  delay(1000);                                  // pause 1 seconde
  digitalWrite(led, LOW);      // éteint la LED (le niveau de tension est LOW)
  delay(1000);                                  // pause 1 seconde
}
```

Notez qu'une grande partie du texte qui a été traduit en français pour les besoins de l'explication est précédé des signes `//`. Cela signifie que le texte qui suit `//` jusqu'à la fin de la ligne doit être considéré comme un commentaire. Ce n'est pas un code de programme, il sert simplement à expliquer au lecteur ce qu'il se passe dans le programme.

Les lignes de code à l'intérieur de la fonction `setup()` ne sont exécutées qu'une seule fois (ou, plus précisément, chaque fois que la carte Arduino est mise sous tension ou que vous appuyez sur le bouton de réinitialisation). Par conséquent, on utilise la fonction `setup()` pour effectuer toutes les choses qui ne doivent être faites qu'une seule fois au démarrage du programme. Dans le cas de *Blink*, cela consiste simplement à préciser que la broche LED est définie comme sortie.

Les commandes à l'intérieur de la fonction `loop()` sont exécutées en boucle – dès que la dernière ligne de commande `loop()` a été exécutée, elle recommence à la première ligne.

Je n'ai pas expliqué le rôle des commandes à l'intérieur de `setup()` et de `loop()`, mais j'y arrive bientôt.

Variables

Les variables permettent de nommer des valeurs. La première ligne du sketch *Blink* (si l'on ne tient pas compte des commentaires) est la suivante :

```
int led = 13;
```

Cela définit une variable nommée `led` en lui donnant une valeur initiale de 13. La valeur 13 a été choisie, car elle correspond au nom de la broche à laquelle la LED L est connectée et `int` désigne le type de variable. Le mot `int` est l'abréviation d'*integer* (« nombre entier », en anglais).

Même s'il n'est pas nécessaire d'utiliser un nom variable pour toutes les broches utilisées, il est judicieux de le faire, car il est alors plus facile de savoir à quoi sert la broche. En outre, si vous voulez utiliser une autre broche, il vous suffit de changer la valeur de la variable à un seul endroit.

Peut-être avez-vous remarqué que dans les sketches qui accompagnent ce livre, lorsque l'on déclare des variables comme celle-ci qui définit la broche utilisée, la ligne commence par `const` :

```
const int led = 13;
```

Le mot-clé `const` indique à l'IDE Arduino que la variable n'en est pas vraiment une, mais plutôt une constante ; en d'autres termes, sa valeur demeurera 13. Cela aboutit à des sketches légèrement plus petits qui sont exécutés plus rapidement. C'est généralement considéré comme une bonne habitude à prendre.

Sorties numériques

Le sketch *Blink* offre un bon exemple de sortie numérique. La broche 13 est configurée comme sortie dans la fonction `setup()` par cette ligne (la variable `led` ayant préalablement été définie sur 13) :

```
pinMode(led, OUTPUT);
```

Cette commande se situe dans la fonction `setup()`, car la configuration n'a besoin d'être effectuée qu'une seule fois. Lorsque la broche a été définie comme sortie, elle le restera jusqu'à ce qu'une autre instruction la configure différemment.

Pour clignoter, la LED doit être allumée et éteinte plusieurs fois. Par conséquent, le code est le suivant :

```
digitalWrite(led, HIGH);      // allume la LED (le niveau de tension est HIGH)
delay(1000);                              // pause 1 seconde
digitalWrite(led, LOW);       // éteint la LED (le niveau de tension est LOW)
delay(1000);                              // pause 1 seconde
```

La fonction `digitalWrite()` a deux paramètres (entre parenthèses et séparés par une virgule). Le premier paramètre est la broche Arduino qui sera écrite et le second paramètre est la valeur qui y sera écrite. Par conséquent, la valeur `HIGH` appliquera une tension de 5 V à la sortie (la LED s'allume) et la valeur `LOW` met la broche hors tension, à 0 V (la LED s'éteint).

La fonction `delay()` interrompt le programme pendant la durée (en millisecondes) précisée comme paramètre. Il y a 1 000 millisecondes dans 1 seconde, donc chacune des fonctions `delay()` présentées interrompt le programme pendant 1 seconde.

Dans l'expérience « Pilotage d'une LED », décrite au chapitre 4, au lieu de la LED de la carte, vous utiliserez une sortie numérique connectée à une LED externe que vous ferez clignoter.

Entrées numériques

Dans ce livre, nous nous intéressons davantage aux sorties qu'aux entrées, donc nous utilisons surtout la fonction `digitalWrite()`. Les entrées numériques permettent de connecter des boutons et des capteurs à une carte Arduino.

Vous pouvez définir une broche Arduino comme entrée numérique à l'aide de la fonction `pinMode()`. L'exemple suivant définit la broche 7 comme entrée. Vous pouvez évidemment aussi utiliser un nom de variable à la place du chiffre 7.

Comme pour une sortie, la broche d'entrée est définie dans la fonction `setup()`, car il est rare de changer le mode d'une broche en cours d'exécution du sketch :

```
pinMode(7, INPUT)
```

Une fois la broche définie comme entrée, vous pouvez la lire pour déterminer si la tension y est plus proche de 5 V (`HIGH`) ou de 0 V (`LOW`). Dans cet exemple, la LED s'allume si l'entrée est LOW au moment de la lecture (une fois allumée, la LED le reste, car le code ne précise pas qu'il faut l'éteindre) :

```
void loop())
{
  if (digitalRead(7) == HIGH)
  {
    digitalWrite(led, LOW)
  }
}
```

À partir de là, le code se complique. Nous le parcourons donc ligne par ligne.

À la deuxième ligne, nous trouvons le symbole `{`. Il arrive qu'il soit placé sur la même ligne que `loop()` ou sur la ligne suivante. Ce n'est qu'une question de préférence personnelle et cela n'a aucune incidence sur l'exécution du code. Le symbole `{` signale le début d'un bloc de code qui se termine par le symbole `}` correspondant. C'est une façon de regrouper toutes les lignes de code qui appartiennent à la fonction `loop`.

La première de ces lignes utilise l'instruction `if` qui est immédiatement suivie d'une condition. Dans ce cas, la condition est `(digitalRead(7) == HIGH)`. Le double signe égal (`==`) permet de comparer les deux valeurs qui figurent de chaque côté. Donc, ici, si la broche 7 est égale à HIGH, alors le bloc de code délimité par `{` et `}` qui figure après `if` est exécuté. Sinon, il ne l'est pas. L'alignement des accolades permet d'associer plus facilement les paires.

Nous avons déjà examiné le code qui est exécuté si la condition est remplie. Il s'agit de la fonction `digitalWrite()`utilisée pour allumer la LED.

Dans cet exemple, on suppose que l'entrée numérique est strictement HIGH ou LOW. Si vous avez connecté un interrupteur à une entrée numérique, il ne pourra que fermer la connexion. D'ordinaire, il faut connecter l'entrée numérique à GND (0 V). Si la liaison établie par l'interrupteur est ouverte, on dira que l'entrée numérique est flottante. Au sens strict, elle n'est pas raccordée électriquement à un composant. L'entrée percevra un bruit électrique et elle alternera souvent entre l'état haut et l'état bas. Pour l'éviter, on utilise généralement une résistance pull-up, comme illustré à la figure 2-5.

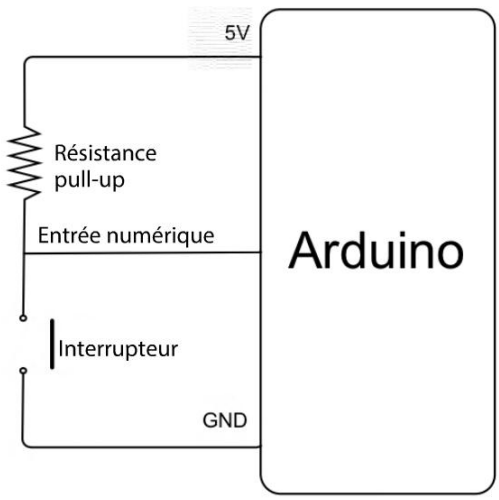


Figure 2-5. *Utilisation d'une résistance pull-up avec une entrée numérique*

Quand l'interrupteur est ouvert (comme illustré à la figure 2-5), la résistance tire la broche d'entrée à 5 V. Quand l'interrupteur est fermé, le tirage faible de l'entrée est contré puisque l'interrupteur relie l'entrée numérique à la masse (GND).

Même si vous pouvez effectivement utiliser une résistance pull-up, sachez que les entrées de la carte Arduino intègrent déjà des résistances pull-up internes d'environ 40 kΩ qui sont activées lorsque le mode de la broche est défini sur INPUT_PULLUP au lieu de INPUT. Le code suivant montre comment définir le mode d'une entrée numérique connectée à un interrupteur sans employer de résistance pull-up externe comme illustré à la figure 2-5 :

```
pinMode(switchPin, INPUT_PULLUP);
```

Entrées analogiques

Les entrées analogiques étiquetées de A0 à A5 peuvent mesurer une tension comprise entre 0 et 5 V. Contrairement aux entrées et sorties numériques, il n'est pas nécessaire d'utiliser `pinMode()` dans la fonction `setup` pour une entrée analogique.

Pour lire la valeur d'une entrée analogique, on utilise la fonction `analogRead()` en indiquant comme paramètre le nom de la broche qui doit être lue. Contrairement à `digitalRead()`, `analogRead()` renvoie un nombre et pas seulement un état vrai ou faux. Le nombre renvoyé par `analogRead()` est compris entre 0 et 1 023. Le résultat 0 correspond à 0 V et 1 023 à 5 V. Pour convertir le nombre reçu en une tension, il faut le multiplier par 5, puis le diviser par 1 023 – ou simplement le diviser par 204,6. Voici comment procéder en Arduino C :

```
int raw = analogRead(A0);
float volts = raw / 204.6;
```

La variable `raw` est un nombre entier (`int`), car le nombre transmis par une entrée analogique est toujours un nombre entier. Toutefois, pour convertir la valeur brute en nombre décimal, la variable doit être de type `float` (virgule flottante).

Différents capteurs peuvent être reliés à des entrées analogiques – par exemple, nous verrons plus loin comment utiliser une entrée analogique avec une photorésistance (voir le projet « Système d'arrosage Arduino pour plantes d'intérieur » du chapitre 7 page 114) et comment connecter une résistance variable (voir le projet « Système thermostatique de réfrigération de boisson » du chapitre 12 page 249).

Sorties analogiques

Une sortie numérique permet par exemple d'allumer ou d'éteindre une LED. Pour contrôler l'alimentation électrique de façon progressive, il faut utiliser une sortie analogique. Ce type de sortie permet notamment de régler la luminosité d'une LED ou la vitesse d'un moteur. Nous nous en servirons souvent dans ce livre.

Toutes les broches d'une carte Arduino Uno ne sont pas capables de se comporter comme des sorties analogiques – seules les broches D3, D5, D6, D9, D10 et D11 le peuvent. Elles sont repérées par un tilde ~ sur l'Arduino.

Pour contrôler une sortie analogique, utilisez la fonction `analogWrite()` en saisissant comme paramètre un nombre compris entre 0 et 255. Une valeur de 0 correspond à 0 V (complètement éteint) et une valeur de 255 à « complètement allumé ».

On pourrait croire qu'une sortie analogique fournit une tension comprise entre 0 et 5 V. Par conséquent, si vous branchez un voltmètre entre une broche utilisée comme sortie analogique et la broche GND, vous pourriez vous attendre à ce que la tension passe de 0 à 5 V lorsque vous changez la valeur définie par `analogWrite()`. En fait, c'est un peu plus compliqué que cela. La figure 2-6 montre ce qu'il se passe effectivement avec ce type de sortie à modulation de largeur d'impulsion (MLI, en anglais PWM pour *Pulse-Width Modulation*).

Toutes les broches de sorties analogiques génèrent 490 impulsions par seconde, sauf les broches D5 et D6 qui fonctionnent à 980 impulsions par seconde. La largeur des impulsions varie. Plus l'impulsion reste longtemps haute proportionnellement, plus la puissance délivrée à la sortie est importante et plus la LED est lumineuse ou plus le moteur est rapide.

Le voltmètre le signale comme un changement de tension, car cet instrument de mesure ne réagit pas suffisamment vite. Par conséquent, il calcule une sorte de moyenne pour produire une tension qui semble varier progressivement.

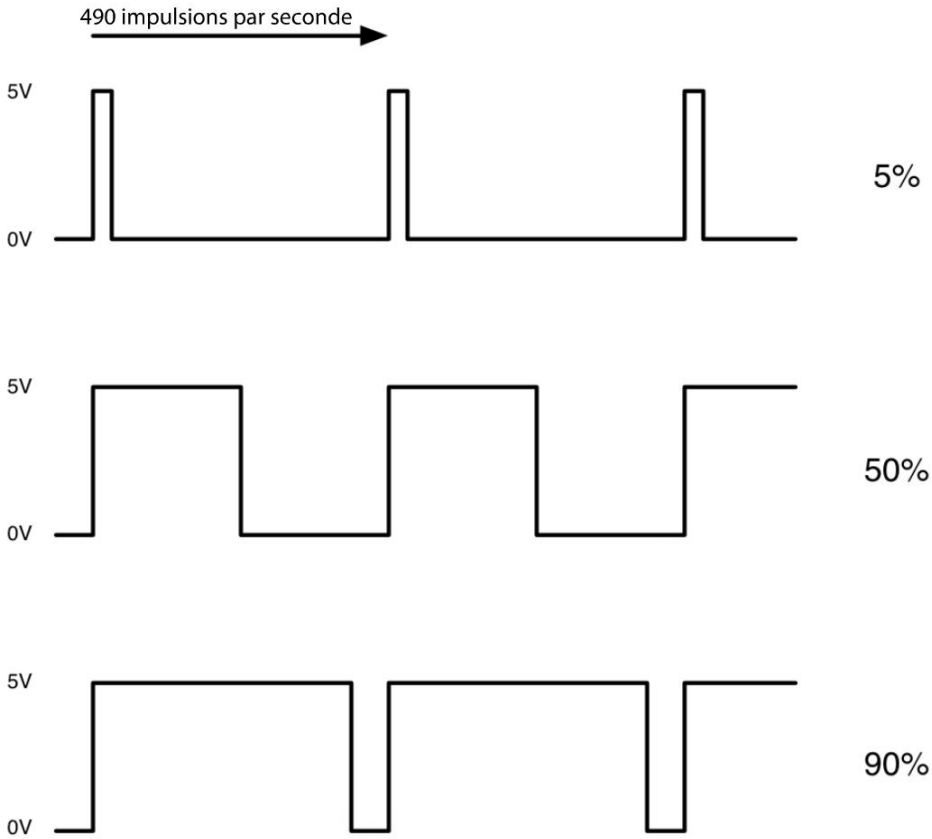


Figure 2-8. *Modulation de largeur d'impulsion des sorties analogiques*

If/Else

Comme nous l'avons vu à la section « Entrées numériques » page 17, nous avons utilisé une instruction `if` pour exécuter une action spécifique si une certaine condition est remplie. Pour contrôler davantage le déroulement du code, vous pouvez utiliser `if/else` afin d'exécuter un ensemble de code si la condition est remplie et un autre ensemble de code si elle ne l'est pas.

L'exemple suivant allume et éteint une LED selon qu'une valeur lue sur une broche analogique est supérieure à 500 ou inférieure ou égale à 500 :

```
if (analogRead(A0) > 500)
{
  digitalWrite(led, HIGH);
}
else
```

```
{
  digitalWrite(led, LOW);
}
```

Pour l'instant, nous avons vu deux types d'opérateurs de comparaison : == (égal à) et > (supérieur à). Toutefois, il en existe d'autres :

- <= (inférieur ou égal)
- >= (supérieur ou égal)
- != (différent de)

En plus de comparer deux valeurs seulement, vous pouvez aussi effectuer des comparaisons plus complexes à l'aide de `&&` (ET) et `||`(OU). Par exemple, si vous voulez uniquement allumer la LED si la valeur mesurée est comprise entre 300 et 400, par exemple, vous pourriez écrire le code suivant :

```
int reading = analogRead(A0);
if ((reading >= 300) && (reading <=400))
{
  digitalWrite(led, HIGH);
}
else
{
  digitalWrite(led, LOW);
}
```

Boucles

Une boucle permet de répéter un code un certain nombre de fois ou jusqu'à ce qu'une condition change. Dans ce cas, vous utilisez une boucle `for` ou une boucle `while` : les boucles `for` sont utilisées pour les actions répétées un nombre de fois prédéfini, tandis que les boucles `while` sont utilisées pour exécuter une action aussi longtemps qu'une condition est remplie.

La boucle `for` dans l'exemple suivant fait clignoter la LED dix fois (notez que la boucle `for` se trouve dans `setup()` plutôt que dans `loop()` ; si elle se trouvait dans `loop()`, elle clignoterait à nouveau 10 fois après avoir terminé la première série de 10 clignotements, ce qui n'est pas l'effet recherché) :

```
for (int i = 0; i < 10; i++)
{
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Si vous voulez que la LED continue à clignoter tant qu'un interrupteur connecté à une entrée numérique est relâché, vous utilisez une boucle `while` :

```
while (digitalRead(9))
```

```
{
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Ce code présuppose que la broche D9 est connectée à un interrupteur (figure 2-5).

Fonctions

Les fonctions peuvent être source de confusion pour les débutants en programmation. Pour simplifier, elles permettent de regrouper des lignes de code sous un nom global, ce qui permet de les réemployer facilement.

Si vous jetez un œil aux rouages internes d'Arduino, vous découvrirez que les fonctions intégrées comme `digitalWrite()` sont en fait complexes. Par exemple, voici le code de `digitalWrite()` (ne vous demandez pas à quoi il sert, soyez simplement heureux de ne pas avoir à saisir tout ce code chaque fois que vous voulez que des broches changent d'état) :

```
void digitalWrite(uint8_t pin, uint8_t val)
{
    uint8_t timer = digitalPinToTimer(pin);
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *out;

    if (port == NOT_A_PIN) return;

    // Si la broche est une broche de sortie PWM, elle doit
    // préalablement être éteinte.
    if (timer != NOT_ON_TIMER) turnOffPWM(timer);

    out = portOutputRegister(port);

    uint8_t oldSREG = SREG;
    cli();

    if (val == LOW) {
        *out &= ~bit;
    } else {
        *out |= bit;
    }

    SREG = oldSREG;
}
```

Le fait de nommer cet ensemble de code permet ensuite d'y faire facilement référence par son nom.

En plus des fonctions intégrées, comme `digitalWrite`, vous pouvez créer vos propres fonctions pour regrouper des commandes. Par exemple, vous pourriez créer une fonction qui fera clignoter la LED un certain nombre de fois défini comme paramètre. La broche qui doit clignoter peut aussi être définie comme paramètre. C'est ce que montre le sketch suivant en créant une fonction intitulée `blink` et en l'appelant durant `startup()` afin que la LED « L » de la carte Arduino clignote 5 fois après la réinitialisation :

```
const int ledPin = 13;
void setup()
{
  pinMode(ledPin, OUTPUT);
  blink(ledPin, 5);
}
void loop() {}

void blink(int pin, int n)
{
  for (int i = 0; i < n; i++)
  {
    digitalWrite(ledPin, HIGH);
    delay(500);
    digitalWrite(ledPin, LOW);
    delay(500)
  }
}
```

La fonction `setup()` définit `ledPin` comme sortie puis appelle la fonction `blink` en précisant le numéro de la broche qui doit clignoter suivi de nombre de clignotements. La fonction `loop()` est vide et ne joue aucun rôle, mais l'IDE Arduino exige sa présence.

La fonction `blink` commence par le mot `void` qui indique que la fonction ne renvoie rien – en d'autres termes, vous ne pouvez pas affecter le résultat de l'activation de cette fonction à une variable, comme vous pourriez le faire si la fonction exécutait un calcul quelconque. Ensuite vient le nom de la fonction (`blink`) suivi des paramètres nécessaires à la fonction qui sont placés entre parenthèses et séparés par des virgules. Lorsque vous définissez une fonction, vous devez préciser le type de chacun des paramètres (`int`, `float` ou autre). Dans ce cas, la broche qui doit clignoter (`pin`) et le nombre de clignotements (`n`) sont tous les deux des nombres entiers `int`.

Comme la plupart des langages de programmation, le langage C connaît le concept de variables globales et locales. Les variables globales (telles que `ledPin` dans l'exemple précédent) peuvent être utilisées n'importe où dans le programme. Par ailleurs, les variables locales, telles que les paramètres de fonctions (`pin` et `n` dans cet exemple), ou encore `i` à l'intérieur de la boucle `for`, sont uniquement accessibles à l'intérieur de la fonction qui les définit.

Par conséquent, dans `setup()`, la ligne `blink(ledPin, 5)` transmet la variable globale `ledPin` à la fonction `blink` où elle est affectée à la variable locale `pin`. Pourquoi faut-il procéder de cette façon ? Parce qu'ainsi, la fonction `blink` est polyvalente : elle peut servir à faire clignoter n'importe quelle broche et pas seulement `ledPin`.

Dans le corps de la fonction `blink`, on trouve une boucle `for` qui répète `n` fois les fonctions `digitalWrite()` et `delay()` qui y sont incluses. Donc, si `n` est 3, la LED clignotera 3 fois.

Résumé

Dans ce chapitre, vous avez appris à installer l'IDE Arduino et à utiliser quelques commandes de programmation de base. Nous avons également vu comment contrôler les broches d'entrées/sorties de la carte Arduino à l'aide du code.

Si vous débutez en programmation, vous devez vous familiariser avec certains des concepts abordés dans ce chapitre. L'une des meilleures façons d'apprendre à programmer est d'examiner des codes existants et de les modifier pour comprendre le rôle des différentes parties.

Les programmes utilisés dans ce livre peuvent être téléchargés sur le site web du livre (*https://github.com/simonmonk/make_action*) ou sur *www.serialmakers.com* pour les sketches Arduino en français. Vous n'avez donc pas besoin de les réécrire. De plus, vous pourrez vous servir des expériences et des projets présentés ici dans vos montages personnels.

Dans le chapitre suivant, nous présenterons les bases du Raspberry Pi.

Raspberry Pi

3

Ce chapitre vous aidera à bien démarrer avec votre Raspberry Pi. Si vous connaissez déjà le fonctionnement de ce nano-ordinateur et si vous avez déjà essayé la programmation en Python sur le Raspberry Pi, vous pouvez passer directement au chapitre 4.

Qu'est-ce qu'un Raspberry Pi ?

Un Raspberry Pi est un ordinateur monocarte doté du système d'exploitation Linux, de prises USB pour raccorder un clavier et une souris, ainsi que d'un connecteur HDMI pour raccorder un écran.

Il existe quelques variantes dont certaines ne sont plus disponibles. Tous les modèles de Raspberry Pi sont plus ou moins compatibles et la réalisation des exemples présentés dans ce livre ne devrait pas vous poser de difficultés, même si vous avez un vieux Raspberry Pi.

La figure 3-1 présente un Raspberry Pi 2 modèle B. Sur le côté droit de la carte, vous trouverez quatre ports USB qui peuvent servir à raccorder un clavier et une souris ou d'autres périphériques, comme une imprimante, un scanner ou une clé USB.

Au-dessous des ports USB, vous trouverez une prise RJ45 Ethernet qui permet de relier le Raspberry Pi à votre routeur domestique via un câble. Vous devrez connecter votre Raspberry Pi au réseau pour avoir accès à Internet et installer des logiciels sur votre nano-ordinateur. Peut-être préférerez-vous vous passer de câble et utiliser une clé Wi-Fi USB ? Un adaptateur de ce type ne coûte que quelques euros et se branche sur un port USB. Veillez toutefois à choisir un adaptateur compatible. Vous trouverez une liste de composants compatibles sur *http://elinux.org/RPi_VerifiedPeripherals*.

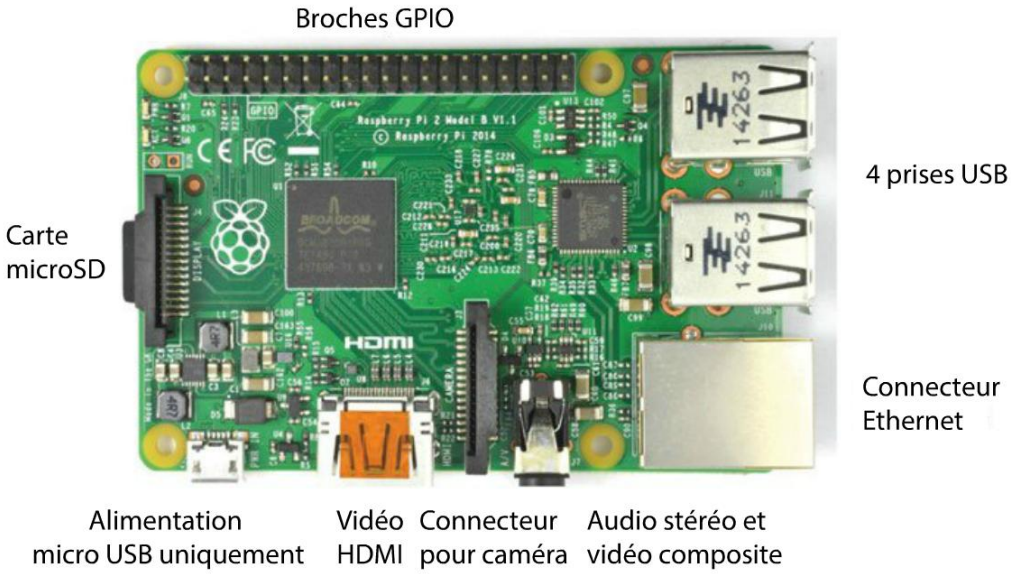


Figure 3-1. *Un Raspberry Pi 2 modèle B*

Si nous continuons à faire le tour de la carte Raspberry Pi dans le sens horaire, nous arrivons au connecteur audio stéréo et vidéo composite. Cette prise sert surtout au raccordement d'un casque audio ou au branchement d'un câble auxiliaire connecté à des haut-parleurs, mais le connecteur contient aussi une prise supplémentaire qui vous permet de raccorder un moniteur vidéo composite utilisant un câble spécial. En général, le connecteur vidéo HDMI sert surtout à brancher un moniteur ou un téléviseur, car sa qualité est bien meilleure que la vidéo composite. Entre les prises HDMI et audio, vous trouverez un connecteur plat permettant de brancher une caméra spécialement conçue pour le Raspberry Pi.

À côté de la prise HDMI se trouve un connecteur micro USB qui sert uniquement à l'alimentation électrique du Raspberry Pi à l'aide d'un adaptateur 5V.

Au-dessus du connecteur d'alimentation micro USB, sur la face inférieure de la carte, se trouve un connecteur microSD. Le Raspberry Pi n'intègre pas de disque dur ordinaire ; à la place, le système d'exploitation et tous les fichiers sont stockés sur une carte microSD.

Sur la face supérieure de la carte, vous trouverez un ensemble de broches GPIO *(General-Purpose Input/Output)* qui peuvent être utilisées pour raccorder le Raspberry Pi à divers circuits électroniques.

Avant la sortie du modèle Raspberry Pi B+, les Raspberry Pi n'étaient dotés que de 26 broches sur le connecteur GPIO au lieu des 40 illustrées à la figure 3-1. Tous les projets présentés dans ce livre se contentent des 26 broches d'origine qui ont été conservées sur les modèles plus récents. Donc les projets devraient fonctionner même si vous avez un ancien Raspberry Pi.

Configuration du Raspberry Pi

Vous devez raccorder un clavier, une souris et un moniteur au Raspberry Pi pour pouvoir le configurer. Ensuite, vous pourrez débrancher tous ces périphériques et vous connecter à la nanocarte par le biais d'une connexion *Secure Socket Shell* (SSH) depuis un autre ordinateur. Mais nous n'en sommes pas encore là.

Vous avez besoin des éléments suivants pour configurer votre Raspberry Pi :

- un clavier et une souris USB (les périphériques PC standards sont parfaits)
- un moniteur ou un téléviseur avec une entrée HDMI et un câble HDMI
- une alimentation 5 V micro USB (délivrant au moins 1 ampère)
- un câble Ethernet pour le raccordement à votre routeur ou une clé Wi-Fi USB
- une carte microSD (4 Go suffisent, mais avec 8 Go vous aurez plus d'espace mémoire pour vos fichiers personnels et les programmes que vous téléchargerez ; choisissez une carte microSD de classe 10 pour des performances accrues)
- un second ordinateur doté d'un adaptateur de carte microSD pour configurer la carte mémoire (sinon, vous pouvez acheter une carte microSD avec adaptateur NOOBS – *New Out Of the Box Software* – préinstallé)

La figure 3-2 présente une configuration Raspberry Pi habituelle.

Pendant l'installation du système d'exploitation, vous devrez placer votre Raspberry Pi à proximité du routeur afin de pouvoir l'y connecter directement pour bénéficier d'une connexion Internet. Lorsque le système d'exploitation sera installé, vous pourrez configurer une clé USB Wi-Fi.

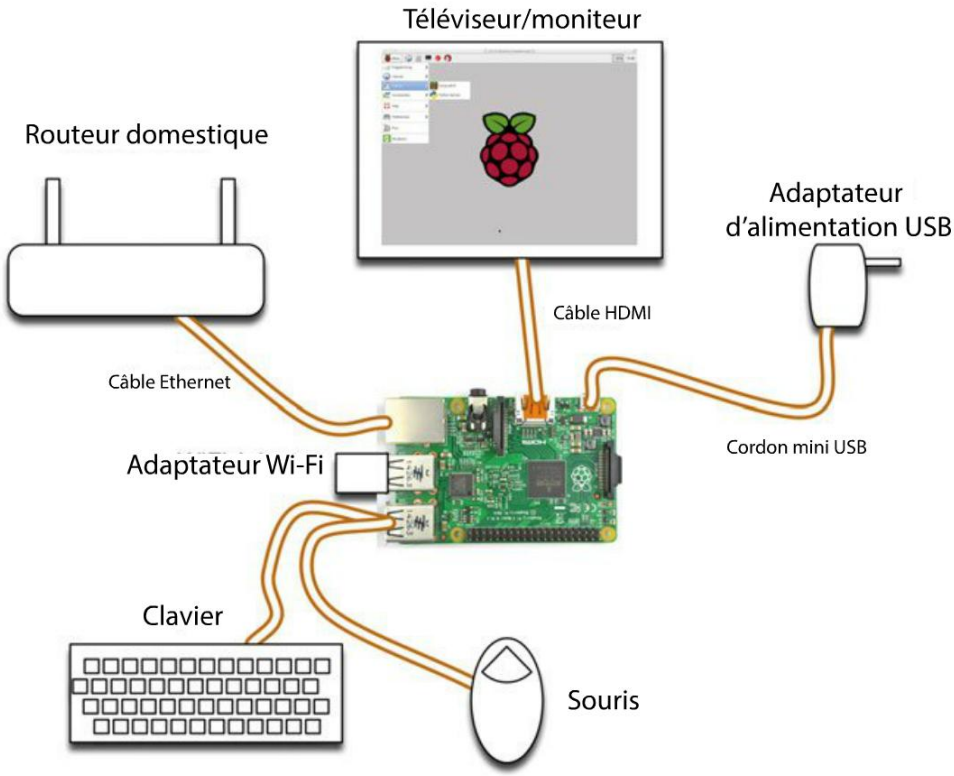


Figure 3-2. *Configuration classique d'un Raspberry Pi*

Préparation d'une carte microSD avec NOOBS

Avec la première version du Raspberry Pi, il fallait utiliser un logiciel de création d'images *(image writer)* pour configurer une carte SD. Ce n'est plus nécessaire avec NOOBS – pour l'installer, il suffit de copier les fichiers sur la carte SD sans formatage préalable.

Vous trouverez les dernières instructions d'utilisation de NOOBS pour la configuration de votre carte microSD à l'adresse suivante : *https://www.raspberrypi.org/help/noobs-setup/*.

Lorsque le Raspberry Pi est initialisé depuis NOOBS, vous avez le choix entre plusieurs systèmes d'exploitation (figure 2-3). Sélectionnez l'option Raspbian recommandée.

À l'issue d'un long processus d'écriture de fichiers et d'un ultime redémarrage, votre Raspberry Pi est prêt. Si vous disposez d'une clé Wi-Fi, c'est le moment de la configurer afin de vous connecter à votre réseau sans fil à l'aide de l'utilitaire Wi-Fi Config que vous trouverez sous la rubrique Preferences du menu du Bureau.

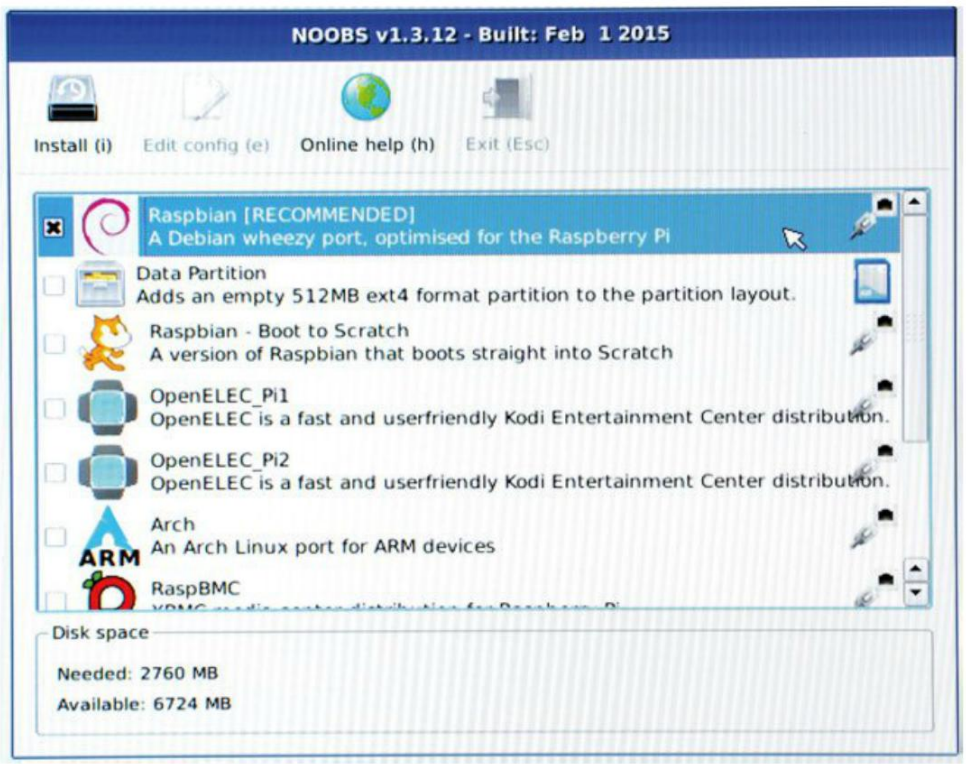


Figure 3-3. *NOOBS permet d'installer plusieurs systèmes d'exploitation au choix*

Configuration de SSH

Pour une grande partie des projets et expériences proposés dans ce livre, la nécessité de raccorder un clavier, une souris et un écran au Raspberry Pi sera un inconvénient. SSH vous permet d'accéder par des lignes de commandes à votre Raspberry Pi depuis un second ordinateur par l'intermédiaire du réseau.

Une fois la configuration terminée, les seules choses que vous devrez raccorder à votre Raspberry Pi seront son alimentation et un câble réseau ou une clé Wi-Fi.

Cliquez sur l'icône LXTerminal qui se trouve en haut du Bureau Raspberry Pi (figure 3-4). Saisissez la commande suivante (sans le signe $ – il s'agit de l'invite de commande) dans le terminal qui apparaît :

```
$ sudo raspi-config
```

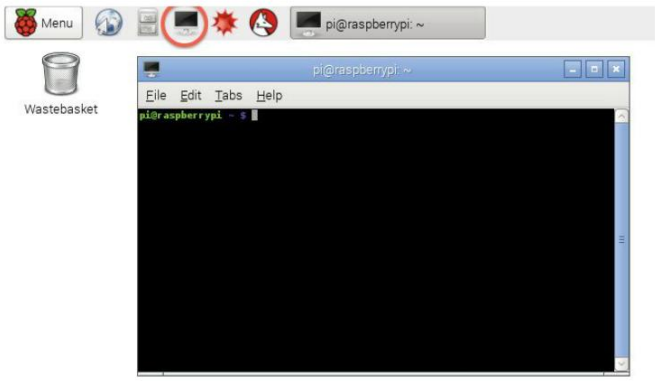


Figure 3-4. *Ouverture de LXTerminal*

Cette commande ouvre l'outil raspi-config qui permet de configurer le Raspberry Pi. Utilisez les touches fléchées pour sélectionner l'option Advanced, puis appuyez sur Entrée. Ensuite, utilisez encore les touches fléchées pour sélectionner SSH (figure 3-5).

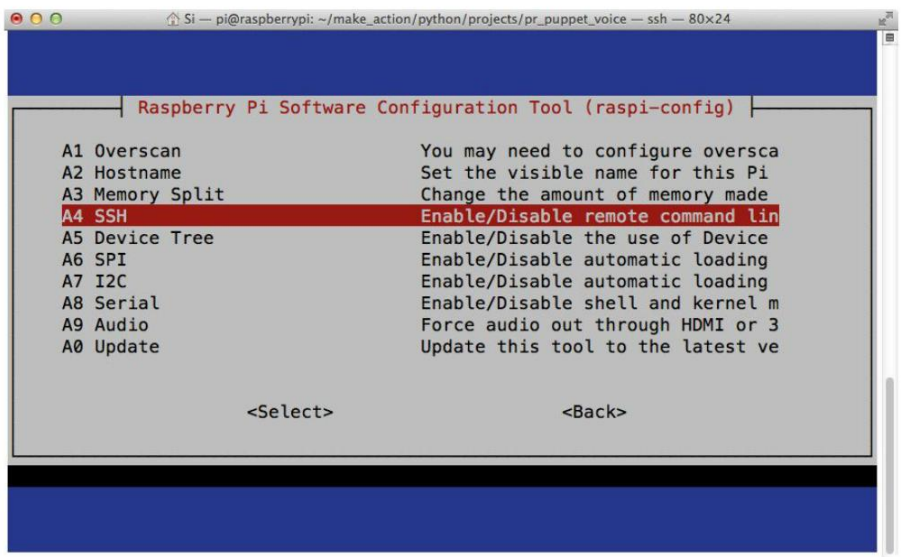


Figure 3-5. *Activation de SSH avec raspi-config*

Cliquez sur Enable, puis sélectionnez la commande Finish.

L'accès à distance par SSH est désormais activé sur votre Raspberry Pi et vous pouvez donc vous y connecter depuis votre ordinateur.

Si votre ordinateur est sous Mac ou Linux, il dispose de son propre terminal que vous pouvez utiliser pour vous connecter au Raspberry Pi. Vous pouvez donc passer directement à la section « SSH sur Mac ou Linux » page 32.

Obtenir l'adresse IP d'un Raspberry Pi

Pour pouvoir vous connecter à votre Raspberry Pi par SSH depuis un autre ordinateur connecté au réseau, vous devez connaître l'adresse IP du Raspberry Pi.

Pour l'obtenir, exécutez la commande suivante dans LXTerminal sur votre Raspberry Pi :

```
$ hostname -I
```

Vous obtenez alors un numéro en quatre parties sur le modèle suivant (il s'agit de l'adresse IP) :

```
192.168.1.23
```

SSH sur un ordinateur sous Windows

Si vous utilisez Microsoft Windows, vous devez télécharger et installer Putty (*http://www.putty.org/*).

L'installation paraît un peu étrange, mais en fait, vous téléchargez uniquement `putty.exe` qui correspond au programme proprement dit. Enregistrez-le sur le Bureau, par exemple, et lancez-le par un double-clic. La fenêtre PuTTY Configuration apparaît (figure 3-6).

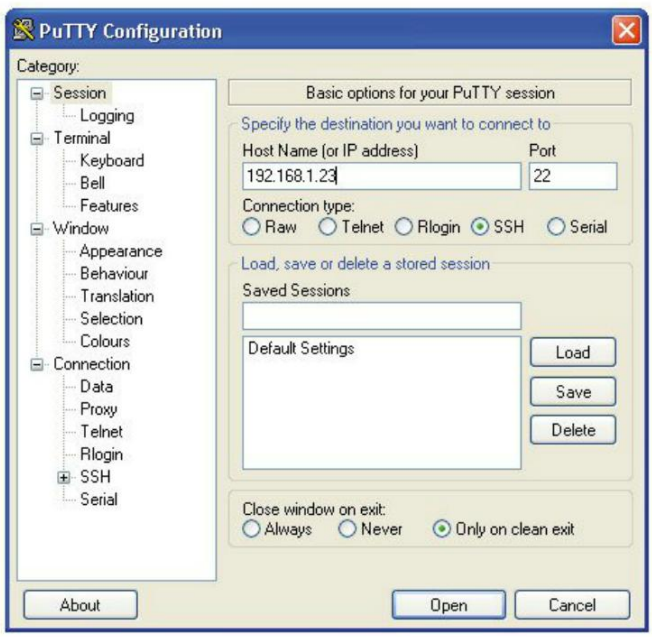


Figure 3-6. *Fenêtre PuTTY Configuration*

Saisissez l'adresse IP de votre Raspberry Pi dans le champ Host Name (or IP address) (voir l'encadré ci-dessus « Obtenir l'adresse IP d'un Raspberry Pi »), puis cliquez sur Open. Vous devez ensuite saisir votre identifiant et votre mot de passe pour vous connecter à votre Pi (figure 3-7). Comme

identifiant, saisissez `pi` et comme mot de passe `raspberry`. Cela suffit pour vous connecter. Vous pouvez maintenant saisir des commandes dans PuTTY sur votre ordinateur principal afin de les exécuter sur votre Raspberry Pi.

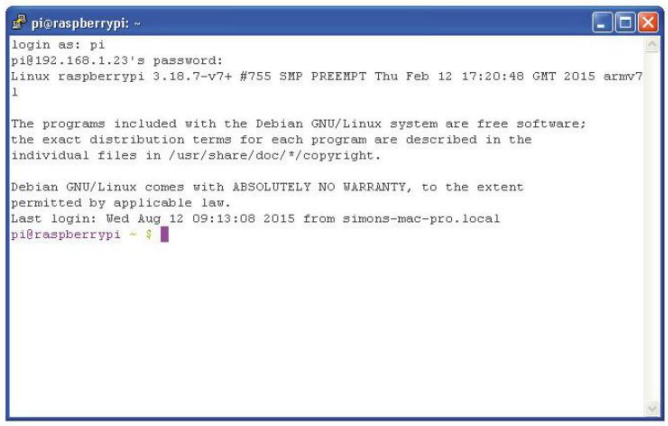


Figure 3-7. *Contrôle à distance du Raspberry Pi par SSH*

SSH sur Mac ou Linux

Si vous utilisez un ordinateur sous Mac ou Linux, le logiciel nécessaire à la connexion au Raspberry Pi se trouve déjà sur l'ordinateur. Ouvrez le Terminal et saisissez la commande suivante en remplaçant l'adresse IP de mon Raspberry Pi (192.168.1.23) par celle du vôtre :

```
$ ssh 192.168.1.23 -l pi
```

La première fois que vous le ferez, vous verrez apparaître le message suivant :

```
The authenticity of host '192.168.1.23 (192.168.1.23)' can't be established.
RSA key fingerprint is 48:8f:c3:07:c2:04:9e:8b:59:ed:53:2b:0b:d0:aa:e5.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.23' (RSA) to the list of known hosts.
pi@192.168.1.23's password:
```

Continuez en confirmant l'authenticité de l'ordinateur en saisissant `yes`. Désormais, toutes les commandes saisies sont exécutées sur le Raspberry Pi et non sur votre ordinateur.

La ligne de commande Linux

Si vous utilisez un ordinateur sous Windows ou Mac, vous n'aurez peut-être jamais eu besoin d'utiliser la ligne de commande pour interagir avec votre ordinateur. Le système d'exploitation Linux qui fonctionne sur votre Raspberry Pi nécessite toutefois que vous saisissiez des commandes dans une ligne de commande pour installer des logiciels, copier, renommer et modifier des fichiers ou exécuter des programmes.

Comme vous avez déjà utilisé la ligne de commande du Raspberry Pi dans LXTerminal pour configurer le protocole SSH, vous pouvez maintenant exécuter des commandes sur le Raspberry Pi par SSH ou directement dans LXTerminal.

Peut-être avez-vous remarqué que lorsque LXTerminal ou la session SSH est prêt à recevoir une commande, le dernier caractère de la ligne est le signe `$`. Il s'agit de l'invite dollar par laquelle Linux vous indique qu'il est prêt à traiter la commande suivante.

La ligne de commande Linux reprend le concept du dossier courant, c'est-à-dire du dossier dans lequel vous travaillez actuellement. Par conséquent, si vous voulez exécuter un programme Python qui se trouve dans un dossier précis, vous devez accéder à ce dossier avant d'exécuter le programme qui s'y trouve. La commande qui permet de le faire se nomme `cd` (*change directory)*.

Au premier démarrage d'une session LXTerminal, le dossier courant est `/home/pi`. C'est ce que l'on appelle le dossier home. Si votre dossier `home` contient un sous-dossier nommé `make_action`, vous pouvez y accéder en saisissant la commande suivante afin de changer de dossier par rapport au dossier courant) :

```
$ cd make_action
```

Vous pouvez aussi saisir le chemin d'accès complet au dossier comme suit :

```
$ cd /home/pi/make_action
```

Dans ce livre, tout le code Raspberry Pi est écrit en langage de programmation Python. Pour exécuter un programme Python qui s'appelle `test.py`, vous utiliseriez la commande :

```
$ python test.py
```

Parmi les autres commandes fréquemment employées, il y a `sudo` *(substitute user do)*. Elle vous permet de demander l'exécution de la commande qui la suit en tant que superutilisateur. Linux s'efforce de vous empêcher d'endommager votre système par inadvertance et exige un accès en tant que superutilisateur pour supprimer des fichiers importants ou pour exécuter certaines actions. Ce niveau d'accès est aussi requis pour accéder aux broches GPIO que vous utiliserez souvent pour les projets de ce livre.

Par exemple, si test.py utilise les broches GPIO, la commande suivante permet de l'exécuter :

```
$ sudo python test.py
```

Même si tout le code Python de ce livre est disponible en téléchargement (voir la section « Le code du livre » page suivante), vous voudrez parfois modifier un programme ou un fichier quelconque. Vous pourrez le faire dans l'éditeur nano par l'intermédiaire de SSH. Pour modifier un fichier, saisissez la commande `nano` suivie du nom du fichier (si le fichier n'existe pas, nano le crée lorsque vous enregistrez vos saisies) :

```
$ nano test.py
```

La figure 3-8 montre l'éditeur nano en action.

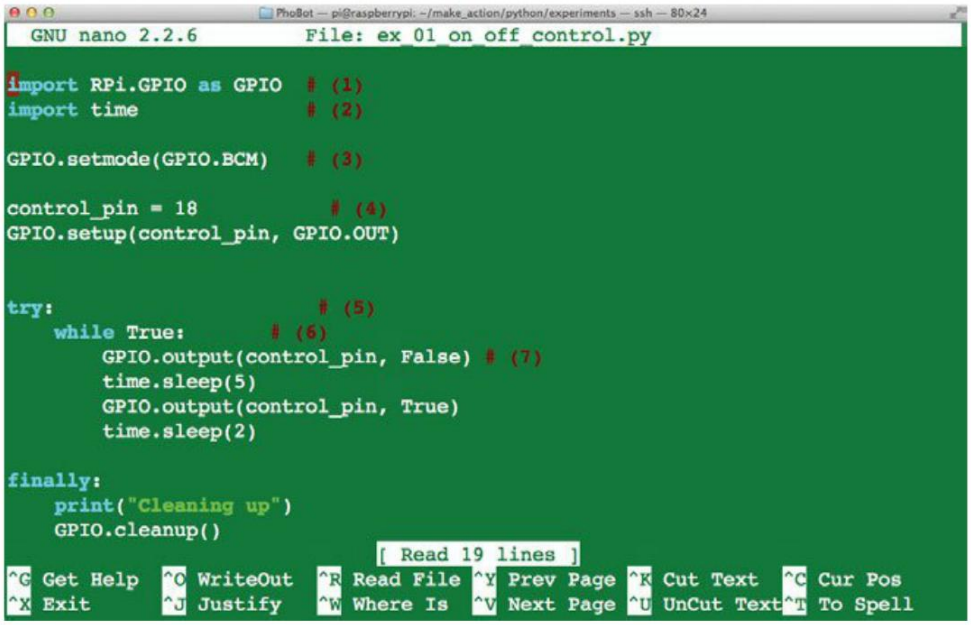


Figure 3-8. *Éditeur nano*

Comme nano a été conçu pour fonctionner dans un environnement à ligne de commande, comme LXTerminal ou par SSH, vous devez vous servir des touches fléchées pour parcourir le fichier au lieu d'utiliser la souris. Quand vous êtes prêt à enregistrer le fichier, appuyez sur Ctrl-X, puis sur Y suivie de la touche Entrée pour confirmer l'enregistrement.

Le code du livre

Le Raspberry Pi ne nécessite pas d'ordinateur distinct pour la programmation, donc il vous suffit de transférer tous les fichiers de programme utilisés dans ce livre sur le Raspberry Pi.

Même si vous pouvez les transférer sous la forme d'une archive ZIP, comme vous le feriez pour le code Arduino (chapitre 2), mieux vaut utiliser Git, qui est un outil de gestion de versions.

Git conserve la chronologie de toutes les modifications apportées au code et cet utilitaire permet même de fusionner le code de différents programmeurs. L'hébergeur de code source public GitHub utilise Git. L'outil étant préinstallé sur Raspbian, pour récupérer tout le code sur votre Raspberry Pi, il vous suffit d'ouvrir une session LXTerminal et d'exécuter la commande suivante :

```
$ git clone https://github.com/simonmonk/make_action.git
```

Cela crée un dossier intitulé `make_action` dans lequel vous trouverez un dossier `python` contenant deux sous-dossiers intitulés `experiments` et `projects`.

L'avantage d'utiliser Git pour récupérer le code du programme est que chaque fois que vous voulez vérifier la disponibilité de corrections mineures ou d'améliorations apportées au code du livre, vous pourrez les obtenir en saisissant simplement :

```
$ git pull
```

Guide de programmation

La meilleure façon d'apprendre à programmer est de commencer par modifier des programmes simples existants afin de comprendre progressivement comment ils fonctionnent. Tous les programmes dont vous aurez besoin pour réaliser les projets de ce livre sont disponibles en téléchargement, donc nul besoin d'être un expert en programmation. Toutefois, il est préférable d'avoir une vague idée de la façon dont cela fonctionne.

Hello, World

Traditionnellement, lorsque l'on débute dans un nouveau langage, le premier programme que l'on apprend à écrire permet d'afficher les mots « Hello, World » à l'écran. Pour effectuer ce test, ouvrez nano à l'aide de la commande suivante :

```
$ nano hello.py
```

L'extension de fichier `.py` identifie le fichier de programme comme étant Python. Ensuite, saisissez le texte suivant dans l'éditeur nano, puis enregistrez le fichier :

```
print('Hello, World')
```

Enfin, exécutez le programme :

```
$ python hello.py
Hello, World
```

Python 2 contre Python 3

Les utilisateurs de Python ont du mal à se séparer des anciennes versions du langage. Beaucoup en sont restés à Python 2 même si la version 3 est disponible.

Les deux versions sont incluses dans Raspbian. Pour exécuter votre programme « Hello, World » en Python 3, saisissez :

```
$ python3 hello.py
```

Le résultat est exactement le même.

Alors pourquoi tant d'utilisateurs de Python s'en tiennent-ils à la version 2, comme je le fais dans de ce livre ?

Cela s'explique par le fait que les bibliothèques Python n'ont pas toutes été converties en Python 3. C'est notamment le cas de la bibliothèque d'interfaces série de I2C que vous utiliserez au chapitre 14 qui n'est pas compatible avec Python 3 au moment où j'écris ces lignes.

Tabulations et retraits

Les programmeurs présentent leur code de façon à le rendre plus compréhensible. En Arduino C, les blocs de code qui se rapportent à une fonction ou à une instruction `if` sont mis en retrait. Un simple coup d'œil suffit pour savoir quelles lignes de code appartiennent à quelle commande ou fonction. En Python, ce n'est pas qu'une question de préférence personnelle.

En Python, il n'y a pas d'accolades { ou } permettant d'identifier le début et la fin d'un bloc de code – il faut utiliser des retraits pour savoir quelles lignes appartiennent à la même fonction.

Examinez par exemple le fragment de code suivant :

```
while True :
   GPIO.output(control_pin, False)
   time.sleep(5)
   GPIO.output(control_pin, True)
   time.sleep(2)
print(«Finished»)
```

Il s'agit d'une boucle `while` (comme en Arduino C, voir la section « Boucles » du chapitre 2 page 21). Ici, la condition est `True`. La valeur `True` s'applique toujours et la boucle continue à l'infini. La première ligne se termine par deux points (`:`). Ce signe de ponctuation indique que ce qui suit sera un bloc de code. C'est donc l'équivalent en Python de l'accolade `{` en Arduino C, mais il n'y a pas de signe de fermeture.

Les lignes qui se trouvent à l'intérieur du bloc doivent être mises en retrait. Du moment que le retrait est homogène, le nombre d'espaces (ou de tabulations) n'a pas d'importance. Les programmeurs ont tendance à utiliser quatre caractères d'espace pour chaque niveau de retrait.

Il n'y a pas de marque à la fin du bloc, mais le code n'est plus mis en retrait. Dans l'exemple précédant, la commande `print` finale n'est pas comprise dans la boucle. Comme la boucle `while` continue à l'infini, cette dernière ligne ne sera effectivement jamais exécutée.

Variables

Les variables en Python ressemblent à celles d'Arduino C. Toutefois, lorsque vous en utilisez une pour la première fois, vous n'avez pas besoin de préciser s'il s'agit d'un `int`, d'un `float` ou autre. Vous devez lui affecter une valeur et rien ne nous empêche de définir la même variable comme étant un nombre entier ou une chaîne de caractères. Par exemple, le code suivant est parfaitement admis, même s'il n'a aucun sens :

```
a = 123.45
a = «message»
```

Par ailleurs, lorsque vous utilisez une chaîne de caractères, vous pouvez la mettre entre guillemets (comme dans l'exemple précédent) ou entre apostrophes.

if, while, etc.

Comme Arduino C, Python dispose aussi d'une structure `if` et `else`, mais les blocs de code à l'intérieur de l'instruction `if` sont repérés par le signe deux points et des retraits :

```
if x > 10 :
    print("x is big!")
else:
   print("x is small")
```

Au fil de la lecture de ce livre, vous en apprendrez davantage sur les boucles et les autres structures de programmation.

La bibliothèque RPi.GPIO

Comme Arduino C, Python utilise des bibliothèques (*libraries*, en anglais) qu'il vous suffit d'importer pour les intégrer à un programme.

Dans ce livre, nous utiliserons souvent le connecteur GPIO du Raspberry Pi. La bibliothèque Python généralement employée pour contrôler les broches GPIO, et celle que l'on utilise aussi dans ce livre, se nomme RPi.GPIO.

Cette bibliothèque est préinstallée avec Raspbian, donc vous n'avez rien de plus à installer ; vous pouvez simplement l'importer dans votre programme pour commencer à l'utiliser.

Pour compliquer les choses, il y a deux conventions de désignation des broches du connecteur GPIO du Raspberry Pi : l'une se base sur les positions des broches sur le connecteur (1, 2, 3, 4, etc.) et l'autre utilise les fonctions des broches. Aux débuts du Raspberry Pi, elles étaient toutes les deux utilisées assez fréquemment. Aujourd'hui, les broches sont généralement désignées d'après leur fonction. La bibliothèque RPi.GPIO est compatible avec les deux méthodes, mais vous devez préciser laquelle est employée. C'est pourquoi vous trouverez le code suivant au début de tous les programmes Python figurant dans ce livre :

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
```

La première ligne importe la bibliothèque RPi.GPIO et la seconde spécifie la puce Broadcom (BMC) qui se trouve au cœur du Raspberry Pi.

Contrairement à l'Arduino, le Raspberry Pi n'a pas d'entrées analogiques. La bibliothèque RPi.GPIO est néanmoins compatible avec des sorties analogiques PWM (MLI, en français).

Le connecteur GPIO

La figure 3-9 présente les ports disponibles sur le connecteur GPIO. Si vous avez un ancien modèle de Raspberry Pi à 26 broches seulement, vous n'y trouverez pas les broches qui figurent au-dessous de la ligne en pointillé. Comme les projets présentés dans ce livre peuvent être réalisés sur tous les modèles de Raspberry Pi, nous n'utiliserons que ces 26 broches.

Ce schéma est repris à l'annexe B à la fin du livre. Contrairement à l'Arduino, le nom des broches ne figure pas sur le circuit imprimé du Raspberry Pi. Il est donc moins facile de s'y retrouver. Néanmoins, vous pouvez vous procurer ou fabriquer un gabarit à poser sur les broches, comme le Raspberry Leaf.

Notez que certaines broches sont désignées par un numéro (2, par exemple), mais aussi une fonction (SDA, par exemple). Quand vous utilisez les broches GPIO dans votre code, vous y faites uniquement référence par leur numéro.

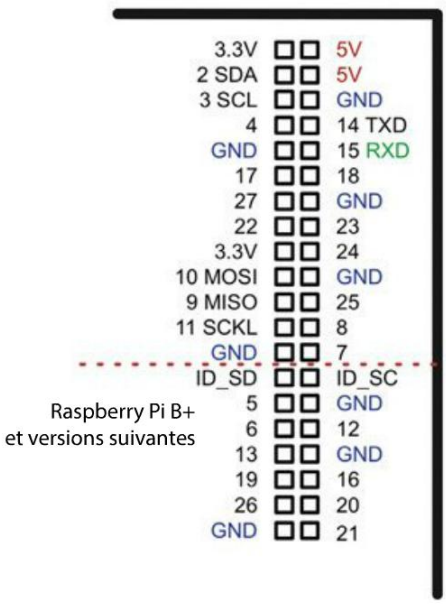


Figure 3-9. *Le connecteur GPIO*

Sorties numériques

L'exemple ci-dessous montre comment définir la broche GPIO 18 (figure 3-9) en tant que sortie numérique, puis comment régler la sortie sur l'état haut :

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
GPIO.output(18, True)
```

Lorsque vous définissez une sortie à l'aide de `GPIO.output`, vous pouvez la régler sur l'état haut à l'aide de la valeur `True` ou `1` et sur l'état bas à l'aide de `False` ou `0`.

Entrées numériques

Les entrées numériques du Raspberry Pi fonctionnent de la même façon que celles d'une carte Arduino :

```
GPIO.setup(18, GPIO.IN)
value = GPIO.input(18)
print(value)
```

Vous pouvez aussi préciser que l'entrée est munie d'une résistance pull-up (voir la section « Entrées numériques » du chapitre 2 page 17) comme suit :

```
GPIO.setup(switch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

Sorties analogiques

Pour utiliser des sorties analogiques sur un Raspberry Pi, il faut procéder en deux temps. Il faut d'abord configurer la broche comme sortie, puis il faut définir un canal PWM qui utilise cette sortie.

Vous en trouverez un exemple commenté à l'expérience « Arc-en-ciel de couleurs », décrite au chapitre 6.

Résumé

Dans ce chapitre, vous avez appris à configurer votre Raspberry Pi et à maîtriser la ligne de commande Linux. Vous vous êtes également familiarisé avec les bases du langage Python dont vous aurez besoin pour contrôler des composants via les broches GPIO.

Il est donc grand temps de passer à la pratique. Au chapitre 4, vous allez pouvoir commencer à utiliser votre Arduino et votre Raspberry Pi.

Premières expériences

4

Il est toujours préférable de compléter des connaissances théoriques par de la pratique, quand on en a la possibilité. Et c'est aussi beaucoup plus amusant. Dans ce chapitre, vous apprendrez à utiliser une plaque d'essai sans soudure ou *breadboard* avec quelques composants électroniques. Vous commanderez une LED, puis un moteur, depuis votre Arduino ou votre Raspberry Pi.

Plaque d'essai sans soudure (breadboard)

En général, lorsque vous raccordez des composants électroniques externes à un Raspberry Pi ou un Arduino, il n'est pas possible de connecter l'appareil (un moteur, par exemple) directement sur la platine. D'autres composants électroniques sont nécessaires pour que l'ensemble fonctionne. Même si vous vous contentez d'allumer une LED, vous devrez la relier au Raspberry Pi ou à l'Arduino.

Pour raccorder facilement des composants, vous pouvez utiliser une plaque d'essai sans soudure ou breadboard. À l'origine, il s'agit d'un outil de prototypage destiné aux électroniciens qui s'en servent pour tester leurs montages avant d'en réaliser une version soudée qui est permanente. Les platines d'expérimentation vous permettent de tester différents montages et de réaliser vos propres projets sans souder les composants.

La figure 4-1 présente une plaque d'essai sur laquelle sont connectés les composants nécessaires pour la réalisation de l'expérience « Pilotage d'un moteur », présentée plus loin.

La platine d'expérimentation est pratique, car elle sert à la fois de support et au branchement de composants et de fils.

La plaque d'essai utilisée pour les montages de ce livre est un modèle demi-taille de 400 points (car elle compte 400 trous). Il existe des modèles plus petits et plus grands, mais cette taille convient pour tous les projets et expériences de ce livre.

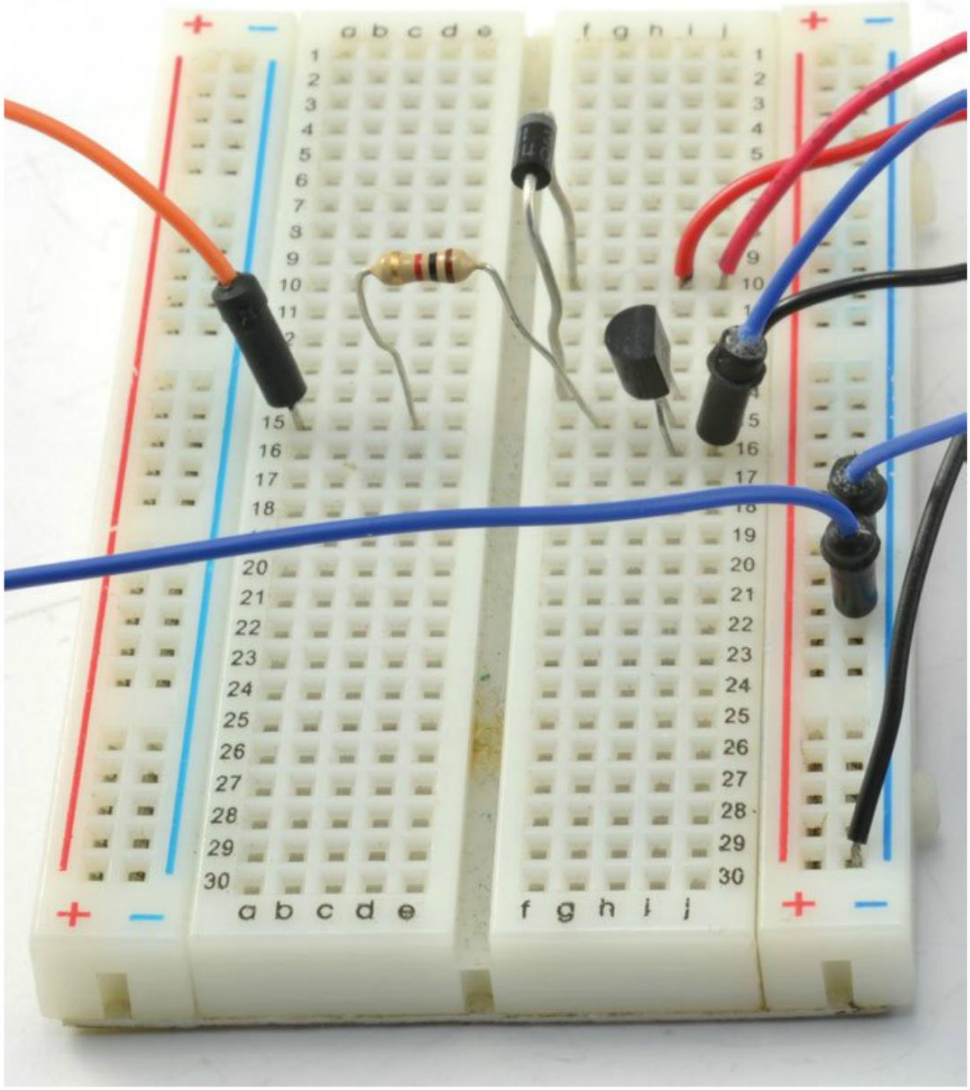


Figure 4-1. *Utilisation d'une plaque d'essai sans soudure*

Ce type de plaque dispose de séries de trous organisées sur deux colonnes de part et d'autre de la platine qui sont repérées par des lignes rouges et bleues. Tous les trous de chacune de ces colonnes sont raccordés électriquement sous la surface plastique. Même si ces colonnes sont polyvalentes, elles sont généralement utilisées pour l'alimentation positive et négative de votre circuit.

La partie principale de la carte est divisée en deux blocs ou rangées de cinq trous sur toute la longueur de la carte. Les cinq trous de chacune de ces rangées sont reliés entre eux par une barrette cachée sous le plastique. Pour connecter la patte d'un composant à celle d'un autre, les deux conducteurs doivent simplement être enfichés dans les trous de la même rangée sur la plaque.

Fonctionnement d'une plaque d'essai

Des barrettes métalliques se cachent derrière les trous de la façade en plastique d'une plaque d'essai. C'est là que viennent s'enficher les liaisons électriques et les conducteurs des composants.

La figure 4-2 présente une platine démontée dont l'une des barrettes a été extraite. Je vous déconseille de démonter votre plaque, car cela risque de nuire à son bon fonctionnement.

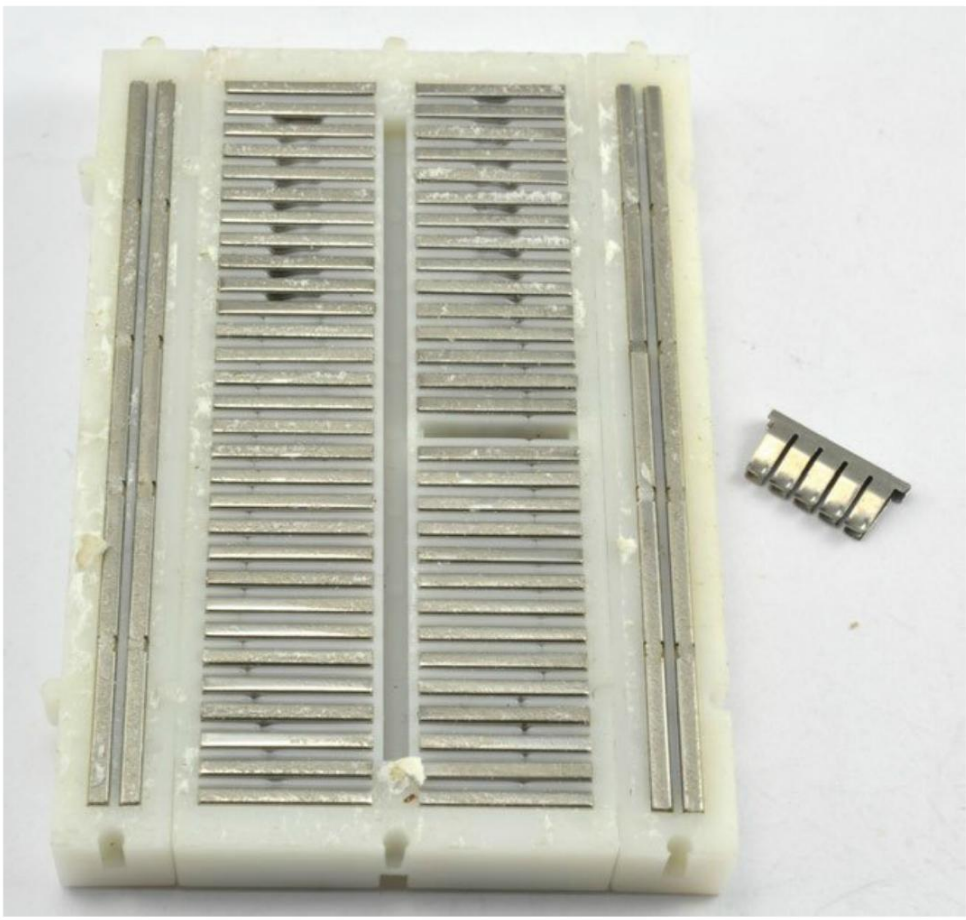


Figure 4-2. *L'intérieur d'une breadboard dont l'une des barrettes a été retirée.*

Raccordement d'une plaque d'essai à l'Arduino

Les connexions GPIO de l'Arduino (souvent appelées broches, ce qui prête à confusion) sont des fiches. Pour raccorder l'une de ces broches à l'une des rangées d'une plaque d'essai, vous pouvez utiliser un câble flexible mâle/mâle, comme illustré à la figure 4-3.

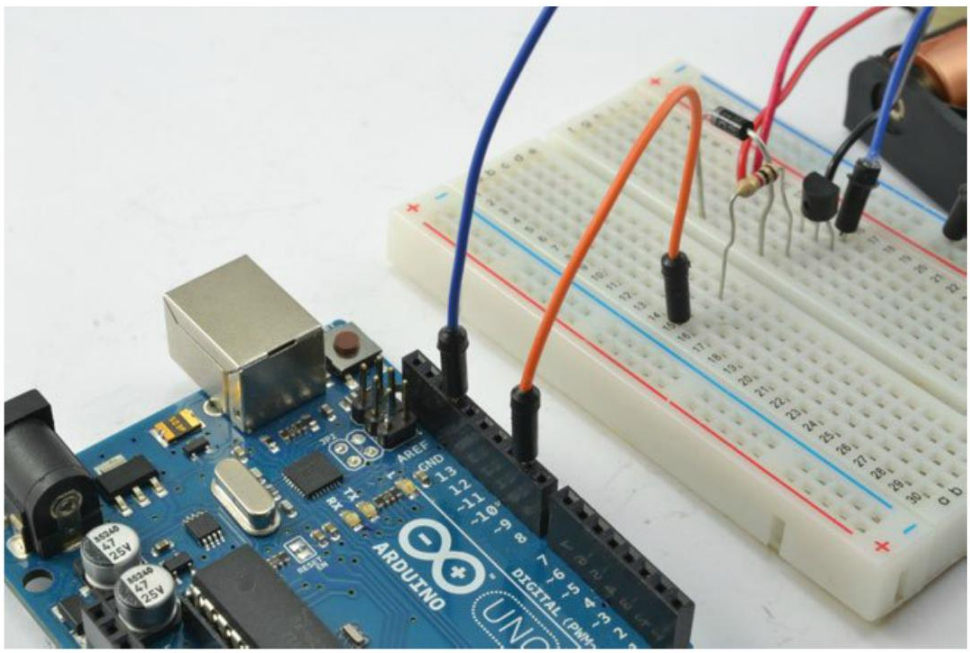


Figure 4-3. *Raccordement de l'Arduino à une breadboard*

Ces câbles flexibles sont composés d'un fil souple muni d'un embout à chaque extrémité. Prévoyez une réserve suffisante de câbles de différentes longueurs.

Vous pouvez acheter des lots de câbles flexibles de différentes tailles et couleurs (voir l'annexe A). Procurez-vous un kit composé d'une plaque d'essai et d'un assortiment de câbles. Les différentes couleurs permettent de mieux identifier les liaisons, surtout lorsque la plaque d'essai est recouverte de fils.

Raccordement d'une plaque d'essai au Raspberry Pi

Le connecteur GPIO d'un Raspberry Pi se compose de broches et non de fiches. Par conséquent, vous ne pouvez pas utiliser les câbles flexibles mâles/mâles employés pour les branchements sur l'Arduino. À la place, vous devez utiliser des câbles flexibles femelles/mâles : l'une des extrémités est munie d'un embout qui vient se brancher sur une broche du connecteur GPIO du Raspberry Pi et l'autre extrémité du conducteur se termine par une broche qui vient s'enficher dans la plaque d'essai.

La figure 4-4 montre comment connecter les broches GPIO d'un Raspberry Pi à une rangée de la plaque d'essai à l'aide de câbles flexibles femelles-mâles.

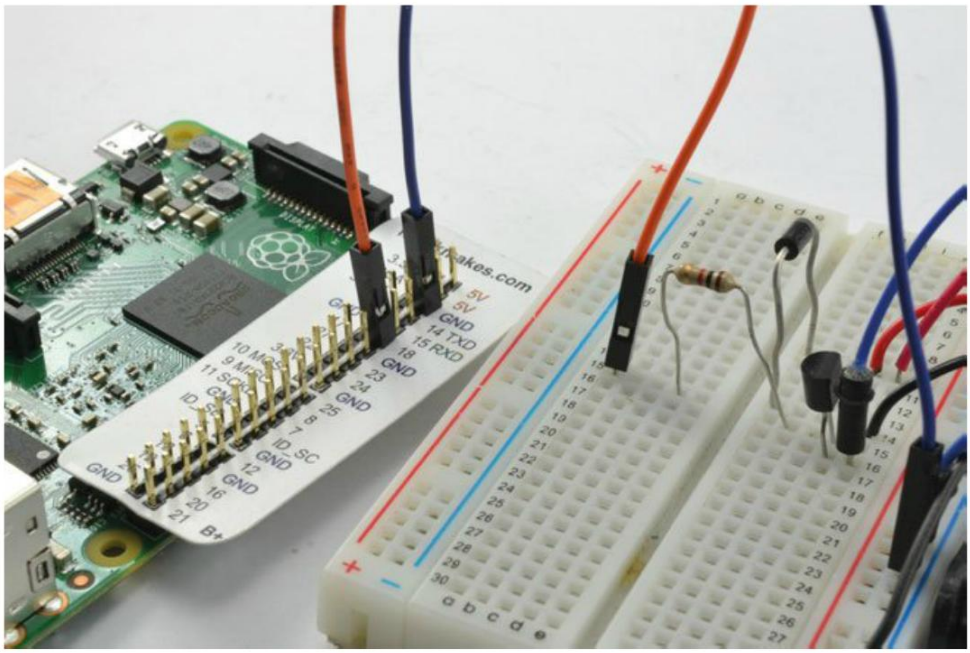


Figure 4-4. *Raccordement d'un Raspberry Pi*

Identification des broches GPIO d'un Raspberry Pi

Les broches GPIO d'un Raspberry Pi ne sont pas repérées sur la carte. Par conséquent, pour éviter de passer votre temps à les compter, vous pouvez enfiler un gabarit sur le connecteur GPIO. Celui illustré à la figure 4-4 est un Raspberry Leaf qui est disponible chez Adafruit et MonkMakes.com. Il en existe aussi d'autres modèles.

Téléchargement des programmes

Tous les programmes de ce livre – que ce soit les sketches Arduino ou les programmes Python pour le Raspberry Pi – sont mis à disposition sur la page GitHub du livre (*https://github.com/simonmonk/make_action*).

Pour plus d'informations sur le transfert des sketches Arduino depuis votre ordinateur sur la carte Arduino, voir la section « Le code du livre » du chapitre 2 page 14.

Pour plus d'informations sur le transfert des programmes Python sur votre Raspberry Pi, voir la section « Le code du livre » du chapitre 3 page 34.

Expérience : pilotage d'une LED

Dans le monde de l'Arduino, la coutume veut que le premier exercice réalisé consiste à faire clignoter une LED. Dans cette première expérience, vous commanderez d'abord une LED à l'aide d'une carte Arduino, puis à partir d'un Raspberry Pi.

C'est un projet facile destiné aux débutants. Deux composants seulement doivent être connectés sur la platine d'essai : une LED et une résistance. Toutes les LED nécessitent une résistance afin de limiter le courant qui les traverse. Nous y reviendrons au chapitre 6.

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou un Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser l'expérience.

Composant	Source
LED rouge	Adafruit : 297 Sparkfun : COM-09590
Résistance 470 Ω ¼ W	Mouser : 291-470-RC
Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
Câbles flexibles mâles/mâles (Arduino uniquement)	Adafruit : 758
Câbles flexibles femelles/mâles (Pi uniquement)	Adafruit : 826

Ce tableau reprend le nom du fournisseur et la référence de chaque composant. Vous trouverez plus d'informations sur tous les composants utilisés dans ce livre à l'annexe A.

Réalisation du circuit

Le circuit de ce projet est illustré à la figure 4-5. La réalisation est la même, que vous utilisiez un Arduino ou un Raspberry Pi, mais le raccordement de la plaque d'essai au Raspberry Pi ou à l'Arduino ne s'effectue pas de la même façon.

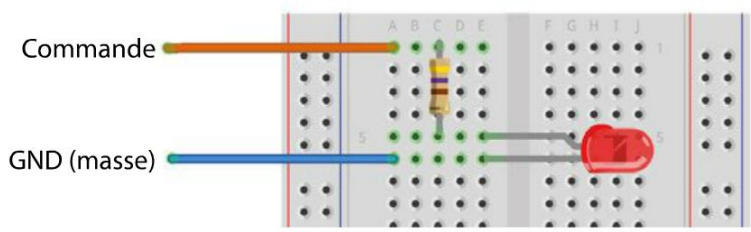


Figure 4-5. *Réalisation du circuit de pilotage d'une LED*

La figure 4-5 montre que le courant transmis par une broche de sortie d'un Arduino ou d'un Raspberry Pi traverse d'abord la résistance avant d'atteindre la LED et l'allumer.

Le sens de circulation du courant dans la résistance n'a pas d'importance, mais le conducteur positif de la LED doit être dirigé vers le haut de la platine. Le conducteur positif d'une LED est légèrement plus long que le conducteur négatif. En outre, la LED a une face plate du côté du conducteur négatif.

Les résistances seront traitées en détail au chapitre 5.

Montage Arduino

Raccordez les ports GND et D9, comme illustré à la figure 4-6. La figure 4-7 montre l'Arduino et la carte d'essais.

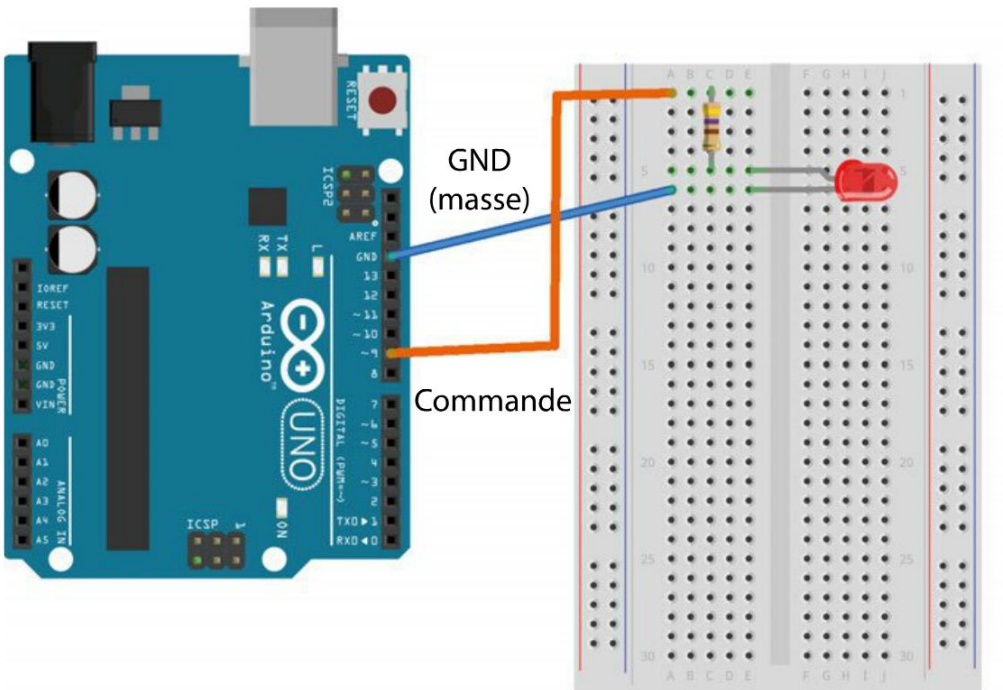


Figure 4-6. *Réalisation du circuit de pilotage d'une LED avec Arduino*

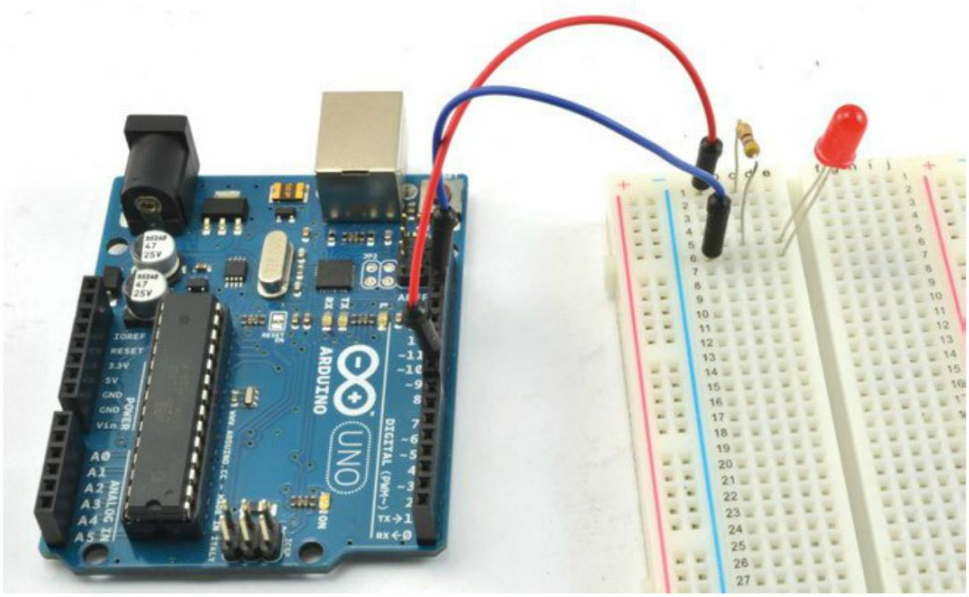


Figure 4-7. *Utilisation d'une plaque d'essai avec Arduino*

Programme Arduino

Vous trouverez le sketch Arduino dans le dossier `arduino/experiments/on_off_control` à l'endroit où vous avez téléchargé le code du livre (voir la section « Téléchargement des programmes » page 45).

Le programme allume la LED pendant 5 secondes, puis il l'éteint pendant 2 secondes, avant de recommencer. Voici l'intégralité du code :

```
const int controlPin = 9; ❶

void setup() { ❷
  pinMode(controlPin, OUTPUT);
}

void loop() { ❸
  digitalWrite(controlPin, HIGH);
  delay(5000);
  digitalWrite(controlPin, LOW);
  delay(2000);
}
```

❶ La première ligne définit une constante nommée `controlPin` comme étant la broche 9. Bien que l'Arduino Uno a des broches numériques numérotées de 0 à 13 et des broches analogiques numérotées de 0 à 5, lorsque le code Arduino fait référence à un numéro seulement (ici, 9), l'usage veut qu'il s'agisse d'une broche numérique. Pour désigner l'une des six broches analogiques, vous devez précéder le numéro de la broche de la lettre A.

❷ La fonction `setup()` définit la broche comme étant une sortie numérique à l'aide de `pinMode`.

❸ La fonction `loop()` qui est répétée à l'infini définit tout d'abord le niveau haut (5 V) pour `controlPin` (9) afin d'allumer la diode. Elle impose ensuite un délai de 5 000 millisecondes (5 secondes) avant de définir `controlPin` sur le niveau bas (0 V) afin d'éteindre la diode. La ligne suivante impose un nouveau délai de 2 secondes avant de recommencer la boucle au début.

Expérimentation Arduino

Téléversez le programme sur votre carte Arduino. Dès que la carte aura été réinitialisée, à la fin du transfert, le code sera exécuté et la LED clignotera.

Si la diode ne clignote pas, vérifiez les branchements et assurez-vous que la LED est enfichée dans le bon sens (le conducteur le plus long doit être dirigé vers le haut de la plaque d'essai).

Essayez de changer les nombres saisis dans les fonctions de délais pour modifier la durée d'activation de la diode dans chaque cycle. Vous devez téléverser à nouveau le programme à chaque correction.

Montage Raspberry Pi

Contrairement aux broches de la carte Arduino, les connecteurs GPIO du Raspberry Pi ne sont pas étiquetés. Vous avez deux possibilités : vous pouvez vous référer à un schéma du brochage GPIO (voir l'annexe B) pour retrouver la broche voulue ; ou bien, vous pouvez utiliser un gabarit d'identification des broches que vous placerez sur le connecteur GPIO, comme le Raspberry Leaf illustré à la figure 4-8. La figure 4-9 montre la réalisation du circuit et les connexions entre la plaque d'essai et le Raspberry Pi.

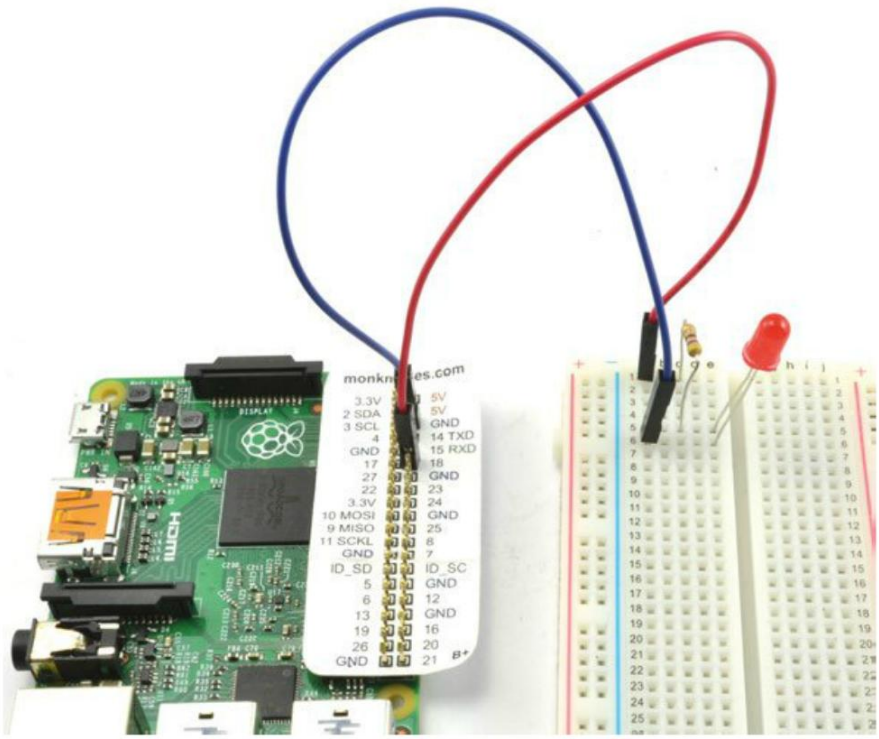


Figure 4-8. *Raccordement d'une plaque d'essai à un Raspberry Pi*

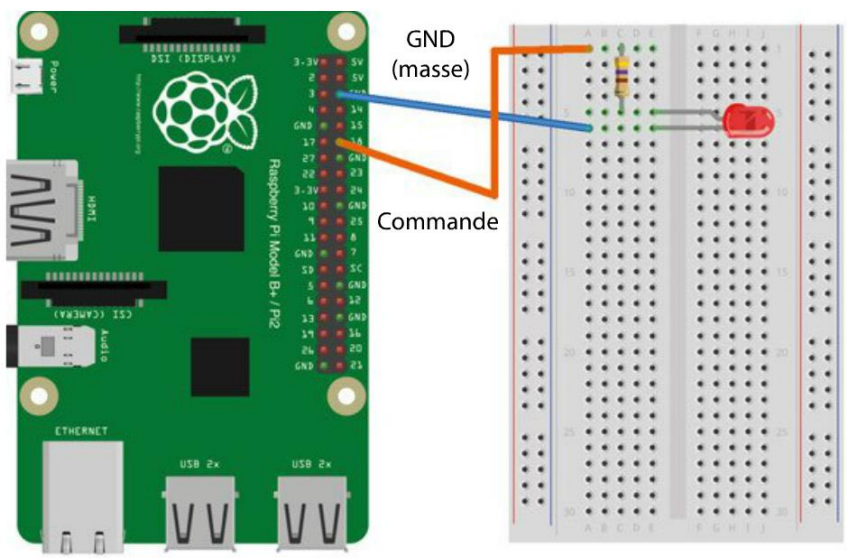


Figure 4-9. *Réalisation du circuit de pilotage d'une LED avec un Raspberry Pi*

Programme Raspberry Pi

Inutile de passer par un autre ordinateur pour programmer le Raspberry Pi – vous pouvez écrire et exécuter directement le programme sur le nano-ordinateur. Utilisez le programme suivant que vous trouverez dans le fichier `on_off_control.py` du dossier `python/experiments` :

```
import RPi.GPIO as GPIO                    ❶
import time                                ❷

GPIO.setmode(GPIO.BCM)                     ❸

control_pin = 18                           ❹
GPIO.setup(control_pin, GPIO.OUT)

try:                                       ❺
    while True:                            ❻
        GPIO.output(control_pin, False)    ❼
        time.sleep(5)
        GPIO.output(control_pin, True)
        time.sleep(2)

finally:
    print("Cleaning up")
    GPIO.cleanup()
```

Le programme ressemble beaucoup au sketch Arduino :

❶ Pour accéder aux broches GPIO du Raspberry Pi, on utilise une bibliothèque Python intitulée RPi.GPIO qui a été créée par Ben Croston, passionné du Raspberry Pi. La première ligne de code importe cette bibliothèque afin qu'elle puisse être utilisée dans le programme. La bibliothèque RPi.GPIO est préinstallée avec toutes les versions récentes de la distribution standard de Raspbian. Vous n'avez donc pas besoin de l'installer, à moins que vous n'utilisiez une ancienne version de Raspbian. Dans ce cas, mieux vaut mettre à jour votre système en saisissant la commande suivante dans le terminal :

```
$ sudo apt-get upgrade
```

❷ La bibliothèque `time` doit aussi être importée, car elle permet de définir les délais d'activation et de désactivation de la diode.

❸ La ligne `GPIO.setmode(GPIO.BCM)` doit être incluse dans tous les programmes Python servant à commander les broches GPIO avant de définir le mode des broches ou de les utiliser d'une quelconque façon. La commande indique à la bibliothèque GPIO que les broches doivent être identifiées par leur nom Broadcom (BCM) plutôt que par leur position. La bibliothèque RPi.GPIO admet les deux méthodes de désignation, mais la convention Broadcom est la plus répandue. C'est aussi celle qui est utilisée dans ce livre.

Il n'y a pas de fonctions `setup()` et `loop()` distinctes comme en programmation Arduino. À la place, tout ce qui serait défini dans la fonction `setup()` apparaît simplement au début du programme et une boucle `while` exécutée à l'infini se charge de tout ce qui se trouverait normalement dans la fonction `loop()` en Arduino.

❹ La variable `control_pin` désigne la broche GPIO 18 comme étant la broche utilisée pour commander la LED. Puis cette broche est définie en tant que sortie à l'aide de `GPIO.setup`.

❺ On arrive ici à l'équivalent de la fonction `loop` en Arduino. Les instructions sont contenues dans une construction `try/finally`. Ainsi, en cas d'erreur dans l'exécution du programme ou d'interruption de son exécution par la saisie du raccourci Ctrl-C dans la fenêtre de terminal utilisée pour l'exécution du programme, le code de nettoyage contenu dans le bloc `finally` sera exécuté.

Vous pourriez omettre ce code et vous contenter de la seule boucle `while`, mais le code `cleanup` redéfinit automatiquement toutes les broches GPIO comme entrées, ce qui limite les risques de court-circuit accidentel ou d'erreur de câblage de la breadboard qui endommagerait le Raspberry Pi.

❻ La boucle `while` a la condition `True`. Cela peut vous paraître étrange, mais c'est ainsi que l'on définit l'exécution d'un code à l'infini en Python. Le programme exécute en boucle les commandes définies à l'intérieur de la boucle `while` jusqu'à ce que vous l'interrompiez en saisissant Ctrl-C ou en débranchant le Raspberry Pi.

❼ À l'intérieur de la boucle, le code ressemble beaucoup à l'équivalent en Arduino. La broche GPIO est définie comme `True` (niveau haut), puis un délai de 5 secondes est appliqué avant que la broche GPIO ne soit définie sur `False` (niveau bas). Un autre délai de 2 secondes se produit alors avant que le cycle ne recommence.

Expérimentation Raspberry Pi

Pour accéder aux broches GPIO, il faut bénéficier des privilèges de superutilisateur Linux. Ouvrez le dossier contenant le fichier `on_off_control.py`, puis démarrez l'exécution du programme à l'aide de la commande suivante :

```
$ sudo python on_off_control.py
```

Lorsque vous saisissez `sudo` au début de la commande, vous lancez l'exécution en tant que superutilisateur. Pour interrompre le clignotement de la diode, utilisez le raccourci Ctrl-C pour quitter le programme.

Comparaison du code

La structure du code Arduino C et Python est globalement la même, mais le style du code lui-même est différent. De plus, pour nommer des variables ou des fonctions, C utilise la convention typographique CamelCase (c'est-à-dire que tous les mots qui se trouvent après le premier sont écrits avec une majuscule initiale), alors que Python applique la convention snake_case (les mots sont séparés par des tirets bas ou underscores).

Le tableau 4-1 compare les principales différences entre les deux programmes.

Tableau 4-1. *Arduino C et Python*

Commande	Code Arduino C	Python code
Définit une constante pour une broche	`const int controlPin = 9;`	`control_pin = 18`
Définit une broche comme sortie	`pinMode(controlPin, OUTPUT)`	`GPIO.setup(control_pin, GPIO.OUT)`
Définit une sortie en niveau haut	`digitalWrite(controlPin, HIGH);`	`GPIO.output(control_pin, True)`
Définit une sortie en niveau bas	`digitalWrite(controlPin, LOW);`	`GPIO.output(control_pin, False)`
Pause de 1 seconde	`delay(1000);`	`time.sleep(1);`

Expérience : pilotage d'un moteur

Vous savez maintenant vous servir du Raspberry Pi et d'Arduino pour allumer et éteindre une diode. Vous allez mettre vos connaissances en pratique pour allumer et éteindre un moteur. Le programme est identique à celui de l'expérience « Pilotage d'une LED », réalisée précédemment, à la différence que vous utiliserez un transistor pour allumer un moteur à courant continu (CC).

Vous apprendrez davantage sur les moteurs à courant continu au chapitre 7. Ces petits moteurs qui équipent généralement les ventilateurs de poche ou les voitures téléguidées sont très simples d'emploi – une tension appliquée à leurs deux bornes fait tourner l'arbre.

Comme la plupart des moteurs nécessitent trop de courant pour être alimentés directement depuis la sortie numérique du Raspberry Pi ou d'Arduino, on utilise un transistor de façon à ce que le courant faible du Raspberry Pi ou d'Arduino parvienne à réguler un courant bien plus fort vers le moteur.

Les mêmes composants électroniques sont nécessaires pour le Raspberry Pi et l'Arduino et ils sont montés sur la même platine d'essais sans soudure.

Comme il s'agit ici d'un chapitre d'introduction, certains aspects de l'expérience ne seront expliqués que dans les chapitres suivants. Le montage nécessite plus de composants que l'expérience précédente. Vérifiez que les conducteurs sont connectés aux emplacements indiqués et que les composants sont dans le bon sens.

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou un Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser l'expérience.

Nom	Source
Transistor Darlington MPSA14	Mouser : 833-MPSA14-AP
Diode 1N4001	Adafruit : 755 Sparkfun : COM-08589 Mouser : 512-1N4001
Résistance 470 Ω ¼ W	Mouser : 291-470-RC
Petit moteur c.c. 6V	Adafruit : 711
Support pour piles (4 × AA) 6V	Adafruit : 830
Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
Câbles flexibles mâles/mâles	Adafruit : 758
Câbles flexibles femelles/mâles (Raspberry Pi uniquement)	Adafruit : 826

Si vous comptez tenter cette expérience avec un Raspberry Pi, vous aurez besoin de câbles flexibles femelles/mâles pour connecter les broches GPIO du Raspberry Pi à la plaque d'essai.

Réalisation du circuit

Le circuit du projet est illustré à la figure 4-10.

Quand vous placez les composants sur la plaque d'essai, vérifiez que le transistor est positionné correctement – le côté plat correspond au côté droit. Vérifiez aussi que la diode est enfichée dans le bon sens – l'extrémité repérée par un anneau doit être orientée vers le haut de la carte.

Comme il est toujours plus agréable de bidouiller avec des composants que d'en comprendre tous les principes de fonctionnement, nous n'étudierons cette expérience en détail qu'au chapitre 5.

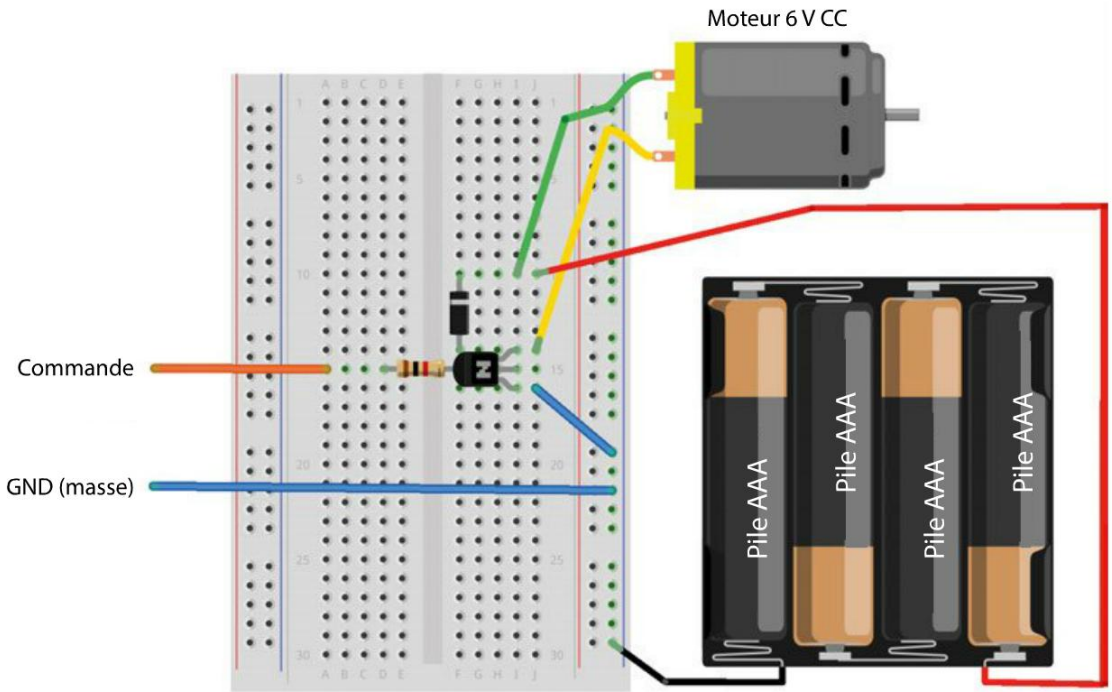


Figure 4-10. *Réalisation du circuit de commande d'un moteur*

Expérimentation sans Arduino ni Raspberry Pi

Avant de commencer à connecter la plaque d'essai à un Arduino ou un Raspberry Pi, vous pouvez réaliser quelques petites expériences pour tester le montage.

Le transistor fonctionne comme un commutateur (chapitre 5). Deux fils seront connectés à l'Arduino ou au Raspberry Pi : GND et Commande. La liaison GND (masse) correspond à zéro volt pour le circuit de la plaque d'essai comme pour l'Arduino et le Raspberry Pi. La liaison de commande allume le moteur si elle est connectée à une tension supérieure à 2 V environ et le moteur sera éteint si la tension est inférieure à ce seuil.

Vous pouvez le vérifier à l'aide d'un câble flexible mâle/mâle avant même de connecter la plaque à un Arduino ou un Raspberry Pi. Branchez l'une des extrémités du câble sur la même rangée que le conducteur gauche de la résistance et, avec l'autre extrémité, touchez le conducteur du haut de la diode qui est aussi connecté au + des piles (figure 4-11). Le moteur démarre. Lorsque vous éloignez le conducteur de la diode, le moteur s'arrête.

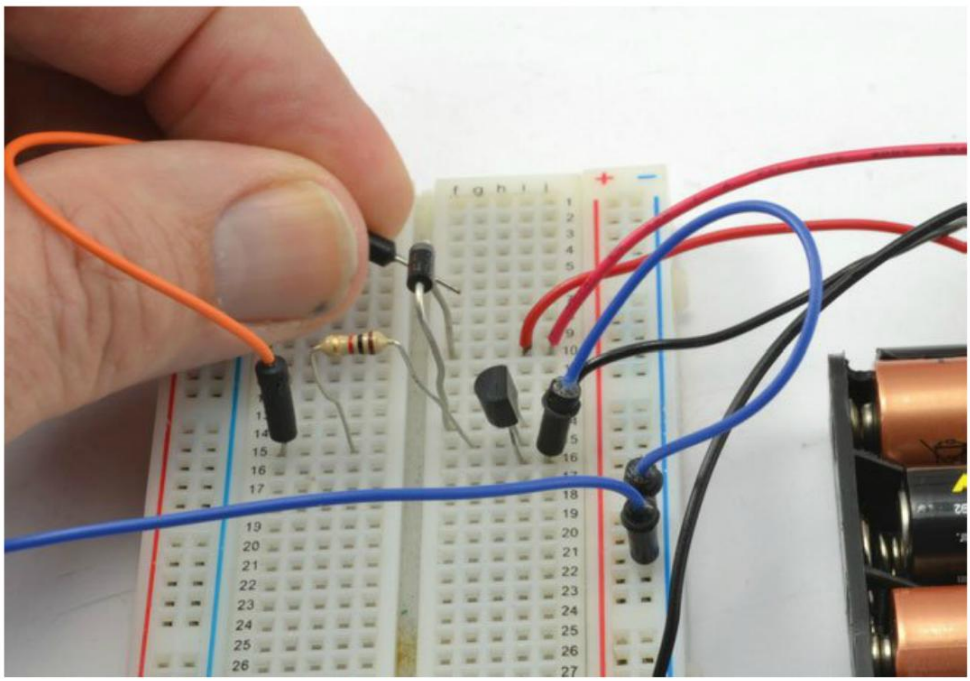


Figure 4-11. *Test du circuit avant le raccordement à un Arduino ou un Raspberry Pi*

Montage Arduino

Après avoir vérifié que le fil de commande de la breadboard allume et éteint le moteur, vous pouvez le connecter à l'une des broches GPIO de la carte Arduino à l'aide d'un câble flexible mâle/mâle. Utilisez la broche 9 de la carte Arduino, comme illustré à la figure 4-12. Notez qu'il s'agit de la même broche de commande que celle utilisée pour la diode dans l'expérience « Pilotage d'une LED », réalisée précédemment.

Vous devrez aussi connecter l'autre fil GND à la broche GND de l'Arduino, comme illustré à la figure 4-12.

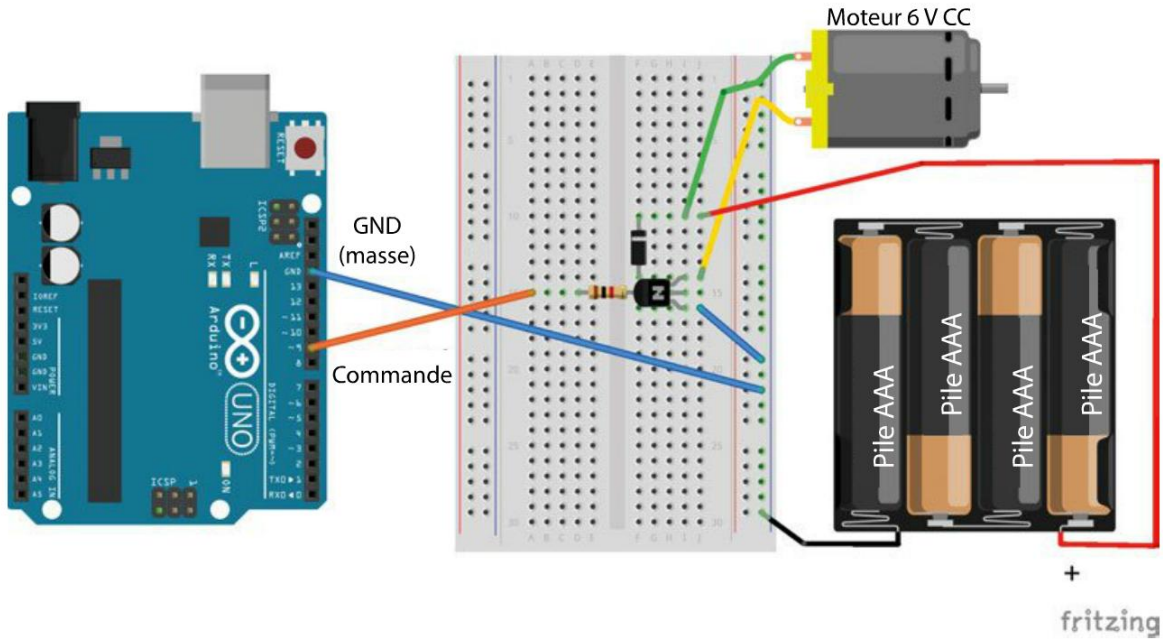


Figure 4-12. *Réalisation du circuit de commande d'un moteur avec un Arduino*

Expérimentation Arduino

Si le programme de l'expérience « Pilotage d'une LED », réalisée précédemment, est toujours chargé sur votre carte Arduino, vous n'avez rien de plus à téléverser. S'il ne s'y trouve plus, reportez-vous aux explications figurant dans cette section.

Comme vous l'avez fait pour la diode, essayez de modifier les nombres définis dans les fonctions de délais pour changer la durée d'allumage du moteur dans chaque cycle.

Montage Raspberry Pi

Vous pouvez maintenant déconnecter la plaque d'essai de l'Arduino et la raccorder au Raspberry Pi. Connectez la broche GPIO 18 (la broche de commande) et GND (la masse), comme illustré à la figure 4-13.

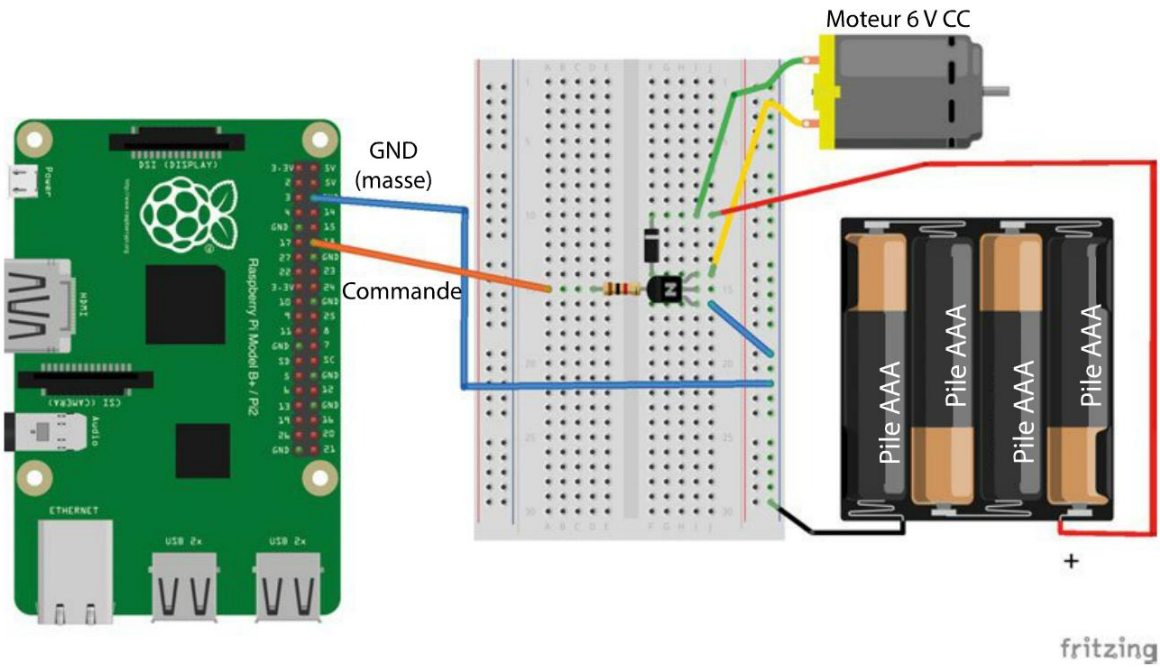


Figure 4-13. *Réalisation du circuit de commande d'une LED avec un Raspberry Pi*

Expérimentation Raspberry Pi

Le programme Raspberry Pi de cette expérience est identique à celui de pilotage de la diode.

Ouvrez le dossier contenant le fichier `on_off_control.py`, puis exécutez le programme à l'aide de la commande suivante :

```
$ sudo python on_off_control.py
```

Quand le moteur a suffisamment fonctionné, appuyez sur Ctrl-C pour quitter le programme.

Résumé

L'objectif de ce chapitre était de vous aider à réaliser votre premier montage. Dans le chapitre suivant, nous nous intéresserons davantage aux connaissances théoriques. Nous verrons aussi comment choisir un transistor et identifier les composants.

Notions d'électronique

5

Dans l'expérience « Pilotage d'un moteur », réalisée au chapitre 4, vous avez vu comment commander un moteur à l'aide d'un transistor sans en comprendre les principes de base. Si vous disposez déjà de notions de base en électronique, vous savez probablement ce qu'est un transistor et vous pourrez survoler une grande partie de ce chapitre. En revanche, si vous débutez en électronique, ces explications vous seront utiles.

Courant, tension et résistance

Dans cette partie, vous devrez vous référer au schéma du circuit de l'expérience « Pilotage d'un moteur », réalisée au chapitre 4, que vous avez construit sur une plaque d'essai, comme illustré à la figure 5-1.

Le schéma est simplement une représentation plus abstraite du montage réalisé sur la plaque d'essai. Au lieu d'en restituer une représentation fidèle à la réalité, le schéma représente le rôle des composants dans le circuit.

Courant

La ligne en zigzag de la résistance R1 indique que ce composant limite le flux du courant. En électronique, le mot « courant » désigne un flux d'électrons qui traverse des fils ou des composants. Imaginez que les électrons circulent d'un emplacement du circuit à un autre. Par exemple, le courant est transmis par la sortie GPIO d'un Arduino ou d'un Raspberry Pi à une résistance R1. Le courant qui sort de R1 passe par la connexion centrale (la base) du transistor Q1.

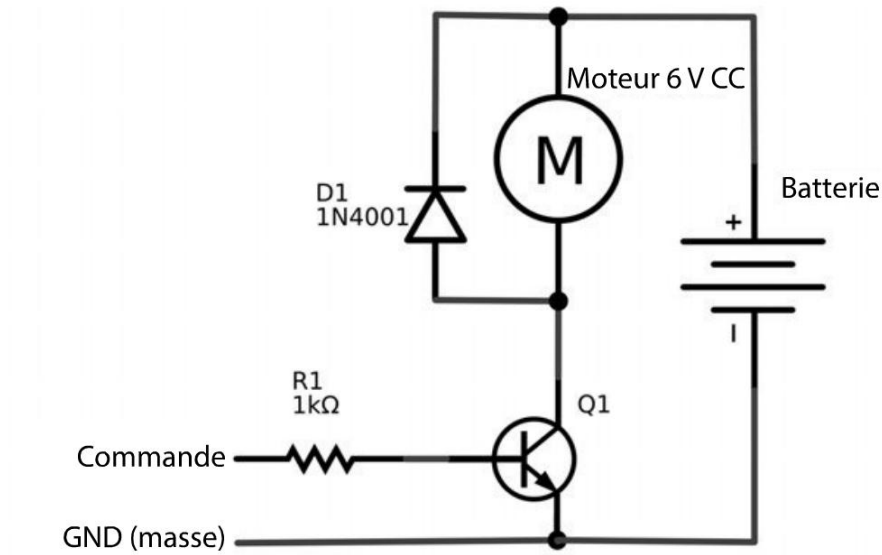


Figure 5-1. *Schéma du circuit de commande d'un moteur*

Un courant de faible intensité transmis à la base d'un transistor permet à un courant de beaucoup plus forte intensité de traverser les deux connecteurs de droite du transistor, le collecteur (en haut) et l'émetteur (en bas). C'est ce qui permet au courant de faible intensité transmis par la broche GPIO de commander le courant de beaucoup plus forte intensité nécessaire au fonctionnement d'un moteur, par exemple. On pourrait le comparer à un interrupteur numérique qui pourrait être allumé ou éteint grâce à un courant de faible intensité.

L'unité de mesure de l'intensité du courant est l'ampère, ou A, dans sa forme abrégée. Un courant de 1 A a une intensité relativement forte pour un Arduino ou un Raspberry Pi, donc on utilise généralement le milliampère (mA) comme unité de mesure. 1 mA équivaut à 1/1 000e d'ampère.

Nous utilisons la résistance R1 pour limiter le flux de courant, car les broches GPIO d'un Raspberry Pi ou d'un Arduino ne peuvent pas fournir assez de courant pour alimenter directement un moteur. D'ailleurs, si vous essayez de le faire, vous risquez d'endommager voire de détruire votre Arduino ou votre Raspberry Pi. Un Raspberry Pi peut fournir un courant d'environ 16 mA, tandis qu'un Arduino peut transmettre un courant d'environ 40 mA en toute sécurité.

Tension

De la même façon que l'eau circule d'un point haut vers un point bas, le courant électrique circule toujours à partir d'emplacements du circuit ayant une tension plus élevée vers des emplacements ayant une tension plus basse. Donc, dans le cas de la broche de commande GPIO illustrée à la figure 5-1, quand la broche est au niveau bas (0V), aucun courant n'est transmis par la broche à la résistance, puis au transistor et enfin, à la masse, car la broche GPIO et la masse ont le même niveau de tension (0 V). Cependant, quand la broche est au niveau haut (3,3 V sur un Raspberry Pi ou 5 V sur un Arduino), le courant circule à travers la résistance et le transistor jusqu'à la masse.

Remarque importante à propos des schémas et de la tension : tous les points reliés par une ligne ont une tension identique.

L'unité de mesure de la tension est le volt, abrégé en V. Les ports GPIO d'un Raspberry utilisés comme sorties ont une tension de 3,3 V (niveau haut) ou 0 V (niveau bas) et les broches d'un Arduino ont une tension de 5 V ou 0 V.

Masse

La ligne qui se trouve en bas de la figure 5-1 est désignée comme étant la masse (GND). Elle correspond à une tension de zéro volt dans un circuit ; c'est la tension de référence utilisée pour la mesure des autres tensions du circuit. Par exemple, on dira que la borne positive de la pile à la figure 5-1 a une tension de 6 V parce que sa tension est supérieure de 6 V par rapport à la masse.

Lorsque vous devez assembler différentes parties d'un projet, les masses de chaque partie sont toutes reliées ensemble. Supposons que vous vouliez relier ce module de commande d'un moteur à un Arduino ou un Raspberry Pi. Dans ce cas, la masse du module serait reliée à l'une des broches GND de l'Arduino ou du Raspberry Pi.

Résistance

La valeur de résistance des résistances est mesurée en ohms, abrégés par la lettre grecque oméga (Ω). Les résistances couvrent une vaste plage de valeurs pouvant même atteindre plusieurs milliers (kΩ) ou millions d'ohms (MΩ).

La résistance utilisée dans l'expérience « Pilotage d'un moteur », réalisée au chapitre précédent, est une résistance de 1 kΩ. Pour calculer la limitation du courant appliquée par cette résistance de 1 kΩ, il faut faire appel à la loi d'Ohm : le courant traversant une résistance est égal à la différence de tension entre les bornes de la résistance (en volts) divisée par la valeur de la résistance en ohms. Dans le cas d'un Raspberry Pi, la limitation de tension la plus élevée entre la broche GPIO et la masse GND est constatée lorsque la broche est au niveau haut (3,3 V). Donc le courant maximum pouvant circuler est de 3,3 V/1 000 Ω = 3,3 mA.

Courant aux broches GPIO du Raspberry Pi

Le courant maximum admis par une broche GPIO d'un Raspberry Pi est un sujet très controversé. Je m'en tiens généralement aux données communiquées par les constructeurs du Raspberry Pi (c'est-à-dire 3 mA par broche GPIO). Ce chiffre de 3 mA s'explique par le fait que le Raspberry Pi original n'était doté que de 14 broches GPIO et son régulateur de tension 3 V ne pouvait fournir que 50 mA aux broches GPIO, soit 3 mA par broche GPIO, si toutes les broches GPIO étaient utilisées.

Cela ne s'applique plus aux modèles récents (A+, B+ et Pi 2) qui ont 24 broches pouvant être utilisées comme broches GPIO et un régulateur 3 V qui, en théorie, peut fournir une tension allant jusqu'à 1 A. Si vous utilisez un modèle récent, n'essayez pas de tirer 1 A/24 = 41 mA par broche GPIO, car la puce du système Broadcom (SoC) est aussi limitée à 16 mA par broche GPIO.

Sur un Raspberry Pi A+, B+ ou Pi 2, vous pouvez en théorie utiliser jusqu'à 16 mA sur autant de broches GPIO que nécessaire. Néanmoins, d'autres facteurs entrent en ligne de compte sur la quantité de courant pouvant être utilisée sans endommager la carte. Il s'agit notamment de la fréquence de commutation et du courant GPIO total admis par la puce Broadcom.

En résumé, tenez-vous-en aux consignes suivantes.

- Sur un Raspberry Pi 1 original, vous pouvez utiliser 16 mA par broche dans la limite de 40 mA.
- Sur un Raspberry Pi A+, B+ ou Pi 2, par prudence, ne dépassez pas 16 mA par broche jusqu'à un maximum de 100 mA au total pour toutes les broches utilisées.

La résistance protège la broche GPIO pour limiter le courant qu'elle peut fournir. Si vous munissez toutes les sorties numériques du Raspberry Pi d'une résistance de 1kΩ, vous ne risquez pas d'endommager votre nano-ordinateur. Vous pourrez souvent utiliser une résistance plus faible, surtout pour des diodes, car ces dernières absorberont une partie de la chute de tension et réduiront le courant.

Puissance

Quand un courant passe à travers une résistance, cela crée de la chaleur. L'énergie électrique est convertie en énergie thermique. La puissance mesure la quantité de cette énergie qui est convertie par seconde. Son unité est le watt (W). La chaleur générée par un composant est calculée en multipliant la tension (en volts) aux bornes du composant par l'intensité du courant qui le traverse (en ampères).

Reprenons la résistance de 1 kΩ de l'expérience précédente. Supposons qu'elle reçoive une tension de 2,2 V et qu'elle soit traversée par un courant d'une intensité de 2,2 mA. Elle produira alors une puissance de 4,8 mW, ce qui est très faible. Toutefois, les transistors génèrent aussi de l'énergie thermique qui est calculée en multipliant la tension à leurs bornes par l'intensité du courant qui les traverse. Si le moteur utilisé pour l'expérience « Pilotage d'un moteur », décrite au chapitre précédent, est relativement puissant (800 mA, par exemple), la perte de tension entre le

collecteur et l'émetteur du transistor sera d'environ 1,2 V. Donc la puissance convertie en énergie thermique sera de 800 mA × 1,2 V = 960 mW. Le transistor chauffera. S'il chauffe trop, il finira par fondre et ne fonctionnera plus. Pour l'éviter, les transistors ont une intensité maximale de courant, comme les broches de sortie du Raspberry Pi et de l'Arduino. Les transistors physiquement plus gros accepteront des intensités de courant supérieures. C'est l'un des facteurs devant être pris en compte au moment de choisir un transistor pour commander un servomoteur.

Le transistor Darlington MPSA14, employé dans l'expérience « Pilotage d'un moteur » au chapitre précédant, supporte une intensité maximale du courant de 1 A.

Principaux composants

Nous allons maintenant passer en revue les principaux composants utilisés dans ce livre. Nous verrons comment les utiliser et les choisir.

Résistances

Les résistances sont de petits composants très colorés. Pour en connaître la valeur, vous pouvez mesurer leur résistance à l'aide d'un multimètre ou vous référer à leurs anneaux colorés.

Chaque couleur est associée à un nombre, comme le montre le tableau 5-1.

Tableau 5-1. *Codes des couleurs des résistances*

Noir	0
Marron	1
Rouge	2
Orange	3
Jaune	4
Vert	5
Bleu	6
Violet	7
Gris	8
Blanc	9
Or	1/10
Argent	1/100

En plus de représenter les fractions 1/10 et 1/100, les marquages or et agent sont aussi utilisés pour indiquer le niveau de précision (tolérance) de la résistance. L'or correspond à ±5 % et l'argent à ±10 %.

Trois anneaux sont généralement réunis à l'une des extrémités de la résistance. Ils sont suivis d'un espace. Un anneau isolé se trouve à l'extrémité opposée. Il indique la tolérance de la valeur de la résistance.

La figure 5-2 illustre la disposition des anneaux colorés. La valeur de résistance est indiquée par les trois anneaux de gauche. Le premier anneau correspond au premier chiffre, le deuxième au second et le troisième, le multiplicateur, indique le nombre de zéros devant être ajoutés après les deux premiers chiffres.

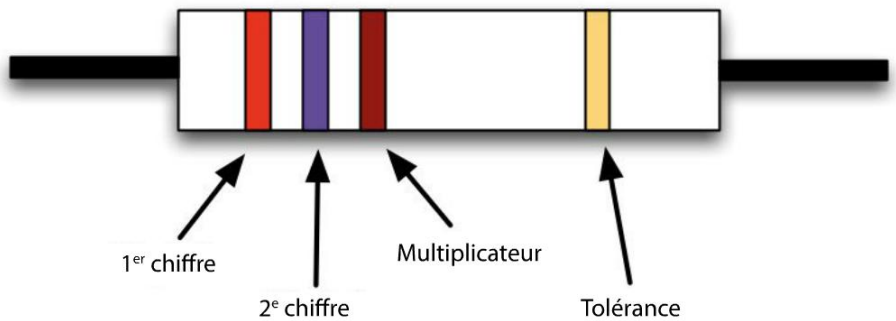


Figure 5-2. *Lecture du code couleur des résistances*

Par exemple, le premier chiffre d'une résistance de 270 Ω est 2 (rouge), le deuxième est 7 (violet) et il y a un seul zéro, donc le multiplicateur est 1 (marron). De même, une résistance de 1 kΩ aura des anneaux marron, noir et rouge (1, 0, 00).

Les résistances ont aussi une puissance nominale. Les résistances à couche métallique du type de celles utilisées dans ce livre ont presque toutes une puissance de 1/4 W. On trouve aussi des modèles ayant une puissance nominale de 1/2 W, 1 W et 2 W. La taille de la résistance s'accroît avec sa puissance.

Transistors

Les fournisseurs de composants électroniques proposent généralement une vaste gamme de transistors. Dans ce livre, j'ai limité le choix à quatre modèles qui répondent aux principaux usages.

Le transistor de l'expérience « Pilotage d'un moteur », réalisée au chapitre précédent, permet à un courant de très faible intensité (quelques milliampères) de réguler les centaines de milliampères requis par le moteur. Même si les transistors peuvent remplir d'autres rôles, dans ce livre, nous les utiliserons comme commutateurs. Un courant de faible intensité arrivant à la base et allant jusqu'à la masse en passant par l'émetteur du transistor pilotera un courant plus fort circulant du collecteur à l'émetteur.

La figure 5-3 montre différents types de transistors aux capacités de pilotage de puissance variées.

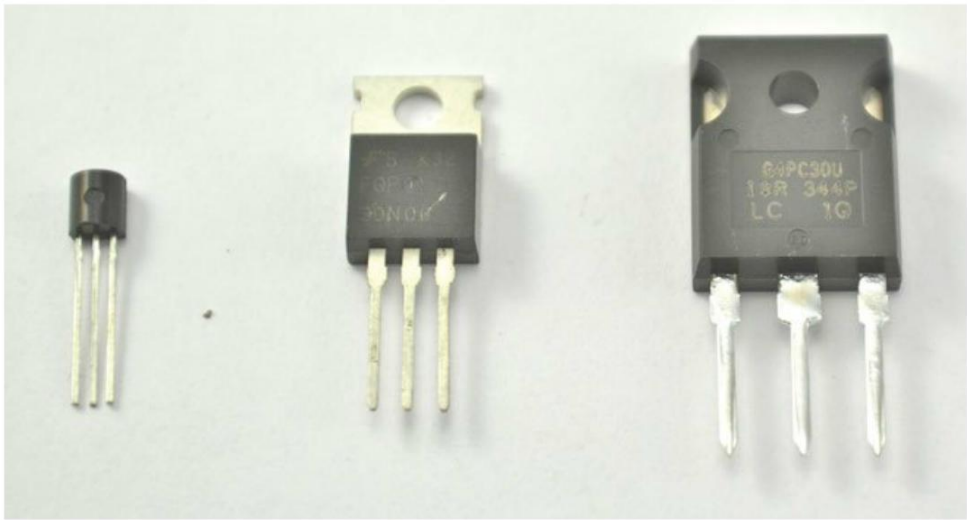


Figure 5-3. *Une sélection de transistors*

Les transistors sont disponibles sous la forme d'un assez petit nombre de boîtiers standards. Pour identifier un transistor, vous ne pouvez donc pas vous fier à son apparence ; vous devez lire la désignation qui y est inscrite.

Les boîtiers les plus courants sont le TO-92, à gauche, et le TO-220, au centre. Parfois, pour des applications à très forte puissance, on peut utiliser un transistor ayant un boîtier plus grand, comme le modèle TO-247, à droite.

Les boîtiers TO-220 et TO-247 ont été conçus pour être fixés sur des dissipateurs thermiques. Il n'est pas nécessaire de le faire si vous les utilisez à des courants inférieurs à leur intensité nominale maximale.

Transistors bipolaires

Les transistors utilisent différentes technologies qui ont chacune des avantages et des inconvénients qui font qu'ils sont adaptés à certaines situations, mais pas à d'autres.

Le transistor bipolaire est le plus utilisé par les novices. Il n'a guère évolué depuis ses débuts. Il présente l'avantage d'être bon marché et simple d'emploi pour les courants assez faibles. Toutefois, bien qu'un courant faible circulant de la base à l'émetteur du transistor aboutisse à un courant plus fort passant du collecteur à l'émetteur, le courant du collecteur est limité à un multiple du courant de base et ce multiple (appelé gain ou hFE) est généralement compris entre 50 et 200. Donc, si un Raspberry Pi ne fournit que 2 mA à la base, le courant passant par le collecteur ne peut être que de 100 mA. Peut-être vous attendiez-vous à bien davantage, car le transistor peut avoir une capacité de conduction de courant bien supérieure (telle que 500 mA), mais cette limite ne sera jamais atteinte puisque le courant de base est insuffisant. Ce problème ne se pose généralement pas avec un Arduino, car il peut fournir plus de courant à la base (jusqu'à 40 mA) en employant une résistance de valeur plus basse à la place de la résistance de

1 kΩ illustrée à la figure 5-1. Si vous choisissez une valeur de résistance de 150 Ω, par exemple, le courant de base augmentera à I = V/R = (5 – 0,5)/150 = 30 mA. Même si le transistor n'a qu'un gain de 50, un courant de base de 30 mA résultera tout de même en un courant collecteur de 1,5 A.

Le calcul de la tension dans l'équation précédente est (5 – 0,5), car la tension entre la base et l'émetteur d'un transistor bipolaire sera d'environ 0,5 V quand le transistor est conducteur.

Dans ce livre, nous n'utilisons qu'un seul modèle de transistor bipolaire, le 2N3904 qui est très répandu. Même s'il existe des transistors bipolaires à courant plus fort, il est préférable d'avoir recours à des technologies plus évoluées lorsque le courant est plus élevé.

Transistors Darlington

Si le gain doit être supérieur, parce que le Raspberry Pi doit piloter un petit moteur alors que la base ne reçoit que quelques milliampères, un transistor Darlington peut être utilisé à la place d'un transistor bipolaire ordinaire. Son gain est généralement de l'ordre de 10 000.

Un transistor Darlington se compose de deux transistors bipolaires réunis à l'intérieur d'un même boîtier (figure 5-4). Cette combinaison lui confère un gain plus élevé.

Comme il y a maintenant deux jonctions base-émetteur, chacune chute d'au moins 0,5 V quand le transistor est allumé, donc la chute de tension est d'environ 1 V, au lieu des 0,5 V d'un transistor bipolaire ordinaire.

Le transistor utilisé dans l'expérience « Pilotage d'un moteur », décrite au chapitre précédent, (un MPSA14) est de type Darlington. La chute de tension sur R1 avec un Raspberry Pi ne sera pas de 3,3 V, mais de 3,3 V – 1 V, soit 2,2 V. Donc, le courant devant être fourni par le Raspberry Pi est de 2,2 V/1 kΩ = 2,2 mA.

Outre le MPSA14 de faible puissance qui permet de commander des charges maximales de 0,5 A, je recommande aussi le transistor Darlington TIP120 qui est plus puissant et qui peut être utilisé comme transistor standard.

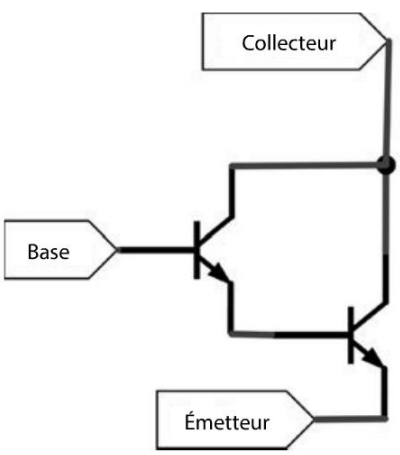


Figure 5-4. *Un transistor Darlington*

MOSFET

Les transistors bipolaires sont des composants électriques contrôlés essentiellement par l'intensité du courant — le courant de base faible est amplifié en un courant collecteur plus élevé. Il existe un autre type de transistor, le transistor à effet de champ à grille isolée (ou MOSFET, *Metaloxide-Semiconductor Field-Effect Transistor*) qui nécessite très peu de courant pour commuter, mais qui conduit tant que la tension à sa grille *(gate)* est supérieure à un certain seuil.

La figure 5-5 présente le symbole électrique du MOSFET. Comme vous pouvez le constater, le symbole implique que la grille n'est pas connectée directement au reste du transistor.

Figure 5-5. *Le symbole d'un MOSFET*

Notez qu'au lieu d'une base, un collecteur et un émetteur, un MOSFET a une grille, un drain et une source. On pourrait s'attendre à ce que la source soit l'équivalent du collecteur, alors qu'en fait, c'est le drain qui correspond au collecteur d'un transistor bipolaire.

L'utilisation d'un MOSFET pour la commutation est un bon choix avec un Arduino ou un Raspberry Pi, car il nécessite très peu de courant. Assurez-vous simplement que la tension à la grille est supérieure à la tension de seuil de la grille du transistor. La tension de seuil de la grille correspond à la tension à partir de laquelle le MOSFET crée un canal de conduction entre le drain et la source. La figure 5-6 montre comment connecter un MOSFET pour commander une charge. Le montage est le même que celui d'un transistor bipolaire.

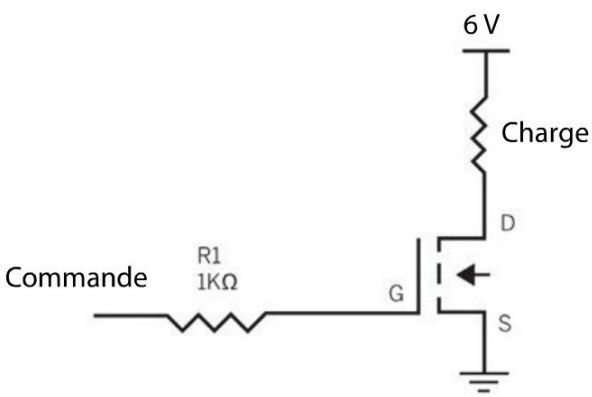


Figure 5-6. *Utilisation d'un MOSFET*

Le schéma illustré à la figure 5-6 utilise deux symboles que nous n'avons pas encore vus. Une série de trois traits parallèles de longueur décroissante est connectée à la source du transistor (S). C'est le symbole de la masse (GND). Son utilisation dans un schéma a l'avantage de réduire le nombre de lignes de liaisons devant être tracées.

L'autre symbole se trouve en haut du schéma : il s'agit simplement d'un trait horizontal portant la mention 6 V. Cela indique qu'il y aura une alimentation de 6 V à ce point du circuit, ce qui évite d'avoir à dessiner des piles.

Les brochages des transistors varient

Même si dans ce livre, nous essayons de nous en tenir aux transistors ayant des brochages compatibles entre eux, ce n'est pas toujours le cas. Tous les transistors présentés dans le même boîtier n'ont pas nécessairement le même brochage ; vérifiez sa fiche technique avant d'utiliser un nouveau transistor. Les brochages de nombreux composants utilisés dans ce livre sont repris à l'annexe A.

Si vous observez la figure 5-6, vous vous demanderez peut-être pourquoi R1 est nécessaire, puisque la grille ne reçoit pas de courant. Il est préférable de prévoir une résistance, car, lorsque la tension de la grille augmente pour la première fois, il y a un très rapide afflux de courant pendant une fraction de seconde. La résistance évite que le courant ne dépasse les capacités de la broche GPIO.

L'inconvénient des MOSFET est que la tension de seuil est parfois trop élevée pour commuter avec les 3,3 V ou 5 V d'un Raspberry Pi ou d'un Arduino. Les MOSFET qui ont un seuil de grille assez bas pour être utilisés directement à partir d'une broche GPIO sont des MOSFET de niveau logique. Dans ce livre, nous utiliserons essentiellement deux MOSFET : le modèle 2N7000 pour les faibles puissances ; et pour les applications nécessitant des puissances supérieures, le FQP30N06L. Ils ont tous les deux une tension de seuil à la grille inférieure à 3 V, ce qui permet de les utiliser avec un Arduino et un Raspberry Pi.

En général, les MOSFET chauffent moins en fonctionnement que les transistors bipolaires. L'une des principales propriétés à prendre en compte au moment de l'achat d'un MOSFET est sa résistance lorsqu'il conduit. Un MOSFET ayant une résistance très basse commutera des courants plus élevés sans chauffer. Comme vous devez vous en douter, plus la résistance d'un MOSFET est basse, plus son prix est élevé.

Dans ce livre, nous utilisons assez souvent des MOSFET. Je m'en tiens généralement au modèle FQP30N06L (jusqu'à 30 A), même si pour ce type de courant, il vous faudra un gros dissipateur thermique.

En fait, les broches de l'émetteur, de la base et du collecteur du Darlington TIP120 et la source, la grille et le drain d'un MOSFET FQP30N06L se trouvent aux mêmes emplacements. Donc vous pouvez vous contenter de retirer un TIP120 d'une plaque d'essais pour le remplacer par un FQP30N06L en conservant la même configuration et le circuit devrait continuer à fonctionner.

Transistors PNP et à canal P

Les transistors que nous venons de décrire sont de deux types. Jusqu'à présent, nous n'en avons vu qu'un seul : les transistors négatif-positif-négatif (NPN) ou à canal N dans le cas des MOSFET. Il s'agit du type le plus répandu et vous vous en contenterez généralement.

L'autre type de transistors regroupe les composants PNP ou à canal P. Quand des composants de type N sont utilisés pour commuter la charge vers la masse, des composants de type P la commutent vers la source positive. Au chapitre 8, nous utiliserons des MOSFET à canal P pour le pilotage de moteurs avec un pont en H.

Modèle de transistor

Il n'est pas toujours facile de choisir le transistor adapté. Le tableau 5-2 compare les caractéristiques de cinq modèles.

Avec un Raspberry Pi, on considère qu'une résistance 1 kΩ se trouve entre la broche GPIO et la base ou la grille du transistor. Pour un Arduino, on considère que cette résistance est de 150 Ω. Les courants indiqués ont été obtenus en testant les composants et les tensions maximales proviennent des caractéristiques techniques des composants.

Tableau 5-2. *Une sélection de transistors utiles*

Transistor	Type	Boîtier	Courant maxi. (Pi 3,3 V)	Courant maxi. (Arduino 5 V)	Tension maxi
2N3904	Bipolaire	TO-92	100 mA	200 mA	40 V
2N7000	MOSFET	TO-92	200 mA	200 mA	60 V
MPSA14	Darlington	TO-92	1 A	1 A	30 V
TIP120	Darlington	TO-220	5 A	5 A	60 V
FQP30N06L	MOSFET	TO-220	30 A	30 A	60 V

Au moment d'acheter un MOSFET FQP30N06L, vérifiez qu'il est de type L (logique) : la lettre L figure à l'extrémité de sa dénomination ; sinon, la tension de seuil à la grille risque d'être trop élevée.

Le MPSA14 est un composant relativement universel pour les courants inférieurs à 1 A. Toutefois, à ce niveau, la chute de tension est d'environ 3 V et la température du composant peut atteindre 120 °C ! À 500 mA, la chute de tension est plus raisonnable (de l'ordre de 1,8 V) et la température est de 60 °C.

Pour résumer, si vous devez seulement commuter 100 mA, un 2N3904 devrait vous suffire. Si vous devez aller jusqu'à 1 A, utilisez un MPSA14. Au-delà, le FQP30N06L est probablement le meilleur choix, à moins que le prix n'entre aussi en considération, car le TIP120 est moins cher.

Diodes

La diode de la première expérience (voir figure 5-1) sert à protéger le Raspberry Pi ou le Arduino et le transistor.

Les moteurs créent des pics de tension et toutes sortes de bruits électriques qui peuvent nuire au bon fonctionnement des circuits délicats comme le Raspberry Pi ou l'Arduino. Une diode permet d'éviter que ces pics électriques causés par le moteur n'inversent momentanément le flux du courant, ce qui risquerait de griller le transistor. Une diode ne laisse passer le courant que dans une direction, à la façon d'un clapet antiretour. Le courant ne peut circuler que dans le sens indiqué par sa forme ressemblant à une flèche.

C'est pourquoi on place fréquemment une diode entre les bornes d'un moteur. Le courant circule généralement dans le moteur dans le sens inverse à la direction autorisée par la diode. En cas de pic de tension négatif, la diode jouera son rôle en conduisant et court-circuitant le bref flux de courant, ce qui aura pour effet de l'annuler.

LED

Vous en apprendrez davantage sur les LED au chapitre 6. Le terme LED signifie *Light-Emitting Diode* (diode électroluminescente ou DEL).

Comme vous pouvez vous en douter, une LED fonctionne comme une diode ordinaire, à la différence que lorsque le courant la traverse, elle émet de la lumière. Son symbole est identique à celui de la diode ordinaire, mais il est doté de flèches signalant l'émission de lumière (figure 5-7).

Figure 5-7. *Le symbole d'une LED*

Il existe différentes couleurs et tailles de LED. Elles peuvent être commandées directement depuis une broche GPIO de l'Arduino ou du Raspberry Pi. Mais il faut prévoir une résistance pour limiter le courant, comme pour commander un transistor. Vous en apprendrez davantage au chapitre 6.

Condensateurs

Les condensateurs permettent de stocker temporairement de l'électricité – comme des batteries de très faible capacité pouvant conserver une petite réserve de charge. Nous nous en servirons dans ce livre en leur faisant remplir divers rôles allant d'aider à supprimer les interférences électriques à conserver une petite réserve d'énergie électrique de façon à pouvoir répondre aux hausses soudaines dans la demande en électricité.

La figure 5-8 présente les symboles d'un condensateur qui est généralement polarisé lorsqu'il a une capacité élevée. Les condensateurs ayant une faible capacité n'ont pas de côtés positif et négatif.

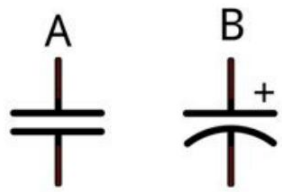


Figure 5-8. *Les symboles de condensateurs : non polarisé (A) et polarisé (B)*

Vous rencontrerez aussi des symboles légèrement différents qui comportent un rectangle vide pour le côté positif d'un condensateur polarisé et un rectangle plein pour le côté négatif. Dans ce livre, nous appliquons la convention américaine des symboles de condensateurs tels qu'illustrés à la figure 5-8.

Circuits intégrés

Les circuits intégrés, aussi appelés puces électroniques, sont constitués de nombreux transistors disposés sur une couche de silicium et sont contenus dans un même bloc. Les cartes Raspberry Pi et l'Arduino sont composées de nombreux circuits intégrés et d'autres composants montés sur un circuit imprimé (ou PCB, *Printed Circuit Board*).

Les circuits intégrés sont spécialement conçus pour répondre à des besoins électroniques particuliers. Dans ce livre, nous les utiliserons surtout pour commander des composants.

Ce type de circuits intégrés associe souvent des transistors pour des courants élevés et de la logique de commande, sur un même support.

Les tenants et les aboutissants des connexions

Après avoir présenté les bases de l'électronique, nous allons voir comment ces composants sont interfacés au Raspberry Pi ou à l'Arduino. Vous trouverez des explications plus détaillées, surtout en ce qui concerne la programmation, aux chapitres 2 et 3.

Sorties numériques

Comme nous l'avons vu au chapitre 4, les sorties numériques servent à piloter (allumer et à éteindre) des composants. Si vous utilisez un Raspberry Pi, une sortie numérique pourra avoir un niveau bas de 0 V ou un niveau haut de 3,3 V. Sur une carte Arduino, le niveau haut est de 5 V au lieu de 3,3 V, mais le principe est le même. La tension ne peut pas se situer entre le niveau haut et le niveau bas.

Une autre différence entre l'Arduino et le Raspberry Pi est que l'Arduino peut fournir plus de courant (40 mA au lieu de 16 mA).

Ce livre contient de nombreux exemples d'utilisation de sorties numériques puisque c'est le principal mécanisme utilisé pour piloter des composants.

Entrées numériques

Les entrées numériques sont souvent connectées à des commutateurs ou aux sorties numériques d'autres composants. Une entrée numérique a une tension de seuil généralement située au milieu de la plage de tensions haute et basse. Pour une carte Arduino, le seuil se situera donc aux alentours de 2,5 V et pour un Raspberry Pi, il sera d'environ 1,65 V. Lorsque le programme exécuté sur la carte Arduino ou le Raspberry Pi lit une entrée numérique, si elle est au-dessus du seuil, elle sera considérée comme étant au niveau haut ; dans le cas contraire, l'entrée sera au niveau bas.

Comme pour les sorties numériques, il n'y a pas de demi-mesures avec les entrées numériques – elles sont soit hautes, soit basses.

Vous trouverez des informations sur l'utilisation des entrées numériques de la carte Arduino au chapitre 2 et sur celles du Raspberry Pi au chapitre 3.

Entrées analogiques

Les entrées analogiques vous permettent de mesurer une tension située entre le niveau haut et le niveau bas sur une broche analogique. Le Raspberry Pi n'a pas d'entrées analogiques, mais l'Arduino en a six, numérotées de A0 à A5.

Sur un Arduino, la tension comprise entre 0 et 5 V est traduite en un chiffre compris entre 0 et 1 023. 0 V correspond à 0 et 5 V à 1 023. Ainsi, 2,5 V correspond à 511.

Reportez-vous au chapitre 2 pour plus d'informations sur les entrées analogiques de l'Arduino.

Sorties analogiques

Même si vous supposez que les sorties analogiques devraient vous permettre de définir une broche de sortie à une tension quelconque située entre le niveau bas et le niveau haut, ce n'est pas aussi simple. Ces sorties utilisent la technique de la modulation de largeur d'impulsion MLI (ou PWM, *Pulse-Width Modulation*) pour réguler la puissance moyenne arrivant à une sortie numérique ordinaire.

La modulation de largeur d'impulsion permet de contrôler la vitesse d'un moteur et la luminosité d'une LED. Vous en apprendrez plus sur la modulation de largeur d'impulsion à la section « Régulation de la vitesse par MLI » du chapitre 7 page 96.

Communication série

Les techniques d'interface que nous venons de décrire sont assez basiques. Certains composants que vous voudrez contrôler avec une carte Arduino ou un Raspberry Pi utilisent des interfaces série qui transmettent des données binaires bit par bit depuis la sortie numérique d'un élément à l'entrée numérique d'un autre élément.

Par exemple, au chapitre 14, nous utiliserons des afficheurs qui exigent des données au format série.

Il existe plusieurs protocoles d'interface série qui jouent tous le même rôle, mais de façons légèrement différentes. Il s'agit notamment du protocole série ou série TTL, du protocole I2C et de la liaison SPI *(Serial Peripheral Interface)*.

Résumé

Ce chapitre vous a permis de vous familiariser avec les concepts électroniques fondamentaux de la tension et de la résistance du courant. Nous avons également passé en revue quelques composants électroniques que nous utiliserons dans ce livre. Nous avons vu assez de théorie pour l'instant. Passons à la pratique ! Dans le prochain chapitre, vous apprendrez à utiliser différents types de LED avec votre carte Arduino et le Raspberry Pi.

LED

Le pilotage de diodes électroluminescentes (LED, ou *Light-Emitting Diode*) figure en tête de liste des tâches que souhaitent accomplir la majorité des *makers*. Il existe un très vaste choix de LED, allant des plus simples aux plus compliquées, en passant par des modèles à haute puissance, infrarouges ou ultraviolets.

Les caractéristiques des LED (figure 6-1) sont extrêmement variables : certaines fonctionnent parfaitement depuis une sortie numérique, tandis que d'autres nécessitent un transistor ou sont commandées par d'autres composants.

Figure 6-1. *Une sélection de LED*

LED ordinaires

Par LED ordinaires, j'entends les petits composants colorés qui mesurent généralement 5 mm de diamètre (parfois 3 mm ou 10 mm) et qui nécessitent un courant modeste pour s'illuminer. Elles peuvent donc être pilotées directement depuis la sortie de l'Arduino ou du Raspberry Pi.

La figure 6-2 montre le raccordement d'une LED à une sortie numérique d'Arduino ou Raspberry Pi.

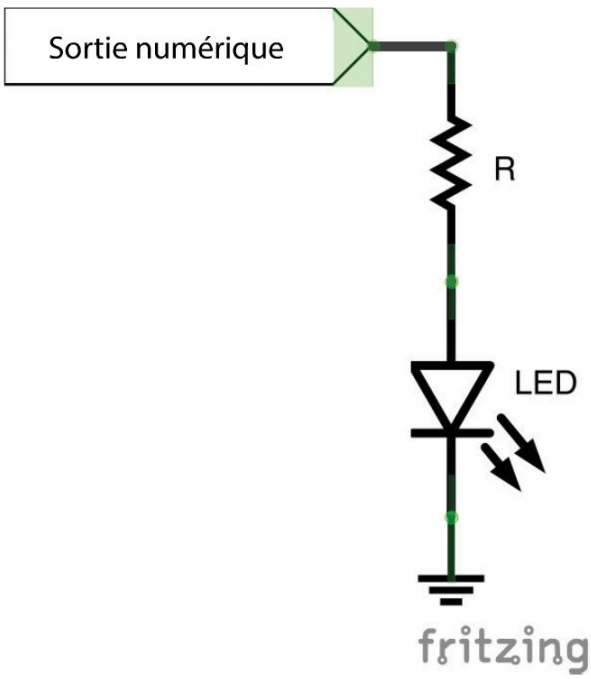


Figure 6-2. *Branchement d'une LED sur une sortie numérique*

La résistance est nécessaire pour limiter le courant traversant la LED pour deux raisons : d'une part, pour éviter de dépasser le courant maximum de la LED (ce qui réduirait sa durée de vie) ; et, d'autre part, pour éviter de dépasser la capacité de courant de la broche de sortie ou le total cumulé de toutes les broches de sortie.

Limitation de courant

Quand une LED est connectée comme illustré à la figure 6-2, la tension sera plus ou moins constante sur la LED. C'est ce que l'on appelle la tension directe de la LED (Vf, ou *forward voltage*). Les différentes couleurs de LED ont des tensions directes différentes. Les LED rouges ont généralement la tension directe la plus faible et les LED bleues et blanches ont la tension directe la plus élevée parmi les LED produisant de la lumière visible (tableau 6-1).

Il existe aussi des LED à infrarouge (IR) qui sont utilisées dans les télécommandes des téléviseurs et des LED à ultraviolet (UV) qui sont souvent utilisées pour faire briller les vêtements blancs lors des soirées ou pour détecter les faux billets de banque.

En plus de sa tension directe, l'autre caractéristique à connaître à propos d'une LED est le courant direct qui doit la traverser. La plupart des LED émettent de la lumière lorsque le courant a une intensité de 1 mA ou moins, mais la plupart atteindront un niveau de luminosité optimal à 20 mA. La plage est vaste. C'est pourquoi, pour plus de sécurité, il vaut mieux utiliser une résistance 470 Ω avec toutes les LED sur un Raspberry Pi ou un Arduino, bien que la LED n'aura pas toujours sa luminosité maximale.

La méthode la plus simple pour calculer la valeur des résistances série est de faire appel à un service web qui fera le calcul à votre place. Un service de ce type (*http://led.linear1.org/1led.wiz*) est illustré à la figure 6-3.

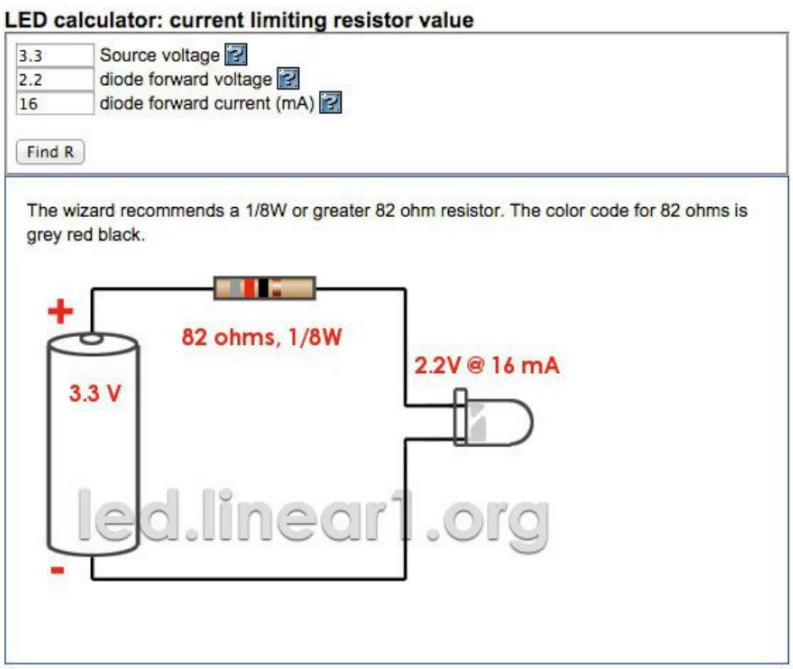


Figure 6-3. *Calculateur en ligne de résistances série*

Ici, la tension source est de 3,3 V parce que l'on utilise un Raspberry Pi et que l'on utilise aussi le courant maximum de 16 mA d'un Raspberry Pi sur une broche. Le calculateur indique qu'il faut utiliser une résistance 82 Ω pour une LED ayant une Vf de 2,2 V.

Si vous voulez faire vous-même le calcul, vous devez d'abord soustraire la valeur Vf (2,2 V) de la LED de la tension de commande (3,3 V). On obtient 1,1 V. Ensuite, utilisez la loi d'Ohm pour calculer la valeur de la résistance R = V/I = 1,1 V/16 mA = 68,75 Ω.

Vous pouvez aussi utiliser le tableau 6-1 pour vous aider à choisir une résistance. Ce tableau indique aussi la plage de Vf approximative des LED de différentes couleurs.

Tableau 6-1. *Résistances de limitation de courant pour les LED*

	IR	Rouge	Orange/ Jaune/Verte	Bleu/ Blanche	Violette	UV
Vf	1,2-1,6 V	1,6-2 V	2-2,2 V	2,5-3,7 V	2,7-4 V	3,1-4,4 V
Pi 3,3 V 3 mA	X	680 Ω	470 Ω	270 Ω	220 Ω	68 Ω
Pi 3,3 V 16 mA	150 Ω	120 Ω	82 Ω	56 Ω	39 Ω	15 Ω
Arduino 5 V 20 mA	220 Ω	180 Ω	150 Ω	150 Ω	120 Ω	100 Ω

Notez que les valeurs de résistance sont arrondies à la valeur standard la plus proche.

Le X dans la colonne IR à 3 mA indique que les LED IR des télécommandes nécessitent généralement au moins 10 mA et la plupart sont conçues pour fonctionner à 100 mA ou plus pour atteindre une portée de signal suffisante.

Projet : feu tricolore

Parmi toutes les couleurs de LED disponibles (figure 6-4), vous en choisirez une rouge, une orange et une verte pour fabriquer un feu tricolore piloté par Arduino ou Raspberry Pi.

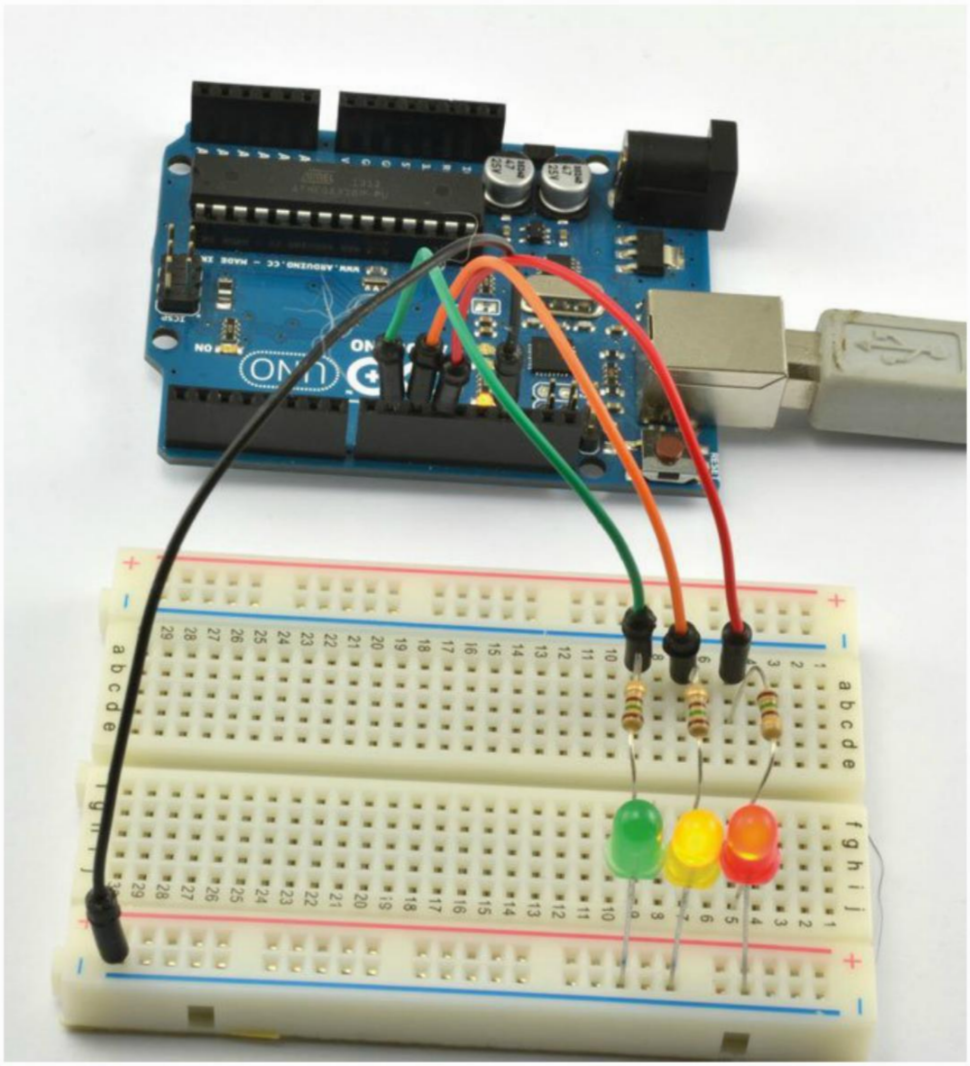


Figure 6-4. *Un feu tricolore Arduino*

La séquence d'éclairage des LED est la suivante :

1. Rouge
2. Rouge et orange ensemble
3. Verte
4. Orange

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou un Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser ce projet.

Nom	Composant	Source
LED1	LED rouge	Adafruit : 297 Sparkfun : COM-09590
LED2	LED orange	Sparkfun : COM-09594
LED3	LED verte	Adafruit : 298 Sparkfun : COM-09650
R1-3	Résistances 150 Ω	Mouser : 291-150-RC
	Plaque de montage rapide à 400 contacts	Adafruit : 64
	Câbles flexibles mâles/mâles	Adafruit : 758
	Câbles flexibles femelles/mâles (Pi uniquement)	Adafruit : 826

Vous vous constituerez peu à peu une collection de résistances de différentes valeurs. Ici, je conseille d'utiliser des résistances d'une valeur de 150 Ω pour les LED, pour l'Arduino et le Raspberry Pi, ainsi que pour les trois couleurs de LED. Si vous voulez ajuster ces valeurs pour profiter d'une luminosité maximale, reportez-vous au tableau 6-1 pour choisir les valeurs de résistances optimales.

Montage

Les trois LED sont chacune connectées à une broche de sortie distincte de l'Arduino ou du Raspberry Pi.

Montage Arduino

La figure 6-5 montre la réalisation du circuit et les connexions entre la plaque d'essai et la carte Arduino.

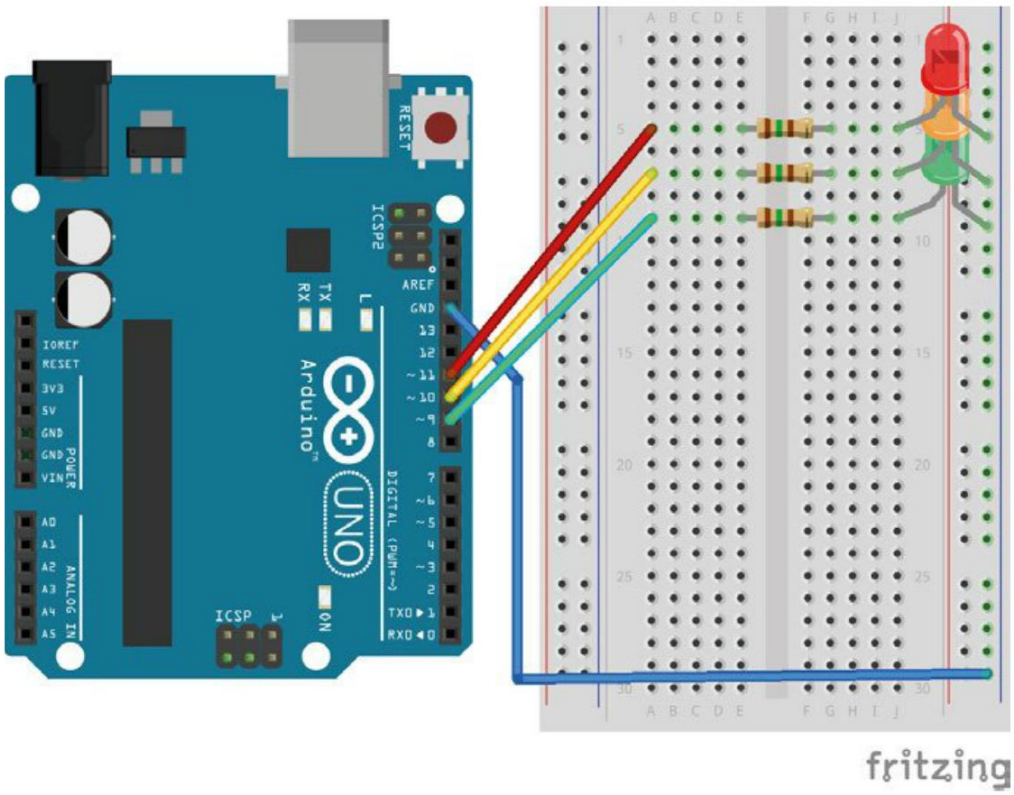


Figure 6-5. *Réalisation du circuit des feux tricolores sur la carte Arduino*

N'oubliez pas que la longue patte de la LED correspond au conducteur positif qui doit être placé du côté gauche de la plaque d'essai et une extrémité de la résistance de la LED. Le conducteur négatif plus court doit être placé dans la colonne d'alimentation négative qui se trouve le long du côté droit de la plaque d'essai.

Programme Arduino

Vous trouverez le sketch Arduino de ce projet dans le dossier `arduino/projects/traffic_signals` à l'endroit où vous avez téléchargé le code du livre (voir la section « Le code du livre » du chapitre 2 page 14). Examinons le code de plus près :

```
const int redPin = 11;                    ❶
const int orangePin = 10;
const int greenPin = 9;

void setup() {                            ❷
  pinMode(redPin, OUTPUT);
  pinMode(orangePin, OUTPUT);
  pinMode(greenPin, OUTPUT);
}
```

```
void loop() {                    ❸
  setLEDs(1, 0, 0);
  delay(3000);
  setLEDs(1, 1, 0);
  delay(500);
  setLEDs(0, 0, 1);
  delay(5000);
  setLEDs(0, 1, 0);
  delay(500);
}

void setLEDs(int red, int orange, int green) {  ❹
  digitalWrite(redPin, red);
  digitalWrite(orangePin, orange);
  digitalWrite(greenPin, green);
}
```

Le sketch présente de nombreux points communs avec le sketch permettant de faire clignoter une diode, sauf qu'au lieu de commander une seule LED, il en pilote trois.

❶ Les constantes sont définies pour chacune des broches de l'Arduino qui sont connectées à une LED.

❷ La fonction `setup()` définit les broches comme des sorties.

❸ La fonction `loop()` appelle une fonction `setLEDs()` pour allumer et éteindre (1 ou 0) les trois LED. La pause entre chaque appel de `setLEDs()` détermine la durée d'éclairage durant cette phase.

❹ La fonction `setLEDs()` sert à raccourcir la fonction `loop` afin de la rendre plus lisible en permettant de placer les trois fonctions `digitalWrite()` sur une seule ligne.

Montage Raspberry Pi

Pour la version Raspberry Pi de ce projet, les câbles flexibles mâles/mâles sont remplacés par des câbles flexibles femelles/mâles et le même circuit est réalisé sur les broches GPIO 18, 23 et 24, comme illustré à la figure 6-6.

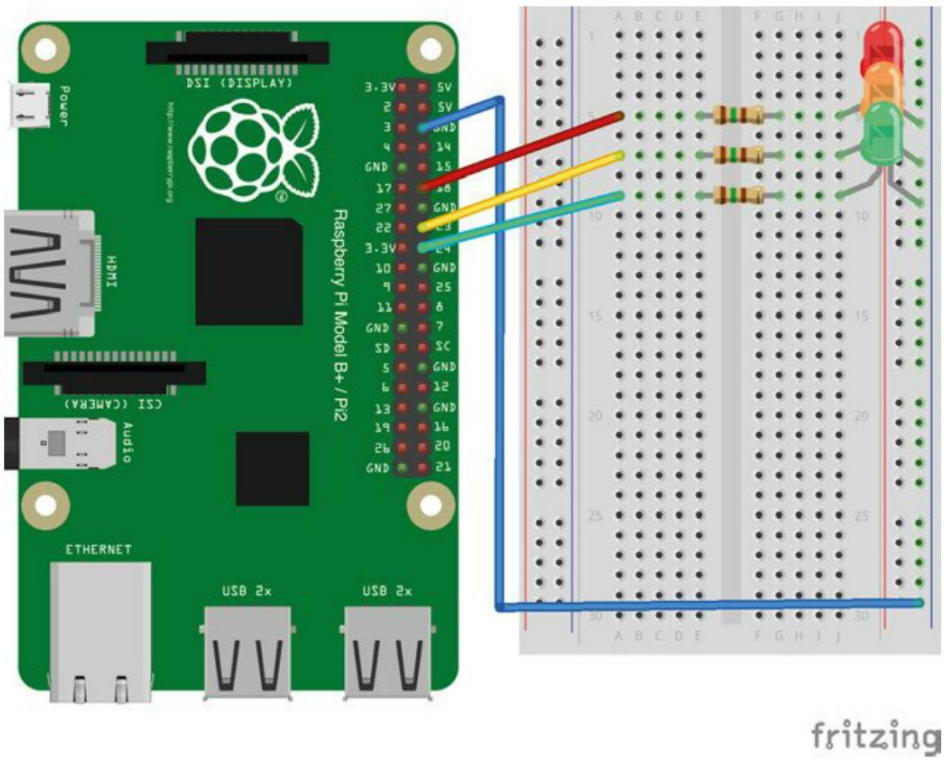


Figure 6-6. *Réalisation du circuit des feux tricolores sur le Raspberry Pi*

Programme Raspberry Pi

Vous trouverez le programme Python de ce projet dans le fichier `traffic.py` du dossier `python/projects/` (pour plus d'informations sur l'installation des programmes Python du livre, voir la section « Le code du livre » du chapitre 3 page 34) :

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

red_pin = 18
orange_pin = 23
green_pin = 24

GPIO.setup(red_pin, GPIO.OUT)
GPIO.setup(orange_pin, GPIO.OUT)
GPIO.setup(green_pin, GPIO.OUT)

def set_leds(red, orange, green):     ❶

GPIO.output(red_pin, red)
GPIO.output(orange_pin, orange)
```

```
GPIO.output(green_pin, green)

try:
    while True:
        set_leds(1, 0, 0)
        time.sleep(3)
        set_leds(1, 1, 0)
        time.sleep(0.5)
        set_leds(0, 0, 1)
        time.sleep(5)
        set_leds(0, 1, 0)
        time.sleep(0.5)

finally:
    print("Cleaning up")
    GPIO.cleanup()
```

❶ Comme la version Arduino, une fonction (`set_leds`) est utilisée pour éviter de surcharger la boucle principale avec de nombreuses fonctions `GPIO.output`.

MLI et LED

Vous aurez du mal à contrôler la luminosité d'une LED en ajustant la tension qui la traverse, car il y a une grande zone neutre avant que la tension ne soit suffisamment élevée pour que la LED s'illumine.

Les sorties analogiques MLI (ou PWM, en anglais) (voir l'encadré suivant) sont idéales pour réguler la luminosité des LED. Les LED peuvent s'allumer et s'éteindre très rapidement, avec des intervalles de moins d'un millionième de seconde. Par conséquent, lorsqu'elles sont raccordées aux sorties MLI, elles clignotent à la fréquence PWM. L'œil les voit allumées, mais avec une luminosité variable, car la proportion de la durée d'éclairage de la LED varie.

Modulation de largeur d'impulsion

Jusqu'ici, vous avez commandé des composants de façon très numérique – c'est-à-dire que vous les avez allumés, puis éteints. Et si vous vouliez les piloter de façon plus analogique ? Par exemple, peut-être voudrez-vous contrôler la vitesse d'un moteur ou la luminosité d'une LED. Dans ce cas, vous devez contrôler la quantité d'énergie fournie au composant que vous voulez commander.

La technique employée pour contrôler l'énergie de cette manière se nomme « modulation de largeur d'impulsion » (MLI, ou PWM, *Pulse-Width Modulation*). Elle utilise une sortie numérique pour produire une série d'impulsions hautes et basses. En contrôlant la durée pendant laquelle l'impulsion est haute, on peut contrôler la quantité globale d'énergie transmise à un moteur ou à une LED.

La figure 6-7 illustre le fonctionnement de la modulation de largeur d'impulsion en se basant sur la sortie 3,3 V d'une broche GPIO du Raspberry Pi. Un transistor doit aussi être connecté à la broche GPIO pour fournir suffisamment de courant pour entraîner le moteur.

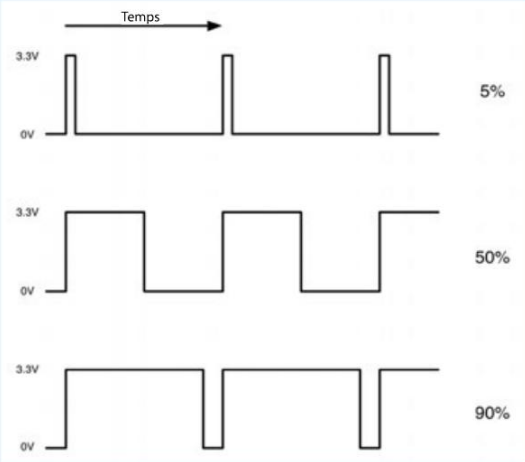


Figure 6-7. *Modulation de largeur d'impulsion*

La proportion de la durée pendant laquelle les impulsions sont hautes se nomme le rapport cyclique. Si les impulsions ne sont hautes que pendant 5 % de la période (rapport cyclique de 5 %), très peu d'énergie sera transmise au moteur. Il tournera donc très lentement (ou la luminosité de la LED sera faible). Si l'on augmente le rapport cyclique à 50 %, le moteur recevra la moitié de l'énergie et tournera probablement à la moitié de la vitesse (ou la LED brillera à la moitié de sa luminosité) maximale. Si le rapport cyclique passe à 90 %, le moteur sera presque à sa vitesse maximale (et la LED aura une forte luminosité).

Pour éteindre le moteur ou la LED, il suffit de mettre le rapport cyclique à 0. La vitesse maximale correspond à 100 %. Quand le rapport cyclique est de 100 %, la broche GPIO est à son niveau haut en permanence.

Les broches de sortie d'un Arduino et d'un Raspberry Pi peuvent produire des modulations de largeur d'impulsion. Sur une carte Arduino Uno, seules les broches D3, D5, D6, D9, D10 et D11 en sont capables. Elles sont repérées par un tilde ~ sur la carte Arduino.

La fréquence de ces impulsions peut varier sur l'Arduino et le Raspberry Pi. Sur une carte Arduino Uno, elle est de 490 Hz (impulsions par seconde) sur la plupart des broches, sauf pour les broches 5 et 6, qui atteignent une fréquence de 980 Hz. Pour contrôler la luminosité d'une LED ou la vitesse d'un moteur, la fréquence PWM par défaut de l'Arduino, qui est de 490 Hz, est suffisante.

LED RGB

La LED rouge, verte et bleue (LED RGB) se présente sous la forme d'un boîtier qui contient en fait trois LED de couleur rouge, verte et bleue. On utilise la MLI pour contrôler la luminosité de chacune des couleurs de la LED.

Bien qu'une LED RGB contienne trois LED ordinaires à deux broches, cela ne signifie pas pour autant que le boîtier de la LED compte six broches. En effet, une extrémité de chaque LED peut être reliée aux autres pour former une broche commune (figure 6-8).

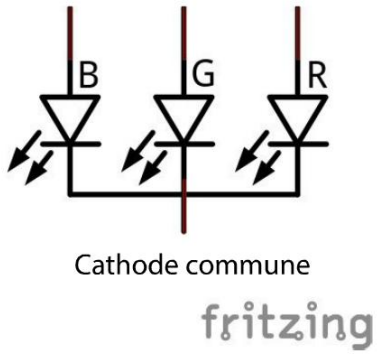


Figure 6-8. *Schéma du circuit d'une LED RGB à cathode commune*

Lorsque les connecteurs négatifs de chaque LED sont reliés ensemble, le conducteur qui en résulte se nomme une cathode commune. Lorsque les connecteurs positifs sont communs, le conducteur se nomme une anode commune.

Le boîtier de la LED RGB est transparent ou à diffusion. S'il est transparent, vous verrez les LED rouge, verte et bleue à l'intérieur du boîtier et les couleurs ne se mélangeront pas aussi bien. Les boîtiers à diffusion mélangent bien mieux la lumière des trois LED.

Au chapitre 14, vous utiliserez des afficheurs composés de circuits imprimés à LED RGB qui contiennent une puce qui limite le courant vers les LED rouge, verte et bleue et qui sert de port série permettant à un Arduino ou un Raspberry Pi de piloter un grand nombre de LED à partir d'une seule broche de sortie.

Expérience : arc-en-ciel de couleurs

Dans cette expérience, vous utiliserez un Arduino et un Raspberry Pi pour contrôler la couleur d'une LED RGB. Dans la version Raspberry Pi du projet, une interface utilisateur graphique munie de trois curseurs est utilisée pour définir la couleur. La figure 6-9 montre la version Raspberry Pi de l'expérience.

Matériel

La figure 6-10 présente le schéma du circuit de l'expérience.

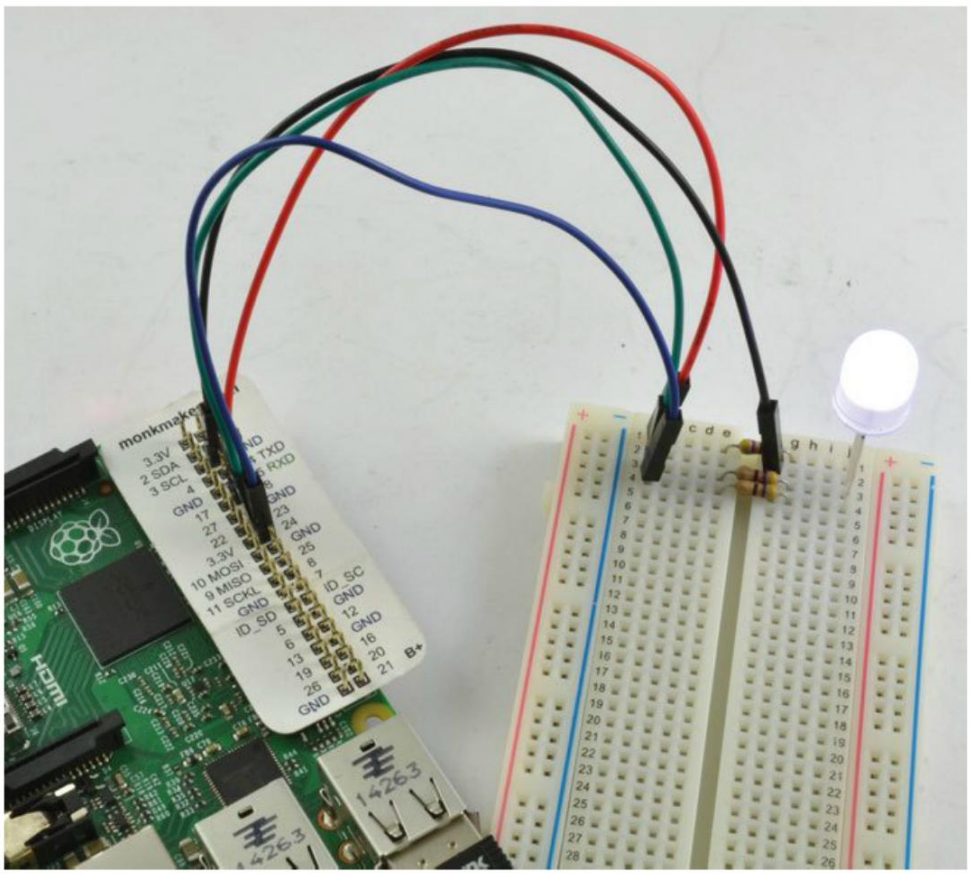


Figure 6-9. *Un arc-en-ciel de couleurs avec un Raspberry Pi*

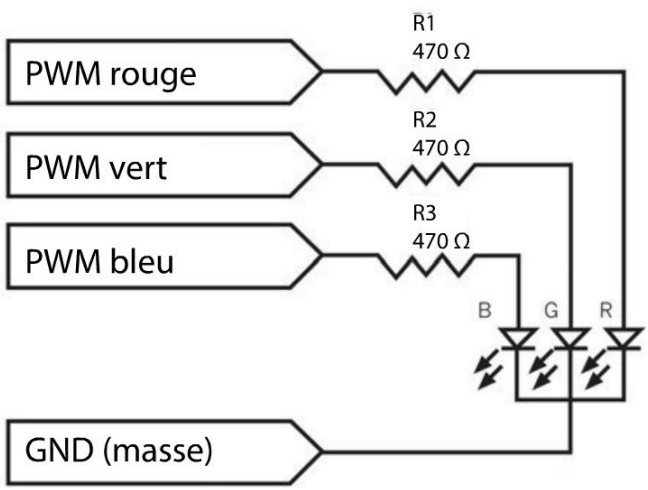


Figure 6-10. *Le schéma du circuit de l'expérience avec une LED RGB*

Choisissez attentivement les valeurs des résistances pour une luminosité optimale et un meilleur équilibre des couleurs. Toutefois, il est plus simple d'utiliser la même valeur pour les trois canaux, car cela vous évite d'acheter plusieurs modèles de résistances. Ici, je recommande des résistances 470 Ω comme solution « universelle » valable pour le Raspberry Pi ou l'Arduino.

La luminosité et l'efficacité d'une LED RGB est telle que même avec seulement 3 mA (6 mA avec une carte Arduino), la LED sera suffisamment lumineuse.

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou un Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser l'expérience.

Nom	Composant	Source
LED1	LED RGB à diffusion et à cathode commune	Sparkfun : COM-11120
R1-3	Résistance 470 Ω	Mouser : 291-470-RC
	Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
	Câbles flexibles mâles/mâles	Adafruit : 758
	Câbles flexibles femelles/mâles (Pi uniquement)	Adafruit : 826

Si vous voulez mener cette expérience avec un Raspberry Pi, vous aurez besoin de câbles flexibles femelles/mâles pour connecter les broches GPIO du Raspberry Pi à la plaque d'essai.

Tous les composants énumérés sont fournis avec le Kit de démarrage pour Raspberry Pi, de Monk Make Ltd (voir l'annexe A).

Montage Arduino

La version Arduino de cette expérience utilise le moniteur série pour définir les proportions de rouge, vert et bleu.

La figure 6-11 montre la réalisation du circuit et les connexions entre la plaque d'essai et l'Arduino.

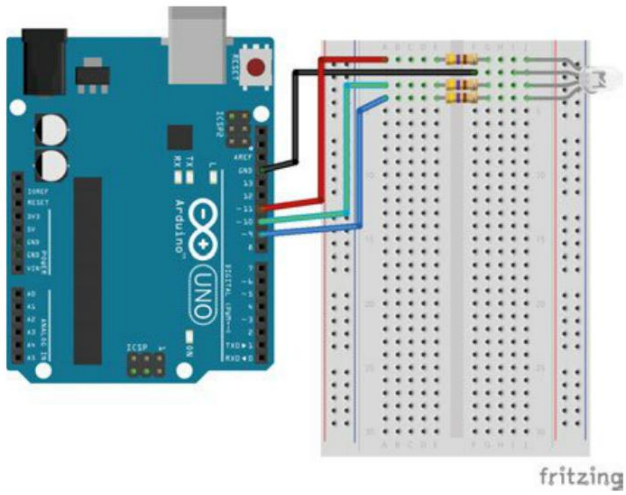


Figure 6-11. *Réalisation du circuit Arduino pour l'arc-en-ciel de couleurs RGB*

Le conducteur le plus long de la LED est la cathode commune qui est connectée à la broche GND (figure 6-11). L'ordre des autres conducteurs peut être différent de celui illustré. Dans ce cas, vous devrez modifier les connexions en fonction de cet ordre.

Programme Arduino

Le sketch Arduino de cette expérience utilise trois canaux PWM pour contrôler la luminosité de chacune des trois couleurs. Vous trouverez le sketch Arduino de ce projet dans le fichier `ex_12_mixing_colors` à l'endroit où vous avez téléchargé le code du livre :

```
const int redPin = 11;
const int greenPin = 10;
const int bluePin = 9;

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT););
  Serial.begin(9600);
  Serial.println(«Saisir R G B (ex. 255 100 200)»);
}
void loop() {
  if (Serial.available()) {
    int red = Serial.parseInt();      ❶
    int green = Serial.parseInt();
    int blue = Serial.parseInt();
    analogWrite(redPin, red);         ❷
    analogWrite(greenPin, green);
    analogWrite(bluePin, blue);
  }
}
```

❶ Chacune des trois valeurs PWM comprises entre 0 et 255 sont lues dans des variables.

❷ La sortie PWM de chaque canal est ensuite définie.

Expérimentation Arduino

Une fois le sketch chargé, ouvrez le moniteur série de l'IDE Arduino. Saisissez trois nombres compris entre 0 et 255 et séparés par des espaces, puis cliquez sur le bouton Envoyer (ou Send dans la version anglaise).

La LED doit changer de couleur en fonction des nombres saisis.

Vous pouvez vérifier le rôle de chaque canal séparément en saisissant 255 0 0 (rouge), puis 0 255 0 (vert) et 0 0 255 (bleu).

Montage Raspberry Pi

Au lieu de faire appel à des commandes pour changer la couleur, la version Raspberry Pi de ce projet crée une petite boîte de dialogue comportant trois curseurs, un par canal.

Lorsque vous modifiez la position des curseurs, la couleur de la LED change. Puisque ce logiciel utilise une interface graphique, vous aurez besoin d'un clavier, d'une souris et d'un écran qui devront être connectés à votre Raspberry Pi, car SSH ne propose pas d'interface graphique.

La réalisation de ce circuit sur le Raspberry Pi (figure 6-12) est la même que sur l'Arduino, à la différence qu'avec le Raspberry Pi, vous devez utiliser des câbles flexibles femelles/mâles. Les broches GPIO 18, 23 et 24 sont utilisées comme sorties PWM.

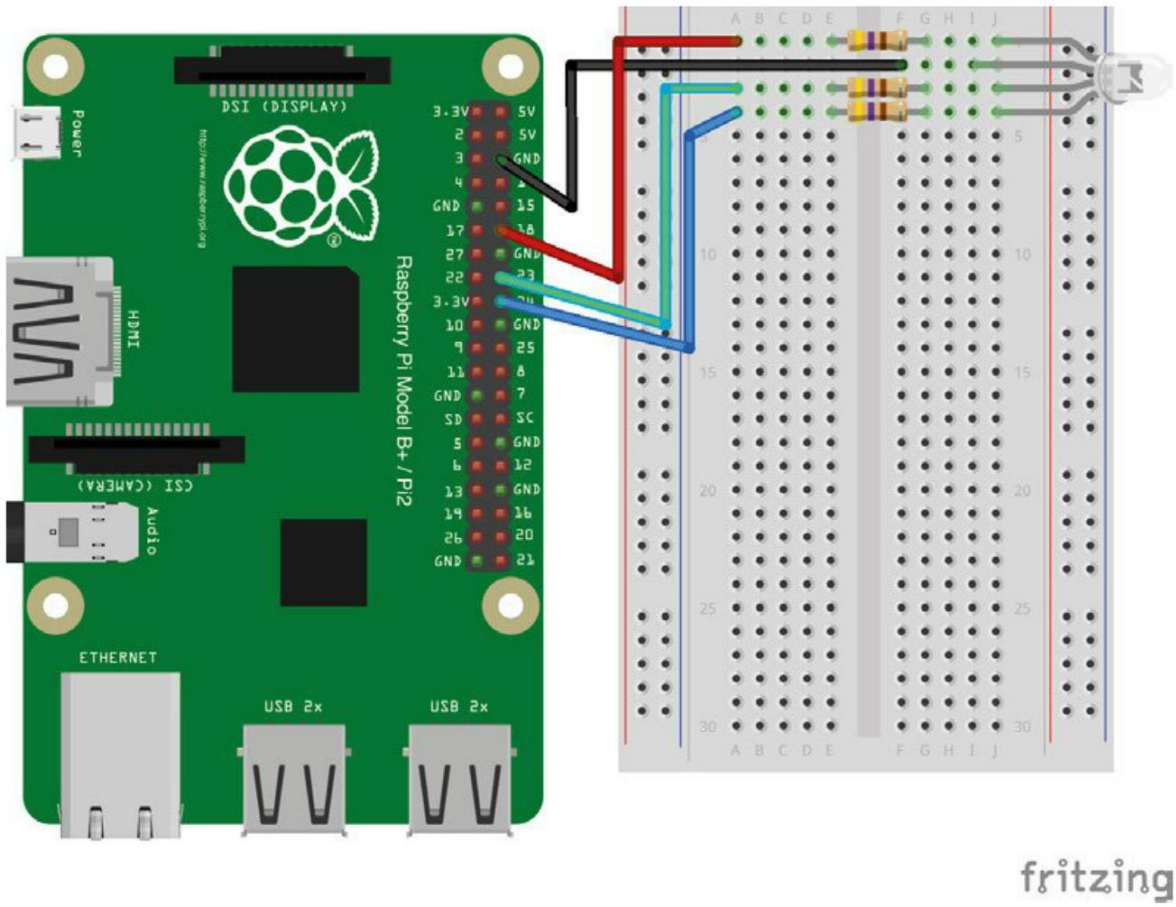


Figure 8-12. *Réalisation du circuit Raspberry Pi pour l'arc-en-ciel de couleurs RGB*

Programme Raspberry Pi

Le programme Python de cette expérience utilise un framework (structure logicielle) nommé Tkinter qui vous permet de créer des applications qui sont exécutées dans une fenêtre et qui ont une interface utilisateur graphique au lieu de la simple ligne de commande que vous avez utilisée jusqu'à présent. Cela explique pourquoi le programme est un peu plus long que d'habitude. Il utilise aussi de la programmation un peu plus avancée.

Examinons le code du programme (que vous trouverez dans le fichier `mixing_colors.py`) :

```
from Tkinter import *
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)      ❶
GPIO.setup(18, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
```

```
GPIO.setup(24, GPIO.OUT)

pwmRed = GPIO.PWM(18, 500)    ❷
pwmRed.start(100)

pwmGreen = GPIO.PWM(23, 500)
pwmGreen.start(100)

pwmBlue = GPIO.PWM(24, 500)
pwmBlue.start(100)

class App:

    def __init__(self, master): ❸
        frame = Frame(master)    ❹
        frame.pack()

        Label(frame, text='Red').grid(row=0, column=0)  ❺
        Label(frame, text='Green').grid(row=1, column=0)
        Label(frame, text='Blue').grid(row=2, column=0)

        scaleRed = Scale(frame, from_=0, to=100,  ❻
              orient=HORIZONTAL, command=self.updateRed)
        scaleRed.grid(row=0, column=1)
        scaleGreen = Scale(frame, from_=0, to=100,
              orient=HORIZONTAL, command=self.updateGreen)
        scaleGreen.grid(row=1, column=1)
        scaleBlue = Scale(frame, from_=0, to=100,
              orient=HORIZONTAL, command=self.updateBlue)
        scaleBlue.grid(row=2, column=1)

    def updateRed(self, duty):  ❼
        # change la luminosité de la led en fonction du curseur
        pwmRed.ChangeDutyCycle(float(duty))

    def updateGreen(self, duty):
        pwmGreen.ChangeDutyCycle(float(duty))

    def updateBlue(self, duty):
        pwmBlue.ChangeDutyCycle(float(duty))

root = Tk()  ❽
root.wm_title('RGB LED Control')
app = App(root)
root.geometry("200x150+0+0")
try:
    root.mainloop()
finally:
    print("Nettoyage")
    GPIO.cleanup()
```

❶ Configure le Pi pour utiliser les noms de broches Broadcom (BCM) au lieu de les désigner par leurs positions.

❷ Démarre la modulation de largeur d'impulsion (MLI) sur les canaux rouge, vert et bleu pour piloter la luminosité des LED.

❸ Cette fonction est appelée lors de la création de l'appli.

❹ Un cadre contient les différentes commandes de l'interface graphique.

❺ Crée les étiquettes et les positionne à l'intérieur d'une grille.

❻ Crée les curseurs et les positionne à l'intérieur d'une grille. L'attribut de la commande définit une méthode qui sera appelée en cas de déplacement d'un curseur.

❼ Cette méthode et les méthodes similaires pour les autres couleurs sont appelées en cas de déplacement de leur curseur respectif.

❽ Ouvre l'interface graphique et définit le titre, la taille et la position de la fenêtre.

Expérimentation Raspberry Pi

Exécutez le programme en tant que superutilisateur à l'aide de la commande suivante :

```
$ sudo python mixing_colors.py
```

Au bout de quelques instants, la fenêtre illustrée à la figure 6-13 apparaît.

Figure 6-13. *Déplacez les curseurs pour changer la couleur de la LED*

Lorsque vous déplacez les curseurs, la couleur de la LED change.

Résumé

Dans ce chapitre, vous avez appris à allumer et éteindre une LED, ainsi qu'à piloter sa luminosité à l'aide d'un Arduino et d'un Raspberry Pi. Maintenant que vous savez moduler la lumière avec l'Arduino et le Raspberry Pi, au chapitre suivant, nous nous intéresserons au mouvement. Nous étudierons les moteurs CC (à courant continu) qui sont les plus répandus et nous verrons comment les piloter.

Moteurs, pompes et vérins

7

Nous avons déjà utilisé des moteurs à courant continu (CC) au chapitre 4. La majorité des principes avec lesquels vous vous familiariserez en utilisant des moteurs CC valent aussi pour de nombreux composants pouvant être pilotés avec un Arduino ou un Raspberry Pi.

La figure 7-1 présente une sélection de moteurs CC Comme vous pouvez le voir, ils peuvent avoir des formes et des tailles variées.

Figure 7-1. *Différents moteurs CC*

Les moteurs sont aussi la force motrice indispensable à d'autres composants, comme les pompes et les vérins linéaires, que nous étudierons plus loin.

Comme vous l'avez constaté dans l'expérience « Pilotage d'un moteur » au chapitre 4, les moteurs CC exigent trop de courant pour être alimentés directement depuis la broche de sortie d'un Raspberry Pi ou d'un Arduino, mais ils peuvent être mis en marche ou arrêtés à l'aide d'un transistor.

Dans ce chapitre, vous commencerez par réguler la vitesse d'un moteur CC.

Fonctionnement des moteurs CC

Les moteurs CC ont généralement trois composants majeurs, comme illustrés à la figure 7-2. Ils ont des aimants permanents (stators) situés à l'extérieur du moteur, un rotor (la partie mobile) et un collecteur.

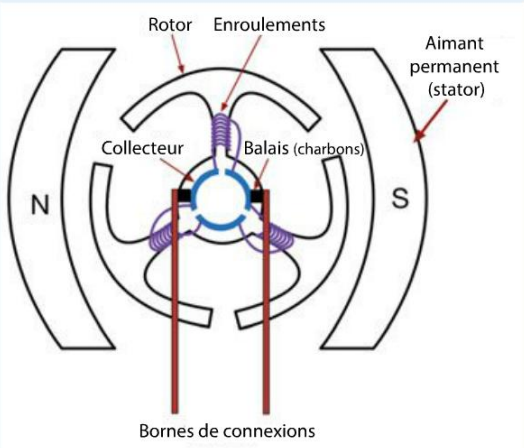


Figure 7-2. *L'intérieur d'un moteur CC*

Des bobines de fil sont enroulées autour du rotor. À la figure 7-2, il y a trois bobines autour de trois parties du rotor. Ces bobines sont reliées au collecteur. Le collecteur a pour rôle d'alimenter les bobines en électricité ayant la bonne polarité, successivement, pendant que le rotor tourne. Ainsi, la bobine suivante est toujours poussée et tirée par les aimants permanents du stator. Cela a globalement pour effet de faire tourner le rotor.

Le collecteur est une bague divisée en segments (ici, il y en a trois). Des balais connectent les bornes aux différents segments du commutateur qui tournent avec le rotor.

Ce modèle comptant trois jeux de bobines est assez fréquent dans les petits moteurs CC tels que ceux illustrés à la figure 7-1.

Une caractéristique utile des moteurs CC est que lorsque vous inversez la polarité de la tension sur ses bornes, il tourne en sens inverse.

Régulation de la vitesse par MLI

Dans l'expérience « Arc-en-ciel de couleurs » au chapitre 6, vous avez utilisé la modulation de largeur d'impulsion (voir l'encadré « Modulation de largeur d'impulsion » du chapitre 6 page 84) pour piloter la luminosité d'une LED. Vous pouvez vous servir de la même méthode pour réguler la vitesse d'un moteur.

Expérience : régulation de la vitesse d'un moteur CC

Cette expérience utilise les mêmes composants que l'expérience « Pilotage d'un moteur » au chapitre 4, mais au lieu de nous contenter d'allumer et d'éteindre le moteur, nous allons en réguler la vitesse.

Matériel

Réunissez le matériel nécessaire pour expérience « Pilotage d'un moteur » au chapitre 4. Le circuit réalisé sur la plaque d'essai est illustré à la figure 7-3.

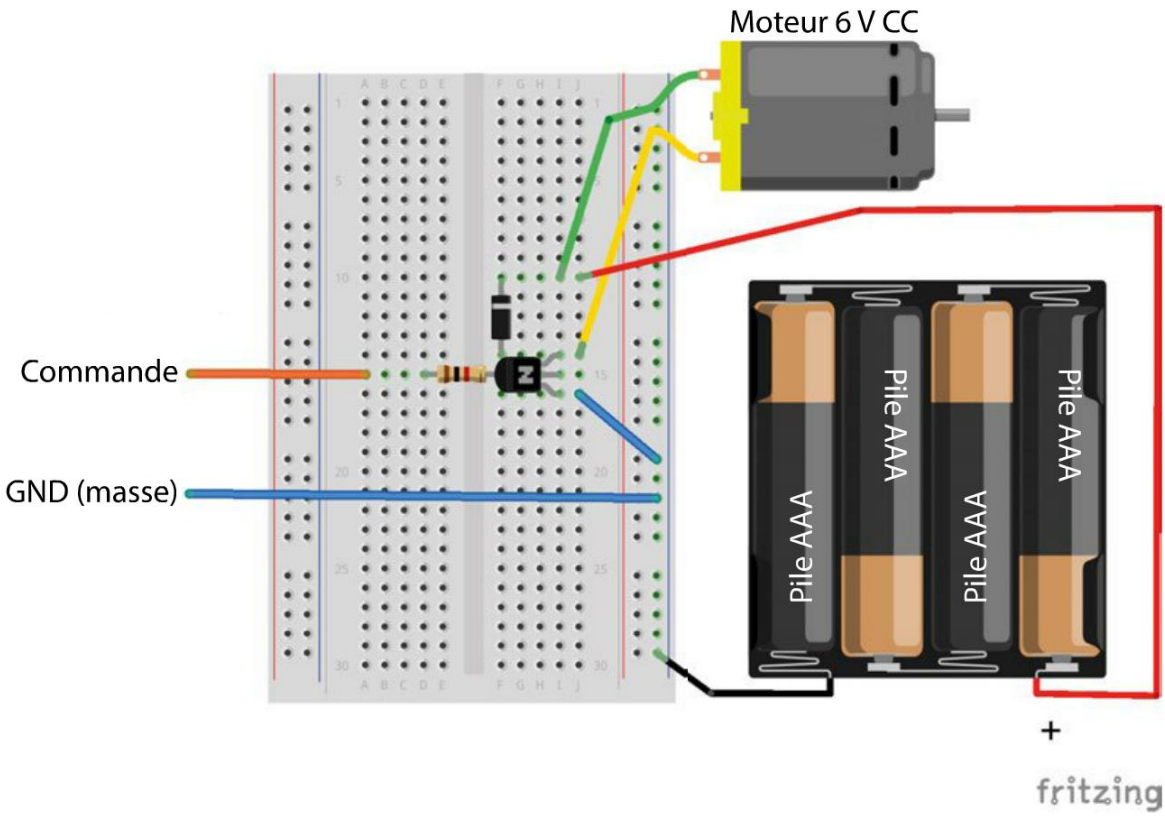


Figure 7-3. *Réalisation du circuit de commande de l'expérience*

Montage Arduino

Réalisez les connexions Arduino comme illustré à la figure 7-4. Connectez la broche GND de la plaque d'essai au GND et la liaison de commande de la plaque d'essai à la broche D9 de l'Arduino Uno.

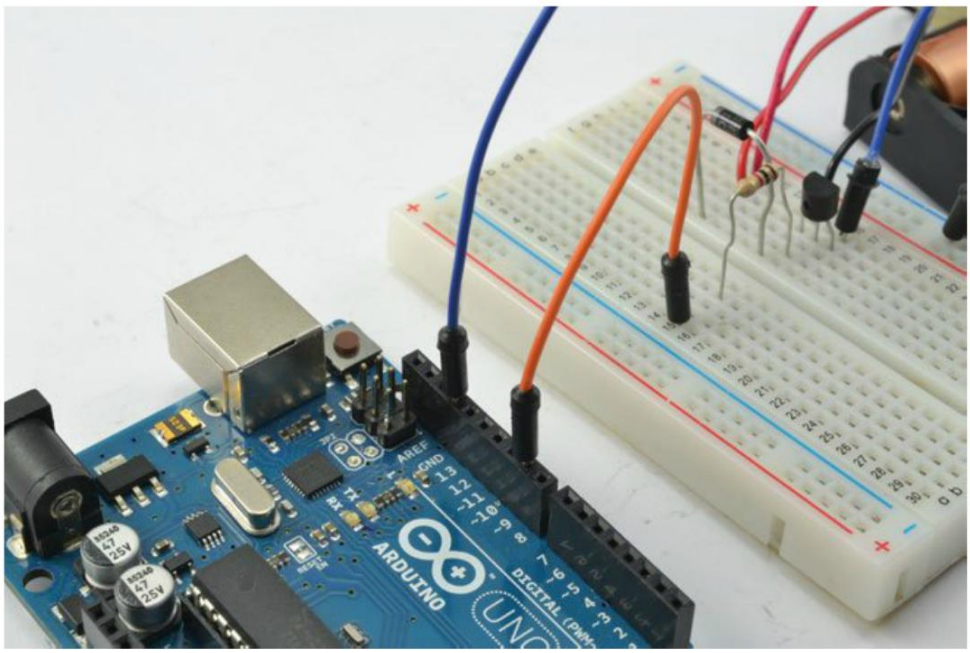


Figure 7-4. *Raccordement de l'Arduino à la plaque d'essai*

Programme Arduino

Vous trouverez le sketch Arduino de ce projet dans le dossier `/experiments/pwm_motor_control` à l'endroit où vous avez téléchargé le code du livre (voir la section « Le code du livre » du chapitre 2 page 14).

Le programme utilise le moniteur série de l'IDE Arduino pour vous permettre de saisir un rapport cyclique afin de réguler la vitesse du moteur :

```
const int controlPin = 9;

void setup() {                          ❶
    pinMode(controlPin, OUTPUT);
    Serial.begin(9600);
    Serial.println(«Saisir le rapport cyclique (0 à 100)»);
}

void loop() {                           ❷
    if (Serial.available()) {           ❸
      int duty = Serial.parseInt();
      if (duty < 0 || duty > 100) {     ❹
        Serial.println("0 à 100");
      }
      else {
        int pwm = duty * 255 / 100;
        analogWrite(controlPin, pwm);
```

```
        Serial.print(«Rapport cyclique de «);
        Serial.println(duty);
      }
    }
}
```

❶ La fonction `setup` définit `controlPin` comme sortie et démarre la communication série à l'aide de `Serial.begin`. Cela vous permet de transmettre des valeurs de rapport cyclique à l'Arduino à partir de votre ordinateur.

❷ La fonction `loop()` vérifie la présence éventuelle d'une communication série en attente qui aurait été transmise par le port USB à l'aide de `Serial.available`.

❸ S'il y a un message sous forme de nombre, le nombre est lu dans la chaîne de caractères et converti en `int` à l'aide de `parseInt`.

❹ La valeur est analysée afin de vérifier qu'elle est comprise entre 0 et 100. Si ce n'est pas le cas, un rappel est envoyé au moniteur série par le port USB.

Texte et nombre

Comme nous utiliserons assez souvent cette technique de lecture de nombres dans le moniteur série de l'IDE Arduino, cela vaut la peine de s'attarder ici sur le rôle de la fonction `parseInt` et sur ce qu'il se passe lorsque nous envoyons un message à l'Arduino par le port USB.

Dans la section « Communication série » du chapitre 7 page 73, j'ai mentionné qu'il est possible d'interfacer l'Arduino (et le Raspberry Pi) en communicant par une interface série. L'Arduino Uno possède une interface série sur les broches numériques D0 et D1. Ces broches ne doivent pas être utilisées comme des entrées et sorties numériques générales parce qu'elles constituent l'interface série entre l'Arduino et votre ordinateur via un circuit intégré d'interface USB qui se charge de la conversion entre le port série USB et la liaison série directe comprise par l'Arduino.

Lorsque vous saisissez un message dans le moniteur série et que vous l'envoyez à l'Arduino, le texte du message est converti en un flux de bits (signaux hauts et bas) qui sont reconstruits en groupes de huit bits (octets) à leur réception. Chacun de ces octets est un code numérique constitué d'une lettre de l'alphabet romain. Toutes les lettres et les chiffres ont un code unique défini par la norme ASCII *(American Standard Code for Information Interchange)*.

Lorsqu'un message est reçu via le port série par le programme Arduino en cours d'exécution, celui-ci peut lire un seul octet (lettre) à la fois. Ou bien, il utilise la fonction `parseInt` qui lit les caractères en continu. Tant que le caractère est un chiffre, il construit un nombre à partir de ce caractère. Par exemple, le nombre 154 est envoyé sous la forme de trois caractères (1, 5 et 4). Si le caractère est suivi d'un caractère de retour à la ligne, d'un espace ou d'un caractère qui n'est pas un chiffre, la fonction `parseInt` en déduit que toutes les lettres du nombre ont été reçues et renvoie la valeur 154 comme `int`. Par ailleurs, même s'il n'y a qu'une courte pause après la transmission de la dernière lettre, c'est suffisant pour que la fonction `parseInt` en déduise que le nombre est terminé.

Si le nombre est compris entre 0 et 100, la valeur est convertie en un nombre compris entre 0 et 255, puis la fonction `analogWrite()` est utilisée pour définir la valeur PWM. C'est nécessaire, car la fonction Arduino `analogWrite` accepte une valeur de rapport cyclique comprise entre 0 et 255, où 0 correspond à 0 % et 255 équivaut à 100 % du rapport cyclique.

Expérimentation Arduino

Sur l'ordinateur depuis lequel vous avez programmé la carte Arduino, ouvrez le moniteur série de l'IDE Arduino en cliquant sur la loupe qui se trouve dans le coin supérieur droit de l'IDE. L'icône est entourée à la figure 7-5.

Vous devez alors saisir une valeur de rapport cyclique comprise entre 0 et 100. Saisissez différentes valeurs pour en observer l'effet sur la vitesse du moteur.

Vous remarquerez que lorsque la valeur est de 10 ou 20, le moteur ne tourne pas, mais émet une sorte de couinement indiquant que la puissance est insuffisante pour surmonter la friction et pour entrer en mouvement.

Si vous voulez utiliser le moteur à des fins pratiques, ce sketch est un excellent moyen de déterminer la valeur minimale utile du rapport cyclique du moteur.

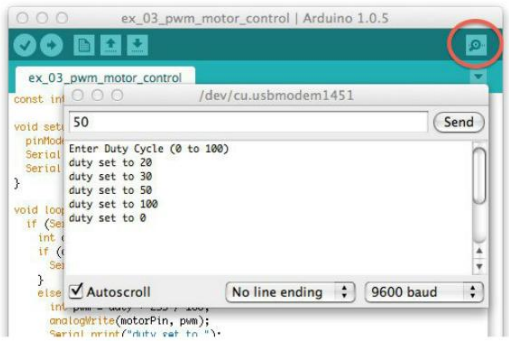


Figure 7-5. *Utilisation du moniteur série pour réguler la vitesse du moteur*

Pour arrêter le moteur, saisissez un rapport cyclique de 0.

Montage Raspberry Pi

Raccordez la plaque d'essai au Raspberry Pi, comme illustré à la figure 7-6. À l'aide de câbles flexibles femelles/mâles, reliez la broche GND de la plaque d'essai à l'une des broches GND du connecteur GPIO et la liaison de commande de la plaque d'essai à la broche 18 du Raspberry Pi.

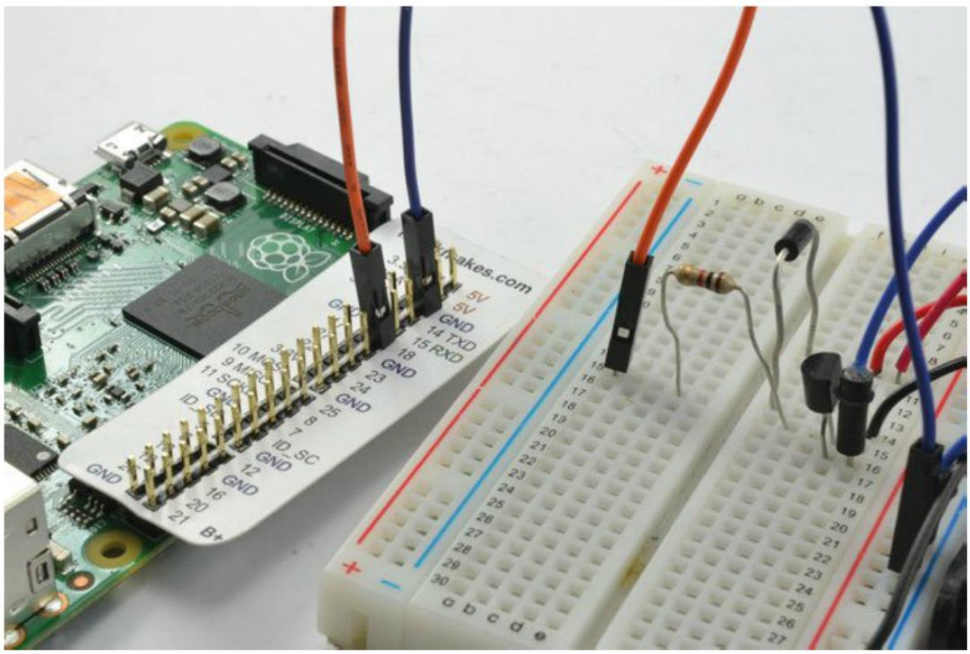


Figure 7-6. *Raccordement de la plaque d'essai de régulation du moteur à un Raspberry Pi*

Programme Raspberry Pi

La structure du programme Raspberry Pi ressemble beaucoup à celle du code Arduino. Ici, le programme vous demande de saisir une valeur de rapport cyclique et il pilote la broche 18 en conséquence.

Le code du programme suivant se trouve dans le fichier `pwm_motor_control.py` du dossier python/experiments, à l'endroit où vous avez téléchargé le code du livre (voir la section « Le code du livre » du chapitre 3 page 34).

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

control_pin = 18        ❶

GPIO.setup(control_pin, GPIO.OUT)
motor_pwm = GPIO.PWM(control_pin, 500) ❷
motor_pwm.start(0)                     ❸

try:
    while True:                        ❹
        duty = input('Saisir le rapport cyclique (0 à 100) : ')
        if duty < 0 or duty > 100:
            print('0 à 100')
```

```
            else:
                motor_pwm.ChangeDutyCycle(duty)

    finally:
    print("Nettoyage")
    GPIO.cleanup()
```

❶ La première partie du programme est identique au code de l'expérience « Pilotage d'un moteur » au chapitre 4, mais la broche 18 est définie comme sortie.

❷ Cette ligne définit la broche comme sortie PWM. La bibliothèque RPi.GPIO vous permet de définir n'importe quelle broche GPIO comme sortie PWM. Le paramètre 500 règle la fréquence PWM sur 500 Hz (impulsions par seconde).

❸ La sortie PWM ne débute pas avant l'appel de `start`. Elle a pour paramètre le rapport cyclique initial. Comme, au départ, nous voulons que le moteur soit éteint, le paramètre est défini sur 0.

❹ À l'intérieur de la boucle principale, un message s'affiche pour vous demander de saisir la valeur de `duty`. La plage de la valeur saisie est vérifiée. Si elle est comprise entre 0 et 100, elle est utilisée pour définir le rapport cyclique à l'aide de la fonction `ChangeDutyCycle`.

Expérimentation Raspberry Pi

Exécutez le programme en tant que superutilisateur en saisissant `sudo`, puis saisissez différentes valeurs de rapport cyclique. La vitesse du moteur devrait changer de la même façon qu'elle l'a fait avec le sketch Arduino :

```
pi@raspberrypi ~/make_action/python $ sudo python pwm_motor_control.py
Enter Duty Cycle (0 to 100): 50
Enter Duty Cycle (0 to 100): 10
Enter Duty Cycle (0 to 100): 100
Enter Duty Cycle (0 to 100): 0
Enter Duty Cycle (0 to 100):
```

Pilotage d'un moteur CC à l'aide d'un relais

Si vous avez simplement besoin que la carte Arduino ou le Raspberry Pi allume et éteigne occasionnellement un moteur, une méthode consiste à utiliser un relais. Même si cette façon de faire peut paraître démodée, elle présente néanmoins quelques avantages.

- Sa réalisation est simple, car elle nécessite peu de composants.
- Extrêmement bonne isolation entre le moteur, qui génère du bruit électrique et nécessite un courant élevé, et les circuits sensibles Pi ou Arduino.
- Traitement de courants élevés (avec le relais adéquat).
- Vous pouvez utiliser des modules de relais prêts à l'emploi directement avec un Raspberry Pi ou un Arduino.

Parmi les inconvénients dus à l'utilisation d'un relais ou d'un module de relais, on peut citer :

- Ce sont des composants relativement volumineux.
- Ils peuvent uniquement allumer ou éteindre le moteur, mais ne peuvent pas contrôler la vitesse.
- Ce sont des organes électromécaniques extrêmement solides ayant une grande longévité.

Relais électromécaniques

La figure 7-7 présente le type de relais le plus répandu qui est surnommé « relais morceau de sucre » à cause de sa forme, mais pas de sa couleur qui est généralement noire.

Le principe général d'un relais électromécanique comme celui-ci est que lorsqu'un courant (d'environ 50 mA) alimente sa bobine, il agit comme un électroaimant et rapproche deux contacts pour établir une liaison. Ces contacts peuvent être adaptés à des courants et des tensions élevés afin de commuter des dizaines d'ampères.

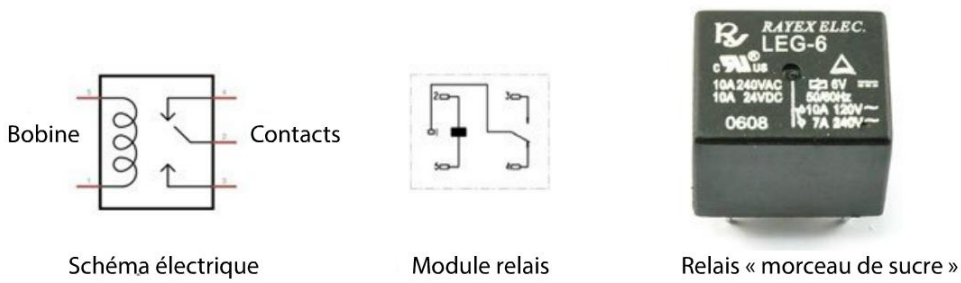


Schéma électrique | Module relais | Relais « morceau de sucre »

Figure 7-7. *Relais*

Dans ce chapitre, le relais sert uniquement de commutateur. Vous pouvez donc vous en servir pour piloter divers composants.

Les relais de ce type sont dits à *Single Pole Change Over* (SPCO)ou inverseurs, car au lieu de n'avoir que deux bornes qui peuvent soit être connectées soit ne pas être connectées, ils en ont trois : la broche commune (COM), la broche NO (*Normally Open*, normalement ouverte) et la broche NC (*Normally Closed*, normalement fermée). Dans ce contexte, l'état « normal » signifie que la bobine n'est pas alimentée. Les contacts NO et COM seront donc « ouverts » (non connectés) jusqu'à ce que la bobine soit alimentée. Le contact NC présente le comportement inverse : les bornes NC et COM sont « normalement » connectées et elles sont déconnectées lorsque la bobine est alimentée.

En général, seules les bornes NO et COM du relais sont utilisées pour la commutation.

Commutation d'un relais avec Arduino ou Raspberry Pi

Lorsque vous utilisez un relais avec un Raspberry Pi ou un Arduino, choisissez un modèle ayant une tension de 5 V sur la bobine. Les bobines des relais demandent trop de courant (environ 50 mA) pour être alimentées directement depuis un Raspberry Pi ou un Arduino. Dans les deux cas, on utilise un petit transistor pour commuter la bobine du relais à 5 V.

La figure 7-8 présente le schéma du circuit de commutation d'un relais.

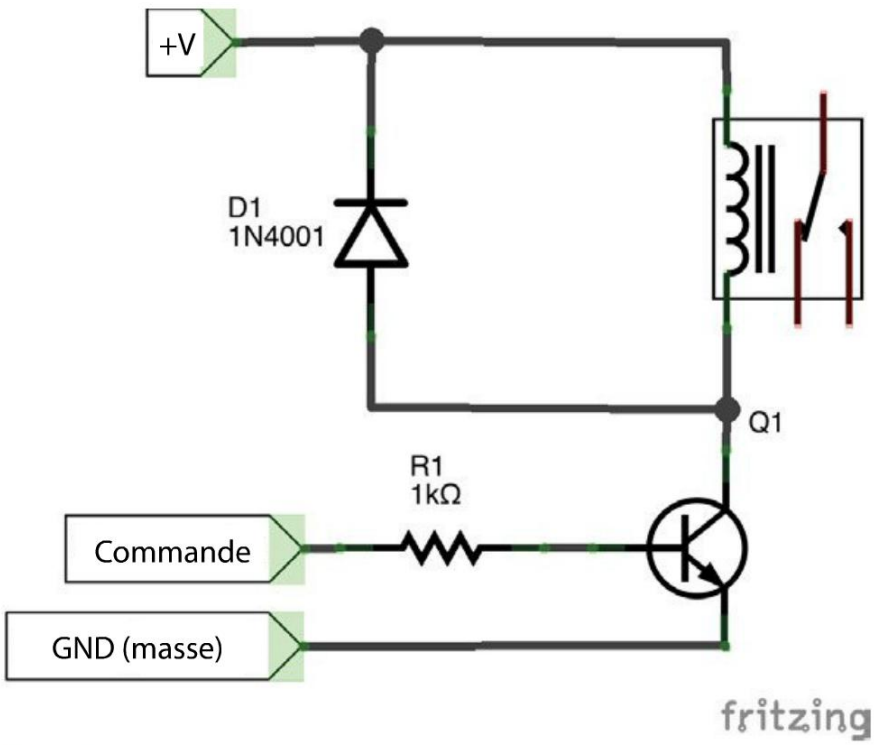


Figure 7-8. *Utilisation d'un transistor pour commuter un relais*

La bobine d'un relais conçue pour fonctionner à 5 V nécessite environ 50 mA de courant. Cela dépasse les capacités de l'Arduino et c'est beaucoup trop pour la broche GPIO d'un Raspberry Pi. Comme dans l'expérience « Pilotage d'un moteur » au chapitre 4, vous devrez donc utiliser un transistor pour piloter le moteur (ici, la bobine du relais remplacera le moteur en tant que « charge ».

Cette précaution n'a de sens que si le moteur (ou toute charge devant être pilotée) consomme tant de courant qu'il ne peut pas être piloté directement à l'aide du transistor.

De la même façon qu'un moteur, la bobine d'un relais risque de produire des pics de tension quand elle est allumée et éteinte. Par conséquent, ce montage nécessite aussi une diode.

La figure 7-8 montre que la partie commutation du relais est entièrement isolée électriquement de la partie bobine. Cela signifie qu'il y a moins de chance que des bruits électriques, des pics de tension ou d'autres perturbations électriques ne remontent jusqu'à l'Arduino et le Raspberry Pi.

Comme le relais n'exige qu'une alimentation de 50 mA, un modeste transistor 2N3904, qui ne coute que quelques centimes, devrait suffire.

Modules relais

Si vous voulez piloter plusieurs composants compatibles avec les limitations des relais décrites plus haut (états allumé/éteint uniquement), mieux vaut acheter un module relais comme celui illustré à la figure 7-9.

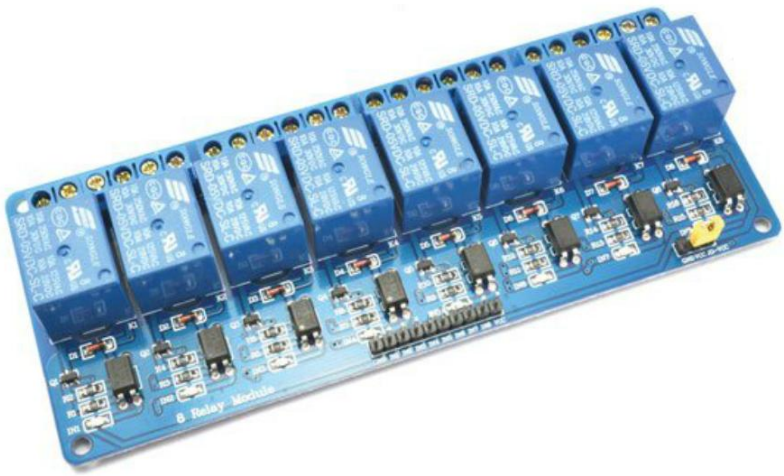


Figure 7-9. *Un module de relais à 8 canaux*

Ces modules bon marché sont disponibles sur eBay et Amazon. Ils se composent de relais et de transistors pour les piloter, ainsi que de petites LED qui indiquent qu'un relais particulier a été activé. Vous pouvez donc les connecter directement à un Raspberry Pi ou un Arduino.

Ces modules peuvent contrôler entre un et plus de huit relais, comme celui illustré à la figure 7-9.

Les modules possèdent habituellement les broches suivantes.

- `GND` (Ground, masse)
- `VCC` ou `5V` Connectez-la à la broche 5 V du Raspberry Pi ou de l'Arduino pour alimenter les bobines des relais en cas d'activation.
- `Broches de données` Chaque broche de données contrôle l'un des relais. Ces broches sont tantôt « actives hautes » (vous devez alors définir la broche GPIO à laquelle elles sont connectées au niveau haut pour les activer) tantôt « actives bas » (la bobine du relais est activée quand la broche est à l'état bas.

Le module possède aussi une rangée de bornes à vis qui sont directement connectées aux contacts du relais.

Expérience : pilotage d'un moteur CC à l'aide d'un module de relais

Dans cette expérience, vous activerez et arrêterez la rotation d'un moteur à l'aide d'un relais.

Ce projet utilise un module de relais prêt à l'emploi qui ne nécessite qu'un seul relais. Le module que j'utilise dispose de huit relais, mais vous pouvez utiliser un module qui en compte moins.

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou un Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser l'expérience.

Composant	Source
Petit moteur CC 6 V	Adafruit : 711
Module de relais	eBay
Support pour piles (4 × AA) 6 V	Adafruit : 830
Câbles flexibles adaptés au module de relais	Voir l'annexe A

Certains modules de relais ont des prises femelles et d'autres des fiches mâles pour être connectés à l'Arduino ou au Raspberry Pi. Choisissez des câbles flexibles adaptés. Pour connecter un module de relais avec des broches mâles à un Arduino, vous aurez besoin de câbles flexibles femelles/mâles (Adafruit : 826). Pour le connecter à un Raspberry Pi, vous aurez besoin de câbles flexibles femelles/femelles (Adafruit : 266).

Câblage

Le schéma de câblage de cette expérience est illustré à la figure 7-10.

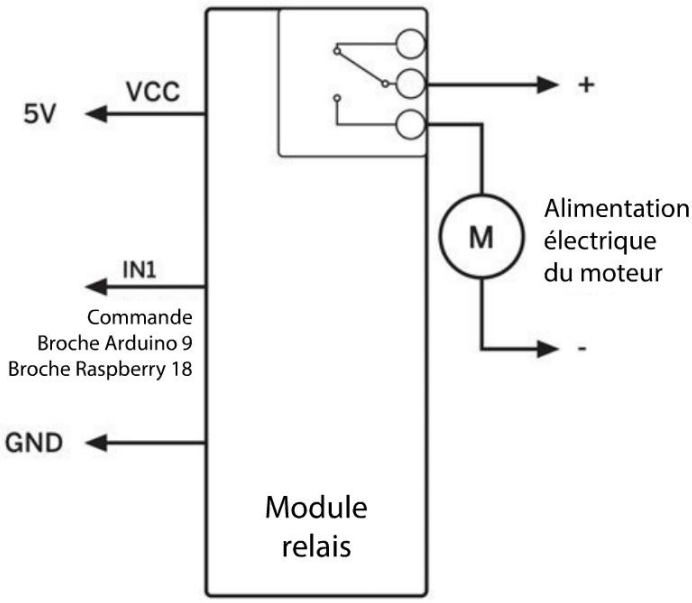


Figure 7-10. *Schéma de câblage pour le pilotage d'un moteur CC avec un module de relais*

Les contacts du relais fonctionnent comme un commutateur pouvant avoir deux positions. Il dispose des bornes suivantes : commune, normalement ouverte et normalement fermée. Quand la bobine du relais n'est pas excitée, la borne commune est connectée à la borne normalement fermée. Quand l'électricité parvient à la bobine, le commutateur s'inverse et la borne commune se retrouve connectée à la borne normalement ouverte.

Le programme Arduino et Raspberry Pi est très proche de celui de l'expérience « Pilotage d'un moteur » au chapitre 4, sauf si le module de relais a une logique « active bas » comme le mien.

Programme Arduino

Vous trouverez le sketch Arduino de ce projet dans le dossier `arduino/experiments/relay_motor_control` à l'endroit où vous avez téléchargé le code du livre (voir la section « Le code du livre » du chapitre 2 page 14).

Le programme allume le relais (et donc le moteur) pendant 5 secondes, puis il l'éteint pendant 2 secondes, avant de recommencer. Voici l'intégralité du code :

```
const int controlPin = 9;

void setup() {
  pinMode(controlPin, OUTPUT);
}
```

```
void loop() {
  digitalWrite(controlPin, LOW);    ❶
  delay(5000);
  digitalWrite(controlPin, HIGH);
  delay(2000);
}
```

❶ Ce code est identique à l'expérience « Pilotage d'une LED » au chapitre 4, sauf que `LOW` et `HIGH` sont intervertis sur les fonctions `digitalWrite`. Si vous constatez que le moteur tourne pendant 2 secondes puis s'éteint pendant 5 secondes, au lieu du contraire, cela signifie que la logique de votre module de relais est « active haute ». Vous devez donc inverser les constantes `LOW` et `HIGH`, comme dans l'expérience « Pilotage d'une LED » au chapitre 4.

Programme Raspberry Pi

Le code du programme suivant se trouve dans le fichier `relay_motor_control.py` du dossier `python/experiments` à l'endroit où vous avez téléchargé le code du livre.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

control_pin = 18

GPIO.setup(control_pin, GPIO.OUT)

try:
    while True:
        GPIO.output(control_pin, False)
        time.sleep(5)
        GPIO.output(control_pin, True)
        time.sleep(2)

finally:
    print("Nettoyage")
    GPIO.cleanup()
```

Choix d'un moteur

Il existe des moteurs de différentes formes et dimensions. Il est donc important d'en choisir un qui soit suffisamment puissant pour votre projet. Les deux caractéristiques essentielles à connaître à propos d'un moteur sont sa force de torsion (le couple) et sa vitesse de rotation.

Lorsqu'il tourne plus vite que nécessaire, mais que sa force est insuffisante, vous pouvez compenser les deux à l'aide d'une boîte à engrenages.

Couple

Pour simplifier, le couple est la force de torsion d'un moteur. Plus le couple est élevé, plus sa force de torsion est importante.

D'un point de vue scientifique, le couple est défini comme une force multipliée par une distance, la force étant mesurée en newtons (N) et la distance en mètres (m). Concrètement, la force du couple est souvent exprimée comme étant la force nécessaire pour soulever un certain poids en kilos sur une distance mesurée en centimètres.

La distance intervient aussi dans l'équation, car plus la distance par rapport au rotor du moteur est grande, moins celui-ci pourra exercer de force. Par exemple, si un moteur a un couple de 15 kg/cm, alors à 1 cm du centre du rotor, il pourra porter un poids de 15 kg — c'est-à-dire que le moteur ne pourra pas soulever le poids plus haut, mais le poids ne tombera pas. À 10 cm du rotor, il pourra seulement porter un poids de 1,5 kg (10 cm × 1,5 kg = 15 kg/cm).

La figure 7-11 illustre la relation entre le poids et la distance de l'axe du moteur.

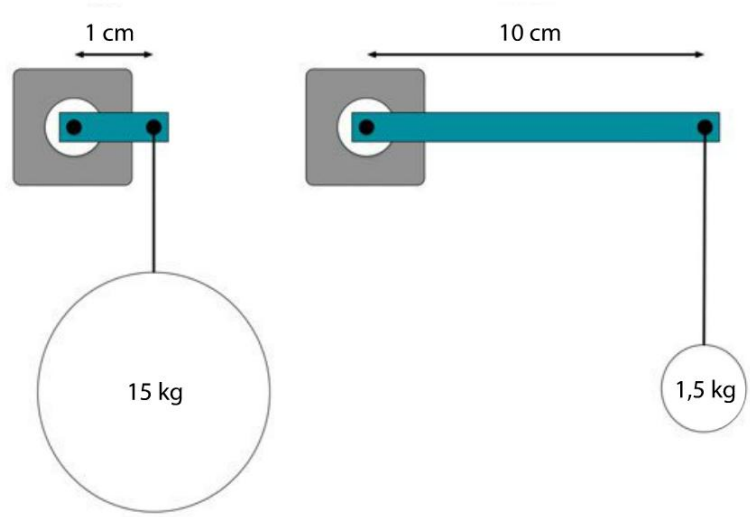


Figure 7-11. *Le couple d'un moteur*

Tr/min

Comme les moteurs tournent assez vite, ils sont souvent utilisés avec des engrenages ou comme moteurs à engrenages (voir les deux sections suivantes). Un moteur CC basse tension ordinaire peut tourner à 10 000 tours par minute. Cela signifie que l'arbre du moteur tourne 166 fois par seconde.

Même si vous pouvez réguler la vitesse d'un moteur par MLI, cela réduit aussi son énergie, donc le couple généré par le moteur reste faible.

Engrenages

Les engrenages permettent de produire une rotation plus lente ayant pour effet de produire un couple supérieur. Supposons que vous utilisiez une boîte à engrenages 5:1 (figure 7-12) avec 50 dents sur un rouage et 10 sur l'autre. Dans ce cas, pour cinq tours de moteur, il n'y aura qu'un seul tour à la sortie de la boîte à engrenages, mais le couple disponible à la sortie de la boîte sera dix fois supérieur à celui disponible directement sur le moteur.

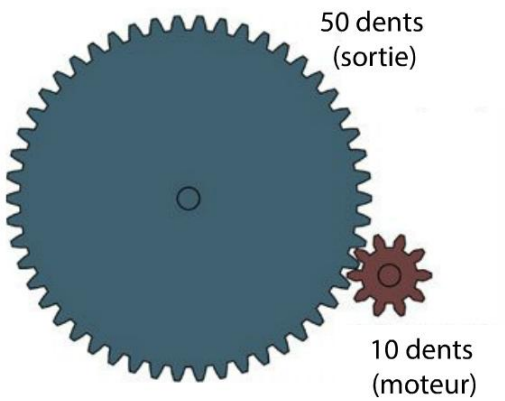


Figure 7-12. *Engrenages*

Moteurs à engrenages

Comme les moteurs sont souvent utilisés avec des engrenages, pour de nombreuses applications, il est préférable d'acheter un moteur à engrenages qui combine un moteur avec une boîte à engrenages dans un seul boîtier.

Vous trouverez notamment un vaste choix de moteurs à engrenages sur le site de Pololu (*https://www.pololu.com/*).

Vous trouverez des modèles extrêmement bon marché employant des rouages en plastique. Ils ne sont pas aussi résistants et ne produiront pas autant de couple que les modèles à engrenages métalliques.

Pompes

Les pompes sont généralement des moteurs CC, ou parfois des moteurs CC sans balais (voir la section « Moteurs CC sans balais » du chapitre 10 page 203), qui actionnent un mécanisme permettant de déplacer un liquide.

Parmi les types de pompes fréquemment utilisées par les amateurs, il y a la pompe péristaltique et la pompe rotodynamique. Elles sont illustrées côte à côte à la figure 7-13. La pompe péristaltique se trouve à gauche.

Figure 7-13. *Une pompe péristaltique (à gauche) et une pompe rotodynamique (à droite)*

Ces deux types de pompes sont actionnés par des moteurs CC, mais leurs propriétés varient. Pour un mouvement lent et mesuré, utilisez une pompe péristaltique ; pour un mouvement rapide, préférez une pompe rotodynamique.

Pompes péristaltiques

Les pompes péristaltiques sont conçues pour déplacer du liquide de façon très contrôlée. D'ailleurs, elles sont souvent utilisées dans des applications médicales et scientifiques pour déplacer une quantité de liquide très précise. Pour une régulation encore plus précise du flux, les pompes péristaltiques sont parfois actionnées par des moteurs pas-à-pas (chapitre 10).

La figure 7-14 présente le fonctionnement d'une pompe péristaltique.

La pompe utilise un moteur à engrenages qui actionne des galets qui pressent un tube flexible pour pousser le liquide qui se trouve à l'intérieur du tube. Comme il est soumis à des pressions constantes, le tube doit être remplacé régulièrement. Les pompes sont généralement conçues pour permettre de remplacer facilement le tube.

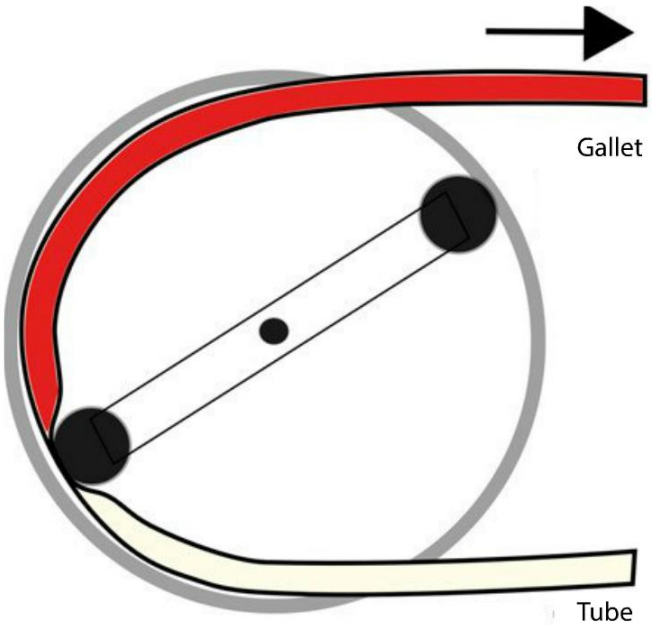


Figure 7-14. *Fonctionnement d'une pompe péristaltique*

Si vous pilotez le moteur à engrenages d'une pompe péristaltique par MLI et à l'aide d'un pont en H (chapitre 8), vous pouvez contrôler à la fois le débit et le sens d'écoulement du liquide.

Une pompe péristaltique s'amorce automatiquement – c'est-à-dire que si la pompe se trouve un peu plus haut que la source de liquide, elle créera suffisamment de succion pour attirer l'eau dans la pompe et commencer à pomper.

Débit volumique

Le débit volumique correspond à la quantité de liquide qu'une pompe peut déplacer en une unité de temps. Diverses unités sont employées pour le volume de liquide et l'unité de temps. Une petite pompe péristaltique peut avoir un débit volumique de 50 ml/min (millilitres/minute). Une pompe rotodynamique pour le jardin peut atteindre un débit volumique de 5 l/min (5 litres/minute).

Pompes rotodynamiques

Si vous avez besoin de déplacer rapidement une grande quantité de liquide, vous aurez besoin d'une pompe rotodynamique. Il existe différents modèles de pompes rotodynamiques, mais la plus répandue est probablement la pompe centrifuge (figure 7-15).

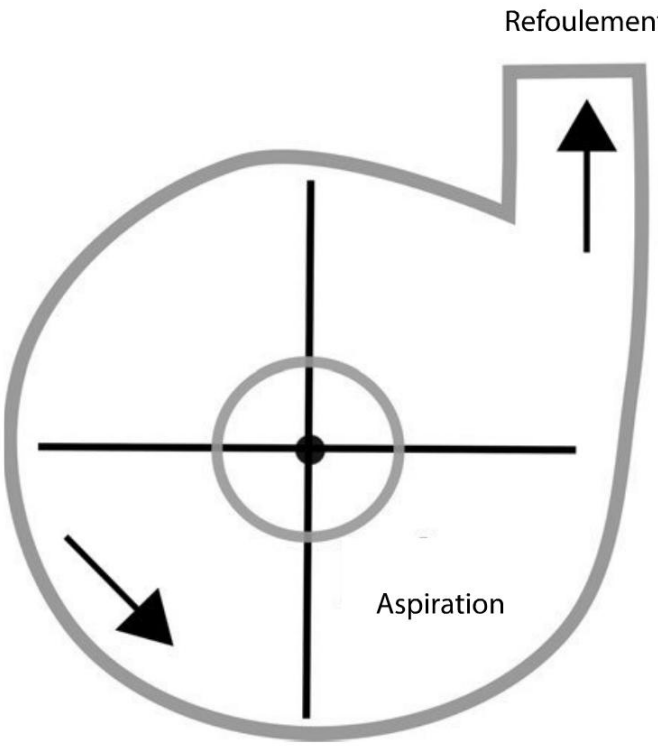


Figure 7-15. *Fonctionnement d'une pompe centrifuge*

À la figure 7-15, le liquide entre dans la pompe par l'avant, dans l'axe du moteur qui actionne une aube. Cette aube transmet une force centrifuge au liquide qui est expulsé du logement de la pompe par l'échappement.

Les pompes rotodynamiques ne sont pas à amorçage automatique. L'aspiration doit déjà contenir de l'eau afin de pouvoir pomper. Contrairement aux pompes péristaltiques, l'eau circule aussi à l'intérieur de la pompe, même lorsqu'elle ne pompe pas. Ces pompes peuvent être utilisées dans les mares ou les aquariums et elles peuvent être entièrement submergées.

Contrairement aux pompes péristaltiques, ces pompes ne peuvent pas être inversées. Certains modèles utilisent un moteur CC sans balais doté de composants électroniques enfermés dans un même boîtier pour une puissance maximale et un encombrement réduit.

Projet : système d'arrosage Arduino pour plantes d'intérieur

Ce projet Arduino très simple (figure 7-16) utilise une pompe péristaltique pour fournir quotidiennement une quantité précise d'eau à vos plantes d'intérieur (ce qui est très utile lorsque vous partez en vacances).

Figure 7-16. *Système d'arrosage pour plantes d'intérieur*

Ce projet n'utilise pas de minuterie pour définir à quel moment arroser la plante ; à la place, il mesure l'intensité lumineuse de façon à ce que plante soit arrosée dès qu'il fait nuit.

Montage

La figure 7-17 présente le schéma du circuit du projet.

Un transistor MPSA14 est utilisé sur l'Arduino pour démarrer et arrêter le moteur de la pompe. La diode D1 assure une protection contre les pics de tension négatifs.

Sur la gauche du schéma, vous pouvez voir une photorésistance et une résistance à valeur fixe qui forment un diviseur de tension pour mesurer l'intensité lumineuse sur la broche analogique A0 de la carte Arduino.

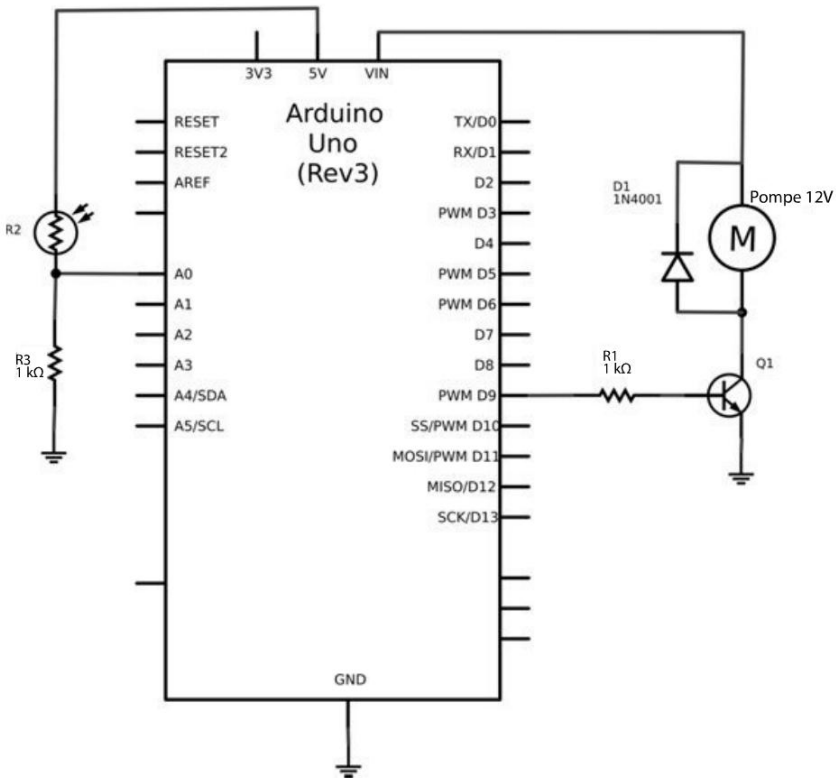


Figure 7-17. *Schéma du circuit du système d'arrosage pour plantes d'intérieur*

Plus la quantité de lumière qui atteint la photorésistance est importante, plus la résistance est faible, ce qui fait monter la tension sur la broche A0 jusqu'à 5 V.

Ce projet est assez facile à réaliser. Comme il est peu probable que des conducteurs soient déjà prévus sur la pompe, vous devrez souder quelques fils au moteur.

Composants nécessaires

Les composants suivants sont nécessaires pour la réalisation du projet.

Nom	Composant	Source
	Arduino Uno	
Q1	Transistor Darlington MPSA14	Mouser : 833-MPSA14-AP
R1, R3	Résistance 1 kΩ	Mouser : 291-1k-RC
R2	Photorésistance (1 kΩ)	Adafruit : 161 Sparkfun : SEN-09088

D1	Diode 1N4001	Adafruit : 755 Sparkfun : COM-08589 Mouser : 512-1N4001
	Pompe péristaltique 12 V	eBay
	Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
	Câbles flexibles mâles/mâles	Adafruit : 758
	1 m de tube d'un diamètre adapté à la pompe	Magasin de bricolage
	Alimentation électrique 12 V 1 A	Adafruit : 798
	Grand réservoir d'eau	
	Câbles de connexion à souder sur le moteur de la pompe	Adafruit : 1311

La pompe péristaltique bon marché utilisée dans ce projet a été conçue pour une utilisation dans un aquarium.

Le tube utilisé dans ce projet fait partie d'un kit d'arrosage acheté dans un magasin de bricolage. L'ensemble comprenait également de petits raccords en plastique pour assembler les longueurs de tube. Ces raccords m'ont permis de fixer le tube de la pompe aux longueurs de tubes supplémentaires. Pour que la pompe fonctionne correctement, les tubes doivent être étanches.

Construction

La construction de ce projet nécessite quelques montages simples sur la plaque d'essai et un peu de bricolage pour adapter le réservoir d'eau.

Étape 1 : soudage des liaisons sur le moteur

Soudez si nécessaire les liaisons sur les bornes de la pompe. Les câbles doivent être suffisamment longs pour relier la pompe à la plaque d'essai et l'Arduino. Une longueur de 50 cm devrait suffire.

Étape 2 : réalisation du circuit sur la plaque d'essai

Reportez-vous au circuit illustré à figure 7-18 pour monter les composants sur la plaque d'essai.

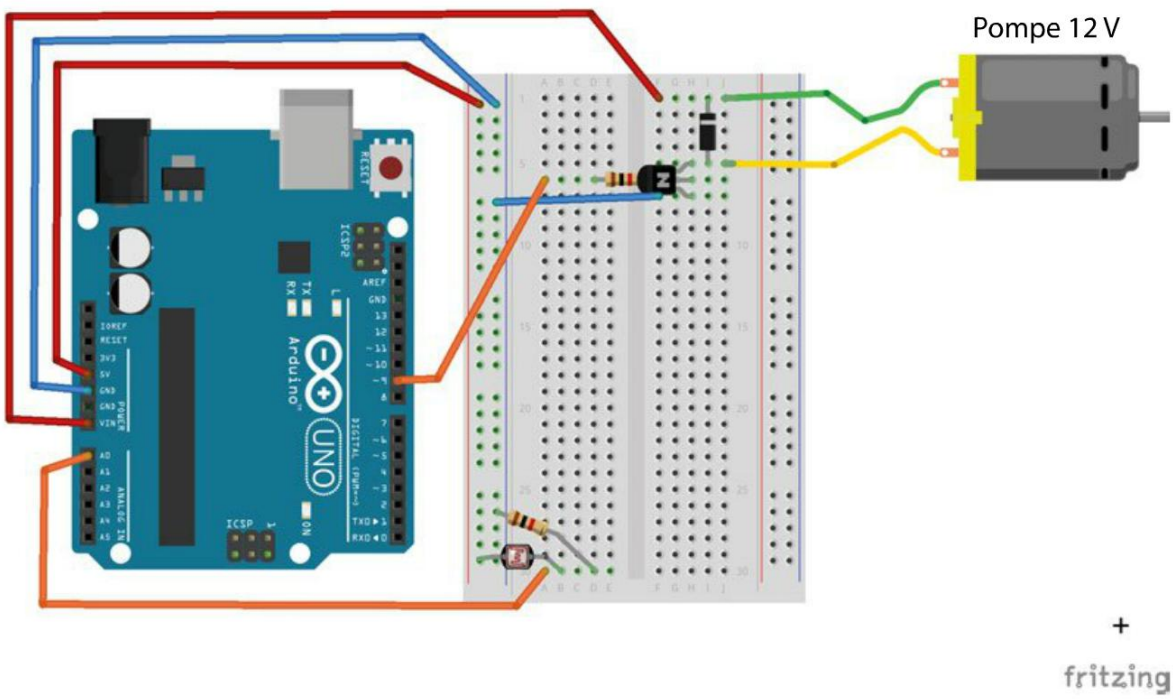


Figure 7-18. *Réalisation du circuit du système d'arrosage pour plantes d'intérieur*

Vérifiez que le transistor et la diode ont été montés dans le bon sens.

Étape 3 : fixation du tube sur la pompe

Vous avez besoin de deux longueurs de tube. L'un aura approximativement la longueur du réservoir d'eau. Il sera fixé sur l'aspiration de la pompe et descendra dans le réservoir. L'autre ira de l'orifice d'échappement de la pompe jusqu'à la plante. La figure 7-19 montre le raccordement des tubes sur la pompe.

L'aspiration et l'échappement de la pompe ne sont généralement pas identifiés, mais les pompes péristaltiques sont réversibles. Donc si vous constatez qu'elle aspire au lieu d'expulser, il sera probablement plus facile d'intervertir le câblage du moteur que d'échanger les tuyaux.

Figure 7-13. *Raccordement des tubes à la pompe*

Étape 4 : assemblage final

J'ai fixé la pompe en haut du réservoir d'eau, tête en bas, avec le moteur vers le haut. Le tube d'aspiration descend directement dans le réservoir et le tube d'échappement sort sur le côté et va jusqu'à la plante. Il m'a fallu couper le goulot de la bouteille de lait.

La figure 7-16 présente l'assemblage final. Si vous le souhaitez, vous pouvez placer la pompe au même niveau que la plaque d'essai, mais il vous faudra un tube plus long.

Programme

Vous trouverez le sketch Arduino de ce projet dans le fichier `arduino/projects/waterer/pr_01_waterer.ino` à l'endroit où vous avez téléchargé le code du livre (vous trouverez aussi un second sketch intitulé `waterer_test` dans le dossier `projects/ folder` ; je m'en suis servi pour étalonner la photorésistance) :

```
const int motorPin = 9;          ❶
const int lightPin = A0;

const long onTime = 60 * 1000; // 60 secondes  ❷
const int dayThreshold = 200;                ❸
const int nightThreshold = 70;

boolean isDay = true;                        ❹

void setup() {
  pinMode(motorPin, OUTPUT);
}
```

```
void loop() {                    ❺
  int lightReading = analogRead(lightPin);
  if (isDay && lightReading < nightThreshold) { //la nuit tombe ❻
    pump();
    isDay = false;
  }
  if (!isDay && lightReading > dayThreshold) {    ❼
    isDay = true;
  }
}

void pump() {                    ❽
  digitalWrite(motorPin, HIGH);
  delay(onTime);
  digitalWrite(motorPin, LOW);
}
```

❶ Le sketch commence par définir des constantes pour les deux broches Arduino utilisées : la broche de commande du moteur et l'entrée analogique (`lightPin`) qui se sert de la photorésistance pour mesurer l'intensité lumineuse.

❷ La constante `onTime` détermine la durée de fonctionnement de la pompe chaque soir. Réglez une période courte pendant la phase de test, de l'ordre de 10 secondes pour limiter le temps d'attente.

La partie la plus intéressante de ce sketch est celle qui détecte la tombée de la nuit. Comme la carte Arduino n'a pas d'horloge intégrée, elle ne connaît pas l'heure à moins que vous n'y ajoutiez une horloge temps réel (HTR ou RTC pour *Real Time Clock*). Dans ce projet, nous voulons que la plante soit arrosée une fois par jour, donc la tombée de la nuit déclenchera le début de l'arrosage. Une fois la plante arrosée, vous ne voulez pas que le prochain arrosage soit déclenché avant la nuit prochaine, dont une période intermédiaire de luminosité (donc de jour) est nécessaire.

❸ Pour aider à faire la différence entre la nuit et le jour, deux constantes sont définies : `dayThreshold` et `nightThreshold`. Vous devrez probablement corriger ces valeurs en fonction de l'emplacement de la plante et de la sensibilité de la photorésistance. Le principe est le suivant : si la luminosité est supérieure à `dayThreshold`, cela signifie qu'il fait jour ; si elle est inférieure à `nightThreshold`, alors il fait nuit. Peut-être vous demandez-vous pourquoi il y a deux constantes au lieu d'une seule. La réponse est qu'au crépuscule, lorsque la nuit commence à tomber, le niveau de luminosité peut être supérieur ou inférieur au seuil pendant une certaine période, ce qui risquerait de déclencher l'arrosage à plusieurs reprises.

❹ La variable booléenne `isDay` correspond à l'état courant de la journée. Si `isDay` est `true`, alors le système d'arrosage considère qu'il fait jour.

❺ La logique déterminant la nécessité d'un arrosage se trouve dans la fonction `loop`. Cela nécessite d'abord une mesure de la luminosité.

❻ S'il fait jour actuellement, mais que la luminosité mesurée est inférieure à `nightThreshold`, cela signifie que la nuit vient de tomber, donc la fonction `pump` est activée pour effectuer un arrosage. La variable `isDay` est alors définie sur `false` pour indiquer qu'il fait nuit et pour éviter que l'arrosage ne se poursuive.

❼ La deuxième instruction `if` dans `loop` vérifie s'il fait jour (`!isDay`) et si le niveau de luminosité est maintenant supérieur à `dayThreshold`. Si ces deux conditions sont remplies, alors `isDay` est défini sur `true`.

❽ Enfin, la fonction `pump` a pour rôle d'allumer la pompe, d'attendre pendant la durée définie dans `onTime`, puis d'éteindre la pompe.

Application du projet

Avant d'exécuter le projet à proprement dit, chargez le programme `waterer_test.ino` sur la carte Arduino, afin de définir des valeurs adaptées pour `dayThreshold` et `nightThreshold`. Téléversez ce sketch et ouvrez le Moniteur série.

Une nouvelle série de valeurs s'affichent toutes les demi-secondes dans le Moniteur série. Elles correspondent au niveau de luminosité actuel. Notez les valeurs mesurées pendant la journée par temps couvert. Divisez cette valeur par deux pour `dayThreshold` pour tenir compte des jours gris.

Ensuite, attendez qu'il fasse le plus nuit possible à l'emplacement de la plante et notez la valeur mesurée. Vous pouvez aussi poser le doigt sur le capteur ou estimer la valeur. Définissez cette valeur pour `nightThreshold`. N'oubliez pas que `nightThreshold` doit être nettement inférieur à `dayThreshold`. Ajustez au besoin vos estimations de ces deux valeurs.

Vous pouvez maintenant modifier `dayThreshold` et `nightThreshold` dans le sketch (`pr_01_waterer.ino`), puis chargez le code sur l'Arduino.

Vous pouvez imiter la tombée de la nuit en posant le doigt sur la photorésistance. La pompe devrait alors fonctionner pendant la durée définie.

La pompe que j'ai utilisée a un débit d'environ 90 ml/min. Pour définir la durée d'arrosage, servez-vous d'un verre mesureur et d'un chronomètre pour évaluer le débit volumique de votre pompe et ajustez `onTime` pour fournir la quantité d'eau nécessaire à la plante.

Vérins électriques

Les vérins électriques convertissent la rotation d'un moteur CC en mouvement linéaire. Ils sont souvent utilisés pour ouvrir et fermer des portes ou des fenêtres.

Ils se composent d'une tige filetée et d'un écrou qui ne tourne pas, mais qui peut monter ou descendre le long de l'axe fileté en faisant entrer ou sortir l'extrémité de l'arbre. La figure 7-20 en illustre le fonctionnement ; la figure 7-21 représente un vérin électrique.

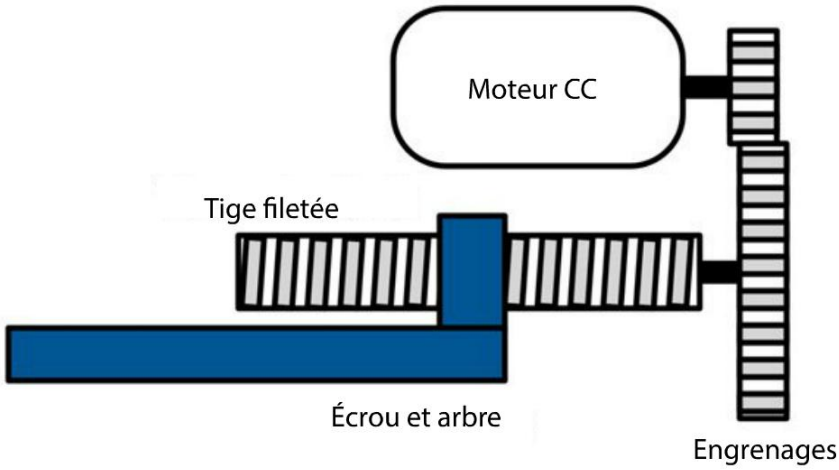


Figure 7-20. *Fonctionnement d'un vérin électrique*

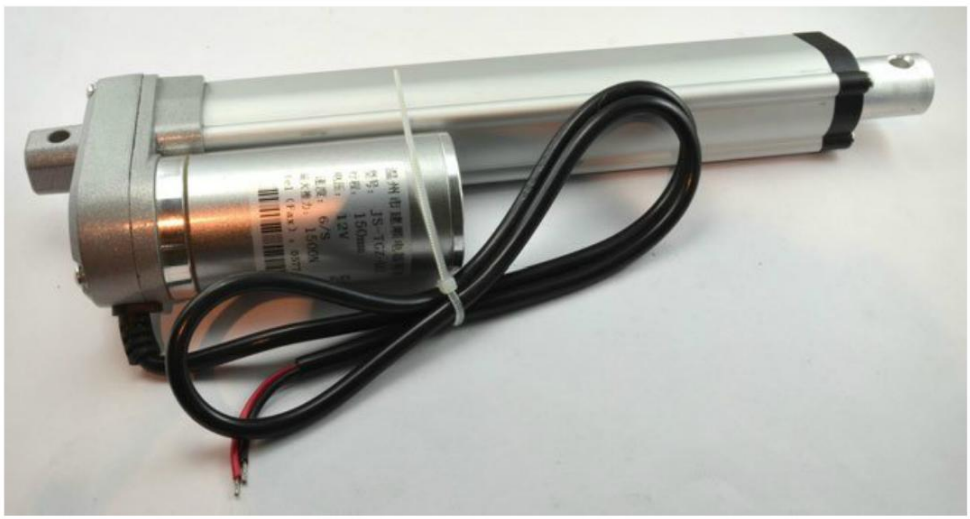


Figure 7-21. *Un vérin électrique*

Les vérins électriques déplacent l'arbre assez lentement, car le long axe fileté et l'écrou constituent en fait un réducteur à engrenages. En outre, une boîte à engrenages se trouve aussi à l'extrémité du moteur. Du fait de cette vitesse réduite et de la puissance du moteur CC, le vérin électrique peut produire une force de poussée ou de traction assez élevée. Le modèle illustré à la figure 7-21 a une force de 1 500 N (newtons), ce qui permet de soulever un poids de 150 kg. À pleine charge, le moteur d'un vérin électrique comme celui-ci peut nécessiter 5 A sous 12 V.

Le moteur d'un vérin électrique doit généralement être régulé à l'aide d'un pont en H (courant maxi de 5 A au moins) pour pouvoir être actionné dans les deux sens. Pour éviter d'endommager le vérin lorsqu'il arrive en fin de course, le dispositif intègre habituellement un contact de fin de

course qui coupe automatiquement l'alimentation quand l'arbre atteint une butée. Cela facilite la régulation du moteur, car vous avez simplement besoin de demander au pont en H d'alimenter le moteur dans un sens ou dans l'autre pendant une période donnée qui est suffisamment longue pour que le vérin termine sa course.

Dans le projet « Compacteur de canettes Arduino » au chapitre 8, nous utilisons un vérin électrique tel que celui illustré à la figure 7-21 pour fabriquer un compacteur de canettes.

Solénoïdes

Les solénoïdes sont utilisés pour les loquets de portes et les valves. Comme les vérins électriques, ils produisent un mouvement linéaire, mais ces dispositifs sont beaucoup plus simples. Il s'agit en fait d'électroaimants qui font entrer ou sortir une armature. La course très courte ne dépasse pas quelques centimètres. La figure 7-22 illustre le fonctionnement d'un solénoïde ; la figure 7-23 montre une valve 12 V utilisant un solénoïde.

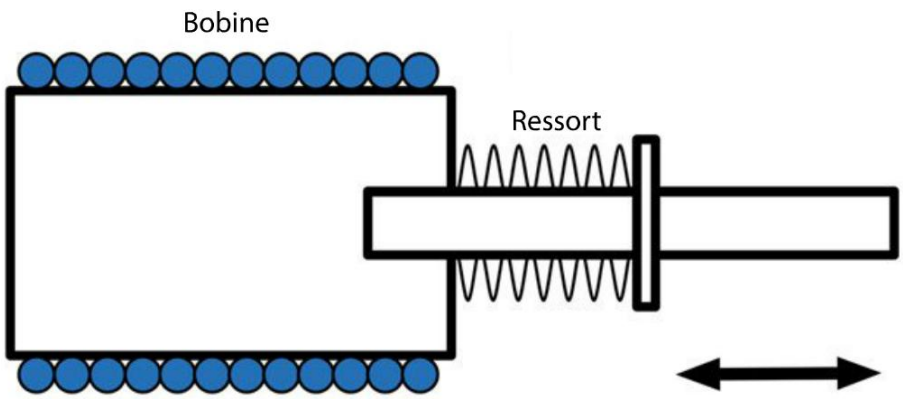


Figure 7-22. *Fonctionnement d'un solénoïde*

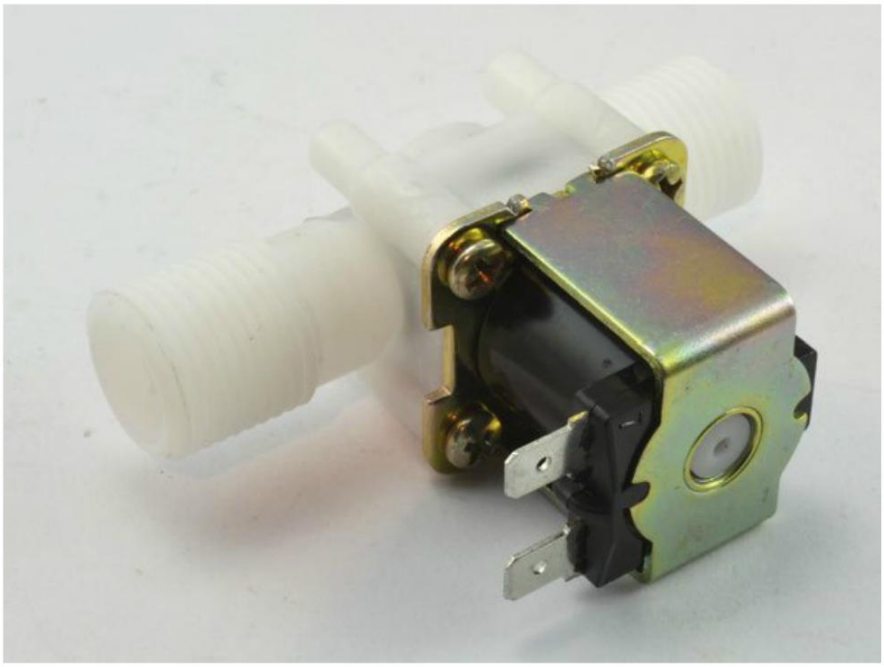


Figure 7-23. *Une valve 12 V*

Quand la bobine est alimentée, elle tire le plongeur dans la bobine en s'opposant à la force du ressort. Quand l'alimentation de la bobine est coupée, le plongeur peut reprendre sa position initiale.

La valve illustrée à la figure 7-23 permet de couper une alimentation en eau sous pression. Quand la valve n'est pas alimentée, l'eau ne s'écoule pas. Quand la bobine est alimentée, un plongeur se rétracte, ce qui permet à l'eau de passer par la valve tant que celle-ci est alimentée.

L'élément illustré est un composant 12 V du type de ceux qui équipent les machines à laver. On en trouve aussi des versions 120 V ou 220 V en CA (courant alternatif) dans les appareils ménagers. Pour vos projets, je vous conseille vivement de vous en tenir aux modèles fonctionnant en 12 V.

Vous trouverez aussi des loquets de portes actionnés par un solénoïde.

Résumé

Dans ce chapitre, vous vous êtes familiarisé avec le fonctionnement des moteurs CC. Vous avez appris à éteindre et à démarrer un moteur à l'aide d'une carte Arduino ou d'un Raspberry Pi, ainsi qu'à en réguler la vitesse.

Dans le chapitre suivant, nous verrons comment utiliser un pont en H pour piloter le sens de rotation d'un moteur.

Régulation avancée d'un moteur

Dans le chapitre précédent, vous avez appris à réguler la vitesse d'un moteur, mais pas son sens de rotation. Dans ce chapitre, nous examinerons différentes possibilités de contrôle du sens d'un moteur. Nous étudierons notamment quelques circuits intégrés et des modules spécialement conçus pour faciliter le contrôle de la direction, ainsi que de la vitesse des moteurs CC.

Il est souvent pratique de pouvoir inverser le sens de rotation d'un moteur. Les vérins, par exemple, qui ouvrent et ferment des fenêtres ou des portes actionnent un moteur CC dans un sens pour l'ouvrir, puis dans l'autre sens pour la fermer. De même, si vous fabriquez un petit robot, vous voudrez probablement que ses roues tournent dans les deux sens.

Imaginez un moteur avec deux conducteurs, A et B (figure 8-1).

Si A est positif et B négatif, le moteur tournera dans un sens. Si vous inversez la polarité des connexions, le moteur tournera en sens inverse.

Par conséquent, si vous voulez contrôler le sens de rotation d'un moteur, vous devez trouver le moyen d'inverser la polarité du courant électrique. Cela s'effectue à l'aide d'un circuit qui se nomme un pont en H.

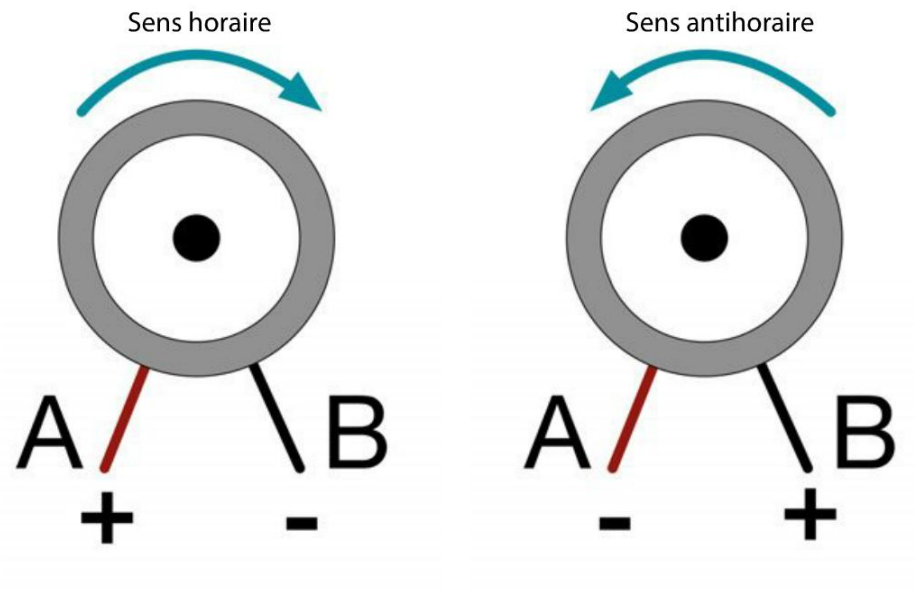


Figure 8-1. *Contrôle du sens d'un moteur*

Ponts en H

La figure 8-1 illustre le fonctionnement d'un pont en H. Nous utiliserons d'abord des commutateurs avant de passer aux transistors ou aux circuits imprimés.

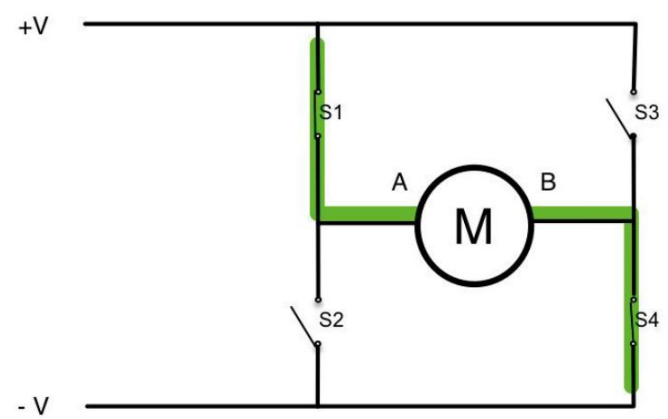


Figure 8-2. *Un pont en H utilisant des commutateurs*

Si les quatre commutateurs sont ouverts, le courant ne circule pas jusqu'au moteur. Cependant, si S1 et S4 sont fermés, mais S2 et S3 sont ouverts (figure 8-2), le courant circulera depuis l'alimentation positive jusqu'à la borne A du moteur, il traversera le moteur et S4 jusqu'à l'alimentation négative. Le moteur tournera dans un sens.

Si S1 et S4 sont ouverts et S3 et S2 fermés, l'alimentation positive sera appliquée à la borne B du moteur. Le courant traversera le moteur et S2 en sens inverse.

Le tableau 8-1 résume le comportement du moteur ; 0 signifie que le commutateur est ouvert, 1 signifie que le commutateur est fermé (conducteur) et X signifie que l'état n'a pas d'importance.

Tableau 8-1. *Combinaisons de commutateurs*

S1	S2	S3	S4	Moteur
1	0	0	1	Tourne dans le sens horaire
0	1	1	1	Tourne dans le sens antihoraire
0	0	0	0	Moteur arrêté
1	1	X	X	COURT-CIRCUIT
X	X	1	1	COURT-CIRCUIT
1	0	1	0	Freinage
0	1	0	1	Freinage

Nous avons vu comment utiliser le pont en H pour changer le sens de rotation du moteur. Toutefois, il y a d'autres combinaisons possibles pour la position des commutateurs.

D'abord, et cela paraît évident, si tous les commutateurs sont ouverts, l'électricité ne parviendra pas jusqu'au moteur et il s'arrêtera rapidement.

Par ailleurs, certaines combinaisons de commutateurs connectent directement l'alimentation positive à l'alimentation négative. Cela provoque un court-circuit qui peut avoir des conséquences désastreuses, car un courant fort circule.

Il existe une autre situation qui ne provoque pas de court-circuit, bien que les broches du moteur soient effectivement connectées ensemble. Cela a pour effet intéressant de faire freiner le moteur, en le faisant ralentir plus vite s'il est en mouvement, ou en le poussant à résister, s'il est déjà à l'arrêt. Si le moteur actionne les roues d'une petite voiture, par exemple, l'utilisation de ce mode de freinage empêche la voiture d'avancer lorsqu'elle est arrêtée sur une route en pente.

Pont en H sur un circuit intégré

Le L293D est un circuit intégré de pont en H simple d'emploi très apprécié des amateurs. Vous vous en servirez dans l'expérience « Commande du sens et de la vitesse d'un moteur », plus loin. Ce composant convient plus particulièrement aux petits moteurs ayant un courant maximum de 600 mA et une tension de 36 V. Vous trouverez toutes les caractéristiques de ce circuit intégré sur ses spécifications techniques.

Le L293D contient deux ponts en H, ainsi des composants supplémentaires qui coupent automatiquement le circuit en cas de surchauffe.

Les principales caractéristiques de ce circuit sont les suivantes.

- Plage de tension du moteur comprise entre 4,5 V et 36 V
- Intensité du moteur de 600 mA en permanence
- Pointe de courant de 1,2 A
- Diodes sur toutes les sorties pour assurer la protection contre les surtensions du moteur
- Protection thermique
- Compatible avec logique 3 V et 5 V (Pi et Arduino)

La figure 8-3 montre le schéma de ce circuit et son utilisation pour commander deux moteurs CC Le circuit intégré se compose en fait de quatre demi-ponts en H au lieu de deux ponts en H entiers. Chaque demi-pont en H est comparable à une sortie numérique de forte puissance capable de fournir ou d'absorber des courants jusqu'à 600 mA. Cela offre davantage de flexibilité pour l'utilisation du circuit intégré.

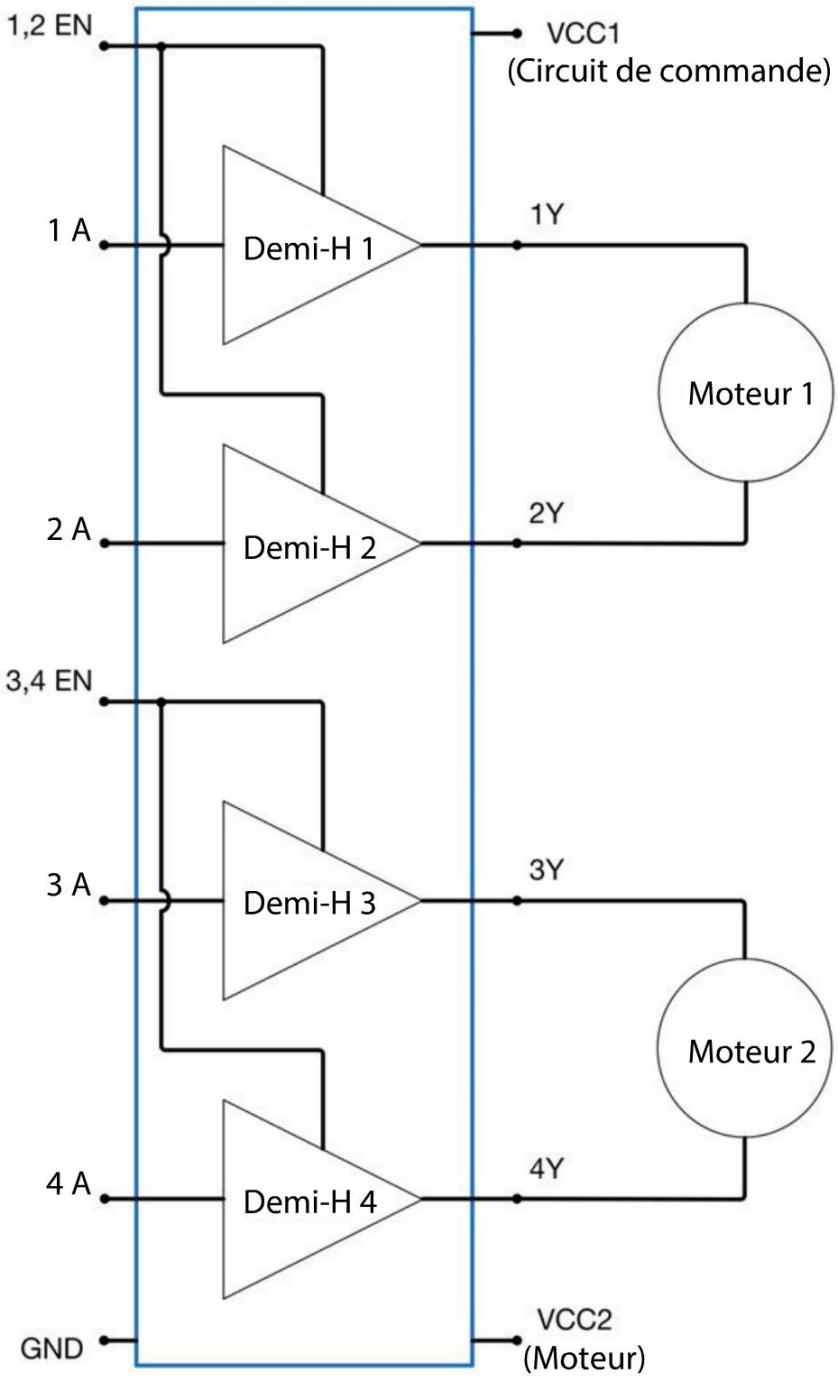


Figure 8-3. *Brochage du L293D*

Ce circuit intégré a des broches d'alimentation séparées pour les circuits de commande et de puissance du moteur, ce qui permet, par exemple, de piloter un moteur 6 V à partir du circuit de commande en 3,3 V d'un Raspberry Pi (voir l'expérience « Commande du sens et de la vitesse d'un moteur », ci-après).

La fonction de chacune des broches utilisées à la figure 8-3 est décrite dans le tableau 8-2.

Tableau 8-2. *Brochage du L293D*

Numéro de broche	Nom de la broche	Description
1	1,2EN	Cette broche active les sorties des demi-ponts 1 et 2 (sauf si cette broche est au niveau haut, ces sorties ne feront rien) ; elle est souvent utilisée avec un signal PWM pour réguler la vitesse globale du moteur.
2	1A	Commande d'entrée du demi-pont 1 ; si elle est au niveau haut, la sortie sur la broche 3 sera au niveau haut (transistor du haut actif).
3	1Y	Sortie du demi-pont 1.
4,5, 12, 13	GND	Masse (sur un circuit imprimé, toutes ces broches sont soudées sur un grand dissipateur thermique).
6	2Y	Sortie du demi-pont 2.
7	2A	Commande d'entrée du demi-pont 2.
8	VCC2	Alimentation électrique des moteurs, jusqu'à 36 V ; la séparation du circuit de puissance et du circuit de commande préserve la stabilité.
9	3,4EN	Active les demi-ponts 3 et 4.
10	3A	Commande d'entrée du demi-pont 3.
11	3Y	Sortie du demi-pont 3.
14	4Y	Sortie du demi-pont 4.
15	4A	Commande d'entrée du demi-pont 4.
16	VCC1	Alimentation électrique du circuit de commande ; elle peut être plus basse que l'alimentation électrique du moteur sur la broche 8 et elle est généralement de 5 V.

Vous utiliserez ce circuit imprimé dans l'expérience « Commande du sens et de la vitesse d'un moteur » pour piloter à la fois la vitesse et le sens d'un moteur CC.

Expérience : commande du sens et de la vitesse d'un moteur

Dans cette expérience, vous utiliserez un circuit intégré L293D monté sur une plaque d'essai. La figure 8-4 montre la plaque d'essai connectée à un Raspberry Pi.

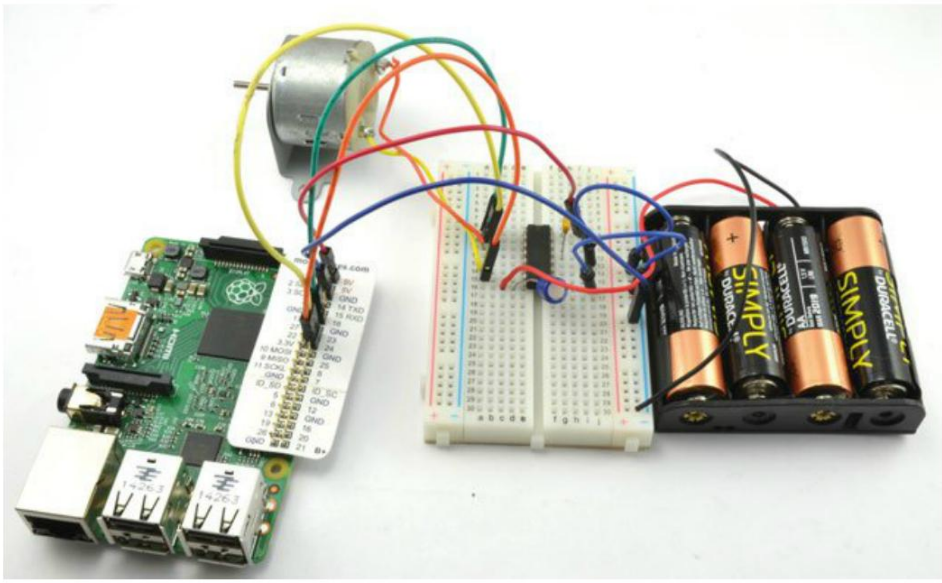


Figure 8-4. *Commande du sens et de la vitesse d'un moteur avec Raspberry Pi*

Les composants de ce projet sont identiques pour le Raspberry Pi et l'Arduino et vous pourrez tester le matériel avant de le connecter à votre carte.

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou un Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser l'expérience.

Nom	Composant	Source
IC1	Circuit intégré de pont en H L293D	Adafruit : 807 Mouser : 511-L293D
C1	Condensateur 100 nF	Adafruit : 753 Mouser : 810-FK16X7R2A224K
C2	Condensateur 16 V 100 µF	Adafruit : 2193 Sparkfun : COM-00096 Mouser : 647-UST1C101MDD
M1	Petit moteur 6 V CC	Adafruit : 711
	Support pour piles (4 × AA) 6 V	Adafruit : 830
	Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
	Câbles flexibles mâles/mâles	Adafruit : 758
	Câbles flexibles femelles/mâles (Pi uniquement)	Adafruit : 826

Pour ce projet, nous nous passerons des condensateurs C1 et C2. Mais, tôt ou tard, vous devrez prendre l'habitude d'utiliser des condensateurs, surtout si vous comptez dépasser le stade de l'expérimentation.

Si vous voulez mener cette expérience avec un Raspberry Pi, vous aurez besoin de câbles flexibles femelles/mâles pour connecter les broches GPIO du Raspberry Pi à la plaque d'essai.

Montage

La figure 8-5 présente le schéma du circuit du projet.

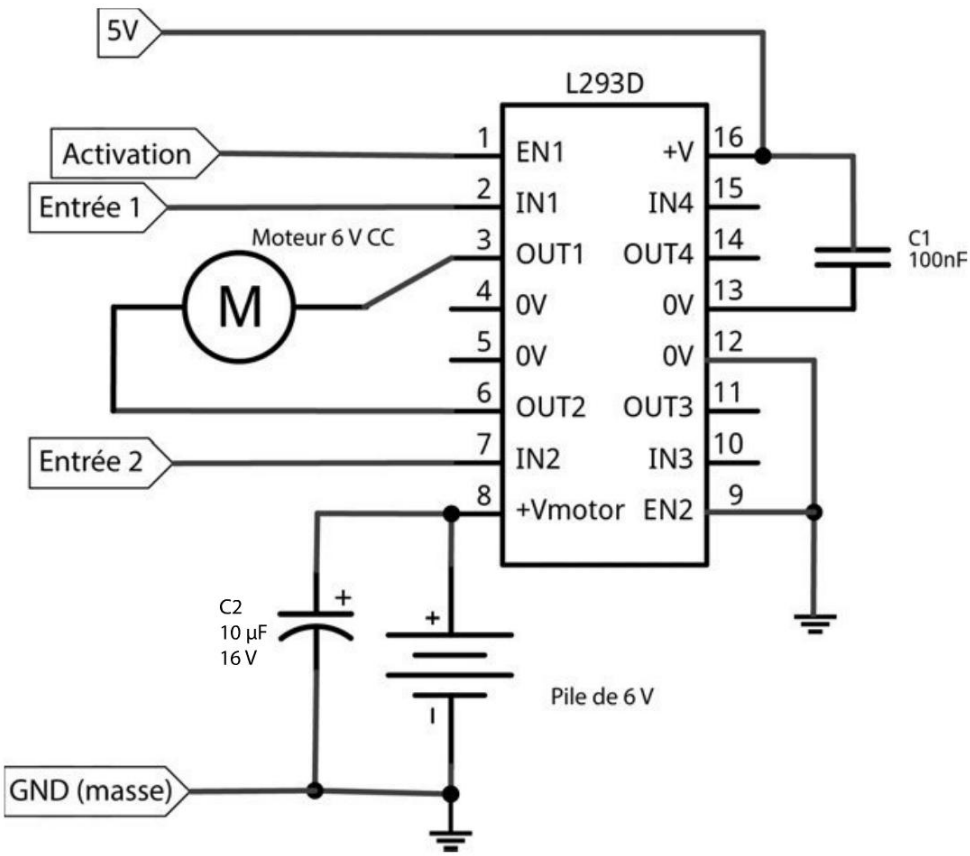


Figure 8-5. *Le schéma du circuit d'un pont en H L293D*

Le Raspberry Pi ou l'Arduino transmet l'alimentation du circuit de commande en 5 V sur la broche 16 et l'alimentation de puissance du moteur est transmise sur la broche 8 par des piles 6 V.

Un seul des ponts en H du circuit intégré est utilisé. La broche EN2 est donc reliée à la masse pour désactiver la moitié inutilisée du circuit.

Les broches EN1, IN1 et IN2 sont toutes connectées aux broches de sorties numériques du Raspberry Pi ou de l'Arduino.

Condensateurs

Les condensateurs sont facultatifs parce que si vous ne faites que tester ce circuit pendant quelques heures, la fiabilité conférée par ces composants n'est pas problématique.

Cette disposition de condensateurs est typique d'un circuit intégré à pont en H. C1 est un condensateur dit de découplage. Il doit être placé aussi près que possible du circuit et entre l'alimentation de commande et GND. Il peut se limiter à 100 nF (faible capacité) et il élimine les bruits électriques qui risquent d'interférer avec le bon fonctionnement du circuit de commande.

C2 fournit une réserve d'énergie à court terme, davantage destinée aux moteurs qu'au circuit de commande. La valeur de ce condensateur, qui est généralement de 100 µF ou plus, est souvent supérieure à celle de C1.

Réalisation du circuit

Avant de connecter un pont en H sur un Arduino ou un Raspberry Pi, vous pouvez le tester seul avec l'alimentation de puissance du moteur et du circuit de commande à partir du même support de batterie 6 V. Ce montage convient pour essayer le composant isolément, mais lorsque vous l'utiliserez avec un Arduino ou un Raspberry Pi, vous séparerez les alimentations : le moteur sera alimenté par le support de batterie et l'alimentation du circuit de commande sera fournie par l'Arduino ou le Raspberry Pi.

La figure 8-6 présente le circuit d'essai du pont en H. Seuls certains câbles flexibles changent lorsque vous connectez le composant à un Arduino ou à un Raspberry Pi.

Lorsque vous connectez la plaque d'essai, faites particulièrement attention au circuit intégré. Veillez à le placer correctement – la petite encoche d'un côté du circuit doit être orientée vers le haut de la plaque d'essai sur la rangée 10. La broche située en haut à gauche de l'encoche est la broche 1.

Vous pouvez aussi connecter la batterie. Au départ, le moteur ne doit pas tourner.

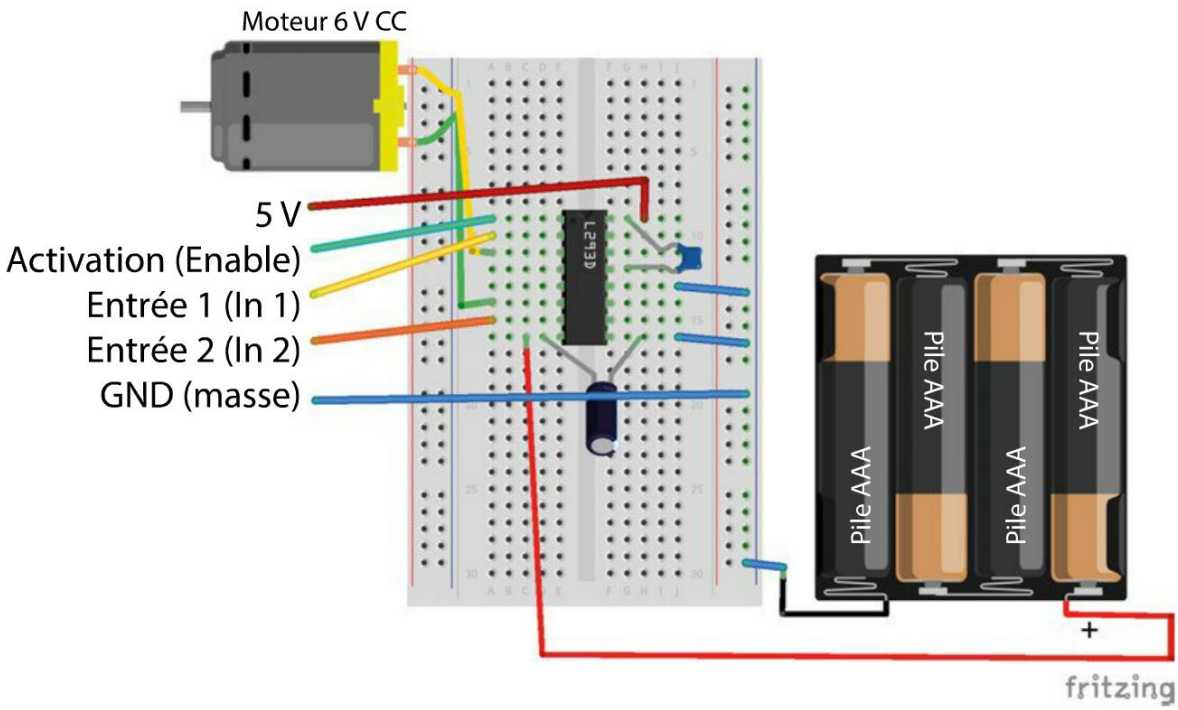


Figure 8-8. *Réalisation du circuit de test isolé du pont en H*

Sens horaire et antihoraire

Même si j'écris que le moteur tourne dans le sens horaire et antihoraire, cela dépend du moteur. Si votre moteur tourne dans le sens antihoraire alors que j'écris qu'il devrait tourner dans le sens horaire, cela ne pose pas de problème tant qu'il tourne dans le sens horaire quand j'écris qu'il devrait tourner dans le sens antihoraire.

Vous pouvez intervertir les conducteurs du moteur si vous préférez que le sens de rotation corresponde à la description.

Il n'est pas toujours facile de reconnaître le sens de rotation du moteur s'il ne comporte qu'un simple arbre métallique nu. Pincez délicatement l'arbre entre le pouce et l'index pour sentir dans quel sens il tourne.

Une autre technique consiste à coller un petit morceau de ruban adhésif sur l'axe.

Expérimentations

Quel que soit le sens de rotation du moteur, la broche Enable doit être connectée à +V. Pour que le moteur tourne dans un sens, connectez l'autre extrémité du câble flexible connecté à In 1 sur +V et connectez In 2 sur la colonne de masse, du côté droit de la plaque d'essai (figure 8-7). Notez qu'aux figures 8-7 et 8-8, les traits correspondants aux câbles flexibles In 1 et In 2 sont plus épais pour les différencier des autres.

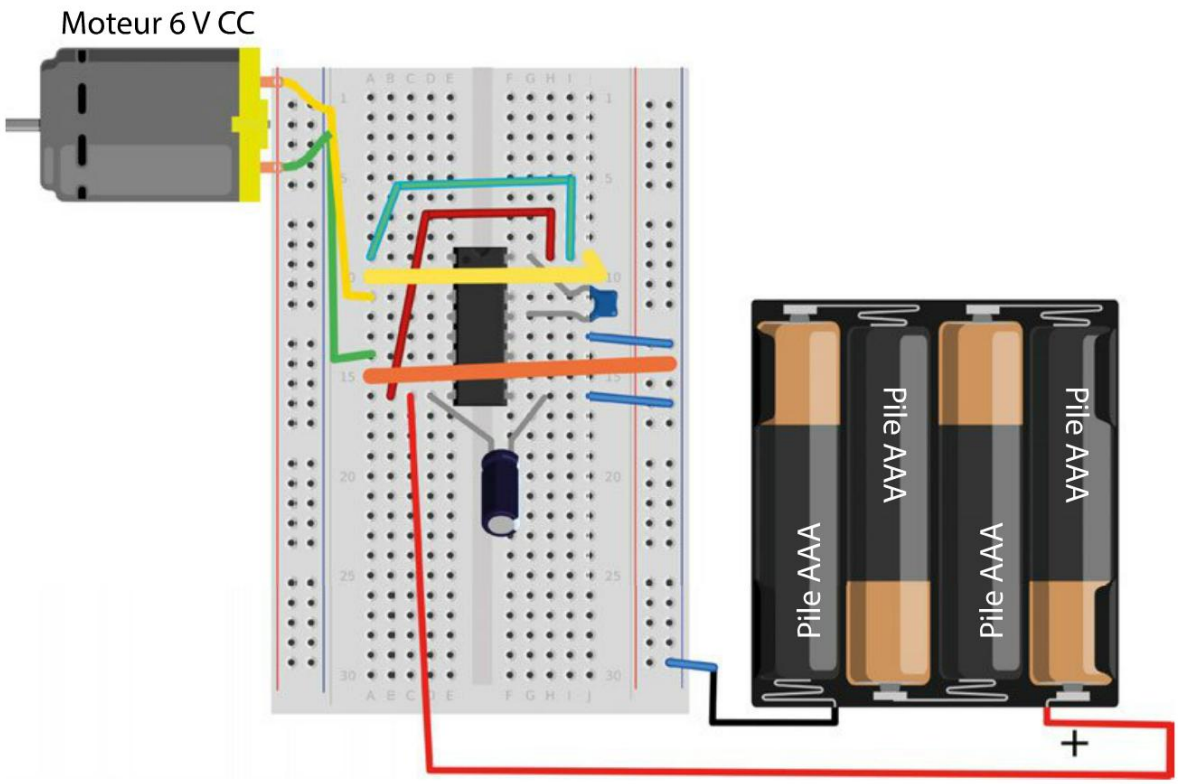


Figure 8-7. *Rotation dans le sens horaire du moteur*

Ensuite, inversez le sens de rotation du moteur. Il est inutile de changer la connexion à Enable, mais vous devez inverser In 1 et In 2 : In 1 est maintenant connecté à la masse et In 2 à +V, comme illustré à la figure 8-8.

Maintenant que nous avons vérifié le bon fonctionnement du pont en H, vous pouvez l'utiliser avec un Arduino (ou bien, si vous préférez utiliser le Raspberry Pi, passez directement à la section « Raccordement du Raspberry Pi » page 140).

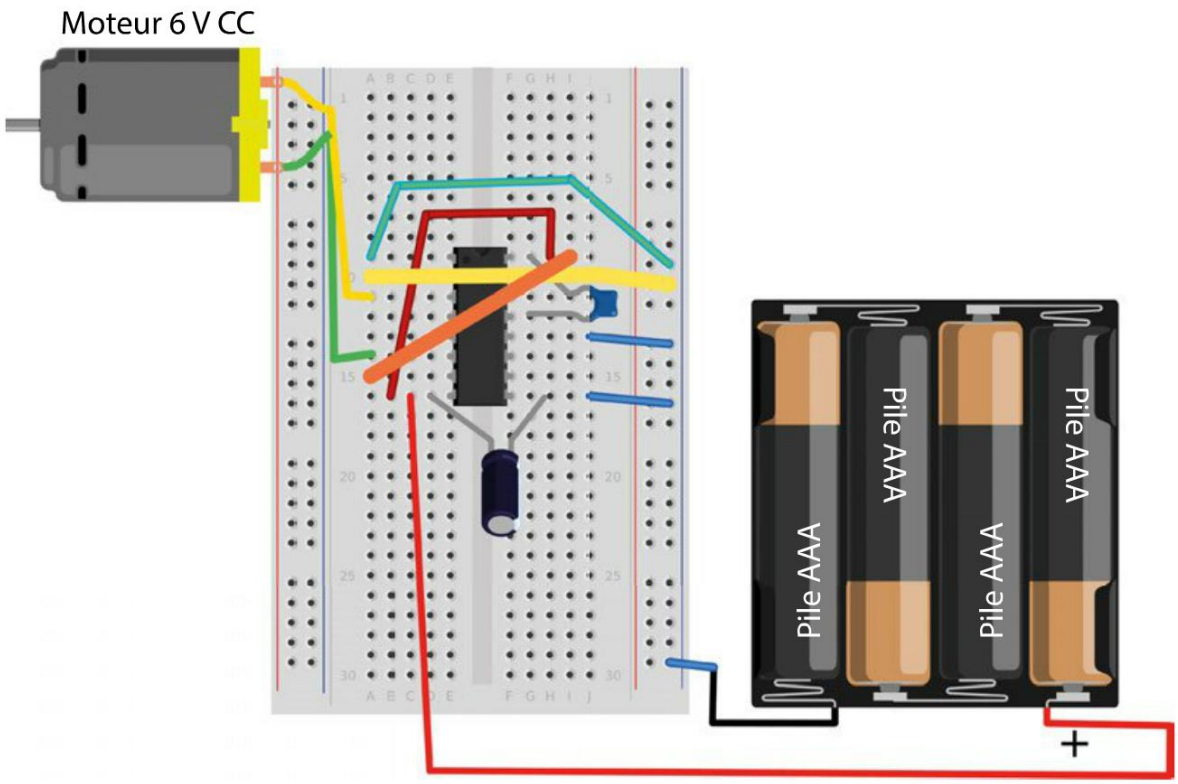


Figure 8-8. *Rotation dans le sens antihoraire du moteur*

Montage Arduino

La figure 8-9 montre comment connecter la plaque d'essai à une carte Arduino Uno à l'aide de câbles flexibles.

La broche Arduino 11 est utilisée pour la liaison Enable, car l'Arduino prend en charge la MLI et nous utiliserons la broche Enable du L293D pour piloter la vitesse du moteur. Vous pouvez connecter n'importe quelles broches numériques Arduino à In 1 et In 2 ; les broches 10 et 9 ont été choisies pour la simple raison qu'elles sont voisines de la broche 11 et que les câbles sont alignés.

L'Arduino alimente le circuit de commande du L293D en 5 V. Mais, surtout, il n'alimente pas le moteur. L'alimentation de puissance du moteur provient de la batterie.

La figure 8-10 montre la plaque d'essai connectée à la carte Arduino avec des câbles flexibles mâles/mâles.

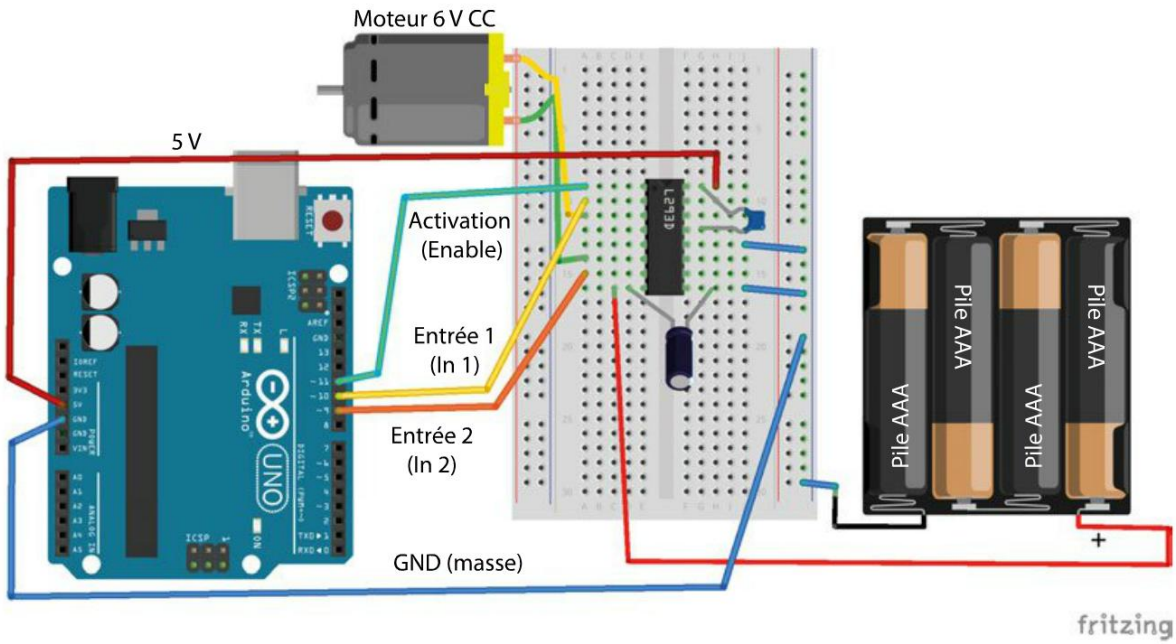


Figure 8-9. *Raccordement de l'Arduino et du pont en H*

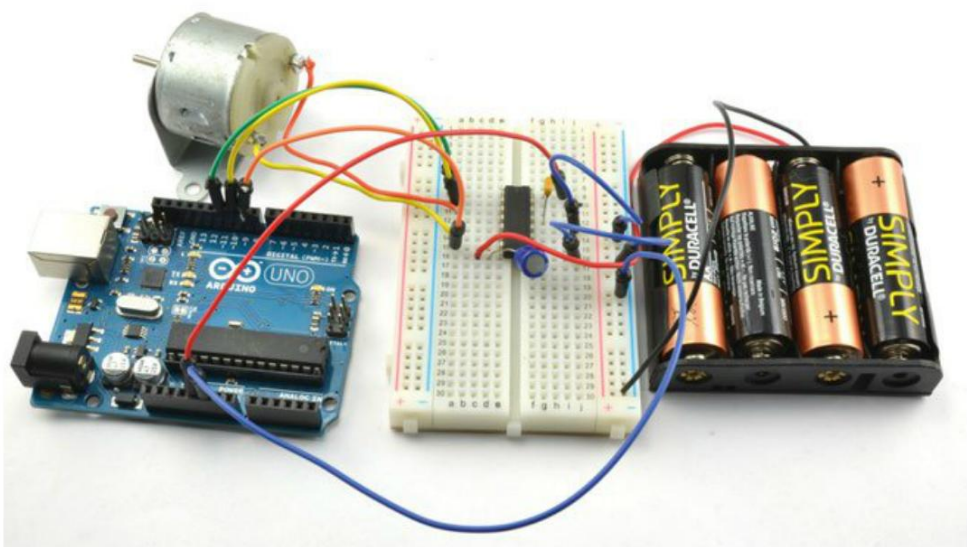


Figure 8-10. *Commande d'un moteur avec un Arduino*

Programme Arduino

Dans le programme Arduino de l'expérience « Régulation de la vitesse d'un moteur CC » au chapitre 7 (commande MLI d'un moteur), le moniteur série est utilisé pour envoyer des commandes de régulation de la vitesse au moteur. Dans cette expérience, le programme est complété afin de permettre également la transmission d'une commande de sens de rotation.

Vous trouverez le sketch Arduino de ce projet dans le dossier `/arduino/experiments/full_motor_control` à l'endroit où vous avez téléchargé le code du livre (voir la section « Le code du livre » du chapitre 2 page 14).

```
const int enablePin = 11;                    ❶
const int in1Pin = 10;
const int in2Pin = 9;

void setup() {                               ❷
  pinMode(enablePin, OUTPUT);
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  Serial.begin(9600);
  Serial.println(«Saisir s (stop) ou f ou r suivi du rapport cyclique (0 à 255). Par ex.
f120”);
}

void loop() {                                ❸
  if (Serial.available()) {
    char direction = Serial.read();          ❹
    if (direction == ‘s’) {                  ❺
      stop();                                ❻
      return;
    }
    int pwm = Serial.parseInt();             ❼
    if (direction == ‘f’) {                  ❽
      forward(pwm);
    }
    else if (direction == ‘r’) {
      reverse(pwm);
    }
  }
}

void forward(int pwm)                        ❾
{
    digitalWrite(in1Pin, HIGH);
    digitalWrite(in2Pin, LOW);
    analogWrite(enablePin, pwm);
    Serial.print(“Avant”);
    Serial.println(pwm);
}

void reverse(int pwm)                        ❿
{
```

```
  digitalWrite(in1Pin, LOW);
  digitalWrite(in2Pin, HIGH);
  analogWrite(enablePin, pwm);
  Serial.print("Arrière");
  Serial.println(pwm);
}

void stop()                          ⓫
{
digitalWrite(in1Pin, LOW);
digitalWrite(in2Pin, LOW);
analogWrite(enablePin, 0);
Serial.println("Stop");
}
```

Ce sketch est relativement long, mais il est structuré en fonctions qui forment une bonne base pour modifier le code ou le réutiliser dans vos projets.

❶ Le sketch commence par définir des constantes pour les trois broches de commande.

❷ La fonction `setup()` définit ces broches comme sorties, puis démarre la communication série à 9 600 bauds et transmet un message au moniteur série vous informant du format des données à transmettre pour piloter le moteur.

❸ La fonction `loop()` ne fait rien sauf si `Serial.available` rapporte l'arrivée d'un nouveau message transmis par le moniteur série.

❹ Le premier caractère de ce message est interprété comme une indication du sens de rotation : il peut s'agir de la lettre `s` (*stop*), `f` (*forward*, marche avant) ou `r` (*reverse*, marche arrière).

❺ Si la lettre de direction est un `s`, la fonction `stop` est appelée et la commande `return` empêche l'exécution de la suite du code contenu dans la fonction `loop`.

❻ La commande `s` ne nécessite pas de paramètre `pwm`, donc nous n'essayons pas de la lire dans le message puisqu'elle n'y figure pas.

❼ Si le code de sens de rotation est la lettre `f` ou `r` (au lieu de `s`), la valeur de `pwm` est prélevée dans le reste du message à l'aide de `parseInt`.

❽ L'une des fonctions `forward` ou `reverse` est ensuite appelée selon la lettre de direction.

❾ La fonction `forward` définit `in1Pin` sur `HIGH` et `in2Pin` sur `LOW` pour piloter le sens de rotation du moteur, puis utilise `analogWrite` pour réguler la vitesse du moteur à l'aide d'`enablePin` et du paramètre `pwm` défini sur `forward`. Enfin, un message de confirmation de l'action est renvoyé au moniteur série.

❿ La fonction `reverse` est presque identique à `forward` sauf qu'`in1Pin` est défini sur `LOW` et `in2Pin` sur `HIGH` pour faire tourner le moteur en sens inverse.

⓫ La fonction `stop` positionne toutes les broches de commandes à la valeur `LOW`.

Expérimentation Arduino

Pour réaliser l'expérience avec un Arduino, connectez le câble USB de l'Arduino et téléversez le sketch. Ouvrez le moniteur série (figure 8-11), saisissez la commande `f100` dans la partie supérieure du moniteur série, puis cliquez sur Envoyer. Le moteur doit commencer à tourner assez lentement en marche avant.

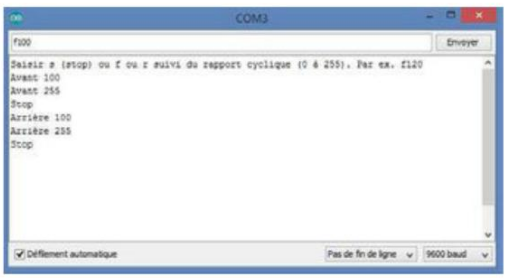


Figure 8-11. *Commande d'un moteur avec des commandes série*

Ensuite, saisissez la commande `f255`. Le moteur devrait tourner à sa vitesse maximale. La commande `s` arrête le moteur et `r100` le fait tourner assez lentement en marche arrière. Avec `r255`, le moteur tourne en marche arrière à sa vitesse maximale.

Raccordement du Raspberry Pi

Un avantage de l'utilisation d'un circuit intégré de pont en H, comme le L293D, est que les broches de commande ne nécessitent que très peu de courant pour piloter le moteur. D'après les spécifications techniques, ce courant est toujours inférieur à 100 µA (0,1 mA). Par conséquent, rien n'empêche d'utiliser les sorties en courant faible du Raspberry Pi.

Si vous avez à la fois un Raspberry Pi et un Arduino, et que vous venez de terminer la partie Arduino de cette expérience, il vous suffit de remplacer les câbles flexibles mâles/mâles utilisés pour raccorder la plaque d'essai à la carte Arduino par des câbles flexibles femelles/mâles pour pouvoir connecter la plaque d'essai aux broches GPIO du Raspberry Pi.

La figure 8-12 présente le raccordement du Raspberry Pi à la plaque d'essai. La figure 8-4 est une photographie de cette même configuration.

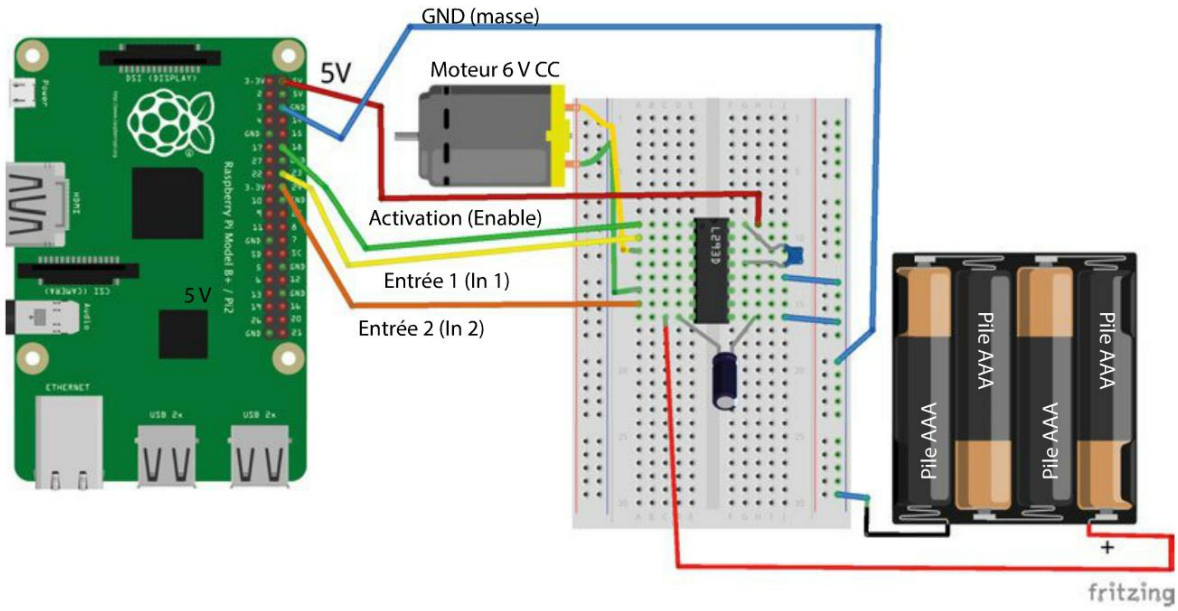


Figure 8-12. *Raccordement du Raspberry Pi et du pont en H*

Programme Raspberry Pi

Le code du programme suivant se trouve dans le fichier `full_motor_control.py` du dossier `python/experiments` à l'endroit où vous avez téléchargé le code du livre.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

enable_pin = 18                    ❶
in_1_pin = 23
in_2_pin = 24

GPIO.setup(enable_pin, GPIO.OUT)
GPIO.setup(in_1_pin, GPIO.OUT)
GPIO.setup(in_2_pin, GPIO.OUT)
motor_pwm = GPIO.PWM(enable_pin, 500)
motor_pwm.start(0)

def forward(duty):                 ❷
    GPIO.output(in_1_pin, True)
    GPIO.output(in_2_pin, False)
    motor_pwm.ChangeDutyCycle(duty)

def reverse(duty):                 ❸
    GPIO.output(in_1_pin, False)
```

```
    GPIO.output(in_2_pin, True)
    motor_pwm.ChangeDutyCycle(duty)

def stop():
    GPIO.output(in_1_pin, False)
    GPIO.output(in_2_pin, False)
    motor_pwm.ChangeDutyCycle(0)

try:
    while True:                    ❹
        direction = raw_input('Saisir la lettre de sens de rotation (f - forward/avant, r -
reverse/arrière, s - stop/arrêt): ')
        if direction[0] == 's':
            stop()
        else:
            duty = input('Saisir le rapport cyclique (0 à 100): ')
        if direction[0] == 'f':
                forward(duty)
            elif direction[0] == 'r':
                reverse(duty)

finally:
    print(«Nettoyage»)
    GPIO.cleanup()
```

Ce code emprunte beaucoup de lignes au code Python des expériences précédentes.

❶ Au début du fichier, on trouve le code habituel de configuration GPIO et les définitions des broches. La broche Enable du L293D sert à réguler la vitesse du moteur, donc la broche 18 qui y est connectée est configurée comme sortie PWM.

❷ La fonction `forward` définit les broches IN1 et IN2 pour piloter le sens de rotation, puis rattache le rapport cyclique au canal PWM.

❸ Si vous comparez avec la fonction `reverse`, vous constaterez que les valeurs des broches IN1 et IN2 sont interverties. La fonction `stop` définit les broches de direction sur la position arrêt (elles sont toutes les deux sur `LOW`) et le rapport cyclique est sur 0.

❹ La boucle principale `while` invite l'utilisateur à saisir une commande puis appelle `stop`, `forward` ou `reverse`, selon le cas.

Expérimentation Raspberry Pi

Le programme d'expérimentation avec le Raspberry Pi se nomme `ex_04_full_motor_control.py`. Lorsque vous l'exécuterez, vous constaterez qu'il fonctionne de la même façon que la version Arduino en vous invitant à saisir un sens de rotation et une vitesse.

```
$ sudo python ex_04_full_motor_control.py
Saisir la lettre de sens de rotation (f - forward/avant, r - reverse/arrière, s - stop/arrêt): f
Saisir le rapport cyclique (0 à 100): 50
la lettre de sens de rotation (f - forward/avant, r - reverse/arrière, s - stop/arrêt): f
Saisir le rapport cyclique (0 à 100): 100
Saisir la lettre de sens de rotation (f - forward/avant, r - reverse/arrière, s - stop/arrêt): s
Saisir la lettre de sens de rotation (f - forward/avant, r - reverse/arrière, s - stop/arrêt): f
Saisir le rapport cyclique (0 à 100): 50
la lettre de sens de rotation (f - forward/avant, r - reverse/arrière, s - stop/arrêt): f
Saisir le rapport cyclique (0 à 100): 100
Saisir la lettre de sens de rotation (f - forward/avant, r - reverse/arrière, s - stop/arrêt): s
Saisir la lettre de sens de rotation (f - forward/avant, r - reverse/arrière, s - stop/arrêt):
```

Ménagez votre moteur

Imaginez qu'une voiture roule tranquillement et que son conducteur passe brusquement la marche arrière — c'est ce que vous faites lorsque vous inversez soudain le sens de rotation d'un moteur. Pour les petits moteurs, qui ne transportent pas une masse importante, cela ne pose généralement pas trop de problèmes. Si vous utilisez un Raspberry Pi ou un Arduino alimenté par la même source d'alimentation que le moteur, vous constaterez éventuellement que le Raspberry Pi dysfonctionne ou que l'Arduino est réinitialisé. Cela est dû au courant élevé qui circule lorsque vous inversez brusquement le sens de rotation et qui produit une chute de la tension d'alimentation.

Pour les plus gros moteurs entraînant des masses ayant une grande inertie, des changements brusques de vitesse ou de sens peuvent engendrer des problèmes plus graves. Outre des courants élevés qui circuleront et risquent d'endommager le pont en H, cela provoque aussi un choc mécanique sur les engrenages du moteur.

Gardez ces remarques à l'esprit au moment de l'écriture du programme de commande des gros moteurs. Pour les préserver, vous pouvez précéder tout changement de sens par une mise à l'arrêt du moteur en prévoyant une pause suffisante pour un arrêt complet avant d'inverser le sens de rotation.

En Arduino, si vous reprenez les fonctions `forward` et `reverse` de l'expérience « Commande du sens et de la vitesse d'un moteur », plus haut, vous obtiendrez par exemple :

```
forward(255);
delay(200);
reverse(255);
```

Autres circuits intégrés de ponts en H

Il existe de nombreux modèles de circuits intégrés de commande des moteurs. Dans cette section, nous examinerons quelques-uns des modules les plus répandus.

L298N

À 600 mA, le courant maximum du L293D est assez bas. S'il vous en faut plus, vous pouvez faire appel à son grand frère, le L298N (*https://www.sparkfun.com/datasheets/Components/General/L298N.pdf*).

Les principales caractéristiques de ce circuit sont les suivantes.

- Plage de tension du moteur pouvant atteindre 50 V
- Courant du moteur de 2 A CC par moteur
- Courant moteur maxi de 3 A par moteur
- Compatible avec un circuit logique 3 V et 5 V (Pi et Arduino)

La figure 8-13 présente le brochage du module. La configuration est très proche de celle du L298D, mais le L298D permet aussi de réguler le courant transmis au moteur en ajoutant deux résistances de faible valeur.

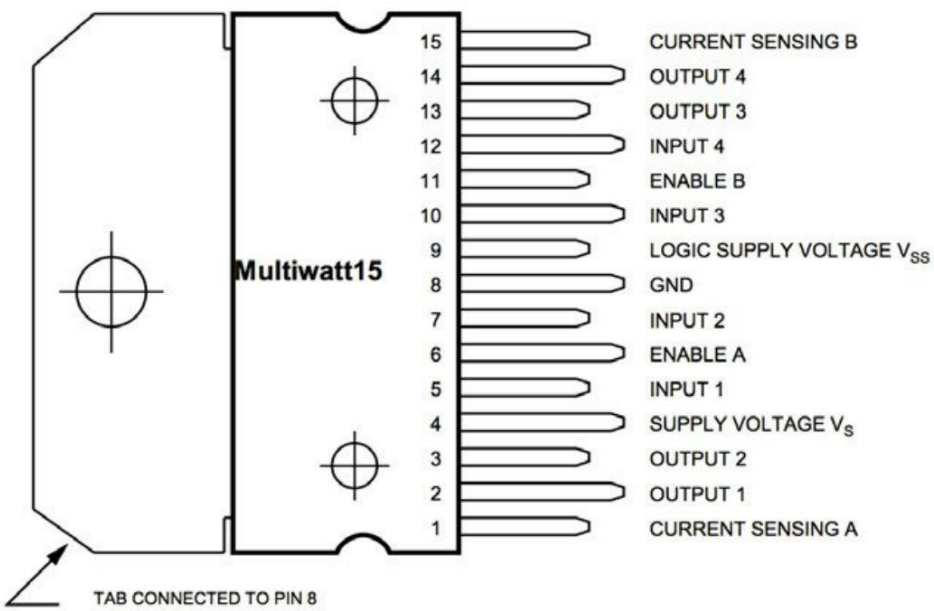


Figure 8-13. *Le circuit intégré du double pont en H L298N*

Comme vous pouvez le voir à la figure 8-13, le circuit intégré est un boîtier conçu pour être monté sur un dissipateur thermique en cas de commutation de puissance élevée. Les broches sont disposées sur deux rangées décalées. Par conséquent, elles ne peuvent pas être enfichées sur une plaque d'essai standard. La méthode la plus simple pour utiliser ce composant est d'acheter un module de pont en H bon marché sur lequel il est préinstallé. Les broches du L298N sont décrites dans le tableau 8-3.

Tableau 8-3. *Brochage du L298N*

Numéro de broche	Nom de la broche	Description
1	CURRENT SENSE A	Voir la discussion à la suite de ce tableau
2	OUTPUT 1	Sortie du demi-pont 1
3	OUTPUT 2	Sortie du demi-pont 2
4	Vs	Tension d'alimentation des moteurs
5	INPUT 1	Commande d'entrée du demi-pont 1
6	ENABLE A	Active les demi-ponts 1 et 2
7	INPUT 2	Commande d'entrée du demi-pont 2
8	GND	Masse
9	Vss	Alimentation électrique du circuit de commande ; elle peut être plus basse que l'alimentation électrique du moteur sur la broche 8 et elle est généralement de 5 V
10	INPUT 3	Commande d'entrée du demi-pont 2
11	ENABLE B	Active les demi-ponts 1 et 2
12	INPUT 4	Commande d'entrée du demi-pont 4
13	OUTPUT 3	Sortie du demi-pont 3
14	OUTPUT 4	Sortie du demi-pont 4
15	CURRENT SENSE B	Voir la discussion à la suite de ce tableau

La plupart des broches ont des équivalents directs sur le L293D. Comme ce dernier, les demi-ponts peuvent être activés par paire pour permettre la régulation MLI de la puissance transmise au moteur lorsque le module est utilisé comme pont en H.

Il peut être utile de pouvoir mesurer le courant fourni aux moteurs. Cela vous permet par exemple de détecter qu'un moteur ne tourne pas parce que quelque chose l'en empêche. Dans ce cas, le courant augmente fortement dans le moteur. Cela s'appelle le courant de calage. Si vous n'avez pas besoin de mesurer le courant passant par chacun des deux ponts en H, vous devez connecter les broches 1 et 15 à GND.

Pour mesurer le courant, vous devez connecter des résistances de faible valeur, mais ayant une puissance nominale élevée entre la broche 1 et GND, ainsi qu'entre la broche 15 et GND. Les résistances doivent être de faible valeur afin de ne pas interférer de façon notable avec le fonctionne-

ment du moteur, mais elles doivent avoir une puissance suffisamment élevée pour supporter la chaleur qu'elles généreront. La tension sur la résistance sera proportionnelle au courant qui circule. Cette tension pourra ensuite être mesurée par l'entrée analogique de l'Arduino (voir la section « Entrées Analogiques » du chapitre 2 page 18).

Supposons, par exemple, que vous pilotiez un moteur 12 V qui utilise normalement un courant de 500 mA mais qui, lorsqu'il cale, consomme un courant de 2 A. Vous pouvez vous permettre de sacrifier jusqu'à 0,5 V pour pouvoir détecter que le moteur cale.

La figure 8-14 montre le circuit correspondant utilisant un L298N, une résistance et un moteur pour l'un des deux ponts en H.

La broche CURRENT SENSE A est connectée à une entrée analogique Arduino (par exemple, A0) et les deux broches IN1 et IN2 sont connectées aux sorties numériques Arduino D2 et D3 pour piloter le sens de rotation du moteur.

Le courant maximum est de 2 A et vous voulez obtenir 0,5 V sur la résistance. Vous devez donc utiliser une résistance ayant la valeur R = V/I = 0,5/2 = 0,25 Ω. La conversion à la valeur de résistance standard la plus proche aboutit à 0,27 Ω ou 270 mΩ (milliohms – à ne pas confondre avec des mégohms). Par conséquent, à 2 A, la tension serait de V = I × R = 2 A × 0,27 Ω = 0,54 V.

Pour calculer la puissance nominale d'une résistance, multipliez la tension (0,54 V) par le courant (2 A) ; vous obtenez une puissance nominale d'à peine plus de 1 W. En tenant compte d'une marge d'erreur, vous utiliserez donc une résistance de 2 W.

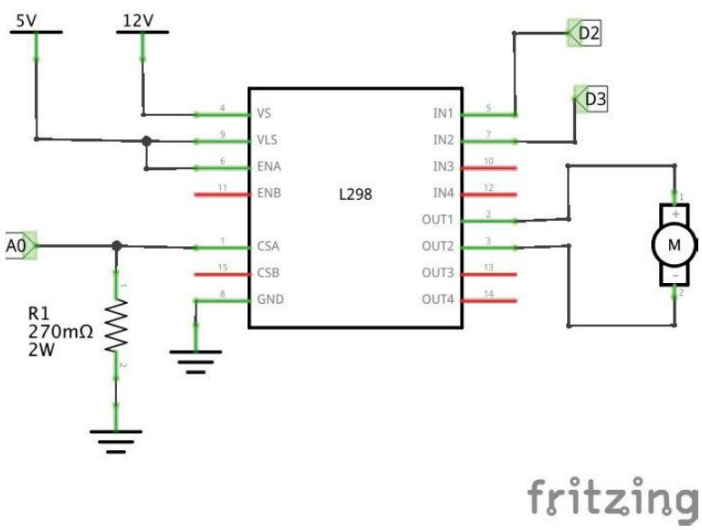


Figure 8-14. *Mesure du courant du moteur*

La broche CURRENT SENSE A peut être connectée directement sur une entrée analogique Arduino (le Raspberry Pi n'a pas d'entrée analogique) pour mesurer la tension. Comme la tension est inférieure à la plage de 5 V des entrées analogiques Arduino, à 2 A (0,5 V), la valeur brute `analogInput` mesurée sera d'environ 100. Le niveau de précision reste suffisant.

Le sketch Arduino suivant montre comment couper automatiquement les moteurs si le courant dépasse 1,5 A, ce qui signifie que le moteur cale :

```
const float R = 0.27;
const int in1pin = 2;
const int in2pin = 3;
const int sensePin = A0;

void setup() {
  pinMode(in1pin, OUTPUT);
  pinMode(in2pin, OUTPUT);
  // moteur en marche avant
  digitalWrite(in1pin, HIGH);
  digitalWrite(in2pin, LOW);
}

void loop() {
  int raw = analogRead(sensePin);
  float v = raw / 204.6; // 204.6 = 1024 / 5V
  float i = v / R;
  if (i > 1.5) {
    // arrêt du moteur
    digitalWrite(in1pin, LOW);
    digitalWrite(in2pin, LOW);
  }
}
```

La fonction `loop` lit une entrée analogique, ce qui fournit une valeur maximale de 1 023 pour 5 V. Donc pour convertir la valeur analogique brute lue en tension, il faut la diviser par 204,6, soit 1 023/5.

Le courant peut alors être calculé à l'aide de la loi d'Ohm.

TB6612FNG

Les modules L293D et L298N existent depuis des années. Ces composants un peu démodés qui utilisent des transistors bipolaires au lieu de MOSFET ont tendance à chauffer même à des courants relativement bas.

Le TB6612FNG est un dispositif beaucoup plus moderne dont la puissance nominale est identique à celle du L293D. Le TB6612FNG intègre aussi un double pont en H.

Il n'admet qu'une tension moteur maximale de 15 V et le courant combiné des deux ponts en H doit être inférieur à 1,2 A (3,2 A en pointe). Le composant intègre une coupure thermique.

Il est également disponible sous la forme d'un circuit monté en surface, mais vous pouvez acheter des modules prêts à l'emploi qui utilisent ce circuit intégré et peuvent être montés sur une plaque d'essai.

Modules à ponts en H

Au lieu de fabriquer votre propre pont en H, vous pouvez utiliser un module dédié. Il en existe de toutes formes et dimensions qui s'adaptent aux différents courants des moteurs. La figure 8-15 présente différents modules de contrôle de moteurs à ponts en H.

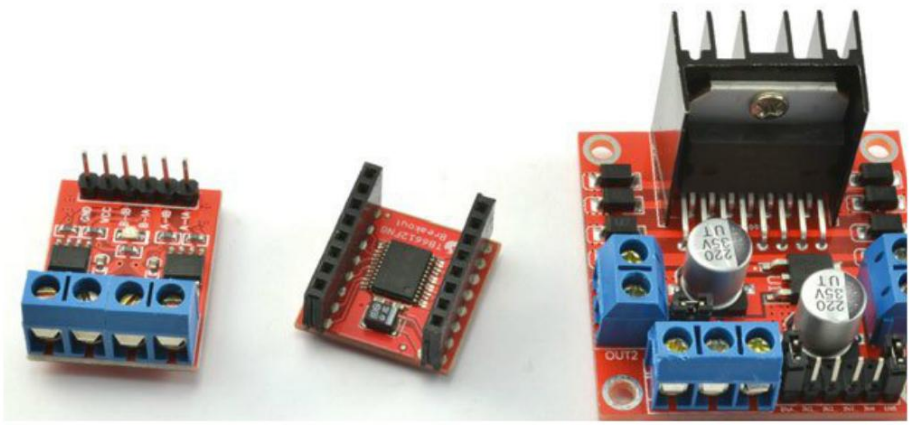


Figure 8-15. *Modules à ponts en H*

Le module sur la gauche de la figure 8-15 est un modèle très bon marché vendu sur eBay qui utilise deux L9110S, chaque circuit intégré étant un seul pont en H. Les quatre bornes à vis se connectent aux deux moteurs CC ; les broches du connecteur sont simplement des ports GND, VSS (courant faible 5 V), ainsi que quatre broches de commande de la direction, comme le L293D.

Le module au centre de la figure 8-15 utilise un TB6612FNG. Il est fabriqué par Sparkfun (référence ROB-09457). Vous pouvez souder votre propre barrette de broches sur les connecteurs pour fixer le module sur une plaque d'essai ou, comme ici, des fiches permettant de connecter des câbles flexibles mâles/mâles.

Sur la droite de la figure 8-15, vous pouvez voir un module qui utilise le L298N et qui est doté d'un dissipateur thermique. Le module comporte également des diodes de protection de chaque côté du dissipateur thermique et il intègre même un régulateur de tension pouvant fournir une alimentation de 5 V à l'Arduino, mais qui n'est probablement pas suffisante pour un Raspberry Pi. Il n'y a pas de résistance de détection du courant.

Il existe des modules de contrôle encore plus puissants. Vous en trouverez notamment sur le site web de Pololu (*https://www.pololu.com/*). Vous y trouverez même des ponts en H résistant à des dizaines d'ampères. Notez que le prix augmente en même temps que le courant.

En plus de modules de ponts en H séparés, vous trouverez aussi des shields pouvant être enfichées sur un Arduino et des cartes enfichables compatibles avec le Raspberry Pi. La figure 8-16 présente trois de ces cartes.

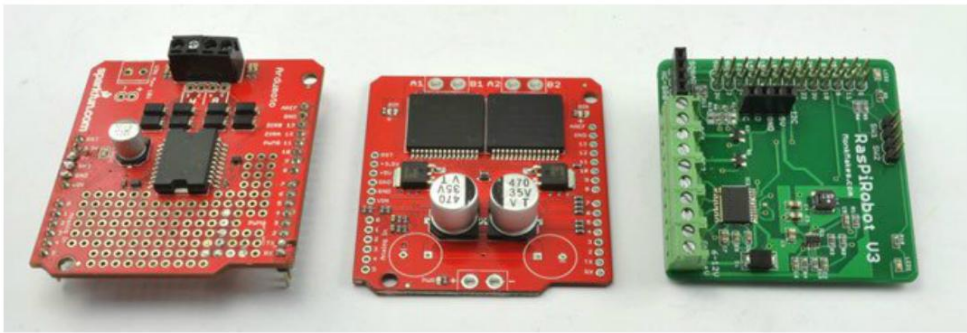


Figure 8-16. *Shields de ponts en H pour Arduino et Raspberry Pi*

La carte sur la gauche de la figure 8-16 est un shield de moteur Arduino fabriqué par Sparkfun (référence DEV-09815) basé sur le L298P, une version montée en surface du L298N. Cette carte comporte une zone de prototypage permettant d'y monter d'autres composants. La carte du milieu est un shield de pont en H de puissance élevée pour Arduino, vendue par Polulu, capable de commuter des charges jusqu'à 30 A.

Sur la droite de la figure 8-16, vous pouvez voir la carte RasPiRobot V3 qui s'enfiche sur les connecteurs GPIO d'un Raspberry Pi et utilise un TB6612FNG pour piloter deux moteurs CC.

Projet : compacteur de canettes Arduino

Vous souvenez-vous des vérins que nous avons étudiés au chapitre 7 ? En bricolant un peu, vous pouvez en assembler un avec un module de pont en H et un Arduino pour fabriquer un compacteur de canettes (figure 8-17).

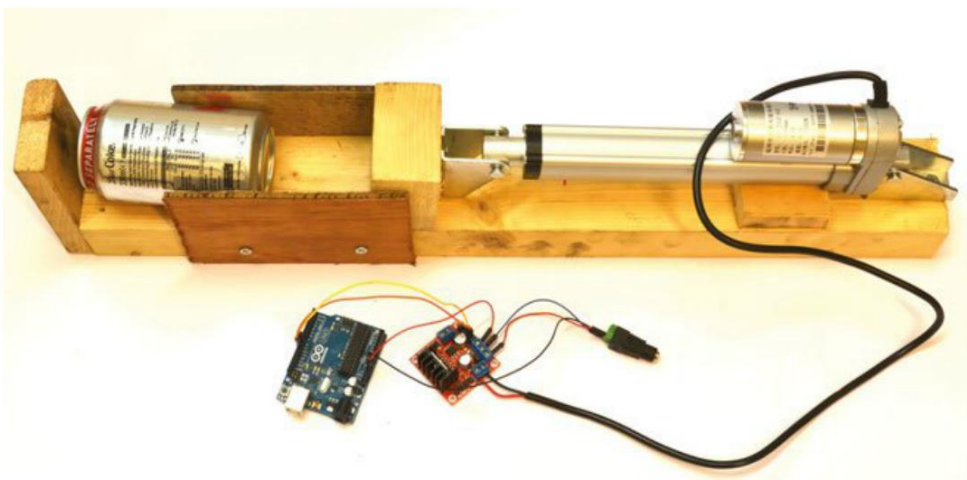


Figure 8-17. *Un compacteur de canettes Arduino*

Vous pouvez voir une vidéo d'un prototype de compacteur en action sur *https://youtu.be/qbWMElFnq2I*.

Les vérins sont costauds

Le but de ce projet est de fabriquer un compacteur de canettes de boisson, mais le vérin est parfaitement capable de vous écraser la main ou tout ce que vous aurez égaré à proximité. Soyez prudent !

Composants nécessaires

En plus d'un Arduino Uno, les composants suivants sont nécessaires pour la réalisation du projet.

Composant	Source
Vérin 12 V de 152 mm	eBay
Module à ponts en H L298	eBay
2× câbles flexibles mâles/femelles	Adafruit : 826
4× câbles flexibles mâles/mâles	Adafruit : 758
Jack femelle pour visser l'adaptateur de bornes	Adafruit : 368
Alimentation électrique (12 V à 3 A au minimum)	Adafruit : 352
Planche en bois de 50 × 100 mm et du contreplaqué	Magasin de bricolage
Vis à bois et outils de menuiserie	Magasin de bricolage

Le prix des vérins est extrêmement variable. Pour le compacteur de canettes, choisissez un modèle ayant une course de 150 mm. Vérifiez son courant maximum et choisissez un module de pont en H adéquat. Comme le modèle de vérin que j'utilise habituellement a un courant maximum de 3 A, je choisis un pont en H basé sur un circuit L298.

Câblage

La figure 8-18 présente le schéma de câblage du projet et la figure 8-19 montre l'Arduino et le module de pont en H.

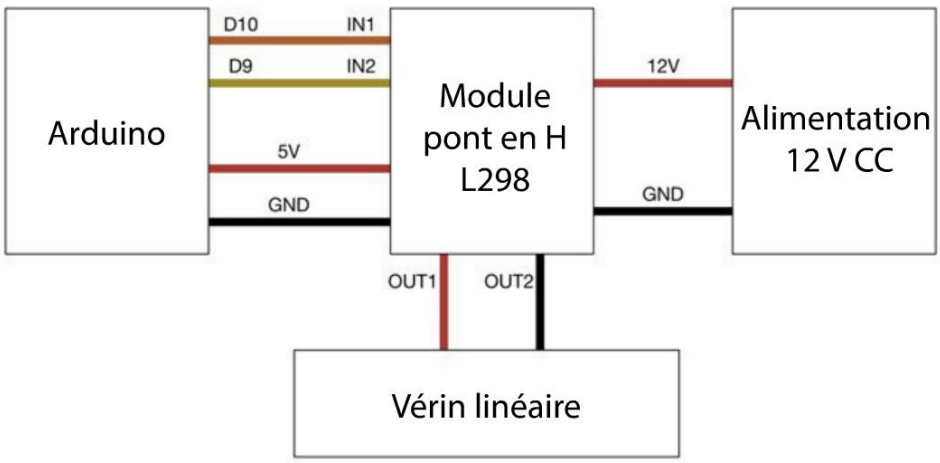


Figure 8-18. *Schéma de câblage du compacteur de canettes*

Par défaut, des cavaliers du module L293 conservent les deux ponts en H activés. Mais, seules les deux sorties Arduino connectées à IN1 et IN2 suffisent.

Le module de pont en H intègre un régulateur de tension qui comporte une sortie 5 V pouvant être connectée à la broche 5 V de la carte Arduino pour son alimentation électrique.

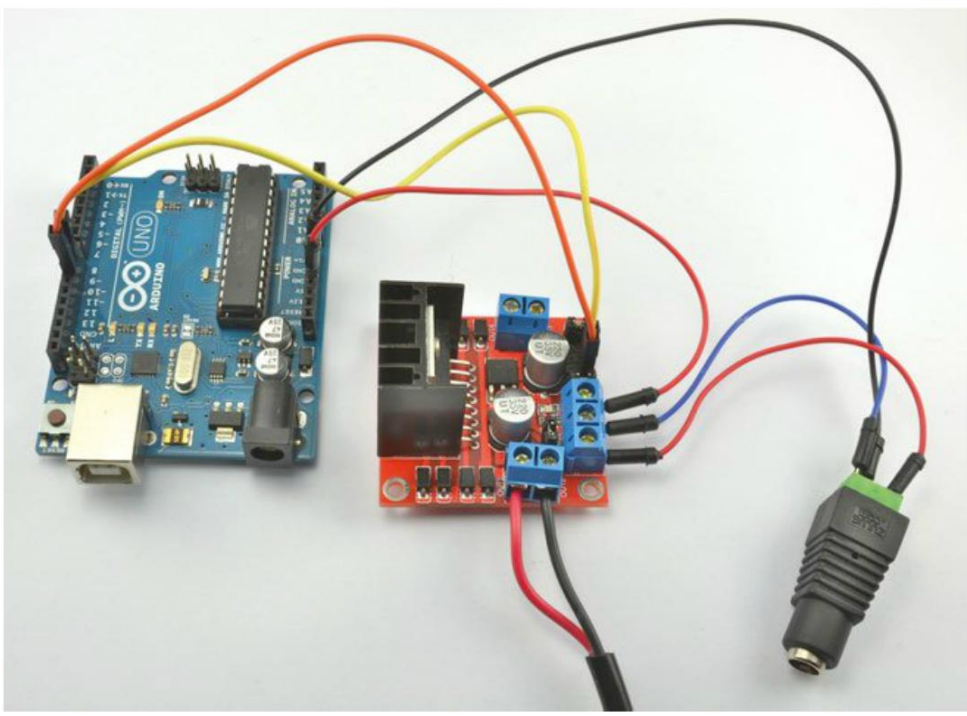


Figure 8-19. *La carte Arduino et le module du pont en H*

Construction

Comme vous pouvez le voir à la figure 8-17, la structure de base du projet est une planche en bois de 50 × 100 mm. Le vérin a été fixé à l'une des extrémités. Un bloc de bois a été fixé à l'extrémité de l'arbre du vérin. Ce bloc de bois écrase la canette contre l'extrémité opposée de la structure. Deux parois latérales en contreplaqué empêchent la canette de s'échapper lorsqu'elle est écrasée.

Je n'ai pas fourni de dimensions précises, car la taille de votre vérin peut varier légèrement. Pour définir les dimensions exactes, posez le vérin sur la planche en bois, puis calculez les distances appropriées. Prévoyez un espace suffisant entre l'arbre entièrement déployé et l'extrémité de la structure afin d'éviter que la machine n'écarte la structure elle-même.

Programme Arduino

Le bouton de réinitialisation de la carte Arduino sert à démarrer le compactage. Lorsque l'Arduino est réinitialisé, il commence automatiquement à actionner le vérin. Le code du projet se trouve dans le fichier `pr_02_can_crusher` :

```
const int in1Pin = 10;
const int in2Pin = 9;

const long crushTime = 30000;  ❶

void setup() {                 ❷
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  crush();
  stop();
  delay(1000);
  reverse();
  stop();
}

void loop() {
}

void crush()
}
  digitalWrite(in1Pin, LOW);
  digitalWrite(in2Pin, HIGH);
  delay(crushTime);
}

void reverse()
}
  digitalWrite(in1Pin, HIGH);
  digitalWrite(in2Pin, LOW);
  delay(crushTime);
}
```

```
void stop()
}
  digitalWrite(in1Pin, LOW);
  digitalWrite(in2Pin, LOW);
}
```

❶ Même si le vérin s'arrête automatiquement lorsqu'il arrive en fin de course, cette période (de 30 secondes pour mon moteur) définit la durée de fonctionnement du moteur avant le changement de sens.

❷ La fonction `setup` contrôle la totalité du déroulement du projet. Après avoir défini les deux broches de commande comme sortie, elle démarre immédiatement le compactage à l'aide de la fonction `crush`.

Résumé

Dans ce chapitre, vous avez appris à piloter le sens de rotation des moteurs CC à l'aide d'un pont en H. Les ponts en H peuvent même être utilisés pour piloter d'autres types de moteurs, dont des moteurs pas-à-pas (chapitre 10). Les ponts en H peuvent aussi servir de commutateur de puissance pour d'autres composants, comme des modules thermoélectriques Peltier (chapitre 11).

Servomoteurs

Les servomoteurs (à ne pas confondre avec les moteurs pas-à-pas que nous étudierons au chapitre 10) se composent d'un petit moteur CC, d'engrenages et d'un circuit de commande muni d'une résistance variable pour être capable de positionner le bras d'un servomoteur selon un angle précis.

Ils trouvent leur utilité dans les projets qui consistent à déplacer un objet de façon assez rapide et relativement précise.

Servomoteurs

Même si les servomoteurs continus peuvent tourner de façon continue, la plupart des servomoteurs (ou « servos ») tournent uniquement sur 180°. Ils sont souvent utilisés en modélisme pour commander la direction ou les angles des ailerons d'un avion téléguidé, par exemple. La figure 9-1 présente deux tailles de servos.

Figure 9-1. *Un miniservomoteur de 9 g et un servomoteur standard*

Le servo de droite est un modèle standard ; c'est la taille la plus répandue. Ses dimensions, et même celles des pattes de fixation, sont assez standardisées. Le modèle de gauche, beaucoup plus petit et léger, est employé dans les engins volants.

Les servos illustrés à la figure 9-1 sont souvent utilisés en modélisme. Ils sont de qualité variable. Les modèles haut de gamme, offrant un couple élevé, ont souvent des engrenages en métal à la place du Nylon. La majorité des servos fonctionnent en 5 V environ, souvent avec une plage de tension admise comprise entre 4 et 7 V. En modélisme, les câbles des servos se terminent par une prise à trois fils (+, - et signal de commande).

Il existe aussi des servomoteurs beaucoup plus gros pour les applications nécessitant des puissances élevées, mais ils ne sont pas aussi normalisés que ceux en modélisme.

Fonctionnement d'un servomoteur

La figure 9-2 présente le fonctionnement d'un servomoteur.

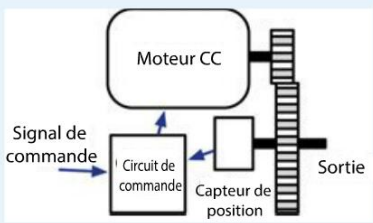


Figure 9-2. *Fonctionnement d'un servomoteur*

Un servo est composé d'un moteur CC qui actionne des réducteurs de vitesse à engrenages afin de réduire la vitesse de rotation du moteur tout en augmentant le couple. L'arbre de transmission est connecté à un capteur de position (généralement, une résistance variable) pour contrôler la position de l'arbre. Un circuit de commande se sert des données transmises par le capteur de position, ainsi que d'un signal de commande qui règle la position voulue, pour commander la puissance et le sens de rotation du moteur CC.

En déduisant la position actuelle de la position voulue, l'unité de commande calcule la déviation qui peut être positive ou négative. Cette déviation est ensuite utilisée pour l'alimentation du moteur. Plus la différence entre la position actuelle et la position voulue de l'arbre est importante, plus le moteur tourne vite pour parvenir à la position désirée. À l'approche de la consigne, la puissance du moteur diminue.

Commande d'un servo

Comme vous devez vous y attendre, le signal de commande du servo n'est pas une tension, mais un signal MLI. Ce signal, illustré à la figure 9-3, est normalisé pour tous les servos utilisés en modélisme.

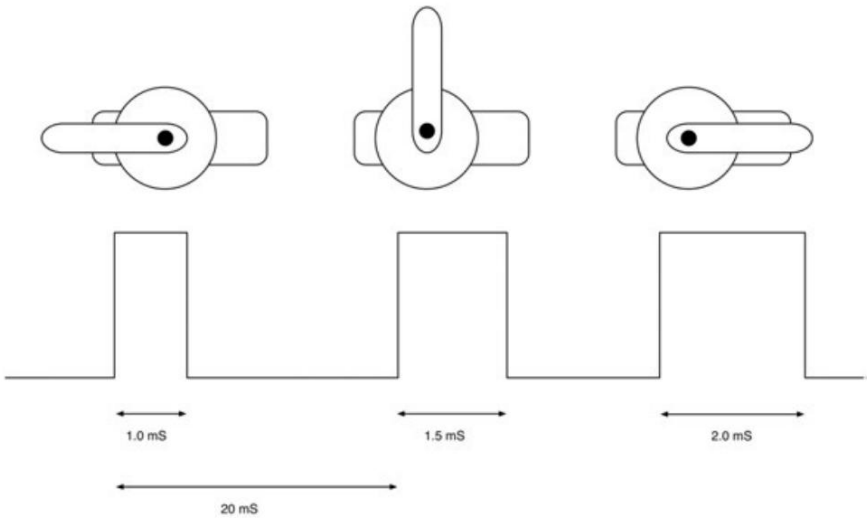


Figure 9-3. *Commande d'un servomoteur*

Un train d'impulsions de 1,5 milliseconde place le servo en position centrale à 90°. Des impulsions de 1 milliseconde le placent à 0° et des impulsions de 2 millisecondes l'orientent à 180°. En réalité, cette course peut être légèrement inférieure à 180°, sans que les impulsions ne soient plus courtes à une extrémité et plus longues à l'autre. En fait, il n'est pas rare que l'impulsion 0° doive être de 0,5 milliseconde et celle de 180° soit effectivement de 2,5 millisecondes.

Le servo s'attend à recevoir une impulsion toutes les 20 millisecondes.

Expérience : commande de la position d'un servomoteur

Dans cette expérience, vous vous servirez d'un Raspberry Pi et d'un Arduino pour régler la position du bras d'un servomoteur selon un angle particulier.

L'Arduino a une bibliothèque Servo qui permet de générer des impulsions sur toutes les broches ; vous n'avez donc pas besoin d'utiliser de broche PWM spécifique.

Pour tester le servo, vous communiquerez l'angle de position du servomoteur à l'aide du moniteur série.

Il est beaucoup plus difficile de générer des impulsions d'une durée précise sur un Raspberry Pi que sur un Arduino. Ce dernier est doté de composants qui génèrent des impulsions, tandis qu'un Raspberry Pi crée ces impulsions de façon logicielle. Comme le nano-ordinateur a un système d'exploitation qui permet à de nombreux processus de se partager le temps de calcul du processeur, les impulsions MLI sont parfois plus longues que prévues. Le mouvement du servo est donc moins précis. Même s'il n'en reste pas moins utilisable, si vous avez besoin de davantage de précision, vous pouvez utiliser des composants MLI externes, comme décrit plus loin dans le projet « Pepe, la marionnette Raspberry Pi qui danse ».

Matériel

L'intérêt des servomoteurs est que l'électronique de commande du moteur est incluse dans un unique boîtier. Il est donc inutile de prévoir un pont en H ou des transistors de commande distincts pour le moteur. Il suffit d'alimenter les broches en 5 V ou 6 V et de transmettre des impulsions de faible intensité sur une sortie numérique.

La figure 9-4 présente un servo connecté à un Raspberry Pi.

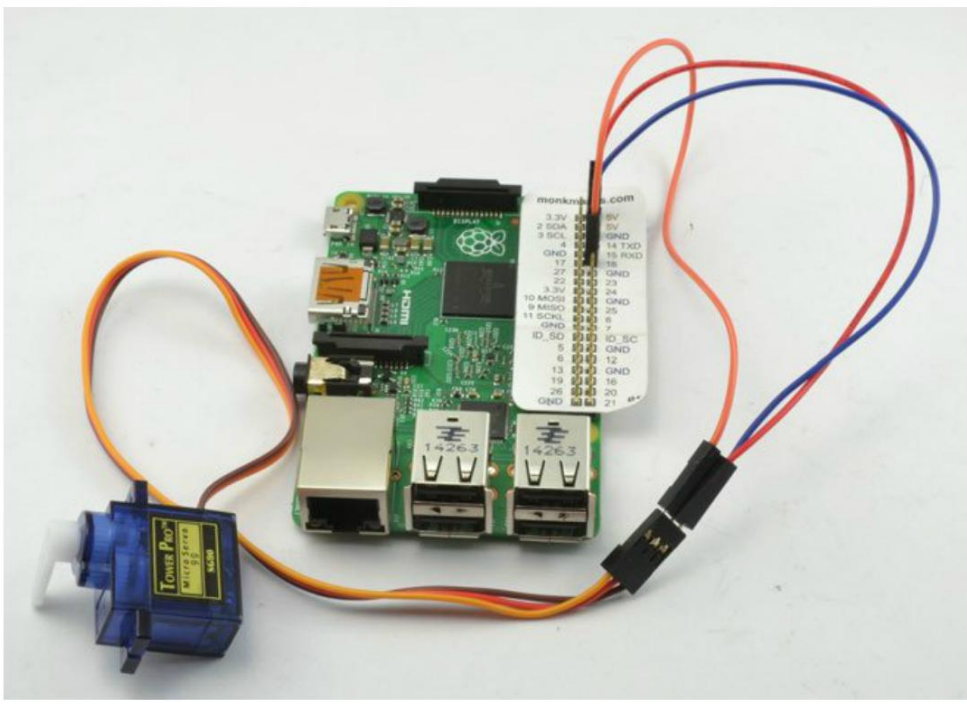


Figure 9-4. *Un servomoteur commandé par un Raspberry Pi*

À la figure 9-4, vous pouvez voir que l'un des bras (fourni avec le servo) a été fixé sur le servo afin de mieux visualiser la position du moteur.

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou un Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser l'expérience.

Composant	Source
Miniservomoteur 9 g	eBay, Adafruit : 169
Câbles flexibles mâles/mâles	Adafruit : 758
Câbles flexibles femelles/mâles (Pi uniquement)	Adafruit : 826

Si vous comptez tenter cette expérience avec un Raspberry Pi, vous aurez besoin de câbles flexibles femelles/mâles pour connecter les broches GPIO du nano-ordinateur au servo.

Le petit servo de 9 g fonctionnera correctement en étant alimenté en 5 V par le Pi ou l'Arduino. Mais si vous avez besoin d'un plus gros servo, vous devrez éventuellement utiliser une alimentation séparée, telle que le support de batterie 6 V que vous avez utilisé dans d'autres expériences.

Raccordement de l'Arduino

La figure 9-5 montre le raccordement de l'Arduino avec le servomoteur.

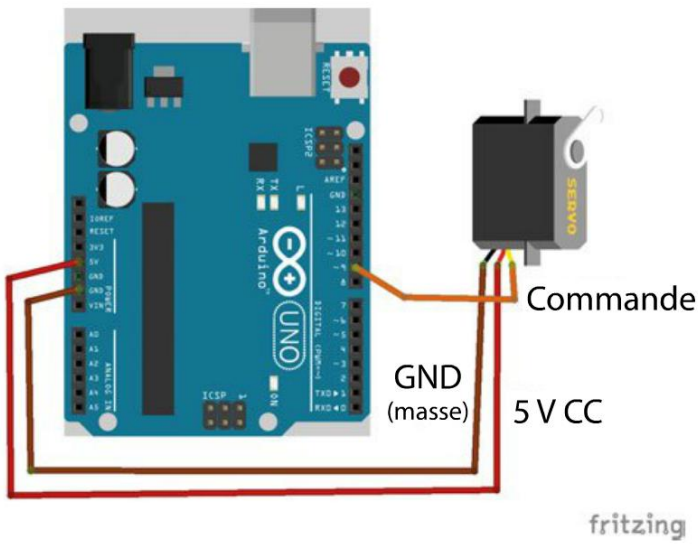


Figure 9-5. *Raccordement d'un servomoteur à une carte Arduino*

Des câbles flexibles mâles/mâles sont utilisés pour connecter la prise à trois fils du servomoteur à l'Arduino.

Connexion du servomoteur

La figure 9-6 présente un servomoteur avec son connecteur. Comme d'autres aspects des servos, ces connexions sont identiques, ou presque, chez tous les fabricants.

Les fils du moteur sont identifiés par des couleurs et sont disposés dans l'ordre suivant :

- Marron (parfois noir) pour la masse
- Rouge pour +V
- Orange (parfois jaune) pour le circuit de commande

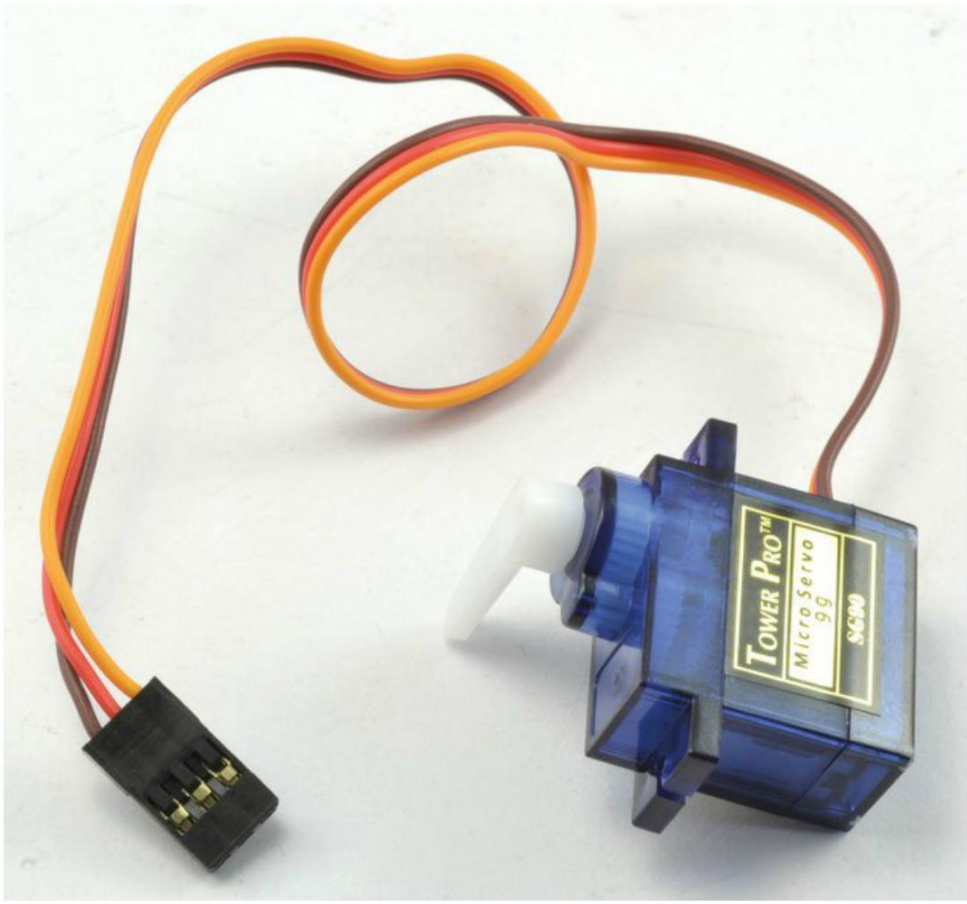


Figure 9-6. *Connexions du servomoteur*

Programme Arduino

La bibliothèque Servo Arduino, qui est incluse avec l'IDE Arduino, facilite grandement le travail avec les servos.

Vous trouverez le sketch Arduino de ce projet dans le dossier `arduino/experiments/servo` à l'endroit où vous avez téléchargé le code du livre (voir la section « Le code du livre » du chapitre 2 page 14).

```
#include <Servo.h>

const int servoPin = 9;          ❶

Servo servo;                     ❷

void setup() {
  servo.attach(servoPin);        ❸
```

```
  servo.write(90);              ❹
  Serial.begin(9600);           ❺
  Serial.println(«Saisir un angle en degrés»);
}

void loop() {                   ❻
  if (Serial.available()) {
    int angle = Serial.parseInt();
    servo.write(angle);         ❼
  }
}
```

Même en utilisant la modulation de longueur d'impulsions et `analogWrite` pour générer une impulsion de la durée voulue pour commander un servo, cela impliquerait aussi de changer la fréquence MLI et de limiter le choix de broches de commande du servo aux broches compatibles. C'est la méthode adoptée dans la section « Programme Raspberry Pi » page 163, mais sur l'Arduino, il est plus facile d'utiliser la bibliothèque Servo.

❶ Après l'importation de la bibliothèque Servo, une constante (`servoPin`) est définie comme broche connectée à la borne de commande du servo.

❷ La variable `servo` de type `Servo` est déclarée. Cette variable est utilisée dès qu'il faut changer la position du servo.

❸ La fonction `setup()` commence les impulsions en associant la broche de commande à la génération d'impulsions à l'aide de `servo.attach`.

❹ Définit l'angle du servo sur 90° (position centrale).

❺ `setup` démarre aussi la communication série de sorte que vous pouvez envoyer des commandes d'angle via le moniteur série.

❻ La fonction `loop` attend de recevoir une commande d'angle, puis elle la convertit en un nombre à l'aide de `parseInt`.

❼ Le nouvel angle du servo est alors défini à l'aide de `servo.write`.

Expérimentation Arduino

Téléversez le sketch sur l'Arduino. Le bras du moteur doit se placer immédiatement en position centrale (90°). Ouvrez le moniteur série (figure 9-7) et saisissez des angles compris entre 0 et 180°. Le moteur doit changer de position à chaque nouvelle saisie.

Figure 9-7. *Commande de la position du servo à l'aide du moniteur série*

Raccordement du Raspberry Pi

Pour raccorder le Raspberry Pi, utilisez des câbles flexibles femelles/mâles pour relier les broches du connecteur GPIO au servo, comme illustré à la figure 9-8.

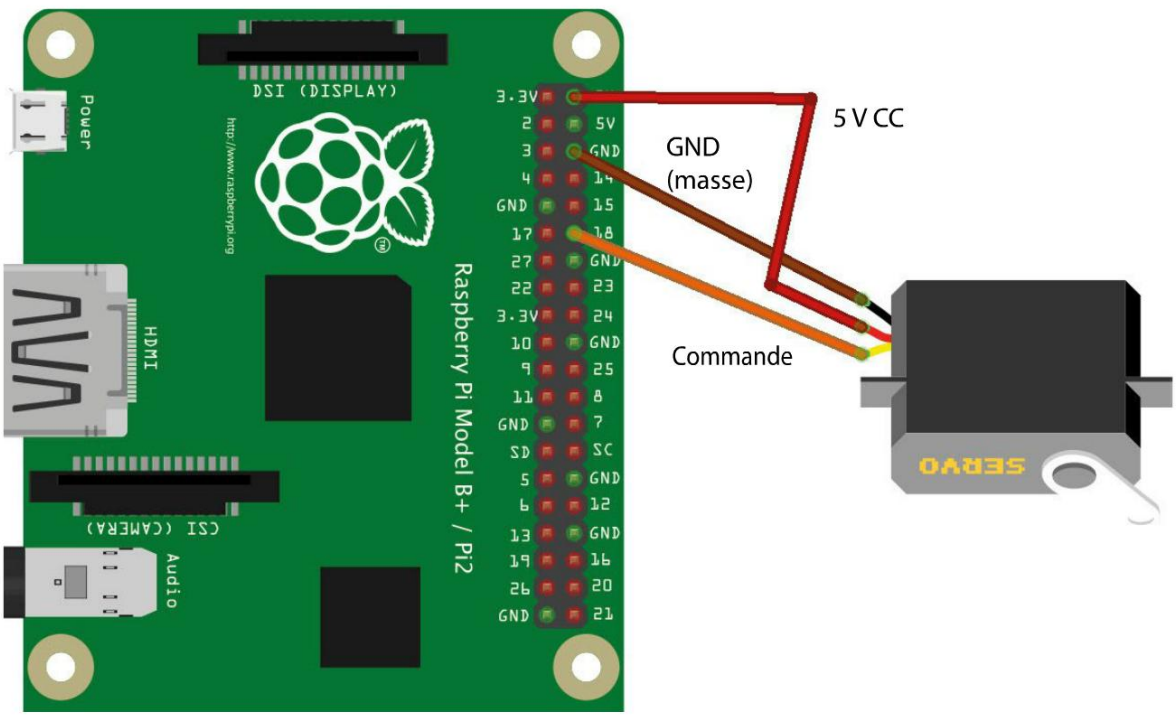


Figure 9-8. *Raccordement d'un Raspberry Pi à un servo*

Programme Raspberry Pi

Vous trouverez le programme Python de cette expérience dans le fichier `python/experiments/servo.py`. Pour plus d'informations sur l'installation des programmes Python sur le Raspberry Pi, voir la section « Le code du livre » du chapitre 3 page 34.

Ce programme utilise la fonction PWM de la bibliothèque RPi.GPIO pour générer les impulsions de commande du servo.

Beaucoup de calculs sont nécessaires pour parvenir à la longueur d'impulsions correctes. Vous n'avez pas besoin de tout suivre en détail. Il vous suffit de copier le code dans vos projets personnels et d'appeler `set_angle` pour définir l'angle du bras du servo :

```
import RPi.GPIO as GPIO
import time

servo_pin = 18

# Corriger ces valeurs en fonction de l'amplitude des mouvements du servo
deg_0_pulse = 0.5 # ms                    ❶
deg_180_pulse = 2.5 # ms
f = 50.0 #50Hz = 20 ms entre les impulsions  ❷
# Calculs des paramètres de la longueur d'impulsion
period = 1000 / f # 20 ms                 ❸
k = 100 / period       # rapport cyclique 0..100 sur 20 ms  ❹
deg_0_duty = deg_0_pulse * k              ❺
pulse_range = deg_180_pulse - deg_0_pulse
duty_range = pulse_range * k              ❻

# Initialise la broche GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(servo_pin, GPIO.OUT)           ❼
pwm = GPIO.PWM(servo_pin, f)
pwm.start(0)

def set_angle(angle):                     ❽
    duty = deg_0_duty + (angle / 180.0) * duty_range
    pwm.ChangeDutyCycle(duty)

try:
    while True:                           ❾
        angle = input(«Saisir l'angle (0 à 180): «)
        set_angle(angle)
finally:
    print("Nettoyage")
    GPIO.cleanup()
```

❶ Comme chaque servo nécessite des longueurs d'impulsions légèrement différentes pour étendre au maximum son champ d'angles, deux constantes (`deg_0_pulse` et `deg_180_pulse`) sont utilisées pour définir les durées des impulsions pour un angle de 0 et 180°, respectivement. Ces paramètres peuvent être ajustés en fonction de l'amplitude des mouvements du servo.

Le bloc de code suivant effectue des calculs relatifs à la longueur d'impulsions.

❷ Pour obtenir une impulsion toutes les 20 millisecondes, il faut définir la fréquence PWM (`f`) sur 50 impulsions par seconde.

❸ La période (20 millisecondes) correspond à 1 000 divisé par `f`. Si vous voulez utiliser une autre fréquence d'impulsions, changez la valeur `f` ; les calculs sont automatiquement corrigés.

❹ La valeur du rapport cyclique MLI doit être comprise entre 0 et 100. La constante `k` est donc définie à 100 sur la période et la constante peut être utilisée pour corriger l'angle en fonction de la valeur du rapport cyclique.

❺ Pour convertir la longueur d'impulsions pour l'angle zéro en une valeur de rapport cyclique correspondante comprise entre 0 et 100, la longueur d'impulsions est multipliée par `k`.

❻ De même, la plage de valeurs de rapport cyclique est aussi calculée en multipliant l'amplitude des longueurs d'impulsions `pulse_range` par `k`.

❼ Ensuite, la broche GPIO est définie et la MLI commence.

❽ La fonction `set_angle` convertit l'angle en valeur de rapport cyclique puis appelle `ChangeDutyCycle` pour définir la nouvelle longueur d'impulsions.

❾ La boucle principale ressemble beaucoup à la version Arduino de ce programme : elle invite à saisir un angle avant de le régler.

Expérimentation avec le Raspberry Pi

Exécutez le programme à l'aide de la commande :

```
$ sudo python servo.py
```

L'interaction avec le programme ressemble à la commande du servo avec une carte Arduino par le biais du moniteur série. Saisissez des angles et observez le mouvement du servo pour se placer selon l'angle voulu :

```
$ sudo python servo.py
Saisir l'angle (0 à 180): 90
Saisir l'angle (0 à 180): 0
Saisir l'angle (0 à 180): 180
Saisir l'angle (0 à 180): 90
Saisir l'angle (0 à 180): 0
```

En général, le servo réagit correctement, mais si vous l'observez attentivement, vous devriez remarquer des mouvements saccadés, surtout si le Raspberry Pi traite plusieurs processus simultanément. Cela s'explique par le fait que le nano-ordinateur génère parfois une impulsion plus longue que d'habitude, car son processeur est occupé par d'autres activités.

Si le servo a un mouvement trop saccadé, au lieu de générer les impulsions sur le Raspberry Pi de façon logicielle, vous pouvez utiliser un module matériel comme le contrôleur PWM/Servo 16 canaux d'Adafruit. Ce module n'utilise que deux broches pour l'interface avec le Raspberry Pi et permet de commander jusqu'à 16 servos à l'aide d'une bibliothèque Python également fournie par Adafruit.

Projet : Pepe, la marionnette Raspberry Pi qui danse

Les servomoteurs réagissent très rapidement aux commandes. Dans ce projet, nous nous en servirons pour tirer les ficelles d'une marionnette afin de la faire danser (figure 9-9).

Figure 9-9. *Pepe la marionnette Pi*

Ce projet nécessite simplement de définir la liste des mouvements des moteurs afin que le programme commande leur exécution à la façon de la lecture d'un enregistrement. Par la suite, la marionnette apprendra à parler (dans le projet « Pepe la marionnette qui parle », décrit au chapitre 15). Puis, au chapitre 16, viendra s'y ajouter une dimension Internet et la marionnette dansera en réaction à un hashtag contenu dans un tweet.

Composants nécessaires

Pour réaliser ce projet, il vous faut les composants suivants.

Composant	Source
Contrôleur PWM/Servo 12 bits 16 canaux Adafruit	Adafruit : 815
4× servo de 9 g	eBay, Adafruit : 169
Jack femelle pour l'adaptateur de bornes à vis	Adafruit : 368
Câbles flexibles femelles/femelles	Adafruit : 266
Câbles flexibles mâles/mâles	Adafruit : 758
Alimentation électrique 5 V 2 A	Adafruit : 276
4× piques en bois (environ 7 cm)	Supermarché
Petite marionnette (une ficelle par membre)	eBay
Feuille de carton épais de format A4	Magasin de fournitures pour loisirs créatifs
Pistolet à colle ou colle époxy et une perceuse	Magasin de bricolage

Montage

Ce projet utilise un Raspberry Pi, essentiellement parce qu'il sera ensuite adapté pour la communication sur Internet, comme nous le verrons au chapitre 16. Cependant, vous pouvez aussi utiliser un Arduino ; dans ce cas, vous n'avez pas besoin du contrôleur Adafruit et vous pouvez connecter les servos à la carte Arduino par l'intermédiaire d'une plaque d'essai et de câbles flexibles mâles/mâles.

Ce projet utilise quatre petits servos de 9 g pour les bras et les jambes de la marionnette. Un contrôleur servo 16 canaux Adafruit simplifie la commande. Grâce à ce module, les prises des conducteurs des servos peuvent être enfichées directement sur les broches de la carte du contrôleur.

La figure 9-10 présente le schéma du circuit du projet.

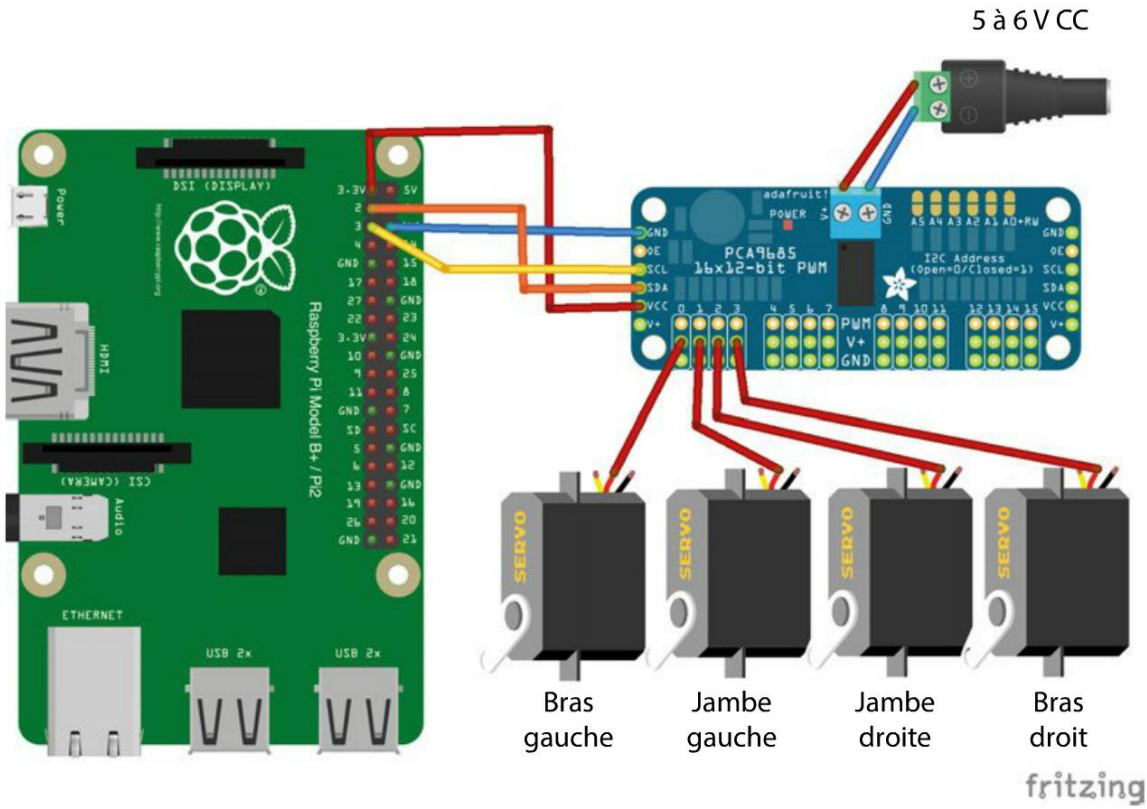


Figure 9-10. *Schéma du circuit du projet de la marionnette*

Il est possible d'enficher directement les prises des servomoteurs sur les broches du connecteur (pour simplifier, un seul conducteur est illustré pour chaque servo à la figure 9-10).

La commande simultanée de quatre servomoteurs implique de prévoir une alimentation électrique séparée pour les moteurs afin que les perturbations produites ne nuisent pas au bon fonctionnement du Raspberry Pi.

Les bras en plastique fournis avec les servomoteurs ne sont pas assez longs pour actionner les bras et les jambes de la marionnette. C'est pourquoi nous les prolongeons à l'aide de piques en bois.

Construction

Ce projet nécessite autant de bricolage que de bidouillage électronique ou informatique. Voici les étapes requises pour la fabrication de la marionnette qui danse.

Étape 1 : allongement des bras du servo

Les servomoteurs sont livrés avec un sachet contenant différents types de bras en plastique qui s'adaptent sur le servomoteur. Choisissez l'un des modèles droits et collez une baguette en bois comme illustré à la figure 9-11. Utilisez de préférence un pistolet à colle ou de la colle époxy.

Figure 9-11. *Allongement des bras du servo*

Notez que j'ai aussi posé une goutte de colle à l'extrémité de chaque pique pour empêcher la ficelle de glisser après avoir été nouée sur la baguette.

Étape 2 : fabrication d'un châssis

Les quatre bras des servomoteurs doivent pouvoir monter et descendre librement pour actionner les membres. Pour que tout reste à sa place, j'ai découpé une plaque de carton épais.

Pour connaître l'emplacement des découpes, imprimez le gabarit que vous trouverez dans le fichier `puppet.svg` dans le dossier `python/projets/puppet`, puis collez-le sur le carton avant de le découper à l'aide d'un cutter (figure 9-12).

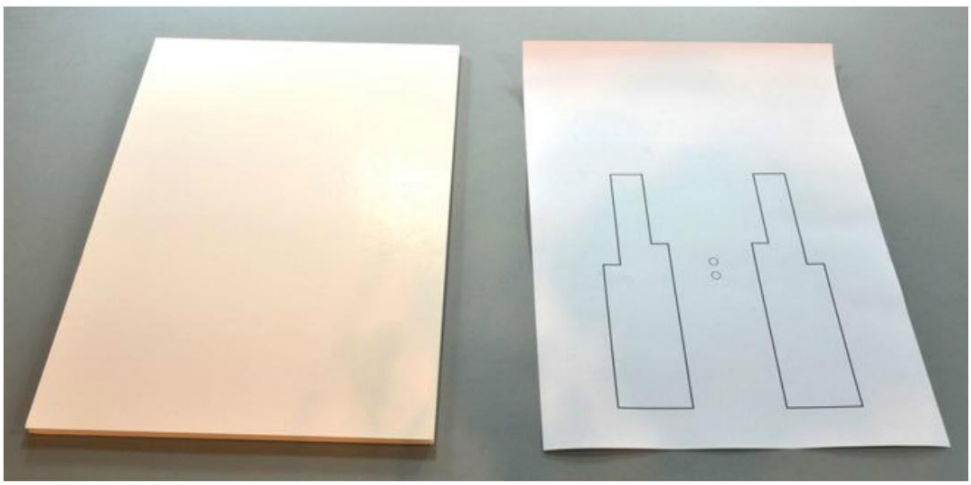


Figure 9-12. *Utilisation d'un gabarit pour le châssis*

Collez le papier sur le carton à l'aide de colle repositionnable afin de pouvoir retirer le gabarit avant de fixer les moteurs sur le châssis. Percez également deux petits trous pour les ficelles reliées à la tête et au corps de la marionnette (figure 9-13).

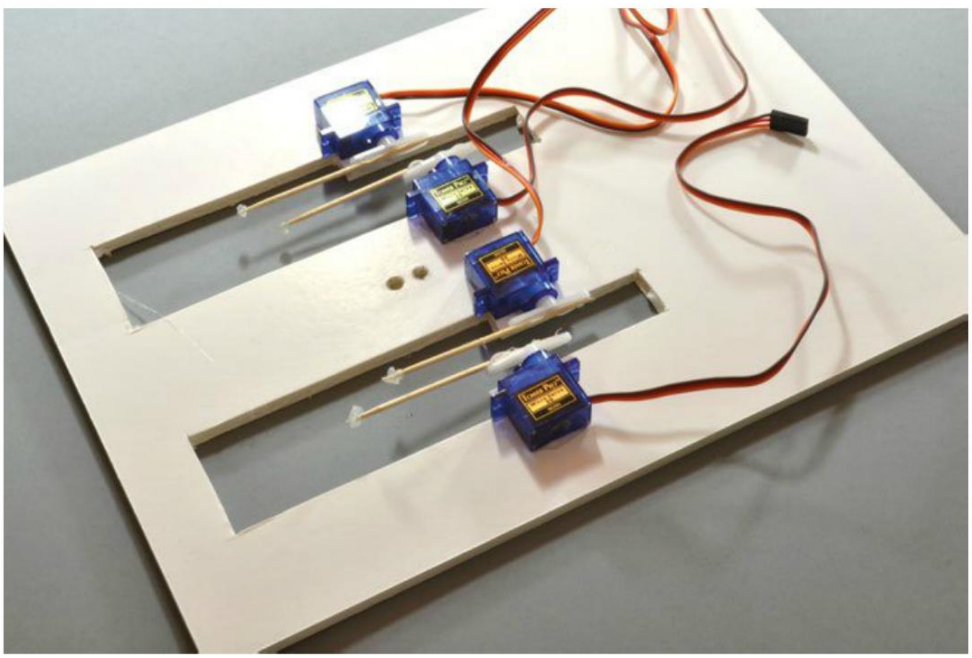


Figure 9-13. *Finition du châssis*

Étape 3 : collage des servos

Décollez le modèle en papier du carton du châssis, puis fixez les bras modifiés sur les servomoteurs. Ne les serrez pas trop, car vous devrez les démonter pour pouvoir positionner correctement les servos.

Alignez les servos comme illustré à la figure 9-14 de façon à ce que les bras puissent bouger librement sans se gêner.

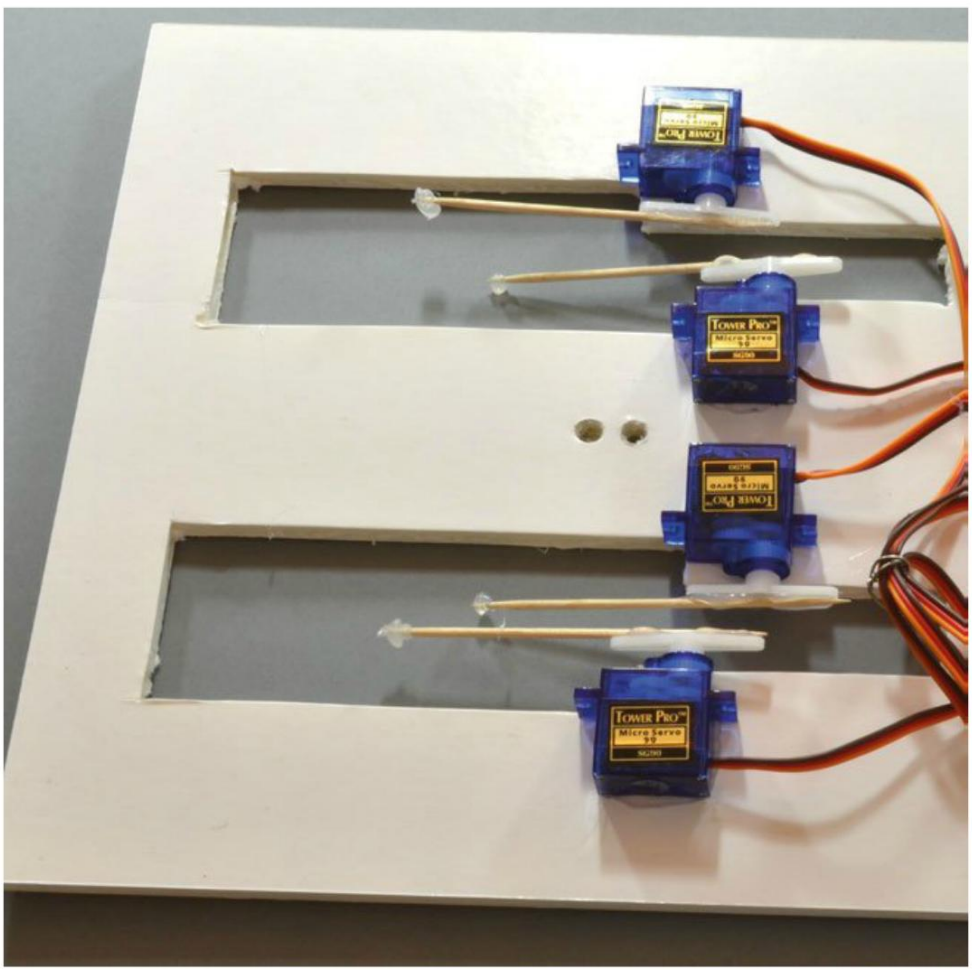


Figure 9-14. *Mise en place des servos*

Retirez la protection autocollante sur l'une des faces des servos avant de les mettre en place.

Étape 4 : préparation de la marionnette

La figure 9-15 présente la marionnette que j'ai utilisée.

Figure 9-15. *Pepe la marionnette*

La ficelle qui part de la tête supporte tout le poids de la marionnette. C'est cette ficelle qui sera passée dans les deux trous du châssis et nouée sur la face inférieure.

Chacune des jambes de la marionnette est reliée à une ficelle et les bras sont tous les deux reliés à la même ficelle qui passe par l'une des branches de la croix en bois. Cette ficelle doit être coupée en son milieu. Brûlez l'extrémité de la ficelle pour la faire fondre ou empêchez-la de s'effilocher par un point de colle.

Si vous le pouvez, défaites les nœuds à l'extrémité des ficelles reliées à la branche en bois. Sinon, coupez-les.

Avant de nouer la marionnette sur son nouveau châssis, vous devez d'abord connecter les servos et exécuter des programmes afin de positionner correctement les bras des servos.

Étape 5 : connexions

Le module Adafruit est vendu sous la forme d'un kit qui est assemblé en grande partie. Il ne vous reste plus qu'à souder les broches des connecteurs. Vous trouverez des instructions détaillées sur le page Adafruit de ce produit : *http://www.adafruit.com/products/815*.

J'ai fixé les broches des connecteurs du port I2C vers le Raspberry Pi du côté opposé à la carte du contrôleur servo afin de pouvoir la fixer proprement sur une plaque d'essai lorsque les composants électroniques du projet seront réutilisés au chapitre 15 pour faire parler Pepe.

Si vous voulez réduire la quantité de soudure requise, commencez par souder les broches des connecteurs des quatre premiers servos puisque ce projet n'en utilise pas d'autres.

Une fois que vous avez branché tous les fils en vous reportant à la figure 9-10, le projet ressemblera à la figure 9-16.

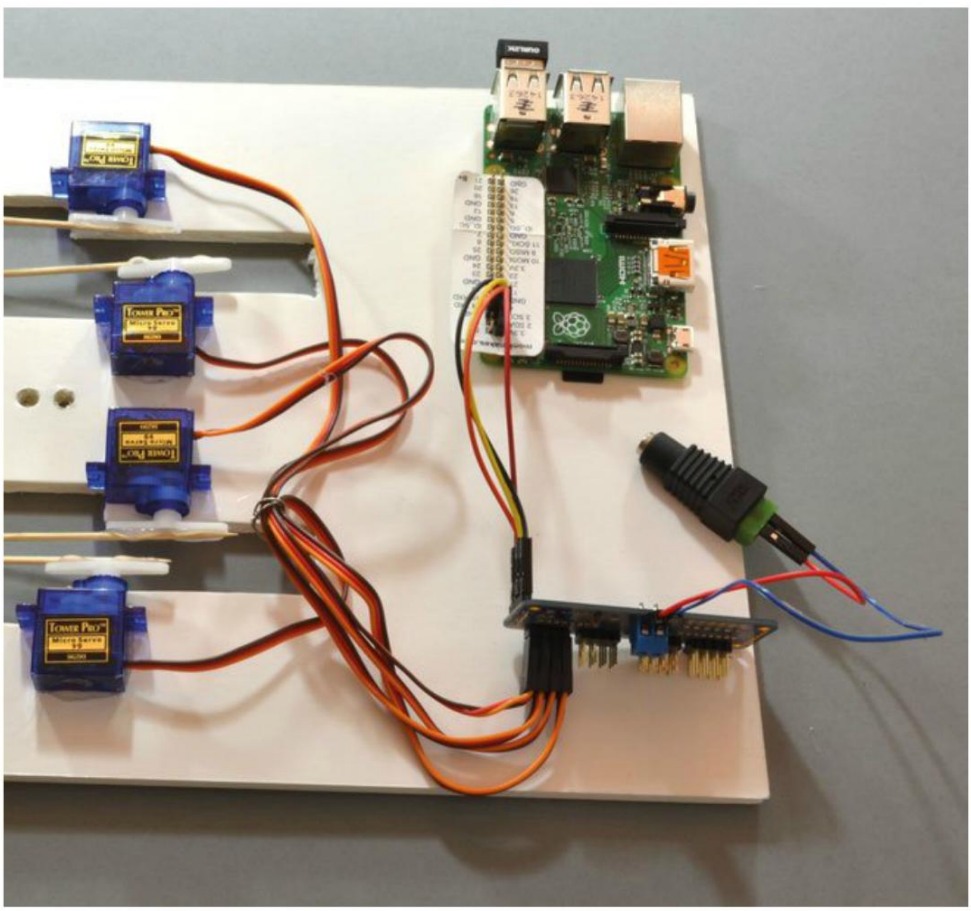


Figure 9-16. *Circuit complet du projet de la marionnette*

Vérifiez que les prises des servos sont dans le bon sens, avec le fil de commande orange en haut et le fil de masse marron ou noir en bas.

Vous pouvez alors raccorder l'alimentation électrique 5 V, ainsi que l'alimentation USB séparée du Raspberry Pi.

Étape 6 : exécution du programme test

Pour positionner correctement les servos, vous devez utiliser un programme séparé (`set_servos.py`) qui placera tous les bras des servos selon un certain angle. Ce programme et le programme principal de la marionnette se trouvent dans le dossier `/python/projets/puppet`.

Le module servo Adafruit utilise l'interface I2C du Raspberry Pi qui n'est pas activée par défaut.

Pour configurer le Pi pour I2C, reportez-vous à l'encadré ci-dessous.

Configuration du protocole I2C sur le Raspberry Pi

Commencez par vérifier que le gestionnaire de paquets est à jour à l'aide de la commande suivante :

```
$ sudo apt-get update
```

Ensuite, exécutez l'outil raspi-config :

```
$ sudo raspi-config
```

Dans le menu qui apparaît, choisissez Advanced puis I2C. Un message s'affiche pour vous demander si vous voulez activer l'interface I2C ARM. Répondez par yes. Un nouveau message s'affiche pour vous demander si vous voulez que le module noyau d'I2C soit chargé par défaut. Répondez encore par l'affirmative. Sélectionnez Finish pour quitter raspi-config.

Exécutez la commande suivante depuis le dossier racine pour installer quelques outils I2C :

```
sudo apt-get install python-smbus i2c-tools
```

Vérifiez que le Raspberry Pi est bien connecté au module servo Adafruit à l'aide de la commande suivante :

```
$ sudo i2cdetect -y 1
```

Vous devriez voir apparaître le tableau suivant (les numéros 40 et 70 indiquent que le module est connecté) :

```
$ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

Une fois que vous avez configuré le protocole I2C, retirez les bras des servos et exécutez le programme `set_servos.py`. À l'invite, saisissez un angle de 90 :

```
$ sudo python set_servos.py
Angle:90
Angle:
```

Les servos se mettent en mouvement. Ensuite, vous pouvez remettre les bras en place. Ils devraient être aussi proches de l'horizontale que le permettent les engrenages de l'arbre.

Étape 7 : fixation de la marionnette

Maintenant que tous les servos sont en position avec un angle de 90°, nouez la ficelle de la tête dans les orifices percés au milieu du châssis, puis nouez les ficelles de chacun des membres à un bras de servo de façon à ce que les bras et les jambes soient à moitié levés.

Une fois fait, vous pouvez de nouveau exécuter `set_servos.py` en saisissant différents angles pour vérifier l'amplitude de mouvements des membres.

Programme

Le programme de ce projet ne constitue qu'un point de départ. Il utilise différentes positions de servos que vous pouvez remplacer par des données personnalisées pour commander les mouvements de la marionnette.

Testez le programme avant d'examiner le code. Vous le trouverez dans le fichier `dance.py` du dossier `puppet/`. Pensez à l'exécuter en utilisant `sudo` :

```
from Adafruit_PWM_Servo_Driver import PWM  ❶
import time

pwm = PWM(0x40)

servoMin = 150 # Longueur d'impulsions mini sur 4 095  ❷
servoMax = 600 # Longueur d'impulsions maxi sur 4 095

dance = [          ❸
  #lh lf rf rh
  [90, 90, 90, 90],
  [130, 30, 30, 130],
  [30, 130, 130, 30]
]

delay = 0.2        ❹

def map(value, from_low, from_high, to_low, to_high):  ❺
  from_range = from_high - from_low
  to_range = to_high - to_low
  scale_factor = float(from_range) / float(to_range)
  return to_low + (value / scale_factor)

def set_angle(channel, angle):  ❻
  pulse = int(map(angle, 0, 180, servoMin, servoMax))
  pwm.setPWM(channel, 0, pulse)

def dance_step(step):     ❼
  set_angle(0, step[0])
  set_angle(1, step[1])
  set_angle(2, step[2])
  set_angle(3, step[3])

pwm.setPWMFreq(60)        ❽

while (True):             ❾
  for step in dance:
    dance_step(step)
    time.sleep(delay)
```

En plus du programme proprement dit (`dance.py`), le dossier contient aussi des fichiers Adafruit qui sont utilisés par le programme. Ils proviennent aussi de GitHub.

❶ Le module Adafruit ne sert pas uniquement aux servomoteurs ; ses sorties peuvent aussi être utilisées pour la commande PWM de LED ou d'autres sorties. C'est pourquoi le code importe une classe intitulée `PWM`.

❷ Les deux constantes `servoMin` et `servoMax` sont des longueurs d'impulsions comprises entre 0 et 4 095, où 4 095 correspond à un rapport cyclique de 100 %. Cette plage de valeurs devrait convenir à la plupart des servos, car elle correspond à une amplitude de près de 180°.

❸ Le bloc `dance` contient trois pas de danse. Ajoutez autant de ligne que vous le souhaitez. Chaque ligne compte quatre valeurs qui correspondent aux angles de la main gauche, du pied gauche, du pied droit et de la main droite, respectivement. Comme les servos des bras et des jambes sont montés en sens inverse, un angle supérieur à 90 lève un bras, mais descend une jambe.

❹ La variable `delay` définit la durée de la pause entre chaque pas de danse. Plus le nombre est petit, plus la marionnette bouge vite.

❺ La fonction `map` est expliquée en détail dans l'encadré « La fonction map Arduino » du chapitre 12 (page 257). Elle permet d'étendre la valeur d'angle à la valeur correcte de longueur d'impulsions qui est utilisée par la fonction `set_angle`.

❻ `set_angle` définit la position du canal servo (0 à 3) dans son premier paramètre selon un angle précisé dans le deuxième paramètre.

❼ La fonction `dance_step` reprend les angles des quatre servos des membres et règle chaque servo sur cet angle.

❽ Une fréquence PWM de 1/60^{e} s produit un train d'impulsions de 17 millisecondes, ce qui convient pour un servo.

❾ La boucle principale répète tous les pas de danse en réglant les angles définis, puis en marquant une pause avant de passer au pas suivant. Quand tous les pas ont été exécutés, la boucle recommence au début.

Utilisation de Pepe la marionnette

Modifiez le contenu du bloc `dance` afin d'ajouter des mouvements personnalisés. Essayez de faire marcher la marionnette, de lui faire agiter la main ou de la faire tenir sur un pied. En réglant une pause plus longue, vous aurez le temps de mieux voir les mouvements de la marionnette afin de corriger les pas.

Nous utiliserons à nouveau la marionnette au chapitre 15, où nous la ferons parler, et au chapitre 16, où elle dansera en réponse à des tweets.

Résumé

Les servomoteurs sont assez faciles à programmer et à relier à des éléments mécaniques.

Dans le chapitre suivant, nous présenterons un autre type de moteur : le moteur pas-à-pas.

Moteurs pas-à-pas

10

Si vous avez une imprimante ordinaire (ou même une imprimante 3D), elle contient probablement un ou plusieurs moteurs pas-à-pas. Celui illustré à la figure 10-1 est chargé d'amener le plastique à l'extrudeuse de l'imprimante 3D. Les moteurs pas-à-pas sont fréquemment utilisés dans les imprimantes parce que leurs mouvements sont extrêmement précis.

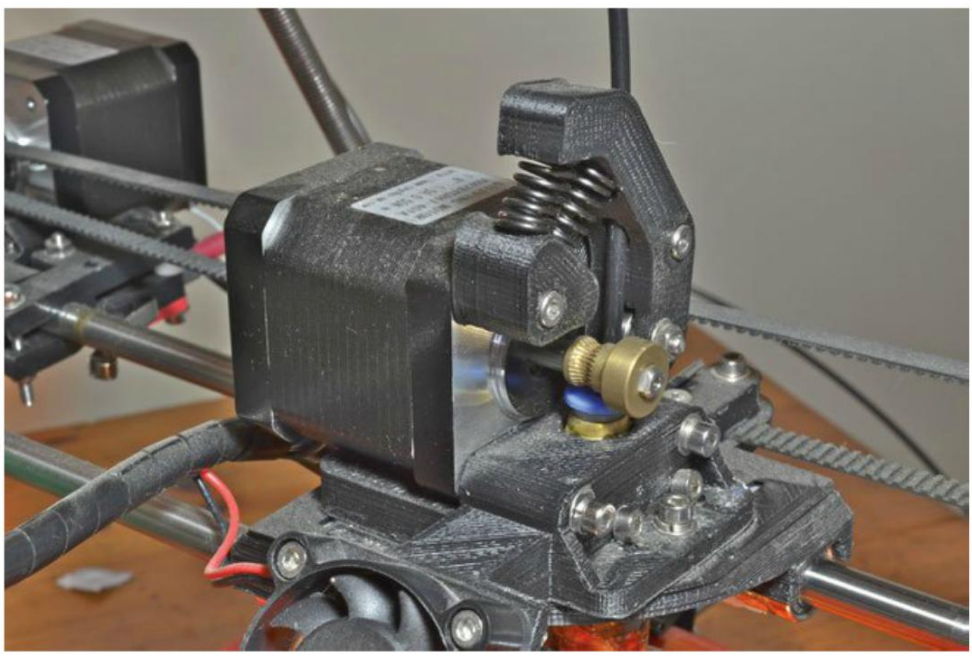


Figure 10-1. *Le moteur pas-à-pas d'une imprimante 3D*

Les techniques de régulation des moteurs pas-à-pas sont assez différentes de celles des moteurs à courant continu. Dans ce chapitre, nous utiliserons aussi bien des moteurs pas-à-pas unipolaires que bipolaires et nous examinerons les différents circuits de contrôleurs nécessaires à leur régulation.

Moteurs pas-à-pas

Comme leur nom l'indique, les moteurs pas-à-pas unipolaires tournent en faisant une série de petits pas. Cela présente l'avantage que lorsque le moteur tourne librement, il suffit de compter les pas pour savoir quelle distance a été parcourue. C'est l'une des raisons pour lesquelles ces moteurs sont utilisés dans les imprimantes, aussi bien 2D que 3D, pour positionner précisément le papier, la table ou les buses. La figure 10-2 montre trois types de moteurs pas-à-pas.

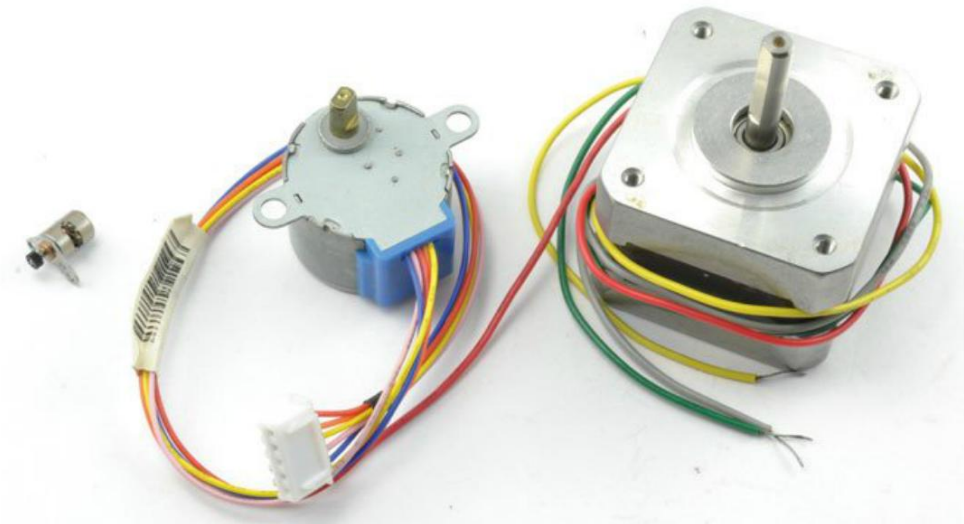


Figure 10-2. *Différents moteurs pas-à-pas*

Le petit moteur de gauche est utilisé pour déplacer les lentilles à l'intérieur de l'objectif d'un appareil photo compact ou d'un appareil photo de smartphone. Au centre, vous pouvez voir un moteur pas-à-pas 5 V à engrenages. Le modèle de droite est fréquemment employé dans les imprimantes.

Moteurs pas-à-pas bipolaires

La figure 10-3 illustre le fonctionnement des moteurs pas-à-pas et, plus précisément, celui des moteurs pas-à-pas bipolaires. Un peu plus loin dans ce chapitre, nous vous présenterons les moteurs pas-à-pas unipolaires.

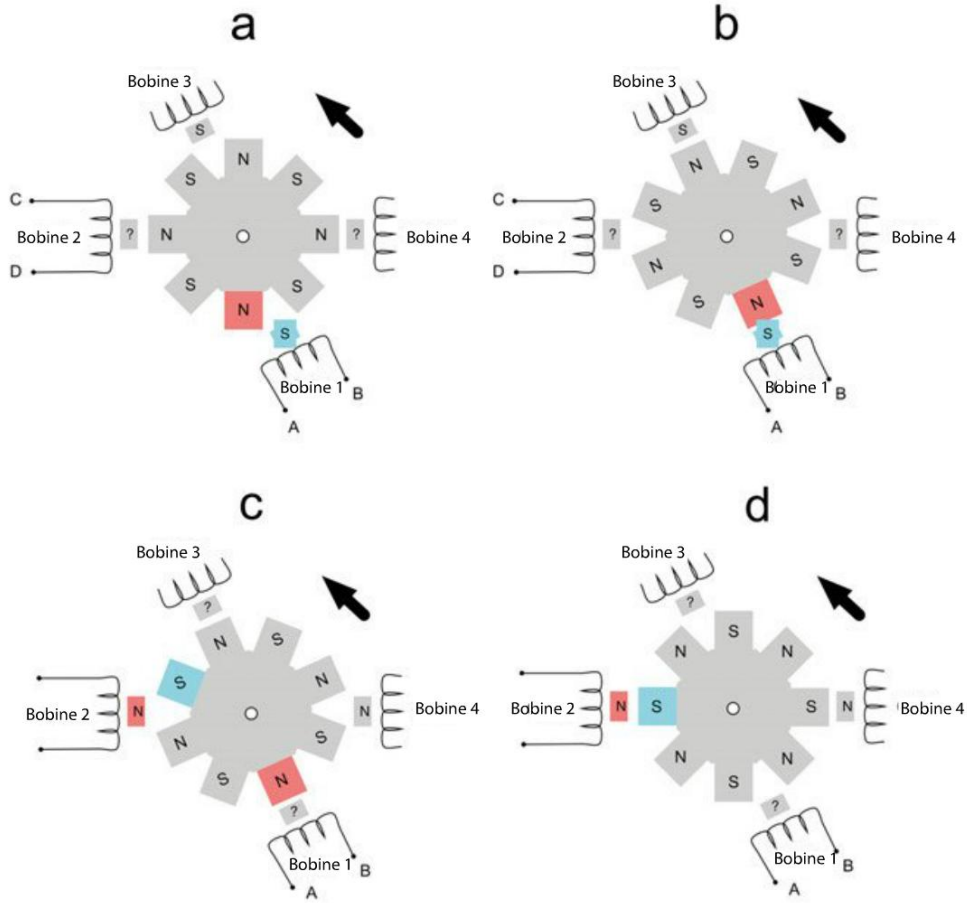


Figure 10-3. *Fonctionnement d'un moteur pas-à-pas bipolaire*

En règle générale, un moteur pas-à-pas contient quatre bobines, les bobines opposées étant reliées pour fonctionner à l'unisson. Les bobines sont placées sur la paroi du stator (à l'extérieur du moteur). Il n'y a donc pas besoin de collecteur et de balais comme dans un moteur CC.

Dans un moteur pas-à-pas, le rotor a la forme d'un rouage avec des pôles nord et sud disposés en alternance et repérés par les lettres N et S. Il y a généralement beaucoup plus d'encoches sur le rotor que celles illustrées à la figure 10-3. Chacune des bobines peut être excitée pour être un nord ou un sud magnétique en fonction du sens du courant qui la traverse. Les bobines 1 et 3 fonctionnent ensemble, donc si la bobine 1 est polarisée au nord, la bobine 3 le sera aussi. Il en va de même pour les bobines 2 et 4.

En commençant par la figure 10-3a, sachant que les opposés s'attirent et que les pôles identiques se repoussent, si la bobine 1 (et donc la bobine 3) est excitée pour être polarisée au sud, le rotor tournera dans le sens antihoraire jusqu'à ce que le sud de la bobine 1 soit aligné avec l'encoche

nord la plus proche, comme illustré à la figure 10-3b. Pour conserver la rotation horaire, la prochaine étape consiste à exciter les bobines 2 et 4 afin qu'elles soient polarisées au N (figure 10-3c), en attirant l'encoche S la plus proche de la bobine 2.

Tous ces efforts ont permis au moteur d'avancer d'un pas. Pour poursuivre la rotation antihoraire, la bobine 1 doit maintenant être excitée de façon à être polarisée au nord.

Le tableau 10-1 montre la séquence d'activation des bobines pour une rotation antihoraire du moteur.

Tableau 10-1. *Séquence pour une rotation antihoraire d'un moteur pas-à-pas*

Bobines 1 et 3	Bobines 2 et 4
S	-
-	N
N	-
-	S

Un tiret indique que la bobine n'a pas d'effet sur la rotation et n'a pas besoin d'être excitée. Dans ce cas d'un tiret, la polarité peut être placée de façon à être identique à l'encoche qui lui sera directement opposée, pour donner une poussée supplémentaire au moteur. On obtiendra alors le tableau révisé (tableau 10-2).

Tableau 10-2. *Séquence révisée d'activations pour la rotation d'un moteur pas-à-pas*

Bobines 1 et 3	Bobines 2 et 4
S	N
N	N
N	S
S	S

Vous vous demandez peut-être ce qu'il se serait passé si nous avions commencé par la bobine 2 au lieu de la bobine 1 ? Dans ce cas, l'excitation de l'autre bobine poussera l'encoche dans la direction voulue.

Pour inverser le sens de rotation du rotor, il suffit d'exciter les bobines dans le sens inverse à celui indiqué au tableau 10-2.

Expérience : pilotage d'un moteur pas-à-pas bipolaire

Nous allons réguler deux bobines (il s'agit en fait de deux paires, soit un total de quatre bobines) et nous devons inverser la direction du courant dans les deux bobines. Nous avons donc besoin de deux ponts en H. Un L293D paraît parfaitement indiqué pour ce rôle.

Dans cette expérience (figure 10-4), vous monterez un L293D sur la plaque d'essai pour contrôler un moteur pas-à-pas bipolaire à l'aide d'un Arduino et d'un Raspberry Pi.

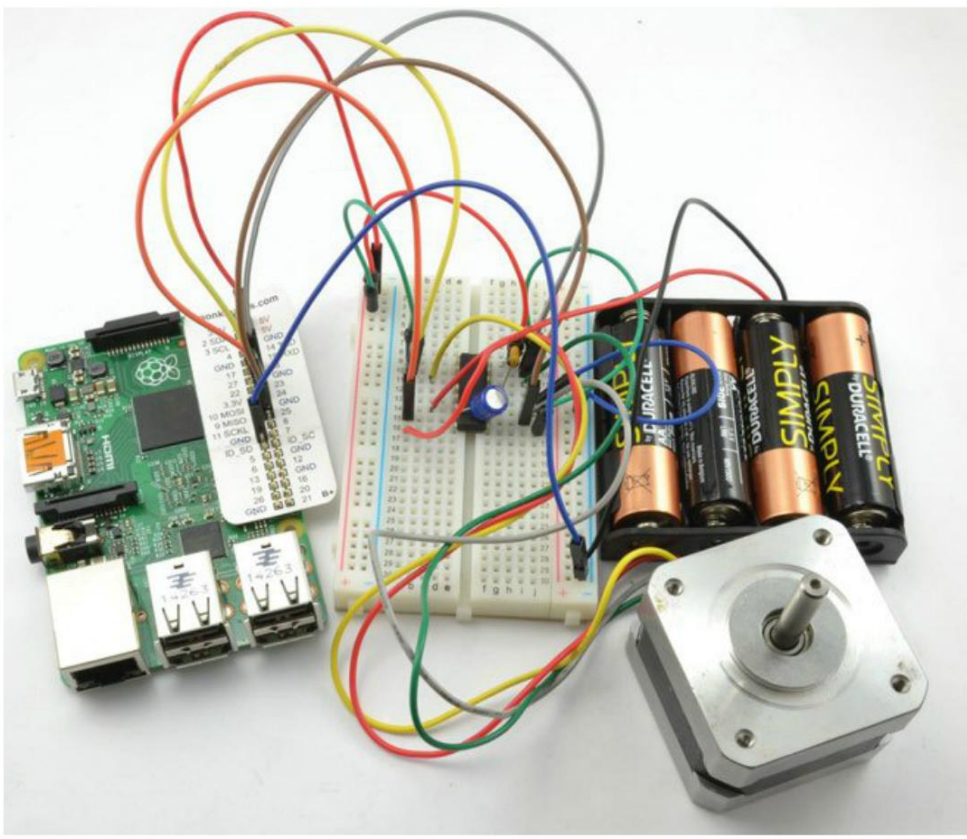


Figure 10-4. *Pilotage d'un moteur pas-à-pas bipolaire*

Bien qu'un moteur pas-à-pas 12 V soit utilisé dans cette expérience, cela peut aussi être réalisé avec une batterie 6 V pour l'alimentation électrique du moteur. Le couple sera inférieur, mais le moteur tournera parfaitement.

Identification des conducteurs d'un moteur pas-à-pas

Si vous venez d'acheter un nouveau moteur pas-à-pas, il devrait être fourni avec ses spécifications techniques ou être muni d'une étiquette permettant d'identifier le rôle de chacune de ses quatre broches. Il est particulièrement important d'identifier les paires de conducteurs qui sont reliées à la même bobine.

Pour identifier les paires, prenez deux fils que vous tenez entre le pouce et l'index tout en faisant tourner l'arbre du moteur. Si vous sentez une résistance, cela signifie que les deux forment une paire.

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou une carte Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser l'expérience.

Nom	Composant	Source
IC1	Circuit intégré de pont en H L293D	Adafruit : 807 Mouser : 511-L293D
C1	Condensateur 100 nF	Adafruit : 753 Mouser : 810-FK16X7R2A224K
C2	Condensateur 16 V 100 µF	Adafruit : 2193 Sparkfun : COM-00096 Mouser : 647-UST1C101MDD
M1	Moteur pas-à-pas bipolaire 12 V	Adafruit : 324
	Support pour piles (4 × AA) 6 V	Adafruit : 830
	Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
	Câbles flexibles mâles/mâles	Adafruit : 758
	Câbles flexibles femelles/mâles (Raspberry Pi uniquement)	Adafruit : 826

Si vous comptez tenter cette expérience avec un Raspberry Pi, vous aurez besoin de câbles flexibles femelles/mâles pour connecter les broches GPIO du Raspberry Pi à la plaque d'essai.

Montage

La figure 10-5 présente le schéma du circuit de cette expérience.

Comme nous n'utiliserons pas la MLI, les deux broches Enable du L293D sont connectées à 5 V de façon à ce que les deux ponts en H soient activés en permanence. Le moteur pas-à-pas est régulé par les quatre connexions In 1, In 2, In 3 et In 4 qui sont pilotées par les sorties numériques de la carte Arduino ou du Raspberry Pi.

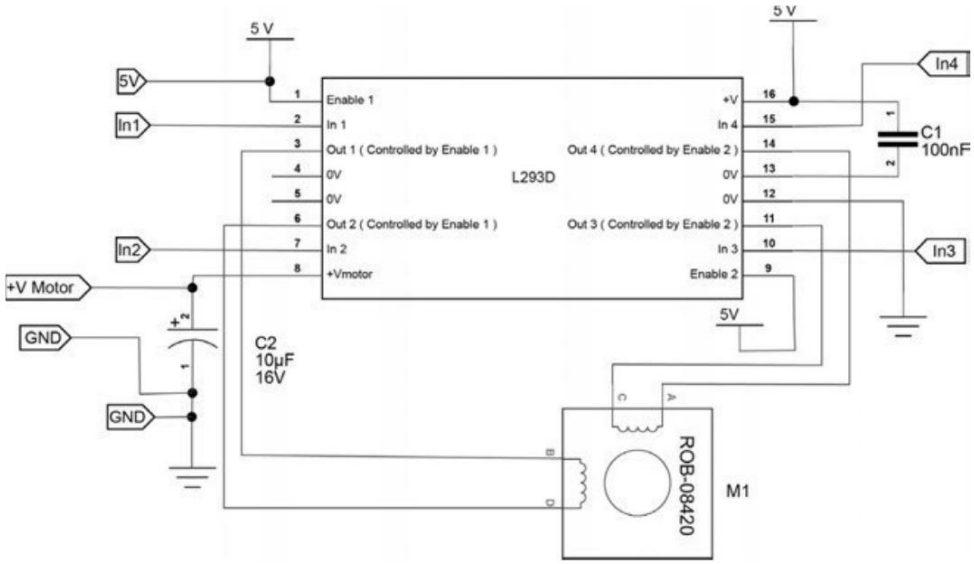


Figure 10-5. *Schéma du circuit de régulation d'un moteur pas-à-pas bipolaire*

Arduino

La version Arduino de cette expérience utilise le moniteur série pour transmettre des commandes à la carte Arduino afin de réguler le moteur. Les trois commandes sont représentées par une lettre suivie d'un nombre. Par exemple :

- `f100` fait tourner le moteur en avant de 100 pas ;
- `r100` fait tourner le moteur en sens inverse de 100 pas ;
- `p10` définit 10 millisecondes de pause entre les impulsions de chaque pas.

La figure 10-6 présente la version Arduino de l'expérience.

Montage Arduino

Dans la version Arduino de cette expérience, les broches suivantes sont connectées au L293D :

Nom de la broche L293D	Numéro de broche L293	Broche Arduino
In1	2	10
In2	7	9
In3	10	11
In4	15	8

La figure 10-7 montre la réalisation du circuit avec un Arduino.

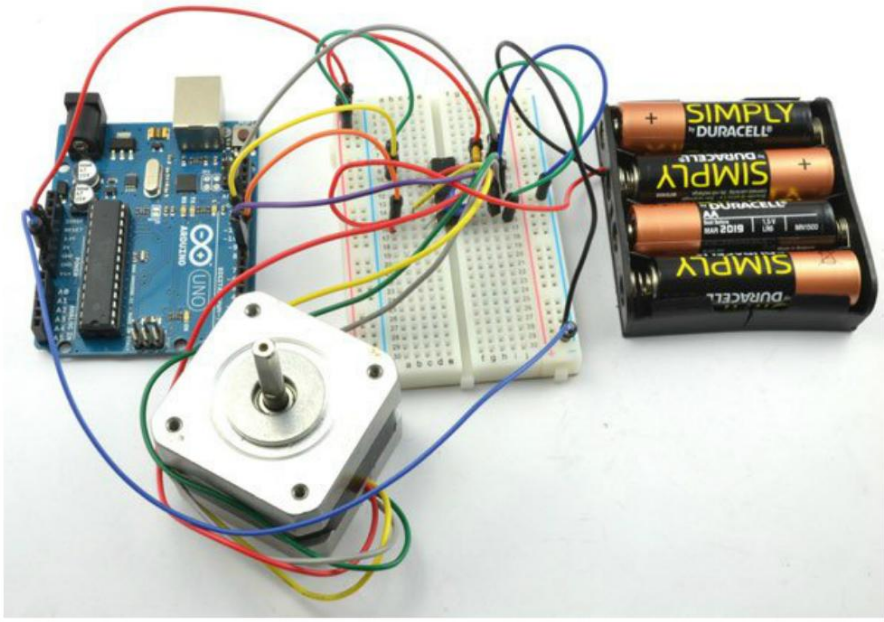


Figure 10-6. *Pilotage Arduino du moteur pas-à-pas*

Vérifiez que le circuit intégré est correctement positionné (encoche vers le haut) et que la borne positive de C2, le condensateur de 100 µF, est connectée à la broche 8 du L293D. En général, le fil positif d'un condensateur électrolytique, comme C2, est plus long que le fil négatif. Il est probable que le conducteur négatif de C2 soit repéré sur le boîtier du condensateur par un signe moins ou une étoile.

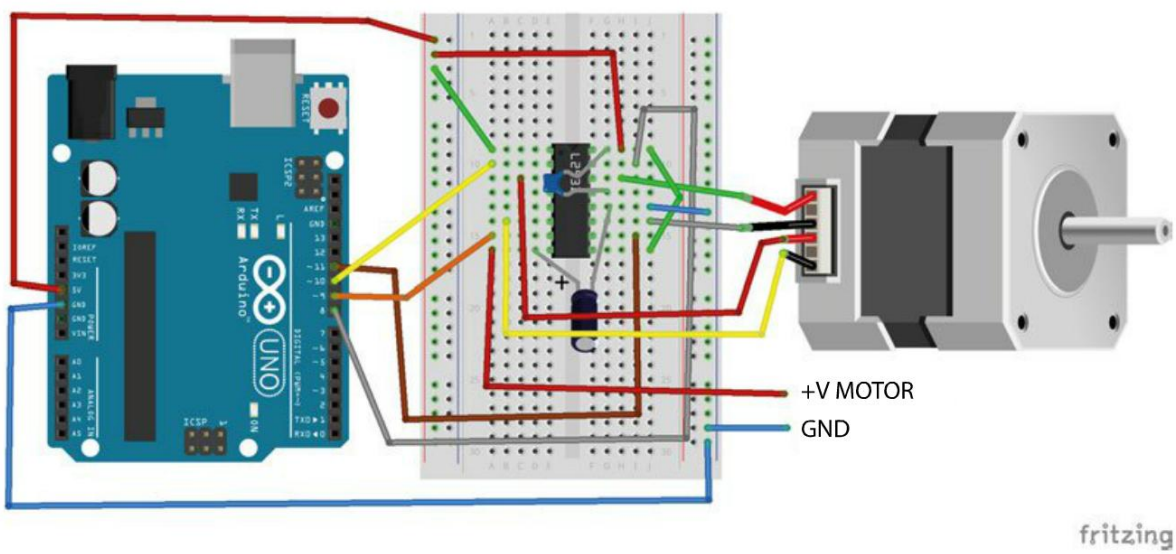


Figure 10-7. *Réalisation du circuit d'un moteur pas-à-pas avec un Arduino*

Programme Arduino (version compliquée)

Il existe deux versions du programme Arduino pour cette expérience. La première est plus compliquée : les broches de commande du L293D sont définies comme décrit à la section « Moteurs pas-à-pas bipolaires » page 178. Cet exercice permet de se familiariser avec le fonctionnement d'un moteur pas-à-pas.

Le second sketch utilise la bibliothèque Arduino Stepper et il est donc beaucoup plus court.

Le premier sketch Arduino se trouve dans `arduino/experiments/bi_stepper_no_lib` :

```
const int in1Pin = 10;          ❶
const int in2Pin = 9;
const int in3Pin = 11;
const int in4Pin = 8;

int period = 20;                ❷

void setup() {                  ❸
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  pinMode(in3Pin, OUTPUT);
  pinMode(in4Pin, OUTPUT);
  Serial.begin(9600);
  Serial.println(«lettre suivie d'un nombre»);
  Serial.println(«p20 - définit une pause de 20 ms entre les pas (pilotage de la vitesse)»);
  Serial.println(«f100 - en avant de 100 pas»);
  Serial.println(«r100 - en arrière de 100 pas»);
}

void loop() {                   ❹
  if (Serial.available()) {
    char command = Serial.read();
    int param = Serial.parseInt();
    if (command == 'p') {                ❺
      period = param;
    }
    else if (command == 'f') {           ❻
      stepForward(param, period);
    }
    else if (command == 'r') {
      stepReverse(param, period);
    }
  }
  setCoils(0, 0, 0, 0); // power down
}

void stepForward(int steps, int period) {     ❼
  for (int i = 0; i < steps; i++) {
    singleStepForward(period);
  }
}
```

```
void singleStepForward(int period) {          ❽
  setCoils(1, 0, 0, 1);
  delay(period);
  setCoils(1, 0, 1, 0);
  delay(period);
  setCoils(0, 1, 1, 0);
  delay(period);
  setCoils(0, 1, 0, 1);
  delay(period);
}

void stepReverse(int steps, int period) {
  for (int i = 0; i < steps; i++) {
    singleStepReverse(period);
  }
}

void singleStepReverse(int period) {          ❾
  setCoils(0, 1, 0, 1);
  delay(period);
  setCoils(0, 1, 1, 0);
  delay(period);
  setCoils(1, 0, 1, 0);
  delay(period);
  setCoils(1, 0, 0, 1);
  delay(period);
}

void setCoils(int in1, int in2, int in3, int in4) {  ❿
  digitalWrite(in1Pin, in1);
  digitalWrite(in2Pin, in2);
  digitalWrite(in3Pin, in3);
  digitalWrite(in4Pin, in4);
}
```

❶ Le code commence par définir les broches qui seront utilisées pour piloter le moteur. Rien ne vous empêche d'en changer tant que vous modifiez la plaque d'essai en conséquence.

❷ La variable `period` correspond au délai entre chaque activation de bobine pour un pas de rotation du moteur. Le délai initial est de 20 millisecondes. Vous pouvez corriger cette valeur dans le moniteur série pour accélérer ou ralentir le moteur.

❸ La fonction `setup()` définit les broches de sortie numériques comme broches de commande. Elle démarre la communication série avec le moniteur série, puis affiche un message expliquant le format des commandes que vous pouvez transmettre.

❹ La fonction `loop()` attend que des commandes arrivent par la liaison série, puis elle les traite. Si une commande a été reçue (ce qui est indiqué par `Serial.available`), la fonction `loop()` lit d'abord la lettre de la commande dans la variable `command`, puis elle lit le paramètre sous la forme d'un nombre qui la suit.

Ensuite, une série d'instructions `if` définit l'action appropriée en fonction de la commande.

❺ Si la commande est `'p'`, la variable `period` est définie en fonction du nombre fourni en tant que paramètre.

❻ En revanche, si la commande est `'f'` ou `'r'`, `stepForward` ou `stepReverse` est appelé, le cas échéant, accompagné comme paramètres par le nombre de pas à effectuer et le délai entre chaque changement de polarité.

❼ `stepForward` et `stepReverse` sont très proches et appellent `singleStepForward` ou `singleStepReverse` autant de fois que défini dans `steps`.

❽ Ensuite, nous arrivons à la fonction `singleStepForward` qui contient le motif des polarités requis pour les quatre phases pour faire avancer le moteur d'un pas. Le motif apparaît dans les paramètres de `setCoils`.

❾ `singleStepReverse` est identique à `singleStepForward`, mais la séquence est inversée. Comparez les pas avec ceux énumérés dans le tableau 10-2.

❿ Enfin, la fonction `setCoils` définit les broches de commande en fonction du motif fourni par les paramètres.

Programme Arduino (version simplifiée)

L'IDE Arduino comprend la bibliothèque Stepper qui fonctionne très bien et qui permet de réduire considérablement la taille du sketch. Vous trouverez ce sketch dans le dossier `arduino/experiments/ex_07_bi_stepper_lib/` à l'endroit où vous avez téléchargé le code du livre (voir la section « Téléchargement des programmes » du chapitre 4 page 45).

```
#include <Stepper.h>                    ❶

const int in1Pin = 10;
const int in2Pin = 9;
const int in3Pin = 8;
const int in4Pin = 11;

Stepper motor(200, in1Pin, in2Pin, in3Pin, in4Pin);  ❷

void setup() {                          ❸
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  pinMode(in3Pin, OUTPUT);
  pinMode(in4Pin, OUTPUT);
  while (!Serial);
  Serial.begin(9600);
  Serial.println(«lettre suivie d'un nombre»);
  Serial.println(«p20 - définit une pause de 20 ms entre les pas (pilotage de la vitesse)»);
  Serial.println(«f100 - en avant de 100 pas»);
  Serial.println(«r100 - en arrière de 100 pas»);
  motor.setSpeed(20);                   ❹
}
```

```
void loop() {      ❺
  if (Serial.available()) {
  char command = Serial.read();
  int param = Serial.parseInt();
  if (command == 'p') {
    motor.setSpeed(param);
  }
  else if (command == 'f') {
    motor.step(param);
  }
  else if (command == 'r') {
    motor.step(-param);
    }
  }
}
```

❶ La première ligne du sketch importe la bibliothèque Stepper qui est fournie avec l'IDE Arduino et qui n'a pas besoin d'être préalablement installée.

❷ Pour utiliser la bibliothèque, une variable `motor` est définie comme étant de type `Stepper`. Les paramètres fournis à `motor` définissent l'utilisation du moteur. Le premier paramètre correspond au nombre de pas par tour. Pour le moteur pas-à-pas Adafruit que j'utilise, cela correspond à 200 pas, ce qui est un nombre répandu. En fait, cela signifie 200 changements de phases par révolution, soit 50 positions effectives de l'arbre par tour. Les quatre autres paramètres sont les broches des bobines.

❸ La fonction `setup()` est presque identique à celle de la version longue du sketch. Le message est légèrement différent, car ici, la commande `'p'` définit la vitesse du moteur en rpm (*revolutions per minute*, tours par minute) à la place du temps de pause. Toutefois, ils ont tous les deux pour effet de changer la vitesse du moteur.

❹ La vitesse par défaut est aussi définie sur 20 tr/min dans `setup()` à l'aide de `motor.setSpeed`.

❺ La fonction `loop()` est presque identique à celle de la version longue du sketch ; toutefois, ici, le nombre de pas et la direction du moteur sont définis sous la forme d'un nombre positif pour la marche avant et un nombre négatif pour la marche arrière.

Le nombre de quarts de pas définis dans le paramètre `'f'` ou `'r'` correspond ici au nombre de changements de phases. Donc pour un tour complet du moteur Adafruit, vous devrez saisir la valeur 200.

Expérimentation Arduino

Je vous recommande d'essayer les deux versions du code. Ici, je supposerai que vous avez téléversé `ex_05_bi_stepper_lib` sur votre carte Arduino. À l'ouverture du moniteur série, vous devriez être accueilli par le message illustré à la figure 10-8.

Figure 10-8. *Pilotage du moteur pas-à-pas dans le moniteur série*

Saisissez la commande `f200`, puis cliquez sur Envoyer. Le moteur doit faire un tour complet. S'il ne tourne pas, mais qu'il bourdonne, cela signifie probablement que vous avez mal positionné l'un des fils des bobines. Intervertissez les fils connectés aux broches 3 et 6 du L293D et saisissez de nouveau la commande.

Si vous saisissez `r200`, le moteur doit faire un tour complet dans la direction inverse.

Ensuite, essayez de notifier la vitesse à l'aide de la commande `p` suivie de la vitesse en tr/min. Saisissez, par exemple, `p5` suivi de `f200`. Le moteur fera un tour complet, mais très lentement. Vous pouvez augmenter la vitesse en saisissant une valeur plus élevée, telle que `p100`, mais vous constaterez qu'il y a une vitesse maximale de l'ordre de 130 tr/min au-delà de laquelle le moteur ne fait pas un tour complet.

Raspberry Pi

Le pilotage d'un moteur pas-à-pas à l'aide de Python ou d'un Raspberry Pi est très proche de la méthode Arduino qui ne fait pas appel à la bibliothèque. La commande du L293D utilise quatre sorties de commande du Raspberry Pi.

Le programme Python vous invite à saisir le délai entre les phases, la direction et le nombre de pas.

Montage Raspberry Pi

Vous pouvez conserver le même circuit réalisé sur la plaque d'essai avec l'Arduino, mais les broches des connecteurs du Raspberry Pi nécessitent des câbles flexibles femelles/mâles au lieu des câbles mâles/mâles. La figure 10-9 présente le circuit de connexion de la plaque d'essai à un Raspberry Pi.

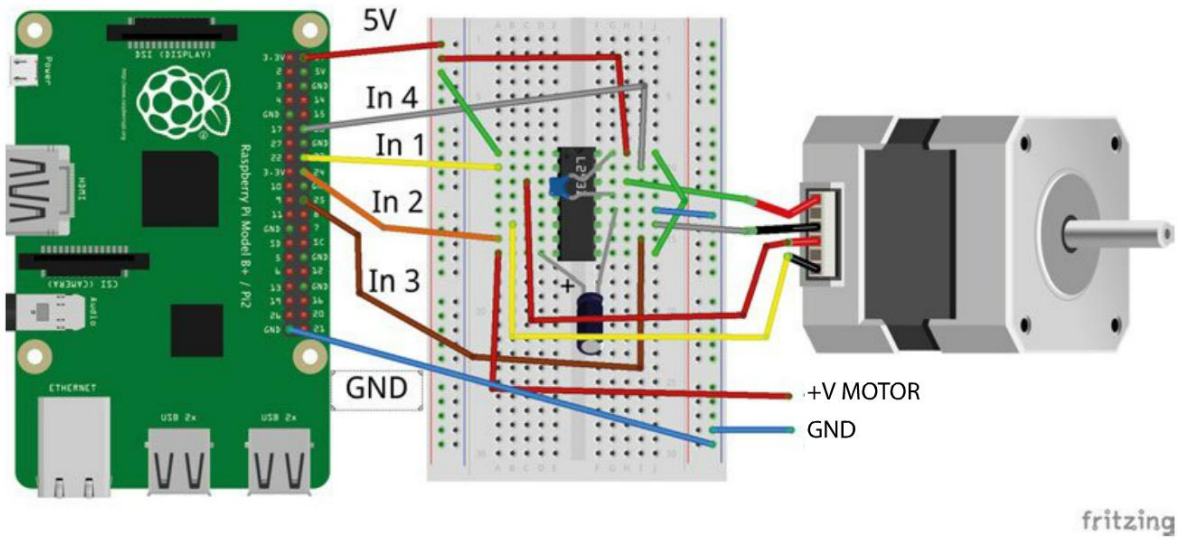


Figure 10-9. *Réalisation du circuit Raspberry Pi pour un moteur pas-à-pas*

Programme Raspberry Pi

La version Raspberry Pi du code est presque une traduction mot-à-mot du code Arduino qui ne fait pas appel à la bibliothèque. Ce programme qui se nomme `bi_stepper.py` se trouve dans le dossier `python/experiments` (à l'endroit où vous avez téléchargé le code du livre) :

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

in_1_pin = 23          ❶
in_2_pin = 24
in_3_pin = 25
in_4_pin = 18

GPIO.setup(in_1_pin, GPIO.OUT)
GPIO.setup(in_2_pin, GPIO.OUT)
GPIO.setup(in_3_pin, GPIO.OUT)
GPIO.setup(in_4_pin, GPIO.OUT)

period = 0.02

def step_forward(steps, period):        ❷
  for i in range(0, steps):
    set_coils(1, 0, 0, 1)
    time.sleep(period)
    set_coils(1, 0, 1, 0)
    time.sleep(period)
    set_coils(0, 1, 1, 0)
```

```
    time.sleep(period)
    set_coils(0, 1, 0, 1)
    time.sleep(period)

def step_reverse(steps, period):
  for i in range(0, steps):
    set_coils(0, 1, 0, 1)
  time.sleep(period)
  set_coils(0, 1, 1, 0)
  time.sleep(period)
  set_coils(1, 0, 1, 0)
  time.sleep(period)
  set_coils(1, 0, 0, 1)
  time.sleep(period)

def set_coils(in1, in2, in3, in4):    ❸
  GPIO.output(in_1_pin, in1)
  GPIO.output(in_2_pin, in2)
  GPIO.output(in_3_pin, in3)
  GPIO.output(in_4_pin, in4)

try:
  print('Lettre suivie d'un nombre');
  print('p20 - définit une pause de 20 ms entre les pas (pilotage de la vitesse)');
  print('f100 - en avant de 100 pas');
  print('r100 - en arrière de 100 pas');

  while True:          ❹
    command = raw_input('Saisir la commande : ')
    parameter_str = command[1:] # from char 1 to end
    parameter = int(parameter_str)       ❺
    if command[0] == 'p':        ❻
      period = parameter / 1000.0
    elif command[0] == 'f':
      step_forward(parameter, period)
    elif command[0] == 'r':
      step_reverse(parameter, period)

finally:
  print('Nettoyage')
  GPIO.cleanup()
```

❶ Le programme définit des constantes pour les quatre broches de commande et les configure comme sorties.

❷ Les fonctions `step_forward` et `step_reverse` sont identiques à leurs équivalents en Arduino. Les quatre bobines sont excitées dans l'ordre établi à l'aide de la fonction `set_coils`. Cette action est répétée autant de fois que défini par `steps` avec un délai défini par `period` entre chaque changement d'activation des bobines.

❸ `set_coils` définit les broches de commande des sorties numériques en fonction de ses quatre paramètres.

❹ La boucle principale du programme lit la chaîne de caractères de la commande à l'aide de `raw_input`. Le paramètre qui suit la lettre est d'abord coupé de la commande à l'aide de la syntaxe `[1:]` qui, dans une chaîne Python, désigne la chaîne de la position 1 (le second caractère) à la fin de la chaîne.

❺ Ce paramètre de chaîne est ensuite converti en nombre à l'aide de la fonction intégrée `int`.

❻ Une série de trois instructions `if` exécute l'action demandée par la commande en changeant la valeur de la variable `period` ou en faisant tourner le moteur dans un sens ou dans l'autre.

Expérimentation Raspberry Pi

Exécutez le programme Python à l'aide de la commande `sudo python ex_07_bi_stepper.py`. Un message s'affiche pour vous inviter à saisir les commandes possibles. Elles sont identiques à celles de la version Arduino.

Essayez de faire tourner le moteur dans les deux directions et déterminez la valeur de pause la plus basse pouvant être réglée sans que le moteur ne rate de pas :

```
$ sudo python ex_05_bi_stepper.py
Lettre suivie d'un nombre
p20 - définit une pause de 20 ms entre les pas (pilotage de la vitesse)
f100 - en avant de 100 pas
r100 - en arrière de 100 pas
Saisir la commande: p5
Saisir la commande: f50
Saisir la commande: p10
Saisir la commande: r100
Saisir la commande:
Enter command: f50
Enter command: p10
Enter command: r100
Enter command:
```

Moteurs pas-à-pas unipolaires

Le fonctionnement des moteurs pas-à-pas unipolaires est très proche de celui des moteurs bipolaires, à la différence qu'ils ne nécessitent pas de ponts en H en raison de la disposition de leurs bobines. La figure 10-10 illustre le fonctionnement d'un moteur pas-à-pas unipolaire.

Un moteur pas-à-pas unipolaire a cinq conducteurs, au lieu des quatre d'un moteur bipolaire. Quatre des conducteurs sont identiques à ceux d'un moteur bipolaire ; ce sont simplement les extrémités des bobines, A, B, C et D. D'ailleurs, il y a une autre paire de bobines, qui n'est pas illustrée à la figure 10-10, comme dans un moteur bipolaire. Si vous le souhaitez, vous pouvez utiliser les conducteurs A, B, C et D avec un pont en H, comme si c'était un moteur pas-à-pas bipolaire.

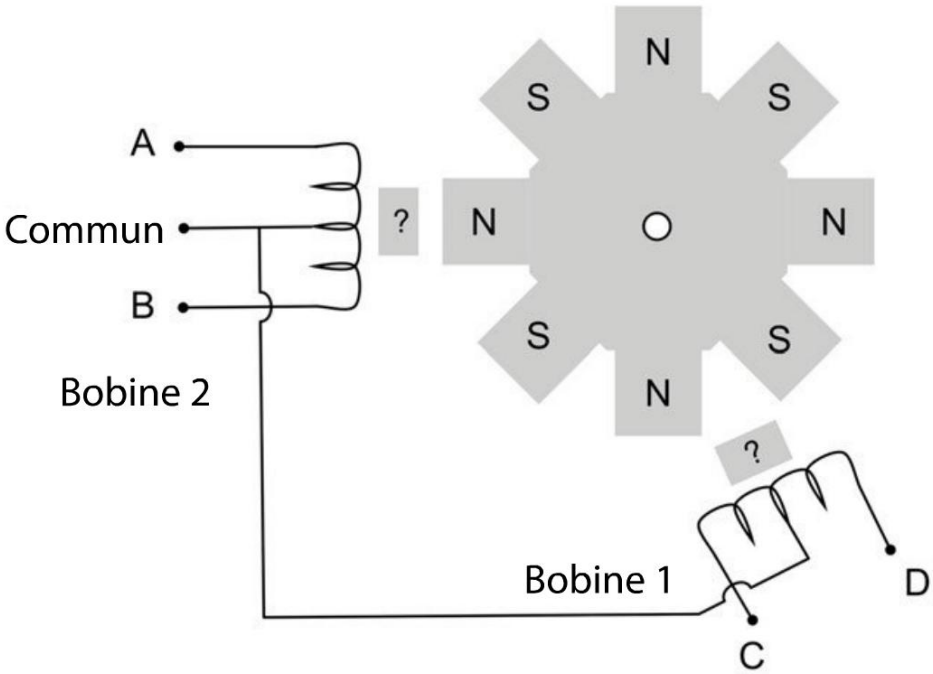


Figure 10-10. *Un moteur pas-à-pas unipolaire*

Le cinquième conducteur est raccordé au point milieu de chacune des bobines. Si vous imaginez que ce conducteur est relié à la masse, vous pouvez polariser la bobine au nord en l'alimentant sur A ou au sud en l'alimentant sur B. Inutile de prévoir un pont en H.

Réseaux de transistors Darlington

Bien que vous n'ayez pas besoin d'un pont en H avec un moteur pas-à-pas unipolaire, les bobines du moteur nécessitent toutefois trop de courant pour être pilotées directement à partir des broches d'un Arduino ou d'un Raspberry Pi.

La méthode la plus évidente pour booster le courant est d'utiliser des transistors, comme vous l'avez fait à l'expérience « Pilotage d'un moteur » au chapitre 4. Vous aurez bien évidemment besoin de quatre transistors, un pour chaque connexion des bobines A, B, C et D. Vous avez aussi besoin de résistances de limitation du courant connectées aux bases de ces transistors, ainsi que des diodes de protection sur chaque bobine. La figure 10-11 montre le schéma de raccordement utilisé pour connecter chaque bobine d'un moteur pas-à-pas unipolaire. Ce branchement devra être répété quatre fois pour un système complet.

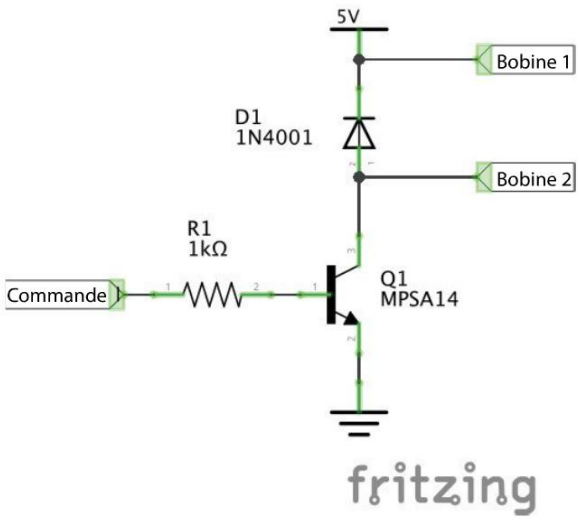


Figure 10-11. *Régulation d'un moteur pas-à-pas unipolaire à l'aide de transistors séparés*

Ce câblage peut être réalisé sur la plaque d'essai, mais il y aura beaucoup de fils à brancher.

Il est préférable d'utiliser un contrôleur, comme l'ULN2803. Ce circuit intégré bon marché réunit huit transistors Darlington dans un même boîtier et il intègre aussi une diode de protection et des résistances de base. Vous n'avez donc plus qu'à y ajouter un condensateur de filtrage.

L'ULN2803 peut fournir jusqu'à 500 mA par canal et une tension maximale de 50 V.

Dans l'expérience suivante « Pilotage d'un moteur pas-à-pas unipolaire », vous utiliserez l'un de ces composants pour contrôler un moteur pas-à-pas unipolaire à l'aide d'un Arduino et d'un Raspberry Pi.

Expérience : pilotage d'un moteur pas-à-pas unipolaire

La figure 10-12 montre une carte Arduino pilotant un moteur pas-à-pas unipolaire. Le moteur utilisé est un autre type de moteur pas-à-pas très répandu. Il inclut un réducteur à engrenages 1:16. Par conséquent, même si le moteur lui-même n'a que 32 pas, avec le réducteur, il faut 513 changements de phases par tour, soit 128 pas par tour.

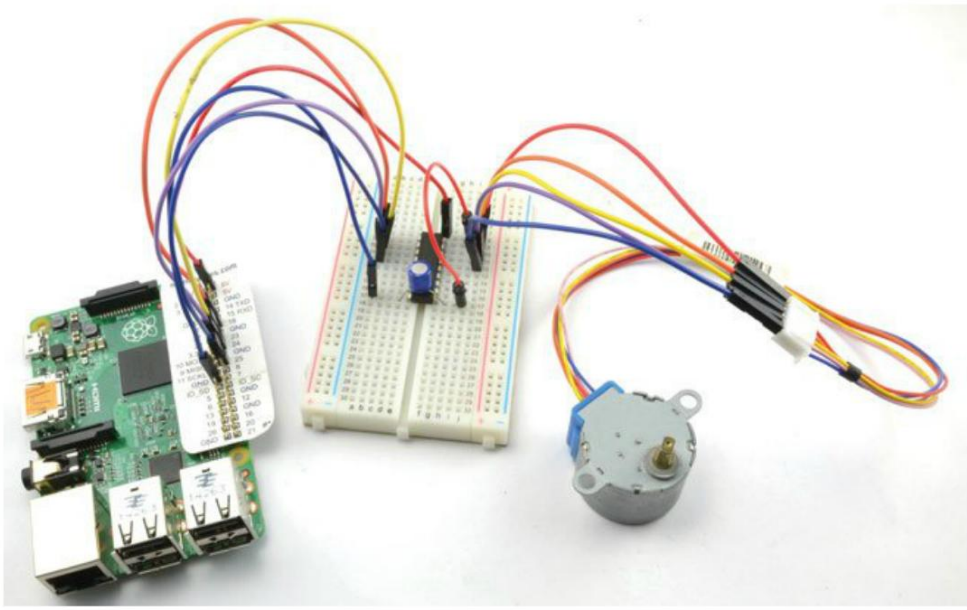


Figure 10-12. *Pilotage d'un moteur pas-à-pas unipolaire avec un Raspberry Pi*

Matériel

La figure 10-13 présente le circuit de la figure 10-11 réalisé à l'aide d'un ULN2003. Notez qu'il y a toujours quatre transistors Darlington inutilisés qui pourraient être employés pour piloter un second moteur pas-à-pas. Pour les moteurs pas-à-pas à courant élevé, vous pouvez même doubler les sorties en connectant ensemble deux entrées correspondantes aux deux sorties.

Le moteur utilisé ici est un modèle 5 V dont le courant est assez faible pour être alimenté par le Raspberry Pi ou la carte Arduino. Il consomme environ 150 mA lorsque deux bobines sont excitées. Si votre moteur est plus gourmand ou s'il exige une tension supérieure, vous devrez utiliser une alimentation séparée, comme pour le moteur pas-à-pas bipolaire.

Pour trouver quels conducteurs sont connectés à quels éléments du moteur, vous pouvez procéder de la même façon que pour les moteurs bipolaires (voir l'encadré « Identification des conducteurs d'un moteur pas-à-pas », plus haut page 182), sachant qu'il y a un conducteur commun. Par conséquent, commencez par identifier le conducteur commun qui résiste au mouvement de rotation lorsqu'il est connecté à un autre conducteur. Ensuite, pour identifier les autres conducteurs, utilisez la même méthode que pour un moteur bipolaire.

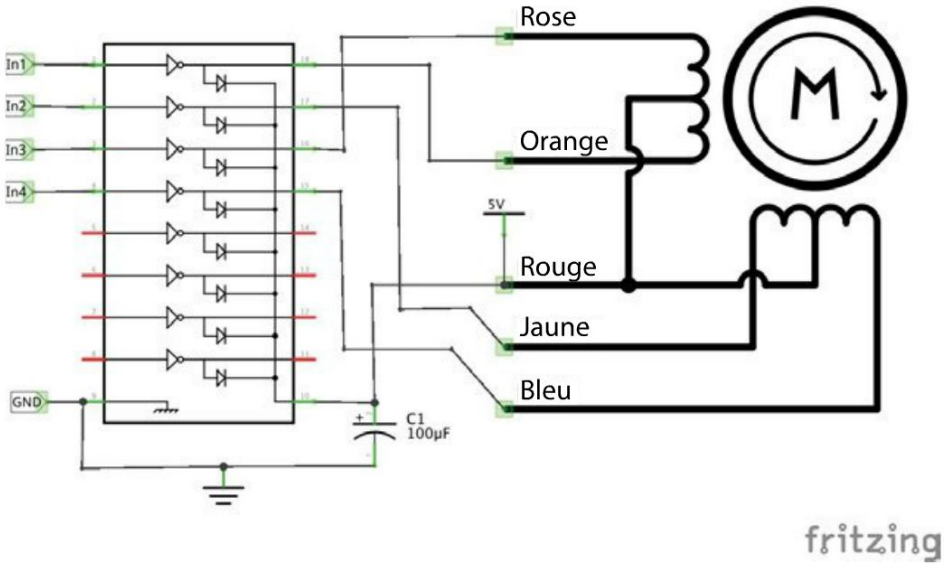


Figure 10-13. *Schéma de pilotage d'un moteur unipolaire (la couleur des conducteurs peut varier)*

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou une carte Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser l'expérience.

Nom	Composant	Source
IC1	ULN2803	Adafruit : 970 Mouser : 511-ULN2803A
C2	Condensateur 16 V 100 µF	Adafruit : 2193 Sparkfun : COM-00096 Mouser : 647-UST1C101MDD
M1	Moteur pas-à-pas unipolaire 5 V	Adafruit : 858
	Support pour piles (4 × AA) 6V	Adafruit : 830
	Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
	Câbles flexibles mâles/mâles	Adafruit : 758
	Câbles flexibles femelles/mâles (Raspberry Pi uniquement)	Adafruit : 826

Si vous comptez tenter cette expérience avec un Raspberry Pi, vous aurez besoin de câbles flexibles femelles/mâles pour connecter les broches GPIO du Raspberry Pi à la plaque d'essai.

Montage Arduino

La figure 10-14 montre le raccordement d'une carte Arduino Uno à un ULN2803.

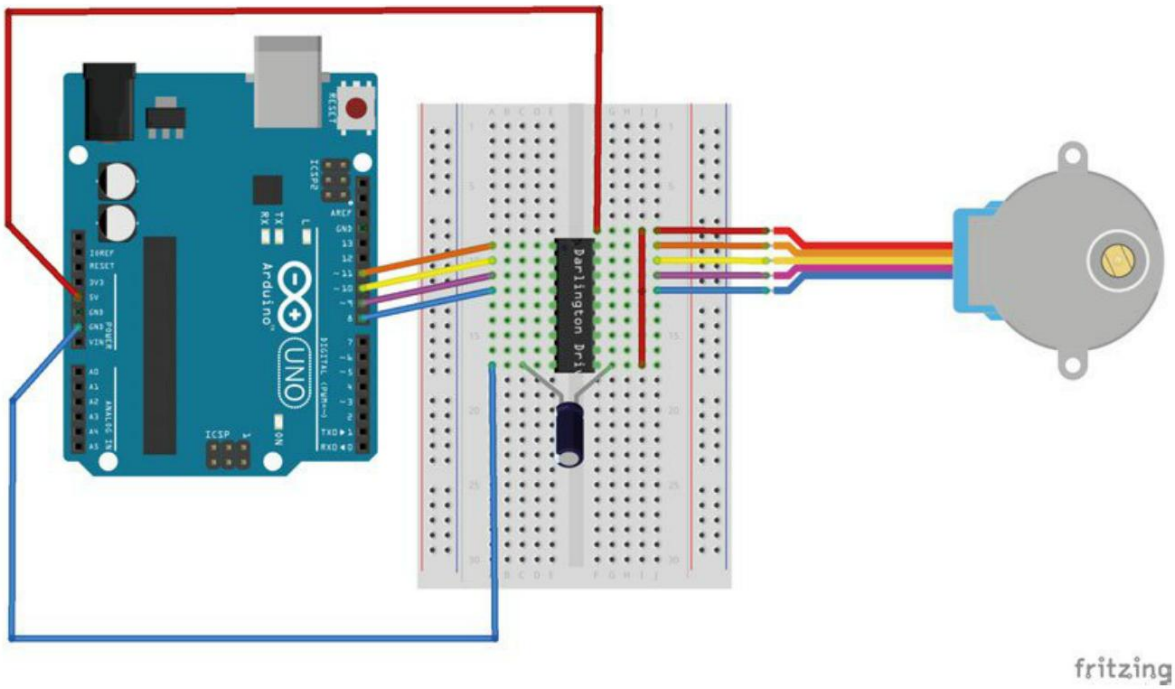


Figure 10-14. *Raccordement d'une carte Arduino à un ULN2803*

Vérifiez que le condensateur est positionné correctement avec le long conducteur positif vers la droite. Le circuit intégré doit être placé avec l'encoche vers le haut de la carte.

Le montage est bien plus ordonné que celui du moteur bipolaire.

Montage Raspberry Pi

La figure 10-15 montre la réalisation du circuit et des connexions avec le Raspberry Pi.

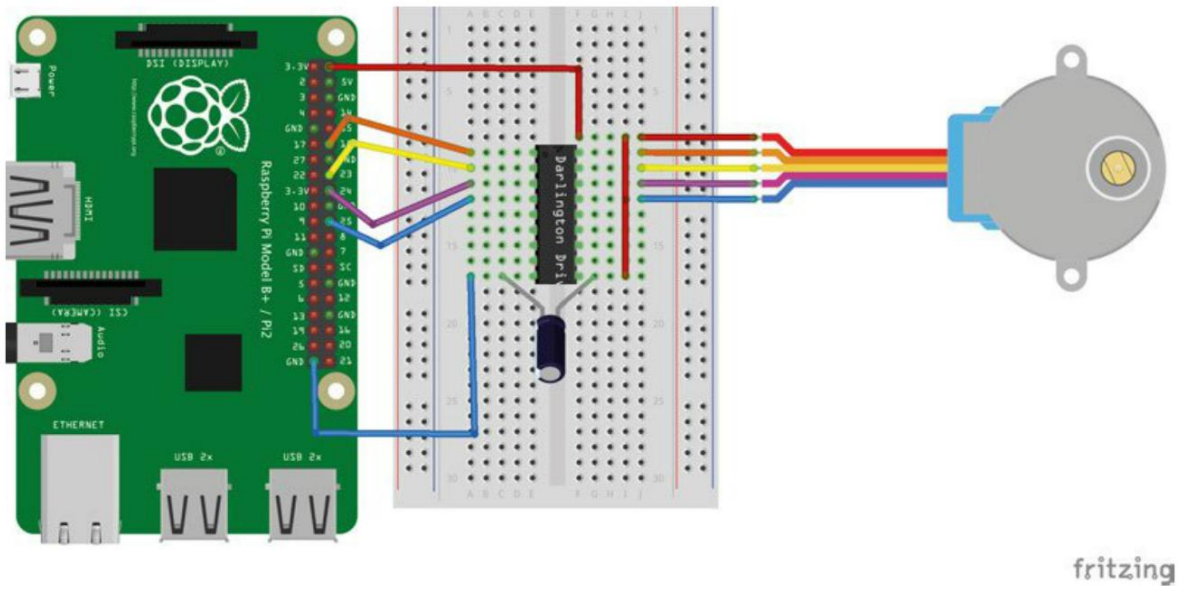


Figure 10-15. *Raccordement d'un Raspberry Pi à un ULN2803*

Programme

Hormis le fait que quelques broches ont été interverties, les programmes de l'Arduino et du Raspberry Pi sont exactement les mêmes que ceux de l'expérience « Pilotage d'un moteur pas-à-pas bipolaire », plus haut. Les programmes avec les connexions modifiées sont :

- Pour l'Arduino : `uni_stepper_lib.ino`
- Pour le Raspberry Pi : `uni_stepper.py`

Micropas

Vous avez sans doute remarqué que la rotation du moteur pas-à-pas n'est pas très fluide, même lorsqu'il tourne assez vite. Cela ne pose pas vraiment de problème pour la plupart des applications. Mais on aimerait parfois que les mouvements soient un peu plus lisses.

Le micropas est une technique qui permet de lisser les mouvements du moteur. Au lieu de se contenter d'activer ou de désactiver la rotation des bobines, la MLI est utilisée pour exciter les bobines de façon plus régulière, ce qui crée une rotation plus régulière du moteur.

Même si c'est possible de façon logicielle, il est généralement plus facile d'utiliser des composants matériels spécifiquement conçus pour le pilotage des micropas, comme EasyDriver conçu par Brian Schmalz. Cette carte se base sur le circuit intégré A3967.

Vous trouverez un excellent tutoriel à propos de l'utilisation de cette carte avec un Arduino sur la page de projet correspondante de Sparkfun (produit ROB-12779). Je ne répéterai pas ces informations ici. À la place, je vous propose une alternative Raspberry Pi sous la forme de l'expérience suivante « Micropas avec le Raspberry Pi ».

Expérience : micropas avec le Raspberry Pi

Dans cette expérience, vous utiliserez une carte de moteur pas-à-pas EasyDriver pour piloter les micropas du moteur bipolaire 12 V, que vous avez utilisé dans l'expérience « Pilotage d'un moteur pas-à-pas bipolaire », plus haut, avec un Raspberry Pi. Vous pourriez aussi utiliser un moteur unipolaire, mais ne raccordez pas le conducteur commun et utilisez le moteur unipolaire comme s'il était bipolaire.

La carte EasyDriver est fournie avec des connecteurs qui permettent de l'enficher directement sur la plaque d'essai.

La figure 10-16 présente le montage du circuit de l'expérience.

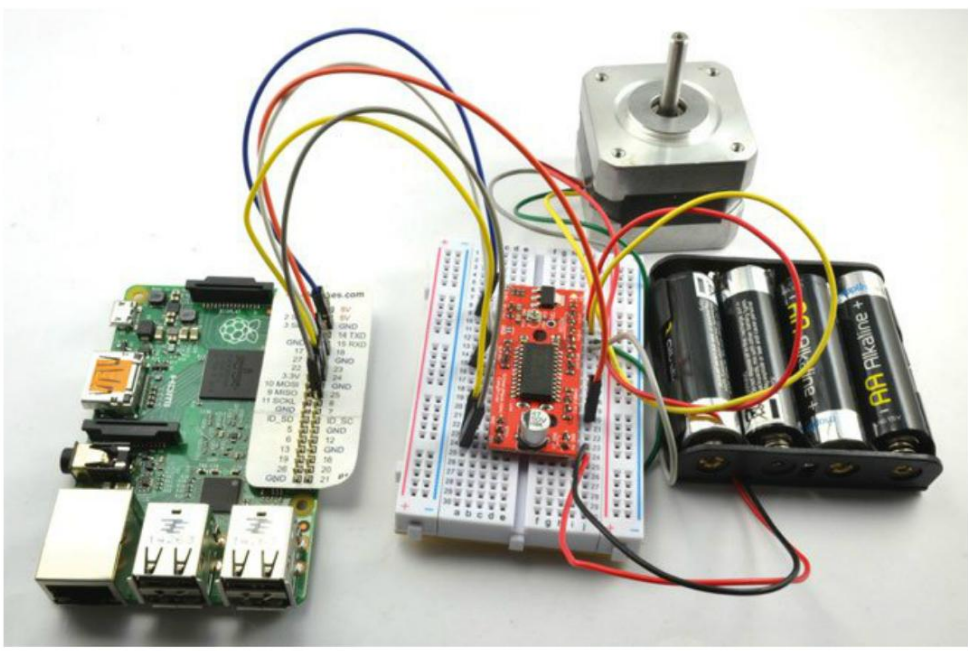


Figure 10-16. *Micropas avec un Raspberry Pi ?*

Composants nécessaires

Les composants suivants sont nécessaires pour la réalisation de l'expérience.

Composant	Source
Carte de moteur pas-à-pas EasyDriver	Sparkfun : ROB-12779
Moteur pas-à-pas bipolaire 12 V	Adafruit : 324
Support pour piles (4 × AA) 6 V	Adafruit : 830
Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
Câbles flexibles femelles/mâles	Adafruit : 826

La plupart des moteurs pas-à-pas 12 V fonctionnent correctement en 6 V, mais vous pouvez remplacer la batterie par une alimentation électrique 12 V.

Montage Raspberry Pi

La figure 10-17 montre la réalisation du circuit et les connexions entre la plaque d'essai et le Raspberry Pi.

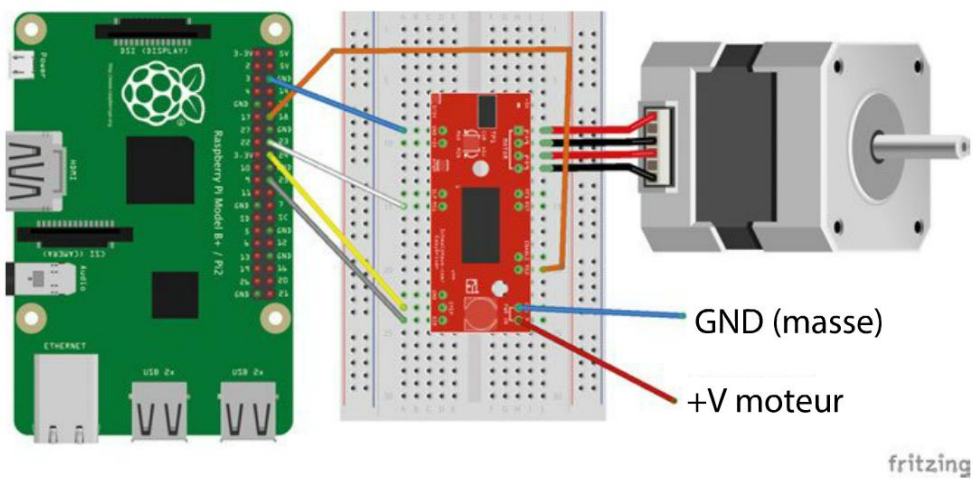


Figure 10-17. *Réalisation du circuit sur la plaque d'essai et connexions au Raspberry Pi pour les micropas*

Comme la carte EasyDriver est déjà munie de condensateurs, vous n'avez pas besoin de les ajouter sur votre circuit. Elle intègre aussi un régulateur de tension qui alimente son pilote A3967. Vous n'avez donc pas non plus besoin d'alimenter la carte à partir du Pi. Il suffit d'une masse commune et de quatre signaux de commande.

Programme

Comme son nom l'indique, le pilote de moteur pas-à-pas EasyDriver est extrêmement facile à piloter. Le processus principal qui consiste à demander au moteur d'avancer pas-à-pas est accompli à l'aide de deux broches de commande. Lorsque la broche `step` reçoit une impulsion HIGH, le moteur fait un pas dans la direction indiquée par la broche `direction`.

Deux autres broches, ms1 et ms2, commandent le degré de micropas compris entre aucun et 1/8e. Le code du programme suivant se trouve dans le fichier `microstepping.py` à l'endroit où vous avez téléchargé le code du livre :

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

step_pin = 24
dir_pin = 25
ms1_pin = 23

GPIO.setup(step_pin, GPIO.OUT)
GPIO.setup(dir_pin, GPIO.OUT)
GPIO.setup(ms1_pin, GPIO.OUT)
GPIO.setup(ms2_pin, GPIO.OUT)

period = 0.02

def step(steps, direction, period):      ❶
  GPIO.output(dir_pin, direction)
  for i in range(0, steps):
    GPIO.output(step_pin, True)
    time.sleep(0.000002)
    GPIO.output(step_pin, False)
    time.sleep(period)

def step_mode(mode):                     ❷
  GPIO.output(ms1_pin, mode & 1)         ❸
  GPIO.output(ms2_pin, mode & 2)

try:
    print('Lettre suivie d'un nombre');
    print('p20 - définit une pause de 20 ms entre les pas (pilotage de la vitesse)');
    print('m - définit le mode de pas (0-aucun 1-demi, 2-quart, 3-huitième)');
    print('f100 - en avant de 100 pas');
    print('r100 - en arrière de 100 pas');

    while True:                          ❹
      command = raw_input('Saisir la commande : ')
      parameter_str = command[1:] # from char 1 to end
      parameter = int(parameter_str)
      if command[0] == 'p':
        period = parameter / 1000.0
```

```
        elif command[0] == 'm':
          step_mode(parameter)
        elif command[0] == 'f':
          step(parameter, True, period)
        elif command[0] == 'r':
          step(parameter, False, period)

    finally:
        print('Nettoyage')
        GPIO.cleanup()
```

Vous devez maintenant tout connaître de la configuration des broches GPIO et du code d'initialisation placé au début de la plupart des programmes Python de ce livre. Si ce n'est pas le cas, je vous invite à consulter les chapitres précédents.

❶ La fonction `step` contient le code indiquant au moteur d'avancer d'un certain nombre de pas (ou micropas). Les paramètres correspondent au nombre de pas, à la `direction` (0 à 1) et à la `period` de pause entre chaque pas. La partie principale de la fonction configure tout d'abord la broche de direction (`dir_pin`) du pilote EasyDriver, puis génère le nombre d'impulsions requises sur `step_pin`. La durée de chaque impulsion `step_pin` n'est que de 2 microsecondes. D'après les spécifications techniques de l'A3967, une impulsion doit avoir une durée minimale de 1 microseconde.

❷ La fonction `step_mode` configure les broches du mode de pas en fonction de la valeur de mode qui doit être comprise entre 0 et 3.

❸ Le code à l'intérieur de `step_mode` sépare les deux bits du nombre de mode et configure `ms1_pin` et `ms2_pin` à l'aide de l'opérateur logique `and` (`&`). Le tableau 10-3 présente la relation entre le nombre de mode, les broches de micropas et le comportement du moteur.

Tableau 10-3. *Broches de commande des micropas*

Valeur de mode	ms1_pin	ms2_pin	Mode de micropas
0	LOW	LOW	Aucun
1	HIGH	LOW	Demi-pas
2	LOW	HIGH	Quart de pas
3	HIGH	HIGH	Un huitième de pas

❹ La fonction `main loop()` ressemble beaucoup à celle du programme Python de l'expérience « Pilotage d'un moteur pas-à-pas bipolaire », plus haut, à la différence qu'une commande supplémentaire `m` est ajoutée pour définir le mode de micropas.

Expérimentations

Lorsque vous exécutez le programme `microstepping.py`, vous êtes invité à saisir une commande :

```
$ sudo python microstepping.py
Lettre suivie d'un nombre
p20 - définit une pause de 20 ms entre les pas (pilotage de la vitesse)
m - définit le mode de pas (0-aucun 1-demi, 2-quart, 3-huitième)
f100 – en avant 100 pas
r100 – en arrière 100 pas
Saisir la commande: m0
Saisir la commande: p8
Saisir la commande: f800
```

Saisissez les commandes `m0`, `p8`, puis `f800`. Le moteur devrait faire quatre tours (dans le cas d'un moteur de 200 pas). La rotation ne manque-t-elle pas de fluidité ? Ensuite, tapez les commandes suivantes : `m3`, `p1`, `f6400`.

Le moteur devrait à nouveau faire quatre tours à la même vitesse, mais de façon beaucoup plus fluide.

Notez que pour conserver la même vitesse, la période de pause entre les pas a été réduite d'un facteur de 8 et le nombre de pas (ou ici, de micropas) a été augmenté d'un facteur de 8.

Moteurs CC sans balais

Les moteurs CC sans balais (ou *brushless*) sont ces petits moteurs puissants que l'on retrouve dans les quadrirotors et les avions télécommandés (figure 10-18). À poids égal, ils peuvent créer beaucoup plus de couple qu'un moteur CC ordinaire. Nous les traitons dans ce chapitre, car même si, techniquement, ce sont des moteurs à courant continu, ils ont davantage de points communs avec des moteurs pas-à-pas.

Figure 10-18. *Un moteur CC sans balais*

Contrairement aux moteurs CC « à balais » (chapitre 7), les moteurs CC sans balais n'ont pas de collecteur tournant qui inverse le sens du courant lorsque le moteur tourne. De par leur conception, ils ressemblent davantage à des moteurs pas-à-pas, sauf qu'au lieu d'avoir deux bobines (par paires), ils en ont trois, répétées plusieurs fois. On peut donc les ranger dans la catégorie des moteurs triphasés et il ne suffit pas d'un pont en H pour les piloter. En fait, chacune des trois connexions d'une bobine doit être capable de fournir et d'absorber du courant, mais aussi d'adopter un troisième état à « haute impédance » dans lequel la connexion de la bobine est effectivement déconnectée. Pour réguler le moteur, le circuit du pilote mesure la tension sur la sortie « déconnectée » d'une bobine particulière afin d'ajuster la durée de commutation vers la prochaine phase.

Il ne semble malheureusement pas y avoir de circuits intégrés pouvant être montés sur une plaque d'essai et capables de jouer le rôle de pilote de ces moteurs. La meilleure solution est donc de se procurer une carte de pilote toute-prête dont différents modèles sont commercialisés sur eBay.

Si vous voulez tirer profit de la puissance et de la taille d'un moteur sans balais sans compliquer les choses avec une carte de pilote, vous pouvez aussi acheter un moteur qui intègre déjà un contrôleur. Ces modèles n'ont que deux fils sortant de leur boîtier, donc vous pouvez l'utiliser de la même façon qu'un moteur. D'ailleurs, la pompe centrifuge qui était illustrée à la figure 7-13 utilise un moteur de ce type.

Résumé

Dans ce chapitre, nous avons étudié les moteurs pas-à-pas et nous avons appris à les utiliser avec une carte Arduino et un Raspberry Pi. Dans le chapitre suivant, vous apprendrez à réguler le chauffage et le refroidissement.

Chauffage et refroidissement

11

Dans ce chapitre, nous examinerons des systèmes de chauffage et de refroidissement, tels que les éléments résistifs chauffants et les modules thermoélectriques Peltier.

Ici, nous traiterons plus particulièrement du fonctionnement de ces systèmes, alors qu'au chapitre 12, nous nous intéresserons davantage à leur utilisation avec un Arduino et un Raspberry Pi pour un contrôle thermostatique précis.

Résistances chauffantes

Au chapitre 5, nous avons mentionné le fait que lorsque les résistances limitent le flux du courant, elles génèrent de la chaleur. Vous vous souvenez certainement que la quantité de chaleur produite (mesurée en watts) est égale à la quantité de courant (mesurée en ampères) circulant dans la résistance multipliée par la tension (mesurée en volts) aux bornes de la résistance.

Expérience : résistance chauffante

Vous devez maintenant savoir comment allumer et éteindre le courant pour démarrer un moteur. Les mêmes principes s'appliquent à la commutation de courant sur une résistance. Les expériences précédentes qui ont permis de réguler la puissance d'un moteur peuvent être menées de la même façon avec une résistance. On pourrait aussi utiliser la modulation de longueur d'impulsion pour réguler la puissance d'une résistance chauffante. Pour mener cette expérience, il est possible de se passer d'un Arduino ou d'un Raspberry Pi et de se contenter d'une batterie et d'une résistance.

Notez que ce projet utilise une résistance comme élément chauffant et une batterie comme source d'alimentation. La capacité de chauffage sera assez limitée, mais elle devrait suffire à des fins d'expérimentation.

Composants nécessaires

Les composants suivants sont nécessaires pour mener cette expérience.

Composant	Source
Résistance 100 Ω	Mouser : 291-100-RC
Thermomètre	Magasin de bricolage
Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
Support pour piles (4 × AA) 6V	Adafruit : 830

Construction

Dans ce projet, vous vous contenterez de connecter la résistance aux bornes des piles et vous utiliserez la plaque d'essai pour maintenir la résistance en place, comme illustré à la figure 11-1.

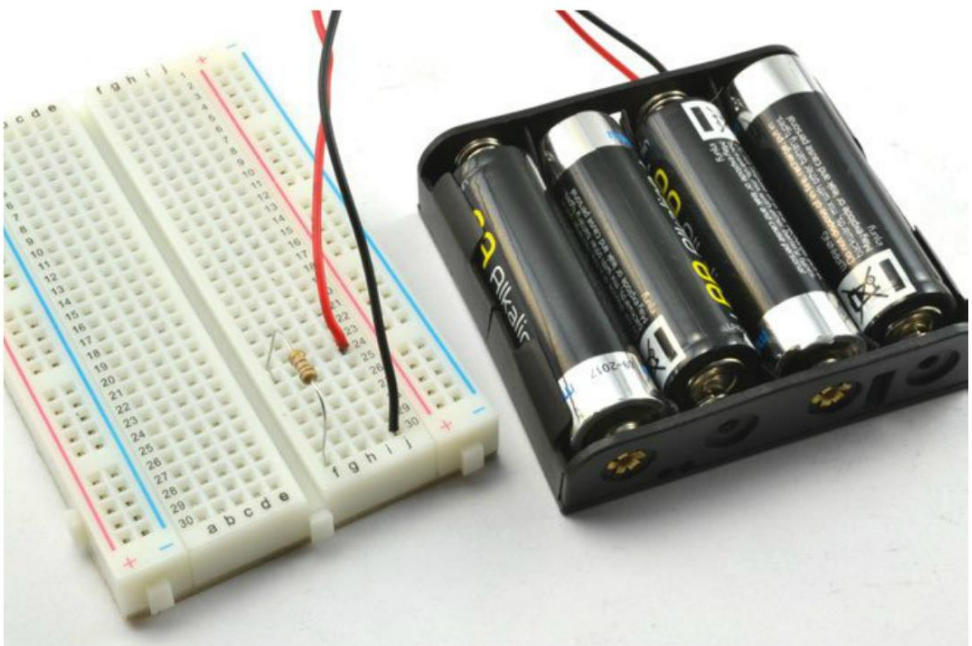


Figure 11-1. *Expérience de la résistance chauffante*

Ne connectez pas encore les conducteurs de la pile, car dès que vous le ferez, la résistance commencera à chauffer.

Expérimentations

Branchez la batterie et utilisez le thermomètre pour mesurer la vitesse à laquelle la température augmente sur la résistance. La température devrait atteindre environ 75 °C. Soyez prudent, car c'est suffisant pour se brûler !

Sachant que la résistance est de 100 Ω et la tension de 6 V, nous pouvons donc calculer le courant comme suit : I = V/R = 6/100 = 0,06 A. La puissance est égale à la tension multipliée par le courant, soit 6 × 0,06 = 0,36 W ou 360 mW.

C'est légèrement supérieur à la puissance nominale maximale de la résistance, qui est de 250 mW (1/4 W), donc le composant ne résistera pas longtemps.

Projet : éclateur aléatoire de ballons Arduino

Ce projet s'adresse aux personnes qui ont les nerfs solides ! Un Arduino pilote une résistance utilisée comme élément chauffant collé sur un ballon. Au bout d'une période d'une durée aléatoire, la résistance est alimentée et le ballon éclate (figure 11-2).

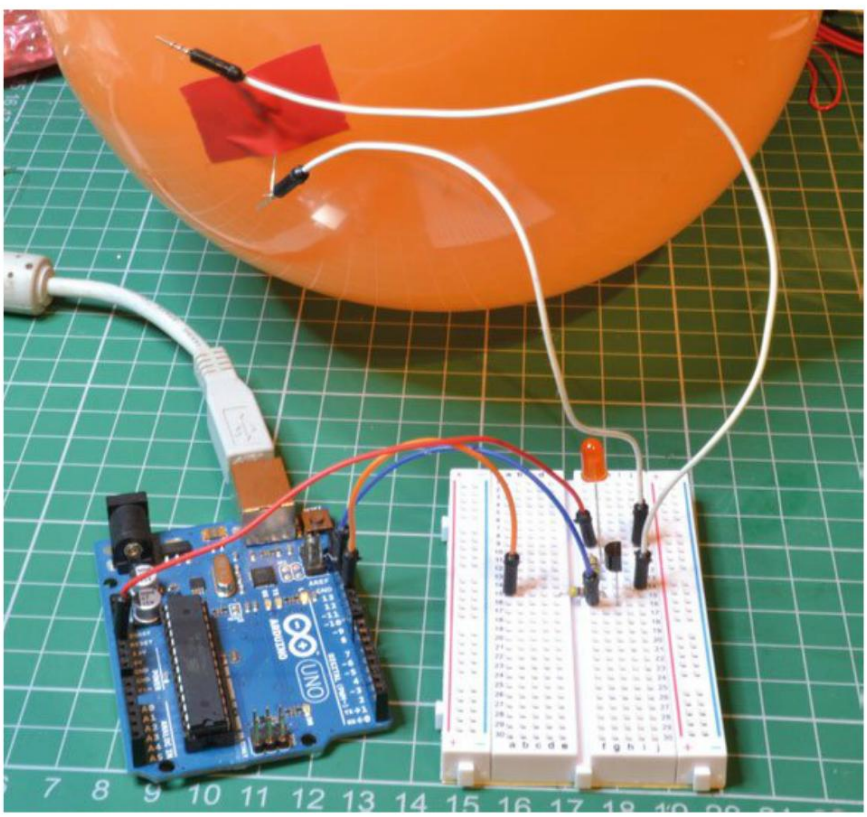


Figure 11-2. *Un éclateur aléatoire de ballons*

C'est un projet assez destructeur, car le fait que la résistance devienne assez chaude pour exploser un ballon implique aussi que sa température augmente assez pour la détruire – vous risquez de voir de la fumée.

Risque de brûlures

Ce projet utilise une résistance pour éclater un ballon. Attention, la température de la résistance augmente assez pour brûler les doigts. Ne la touchez donc pas lorsqu'elle est sous tension. Après avoir cessé de l'alimenter, attendez au moins 30 secondes avant de la manipuler.

Ce projet peut aussi poser un risque d'incendie. Prévoyez un extincteur à portée de main. Protégez-vous également les yeux avec des lunettes de sécurité, car la résistance risque de s'envoler lors de l'explosion du ballon.

Composants nécessaires

Les composants suivants sont nécessaires pour la réalisation du projet.

Nom	Composant	Source
R1, R3	Résistance 470 Ω	Mouser : 291-470-RC
R2	Résistance 10 Ω 1/4 W	Mouser : 291-10-RC
Q1	Transistor Darlington MPSA14	Mouser : 833-MPSA14-AP
LED1	LED rouge	Adafruit : 297 Sparkfun : COM-09590
	Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
	Câbles flexibles mâles/mâles	Adafruit : 758

Procurez-vous plusieurs résistances 10 Ω, car elles risquent de partir en fumée.

Matériel

Lorsque le transistor Darlington aura joué son rôle, la tension atteindra environ 3,5 V sur la résistance au moment où le transistor devient conducteur. Cela correspond à un courant de 3,5 V/10 Ω = 350 mA. Cela ne devrait pas endommager les composants connectés sur le port USB.

La chaleur dégagée par la résistance sera de I × V = 350 mA × 3,5 V = 1,225 W. C'est bien supérieur à la puissance nominale de 1/4 W de la résistance, mais elle devrait résister suffisamment longtemps pour éclater le ballon.

La figure 11-3 montre la réalisation du circuit du projet.

La résistance doit être solidement attachée aux bornes des câbles flexibles (figure 11-4) afin d'éviter de la perdre au moment où le ballon éclate.

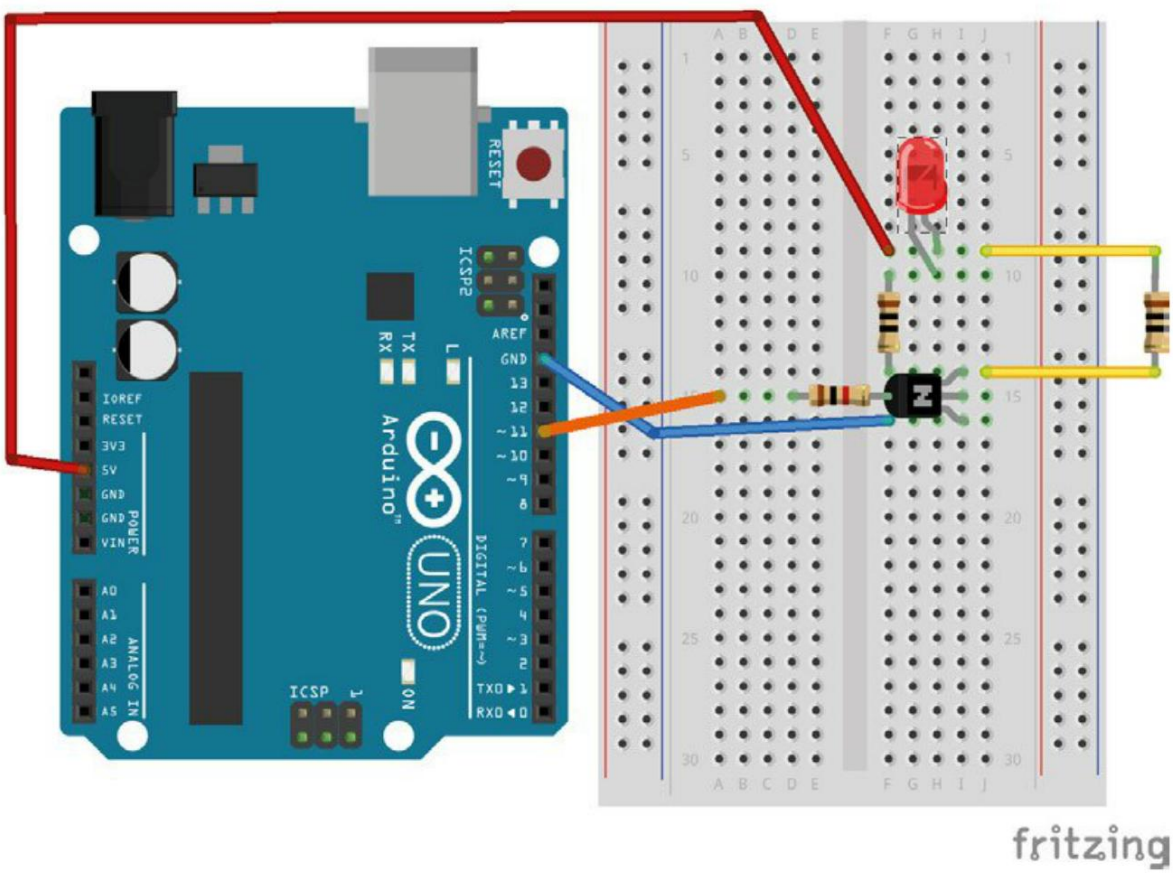


Figure 11-3. *Réalisation du circuit de commande de l'éclateur de ballon*

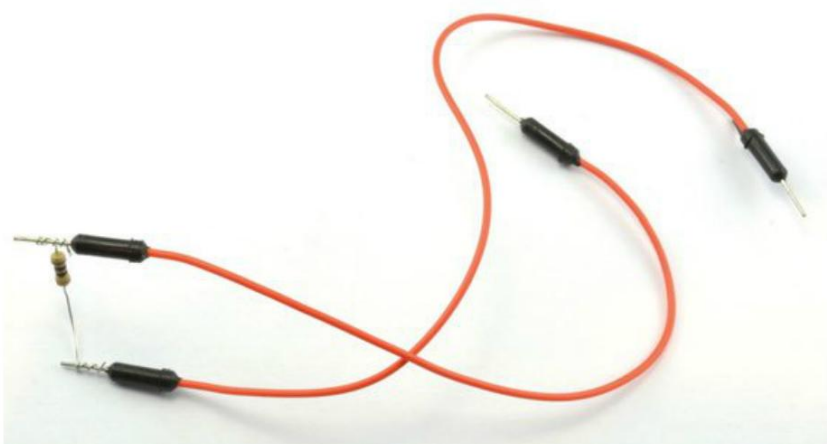


Figure 11-4. *Fixation de la résistance*

Programme

À part le bouton de réinitialisation de la carte Arduino, il n'y a pas de bouton marche-arrêt pour ce projet. Le sketch Arduino du projet démarre dès que la carte est branchée. Par conséquent, attendez d'être prêt avant de connecter l'une des extrémités des câbles flexibles à la résistance.

Ouvrez `/arduino/projects/pr_balloon_popper` (que vous trouverez à l'endroit où vous avez téléchargé le code du livre) :

```
const int popPin = 11;

const int minDelay = 3; // Secondes     ❶
const int maxDelay = 5; // Secondes
const int onTime = 3;   // Secondes     ❷

void setup() {
  pinMode(popPin, OUTPUT);
  randomSeed(analogRead(0));           ❸
  long pause = random(minDelay, maxDelay+1);    ❹
  delay(pause * 1000);                 ❺
  digitalWrite(popPin, HIGH);          ❻
  delay(onTime * 1000);
  digitalWrite(popPin, LOW);
}

void loop() {
}
```

❶ Les constantes `minDelay` et `maxDelay` définissent la plage de délais d'excitation de la résistance et d'éclatement du ballon.

❷ La constante `onTime` définit la durée pendant laquelle la résistance doit être allumée. Quelques essais seront éventuellement requis pour déterminer la durée correcte. La résistance ne doit pas chauffer trop longtemps afin d'éviter qu'elle ne grille trop vite.

❸ Les nombres aléatoires en Arduino C ne sont pas vraiment aléatoires, mais ils font partie d'une longue séquence. Pour veiller à ce qu'un délai différent soit défini à chaque réinitialisation de la carte Arduino, cette ligne définit la position de cette séquence en fonction de la valeur lue sur la broche analogique A0. Comme cette broche n'est connectée à rien, les valeurs seront plus ou moins aléatoires.

❹ La pause qui précède l'activation de la résistance est définie par la fonction `random` qui renvoie un nombre aléatoire compris entre les deux valeurs qui lui sont fournies comme paramètres. 1 est ajouté au second paramètre parce que la plage est exclusive. La variable utilisée est de type `long`, car `int` peut uniquement contenir des chiffres jusqu'à 32 767. Dans ce cas, vous ne pourriez pas définir de délai au-delà de 32 secondes.

❺ Délai de `pause` en secondes.

❻ Provoque la conduction du transistor et alimente la résistance. Délai défini par `onTime()`, puis coupure de l'alimentation.

Utilisation de l'éclateur de ballon

Lorsque vous testerez le projet, vous voudrez peut-être davantage de certitude quant au moment où le ballon éclate. Dans ce cas, corrigez `minDelay` et `maxDelay` en saisissant 2 et 3, respectivement.

Collez la résistance sur le ballon, puis branchez les fils de la résistance sur la plaque d'essai. Appuyez sur le bouton de réinitialisation et attendez le « bang ».

Éléments chauffants

Les résistances ne sont pas des éléments chauffants très pratiques. Pour chauffer un certain volume, il faut utiliser un élément chauffant. Ce sont de grosses résistances fabriquées dans des matériaux qui sont conçus pour chauffer et pour transférer la chaleur vers ce que vous essayez de chauffer.

En général, le chauffage consomme beaucoup d'énergie, comparé à la production de lumière, de son ou même de mouvement. Par conséquent, les éléments chauffants utilisés dans des bouilloires électriques, les lave-vaisselle et d'autres appareils d'électroménager qui chauffent l'eau utilisent directement le courant alternatif haute tension fourni par le réseau électrique pour obtenir la puissance requise pour accomplir leur mission. Reportez-vous au chapitre 13 si vous voulez commuter des éléments chauffants de taille substantielle.

La puissance d'un élément chauffant est mesurée en watts (W) ou kW (milliers de watts), comme la puissance nominale d'une résistance. Cette dernière correspond à la quantité de chaleur que la résistance peut produire par seconde avant de devenir trop chaude et se détruire.

Dans l'expérience « Résistance chauffante », plus haut, nous nous sommes servis de la loi d'Ohm et de la loi de puissance (P = I V) pour calculer la puissance. Ces lois peuvent être réunies dans une seule et unique formule qui vous permettra de calculer la puissance si vous connaissez la résistance de l'élément chauffant et la tension sur ses bornes :

$$P = \frac{V^2}{R}$$

En d'autres termes, la puissance de chauffage produite par un élément chauffant équivaut au carré de la tension en volts divisé par la résistance en ohms. Cette formule s'applique aux tensions en courant continu et en courant alternatif.

Ainsi, une résistance (ou un élément chauffant) de 10 Ω alimentée par une tension de 12 V génèrera (12 × 12)/10 = 14,4 W de chaleur. Si la tension était de 120 V CA, la puissance serait de (120 × 120)/10 = 1 440 W, soit 1,44 kW.

Un élément chauffant est donc défini par :

- sa tension de service (12 V, 120 V, 220 V) ;
- sa puissance (50 W, 1 kW, 5 kW).

Si cet élément chauffant est destiné à chauffer de l'eau, il sera sans doute immergé afin que la chaleur puisse se propager dans l'eau. S'il ne l'est pas, il risque de trop chauffer et de casser.

Puissance et énergie

Les mots « puissance » et « énergie » sont souvent utilisés l'un pour l'autre dans le langage courant. D'un point de vue scientifique, la relation entre la puissance et l'énergie est aussi différente que la relation entre la vitesse et la distance.

La puissance correspond au niveau de conversion de l'énergie. La puissance d'un élément chauffant est la quantité d'énergie qui est libérée sous forme de chaleur par seconde.

L'unité de mesure de l'énergie est le joule ; une puissance de 1 watt correspond à une énergie de 1 joule par seconde.

De la puissance à la hausse de température

Si vous connaissez la puissance en watts pouvant être produite par votre élément chauffant et la matière qui est chauffée, vous pouvez en déduire la durée requise pour augmenter la température d'un certain niveau.

Comme pour beaucoup de choses à caractère scientifique, des unités de mesure internationales, comme le watt, le degré Celsius, le gramme et la seconde sont généralement utilisées pour ces calculs. Rien ne vous empêche de convertir ensuite le résultat dans une unité qui vous est plus familière.

Les matériaux ont une propriété qui s'intitule la capacité thermique massique et qui correspond à la quantité d'énergie requise pour augmenter la température d'un gramme de matière de 1 °C. Par exemple, l'eau a une capacité thermique massique de 4,2 J/g/degré C. En d'autres termes, il faut une énergie de 4,2 J pour augmenter la température de 1 gramme d'eau de 1 °C. Si vous voulez augmenter la température de 100 grammes d'eau de 1°C, cela demande une énergie de 420 J ; si vous voulez augmenter la température de ces mêmes 100 grammes de 10 °C, l'énergie requise atteindrait 4 200 joules.

L'air a une capacité thermique massique d'environ 1 J/g/degré C ; celle du verre est d'environ 0,8 J/g/degré C. Tous les matériaux sont différents.

Eau bouillante

Nous allons voir si notre petite résistance est capable de faire bouillir de l'eau.

Supposons que nous voulions faire bouillir une tasse d'eau (environ 250 g d'eau). L'eau est à température ambiante (20°C) et bout à 100°C ; la température doit donc augmenter de 80°C.

L'énergie totale requise pour augmenter la température de 250 g d'eau de 80°C est :

$$4{,}2 \times 250 \times 80 = 84\,000 \text{ joules}$$

Dans l'expérience « Résistance chauffante », plus haut, la résistance produit 0,36 W ou 0,36 joules par seconde. Il faudrait donc 84 000/0,36 = 233 333 secondes, soit environ 64 heures pour faire bouillir l'eau !

Pour des raisons pratiques décrites plus bas, l'eau ne bouillira jamais dans ces conditions.

Déperdition de chaleur et équilibre

Il est important de s'assurer que l'élément chauffant (ou refroidissant) est suffisamment puissant pour accomplir sa mission.

Vous ne réussirez jamais à faire bouillir une tasse d'eau avec une résistance de 1/4 W à cause de la déperdition de chaleur. Si l'eau se trouvait dans un récipient bénéficiant d'une parfaite isolation thermique dont la chaleur ne peut pas s'échapper, alors oui, l'eau continuerait à chauffer jusqu'à l'ébullition.

La déperdition de chaleur d'un corps est proportionnelle à l'écart de température entre ce corps et son environnement. Par conséquent, plus l'eau se réchauffe, plus vite elle perd de la chaleur. On finit donc par parvenir à un équilibre dans lequel le récipient d'eau perd de la chaleur au même rythme (watts) auquel l'élément chauffant ajoute de l'énergie (watts) et la température se stabilise. C'est pourquoi la résistance de l'expérience « Résistance chauffante », plus haut, chauffe jusqu'à 75 °C environ et ne va pas au-delà.

Éléments Peltier

Les éléments Peltier (figure 11-5) ont la propriété très utile de chauffer d'un côté et de refroidir de l'autre lorsqu'ils sont traversés par un courant.

Une quantité assez importante de courant est nécessaire pour que ce phénomène se produise (de l'ordre de 2 à 6 A à 12 V). Vous aurez donc besoin d'une alimentation électrique assez puissante si vous voulez utiliser un élément Peltier. Ces éléments sont souvent utilisés dans les réfrigérateurs de camping et les refroidisseurs de boisson. Par rapport aux réfrigérateurs ordinaires, ils présentent l'avantage de ne pas avoir de pièces mobiles risquant de dysfonctionner.

Fonctionnement des éléments Peltier

Lorsqu'un courant traverse une jonction entre deux matériaux conducteurs différents, l'un des côtés de la jonction devient légèrement plus chaud et l'autre devient légèrement plus froid. C'est l'effet thermoélectrique ou effet Peltier, qui doit son nom au physicien français Jean Peltier qui l'a découvert en 1834.

L'effet étant de relativement faible amplitude, il doit être multiplié pour être suffisamment efficace pour refroidir une boisson. On y parvient en plaçant plusieurs jonctions alternées les unes à la suite des autres de façon à ce que le courant les traverse successivement et que chacune contribue à l'effet global. Les éléments communs bon marché ont généralement une douzaine de jonctions (figure 11-6). De part et d'autre des éléments, il y a un matériau de base qui constitue le pain du sandwich de la jonction.

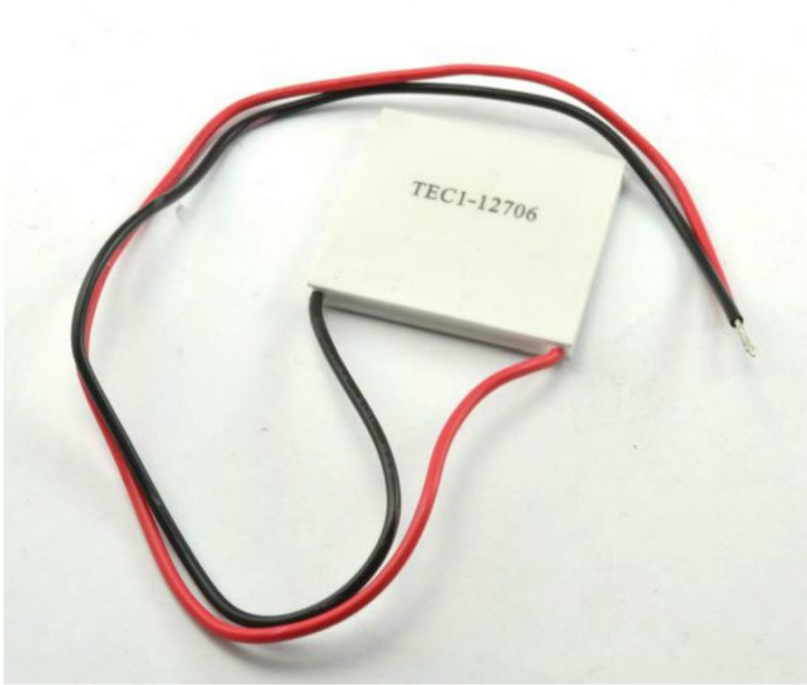


Figure 11-5. *Un élément Peltier*

Les différents types de matériaux utilisés pour les jonctions sont deux types de semiconducteurs, comme ceux employés dans la fabrication de transistors et de circuits intégrés, mais optimisés pour leur effet thermoélectrique. Ils sont dits de type N ou P (négatif et positif).

Une caractéristique très intéressante des éléments Peltier est qu'en plus d'être utilisés pour refroidir des corps, si vous faites en sorte qu'un côté soit plus chaud que l'autre, une petite quantité d'électricité est générée.

Considérations pratiques

Le principal problème posé par les éléments Peltier est que les côtés chauds et froids sont très rapprochés. Donc le côté chaud ne tarde pas à chauffer le côté froid à moins que la chaleur soit éliminée le plus vite possible.

Pour commencer, on peut utiliser un dissipateur thermique, comme illustré à la figure 11-7.

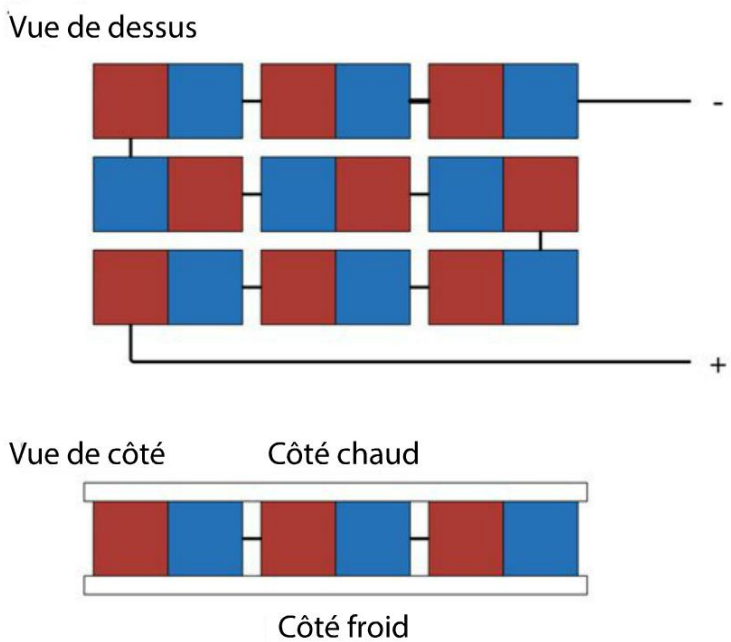


Figure 11-8. *À l'intérieur d'un élément Peltier*

Dans le système de refroidissement illustré à la figure 11-7, le dissipateur thermique est un radiateur en aluminium muni de lames qui augmentent la surface de dissipation de la chaleur. Le côté froid du dispositif correspond au bloc qui dépasse dans le compartiment réfrigéré thermiquement isolé.

Un dissipateur thermique isolé est loin d'être aussi efficace qu'un dissipateur thermique muni d'un ventilateur pour extraire l'air qui a été réchauffé par convection et le remplacer par de l'air frais. D'ailleurs, certains dispositifs sont munis de ventilateurs des deux côtés de l'élément Peltier pour le rendre encore plus efficace (figure 11-8).

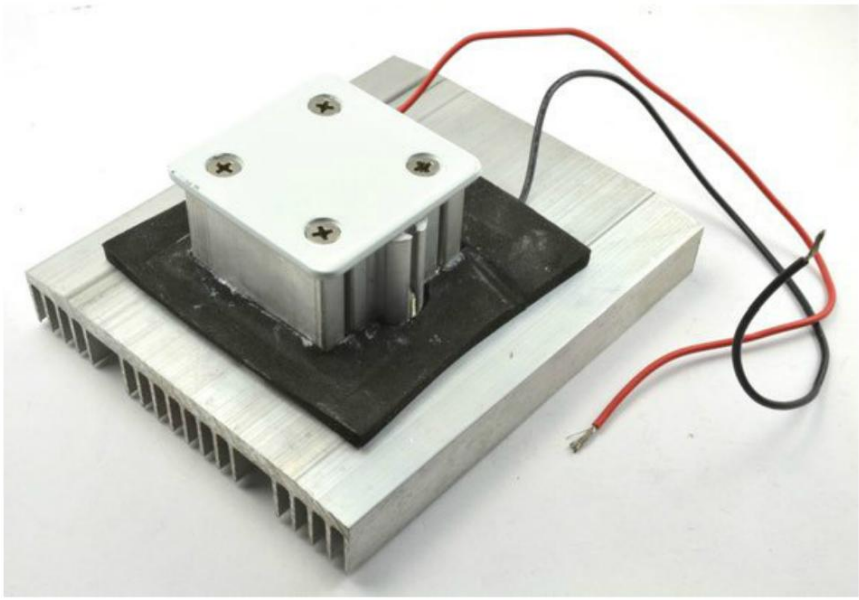


Figure 11-7. *Utilisation d'un dissipateur thermique avec un élément Peltier*

Figure 11-8. *Un système de réfrigération Peltier à deux ventilateurs*

Projet : réfrigération de boisson

Ce projet n'utilise ni Raspberry Pi ni Arduino. Il sert uniquement à vous montrer comment connecter un système de refroidissement Peltier pour fabriquer un système rudimentaire de réfrigération de boisson (figure 11-9). Au chapitre 12, ce projet élémentaire sera complété par un contrôle thermostatique de la température. Puis, au chapitre 14, un afficheur OLED y sera ajouté pour indiquer la température.

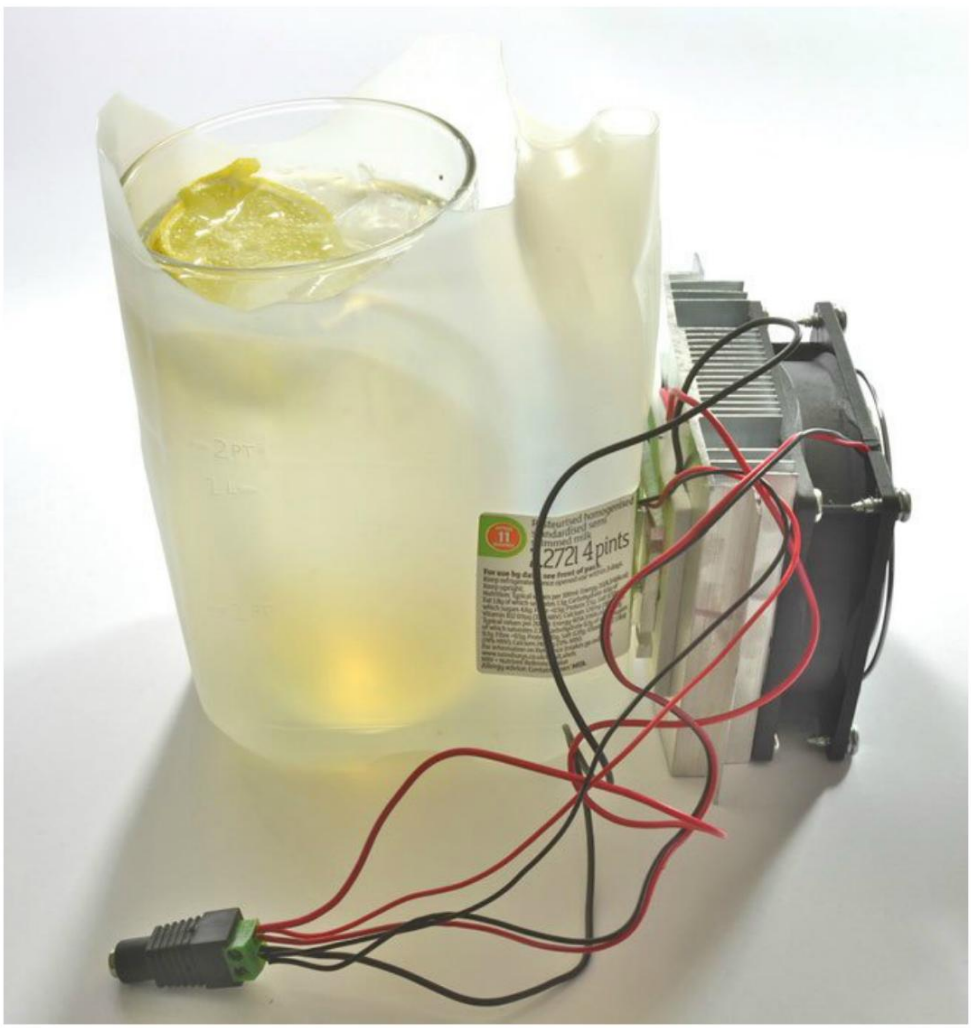


Figure 11-9. *Le projet de réfrigération de boisson*

Composants nécessaires

Les composants suivants sont nécessaires pour la réalisation du projet.

Composant	Source
Système de refroidissement Peltier à double ventilateur 4 A ou moins	eBay
Jack femelle pour adaptateur de bornes à vis	Adafruit : 368
Alimentation électrique (12 V à 5 A)	Adafruit : 352
Grande bouteille de lait ou de jus de fruit	Récupération
Une boisson rafraîchissante	

Si vous voulez utiliser un élément Peltier de plus de 4 A, pensez aussi à utiliser une alimentation électrique plus puissante pour disposer d'un courant nominal supérieur à celui requis par le système de refroidissement. Prévoyez au moins un demi-ampère supplémentaire pour les ventilateurs et un demi-ampère de plus par sécurité.

Construction

Si vous dénudez les fils de votre système de refroidissement, vous trouverez trois paires de fils : une paire pour l'élément Peltier et une paire pour chacun des ventilateurs. Tout cela nécessite une alimentation électrique en 12 V. Pour l'obtenir, le plus simple est d'utiliser un adaptateur avec une borne à vis d'un côté et une prise jack CC de l'autre. Raccordez les trois fils rouges du système de refroidissement à la borne à vis marquée + et les trois fils noirs à la borne à vis marquée – comme illustré à la figure 11-10.

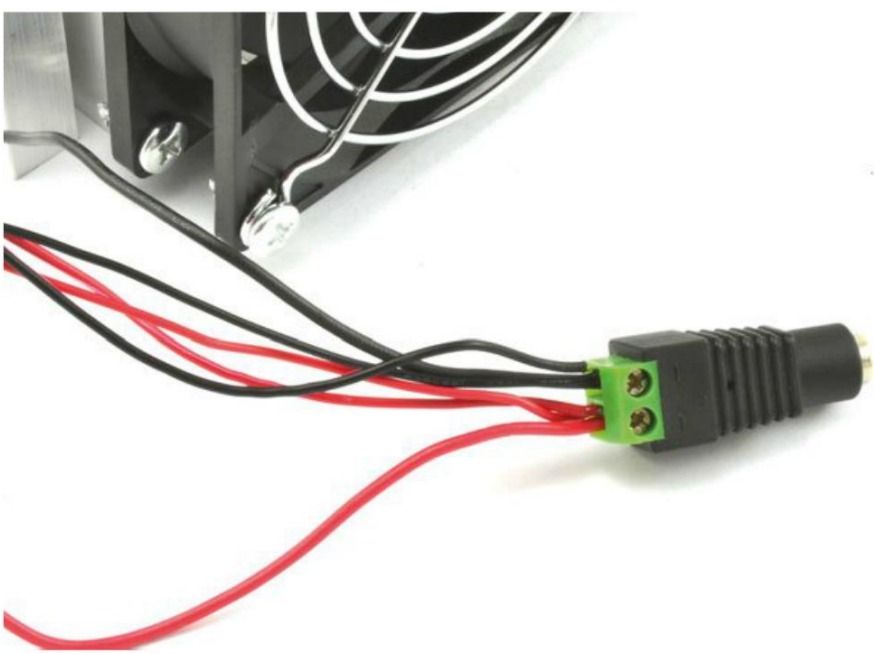


Figure 11-10. *Raccordement du projet de réfrigération de boisson*

Vous devrez couper la partie haute de la bouteille en plastique et découper une ouverture carrée sur le côté de façon à ce que la partie réfrigérante du système de refroidissement Peltier dépasse à l'intérieur de la bouteille (figure 11-11). Choisissez une bouteille en plastique suffisamment grande pour contenir le récipient contenant la boisson à refroidir.

Figure 11-11. *Préparation de la bouteille en plastique*

Si des vis dépassent du côté froid du ventilateur, prévoyez aussi de percer quelques trous dans la bouteille en plastique de façon à ce que le système de refroidissement soit solidement fixé dessus. Assurez-vous que l'orifice prévu pour le côté froid du module Peltier est percé de façon à ce que le bas du module et le bas de la bouteille soient au même niveau pour éviter que l'ensemble ne se renverse.

L'avantage de l'utilisation d'une bouteille en plastique de récupération est que si vous vous trompez, vous pouvez toujours recommencer en en prenant une nouvelle.

Application du projet

Une fois que tous les éléments sont assemblés, branchez l'alimentation électrique. Les deux ventilateurs se mettent en marche. Si vous placez la main sur la bouteille, vous sentirez l'air froid provenant du petit ventilateur.

Comme nous l'avons découvert lorsque nous avons essayé de faire bouillir de l'eau, il faut du temps pour augmenter ou baisser la température ne serait-ce que d'une quantité relativement faible d'eau. Même si ce projet finira par refroidir une boisson chaude, il réussira beaucoup mieux à maintenir une boisson fraîche qui est déjà froide au départ.

Ce projet est assez énergivore, car il consomme jusqu'à 50 W d'électricité simplement pour conserver une boisson fraîche. Au chapitre 12, vous rendrez ce projet un peu plus efficace en y ajoutant un contrôle thermostatique.

Résumé

Le chauffage et le refroidissement nécessitent une puissance relativement élevée pour fonctionner rapidement, mais les techniques de commutation à base de transistors et de relais peuvent toutes être utilisées pour la commutation d'éléments chauffants et de modules Peltier.

Au chapitre suivant, vous apprendrez à réguler précisément la température et à améliorer le projet de réfrigération de boisson à l'aide d'un contrôle thermostatique.

Boucles de commande

12

Même si ce livre traite surtout d'actionneurs (moteurs, chauffages, lumières et autres « sorties »), de nombreux systèmes qui les emploient utilisent aussi des capteurs pour les contrôler. Comme exemple, nous pourrions citer un radiateur à commande thermostatique : la puissance de l'élément chauffant est régulée en réponse aux mesures réalisées par un capteur de température. Il peut aussi bien s'agir d'une simple commande marche/arrêt que d'une technique plus avancée comme un régulateur PID (Proportionnel-Intégral-Dérivé).

Dans ce chapitre, nous verrons comment associer des capteurs à des actionneurs pour fabriquer un système de commande. Puis, nous appliquerons cette technique au système de réfrigération de boisson réalisé au chapitre 11.

Le thermostat simple

Pour expliquer comment utiliser un capteur et un actionneur dans un système de commande, nous commencerons par employer un capteur de température avec un élément chauffant et nous tenterons de maintenir une température constante. Les mêmes principes peuvent être appliqués pour piloter la position d'un moteur ou réguler le niveau d'eau dans un réservoir, ou n'importe quelle propriété qui peut être à la fois mesurée et commandée électroniquement.

La figure 12-1 présente une boucle de commande ou de régulation.

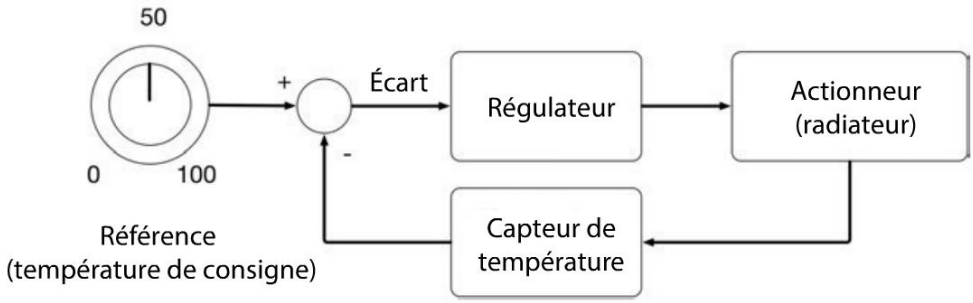


Figure 12-1. *Une boucle de régulation (thermostat)*

Ce type de montage se nomme une boucle de commande ou de régulation. Dans l'exemple du thermostat, vous réglez la température d'ambiance voulue ; c'est la référence, ou température de consigne. Un capteur de température mesure la température ambiante. L'écart de température est mesuré en soustrayant la température ambiante à celle de consigne. À partir de cette valeur d'écart ou d'erreur, le régulateur calcule la puissance devant être fournie au radiateur.

Un radiateur de base sera simplement allumé ou éteint. Si cet écart est positif (c'est-à-dire que la température de consigne est supérieure à celle ambiante), cela signifie qu'il fait trop froid : le radiateur est allumé. Au contraire, si la température ambiante est supérieure à la consigne, il fait trop chaud : le radiateur est éteint.

Dans l'expérience « Précision de la régulation thermostatique marche/arrêt », ci-après, vous fabriquerez un simple thermostat Arduino avec une résistance comme corps chauffant et un capteur numérique de température. Nous réutiliserons ce même dispositif lorsque nous passerons à des méthodes plus précises de régulation d'un actionneur.

Expérience : précision de la régulation thermostatique marche/arrêt

Pour la première expérience de ce chapitre, nous utiliserons un Arduino et un circuit numérique de sonde de température DS18B20 pour réaliser un système rudimentaire de régulation de température. Une résistance 100 Ω sert d'élément chauffant. Elle est connectée physiquement au DS18B20 qui mesure la température de la résistance.

Le moniteur série sera utilisé pour ajuster la température de consigne et la température mesurée y sera également affichée. Vous pouvez copier-coller les valeurs de température dans une feuille de calcul afin de générer des tableaux illustrant le niveau de précision de la régulation de la température.

Ce petit système, qui présente l'avantage de tout offrir à portée de main, est suffisant pour mener des expériences de régulation de la température.

La figure 12-2 illustre le montage réalisé pour l'expérience. Vous pouvez voir la pince maintenant solidement la résistance en place contre la puce de contrôle de la température.

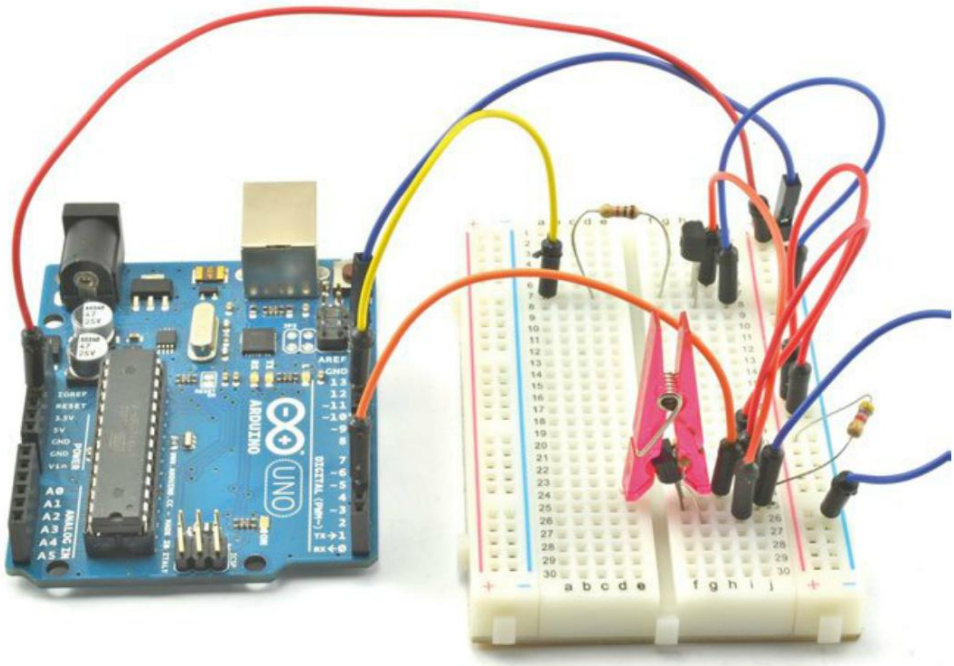


Figure 12-2. *Expérience de régulation thermostatique marche-arrêt Arduino*

Composants nécessaires

Que vous utilisiez un Raspberry Pi ou un Arduino (ou les deux), vous aurez besoin des composants suivants pour réaliser l'expérience.

Nom	Composant	Source
IC1	Module capteur de température numérique DS18B20	Adafruit : 374 (résistance 4,7 kΩ incluse)
R1	Résistance 4,7 kΩ	Mouser : 291-4.7k-RC
R2	Résistance 1 kΩ	Mouser : 291-1k-RC
R3	Résistance 100 Ω	Mouser : 291-100-RC
Q1	Transistor Darlington MPSA14	Mouser : 833-MPSA14-AP
	Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
	Câbles flexibles mâles/mâles	Adafruit : 758
	Petite pince à papier	

Si vous n'avez pas de petite pince à papier, une pince ordinaire fera l'affaire, mais assurez-vous qu'elle est recouverte de plastique pour éviter les risques de court-circuit avec la résistance ou les autres liaisons du module.

Le DS18B20 est aussi vendu avec une résistance 4,7 kΩ et sous la forme d'un boîtier étanche. Pour cette expérience, nous préférons la puce nue (qui ressemble à un transistor). Mais si vous voulez fabriquer un vrai thermostat, la version ayant la forme d'une sonde étanche, comme celle du projet « Système thermostatique de réfrigération de boisson », plus loin, est idéale.

Montage

La figure 12-3 présente le schéma du circuit de l'expérience.

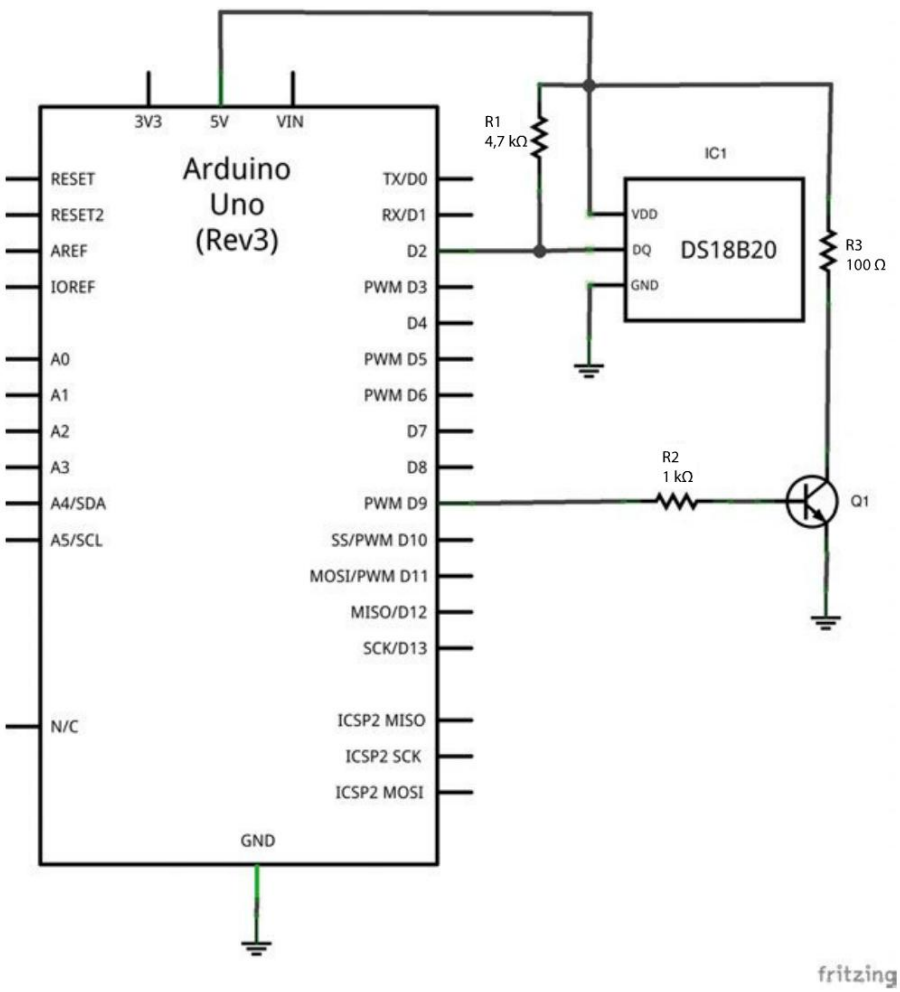


Figure 12-3. *Schéma du circuit de l'expérience de thermostat marche/arrêt*

Le circuit est en deux parties : le capteur de température DS18B20, qui nécessite une résistance R1 pull-up 4,7 k, et la partie chauffage du circuit centrée autour du transistor Q1. R2 limite le courant transmis à la base du transistor ; R3 est la résistance utilisée comme élément chauffant.

DS18B20

Le module DS18B20 est excellent : non seulement, il est assez précis (±0,5 °C), mais il vous permet aussi de connecter plusieurs composants sur une seule broche de l'Arduino ou du Raspberry Pi.

Pour savoir comment connecter plusieurs capteurs DS18B20 et les commander individuellement dans un sketch Arduino, reportez-vous au site de Miles Burton (*https://milesburton.com/Dallas_Temperature_Control_Library*).

Lorsque vous extrayez les données de plusieurs capteurs sur un Raspberry Pi, vous devez utiliser un autre identifiant de composant à l'ouverture du fichier de composant, comme décrit à la section « Programme Raspberry Pi », un peu plus loin dans ce chapitre page 244.

Réalisation du circuit

La figure 12-4 montre la réalisation du circuit de l'expérience.

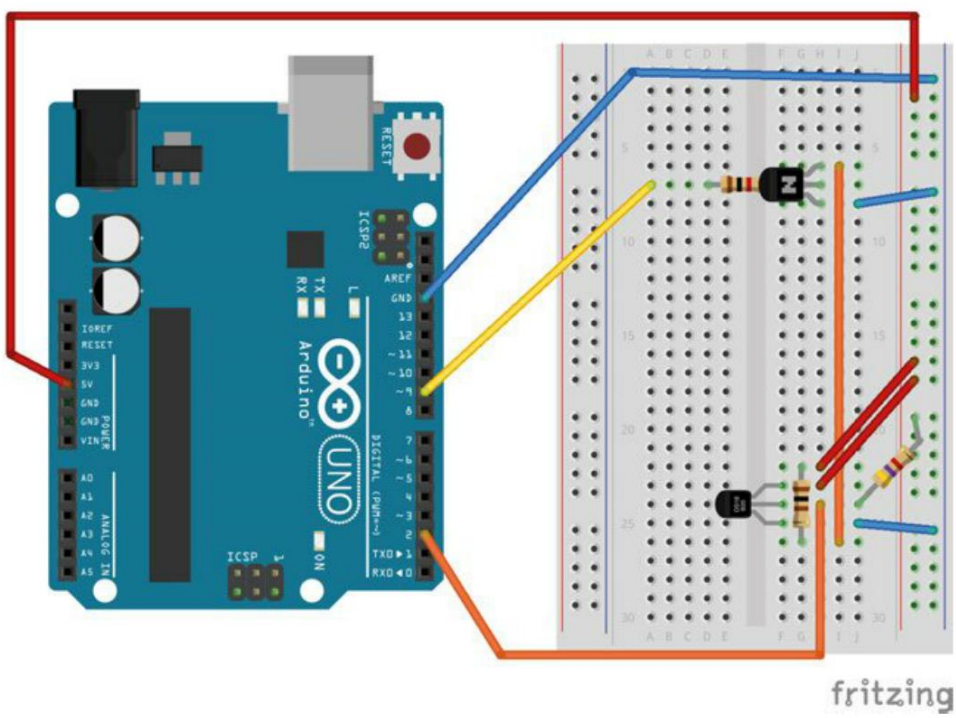


Figure 12-4. *Réalisation du circuit du thermostat marche/arrêt*

La résistance chauffante est placée du côté plat du DS18B20 de façon à ce qu'ils puissent être maintenus ensemble en se touchant à l'aide d'une pince (figure 12-5).

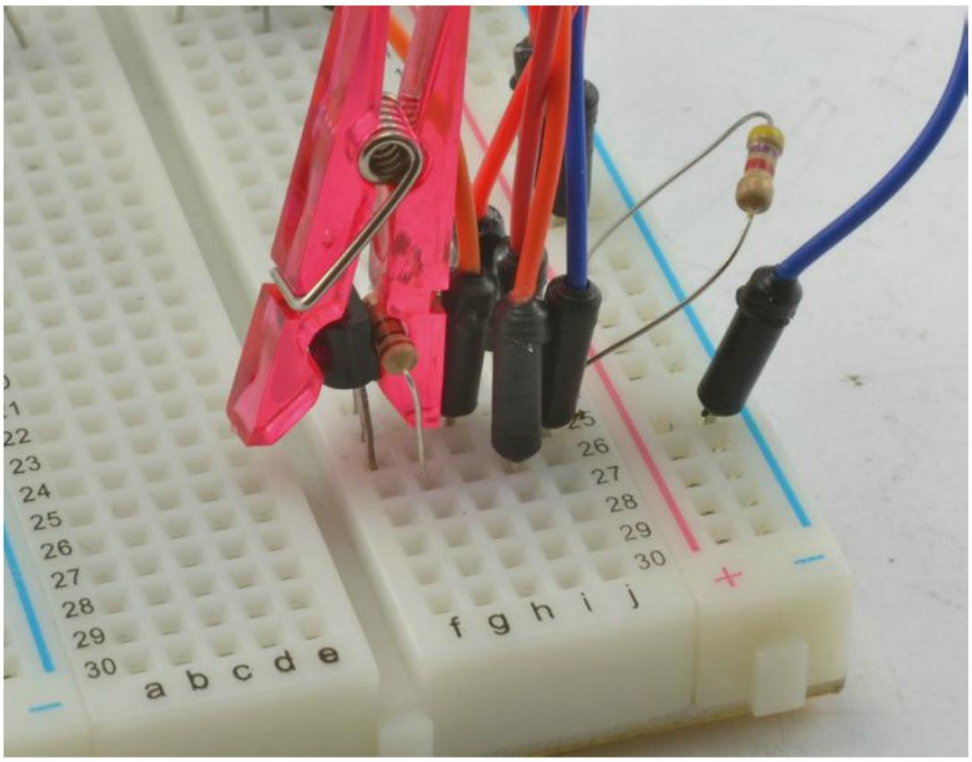


Figure 12-5. *Une petite pince maintient la résistance et le DS18B20*

Deux broches Arduino sont utilisées : la broche D9 contrôle le transistor et sert à allumer/éteindre l'alimentation de la résistance chauffante ; la broche D2 est l'entrée du capteur numérique.

Positionnez soigneusement le transistor et la puce DS18B20. Les deux composants sont placés avec leur côté plat vers la droite de la plaque d'essais.

Programme

Le DB18B20 utilise un type de bus intitulé 1-wire. Pour utiliser ce capteur avec un Arduino, vous devez télécharger deux bibliothèques Arduino et les installer dans votre environnement Arduino.

Téléchargez tout d'abord la bibliothèque OneWire. Décompressez l'archive téléchargée, installez le dossier intitulé `OneWire` dans votre environnement Arduino (voir l'encadré « Installation des bibliothèques Arduino » page suivante).

Vous pouvez aussi télécharger la deuxième bibliothèque du DS18B20 proprement dit. Cliquez sur le bouton Download ZIP sur la page GitHub. Une fois la bibliothèque extraite, vous obtenez un dossier intitulé `dallas-temperature-control` que vous devez renommer `DallasTemperature` avant de le déplacer dans le dossier `libraries`.

Installation des bibliothèques Arduino

L'IDE Arduino comporte de nombreuses bibliothèques préinstallées, mais vous aurez parfois besoin de télécharger une bibliothèque qui n'est pas incluse dans l'IDE. La procédure est assez simple : la bibliothèque est généralement téléchargée sous la forme d'une archive ZIP dont est extrait un dossier. Le nom du dossier correspond généralement à celui de la bibliothèque, mais ce n'est pas toujours le cas, surtout lorsque le dossier a été téléchargé depuis GitHub.

Après avoir renommé le dossier (si nécessaire), placez-le dans le dossier `arduino/libraries`.

L'IDE Arduino a automatiquement créé le dossier `arduino` dans `Mes Documents` (sous Windows) ou `Documents` (sous Mac ou Linux). C'est là qu'Arduino conserve vos sketches.

Si c'est la première bibliothèque que vous créez, vous devez préalablement créer un dossier intitulé `libraries` dans le dossier `arduino` avant d'y placer le dossier téléchargé.

Quittez et redémarrez l'IDE Arduino après avoir installé une nouvelle bibliothèque afin qu'elle soit reconnue.

Vous trouverez le sketch Arduino de ce projet dans le dossier `arduino/experiments/simple_thermostat` à l'endroit où vous avez téléchargé le code du livre (voir la section « Le code du livre » du chapitre 2 page 14).

```
#include <OneWire.h>
#include <DallasTemperature.h>

const int tempPin = 2;          ❶
const int heatPin = 9;
const long period = 1000;       ❷

OneWire oneWire(tempPin);       ❸
DallasTemperature sensors(&oneWire);

float setTemp = 0.0;            ❹
long lastSampleTime = 0;

void setup() {
  pinMode(heatPin, OUTPUT);
  Serial.begin(9600);
  Serial.println(«t30 - règle la température sur 30»);
  sensors.begin();              ❺
}

void loop() {
  if (Serial.available()) {     ❻
```

```
    char command = Serial.read();
    if (command == 't') {
    setTemp = Serial.parseInt();
    Serial.print(«Règle Temp=»);
    Serial.println(setTemp);
  }
  }
  long now = millis();                    ❼
  if (now > lastSampleTime + period) {
    lastSampleTime = now;
    float measuredTemp = readTemp();      ❽
    float error = setTemp - measuredTemp;
    Serial.print(measuredTemp);
    Serial.print(", ");
    Serial.print(setTemp);
    if (error > 0) {                      ❾
      digitalWrite(heatPin, HIGH);
      Serial.println(", 1");
    }
    else {
      digitalWrite(heatPin, LOW);
      Serial.println(", 0");
    }
  }
}

float readTemp() {                        ❿
  sensors.requestTemperatures();
  return sensors.getTempCByIndex(0);
}
```

❶ Le code commence par définir des constantes pour la broche de mesure de la température (`tempPin`) et la broche de régulation du chauffage (`heatPin`).

❷ La constante `period` est utilisée pour définir l'intervalle entre deux mesures de la température. Le module DS18B20 n'est pas très rapide et peut prendre jusqu'à 750 millisecondes. Par conséquent, définissez une valeur de `period` supérieure à 750.

❸ Des variables doivent être définies pour pouvoir accéder aux bibliothèques DallasTemperature et OneWire.

❹ La variable `setTemp` contient la température ambiante souhaitée et la variable `lastSampleTime` consigne le moment où la dernière mesure a été effectuée.

❺ Initialise la bibliothèque DallasTemperature.

❻ La première partie de la fonction `loop()` surveille l'arrivée de commandes série transmises par le moniteur série. En fait, il n'y a qu'une seule commande possible (`'t'`) suivie de la température de consigne souhaitée en degrés Celsius. Quand la nouvelle valeur de `setTemp` a été définie, la nouvelle valeur est renvoyée au moniteur série pour confirmation.

❼ La seconde moitié de `loop()` vérifie tout d'abord s'il est temps d'effectuer une autre mesure – c'est-à-dire si le nombre de millisecondes définies par `period` est écoulé.

❽ La température est mesurée puis l'erreur est calculée. Ensuite, `measuredTemp` et `setTemp` sont transmis au moniteur série.

❾ Si l'erreur est supérieure à 0, le radiateur est allumé et la ligne du moniteur série se termine par `'1'` pour indiquer que la résistance chauffe ; sinon, le radiateur est éteint et un `'0'` est transmis.

❿ Comme vous pouvez connecter plusieurs DS18B20 à une seule broche d'entrée, la commande `requestTemperatures` demande à tous les DS18B20 connectés (ici, il n'y en a qu'un) de mesurer et de communiquer les températures mesurées. La fonction `getTempCByIndex` est ensuite utilisée pour transmettre la mesure du DS18B20.

Expérimentations

Téléversez le programme et ouvrez le moniteur série. Un flux régulier de mesures de température commence à s'afficher :

```
t30 - règle la température sur 30
21.75, 0.00, 0
21.69, 0.00, 0
21.75, 0.00, 0
21.69, 0.00, 0
21.75, 0.00, 0
```

Les trois colonnes correspondent à la température mesurée, la température de consigne (0 pour l'instant) et l'alimentation ou non du radiateur (0 ou 1).

Saisissez la commande `t30` pour définir la température voulue sur 30 °C. La température devrait immédiatement commencer à augmenter, car le radiateur est allumé, comme indiqué par un 1 dans la troisième colonne :

```
21.75, 0.00, 0
21.75, 0.00, 0
Règle Temp=30.00
21.75, 30.00, 1
21.75, 30.00, 1
21.81, 30.00, 1
21.94, 30.00, 1
22.06, 30.00, 1
```

Quand la température atteint 30 °C, le radiateur doit s'éteindre et vous devriez assister à un cycle au cours duquel le radiateur s'allume et s'éteint, car la température oscille autour de 30 °C :

```
29.87, 30.00, 1
29.94, 30.00, 1
29.94, 30.00, 1
30.00, 30.00, 0
30.06, 30.00, 0
30.12, 30.00, 0
```

```
30.06, 30.00, 0
29.94, 30.00, 1
29.81, 30.00, 1
29.75, 30.00, 1
```

Vous avez donc fabriqué un thermostat qui vous rendra de nombreux services. Toutefois, comme vous pouvez le constater, la température « oscille » autour de la température de consigne de 30 °C. Pour avoir une idée plus précise des variations de température, vous pouvez copier un extrait de ces données et les coller dans un fichier à l'aide d'un éditeur de texte. Utilisez l'extension de fichier `.csv`. Ensuite, vous pouvez les importer dans votre tableur préféré et créer un graphique, comme à la figure 12-6.

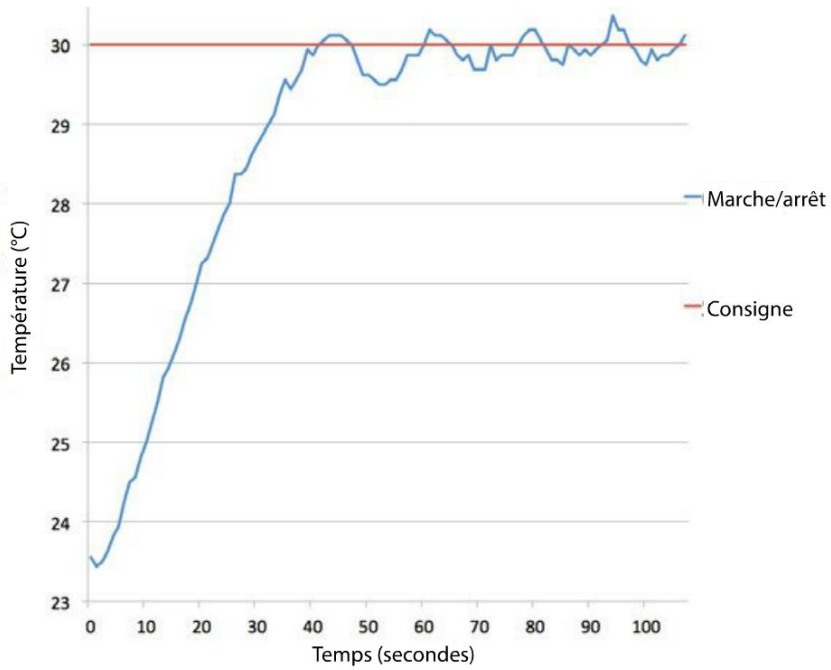


Figure 12-6. *Suivi de température d'un simple thermostat marche/arrêt*

Comme vous pouvez le constater, la température fluctue de près d'un demi-degré au-dessus et en dessous de la température de consigne. Nous pouvons considérablement améliorer ces résultats à l'aide d'une technique de régulation dite Proportionnelle-Intégrale-Dérivée ou PID. Bien que cette technique puisse faire peur de prime abord, les calculs peuvent être simplifiés.

Hystérésis

Un transistor permet de commuter très rapidement l'électricité et d'utiliser la MLI. Toutefois, certains systèmes ne supportent pas bien que l'alimentation soit allumée et éteinte aussi rapidement. C'est notamment le cas des chaudières domestiques qui ne produisent pas de la

chaleur immédiatement après avoir été allumées. En outre, comme elles sont régulées à l'aide de valves électromécaniques et d'autres pièces, leur durée de vie serait considérablement réduite si elles étaient constamment allumées et éteintes.

Pour empêcher une commutation trop rapide, vous pouvez définir une durée de marche minimale et empêcher l'arrêt de l'appareil avant l'expiration de ce délai. Ou bien, vous pouvez faire appel à l'hystérésis (figure 12-7).

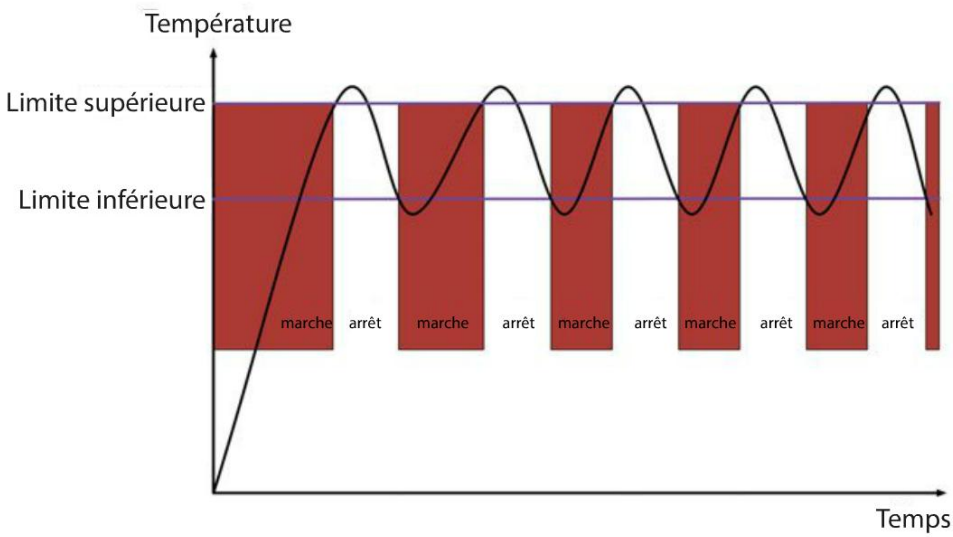


Figure 12-7. *Hystérésis dans un thermostat*

Dans le cas d'un thermostat, l'hystérésis équivaut à définir deux seuils de température au lieu d'un, l'un des seuils étant équivalent à une quantité fixe au-dessus du précédent. Si la température chute au-dessous du seuil le plus bas, le radiateur est allumé. Ensuite, il est uniquement éteint après avoir dépassé le seuil supérieur.

Avec cette méthode, l'inertie naturelle du système est utilisée pour introduire des délais dans la commutation.

Régulation PID

Lorsque le radiateur est éteint quand le capteur de température indique qu'il fait plus chaud que la température de consigne et allumé quand il fait plus froid, la température a tendance à fluctuer autour de la température de consigne, comme indiqué à la figure 12-6.

Si la température de votre système doit être maintenue de façon plus précise, vous devez utiliser la régulation Proportionnelle-Intégrale-Dérivée ou PID.

Au lieu d'allumer ou d'éteindre un radiateur, un régulateur PID varie la puissance transmise au radiateur (ou à d'autres actionneurs) en tenant compte de trois facteurs de nature proportionnelle, intégrale et dérivée.

Attention : cette partie est très théorique. On me pose beaucoup de questions sur la régulation PID et l'un des principaux objectifs de ce chapitre est d'expliquer le fonctionnement et l'utilisation d'un régulateur PID.

Proportionnel (P)

Vous obtiendrez d'assez bons résultats avec de nombreux systèmes utilisant simplement la partie P d'un programme de régulation PID et ignorant les aspects I et D.

La régulation proportionnelle signifie simplement que la puissance transmise au radiateur est proportionnelle à l'écart ou l'erreur de température. Par conséquent, plus l'écart est important – c'est-à-dire plus la température ambiante est éloignée de la température de consigne – plus il y a d'électricité transmise au radiateur. Lorsque la température ambiante se rapproche de la température de consigne, la puissance diminue de sorte que la température ne dépasse pas autant qu'à la figure 12-6. Elle dépassera toujours un peu, mais pas autant qu'avec une simple commande marche/arrêt. C'est un peu comme lorsque l'on veut s'arrêter à un stop : il faut anticiper l'arrêt et freiner avant d'atteindre le panneau plutôt que de piler sur la ligne blanche.

Si la température ambiante est supérieure à la température de consigne, l'écart sera négatif et il faudra donc transmettre une valeur de puissance négative au radiateur. Si l'appareil intègre un élément Peltier (chapitre 11), vous pourriez inverser le courant qui le traverse (à l'aide d'un pont en H) et commencer à refroidir au lieu de chauffer. En pratique, on ne procède généralement pas ainsi, à moins que la température ambiante soit très proche de la température de consigne. On se contente généralement de laisser le système refroidir (ou se réchauffer) naturellement.

Dans le cas d'un thermostat, l'écart sur lequel sont basés les calculs pour la puissance de sortie est une différence de température. Elle peut être exprimée en degrés Celsius. Par conséquent, si la température de consigne est de 30 °C et la température mesurée est de 25 °C, alors l'écart est de 5 °C (30-25 °C). Si vous utilisez la MLI avec une carte Arduino pour définir la puissance de sortie, la sortie attend une valeur comprise entre 0° et 255 °C. Si vous réglez directement la sortie sur la valeur de l'écart (5), très peu de puissance serait transmise au radiateur. Elle ne serait probablement pas suffisante pour faire monter la température jusqu'à 30 °C. C'est pourquoi l'erreur est multipliée par un nombre nommé kp (ou gain) pour obtenir la puissance de sortie. Le réglage de kp détermine la vitesse à laquelle la température montera pour atteindre la température de consigne. Si la valeur kp est faible, il est possible que la température n'atteigne jamais la température de consigne. Si la valeur kp est trop élevée, le système se comportera exactement comme le régulateur de température marche/arrêt et oscillera autour de la température de consigne. La figure 12-8 montre l'influence de différentes valeurs de kp sur le comportement d'un système idéal.

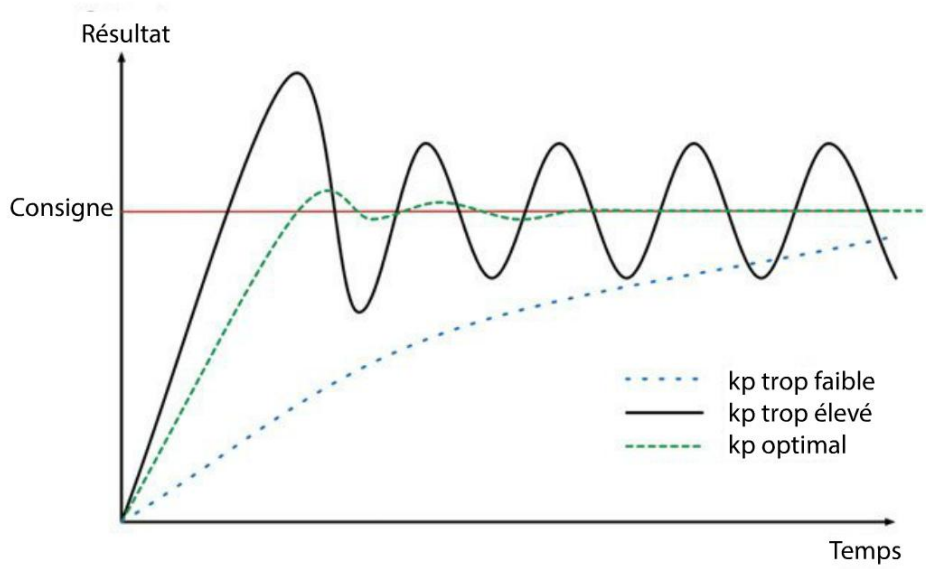


Figure 12-8. *L'effet de kp (gain) sur la sortie d'un régulateur proportionnel*

Si le gain n'est pas assez élevé, la consigne de sortie ne sera jamais atteinte ou cela mettra beaucoup de temps. Par ailleurs, si kp est trop élevé, la sortie oscille et l'amplitude de ces oscillations ne diminuera pas. Une valeur de gain adaptée fera rapidement augmenter la sortie au niveau de la consigne. Peut-être sera-t-elle légèrement dépassée, mais les oscillations s'atténueront rapidement à un niveau acceptable.

Pour déterminer la valeur de gain correct, on procède au réglage du système. Dans la section « Réglage d'un régulateur PID » page suivante, nous verrons comment syntoniser un système de régulation PID.

Reprenons l'exemple d'une température de consigne de 30 °C et d'une température ambiante de 25 °C. Pour l'instant, nous voulons toujours qu'une puissance maximale soit transmise au radiateur – c'est-à-dire un rapport cyclique de 255 (100 %). Si nous avions défini une valeur de gain de 50, l'erreur de 5 aboutit à la puissance de sortie suivante :

$$\text{Sortie} = \text{erreur} \times \text{kp} = \times 50 = 250$$

Une fois que le système n'est plus qu'à 1° de la température de consigne, la puissance de sortie passe à :

$$\text{Sortie} = \text{erreur} \times \text{kp} = 1 \times 50 = 50$$

Cette puissance ne suffit pas toujours pour permettre au système d'atteindre la température de consigne. Tous les systèmes sont différents, d'où la nécessité de régler le régulateur.

Intégral (I)

Si la régulation de puissance proportionnelle de la sortie n'est pas assez précise, vous pouvez ajouter l'aspect I aux calculs. La puissance de sortie est alors calculée comme suit :

$$\text{Sortie} = \text{erreur} \times \text{kp} + \text{I} \times \text{ki}$$

La nouvelle constante ki pondère cette mystérieuse nouvelle propriété nommée I (intégrale). La puissance de sortie est calculée en ajoutant l'aspect intégral à l'aspect proportionnel. Ce type de régulateur se nomme un régulateur PI (parfois P + I) (à ne pas confondre avec le Pi du Raspberry Pi). Comme les régulateurs uniquement proportionnels, les régulateurs PI peuvent produire des résultats satisfaisants sans avoir à ajouter l'aspect D.

L'aspect intégral est calculé en totalisant les erreurs à chaque mesure de la température. Tant que l'erreur est positive (réchauffement), l'aspect I ne cesse de croître et gonfle la réponse initiale. Il commence uniquement à diminuer lorsque l'erreur devient négative, c'est-à-dire lorsque la température de consigne a été dépassée.

Lorsque la température ambiante atteint la température de consigne, l'aspect I exerce un effet calmant et lisse les fluctuations de la température afin qu'elle se stabilise à la valeur voulue.

Le seul problème posé par le parameter I est lié au fait que la puissance augmente fortement lorsque la température augmente. Par conséquent, la température risque d'être largement dépassée. Il faudra donc attendre un certain temps avant que la température de consigne ne soit atteinte et que la température ne se stabilise.

Dérivé (D)

En pratique, l'aspect D n'est souvent pas utilisé dans les véritables systèmes de régulation, car ses avantages en matière de réduction du dépassement sont compensés par un réglage plus difficile.

Pour contrer l'effet de dépassement, un aspect D peut être ajouté au programme de régulation. La puissance de sortie est alors calculée comme suit :

$$\text{Sortie} = \text{erreur} \times \text{kp} + \text{I} \times \text{ki} + \text{D} \times \text{kd}$$

Cet aspect D est une mesure de la vitesse de fluctuation de l'erreur entre chaque mesure de la température. D'une certaine façon, il permet de prédire l'évolution de la température.

Réglage d'un régulateur PID

L'ajustement d'un régulateur PID consiste à déterminer les valeurs de kp, ki et kd qui font que le système se comporte comme vous l'entendez. Je vous conseille de vous simplifier la vie et de vous contenter de la régulation PI en définissant kd sur 0. Il ne vous reste alors plus que deux paramètres à ajuster. En fait, cette simplification facilite la syntonisation de la majorité des systèmes, mais cela n'en reste pas moins une opération fastidieuse.

Dans l'expérience « Régulation thermostatique PID », plus loin, je présenterai une méthode de réglage basée sur des essais qui fonctionne bien avec le type de thermostat utilisé. Dans cette section, nous étudierons l'une des techniques de réglage PID les plus répandues : la méthode de Ziegler-Nichols. Le processus se réduit essentiellement à une série d'essais, puis à quelques calculs simples pour obtenir les valeurs de kp, ki et kd.

Avec la syntonisation par la méthode de Ziegler-Nichols, on commence par régler ki et kd sur 0 afin que le régulateur fonctionne en mode proportionnel. Ensuite, on augmente régulièrement la valeur de kp jusqu'à ce que la sortie commence à osciller. La valeur de kp à laquelle cette oscillation se produit se nomme ku.

Après avoir défini ku, vous devez mesurer la période d'oscillation en secondes (figure 12-9), cette dernière se nomme pu.

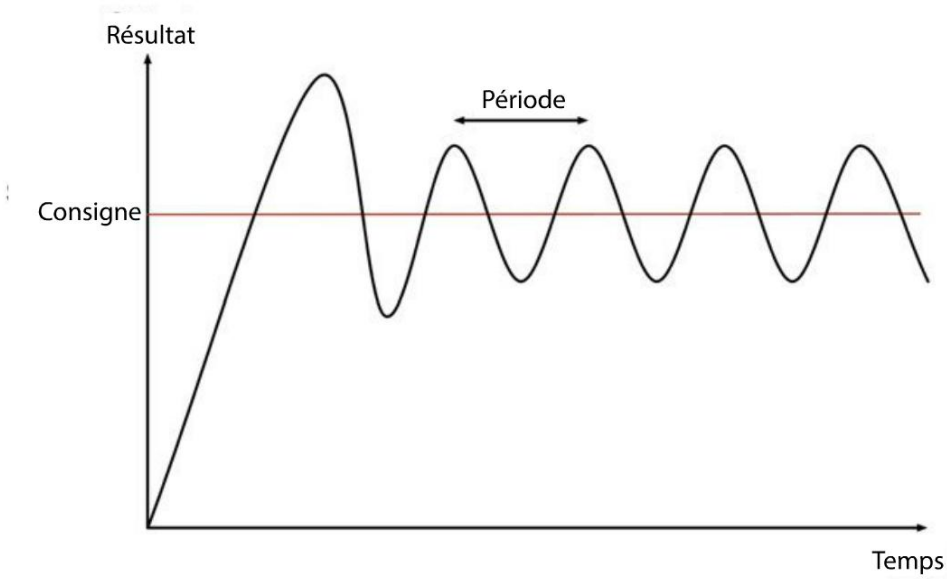


Figure 12-9. *Détermination de la période d'oscillation*

Pour y parvenir, vous devez reporter les mesures sur un graphique, comme vous l'avez fait à l'expérience « Précision de la régulation thermostatique marche/arrêt », au début de ce chapitre.

Ensuite, vous pouvez calculer les valeurs de kp, ki et kd comme suit :

$$kp = 0{,}6 \times ku$$

$$ki = (2 \times kp)/pu$$

$$kd = (kp \times pu)/8$$

Si le régulateur est uniquement PI, effectuez les calculs suivants :

$$Kp = 0{,}45 \times ku$$

$$ki = (1{,}2 \times kp)/pu$$

$$kd = 0$$

Vous trouverez plus d'informations sur la méthode de Ziegler-Nichols sur Wikipédia (*https://fr.wikipedia.org/wiki/Méthode_de_Ziegler-Nichols*).

Expérience : régulation thermostatique PID

Dans cette expérience, vous testerez la régulation PID avec un Arduino et un Raspberry Pi. Le programme de régulation PID Arduino provient de la bibliothèque, mais celui de la version Raspberry Pi a été totalement créé.

Matériel

Cette expérience utilise exactement le même matériel que l'expérience « Précision de la régulation thermostatique marche/arrêt », au début de ce chapitre. Toutefois, soyez prudent au moment de la connexion au Raspberry Pi, car le module DS18B20 et la résistance chauffante doivent être alimentés séparément sur le Pi : le DS18B20 en 3,3 V et la résistance chauffante en 5 V. C'est dû au fait que la sortie numérique du module DS18B20 doit être tirée à 3,3 V au lieu de 5 V pour ne pas risquer d'endommager la broche d'entrée du Raspberry Pi.

Programme Arduino

Cette expérience utilise une bibliothèque PID prête à l'emploi que vous devez télécharger et installer comme expliqué dans l'encadré « Installation des bibliothèques Arduino », plus haut. Vous trouverez une documentation complète sur cette bibliothèque sur le site web Arduino (*http://playground.arduino.cc/Code/PIDLibrary*).

Le sketch Arduino de l'expérience se trouve dans `/arduino/experiments/pid_thermostat` :

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <PID_v1.h>

const int tempPin = 2;
```

```
const int heatPin = 9;
const long period = 1000; // >750

double kp = 0.0;      ❶
double kd = 0.0;
double ki = 0.0;

OneWire oneWire(tempPin);
DallasTemperature sensors(&oneWire);

double setTemp = 0.0;
double measuredTemp = 0.0;
double outputPower = 0.0;    ❷
long lastSampleTime = 0;

PID myPID(&measuredTemp, &outputPower, &setTemp, kp, ki, kd, DIRECT);  ❸

void setup() {
  pinMode(heatPin, OUTPUT);
  Serial.begin(9600);
  Serial.println(«t30 - règle la température sur 30»);
  Serial.println(«k50 20 10 - règle kp, ki et kd respectivement»);
  sensors.begin();
  myPID.SetSampleTime(1000);  ❹
  myPID.SetMode(AUTOMATIC);
}

void loop() {
  checkForSerialCommands();   ❺
  long now = millis();
  if (now > lastSampleTime + period) {   ❻
    lastSampleTime = now;
    measuredTemp = readTemp();
    myPID.Compute();
    analogWrite(heatPin, outputPower);

    Serial.print(measuredTemp);  ❼
    Serial.print(“, “);
    Serial.print(setTemp);
    Serial.print(“, “);
    Serial.println(outputPower);
  }
}

void checkForSerialCommands() {  ❽
    if (Serial.available()) {
    char command = Serial.read();
    if (command == ‘t’) {
      setTemp = Serial.parseFloat();
      Serial.print(“Règle Temp=”);
      Serial.println(setTemp);
    }
    if (command == ‘k’) {
```

```
        kp = Serial.parseFloat();
        ki = Serial.parseFloat();
        kd = Serial.parseFloat();
        myPID.SetTunings(kp, ki, kd);
        Serial.print(«Règle les constantes kp=»);
        Serial.print(kp);
        Serial.print(“ ki=”);
        Serial.print(ki);
        Serial.print(“ kd=”);
        Serial.println(kd);
      }
    }
}

double readTemp() {
  sensors.requestTemperatures();
  return sensors.getTempCByIndex(0);
}
```

Le code chargé d'obtenir une mesure de la température à l'aide du module DS18B20 est identique à celui de l'expérience « Précision de la régulation thermostatique marche/arrêt », plus haut, donc nous nous concentrerons sur le code relatif à la régulation PID.

❶ Trois variables (`kp`, `ki` et `kd`) sont définies en tant que facteurs multiplicateurs pour les aspects P, I et D de la sortie. Des variables sont utilisées parce qu'elles peuvent être corrigées à l'aide d'une commande saisie dans le moniteur série durant l'exécution du programme. Le type `double` est utilisé au lieu de `float`, non seulement parce que les nombres sont plus précis, mais parce que la bibliothèque l'exige.

❷ La variable `outputPower` contient le rapport cyclique du radiateur compris entre 0 et 255.

❸ Une variable nommée `myPID` est définie pour accéder au code de la bibliothèque PID. Vous remarquerez que les trois premiers paramètres de création d'une variable PID portent le nom des variables `measuredTemp`, `outputPower` et `setTemp` précédées du préfixe `&`. Cette astuce propre à C permet à la bibliothèque PID de modifier les valeurs de ces variables même si celles-ci ne font pas partie de la bibliothèque. Pour plus d'informations sur cette technique (pointeurs C), consultez le site Tutorials Point (*http://www.tutorialspoint.com/cprogramming/c_pointers.htm*). Le dernier paramètre (`DIRECT`) définit le mode de fonctionnement PID direct. Avec cette bibliothèque, cela signifie que la sortie est proportionnelle à l'erreur au lieu d'être inversée. En général, cette bibliothèque règle la sortie sur une valeur comprise entre 0 et 255 en MLI.

❹ La période d'échantillonnage doit être définie sur 1 seconde (1 000 millisecondes). Le réglage du mode `AUTOMATIC` démarre les calculs PID.

❺ La surveillance de l'arrivée de commandes série est définie comme une fonction distincte afin d'éviter d'allonger la fonction `loop()` et la rendre peu lisible. Voir aussi `callout` (8).

❻ Si le moment est venu de prélever un autre échantillon, la température est lue dans la variable `measuredTemp` puis la bibliothèque PID actualise ses calculs (`myPID.Compute`). La valeur d'`outputPower` est automatiquement actualisée et elle sera ensuite utilisée pour définir le rapport cyclique de la broche utilisée pour piloter la résistance de chauffage.

❼ Les valeurs sont toutes communiquées au moniteur série, car nous en aurons besoin pour créer des graphiques afin de vérifier les performances du régulateur.

❽ La fonction `checkForSerialCommands` vérifie la présence d'une commande `'t'` afin de régler la température, comme dans l'expérience « Précision de la régulation thermostatique marche/arrêt », plus haut. En outre, la fonction vérifie la réception éventuelle de la commande `k` suivi de trois nombres (`kp`, `ki` et `kd`) afin de définir ces réglages si cette commande a été reçue.

Expérimentation Arduino

Les composants matériels et logiciels de cette expérience réunissent tous les éléments nécessaires pour tester un régulateur PID. Il est possible de changer la température de consigne ainsi que les trois valeurs de `kp`, `ki` et `kd` afin d'enregistrer leur effet sur la sortie. Les résultats fournis par la régulation PI seront suffisants donc nous pouvons nous contenter de régler `kd` sur 0.

Téléversez le sketch sur l'Arduino et ouvrez le moniteur série (figure 12-10).

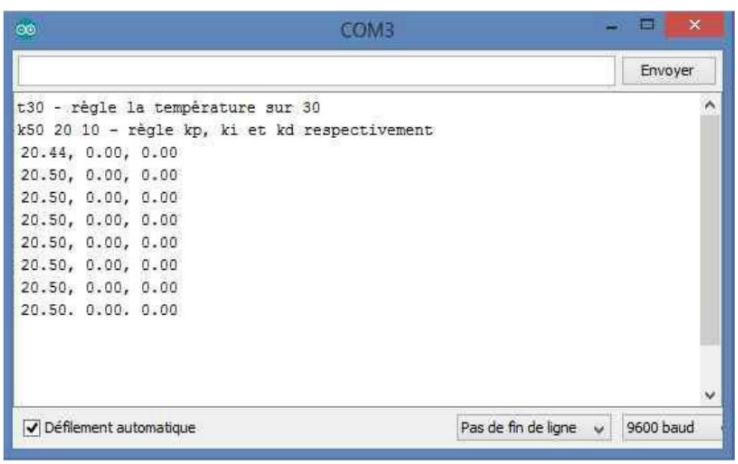


Figure 12-10. *Utilisation du moniteur série pour tester la régulation PID*

L'ajustement du régulateur est assez longue. Vous devez enregistrer les données et les afficher sous la forme d'un graphique pour vérifier le comportement du système. Pour commencer, il faut définir une valeur adaptée pour `kp`. Vous allez d'abord tester les valeurs 50, 500 et 5 000 pour vérifier l'effet produit.

Réglez les paramètres de syntonisation en saisissant la ligne suivante dans le moniteur série :

```
k50 0 0
```

Cela définit `kp` sur 50 et `ki` et `kd` sur 0. Si vous le souhaitez, vous pouvez saisir les valeurs avec une décimale (par exemple, k30.0 0.0 0.0). Dans les deux cas, les nombres seront convertis en virgule flottante.

Ensuite, définissez la température voulue sur 30 °C (cette température m'a paru pratique, car elle se situe à 7° ou 8° au-dessus de la température ambiante) :

```
t30
```

La température devrait commencer à augmenter. Les trois colonnes affichent la température courante, la température de consigne et la valeur de sortie PWM (0 à 255). Dans les données ci-dessous, notez que la sortie PWM est conservée en tant que `float`. Il y a donc des chiffres après le point décimal. Ce nombre à virgule flottante sera tronqué en nombre entier compris entre 0 et 255 lorsque la fonction `analogWrite` sera appelée :

```
25.06, 30.00, 246.88
25.19, 30.00, 240.63
25.31, 30.00, 234.38
25.44, 30.00, 228.13
```

La dernière colonne (la valeur PWM) diminue presque aussitôt que `kp` est réglé sur 50. Cela signifie que 50 est trop bas, mais poursuivez la collecte de données pendant quelques minutes. Ensuite, collez les données dans un fichier texte et importez-les dans votre tableur. J'ai commencé à copier les données lorsque la température a dépassé 25 °C. Vous devriez obtenir un graphique ressemblant à celui illustré à la figure 12-11.

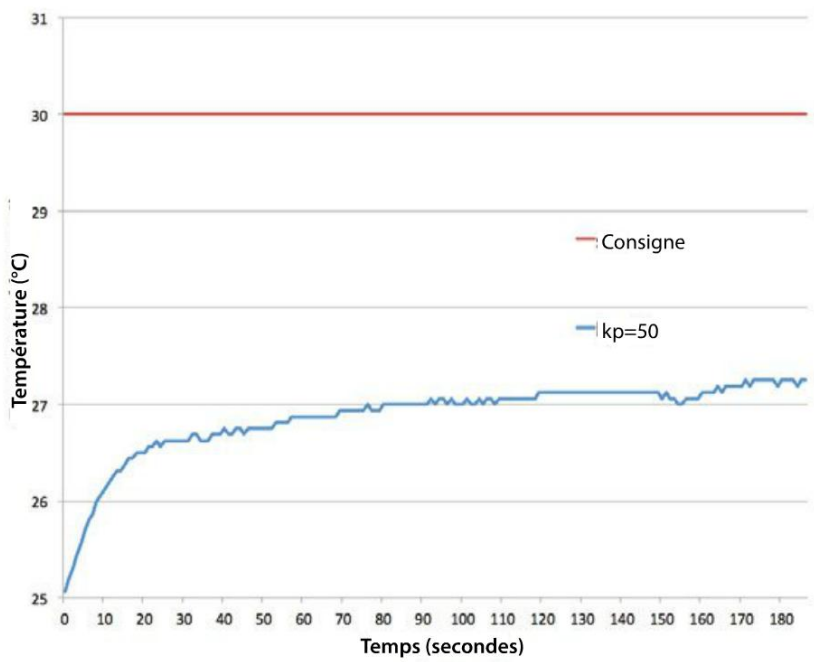


Figure 12-11. *Évolution de la température dans le temps pour kp = 50*

Importation de données dans une feuille de calcul

Il est très utile de pouvoir importer les données brutes depuis le moniteur série dans un tableur afin de créer un graphique pour visualiser l'évolution des données.

Si vous utilisez OpenOffice, sélectionnez les données dans le moniteur série, puis saisissez CTRL-C dans Windows et Linux ou Command-C sur un Mac pour copier les données dans le Presse-papier. Basculez dans une feuille de calcul ouverte dans OpenOffice, sélectionnez la cellule dans laquelle vous voulez insérer les données, puis saisissez CTRL-V dans Windows et Linux ou Command-V sur un Mac.

Comme les données sont réparties dans plusieurs colonnes, OpenOffice affiche la boîte de dialogue illustrée à la figure 12-12 pour définir la séparation des colonnes.

Sélectionnez l'option Comma dans la rubrique Separated by. Lorsque vous appuyez sur OK, les données seront collées dans des colonnes distinctes de la feuille de calcul.

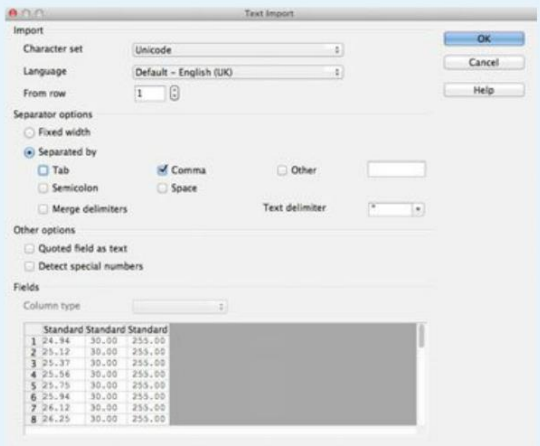


Figure 12-12. *Importation des données dans OpenOffice*

Si vous utilisez Microsoft Excel, vous devrez utiliser un éditeur de texte comme Notepad++ ou Textmate. Collez tout d'abord les données dans un nouveau document texte, puis enregistrez le fichier au format `.csv` (valeurs séparées par des virgules). Vous pourrez ensuite ouvrir directement le fichier dans Excel. Vous pouvez aussi utiliser la commande Importer d'Excel pour importer le fichier.

Avec une valeur `kp` de 50, il est peu probable que la température parvienne à monter jusqu'à 30 °C. Réglez la température sur 0 (`t0` dans le moniteur série) et attendez que le système refroidisse. Ensuite, répétez d'abord la procédure pour un `kp` de 500, puis de 5 000. Les résultats des trois valeurs de `kp` sont illustrés à la figure 12-13.

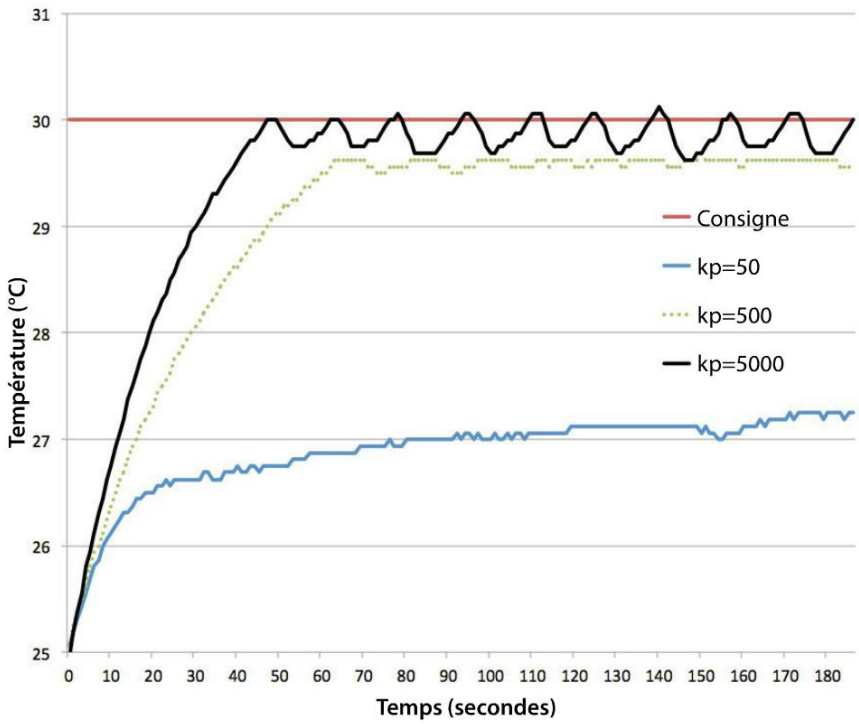


Figure 12-13. *Évolution de la température dans le temps pour trois valeurs de kp*

Comme on pouvait s'y attendre, un kp de 50 est beaucoup trop bas, 500 n'est pas trop mal, mais cela ne permet pas tout à fait d'atteindre la température de consigne, et à 5 000, le système se comporte comme un thermostat marche/arrêt. Il semblerait que 700 soit une bonne valeur de kp, surtout si la hausse de température est aidée par une forte augmentation de la puissance pour atteindre la consigne en ajoutant un peu de I.

La méthode de réglage Ziegler-Nichols suggère une valeur de ki pour un régulateur PI de :

$$ki = (1{,}2 \times kp)/pu$$

Nous avons estimé un kp de 700 et, d'après la figure 12-13, pour une durée d'environ 15 secondes, cela suggère un ki de 56.

Enregistrez une autre série de données avec kp = 700 et ki = 56. Les résultats sont illustrés à la figure 12-14, avec les résultats de la régulation proportionnelle uniquement pour kp = 700.

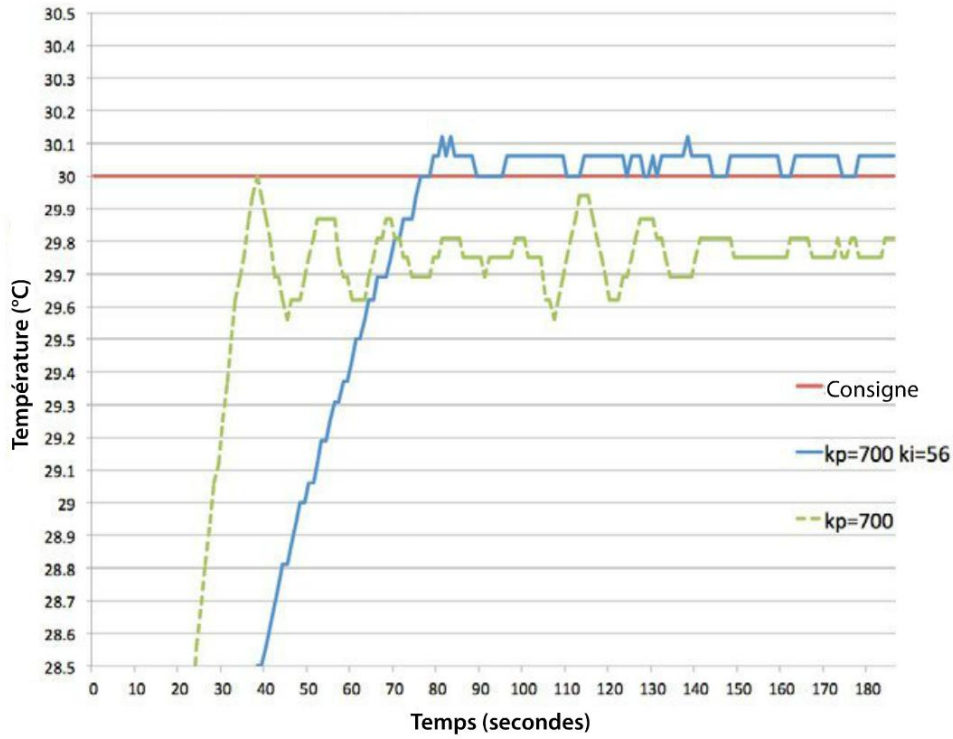


Figure 12-14. *Évolution de la température dans le temps pour la régulation P et PI*

La figure 12-14 présente une vue grossie de l'axe Y, ce qui permet d'apprécier les excellents résultats obtenus pour la régulation PI.

Lorsque la courbe PI atteint la température de consigne, la variation n'est plus que de l'ordre de 0,1°. La température fluctue par pas, et non de façon lisse, car le module DS18B20 est un système numérique avec une précision fixe.

Avec plus d'efforts et d'expérimentations, il serait sans doute possible d'améliorer encore ces résultats, mais cela n'en vaut pas vraiment la peine.

Raccordement du Raspberry Pi

La réalisation du circuit sur le Raspberry Pi (figure 12-15) est légèrement différente de celui de l'Arduino. Bien que la résistance chauffante nécessite une tension de 5 V, le module DS18B20 du capteur de température nécessite une tension de 3,3 V pour être compatible avec les broches GPIO du Raspberry Pi.

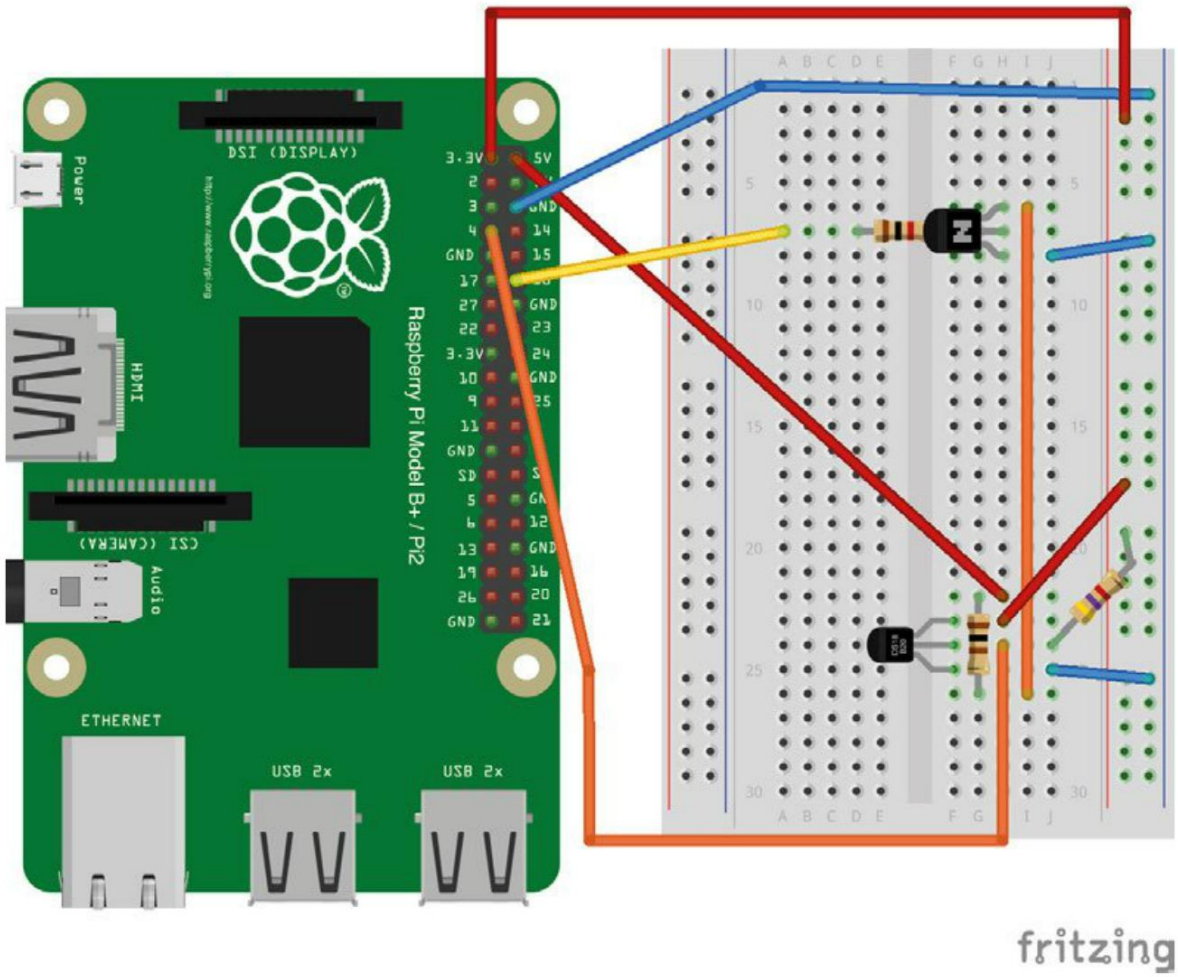


Figure 12-15. *Réalisation du circuit et connexions au Raspberry Pi*

Programme Raspberry Pi

Quelques préparatifs sont nécessaires pour faire fonctionner le module DS18B20 avec le Raspberry Pi. La première étape consiste à activer le bus 1-wire. Ouvrez le fichier `/boot/config.txt` à l'aide de la commande suivante :

```
$ sudo nano /boot/config.txt
```

Ajoutez la ligne suivante à la fin du fichier :

```
dtoverlay=w1-gpio
```

Ensuite, vous devez redémarrer votre Raspberry Pi pour que les modifications prennent effet. Le DS18B20 utilise une interface de style fichier texte, ce qui signifie que le programme Python devra lire un fichier pour en extraire la température mesurée. Faites un essai avant d'exécuter l'intégralité du programme afin de vous familiariser avec le format du message en ouvrant le dossier `/sys/bus/w1/devices` à l'aide de la commande suivante :

```
$ cd /sys/bus/w1/devices
```

Affichez la liste des répertoires contenue dans ce dossier à l'aide de la commande suivante :

```
$ ls
28-000002ecba60 w1_bus_master1
pi@raspberrypi /sys/bus/w1/devices $
```

Ouvrez le répertoire dont le nom commence par 28. Ici, `28-000002ecba60` (notez que le vôtre aura probablement un nom différent) :

```
$ cd 28-000002ecba60
```

Enfin, exécutez la commande suivante pour récupérer la dernière température mesurée :

```
$ cat w1_slave
53 01 4b 46 7f ff 0d 10 e9 : crc=e9 YES
53 01 4b 46 7f ff 0d 10 e9 t=21187
pi@raspberrypi /sys/bus/w1/devices/28-000002ecba60 $
```

La réponse se présente sous la forme de deux lignes. La première partie de chaque ligne est l'identifiant unique du capteur de température et la première ligne se termine par YES, ce qui indique que la mesure a réussi. La deuxième ligne se termine par la température en 1/1 000e degrés C. Ici, 21 187 /(soit 21,187 °C).

Même s'il existe des bibliothèques Python pour la régulation PID, elles ne sont pas aussi simples d'emploi que leur équivalent Arduino. Pour la version Raspberry Pi, l'algorithme PID est implémenté en recommençant à zéro (enfin, pas tout à fait puisque le code a été écrit en se référant à la bibliothèque Arduino afin que le comportement des deux versions soit aussi similaire que possible).

Vous trouverez le code dans le fichier `pid_thermostat.py` (qui se trouve dans le dossier `python/experiments`) :

```
import os
import glob
import time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

heat_pin = 18
base_dir = '/sys/bus/w1/devices/'      ❶
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

GPIO.setup(heat_pin, GPIO.OUT)
heat_pwm = GPIO.PWM(heat_pin, 500)
```

```
heat_pwm.start(0)

old_error = 0    ❷
old_time = 0
measured_temp = 0
p_term = 0
i_term = 0
d_term = 0

def read_temp_raw():      ❸
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():  ❹
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        return temp_c

def constrain(value, min, max):   ❺
    if value < min :
        return min
    if value > max :
        return max
    else:
        return value

def update_pid():    ❻
    global old_time, old_error, measured_temp, set_temp
    global p_term, i_term, d_term
    now = time.time()
    dt = now - old_time    ❼
    error = set_temp - measured_temp    ❽
    de = error - old_error              ❾

    p_term = kp * error                 ❿
    i_term += ki * error                ⓫
    i_term = constrain(i_term, 0, 100)  ⓬
    d_term = (de / dt) * kd             ⓭

    old_error = error
    # print((measured_temp, p_term, i_term, d_term))
    output = p_term + i_term + d_term   ⓮
    output = constrain(output, 0, 100)
    return output
```

```
set_temp = input('Enter set temperature in C ')   ⓯
kp = input('kp: ')
ki = input('ki: ')
kd = input('kd: ')

old_time = time.time()            ⓰
try:
    while True:
        now = time.time()
        if now > old_time + 1 :   ⓱
        old_time = now
        measured_temp = read_temp()
        duty = update_pid()
        heat_pwm.ChangeDutyCycle(duty)

        print(str(measured_temp) + ', ' + str(set_temp) + ', ' + str(duty))

finally:
    GPIO.cleanup()
```

❶ Ce code désigne le dossier contenant le fichier du DS18B20. La méthode ressemble beaucoup à celle décrite précédemment en utilisant la commande `glob` pour trouver le premier dossier dont le nom commence par 28.

❷ Ces variables globales sont utilisées par l'algorithme PID. La variable `old_error` est utilisée pour calculer l'évolution de l'erreur pour l'aspect D.

❸ La fonction `read_temp_raw` lit le DS18B20 sous forme de deux lignes de texte.

❹ La fonction `read_temp` est chargée d'extraire la température à la fin de la deuxième ligne après avoir vérifié que la réponse YES a été obtenue à la fin de la première ligne.

❺ Cette fonction utilitaire limite la valeur du premier paramètre de façon à ce qu'elle soit toujours comprise dans la plage définie par les deuxième et troisième paramètres.

❻ La fonction `update_pid` correspond aux calculs proprement dits de la régulation PID.

❼ Calcule `dt` (combien de temps a passé depuis le dernier appel de `update_pid`).

❽ Calcule l'erreur.

❾ Calcule l'évolution de l'erreur `de`.

❿ Calcule l'aspect proportionnel.

⓫ Ajoute l'`error * ki` courant à `i_term`.

⓬ Contraint `i_term` à la même plage que la sortie (0 à 100).

⓭ Calcule `d_term`.

⓮ Additionne tous les termes, puis les contraint à la plage de sortie 0 à 100.

⓯ Contrairement à la version Arduino qui permet de régler les variables à ajuster pendant le fonctionnement du régulateur, le programme Python vous invite une fois à saisir la température, `kp`, `ki` et `kd`.

⓰ `old_time` est initialisé sur l'heure actuelle avant le démarrage de la boucle de régulation principale.

⓱ Si 1 seconde s'est écoulée depuis le dernier échantillonnage, mesure la température, puis lit la nouvelle valeur de sortie (rapport cyclique) et change le rapport cyclique sur le canal PWM.

Expérimentation Raspberry Pi

L'une des différences entre les versions Python et Raspberry Pi est que le second a une sortie qui va de 0 à 100, tandis que celle de l'Arduino a une plage comprise entre 0 et 255. Par conséquent, les paramètres kp et ki qui ont été obtenus lors de la syntonisation de la configuration Arduino doivent être ajustés pour le Raspberry Pi. Il suffit de diviser kp et ki par 2,5 pour ramener la sortie dans la plage de 0 à 100. On obtient alors une valeur de 280 pour kp et de 22 pour ki.

Exécutez le programme, réglez la température à 30 et saisissez ces nombres. Vous devriez obtenir des résultats similaires à la version Arduino :

```
$ sudo python ex_11_pid_thermostat.py
Enter set temperature in C 30
kp: 280
ki: 22
kd: 0
23.437, 30, 100
23.437, 30, 100
23.5, 30, 100
23.562, 30, 100
23.687, 30, 100
```

Après avoir importé ces résultats dans une feuille de calcul, j'ai obtenu le graphique illustré à la figure 12-16. Il s'agit ici aussi d'une vue grossie de la température dont la régulation est plutôt précise.

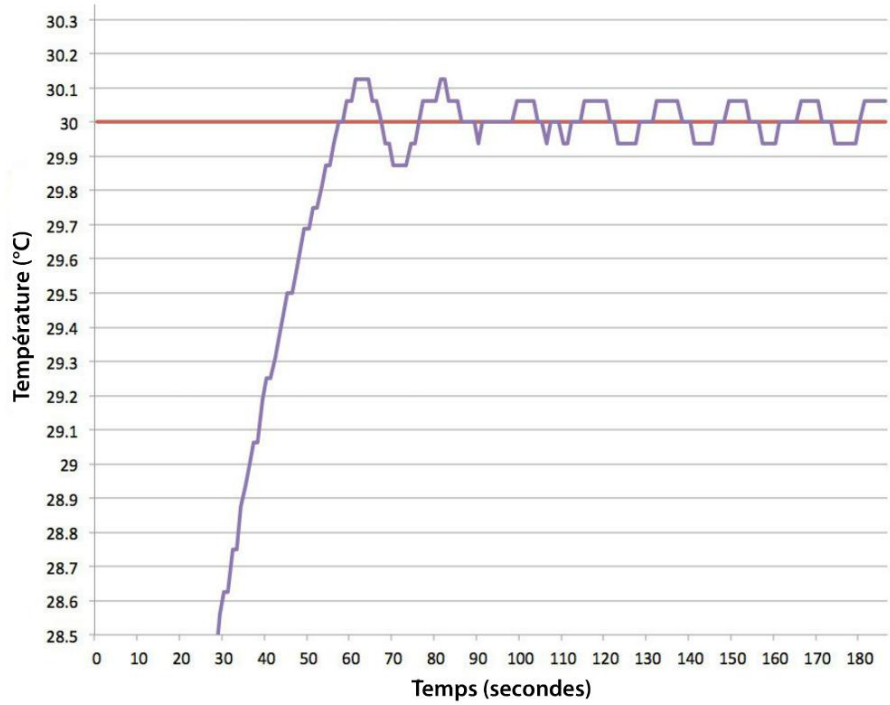


Figure 12-16. *Résultats de la régulation PID avec le Raspberry Pi*

Projet : système thermostatique de réfrigération de boisson

Dans ce projet, nous doterons le réfrigérateur de boisson réalisé dans le projet « Réfrigération de boisson » au chapitre 11, d'un système de régulation thermostatique. Ainsi, votre boisson sera refroidie à la bonne température (figure 12-17). Au chapitre 14, ce projet sera complété par un écran qui affichera la température de consigne et la température actuelle.

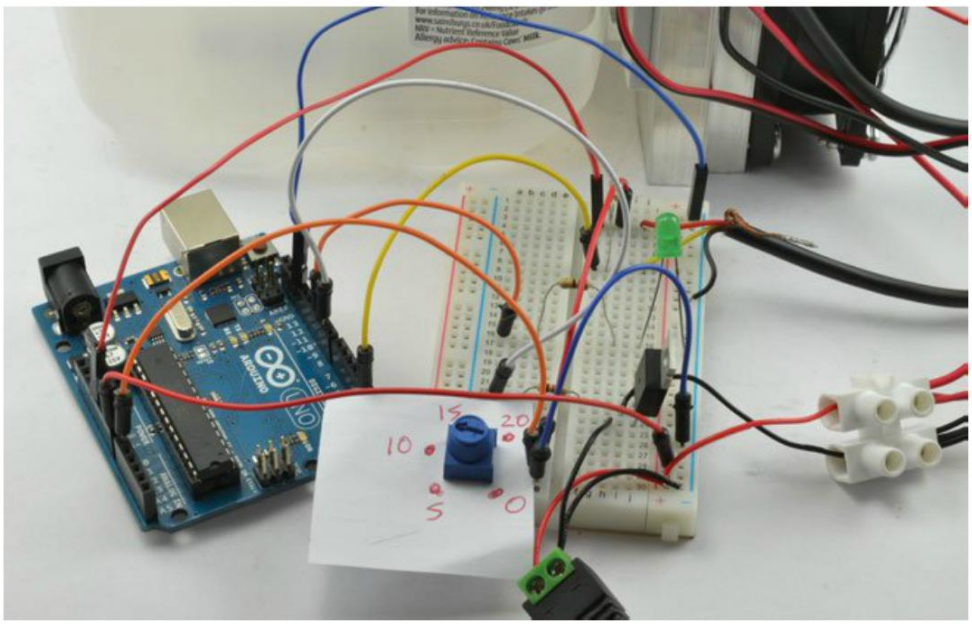


Figure 12-17. *Un réfrigérateur de boisson thermostatique*

Ce projet est réalisé avec un Arduino. Mais, puisque vous avez appris à utiliser le Raspberry Pi avec un module DS18B20, vous devriez réussir à adapter le projet afin qu'il fonctionne avec un Raspberry Pi.

Matériel

Ce projet reprend le projet « Réfrigération de boisson », réalisé à la fin du chapitre 11, en le complétant par un Arduino et un capteur de température DS18B20.

Composants nécessaires

Les composants suivants sont nécessaires pour la réalisation du projet.

Nom	Composant	Source
R1	Résistance 4,7 kΩ	Mouser : 291-4.7k-RC
R2	Résistance 1 kΩ	Mouser : 291-1k-RC
R3	Résistance 270 Ω	Mouser : 291-270-RC
R4	Potentiomètre linéaire 10 kΩ	Adafruit : 356 Sparkfun : COM-09806
	Sonde de température DS18B20	eBay, Adafruit : 381
Q1	MOSFET FQP30N06L	Mouser : 512-FQP30N06L

LED1	LED verte	Adafruit : 298 Sparkfun : COM-09650
	Système de refroidissement Peltier à double ventilateur 4 A ou moins	eBay
	Jack femelle pour l'adaptateur de bornes à vis	Adafruit : 368
	Alimentation électrique (12 V à 5 A)	Adafruit : 352
	Domino électrique	Magasin de bricolage
	Grande bouteille de lait ou de jus de fruit	Récupération

La sonde DS18B20 contient le même circuit que celui utilisé dans les expériences « Précision de la régulation thermostatique marche/arrêt » et « Régulation thermostatique PID », également réalisées dans ce chapitre, à la différence que la puce se présente sous la forme d'une sonde munie d'un long fil que vous pouvez connecter sur la plaque d'essai.

Si vous voulez utiliser un élément Peltier de plus de 4 A, pensez aussi à utiliser une alimentation électrique plus puissante pour disposer d'un courant nominal supérieur à celui requis par le système de refroidissement. Prévoyez au moins un demi-ampère supplémentaire pour les ventilateurs et un demi-ampère de plus par sécurité.

Montage

La figure 12-18 présente le schéma du circuit du projet. Sur la gauche de la figure 12-18, vous pouvez voir R4 qui est une résistance variable, ou potentiomètre linéaire (voir l'encadré ci-dessous « Potentiomètres »). Le curseur du potentiomètre est connecté à l'entrée analogique A0 de l'Arduino (voir la section « Entrées Analogiques » du chapitre 2 page 18). La position du bouton du potentiomètre règle la tension sur A0 qui est mesurée par Arduino, puis l'utilise pour régler la température souhaitée.

Potentiomètres

Les potentiomètres ne vous sont pas inconnus : vous vous en servez par exemple comme boutons de réglage du volume de la radio. Ils sont munis d'un bouton qui fait presque un tour complet.

Sur la figure 12-18, vous pouvez voir le potentiomètre utilisé comme entrée sur un Arduino. Le haut du potentiomètre est connecté à une broche 5 V et le bas à la masse. La connexion du milieu varie entre 0 et 5 V en fonction de la position du bouton.

La partie droite de la figure 12-18 ressemble beaucoup au schéma de l'expérience « Précision de la régulation thermostatique marche/arrêt », au début de ce chapitre, à la différence qu'au lieu d'un transistor MPSA14 courant faible, on utilise un MOSFET FQP30N06L pour puissances élevées. Ce transistor peut commuter un courant de 4 A ou davantage vers le système de refroidissement sans que sa température n'augmente au point de nécessiter un dissipateur thermique.

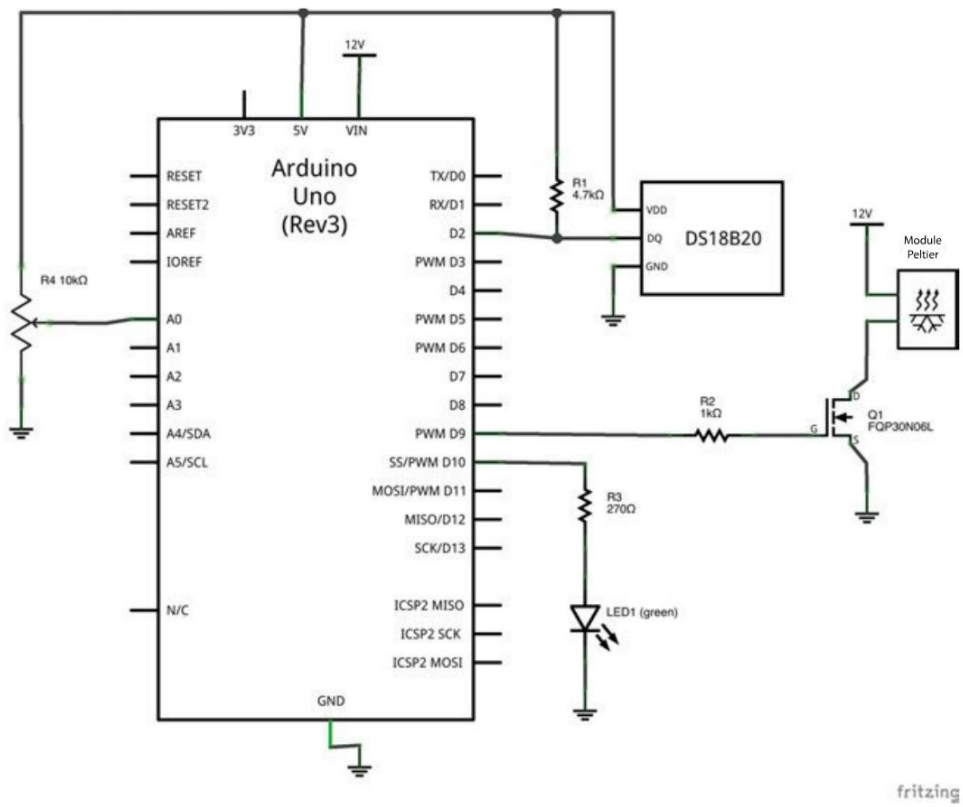


Figure 12-18. *Schéma du circuit de réfrigération de boisson thermostatique*

Construction

Si l'on suppose que vous avez construit le projet « Réfrigération de boisson » au chapitre 11, voici les étapes nécessaires pour compléter le projet.

Étape 1 : ajout de la sonde thermique

Le système de refroidissement est identique au projet « Réfrigération de boisson » au chapitre 11, à la différence que vous ajoutez une sonde de température à la base du récipient. Le verre ou la bouteille à refroidir sera posé sur la sonde (figure 12-19). Ici, j'ai simplement fixé la sonde avec du ruban adhésif.

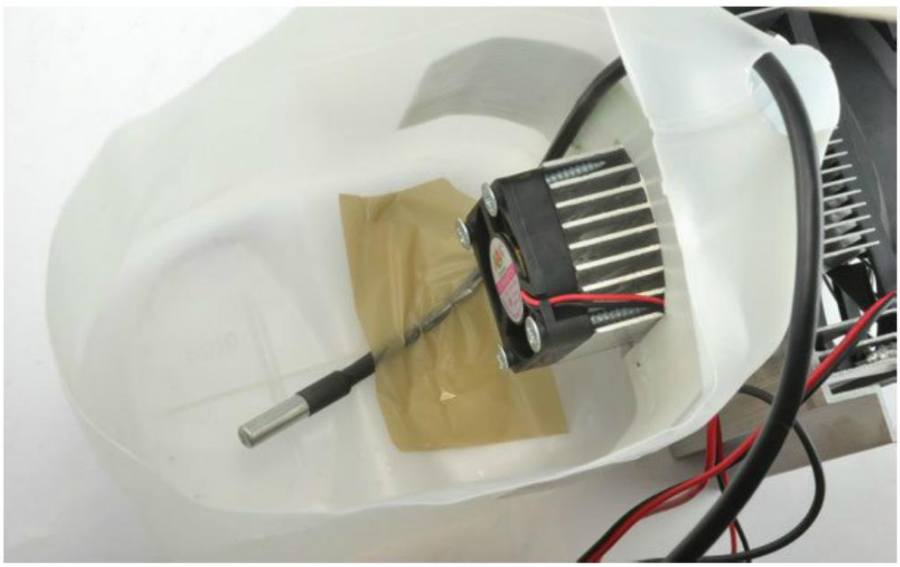


Figure 12-19. *Ajout de la sonde de température*

Étape 2 : réalisation du circuit sur la plaque d'essai

La figure 12-20 montre la réalisation du circuit du projet, ainsi que le raccordement des différents composants.

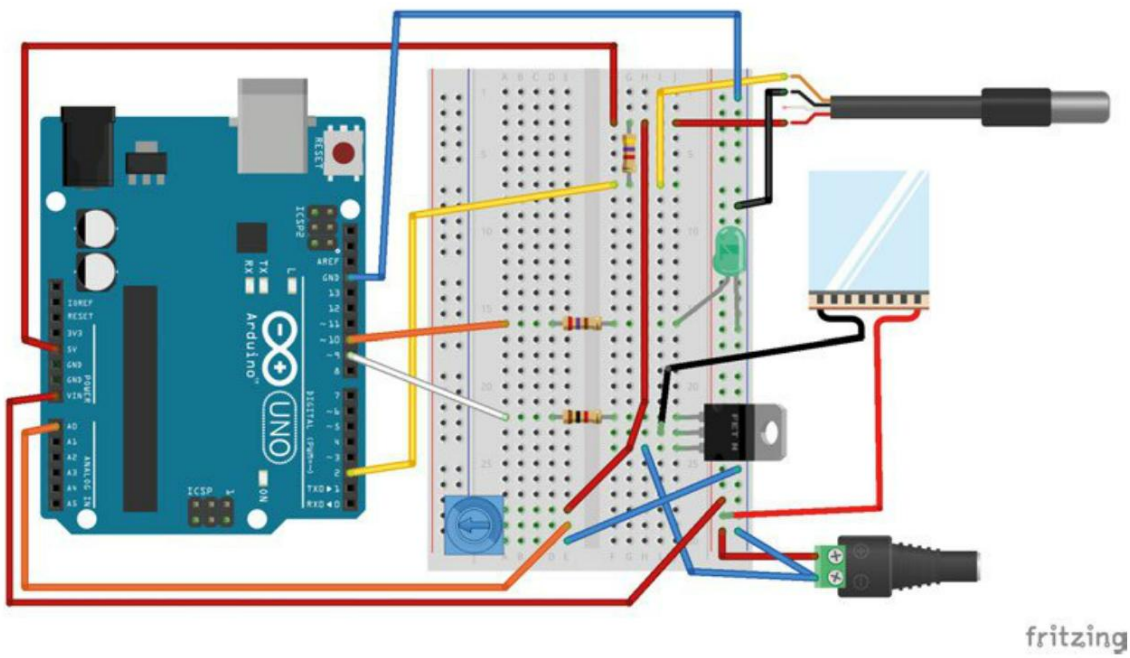


Figure 12-20. *Réalisation du circuit du projet*

Fixez les composants sur la plaque d'essai en veillant à positionner correctement le MOSFET et la LED. Le câble de la sonde de température compte quatre fils. Les fils rouge et noir correspondent aux connexions VCC et GND et le fil jaune est la sortie numérique de la sonde. Le quatrième fil ne doit pas être connecté.

J'ai fait passer les fils du potentiomètre à travers un morceau de papier sur lequel j'ai dessiné des graduations correspondant aux réglages de la température.

Étape 3 : raccordement du système de réfrigération

Le système de réfrigération a trois paires de conducteurs : deux paires pour les ventilateurs et un pour le module Peltier proprement dit. Un domino est utilisé pour faciliter le raccordement du système de réfrigération (figure 12-21) afin de connecter le système à la plaque d'essai à l'aide de deux fils seulement.

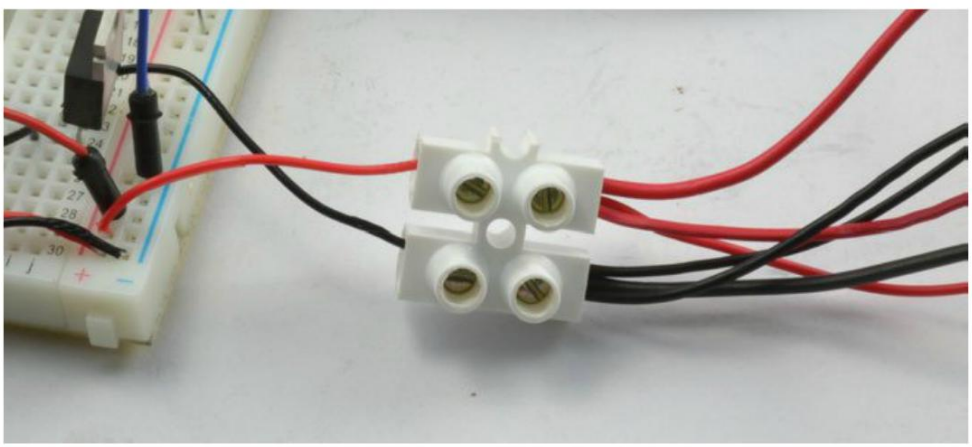


Figure 12-21. *Raccordement du système de réfrigération*

Étape 4 : raccordement de la prise électrique

L'adaptateur jack femelle/bornier à vis peut être connecté à la plaque d'essai à l'aide de câbles flexibles mâles/mâles, à condition que les câbles soient de bonne qualité et que le fil soit relativement épais. Les conducteurs des câbles flexibles sont souvent assez fins et risquent de chauffer. Cela ne pose pas de problème, sauf si les fils chauffent vraiment beaucoup. Toutefois, cela signifie que la totalité du courant de 12 V ne parviendra pas au module Peltier et le système mettra donc plus de longtemps à refroidir.

À la place, vous pouvez utiliser un seul conducteur isolé pour connecter la plaque d'essai à l'adaptateur jack femelle et au système de refroidissement.

Programme Arduino

Certes, l'utilisation de la régulation PID pour un système de réfrigération de boisson pourra sembler un peu excessive. Mais ce n'est qu'une question de programme. Le fait que vos boissons soient maintenues au frais par un super algorithme ne vous coûtera pas plus cher.

Le sketch présente de nombreux points communs avec ceux des expériences « Précision de la régulation thermostatique marche/arrêt » et « Régulation thermostatique PID », également réalisées dans ce chapitre, y compris le code utilisé pour communiquer avec le capteur de température DS18B20. Reportez-vous à ces projets pour plus d'informations sur cette partie du programme.

```
#include <OnveWire.h>
#include <DallasTemperature.h>
#include <PID_v1.h>

const double minTemp = 0.0;      ❶
const double maxTemp = 20.0;
const float tempOKMargin = 0.5;

double kp = 1500;                ❷
double ki = 50.0;
double kd = 0.0;

const int tempPin = 2;
const int coolPin = 9;
const int ledPin = 10;           ❸
const int potPin = A0;
const long period = 1000; // >750

OneWire oneWire(tempPin);
DallasTemperature sensors(&oneWire);

double setTemp = 0.0;
double measuredTemp = 0.0;
double outputPower = 0.0;
long lastSampleTime = 0;

PID myPID(&measuredTemp, &outputPower,
      &setTemp, kp, ki, kd, REVERSE);      ❹

void setup() {
  pinMode(coolPin, OUTPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  sensors.begin();
  myPID.SetSampleTime(1000);
  myPID.SetMode(AUTOMATIC);
}

void loop() {                    ❺
  long now = millis();
```

```
  if (now > lastSampleTime + period) {
    checkTemperature();
    lastSampleTime = now;
  }
  setTemp = readSetTempFromPot();   ❻
}

void checkTemperature() {   ❼
  measuredTemp = readTemp();
  Serial.print(measuredTemp);
  Serial.print(", ");
  Serial.print(setTemp);
  Serial.print(", ");
  Serial.println(outputPower);
  myPID.Compute();
  analogWrite(coolPin, outputPower);
  float error = setTemp - measuredTemp;   ❽
  if (abs(error) < tempOKMargin) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
}

double readSetTempFromPot() {   ❾
  int raw = analogRead(potPin);
  double temp = map(raw, 0, 1023, minTemp, maxTemp);
  return temp;
}

double readTemp() {
  sensors.requestTemperatures();
  return sensors.getTempCByIndex(0);
}
```

❶ Les deux constantes `minTemp` et `maxTemp` définissent la plage de températures pouvant être réglées à l'aide du potentiomètre. La variable `tempOKMargin` détermine le niveau d'écart toléré entre la température courante et la température de consigne avant que la LED verte ne s'éteigne.

❷ `kp` est défini sur une valeur élevée de façon à ce que le chauffage soit allumé et éteint proprement. Cela permet surtout d'éviter le bruit produit par les moteurs du ventilateur quand l'alimentation est insuffisante. Pour l'éviter, il serait aussi possible de connecter les ventilateurs séparément afin qu'ils fonctionnent en continu et de réguler uniquement la puissance transmise au module Peltier.

❸ Des broches supplémentaires sont définies pour la LED et le potentiomètre.

❹ La régulation PID est initialisée en mode `REVERSE` au lieu de `DIRECT` (comme précédemment), car l'ajout de davantage de puissance de sortie réduira la température au lieu de l'augmenter.

❺ La boucle principale vérifie qu'une seconde s'est écoulée, puis appelle `checkTemperature` afin d'allumer ou éteindre le système de refroidissement.

❻ Chaque fois que la boucle est lue (ce qui a lieu plusieurs fois par seconde), la fonction `read SetTempFromPot` est appelée pour régler la variable `setTemp` d'après la position du potentiomètre.

❼ `checkTemperature` mesure la température, la lit, puis corrige la régulation PID. Cette fonction communique aussi les valeurs au moniteur série pour vous permettre d'ajuster le système de refroidissement ou de surveiller ses performances. La carte Arduino ne doit pas nécessairement être connectée par une liaison USB, car elle est alimentée par l'intermédiaire de sa broche Vin. Mais si vous utilisez la prise USB, vous pouvez piloter cette sortie à l'aide du moniteur série.

❽ La suite de cette fonction allume la LED si la température mesurée est comprise dans la plage définie par `tempOKMargin` pour la température de consigne. La fonction `abs` (absolue) élimine les signes moins qui précèdent les nombres.

❾ Code permettant de convertir la position du potentiomètre en valeur comprise entre `minTemp` et `maxTemp` ; la valeur analogique brute (entre 0 et 1 023) est lue à partir de la variable `raw`. La fonction `map` (voir l'encadré ci-dessous) assure ensuite la conversion dans la plage de températures souhaitées.

La fonction map Arduino

Lorsqu'un Arduino ou un Raspberry Pi est utilisé pour piloter un système, vous devrez souvent convertir un nombre appartenant à une plage de valeurs en un nombre correspondant dans une autre plage de valeurs.

Par exemple, une entrée analogique Arduino a une plage comprise entre 0 et 1 023. Si nous voulions convertir cette plage en températures comprises entre 0 et 20, nous pourrions simplement diviser le nombre par 51,15 (1 023/20). Ainsi, 1 023 correspondrait à 1 023/51,15 = 20.

Ce n'est pas aussi simple lorsque les deux plages ne commencent pas à 0. C'est là que la fonction map Arduino intervient. Comme vous pouvez le voir ici, cinq paramètres sont nécessaires pour convertir un nombre compris dans une plage allant de 0 à 1023 en un nombre compris dans une plage allant de 20 à 40 :

```
map(value, 0, 1023, 20, 40);
```

Le premier paramètre est la valeur à convertir, le deuxième et le troisième sont la plage de nombres existants et le quatrième est la plage voulue pour le nombre (ici, entre 20 et 40).

Il n'y a pas de fonction de ce type dans Python, mais il est facile d'en écrire une, puis de s'en servir dans vos programmes. En voici un exemple :

```
def map(value, from_low, from_high,
        to_low, to_high):
    from_range = from_high - from_low
    to_range = to_high - to_low
    scale_factor = from_range / to_range
    return to_low + (value / scale_factor)
```

Vous pouvez ensuite appeler cette fonction Python avec les mêmes paramètres que dans l'équivalent Arduino. Par exemple :

```
map(510, 0, 1023, 20, 40)
```

Vous obtenez ainsi la valeur 30 qui est à mi-chemin entre 20 et 40, tout comme 510 est à mi-chemin entre 0 et 1023.

Résumé

Même si dans ce chapitre, nous nous sommes servis de la température pour expliquer comment réguler précisément un système préalablement étalonné, les mêmes principes s'appliquent aussi au pilotage d'autres aspects, comme la position. C'est ainsi que fonctionne le servomoteur que nous avons étudié au chapitre 9.

Dans le chapitre suivant, nous verrons comment piloter des dispositifs en courant alternatif à haute tension avec le Raspberry Pi ou une carte Arduino.

Contrôle d'appareils à courant alternatif

13

Le pilotage de dispositifs utilisant du courant alternatif présuppose de bien connaître les dangers et les précautions particulières devant être prises en cas de manipulation de tensions élevées. Dans ce chapitre, vous apprendrez à contrôler des systèmes utilisant du courant alternatif en toute sécurité à l'aide de relais électromécaniques et de contacteurs statiques, ainsi que de techniques telles que la commutation avec détection de passage par zéro.

Danger de mort !

Aux États-Unis, des centaines de personnes meurent chaque année des suites d'une électrocution. Encore plus de personnes subissent de graves brûlures et de nombreux incendies domestiques sont provoqués par des circuits défectueux. L'alimentation électrique de la maison utilise une tension élevée qui est capable de fournir des courants forts.

Lorsque vous en serez arrivé à l'aspect pratique de ce chapitre, débranchez toujours le montage et posez la prise près de vous. Ainsi, vous pouvez vérifier d'un seul coup d'œil que vous pouvez travailler en toute sécurité.

Utilisez toujours une prise dotée d'une mise à la terre lorsque vous travaillez sur un projet et ne laissez jamais de fils dénudés ou de circuits exposés sur des projets servant à contrôler des courants alternatifs sans les protéger dans un boîtier isolé fermé.

N'utilisez pas de plaques d'essai pour vos montages en courant alternatif, car elles ne sont pas conçues pour résister à des tensions et des courants aussi élevés.

En résumé, à moins de disposer des compétences requises pour travailler avec du courant alternatif, utilisez un module prêt à l'emploi comme le module PowerSwitch Tail.

Théorie de la commutation du courant alternatif

Dans cette partie, nous étudierons des aspects théoriques de la commutation de courant alternatif et nous passerons en revue divers circuits et composants utilisés dans ce domaine, pour terminer par la pratique.

Qu'est-ce que le courant alternatif ?

Alors que le courant continu CC circule dans une seule direction, le courant alternatif CA est beaucoup moins prévisible. Le sens du flux de courant s'inverse 120 fois par seconde dans certaines parties du monde (dont les États-Unis) et 100 fois par seconde dans d'autres. Cette inversion du flux du courant est obtenue par une inversion de la tension de la charge (figure 13-1). Il y a deux inversions du flux du courant dans chaque cycle complet. La fréquence du courant alternatif est donc de 60 Hz ou 50 Hz (cycles par seconde). Le hertz est l'unité de mesure de la fréquence ; 1 Hz correspond à un cycle par seconde.

Aux États-Unis et dans d'autres pays, l'alimentation en courant alternatif est généralement en 120 V. Dans la majeure partie du monde, les prises en CA sont alimentées en 220 V, une tension potentiellement beaucoup plus dangereuse.

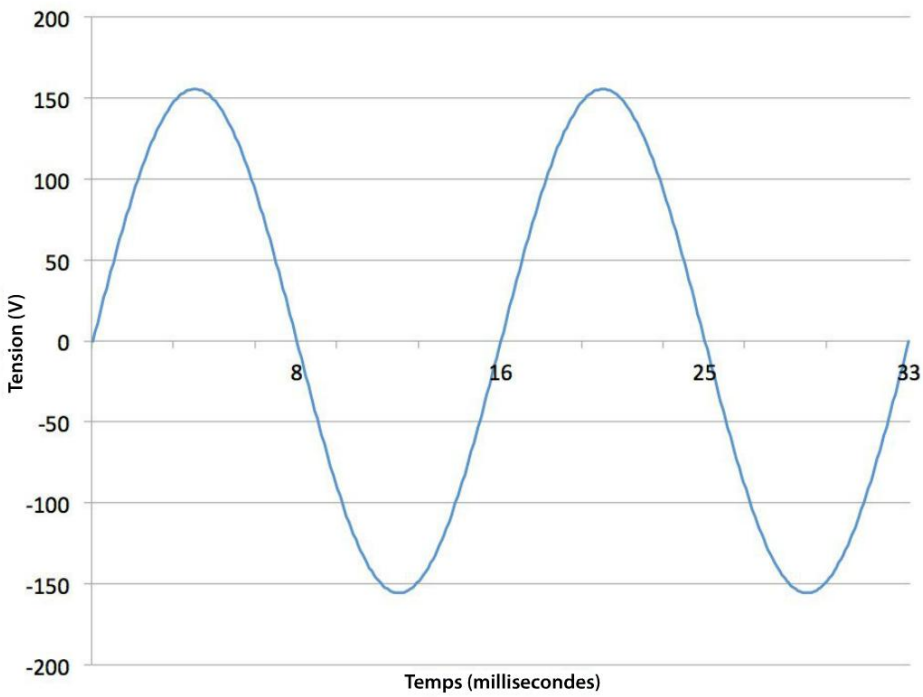


Figure 13-1. *Courant (et tension) alternatif*

Comme vous pouvez le voir à la figure 13-1, les pics de tensions positifs et négatifs dépassent largement 120 V. Ce chiffre correspond en fait à une sorte de tension moyenne ou tension efficace. C'est la tension CC équivalente qui serait capable de fournir la même quantité de puissance à une charge. Ainsi, un courant continu de 120 V pourrait faire briller une ancienne ampoule à filament avec la même intensité qu'un courant alternatif de 120 V.

Relais

Nous avons déjà abordé les relais dans la section « Pilotage d'un moteur CC à l'aide d'un relais » du chapitre 7 page 102, où vous aviez utilisé un relais pour la commutation d'un moteur CC. La bobine d'un relais est isolée de la partie commutation, ce qui est indispensable du point de vue de la sécurité. Les capacités des relais sont généralement précisées sur leur boîtier. Par exemple, un « morceau de sucre » ordinaire peut commuter un courant d'une intensité de 10 A en 250 V CA et de 10 A en 24 V CC.

La figure 13-2 illustre l'utilisation d'un relais pour la commutation d'une charge en CA Souvenez-vous que la bobine du relais nécessite un peu trop de courant pour être pilotée directement depuis la sortie numérique de la carte Arduino ou du Raspberry Pi.

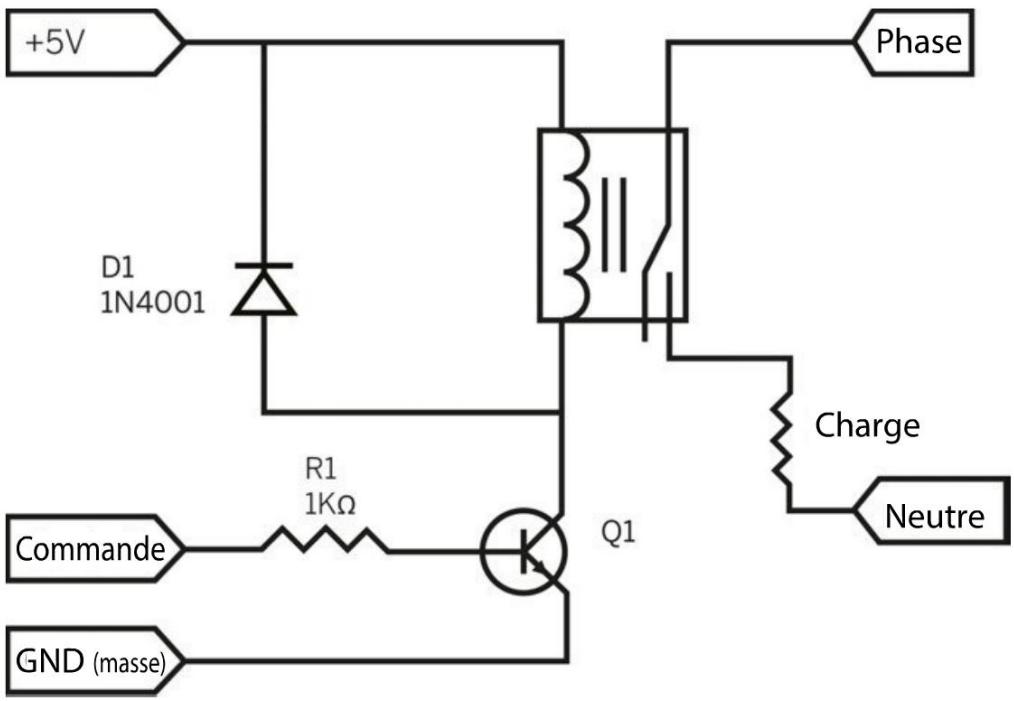


Figure 13-2. *Commutation en CA avec un relais*

Les liaisons de commande, 5 V et GND sont connectées à l'Arduino ou au Raspberry Pi. Quand la liaison de commande est à l'état haut, le transistor Q1 est conducteur et excite la bobine du relais qui ferme le contact de ce dernier afin d'établir la liaison avec la phase de la ligne électrique en CA à une extrémité de la charge (dispositif que vous voulez allumer et éteindre). L'autre extrémité de la charge est connectée au neutre de la ligne en CA.

Optocoupleurs

Les relais utilisent une technologie ancienne ; les contacteurs statiques, que nous étudierons un peu plus loin, les remplacent dans de nombreuses applications de commutation de courant alternatif. L'optocoupleur est un composant clé des contacteurs statiques. Il a essentiellement pour rôle de séparer la partie de commutation basse tension du projet de la partie en CA à haute tension qui est beaucoup plus dangereuse.

La figure 13-3 présente un optocoupleur basé sur un transistor.

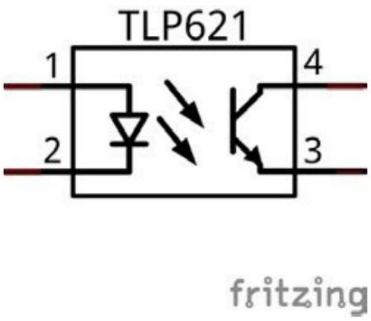


Figure 13-3. *Un optocoupleur*

Un optocoupleur se compose d'une LED et d'un élément photosensible (généralement un phototransistor) qui sont réunis dans un boîtier en plastique.

L'aspect essentiel est qu'il n'y a pas de connexion électrique entre la LED et le phototransistor, mais uniquement une liaison optique. Quand la LED émet de la lumière, le phototransistor conduit l'électricité. Le phototransistor est un composant à faible puissance qui doit être associé à d'autres composants pour pouvoir piloter un système en courant alternatif.

Les dispositifs sont assez sensibles et vous pouvez piloter le côté LED par une broche GPIO d'une carte Arduino voire d'un Raspberry Pi à l'aide d'une résistance de 1 kΩ (limitant le courant à quelques milliampères).

Triacs et optocoupleurs à sortie triac

Les optocoupleurs qui sont conçus pour commuter du courant alternatif présentent quelques spécificités. À l'extrémité photosensible, ils n'ont pas un phototransistor bipolaire ordinaire, mais utilisent un photo-TRIAC (*TRIode for AC*). La figure 13-4 illustre la conception interne d'un composant de ce type (comme le MOC3031). Vous pouvez télécharger les spécifications techniques de ce dispositif au format PDF sur *http://www.farnell.com/datasheets/1639837.pdf*.

Un TRIAC est un transistor conçu pour commuter du courant circulant dans les deux sens, ce qui est le cas du courant alternatif.

Une caractéristique d'un TRIAC est qu'une fois enclenché, il laisse passer le courant jusqu'à ce que le courant qui le traverse devienne trop faible. Il ne convient donc pas à une utilisation avec du courant continu. Comme le courant alternatif change de polarité 120 fois par seconde, le TRIAC a autant de possibilités de se désactiver.

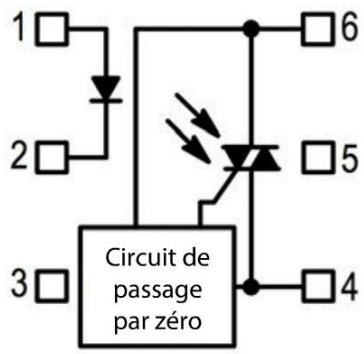


Figure 13-4. *Un optocoupleur avec sortie triac*

Comme le TRIAC se désactive uniquement lorsque le courant (et donc la tension) qui le traverse est faible, les intensités commutées sont faibles. Parmi les avantages de ce mécanisme d'enclenchement, il faut également citer le fait que cela réduit les interférences électriques. Cette commutation qui se fait en douceur peut encore être améliorée par l'utilisation d'un circuit de passage par zéro inclus dans certains optocoupleurs. Cela diffère l'enclenchement du TRIAC jusqu'à ce que la tension passe par zéro ; l'enclenchement et la désactivation se font donc sans heurts.

La figure 13-5 présente un circuit utilisant un TRIAC avec passage par zéro pour réguler un TRIAC plus puissant.

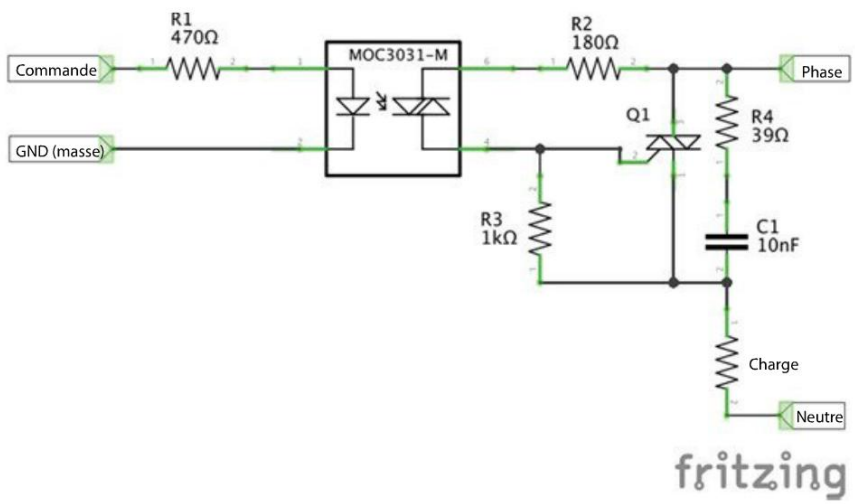


Figure 13-5. *Commutation de CA avec un optocoupleur avec passage par zéro*

Le TRIAC qui est intégré à un optocoupleur comme le MOC3031 est destiné à être utilisé avec un courant faible pour commander un TRIAC plus puissant qui commute le courant alternatif.

Commander une charge à partir d'un Arduino ou d'un Raspberry Pi se résume alors à fournir un ou deux milliampères à la LED qui se trouve à l'intérieur de l'optocoupleur.

R2 et R3 limitent le courant qui circule par le photo-TRIAC basse puissance qui se trouve à l'intérieur de l'optocoupleur. R4 et C1 sont là pour absorber les tensions transitoires qui se produiraient malgré la commutation en douceur.

Commutation de courant alternatif en pratique

Attention, danger ! N'essayez pas de monter les circuits décrits dans ce chapitre sur une plaque d'essai. Si vous voulez commuter des charges en CA avec un Arduino ou un Raspberry Pi, respectez les consignes fournies dans cette partie.

Modules relais

Dans la section « Modules relais » du chapitre 7 page 105, nous avons examiné des modules relais prêts à l'emploi. En courant alternatif, ils présentent l'avantage de permettre de connecter le dispositif à commuter à l'aide des bornes à vis. Le relais isole naturellement le côté basse tension du projet de la partie haute tension.

Comme les relais contiennent des pièces métalliques, en cas de dysfonctionnement, la partie qui est reliée au secteur risque de se retrouver connectée aux bobines du relais. Cela pourrait notamment se produire en cas de choc violent ou d'écrasement accidentel du relais. Pour une sécurité accrue, les relais utilisent souvent un optocoupleur.

Utilisation d'un module relais en toute sécurité

Souvenez-vous qu'un module relais comme celui illustré à la figure 13-6 comporte des conducteurs métalliques dénudés et des soudures sur la face supérieure et inférieure du circuit qui sera raccordé au secteur. Pour éviter de les toucher par inadvertance, placez toujours le module, ainsi que de toutes les parties du projet, dans un boîtier en plastique. Les composants placés à l'intérieur de la boîte doivent être fixés solidement afin d'éviter qu'ils ne se déplacent.

Les traversées avec passe-câble sécurisé empêchent les câbles d'être arrachés et de se détacher des bornes à vis du module, ce qui créerait un risque de court-circuit.

Ne modifiez pas les branchements du module relais et ne connectez pas de câbles sur les bornes à vis quand l'installation est branchée sur le courant alternatif. Fermez la boîte dès que le raccordement a été effectué.

Figure 13-6. *Un module relais dans une boîte en plastique*

Méfiez-vous aussi des modules relais bon marché même si, d'après leurs spécifications techniques, ils peuvent être utilisés avec du courant alternatif, car ils ne sont pas sûrs.

S'il est écrit sur le relais qu'il est compatible avec un courant de 10 A à 25 V, n'allez pas imaginer que c'est aussi le cas du module relais. Les petites bornes à vis de nombreux modules bon marché ne tolèrent qu'un courant de 2 A. Par ailleurs, sur certains circuits bas de gamme, les soudures des contacts du relais sont très proches du côté haute tension du circuit. C'est dangereux avec des tensions élevées en CA. N'utilisez ces relais qu'avec une basse tension continue et avec des courants faibles. Les meilleurs modules relais comportent un optocoupleur et une fente est prévue dans le circuit autour du contact de relais COM pour une isolation maximale.

Pour commuter du courant alternatif, la méthode la plus sûre consiste à utiliser un module prêt à l'emploi enfermé dans un boîtier, comme le PowerSwitch Tail (voir plus loin).

Vérifiez si votre module relais est actif-bas ou actif-haut. S'il est actif-bas, le relais s'active quand la sortie numérique est à l'état LOW. Par conséquent, dès que vous définissez la broche comme sortie numérique, vous devez aussi définir la sortie comme étant HIGH sur la ligne suivante afin d'éviter que le relais ne se déclenche brièvement à l'initialisation de l'Arduino :

```
pinMode(relayPin, OUTPUT);
digitalWrite(relayPin, HIGH);
```

Si vous utilisez un module relais actif-bas avec un Raspberry Pi, vous pouvez vous servir du paramètre `initial` pour définir la sortie comme HIGH:

```
GPIO.setup(relay_pin, GPIO.OUT, initial=True)
```

Sur les modules relais qui comportent un optocoupleur, il est généralement possible de retirer un cavalier afin que l'alimentation positive de la bobine du relais soit assurée indépendamment du côté positif de la LED de l'optocoupleur. Cela offre un niveau d'isolation supplémentaire, mais nécessite une alimentation électrique séparée.

Relais statiques

La figure 13-7 présente un module SSR (*Solid State Relays*, ou contacteur statique). Il s'agit d'un boîtier scellé qui contient un circuit probablement très proche de celui illustré à la figure 13-5.

Figure 13-7. *Un relais statique pour courant alternatif*

Ces modules très répandus facilitent considérablement la commutation du courant alternatif. Comme il y a néanmoins des parties métalliques exposées, le module doit être enfermé dans un boîtier isolant.

Le côté basse tension du module peut être connecté directement à un Raspberry Pi ou un Arduino, car ces derniers contiennent une série de résistances adaptées pour la LED.

PowerSwitch Tail

Le PowerSwitch Tail (figure 13-8) est un contacteur statique muni d'une prise d'alimentation en CA d'un côté et d'une sortie en CA de l'autre.

Figure 13-8. *Un PowerSwitch Tail*

Des bornes à vis connectent le côté LED de l'optocoupleur (série de résistances intégrées) et une petite LED rouge s'allume quand le SSR est activé. Vous utiliserez l'un de ces dispositifs bien pratiques dans le projet « Interrupteur à minuterie Raspberry Pi », ci-après.

Projet : interrupteur à minuterie Raspberry Pi

Ce projet utilise un Raspberry Pi et un PowerSwitch Tail pour piloter l'alimentation d'un petit appareil électrique. Au chapitre 16, ce projet très basique sera complété par une interface web afin de vous permettre d'allumer et d'éteindre l'appareil depuis un navigateur (voir le projet « Un interrupteur web Raspberry Pi » du chapitre 16, page 305).

Ce projet est très facile à réaliser ; le seul outil nécessaire est un tournevis.

Composants nécessaires

En plus d'un Raspberry Pi, les composants suivants sont nécessaires pour la réalisation du projet.

Composant	Source
PowerSwitch Tail	Adafruit : 268
Straps flexibles femelles/mâles	Adafruit : 826
Lampe ou autre petit appareil	

Construction

La figure 13-9 présente le raccordement du circuit du projet.

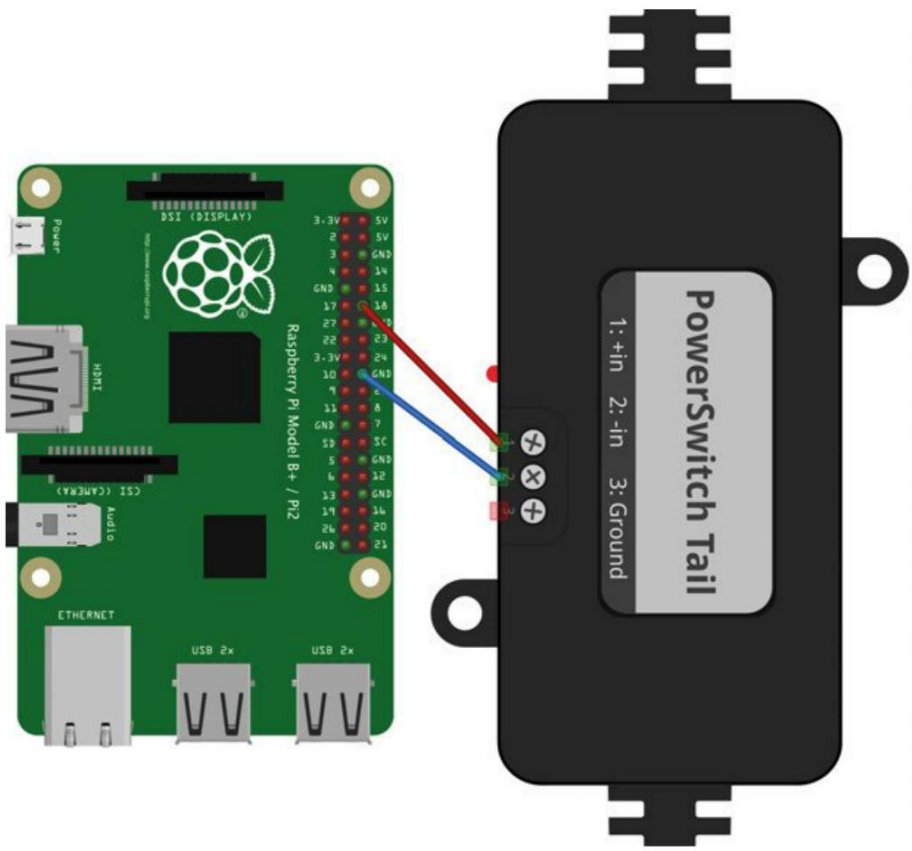


Figure 13-9. *Schéma de raccordement du projet d'interrupteur à minuterie*

Si vous examinez attentivement l'étiquette du PowerSwitch Tail, vous y lirez que l'entrée est comprise entre 3 et 12 V CC pour 3-30 mA. Le courant requis pour l'entrée varie en fonction de la tension, donc l'extrémité la plus basse de la plage correspond à l'entrée 3 V. En fait, à 3,3 V, le PowerSwitch Tail nécessite environ 6 mA, ce qui est compatible avec une broche GPIO du Raspberry Pi.

Il est inutile de raccorder au PowerSwitch Tail un appareil fonctionnant en CA pour réaliser des essais, car sa LED d'état s'allume dès que le SSR est activé.

Programme

Vous trouverez le programme de ce projet dans le fichier `/python/projects/ac_timer_switch.py` (pour plus d'informations sur l'installation des programmes Python du livre, voir la section « Le code du livre » du chapitre 3 page 34) :

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

control_pin = 18

GPIO.setup(control_pin, GPIO.OUT)

try:
    while True:     ❶
        duration_str = input(«Durée allumée en minutes : «) #  ❷
        duration = int(duration_str) * 60     ❸

        GPIO.output(control_pin, True)    ❹
        time.sleep(duration)
        GPIO.output(control_pin, False)    ❺

finally:
    print(“Nettoyage”)
    GPIO.cleanup()
```

Le programme est aussi très simple ; il commence par les importations et les définitions de constantes habituelles.

❶ Avec `while True`, la boucle est exécutée à l'infini, car la condition `True` n'est jamais `False`. C'est la seule raison pour laquelle une boucle while se termine (à moins d'appuyer sur Ctrl-C).

❷ La boucle principale vous invite à saisir une durée en minutes pendant laquelle la lampe doit être allumée.

❸ Convertit la valeur string `duration` en un nombre entier de minutes à l'aide d'`int`, puis ce nombre est multiplé par 60 pour être converti en secondes.

❹ La broche GPIO 18 est définie sur l'état `HIGH` (True) pour activer le PowerSwitch Tail et allumer l'appareil qui est branché sur sa prise CA

❺ Passé le délai fixé, la broche GPIO est définie sur l'état `LOW` pour désactiver le SSR. La boucle redémarre et vous invite à saisir une nouvelle « durée allumée. »

Application du projet

Le PowerSwitch Tail peut commuter jusqu'à 15 A. Vous pouvez donc y brancher divers appareils, à part les plus puissants, comme les bouilloires électriques ou les sèche-cheveux. Une petite lampe de chevet est un bon point de départ.

Exécutez le programme et lorsque s'affiche l'invite vous demandant de saisir une durée, tapez le chiffre 1, soit 1 minute, puis appuyez sur Entrée. L'appareil branché sur le PowerSwitch Tail devrait alors s'allumer ; la petite LED d'état du PowerSwitch Tail s'allume aussi. Au bout d'une minute, le Power-Switch Tail s'éteint.

Résumé

Ce chapitre énumère quelques-uns des dangers auxquels on s'expose lorsque l'on manipule du courant alternatif haute tension. Nous avons également vu qu'il est assez facile d'allumer et d'éteindre des appareils à l'aide des composants adaptés.

Dans le chapitre suivant, vous apprendrez à utiliser des afficheurs avec un Arduino ou un Raspberry Pi.

14 Afficheurs

Vous pouvez non seulement raccorder un moniteur à votre Raspberry Pi, mais aussi divers périphériques de sortie afin d'afficher du texte ou des nombres, des images, ou simplement pour contrôler plusieurs LED simultanément. Ces appareils sont bien trop nombreux pour que nous les passions tous en revue dans ce chapitre. Nous nous en tiendrons aux plus pratiques et ludiques et nous utiliserons diverses techniques d'interfaçage.

Rubans LED

Les LED RGB décrites au chapitre 6 sont simplement composées de trois LED émettrices de lumière réunies dans un boîtier. Il existe un autre type de LED : les LED adressables dont le boîtier LED contient aussi une puce. Ce circuit électronique permet de piloter les trois couleurs de la LED par MLI, comme nous l'avons fait avec un Arduino ou un Raspberry Pi au chapitre 6, à la différence que le contrôleur est intégré à la LED.

Ces LED adressables sont conçues pour être contrôlées en grands nombres à l'aide d'un microcontrôleur ou d'un ordinateur comme un Arduino ou un Raspberry Pi. Adafruit commercialise des LED adressables qui se nomment des NeoPixels (le terme NeoPixels est souvent utilisé pour désigner des LED adressables qui n'ont pas été produites par Adafruit et qui sont vendues sur eBay). Les modèles les plus répandus sont de type WS2812. Elles utilisent une norme de communication série qui permet de connecter de longs rubans de LED pour créer de grands afficheurs. Il existe des LED adressables RGB et monochromes.

Elles sont parfois disposées en matrice, mais vous pouvez aussi les acheter sous la forme d'un ruban que vous pouvez couper à vos mesures (figure 14-1) en fonction du nombre de LED dont vous avez besoin pour votre projet.

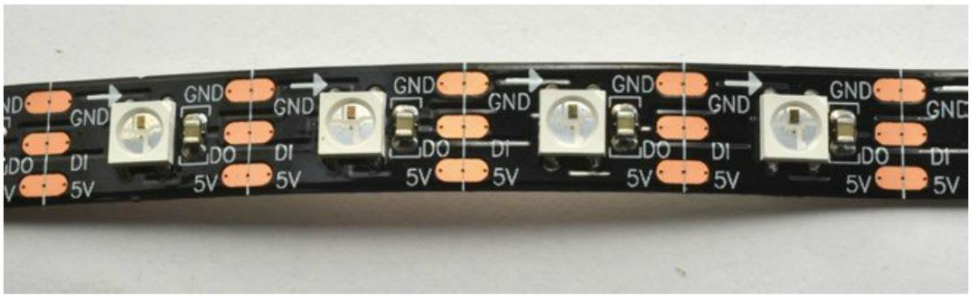


Figure 14-1. *Ruban de LED NeoPixel*

Si vous examinez attentivement la figure 14-1, vous pouvez voir que le ruban se divise en segments. Les trois plots soudés de part et d'autre d'une LED servent à l'alimentation de puissance, la masse et l'alimentation en 5 V. Le port série D0 qui est connecté en chaîne d'une LED à la suivante, c'est-à-dire que la sortie d'une LED est connectée à l'entrée de la suivante. Si vous voulez raccourcir la bande de LED, le ruban peut être coupé le long de la ligne continue qui passe par le milieu des soudures.

La flèche indique le sens de circulation des données série. Par conséquent, la connexion doit être réalisée à gauche pour contrôler les LED de droite.

Expérience : pilotage d'un ruban de LED RGB

Après avoir acheté un mètre ou deux de ruban de LED sur eBay ou Adafruit, vous aurez probablement hâte de l'essayer. Il est facile de connecter un Arduino (figure 14-2), mais le raccordement d'un Raspberry Pi est un peu plus compliqué.

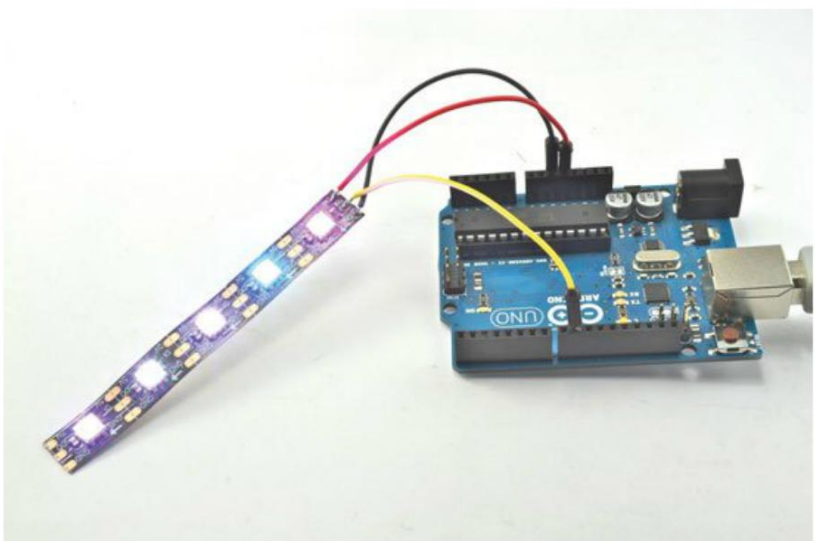


Figure 14-2. *Ruban LED et Arduino*

Composants nécessaires

Voici les composants nécessaires pour la réalisation de cette expérience avec un Arduino.

Composant	Source
Bande LED adressable WS2812	eBay, Adafruit : 1376
3× câbles flexibles mâles/mâles	Adafruit : 758

Si vous utilisez un Raspberry Pi, vous aurez également besoin des composants énumérés dans le tableau ci-dessous, car l'entrée logique des NeoPixels n'est pas spécifiée comme étant compatible avec la logique 3 V. Cela dit, avant d'acheter les composants supplémentaires, vous pouvez toujours essayer de vous passer d'un convertisseur de niveau logique, car il est possible que votre ruban LED fonctionne parfaitement avec la logique 3 V.

Pour réaliser le projet avec un Raspberry Pi, vous aurez également besoin des composants suivants.

Nom	Description	Source
	Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
R1, R2	2× résistances 470 Ω 1/4 W	Mouser : 291-470-RC
Q1	Transistor MOSFET 2N7000	Mouser : 512-2N7000
	Câbles flexibles femelles/mâles	Adafruit : 826

Ce projet n'emploie pas les transistors habituels et utilise plutôt un transistor MOSFET basse puissance pour la conversion de niveau logique, car il réagit bien mieux aux données série haute fréquence qu'un 2N3904, par exemple. Si vous préférez, vous pouvez utiliser un FQP30N06L, mais ce composant haute puissance est un peu surdimensionné pour ce projet.

Montage Arduino

L'une des extrémités du ruban se termine souvent par un câble muni d'une prise à trois broches connectées aux liaisons GND, 5 V et D0 du ruban. Si tel est le cas, vous pouvez utiliser des câbles flexibles femelles-mâles pour connecter le composant à une plaque d'essai ou à votre Arduino. Mais si vous avez déjà coupé quelques pixels pour un autre projet, vous devrez souder des fils à l'afficheur.

J'ai sacrifié trois câbles flexibles femelles-mâles : j'ai coupé les cavaliers à une extrémité, puis j'ai soudé les fils au ruban de cinq NeoPixels, comme illustré à la figure 14-3.

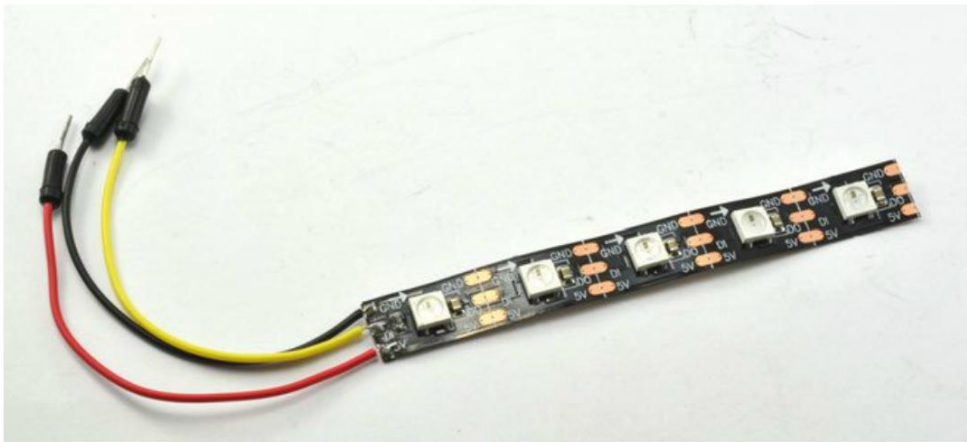


Figure 14-3. *Un afficheur NeoPixel*

Consommation électrique des LED adressables

À leur niveau de luminosité maximale et lorsqu'elles sont blanches, les LED adressables nécessitent beaucoup de courant (environ 60 mA par LED). Sachant que cinq LED adressables peuvent consommer jusqu'à 300 mA (ce qui correspond à la limite de l'énergie pouvant être fournie directement par une carte Arduino ou un Raspberry Pi), si vous en utilisez davantage, vous devrez éventuellement prévoir d'alimenter directement le ruban de LED en 5 V.

Les trois connecteurs peuvent être enfichés directement sur la carte Arduino en connectant la liaison de données D0 à l'entrée/sortie D9 de la carte Arduino (figure 14-2).

Programme Arduino

Il existe une bibliothèque NeoPixel Adafruit qui facilite le pilotage de longues chaînes de LED adressables. Vous pouvez la télécharger à l'adresse *https://github.com/adafruit/Adafruit_NeoPixel* et l'installer dans votre environnement Arduino (voir l'encadré « Installation des bibliothèques Arduino » du chapitre 12 page 227).

Vous trouverez le sketch Arduino de cette expérience dans le dossier `/arduino/experiments/neopixel` (pour plus d'informations sur l'installation des sketchs Arduino du livre, voir la section « Le code du livre » du chapitre 2 page 14) :

```
#include <Adafruit_NeoPixel.h>     ❶

const int pixelPin = 9;      ❷
const int numPixels = 5;     ❸

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(numPixels, pixelPin, NEO_GRB + NEO_KHZ800);  ❹
```

```
void setup() {
  pixels.begin();  ❺
}

void loop() {
  for (int i = 0; i < numPixels; i++) {    ❻
    int red = random(255);
    int green = random(255);
    int blue = random(255);
    pixels.setPixelColor(i, pixels.Color(red, green, blue));  ❼
    pixels.show();
  }
  delay(100L);
}
```

❶ Importe la bibliothèque NeoPixel Adafruit.

❷ Corrigez si vous voulez utiliser une autre broche pour piloter les NeoPixels.

❸ Corrigez en fonction du nombre de LED sur votre ruban. Reportez-vous à l'encadré « Consommation électrique des LED adressables » plus haut, avant d'ajouter des LED.

❹ Initialise la bibliothèque NeoPixel pour votre configuration.

❺ Commence à envoyer les données d'affichage.

❻ Attribue une couleur aléatoire à chacun des pixels, actualise l'afficheur, puis attend 1/10e de seconde avant de changer à nouveau toutes les couleurs des LED.

❼ La fonction `setPixelColor` nécessite deux paramètres : la position d'index du pixel et sa couleur, qui est elle-même composée de trois valeurs comprises entre 0 et 255 pour chacun des trois canaux de couleurs.

Connexions Raspberry Pi

Vous constaterez peut-être que votre Raspberry Pi fonctionne très bien sans conversion de niveau. Faites l'essai avant d'utiliser le convertisseur de niveau sur la plaque d'essai .

Reprenez l'afficheur à cinq NeoPixels que vous venez d'utiliser avec un Arduino et utilisez des câbles flexibles femelles-femelles pour effectuer les connexions suivantes :

- GND NeoPixel à GND Raspberry Pi
- 5 V NeoPixel à 5 V Raspberry Pi
- D0 NeoPixel à GPIO 18

La figure 14-4 présente le raccordement direct du ruban de LED au Raspberry Pi.

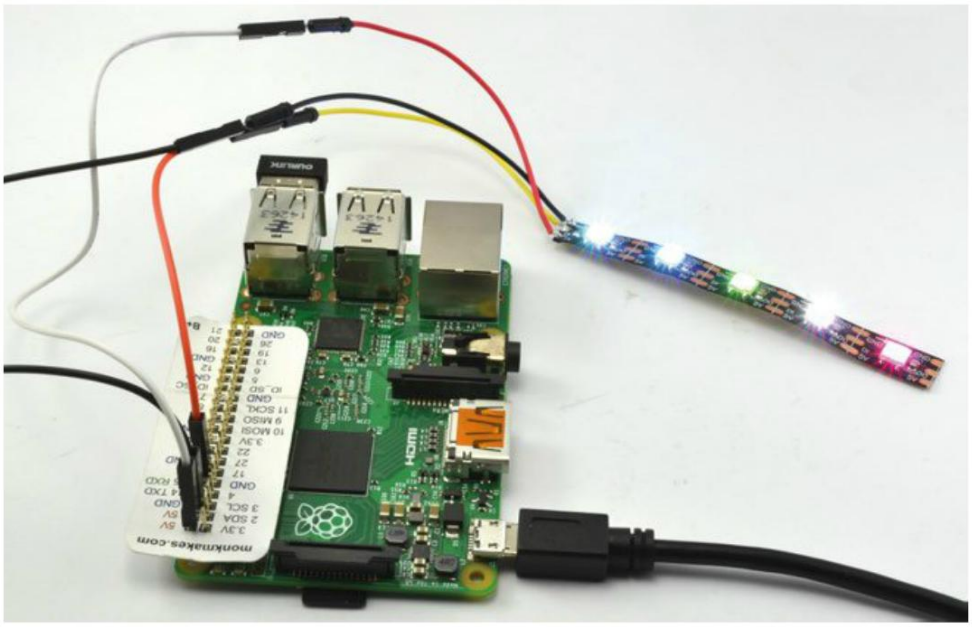


Figure 14-4. *Raccordement direct des NeoPixels au Raspberry Pi*

Passez directement à la section « Programme Raspberry Pi » page suivante, et testez le programme `neopixel_no_level_conv.py`.

Si le ruban de LED ne s'illumine pas dans un arc-en-ciel de couleurs, vous devez probablement procéder selon les règles de l'art et utiliser un convertisseur de niveaux pour amplifier le niveau du signal de 3 à 5 V.

De 3 à 5 V

Vous aurez peut-être de la chance, mais le signal 3 V du Raspberry Pi est inférieur aux 4 V minimum nécessaires pour le niveau HAUT d'une LED adressable WS2812. Le montage de la figure 14-5 illustre l'utilisation d'un transistor MOSFET pour amplifier le niveau du signal à 5 V.

Un effet secondaire de ce décalage de niveau est que la sortie est inversée, c'est-à-dire que si le Raspberry Pi fournit un signal logique LOW (0 V), la sortie vers le ruban de LED sera de 5 V. Au contraire, quand la broche GPIO du Raspberry Pi est à l'état `HIGH` (3,3 V), la sortie vers le ruban de LED sera de 0 V.

Heureusement, c'est facile à corriger dans le programme.

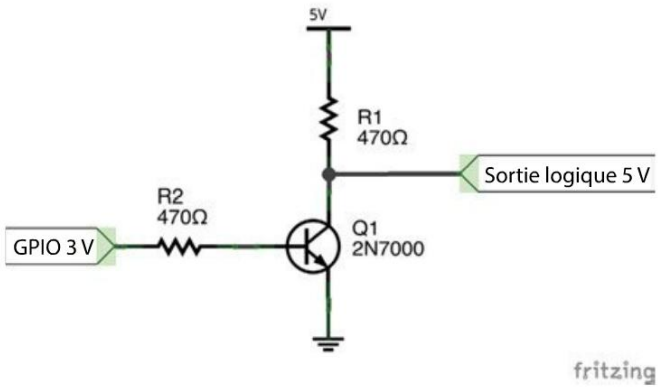


Figure 14-5. *Conversion de niveau de 3 V en 5 V*

La figure 14-6 montre la réalisation du circuit de l'expérience avec la conversion de niveau.

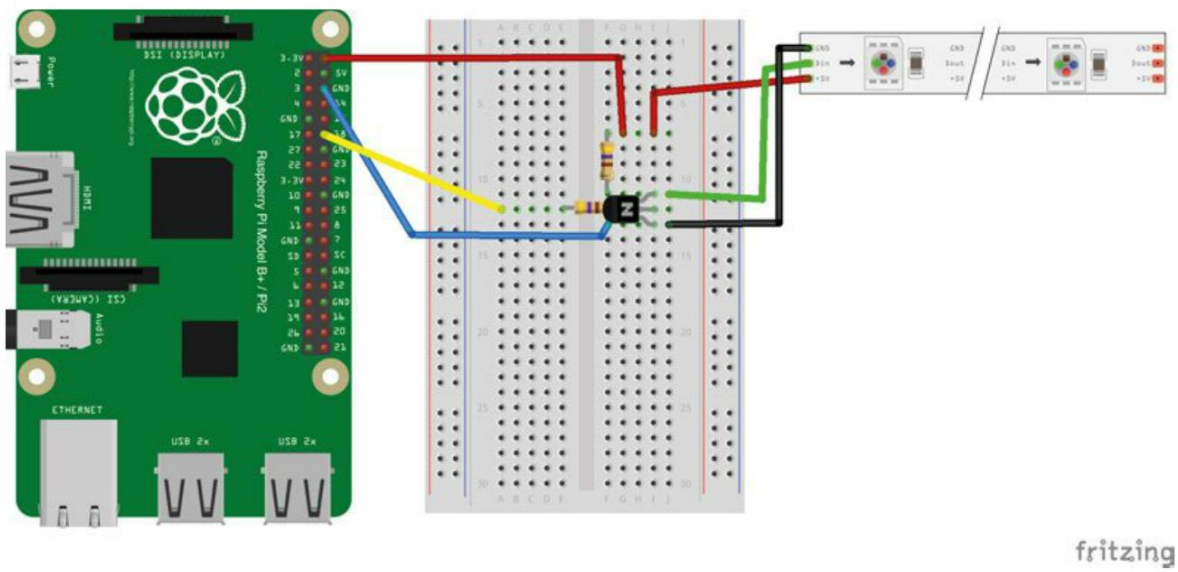


Figure 14-6. *Raccordement d'un Raspberry Pi avec conversion de niveau*

Programme Raspberry Pi

Le programme utilisé ici est basé sur le tutoriel Adafruit. Toutefois, la bibliothèque C employée dans le tutoriel Adafruit n'est pas compatible avec le Raspberry Pi 2 (au moment où nous écrivons ces lignes).

Heureusement pour les utilisateurs du Raspberry Pi 2, Richard Hurst a créé une version du programme qui fonctionne avec le nano-ordinateur. Cette version fonctionne aussi avec les modèles de Raspberry Pi précédents. Installez les programmes et la bibliothèque à l'aide de la commande suivante :

```
$ sudo apt-get install build-essential python-dev git scons swig
```

L'étape suivante ne suit pas le tutoriel Adafruit, car vous devez utiliser la version du logiciel compatible avec le Raspberry Pi 2.

Récupérez le code NeoPixel modifié sur GitHub à l'aide de la commande :

```
$ git clone https://github.com/richardghirst/rpi_ws281x.git
```

Ouvrez le dossier contenant le logiciel récupéré sur GitHub, puis compilez le code C à l'aide de la commande :

```
$ scons
```

Quand le code C est compilé, installez la bibliothèque Python contenant l'interface avec le code C à l'aide de la commande :

```
$ cd python
$ sudo python setup.py install
```

Il y a deux versions presque identiques du programme Python : l'une optimisée pour fonctionner avec un convertisseur-inverseur de niveau et l'autre conçue pour fonctionner avec le ruban de LED directement connecté au Raspberry Pi. Par conséquent, exécutez `neopixel_no_level_conv.py` ou `neopixel.py` (si vous avez réalisé le circuit sur la plaque d'essai) selon votre configuration matérielle. Les deux programmes se trouvent dans `python/experiments/`.

Le programme suivant correspond à la version utilisant un circuit de convertisseur de niveau produisant une inversion :

```
import time, random
from neopixel import *    ❶

# LED strip               ❷
LED_COUNT      = 30      # Nombre de pixels LED.
LED_PIN        = 18      # Broche GPIO connectée aux pixels (doit être compatible PWM!).
LED_FREQ_HZ    = 800000  # Fréquence du signal LED en hertz (généralement 800 khz)
LED_DMA        = 5       # Canal DMA utilisé pour générer le signal (essayez 5)
LED_BRIGHTNESS = 255     # 0 = le moins lumineux ; 255 le plus lumineux
LED_INVERT     = True    # True pour inverser le signal (en cas de décalage de niveau
                         # par transistor NPN)

# Initialise l'affichage
strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT, LED_BRIGHTNESS)
strip.begin()

while True:              ❸
    for i in range(strip.numPixels()):
        red = random.randint(0, 255)
        green = random.randint(0, 255)
        blue = random.randint(0, 255)
        strip.setPixelColor(i, Color(red, green, blue))
        strip.show()
    time.sleep(0.1)
```

❶ Importe la bibliothèque NeoPixel.

❷ Il est inutile de corriger ce groupe de paramètres, sauf si vous voulez utiliser une autre `LED_PIN`.

❸ La boucle principale du programme ressemble beaucoup à l'équivalent Arduino : elle génère une couleur aléatoire, puis l'attribue à un pixel.

Écrans OLED I2C

Bien qu'un Raspberry Pi puisse être connecté à n'importe quel moniteur (grand ou petit) muni d'un port HDMI ou AV, vous n'avez parfois besoin que d'afficher quelques lignes de texte. Toutefois, l'Arduino Uno n'a pas de sortie vidéo. Par conséquent, un petit écran permettant d'afficher une image ou quelques lignes de texte peut se révéler très utile.

Les petits écrans à LED organiques (OLED) sont bon marché, ils ne consomment pas beaucoup d'énergie et ils sont très lisibles (figure 14-7). Ils remplacent les écrans LCD sur de nombreux produits courants. Ils sont disponibles en écrans monochromes et couleur.

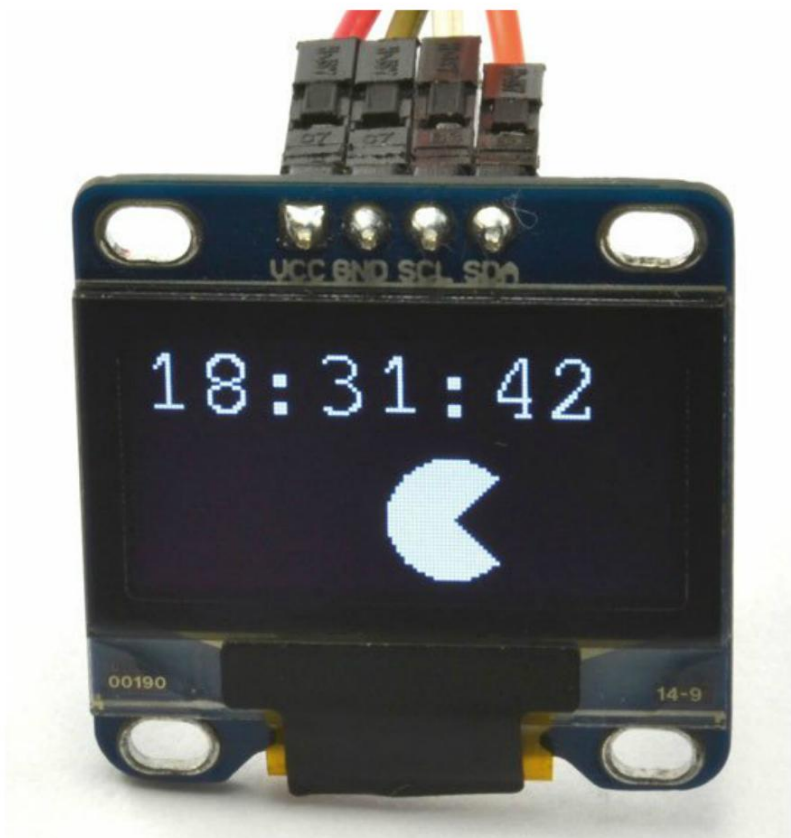


Figure 14-7. *Un afficheur OLED*

Un modèle très répandu d'écran OLED se présente sous la forme d'un module composé de l'écran proprement dit et d'un contrôleur sur un circuit imprimé. La carte de pilote utilise généralement une puce SSD1306 dotée d'une interface I2C qui ne nécessite que deux broches pour la partie commande et deux broches pour la partie puissance.

Ces petits écrans ont une résolution élevée. Par conséquent, si vous utilisez un Arduino, vous risquez d'arriver vite à court de mémoire.

Expérience : utilisation d'un module d'affichage I2C avec un Raspberry Pi

Les écrans OLED I2C peuvent être utilisés avec un Arduino et un Raspberry Pi. Dans le projet « Ajout d'un écran au système de réfrigération de boisson » (voir plus loin), vous remplacerez la LED de température par un écran OLED qui affiche à la fois la température courante et la température de consigne.

Cette expérience sert simplement à vous montrer comment utiliser ce type d'afficheur avec un Raspberry Pi. Le code affiche l'heure et une petite animation (figure 14-8).

Figure 14-8. *Une horloge à écran OLED*

Composants nécessaires

L'écran OLED possède un connecteur à quatre broches qui peuvent être reliées directement au Raspberry Pi à l'aide de quatre câbles femelles/femelles.

Composant	Source
Écran OLED I2C 128 × 64 pixels	eBay
4× câbles flexibles femelles/femelles	Adafruit : 266

Choisissez un écran de 128 × 64 pixels doté d'un contrôleur SSD1306. Certains modèles utilisent des broches supplémentaires du SSD1306 qui ne sont pas indispensables. Pour simplifier, choisissez un écran à quatre broches seulement : GND, VCC (+V), SDA (données) et CLK (horloge). J'ai choisi un modèle monochrome, mais rien n'empêche d'utiliser un écran couleur.

Si vous utilisez l'un des écrans Adafruit, suivez les instructions fournies sur le site d'Adafruit pour le raccordement des broches supplémentaires.

Connexions

Le modèle SSD1306 fonctionne en 3 V ou 5 V. Les anciens Raspberry Pi pouvaient uniquement fournir des courants faibles de 3 V, il est donc préférable de l'alimenter en 5 V.

Connectez les ports comme suit :

- GND de l'écran à GND du Raspberry Pi
- VCC de l'écran à 5 V du Raspberry Pi
- SCL de l'écran à GPIO 3 du Raspberry Pi
- SDA de l'écran à GPIO 2 du Raspberry Pi

Programme

Si vous n'avez pas encore configuré le Pi pour la prise en charge du protocole I2C, reportez-vous à l'encadré « Configuration du protocole I2C sur le Raspberry Pi » au chapitre 9.

Adafruit a produit une excellente bibliothèque Python pour faciliter l'utilisation de ces écrans. Elle est compatible avec la plupart des écrans SSD1306, qu'ils soient commercialisés par Adafruit ou un autre vendeur. Pour télécharger la bibliothèque, vous pouvez cloner directement l'archive de la bibliothèque sur votre Raspberry Pi à l'aide de la commande :

```
$ git clone https://github.com/adafruit/Adafruit-SSD1331-OLED-Driver-Library-for-Arduino.git
```

Pour installer la bibliothèque, placez-vous dans le dossier créé par la commande `clone`, puis exécutez le script d'installation :

```
$ cd Adafruit_Python_SSD1306
$ sudo python setup.py install
```

Vous trouverez le programme de l'expérience dans le fichier `python/experiments/oled.py`.

Coordonnées

L'afficheur OLED est un écran graphique capable d'afficher des images et du texte. Toutefois, il est indispensable de préciser la position de ce qui doit être affiché. La bibliothèque Adafruit utilise le système de coordonnées présenté à la figure 14-9.

Toutes les positions des pixels sont indiquées par rapport au coin supérieur gauche de l'écran. Les coordonnées du pixel situé dans le coin inférieur droit sont donc x = 127, y = 63.

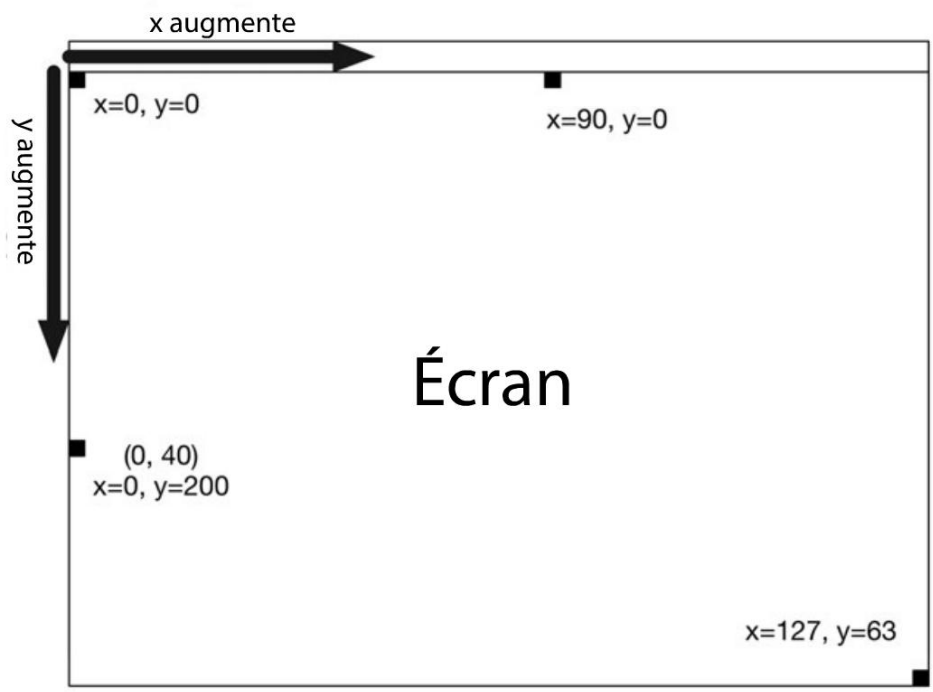


Figure 14-9. *Le système de coordonnées de l'afficheur*

```
from oled.device import ssd1306      ❶
from oled.render import canvas
from PIL import ImageFont            ❷
import time

device = ssd1306(port=1, address=0x3C)    ❸
large_font = ImageFont.truetype('FreeMono.ttf', 24)    ❹

x = 0
```

```
while True:
    with canvas(device) as draw:    ❺
        draw.pieslice((x, 30, x+30, 60), 45, -45, fill=255)    ❻
        x += 10    ❼
    if x > 128:
        x = 0
    now = time.localtime()          ❽
    draw.text((0, 0), time.strftime('%H:%M:%S', now), font=large_font, fill=255)
time.sleep(0.1)
```

❶ Deux importations sont effectuées à partir de la bibliothèque OLED : l'une de `ssd1306` pour le composant lui-même et l'autre pour `canvas`, qui est l'objet Python sur lequel des formes et du texte sont tracés.

❷ La bibliothèque PIL *(Python Image Library)* est également importée.

❸ Crée une variable (`device`) pour accéder à l'afficheur. La valeur `0x3C` utilisée ici correspond à l'adresse I2C de l'écran. C'est l'adresse généralement utilisée par les écrans bon marché achetés sur eBay, mais d'autres modèles, y compris ceux d'Adafruit, utilisent une autre adresse. Reportez-vous à la documentation de votre module d'affichage.

❹ Précise la police à utiliser avec une hauteur de 24 pixels. Elle servira à écrire l'heure sur l'écran.

❺ Cette construction `with as` permet de regrouper tout le code d'affichage du texte.

❻ Dessine une forme de PacMan mesurant 30 × 30 pixels avec un angle de 45° à -45° pour la bouche. La couleur de remplissage est 255, ce qui correspond au blanc.

❼ Ajoute 10 à `x`. La variable `x` définit la position de départ du graphique `pieslice`. En l'augmentant de 10 à chaque exécution de la boucle, on obtient une animation rudimentaire.

❽ Lit l'heure actuelle, la formate en chaîne de caractères, puis l'écrit sur l'afficheur.

Expérimentation

Exécutez le programme à l'aide de la commande :

```
$ sudo python oled.py
```

Si l'écran reste noir, le problème est probablement dû à une adresse I2C incorrecte.

Cet exemple utilise la fonction `piesclice` qui dessine un cercle dont une part a été coupée. Vous trouverez d'autres fonctions graphiques sur *http://effbot.org/imagingbook/imagedraw.htm*. Testez-les pour améliorer votre affichage.

Projet : ajout d'un écran au système de réfrigération de boisson

Ce projet illustre l'utilisation d'un écran OLED avec une carte Arduino. Il se base sur le projet « Système thermostatique de réfrigération de boisson » au chapitre 12. La LED verte, qui indique que le système de réfrigération est à la bonne température, est remplacée par un écran OLED qui affiche les températures courantes et de consigne (figure 14-10).

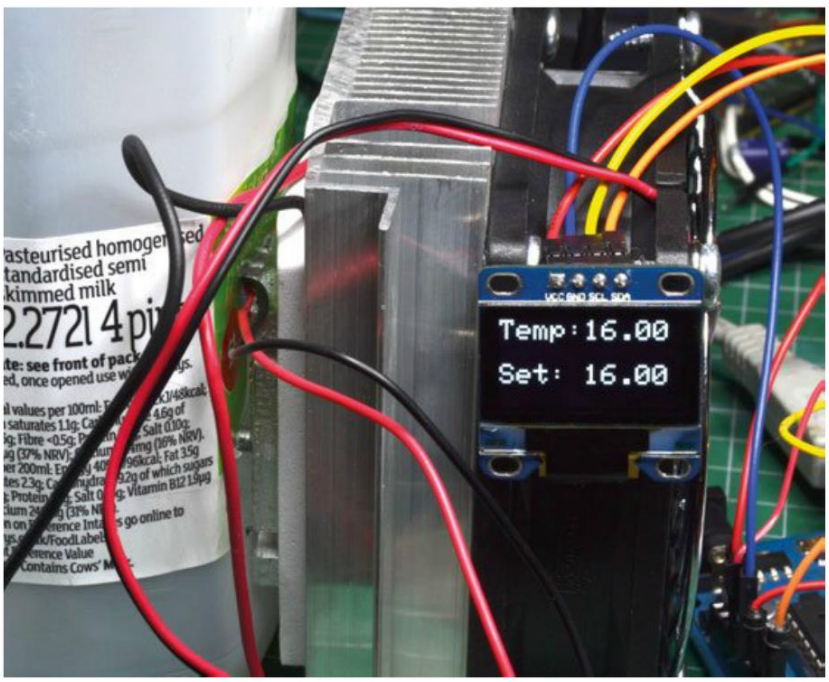


Figure 14-10. *Ajout d'un écran OLED au projet de réfrigération de boisson*

Composants nécessaires

En plus des pièces utilisées pour le projet « Système thermostatique de réfrigération de boisson » au chapitre 12, (moins la LED et R3), vous avez besoin d'un écran OLED du type de celui utilisé dans l'expérience « Utilisation d'un module d'affichage I2C avec un Raspberry Pi », plus haut, et quatre câbles flexibles femelles/mâles supplémentaires.

Connexions

Si vous ne l'avez pas déjà fait, vous devez réaliser le projet « Système thermostatique de réfrigération de boisson » au chapitre 12. Vous n'avez pas besoin de monter la LED et R3.

L'écran OLED peut être connecté directement sur l'Arduino, ce qui limite le nombre de composants. La figure 14-11 présente uniquement la nouvelle partie du projet avec l'écran OLED connecté à l'Arduino.

Pensez à vérifier le brochage du module OLED. Sur certains modèles, les broches 5 V et GND sont interverties.

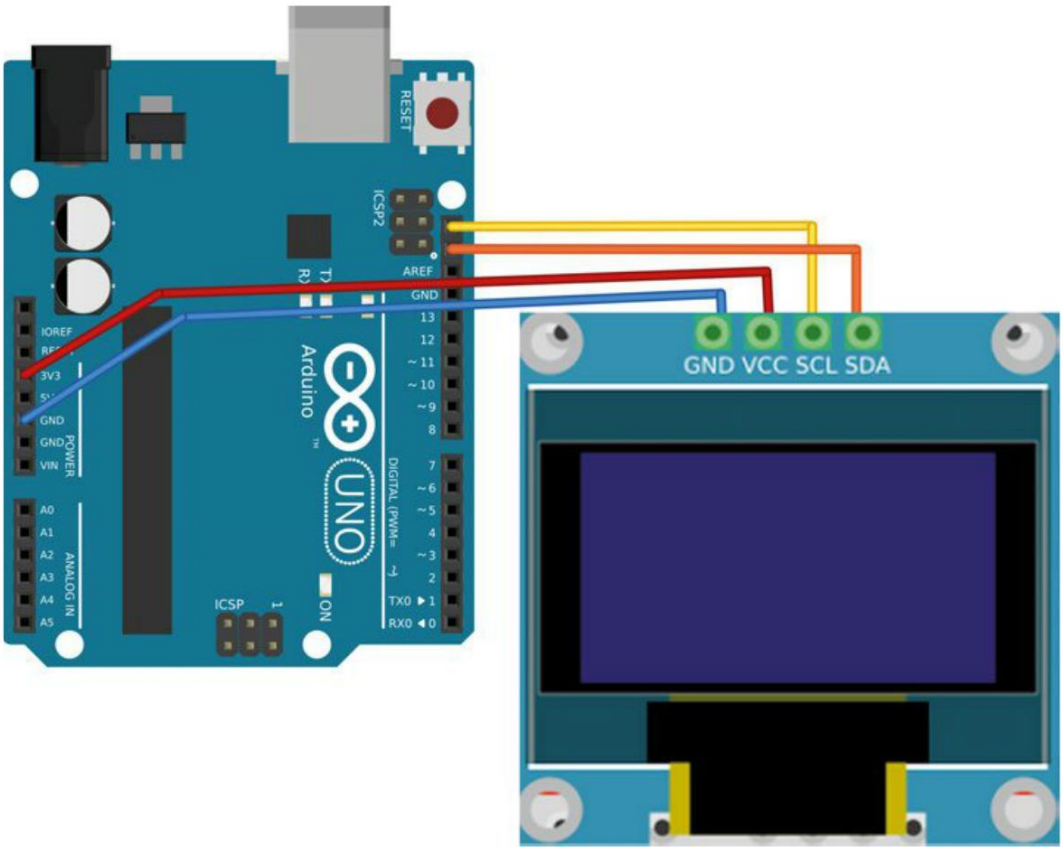


Figure 14-11. *Raccordement de l'écran OLED à la carte Arduino*

Programme

Comme vous devez vous en douter, il existe aussi une bibliothèque Arduino pour l'écran OLED. Vous pouvez la télécharger et l'installer sur votre IDE Arduino (voir « Installation des bibliothèques Arduino » au chapitre 12).

Le sketch de cette version du système de réfrigération ressemble beaucoup à celui du système sans affichage. Vous le trouverez dans `pr_thermostatic_cooler_display`. Les modifications concernent notamment l'ajout des bibliothèques Adafruit GFX et SSD1306. Cette dernière est destinée à l'écran proprement dit et la bibliothèque GFX réunit des fonctions utiles pour l'affichage de texte et de graphiques :

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

La variable `display` est définie pour servir de référence à la bibliothèque. Le paramètre 4 correspond au numéro de la broche Enable de l'écran, le cas échéant. Notre écran n'en a pas, donc nous avons choisi la broche 4 puisqu'elle n'est pas utilisée par la carte Arduino :

```
Adafruit_SSD1306 display(4);
```

`setup()` réunit les lignes suivantes qui initialisent l'afficheur :

```
display.begin(SSD1306_SWITCHCAPVCC, 0x3c);
```

La fonction `checkTemperature` inclut désormais un appel à la nouvelle fonction `updateDisplay` pour actualiser la température courante qui est affichée toutes les secondes :

```
void updateDisplay() {
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.print("Temp:");
  display.println(measuredTemp);
  display.print("Consgne: ");
  display.println(setTemp);
  display.display();
}
```

La fonction `readSetTempFromPot` a été légèrement modifiée. Désormais, dès que la température de consigne change, `updateDisplay` est appelé immédiatement. Lorsque vous tournez le potentiomètre, la température de consigne est mise à jour sans attendre la seconde suivante :

```
double readSetTempFromPot() {
  static double oldTemp = 0;
  int raw = analogRead(potPin);
  double temp = map(raw, 0, 1023, minTemp, maxTemp);
  if (oldTemp != temp) {
    updateDisplay();
    oldTemp = temp;
  }
  return temp;
}
```

Le mot-clé `static` placé avant la définition d'`oldTemp` signifie qu'`oldTemp` conserve sa valeur entre des appels successifs de `readSetTempFromPot`. Les variables `temp` et `oldTemp` sont toutes les deux de type `double` *(double-precision float)*, car c'est le type utilisé par la bibliothèque DS18B20.

Résumé

Il existe de nombreux types d'afficheurs et ce chapitre n'a abordé que certains des plus répandus.

Dans le chapitre suivant, vous apprendrez à utiliser une carte Arduino et Raspberry Pi pour créer des sons.

15 Son

Après avoir créé du mouvement, de la lumière, de la chaleur et du froid, mais aussi des affichages, il est temps de nous intéresser au son.

Il est facile de produire un son de qualité sur le Raspberry Pi, car il est possible d'y relier des enceintes actives sur la prise jack audio. En revanche, c'est un peu plus compliqué avec l'Arduino.

Expérience : enceinte non amplifiée et Arduino

Tôt ou tard, si vous voulez produire un son complexe, vous aurez besoin d'une enceinte acoustique. Les haut-parleurs existent depuis près d'un siècle et ils fonctionnent à la façon d'un solénoïde (voir la section « Solénoïdes » du chapitre 7 page 122) qui pousse un cône rigide à une fréquence suffisamment élevée pour créer des ondes sonores.

Les enceintes sont généralement désignées par une valeur en ohms. Cette valeur correspond à leur impédance. Comme son nom l'indique, l'impédance ressemble à la résistance, mais elle s'applique à des choses qui ne sont pas des résistances pures. En effet, la bobine qui se trouve à l'intérieur de l'enceinte ne se comporte pas tout à fait comme une résistance. Si le sujet vous intéresse, je vous invite à faire des recherches sur l'inductance.

Les enceintes ont généralement une impédance de 4 Ω ou 8 Ω. Si vous raccordez un haut-parleur de 8 Ω sur la sortie 5 V d'une carte Arduino, vous pourriez vous attendre à obtenir un courant de I = V/R = 5/8 = 625 mA. C'est beaucoup plus que les 40 mA recommandés pour une broche de sortie Arduino. On aurait bien besoin d'une résistance !

Dans cette expérience, vous connecterez une enceinte sur un Arduino via une résistance, puis vous utiliserez le moniteur série pour demander à l'Arduino de produire des sons ayant une certaine fréquence.

Fréquences audio

La fréquence de l'onde sonore s'appelle la hauteur (ou *pitch*, en anglais) dans le jargon de la musique : elle correspond au nombre d'ondes sonores parvenant à nos oreilles par seconde. Les ondes sonores ressemblent aux rides qui se forment à la surface de l'eau d'une mare. Un son haute fréquence de 10 kHz (kilohertz), par exemple, aura 10 000 ondes sonores par seconde et un son basse fréquence à 100 Hz, par exemple, aura seulement 100 ondes par seconde. On considère généralement que les limites de l'ouïe humaine se situent entre 20 Hz et 20 kHz. Toutefois, la limite haute diminue avec l'âge. Les sons de plus de 20 kHz s'appellent des ultrasons.

D'autres espèces animales sont sensibles à différentes plages de fréquences. Par exemple, les chats peuvent entendre des fréquences pouvant s'étendre de 55 kHz à 79 kHz. Sans parler des chauves-souris qui utilisent l'écholocalisation par ultrasons.

En musique, le do le plus grave d'un piano standard a une fréquence de 32,7 Hz et pour le do plus haut, elle est de 4 186 Hz. La fréquence est doublée lors du passage d'une octave à l'octave supérieure. Si vous prenez deux do voisins sur le clavier du piano, le second a le double de la fréquence du premier.

Composants nécessaires

Vous n'avez pas besoin de grand-chose pour réaliser cette expérience, juste une enceinte et une résistance, même si une plaque d'essai et quelques câbles flexibles faciliteront le montage :

Composant	Source
Petite enceinte 8 Ω	Adafruit : 1891
Résistance 270 Ω 1/4 W	Mouser : 291-270-RC
Plaque d'essai sans soudure à 400 contacts	Adafruit : 64
Câbles flexibles mâles/mâles	Adafruit : 758

Je me suis servi d'un haut-parleur récupéré sur un vieux poste de radio. Il était muni d'un connecteur dans lequel il était possible d'insérer un câble flexible mâles/mâles. Vous pouvez aussi vous procurer une petite enceinte munie de fils qui s'enfichent sur une plaque d'essai ou sur un Arduino. Sinon, vous pouvez aussi souder des câbles suffisamment fins pour pouvoir être enfoncés dans les trous de la plaque d'essai.

Réalisation du circuit

La figure 15-1 montre la réalisation du circuit de l'expérience.

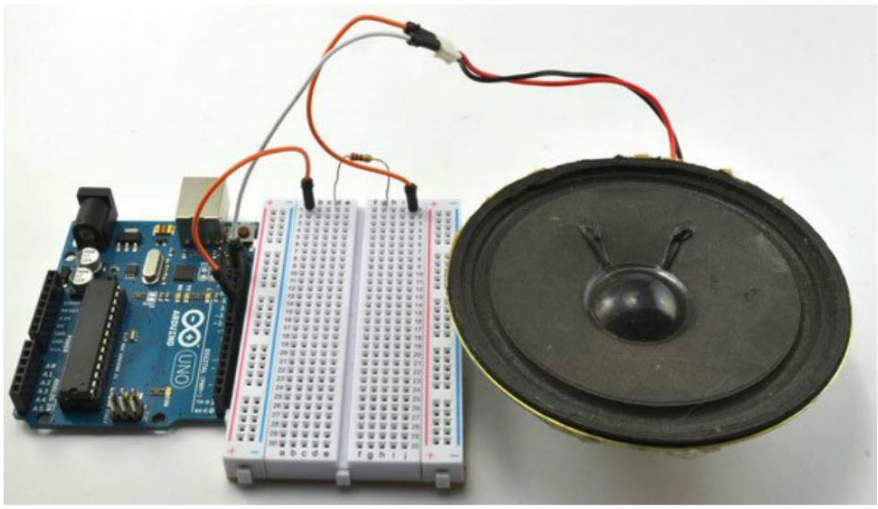


Figure 15-1. *Arduino et enceinte*

Un fil de l'enceinte est raccordé à la broche GND de l'Arduino et l'autre est connecté via la résistance à la broche D11.

Programme Arduino

Voici le sketch de ce projet que vous trouverez dans /arduino/experiments/ex_speaker (pour plus d'informations sur l'installation des sketches Arduino du livre, voir la section « Le code du livre » du chapitre 2 page 14) :

```
const int soundPin = 11;

void setup() {
  pinMode(soundPin, OUTPUT);
  Serial.begin(9600);
  Serial.println(«Saisir la fréquence»);
}

void loop() {
  if (Serial.available()) {
    int f = Serial.parseInt();
    tone(soundPin, f);      ❶
    delay(2000);
    noTone(soundPin);       ❷
  }
}
```

La partie du code qui doit vous paraître moins familière se trouve à l'intérieur de `loop()`.

❶ `tone` définit l'une des broches de sortie de l'Arduino de telle sorte qu'un son soit émis à la fréquence spécifiée (ici, la fréquence saisie dans le moniteur série).

❷ Au bout d'un délai de deux secondes, la commande `noTone` annule le ton et le silence revient.

Expérimentation Arduino

Téléversez le programme, puis ouvrez le moniteur série. Saisissez 1000. Cela devrait produire un son qui n'est pas particulièrement agréable avec un volume raisonnable, mais pas suffisamment fort pour être entendu dans une pièce bruyante.

Saisissez différentes fréquences et écoutez la façon dont la note change.

Même si vous pourriez être tenté de tester la plage de fréquences de votre audition, ce n'est malheureusement pas possible, car l'enceinte a elle aussi une plage de fréquences limitée. À des fréquences supérieures à 10 kHz environ, le niveau du son produit chute nettement. En outre, un petit haut-parleur ne peut normalement pas produire un son inférieur à 100 Hz.

Ondes sinusoïdales et ondes carrées

Le son obtenu en connectant un Arduino directement à partir d'une broche de sortie peut paraître un peu haché. Cela s'explique par le fait qu'une sortie numérique peut uniquement être allumée ou éteinte. Donc la forme de l'onde sonore produite s'appelle une onde carrée.

Un son ayant une onde carrée ne paraît pas très naturel. Au contraire, les instruments de musique produisent un son plus lisse, qui est plus proche d'une onde sinusoïdale (voir figure 15-2).

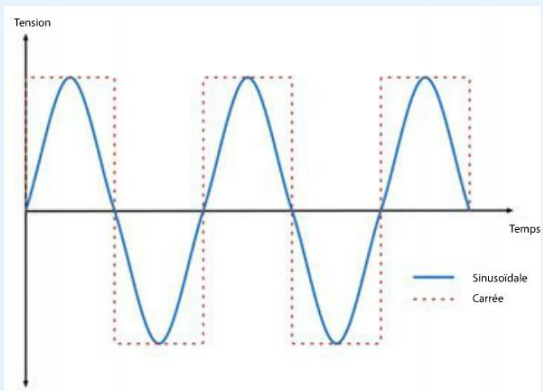


Figure 15-2. *Ondes sinusoïdales et ondes carrées*

Amplificateurs

Si vous voulez que le son soit plus fort, vous devez fournir plus de puissance à votre haut-parleur. En d'autres termes, vous devez amplifier le signal afin que davantage de puissance atteigne le haut-parleur.

Dans l'installation réalisée pour l'expérience « Enceinte non amplifiée et Arduino », plus haut, il était possible d'obtenir beaucoup plus de puissance dans l'enceinte à l'aide d'un seul transistor, comme si vous commutiez un relais ou un moteur. Cela ne rend pas le son plus agréable, mais son volume est plus élevé.

Si vous voulez générer une plus jolie forme d'onde (pour le son de la musique ou de la voix), la méthode par commutation ne permet pas d'obtenir des résultats satisfaisants. Vous devez envisager d'utiliser un amplificateur audio digne de ce nom.

Même si vous pouvez fabriquer votre amplificateur de toutes pièces, il est beaucoup plus facile d'utiliser un module prêt à l'emploi ou des enceintes actives pour PC. Il est particulièrement intéressant d'utiliser des enceintes actives (amplifiées) avec un Raspberry Pi, car la liaison AUX peut être reliée directement à la prise audio du nano-ordinateur.

Cette méthode sera utilisée plus loin dans ce chapitre pour faire parler Pepe la marionnette (voir le projet « Pepe, la marionnette Raspberry Pi qui danse » du chapitre 9 page 165, et le projet « Pepe, la marionnette Raspberry Pi qui parle » page 297).

Expérience : lecteur de fichiers son sur un Arduino

Il est possible de lire des fichiers audio au format WAV sur un Arduino muni des composants utilisés pour l'expérience "Enceinte non amplifiée et Arduino » plus haut, et d'une bibliothèque Arduino intitulée PCM. La technique employée ressemble un peu à celle de la modulation de largeur d'impulsion MLI pour générer un son approximatif. La mémoire flash de l'Arduino ne peut contenir que 4 secondes environ d'enregistrement. Si vous voulez lire des morceaux plus longs, vous devrez ajouter un lecteur de carte SD à l'Arduino et utiliser une méthode comme celle décrite sur le site web d'Arduino (*https://www.arduino.cc/en/Tutorial/SimpleAudioPlayer*).

Vous pouvez enregistrer le son sur votre ordinateur à l'aide du logiciel Audacity, puis vous servir d'un utilitaire pour convertir le fichier audio en une série de nombres qui représentent le son afin de les coller dans un sketch Arduino qui sera ensuite lu.

L'article original qui décrit cette approche est paru sur High-Low Tech (*http://highlowtech.org/?p=1963*). Cette expérience est légèrement différente, car elle utilise le logiciel gratuit Audacity pour enregistrer un clip audio.

Composants nécessaires

Le matériel nécessaire pour cette expérience est identique à celui de l'expérience « Enceinte non amplifiée et Arduino » plus haut. Toutefois, vous devrez installer les logiciels suivants sur votre ordinateur pour pouvoir enregistrer et traiter un clip audio :

- Audacity
- L'utilitaire Audio Encoder (le lien correspondant à votre système d'exploitation est disponible sur *http://highlowtech.org/?p=1963*)

Création des données audio

Si vous ne voulez pas enregistrer votre propre clip audio, passez directement à la section « Expérimentation Arduino » page 294, et exécutez le sketch `ex_wav_arduino` dans lequel un petit message a été encodé.

Pour créer le fichier audio, vous devez tout d'abord démarrer Audacity. Avant de lancer l'enregistrement, définissez le mode Mono et le débit du projet sur 8 000 Hz. Ces options sont entourées en rouge sur la figure 15-3.

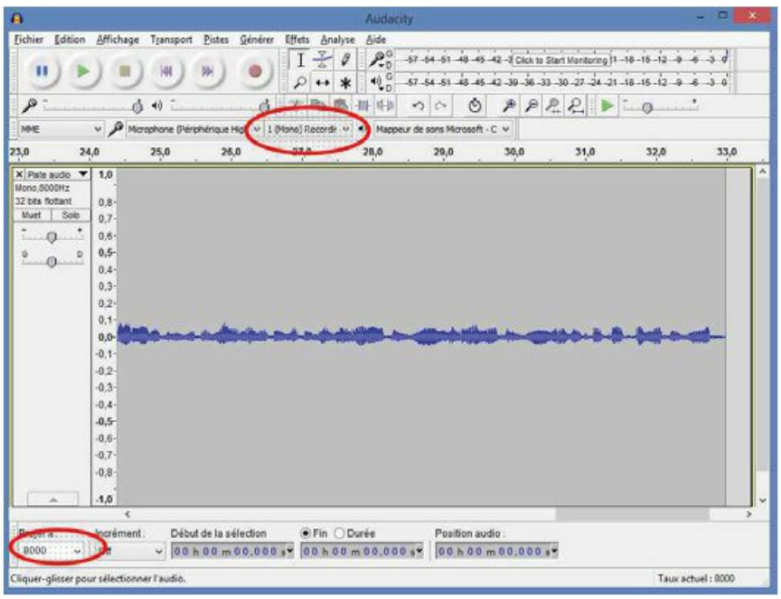


Figure 15-3. *Enregistrement d'un clip audio*

Cliquez sur le bouton rouge pour démarrer l'enregistrement de votre annonce. Notez que sa durée maximale est limitée à 4 secondes. Une fois le message enregistré, le fichier audio s'affiche dans Audacity. Vous pouvez sélectionner les plages de silence au début et à la fin pour les effacer et ne conserver que l'extrait voulu.

L'étape suivante consiste à exporter le fichier audio. Là encore, il faut ajuster quelques options. Dans le menu Fichier, sélectionnez Exporter audio. Puis, dans la liste déroulante Format, sélectionnez Autres formats non compressés. Cliquez sur Options et sélectionnez WAV (Microsoft) et Unsigned 8 bits PCM (figure 15-4). Saisissez un nom de fichier et passez l'écran vous demandant plus d'informations sur l'artiste.

Le fichier que vous venez de créer est composé de données binaires. Il doit être converti en une suite de nombres, chacun séparé par des virgules, qui peuvent être collés dans le sketch. À cette fin, exécutez l'utilitaire Audio Encoder que vous avez téléchargé depuis *highlowtech.org*. Une invite s'affiche pour vous demander de sélectionner le fichier que vous voulez convertir : sélectionnez le fichier que vous venez d'exporter depuis Audacity.

Au bout de quelques instants, un message de confirmation s'affiche pour indiquer que toutes les données se trouvent dans le Presse-papier.

Ouvrez le sketch `/arduino/experiments/wav_arduino`. Vous allez remplacer toute la ligne qui commence par 125, 119, 115 par les données qui se trouvent dans le Presse-papier. Comme c'est une très longue ligne, la meilleure façon de la sélectionner consiste à placer le curseur au début de la ligne, puis à maintenir la touche maj enfoncée tout en appuyant une fois sur la touche Flèche bas. Utilisez la commande Coller pour remplacer le texte sélectionné par les données du Presse-papier.

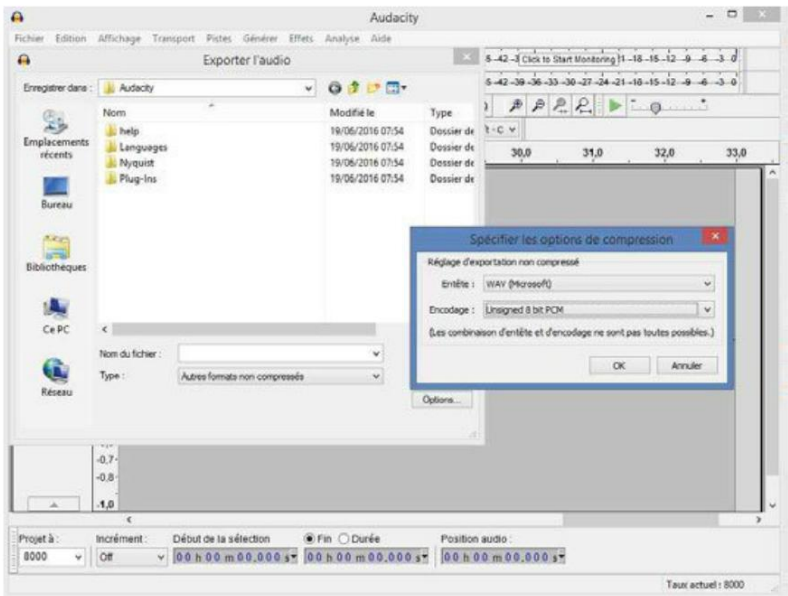


Figure 15-4. *Réglage des options d'exportation*

Si vous affichez ces nombres dans un graphique, la forme obtenue est identique à celle affichée dans Audacity lors de l'enregistrement du clip audio.

Code Arduino

Avant de compiler et exécuter le sketch, vous devez installer la bibliothèque PCM. Téléchargez l'archive ZIP du fichier depuis GitHub ; décompressez-la, renommez le dossier `PCM`, et déplacez-le dans votre dossier de bibliothèques Arduino, comme décrit dans l'encadré « Installation des bibliothèques Arduino » du chapitre 12 page 227.

Le sketch Arduino (si l'on ne tient pas compte des données audio) est extrêmement court :

```
#include <PCM.h>

const unsigned char sample[] PROGMEM = {     ❶
125, 119, 115, 115, 112, 116, 114, 113, 124, 126, 136, 145, 139,
};

void setup() {
  startPlayback(sample, sizeof(sample));     ❷
}
```

```
void loop() {     ❸
}
```

Toutefois, ce bloc de données représente une très longue ligne !

❶ Les données sont contenues dans un bloc de type `char` qui contient des nombres non signés de 8 bits. La commande `PROGMEM` permet de stocker les données dans la mémoire flash de la carte Arduino (qui a une capacité de 32 Ko).

❷ La bibliothèque PCM lit l'échantillon. `startPlayback` reçoit le bloc de données à lire et le volume de données en octets.

❸ Comme le clip audio est lu à chaque réinitialisation de la carte Arduino, la fonction `loop()` est vide.

Expérimentation Arduino

Installez le sketch sur votre carte Arduino. Dès que l'installation est terminée, le clip audio est lu !

Lorsque vous téléversez le sketch, un message s'affiche dans le bas de l'IDE Arduino pour indiquer quelle quantité de la mémoire flash Arduino est utilisée. Si le fichier est trop volumineux, un message d'erreur apparaît.

Raccordement d'un Arduino à un amplificateur

L'expérience précédente fonctionne étonnamment bien si l'on considère qu'elle n'utilise qu'une humble carte Arduino.

Le signal audio transmis par l'Arduino peut passer par une résistance pour maintenir un courant faible, mais la broche Arduino fonctionne à 5 V, ce qui est trop élevé pour servir d'entrée pour un amplificateur audio ordinaire. Par conséquent, avant de pouvoir connecter un Arduino à des enceintes actives afin d'obtenir un volume sonore plus élevé, il faut d'abord réduire sa tension de sortie.

Pour y parvenir, on peut utiliser une paire de résistances comme diviseur de tension.

Diviseur de tension

Un diviseur de tension est un circuit utilisant deux résistances afin de réduire une tension. Si cette tension est variable, comme celle d'un signal audio, le réducteur de tension divise la tension dans une proportion fixe.

La figure 15-5 montre le diviseur de tension utilisé pour réduire le signal de 5 V d'un Arduino à (plus ou moins) un demi-volt, ce qui convient bien mieux à un amplificateur audio.

La tension de sortie à laquelle les deux résistances sont reliées (V_{out}) est calculée à l'aide de la formule suivante :

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in}$$

Dans ce cas, quand la sortie numérique est `HIGH`, $V_{in} = 5$ V, donc $V_{out} = 5 \times 1/(1+10) = 0{,}45$ V.

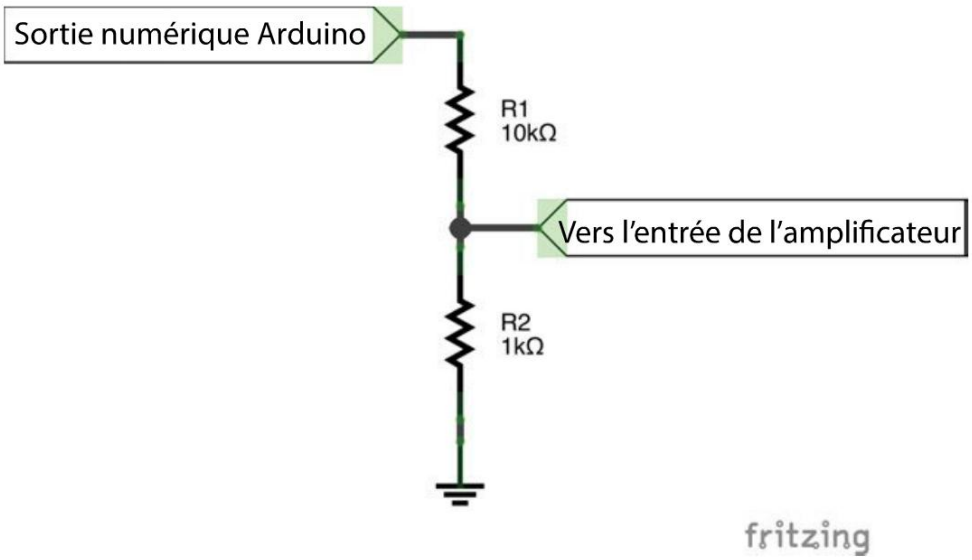


Figure 15-5. *Un diviseur de tension*

Vous pouvez adapter la plaque d'essai de l'expérience « Enceinte non amplifiée et Arduino » réalisée plus haut, en plaçant une résistance au-dessus de l'autre, comme illustré à la figure 15-6 : le haut de la résistance 10 kΩ est connecté à D11 et le bas de la résistance du bas est connecté à GND. Il ne vous reste plus qu'à connecter GND et la rangée de la plaque d'essai sur laquelle les résistances rejoignent l'amplificateur.

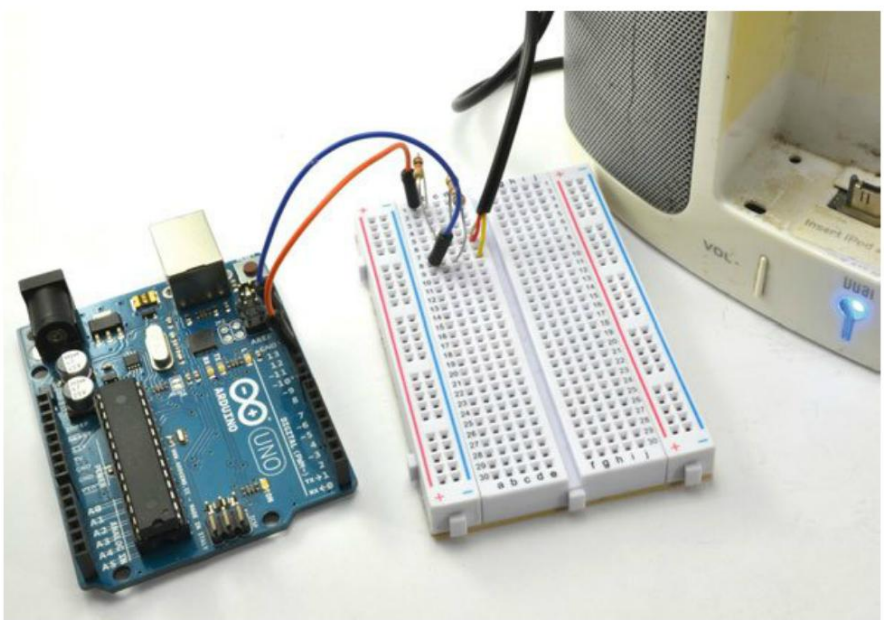


Figure 15-6. *Raccordement d'une carte Arduino à un câble d'entrée AUX*

Une façon d'établir cette connexion est de sacrifier un câble d'entrée AUX : coupez-le en deux et dénudez les fils à l'intérieur du câble pour établir la connexion. En général, les câbles se composent de trois fils, car la plupart sont stéréo. Il y a un fil de masse, et des câbles séparés pour les canaux audio gauche et droit, ces derniers sont souvent rouge et blanc.

Le seul fil que vous avez vraiment besoin d'identifier est le fil de masse, puisque les fils de droite et de gauche seront connectés ensemble pour que le signal mono transmis par l'Arduino soit diffusé sur les deux enceintes. Vous pouvez identifier le fil de masse à l'aide d'un multimètre en mode de contrôle de continuité (figure 15-7).

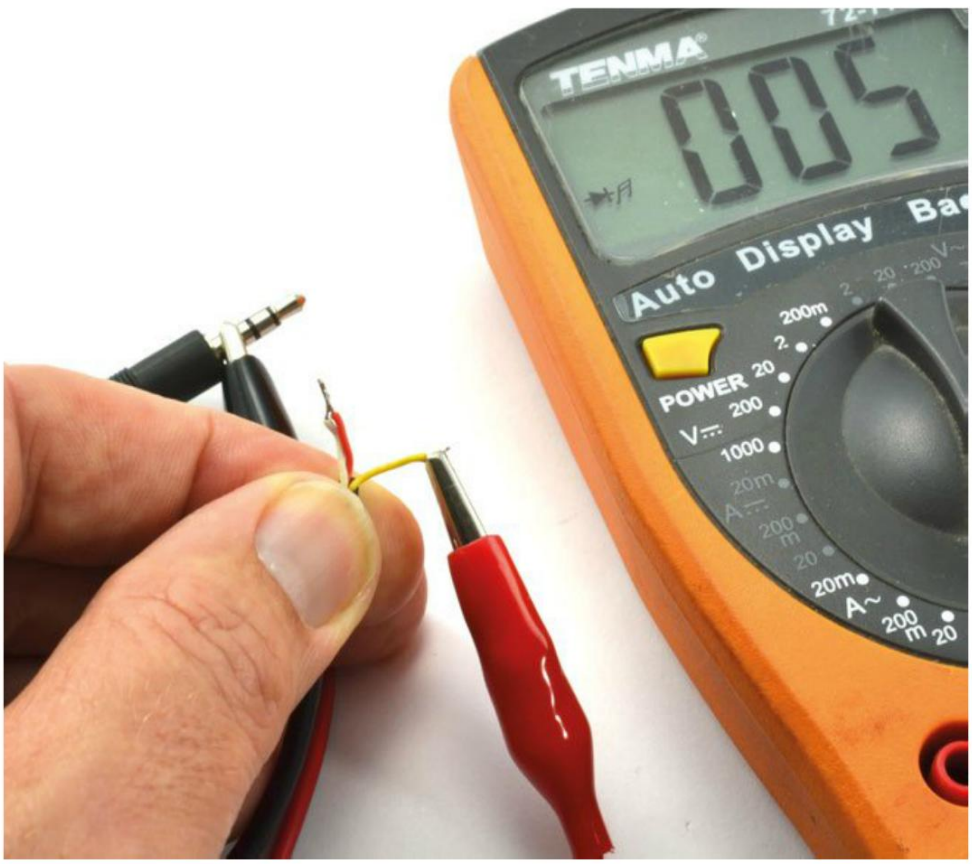


Figure 15-7. *Test du câble auxiliaire*

Avec l'un des fils du multimètre, touchez la borne de la prise la plus éloignée de son extrémité. Puis testez successivement chacun des trois conducteurs jusqu'à ce que le multimètre émette un signal sonore (ou indique qu'il y a une connexion). Il s'agit du fil de masse que vous pouvez enficher dans la rangée de masse (GND) de la plaque d'essai. Les deux autres fils peuvent être entortillés ensemble et enfichés dans la rangée de sortie de la plaque d'essai où les deux résistances se rejoignent.

Essayez de réaliser à nouveau l'une des expériences précédentes de ce chapitre. Le son émis devrait être beaucoup plus fort et de meilleure qualité.

Lecteur de fichiers audio sur le Raspberry Pi

Le Raspberry Pi est un ordinateur à part entière avec un jack de sortie audio. Par conséquent, pour pouvoir lire un fichier audio sur le Raspberry Pi, il vous suffit de trouver le programme adapté.

Vous pouvez procéder de différentes façons qui sont passées en revue au cours d'une discussion sur StackExchange (*http://raspberrypi.stackexchange.com/questions/7088/playing-audio-files-with-python*).

J'ai choisi d'utiliser ici une méthode faisant appel à la bibliothèque Python pygame qui est déjà installée sur le Raspberry Pi.

Les fichiers WAV, même s'ils sont beaucoup plus volumineux que les fichiers MP3, présentent le gros avantage d'être très faciles à décoder pour un Raspberry Pi et donc de ne pas trop le ralentir. Tandis qu'un Arduino ne lira pas n'importe quel fichier WAV, le Raspberry Pi n'a pas une capacité de mémoire ou une vitesse de traitement aussi limitée. Tous les fichiers WAV ou presque devraient pouvoir être lus correctement.

Faites l'essai depuis la ligne de commande Python en utilisant le fichier audio du projet « Pepe la marionnette qui parle », plus loin. Sur la ligne de commande Raspberry Pi, passez dans le répertoire contenant les fichiers téléchargés pour le livre, puis dans le dossier `python/projects/puppet_voice`. Vous y trouverez un fichier nommé `pepe_1.wav`. Pour lire ce fichier, connectez des enceintes actives ou des écouteurs sur la prise audio du Raspberry Pi, puis démarrez la console Python à l'aide de la commande `python` :

```
>>> from pygame import mixer
>>> mixer.init()
>>> mixer.music.load("pepe_1.wav")
>>> mixer.music.play()
```

Vous devriez entendre un petit message de Pepe.

Projet : Pepe, la marionnette Raspberry Pi qui parle

Maintenant que vous pouvez lire des fichiers audio avec votre Raspberry Pi, vous allez mettre en pratique ces connaissances en les associant au projet « Pepe, la marionnette Raspberry Pi qui danse » au chapitre 9, ainsi qu'à un capteur PIR (*Passive Infrared Sensor* ou IRP capteur infrarouge passif) pour faire danser et parler Pepe dès que l'on s'approche de la marionnette (figure 15-8).

Figure 15-8. *Pepe avec détection de mouvements et voix*

Composants nécessaires

Pour réaliser ce projet, vous avez besoin des éléments du projet « Pepe, la marionnette Raspberry Pi qui danse » au chapitre 9, ainsi que des composants suivants.

Composant	Source
Module de capteur à infrarouge passif (IRP)	eBay, Adafruit : 189
Câbles flexibles femelles/mâles	Adafruit : 826
Enceintes actives	
Plaque d'essai sans soudure à 400 contacts	Adafruit : 64

Les capteurs IRP sont utilisés comme détecteurs de mouvement dans les systèmes d'alarme. Ce module bon marché est idéal pour déclencher Pepe lorsqu'une personne s'approche. Il est conseillé d'ajouter une plaque d'essai pour le montage d'une carte de pilote des servos et les connecteurs du capteur IRP.

Réalisation du circuit

Dans ce projet, la plaque d'essai sert de support au câblage du module IRP et la carte de pilote des servos qui est directement enfichée dessus. La plupart des plaques d'essai comportent un adhésif sur la face inférieure. Collez-le sur le boîtier du châssis pour consolider l'ensemble. La figure 15-9 illustre la réalisation du circuit sur la plaque d'essai et la figure 15-10 présente le câblage du projet.

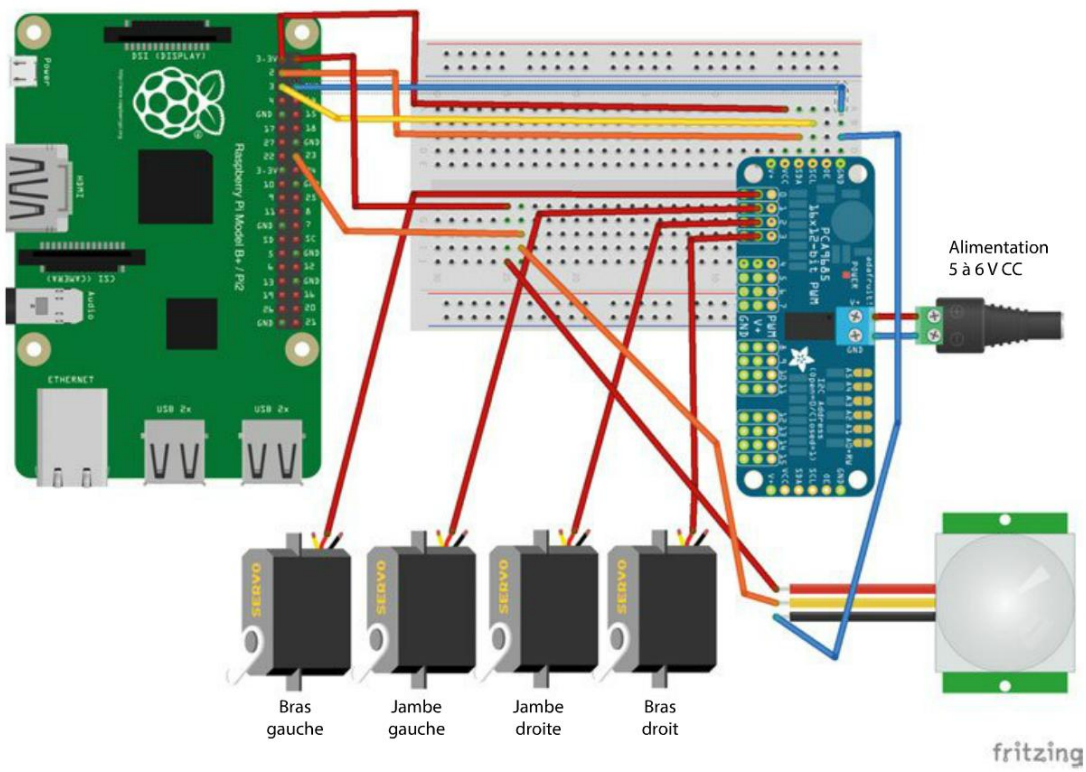


Figure 15-9. *Réalisation du circuit de la marionnette qui parle*

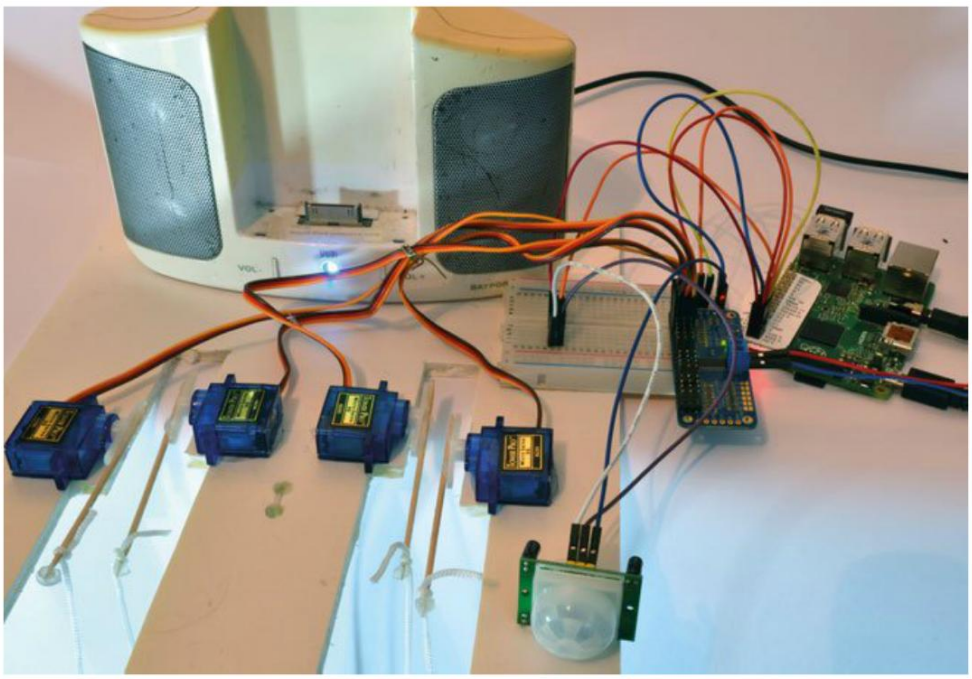


Figure 15-10. *La marionnette qui parle*

Le capteur IRP a une sortie numérique 3 V, mais il nécessite une alimentation en 5 V. Il est donc tout à fait adapté à un Raspberry Pi.

Capteurs IRP

Les capteurs IRP (infrarouge passif) détectent le mouvement de tout ce qui émet de la chaleur (comme les gens). Une sortie numérique du capteur est définie à l'état haut pendant une seconde ou deux chaque fois qu'est détecté un changement de niveau ou du motif infrarouge perçu par le capteur.

Pour le connecter à une carte Arduino ou un Raspberry Pi, il suffit de l'alimenter en électricité et de connecter sa sortie numérique à une entrée numérique du Raspberry Pi ou de la carte Arduino.

Programme

Le programme de ce projet est basé sur le projet « Pepe, la marionnette Raspberry Pi qui danse » au chapitre 9. Reportez-vous aussi à la description du code.

Vous trouverez tous les fichiers de ce projet dans le répertoire `python/projects/puppet_voice`. En plus du code du module servo Adafruit et du programme proprement dit (`puppet_voice.py`), vous trouverez aussi un fichier audio intitulé `pepe_1.wav`. Il s'agit du fichier audio qui sera lu lorsque Pepe est déclenché par un mouvement :

```
from Adafruit_PWM_Servo_Driver import PWM
import RPi.GPIO as GPIO
from pygame import mixer
import time

PIR_PIN = 23      ❶
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIR_PIN, GPIO.IN)

pwm = PWM(0x40)
mixer.init()      ❷
mixer.music.load("pepe_1.wav")

servoMin = 150 # Longueur d'impulsion mini sur 4096
servoMax = 600 # Longueur d'impulsion maxi sur 4096

dance = [
  #lh lf rf rh
  [130, 20, 20, 130],
  [30, 160, 160, 30],
  [90, 90, 90, 90]
]

delay = 0.2

def map(value, from_low, from_high, to_low, to_high):
  from_range = from_high - from_low
  to_range = to_high - to_low
  scale_factor = float(from_range) / float(to_range)
  return to_low + (value / scale_factor)

def set_angle(channel, angle):
  pulse = int(map(angle, 0, 180, servoMin, servoMax))
  pwm.setPWM(channel, 0, pulse)

def dance_step(step):
  set_angle(0, step[0])
  set_angle(1, step[1])
  set_angle(2, step[2])
  set_angle(3, step[3])

def dance_pupet():    ❸
    for i in range(1, 10):
        for step in dance:
            dance_step(step)
            time.sleep(delay)

pwm.setPWMFreq(60)
```

```
while True:
  if GPIO.input(PIR_PIN) == True:    ❹
      mixer.music.play()
      dance_pupet()
      time.sleep(2)
```

❶ Le premier code ajouté est celui qui permet de définir la broche 23 comme entrée numérique.

❷ Il y a toujours un code d'initialisation du mixeur avant la lecture du clip audio.

❸ La nouvelle fonction `dance_puppet` permet de faire répéter les pas de danse dix fois à la marionnette.

❹ Lorsque le capteur IRP est activé (la broche 23 est à l'état `True`), le fichier audio est lu et la danse commence. La fonction `music.play` s'exécute en arrière-plan.

Utilisation de Pepe la marionnette

Vous pouvez enregistrer un nouveau clip audio qui sera lu à l'aide du logiciel Audacity que vous avez utilisé dans la section « Expérimentation Arduino » page 290. Il suffit ensuite de remplacer le fichier `pepe_1.wav`. Vous pourriez aussi enregistrer plusieurs clips audio qui seront lus de façon aléatoire ou en fonction de l'heure.

Résumé

Dans ce chapitre, vous avez appris à utiliser le son avec une carte Arduino ou un Raspberry Pi.

Pour finir, nous verrons comment utiliser un Arduino, mais surtout un Raspberry Pi, avec l'Internet des objets.

L'Internet des objets

18

Les gens interagissent généralement avec l'Internet en se servant d'un navigateur pour parcourir des pages web. L'Internet des objets (*Internet of Things*, ou IoT) se base sur l'idée que des objets peuvent aussi avoir une présence sur la toile. Des systèmes de domotique intelligents sont connectés à l'Internet et peuvent fournir des informations utiles aux propriétaires ou aux fournisseurs d'eau, d'électricité, de gaz, etc. En outre, les utilisateurs d'appareils mobiles, comme des montres et des bracelets connectés, font aussi partie de l'IoT, car des informations personnelles sur leur localisation et leur fréquence cardiaque sont communiquées via Internet.

L'utilisation d'appareils électroniques prend une toute nouvelle dimension lorsque ce contrôle est effectué via l'Internet. Dans ce chapitre, vous apprendrez à piloter les actionneurs décrits dans ce livre par l'intermédiaire d'une interface web.

Concrètement, il existe deux façons de mettre votre Raspberry Pi en réseau et donc de lui donner accès à Internet.

La première est la méthode directe qui consiste à transformer le Raspberry Pi en serveur web. Il peut alors héberger une interface web à laquelle vous pouvez vous connecter et avec laquelle vous pouvez interagir à partir de n'importe quel navigateur. Par exemple, vous pouvez cliquer sur un bouton sur la page web hébergée par le Raspberry Pi pour activer une sortie GPIO. Nous utiliserons cette méthode dans le projet « Un interrupteur web Raspberry Pi ».

La seconde méthode consiste à faire communiquer le Raspberry Pi avec un service web qui agit comme intermédiaire et permet de faire transiter les messages entre les objets et les gens sur Internet. Par exemple, dans le projet « Pepe la marionnette annonce l'arrivée de nouveaux messages », le service web IFTTT (*If This Then That*, Si Ceci Alors Cela) surveille votre compte Twitter et fait danser Pepe la marionnette à chaque nouveau tweet contenant le hashtag *#dancepepe*.

Si vous ne voulez pas utiliser Twitter, vous pouvez faire surveiller votre messagerie, Facebook ou tout ce qui peut être pris en charge par le service If This Then That.

Raspberry Pi et Bottle

Bottle est un framework de serveur web simple d'emploi entièrement écrit en Python. C'est un excellent moyen de créer des applications simples de serveur web pour un Raspberry Pi.

Pour installer Bottle, saisissez les commandes suivantes :

```
$ sudo apt-get update
$ sudo apt-get install python-bottle
```

L'étape suivante consiste à créer un programme Python qui utilise Bottle pour créer un serveur web minimal auquel vous pouvez vous connecter depuis un navigateur situé n'importe où sur votre réseau. Saisissez la commande suivante pour créer un nouveau fichier Python :

```
$ nano test_bottle.py
```

Ensuite, ajouter le texte suivant au fichier :

```
from bottle import route, run

@route('/')
def index():
    return '<h1>Hello World</h1>'

run(host='0.0.0.0', port=80)
```

Le marqueur `@route` placé avant la fonction `index` indique qu'`index` est chargé de générer le code HTML qui sera renvoyé à n'importe quel navigateur qui accède à la racine du serveur web (le site web proprement dit et non une page particulière). Dans ce cas, le texte « Hello World » mis en forme comme un titre de premier niveau s'affiche. Pour vérifier que le programme fonctionne, exécutez la commande suivante :

```
$ sudo python test_bottle.py
Bottle server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:80/
Hit Ctrl-C to quit.
```

Ensuite, ouvrez un navigateur, soit sur le Raspberry Pi, soit sur un autre ordinateur de votre réseau, et utilisez l'adresse IP de votre Raspberry Pi comme URL (figure 16-1). Pour connaître l'adresse IP de votre Raspberry Pi, reportez-vous à l'encadré « Obtenir l'adresse IP d'un Raspberry Pi » au chapitre 3.

Figure 16-1. *« Hello World » dans Bottle*

Projet : un interrupteur web Raspberry Pi

Comme le serveur web Bottle n'est qu'un programme Python, il ne se limite pas à héberger du code HTML pour un navigateur. Il peut aussi utiliser la bibliothèque RPi.GPIO pour piloter les broches GPIO en réponse à des liens hypertextes ou des boutons activés par un clic sur une interface web qu'il héberge (figure 16-2).

Vous pouvez utiliser ce dispositif pour allumer ou éteindre des appareils électriques branchés sur un PowerSwitch Tail, comme décrit dans le projet « Interrupteur à minuterie Raspberry Pi » au chapitre 13.

Figure 16-2. *Une interface web pour le projet d'interrupteur Internet*

Matériel

Ce projet crée une interface web qui permettra de passer la broche GPIO 18 à l'état haut ou à l'état bas à partir d'un navigateur. À vous de choisir ce que vous voulez connecter à la broche 18. Si vous décidez de favoriser la simplicité, vous pouvez n'y connecter qu'une LED. Dans ce cas,

utilisez les composants matériels de l'expérience « Pilotage d'une LED », décrite au chapitre 4. Ou bien, vous pouvez piloter un PowerSwitch Tail et un appareil électrique, comme décrit dans le projet « Interrupteur à minuterie Raspberry Pi » au chapitre 13.

Programme

Bottle sépare le code HTML qui sera transmis au navigateur du programme Python qui contrôle l'ensemble. Ce mécanisme se base sur des *templates* (modèles). Ce projet n'utilise que deux fichiers que vous trouverez dans `projects/pr_web_switch`.

Le fichier `home.tpl` contient le code HTML de l'interface web :

```
<html>
<body>

<h1>Web Switch</h1>

<a href="/on">ON</a>

<a href="/off">OFF</a>

</body>
</html>
```

Une fois affiché dans un navigateur, ce code HTML ressemble à la figure 16-2. Les lignes clés sont délimitées par les deux balises `a` des liens hypertextes. L'attribut `href` des balises `a` contient l'adresse web qui sera affichée en cas de clic sur ON ou OFF. Si vous maîtrisez le langage HTML et les styles CSS, vous pourrez enjoliver ces liens hypertextes en leur donnant l'apparence de vrais boutons.

Dans le cas du lien hypertexte ON, une requête web se terminant par `/on` est transmise à l'adresse IP du Raspberry Pi. Cette requête est alors traitée dans le code Python présenté ci-dessous :

```
from bottle import route, run, template, request
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)      ❶
CONTROL_PIN = 18
GPIO.setup(CONTROL_PIN, GPIO.OUT)

@route('/')                 ❷
def index():
    return template('home.tpl')

@route('/on')               ❸
def index():
    GPIO.output(CONTROL_PIN, 1)
    return template('home.tpl')

@route('/off')              ❹
def index():
```

```
    GPIO.output(CONTROL_PIN, 0)
    return template('home.tpl')


try:
    run(host='0.0.0.0', port=80)    ❺
finally:
    print('Nettoyage GPIO')
    GPIO.cleanup()
```

❶ Définit la broche 18 comme broche de commande de sortie.

❷ Affiche simplement le contenu du template home.tpl lorsqu'un navigateur accède à la racine de votre serveur web.

❸ Il s'agit du code de traitement utilisé lorsque l'URL se termine par `on` (auquel cas la broche de commande passe à l'état haut avant d'afficher à nouveau la page d'accueil).

❹ Code de traitement du lien hypertexte OFF.

❺ Démarre l'exécution du serveur web sur le port 80 (port par défaut des pages web).

Utilisation de l'interrupteur web

Pour démarrer l'exécution du serveur web, saisissez la commande suivante :

```
$ sudo python web_switch.py
```

Ensuite, ouvrez un onglet dans le navigateur en saisissant l'adresse IP de votre Raspberry Pi qui s'affiche comme illustré à la figure 16-2.

Lorsque vous cliquez sur les liens hypertextes ON et OFF, l'objet connecté à la broche GPIO 18 doit s'allumer et s'éteindre.

Arduino et les réseaux

Avec son interface réseau intégrée et la possibilité d'acheter des modules Wi-Fi USB bon marché, le Raspberry Pi est bien plus adapté aux projets IoT qu'un Arduino Uno.

Il existe des shields WiFi pour l'Arduino, mais ils ne sont pas bon marché. D'autres modèles de cartes Arduino, comme l'Arduino Yun, intègrent le Wi-Fi, mais elles sont plus chères et pas vraiment simples d'emploi.

Si vous voulez utiliser un dispositif Wi-Fi ressemblant à une carte Arduino dans vos projets IoT, je vous recommande d'utiliser la carte Photon de Particle.io (figure 16-3).

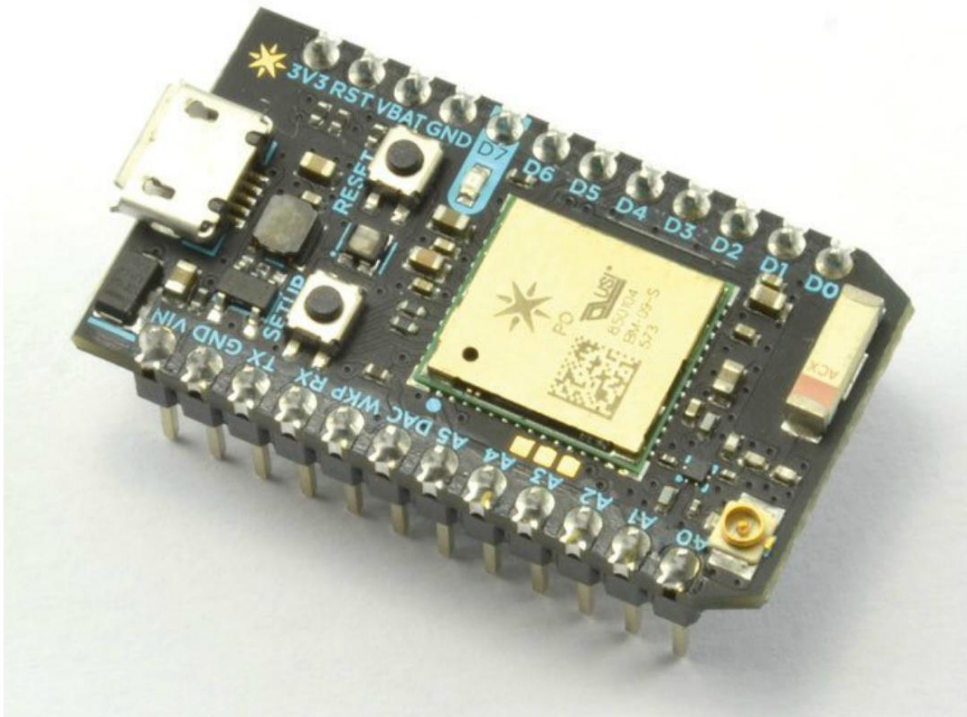


Figure 16-3. *Le Photon*

Le Photon s'inspire de l'Arduino Nano, une version de la carte Arduino, mais il intègre un module Wi-Fi. Il s'agit d'un véritable composant basé sur le cloud : vous pouvez communiquer avec lui et même y installer des logiciels via Internet. Par conséquent, vous pouvez intégrer votre Photon à un projet et le programmer sans y avoir physiquement accès. Il suffit de l'alimenter et de le connecter à votre réseau Wi-Fi.

Le langage de programmation du Photon se base aussi sur Arduino C, mais au lieu d'utiliser l'IDE Arduino standard, il est programmé depuis un IDE basé sur le web. Vous trouverez plus d'informations sur le Photon sur particle.io.

L'ESP8266 est un autre exemple de carte sur le modèle de l'Arduino qui est utilisé dans les projets IoT. Ce module (figure 16-4) extrêmement bon marché peut être configuré pour être programmé depuis l'IDE Arduino, comme une carte Arduino, ou il peut être connecté à un Arduino pour doter la carte d'une connexion Wi-Fi bon marché.

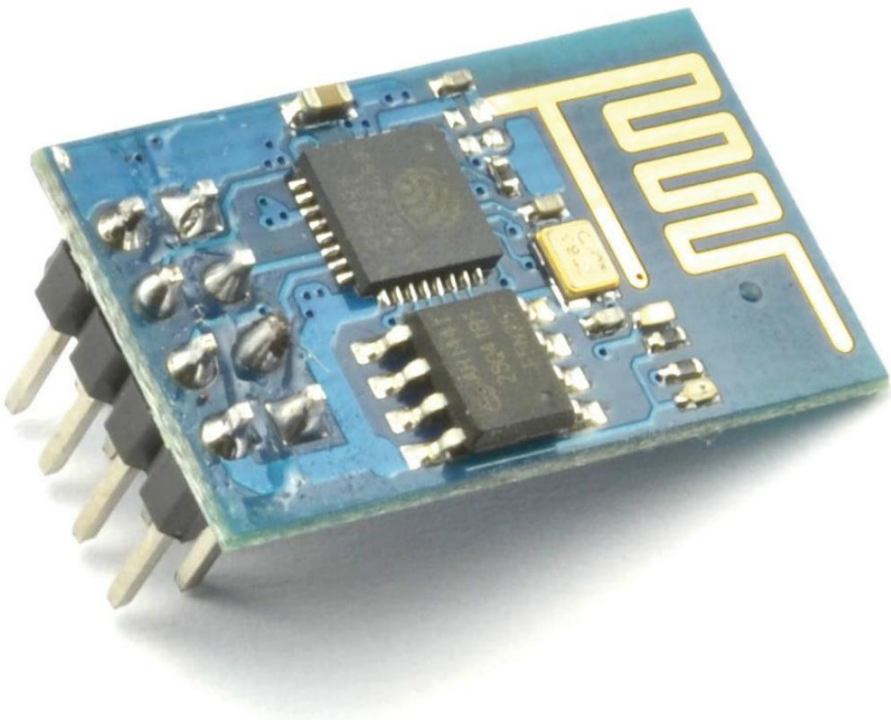


Figure 16-4. *Un module ESP8266*

L'utilisation de ce module a tendance à se simplifier, mais (à l'époque où j'écris ces lignes) sa configuration demeure assez fastidieuse. Vous trouverez plus d'informations sur ce composant, ainsi que des tutoriels, sur Internet.

Projet : Pepe la marionnette annonce l'arrivée de nouveaux messages

Nous allons compléter le projet « Pepe la marionnette qui parle » au chapitre 15, afin que Pepe réagisse aux tweets contenant le hashtag *#dancepepe* : la marionnette exécutera quelques pas de danse et émettra un son. Ce projet utilise les mêmes composants que le projet d'origine, sans le capteur IRP. Seul le logiciel change. La figure 16-5 présente le montage réalisé pour le projet « Pepe la marionnette qui parle » sans le capteur IRP.

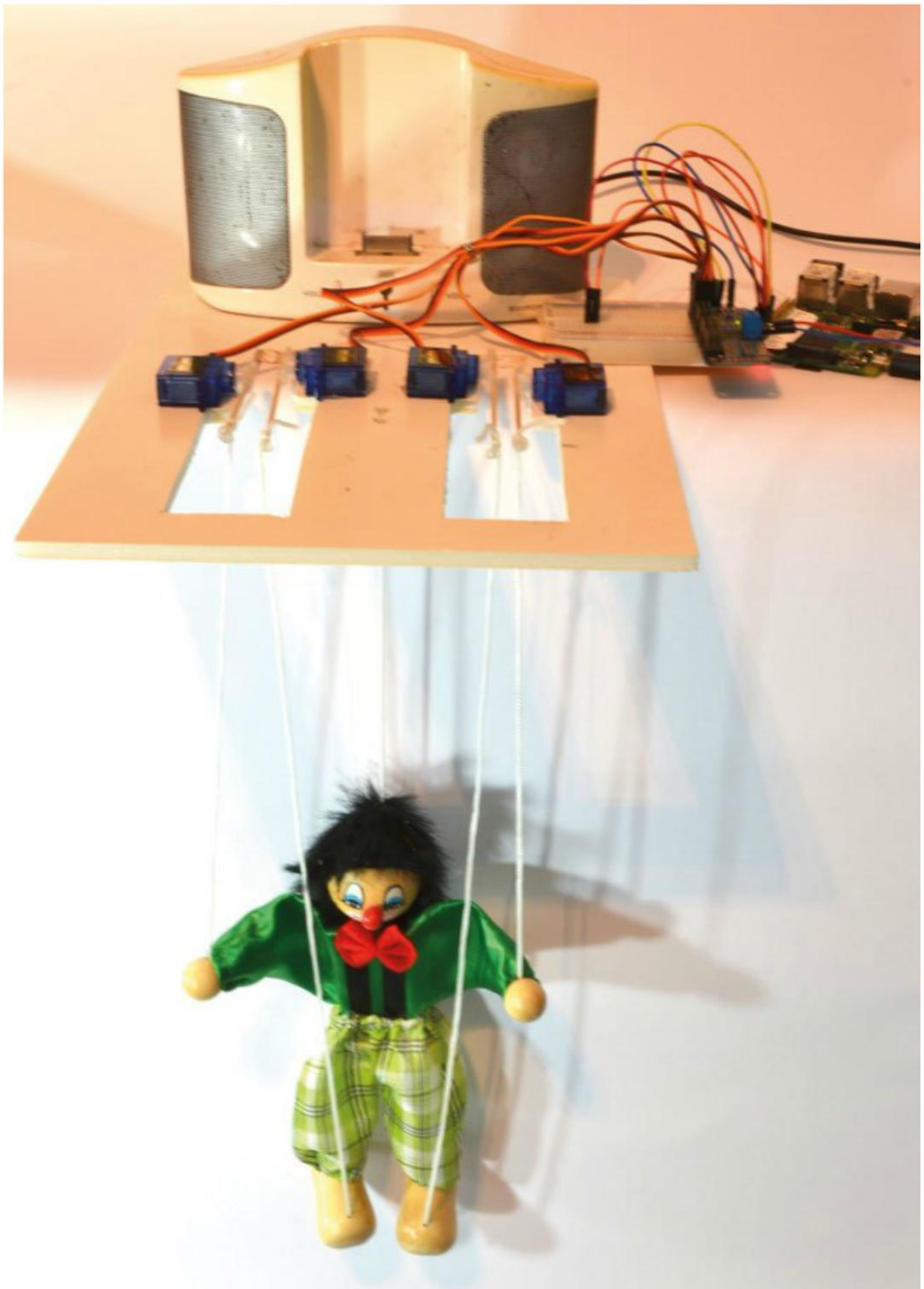


Figure 18-5. *Pepe la marionnette est prête à répondre à Twitter*

Connexion de Pepe à Internet

Deux étapes sont nécessaires pour que Pepe réagisse aux tweets. La première consiste à permettre à Pepe de réagir aux requêtes venant du web afin de déclencher son pas de danse depuis un navigateur web. La seconde étape utilise le service web IFTTT *(If This Then That)* pour détecter la réception d'un message contenant le hashtag *#dancepepe* sur Twitter et envoyer une requête web afin d'établir le lien avec la première étape.

Vous utiliserez le service web dweet.io pour permettre à Pepe de réagir aux événements web. Ce service gratuit (limité à un certain nombre de messages pouvant être envoyés par mois) se décrit comme étant l'équivalent de Twitter pour l'IoT. Il n'est pas nécessaire de s'identifier et il est très simple d'emploi. Il dispose aussi d'une bibliothèque Python, ce qui facilitera son intégration avec le programme de Pepe.

Pour installer dweepy, la bibliothèque Python pour dweet.io, exécutez les commandes suivantes :

```
$ git clone git://github.com/paddycarey/dweepy.git
$ cd dweepy
$ sudo python setup.py install
```

L'utilisation de cette bibliothèque avec Python 2 et SSL pose un petit problème qui est facile à résoudre à l'aide des commandes suivantes pour changer la version des requêtes HTTP utilisées par Python 2 :

```
$ sudo apt-get install python-pip
$ sudo pip install requests==2.5.3
```

Vous devriez maintenant pouvoir exécuter le programme du projet (`puppet_web.py`), disponible dans `python/projects/puppet_web`, à l'aide de la commande suivante :

```
$ sudo python puppet_web.py
```

Vous pouvez tester la progression de ce projet à l'aide d'un navigateur web en vous rendant à l'adresse *https://dweet.io/dweet/for/pepe_the_puppet* (figure 16-6).

Figure 16-6. *Manipulation de la marionnette à partir d'un navigateur web*

Dès que vous accédez à cette URL, Pepe doit prendre vie.

Le programme présente de nombreux points communs avec celui du projet « Pepe la marionnette qui parle » au chapitre 15. Je vous invite à vous reporter à ce projet pour la partie principale du code :

```
from Adafruit_PWM_Servo_Driver import PWM
from pygame import mixer
import time
import dweepy      ❶

pwm = PWM(0x40)
mixer.init()
mixer.music.load("pepe_1.wav")

dweet_key = 'pepe_the_puppet'      ❷

servoMin = 150 # Longueur d'impulsion mini sur 4096
servoMax = 600 # Longueur d'impulsion maxi sur 4096

dance = [
  #lh lf rf rh
  [130, 20, 20, 130],
  [30, 160, 160, 30],
  [90, 90, 90, 90]
]

delay = 0.2

def map(value, from_low, from_high, to_low, to_high):
  from_range = from_high - from_low
  to_range = to_high - to_low
  scale_factor = float(from_range) / float(to_range)
  return to_low + (value / scale_factor)


def set_angle(channel, angle):
  pulse = int(map(angle, 0, 180, servoMin, servoMax))
  pwm.setPWM(channel, 0, pulse)

def dance_step(step):
  set_angle(0, step[0])
  set_angle(1, step[1])
  set_angle(2, step[2])
  set_angle(3, step[3])

def dance_pupet():
    for i in range(1, 10):
        for step in dance:
            dance_step(step)
            time.sleep(delay)

pwm.setPWMFreq(60)
```

```
while True:
  try:          ❸
      for dweet in dweepy.listen_for_dweets_from(dweet_key):  ❹
          print(«Dance Pepe! Dance!»)
          mixer.music.play()
          dance_pupet()
  except Exception:
    pass
```

❶ Importe la bibliothèque dweepy.

❷ dweet utilise cette valeur en tant que clé pour les dweets qui vous intéressent. Si vous conservez cette clé telle quelle, d'autres lecteurs de ce livre qui réalisent ce projet pourront déclencher votre marionnette (et vice versa), ce qui est plutôt amusant. Sinon, vous pouvez choisir une autre valeur pour cette clé.

❸ Le code à l'intérieur de la boucle principale est contenu dans la fonction de traitement d'erreur `try/except`, car la connexion web surveillée par le programme se désactive au bout d'un certain délai, ce qui cause une exception. Le code `try/except` masque ce comportement et permet au programme de réessayer après ce type d'erreurs.

❹ dweet.io vous permet d'attendre l'arrivée de nouveaux dweets pour votre clé et d'exécuter l'action à l'arrivée des messages.

IFTTT (If This Then That)

IFTTT est un service web qui vous permet de configurer des déclencheurs d'actions. Par exemple, vous pourriez créer une recette IFTTT qui vous envoie un message électronique (l'action) dès que l'on vous cite sur Twitter (le déclencheur).

Dans ce projet, nous nous servirons d'IFTTT pour surveiller l'apparition du hashtag *#dancepepe* sur Twitter. Dès qu'il est utilisé, une requête web est envoyée afin de déclencher le pas de danse de Pepe et la lecture de son clip audio.

Voici comment procéder pour configurer IFTTT.

Étape 1 : création d'une nouvelle recette

Inscrivez-vous au service IFTTT (c'est gratuit). Ensuite cliquez sur le bouton Create a Recipe. La page illustrée à la figure 16-7 apparaît.

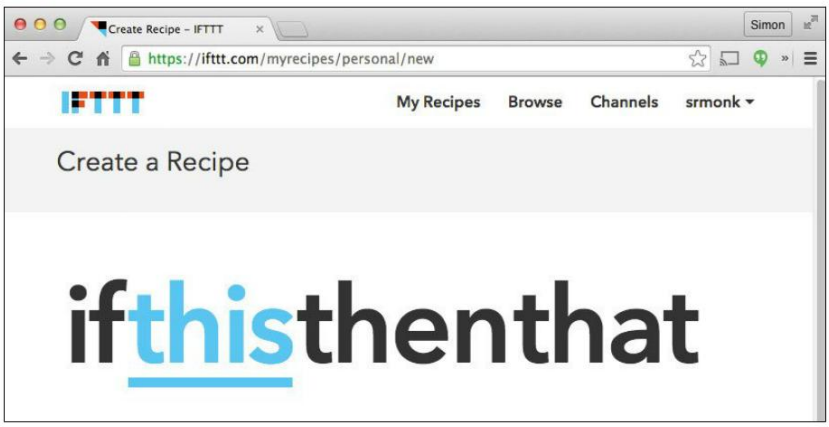


Figure 16-7. *Création d'une nouvelle recette dans IFTTT*

Étape 2 : Définition du déclencheur

Cliquez sur le lien hypertexte this, puis retrouvez la chaîne Twitter dans la liste des chaînes. Dans la chaîne Twitter, retrouvez le déclencheur New tweet from search, puis saisissez `#dancepepe` dans le champ Search for (figure 16-8).

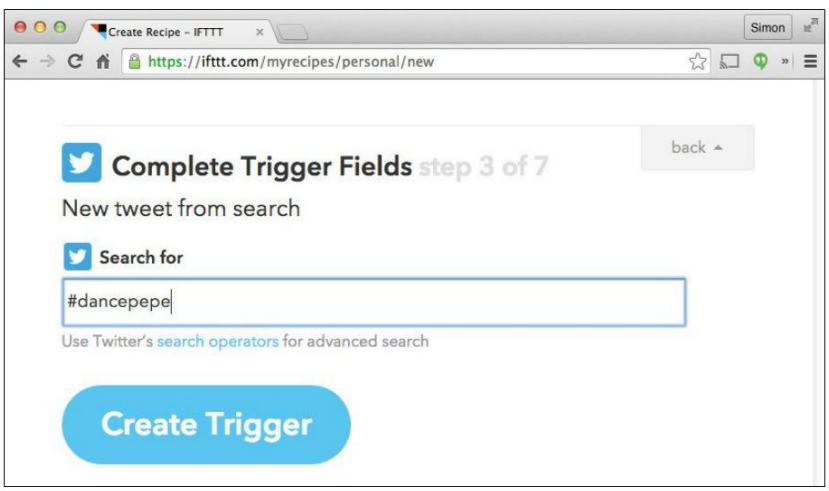


Figure 16-8. *Configuration du déclencheur Twitter*

Étape 3 : ajout de l'action de requête web

Lorsque le déclencheur a été défini, l'écran if this then that réapparaît. Vous devez maintenant définir l'action en cliquant sur le lien hypertexte that, puis recherchez la chaîne d'action « Maker ». Sélectionnez la seule action proposée (Make a web request), puis renseignez le formulaire illustré à la figure 16-9.

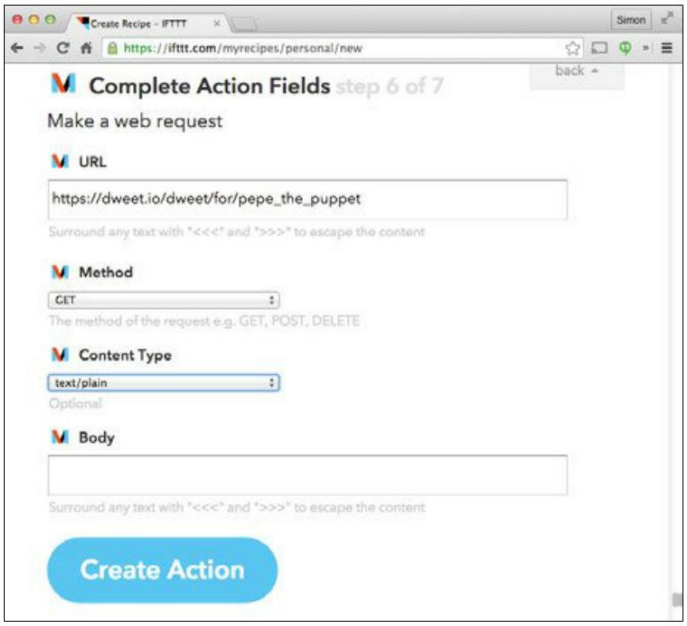


Figure 16-9. *Le formulaire d'action à compléter*

Étape 4 : finalisation de la recette

Cliquez sur le bouton Create Action, puis confirmez la création de la recette en cliquant sur Create Recipe (figure 16-10).

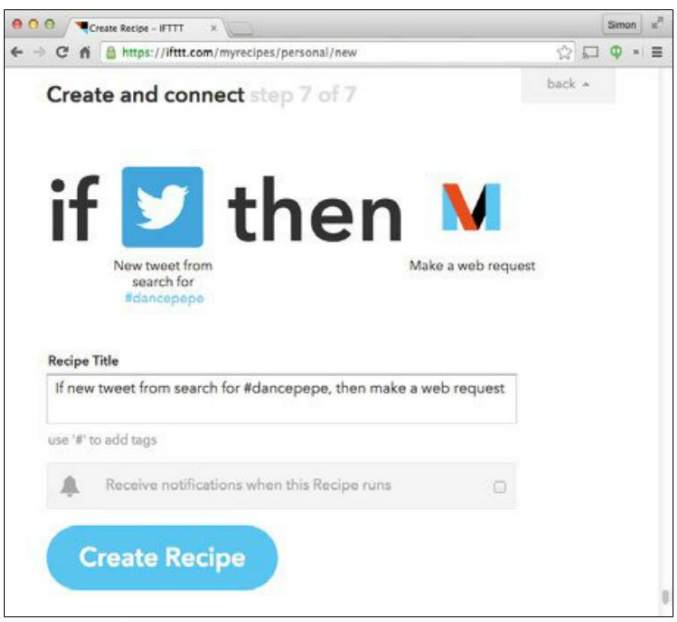


Figure 16-10. *Création de la recette*

Une fois terminée, la recette s'applique automatiquement. Peut-être devrez-vous vous identifier sur la chaîne Twitter à un certain stade du processus.

Utilisation du projet

Le projet est prêt. Vous pouvez le tester en tweetant le hashtag *#dancepepe*. Patientez le temps qu'IFTTT détecte le tweet. Cela peut prendre une ou deux minutes.

Si rien ne se produit, jetez un œil au journal IFTTT pour y retrouver votre recette. Il s'agit de l'icône qui ressemble à une liste à puces sur la page de la recette. Vous saurez alors si la recette a été déclenchée et s'il y a eu des erreurs.

Si vous voulez enregistrer votre propre message qui sera lu par Pepe, reportez-vous à la section « Expérimentation Arduino » du chapitre 15 page 290.

Prenez le temps de parcourir les autres déclencheurs disponibles sur IFTTT. Vous en découvrirez beaucoup d'autres qui pourraient être associés à Pepe la marionnette !

Résumé

Dans ce chapitre, vous avez appris à exécuter un serveur web sur un Raspberry Pi, ainsi qu'à utiliser des services web, comme IFTTT et dweepy, pour créer des projets IoT amusants.

J'espère que ce livre vous aura été utile et que vous y avez trouvé des idées de projets que vous allez pouvoir réaliser sans tarder.

Fournisseurs et composants

Il n'est pas toujours évident de dénicher les composants nécessaires à vos projets. Pour vous y aider, vous trouverez dans cette annexe une liste des principaux fournisseurs, des exemples de références des composants utilisés dans l'ouvrage, ainsi que d'autres informations pratiques comme les brochages de certains composants.

Principaux fournisseurs

Il existe maintenant de nombreux revendeurs de composants électroniques qui peuvent répondre aux besoins des électroniciens amateurs. Les plus populaires sont indiqués dans le tableau A-1.

Vous noterez que la plupart des fournisseurs spécialisés de cette liste vendent l'Arduino Uno R3 ainsi que le Raspberry PI 2 modèle B, qui sont recommandés dans ce livre.

La majorité de ces fournisseurs sont des revendeurs en ligne, mais Selectronic, Go Tronic et Les Cyclades possèdent également des magasins à Paris ou en province.

Il est également possible d'acheter des composants électroniques sur le site eBay (*http://ebay.fr*), ou encore sur le site Aliexpress (*http://aliexpress.com*) qui regroupe un ensemble de revendeurs asiatiques. Attention pour ce dernier : si les prix sont attractifs, le délai de livraison atteint souvent plusieurs semaines.

Pour débuter, vous pouvez choisir d'acheter le kit d'expérimentation Arduino d'Adafruit (référence produit ID 170) ou le kit Sparkfun Arduino Inventor's kit (KIT-11227) qui offrent tous deux une sélection de composants de base accompagnés d'une carte Arduino. Autre possibilité, le kit de composants électroniques préparé par MonkMakes, disponible chez Amazon UK (*https://www.amazon.co.uk/Monk-Makes-Ltd-SKU00050-Component/dp/B01EREYOEW*), réunit la plupart des résistances, condensateurs, transistors et LED utilisés dans ce livre.

Tableau A-1. *Principaux fournisseurs*

Fournisseur	Site web	Remarques
Selectronic	*http://www.selectronic.fr/*	Grand choix de composants et modules, magasins à Paris et Lille
Conrad	*http://www.conrad.fr/ce/*	Grand choix de composants et modules
Les Cyclades Électronique	*https://www.cyclades-elec.fr/*	Grand choix de composants. Magasin à Paris
Digikey	*http://www.digikey.fr/*	Large gamme de composants
Amazon	*https://www.amazon.fr/*	Composants et modules
Adafruit	*http://www.adafruit.com/*	Grand choix de modules, expédition vers la France. Plusieurs distributeurs en France.
SparkFun	*http://www.sparkfun.com*	Grand choix de modules, expédition vers la France
Lextronic	*http://www.lextronic.fr/*	Composants et robotique. Distributeur français des produits Sparkfun.
Go Tronic	*http://www.gotronic.fr/*	Composants et robotique. Magasin à Blagny (Ardennes)
Robotshop	*http://www.robotshop.com/eu/fr/*	Tout ce qui concerne la robotique

Résistances et condensateurs

Les résistances et les condensateurs sont très peu coûteux, mais il est souvent nécessaire de les acheter en quantité minimale chez certains revendeurs. C'est pourquoi il est préférable de vous procurer un assortiment de composants pour débuter, comme le kit de 610 résistances ¼ W Selectronic référence 7654-2.

Tableau A-2. *Références des résistances et condensateurs*

Description	Source
Résistance 10 Ω ¼ W	Kit Selectronic, référence 7654-2
Résistance 100 Ω ¼ W	Kit Selectronic, référence 7654-2
Résistance 150 Ω ¼ W	Kit Selectronic, référence 7654-2
Résistance 270 Ω ¼ W	Kit Selectronic, référence 7654-2
Résistance 470 Ω ¼ W	Kit Selectronic, référence 7654-2
Résistance 1 kΩ ¼ W	Kit Selectronic, référence 7654-2
Résistance 4,7 kΩ ¼ W	Kit Selectronic, référence 7654-2
Potentiomètre linéaire 10 kΩ	Selectronic, référence 5302
Photorésistance	Selectronic, référence 7608
Condensateur de 100 nF	Selectronic, référence 1485
Condensateur de 100 µF	Selectronic, référence 6735

Semi-conducteurs

Il est facile de se procurer des composants ayant une référence simple comme « 2N3904 », mais lorsqu'il s'agit de LED, il peut être fastidieux de les acheter individuellement. Dans ce cas, il vaut mieux en acheter un assortiment sur eBay ou Amazon, ou sinon un lot faisant partie d'un kit de démarrage.

Tableau A-3. *Références des semi-conducteurs*

Description	Source
Transistor 2N3904	Selectronic, référence 3383 Conrad, code produit 163350 – 62
Transistor Darlington MPSA14	Selectronic, référence HD85-623 Digikey, référence MPSA14-ND
Transistor MOSFET 2N7000	Conrad, code produit 151034 – 62
Transistor Darlington TIP120	Conrad, code produit 150872 – 62 Digikey, référence TIP120-ND
Transistor MOSFET Canal-N FQP30N06L	Conrad, code produit 1264199 – 62 Digikey, référence FQP30N06FS-ND
Diode 1N4001	Conrad, code produit 162213 – 62 Digikey, référence 1N4001FSCT-ND
LED rouge	Selectronic, référence 2536
LED verte	Selectronic, référence 2538
LED orange	Selectronic, référence 9652
LED RVB à cathode commune	Selectronic, référence 0567
Circuit intégré pont en H L293D	Conrad, code produit 174003 - 62 Amazon eBay
Circuit intégré pont en H L298D	Conrad, code produit 156128 – 62 Amazon eBay
ULN2803 8 x transistors Darlington	Selectronic, référence 7396 Conrad, code produit 171824 – 62
Capteur de température DS18B20	Conrad, code produit 184024 – 62 Adafruit, 374 (résistance 4,7K comprise).
Capteur de température en boîtier étanche DS18B20	Adafruit, 381 eBay

Autres fournitures

Généralement, une plaque d'essais et un assortiment de câbles flexibles de liaison sont inclus dans les kits mentionnés page 318.

Tableau A-4. *Références des autres fournitures*

Description	Source
Plaque d'essai 400 points (breadboard)	Adafruit, 64 eBay
Assortiment de câbles flexibles mâle-mâle	Adafruit, 758 eBay
Assortiment de câbles flexibles femelle-femelle	Adafruit, 266 eBay
Assortiment de fils de câblage isolés	Adafruit, 1311 eBay
Assortiment de câbles flexibles femelle-mâle	Adafruit, 826 eBay
Coupleur étanche pour 2 piles AA (3 V)	Adafruit, 770 eBay
Coupleur étanche pour 4 piles AA (6 V)	Adafruit, 830 eBay
Jack femelle 2,1 mm à vis	Adafruit, 368, Selectronic, référence 1003-5 eBay
Domino de connexion électrique	Grande surface de bricolage

Modules, moteurs et alimentations

La plupart de ces éléments peuvent être achetés sur eBay ou Amazon.

Tableau A-5. *Références des modules, moteurs et alimentations*

Description	Source
Petit moteur 6 V	Adafruit, 711
Moteur pas-à-pas bipolaire 12 V	Adafruit, 324
Moteur pas-à-pas unipolaire 5 V	Adafruit, 858
Servomoteur 9G SG92R	Adafruit, 169 eBay

Module alimentation secteur 5 V, 2 A	Adafruit, 276 (nécessite un adaptateur de prise US) Amazon eBay
Module alimentation secteur 12 V, 1 A	Adafruit, 798 (nécessite un adaptateur de prise US) Amazon eBay
Alimentation secteur 12 V, 5 A	Adafruit, 798 (nécessite un câble secteur aux normes européennes) Amazon eBay
Module contrôleur de moteur pas-à-pas « EasyDriver »	Sparkfun, ROB-12779
Module de commande de 16 servomoteurs ADA815	Adafruit, 815 Go Tronic, code 31840
Haut-parleur 8 Ω diamètre 39 mm	Adafruit, 1891 eBay
Ruban à LED adressables WS812, 30 LED par mètre	Adafruit, 1376 eBay
Afficheur OLED I2C, 128 × 64 pixels monochrome avec interface SSD1306	eBay
Module infrarouge PIR	Adafruit, 189 eBay
Module relais	eBay

Brochage du connecteur GPIO du Raspberry Pi

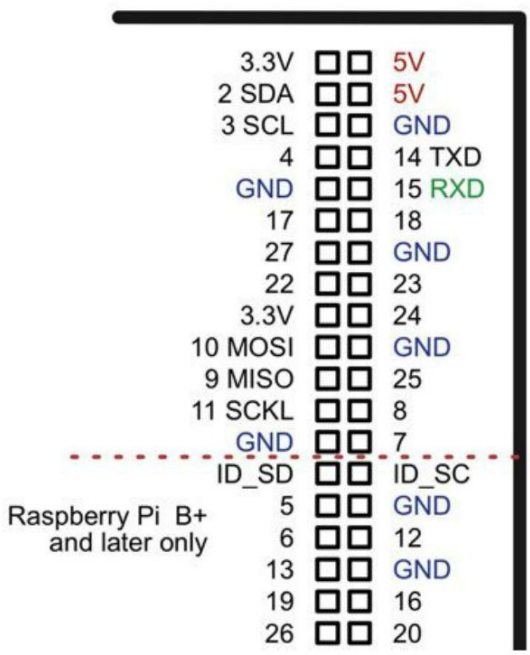


Figure B-1. *Brochage de la carte Raspberry PI*

- Les broches 2 et 3 sont utilisées par le bus I2C.
- La liaison SPI utilise les broches 9, 10 et 11.
- L'interface TTL utilise les broches 14 et 15.
- Les broches ID_SD et ID_SC sont réservées pour l'identification automatique des cartes d'extension HAT.

Brochage des composants

La fig. A-1 représente le brochage des composants utilisés dans ce livre.

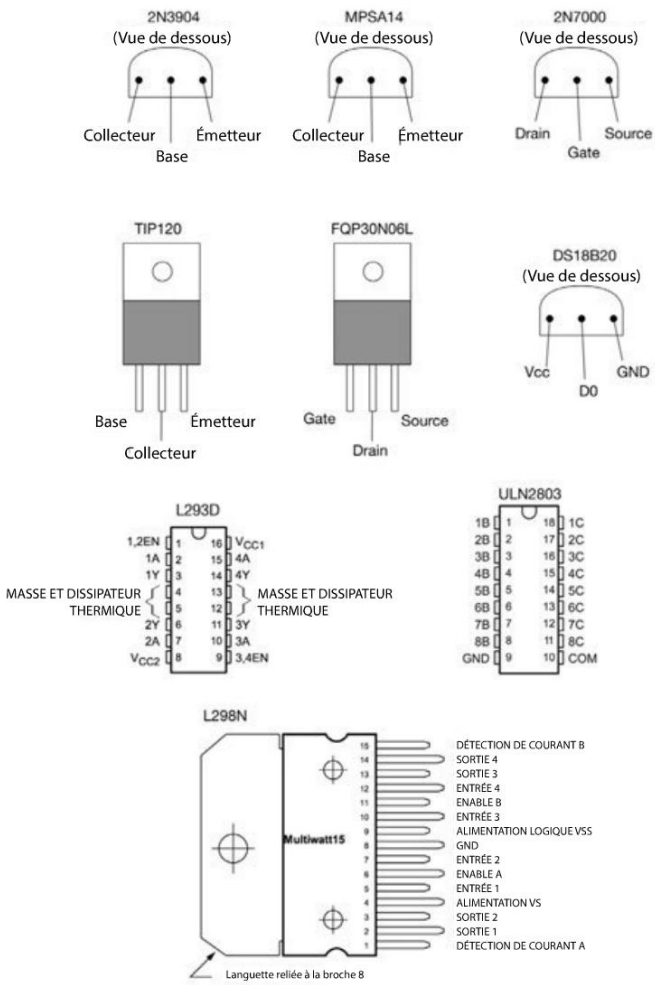


Figure A-1. *Brochage des composants*

Index

Symboles

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

U

V

W

Z