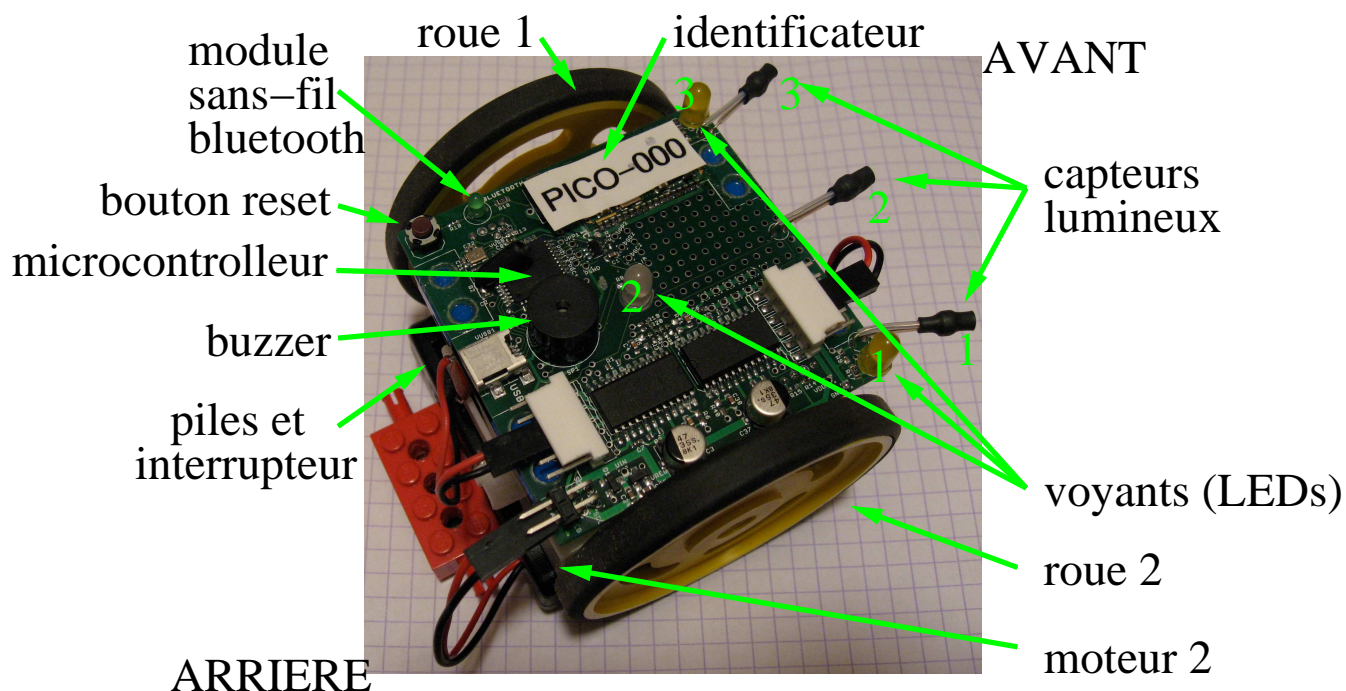


Ce travail a pour but de vous introduire au langage de programmation Scheme et de vous montrer les avantages qu'offrent les langages de programmation de haut niveau pour la réalisation de programmes complexes. Vous devez réaliser quelques programmes de contrôle de robot, allant du simple au complexe.

Le robot miniature qui est utilisé dans ce travail a été conçu au département d'informatique et de recherche opérationnelle de l'Université de Montréal. Il est composé d'un petit ordinateur programmable (microcontrôleur), 2 roues motorisées, 3 voyants lumineux "LEDs" (deux jaunes et un bicolore (vert ou rouge)), un "buzzer", 3 capteurs lumineux, et un système de communication sans-fil Bluetooth permettant de programmer le robot à distance. Voici une image du robot montrant les composantes principales :



Le robot peut être programmé en langage assembleur ou en langage Scheme. Le langage assembleur offre l'avantage de donner accès à toutes les possibilités du microcontrôleur du robot; cependant la programmation en langage assembleur est fastidieuse et il est difficile de réaliser des programmes fiables et robustes de plus de quelques dizaines ou centaines de lignes de code (en fonction du niveau d'expérience du programmeur). Les langages de plus haut niveau, tel que Scheme et Java, ont été conçus pour faciliter le développement de programmes complexes. Dans ce travail, nous utiliserons le langage Scheme et, plus spécifiquement, le compilateur PICOBIT.

PICOBIT est un compilateur pour le langage Scheme qui possède des fonctions prédéfinies pour contrôler les moteurs et capteurs du robot. Voici la liste de ces fonctions :

- (**motor**  $\langle id\text{-moteur} \rangle$   $\langle niveau\text{-de-puissance} \rangle$ )

La fonction **motor** change le sens de rotation et le niveau de puissance d'un moteur. Le paramètre  $\langle id\text{-moteur} \rangle$ , qui est soit 1 ou 2, identifie le moteur tandis que le paramètre  $\langle niveau\text{-de-puissance} \rangle$ , qui est un entier entre -100 et 100 inclusivement, indique le niveau de puissance. Un niveau de puissance 0 arrête le moteur et un niveau négatif fait tourner la roue dans le sens inverse. Nous pouvons utiliser **motor** par exemple pour définir ces deux fonctions qui font avancer ou reculer le robot à mi-vitesse :

```
(define avancer
  (lambda ()
    (motor 1 100)
    (motor 2 100)))

(define reculer
  (lambda ()
    (motor 1 -100)
    (motor 2 -100)))
```

- (**led**  $\langle id\text{-led} \rangle$   $\langle durée \rangle$   $\langle période \rangle$ )

La fonction **led** change l'état d'un voyant lumineux (LED). Le paramètre  $\langle id\text{-led} \rangle$ , qui est soit 1, 2 ou 3, identifie le voyant. Le voyant va clignoter à une cadence dont la période en millisecondes est donnée par le paramètre  $\langle période \rangle$ . Le paramètre  $\langle durée \rangle$  indique le temps pendant lequel le voyant sera illuminé. Dans le cas où  $\langle durée \rangle = \langle période \rangle$  le voyant restera illuminé sans clignotement. Dans le cas où  $\langle durée \rangle = 0$  le voyant restera éteint. Les paramètres  $\langle période \rangle$  et  $\langle durée \rangle$  doivent être entre 0 et 2559.

Par exemple l'appel (**led** 2 500 1000) va faire clignoter la LED 2 une fois par seconde (illuminé pendant 1/2 seconde et éteint pendant 1/2 seconde).

- (**led2-color**  $\langle couleur \rangle$ )

La fonction **led2-color** change la couleur émise par la LED 2. Le paramètre  $\langle couleur \rangle$  doit être le symbole **red** ou **green**.

- (**clock**)

La fonction **clock** retourne le temps qui s'est écoulé depuis le démarrage du robot, en millisecondes. Cette fonction est utile pour provoquer des actions à des moments précis dans le temps.

- (**light**  $\langle id\text{-capteur} \rangle$ )

La fonction **light** fait une lecture du capteur de lumière identifié par  $\langle id\text{-capteur} \rangle$  et retourne un nombre entier entre 0 et 1023 qui indique l'intensité de la lumière qui frappe ce capteur (grande valeur = plus lumineux).

- (**getchar**)

La fonction **getchar** lit un caractère de la console et retourne son code ASCII (un nombre entier). Cette fonction attend qu'un caractère soit tapé à la console avant de retourner un résultat.

- (`getchar-wait`  $\langle attente-max \rangle$ )

La fonction `getchar-wait` est comme `getchar` sauf que l'attente est limitée à  $\langle attente-max \rangle$  (en millisecondes). Si aucun caractère n'est reçu à temps, la fonction `getchar-wait` retourne `#f` (la valeur "faux"). Cette fonction est utile pour faire des actions en même temps qu'on traite des caractères de la console.

- (`putchar`  $\langle x \rangle$ )

La fonction `putchar` envoie le caractère  $\langle x \rangle$  sur la console.  $\langle x \rangle$  peut également être un entier, auquel cas c'est le code ASCII du caractère à afficher. Par exemple, ces deux appels à `putchar` vont afficher le caractère "Q" (notez la syntaxe particulière des caractères en Scheme) :

```
(putchar 81) ;; le code ASCII de 'Q' est 81
(putchar #\Q)
```

- (`sleep`  $\langle attente \rangle$ )

La fonction `sleep` suspend l'exécution du programme pour  $\langle attente \rangle$  millisecondes. Cela est utile pour cadencer les actions du robot à la bonne vitesse.

- (`beep`  $\langle div-freq \rangle$   $\langle durée \rangle$ )

La fonction `beep` débute un son sur le buzzer. La durée du son en millisecondes est indiquée par le paramètre  $\langle durée \rangle$ . Une horloge de 2500 Hz est utilisée pour générer la tonalité. Le paramètre  $\langle div-freq \rangle$  indique la division de fréquence qui sera effectuée. Donc un paramètre  $\langle div-freq \rangle=5$  donnera une tonalité de 500 Hz (2500 divisé par 5).

- (`sernum`)

La fonction `sernum` retourne le numéro de série du robot (par exemple 13 pour le robot PICO-013).

PICOBIT a aussi les fonctions suivantes, plus standard, qu'on a vues en classe : `+`, `-`, `*`, `quotient`, `remainder`, `modulo`, `=`, `<`, `<=`, `>`, `>=`, `min`, `max`, `abs`, `string-length`, `string-append`, `substring`, et `display`.

Cependant, vu la petitesse du microcontrôleur, certaines fonctionnalités de Scheme ne sont pas disponibles. Les calculs numériques sont seulement sur des entiers, et, de plus, ceux-ci sont limités à la plage -8388608 à 8388607 (c'est-à-dire des nombres binaires de 24 bits). Il y a peu de mémoire RAM (seulement 128 cellules de 4 octets sont accessibles) et, si on ne fait pas attention, il est possible de l'épuiser assez facilement (par exemple avec des appels récursifs profonds).

PICOBIT vérifie la validité des opérations avant de les effectuer. Ainsi, si vous passez en paramètre à une fonction une valeur qui n'est pas permise, l'exécution du programme sera terminée (par exemple : `(+ 1 "allo")` ou un débordement de mémoire). Dans ces cas d'erreur, le robot arrête les moteurs et émet une tonalité grave.

Avec si peu d'information sur la nature de l'erreur, la mise au point des programmes (le "débogage") serait difficile. Pour réduire le problème, on peut simuler l'exécution du programme sur l'ordinateur de développement pour s'assurer qu'il fonctionne correctement avant de le télécharger au robot. Cela se fait avec un simulateur qui est intégré à l'environnement de développement.

Les fichiers requis pour ce travail ont déjà été copiés dans votre compte. Ils sont dans le sous-répertoire "lang" de votre répertoire principal.

# 1 Exercice 1 : Tourner en rond

Le premier exercice consiste à faire tourner le robot en rond. L'idée, c'est d'activer seulement un des deux moteurs de sorte que le robot pivote sur la roue immobile.

## Exercice 1-a

Après vous être branché sur l'ordinateur vous devez démarrer l'environnement de développement. Cela peut se faire en double-cliquant l'icone "**start-dev**". Cela ouvre une fenêtre "shell" qui vous permet d'exécuter des commandes textuelles et aussi un éditeur texte (SciTE) qui vous permet d'écrire, modifier et exécuter des programmes.

Dans SciTE, dans le menu "File" ouvrez le fichier "**ex1-a.scm**" du sous-répertoire "**pico\prog**". Le contenu du fichier "**ex1-a.scm**" est le suivant :

```
(motor 1 50)
(sleep 3000)
(motor 1 0)
```

L'appel de fonction "**(motor 1 50)**" fera avancer la roue de gauche à 50% de la puissance maximale. Après une attente de 3 secondes (3000 millisecondes), la roue gauche sera immobilisée grâce à l'appel de fonction "**(motor 1 0)**".

Pour exécuter le programme, vous devez utiliser la touche F7. Cette touche enregistre automatiquement le fichier si vous avez fait des modifications, et démarre le compilateur PICOBIT pour produire le code hexadécimal du programme à télécharger au robot (le nom du fichier se termine avec "**.hex**"). La partie inférieure de la fenêtre vous indique le déroulement de la compilation. Si le programme contient une erreur vous devez le corriger et faire la compilation à nouveau.

Si la compilation termine sans erreur, une fenêtre s'ouvre immédiatement pour vous montrer la **simulation** de l'exécution du programme sur le robot. Les traces à l'écran vous indiquent les actions que le robot prendrait (changement d'état de la LED, action des moteurs, et affichages/lectures à la console). Une trace vous indique aussi la quantité de mémoire nécessaire à l'exécution de votre programme. S'il y a une erreur d'exécution, le simulateur affiche un message explicatif et termine. Notez que vous pouvez arrêter l'exécution du simulateur à tout moment en fermant la fenêtre.

Puisque le programme fonctionne tel que prévu (à moins que vous ne l'ayez changé!), il terminera sans erreur. Dans la fenêtre de shell entrez la commande suivante pour programmer votre robot :

```
robot pico\prog\ex1-a.hex
```

Le robot devrait tourner pendant 5 secondes. Maintenant, utilisez la touche F1 pour arrêter l'exécution du programme sur le robot, puis la touche F4 pour quitter le programme "**robot**".

## Exercice 1-b

Le programme fonctionne mais n'est pas très lisible. Pour qu'il soit plus facile à comprendre pour un autre programmeur nous pouvons ajouter des définitions pour qu'on puisse utiliser les noms "d" et "g" pour identifier le moteur droit et le moteur gauche, et une définition de la puissance envoyée aux moteurs.

Ouvrez le fichier "**ex1-b.scm**" qui contient le programme amélioré, qui cette fois-ci tourne dans l'autre direction :

```

(define g 1)
(define d 2)
(define puissance 50)

(motor d puissance)
(sleep 3000)
(motor d 0)

```

Exécutez ce nouveau programme pour vous assurer qu'il fonctionne.

### Exercice 1-c

Nous pouvons ajouter des définitions de fonction et des commentaires pour que les actions du programme soient plus compréhensibles.

Ouvrez le fichier “ex1-c.scm” qui contient le programme amélioré, qui cette fois-ci pivote sur place :

```

(define g 1) ;; index du moteur de gauche
(define d 2) ;; index du moteur de droite
(define puissance 50) ;; puissance envoyee aux moteurs

;; (avancer m) fait avancer le moteur m.

(define avancer
  (lambda (m)
    (motor m puissance)))

;; (reculer m) fait reculer le moteur m.

(define reculer
  (lambda (m)
    (motor m (- puissance))))

;; (stop m) arrete le mouvement du moteur m.

(define stop
  (lambda (m)
    (motor m 0)))

;; Debut des actions:

(avancer g)
(reculer d)
(sleep 3000) ;; attendre 3 secondes
(stop g)
(stop d)

```

Ce programme définit les fonctions “avancer”, “reculer”, et “stop”. Chaque fonction reçoit un paramètre qui est l'index du moteur qui est concerné. Donc l'appel de fonction “(avancer g)” fera avancer la roue de gauche.

Exécutez ce nouveau programme pour vous assurer qu'il fonctionne.

## 2 Exercice 2 : En rond et plus

Maintenant, vous devez modifier le fichier “`ex2.scm`” (dont le contenu est identique au fichier “`ex1-c.scm`”) pour changer le comportement du robot. Pour chacun des cas suivants, essayez votre programme (avec la touche F7 et la commande “`robot`”) pour vous assurer que le programme fonctionne correctement :

- Faites pivoter le robot dans l’autre direction.
- Faites pivoter le robot plus rapidement.
- Faites pivoter le robot plus lentement. Quelle est la vitesse la plus lente possible?
- Faites tourner les roues dans le même sens et à des vitesses différentes. Que se passe-t-il?
- Faites tourner les roues en sens inverse et à des vitesses différentes. Que se passe-t-il?

## 3 Exercice 3 : La LED clignotante

Cet exercice consiste à faire clignoter la LED 2 en alternant entre le vert et le rouge. Ouvrez le programme “`ex3.scm`”. Ce programme contient la définition de la fonction `clignoter` et un appel à cette fonction :

```
(define clignoter
  (lambda ()
    (led2-color 'green) ;; allumer la LED en vert
    (sleep 250)         ;; attendre 1/4 seconde (250 millisecondes)
    (led2-color 'red)   ;; allumer la LED en rouge
    (sleep 250)         ;; attendre 1/4 seconde
    (clignoter)))

(led 2 100 100) ;; allumer la LED 2
(clignoter)
```

Remarquez que la fonction `clignoter` est récursive. Elle se termine par un appel à elle-même pour effectuer le prochain cycle de clignotement (c’est un *appel terminal*). Les appels terminaux sont essentiels pour exécuter des commandes répétitivement. Remarquez aussi l’utilisation de la fonction `sleep` pour fixer la fréquence de clignotement.

Essayez le programme pour vous assurer que le programme fonctionne correctement. Ensuite, modifiez-le pour obtenir les comportements suivants :

- Faites un clignotement différent, par exemple : vert, rouge, noir, vert, rouge, noir, etc).
- Faites clignoter la LED plus lentement.
- Faites clignoter les 3 LEDs en séquence, c’est-à-dire LED 1, ensuite LED 2, ensuite LED 3, puis on recommence.

- d. Faites clignoter la LED plus rapidement. Si la fréquence est très rapide (aucun “sleep”) quelle est la couleur que la LED semble émettre?
- e. Faites clignoter les 3 LEDs en séquence, c’est-à-dire LED 1, ensuite LED 2, ensuite LED 3, puis on recommence. Ajoutez un effet d’accélération.

## 4 Exercice 4 : Le compteur de présence

Les capteurs de lumière permettent de réaliser un petit système de compteur de présence (utilisé par exemple dans les musées pour compter le nombre de personnes qui ont visité le musée). L’idée c’est d’utiliser un seul des capteurs, disons le capteur 2. Il faut illuminer le capteur de lumière avec une lampe de poche de manière à détecter lorsqu’un objet (personne, animal, autre robot, etc) passe entre la lampe de poche et le capteur de lumière. Il faut compter le nombre de fois que la lecture du capteur de lumière passe en dessous d’un certain seuil (qu’il faut déterminer expérimentalement car il peut varier d’un robot à l’autre). La valeur du compteur est affichée à la console à chaque fois qu’elle change. Pour éviter les fausses lectures, il faut que le seuil dépende de l’état du capteur (faisceau lumineux présent ou absent).

Le programme “ex4.scm” qui vous est fourni est le suivant :

```
(define seuil1 700)
(define seuil2 1000)

(define senseur
  (lambda ()
    (light 2)))

(define faisceau-absent
  (lambda (n)
    (if (< (senseur) seuil2) ;; faisceau toujours absent?
        (faisceau-absent n)
        (faisceau-present n))))

(define faisceau-present
  (lambda (n)
    (if (> (senseur) seuil1) ;; faisceau toujours présent?
        (faisceau-present n)
        (let ((x (+ n 1)))
          (display x) ;; afficher le compteur
          (display "\n")
          (faisceau-absent x)))))

(faisceau-absent 0) ;; compteur = 0 et faisceau absent
```

Testez le programme. S’il ne fonctionne pas changez les valeurs des seuils et vérifiez le bon fonctionnement du programme. Il peut être utile d’ajouter une trace à votre programme pour afficher la valeur du capteur de lumière à chaque cycle. Vous pourrez ainsi voir à la console les valeurs minimales et maximales que le capteur retourne. Les seuils devraient être proches de ces valeurs (disons à 10% et 90% dans la plage de valeurs possibles). Le programme suivant est utile pour obtenir la valeur des seuils :

```

(define senseur
  (lambda ()
    (light 2)))

(define boucle
  (lambda ()
    (display (senseur))
    (display "\n")
    (boucle)))

(boucle)

```

## 5 Exercice 5 : La toupie

À vous maintenant d'écrire des programmes et de les tester. Commencez par un programme qui fait pivoter le robot en place. Le robot est initialement au repos, puis se met à tourner de plus en plus vite dans une direction, puis se met à ralentir jusqu'à ce qu'il soit immobile. Puis il doit faire la même chose dans l'autre direction, et ainsi de suite. Ajoutez aussi un effet visuel en contrôlant la LED 2. Lorsque le robot accélère la LED doit être verte. Lorsque le robot ralentit la LED doit être rouge. Chaque cycle doit durer de 10 à 15 secondes environ.

## 6 Exercice 6 : Contrôle par lampe de poche

Écrivez maintenant un programme pour contrôler le robot à l'aide d'une lampe de poche. Le robot est initialement au repos. Lorsque le capteur de lumière 2 reçoit la lumière de la lampe de poche, le robot se met à avancer et s'arrête lorsque la lampe de poche n'illumine plus le capteur. La fois suivante, il faut plutôt faire reculer le robot lorsqu'il y a de la lumière, et ainsi de suite.

## 7 Exercice 7 : Calcul de la vitesse de rotation

Écrivez un programme pour faire pivoter le robot sur place (sens de rotation différent pour les deux moteurs et puissance égale). Utilisez une puissance moyenne.

Pouvez-vous maintenant calculer le temps que prend une rotation (en millisecondes) et l'afficher à la console? Il faut se servir d'un capteur de lumière et d'une lampe de poche comme point de référence. Faites quelques tours pour déterminer quelle est la lecture pour le point le plus lumineux (la lampe de poche). Puis calculez le temps, avec deux appels à la fonction `clock`, entre les moments où l'intensité lumineuse dépasse de 80% puis 90% de l'intensité maximale. Quel est le temps de rotation pour chaque niveau de puissance? Quel est le temps de rotation si seulement un des deux moteurs est actif?

Note : essayez de faire un minimum de communication avec le robot car l'envoi d'information est lente et cela peut perturber la cadence d'exécution de votre programme et donc la fréquence de lecture du capteur. Il est conseillé de faire tourner le robot et de faire la lecture du capteur de lumière pendant 5 ou 10 secondes sans aucun affichage, et puis finalement d'afficher la plus grande valeur lue du capteur.



## 8 Exercice 8 : La danse du robot

Écrivez une fonction pour faire un demi-tour (approximativement). Les résultats de l'exercice 7 seront utiles ici. Notez cependant que l'inertie du robot fait en sorte qu'il tourne moins vite au début d'une rotation qu'une fois qu'il a atteint une certaine vitesse. Écrivez une autre fonction pour faire un quart de tour. Maintenant, écrivez un programme qui utilise ces fonctions pour faire danser le robot : un pas en avant, un demi-tour, deux pas en avant, un demi-tour et on continue. Remplacez les demi-tours par des quarts de tours. Inventez votre propre pas de danse!

## 9 Exercice 9 : Le chercheur de lumière

Combinez maintenant les capteurs de lumière et les moteurs pour concevoir un programme qui va chercher la source de lumière la plus forte et se diriger vers celle-ci. Il faut faire attention car le robot peut se déplacer beaucoup entre deux lectures du capteur de lumière et il est alors difficile de le contrôler précisément. Mieux vaut faire un petit pas, une lecture du capteur, un autre pas, une autre lecture, et ainsi de suite.

Un algorithme simple est le suivant : tourner sur place tant que la valeur du capteur est en dessous d'un certain seuil, et sinon avancer.

Essayez des algorithmes plus intelligents. Comparez la performance de votre programme à celui de votre voisin. Lequel atteint la source de lumière plus rapidement? Quel algorithme est le plus fiable? Est-ce que votre programme arrive à se débrouiller s'il entre en collision avec un obstacle ou un autre robot?