CAROLL : Un Robot Pas Cher pour l'Apprentissage (par Renforcement)

Francesco De Comité

March 27, 2006

Ce document décrit la conception, la construction et la programmation d'un robot autonome bon marché, utilisant un minimum de circuits électroniques et ne demandant pas une grande expertise des montages. Il utilise un micro-contrôleur bon marché mais puissant, possédant assez de mémoire de programme et de données pour s'autoriser à implémenter des algorithmes d'apprentissages non triviaux (apprentissage par renforcement, réseaux de neurones, algorithmes génétiques ...).

Généralités

La figure 1 montre l'aspect général de *Caroll*, tandis que les figures 2 et 3 représentent des vues de dessus et de dessous du robot. Ses caractéristiques principales sont :

- Micro-contrôleur PIC18F2620[1], avec 64K octets de mémoire de programme (soit 32768 instructions), 3986 octets de mémoire de données et une EEPROM de 1024 octets, cadencé entre 16Mhz et 40Mhz.
- Deux drivers de moteurs TC4427A [2], pour isoler l'alimentation du processeur et celle des deux moteurs.
- Quatre capteurs de lumière réglables à base de photo-résistances. Ces capteurs sont démontables.
- Quatre capteurs de proximité digitaux, à base de circuit IS471F, eux aussi démontables.
- Deux moteurs Faulhaber à démultiplication integrée. Ces moteurs sont sans doute la partie la plus chère du montage.

[1] `http://ww1.microchip.com/downloads/en/DeviceDoc/39626b.pdf`
[2] `http://ww1.microchip.com/downloads/en/DeviceDoc/21423F.pdf`

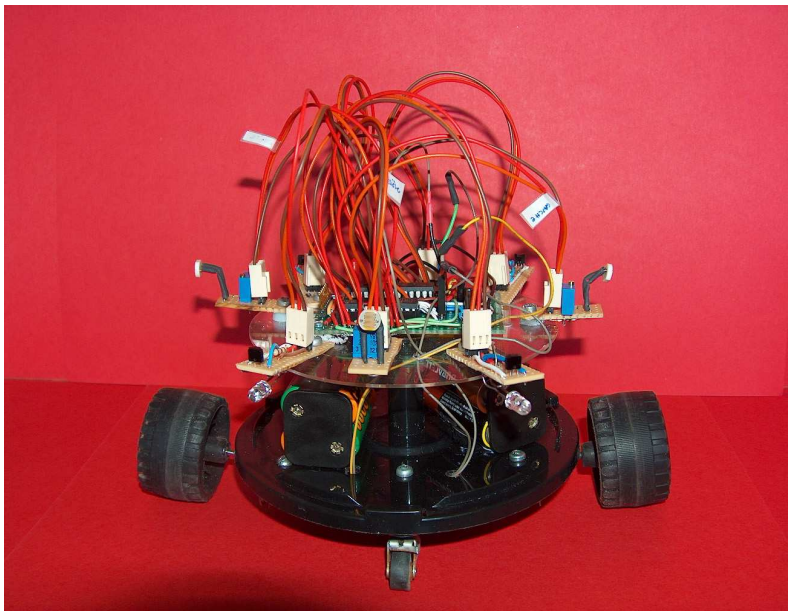

Figure 1: Une vue générale de Caroll

La carcasse du robot est une boîte de DVD, il est alimenté par 2×4 piles AA. On peut envisager d'augmenter la puissance des moteurs en augmentant leur alimentation. La *carte mère* possède les connecteurs qui permettent de programmer le micro-contrôleur par ICSD (i.e. sans avoir à démonter le processeur). On peut ainsi le programmer, utiliser le débuggeur dans l'environnement de développement, lire et écrire dans l'EEPROM.... Personnellement j'utilise le programmateur ICD2 de chez Microchip. Si vous connaissez d'autres programmateurs, faites-le moi savoir !
Le programme est écrit en MPLAB C18.

Construire le robot

La mécanique

La figure 3 montre comment des pièces de Meccano permettent de fixer les roues. Les roues sont aussi des roues de Meccano. Les deux petites roues stabilisatrices sont des galets d'entraînement de bandes magnétiques récupérés dans des walkmans.

- Les piles sont sur le plancher de la boîte.
- La *carte mère* et les capteurs sont vissés sur le disque de protection en plexiglas que l'on trouve dans les boîtes de CD/DVD (attention à ne pas

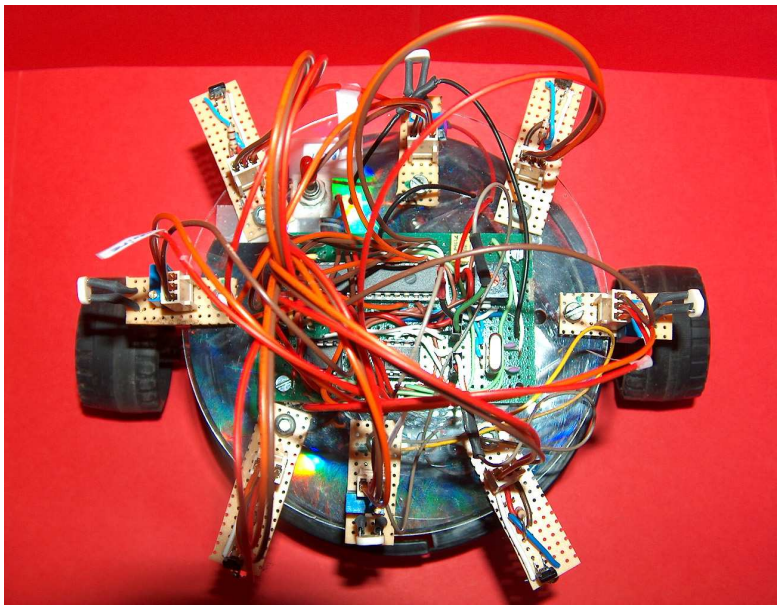

Figure 2: Caroll vu de dessus

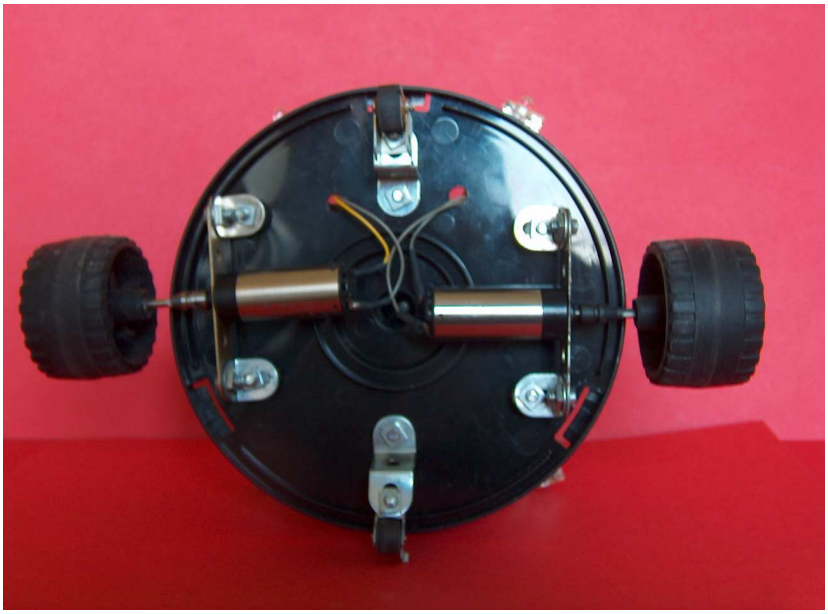

Figure 3: Caroll vu de dessous

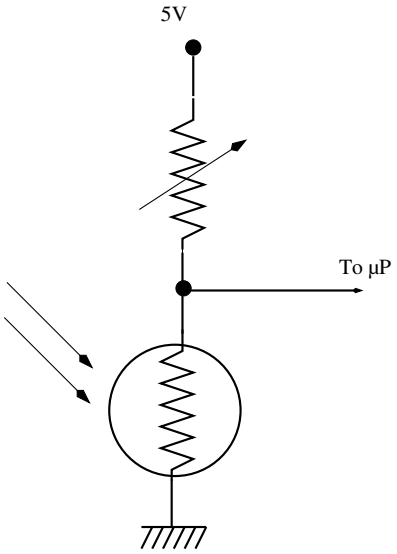

Figure 4: Le schéma du capteur de lumière

visser trop fort, le plastique n'est pas très solide !).

- Les capteurs sont connectés au micro-contôleur par des cables standards à trois fils.
- Les moteurs sont connectés surles sorties de deux drivers de moteur TC4427A.

Les capteurs de lumière

Les capteurs de lumière, à base de photo-résistances montés en diviseurs de tension, fournissent une entrée analogique au micro-contrôleur.
Un capteur de lumière correspond au schéma de la figure 4. Le cablâge effectif, utilisant une résistance réglable de 22KOhm, est visible sur la figure 5. Les figures 6,7 et 8 montrent le composant terminé, sous différents angles, afin de vous donner une idée de comment tout souder ensemble. La figure 9 montre le circuit monté sur le robot.

Capteurs de proximité

Ces capteurs sont construits autour d'un IS471F et d'une diode infra-rouge. Ils détectent des obstacles à une distance d'une dizaine de centimètres, selon la couleur de cet obstacle. La sortie est digitale (0 si obstacle, 1 sinon).

- IS471F datasheet :

  `http://www.datasheetcatalog.com/datasheets_pdf/I/S/4/7/IS471F.shtml`

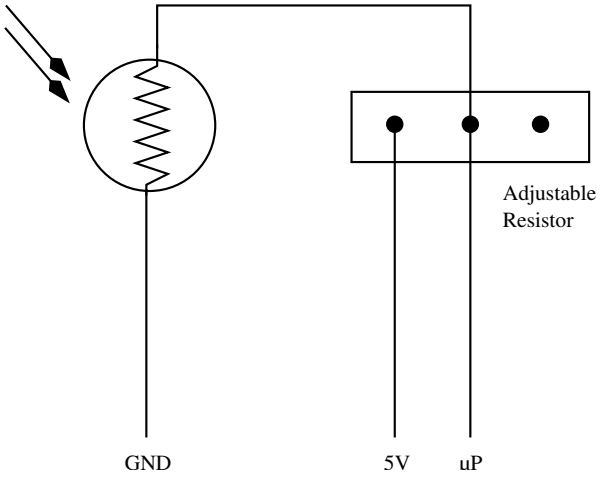

Figure 5: Le cablâge du capteur de lumière

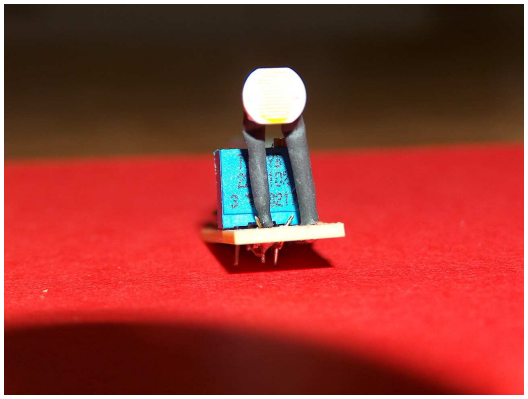

Figure 6: Le capteur de lumière vu de face

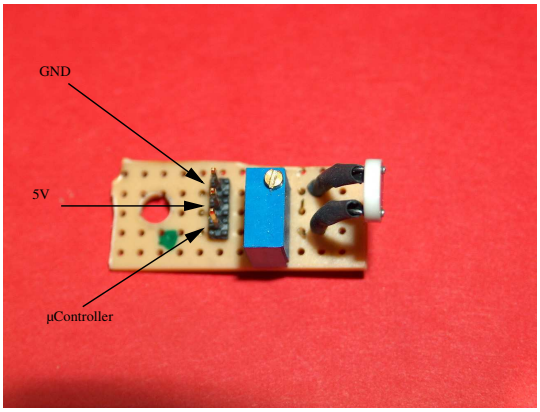

Figure 7: Le capteur de lumière vu de dessus

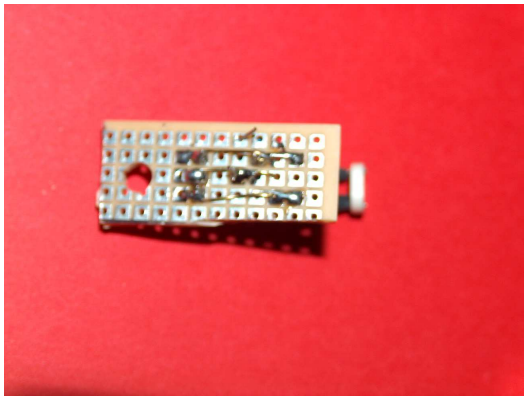

Figure 8: Le capteur de lumière vu de dessous

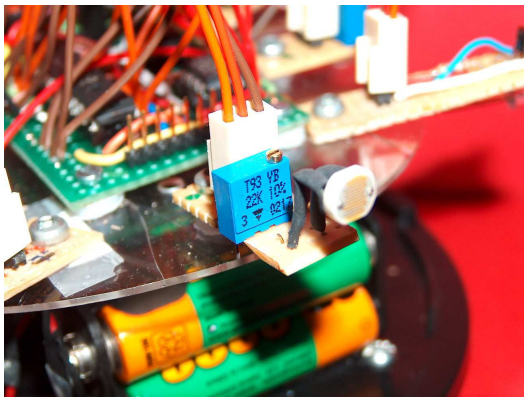

Figure 9: Le capteur de lumière en place sur le robot

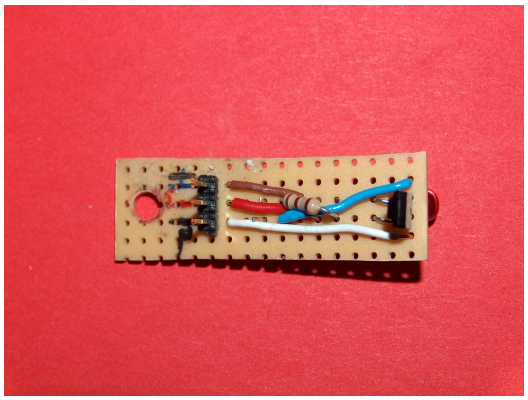

Figure 10: Le capteur de proximité vu de dessus

- Comment connecter et utiliser IS471F

  `http://www.kronosrobotics.com/an150/DAN150.shtml`

Les figures 10 et 11 montrent le module terminé. Les connecteurs sont identiques à ceux des capteurs de lumière : de haut en bas GND, +5V, micro-contrôleur.la figure 12 montre le capteur en place sur le robot.

la *carte mère*

En l'absence de matériel permettant de concevoir des circuits imprimés, l'aspect final de la carte contenant le micro-contrôleur et les drivers des moteurs dépend fortement de vos talents en électronique, soudure et bricolage...

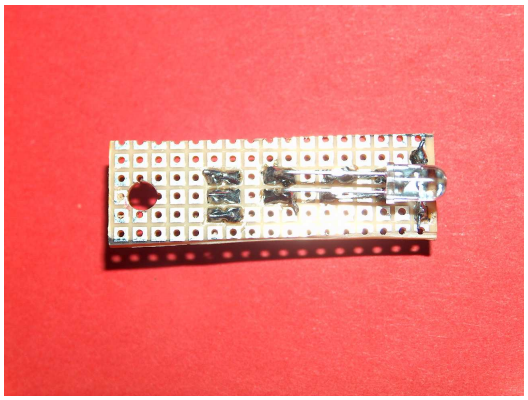

Figure 11: Le capteur de proximité vu de dessous

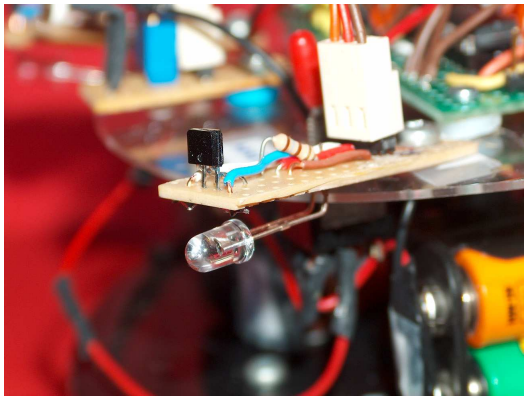

Figure 12: Le capteur de proximité en place sur le robot

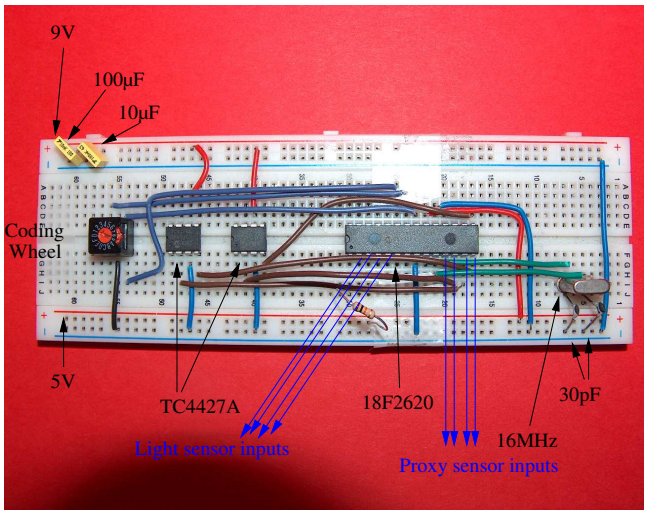

Figure 13: Le cablâge

Afin qu'on y voit encore quelque chose, j'ai choisi de montrer non pas la carte finale, mais l'étape précédente, à savoir le montage sur une platine d'expérimentation. C'est ce qui est représenté sur la figure 13.
La roue codeuse permet de choisir, avant d'allumer le robot, quel sera son comportement : mouvement réflexe (se diriger vers la source de lumière la plus importante), ou apprentissage. Les valeurs utiles de cette roue et leur signification sont en commentaires à l'intérieur du programme
La figure 14 montre comment le micro-contrôleur est connecté au programmateur (ICD2).

Programmation

Le programme implémente deux comportements et des variantes :

- Comportement réflexe : se diriger vers la source de lumière la plus intense.
- Apprendre à se déplacer pour récolter une récompense maximale.

Les deux comportements sont codés à l'intérieur d'un même programme. En fixant la valeur de la roue codeuse (avant d'allumer le robot), on choisit le programme qui sera exécuté.

Déplacement reflexe

On choisit ce comportement lorsque la roue codeuse vaut 4. Le programme se contente de lire les capteurs de lumières, de les ordonner dans le sens des

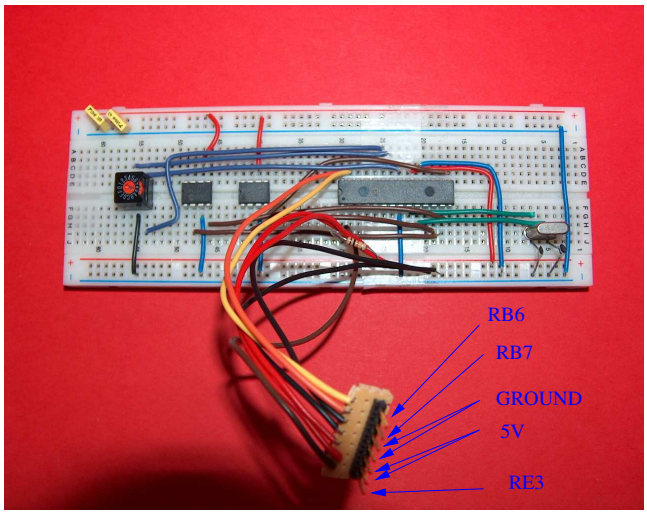

Figure 14: Le cablâge vers le programmateur

intensités de lumière décroissantes, et de choisir le mouvement correspondant à se mouvoir dans la direction d'où provient le plus de lumière.
Une petite modification du programme permettrait de tenir compte des obstacles. . .

Apprendre

L'algorithme implémenté est un Q-Learning simple :

- Chaque fois que le robot reçoit plus de lumière au total qu'au temps précédent, il reçoit une récompense.
- Cette récompense est augmentée si le mouvement choisi est en ligne droite (marche avant ou marche arrière, mais pas tourner).
- Le programme mémorise et estime la récompense qu'il recevra selon son choix de mouvement.
- Il choisit (souvent) l'action qui maximise cette espérance de gain.

Pour plus d'informations sur l'apprentissage par renforcement, consultez le livre de Sutton et Barto[3], et plus particulièrement la description de l'algorithme[4].

- Le source du programme est ici :

  `light2620ProxyQL.c`.

[3] `http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html`
[4] `http://www.cs.ualberta.ca/~sutton/book/ebook/node65.html`

- Le script de linkage : `18f2620.lkr`.
- Le fichier hex résultant de la compilation : `robotLight2620.hex`

Si vous utilisez le fichier hex directement avec un autre programmateur que le MPLAB ICD2, faites-le moi savoir !

Quelques commentaires sur le programme

Un état du système, tel qu'il est perçu par le robot pourrait être décrit par :

- L'ordre des capteurs de lumière (en fonction de l'intensité lumineuse que chacun d'eux a reçue).
- L'état des capteurs de proximité.

Il y a 24 ordres possibles pour les quatre capteurs de lumière, et 16 configurations possibles pour les capteurs de proximité. Celà donne $24 \times 16 = 384$ états différents. Comme il y a 4 actions différentes, celà conduit à considérer 1536 couples (état,action). Comme on a besoin, dans les versions *basiques* du l'apprentissage par renforcement, de stocker les valeurs de ces 1536 couples, il nous faudrait $1536 \times 4 = 6144$ octets pour mémoriser ces nombres réels. C'est supérieur à la capacité de la mémoire de données du micro-contrôleur, et bien au-dessus de ce que l'on peut stocker dans l'EEPROM.
Si l'on ne peut pas stocker dans l'EEPROM ce que le robot a appris, il nous est impossible d'essayer de comprendre ce qu'il appris, et aussi de reprendre l'apprentissage à l'endroit où il l'avait laissé à l'épisode précédent.
A cause de ce problème de place mémoire, nous avons simplifié la description d'un état :

- On garde l'ordre sur les capteurs de lumière.
- On ne conserve que deux booléens qui indiqueront s'il y a un obstacle devant et/ou un obstacle derrière.

On se ramène donc à un total de 96 états, soit 384 couples (état,action). Mais l'espace occupé par 384 flottants est de 1586 octets.
C'est convenable pour ce qui concerne la mémoire de données, mais c'est encore trop pour l'EEPROM !
On décide alors de ne stocker que des approximations des $Q(s, a)$ dans l'EEPROM : comme ces valeurs sont toujours comprises entre 0 et 10 (celà est dû aux valeurs des récompenses élémentaires et à la valeur de gamma), on multiplie $Q(s, a)$ par 6553.5, on en garde la partie entière, et on stocke l'entier obtenu (sur deux bits) dans l'EEPROM. Le procédé inverse permet de reconstruire un flottant.
On n'occupe plus alors que 768 octets de l'EEPROM, ce qui nous laisse même de la place pour éventuellement stocker d'autres informations sur le déroulement de l'apprentissage (valeur de alpha, valeur de epsilon ...).

Conclusions

Il reste encore quelques réglages à effectuer pour améliorer la qualité de l'apprentissage : valeur et décroissance de alpha, valeur et décroissance de epsilon,valeur de gamma...
Il reste beaucoup de place dans la mémoire de programme (le programme dont on parle ici n'en occupe que 10% !). On peut sérieusement envisager l'implantation de programmes non triviaux : réseaux de neurones, algorithmes génétiques...
Contactez-moi pour plus d'infos, détection de bugs ... : decomite_at_lifl.fr