E-Gab

Carte de commande de bas niveau pour un robot

RobotCoco[1]

Auteur : Justin CANO, cano.justin@gmail.com

Ancien président de E-Gab mais électronicien invétéré !!

le 03/08/16

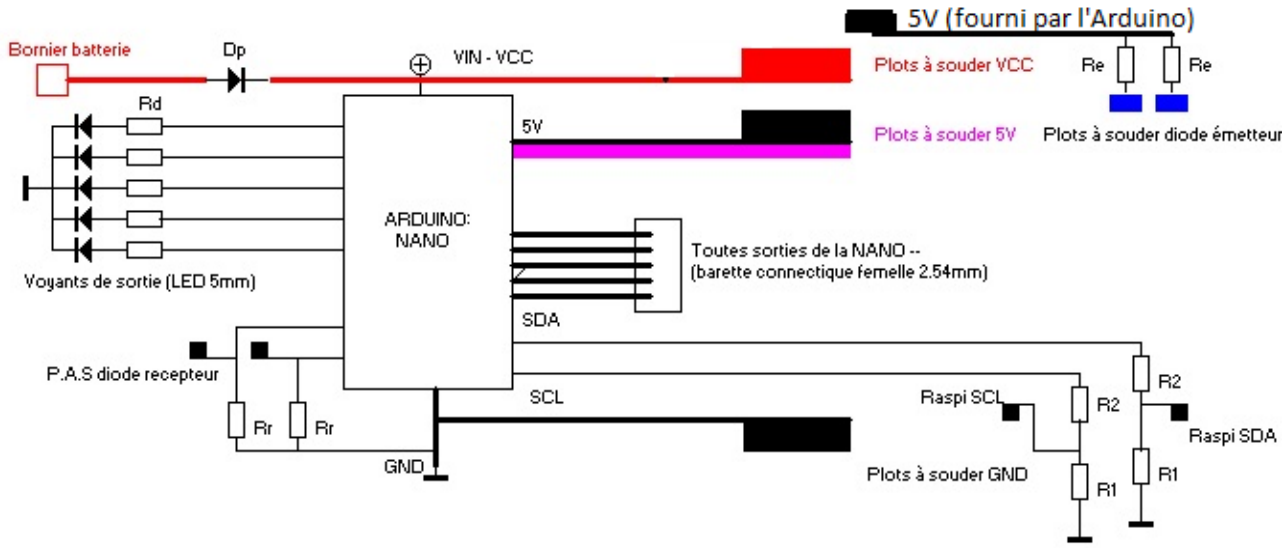
Table des matières

1 Je sais, je pourrais faire mon Fierro avec ce nom. J'ai fait ce précis pour Constanza Fierro avec qui j'ai développé Constantin, notre robot chéri ^^

But

Cette dernière permet de créer un robot avec des effecteurs de bas niveau, des capteurs anti collision Ultrason, et un asservissement des moteurs. Elle repose sur une Arduino Nano et peut être commandée en I2C par une Raspberry PI

Circuit électrique :



Nomenclature :

- D1 à D5 : diodes LED rouge 5mm, classiques. Si l'on suppose qu'il leur faut 2,1V pour fonctionner @10mA (estimation grossière mais suffisante malgré les sensibles variations de caractéristiques suivant les modèles) on peut en déduire la valeur de la résistance de protection Rd. $Rd_{théorique} = \frac{U_{alim} - U\,nominale^{diode}}{I^{diode}_{nominale}} = \frac{5-2,1}{0,01} = 290 \approx Rd \in [220\,\Omega ; 330\,\Omega]$
- Rd = 220 ou 330 ohms (cf précédent)
- Dp : diode de protection anti court-circuit. On peut prendre des 1N4007 ou des 1N4001, il faut en tout cas, par prudence avoir Imax > 1A
- Le couple R1 et R2.
  - **Contrainte :** les « Raspis » vivent entre 0V et 3,3V mais les Arduinos entre 0V et 5V
  - On va donc créer un pont diviseur de tension. On met donc aux extrémités le « fort » voltage (Masse et Arduino) et au centre le « faible » (Raspberry PI)
  - On a donc la relation suivante, qui doit être vérifiée : $\frac{3,3}{5} \leq \frac{R1}{R1+R2}$ on remarque que l'on demande ici, approximativement un rapport 2/3 minoré.

    On prend donc R1 = 2R2 et le tour est joué.

- Attention de ne pas trop tirer de courant inutilement (risques de chauffe et d'épuisement des batteries) ni d'en avoir trop peu (risque de perturbation à faible intensité).

  On va admettre Imin = 0,1mA et Imax = 10mA : il nous faut donc, idéalement que R1 + R2 soit compris (pour U=5V) entre 500 et 50 000 Ohms (on a le choix des deux valeurs).

- Les nominés sont donc (quelques exemples...) :

  - ✔ R1 = 2k, R2 = 1k
  - ✔ R1 = 47k, R2 = 22k (limite pour Imin)
  - ✔ R1 = 4,7k, R2 = 2,2k (je crois qu'il s'agit du meilleur couple possible, longue vie aux mariés !)

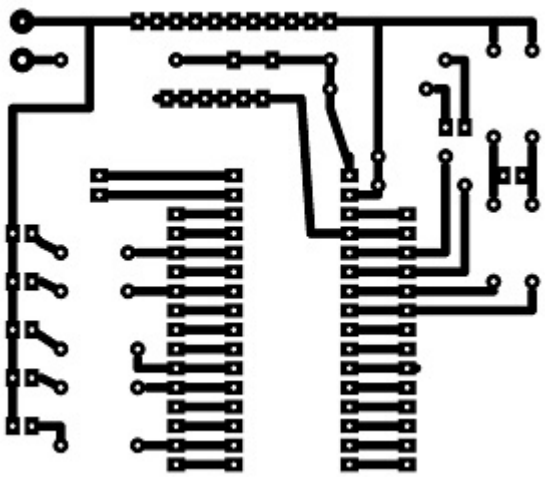
- Re et Rr, pour les émetteurs et récepteurs des capteurs mesurant les rotations du moteur. Cela dépend du cahier des charges, dans le cadre du robot que j'ai réalisé avec Constanza FIERRO, on était en présence de diodes IR adjointes de récepteurs du même type.

  - J'avais choisi une faible résistance pour l'emetteur Re = 150 ohms
  - Une résistance de polarisation du photorécepteur « classique » (polarisation au demi-milliampère 0,5mA) j'ai donc utilisé Rr = 10 k

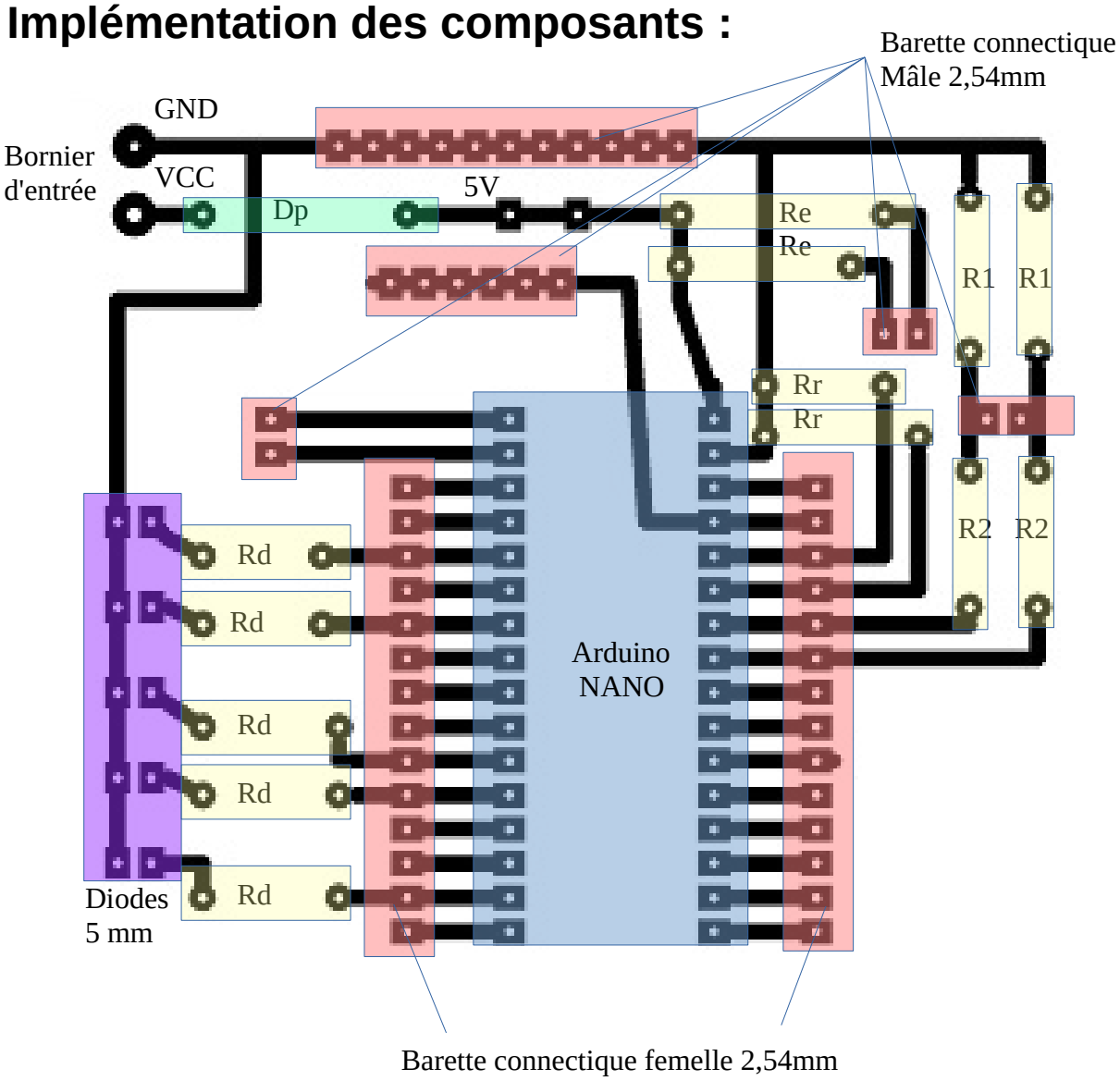
Typon :

Pour refaire le circuit chez vous

NB : sa largeur (Ox) est de 75 mm



Implémentation des composants :



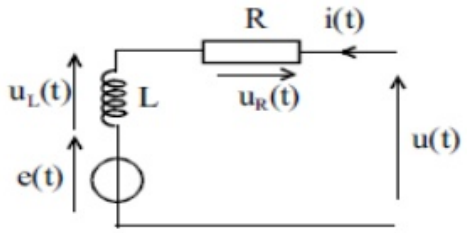
Réaliser l'asservissement des moteurs avec cette carte :

Tout d'abord, on veut asservir en vitesse de rotation un moteur à courant continu.

Modèle continu :

On fait dans un premier temps l'étude du moteur continu dans le domaine de Laplace (on considère d'abord de l'automatisme linéaire et non numérique, et on montre qu'il peut être modélisé par le modèle physique suivant :

En régime variable les équations électriques et mécaniques du modèle dynamique sont :



$$u(t) = e(t) + R.i(t) + L.(di(t) / dt)$$
$$e(t) = k.\Omega(t)$$
$$T_{em}(t) = k.i(t)$$
$$J.(d\Omega(t) / dt) + f.\Omega(t) = T_{em}(t) - T_R(t)$$

Illustration 1: Source : fr.slideshare.net

On a (dans l'ordre) l'équation électrique, l'équation de couplage électromagnétique (induction de la force contre électromotrice) , l'équation résultant des forces de Laplace (circulation du courant qui engendre un couple T, pour torque en anglais) ainsi que l'équation mécanique (théorème du moment cinétique appliqué à l'axe du moteur).

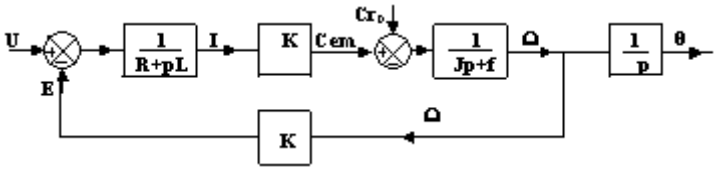


Illustration 2: Source http://sitelec.org/cours/bonnet/machinecc.htm

Avec ces dernières et dans les conditions de Heavyside (variables nulles à t=0), on peut déterminer

la fonction de transfert du moteur (et son schèma-blocs du coup) lorsque ce dernier n'est pas asservi ni perturbé. On montre que ce dernier a un schèma-blocs du type :

Avec Ω, vitesse de rotation du moteur (ce qui nous intéresse) et Θ sa position angulaire (intégrale, ou 1/p dans les conditions de Heavyside de la vitesse angulaire). On note p (notation française) ou s la variable complexe de Laplace.

Pour l'étude de ceci, je peux vous laisser ce site qui résume le cours d'asservissement :

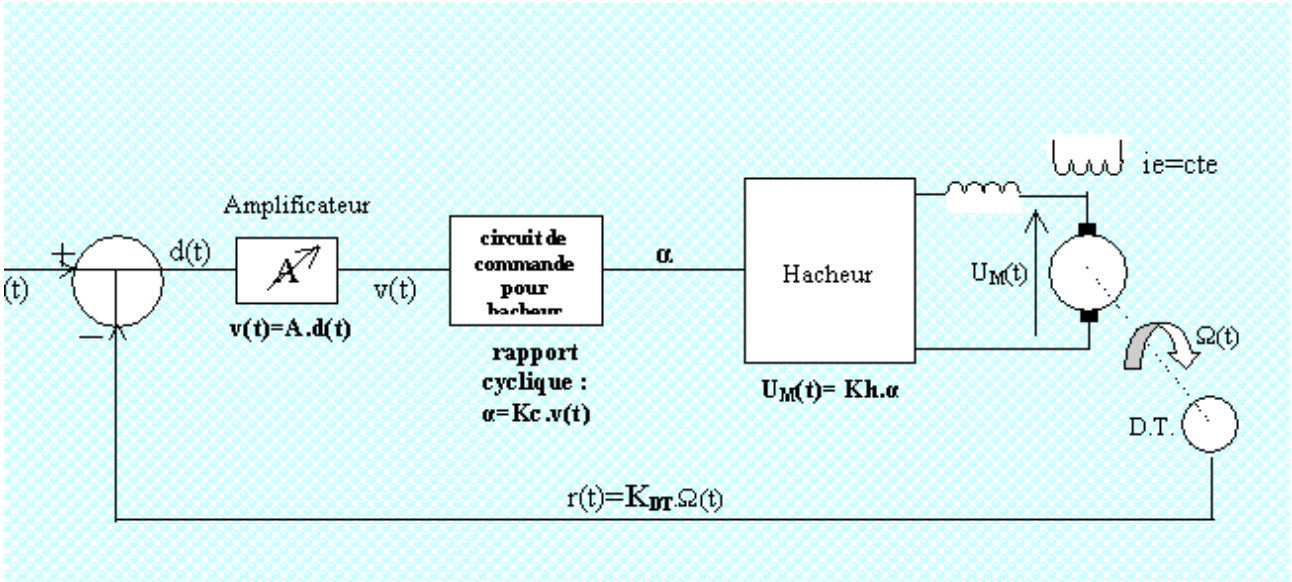
http://michel.lebeau.pagesperso-orange.fr/html/Cours/COURS1_Asservissements/COURS1_Asservissements_cours_corrige.htm

Cas général de l'asservissement en vitesse d'un MCC

On va se contenter d'une correction proportionnelle de l'asservissement dans le système bouclé c'est à dire :

Un capteur mesurera l'écart entre la consigne et la valeur effective de la vitesse. Cette erreur sera ré-injectée puis multiplié par un gain Kp (proportionnel) avant d'être envoyée à l'entrée du moteur (U sur le schéma-blocs précédent).

Dans le schèma, tiré du même site, je montre notre chaîne d'action bouclée (Kp = A dans ce cas) :



Donc, on a besoin d'un MotorShield pour fonctionner avec la carte Robotcoco.

Ce dernier est composé d'un pont en H (https://fr.wikipedia.org/wiki/Pont_en_H) qui permet de commander le moteur de manière omnidirectionnelle et à toute vitesse.

Je donne ici en guise d'exemple le motorshield officiel (et cher donc) d'Arduino https://www.arduino.cc/en/Main/ArduinoMotorShieldR3

Mais tout dépend des moteurs que vous utiliserez : attention aux intensités max et pensez qu'elles sont majorées au départ du moteur(!) et tout se passera pour le mieux.

Sinon un retour de l'asservissement par capteur à effet Hall, diode-phototransistor IR sur les rayons de la roue du robot ou autre est indispensable. Si l'échantillonnage est suffisant, on peut se placer dans l'approximation linéaire, et appliquer la correction proportionnelle à la mesure du retour du capteur qui sera nécessairement proportionnelle à la vitesse de la roue. En effet, ce dernier donne généralement un nombre en tours par secondes.

Détection et comptage de créneaux en Arduino (« tics »)

Ce programme permet le comptage à base de créneaux du nombre de tours, on les détecte à l'aide des fronts descendants.

```
// Variables globales

boolean etatHaut = false ;

long nombreDeTours = 0 ;

// fonction

void countNotch(int pin, long nombreDeTours ; boolean etatHaut) {

      if(digitalWrite(pin)) {

            // si on est dans un état haut

            boolean etatHaut = true ;

       }

      else if(etatHaut) {

      // si on est dans un état bas et qu'on a été dans un état haut précédemment

      nombreDeTours++ ; //incrémentation

      etatHaut = false ;

      }

      else {

      // si l'état est bas sans changement d'état

      etatHaut = false ;

      }

}
```

Correction proportionnelle en Arduino

On fait un cadre de lecture de T millisecondes, on acquiert le nombre de tours durant ce temps que l'on norme. On obtient ainsi une fréquence de rotation en tours par secondes. On indique une consigne cons en tours par secondes, un facteur de proportionnalité Kp, le pin de mesure pin_mesure, le pin d'application de la consigne par PWM (hacheur) [voir le cours d'Arduino sur http://wiki.centrale-marseille.fr/fablab] pin_application. On renseigne aussi pins correspondant aux bits de consigne de sens au nombre de 2.

On a la fonction suivante :

```
automation( float cons, float Kp, int pin_mesure, int pin_consigne, int bit_sens_a, int bit_sens_b, int  float T) {

long nombreDeTours = 0 ;

boolean etatHaut = false ;

long t_debut = millis() ; //date du début de l'acquisition

while( millis() - t_debut < T) { //tant qu'on est dans le cadre de lecture

	countNotch(pin_mesure, nombreDeTours, etatHaut) ; // acquisition

}

float toursParSeconde = (float(nombreDeTours)/T)*1000 ;

boolean sensPositif = boolean( 0 < cons) ; //détection du sens

float erreur = toursParSeconde – cons ; // erreur par rapport à la consigne

// étape de normalisation et de conversion en variable entière entre 0 et 255 pour le PWM

// on renseigne ici les maximum et minimum de vitesse

float maxVitesse = 42.0 ; //exemple 42 tours par secondes est la vitesse max

float minVitesse = 0.0 ;

vitessePWM = int( map( Kp*erreur , maxVitesse, minVitesse, 0, 255)) ;

	if(sensPositif) { //selon la datasheet du motorshield

		boolean Vbit_sens_a = true ;

		boolean Vbit_sens_b = false ;

	}

	else {

	boolean Vbit_sens_a = false ;
	boolean Vbit_sens_b = true ;

	}

//application des consignes

digitalWrite(bit_sens_a, Vbit_sens_a) ;

digitalWrite(bit_sens_b, Vbit_sens_b) ;
```

```
analogWrite(pin, consigne, vitessePWM) ;
}
```

Comment choisir sa valeur de correction ?

Il faut expérimenter en se souvenant que :

- Augmenter Kp revient à approcher les pôles de la fonction de transfert de l'axe des imaginaires, ce qui a pour conséquence de
  - rendre **instable** le système bouclé
  - rendre plus **rapide** le système
  - rendre **précis le système** (erreur statique d'un système asservi avec une correction proportionnelle de classe 0 $E_s = \frac{1}{Kp+1}$)
- Diminuer Kp, revient à les éloigner, cela contribue à :
  - rendre plus **stable** le système
  - à le rendre plus mou, à le **ralentir**
  - à gâcher la précision

Cependant, en robotique, mieux vaut préférer la stabilité dans les bases roulantes. Donc pour régler Kp, je conseille de l'augmenter graduellement jusqu'à obtenir satisfaction avec ce réglage.