Précis Robotique, Asservissements et Arduino

Par Justin CANO,
élève en M.Sc.A. à l'École Polytechnique de Montréal,
Département de Génie Électrique
École Centrale de Marseille – Promo ent. 2014

- ♢♢♢ -

28 mars 2017

Comment construire de α à ω un asservissement en vitesse d'un moteur pour la robotique ?

Hypothèse : Vous ne disposez que d'une carte Arduino, votre moteur CC, sa carte de contrôle, un ordinateur avec MATLAB et du présent document.

Table des matières

Table des figures

1 Avant-propos

1.1 Mot de l'auteur

Avant d'étudier en Génie Électrique à Polytechnique Montréal, j'ai été Président du Club de Robotique de Centrale Marseille. Je me suis donc naturellement confronté au présent problème et il est clair que celui ci n'est pas trivial. En effet, on ne maîtrise pas les frottements lorsqu'on fait rouler par exemple des robots et il faut à tout prix asservir ce dernier si on veut le diriger de manière la plus précise qu'il soit. Je vous souhaite donc une bonne lecture, en espérant que le présent papier sera utile pour vos réalisations. À noter qu'il il est disponible sur le site du wiki du FabLab Marseille : `https://wiki.centrale-marseille.fr/fablab` et sur mes pages personnelles `https://jcano.perso.centrale-marseille.fr`, `http://justincano.com` .

Je précise que ce texte est libre de droits, à condition de me citer, et que son écriture provient de mon expérience personnelle en électronique que j'ai acquise à Centrale et Polytechnique. Veuillez donc m'adresser toute remarque relative à son contenu à l'adresse suivante : `cano.justin@gmail.com`.
Ce papier n'est évidemment pas un cours d'automatique mais il a la seule prétention d'aider l'électronicien désirant asservir un moteur à courant continu. Ce concept est évidemment généralisable à tout système physique à condition de le modéliser convenablement à l'aide d'une fonction de transfert d'ordre peu élevé (1 ou 2), pour l'ordre 2, il faut évidemment revoir la méthode que je présente surtout dans les illustrations (je traite en exemple un cas du premier ordre) mais la démarche à entreprendre est identique.

1.2 Objectifs du présent papier

1. Modéliser un moteur à courant continu en automatique
2. Quelques rappels d'automatique linéaire appliqués à ce problème
3. Corriger le système, comment le faire mathématiquement
4. Implémentation sous un microcontrôleur Arduino du présent algorithme

2 Modélisation d'un moteur à courant continu

Avant toute chose, il faut donner un sens physique au problème que l'on s'apprête à étudier. En effet, mieux vaut connaître ce que l'on veut réguler avant de tenter tout calcul hâtif.

2.1 Modèle du moteur

2.1.1 Équations physiques temporelles

Un moteur est régi par des équations différentielles physiques, on va essayer de les combiner pour obtenir la vitesse, en fonction de la commande (la tension au bornes du

moteur).
Notations :

- $\omega(t)$ vitesse de sortie du moteur
- $C(t)$ couple fourni par le moteur
- $\imath(t)$ intensité traversant le bobinage du moteur
- L, R, e Inductance, Résistance et force contre-électromotrice du moteur
- J, f Moment d'inertie et coefficient de frottement sec du moteur

Modélisation des perturbations : On supposera que la commande U est en fait la vraie tension physiquement appliquée à ses bornes U_{phy} moins une tension "fictive" qui modélisera notre perturbation extérieure U_p (exemple les frottements).

$$u(t) = u_{phy}(t) - u_p(t) \tag{1}$$

Équation électrique Un moteur est un dipôle RLE électriquement donc :

$$u(t) = L\frac{di}{dt} + Ri(t) + e,\ \tau_e = \frac{L}{R} \tag{2}$$

e est la force contre électromotrice induite du moteur (on fait tourner des spires dans un champ, cela provoque de l'induction d'où l'apparition d'un potentiel contre-électromoteur).

Équations magnétique On peut démontrer, dans le cas des moteurs à courant continus pourvus d'aimant permanent la relations suivantes :

$$e(t) = \Phi\omega(t) \tag{3}$$
$$C(t) = \Phi i(t) \tag{4}$$

Ou Φ est la constante d'induction du moteur et s'exprime en Weber

Équation magnétique : Le théorème du moment cinétique nous donne l'équation qui suit :

$$J\frac{d\omega}{dt} = -f\omega(t) + C(t),\ \tau_m = \frac{J}{f} \tag{5}$$

2.1.2 Équations physiques dans le domaine de Laplace

Comme nous sommes en automatique et qui plus est en commande classique, nous allons donc créer un modèle en transformée de Laplace. Je noterais s la variable de Laplace [1] et on supposera que toutes les fonctions respectent les conditions d'Heavyside c'est à

1. En sachant qu'en France c'est p mais je suis parti à l'étranger entre temps.

dire que toutes les conditions initiales sont nulles. On a donc l'ensemble d'équations suivantes, notre but est de trouver la fonction de transfert qui relie u et ω :

$$U(s) = (\tau_e s + 1)I(s) + e = (\tau_e s + 1)C(t)\Phi + \Phi\Omega = (\tau_e s + 1)(\tau_m s + 1)\Omega\Phi + \Phi\Omega \tag{6}$$

$$\frac{\Omega(s)}{U(s)} = \frac{1}{(\tau_e * s + 1)(\tau_m s + 1)\Phi + \Phi} = \frac{\frac{1}{\Phi}}{\tau_e \tau_m s + (\tau_e + \tau_m)s + 1} \tag{7}$$

Or, en pratique la constante de temps électrique τ_e est négligée par rapport à τ_m. En effet, une résistance de quelques dizaines d'ohm et une inductance ne dépassant pas le milihenri ne peuvent pas rivaliser avec l'inertie mécanique. En plus, en robotique on utilise des motoréducteurs et des roues ce qui l'augmente de surcroit. On peut donc simplement écrire la fonction de transfert de notre bloc physique représentant le moteur comme suit :

$$F(s) = \frac{\Omega(s)}{U(s)} = \frac{K_m}{s\tau_m + 1}$$

C'est un système premier ordre avec constante de temps τ_m et gain statique K_m à identifier. Il s'agit du modèle physique sans perturbations, toutes les étapes se font donc avec le moteur plus les roues tournant dans le vide (si vous modifiez les roues vous modifiez l'inertie et donc la constante de temps ! !).

2.2 Modélisez votre moteur grâce à une fonction de transfert

Pour réguler un système, il est de bon goût de savoir à quoi il ressemble numériquement : il faut donc trouver le couple K, τ voici une méthode (parmi d'autres) permettant de le faire :

2.2.1 Mesurer la vitesse ?

Afin d'asservir votre robot, vous allez utiliser des capteurs angulaires tels que des encodeurs ou des capteurs à effet Hall. Ces capteurs sont actifs donc nécessitent une alimentation pour leur bon fonctionnement. J'insiste, ces derniers sont angulaires, c'est à dire qu'il faut donner une valeur approchée de leur dérivée pour donner la vitesse angulaire. Deux fronts montants (ou ticks) en sortie du capteur signifie qu'une rotation de $2\pi/a$ a eu lieu a étant un entier non-nul et généralement relativement grand, par exemple : $a = 100 ticsk/tour$. Pour estimer la vitesse il vous suffit de compter le nombre de tours $N_{ticks}(t)^T$ durant une période T donnée (vous la fixez ex : $T = 20ms$) et appliquer la formule suivante :

$$\omega(t) \approx \frac{1}{a \times T} \times N_{ticks}(t)^T \tag{8}$$

Je donne ici une fonction en langage Arduino permettant de faire ce travail :

```
1  const int pin_acquisition = 12;
   const int pin_moteur = 3;
3  static int temps_absolu;
   static int periode = 20;
5  static int a = 120;

7  void setup(){
   Serial.begin(9600);
9  pinMode(pin_acquisition,INPUT);
   pinMode(pin_moteur,OUTPUT);
11 }

13 float mesureVitesse() {
   digitalWrite(pin_moteur,HIGH);
15 //la PIN 3 est a relier a l'entree du montage amplificateur
   //(pont en H) alimentant le moteur
17 temps_absolu = millis(); // le temps absolu au debut de l'acquisition
   int ticks=0;
19 boolean b;
   while(millis()-temps_absolu<periode){ //on mesure les ticks pendant T
21   b = digitalRead(pin_acquisition);
     delayMicroseconds(20);
23   if(b==0 && digitalRead(pin_acquisition)==1){ // detection d'un front
     montant
       ticks++;
25   }
   }
27 float vitesse = (float) ticks*6000/(a*periode); // conversion de la
     vitesse
   return vitesse;
29 }

31
   void loop() {
33   Serial.println(mesureVitesse()); // affichage de la vitesse toutes les
     20 ms
     Serial.print(millis()); // affichage du temps en ms
35 }
```

Initialisation : On suppose que la lecture se fait sur la `pin 12` de l'Arduino. On règle la période $T = 20ms$ et on suppose que le capteur nous donne 120 fronts montants par tour $a = 120$. On initialise la liaison USB sérielle qui nous renverra nos valeurs, la `pin 12` est configurée en entrée.

Routine : La fonction `mesureVitesse()` est constituée d'une boucle qui détecte les fronts montants (et donc le nombre d'impulsion par seconde retournée par le capteur. On incrémente le compteur à chaque front montant. Une fois la période de comptage écoulée,

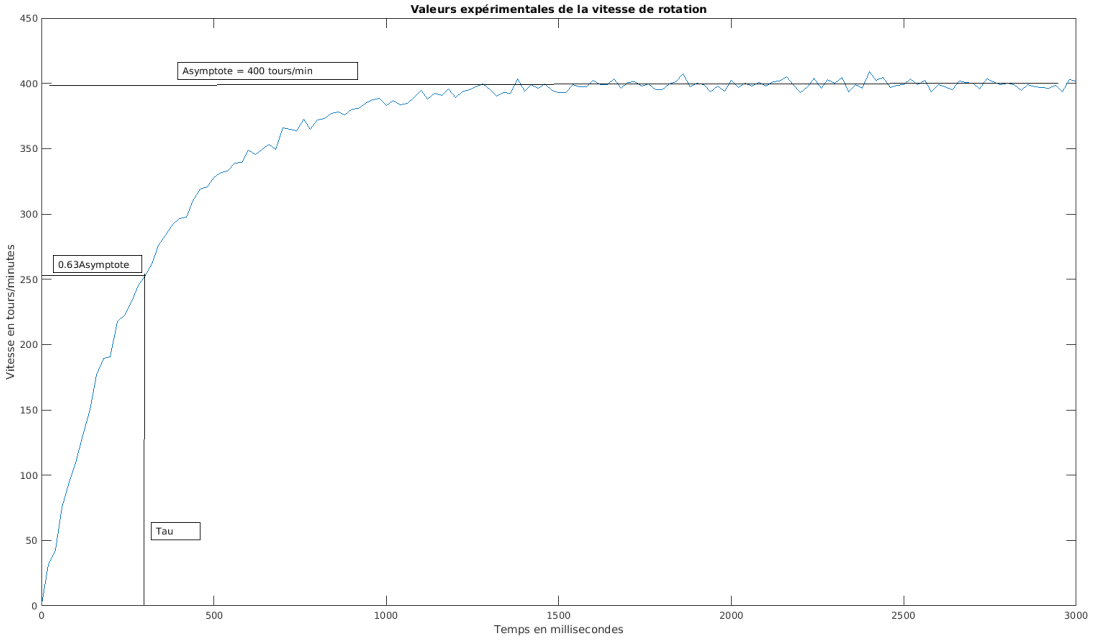

FIGURE 1 – Acquisition expérimentale de la vitesse et identification des paramètres.

on applique la formule ci-haut qui nous donne la vitesse en *tours/minute*. Cette fonction est dans la fonction `loop()` et donc s'exécutera jusqu'à ce qu'on l'arrête. Je suggère de stopper l'acquisition en régime permanent c'est à dire si les valeurs semblent plus trop évoluer. Après, il suffit de faire l'identification qui suit :
Il nous suffit de récupérer le flux de données provenant du câble sériel de l'Arduino (Cf. `http://wiki.centrale-marseille/fablab`) en copiant directement le contenu du terminal, afin de procéder au tracé de la courbe. Un logiciel de tableur tel que `OpenOfficeCalc` ou encore `Scilab` peut faire l'affaire pour séparer les valeurs du temps avec celle de la vitesse. Parvenus ici, tracer la courbe de vitesse en fonction du temps ne devrait pas poser de soucis, on devrait obtenir un résultat analogue à la figure 1.

2.2.2 Identification des paramètres

Comme nous le montre la figure 1, la réponse indicielle du moteur nous montre généralement qu'elle peut être assimilable à une réponse d'un système du premier ordre soumis à un échelon E. C'est à dire une fonction temporelle du type :

$$\omega(t) = E \times K \times (1 - e^{-t/\tau})$$

On sait que pour un échelon, c'est à dire une mise à une certaine tension E à une valeur donnée, le système tendra vers une valeur asymptotique valant :

$$\omega_{max} = \omega(\infty) = K \times E$$

Dans notre cas précis, dans la figure 1, on a, si l'échelon vaut $E = 10V$:

$$K = \frac{\omega_{max}}{E} \approx \frac{400}{10} = 40 \ \ min^{-1}.V^{-1}$$

On sait de plus que :

$$\omega(\tau) = K \times E \times (1 - e^{-1}) \approx 0.63\omega_{max}$$

Ce qui permet d'identifier graphiquement τ sur l'axe des abscisses : ici $\tau \approx 0.3$ en **secondes**. Les constructions nécessaires à ces considérations sont sur la figure 1.

3 Conception du contrôleur

De multiples contrôleurs existent dans le monde de l'Automatique. On se donne un petit cahier des charges qui peut correspondre à une application en robotique, afin de se donner quelques contraintes que l'on respectera par la suite.

- On veut un temps de réponse [2] à 5 % $T_5 \leq 0.5s$. Le moteur a une demi-seconde pour répondre quelle que soit sa valeur.
- On voudrait également éviter les trop grands dépassements [3] , c'est à dire que le pourcentage de dépassement (*Overshot*) soit tel que $P_{os} \leq 5$
- Il ne faut pas faire saturer l'alimentation du système . [4].
- Bien entendu, il faut que le système soit stable.

3.1 Stratégie employée

Nous allons diviser en plusieurs étapes cette conception :

1. Conception en temps continu du système
   - Création du modèle sous `Matlab`
   - Ajustement en simulation avec l'outil `pidtool` (option)
2. Discrétisation du système
   - Transformée en Z du contrôleur sous `Matlab`
   - Obtention de l'équation aux récurrences
3. implémentation en C++ de l'algorithme sur Arduino

3.2 Rappels : Théorie de la conception dans le plan S

Je vous conseille de lire un quelconque polycopié de cours si vous n'avez jamais traité ce sujet. L'objectif du présent papier n'étant pas un cours généraliste mais quelques éléments théoriques pour mieux comprendre le problème d'asservissement numérique d'un premier ordre.

2. Le temps de réponse à cinq pourcent est le temps de réponse au bout duquel pour un signal de sortie d'un système stable (au sens convergeant vers une valeur de sortie, voir la section 3.2) $y(t)$ convergeant vers une valeur $y(\infty)$ respecte l'inégalité $0.9 * y(\infty) \leq y(t) \leq 1.05y(\infty)$.

3. On ne veut pas que le moteur accélère et décélère trop fortement ce qui risquerait d'endommager le robot sur lequel il est implémenté.

4. Si la commande dépasse sa valeur maximale qui est la tension de ses batteries, le système devient non-linéaire et donc les hypothèses de calcul que l'on va poser vont devenir fausses.

3.2.1 Notions de pôles et de zéros

Dans le domaine de Laplace, les fonctions de transfert des systèmes continus et invariants peuvent s'écrire sous forme de fonction de transfert comme suit :

$$H(s) = \frac{N(s)}{D(s)}$$

Où N, D sont des polynômes en s.

Pôle : Les pôles sont les valeurs de s pour lesquelles $D(s) = 0$.

Zéro : Les pôles sont les valeurs de s pour lesquelles $N(s) = 0$.

Note : Toutes ;es valeurs ici sont complexes ! Il existe autant de racines fois leur multiplicité que les degrés des polynômes précédents.

3.2.2 Notations employées

On notera respectivement :
- p_i les pôles du système.
- z_i les zéros du système.

3.2.3 Propriétés du système en fonction des pôles

Note : Pour mieux comprendre les pôles et leur propriété, j'ai fait une petite feuille de "révision" présente sur mon site personnel `http://jcano.perso.ec-m.fr`

Stabilité : On veut que le système après correction soit stable tout d'abord. Ceci est assuré par la relation suivante à respecter pour tous les pôles :

$$Re(p_i) < 0 \Leftrightarrow STABILITE$$

Interdiction de déroger à cette règle, sinon on ne contrôle plus rien. Donc on placera les pôles dans **le demi-plan gauche** du plan de Laplace.

Oscillations : Dans le plan complexe de Laplace, on peut définir les coordonnées de pôles (x, y) cartésiennes comme étant les suivantes :

$$p_i = x + jy = \sigma + j\omega \qquad j^2 = -1$$

On peut également, comme pour toute géométrie du Plan, définir des coordonnées polaires (ω_0, θ) et il se trouve que théta est directement lié au caractère oscillatoire des

pôles. En effet, on peut définir le facteur d'amortissement ξ d'une fonction de transfert du deuxième ordre [5] comme suit :

$$H(s) = \frac{Y(s)}{R(s)} = \frac{K}{\frac{1}{\omega_0^2}s^2 + \frac{2\xi}{\omega_0}s + 1} \tag{9}$$

$Y(s)$ est la sortie du système et $R(s)$ la référence à poursuivre.
Mais ce fameux facteur d'amortissement est lié (voir la section 3.2.4) directement à l'angle polaire θ. En effet on pose :

$$cos(\theta) = \xi \tag{10}$$

De plus,on a la relation suivante entre le facteur d'amortissement ξ et le pourcentage de dépassement maximum P_{OS}.

$$\xi = ln(\frac{100}{P_{OS}}) \times \frac{1}{\sqrt{\pi^2 + ln^2(\frac{100}{P_{OS}})}} \tag{11}$$

Rapidité La rapidité d'un système est donnée par la relation approximative suivante [6] :

$$T_5 \approx \frac{3}{\omega_0 \xi} \tag{12}$$

Généralement, on dit que le temps de réponse d'un système est inversement proportionnel à l'opposé de la partie réelle de ses pôles. On négligera les pôles dits rapides face aux pôles dits lents (ce qu'on a fait dans la section 2.1.2, pour la constante de temps électrique beaucoup plus rapide que la mécanique, ce qui a permis de créer notre fonction du premier ordre dits rapide), c'est la notion de dominance des pôles.

Précision L'erreur à l'infini d'un système est caractérisée par : $e_{ss} = y(\infty) - r(\infty)$. Si le correcteur implémente un intégrateur (ce qui est notre cas) on peut prouver qu'elle est nulle à l'infini pour poursuivre un échelon [7] de référence ou rejeter un échelon de perturbation. On poursuit parfaitement les échelons : quoi de mieux pour notre robot ?

5. On tente toujours de se ramener à un deuxième ordre à pôles complexes conjugués dans la conception de systèmes asservis dans le plan S. Bien que cela nécessite des approximations.

6. En supposant que $\xi < 0.9$.

7. Un échelon $U_E(t)$ d'amplitude E est une fonction telle que $U_E(t) = E$ pour $t \geq 0$ ou $U_E(t) = 0$ sinon.

3.2.4 Une figure résumé

Oui, ceci est dans mes compétences d'illustrateur ! Voilà notre plan S, on notera le passage de coordonnées cartésiennes à polaires dans le présent plan pour un système du deuxième ordre tel que décrit par l'équation 9.

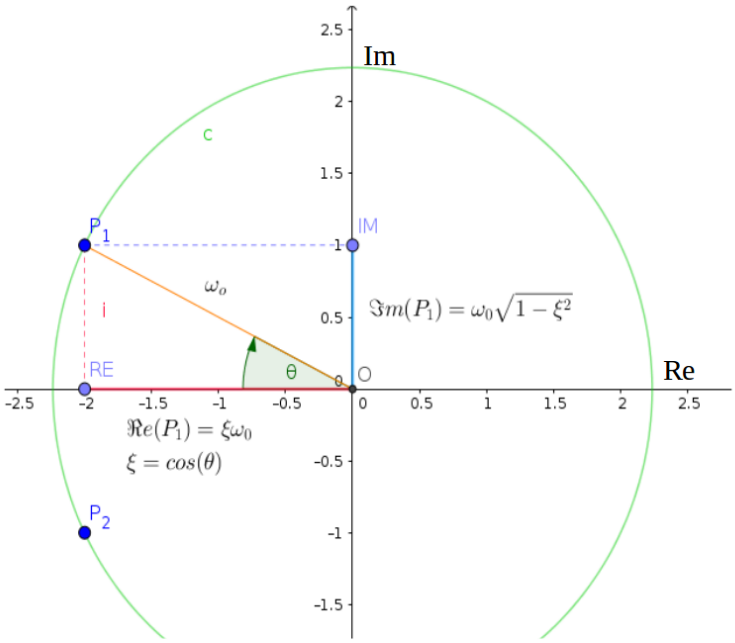

FIGURE 2 – Le plan S - Figure résumé

3.3 Méthode pour une conception directe dans le plan S :

On cherchera donc à placer des pôles en boucle fermée tels que défini à la figure 2. On va chercher à faire en boucle fermée un deuxième ordre approché qui aurait deux pôles dominants tels que (traduction des contraintes du cahier des charges) :

$$P_{OS} \leq 5 \Leftrightarrow \xi \geq 0.69 \tag{13}$$

On choisit $\xi = 0.707 = \frac{\sqrt{2}}{2} = cos(\theta)$ qui a le bon gout de respecter l'inéquation précédente et qui correspond à un angle de $\theta = 45\ deg$ et qui de plus impose que les **parties réelles et imaginaires des pôles sont égales.**. Voyons la contrainte de rapidité maintenant :

$$T_5 \leq 0.5s \Leftrightarrow \frac{3}{\omega_0\xi} \leq 0.5 \tag{14}$$

Or, $-Re(P_i) = \xi\omega$ par définition, voir la figure 2. On va se placer à la limite de cette frontière, définie par l'équation 14, car rappelons nous-le, créer des pôles rapide peut

faire saturer la commande ce que l'on doit à tout prix éviter. On a donc :

$$Re(p_i) = -3/0.5 = -6 \tag{15}$$

On va donc chercher à placer les deux pôles en boucle fermée tels que :

$$p_1 = -6 + 6j \qquad p_2 = -6 - 6j \tag{16}$$

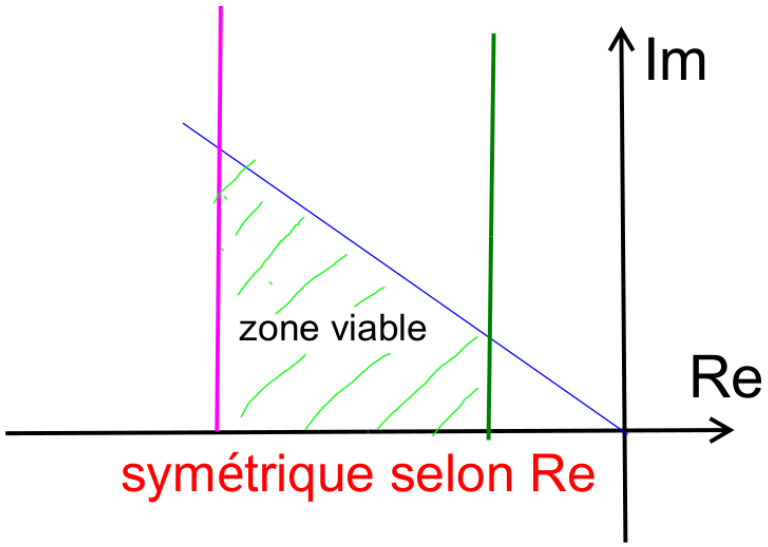

FIGURE 3 – Résumé de la géométrie que doivent respecter les pôles en boucle fermée. On notera que la conception est symétrique suivant l'axe des réels car les pôles sont dans notre cas toujours complexes conjugués.

Pour réaliser ceci il suffit de concevoir un compensateur de la façon suivante :

- On doit placer deux pôles situés à $-6 \pm 6j$, par exemple. [8]
- Pour garantir la précision un intégrateur, rendant le système de classe 1, devra être présent au sein du compensateur.
- Un zéro indésirable sera introduit par la dans notre contrôleur : nous négligerons dans une première approche sa présence (voir section 3.5).

3.4 Architecture globale du système

Nous alors pour cela concevoir un contrôleur proportionnel-intégral afin de réguler au mieux toutes les perturbations modélisées par l'entrée du signal $P(s)$, la référence que l'on cherche à poursuivre étant $R(s)$ et la sortie étant $Y(s)$.

Fonction de transfert du système global en boucle fermée :

$$H(s) = \frac{F(s)C(s)}{1 + F(s)C(s)} = \frac{\frac{K}{\tau s+1}(K_p + \frac{K_i}{s})}{1 + \frac{K}{\tau s+1}(K_p + \frac{K_i}{s}+)} \tag{17}$$

8. De manière générale il s'agit de placer deux pôles respectant la géométrie proposée à la figure 3.

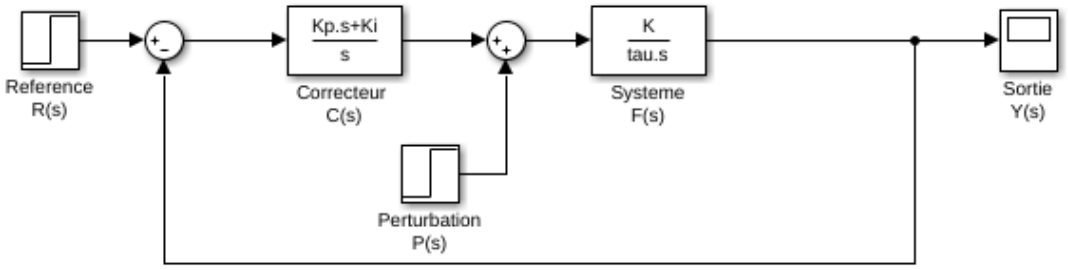

FIGURE 4 – Architecture globale du système en continu. Nous noterons que tout se passera dans l'Arduino excepté la fonction F(s)

Problématique : La question est... comment faire ? Comment régler le triplet K_p, K_d, K_i efficacement ? Comment l'implémenter dans une carte Arduino en C++ ?

3.5 Calcul des coefficients du correcteur en temps continu C(s)

On va tout d'abord essayer de calculer les coefficients comme si le système était un deuxième ordre pur. On connait grâce aux développements faits dans la section 2.2.2 les valeurs de K et τ. De plus, on peut écrire :

$$H(s) = \frac{KK_p(s + \frac{K_i}{K_p})}{s^2 + \frac{1+KK_p}{\tau}s + \frac{KK_i}{\tau}} \tag{18}$$

Mais, grâce aux développements de la section 3.3 on connait les pôles que l'on veut, le dénominateur de l'expression 18 doit être égal à :

$$D(s) = (s + p_1)(s + p_2) = (s - 6 + 6j)(s - 6 - 6j) = s^2 + 12s + 72$$

Nous avons donc deux équations et deux inconnus en identifiant les coefficients au dénominateur... formidable !

$$D(s) = s^2 + \frac{1 + KK_p}{\tau}s + \frac{KK_i}{\tau}$$

On fait l'identification en prenant les valeurs suivantes (comme dans mon exemple) $K = 40 \quad \tau = 0.3$

$$12 = \frac{1 + KK_p}{\tau} \Leftrightarrow K_p = \frac{12\tau - 1}{K} \approx 0.065$$
$$72 = \frac{KK_i}{\tau} \Leftrightarrow K_i = \frac{72\tau}{K} \approx 0.54$$

Par ailleurs, le zéro vaut : $z_1 = -Ki/Kp \approx -8.3$

3.6 Environnement MatLab

Vous pouvez utiliser le logiciel Simulink, dépendance de MATLAB [9] en vue de vérifier la réponse du système en continu. En fait, il faut regarder ce qu'il se passe en sortie du système pour bien visualiser le tout. On regardera ce qui se passe en sortie du bloc $C(s)$ également. Si vous avez fait votre identification avec une source de $12V$ il faudra veiller à ce que cette sortie ne s'approche pas trop du voltage max. On risque d'obtenir des saturations sinon et cela est problématique pour la dynamique du système.

On entrera les paramètres directement dans l'invite de commande ou dans un script MATLAB ce qui aura pour résultat le stockage dans l'espace de travail de ces variables. Par exemple :

```
K = 200;
tau = 0.3;
... etc
```

On pourra se servir de la figure 5 pour faire ceci. À noter que je donne un script résumé de la conception complète complet sous Matlab à la section 4.2. Ce dernier n'utilise pas Simulink mais rentre le modèle analytique directement.

Note : On peut également utiliser la fonction MatLab `pidtool(F)` pour régler le gain du contrôleur, `F` étant ici la représentation de la fonction $F(s)$ en fonction de transfert sous MatLab (*Transfer function*) . On ne passe pas par les formules et le résultat est plus immédiat, par contre, cet outil n'existe que dans MatLab depuis la version 2010b.

9. D'une manière analogue, on peux utiliser le freeware Scilab avec son logiciel de simulation XCos avec la palette (bibliothèque) CPGE, installable par l'outil de gestion des modules ATOMS

4 Implémentation dans le système avec Arduino

Maintenant, passons des milieux continus aux milieux échantillonnés puisque le micro-contrôleur Arduino en est un.

4.1 Éléments de théorie : Asservissements Numériques

4.1.1 Signal échantillonné

En asservissement numérique, on échantillonne les signaux avec une période d'échantillonnage T_e. Le signal est mesuré au départ de chaque période puis, la valeur est bloquée jusqu'à un nouvel échantillonnage :

$$U_e(t) = U_e(nT_e) \qquad t \in [nT_e, (n+1)T_e]$$

Cette méthode est appelée Bloqueur d'Ordre Zéro (ou BOZ)

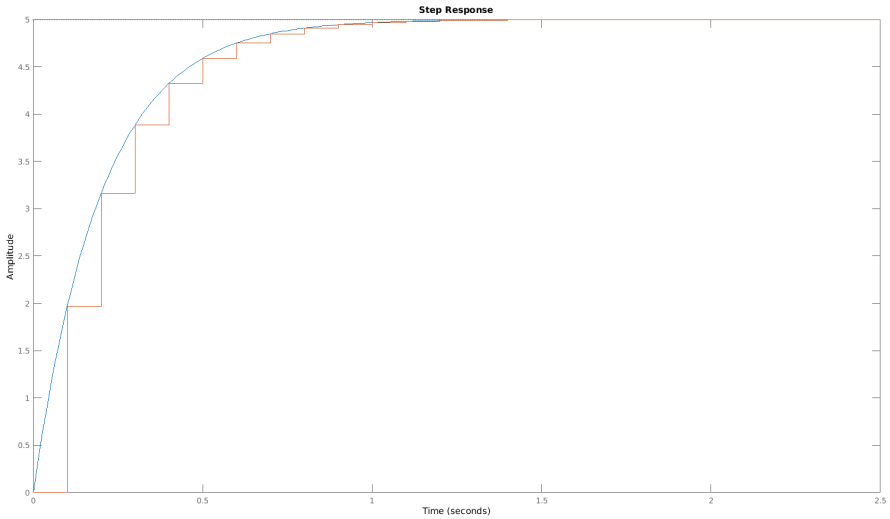

FIGURE 5 – Signal continu et échantillonné par BOZ avec une période de 0.1s

4.1.2 Transformation en Z

Une transformée en Z d'un signal causal x(n) La transformée en Z est pour les systèmes échantillonnés ce que la transformée de Laplace est aux systèmes continus.

Définition mathématique : Pour tout système causal, elle vaut :

$$Z(u(nT_e)) = U(z) = \sum_{n \in \mathbb{N}} u(nTe) \times z^{-n} \qquad z \in \mathbb{C} \tag{19}$$

Nous nous focaliserons pas sur toutes ses propriétés mais seulement celles qui sont utiles dans notre cas précis. Des livres et tout un pan de l'automatique traitant du sujet, je ne donne ici que des éléments pour mieux comprendre.

4.1.3 Quelques subtilités par rapport aux asservissements continus classiques

Stabilité : Le critère de Cauchy sur les séries entières s'applique ici. Un système est stable si et seulement si ses pôles (valeurs critiques) appartiennent au cercle unitaire. Autrement dit :

$$|p_i| < 1$$

Il faudra faire attention de vérifier que les pôles se trouvent bien dans cette région en concevant le système sur MatLab. Si le système diverge, il faudra sans doute concevoir un système plus robuste en continu, voire échantillonner plus pour s'approcher du système précédemment conçu. Il faut être prudent, même un premier ordre peut devenir instable si il est bouclé avec des gains inadéquats !

Théorème de Shannon : Pour garantir une bonne période d'échantillonnage, par ce biais une bonne stabilité en utilisant une approche tendant à ressembler à la conception en continu, on doit respecter le critère suivant :
Les pôles en boucle fermée en continu doivent appartenir à une bande B du plan S de Laplace définie comme suit :

$$B \ t.q. \quad \forall p_i \in B, \quad -\frac{\omega_e}{2} < \Im(p_i) < \frac{\omega_e}{2}$$

Avec ω_e, pulsation d'échantillonnage, relié à la période d'échantillonnage par la relation :

$$\omega_e = \frac{2\pi}{T_e}$$

Si on ne respecte pas ceci, des complications peuvent apparaitre :

- On peut sous-échantillonner le système et ainsi fausser complètement le système en temps discret. On perd de l'information si la fréquence d'échantillonnage ne vaut pas **strictement** plus du double de la fréquence caractéristique la plus élevée du système :

$$Pas\ de\ pertes \Leftrightarrow f_e > 2 \times max(f_{pole})$$

- Le signal ainsi produit ne correspond pas à ce qui se passe réellement dans le système amenant des erreurs de précisions ou pire : des instabilités.

Exemple : Un système $F(s) = \frac{1}{s^2+1}$ oscillateur harmonique de fréquence $f_{pole} = 1$. On l'échantillonne à $T_e = 1/f_e = 4$ on obtient $F_4(z) = \frac{1.654z+1.654}{z^2+1.307z+1}$ (ne respecte pas le théorème de Shannon) et pour $f_e = 3 \Leftrightarrow T_e \approx 0.33$ on obtient $F_{0.33}(z) = \frac{0.05396z+0.05396}{z^2-1.892z+1}$ (respecte le théorème) ceci nous donne la figure 6 suivante pour la réponse à un échelon.

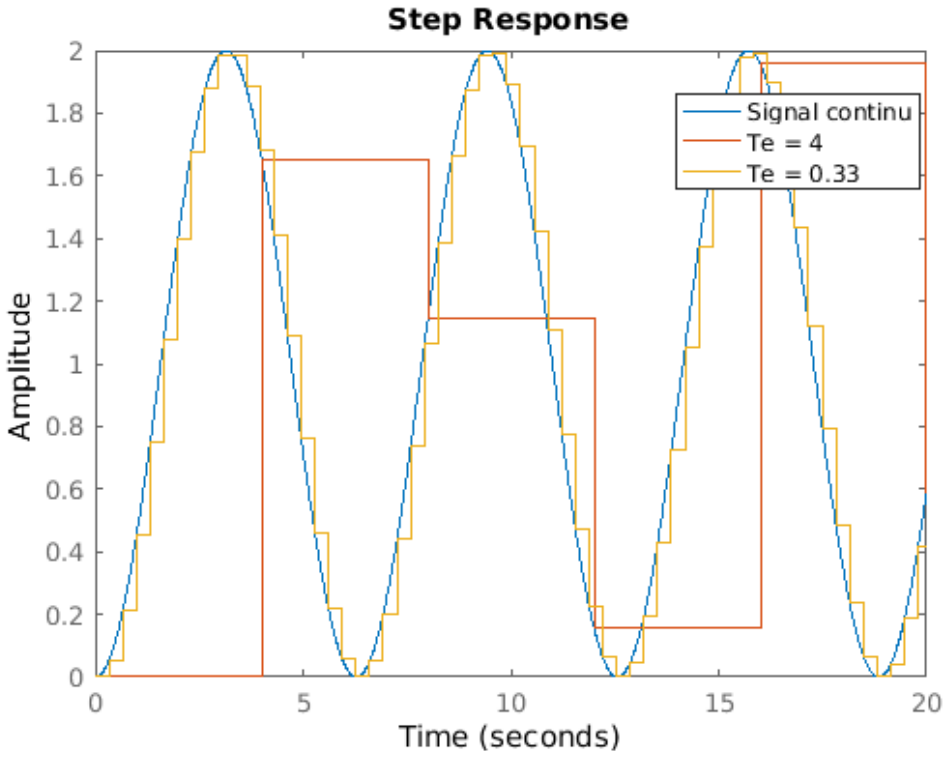

FIGURE 6 – Exemple de discrétisation de systèmes continus en adéquations ou non avec Shannon. La courbe pour $T_e = 4$ met en valeur un système clairement sous-échantillonné : on est plus capable de voir la sinusoïde à la bonne fréquence.

Théorème du retard : Soit un signal $u(n)$, la transformée d'un signal retardé de k fois le temps d'échantillonnage en Z vaut :

$$Z(u(n - kT_e)) = z^{-k}Z(u(n))$$

C'est pour cela que l'on utilise la transformée en Z, on peut déterminer une équation de récurrence facilement à partir de cette dernière

4.2 Discrétisation de l'équation à l'aide de MATLAB

4.2.1 Code MATLAB

On devra ainsi utiliser la fonction `c2d` afin de transformer les valeurs de la fonction de transfert continue du correcteur en fonction de transfert discrète en Z. On pourra utiliser le script MATLAB suivant :

```
% Parametres du moteur
K = 40;
tau = 0.3;

%Coefficients du polynome des poles choisis D = (s-p1)(s-p2)
a1 = 12;
a0 = 72;

Kp = (a1*tau - 1)/K;
Ki = (a0*tau)/K;

%Création des fonctions de transfert en continu
s = tf('s');
F = K/(s*tau + 1)
C = (s*Kp + Ki)/s
H = (C*F)/(1+C*F);
Rejet = -0.05*F/(1+C*F);

%Simulation temporelle en continu du systeme en BF (poursuite et rejet)
step(H, Rejet); %On poursuit un échelon unitaire et on rejette -5% de sa valeur

%discrétisation du contrôleur
Ts = 0.05;
C_e = c2d(C,Ts)
F_e = c2d(F,Ts);

% Vérification de la stabilité
figure
H_e = (C_e*F_e)/(1+C_e*F_e);
step(H_e)
```

4.2.2 Explications à propos du précédent code

1. On entre les bons paramètres (**et pas ceux de MON exemple)** du système identifié à la section 2.
2. On calcule les gains du contrôleur PI :

— On connait les pôles choisis (ici $p_i = -6 \pm 6j$) et donc les coefficients du polynôme caractéristique (dénominateur de la fonction de transfert en boucle fermée)
— On peut donc, connaissant le système en boucle fermée, déterminer (k_i, k_p) facilement pour placer les pôles au bon endroit (identification sur les coefficients du dénominateur de la fonction de transfert).

3. On calcule les fonctions de transferts en continu du système (correcteur et système physique [10]).
   — La fonction MatLab `tf()` (Transfer Function) permet de définir le symbole s comme étant la variable de Laplace.
   — Ensuite, on construit à l'aide de fractions rationnelles les différentes fonctions de transfert dont on a besoin.
   — On testera avec la fonction `step()` les réponses à des échelons de référence unitaire.
   — On testera également le rejet d'une perturbation de cinq pourcent par le PI (frottements arrivant d'un coup sur le robot, par exemple : traversée brutale d'une bande rugueuse).
4. On a choisi comme temps d'échantillonnage [11] $T_e = T_s = 50ms$ (ajustable) car on estime la mesure de vitesse suffisamment précise sur ce laps de temps. Mais cela dépend de votre capteur, si votre acquisition à la section 2 était trop bruité, il faudrait l'augmenter mais gare au sous-échantillonnage car il peut vous jouer des tours en terme de robustesse. Il faudra, le cas échéant alors modifier les pôles pour avoir un système plus lent mais plus robuste. **On conçoit le système en discret car notre mesure en vitesse est discrète**, une nouvelle mesure n'est disponible que toutes les $50ms$, c'est le cœur du problème.
5. On discrétise le système en utilisant la fonction `c2d()` (continuous to discrete) en renseignant le temps d'échantillonnage [12]. La transformée en Z de la fonction devrait s'afficher sur l'invite de commande MATLAB (absence de point virgule en MATLAB impliquant affichage).
6. Une dernière simulation est effectuée, on trace la réponse indicielle de la fonction de transfert discrétisée en boucle fermée pour s'assurer qu'elle est stable, si elle ne diverge pas, on peut donc garder ces valeurs. Sinon, il faut soit augmenter T_e soit trouver des pôles plus robustes en continu.

Note importante : Pour avoir de l'aide sur les syntaxes des fonctions MATLAB il suffit de taper sur l'invite de commande `help maFonction`.

10. Non nécessaire pour le système physique mais je préconise une simulation de son comportement pour vérifier sa stabilité. Donc il y a besoin de connaitre $F(s)$ pour calculer la fonction du système en boucle fermée.
11. Sampling Time en anglais.
12. Rappel : une transformée en Z d'un même système continu est en général différente avec deux temps d'échantillonnage différents.

4.2.3 Résultats obtenus avec mes valeurs :

En faisant simplement tourner le script précédent, on obtient :

Sur l'invite de commande

```
F =

     40
  ---------
  0.3 s + 1

Continuous-time transfer function.


C =

  0.065 s + 0.54
  --------------
        s

Continuous-time transfer function.


C_e =

  0.065 z - 0.038
  ---------------
       z - 1

Sample time: 0.05 seconds
Discrete-time transfer function.
```

Ceci signifie que la fonction de transfert de mon correcteur en Z, avec $T_e = 50ms$ vaut $C_e(z) = \frac{0.065z-0.038}{z-1}$.

Figures

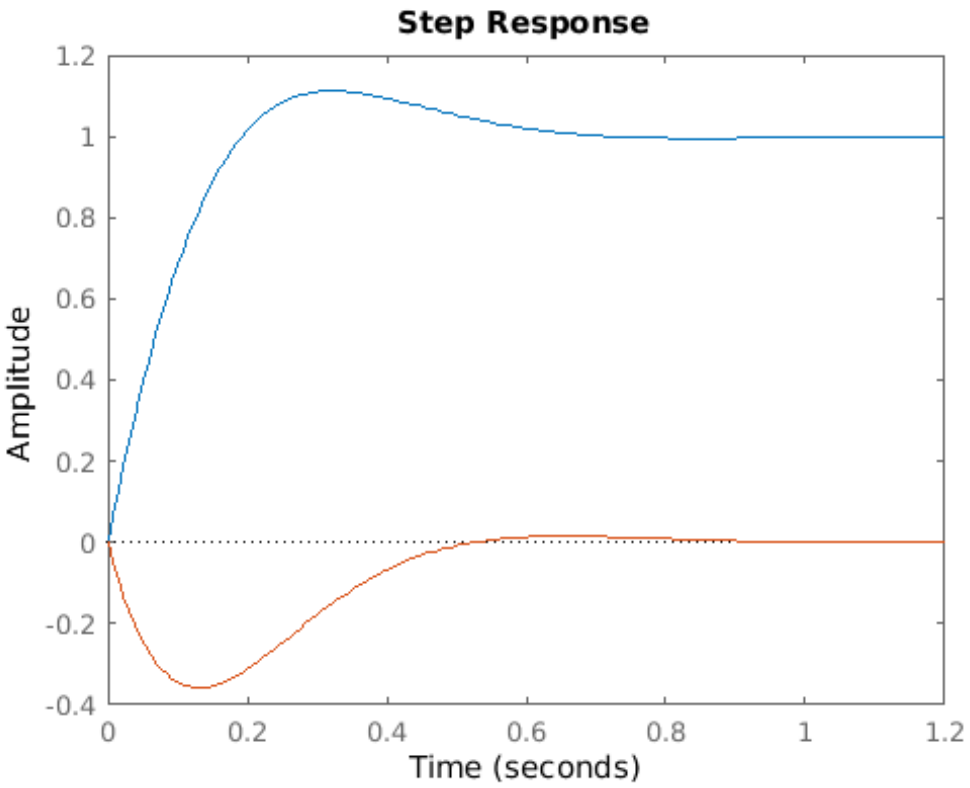

FIGURE 7 – Réponse continue, suivi en bleu, rejet d'un échelon de -5 pourcent en orange. Les performances sont celles fixées par le placement des pôles.

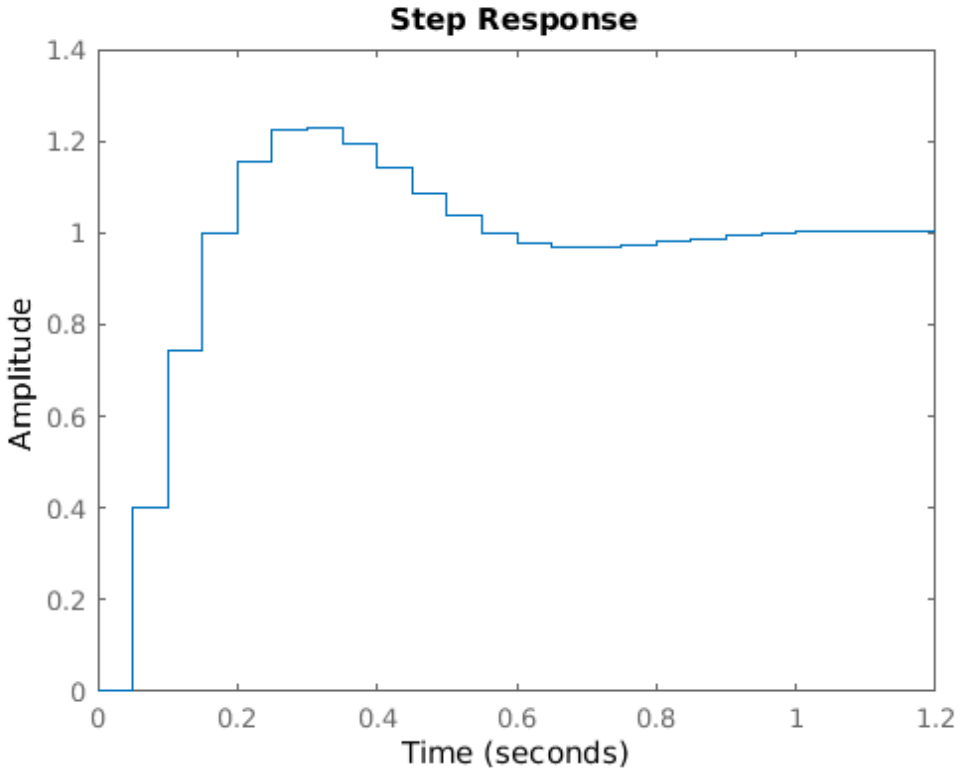

FIGURE 8 – Le système converge et a le bon goût de conserver les performances désirées.

4.3 Création de l'équation aux différences du contrôleur C(z)

L'étape est toute simple, nous disposons déjà de la transformée en Z. Il faut procéder ainsi :

1. Normaliser par z^{-k}, k étant le degré le plus élevé du polynôme du dénominateur, de manière à n'obtenir que des z^i avec i des entiers négatifs.
2. Séparer entrée et sortie.
3. Transformer en inverse en appliquant le théorème du retard.
4. Isoler U(n)

Exemple : Pour notre système, on a :

$$C_e(z) = \frac{U(z)}{E(z)} = \frac{0.065z - 0.038}{z - 1} = \frac{0.065 - 0.038z^{-1}}{1 - z^{-1}}$$

On a donc l'équation :

$$U(z)(1 - z^{-1}) = E(z)(0.065 - 0.038z^{-1})$$

Le théorème du retard nous donne (la transformée en Z étant linéaire) :

$$U(n) - U(n - Te) = 0.065E(n) - 0.038E(n - Te)$$

Une équation de récurrence peut donc être trouvée en notant les signaux comme suit : $X_{n-k} = X(n - kT_e)$

$$U_n = U_{n-1} + 0.065E_n - 0.038E_{n-1}$$

4.4 Implémentation sous Arduino

Pour implémenter cette fonction, la difficulté réside dans le fait de l'implémenter sur une sortie PWM de l'Arduino (voir l'explication plus complète sur http://wiki.centrale-marseille.fr/fablab). En effet, le PWM (*Power Wave Modulation*) délivre des carrés entre 0 et $5V$ à une fréquence d'environ $490Hz$.
En Arduino, pour faire ceci, on a la fonction `analogWrite(pin,value)`, le premier argument `pin` est un entier représentant la pin à utiliser. Pour une Arduino UNO, les sorties équipées du PWM à $490Hz$ sont les suivantes : 3, 5, 6, 9, 10, et 11. On notera que les pins 5 et 6 supportent également le PWM mais à $980Hz$, on utilisera ces dernières. Pour les autres cartes, vérifiez avec la datasheet sur le site officiel d'Arduino pour savoir quelles pin prendre.

Tension moyenne du PWM : Le deuxième argument, `value` est codé sur 8 bits et est directement relié à la tension de moyenne de sortie de l'Arduino. On a fait l'identification du moteur avec comme entrée directement la sortie de l'arduino donc :

$$u = \frac{value}{255} \Leftrightarrow value = 255 \times u$$

En supposant que `value` est entre 0 et 255 si ce n'est pas le cas, **il faut faire saturer value sinon on risque d'envoyer une tension erronée**. Exemple $256 = 0x100$ sera interprété comme $0x00 = 0$ et donnera une tension de $0V$ en sortie !

Code Arduino : On reprend la routine précédente plus quelques modifications :

```
const int pin_acquisition = 12;
const int pin_moteur = 5; // on branche le moteur sur la PIN 5
static int temps_absolu;
const int periode = 50; // 50 ms = Te (par exemple)
const int a = 120;
float v; //vitesse
float e; float e1= 0; float u=0; float u1=0;
// e(n), e(n-1), u(n), u(n-1)
float cons = 42.314; // consigne (par exemple)

void setup(){
  Serial.begin(9600);
  pinMode(pin_acquisition,INPUT);
  pinMode(pin_moteur,OUTPUT);
}

float mesureVitesse() {
  digitalWrite(pin_moteur,HIGH);
  //la PIN 3 est a relier a l'entree du montage amplificateur
  //(pont en H) alimentant le moteur
  temps_absolu = millis(); // le temps absolu au debut de l'acquisition
  int ticks=0;
  boolean b;
  while(millis()-temps_absolu<periode){ //on mesure les ticks pendant T
    b = digitalRead(pin_acquisition);
    delayMicroseconds(20);
    if(b==0 && digitalRead(pin_acquisition)==1){ // detection d'un front
    montant
      ticks++;
    }
  }
  float vitesse = (float) ticks*6000/(a*periode); // conversion de la
    vitesse
  return vitesse;
}
void correction(float vitesse, float consigne){
  float e = vitesse - consigne; //comparateur
  u = u1 + 0.065*e - 0.038*e1; //equation de recurrence
  int value = (int)u*255; //conversion sur 8 bits
  value = constrain(value,0,255); //saturation si on sort des 8 bits
  digitalWrite(pin_moteur,value);
  u1 = u; e1 = e; // memorisation des anciennes valeurs
}
void loop() {
   v = mesureVitesse();
 //  Serial.println(v); // affichage de la vitesse toutes les 50 ms
 //  Serial.print(millis()); // affichage du temps en ms (debug et test)
   correction(v,cons); // application de votre controleur (temps d
    execution neglige)
}
```

Pour deux moteurs sur la même carte : Recommencez l'étude pour le deuxième moteur et implémentez la deuxième acquisition de vitesse en même temps que la première, doublez vos variables, vous pouvez faire l'hypothèse que votre processeur sera assez rapide pour gérer ces deux choses à la fois...

5 Références

— Kilidjian, A., Cours d'asservissements numériques, option de 2A à l'ECM, 2015.
— Guchuan, Z., Cours d'asservissements numériques, Polytechnique Montréal, ELE 8200, 2017.
— Moudgalya, K.M., Digital Control, éd Willey, 2007.
— Jazzar, C., Cours de C/C++, option de 2A à l'ECM, 2015.
— `http://wiki.centrale-marseille.fr/fablab`, Cano, J. , Salles, P. Wiki du FabLab Marseille, page de la théorie Arduino, 2014, rév 2017.
— `http://arduino.cc`, Site officiel d'Arduino, rév 2017.