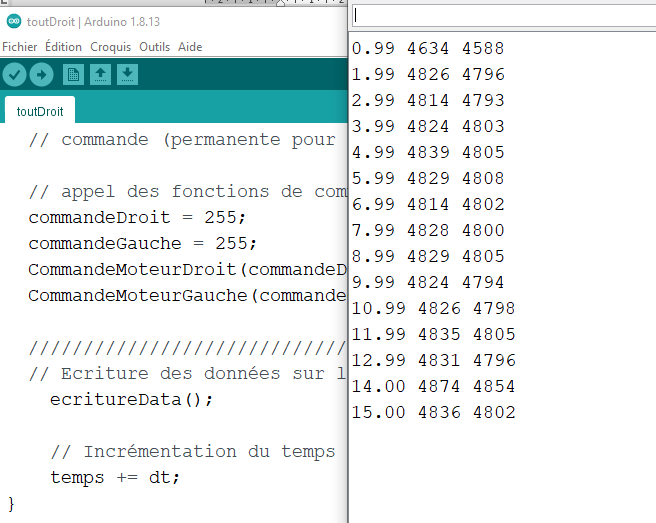
Vous trouverez un court topo sur l’asservissement PID ici par exemple <http://projet.eu.org/pedago/sin/term/6-asservissement_arduino.pdf>).

Le principe d’un PID (Position, Proportionnel, Integral, que l’on limité à PI ici) est de contrôler l’atteinte d’une consigne par un système dynamique. Ici, cela sera le contrôle de la rotation d’un moteur pour l’amener à rejoindre une consigne en vitesse. Ce contrôle est effectué en mesurant en direct l’état du moteur (position et vitesse) à partir de comptages de ticks (un tick est équivalent à un angle mesuré par le codeur interne).

Pour démarrer, plusieurs programmes vous sont fournis, à jouer dans l’ordre pour comprendre progressivement les commandes (et au final pour pouvoir les adapter à votre cas).

ATTENTION : Chaque programme intègre de nombreux commentaires qu’il s’agit de lire en détail. Ces commentaires vous aident à comprendre le code, vous donnent les indications pour les câblages des matériels, vous indique des procédures d’utilisation et vous indiquent enfin parfois les quelques consignes de sécurité à respecter.

* ComptTicks : Un programme pour juste compter les ticks renvoyés par les codeurs internes de vos moteurs  
  Si le nombre de ticks comptés par tour de roue renvoyés par les codeurs ne sont pas identiques, une règle de 3 pourra être appliqué dans vos codes (il est nécessaire que les nombres de ticks par tour exploités soit identiques pour les deux moteurs).
* toutDroit : un programme qui permet de vérifier que les moteurs sont correctement alimentés en fonction de consignes électriques. Vous pourrez parfois constater que même avec une même consigne de tension, les alimentations des moteurs peuvent différer et que même avec la même tension appliquée, les moteurs peuvent avoir des vitesses légèrement différentes. Ces différences devront être compensées.



Exemple de production en console. On voit que le nombre de ticks comptés sur chaque codeur est quasi identique et acceptable. La différence peut venir soit d’une différence de tension aux bornes des moteurs, soit de compteurs qui comptent un peu différemment. Vérifiez tout ce que vous pouvez et corrigez logiciellement au mieux.

* parcoursTemps : un programme qui permet d’aller tout droit pendant un temps, de faire une rotation, plus tout droit pendant un temps. Vous verrez que le comportement n’est pas si systématique et simple à mettre en œuvre.
* parcoursTicks : un programme qui donne des consignes en fonctions du nombre de ticks comptés (rappelons qu’un nombre de ticks correspond à une rotation et donc un déplacement).
* Deplacement : un programme intégrant un PI. Les commandes sont calculées en fonctions de consignes de vitesse et rotation. Tout se complique …  
  \* Ce programme est le programme d'un robot de l'année passée qui maitrisait parfaitement ses déplacements.  
  \* Ce robot utilisait un driver moteur différent de celui qui vous est fourni de base. Ceci vous   
  \* quelques modifications de ce programme pour correspondre à vos driver : Le programme \* n'utilise pas la bibliothèque AFmotor.h ==> Voyez en entête du code les modifications   
  \* minimales à faire  
  \* Ce programme ne correspond évidemment pas à votre robot ni à ses trajectoires mais peut   
  \* vous aider. ==> Voyez en entête du code les attentions minimales à apporter  
  \* Prenez le temps de bien lire en détail ce code pour l'adapter, cette adaptation est   
  \* obligatoire   
  \* sinon cela ne marchera pas.  
    
  \* Pour une exploitation progressive, faites des tests simples au départ : tout droit et je   
  \* m'arrête, tout droit, je m'arrête et je repars puis m'arrête à nouveau, puis introduisez un   
  \* virage, etc.  
    
  \* Versionnez votre code. Vous devez pouvoir revenir à un code précédent qui fonctionnait.  
  \* En cas de comportement étrange, essayez de mettre des affichages (fonction ecritureData)   
  \*qui par exemple validerait que vous n'êtes pas en saturation et donc algorithme ne   
  \* fonctionnant plus.