Centre professionnel du Nord vaudois
Filière d'automatique et d'électronique
Modules complémentaires techniques

CPNV - Centre Professionnel du Nord Vaudois

MCT - Modules complémentaires techniqeus

Explication code Arduino

P2213

Rédacteur : Quentin Surdez

Relecture : Rafael Dousse

École : CPNV

Date : Yverdon-Les-Bains, le 29 mai 2022

Table des matières

Table des figures

Liste des codes sources

1 Introduction

Ce document a pour but d'expliciter les commentaires et l'architecture du code de l'Arduino Nano pour le contrôle des moteurs. Il permet un approfondissement du code et de comprendre la logique derrière les différents choix effectués.

2 Asservissement des moteurs

L'asservissement de moteurs à courant continu est au coeur de notre projet. En effet, chaque fonction que nous construisons par la suite reposera totalement ou en partie sur la fonction d'asservissement.

L'asservissement permet de contrôler un certain paramètre dans un système. Il se caractérise par le besoin d'un système de maintenir une consigne donnée, qu'importe les perturbations infligées au système. Nous avons donc asservi notre système au niveau des moteurs. En effet, nous souhaitons que le robot aille tout droit lorsqu'on le lui demande, ce qui se traduit par le besoin d'avoir la même vitesse constante aux deux moteurs.

2.1 PID

Pour asservir nos moteurs à courant continu, nous avons choisi d'utiliser la méthode des facteurs PID. L'acronyme PID signifie proportionnel, intégral, dérivateur. Ce sont les 3 facteurs que nous utilisons pour construire notre asservissement.

Pour utiliser les différents facteurs, nous devons premièrement calculer l'erreur. Elle est la différence entre la consigne et la vitesse observée. Nous appliquons le facteur proportionnel directement à l'erreur. Le facteur intégratif est appliqué à la somme des erreurs sur le temps. Enfin, le facteur dérivateur s'applique sur la différence des erreurs sur le temps.

L'équation pour le calcul de la valeur de contrôle est la suivante :

$$u = k_p e + k_i \int e \cdot dt + k_d \frac{de}{dt}$$

FIGURE 2.1 – Équation du PID

Les différents calculs se font par rapport à un dt, cette particularité est très importante et a permis de diriger le développement du code. Pour mieux comprendre l'intégration de l'équation dans le code une image permet de visualiser le processus :

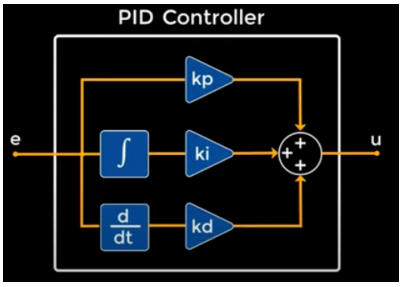

FIGURE 2.2 – Intégration de l'erreur et du PID

3 Interruptions

Comme expliqué au-dessus, la composante du temps est très importante. C'est grâce à elle que l'asservissement peut être fonctionnel et robuste. Pour intégrer ce besoin, il existe une fonctionnalité des microcontrôleurs MegaAVR appelée interruption. Les interruptions permettent, comme leur nom l'indique, d'interrompre le programme pour effectuer une tâche bien précise. Souvent, cette tâche est une incrémentation de valeur, ainsi l'interruption ne dure qu'un très court instant. Après avoir fini la routine d'interruption, le programme reprend exactement là où il en était.

3.1 Les interruptions attachées

La librairie Arduino permet d'utiliser une fonction appelée attachInterrupt(). Cette fonction permet de donner à un pin la responsabilité de lancer une routine d'interruption. Cette interruption sera alors attachée à un pin. Ainsi, si un pin est activé à un internal de temps régulier, la routine d'interruption se fera à un interval de temps régulier.

Cela a été mis en place pour notre projet en incorporant un Arduino gérant exclusivement le temps. Toutes les x secondes, un signal est envoyé à l'Arduino se chargeant de calculer le PID. Un schéma permet de visualier le processus :

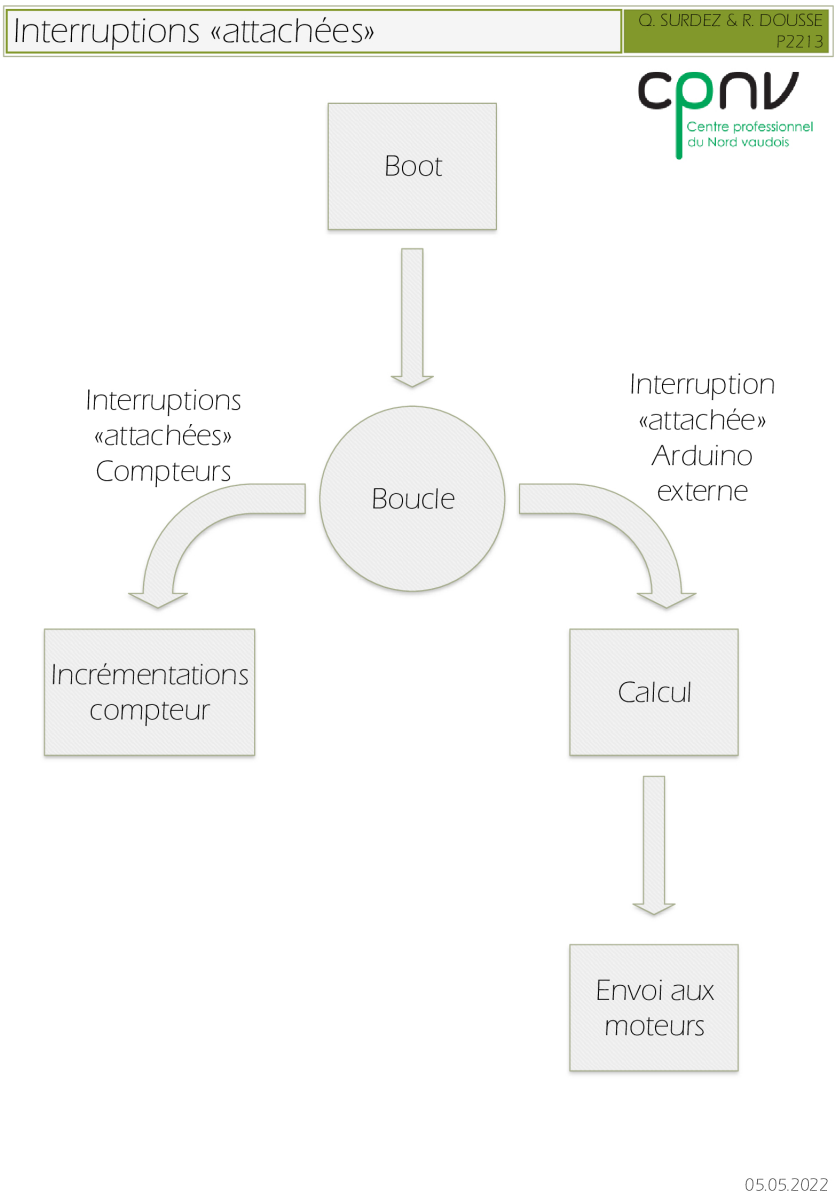

FIGURE 3.1 – Schéma des interruptions attachées

4 Code

pygme Le code a été écrit dans le but de répondre au besoin de notre projet. Les fonctions le composant et la logique appliquée seront discutés dans les prochains sous-chapitres. Premièrement, une explication du code nécessaire au setup de la communication I2C, puis donner de plus grandes explications que les commentaires sur les fonctions créées. Enfin, une une explication sur la remise à zéro des différentes valeurs pour que les différentes fonctions puissent intéragir entre elles sans compromettre l'intégrité de l'ensemble.

4.1 Communication I2C

```
// Set up du programme tourner sur soi ----------------------------
void tourneSurSoi(){


  attachInterrupt(digitalPinToInterrupt(photoElectricSensor), compteurDistance, RISING);
  attachInterrupt(digitalPinToInterrupt(pin_PID), asservissement, RISING);


  consigne_moteur = 2;
  consigne_moteur1 = 0;

  tourner = 1;

}
```

Listing 1 – Fonction du robot en C

4.2 Fonctions appelées

4.3 Remise à zéro des valeurs