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

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 :

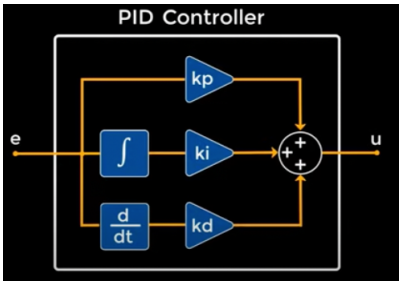

FIGURE 2.2 – Intégration de l'erreur et du PID

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 :

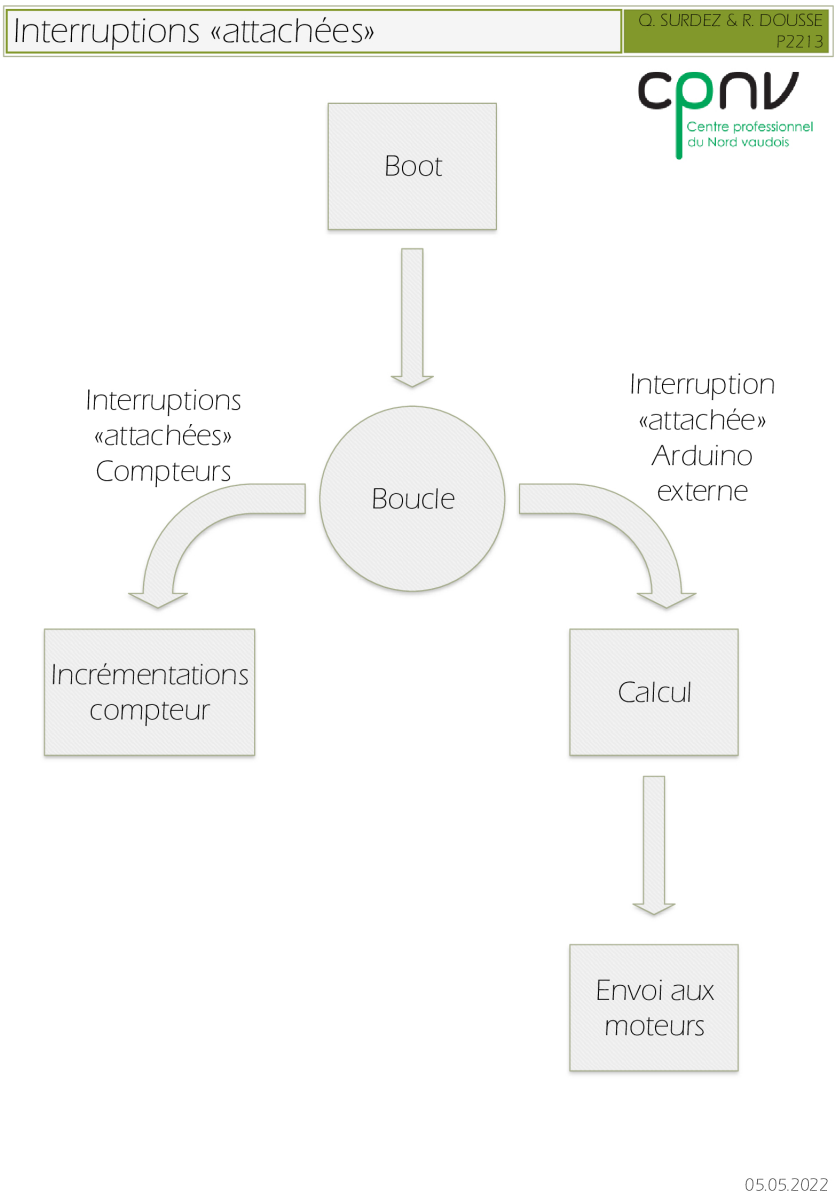

FIGURE 3.1 – Schéma des interruptions attachées

4 Code

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

La communication I2C est celle que nous utilisons dans le cadre de notre projet. Ce mode de communication est rapide (moins d'une seconde) et permet de rajouter des slaves facilement dépandamment de nos besoins. La librairie Wire.h avec Arduino permet une mise en place simple des différentes fonctions utilisées.

Pour commencer, son setup est composé de deux fonctions. La première est celle déterminant l'adresse donnée au microcontrôleur slave et la deuxième est celle qui détermine quelle fonction appelée lorsque le slave reçoit un ordre du master. Les voici :

```
// Set up de la communication avec le Raspberry PI via I2C -----------------
Wire.begin(0x08);
Wire.onReceive(receiveEvent);
```

FIGURE 4.1 – Setup I2C

Nous pouvons voir que nous choisissons l'adresse 08 et qu'une fonction (receiveEvent) est appelée à chaque fois que l'Arduino reçoit une information de la part du maître soit le Raspberry PI dans notre cas.

4.2 Fonctions appelées

La première fonction appelée lors de la réception d'un ordre est donc, comme vu ci-dessus, receiveEvent(). Voici sa structure :

```
// Communication avec le Raspberry via I2C --------------
void receiveEvent(int howMany){
  while(Wire.available()){
    int c = Wire.read();
    switch (c)
    {
    case 3:
      cinqMenAvant();
    break;
    case 4:
      tourneSurSoi();
    break;
    case 10:
      toutDroit();
    break;
    }
  }
}
```

FIGURE 4.2 – Fonction de réception et de commande

Nous pouvons voir que la fonction vérifie en premier lieu que le bus I2C est libre. Ensuite on stock la valeur qu'on lit sur le bus dans une variable locale. Cette dernière, dépendamment de sa valeur permet d'enclencher une fonction ou une autre. Nous utilisons un switch pour choisir la fonction. Nous avons ici trois cas :

- cinqMenAvant(), la fonction qui permet au robot d'aller tout droit pendant $5m$, puis de s'arrêter
- tourneSurSoi(), la fonction qui permet au robot de faire un tour sur lui-même, puis de s'arrêter
- toutDroit(), le robot se met en marche avec le PID et ne s'arrête pas

Nous allons observer la structure commune de ces fonctions avec cinqMenAvant() :

```
// Set up du programme 5m en avant ------------------------------------
void cinqMenAvant(){

  tick_distance=0;
  tick_distance1=0;
  enAvant = 1;

  //Interruptions
  attachInterrupt(digitalPinToInterrupt(photoElectricSensor), compteurDistance, RISING);
  attachInterrupt(digitalPinToInterrupt(photoElectricSensor1), compteurDistance1, RISING);
  attachInterrupt(digitalPinToInterrupt(pin_PID), asservissement, RISING);

  //On donne une consigne petite pour être sûr que le robot va tout droit
  consigne_moteur = 2;
  consigne_moteur1  = 2;
}
```

FIGURE 4.3 – Fonction pour se déplacer 5m en avant

Nous pouvons voir ci-dessus que les premières variables que nous changeons sont celles qui sont inhérentes à la fonction. Les $tick\_distance$ permettent de s'assurer qu'ils soient à zéro au lancement. Ce sont eux qui gèreront la distance parcourue. Ensuite, la variable $enAvant$ sera approfondie dans le prochain chapitre.

Nous voyons enfin ce qui nous intéresse particulièrement, les interruptions attachées. Elles sont à gérer pour chaque appel de fonction. Le setup des interruptions se fait exculsivement à ce moment précis. Nous choisissons les pins liés aux interruptions, quelle fonction la routine appelera et le mode sur lequel la routine se déclenchera. Ensuite, les consignes moteurs sont définies pour le calcul du PID.

Les différentes fonctions utilisées, ont toutes la même strucutre avec de léger changement comme la consigne moteur qui est à 0 sur un moteur pour que le robot tourne sur lui-même. Cependant, si un clean up n'est pas correctement effectué avant l'appel d'une autre fonction, les interruptions ne se comporteront pas de la bonne manière. Ainsi, il faut remettre les différentes valeurs à zéro.

4.3 Remise à zéro des valeurs

La remise à zéro des valeurs s'effectue à la sortie des fonctions, ou plus précisément, lorsque ces dernières ont rempli le rôle qui leur était attribué. La limite de ce système est qu'il ne gère pas le cas où une fonction n'est pas finie et une autre est appelée.

Les variables que nous changeons dans les fonctions comme $enAvant$ nous servent à désattacher les interruptions lorsque le travail de la fonction est terminée. Cette action se fait dans le loop() de notre script Arduino.

Voici un exemple pour la fonction cinqMenAvant() :

```
if (enAvant == 1){

  // Limite du nombre d'incrémentations pour que le robot fasse 5m depuis l'allumage
  if (tick_distance >= 580){
        setMotor(1, 0, Enable1, In1, In2);
        setMotor(1, 0, Enable2, In3, In4);
        detachInterrupt(digitalPinToInterrupt(photoElectricSensor));
        detachInterrupt(digitalPinToInterrupt(photoElectricSensor1));
        detachInterrupt(digitalPinToInterrupt(pin_PID));
        Serial.println("5m atteint");
        enAvant= 0;
  }
}
```

FIGURE 4.4 – Clean up de la fonction cinqMenAvant()

Pour rentrer dans le if principal, il faut que la fonction cinqMenAvant() ait été appelée. Ensuite, nous avons la condition qui déclenche le clean up. Au début du clean up, les moteurs sont forcés à 0 par la fonction setMotor(). Ensuite, les interruptions sont détachées. Celle qui permet de calculer le PID est la plus importante pour être certain qu'aucun calcul ne sera effectué. Enfin, une mise à zéro de la variable $enAvant$ permet de sortir du clean up.

5 Conclusion

Nous avons pu nous intéresser aux différentes composantes du script de notre Arduino Nano Every. Cela permettra au lecteur attentif de gérer et adapter le code écrit selon ses besoins. Le peu de fonctions qui n'ont pas été introduites dans le document, possèdent suffisamment de commentaires pour être clair et compréhensible par le lecteur.

Le développement du code se sera fait à travers maintes itérations. La version présentée aujourd'hui est dans sa phase finale. Incorporant la communication avec le Raspberry PI, elle se veut prête à répondre aux besoins du projet. La flexibilité des attachInterrupt() et detachInterrupt() permettent facilement de passer d'une fonction à l'autre, aussi longtemps que le clean up est correctement effectué.