POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF1900
Projet initial de système embarqué

Rapport final de projet

Simulation du comportement du robot

Équipe No < **28, 68** >

Section de laboratoire < **01** >

< Félix Duguay, Florence Cloutier,
Yazid Ben Saïd, Dali Messedi >

17 Avril 2020

1. Description de la structure du code et de son fonctionnement

Pour la conception de notre projet, nous avons décidé de séparer le projet en plusieurs petites étapes qui seront beaucoup plus faciles à coder et à implémenter. Voici les principales étapes de la conception de ce projet.

Makefile

La première étape est de modifier le makefile que nous avons conçu lors du TP7 en classe. En effet, nous avons d'abord corrigé nos erreurs soulignées par le correcteur afin d'améliorer la lisibilité du code et pour qu'il soit le plus concis possible.
De plus, pour améliorer la compilation de notre makefile, nous avons décidé d'utiliser un « wildcard » afin que notre makefile prennent tous les fichiers cpp automatiquement. Ce « wildcard » est de la forme :

Aussi, nous avons retiré quelques répétitions qui étaient présentes dans nos makefiles. En effet, nous avons opté pour faire 3 makefiles distincts. Nous avons un makefile général, qui contient toutes les répétitions de code entre les deux autres : le makefile des sources et le makefile des librairies. Le makefile des librairies sert à compiler tous nos fichiers cpp représentant différentes composantes matérielles de notre robot. Enfin, le makefile des sources permet de compiler le « main » qui donne des exécutions à notre robot.

```
PRJSRC= $(wildcard *.cpp)
```

Sonar

En deuxième temps, nous avons décidé de commencer à travailler sur le sonar qui sert à calculer les distances entre le robot et les objets environnants. Nous avons d'abord créé des fichiers qui se nomment respectivement « sonar.cpp » et « sonar.h », qui contiennent la classe sonar qui elle contient toutes les méthodes utiles pour contrôler le sonar. Nous avons décidé de faire cette classe en mode interruption, car il sera beaucoup plus facile d'implémenter ce code et de l'utiliser dans notre programme.

La classe sonar comporte donc 6 méthodes afin de contrôler les capteurs efficacement :

- sendTrigger ()

Cette méthode permet d'envoyer un signal de 10 ms sur la pin de « trigger » du sonar pour démarrer la lecture.

- partirMinuterie()

Cette méthode permet de partir une minuterie en affectant les bonnes valeurs aux registres. Aussi, cette méthode arrête les interruptions le temps de l'affectations pour les redémarrer ensuite.

- arreterMinuterie0()

Cette méthode permet de désactiver la minuterie et les interruptions engendrées par celle-ci en affectant aux registres de cette dernière la valeur de zéro.

- setupSonarAvecMinuterie()

Cette méthode affecte les bonne valeurs à DDRA et DDRB pour que le port A soit en entrée et que le port B soit en sortie.

- calculDistanceAvecMinuterie(uint8_t sonarID)

Cette méthode permet de calculer la distance d'un mur ou d'un objet par scrutation. Notre méthode ne teste qu'un sonar à la fois, et il nous faut donc déterminer quel sonar parmi les 3 nous voulons tester. La variable sonarID sert donc à déterminer si nous voulons effectuer une opération sur le sonar de droite, du centre ou bien de la gauche.

- determinerQuelleManoeuvreEffectuer(float distanceGauche, float distanceAvant, float distanceDroite)

Cette méthode détermine quelle manœuvre doit être effectuée en fonction des distances entrées en paramètre.

De plus, l'implémentation de l'interruption générée par le overflow du timer/counter 0 est dans le fichier sonar.cpp. Cette dernière incrémente un compteur qui est ensuite utilisé dans la méthode pour le calcul de la distance.

Bouton

Nous avons ensuite décidé de créer la classe bouton. Cette classe est très simple, elle permet seulement de déterminer si le bouton connecté au port D de notre robot est enfoncé. Nous avons donc créé une classe bouton, qui permet de déterminer cette condition.

- Bouton()

Ceci est le constructeur de l'objet bouton. Cela permet d'initialiser une constante qui correspond à la 3e pin que nous voulons utiliser, puisque le bouton est connecté à la 3e pin du port D.

- RegarderSiAppuye()

Cette méthode permet de déterminer si notre bouton poussoir est appuyé. Cette méthode retourne true s'il est enfoncé, et false s'il n'est pas enfoncé. Cette méthode prend aussi en charge le rebond possible d'un bouton matériel qui ne ferais pas un bon contact.

Afficheur 7 Segments

Le but de cet afficheur est d'afficher la puissance qui est donnée aux moteurs des roues en tout temps. Puisque nous ne pouvons pas toujours afficher deux chiffres différents dans cet afficheur, il nous faut passer les chiffres rapidement les uns après les autres pour créer une « illusion » que tous les chiffres sont toujours allumés. Cette classe comporte des attributs qui sont les vitesses des deux roues et le numéro du segment affiché présentement.

Cette classe contient un constructeur, deux setters pour les attributs ainsi que 5 méthodes :

- Afficheur7Seg(int vitesseRoueGauche, int vitesseRoueDroite)

Le constructeur affecte des valeurs de base aux attributs et initialise les pins A3, A4, A5, A6, A7 et le port C en sortie.

- partirMinuterie2()

Initialise le timer/counter0.

- arreterMinuterie2()

Arrete la minuterie.

- enableSeg()

Met en sortie le port A voulu pour afficher sur le 7 segments.

- displaySeg(uint8_t vitesse)

Allume les segments voulus pour afficher le bon chiffre.

- ConversionVitesseEn7Seg()

Cette méthode permet de convertir une vitesse en une vitesse mais en mode 7 segments, pour allumer les bons segments sur l'afficheur.

Moteur

Le but de cette classe et de ses fichiers créés est de faire tourner les moteurs de notre robot en utilisant un PWM matériel afin de changer les vitesses des moteurs. Nous avons donc besoin du constructeur de la classe moteur, du destructeur et de 2 autres méthodes pour contrôler adéquatement les moteurs :

- Moteur()

Ceci est le constructeur de l'objet moteur. Premièrement, on met le port D en sortie. Aussi, il nous faut mettre tous les registres de notre robot pour initialiser la division de l'horloge et mettre ce port en mode « Phase correct », afin de faire un PWM matériel avec notre robot en utilisant la minuterie

- ~ Moteur()

Ceci est le destructeur de l'objet moteur. Pour notre projet, le destructeur par défaut fait tout ce dont nous avons besoin. Nous n'avons donc pas besoin de modifier quoi que ce soit sur ce destructeur.

- ajustementPWM(float roueGauche, float roueDroite)

Cette méthode permet d'ajuster le PWM des moteurs des robots selon la vitesse passée en paramètre. Cette méthode affectera donc la puissance de la roue gauche et de la roue droite avec les deux paramètres donnés.

Del

Cette méthode permet de contrôler les deux leds qui sont présentes sur notre robot afin de visualiser lorsque les moteurs tournent et de quel sens ceux-ci sont alimentés. Nous avons besoin du constructeur de la classe del ainsi que d'une méthode :

- Del()

Le constructeur de la classe del. Ce constructeur permet seulement de mettre le port D pour les pin 0,1, 2 et 7 en sortie.

- changerCouleur(Couleur couleur, IdDel idDel)

Méthode qui permet de changer la couleur d'une del. L'id que nous fournissons nous dit quelle del nous voulons modifier. Nous pouvons donc choisir de changer la led gauche ou bien la led droite.

Écran LCD

Puisque le Code pour l'écran LCD nous est donné, nous n'avons pas conçu ces algorithmes et le code.

- Lcd operator<<

Pour afficher quelque chose sur l'écran, il suffit d'utiliser l'opérateur << avec l'objet lcd pour afficher la chaine que l'on veut sur l'écran.

Main

Le main est la partie du projet qui sera exécutée. Il nous faut donc appeler toutes les méthodes des différentes classes afin de contrôler les différentes composantes de notre robot.
Premièrement, nous avons défini toutes les constantes que nous allions utiliser dans ce programme afin d'éviter les « chiffres magiques ». Ensuite, nous avons créé des objets simples pour chaque classe que nous allions utiliser (ex : créer un bouton, une del, etc…).
Après ces déclarations, nous avons enfin entamer la partie exécutable de notre robot.
La partie en mode détection est assez simple. Le robot doit regarder les sonars et déterminer dans quel état il est. De ce fait, il nous suffit d'afficher cet état sur l'écran LCD. Lorsque le bouton est enfoncé, le robot quitte ce mode. Cela veut dire que nous aurons forcément un : while(!bouton), qui veut dire que le robot reste en mode détection tant que le bouton n'est pas enfoncé.
La partie en mode manœuvre est un peu plus complexe. En effet, il nous faut déterminer dans quel cas le robot se trouve et quelle sera la manœuvre appropriée. Cette étape peut se faire facilement avec un switch case. Ensuite, il nous faut alimenter les roues du robot de façon à exécuter la manœuvre, tout en affichant la puissance en temps réel qui sera envoyée aux roues. De plus, il faut afficher quelle manœuvre est exécutée sur l'écran LCD.
Finalement, lorsque la manœuvre est terminée, on retourne en mode détection.

2. Expérience de travail à distance

Nous avons trouvé qu'il était quelquefois plus difficile de se coordonner à distance et de comprendre le travail fait par les autres. On peut alors voir que le travail n'avance pas aussi rapidement, car nous devons passer beaucoup plus de temps à expliquer ce que les autres n'ont pas bien compris. De ce fait, la charge de travail est un peu plus grosse. En revanche, le projet était très bien réalisable en ligne, même si nous aurions préféré avoir un vrai robot physique. En effet, la présence physique d'un robot offre une expérience beaucoup plus concrète et nous trouvons qu'il serait beaucoup plus facile de déboguer et de trouver nos erreurs ainsi.

Aussi, il est plus facile d'expliquer nos questions et nos interrogations au chargés de laboratoire en présence physique, mais la plateforme « Discord » étant assez versatile, nous avons réussi à utiliser le partage d'écran pour expliquer de la façon la plus optimale possible nos questions et notre avancement sur le robot.

De plus, l'absence de robot matériel et l'obligation de rester au domicile nous a permis de mettre en place de nouvelles manières de travailler tous ensemble. Nous pensons que cela à grandement augmenter la cohésion dans l'équipe.