POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF1900
Projet initial de système embarqué

Rapport final de projet

Projet dans SimulIDE

Équipe No < **204208** >

Section de laboratoire < **5** >

< Nicolas Deloumeau, Laurent Langlois, Hisham Boulifa, Thomas Duperron >

22 avril 2021

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

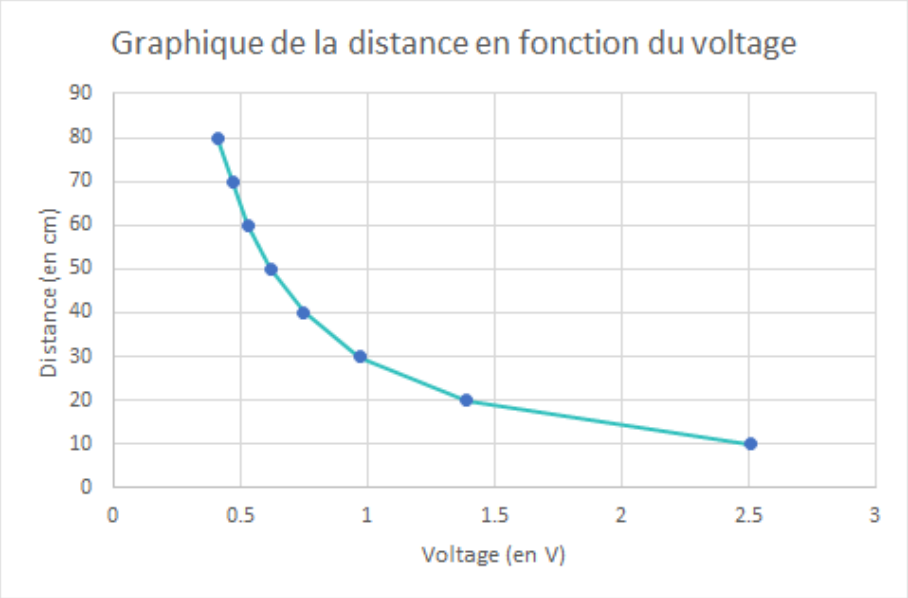
Pour commencer, le projet final comprend deux dossiers : *librairie* et *projet*.
D'une part, le dossier *librairie* contient tous les fichiers .cpp et .h ainsi qu'un Makefile réalisé tout au long de la progression de notre projet. Les fichiers contenus dans la librairie permettent le bon fonctionnement de notre robot. Le code a été écrit pour le projet final mais aussi pour les tp précédents.
D'une autre part, le dossier *projet* possède 3 fichiers .cpp (clavier3x3.cpp, septSegment.cpp et main.cpp), 2 fichiers .h (clavier3x3.h, septSegment.h et en relation avec leur fichier .cpp respectif), d'un Makefile ainsi que d'un fichier LISEZ-MOI.txt puis du fichier *circuit.simu* à ouvrir avec le logiciel SimulIDE (contenant le circuit de notre projet). Le code est spécifique au robot.

Makefile
Pour le projet final, le Makefile contenu dans le dossier du tp7 a été repris et modifié à quelques niveaux afin de l'adapter à notre projet.
Tout d'abord, le nom du projet (PROJECTNAME) a été changé à *projet*.
De plus, nous avons fait des modifications au niveau de PRJSRC, permettant d'inclure les fichiers désirés en source. En effet, en plus du main.cpp, les fichiers clavier3x3.cpp et septSegment.cpp sont inclus.
Nous y avons aussi ajouté une commande *debug* afin de pouvoir déboguer notre code dans le but vérifier s'il s'y trouve des failles et de les fixer (si c'est le cas).

Clavier
Implémentée sous le nom *clavier3x3* (dans les fichiers clavier3x3.cpp et .h), cette classe permet de savoir la position exacte du bouton appuyé du clavier à neufs boutons. Pour cela, il faut savoir si une des touches d'une colonnes a été pressée, ensuite, en inversant les colonnes et les rangées, nous pouvons obtenir la position du bouton dans le clavier.

Capteur de distance (ou détection)
Nous avons ajouté le capteur de distance dans notre projet dans le *main* (implémenté au début de la boucle infinie). Ce dernier est relié au convertisseur analogique-numérique dont la classe *Can* est comprise dans le dossier librairie.
Tout d'abord, le capteur retourne un certain voltage. Par la suite, grâce à un multiplexeur, un des trois capteurs GP2D12 est sélectionné afin d'être lu. Le voltage est alors lu par un convertisseur analogique-numérique interne ou externe dépendamment du choix de l'utilisateur. Ce choix peut être changé durant l'exécution selon la commande de l'utilisateur. Enfin, dépendamment du voltage, une distance est déterminée. Donc, une distance est associée à chaque voltage. Rappelons que le convertisseur analogique-numérique a pour but de lire la valeur analogique des capteurs de distance et de la convertir en valeur numérique.

Le graphique ci-dessous nous permet de visualiser la distance que nous pouvons obtenir en dépendamment du voltage retourné par les capteurs de distance. Nous constatons que moins le voltage est élevé, plus la distance est grande.



Afficheur 7 segments

Dans les fichiers *septSegment* .h et .cpp (contenu dans le dossier projet) sont implémentées quatre classes correspondant aux quatre afficheurs. Les fonctions comprises sont presque identiques, le seul élément devant changer est la pin connectée à l'entrée IE du décodeur BCD, allant de gauche à droite (dans l'ordre : B5, B6, C0, C1). Quand l'entrée IE est sous tension, la valeur comprise entre A4 et A7 est mise en mémoire dans le décodeur BCD correspondant.

Bouton-poussoir

La classe du bouton-poussoir (dans le fichier main) a pour but de changer un bool à true afin de savoir quand il faut passer aux manœuvres.

Au niveau du circuit, lorsque le bouton-poussoir est appuyé, cela change la pin D2 à 0. Cette dernière est donc toujours alimentée jusqu'à ce que l'on appuie sur le bouton-poussoir, ce qui va couper l'alimentation.

Moteurs (manoeuvres)

Dans le mode manœuvre, les moteurs seront contrôlés afin de pouvoir effectuer des manœuvres dans le but de ne pas être en situation de danger (lorsqu'il y a un obstacle).

Dans les fichiers *bimoteurs* (contenus dans la librairie), nous avons implémenté deux fonctions nommées *ajustementGauche* et *ajustementDroite* ainsi que deux "directions" : avancer et reculer. Les fonctions prennent deux mêmes paramètres étant la vitesse et la direction.

Pour être plus précis, sachant que l'on utilise ici la classe bimoteur, cela veut dire que la classe se base sur un timer et contrôle deux PWM différents. Donc, chaque roue a une vitesse indépendante en utilisant un seul timer.

Main

- Suite à l'inclusion des différents fichiers nécessaires à notre *main*, un initialisation des variables est effectuée afin de pouvoir les utiliser au cours de notre code.

- Dans la fonction *initialisation()*, comme son nom l'indique, nous initialisons les *interrupts* afin de pouvoir utiliser les ISR (ici, le bouton et le clavier à neuf boutons).

- Après avoir initialisé les *interrupts*, nous pouvons finalement utiliser les ISR (ou Interrupt Service Routine). Dans le premier ISR, nous effectuons une lecture du bouton-poussoir (*boutonappuye*). Par après, dans le deuxième ISR, nous réalisons la lecture du clavier à neuf boutons par l'intermédiaire d'un switch case contenant 9 cases, un case pour chaque bouton possible, et un default, contenant juste un *break*.

- A posteriori, dans le main, les lignes se trouvant avant le commentaire *Demarrage* servent à initialiser les différentes variables dont nous allons faire usage. D'ailleurs, nous appelons la fonction *intialisation()*.

- Dans le main se trouve le commentaire *Demarrage*, indiquant donc le démarrage. Sous ce commentaire, l'affichage des lettres A, B, C et D est exécuté. Ensuite, le code indique aux roues de tourner en sens horaire puis anti-horaire.

- Nous avons implémenté dans le main une boucle infinie (for (;;)) dans laquelle se trouvent la détection (décrite dans un paragraphe plus haut) et un switch case concernant *manoeuvre* est appelée si le bouton-poussoir est appuyé (donc une condition if(boutonappuye)).
  Il y a six cases définis dans le switch case. Dans le premier case (case 0), manoeuvre n'est pas demandé, ce case est utilisé lorsqu'il y a un combinaison non reconnue de distances.
  Les cinq cases suivantes correspondent quant à elles les cinq manœuvres. Donc, les cases 1 à 5 représentent de manière respective les manœuvres 1 à 5.

2. Expérience de travail à distance

Vu le contexte actuel, il peut sembler difficile d'imaginer la réalisation d'un projet d'équipe sans être en présentiel. Tout d'abord, cela aurait pu avoir un impact sur la communication, mais aussi sur la qualité de notre travail. En l'occurrence, hormis le manque d'aller à l'université afin de pouvoir se regrouper pour travailler (ou autre), nous pouvons affirmer que nous avons pu réaliser notre projet tout en nous acclimatant aux règles sanitaires.

Pour commencer, le fait de disposer du logiciel SimulIDE nous a permis de pouvoir réaliser un robot virtuel, ce qui nous a permis de comprendre comme si on était en face de notre robot (ou presque). Aussi, SimulIDE nous a permis de pouvoir accéder au robot en tout temps, ce qui peut être un avantage. Effectivement, lorsqu'une partie de code était ajoutée, nous pouvions directement la tester sur le robot. Ensuite, la communication s'est principalement déroulée sur Discord afin de pouvoir parler entre nous, puis sur Git pour pouvoir push le code réalisé et ce pour que chaque membre de l'équipe soit à jour.
Lors de nos séances de travail, nous codions ensemble sur Visual Studio Code grâce à la fonctionnalité Live Share, ce qui permet un accès aux fichiers communs de manière simultanée.

Toutefois, il nous arrivait de temps à autre d'avoir des soucis techniques. Il en existe d'ailleurs plusieurs qui peuvent plus ou moins nuire au rythme de travail comme une rupture de réseau (notamment avec le Wi-Fi), SimulIDE qui peut rejeter un thème Linux ou encore les bogues sur Discord (qui a tendance à s'arrêter soudainement sur Linux).

Finalement, en ce qui concerne notre équipe, nous avons tout de même réussi à bien communiquer et de mettre au clair les différents points dans le but d'avancer dans ce premier projet. La communication était une des bases fondamentales de notre projet ce qui fait que l'on a pu s'entendre sur les décisions à prendre par rapport au travail.