Département de génie informatique et génie logiciel

INF1900
Projet initial de système embarqué

Rapport final de projet

Simulation du comportement du robot

Équipe No < **63121** >

Section de laboratoire < **3** >

< Dana Louka, Emile Lefebvre, Oussama Ressak, Louis Dutheil >

17 Avril 2020

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

Main, inclusions et définitions :

Notre fichier main.cpp qui exécute notre programme comporte les inclusions des quelques fichiers trouvables dans notre librairie ou sont des en-têtes de la bibliothèque standard du C. Ces fichiers comportent des fonctions utiles pour initialiser des registres, faire des calculs, afficher des informations et d'autres fonctionnalités. Pour les définitions, nous avons défini la fréquence de l'horloge et nous avons initialisé deux variables volatiles globales importantes. *mode* et *compteur* sont deux variables qui changent dans les ISR du bouton-poussoir et de la minuterie. *mode* est responsable de changer entre le mode de détection et le mode de manœuvres. c*ompteur* est utile pour les sonars.

Dans la fonction main, nous avons une boucle qui roule à l'infini. Dans cette boucle, nous avons choisi d'utiliser un switch-case qui permet de changer entre les deux modes de fonctionnement.

Afficheur 7 segments :

Le robot comporte 5 afficheurs 7 segments, leur rôle est d'afficher l'intensité des moteurs en temps réel lors des manœuvres. La particularité de l'afficheur 7 segments est que l'on ne peut afficher qu'un chiffre à la fois. Ainsi, pour donner l'impression que plusieurs chiffres s'affichent au même moment, nous devrons donner une différence de potentiel sur un très court lapse de temps aux cinq ports positions. Cette différence de potentiel sera répétée en boucle pour combler le temps demandé. Bref, les principales fonctions créées pour contrôler l'afficheur sont les suivantes :

enum valeurChiffre :

Table de traduction entre les 10 chiffres et la valeur hexadécimale d'afficheur 7 segments

determinerChiffre(uint8_t chiffre**) :**

Transforme le *chiffre* reçu pour retourner une valeur hexadécimale correspondant à ce chiffre selon l'affichage 7 segments. On peut utiliser un chiffre à la place de l'énumération valeurChiffre; allège ainsi le nombre de paramètres dans la fonction afficherSequenceChiffres (cinq paramètres vs neuf paramètres).

affichageChiffreSurLapseTemps(unsigned int *position,* **uint8_t** *chiffre,* **uint8_t** *temps***) :**

Affiche le *chiffre* sur l'afficheur ciblé à une certaine *position* pendant une durée *temps* en ms.

afficherSequenceChiffres (uint8_t *temps*, **uint8_t** *chiffre1*, **uint8_t** *chiffre2*, **uint8_t** *chiffre3*, **uint8_t** *chiffre4*) **:**

Affiche la séquence composée des *chiffres* dans l'ordre sur les afficheurs 7 segments pendant une durée *temps* en ms. Les valeurs *chiffres* sont des chiffres déjà « transformés » pour correspondre au « langage 7 segments ».

void passageVitesse(double ratio, **uint8_t** *premierNombre*, **uint8_t** *deuxiemeNombre*, **uint8_t** *troisiemeNombre*, **uint8_t** *quatriemeNombre*) **:**

Fonction qui gère l'affichage des vitesses des moteurs lors de transitions entre 2 vitesses pour chaque moteur selon un *ratio.* Le premier et troisième nombre agissent comme nombres de départ alors que le deuxième et quatirième nombre sont respectivement ceux que l'on veut afficher.

Mode détection :

Lorsqu'il n'a pas à exécuter des manœuvres, le robot est en permanence en mode détection. Le principe de ce mode est de constamment évaluer les distances qui séparent le robot et d'autres objets, tout cela à l'aide des sonars droite, gauche et devant. Comme écrit dans l'énoncé, nos sonars fonctionnent selon la séquence suivante :

Émission d'une onde des trois sonars -> Réception de l'écho par un seul des sonars.

Le radar receveur change, donc au bout de 3 cycle chaque sonar a reçu une information.

determinerCatégorie (int *distance*, **char*** *outStr*) **:**

Fonction qui détermine le niveau de danger du robot à partir d'une *distance.* La fonction modifie en retour un tableau de caractères (*outStr)* pour qu'il contienne le message OK, ATTN ou DNGR à afficher sur l'écran LCM.

determinerManoeuvre(int *distanceGauche*, **int** *distanceAvant,* **int** *distanceDroite*) **:**

Détermine la manœuvre à exécuter selon les *distances* passées en paramètres, si la combinaison des distances ne correspond pas à une manœuvre, la fonction retourne 0.
La fonction est couplée à **afficherManoeuvre** pour afficher la manœuvre.

affichageSensorDetection(int *distance*, **int** *dixiemeDeDistance*, **LCM** *ecran*,

int *deplacement***) :**

Fonction qui affiche une *distance* précise à 0.1m sur un *ecran* LCM.

modeDetection() :

Fonction qui opère les 3 sonars de façon continue. Elle utilise ensuite la fonction **determinerCategorie** et **affichageSensorDetection** pour gérer l'affichage des distances sur l'écran LCD en temps réel**.** Si l'utilisateur presse le bouton, alors la fonction retourne un int qui sera « traduit » en une manœuvre par **modeManoeuvre**.

Manœuvres :

Une fois rentré dans le mode manœuvre, le robot va exécuter une suite d'action qui lui permettront de s'extraire de sa position dangereuse. Lors de ces manœuvres, les afficheurs 7 segments affichent l'intensité de chaque moteur en temps réel et l'écran nous informe du numéro de la manœuvre en cours. Cette partie comporte moins de fonctions dédiées puisqu'elle utilise principalement des fonctions dont on a déjà parlé plus haut ou dans le rapport de la librairie.

modeManoeuvre(int *numeroManoe***) :**

Lance le mode manœuvre à partir d'un numéro de manœuvre, elle utilise **afficherManoeuvre** ainsi qu'un switch-case pour choisir la manœuvre à effectuer.

afficherManoeuvre(int *numeroManoeuvre***) :**

Affiche sur l'écran LCD le numéro de la manœuvre, si *numeroManoeuvre* = 0 affiche « Combinaison non évaluée »

manoeuvre1(), **manoeuvre2**(), **manoeuvre3**(), **manoeuvre4**(), **manoeuvre5**(), **manoeuvre6**() :
Ces méthodes contiennent le code des différentes manœuvres qui seront effectuées par le robot. Pour pouvoir effectuer les délais nécessaires, il fallait prendre en compte les délais effectués dans l'afficheur 7 segments qui était de 25ms par cycle d'affichage. De sorte, il fallait diviser nos délais voulu par 25 afin d'obtenir la valeur qu'il faudra passer en paramètre à la méthode **afficherSequenceChiffres** qui va boucler l'affichage par le nombre que l'on a trouvé afin d'obtenir le délai souhaité.

2. Expérience de travail à distance

Au début du travail nous avons mis en place un serveur Discord afin de pouvoir communiquer facilement (appels, partages d'écran). On s'est partagé le travail puis on a choisi un mode de travail centralisé : trois étudiants s'occupent de rédiger le code, le dernier s'occupe du rapport.

Les conditions ont été vraiment particulières, entre les problèmes de connexion, les difficultés à communiquer et les différents rythmes au sein du groupe (certains préfèrent travailler le soir, d'autres le matin). Fedora a aussi apporté son lot de difficultés pour certains d'entre nous. Une semaine après le partage des tâches, nous n'avions utilisé le discord qu'une fois.

Ne pas se voir en vrai a vraiment compliqué certains points, comme les membres du groupe ne se connaissent pas encore très bien, on n'osait pas vraiment déranger l'autre pour lui poser une question sur son code, l'équipe se trouvant vraiment dans une phase de forming comme vu en HPR.

Le problème étant qu'en interagissant seulement à distance, la situation évolue lentement et le groupe se retrouve avec un travail d'équipe moins performant que si l'on avait pu se voir.

Cependant nous avons fini par nous fixer des horaires de communication sur discord ce qui a permis d'échanger et de trouver des solutions ensemble à nos problèmes, ces horaires ont permis de répondre aux questionnements laissés en suspens par peur de déranger.