Polytechnique Montréal

Département de génie informatique et génie logiciel

Cours INF1900:

Projet initial de système embarqué

Travail pratique 7

Makefile et production de librairie statique

Par l'équipe

No 39-40

Noms:

Ouassim Ouali

Kevin Wassef

Anass Laaz

Alexandre Ramtoula

Date:

22 octobre 2019

Partie 1 : Description de la librairie

Décrire la librairie construite et formée (définitions, fonctions ou classes, utilité, etc...) pour que cette partie du travail soient bien documentées pour la suite du projet pour le bénéfice de tous les membres de l'équipe.

Notre librairie nous permet d'utiliser des codes déjà prêts que nous avons codés et utilisés lors de nos 6 premiers travaux pratiques, par exemple le générateur des signaux PWM pour les roues. Ainsi nous aurons accès à des classes déjà prêtes qui vont nous aider pour formater le robot pour le parcours final de la session. Cette librairie contient alors principalement des fichiers de type .cpp et certains possèdes des fichiers de type .h qui leurs correspondent. De plus, nous avons un makefile qui nous permet de construire une librairie en .a, une librairie statique. On peut donc retrouver les fichiers générés par le makefile (.o, .d , .a).

Voici les différentes classes comprises dans la librairie et les descriptions des méthodes qu'elles utilisent que nous avons écrites nous-mêmes :

La classe memoire_24 contient les méthodes utilisées permettant un accès en lecture et en écriture des éléments de la mémoire externes. Cette classe permet de recevoir des données du robot, ce qui permettra dans le futur de procéder à un débogage de la mémoire et des autres fonctionnalités.

La classe can permet de prendre une source analogique et de la convertir en source numérique. Par exemple, capter l'intensité lumineuse grâce à une photorésistance puis effectuer une conversion analogique - numérique afin que le programme soit en mesure de traiter cette information (celui-ci fonctionnant exclusivement de manière numérique). Cette classe est utilisée dans la méthode detecterLumiere() ci-dessous.

La méthode detectionLumiere permet de recevoir un signal lumineux puis de convertir ce signal analogique en des valeurs numériques qui pourront être utilisées par le programme. Ce programme permet d'activer la DEL et d'en obtenir une réponse différente (soit une couleur qui variera) en fonction de l'intensité de la lumière ressentie par la photorésistance.

La méthode PWMDEL() permet de générer un signal PWM sur la DEL.

La méthode my-delay() prend en paramètre un nombre de millisecondes sous forme de uint16_t qu'on pourra par la suite utiliser dans d'autres fonctions.

La méthode boucle60Hz() prend en paramètre (uint32_t a, uint32_1 b) afin de générer un signal PWM de 60 Hertz selon les valeurs de a et b.

La méthode boucle400Hz() prend en paramètre uint32_t b et uint32_1 b pour générer un signal PWM de 400 Hertz selon a et b.

La méthode PWM-roue() permet de démarrer les roues selon deux signaux PWM distincts.

La méthode void initialisation() mets les PORTS D et A en sortie.

La méthode void ajustementPWM prend les paramètres (float vitesse1, float vitesse2) afin de créer deux signaux PWM qui vont êtres utilisés par les autres fonctions afin de faire tourner les roues.

La méthode void accelerer() fait accélérer les roues à partir d'une vitesse nulle en augmentant la vitesse de 25% toutes les 2 secondes jusqu'à atteindre une vitesse maximale, puis retourne à la vitesse nulle.

La méthode void reculerVitesseConst(float vitesse) mets les bits 6 et 7 (les deux derniers) a 1 pour changer le sens de rotation et après fait tourner les roues a l'aide de la méthode ajustementPWM().

La méthode void avancerVitesseConst(float vitesse) fait avancer le robot avec une vitesse constante.

La méthode transmissionChaine() utilise la classe mémoire pour lire un message stocké dans la mémoire externe du robot puis la transmet vers le PC à l'aide de la connexion UART

La méthode afficherChaine() lit une chaine de caractères stockés dans la mémoire externe.

La méthode transmissionUART() prend uint_8 données et transmet celle-ci à l'ordinateur.

La méthode InitialisationUART() permet d'initialiser l'UART.

Partie 2 : Décrire les modifications apportées au Makefile de départ

Décrire les quelques modifications apportées au Makefile de la librairie pour démontrer votre compréhension de la formation des fichiers. Faire de même pour les modifications apportées au Makefile du code (bidon) de test qui utilise cette librairie.

Pour le makefile de la librairie

Nous avons changé le nom du projet <<PROJECTNAME>> pour <<librobot>> afin de terminer avec une librairie nommée librobot.

Nous nous sommes servis de la fonction PRJSRC = $(wildcard *.cpp) afin d'inclure tous les fichiers qui se terminent par les cpp dans le dossier concerné.

Concernant l'Implémentation de la cible avec le target TRG au lieu de mettre (PROJECTNAME).out, nous devons mettre les extensions .a car les librairies sont des fichiers utilisant l'extension .a. De plus, nous avons remplacé la création de l'exécutable à partir des .o par la fonction ar -crs $(TRG) $(OBDJEPS) qui créé une librairie à partir des objets créés par les cpp.

Pour la règle clean, nous avons ajouté au fichier la fonction de supprimer toutes les extensions .a en remplaçant *.out par *.a dans la ligne suivante <<`$(REMOVE) $(TRG) $(TRG).map $(OBJDEPS) $(HEXTRG) *.d *.out`>>.

Pour le makefile du code :

PRJSRC jouera le rôle de notre main.cpp puisqu'il y aura optimalement seulement un fichier main qui utilisera la librairie pour fonctionner.

Nous avons inclus la librairie statique qu'on a créée en implémentant INC = -I ../librobot.

LIBS= -L ../librobot/ -lrobot nous as permis de faire une liaison avec la librairie statique créée, dans notre cas librobot.a, d'où l'utilisation de -L ../-lrobot.