Département de génie informatique et génie logiciel

INF1900

Projet initial de système embarqué

Rapport final de projet

Simulation du comportement du robot

Équipe No 117 et 119

Section de laboratoire 01

Abderraouf Benchoubane, Samy Cheklat, William Ringuet,
Philippe Bonhomme

17 Avril 2020

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

Pour la conception de notre projet final, nous avons séparé notre code en trois parties. Nous avons la première partie qui est un dossier nommé *src* (source) qui contient notre *main* qui est le fichier principal de notre programme et qui contrôle le comportement de notre code. Un fichier *Makefile* permettant la compilation de notre *main* est aussi inclus dans ce dossier.

La deuxième partie est un dossier nommé *lib* (librairie) qui contient tous les fichiers *.cpp* et *.h* qui seront utilisés dans la programmation de notre *main*. Ce dossier inclut aussi son propre *Makefile* pour permettre la compilation de tous les fichiers contenus dans la librairie.

La troisième partie est un fichier *Makefile* qui regroupe les parties communes des deux autres *Makefile* contenus dans les dossiers *lib* et *src*. Globalement, notre code doit se servir des différents fichiers de la librairie que nous avons conçus pour pouvoir réaliser une série d'actions contrôlées dans notre *main*. Dans cette librairie, nous avons implémenté plusieurs classes mais parmi les plus importantes figurent le sonar, le bouton poussoir, l'affichage du sept segments, l'affichage de la *DEL,* la création du PWM ainsi que nos différentes manœuvres. Ces classes seront expliquées plus en profondeur plus loin dans ce rapport.

Concernant notre implémentation générale, notre robot fonctionne comme une machine à états. Notre robot a deux états : un état *détection* et un état *manœuvre*. Dépendamment, de l'entrée que notre robot reçoit, la sortie sera différente lorsqu'il sera dans l'état *manœuvre*. Il est possible de voir le fonctionnement du robot dans la figure 1 ci-dessous

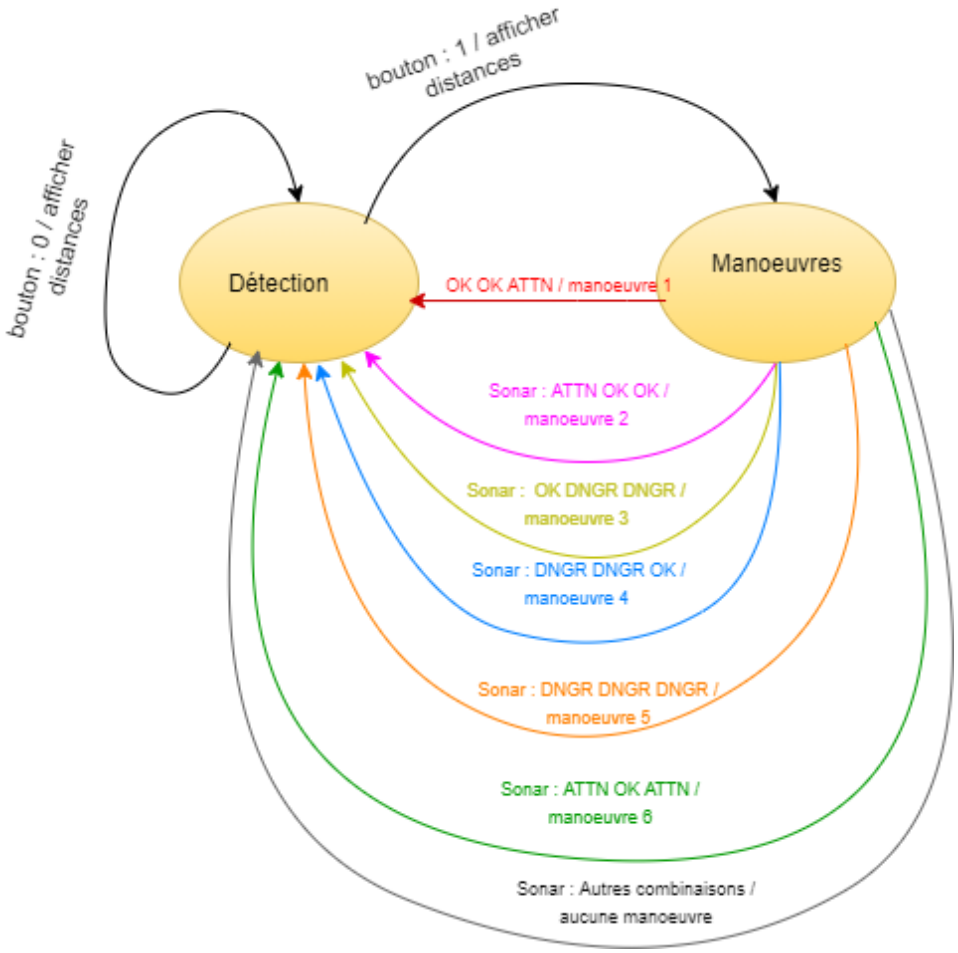

Figure 1 : Diagramme d'états du robot

Conception de la librairie

Sonar :

Cette classe permet de faire fonctionner les sonars, premièrement le constructeur permet d'initialiser les attributs de distances et de mettre les « pins » nécessaire, *PIN_B0* en sortie et les « pins » *PINA0*, *PINA1*, *PINA2* en entrées. Deuxièmement, la classe contient des méthodes qui déterminent la longueur des ondes échos, les procédures attendent la réception de l'écho dans les « pins » A. Une fois que le signal est reçu, un compteur est incrémenté chaque 100 uS permettant ainsi d'avoir la durée du signal écho. Troisièmement une méthode *determinerDistances()* permet de calculer les distances à partir de la longueur de l'écho associé à chaque sonar. Quatrièmement nous avons une méthode qui permet d'afficher les distances avec les commentaires associés. Cette méthode utilise une classe incluse dans notre librairie qui permet d'écrire et d'afficher des

caractères aux endroits voulus sur l'écran LCM. Les distances par rapport aux sonars ainsi que les messages d'avertissements "ATTN", "OK" et "DNGR" sont affichés grâce à cette classe. Pour finir, nous avons implémenté des méthodes qui retournent les valeurs des attributs de la classe (les distances) pour qu'elles puissent être récupérées et analysées par le *main* afin de déterminer quelle manœuvre devra être adoptée.

DEL:

Cette classe permet d'allumer ou de fermer les DEL à l'aide de méthodes. Son constructeur prend en paramètre le port où sont branché les DEL. Ensuite, on initialise les « pins » nécessaires en sortie *DDR*. La procédure *rougeDroit()* ajoute à l'aide d'un masque une valeur au port passé en paramètre. Cette valeur allume la DEL de droite en rouge. Pour ce faire, on doit s'assurer que la « pin » qui est associée la couleur verte de la DEL droite soit fermée (0), car la DEL ne peut pas afficher les deux couleurs en même temps. On met donc un 0 à la position de la « pin » associée à la DEL verte de droite. On utilise le même principe pour *vertDroit()*, *rougeGauche()* et *vertGauche()*. Pour la procédure *eteintDELS()*, on met toutes les « pins » du port où sont branchées les DEL à 0.

Bouton Poussoir:

Le constructeur de cette classe prend en paramètre la « pin » auquel du bouton et la valeur en hexadécimale de la position de la « pin » par rapport au port (celle-ci servira de masque dans la fonction *estAppuyer())*. Dans la fonction *estAppuyer()*, on compare la « pin » et le masque. Si le bouton est appuyé, la position du port associé au masque est à 0 de sortes qu'on inverse la valeur de retour de la comparaison avec un « *!* ». On refait une deuxième fois la comparaison après 10 ms (*anti-rebond*). On retourne la valeur « *true* » si les conditions sont vraies, dans le cas contraire on retourne « *false* ». La procédure *initialisationInteruption()* initialise les registres pour effectuer des interruptions si on appuie lorsque l'on appuie sur le bouton.

PWM

Pour le *PWM*, une seule méthode nommée *ajustementPWM()* sera utilisée. Cette méthode initialise les registres ainsi que la minuterie 1 et nous permet d'ajuster la vitesse du robot. Cela se fait à l'aide d'une formule qui se base sur la valeur maximale du registre *OCR1A et OCR1B*. Elle prend en paramètre le pourcentage de chaque roue.

Manoeuvre:

Cet ensemble de fonctions permet de décrire les actions dans chaque manœuvre. En premier lieu, on a une fonction qui initialise les ports pour les manœuvres intitulée *initaliserManoeuvre().* Celle-ci permet d'initialiser l'afficheur sept segments ainsi que la minuterie deux. Ensuite nous avons implémenté chaque manœuvre selon leur description. Finalement la fonction *clearPort()* permet d'arrêter la minuterie pour les afficheurs sept segments, et de mettre les ports associés aux manœuvres en entrée pour les arrêter. Enfin, les méthodes *manoeuvreX()* appellent les diverses méthodes, comme *ajustementPWM()*, *convertisseurVersSeg()* qui permettent de créer les manœuvres en questions.

Sept segments:

Cet ensemble de fonctions permet de décider du comportement de l'afficheur à sept segments. Cette classe a pour but d'afficher les vitesses des roues de chaque côté du robot en indiquant leur pourcentage de puissance (PWM). D'abord, nous avons la méthode *initseptsegment()* qui initialise les ports d'alimentation (*PIN_A3 à PIN_A7*) des sept segments en sortie, ainsi que le port de données en sortie (*PIN_C0 à PIN_C7*). Ensuite, nous avons implémenté une fonction qui permet d'associer le segment à la valeur à afficher nommée *transitionPourcentage().* En continuant, nous avons la fonction *convertisseurversseptseg()* qui assigne la valeur à afficher au tableau de segments. Finalement les fonctions *initminutriedeux()* et *arreterminuteriedeux()* contrôlent la minuterie pour l'affichage des sept segments. Ces fonctions délimitent la durée de la période d'utilisation de notre afficheur sept segments.

2. Expérience de travail à distance

Le travail à distance s'est avéré un peu plus complexe que ce que nous pensions, car nous n'étions pas toujours disponibles aux mêmes moments. De plus, il y avait quelques « *bugs* » dans SimuliDE qui nous ont ralentis. Cette expérience fut tout de même très enrichissante et nous avons appris de nouvelles manières de travailler en équipe. Nous avons développé de nouvelles méthodes de travail d'équipe et d'organisation. Nous devions être rigoureux dans notre planification et nous devions aussi être en mesure de gérer un grand projet avec peu de soutien et d'informations. Le fait de travailler en ligne avec des vidéoconférences pouvait parfois ralentir le processus mais malgré tout, ce fut une expérience très plaisante. L'utilisation du logiciel Discord fut très utile et bénéfique pour la productivité de l'équipe. Ce fut une expérience assez spéciale puisque le projet était à la base quelque chose de physique mais les chargés et les responsables du cours ont réussi à sauver la situation avec un travail qui fut fort bien amusant au final.