Dossier technique
Coopération de robots

Thibault Vernay, Victor Cazaux, Daphné Porteries

25 juin 2017

Table des matières

1 Spécifications

La spécification qui suit est la spécification que nous avons établi au début du projet. Pas toutes les contraintes spécifiées ont été satisfaites et pas tous les capteurs ont été utilisés mais celle-ci donne une vue globale de ce projet.

L'objectif de ce projet est de créer deux robots capables de coopérer pour venir à bout d'une épreuve définie qui oblige une coopération.

Le robot doit être capable de :

— se déplacer ;
— se repérer dans l'espace ;
— communiquer avec un coéquipier ;
— prendre une décision en concertation avec son coéquipier.

1.1 Spécification de l'épreuve

Chaque robot doit :

— se repérer et repérer son coéquipier ;
— rejoindre son coéquipier ;
— traverser avec son coéquipier la zone de passage afin de valider l'épreuve.

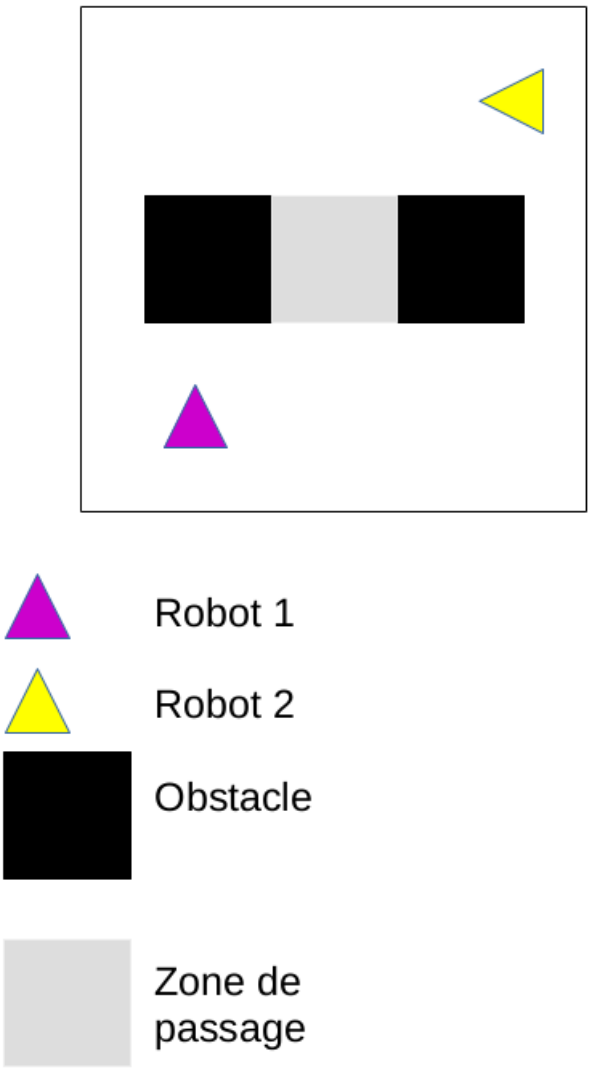


FIGURE 1 – terrain de jeu 80*80 cm

1.2 Spécifications matérielles

Le châssis des robots sont fournis au projet, ainsi que les composants et capteurs. Les robots doivent être identiques en conception matérielle.

1.2.1 Micro-contrôleur

Pour ce projet nous utilisons deux cartes STM32F303K8 comme micro-contrôleurs pour nos robots, ces cartes permettent un encombrement minimal sur le robot et fournissent suffisamment d'entrées-sorties pour nos différents capteurs.

1.2.2 Déplacement

Le déplacement est assuré par des moteurs à courant continus. Un driver moteur fait l'interface de puissance. Pour pivoter il est également important de connaître son orientation. En effet sans orientation

connue notre robot est sensible aux perturbations externes. Nous utilisons donc ici un magnétomètre pour connaître l'orientation de notre robot.

1.2.3 Repérage dans l'espace

Ici deux capteurs SHARP sont utilisés pour mesurer les distances. L'un d'entre eux est fixé sur un servomoteur fixé sur le châssis pour pouvoir balayer l'environnement et l'autre sera fixe, dirigé vers l'avant du robot.

1.2.4 Communication

Pour la communication, le protocole UART sera utilisé mais la communication filaire est trop contraignante sur une épreuve (risques d'accrocher les fils dans le décors et risque de perturbations avec la longueur du fil) ce sont donc deux modules radio à sens unique qui seront utilisés.

1.3 Spécifications logicielles

La conception logicielle est séparée en trois parties :

— une couche bas niveau avec des fonctions permettant de déplacer le robot et de récupérer les valeurs des capteurs ;
— un protocole de communication gérant le transfert de données de manière fiable ;
— une couche haut niveau définissant le comportement du robot.

La couche haut niveau se servira des fonctions des deux autres niveaux pour définir des réactions pour le robots en fonction des données matérielles.

1.3.1 Couche bas niveau

Le but de cette couche est de créer des fonctions élémentaires de déplacement à l'aide de capteurs et des timers pour avancer et tourner, ainsi qu'une fonction renvoyant une structure contenant les valeurs des capteurs la plus à jour possible (100 fois par secondes).

Fonctions élémentaires pour le déplacement :

— avancer de 15 cm ;
— reculer de 15 cm ;
— pivoter de 90 degrés à droite ;
— pivoter de 90 degrés à gauche.

1.3.2 Couche communication

Cette couche doit gérer indépendamment du reste la communication entre les deux robots, elle définie donc la taille des trames envoyées et le protocole de sécurité des données pour vérifier l'intégrité de celui-ci. Il y a plusieurs choix technologiques possibles que nous devons tester en fonction de la longueur de la trame :

— XOR ;
— bit de parité ;
— checksum.

1.3.3 Couche IA

Cette couche définie la connaissance de l'épreuve du robot ainsi que la façon que celui-ci aura de quadriller son environnement. Cette couche utilise toutes les fonctions des couches inférieure et donc se positionne en couche de haut niveau. Elle permet de créer les déplacement du robots en fonctions de son environnement, de définir comment celui-ci va repérer où est le deuxième robot et également quelle méthode de prise de décision ceux-ci vont utiliser.

Quadrillage de la zone :

On utilise le fait que les fonctions élémentaires font avancer de 10 cm pour quadriller la zone en une multitude de carrés de 10*10 cm^2 pour établir une carte de l'épreuve.

Repérage du deuxième robot :

Les robots vont se déplacer en balayant la zone en cherchant un déplacement anormal qui ne soit pas de 10 cm, s'ils ne se trouvent toujours pas, il vont tourner pour chercher dans une autre direction.

Prise de décision :

Pour la prise de décision les robots vont utiliser un tirage pseudo aléatoire et comparer les numéros obtenus, celui qui a le numéro le plus grand deviens prioritaire et indique à l'autre qu'il est prioritaire.

2 Étude du matériel

2.1 Magnétomètre 3 axes

Le magnétomètre que nous aurions aimé avoir est le MAG3110FCR1 qui est un composant nécessitant la création d'une carte d'accueil avec des condensateurs de découplages pour les alimentations (dont les valeurs sont spécifiées dans la documentation du composant). Ce composant utilise le protocole I^2C qui permet d'utiliser que deux lignes, la ligne SDA et la ligne SCL réciproquement pour les données et pour l'horloge. Le composant en I^2C dispose d'une adresse sur 7 bits aligné à gauche à laquelle il faut ajouté le bit d'écriture ou de lecture pour en faire une information sur 8 bits. En plus de l'alimentation et des lignes destinées à l'I^2C ce capteur possède une ligne destinée aux interruptions.

Figure 2 – Schématique du composant

Ce capteur utilise l'effet Hall, mais nous n'avons pas vraiment eu le temps d'en parfaire l'étude.

2.2 Moteurs à courant continus et pont en H

Pour contrôler nos deux moteurs à courant continu nous avons à disposition un driver moteur qui possède pour chaque moteur deux entrées de sens et une entrée

2.3 Capteur de proximité infrarouge

Le capteur de proximité infrarouge est un capteur SHARP. C'est un capteur de mesure de distance, il intègre la combinaison d'un détecteur de position et une led à émissions infrarouge) et un circuit de traitement de signal. Ce capteur exploite la réflexion de l'objet pour calculer la distance. Ce capteur est difficilement influencé par la variation de température ce qu'il fait qu'il a une courbe d'étalonnage pour la distance qui ne change que très peu.

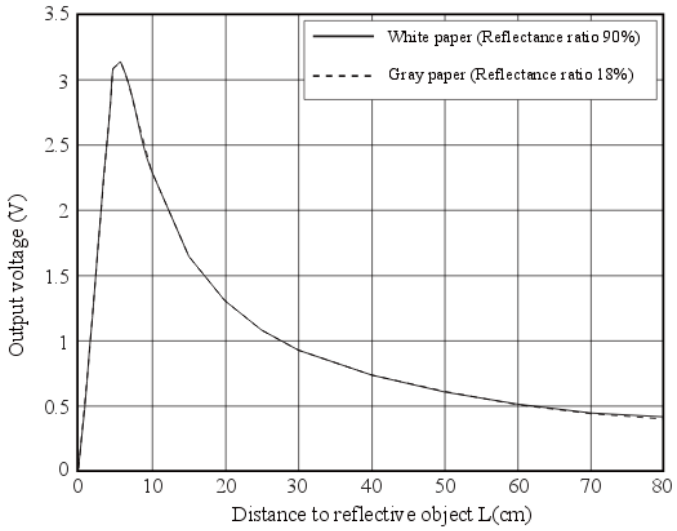


FIGURE 3 – Courbe d'étalonnage de distance du capteur SHARP

3 Étude du logiciel

3.1 Couche bas niveau

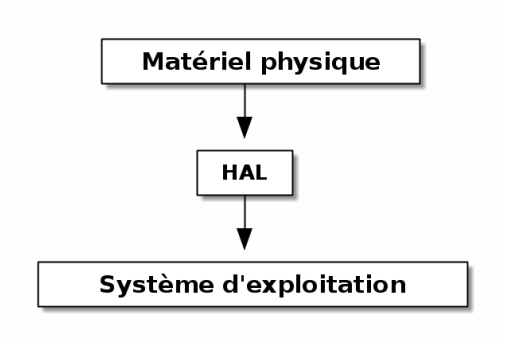
3.1.1 Configuration de STM32CubeMx

Lors de l'utilisation d'une carte microcontrôleur telle que la STM8S ou la Nucléo STM32F3xx, il est nécessaire de configurer les ports, les pins, les différents registres des timers et des ADC avant même de commencer le code à proprement parlé. Dans le cas d'une carte STM32, il existe un outils très pratique nommé STM32CubeMX permettant de configurer préalablement les bons registres, ports et pins et de générer un ensemble de fichiers projet avec un Makefile adapté au projet.

Les fichiers s'organisent de la manière suivante :

— fichier "main" contenant l'initialisation de la carte ;
— fichiers contenant les fonctions de la "HAL" pour chaque module de la carte.

À chaque génération de projet, STM32CubeMx va inclure les bonnes bibliothèques au main et au Makefile. Par exemple si j'allume l'UART2 avec la configuration STM32CubeMX, le fichier stm32f3xx_hal_uart.h sera inclus et son .c associé sera compilé. La "HAL" (Hardware Abstraction Layout) est comme son nom l'indique une couche d'abstraction matérielle qui va se charger de mettre les bonnes valeurs dans les bons registres lors de l'initialisation mais également lors de l'utilisation. Elle embarque donc un certain nombre de fonctions générique de manipulation du matériel facilitant l'utilisation de la carte. La mise en place de la HAL s'illustre de la manière suivante :



STM32CubeMx va aussi permettre autre chose : la mise en place d'un OS temps réel nommé FreeRTOS permettant de paralléliser les tâches. Ainsi nous allons créer plusieurs tâches avec cet OS et profiter

de sa gestion simplifiée des temporisations. Il faudra également gérer le HEAP (le tas) et la taille de STACK (la pile) lors de la mise en place de celui-ci. La "STACK" est la taille dans la mémoire maximum que peu prendre le programme lors de son exécution tandis que le "HEAP" existe lors de cas d'allocations dynamiques de la mémoire. Dans le cas de FreeRTOS toutes tâche créée est allouée dynamiquement dans la mémoire en fonction de la taille de sa Stack notamment. Il faut donc penser à avoir de la marge côté Heap et tester quelles valeurs maximale la carte accepte en fonction de la taille de sa RAM. La valeur max n'est pas mentionnée dans la doc, il faut donc tester empiriquement car une valeur trop élevée du Heap pose des problèmes lors du linkage et crée une erreur de "RAM overflowed".

Niveau matériel, trois tâches indépendantes serons créées : une pour l'ADC, une pour les moteurs et une pour la gestion de l'UART. Nous parlerons dans cette partie des deux premières.

3.1.2 Commande des actionneurs

Le driver moteur nécessite deux pins de commandes par moteur et un PWM par moteur. Le principe du PWM est assez simple. On va utiliser un timer qui au lieu de compter simplement jusqu'à la valeur du registre ARR, va avoir une deuxième valeur de comparaison inférieure à celle de l'ARR. Le signal du PWM change de polarité quand le timer atteint cette valeur et quand le timer fini de compter, ainsi on a un signal carré avec un rapport cyclique variable en fonction de cette valeur de comparaison.

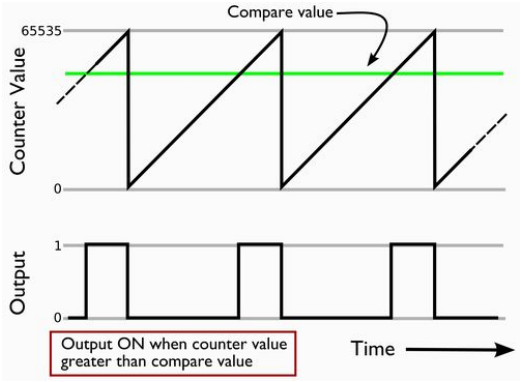


FIGURE 4 – Explication du principe du PWM

Nous avons donc plusieurs calculs à notre disposition pour avoir une certaine fréquence de PWM et un certain rapport cyclique :

$$timer\_tick\_frequency = Timer\_default\_set \div (prescaller\_set + 1)$$
$$PWM\_frequency = timer\_tick\_frequency \div (TIM\_period + 1)$$
$$TIM\_period = timer\_tick\_frequency \div PWM\_frequency - 1$$
$$pulse\_lenght = ((TIM\_period + 1) \times DutyCycle) \div 100 - 1$$

où "DutyCycle" est une valeur exprimée en pourcent qui représente notre rapport cyclique. Ainsi avec notre horloge à 64MHz nous choisissons un prescaller de 999 (pour diviser la fréquence du timer par 1000, on veut maintenant une fréquence de PWM de 100Hz, on effectue donc la troisième ligne de calcul et on trouve que la valeur de l'ARR doit être de 639 pour atteindre cette fréquence. Maintenant si on veut un rapport cyclique de $0,25$ il faut un "DutyCycle" de 25% on obtient donc que 159 est la valeur à mettre pour la comparaison du PWM. Si nous voulons que nos deux moteurs aillent à la même vitesse il faut donc que les deux PWM soient configurés à l'identique. Nous utiliserons ici qu'un seul timer, le timer 2 qui possède quatre lignes sur lesquelles on peut configurer des PWM, ici la ligne 1 et 4.

Ces calculs ont été testés à l'oscilloscope par observation du signal produit et mesure de sa fréquence et de son rapport cyclique.

D'après la datasheet, pour que le moteur fonctionne et s'allume, il faut que la ligne reliée à l'entrée standby soit à l'état haut et que les deux lignes d'entrées soient de valeur opposées. Ainsi on initialise la ligne reliée à l'entrée standby à l'état haut et on crée diverse fonctions permettant d'allumer les moteurs dans un sens ou dans l'autre en fonction du tableau d'états fourni dans la datasheet du pont en H. Ces fonctions vont permettre de créer les autres fonctions de déplacements : avancer, reculer, tourner à droite ou à gauche. Moduler le pwm grâce aux fonctions de la HAL permet également de faire des routines d'accélération et de décélération.

3.1.3 Configuration des capteurs

Le capteur de détection des distances est un capteur infrarouge SHARP. À partir de sa véritable distance minimale pour laquelle la tension renvoyée est maximale il effectue une courbe de tension à allure exponentielle inverse. La valeur en tension doit être récupérée par une entrée analogique, puis traduite en fonction de la précision voulue (ici sur 12 bits donc en valeur numérique entre 0 et 4095 pour $3,3V$) et ensuite étalonné pour que la valeur récupérée par la carte soit traduite en distance. Pour travailler plutôt avec des variables dites "drapeau" nous utiliserons ici l'ADC avec des interruptions toutes les 10 millisecondes (après une tentative ratée d'utilisation de la DMA avec un timer pour commander l'ADC, l'erreur est encore inconnue).

Pour ce faire il faut configurer l'ADC avec une résolution de 12 bits. augmenter le nombre de cycles quand le temps entre deux conversions devient faible (ici le temps entre deux conversions sera de 10 millisecondes) et activer les interruptions avec NVIC qui est un module de la carte gérant les interruptions. Utiliser la conversion continue ne marchait pas dans notre cas sans véritables explications, nous avons décidé de nous en passer. Niveau code, il faut démarrer l'ADC avec la HAL et les interruptions, récupérer dans le code le prototype de la fonction "HAL_ADC_ConvCompltCallback" qui s'exécutera à chaque fois qu'une conversion sera terminée. Cette fonction est déclarée en "__weak" ce qui signifie que si elle est déclarée dans un autre fichier, lors de la compilation seule celle qui n'a pas le mot clé "__weak" sera compilée, l'autre sera ignorée. Elle est vide, il faut donc coder son contenu. Ici nous décidons d'utiliser des variables drapeau pour indiquer l'état de la variable qui stocke la donnée. Cette fonction positionne donc l'état du drapeau à 1 et récupère la valeur de l'ADC dans une variable globale au fichier.

La tâche associée à l'ADC n'a plus qu'à vérifier l'état du drapeau, traiter la valeur de l'ADC et remettre la valeur du drapeau à 0.

3.1.4 Fonctions primitives

Les fonctions primitives sont les fonctions proches du matériel qui sont données aux couches supérieures. Ici ces fonctions primitives sont au nombre de 6 :

- avancer ;
- reculer ;
- pivoter à droite ;
- pivoter à gauche ;
- distance frontale ;
- est présent.

Les fonctions de déplacement mettent en place l'allumage séparé des moteurs, la gestions du PWM avec les routines d'accélération et de décélération et les temporisations permettant pour les déplacement linéaires de parcourir 15 cm (longueur d'une case pour l'IA) et pour les fonctions tournante d'atteindre approximativement un angle de 90 degrés (difficile sans boussole). La fonction "distance frontale" quant à elle va prendre en compte la distance traduite du capteur mais risque des erreurs quand cette distance est inférieure à 5 cm (chute de tension brutale). La fonctions "est présent" permet de détecter un objet dans une distance définie par l'utilisateur.

À ces fonctions s'ajoutent une fonction plus haut niveau nommée "gestion moteur" qui elle va se servir des indicateurs d'état mis en place par l'intelligence artificielle pour décider de quel déplacement est à effectuer et d'effectuer des déplacements tant que les indicateurs d'états ne sont pas nuls. Quand ceux-ci le sont, elle positionne l'indicateur d'état "déplacement fini" ou "rotation finie" à l'état haut.

3.2 Couche communication

Dans l'optique de créer deux robots collaborant entre eux afin d'accomplir des actions qu'un robot unique ne peut réaliser seul une communication entre les robots est nécessaire.

3.2.1 Format des messages

La quantité de message à envoyer étant faible les messages sont codés sur un octet.

3.2.2 Sécurisation des messages

Il est nécessaire de sécuriser la transmission afin de minimiser les mauvaises données envoyées. Pour cela une méthode utilisant des XOR entre les bits de données est utilisée.

3.2.3 Traduction des message

Les messages envoyés sont des uint8_t donc il faut ensuite les "traduire" en actions à effectuer. Afin de simplifier l'utilisation de la communication par l'IA plusieurs fonctions "traduisant" une fonction simple en un message codé sur 8 bits ont été créées.

3.2.4 Fonctions

Fonction EnvoiMessage : Tout d'abord la fonction récupère la valeur de chaque bit du message grâce à des masques, elle le sépare en 3, puis elle applique des XOR entre les bits d'un même paquet. Elle envoie ensuite chaque paquet dans un ordre précis : les 3 bits de poids forts puis les 3 bits centraux et enfin les deux bits de poids faible. L'ordre d'émission est important pour permettre à la réception de recréer le bon message.

Fonction ReceptionMessage : Tout d'abord cette fonction reçoit les 3 paquets. Elle récupère la valeur de chaque bit grâce à des masques. La fonction vérifie que les valeurs des XOR soient cohérentes par rapport aux bits de données. Grace à un Nouveau masque on élimine ensuite les bits de XOR pour garder uniquement les données. On utilise ensuite l'ordre de réception des données pour reformer l'octet d'origine.

Tests effectués :
- envoi de messages d'un micro-contrôleur à lui-même pour tester l'UART ;
- échange de messages entre micro-contrôleur faisant clignoter des leds en fonction des messages reçus ;
- attente d'un message d'un robot pour faire une action.

3.3 Couche intelligence artificielle