Projet
Robotique BA6

Groupe 61 : Frédéric Pili & Lucie Chêne

MICRO-315
Systèmes embarqués et robotique

Table des matières

1. Introduction

Pour ce projet de robotique, nous avons décidé de réaliser un programme qui permet à l'e-puck 2 d'analyser son environnement et de se déplacer. Le but est de se diriger vers l'objet situé le plus haut parmi plusieurs disposés en cercle autour de lui sur un plan incliné, sur lequel l'e-puck 2 est posé. Il se déplace ensuite jusqu'à cet objet pour s'arrêter devant. Ce projet nécessite d'utiliser plusieurs capteurs, comme le capteur de distance TOF et l'IMU, ainsi que de pouvoir exécuter en parallèle plusieurs fonctions.

2. Fonctionnement

Le robot ne démarre que lorsque la position 9 est choisie par l'utilisateur à l'aide du sélecteur de mode. Ensuite, trois phases différentes se succèdent pour l'e-puck 2.

2.1. Phase 1

Durant la phase 1, l'e-puck 2 tourne sur lui-même jusqu'à effectuer un tour complet. Lors de la rotation, l'emplacement des différents objets est déterminé à l'aide du capteur de distance TOF placé à l'avant du robot.
L'e-puck 2 est capable d'effectuer un tour complet avec précision grâce à la possibilité de récupérer la position des moteurs en nombre de pas. Il suffit alors de calculer le nombre de pas nécessaires pour faire un tour complet.
Le capteur TOF permet de détecter une distance avec précision jusqu'à 2 mètres, ce qui en fait un choix cohérent pour détecter la position de différents objets placés autour du robot, pouvant être interprétés comme des obstacles pour le capteur de distance.
La détection d'objet fonctionne en deux temps : la détection d'un début d'un objet (flanc descendant de la mesure distance), puis la détection de la fin d'un objet (flanc montant de la mesure de distance). Les valeurs des positions des moteurs sont alors récupérées et la moyenne est calculée, ce qui nous permet en principe de connaitre la position des moteurs pour que l'e-puck 2 fasse face à l'objet.

2.2. Phase 2

Lors de la phase 2, l'e-puck effectue à nouveau une rotation sur lui-même afin de déterminer quel est l'objet situé le plus haut sur la pente. Cette fois, c'est l'IMU qui est utilisé pour déterminer l'accélération maximum sur l'axe x.
Le robot peut se placer face aux objets détectés grâce à la position calculée lors de la phase 1.
L'accéléromètre de l'IMU est alors utilisé pour mesurer l'accélération sur l'axe x, qui est l'axe allant dans le sens d'avancée du robot. Lors de l'utilisation l'IMU la rotation est mise en arrêt et reprend une fois la mesure terminée.
Les valeurs des accélérations sont enregistrées et traitées pour déterminer quel objet correspond à la valeur d'accélération la plus élevée, c'est-à-dire celui qui est situé le plus haut.

2.3. Phase 3

Pour finir, pendant la phase 3, l'e-puck tourne dans le sens le plus rapide pour se positionner en face de l'objet le plus haut et se déplace en ligne droite jusqu'à lui.
Un régulateur PI est utilisé pour contrôler la vitesse du robot. Une fois qu'il s'est arrêté devant lui, l'e-puck allume sa body led verte pour signifier que la mission est accomplie et que l'objet a été trouvé.

3. Conception du logiciel

Le programme est conçu de façon à respecter la logique de l'exécution en parallèle, en mettant en place des threads. Les moteurs, le capteur de distance TOF et l'IMU possèdent chacun leur thread dédié, eux-mêmes codés dans des fichiers séparés.
Cette organisation possède beaucoup d'avantages pour l'exécution en temps réel d'un robot mais nous devons aussi gérer la synchronisation et le partage de données entre les différentes parties du code.
La synchronisation est effectuée à l'aide de sémaphores, notamment durant la phase 2, où le robot doit alterner entre des rotations et des mesures d'accélération à l'arrêt.
Nous utilisons des fonctions dédiées au passage spécifique d'une variable d'un fichier à un autre, lorsque nous devons définir et utiliser une variable dans deux fichiers différents pour éviter d'avoir recours aux variables globales.
Nous utilisons des messagebus, qui ont un but de partage de données d'un thread à un autre, mais également de synchronisation avec la possibilité d'attendre la publication d'un topic avant de continuer l'exécution. Dans notre programme, les messagebus sont utilisés pour partager les résultats des mesures de distance et d'accélération.
Grâce à cette organisation en threads, la fonction main du programme est uniquement constituée d'initialisations de capteurs et d'autres fonctions, du lancement des threads puis d'une boucle infinie qui ne fait rien.

3.1 Thread Moteur

Ce thread situé dans le fichier *roues.c* est le cœur du programme. C'est là où se situent les trois phases et d'où sont appelés les autres threads (voir Figure 1). De plus, ce thread gère les moteurs et donc le déplacement de l'e-puck.

3.2 Thread traitementTOF

Contenu dans *TOF.c*, ce thread n'est utilisé que pendant la phase 1.
Ce thread consiste à gérer le capteur TOF pour déterminer l'emplacement de tous les objets présents dans un rayon déterminé par la constante *RAYON* choisie à 30 cm. Nous avons décidé de limiter le rayon du capteur pour éviter que l'e-puck détecte les personnes ou un autre objet non présent sur son plan incliné. Grâce à l'utilisation du TOF, nous pouvons détecter des objets de toutes formes et de toutes couleurs ou matériaux. De plus, il faut une différence plus grande que *LIM_DIST* entre l'objet et le fond pour qu'il soit détecté. Une attention particulière est portée à la situation où le robot débute son scan en étant face à un objet, car une fin est détectée mais pas un

début. Il faut donc prendre en compte la première valeur de position mesurée et la dernière mesurée pour reconstituer la position de l'objet.

3.3 Thread captureIMU

Contenu dans le fichier *IMUmesure.c*, ce thread est utilisé pendant la phase 2 du programme. Il consiste à mesurer la valeur de l'accélération sur l'axe x du robot. Il a besoin d'être synchronisé avec le thread moteur et les mesures se font à l'arrêt. On utilise pour cela les sémaphores *IMU_CAPTURE* et *OBJET_SUIVANT*. Ce thread enregistre les mesures dans le vecteur pentes et publie les résultats grâce aux messagebus dans le topic */pentes*. Les résultats sont alors récupérés dans le thread moteur.

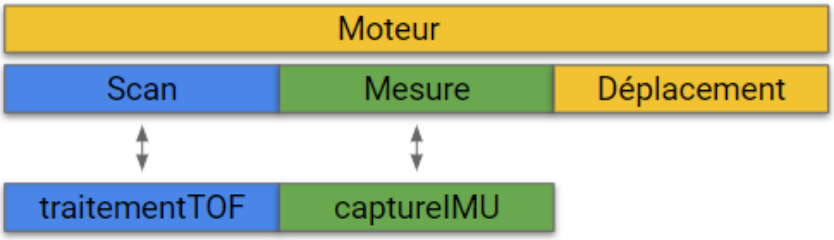

Figure 1 : lien entre les threads et les différentes phases

4. Analyse

La récupération de la valeur de position des moteurs se produit avec un délai, dû à la fonction elle-même mais aussi à l'exécution en parallèle des threads. En conséquence, nous devons ajouter un facteur de correction en nombre de pas sur la position des objets en phase 1 pour avoir une position plus exacte. Nous devons également corriger la position des moteurs après une révolution, car celle-ci n'est pas 0 comme on le voudrait.
La précision de l'IMU pour la mesure d'une seule valeur d'accélération n'est pas parfaite, nous pouvons parfois observer une valeur très incohérente (beaucoup trop grande ou trop petite), ce qui nous oblige à récupérer une moyenne de plusieurs valeurs grâce à la fonction "*get_acc_filtered*" de la librairie. Nous effectuons une mesure sur 50 valeurs afin d'avoir un bon compromis entre précision et temps de mesure.

5. Conclusion

Ce projet nous a permis de mettre en pratique la programmation en C et l'utilisation des threads dans le cadre de l'exécution en temps réel sur le robot e-puck 2. Nous avons notamment pu utiliser plusieurs capteurs comme le capteur de distance TOF et l'IMU ainsi que les moteurs. Nous avons également appris à utiliser des librairies fournies par le constructeur ainsi qu'à utiliser Github pour pouvoir faciliter la

programmation à plusieurs de façon efficace. Grâce à tous ces éléments, nous avons réussi à faire en sorte que le robot trouve l'objet le plus haut parmi plusieurs en cercle autour de lui situé sur un plan incliné.