

1 Introduction

Dans le cadre du cours 'Systèmes embarqués et Robotique', nous avons réalisé un projet utilisant l'E-puck2. Le but était de réaliser un mini-projet en utilisant plusieurs capteurs situés sur le robot en se basant sur les éléments appris lors des TPs. La librairie sur laquelle notre projet est basée correspond à celle des Tp4-5 ChibiOs. Suivant la consigne nous avons choisi d'utiliser les moteurs pas à pas, le capteur de distance IR ainsi que la caméra pour une détection des couleurs. Notre projet consiste en un périmètre dans lequel le robot se déplace vers des objets, s'arrête devant eux, détecte leur couleur puis se déplace vers le but de couleur correspondante située de l'autre côté.

2 Fonctionnement du projet

2.1 L'idée du projet

Dans l'environnement hostile qu'est la jungle, de nombreux bébés animaux, encore dépendants de leurs mamans, se perdent à travers les feuillages. Heureusement, dans notre projet, l'E-puck2 va à la rescoussse des animaux en ramenant les bébés à leurs mamans selon les couleurs.

2.2 Mode d'emploi

1. Placer l'E-puck2 sur le point de départ **n°1**
2. L'allumer (bouton bleu ON/OFF) et le laisser détecter l'objet (le bébé animal) ainsi que sa couleur **n°2**
3. Quand la body-led s'allume, prendre l'objet (le bébé animal) et le poser sur l'E-puck2 **n°2**
4. L'e-puck2 roule vers le but (la maman animal) correspondant à la couleur détectée (rouge à gauche, bleu au centre et vert à droite) **n°3**
5. Quand les leds rouges s'allument, enlever l'objet et le poser près du but **n°3**
6. L'e-puck2 revient à la position de départ et recommence son travail, jusqu'à ce qu'on l'éteigne manuellement (bouton bleu ON/OFF) **n°1**

La maquette est de taille A1

2.3 Implémentation

Pour l'implémentation de notre projet nous avons un certain nombre de fichiers :

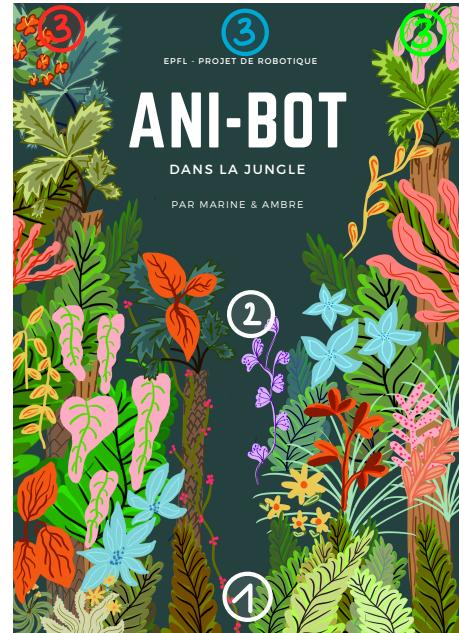


FIGURE 1 - Schéma de la maquette avec les différentes positions que peut prendre le robot

- affichage (.c et .h) : il sert uniquement pour les fonctions d'affichage des leds utilisées lors de notre projet. Il contient une fonction qui indique à l'utilisateur de positionner l'objet sur le robot (body led clignote) et une autre qui indique à l'utilisateur que l'objet est ramené à sa bonne couleur.
- captor_distance (.c et .h) : composé d'une thread et d'une sémaphore, il sert uniquement à lancer proximity_start() ; qui démarre le capteur de distance IR ainsi qu'à le calibrer. Une sémaphore permet de le configurer à wait pour qu'il ne fonctionne que lorsque le signal de celle-ci est appelé.
- go_to_goal (.c et .h) : il s'agit du fichier qui s'occupe du mouvement des moteurs. Il existe des fonctions géométriques différentes, suivant les couleurs. Il contient une thread qui se charge de récupérer la valeur du capteur et de faire fonctionner les moteurs en conséquence. De même que pour le capteur de distance, nous avons mis une sémaphore pour mettre le thread en wait par défaut. Il est activé dans une boucle while dans le main.
- detect_color (.c et .h) : Ce fichier est en charge de la prise des images ainsi que de la détection des couleurs. Il est composé de deux threads. La première sert à capturer les images de la caméra en utilisant la librairie liée à la caméra. De plus, cette thread signale une sémaphore quand une image capturée est prête à être analysée. Cette sémaphore active la deuxième thread, qui l'attendait, et elle peut analyser l'image transmise. Cela permet d'avoir une image complète à analyser. Les threads sont initialisés dans le fichier main.c mais la thread de capture attend une sémaphore qui est signalée seulement dans go_to_goal, quand la distance est inférieure à une valeur donnée. Nous avons pris 150, ce qui correspond à environ 2 cm.

3 Organisation du code

3.1 Les différents Threads et leur priorité

Dans notre projet, nous avons utilisé différents threads. Certaines de ses threads sont appélées dans le main. Grâce à l'utilisation de sémaphore elles sont en wait par défaut, ce qui nous permet de les activer quand on le souhaite.

- Capteur : s'occupe de la détection des obstacles en appelant deux fonctions du fichier proximity.c qui utilise les capteurs IR.
- Goal : s'occupe uniquement des moteurs. Ce thread a pour but de faire avancer les moteurs en fonction de l'environnement. Une fois un objet détecté il appelle désactive le wait de la thread de la camera, ce qui permet à la caméra de ne pas tourner en continu.
- Capture Image : sert à capturer les images de la caméra, elle est lancée une fois l'objet détecté.
- Detect Color : Permet d'analyser les images pour déterminer la couleur de l'objet

Nos threads ont toutes la même priorité. Le cadre de notre projet ne nécessitait pas de priorité entre les threads, nous avons donc choisi de les exécuter toutes avec la même priorité de manière non collaborative et suspension.

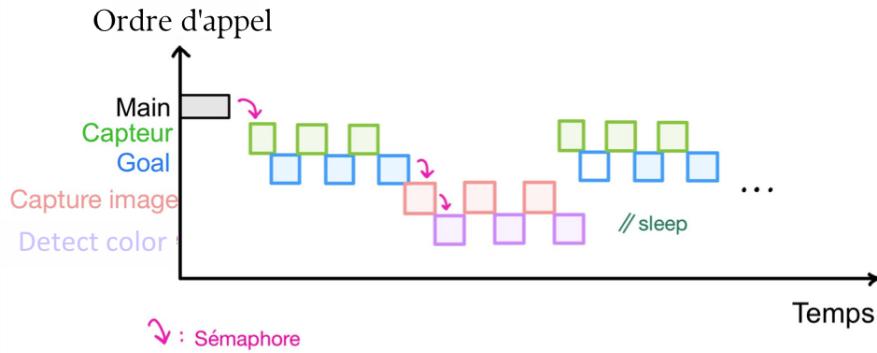


FIGURE 2 - Schéma simplifié des threads (le temps n'est pas à l'échelle)

3.2 Les sémaphores

Il existe des sémaphores dans notre projet. Nous avons fait ce choix car nous voulions implémenter une structure qui se base sur un switch case dans la main, dans lequel une variable d'état active les sémaphores liées aux threads. Cependant, le robot se mettait en panique à ce moment. Nous avons donc enlevé le switch case et cela a fonctionné d'activer les threads au fur et à mesure. Le choix de laisser les sémaphores pour la thread du moteur et du capteur de distance se justifie aussi dans l'idée de vouloir ajouter des étapes au projet, le rendant ainsi plus modulable. Les sémaphores indispensables sont celles entre la thread Capture Image et Detect Color ainsi que entre Goal et Capture Image.

3.3 Les DEFINE et les tests

Les defines sont pour la plupart obtenu de manière expérimentale après de nombreux tests, les define des roues par exemple sont propres à notre robot. Le define lui même nous permet d'optimiser la mémoire mais en plus, nous avons minimiser certaines valeurs (comme le nombre de pixels, le nombre d'images prises,) dans le but d'optimisation.

4 Difficultés rencontrées

Durant ce projet nous avons rencontré une certaine quantité de difficultés. La principale était sur la détection des couleurs. En effet, lorsque nous testions au début, le résultat était aléatoire. Les valeurs des intensités liées aux 3 couleurs, que nous avons observées grâce au debugger, étaient très souvent fausses. Afin de résoudre ce problème nous avons tout d'abord choisi une zone d'intérêt pour traiter la ligne analysée allant de un quart de IMAGE_BUFFER_SIZE (qui est la taille en pixels de la ligne) à trois quarts. Cela nous permettait d'avoir une référence sur toute la ligne car, d'après la datasheet de la caméra, un pixel sur trois prend une couleur différente et les autres sont déduites des pixels voisins. De plus, nous avons réalisé une moyenne de l'intensité des couleurs sur la ligne et un ratio propre au vert (codé sur 6 bits) et au rouge/bleu (sur 5 bits). Ensuite, nous avons désactivé la fonction servant à la balance des blancs ce qui nous a permis d'avoir un résultat plus indépendant de la lumière externe. Enfin, par mesure de sécurité, un certains nombre d'images NUMBER_OF_IMAGE est lu, puis grâce à des compteurs hors de la boucle, la plus fréquente est choisie comme couleur détectée. Cela nous permet de palier à certaines erreurs, pouvant être due à la luminosité ou au mouvement de l'objet.

Par ailleurs nous avons rencontrés une difficulté importante lorsque nous avons essayé de mettre nos fichiers ensemble et de réunir nos codes. La raison principale était la place sur la RAM. En effet le bss lors de la compilation affichait un nombre supérieur à 20Ko ce qui a créé une panique kernel. Pour résoudre ce problème nous avons du réduire l'espace utilisé en RAM et donc se débarasser des couleurs pour les LEDs. La partie qui prenait une place importante était la fonction activant spi, une interface de communication série synchrone, nécessaire pour faire fonctionner les leds rbg. Nous voulions au départ faire fonctionner les leds en suivant la couleur de l'objet détecté. Si l'objet était vert alors le robot suivait la direction du panier vert, et une fois que celui-ci était arrivé devant il clignotait en vert pour signaler à l'utilisateur qu'il fallait qu'il enlève l'objet, de même avec les autres couleurs. Mais l'utilisation des leds rbg prennait environ 10ko d'espace en ram, ce qui faisait avec la caméra et la détection de couleur, un total de plus de 20ko, créant ainsi une panique kernel du robot.

Une autre difficulté était de bien coordonner la détection des couleurs avec la détection de l'objet. Dans le cas contraire, le robot serait continuellement en train de détecter les couleurs, ce qui n'est pas nécessaire et ce qui pourrait également fausser le résultat de la couleur une fois l'objet détecté. Nous avons donc fait appel à des sémaphores pour mettre en wait les threads non utilisées. Cela nous a permis de faire fonctionner les parties en temps et en heure sans gaspiller de ressources.

5 Conclusion

Ce projet c'est globalement très bien déroulé. Nous avons appris à simplifier nos idées et à ajouter des étapes progressivement plutôt que d'attaquer une idée trop compliquée. Nous avons aussi compris l'utilité de git, malgré que nos commits ne soient pas très propres. Nous avons rencontré des difficultés, ce projet nous a appris à parfois revenir à des versions antérieures et à modifier lentement en retestant à chaque fois. Nous avons apprécié ce projet qui était très pratique et qui nous a permis d'appliquer les concepts vus en cours.

6 Remarques

1. Nous avons laissé en commentaire dans le fichier detect_color.c un switch case qui se charge du reboulement du programme. Sans celui-ci, le robot réalise une fois l'ensemble de ses tâches et il suffit de l'éteindre et de le rallumer pour qu'il recommence. Vous pouvez décommenter cette partie et il rebouclera indéfiniment à la suite, il faudra cependant directement replacer un objet sur la position 2, une fois le robot à la case départ.
2. Dans le cas où vous mettez aucun objet/ un objet avec plusieurs couleurs devant le robot, il y a une condition où il exit toutes les Threads et s'arrête. Il se peut qu'il détecte une couleur aléatoirement dû à la luminosité.