Mini-Projet 2024 : Fire Seeker
MICRO-315 Systèmes embarqués et Robotique

Nathann Morand (296190) et Felipe Ramirez (331471)

mai 2024

Contents

1 Introduction

1.1 Contexte

Ce document synthétise notre travail dans le cadre du mini-projet pour notre cour de robotique.

1.2 But

Notre but était de faire un robot qui soit capable de reconnaitre et d'éteindre un feu. Dans les faits cela se ramène à programmer notre ePuck2 pour qu'il se balade tel un roomba dans sur une surface plate et rebondit sur les obstacles rencontrés après avoir vérifié la couleur. Si la couleur est rouge, avant de rebondir, le robot va essayer de rouler sur l'obstacle (le feu) pour l'éteindre.

1.3 Ressource mobilisé

Nous avons eu besoin de comprendre et faire fonctionner les choses suivantes :

- les threads et sémaphore du RTOS : pour un système en temps réel
- la caméra : pour reconnaitre la couleur rouge du feu
- les moteurs : pour explorer
- les capteurs IR de proximité : pour détecter les obstacles
- les LED RGB : car c'est un robot pompier

1.4 approche de développement

Après que nous nous sommes mis d'accord sur ce que nous voulions que le robot fasse et choisit les capteurs que nous voulions utiliser, nous avons créé une branche et avons décidé de partir depuis zéro pour tenter de comprendre ce que nous faisions. Nous avons défini la machine d'état et les fonctions dont nous avions besoin ainsi que la structure générale du code et des threads. À partir de là, nous nous sommes répartis les fichiers et avons commencé à programmer chacun de notre côté. Nous avons rencontré plusieurs problèmes qui sont détaillés dans la section 3.1 bug et workaround. Pour les résoudre, nous nous sommes assis ensemble et avons testé petit à petit de les isoler et de les comprendre. Ce fut l'étape la plus longue du projet. Finalement, nous avons retravaillé l'esthétique du code.

2 Explication technique

2.1 Structure du programme

Nous avons réparti notre code dans différent fichier :

- main
- behaviour
- camera
- movement
- blink
- IR_proximity

2.1.1 Main

Ce fichier sert de point d'entrée. il lance les quatre threads qui gère le robot.

2.1.2 Behaviour

Ce fichier implémente une machine d'état dans un thread dont voici le diagramme en figure 1. pour ce faire, la machine d'état utilise des fonctions de haut niveau fournies par les autres fichiers.

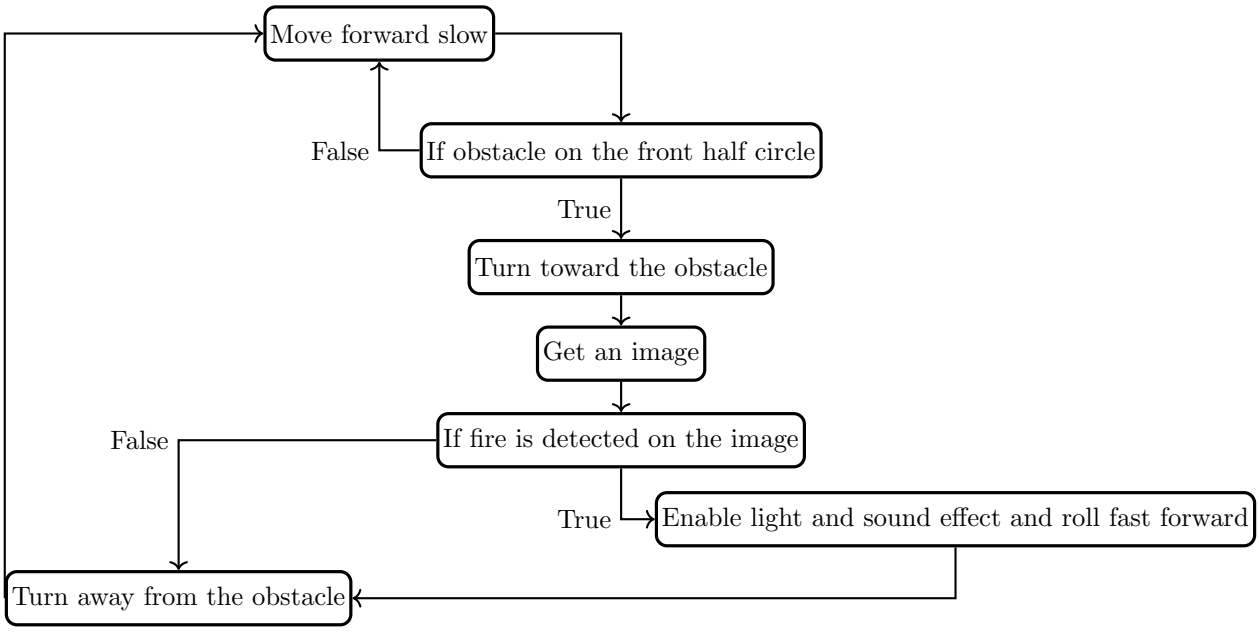

Figure 1: State Machine Diagram

2.1.3 Camera

Ce fichier implémente deux threads, le premier prend en continu des images (en ayant désactivé la balance des blancs). Le second thread récupère un pointeur vers l'image fraichement capturée et extrait dans 3 liste les valeurs pour les pixels rouges, vert et bleu. Le thread somme ensuite l'intensité des pixels rouge, vert et bleu et si le rouge est supérieur dans l'image alors un flag publique indiquant qu'une flamme est détectée est mis à True. Cette approche assez simple semble un bon compris entre rapidité, complexité et le taux d'erreur de détection.

2.1.4 Movement

Ce fichier agit comme un wrapper autours des fonctions de la librairie qui permettent de commander les moteurs. Nous n'avons pas jugé nécessaire de les mettre dans un thread, car l'avance des moteurs est géré par un PWM hardware et celle-ci sont appelés depuis la machine d'état qui est dans un thread. Nous avons implémenté une fonction pour avancer et une pour s'arrêter ainsi que deux fonctions qui permettent de se tourner soit d'un nombre donné de degrés soit vers un capteur spécifique. pour rendre le code plus lisible, nous avons écrit une série de fonctions dont le nom explicite directement vers quel capteur le robot se tourne. Nous avons utilisé le compteur de step et mesurer expérimentalement le nombre de steps nécessaire pour faire un angle donnée sur nous-mêmes. Comme la situation le permettait, nous avons décidé d'utiliser un ratio d'entier plutôt qu'un facteur sous forme de float pour optimiser un peu le code.

2.1.5 blink

Ce fichier implémente un thread qui gère le pattern lumineux du robot de manière asynchrone. un flag global permet de passer d'un mode de clignotement à l'autre. Le thread implémente une simple boucle infinie qui allume soit une led après l'autre en rouge pour donner un effet rotation de la lumière autours du robot, soit en faisait varier l'intensité en bleu de quatre leds autours du robot pendant qu'il patrouille.

2.1.6 IR_proximity

Les capteurs de proximité sont déjà gérés par un thread depuis la librairie donnée. Nous nous sommes contentés d'écrire un wrapper pour rendre l'utilisation de la détection de distance plus explicite et propre.

3 conclusion

3.1 bug et workaround

Nous avons eu affaire à trois bugs majeurs durant le développement. Le premier est apparu lors ce que nous avons voulu utiliser des sémaphores "publique" entre les fichiers camera.c et behaviour.c. Nous avons essayé de les définir de plusieurs manières entre le .h et le .c, mais sans succès. Pour contourner le problème, nous avons utilisé un système de flag global accessible depuis des setters et des getters et nous nous sommes résolus à laisser tourner la caméra en free running. Le second bug venait des moteurs, en effet notre machine d'état vérifie plusieurs fois par seconde son état et appelait ainsi plusieurs fois par seconde la fonction pour changer la vitesse des moteurs, or, il semblerait que le générateur de PWM n'aime pas ça et reste bloqué pendant une demi seconde. Ce problème semble être déjà connu et une modification a été proposé sur la librairie pour ne plus l'avoir. Pour plus de détail, voir la bibliographie : [Morand(2024)]. En attendant, nous vérifions que la valeur de vitesse est différente de la précédente et seulement dans ce cas, nous la mettons à jour. Finalement, le dernier bug venait de la caméra qui ajustait automatiquement la balance des blancs sans que nous en soyons conscients, ce qui nous donnait des résultats aléatoires. Pour le régler, il suffit de forcer le gain de chaque channel de couleur à 1 comme mentionné dans le datasheet de la camera [Pixel Plus Co(2013), page 34].

3.2 lesson learned

Au cours de ce projet, nous avons appris à utiliser les fonctions de base de ChibiOS et mis en pratique les concepts du cours. Notre approche à partir de zéro nous a rendu la tâche plus difficile, mais nous a permis de comprendre la raison d'être du code fourni et à chercher l'information en la remettant en question sans se contenter de copier-coller des lignes de code sans en comprendre la fonction.

4 Bibliographie

References

[Morand(2024)] Nathann Morand. motor.c "resolves a problem when the motors take about 650ms to restart" ¿ not resolved, 2024. URL `https://github.com/e-puck2/e-puck2_main-processor/issues/2`. Accessed: 08.05.24.

[Pixel Plus Co(2013)] Pixel Plus Co. Po8030d datasheet. `https://projects.gctronic.com/E-Puck/docs/Camera/PO8030D.pdf`, 2013. Accessed: 04.05.24.