15.05.2022

Rapport Final

‘Systèmes embarqués et Robotique’

MT-BA6

AMOR Mehdi

BENHASSINE Selma

Table des matières

1. Introduction

L'idée de notre projet s'inspire d'un jeu de labyrinthe. Le robot entre dans celui-ci et l'utilisateur.ice doit le faire sortir en utilisant sa voix. Lorsqu'il se trouve devant un mur, le/la joueur.se peut le tourner à droite en faisant un son aigu ou à gauche en faisant un son grave. Des bandes rouges verticales se trouvant sur les impasses donne un signal au robot afin d'opérer un demi-tour et des autres, bleues, affiche la sortie du labyrinthe. Le but est de sortir le robot avant un temps limite au bout duquel il sonne.

2. Principe de fonctionnement

2.1. Analyse de l'environnement

Le robot utilise plusieurs périphériques et éléments permettant une bonne gestion de ses déplacements et de l'application du projet à l'instar des deux moteurs pas-à-pas qui permettent au robot d'avancer et de tourner, un capteur Time-of-Flight (TOF) qui permet de détecter les obstacles et au robot de s'arrêter, une caméra permettant de détecter les couleurs rouge et bleue, et un microphone permettant la détection de sons sur la base de la fréquence de la voix du/de la joueur.se.

2.2. Fonctionnement du moteur

Les moteurs sont contrôlés par le fichier *motor_pro.c.* Plusieurs fonctions de base permettent de contrôler les moteurs du robot dépendamment des contraintes du capteur TOF, de la détection des sons et des couleurs. Elles se composent de fonctions pour aller tout droit, tourner de 90° à droite ou à gauche. La fonction *motors_init_pos()* initialise la position du robot à 0 après chaque pivotement étant donné que ce dernier nécessite de calculer la distance de pivotement qui équivaut au quart du périmètre du e-puck multiplié par le nombre de pas/périmètre de la roue.

2.3. Détection des murs

La détection des obstacles (des murs du labyrinthe) est faite grâce au capteur TOF. Celui-ci utilise la lumière infrarouge pour déterminer les informations de profondeur. Le capteur émet un signal lumineux qui heurte l'obstacle et revient au capteur. Le temps nécessaire pour faire l'aller-retour est ensuite mesuré et permet de déterminer avec précision les distances [1]. Cela se matérialise dans le code par un thread *TimeOfFlight* du fichier *tof.c* qui est appelé par la fonction *start_tof* dans la main. Dans ce thread, le capteur calcule continuellement la distance avec un éventuel obstacle devant lui grâce à la fonction *VL53L0X_get_dist_mm()* et si celle-ci est inférieure à une distance que l'on prédéfinie, alors le robot s'arrête. Dans le cas contraire, le robot ne fait qu'avancer droit devant lui.

2.4. Détection des sons et des couleurs

Le fichier *audio_processing.c* contrôle la détection des sons dépendamment de leurs fréquences.

Étant donné que le robot doit être contrôlé par la voix, nous avons identifié empiriquement les intervalles de fréquences des voix aigues et graves humaines. Pour le premier cas, l'intervalle va de 550Hz à 650Hz et pour le deuxième de 150Hz à 250Hz. La fonction *sound_remote()* détecte la valeur la plus élevée dans un tampon et où la fonction *init_path()* est appelée. Cette dernière rassemble les informations relevées des périphériques - microphone, caméra, TOF - pour commander le moteur du robot en fonction du projet. En ce qui concerne la caméra, le fichier *detect_color* s'occupe de capturer les images et de les traiter.

L'analyse de l'image est faite deux fois afin de détecter le rouge ou le bleu. La fonction *filter_image()* capturera tous les pixels dans img_bff_ptr(), qui pointe vers le tableau rempli de la dernière image dans RGB565, et les placera et les filtrera dans deux tableaux; bleu et rouge. Le premier tableau sera composé de tous les pixels sauf les pixels rouges – cela permettra de détecter le non-rouge, c'est-à-dire le bleu. De même pour le deuxième tableau qui sera composé de tous les pixels sauf les pixels bleus – cela permettra de détecter le non-bleu, donc le rouge.

Pour les deux tableaux, une fonction *verify_line_color()* permet de détecter la présence d'une largeur de bande prédéfinie (Voir figure 1 et 2) [2]. Pour cela, la fonction calcule la moyenne de tous les pixels en évaluant une ligne en l'axe y. Ensuite, le tableau est parcouru deux à deux de sorte à mesurer chaque pixel et son suivant successivement pour déterminer si la première valeur et au-dessus de la moyenne et si celle d'après et en dessous. Si c'est le cas, le début du fossé (qui représente la largeur de bande) est trouvé, même chose en inverse pour la fin du fossé.

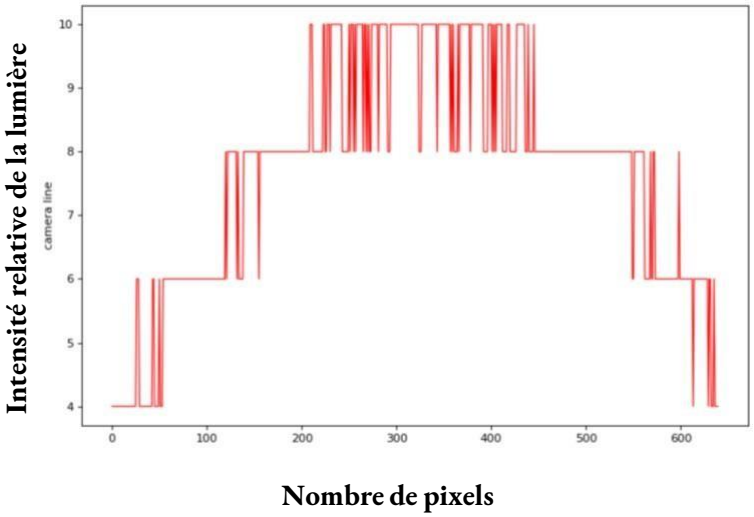


Figure 1: Intensité relative en fonction du nombre de pixels. L'image est prise sur fond blanc et peu est détectable.

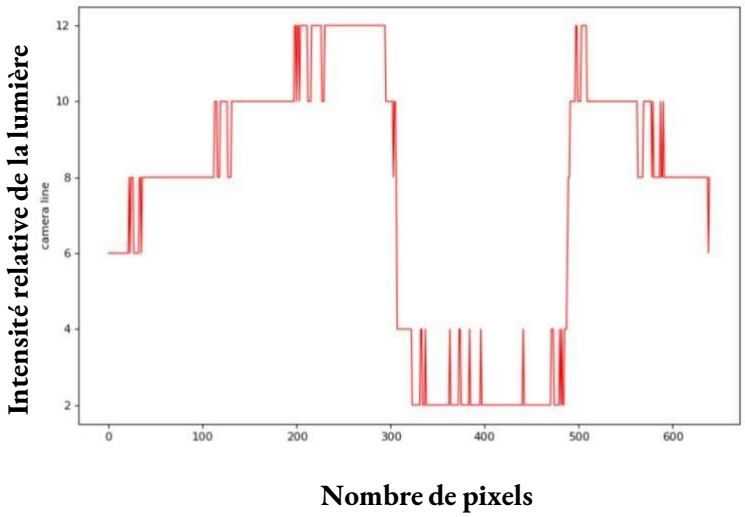


Figure 2: Image prise devant une bande rouge verticale d'où l'apparition d'un fossé, nécessaire pour la détection de couleur.

Afin de minimiser le bruit, trois vérifications sont faites. La première consiste en une double vérification de la détection de la ligne comme expliqué plus haut (deux boucles). La deuxième est faite par une comparaison entre la largeur de bande calculée et une prédéfinie afin d'éviter de prendre en considération les pics aléatoires. Aussi, lors du filtrage, le LSB du *RGB565 binary representation* [3] du rouge ou bleu est éliminé dans le but d'avoir des pentes plus claires (Voirfigure 3, 4 et 5)

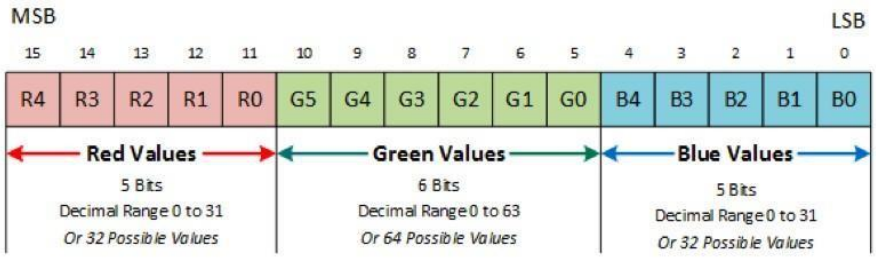


Figure 3: Représentation binaire du RGB 565. Dans le cas du bleu, le bit BO est éliminé. Pour le rouge, un masque est nécessaire

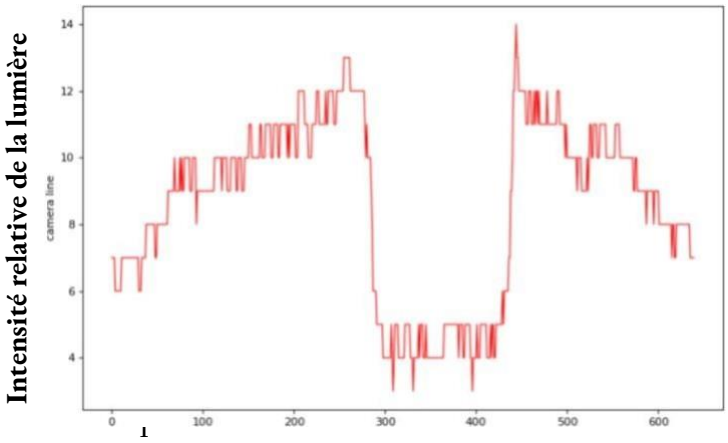


Figure 4: Intensité relative en fonction du nombre de pixels avec le LSB. L'image est prise devant une bande rouge. Présence d'irrégularités

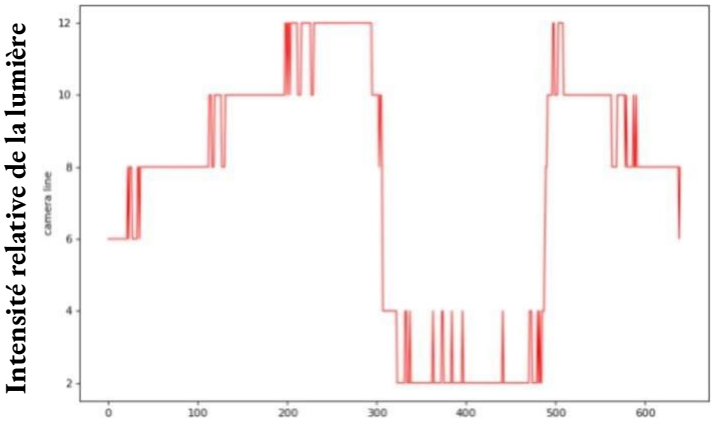


Figure 5: Intensité relative en fonction du nombre de pixels sans le LSB. L'image est prise devant une bande rouge. Valeurs de pics plus uniformes

3. Organisation du code

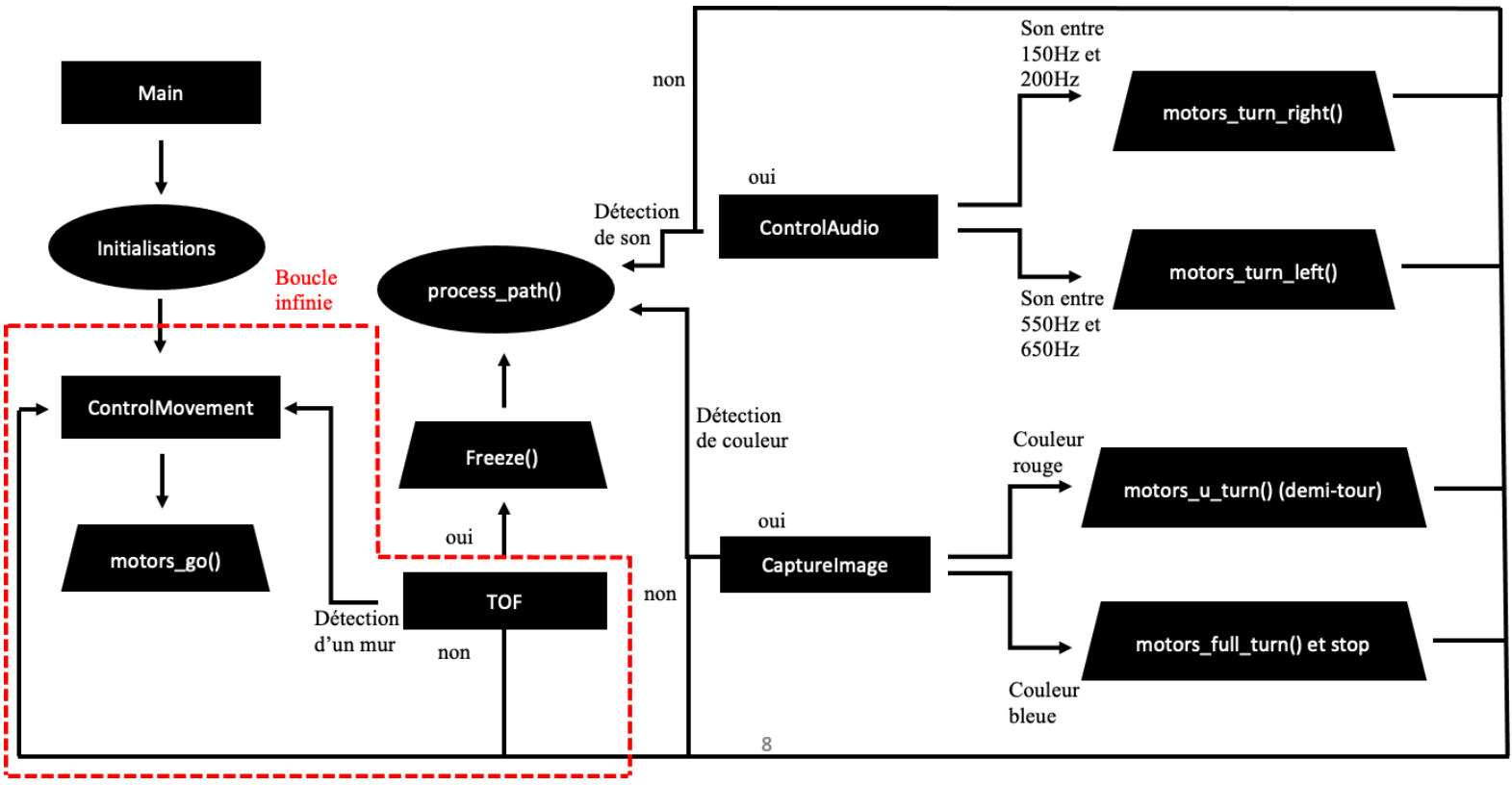
3.1. Interaction entre les threads

Le programme fonctionne avec les threads suivants :

- Main: initialise les threads suivants
- TimeOfFlight : gère la distance entre le robot et le prochain objet devant (que le robot continue ou s'arrête) (NORMALPRIO+1)
- ControlMovement : gère les actions du robot en fonction des conditions (NORMALPRIO)
- ControlAudio : gère l'acquisition et le traitement des sons (NORMALPRIO)
- CaptureImage : gère la capture des images de la caméra (NORMALPRIO)
- ProcessImage : traite les images acquises par la caméra (NORMALPRIO)

Les threads des périphériques sont initialisés dans le fichier main.c. Main appelle dans la boucle infinie la fonction *wait_send_to_computer(),* qui attend une instruction d'une sémaphore se trouvant dans la fonction *processAudioData()* du fichier *audio_processing.c*. Le thread ControlMovement appelle quant à lui la fonction *init_path()* qui, sans contraintes, fait avancer le robot et lit la variable freeze continuellement pour contrôler le cas où le thread TOF, qui une priorité élevée, détecte un obstacle grâce au capteur. Dans ce cas, les moteurs sont arrêtés et la fonction *process_path()* est appelée. Celle-ci va alors utiliser les informations des threads ProcessImage et ControlAudio pour déterminer la prochaine action du e-puck. Ces threads ont une plus petite priorité car le robot doit attendre, dépendamment des conditions, l'afflux de certaines informations. Aussi, puisque le traitement des tableaux de données des images et des sons prend du temps, il est préférable que ces threads soient facilement interrompables. De plus, le thread ProcessImage est relié au thread CaptureImage – une sémaphore *image_ready_sem* empêche de capturer une image sans que celle d'avant n'a été traitée. ProcessAudio sera aussi contrôlée par deux sémaphores : une attends continuellement dans la boucle *while* de la main que les informations acquises par le microphone soient suffisantes pour analyser avec la fonction *wait_send_to_computer()*. L'autre attends que le thread TimeOfFlight détecte un obstacle avant de lancer la fonction *processAudioData()*.

3.2. Flow Chart



4. Conclusion

Le projet fût intéressant et enrichissant puisque nous avons appris les bases nécessaires pour pouvoir coder un projet axé robotique notamment en utilisant un système d'exploitation à tempsréel. Nous avons respecté le cahier des charges qui demande d'utiliser les deux moteurs, un descapteurs de distance et un périphérique au moins. Le projet nous a aussi permis de comprendre avec plus de profondeur les corrélations et liens entre composants hardware et software et que l'influence de facteurs externes ne sont pas négligeables dans ce genre de travail à l'instar de la caméra qui y est sensible.

5. Références

[1] Datasheet du STM-VL53L0X ToF distance sensor, https://projects.gctronic.com/epuck2/doc/VL53L0X-Datasheet.pdf

[2] Mondada, F. (2018) Lausanne, Practical exercise 4: CamReg. https://moodle.epfl.ch/pluginfile.php/22861/mod_resource/content/12/TP4Corrections.pdf

[3] RGB565 binary representation. Taken from: http://www.barth-dev.de/online/rgb565-color-picker/