



SYSTÈMES EMBARQUÉS ET ROBOTIQUE MINIPROJET : DOG BOT

16 mai 2021

Groupe 50
Gloria MELLINAND
Jérémy MARCIACQ

supervisé par
Francesco MONDADA

TABLE DES MATIÈRES

Introduction	2
1 Utilisation des périphériques	3
1.1 Micros	3
1.2 Capteurs infrarouges	7
1.3 Moteurs pas-à-pas	8
1.4 Leds	9
2 Structure du code	11
3 Résultats et conclusion	13
Bibliographie	14

INTRODUCTION

Pour ce miniprojet, nous avons décidé de programmer l'E-PUCK2 pour qu'il agisse à la manière d'un chien. En effet, le but est que ce robot se dirige vers nous lorsqu'on l'appelle¹, tout en évitant d'éventuels obstacles.

Par ailleurs, le but étant de simuler le comportement d'un animal, nous souhaitons imposer à notre robot un fonctionnement que l'on pourrait qualifier de "fluide", malgré son fonctionnement principalement digital.

Nous allons donc utiliser les micros de l'E-PUCK2 pour déterminer la provenance du son à reconnaître, les capteurs infrarouges pour éviter d'éventuels obstacles, les moteurs pas-à-pas pour se diriger vers la source sonore et les leds pour afficher l'humeur de notre DOG BOT.

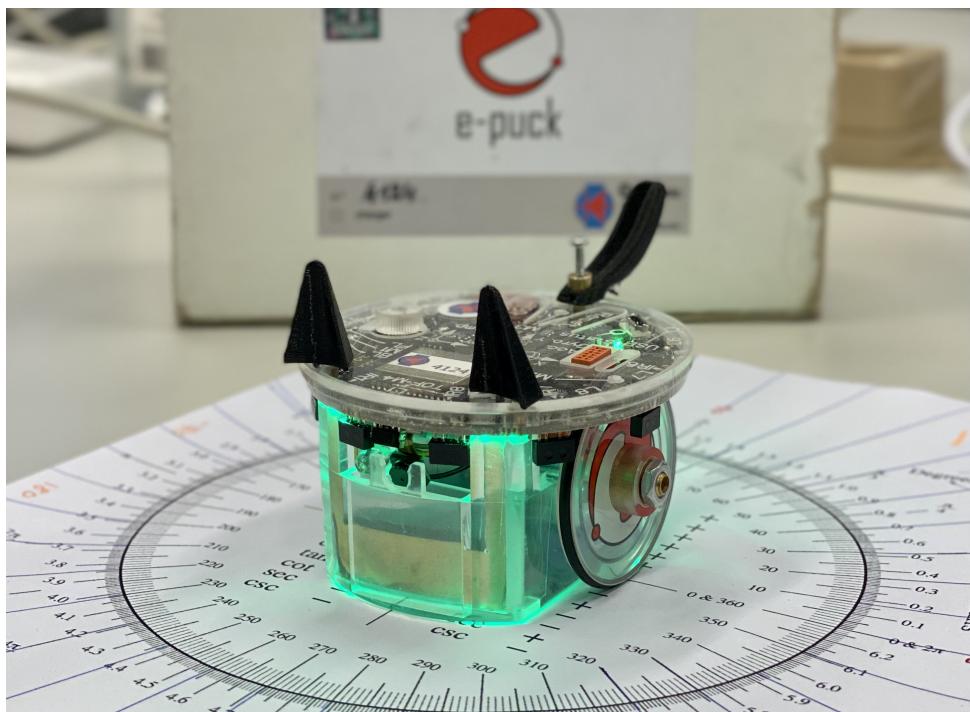


FIGURE 1 – Notre E-PUCK2 en mode stand-by.

1. Nous n'avons pas cherché à ce qu'il reconnaisse des paroles, mais une fréquence bien distincte dont il détermine l'origine.

PARTIE 1

UTILISATION DES PÉRIPHÉRIQUES

1.1 MICROS

Pour que le Dog Bot puisse se mettre en mouvement, nous allons imiter un sifflement grâce à une fréquence sonore de 500Hz. A l'aide des 4 microphones situés sur le robot, nous pourrons en extraire l'amplitude et la phase de deux micros opposés.

Nous avons utilisé la librairie microphone.h fournie dans e-puck2_main-processor [2], pour obtenir les buffers des 4 micros afin de déterminer la provenance du son : les paires de micros seront chacune utilisée pour déterminer certains angles (cf. fig. 1.1). De plus, nous avons eu recours aux librairies math.h et arm_math.h pour calculer respectivement les FFT et amplitudes des valeurs importées depuis les buffers (tel que nous l'avions fait lors du TP5 [1]) et l'angle de provenance du son en fonction de la phase entre deux micros.

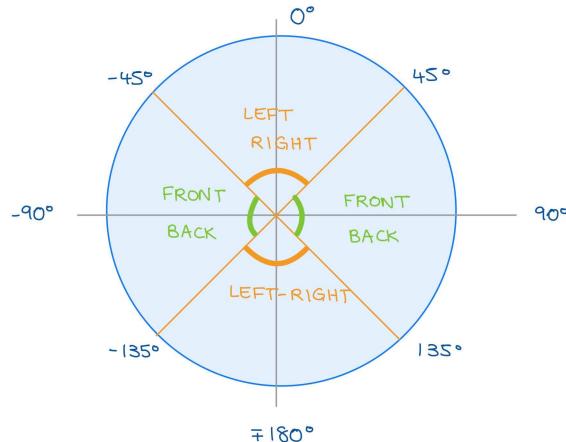


FIGURE 1.1 – Schéma de la paire de micros sélectionnée en fonction de l'angle de provenance du son. L'angle 0° correspond à l'avant de l'E-PUCK2.

Suite à l'extraction des FFT et amplitudes de chaque micro, un filtrage en amplitude permet de réduire le nombre de données à analyser, puis un filtrage temporel élimine les pics sonores isolés. Après accumulation de données, nous calculons la moyenne des valeurs en amplitude pour chaque micro. Nous effectuons ensuite une comparaison de ces valeurs pour en extraire la paire avec les 2e et 3e plus grandes amplitudes. Ensuite viens le calcul de phase, puis le calcul d'angle (cf. fig. 1.2, fig. 1.3).

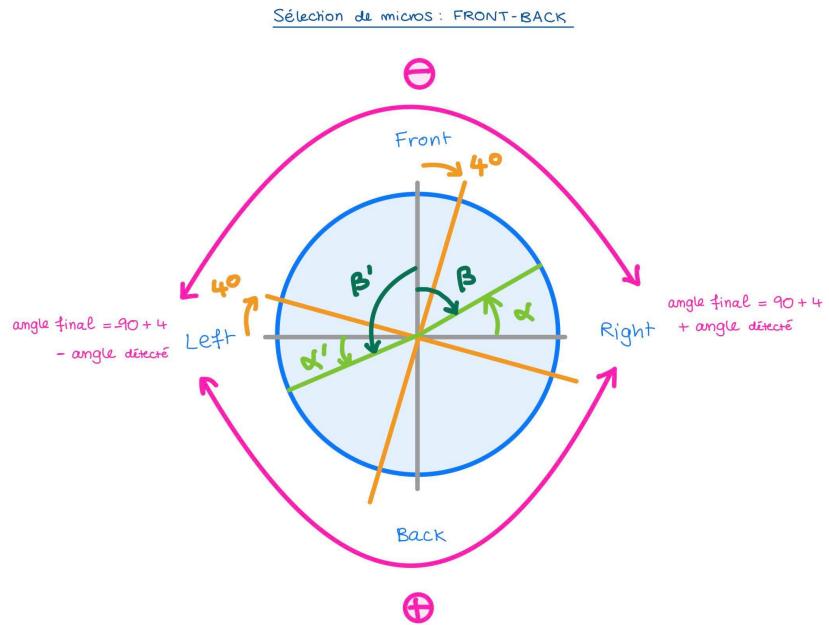


FIGURE 1.2 – Calcul d’angle en fonction de la paire FRONT - BACK. Le signe négatif montre une valeur de phase et d’angle négative. Le décalage de 4° a été implémenté pour compenser la position décalée du micro FRONT sur le PCB. "Angle détecté" (β , β') correspond au résultat de la conversion phase-angle, tandis que "angle final" (α , α') représente l’angle entre l’avant du robot et "angle détecté"

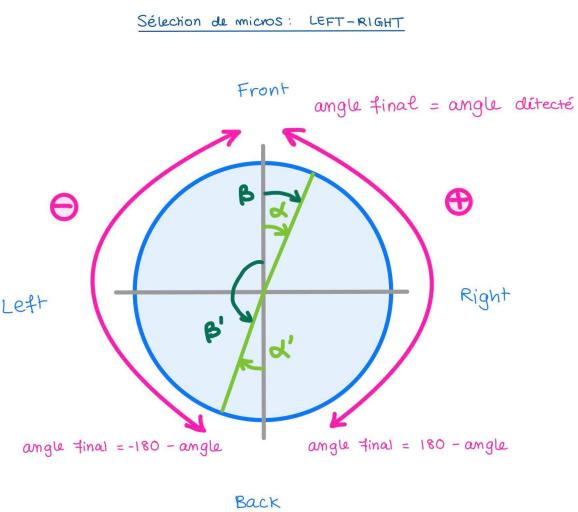


FIGURE 1.3 – Calcul d’angle en fonction de la paire LEFT - RIGHT. Le signe négatif montre une valeur de phase et d’angle négative. "Angle détecté" (β , β') correspond au résultat de la conversion phase-angle, tandis que "angle final" (α , α') représente l’angle entre l’avant du robot et "angle détecté"

Grâce au déphasage entre deux micros ϕ , nous pouvons déterminer l'angle de provenance du son α . La formule 1.1 est déduite du schéma 1.4. La distance entre les deux fronts d'onde arrivant en même temps sur les micros δ se calcule à l'aide du déphasage (équation 1.2), puis nous obtenons la relation finale 1.3¹.

$$\sin(\alpha) = \frac{\delta}{d} \quad (1.1)$$

$$\delta = \frac{\phi * \lambda}{2 * \pi} = \frac{\phi * V_{son}}{2 * \pi * f} \quad (1.2)$$

$$\alpha = \arcsin\left(\frac{\phi * V_{son}}{2 * \pi * f * d}\right) \quad (1.3)$$

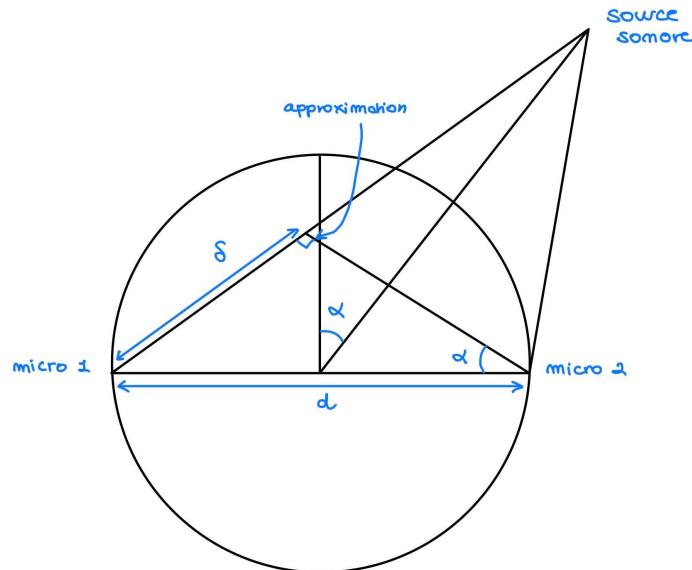


FIGURE 1.4 – Schéma du calcul de l'angle de provenance du son à partir du déphasage entre deux micros.

L'alternance de détection sonore entre deux paires de micros est due au fait que le comportement de la fonction sinusoïdale de la phase en fonction de l'angle est quasi-linéaire sauf aux extrêmes (à + ou - 90°)(cf. fig. 1.5). A cause de cela, il est difficile d'être précis dans les zones non-linéaires en utilisant une seule paire de micros, ce qui explique l'utilisation de la paire FRONT et BACK en plus de la paire LEFT et RIGHT pour n'avoir que des zones linéaires.

De plus, il y a un décalage physique des microphones LEFT et RIGHT par rapport à l'axe horizontal du robot. En laissant la source sonore immobile à 1 mètre du robot, et

1. f : fréquence du signal, V_{son} : vitesse du son, α et ϕ en radians

en faisant tourner l'E-PUCK2 (cf. fig. 1.6), nous avons pu tracer expérimentalement la relation angle-phase (cf. fig. 1.5) et valider l'équation théorique².

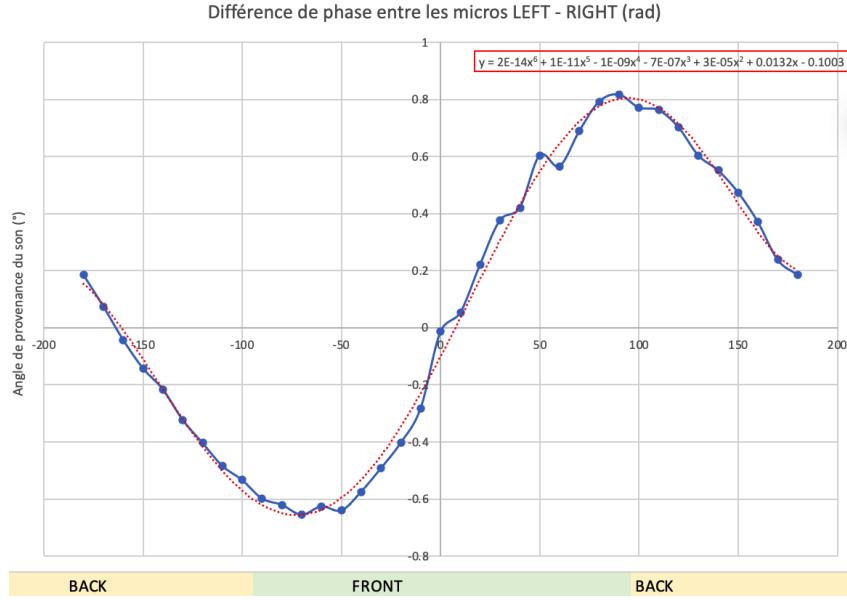


FIGURE 1.5 – Graphe de la différence de phase calculée par les FFT des micros LEFT et RIGHT en fonction de l'angle de provenance du son. Entre -90° et 90° , nous nous situons sur le devant du robot. Entre -90° à -180° et 90° à 180° , il s'agit de l'arrière. En faisant tourner à chaque fois l'E-PUCK2 de 10° , nous obtenons une courbe sinusoïdale correspondant à une rotation de 360° du robot. L'équation encadrée correspond à l'expression de la courbe de tendance polynomiale d'ordre 6 de notre fonction (courbe rouge). Pour plus de précision, chaque point de mesure représente une moyenne de 20 échantillons sonores.

Pendant la sélection de la paire de micros, nous avons pris en compte que la surface d'exposition des capteurs sonores est inégale. En effet, les micros RIGHT et LEFT sont situés sous la plaque protectrice et ont chacun une ouverture identique, ce qui rend leur surface d'exposition égale. En revanche, le micro FRONT est situé sous le PCB, et le micro BACK sur le PCB. Cette inégalité d'exposition rend les valeurs en amplitude de cette paire peu fiable. C'est pour cela que dans la fonction *compare_amps*, nous avons opté pour la paire LEFT et RIGHT comme choix par défaut quand la paire FRONT et BACK nous donne des valeurs illogiques.

L'environnement joue un rôle important dans la détection sonore. Il faut s'assurer que la source soit unidirectionnelle, et garder une distance minimale entre la source sonore et le robot pour ne pas avoir des valeurs d'angles faussées. En effet, une trop grande proximité induit de la résonance au sein de l'E-PUCK2.

2. Nous préférons utiliser l'équation théorique malgré son inexactitude afin de ne pas avoir à refaire les mesures si l'on choisi une autre fréquence



FIGURE 1.6 – Expérience visant à obtenir expérimentalement la relation angle de provenance du son et phase déduite des FFT. Seuls les microphones LEFT et RIGHT ont été activés. La source sonore (gauche) reste immobile, tandis que le robot est tourné avec des intervalles de 10°.

1.2 CAPTEURS INFRAROUGES

Afin d'éviter les obstacles, nous avons choisi d'utiliser les capteurs infrarouges car il nous permettent de déterminer la distance et l'orientation de l'E-PUCK2 par rapport à ces derniers.

Nous avons donc utilisé la librairie proximity.h fournie dans e-puck2_main-processor [2] pour récupérer les données des capteurs sur le bus correspondant³ et les calibrer.

Expérimentalement, nous avons pu calculer une fonction donnant la distance en centimètres en fonction de la valeur retournée par le capteur (c.f. fig. 1.7).

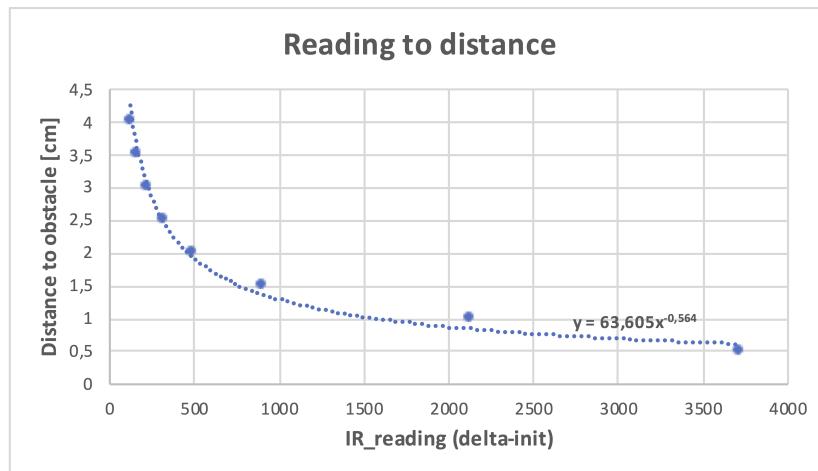


FIGURE 1.7 – Graphique de la distance en centimètres en fonction de la valeur du capteur IR.

Grâce aux distances calculées pour les capteurs avant et latéraux du E-PUCK2 ayant

3. Étant donné la manière dont est structuré notre code, la lecture des capteurs IR n'est pas un élément bloquant pour la boucle principale de l'E-PUCK2. Il est donc inutile d'attendre la publication des données pour continuer notre boucle. Nous avons donc utilisé la fonction messagebus_topic_read à la place de messagebus_topic_wait.

éventuellement un obstacle face à eux⁴, nous pouvons sélectionner les deux distances les plus courtes et calculer à partir de celles-ci l'orientation du robot par rapport à l'obstacle. En effet, connaissant les angles entre les axes des capteurs, la distance entre le centre du robot et chaque capteur, et la distance entre chaque capteur et l'obstacle, nous pouvons trouver précisément l'angle α entre l'axe du capteur IR le plus proche d'un obstacle et l'axe orienté vers l'obstacle (c.f. fig. 1.8).

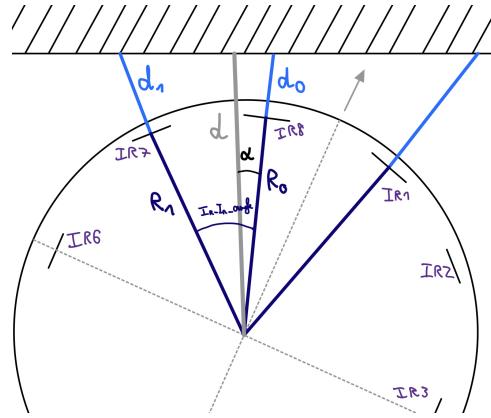


FIGURE 1.8 – Schéma pour le calcul de l'angle entre l'axe du capteur IR le plus proche d'un obstacle et l'axe orienté vers l'obstacle.

Cet angle se calcule alors avec la formule 1.4. De celui-ci, il est aisément de retrouver l'orientation de l'E-PUCK2 par rapport à l'obstacle grâce aux angles entre chaque capteur IR.

$$\alpha = \text{atan}\left(\frac{\frac{R_0+d_0}{R_1+d_1} - \cos(\text{IR_to_IR_angle})}{\sin(\text{IR_to_IR_angle})}\right) \quad (1.4)$$

Finalement, nous avons choisi deux scénarios en cas de présence d'obstacle : soit l'E-PUCK2 s'éloigne à l'opposé de l'obstacle pendant un instant, soit il longe l'obstacle sans le percuter. Le choix est fait en fonction de l'angle qu'a le robot par rapport à l'obstacle.

1.3 MOTEURS PAS-À-PAS

Afin de permettre au robot de se diriger vers la source sonore, nous utilisons la librairie motor.h de e-puck2_main-processor [2]. L'objectif est que le DOG BOT soit appelé ponctuellement puis bouge en conséquence sans forcément recevoir de nouvel appel.

Nous avons tout d'abord défini, dans un soucis de réalisme, que l'E-PUCK2 n'attende pas d'être aligné avec sa destination pour avancer. En effet, sa vitesse frontale est inversement proportionnelle à son erreur de trajectoire.

4. Nous avons choisi d'ignorer les deux capteurs arrières car le robot ne recule jamais.

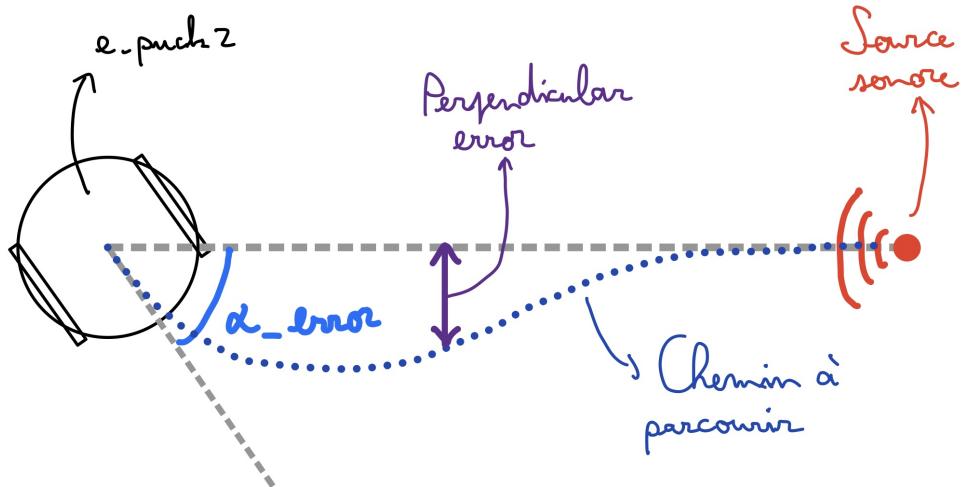


FIGURE 1.9 – Illustration du chemin que doit parcourir le robot s'il entend un son provenant d'une certaine direction.

De là, il nous a fallu rajouter un régulateur proportionnel qui régule la vitesse angulaire du robot (selon son axe pointant vers le haut) en fonction de l'erreur en entrée. Si on avait seulement ce régulateur, le robot se dirigerait dans le sens de la source sonore, mais avec un décalage latéral à cause de sa vitesse frontale qui est déjà positive alors qu'il n'est pas aligné avec sa cible. Afin de contrer ce problème, un second régulateur proportionnel va réguler la référence du premier régulateur en fonction du décalage latéral du robot. Ainsi, l'E-PUCK2 suivra une trajectoire tel qu'elle est décrite dans la figure 1.9. Le choix des coefficients de chaque régulateur a rapidement été fait en essayant diverses valeurs, jusqu'à arriver à un résultat satisfaisant.

Finalement, à chaque boucle, l'E-PUCK2 calcule sa nouvelle orientation ainsi que la distance latérale parcourue à l'aide du compteur de pas de la librairie [2] qui s'est avéré bien assez précis.

1.4 LEDs

Les diverses leds de l'E-PUCK2 ont pour but de mimer l'humeur du robot ainsi que de pointer l'origine du dernier son entendu si celui-ci est encore en train d'être suivi.

Afin de montrer la direction du son, nous avons utilisé les librairies `leds.h` et `spi_comm.h`⁵ de `e-puck2_main-processor` [2] afin d'allumer ou d'éteindre les 8 leds supérieures en rouge en fonction de l'orientation du robot. La front-led est aussi allumée si le robot est face au son qu'il entend.

5. Ce fichier est nécessaire pour la communication SPI avec l'ESP32 afin de contrôler les leds qui lui sont reliées.

Pour les humeurs, nous avons utilisé les leds supérieures et les body_leds. En effet, si notre DOG BOT est énervé, ses leds supérieures vont clignoter en alternance en rouge. Par ailleurs, si le robot est calme, ses body_leds et les leds rgb supérieures vont pulser lentement en vert.

Ce dernier état demande l'utilisation du PWM afin de pouvoir faire varier l'intensité des leds. En effet, nous avons créé deux callbacks⁶ : l'un qui allume les body_leds à chaque période principale, et le second qui les éteint à chaque appel de la période secondaire. Ce moment d'appel varie en fonction du rapport cyclique que l'on fait varier dynamiquement. Afin d'avoir une variation fluide, nous déterminons le rapport cyclique en fonction sinus du temps système. Ainsi, nous avons une pulsation qui inspire le calme.

6. La pin PB2 de body led n'est pas reliée à un timer et on ne peut donc pas simplement utiliser un channel qui change l'état du GPIO sans callback.

PARTIE 2

STRUCTURE DU CODE

Notre librairie DOG BOT est composée de 4 fichiers afin de séparer les diverses fonctions, et fait appel à 4 threads différents pour les capteurs, la communication SPI ainsi que le thread principal¹ faisant appel à toutes les fonctions de notre librairie. La figure 2.1 représente ainsi la structuration de ces fichiers.

Notre mode se lance donc au simple appel de dog_mode_setUp() dans le main.c. Une machine d'état gère quant à elle le comportement du robot en fonction des entrées des capteurs utilisés.

La fréquence du thread principal a été fixé à 100Hz afin d'avoir une variation de l'intensité des leds fluide et pour garder un contrôleur précis. En effet, après plusieurs essais nous avons conclu qu'une fréquence plus élevée mobilisera des ressources supplémentaires inutilement, et qu'une fréquence plus faible rend perceptible la variation de l'intensité des leds par échelon.

1. La taille de la pile du thread a été choisie en essayant plusieurs valeurs. En effet, nous obtenons un Kernel Panic au bout d'un certain temps avec de faibles valeurs de pile, et celui-ci augmente avec des valeurs plus élevées. Nous avons donc évalué la valeur nécessaire à 1024 bytes. Nous n'avons par contre pas réussi à retracer la source du problème.

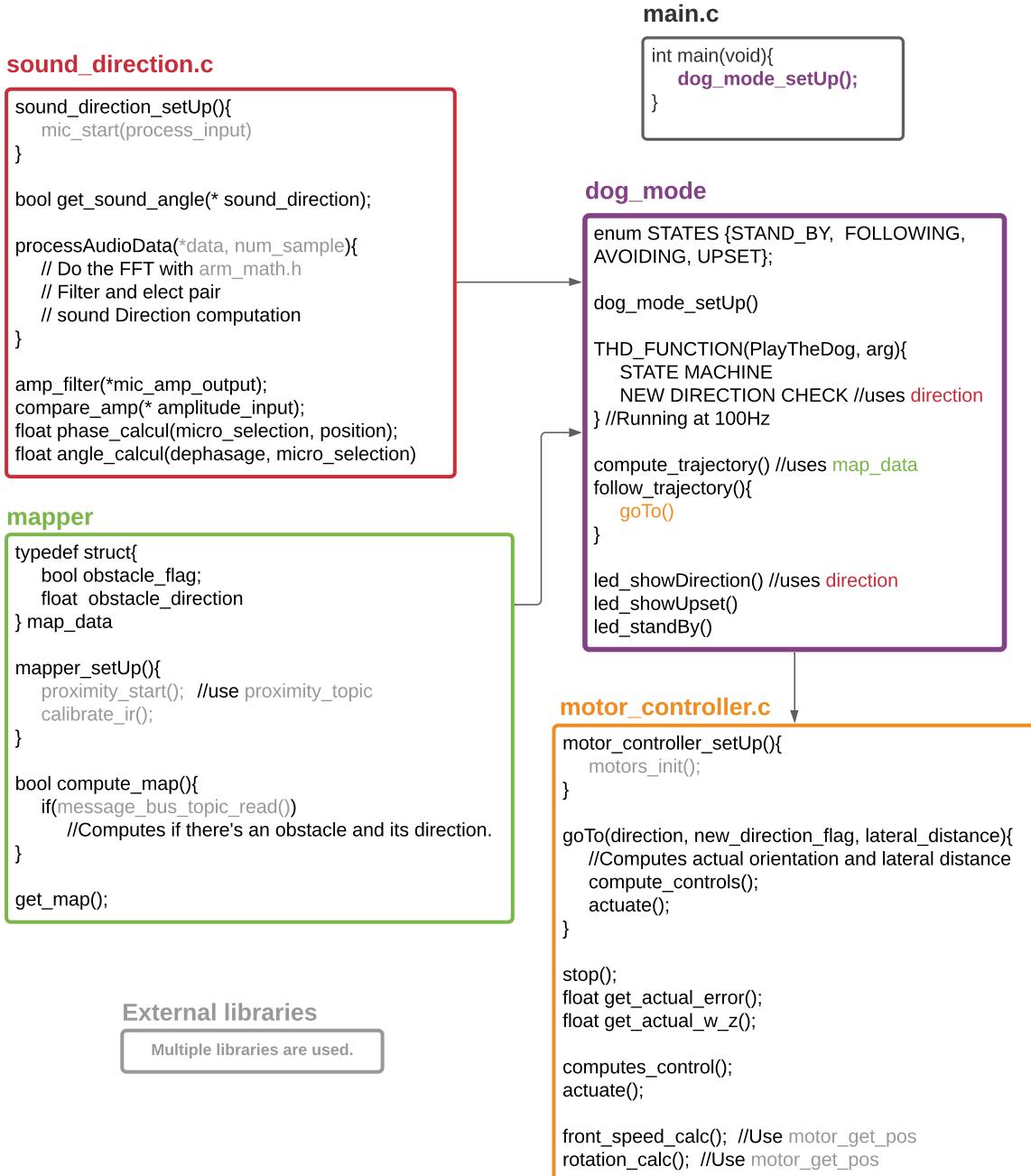


FIGURE 2.1 – Organigramme de la structure générale du code, seuls les éléments importants sont présents.

PARTIE 3

RÉSULTATS ET CONCLUSION

Grâce au contrôleur des moteurs et aux filtres sonores, notre DOG BOT peut poursuivre une trajectoire lisse lors d'un appel. Le seuil de filtrage en amplitude rend le robot plus ou moins susceptible à son environnement. De plus, l'onde sonore peut être réfléchie sur des murs ou de grands obstacles, ce qui peut fausser les mesures. Cet aspect-ci est difficilement corrigible.

Le bruit des capteurs sonores peut causer des oscillations dans les mesures, ce que nous compensons avec une moyenne mobile. Ainsi, ces valeurs aléatoires peuvent être atténuerées, et la stabilité de l'angle est accrue. Nous aurions pu implémenter une fonctionnalité nous permettant de le siffler, en remplaçant la fréquence unique à 500Hz par un filtre passe-bande¹.

Les mesures des capteurs IR ont aussi du mal à être répétables sur tout type de surface. En effet, malgré la calibration, l'éclairage ambiant, la couleur de la surface, ainsi que la texture de cette dernière peuvent influencer la lecture des capteurs et fausser nos mesures.

Ce projet a été une première approche de la mise en mouvement d'un robot à l'aide de multithreading. Pouvoir relier plusieurs concepts à la fois de mathématiques (calcul d'angle du capteur IR, FFT), de système de contrôle (pour les moteurs), d'enrichissements visuels (LEDs avec et sans PWM) et d'analyse auditive était à la fois enrichissant et très intéressant. Il a fallu s'adapter aux contraintes physiques de l'E-PUCK2, tels que l'asymétrie de la position des micros et des capteurs IR, et faire plusieurs essais avec différentes méthodes pour cibler le moyen le plus efficace pour arriver à un résultat précis mais ne débordant pas de complexités.

1. Nous avons malgré tout pu constater que l'E-PUCK2 avait plus de mal à analyser les fréquences plus élevées, nous avons donc continué avec le 500Hz

BIBLIOGRAPHIE

- [1] Francesco MONDADA. *Practical Exercise 5 : Noisy*. Cours de MICROINFORMATIQUE, Section de Microtechnique, Printemps 2020.
- [2] GCtronic. *e-puck2_main-processor*. https://github.com/e-puck2/e-puck2_main-processor, version début 2020.