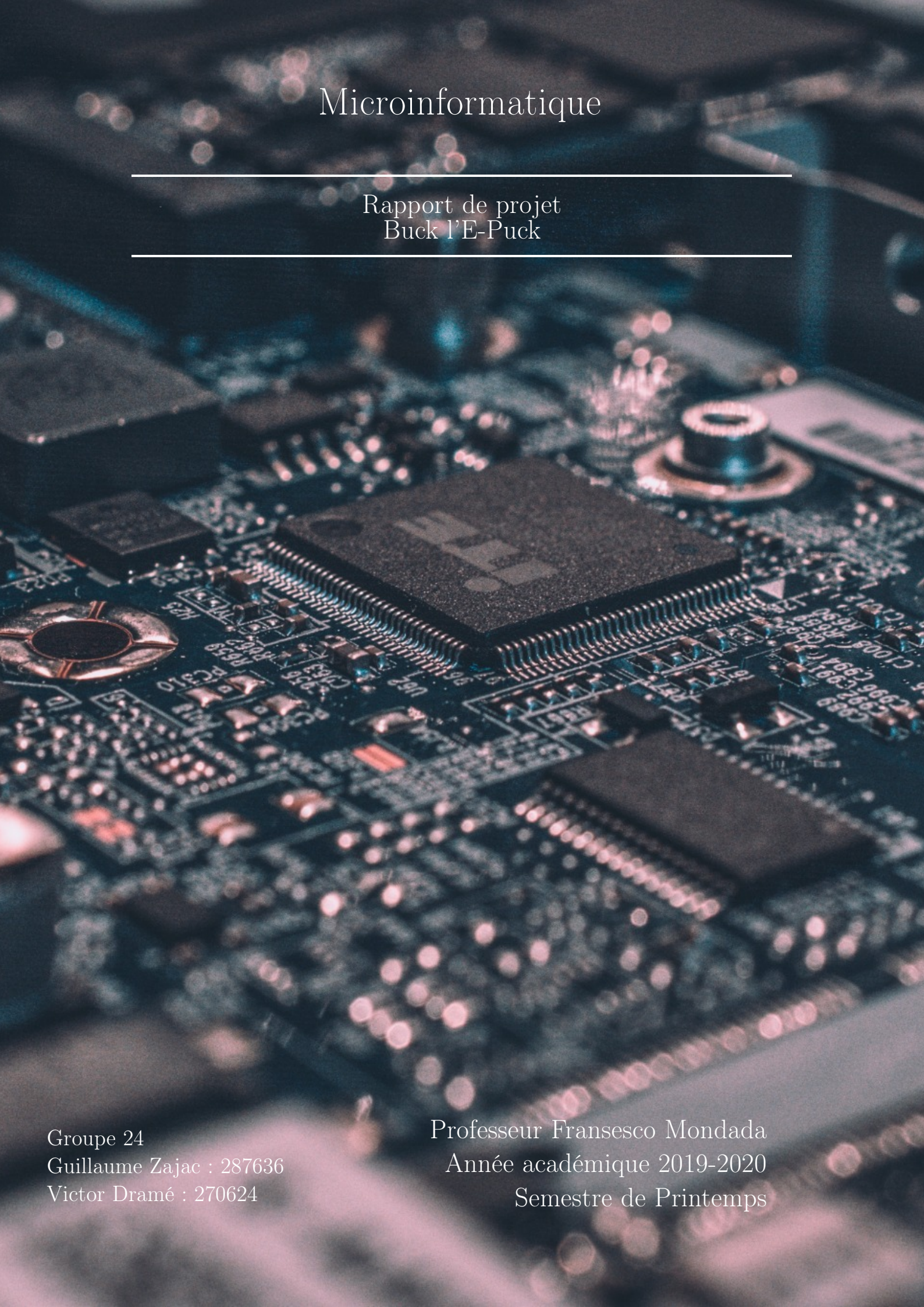

Microinformatique

Rapport de projet
Buck l'E-Puck

Groupe 24
Guillaume Zajac : 287636
Victor Dramé : 270624

Professeur Fransesco Mondada
Année académique 2019-2020
Semestre de Printemps

Table des matières

Table des figures

1 Présentation du Projet

1.1 Cahier des charges

Dans le cadre du cours de Microinformatique ainsi que les séances de TP, il nous a été confié la tâche de réaliser un projet à implémenter sur le robot `E Puck2`. Le but de ce travail consiste à assimiler les connaissances acquises le long du semestre avec de la pratique. Nous devions écrire un programme à charger sur le robot de sorte à ce qu'il réponde aux tâches que nous avions fixé à l'avance.

Bien que le choix des applications à réaliser sur l'e-puck nous était libre, il fallait respecter les contraintes imposées par le cahier des charges à savoir :

- Le projet devait être réalisé sur la base de la librairie `e-puck2_main-processor`, que nous avons étudié lors des TP 4 et 5
- Il était obligatoire d'utiliser les différents modules périphériques présents sur l'e-puck : Moteurs Pas à Pas, au moins un des Capteurs de distance (Infrarouge ou Time of Flight) et au moins un des capteurs utilisés lors des TP(IMU, Microphones, Caméra)
- Chaque capteur ou actuateur doit être géré par un thread
- Le fichier devait être rendu sous forme d'une librairie (avec ses fichiers .c et .h) qui s'intègre avec `ChibiOS`, qui correspond au RTOS de notre MCU

1.2 Idée de notre Projet

Nous sommes au printemps 2020, la crise sanitaire du Covid-19 frappe de plein fouet le monde entier. Confinement ou autres mesures obligent, les citoyens sont incités à rester chez eux et limiter dès lors tout contact extérieur au STRICT MIMINUM ! Bien que ce soit une méthode efficace pour freiner l'épidémie, elle ne laisse pas nos comportements sociaux indifférents. Au fil des jours, un sentiment de solitude s'installe chez les individus et l'isolement affecte la santé des plus sensibles. Vous vous sentez également touché par ce manque de contact social ? Une dépression est-elle sur le point d'imploser en vous au vue du très peu de fréquentations autorisées en cette situation exceptionnelle ? Pas de panique, nous sommes là pour guérir votre santé mentale en vous introduisant notre nouveau prototype. Permettez nous de vous présenter *"Buck l 'E-Puck"*, votre nouveau compagnon du quotidien. Constamment à vos côtés et prêt à vous suivre, vous n'aurez jamais été en de si bonne compagnie. Cette solitude vous paraîtra par ailleurs comme un souvenir lointain !

Il est alors possible de l'appeler pour qu'il vienne auprès de vous, il ne vous lâchera pas d'une seconde. A l'inverse, si vous désirez un peu de tranquillité ou bien d'intimité, vous pouvez demander à votre compagnon de vous laisser en paix. Il partira dès lors dans la direction oppsée pour autant que vous ne lui ordonnez pas de revenir auprès de vous. Enfin, n'oubliez pas de noter que Buck peut s'occuper seul. Ainsi si vous ne lui transmettez aucun ordre sur une période de 5 secondes, ce dernier se mettra en mouvement et effectuera un parcours dont la géométrie prend la forme d'un infini (ou bien d'un 8 tout dépend de votre référentiel). Il s'agit de vous affirmer à travers cette interaction que le robot vous sera fidèle pour une période indéterminée.

Nous avons décidé de simuler "l'appel" du compagnon avec une source sonore en continu. L'interprétation de la tâche confiée se déterminera par la fréquence émise. Il s'agit donc d'une communication utilisateur/robot basée sur les travaux de la 5ème séance des TPs. Quant à la réponse de notre E-Puck, elle se caractérise par des animations spécifiques avec les leds à disposition. Cela permet une transmission simple et rapide de l'information sans passer par un module `UART`. Afin d'ajouter des interactions supplémentaires, il est possible de construire un parcours semé d'obstacles entre vous et le robot. Il sera par exemple alors chargé de vous "retrouver" en franchissant les différents obstructions vous séparant.

Dans le cadre de ce projet ainsi que du cahier des charges, nous utilisons le microphone pour orienter le robot selon le son, mais aussi les capteur Infrarouges pour détecter et contourner d'éventuels obstacles.

2 Implémentation

2.1 Modélisation

2.1.1 Transmission et interprétation des tâches confiées

Comme nous l'avons précisé dans la présentation, l'e-puck répond à 2 tâches à savoir **Suivre le son** ou bien **S'éloigner du son**. L'interprétation des ordres se distingue alors avec la fréquence émise par la source. La 1ère commande correspond à un son émis dans une bande passante de 450 à 600 Hz tandis que la 2ème commande se détermine par une fréquence entre 825 et 975 Hz. Il fallait alors passer par les transformées de Fourier des signaux émis pour étudier leur comportement fréquentiel. Sachant que ces derniers ont des formes sinusoïdales dans le temps, leurs spectres correspondent à des pointes de Dirac. On cherche alors parmi les transformées des signaux échantillonnés celui qui possède la plus grande amplitude et observe la fréquence associée. Il est important de noter que l'e-puck possède 4 micros, soit 4 buffers d'échantillons. Nous analysons ainsi le pic fréquentiel de chaque buffer et vérifions si la pulsation sonore associée est la même pour les 4 mircrophones. Par conséquent il n'est pas possible de transmettre plusieurs signaux avec des fréquences différentes. Cette mesure nous permet de nous déterminer si notre source se trouve bien dans les bandes passantes correspondantes aux 2 tâches possibles.

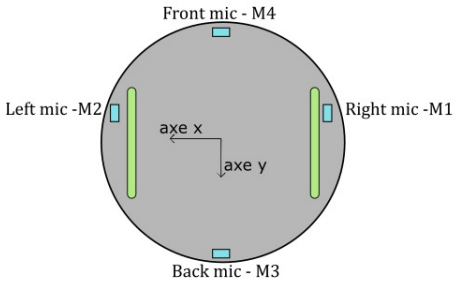

FIGURE 1 – L'emplacement des micros sur l'E-puck, vue du dessus. Le devant du robot (non représenté sur la figure) correspond au vecteur opposé à y

Pour déterminer la direction du son, il faut s'intéresser aux phases des transformées. En effet, en prenant 2 micros opposés sur l'e-puck, le signal mettra plus de temps pour atteindre celui étant le plus loin par rapport à la source. De ce faite, on aura un déphasage dans le domaine fréquentiel entre les 2 échantillons récoltés par les micros concernés. Pour cette mesure, nous observons le delta entre les micros de devant et derrière ainsi que ceux de gauche et de droite, en s'assurant d'abord qu'ils reçoivent tout les 2 la même fréquence. Prenons une des paires de micros, on aura $F_1 = DFT\{Micro\,1\} = Ae^{j\theta_1}$ et $F_2 = DFT\{Micro\,2\} = Ae^{j\theta_2}$, le déphasage se trouve à l'aide de la relation $\theta = atan\left(\frac{\Im\{DFT\}}{\Re\{DFT\}}\right)$.

On peut déterminer les orientations grâce aux équations suivantes

$$\lambda = \frac{v}{f}, \quad \Delta\theta = \frac{2\pi\Delta r}{\lambda} \Leftrightarrow \Delta r = \frac{v\Delta\theta}{2\pi f} \Rightarrow sin(\varphi) = \frac{\Delta r}{l}, l \text{ la distance entre les micros}$$

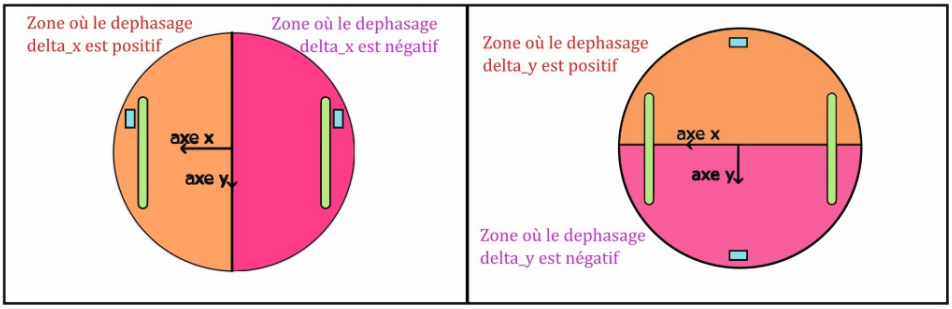

FIGURE 2 – Zones représentant le signe des déphasages gauche/droite donné par x et devant/derrière donné par y, vue du dessus

On obtiendra ainsi deux angles $\Delta\varphi_x$ et $\Delta\varphi_y$ comme définis sur la figure 2 et respectivement donnés par le déphasage des micros M1/M2 et M3/M4. Les signes de ces derniers nous permettront de fixer le quadrant dans lequel se situe le son incident et dès lors établir l'angle relatif entre le devant du robot et la source. L'angle est compris dans l'intervalle $[0; 2\pi]$

2.1.2 Réaction du Robot

Une fois la fréquence ainsi que la direction calculées, Buck se met en mouvement dans le but d'aligner son vecteur directeur (opposé à y) avec la source sonore. Le sens entre les 2 vecteurs diffère selon la tâche. Dans le cas où le robot doit s'approcher de "nous", ils seront en opposition. A l'inverse, si l'e-puck doit s'éloigner, les axes seront dans le même sens. Cette configuration force constamment le mouvement du robot en "marche avant". Afin de rendre ses déplacements le plus fluide possible, la vitesse des moteurs est géré par un régulateur PI. Nous n'avons cependant pas imposé des distances limites entre le robot et la source. Autrement dit il ne s'arrêtera pas s'il se trouve trop proche du son. Lors de son déplacement, Buck nous indique d'où provient le signal en utilisant les leds rouges et rgb, configurées dans la même couleur, situées sur le dessus. Il allume alors la ou les led(s) correspondante(s) selon l'angle séparant le vecteur directeur du robot et celui du son. Le tableau disponible en annexe montre les interactions lumino/sonores (nous n'avons pas réussi à le placer dans cette zone de texte).

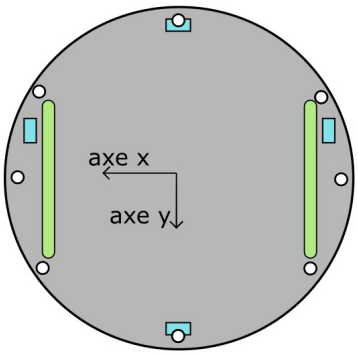

FIGURE 3 – La disposition des 8 leds sur le dessus de Buck pour indiquer la direction du son, vue du dessus. Les leds rouges se trouvent devant, derrière, à droite et à gauche. Les leds rgb, configurées en rouge, se touvent sur les emplacement restants. Le devant du robot est toujours fixé comme le vecteur opposé à y

En absence de signal sonore ou bien en cas de fréquence hors des bandes passantes, Buck s'arrête et attend un nouvel ordre. S'il ne reçoit rien au bout de 5 secondes, il se mettra en situation d'idle. Il effectuera un parcours qui lui est propre, en forme d'infini. Il se mettra par la même occasion à clignoter en bleu, nous indiquant alors qu'il est entré en idle. Les vitesses des moteurs sont définies de sorte à respecter les contraintes géométriques du chemin. Nous avons fixé le parcours comme 2 lignes droites de 30cm ainsi que 2 arcs de cercle d'un angle de $\frac{4\pi}{3}$ et d'un rayon central de 7.65cm. Malgré l'activité apparente, le robot reste attentif à toute nouvelle commande. On peut le sortir de cet idle en émettant à nouveau un son avec une fréquence valable ou bien en insérant un obstacle sur son chemin.

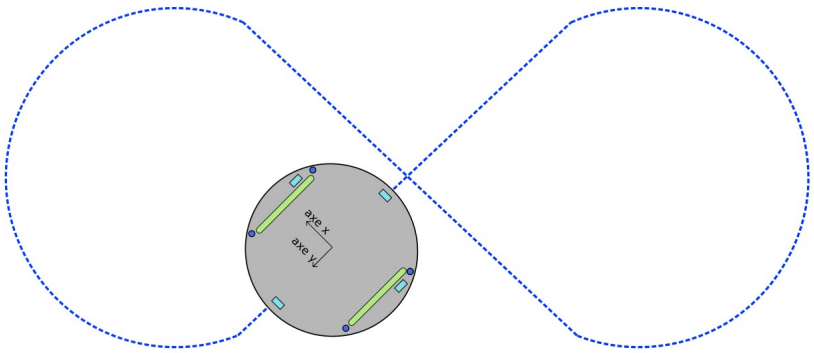

FIGURE 4 – Parcours en forme d'infini entrepris par le robot lorsqu'il rentre en état d'idle

2.1.3 Contournement des obstacles

Comme nous l'avons indiqué dans l'introduction, il ne s'agit pas de résumer le chemin du robot à un simple positionnement avec le son. Nous avons décidé d'utiliser les capteurs de proximité pour gérer des évitements d'obstacles qui pourraient se dresser devant lui. Buck se déplaçant toujours en marche avant, on se sert des 4 capteurs situés à l'avant pour détecter la présence d'objet sur le chemin. A l'instant où l'e-puck détecte la présence d'un obstacle, il met en priorité une routine permettant de l'éviter quitte à ignorer la direction du son et aller dans la direction opposée. Les leds sur le dessus se mettent d'ailleurs à clignoter en rouge/orange/jaune pour signaler qu'il est occupé par l'objet.

L'évitement d'obstacle s'effectue en 3 étapes successives : le changement de direction, le contournement puis le retour sur trajectoire. La 1ère se déclenche lorsqu'un objet se trouve dans la zone rouge, soit si l'un des 4 capteurs de devant le voit. Buck choisit une direction (droite ou gauche) puis tourne sur lui même jusqu'à que l'obstacle ne soit plus l'aire de détection. Vient ensuite le contournement dans lequel le robot se déplace le long de l'objet, jusqu'à ce que son demi-cercle de devant franchisse l'extrémité de l'obstacle. Cette condition équivaut à ce que les capteurs des côtés, situés vers le milieu du robot, ne détectent plus d'obstruction. Enfin l'E-Puck effectue un retour dans sa "trajectoire initiale". En ayant enregistré sa rotation lors du changement de direction, il est capable d'effectuer le virage nécessaire pour revenir quasiment sur son axe initiale. Nous vous résumons le procédés avec les illustrations ci-dessous.

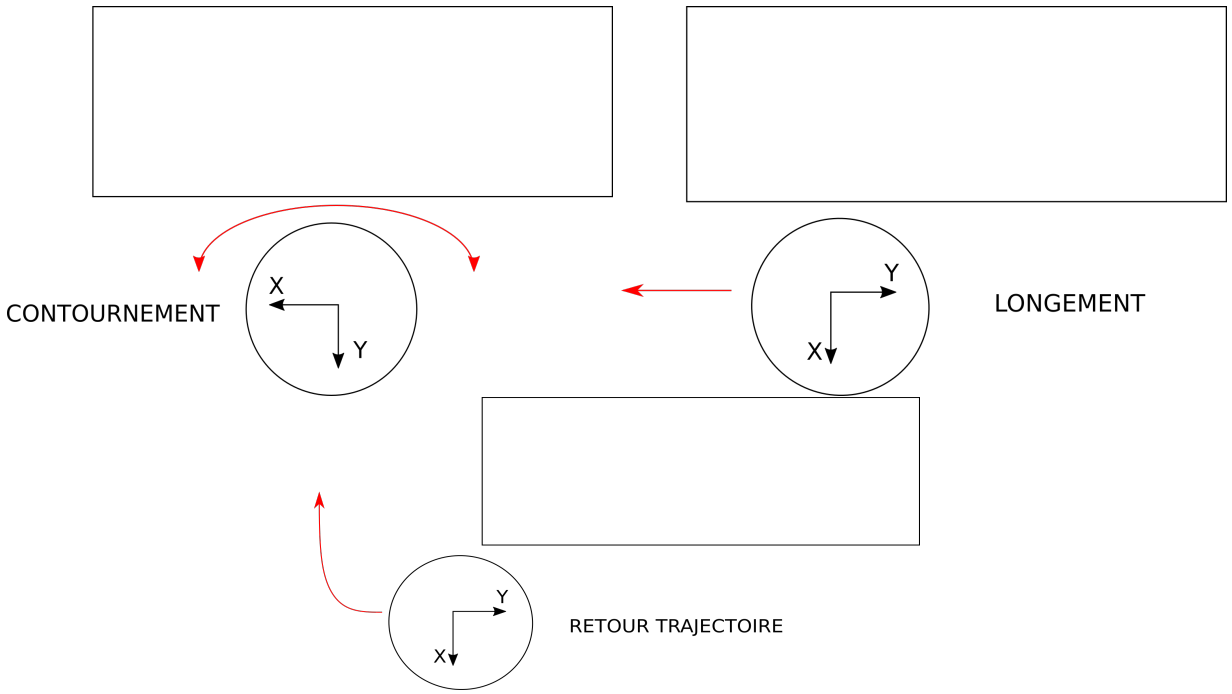

FIGURE 5 – Schémas représentant les 3 étapes successives pour franchir l'obstacle. La flèche rouge représente la direction que notre robot entreprend. Le devant du robot se trouve toujours dans l'axe opposé à y

Le traitement des données en provenance des capteurs s'effectue avec un raisonnement binaire. Autrement dit, le robot "voit" ou "ne voit pas" l'obstacle et est incapable de déterminer la distance à laquelle se trouve ce dernier. La condition `true` se définit lorsqu'un capteur sort une valeur "brute" au dessus d'un seuil fixé préalablement. La valeur threshold correspond à des mesures d'une feuille blanche présente à environ 5 cm de l'E-Puck. Nous avons jugé cet écart suffisant pour que Buck dispose du temps nécessaire pour réagir en conséquence.

2.2 Hierarchie des fichiers

Le fichier principal `main.c` initialise tous les threads et modules de communications nécessaires pour notre programme. Une fois les démarrages terminés, il entre dans sa boucle infinie et laisse alors la main aux autres routines. C'est maintenant qu'interviennent les modules responsables des Réception de données, que l'on peut également considérer comme les entrées de notre programme. Les fichiers `proximity.c` et `microphone.c` sont chargés de récupérer les échantillons respective-

ment issus des capteurs et microphones puis de remplir leurs buffers. Appartenant à la librairie du `main-processor`, tous les threads et fonctions sont définis en interne.

Ces derniers transmettent par la suite leur données au secteur responsable des traitement de données dans lequel se trouvent `traitement_son.c` et `traitement_capteur.c`. Ces fichiers effectuent les conversions nécessaires dans le but représenter l'environnement du robot, par exemple la localisation du son, sa fréquence ou la présence d'objets. Une fois les calculs réalisés tout est envoyé dans le module `parcours.c`, par l'intermédiaire de fonctions `get()`.

Le fichier va s'occuper de la gestion des vitesses des moteurs, soit indirectement du parcours de Buck, selon les paramètres reçus. Il interagit donc avec `motors.c` et lui transmet les valeurs des vitesses à appliquer. Les moteurs agissent en tant qu'une des sorties de notre programme puisqu'elles traduisent la réaction de l'E-Puck face à son environnement. Pour en revenir à la gestion de notre Parcours, c'est également dans ce fichier que Buck doit interpréter quelle tâche il doit effectuer. Il déclenche en parallèle un minuteur de 5 secondes avant d'entrer en idle, et le redémarre constamment pour autant qu'il soit en activité. On peut affirmer que c'est le module responsable de prendre des décisions.

Pour finir, l'état actuel modifié par `parcours.c` est ensuite transmis à `animations.c`. Comme son nom l'indique, il gère toutes les routines d'animations nécessaires pour communiquer à l'utilisateur la situation actuelle de Buck. Il interagit par conséquent avec `leds.c` soit la 2ème sortie de notre programme.

La structure de notre code est résumée dans la figure 8 :

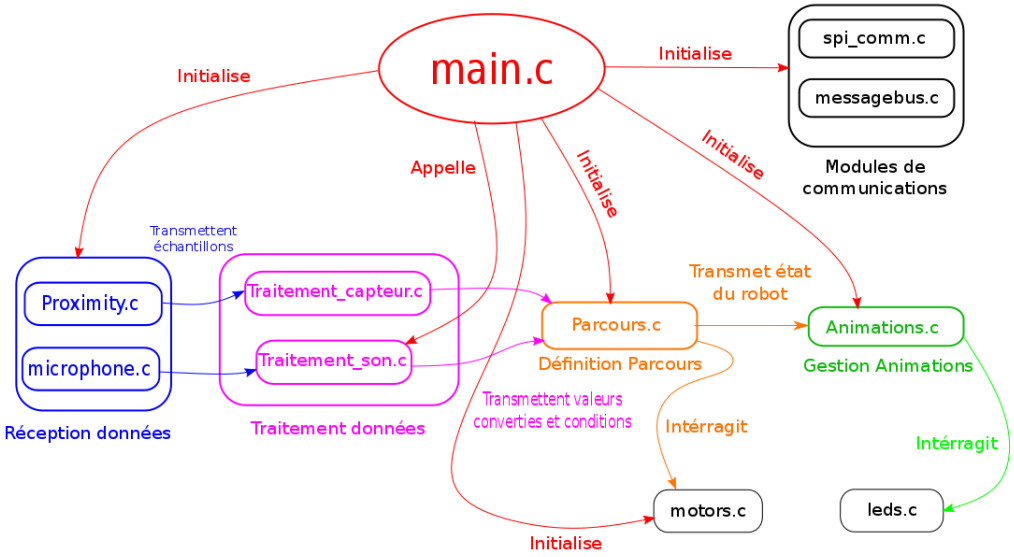

FIGURE 6 – Diagramme représentant la structure du code avec les interdépendances entre les différents fichiers

2.3 Gestion des Threads et fonctions principales

Les capteurs infrarouge et les microphones sont gérés par des threads internes initialisés à leur propre fréquence. Nous avons ajouté 2 threads supplémentaires à savoir `waParcours` et `waAnimations`. L'interêt de leur implémentation était de gérer leurs routines avec des timings bien précis. Enfin nous nous sommes également servis de la main comme un thread notament pour lancer le traitement des échantillons sonores (calculs des DFT et analyse fréquentielle). Tout comme les 2 autres

routines, nous nous assurons que la fréquence ne varie pas. Les détails sont donnés dans les listes ci dessous.

2.3.1 Echantillonnage et traitement

Cela concerne le son, ainsi géré par les 4 microphones, mais aussi l'environnement du robot, traité par les capteurs de proximité. Les données sont reçues et traitées avec les fonctions suivantes :

- `proximity_thd` : Thread interne permettant de récolter les valeurs des 8 capteurs (lumière ambiante, variation lumière réfléchie/ambiante, même variation mais avec une calibration). Les données se renouvellent à une vitesse de 100 Hz.

- `DataProcessing` : Thread interne, s'occupe de l'échantillonage du son grâce aux 4 microphones présents. La fréquence d'échantillonnage s'élève à 16kHz.

- `actualisation_capteurs` : Fonction permettant d'actualiser l'environnement du robot et indiquer la présence d'éventuels obstacles. Elle se charge de modifier les valeurs logiques des 8 capteurs selon s'ils voient un objet ou non

- `processAudioData` Fonction callback récupérant les buffers des 4 microphones pour réaliser les Transformées de Fourier des signaux échantillonnés. Elle est appelée une fois que `traitement_data` laisse la main à l'aide d'une sémaphore.

- `traitement_data` Fonction chargée d'effectuer l'analyse fréquentielle des signaux sonores. Elle cherche notamment les pics d'intensité pour trouver la fréquence associée, mais calcule aussi les déphasages entre les micros appariés. Cette fonction est constamment appelée dans la main tel un thread, dont la fréquence est gérée par le sémaphore. Après plusieurs mesures, nous en concluons que cette dernière stagne autour des 14 Hz.

2.3.2 Réaction du Robot et déplacement

Ces routines attribuent la réaction de notre robot, i.e son état, ainsi que les mouvements à entreprendre. Tous dépendent des données traitées reçues précédemment

- `waParcours` : Thread créé dans le but de gérer le déplacement. Il appelle toutes les fonctions chargées d'actualiser la situation actuelle ainsi que les décisions à prendre. La fréquence de notre Thread est fixée à 100Hz.

- `parcours_obstacle` : Fonction ordonnant le déplacement du robot. Elle ne réagit qu'à 2 phénomènes : la présence de son ou bien d'obstacles. Elle doit par ailleurs actualiser l'étape de l'évitement de l'objet.

- `reponse_sonore` : Fonction appelée dans `parcours_obstacle` et s'occupant du déplacement du robot en accord avec le son. Elle implémente en réalité un controleur PI afin de fluidifier le mouvement au maximum.
- `parcours_en_infini` : Fonction qui lance le chemin en idle une fois les 5 secondes écoulées

- `waAnimations` : Thread créé pour assurer la communication Buck -> utilisateur. Tout comme `waParcours` il appelle toutes les routines nécessaires pour assurer l'animation. Pour des raisons esthétiques, nous avons fixé la fréquence à 2Hz. Ce timing permet de rendre les éventuels clignotements de leds bien visibles.

- `set_tracking_leds` : Fonctions déterminant la direction du son avec les leds rouges du dessus. C'est la seule fonction du fichier n'appartenant pas au thread dédié. Pour cause, une vitesse de 2Hz n'était nettement pas suffisant pour actualiser la position du son. Elle est alors appelée par `reponse_sonore`.

3 Résultats et Conclusion

3.1 Utilisation de la mémoire

Nous avons tenté d'optimiser au mieux le temps d'exécution de notre programme en réduisant le coût mémoire. Pour ce faire, nous avons adapté tous les types de variables selon leurs applications, libérant ainsi quelques octets. En effet la plupart de nos conditions sont résumées par des booléens ce qui prend le moins de place dans la mémoire. Concernant les autres applications, nous avons cherché à déterminer les valeurs maximales possibles dans le but d'employer le type adéquat. Par exemple, on ne gardait que les `float` pour des valeurs demandant des précisions après la virgule telles que la fréquence ou l'angle.
En ce qui est variables globales, elles furent utilisées pour le strict nécessaire tel que les vitesses du moteur ou bien la transmission de données. De plus, ces variables étant déclarées comme `static` seuls les fichiers sources y détiennent l'accès pour les modifier, à l'exception de quelques `set()`. On gardait ainsi toujours le contrôle sur ce qui entraient et sortaient dans chaque module.

Enfin nous avons utilisé le plus de ressources à notre disposition depuis la librairie `main-processor`. Cela nous a permis d'implémenter uniquement 2 threads en plus de la main et dont leur taille en mémoire fut réduite au minimum. En effet, si on fixe la taille de `waParcours` en dessous de 128 octets et et celle de `waAnimations` en dessous de 64 octets, le robot se met en panic, très sûrement suite à un dépassement de mémoire.

3.2 Améliorations

Malgré les efforts fournis pour optimiser au mieux notre code, il reste de nombreux points que l'on aurait pu améliorer. En commençant par la taille des fonctions, notamment dans `parcours.c`, certaines dépassent les 100 lignes. En utilisant les commandes simplifées telles que `switch(conditions)` ou des `if(booléen)`, nous avons du en implémenter des quantités conséquentes. Cela a malheureusement allongé le nombre d'instructions par routines et peut rendre leur lecture moins agréable. En appliquant encore d'avantage le principe de *"Divide and Conquer"* et regroupant plus de fonctionnalités ensemble, nous aurions très probablement allégé les fonctions concernées. Par exemple, il semblerait plus judicieux de déclarer un seul régulateur PI en transmettant les paramètres nécessaires au lieu d'en appeler 2 différents.

Des améliorations concernant l'évitement d'obstacles s'avèrent également possibles. En se penchant sur les valeurs reçues par les capteurs et travaillant sur une conversion en distance, il aurait été fort possible d'adapter un mouvement plus fluide pour Buck. Nous avons tenté l'évitement d'obstacle avec des mouvements en arc de cercle en implémentant un PI. Malheureusement, le résultat ne répondait pas à nos attentes (le robot rentrait constamment en collision), d'où l'idée de fixer une routine avec des étapes.

Mais malgré tout, nous avons tenté de garder une architecture simple dans le mini-projet de sorte à ce qu'il soit facile de lui implémenter de nouvelles fonctions, que ce soit en lui rajoutant des comportements sur certaine fréquence ou en lui ajoutant des animations.

3.3 Synthèse de la présentation visuelle

Nous sommes plutôt satisfaits de notre résultat puisque celui-ci répond très correctement à nos idées et attentes. En effet, Buck arrive très bien à distinguer les différentes tâches demandées et à réagir en conséquence. La gestion d'obstacles s'avère plutôt efficace même avec plusieurs objets mis à la suite. Ils existent néanmoins quelques améliorations pouvant être mise en place. Lorsque la distance robot-obstacle est proche du seuil de détection, le robot oscille entre deux mode de fonctionnement donnant naissance a des petit "zig-zag". Une autre amélioration possible serait de rendre le robot toujours à l'écoute du son lors du contournement d'obstacle. Lui permettant ainsi d'être plus rapide et efficace dans ses décisions.

La géométrie du parcours en idle est respectée et le robot ne peine pas à sortir de cet état en cas de nouvel ordre. Cependant, nous nous sommes rendus compte qu'il arrêtait parfois de son parcours sans qu'on lui en ordonne. Pour cause le bruit des moteurs se situait dans les bandes passantes avec lesquelles on travaillait, et s'interprétait alors comme une commande. Il a fallu changer les fréquences pour ignorer cette source parasite.

3.4 Conclusion

Pour conclure, nous avons pris beaucoup de plaisir et d'intérêts à réaliser ce projet. En effet, ce dernier ainsi que les séances de TP nous ont considérablement familiarisé avec les notions de robotique ainsi que d'automatique. Nous nous sommes de plus rendus compte projet et des TP multitude d'idées réalisables sur l'E-Puck. De plus, l'étude des différents modules présents sur le robot ainsi que leur principe de fonctionnement, nous a fourni un aperçu des nombreux travaux actuels en microtechnique, notamment sur des systèmes embarqués. Cela nous donne un avant goût de ce qui nous attend pour l'année prochaine. Il s'avérait également très intéressant de manipuler avec du code des données issues de l'hardware et comprendre alors les liens unissant les 2 domaines.

Pour finir, ce projet nous a véritablement établi une méthode de travail rigoureuse. En effet, avec la compléxité des tâches demandées, il était essentiel d'étudier au préalable *"Sur Papier"* la structure de notre code ainsi que les dépendances entre les différents modules. Ces quelques heures de réflexion nous ont épargné des périodes interminables consacrées à la recherche de bugs multiples. Par conséquent, nous avons connu très peu d'imprévus allant à l'encontre de nos objectifs et résolu ces derniers avec la méthode des `printf`.

Nous remercions le professeur Fransesco Mondada ainsi que toute l'équipe d'assistanat pour nous avoir accompagné tout au long de ce semestre. Leurs aides nous furent d'un grand précieux et ils ont su nous donner un véritable goût pour la matière.

4 Annexe

Valeur de l'angle \ Led allumée	LED 1	LED 2	LED 3	LED 4	LED 5	LED 6	LED 7	LED 8
$[0; \frac{\pi}{8}]$	OUI	NON	NON	NON	NON	NON	NON	NON
$[\frac{\pi}{8}; \frac{2\pi}{8}]$	OUI	NON	NON	NON	NON	NON	NON	OUI
$[\frac{2\pi}{8}; \frac{3\pi}{8}]$	NON	NON	NON	NON	NON	NON	OUI	OUI
$[\frac{3\pi}{8}; \frac{5\pi}{8}]$	NON	NON	NON	NON	NON	NON	OUI	NON
$[\frac{5\pi}{8}; \frac{6\pi}{8}]$	NON	NON	NON	NON	NON	OUI	OUI	NON
$[\frac{6\pi}{8}; \frac{7\pi}{8}]$	NON	NON	NON	NON	OUI	OUI	NON	NON
$[\frac{7\pi}{8}; \frac{9\pi}{8}]$	NON	NON	NON	NON	OUI	NON	NON	NON
$[\frac{9\pi}{8}; \frac{10\pi}{8}]$	NON	NON	NON	OUI	OUI	NON	NON	NON
$[\frac{10\pi}{8}; \frac{11\pi}{8}]$	NON	NON	OUI	OUI	NON	NON	NON	NON
$[\frac{11\pi}{8}; \frac{13\pi}{8}]$	NON	NON	OUI	NON	NON	NON	NON	NON
$[\frac{13\pi}{8}; \frac{14\pi}{8}]$	NON	OUI	OUI	NON	NON	NON	NON	NON
$[\frac{14\pi}{8}; \frac{15\pi}{8}]$	OUI	OUI	NON	NON	NON	NON	NON	NON
$[\frac{15\pi}{8}; \frac{16\pi}{8}]$	OUI	NON	NON	NON	NON	NON	NON	NON

TABLE 1 – Activation des leds du dessus selon l'angle séparant le devant du robot et le son. L'angle est défini dans le sens anti-horaire, d'où le défilement des leds dans l'odre décroissant, elles définies dans le sens horaire