# Sytèmes embarqués et robotique

## Bachelor VI Microtechnique 2021

Antoine Perrin Lara Laamari Groupe 44

16 mai 2021

# Table des matières

# 1 Introduction et présenation

En ces temps incertains de Covid, certaines de nos habitudes se sont vu chamboulées. Manger une bonne pizza en terrasse c'est trouvé impossible c'est pourquoi en commander et en recevoir une en 30 min est plus important que jamais. C'est pourquoi nous avons développé "GORGIO", plus rapide et autonome qu'une Tesla. Son but : livrer plus vite que son ombre des pizzas succulentes aux 4 coins de la ville de Lausanne afin que les clients puissent se remémorer les saveurs de la liberté pré-covid.

Pour ce faire, afin d'adapter cette vision au robot Epuck, nous avons modélisé la ville par une succession de lignes noires et de carrefours, nous avons remplacé la localisation GPS par une localisation sonore.

À la réception d'une commande (donc quand il entend un son) Gorgio se dirige vers le client tout en respectant le code de la route (suivant les lignes noires). Lorsqu'il arrive au niveau d'un carrefour Gorgio choisi où tourner suivant la position du client. Il en va de soit que Gorgio à passé son permis et donc met le clignotant lorsqu'il tourne. Mais dans une vraie ville il y a d'autres voitures nous direz nous ! C'est pour ça que lorsqu'il voit un obstacle devant lui (une voiture modélisée par une petite boîte en carton) il s'arrête à une distance de sécurité (Covid friendly) prédéfinie et allume ces feux de détresse afin de prévenir les autres automobilistes. Si l'obstacle se met à reculer, Gorgio reculera afin de respecter au mieux les distances de sécurité.

Lors de ses déplacements Gorgio émet des signaux lumineux indiquant un mouvement.

Enfin une fois arrivé devant le client, le signal (le son) s'arrête et Giorgio livre la ou les pizzas avec une petite musique qui met de bonne humeur afin d'être heureux de déguster ces bonnes pizzas.

# 2 Implémentation

## 2.1 Fonctionnalités et capteurs utilisés

**Suivi de ligne noir :**

Afin de pouvoir s'orienter vis à vis des lignes noir au sol nous faisons appelle à la camera. Nous extrayons ensuite deux lignes de pixel du bas de l'image, qui nous suffisent pour l' analyse de la position et de la largeur d'une ligne noir au sol, si celle-ci est présente. Grâce aux informations de la position du centre de la ligne, nous pouvons orienter notre robot face à elle, pour qu'il les suivent. En s'intéressant également à l' élargissement de la route, nous pouvons anticiper les croisements.

**Triangulation du son :**

Le robot détecte un son et calcule l'angle entre lui et la provenance du son en utilisant le déphasage entre les signaux perçus par les micros (calculé à partir de la FFT de ceux-ci).
Nous avons utilisé les 4 micros afin d'avoir une mesure très précise et limiter les erreurs de mesure. Pour ce faire si la fréquence entendue par les micros est la même, Gorgio calcule l'angle entre son axe x et la cible et celui entre son axe y et la cible à partir de la formule suivante :

$$sin(phi) = \frac{v*deph}{e*2*\pi*f}$$

où est la distance entre les micros.

Ensuite nous utilisons une moyenne mobile pour ne pas avoir de trop grandes oscillations d'angle. À partir de ces 2 angles nous calculons l'angle final entre l'avant du robot et la cible.

Enfin quand Gorgio est sur un carrefour, il récupère l'angle et choisit où tourner en fonction de celui-ci.

**S'arrêter avec un obstacle :**

Nous avons utilisé le capteur de distance TOF afin que Gorgio ne rentre pas dans les obstacle qu'il croise. Il avance ou recule de façon fluide afin de rester à une distance de 'sécurité' prédéterminée (5 cm).

**Animation des LEDs et de la musique du buzzer :**

L'animation des LEDs est effectué dans une thread, qui check toutes les 250ms l'état actuel du robot( mouvement tout droit, en train de tourner lors d'une intersection, etc..). Pour chaque état, nous avons implémenté une animation de led différente.

Notre animation de son intervient uniquement lors de l'arrivée à destination du robot, lorsqu'il commence à jouer la musique de flag de mario, géré par une thread.

**PID implementation :**

Nous avons implémenté 1 controleur PI et un PD pour que le mouvement de Gorgio soit le plus fluide possible.

Le régulateur PD à été utilisé pour le suivi de ligne pour que Gorgio n'oscille pas autour de sa position d'équilibre. Nous avons récupéré l'erreur entre la position du robot et le centre de la ligne. Nous l'avons dérivé et calculé la vitesse de correction de rotation à donné aux moteurs à partir de cette formule :

$$rot\_speed = KP\_PD * error + KD * \frac{error - old\_error}{D\_T}$$

où, après essaies, KP = 2 et KD = 2.

Pour éviter les oscillation autour de la position d'équilibre nous avons mis un treshold de 5 pixels.

Le régulateur PI à été utilisé pour la distanciation avec un obstacle. Nous avons récupéré l'erreur de distance vue par le capteur TOF (voir "s'arrêter avec un obstacle"), et nous l'avons intégré afin de trouver la vitesse de rotation à donner aux moteurs du robot à partir de l'équation suivante :

$$speed = KP\_PI * (error) + KI * (sum\_error)$$

où, après essaies, KP = 20 et KD = 0.002
éviter les oscillation autour de la position d'équilibre nous avons mis un treshold de 5 mm

## 2.2 Gestion des threads

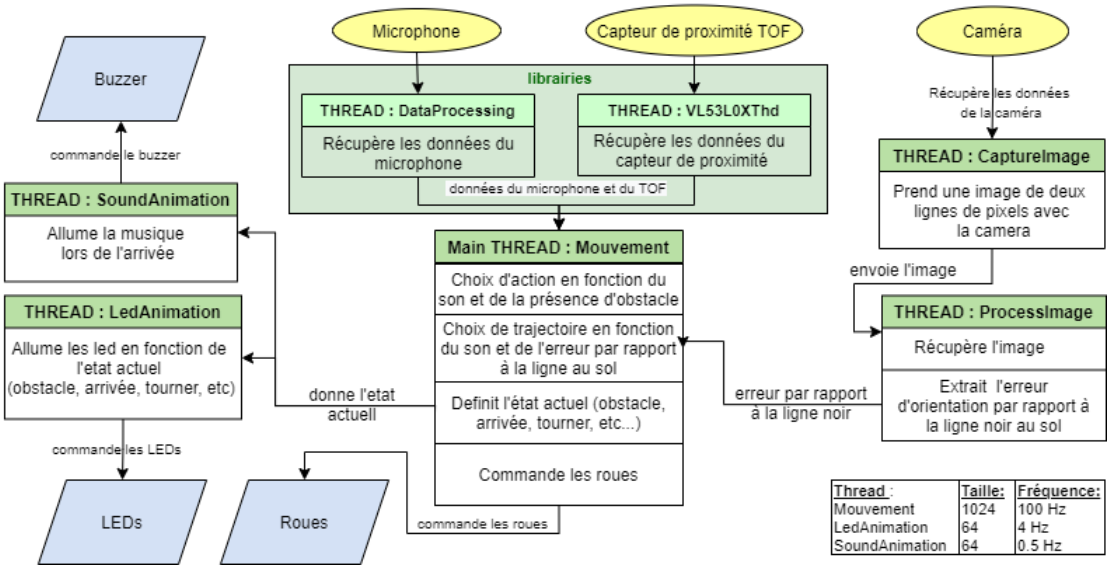


FIGURE 1 – Organisation des threads

Mémoire :

Afin d'optimiser la mémoire utilisée, nous avons adapté les types de variables à chacune des variables crées (par exemple int sur 8 bits au lieu de int sur 64 bits pour des petites variables) et nous avons diminué au maximum la mémoire alloué aux threads tout en vérifiant que le robot ne rentre pas en mode panic.

## 2.3 Gestion des actions

Une fois les données des capteurs récupérées dans les threads adéquates, c'est le fichier "PI_regulator" dans lequel se trouve le Thread "Mouvement" qui est chargé d'organiser le mouvement.

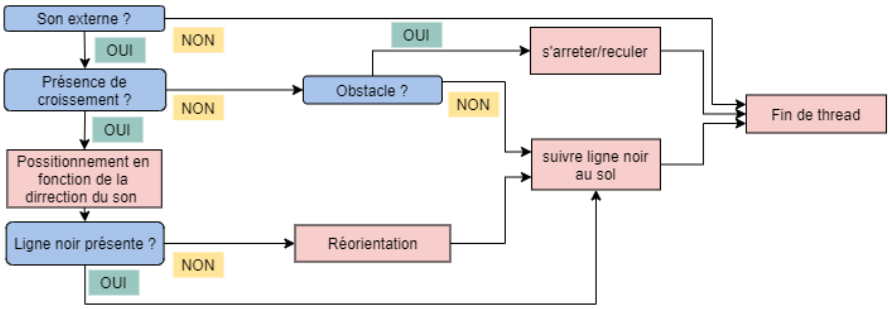


FIGURE 2 – Organisation de la thread mouvement

Le robot regarde tout d'abord s'il reçoit un son. Si Non il ne bouge pas. Si oui alors il suit une ligne en utilisant le régulateur PD qui utilise l'erreur retourné par la thread "Process image". S'il ne voit pas de ligne il avance et tourne jusqu'à en trouver une.

- **Carrefour :**
  - Si le robot détecte un carrefour, autrement dit s' il détecte une hausse de l'épaisseur de la ligne d'une certaine valeur (300 pixels) alors il sait qu'il doit anticiper un croissement. La caméra regardant droit devant, le robot ne peut pas voir à moins de 6cm de lui sans miroir. Une fois qu'il détecte un carrefour il se déplace alors d'une valeur pré mesuré adéquate pour se positionner au centre du carrefour.
  - Il récupère ensuite l'angle qui a avec la cible afin de choisir où tourner (fonction "trun_choice()").
  - Après avoir tourné, s'il y a une route (ligne noire) il continue en la suivant sinon il tourne du côté où l'angle avec la cible est le deuxième plus faible.
- **Obstacle :**
  - Le robot récupère la distance avec les objets devant lui et vérifie si cette distance n'est pas plus petite que la distance critique (5cm) s'il n'y a pas de carrefour.
  - La vitesses des roues est contrôlée par un régulateur PI qui à pour but de positionner le robot à 5 cm de l'obstacle. Si l'obstacle recule alors le robot recule aussi pour éviter la collision.
- **Arrivée :**
  - Gorgio comprend qu'il est arrivé s' il voit un obstacle(le client) et qu'il ne reçoit plus de signal (de son).
  - Il se met en mode arrivé, fait une animation lumineuse et joue une musique au buzzer. Ces annimation sont gérés par leurs Thread (voir gestion des thread)

Pour simplifier au maximum la compréhension du code nous avons créé des fonctions propres aux actions plutôt que d'écire un thread très long. Le thread mouvement fait appel à ces fonctions (turn_choice, PID_regulators, motor_turn, corner_approch, etc..)

## 2.4 Diagramme de hiérarchisation des fichier

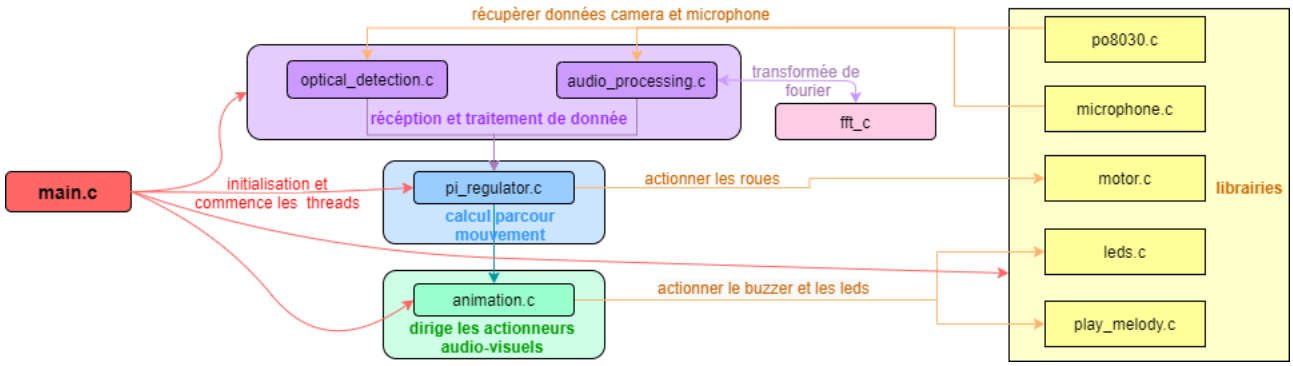


FIGURE 3 – Diagramme de hiérarchisation des fichier

# 3 Oraganisation du travail et utilisation de github

Afin de travailler de façon efficace et organisée, nous avons utilisé github pour l'échange et le partage de notre code et nous nous sommes arrangés afin de se partager epuck régulièrement pour pouvoir tester chaque fonctionnalités développées.

Sur Github nous avons décidé de créer une nouvelle branche pour chaque nouvelle fonctionnalité, nous permettant ainsi de nous y retrouver facilement dans tous les commits. Chacun a donc travaillé et fait des tests en parrellèle sur sa branche avant de la merge à la branche principale, de néttoyer les fichiers modifiés en commun (comme le main pour les initialisation par exemple) et de tester les nouvelles fonctionnalités ensemble. Les branche ont été ensuite suprimés afin de nettoyer notre répertoire.

Au total nous avons utilisés 4 branches (sans compter les 2-3 branches origin-main créés par des erreurs de manipulation) en plus du main : l'analyse des sons, l'analyse du suivi de ligne, l'implémentation de PID et du capteur de distance TOF, les animations leds et buzzers.

Avant d'utiliser ces branches, il a fallu tout d'abord préparer l'environnement sur la branche principale (Main). Une fois le code fini et fonctionnel nous l'avons nettoyé (enlevé les chprint, les commentaires inutiles, etc..) sur cette même branche.

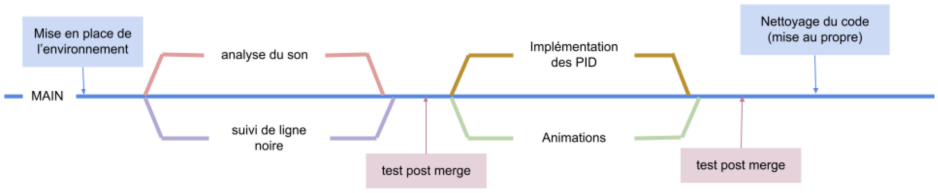


FIGURE 4 – Diagramme de l'utilisation de Github

Lien du repository Github : github.com/antoinetx/Mini_projet-de-Robotique

# 4 Résultats et Conclusion

## 4.1 Problème rencontrés et améliorations

Ce projet de système embarqué et de robotique à était un challenge pour nous, non seulement au niveau du code mais aussi de l'organisation.

Le premier défit a été d'apprendre l'interaction entre un robot et nos ordinateurs. Lors des TPs nous nous sommes plusieurs fois retrouvés dans des situations où la difficulté ne parvenait pas du code, mais simplement de l'implémentation du code sur Epuck. Ces problèmes ont cependant pu être résolus rapidement, à l'aide des assistants entre autres.

Il a également été nécessaire de se rendre compte que la théorie peut se retrouver à ne pas fonctionner comme nous l'aurions prédit. Les conditions lumineuses externes par exemple, jouent un rôle que nous avons appris à ne pas négliger, car les résultats de la caméra en dépendent fortement. Il en est de même pour l'adhésion des roues au sol ou des bruits externes.

Pour finir, l'adaptation à la plateforme de Github a été un challenge pour nous. Nous appréhendions en particulier les "Merges", mais Github s'est révélé très utile, et nous a énormément facilité le travail en binôme.

Pour de futurs projets, nous avons appris à donner un poids plus important aux conditions externes, et qu'une plateforme interactive comme github améliore considérablement la qualité d'un travail à plusieurs sur un même code.

## 4.2 Conclusions

Malgré les défis à surmonter tout au long de ce projet, nous nous retrouvons très satisfait d'avoir pu arriver au résultat que nous espérions. Notre robot accomplit les contraintes que nous lui avons imposées. Nous avons pris beaucoup de plaisir à pouvoir réaliser ce projet du début à la fin et nous permettant de combiner des problèmes

de hardware et de software, ce projet nous a mis dans une situation plus concrète et nous préparer au master et à notre avenir.

Nous avons également pu continuer à apprendre à travailler en binôme, une qualité très précieuse pour notre future, par l'acquisition de l'utilisation de plateformes comme github par exemple.

Nous remercions notre professeur Mondada et toute l'équipe d'assistants, qui nous ont mené, conseillé et aidé tout au long de ce projet, nous permettant de progresser et prendre d'autant plus goût à la matière.

# 5 Références

**Cours et TPs de Systèmes embarqués et robotique** (MICRO-315)
https ://moodle.epfl.ch/course/view.php ?id=467

**Forum d'aide aux TPs** (MICRO-315)
https ://moodle.epfl.ch/mod/forum/view.php ?id=1055600

# 6 Annexe

## 6.1 Cahier des charges

"Le but du miniprojet est de partir sur la base des éléments que vous avez vu lors des TPs 1-5 pour créer plusieurs tâches plus complexes à résoudre par le robot e-puck2."

Liberté dans le choix des tâches et de la présentation.

Éléments requis :

— utilisation de la librairie e-puck2main-processor.
— Utilisation des moteurs pas-à-pas, capteur de distance et 1 des capteurs vues pendant les TPs
— Chaque capteur doit être géré par un thread dans le code
— Le code doit être rendu sous la forme d'une librairie (avec divers .c/.h) qui s'intègre avec la librairie ChibiOS