Microinformatique

Rapport du mini projet

10.05.2020

Groupe 44

Table des matières

1. Introduction

Dans le cadre de ce mini projet, nous avons cherché à mettre en place un labyrinthe. Le robot mis à l'entrée du labyrinthe, va analyser son environnement pour repérer quel chemin prendre tout en restant centré le long de son parcours jusqu'à ce qu'il arrive à la sortie.

2. Principe de fonctionnement et analyse de l'environnement

Cette section décrit les différentes étapes nécessaires à son fonctionnement. L'e-puck est placé dans un labyrinthe avec des murs en blanc. Afin d'analyser son environnement, le robot utilise les microphones (le micro gauche), les moteurs et les capteurs de proximité infrarouges (6 IR out of 8).

2.1. Détection du son

La détection du son est cruciale pour le démarrage du mouvement du robot. En effet, le robot détecte une fréquence située dans un intervalle autour de 550Hz ([500-600Hz]) qu'on lui a imposée. Une fois démarré, le robot ne détecte plus le son, afin d'éviter un calcul constant de la FFT (Fast Fourier Transform) qui risque de remplir la mémoire. Ensuite, le robot commence à analyser son environnement grâce aux capteurs de proximité IR afin d'avancer dans le labyrinthe.

2.2. Détection du chemin à prendre

Nous avons mis en place différents scénarios pour le chemin que le robot doit parcourir : ouvertures à droite et à gauche, virage à 180 degrés et cul-de-sac. Les capteurs de proximité IR sont utilisés pour détecter les obstacles autour du robot afin de déterminer son chemin. Les valeurs mesurées par ces capteurs indiquant la distance entre l'obstacle et le robot, sont obtenues par la fonction *get_prox* et nous avons pu les repérer avec le logiciel *RealTerm* - *Figure 1*. Comme la relation entre la valeur affichée et la valeur réelle ne semblait pas être linéaire, nous avons donc utilisé les valeurs brutes. Pour pouvoir faire correspondre ces valeurs à nos conditions, nous avons procédé à plusieurs tests en plaçant le robot dans différents endroits dans le labyrinthe afin de pouvoir mesurer à la règle et comparer ces mesures avec les valeurs récupérées.

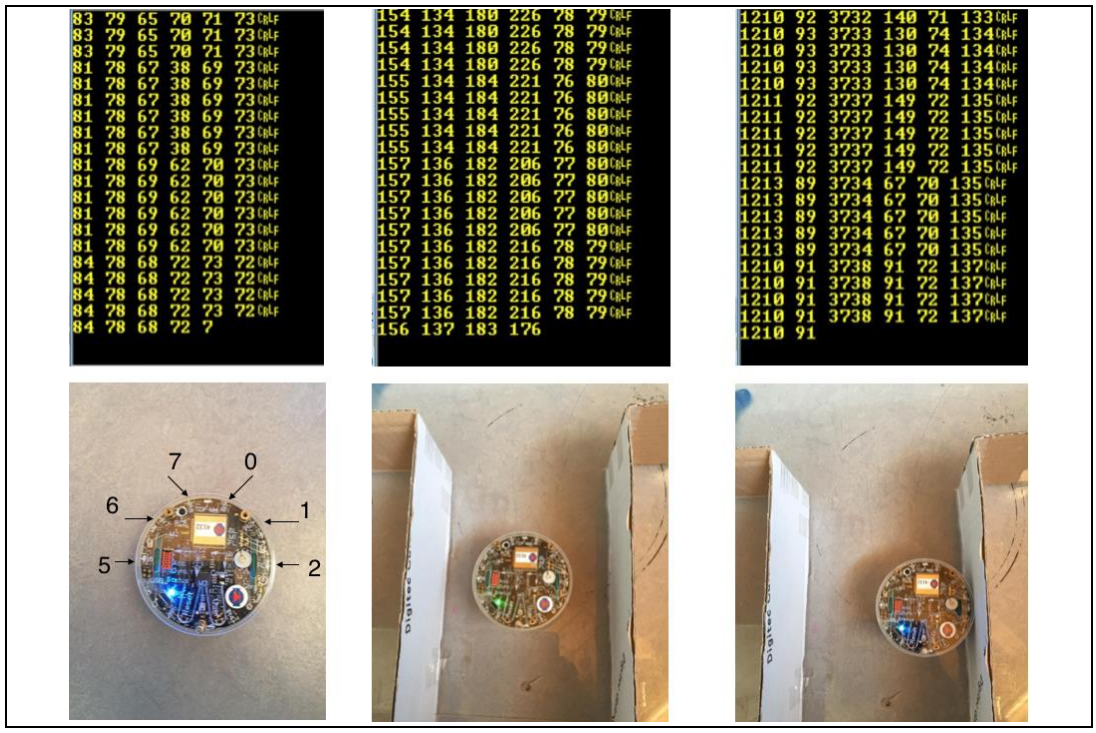

Figure 1 *: Cette figure représente 3 situations différentes : sans rien autour, entre deux murs, coller contre un mur, respectivement de gauche à droite, accompagnés des valeurs récupérées à partir du logiciel RealTerm. Sur ce tableau de données, sont représenté les valeurs brutes renvoyées par les 6 capteurs de proximités IR : 1, 6, 2, 5, 7, 0 respectivement de gauche à droite. Capteurs d'avant : 0 et 7, Capteurs diagonaux : 1 et 6, Capteurs latéraux : 2 et 5. Comme nous pouvons l'observer, plus le robot est proche d'un mur, plus la valeur détectée par le robot est grande.*

Pour savoir ce que le robot doit faire au temps t dans le labyrinthe, il est confronté à tout moment à des tests. Ceux-ci consistent à détecter s'il s'agit de :

- Une ouverture à gauche et un obstacle en face
- Une ouverture à droite et un obstacle en face
- Une ouverture à gauche et rien en face
- Une ouverture à droite et rien en face
- Un cul de sac
- Fin du labyrinthe
- Rien de tout cela

Ces tests sont placés dans cet ordre, ce qui donne une priorité à tourner à gauche s'il s'agit de conflit : ouverture à gauche et à droite par exemple.

Il y a une petite exception pour le rôle du cul-de-sac, car dès le moment où le robot va se trouver dans un cul de sac, il va devoir rebrousser chemin et pour éviter de sortir par où il est entré, il va donc éviter la prochaine sortie qu'il apercevra.

2.3. Stabilisation du robot au centre du parcours

Une fois que le robot a choisi son chemin, il calcule la distance qui le sépare des deux murs des deux côtés. Le but est de stabiliser l'e-puck au centre de son parcours. Pour cela nous calculons la différence entre deux valeurs détectées par les capteurs de proximité IR : la différence entre les deux latéraux et celle entre les deux diagonaux des deux côtés et ensuite on l'évalue.

On procède tout d'abord par les capteurs diagonaux, ce qui va nous permettre de régler l'angle. Si cette différence est plus grande que l'intervalle de [-35,35], ce qui représente un intervalle de [-3.5,3.5] degrés, on fait tourner le robot jusqu'à ce qu'il entre dans cet intervalle. Une fois dans cet intervalle, on fait tourner le robot par des à-coups de 0.5 degré jusqu'à ce que cette différence diminue entrant dans un intervalle plus petit de [-10, 10], correspondant à un intervalle de [-1.5, 1.5] degrés.

Une fois cet intervalle atteint, on s'intéresse aux capteurs latéraux qui vont nous aider à centrer le robot au centre des deux murs. Si la différence des valeurs de ces deux capteurs ne se situe pas dans l'intervalle [-75, 75], correspondant à un intervalle de [-0.5, 0.5] centimètres du centre, le robot va tourner de 50° en direction opposée du mur le plus proche, ensuite avancer de 1cm, puis tourner de 50° dans le sens opposé à tout à l'heure, pour revenir à l'angle initial avant la première rotation. Ces trois étapes sont répétées tant que la différence des valeurs repérées des capteurs IR n'est pas dans l'intervalle voulu : une répétition considérée entre 0 à 3 fois.

Pour la stabilisation de l'angle, nous tournons par des à-coups de 0.5°. Ce processus semble être plus long que de le fait tourner tout simplement sans à-coups, mais cette méthode nous permet d'obtenir une meilleure précision angulaire.

3. Justification des choix

3.1. Choix de la fréquence

Pour la mesure du son, nous devons réussir à identifier les deux peaks présents sur la *Figure 2.* Nous avons donc instauré un seuil minimum de 10'000 pour pouvoir passer outre le bruit ambiant et pouvoir bien détecter nos 2 peaks.

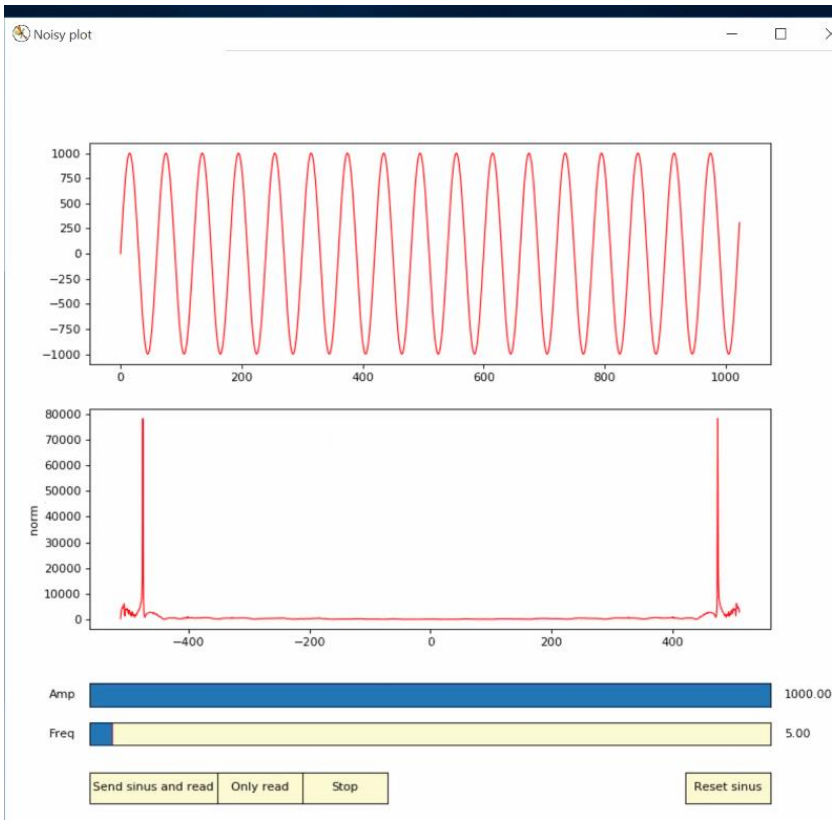

Figure 2 *- Cette figure est prise par Python Script. La partie en haut représente le signal sinusoïdal généré par Python Script plotFFT.py et envoyé au robot. Au-dessous, nous avons les résultats reçus par l'e-puck. Tout en bas, nous avons deux curseurs pour changer la fréquence et l'amplitude du signal. Pour le 1er graphe, l'axe horizontale est en fonction du temps et l'axe verticale est une amplitude, pour le 2ème graphe, l'axe horizontale est en Hz et l'axe verticale est une norme. Nous observons sur le 2ème graphe que le seuil maximum du bruit est autour de 10'000.*

Pour le choix de la fréquence de démarrage du robot, nous avons choisi un intervalle de [500, 600] Hz, pour qu'il ne démarre pas avec des bruits ambiants qui peuvent être crées autour de lui.

3.2. Dimensionnement du labyrinthe

Avant d'avoir dimensionner le labyrinthe, nous devions nous intéresser aux différents capteurs de proximité IR et voir la distance maximum qu'ils peuvent détecter. Cette distance est autour de 9cm. Une fois ces mesures faites, nous avons jugé qu'un labyrinthe dont la largeur est de 14 cm paraît raisonnable. Sachant que le diamètre du robot est de 7cm, il détecterait ainsi le mur s'il était collé contre le mur d'en face.

Les capteurs de proximité IR renvoient une valeur autour de 60-70 lorsque l'obstacle dépasse les 9cm. Nous avons donc décidé de considérer que le robot ne voit pas d'obstacle si le capteur renvoie une valeur inférieure à 80. Pour la détection d'un obstacle en face de lui, afin de tourner en direction du chemin, le robot arrête son parcours lorsque les capteurs d'avant renvoient une valeur de 150 ou plus. Il se retrouve dans ce cas-là à 4.5 cm. A ce moment-là, il tourne de 90 ° (angle droit) en direction de la suite du parcours et se retrouve ainsi au centre des deux murs suivants.

3.3. Déplacement du robot

Nous avons fait le choix d'attribuer au moteur du robot la valeur de 600, pour avancer et tourner. À la suite de ce choix, nous avons pu mesurer le temps que met le robot pour avancer de 1 cm (130 ms) et pour faire un tour de 360° (2088 ms). Cela nous a permis, en activant pendant un certain temps les fonctions de déplacement (pour avancer et tourner), de gérer parfaitement un virage à angle droit ou une distance de déplacement de 10 cm par exemple.

4. Organisation du code

4.1. Principes généraux

Pour stocker les différentes valeurs repérées des capteurs de proximités, nous avons déclaré comme variable globale (en *static uint16_t*) un tableau contenant ces différentes valeurs puisqu'elles sont utilisées à plusieurs reprises le long du fichier qui gère le déplacement. Celles-ci pouvant monter jusqu'à des valeurs de 4'000, 16 bits étaient nécessaires.

Le tableau contenant les données du micro est aussi déclaré en *static* mais cette fois de type *float*, afin d'assurer une meilleure précision.

Pour l'allocation de la mémoire pour le thread *Audio Processing*, étant donné que le thread passe de la détection du son à la récupération des valeurs des capteurs de proximité IR de manière sérielle, nous devions voir laquelle de ces deux fonctions (détection du son et récupération des valeurs des capteurs IR) occupaient plus d'espace dans la mémoire. Etant donnée que le tableau ayant les valeurs du son, est initialisé en *static* et pas dans la fonction elle-même, la fonction en question n'a pas besoin de beaucoup de mémoire. La fonction qui en a besoin de plus est donc celle gérant les capteurs de proximité. Après avoir calculé cette valeur correspondant à 640 bits, nous avons décidé d'en allouer 800 bits pour avoir une petite marge.

4.2. Fonctionnement du programme

Le thread *Audio Processing* gère la machine d'état représentée à la *Figure 3*. Il gère tout d'abord la détection du son. Une fois la bonne fréquence est détectée, il entre dans une fonction nommée *obstacle*. Dans cette fonction, sont générés plusieurs tests pour sonder les alentours du robot. Dans chacun de ces tests, des fonctions sont appelées pour pouvoir déplacer le robot.

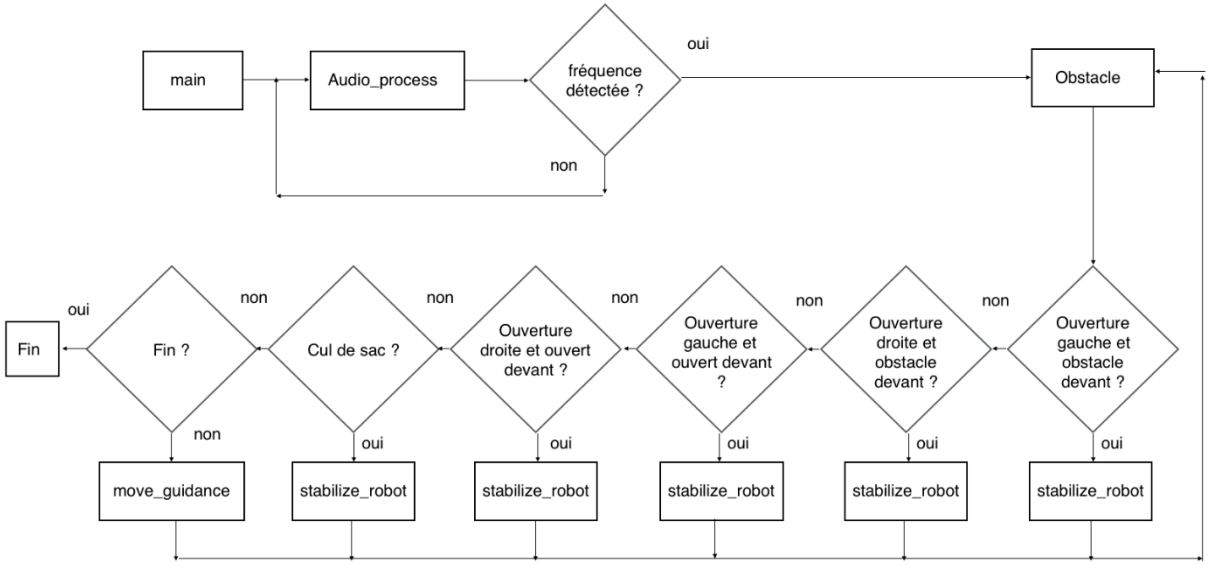

***Figure 3** : Cette figure représente la machine d'état gérée par le thread Audio Processing : d'une partie le processus de détection du son, et d'une autre le cheminement pour détecter les alentours du robot dans le labyrinthe.*

5. Conclusion

Ce mini projet nous a permis de programmer pour la première fois un système embarqué et de réaliser un modèle concret. Cependant, nous avons rencontré quelques difficultés. Premièrement, le travail à distance au début n'était pas évident vu qu'une seule personne parmi nous avait le robot. Mais nous avons pu nous adaptés rapidement à cette situation et l'utilisation de GitHub fut nécessaire et efficace. Deuxièmement, la limitation de la précision du matériel, comme les capteurs de proximité IR et le microphone. Cela nous a engagés à faire plusieurs essais avant de commencer le projet lui-même et qui ont pris une grande partie de notre temps. En effet plusieurs solutions s'offraient à nous pour contrôler ces capteurs et cela était réalisé dans des conditions différentes afin de tester tous les cas possibles. Malgré ces difficultés, ce projet nous a beaucoup appris.

6. Références

- GCtronic, e-puck2, https://www.gctronic.com/doc/index.php/e-puck2 , édité en avril 2020.
- Cours du Professeur Mondada MICRO-315 enseigné aux étudiants de la troisième année de Bachelor en Microtechnique à l'EPFL en 2020.