ECOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

SECTION MICROTECHNIQUE

Systèmes embarqués et robotique
Rapport de Projet

Auteurs :
TNANI Nour
SOUID Zakarya

17 Mai 2021

Table des matières

1 Introduction

Dans le cadre du cours de Systèmes embarqués et robotique, nous avons eu comme projet la programmation d'un robot e-puck2 [1], en se servant de ses librairies internes. Dans ce cadre, nous avons cherché à simuler un labyrinthe en 3D. L'utilisateur devra créer son labyrinthe sur une plate-forme d'au moins 3 cm de hauteur, et faire en sorte que la sortie du labyrinthe soit à la limite de cette plate-forme. Il positionne ensuite le robot à l'entrée du circuit. Le robot va alors se coller au mur à sa droite, et suivra le chemin créé par ces murs. Malheureusement pour notre petit explorateur, à la sortie, il tombe d'une falaise infiniment haute, et crie comme dans Mario en agitant les jambes.

2 Principe de fonctionnement

2.1 Principe algorithmique

Inspirés par Tomb Raider ou Indiana Jones, nous avons voulu rendre notre e-puck capable de sortir de tout labyrinthe traversant, $i.e$ un labyrinthe ayant son entrée et sa sortie sur ses murs extérieurs. Nous avons ainsi retenu l'algorithme de la main droite [5], pour la faible puissance processeur qu'il requiert. Celui-ci fonctionne ainsi :

1. Dès qu'il peut, le robot tourne à droite ;
2. S'il ne peut pas tourner à droite, il essaiera de continuer tout droit ;
3. S'il ne peut ni aller tout droit, ni tourner à droite, alors il tournera à gauche.

2.2 Analyse de l'environnement

Avant de commencer la résolution du labyrinthe, le robot analyse son environnement. Pour cela, il utilise tous ses capteurs infra-rouges (Capteurs IR), le capteur Time Of Flight (TOF) et les moteurs pas à pas. Comme on a choisi de faire suivre au robot le chemin de sa droite, les capteurs essentiels dans la suite du projet seront ceux situés du côté droit, et ceux de devant (1, 2, 3, 4 , 8). Cette section abordera donc plus en détail les différentes étapes nécessaires à son fonctionnement.

2.2.1 Détection du mur le plus proche

À l'allumage du robot, celui-ci rentre dans une phase d'initialisation.
Cette phase se compose d'une première analyse des environs grâce aux capteurs de proximité. Une fonction init_prox() est chargée de comparer les valeurs retournées par les différents capteurs pour trouver celui qui donne la valeur la plus grande ($i.e$. le capteur qui sent l'objet le plus proche). Le capteur retourné par la fonction permet au robot de s'orienter vers l'obstacle le plus proche.

2.2.2 Mouvement vers le mur détecté

Une fois correctement orienté, le capteur TOF est utilisé pour déterminer la distance du robot à l'obstacle. Le robot avance ensuite de cette distance, pour être presque collé au mur. Il pivote ensuite, en utilisant glue_shoulder(), pour coller sa roue droite et son capteur IR_3, ou plutôt sa "main droite" au mur.

2.3 Parcours du labyrinthe

Comme dit au 2.1, nous avons 3 cas principaux différents :

- aller tout droit en suivant le mur ;
- effectuer un virage à droite ;
- effectuer un virage à gauche.

Intéressons-nous d'abord à l'implémentation du premier cas.

2.3.1 Correction du chemin

Pour suivre un mur, il ne suffit pas réellement d'aller tout droit. Nous voulions rendre notre robot capable de s'adapter à une variété d'imprévus, comme des murs courbés et des angles variés. La fonction check_shoulder() nous permet d'accomplir cela. Le but de celle-ci est de vérifier les valeurs des capteurs IR_2 et IR_3 et de décider de l'ajustement de la vitesse des moteurs en fonction des valeurs reçues. Dans le code, nous avons surnommé les capteurs "eye" et "shoulder", et établi expérimentalement les seuils de valeurs que nous estimions trop proche ("close"), trop loin ("far") et sans mur ("nothing").
La réponse des capteurs de proximité n'était pas linéaire.[4] Nous avons donc relevé expérimentalement les valeurs renvoyées par les capteurs, pour corréler le facteur de correction de la vitesse à la proximité au mur du capteur. Nous avons obtenu les courbes de la figure 1 :

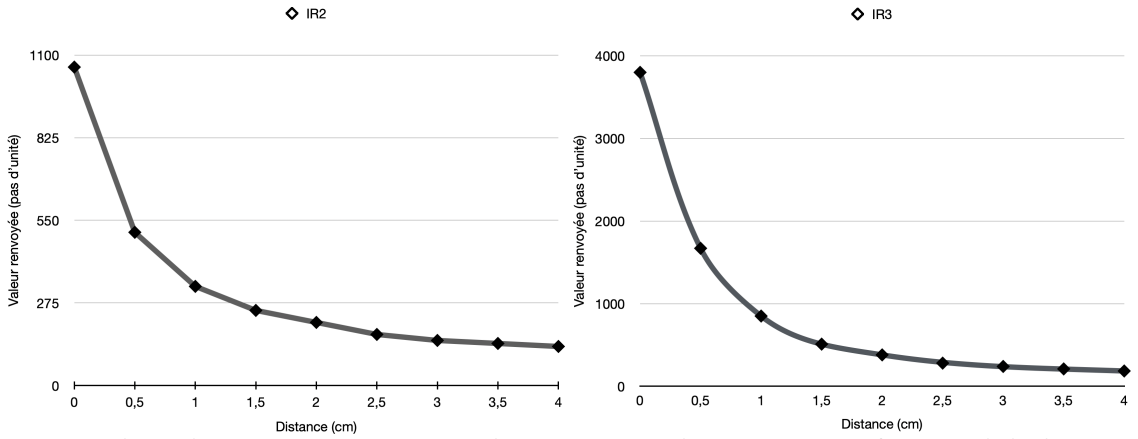

FIGURE 1 – Valeurs de proximité renvoyées par le capteur IR2 et le capteur IR3 en fonction de la distance. Ce diagramme nous a été bien utile lors de la définition de nos limites.

Pour économiser notre petit processeur, nous avons décidé de faire deux approximations linéaires autour de la valeur cible, selon les équations suivantes :

$$\frac{Valeur - Goal}{Goal - Max} * V_{correction} \tag{1}$$

Pour tourner à gauche.

$$\frac{Goal - Valeur}{Goal - Min} * V_{correction} \tag{2}$$

Pour tourner à droite.

Nous avons effectué les mesures plusieurs fois, en changeant la luminosité de l'environnement et en redémarrant le robot, et les variations étaient très faibles, toujours inférieures à 3%. (cf. figure 2)

Valeur des capteurs en fonction de la distance du mur

Distance (cm)	IR2	IR3	IR4	IR3 deuxième mesure	Différence relative
0	1060	3800	225	3780	0,53 %
0,5	510	1670	167	1645	1,50 %
1	330	850	152	845	0,59 %
1,5	250	510	148	513	0,59 %
2	210	380	132	378	0,53 %
2,5	170	280	124	281	0,36 %
3	150	240	122	244	1,67 %
3,5	140	210	111	206	1,90 %
4	130	185	110	189	2,16 %

Variations après redémarrage et changement d'environnement

Distance (cm)	IR3	IR3 deuxième mesure	Différence relative
0	3800	3780	0,53 %
0,5	1670	1645	1,50 %
1	850	845	0,59 %
1,5	510	513	0,59 %
2	380	378	0,53 %
2,5	280	281	0,36 %
3	240	244	1,67 %
3,5	210	206	1,90 %
4	185	189	2,16 %

FIGURE 2 – Détail des valeurs relevées.

2.3.2 Gestion des angles et des virages

Trois situations sont possibles quand il s'agit de tourner (cf figure 3) :

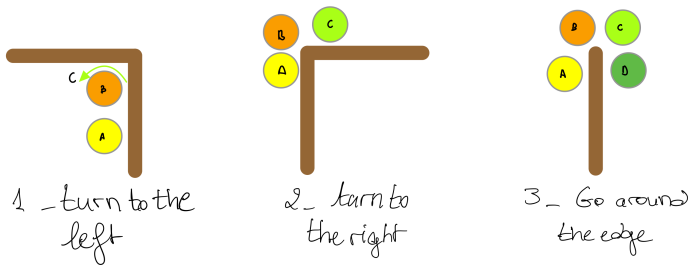

FIGURE 3 – Les différentes situations possibles pour tourner.

1. Pour tourner à gauche : le robot voit un mur à sa droite (A). On doit donc vérifier que les capteurs de devant (IR1 et IR8) renvoient une valeur limite (B). Ceci vérifié, le moteur fait des tours sur lui-même de 20° vers la gauche (C). Il retourne ensuite dans la boucle principale lui permettant d'avancer en suivant le mur
2. Si le robot arrive au bout d'un mur, deux situations sont possibles. Dans les deux, s'il ne voit plus rien de son "oeil", mais toujours de son "épaule", il ne tourne pas et continue jusqu'à ce que IR3 ne capte plus rien lui non plus. Il se met alors à tourner vers la droite, jusqu'à ce qu'il capte à nouveau un obstacle. Cela couvre ainsi les scénarios 2 et 3. Après cela, il reprend son

chemin le long du mur.

2.4 Sortie

À la fin du programme, nous utilisons comme capteur l'Inertial Motion Unit (IMU) [3] et le gyroscope. Quand notre robot sort du labyrinthe, il tombe d'un angle minimum. Ceci induit une rotation selon l'axe X. Une comparaison des valeurs rapportées et des valeurs limites lance donc la musique. Le robot est arrivé à la fin de son périple, mais il n'a pas été assez attentif pour éviter la chute.

2.5 Intéractions entre les Threads

Finalement, le programme fonctionne avec les threads suivantes :

- move_right_start : gère la résolution du labyrinthe (NORMALPPRIO)
- fall_monitoring_start : gère la sortie du robot du circuit (NORMALPRIO)
- Ainsi que les threads de la librairie propres aux capteurs : TOF (NORMALPRIO), capteurs IR (NORMALPRIO), capteurs IMU (NORMALPRIO), et aux buffers (NORMALPRIO).

Ces différentes threads sont appelées dans le Main. Une gestion particulière [2] n'est pas nécessaire dans notre cas, puisque les deux principales threads sont indépendantes. En effet, la condition pour déclencher le deuxième module ne dépendra que des conditions externes (pente), et non de résultats de modules précédents.

3 Résultat

3.1 Premiers résultats

La première implémentation de notre code consistait en une utilisation basique des différents modules évoqués précédemment. Le robot continuait tout droit tant que le mur était à une distance minimale, tournait de 90° à gauche si un obstacle est détecté en face, sinon si il fallait tourner à droite, le robot avançait d'une distance déterminée expérimentalement puis faisait un tour de 90° à droite.
Plusieurs problèmes ont été rencontrés avec ce genre d'implémentation. Entre autres, si pour une quelconque raison le robot avance d'une façon pas droite, il s'éloignera au fur et à mesure du mur à sa droite, et les conditions ne seraient plus vérifiées.

3.2 Améliorations mises en oeuvre

Afin de remédier à ce problème, la fonction check _ shoulder() ,évoquée en section 2, permet de réguler l'angle de mouvement au fur et à mesure, grâce à des comparaisons entre des valeurs limites et les valeurs de proximité renvoyées par les capteurs de droite.
Cette même fonction nous a aussi donc permis de suivre les murs peut importe leur angle, et ne pas être limité à des virages de 90°. Le robot peut donc suivre le chemin du labyrinthe que les murs soient complètement droit, incurvés, ou circulaires.

Le coefficient de rotation SPEEDK a été calculé pour tourner de manière appropriée, peu importe le mur ou son épaisseur. Notre programme est facilement adaptable à n'importe quelle valeur de vitesse de moteur souhaitée, tant qu'elle reste inférieure à 1200 steps/s. Il suffira à ce moment de changer la constante SPEED définie dans le main.h .

3.3 Limites

Le principal défaut de notre implémentation réside selon nous dans l'incapacité de contrôler précisément la distance du robot au mur. Dans les faits, nous sommes presque obligés qu'il soit collé la majeure partie du temps. Nous voulions utiliser l'IMU pour plus, mais il s'est révélé très capricieux et nous n'avons pas réussi à en faire ce que nous avions prévu.

4 Conclusion

Après des projets de programmation de software, et d'autres de conception mécanique, ce projet a été pour nous le premier qui combine ces deux aspects, nous offrant ainsi une première expérience de programmation d'un système physique. Nous nous sommes vite rendus compte des difficultés engendrées par le hardware utilisé. Contrairement à une programmation software à laquelle on s'est habitué, ce projet nécessitait continuellement des expériences et des essais, des prises en compte de variables qu'on peut difficilement contrôler, i.e l'atmosphère ambiante (les variations de la luminosité alors qu'on utilise des capteurs IR). En bref, l'aspect programmation de ce projet ne nous a pas pris énormément de temps. Ce fut plutôt le debuggage qui nous a pris le plus de temps.
Finalement, nous avons beaucoup apprécié travailler sur ce projet, malgré les contraintes de la situation sanitaire.

Références

[1] GCtronic, *e-puck2*. `https://www.gctronic.com/doc/index.php/e-puck2`

[2] *Mondada, F. (2021), TP4 : CamReg*. `https://moodle.epfl.ch/course/view.php?id=467#section-5`

[3] *Mondada, F. (2021), TP3 : IMU*. `https://moodle.epfl.ch/course/view.php?id=467#section-4`

[4] Vishay Semiconductors, *Reflective Optical Sensor with Transistor Output*. `https://www.vishay.com/docs/83752/tcrt1000.pdf`

[5] Wikipedia, *Résolution de labyrinthe* . `https://fr.wikipedia.org/wiki/RÃ©solution_de_labyrinthe`