E-Puck Semi-Autonome : Suivi de ligne et détéction d'intersection

Microinformatique : Miniprojet - Prof. Mondada

Printemps 2019

KIMBLE Thomas SCIPER: 261204
OHANA Adrien SCIPER: 250619

Section: MT

Contents

1 Introduction

Le but de ce miniprojet est d'implémenter un système simplifié de conduite semi-autonome inspiré des options actuellement disponibles chez une grande partie des constructeurs automobiles.

Les options considérées sont la détection de ligne et son suivi, le régulateur de vitesse adaptif, et le changement de voie par activation de clignotant. Nous avons revisité cette dernière option afin d'obtenir une application plus appropriée à notre projet : la détection d'une intersection suivie d'une attente d'instruction par le "conducteur" qui choisit dans quelle direction se diriger après chaque intersection.

Afin de pouvoir suivre les lignes précisément, nous avons fabriqué un support pour un miroir placé devant la caméra.

2 Guide d'Utilisation

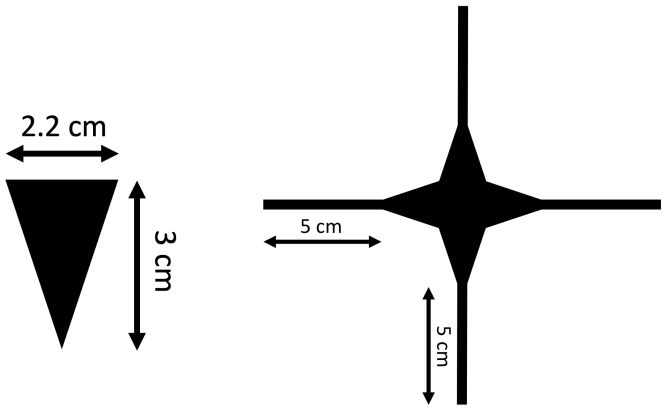
2.1 Contraintes

Nous garantissons le fonctionnement du système dans les conditions suivantes :

- Avant tout : le robot doit porter le support qui contient le miroir.
- Le robot doit suivre une ligne noire sur fond blanc d'une épaisseur comprise entre 4mm et 10mm dont le rayon de courbure est supérieur à 75mm.
- La surface sur laquelle travaille le robot est plane.
- Une intersection sera représentée par un triangle isocèle avec une base de 22mm et une hauteur de 30mm comme sur la Figure 1a. Le robot détecte une intersection si la ligne noire qu'il suit a une épaisseur comprise entre 16mm et 22mm, donc un triangle assure qu'il observe ce cas.
- Les lignes connectées à une intersection doivent être droites sur au moins 5cm et perpendiculaires entre elles. Une intersection type est montrée sur la Figure 1b.
- Les instructions reçues par le robot sont des sinus de fréquences audibles comme indiqué dans la Table 1.

Sound Remote	
Sinus Frequency	Instruction
2000 Hz	Skip Stop
3000 Hz	Left Turn
4000 Hz	Right Turn
5000 Hz	U Turn

Table 1: Instructions Sonores



(a) Triangle Stop (b) Intersection avec 4 lignes

Figure 1: Représentation et dimensionnement d'intersections

- Le robot doit démarrer centré sur une ligne droite d'au moins 5cm et orienté dans sa direction.
- La direction choisie après une intersection doit déboucher sur une ligne.
- La luminosité de l'environnement ne doit être ni extrême ni anisotrope afin de permettre la détection robuste d'intersections. Ceci peut être assuré la plupart du temps avec un enclos en carton.

2.2 Fonctionnement

À la mise sous tension, le robot est en mode intersection, il est donc à l'arrêt. Placer le robot sur une ligne droite d'au moins 5cm, en s'assurant qu'elle soit dans le champ de vision de la caméra et que l'e-puck2 est orienté dans la direction de la ligne. Envoyer l'instruction *Skip Stop* avec un générateur de sinus à 2000Hz. Le robot suivra la ligne jusqu'à la prochaine intersection.

Arrivé à un intersection, le robot s'arrête. Envoyer alors l'instruction désirée : *Skip Stop* pour aller tout droit, *Left Turn* pour tourner à gauche avec indicateur, *Right Turn* pour tourner à droite avec indicateur ou *U Turn* pour faire demi-tour avec indicateur.

3 Structure du Code

3.1 Structure Générale

En plus d'utiliser la librairie *ChibiOS*, nous avons créé notre propre libraire contenant trois fichiers supplémentaires afin de contrôler notre e-puck2 : process_image.c, control.c et audio_processing.c .

Le fichier *process_image.c* utilise la camera afin de détecter la présence et les caractéristiques d'une ligne à suivre. *audio_processing.c* utilise un microphone afin de détecter des maxima de fréquence pour envoyer une instruction a *control.c*, qui contient toutes les fonctions pour contrôler les mouvements des moteurs, le capteur Time of Flight, la recherche d'intersection, ainsi que les LEDs. Nous introduisons dans ce dernier fichier la variable globale *intersect* qui servira comme variable d'état.

- Si *intersect* est nulle nous sommes dans un premier état avec suivi de ligne, régulation PI de la vitesse grâce au Time of Flight et recherche d'intersection, géré donc par *Control* et *Process Image*.
- Si *intersect* est vraie nous sommes dans l'état Stop avec une attente d'instruction sous forme de maxima de fréquence qui va engendrer une fonction moteur, géré par le fichier *Audio Processing*.

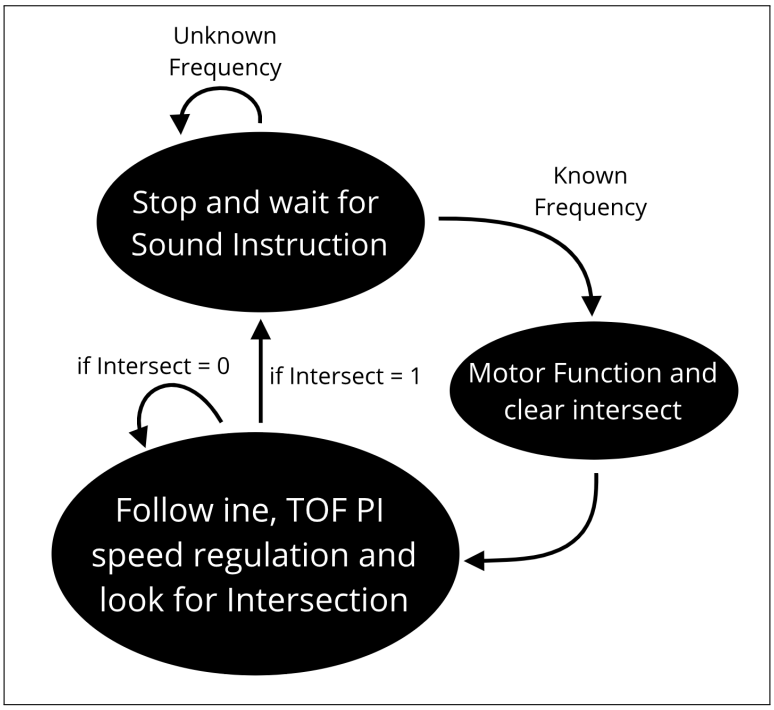


Figure 2: Machine d'état du Miniprojet

3.2 Process Image

Le fichier *process_image.c* contient le Thread *CaptureImage* qui sert a capturer une ligne de la camera po8030 en format RGB565. Nous choisissons la ligne 350 car elle capture les pixels juste devant notre e-puck grâce au miroir.

Une fois la ligne voulue de l'image capturée, le Thread *ProcessImage* va en extraire les pixels verts. En moyennant toutes les valeurs extraites pour ensuite trouver un début (pente

descendante) et une fin (pente ascendante) de ligne comme dans le TP4, l'épaisseur et la position de la ligne sont obtenues et stockées "in place".

3.3 Control

Le fichier *control.c* contient le Thread *SpeedRegulator* qui utilise le capteur Time of Flight et gère les moteurs pour le suivi de ligne. On utilise ici un *get* pour obtenir la largeur de la ligne du fichier *Process Image*. Si la ligne a une largeur entre 300 et 450 pixels et que l'on peut le vérifier une dizaine de fois sur plusieurs dixièmes de millimètres, il s'agit d'une intersection. La variable d'état est fixée vraie et la vitesse des moteurs est mise à zéro.

Si nous ne sommes pas dans un cas d'intersection nous vérifions déjà qu'il n'y a pas d'obstacles devant le e-puck grâce au Time of Flight. Si un obstacle est trop proche, c'est-à-dire à moins de 12 cm du robot, la vitesse est gérée par un régulateur PI qui permet d'avancer à la même vitesse que l'obstacle sans le rattraper. Si l'obstacle est immobile le robot n'avancera plus, nous évitons donc tout accident frontal.

Dans un dernier cas, si nous sommes ni en présence d'intersection, ni en présence d'obstacle, la vitesse est gérée par le sélecteur et la correction pour les virages est gérée par une moyenne mobile de la position de la ligne obtenue aussi avec un *get*. Nous utilisons une moyenne mobile et un coefficient de rotation afin d'éviter des déplacements soudains pour rendre le virage plus lisse.

En dehors du Thread *SpeedRegulator* le fichier contient les fonctions moteurs post-intersection, c'est à dire les virages, le demi-tour ainsi que l'instruction de continuer tout droit. On utilise ici des boucles *for* pour que les moteurs tournent pendant un certain temps choisi expérimentalement pour donner le déplacement voulu. Nous nous permettons d'utiliser ces boucles car elles assurent que le robot ne fait rien d'autre pendant les virages. Ceci nous limite à des virages de 90°, mais nos circuits sont conçus pour ce fait. Finalement la fonction *turn_signal* gère les *LED3* et *LED7* qui représentent des clignotants. Ces fonctions seront utilisés dans le fichier *audio_processing.c* après avoir reçu une instruction sonore.

3.4 Audio Processing

Le fichier *audio_processing.c* contient la fonction *processAudioData* qui a pour but d'écouter un son grâce au microphone gauche du e-puck et en faire une transformée de Fourier optimisée pour pouvoir l'interpréter efficacement, et la fonction *sound_remote* pour agir en fonction. Dans *processAudioData* les valeurs de la FFT sont normalisées et stockées dans un tableau que nous envoyons à la fonction de commande des moteurs *sound_remote* (si et seulement si $intersect = 1$). Ce tableau est balayé afin de trouver un maximum qui, selon sa valeur, commandera les fonctions moteurs (demi-tour, virages...) ainsi que les fonctions de LEDs clignotantes.

4 Conclusion

Au terme de ce projet, nous nous devons de souligner l'atout qu'a été l'utilisation d'un RTOS pour gérer les différentes tâches à exécuter, même pour un si petit projet. Les threads du microphone, du capteur TOF, de la caméra, et de la régulation de vitesse sont tous de même priorité et agissent ou non en fonction de la variable d'état *intersect* statique et globale. Cette variable est modifiée ou lue par les fonctions "in place" de type *get*, *set* et *clear*, tout le reste est géré par ChibiOS.

Enfin, l'aspect conception du support de miroir nous a permis de faire usage de nos connaissances d'autres cours pour proposer une solution originale à notre problème de suivi de ligne. Bien que le temps passé sur ces détails n'ait rien avoir avec ce cours (CAO, impression 3D, différents essais), cela aura été une véritable expérience pluridisciplinaire avec un face à face entre les limites "hardware" et "software".