

Rapport du Miniprojet de Systèmes Embarqués et Robotique

Auteurs

COUYOUMTZELIS Romain Nicolas Paul	340933
DEMIERRE Gautier	340423

Groupe 33, Section MT-BA6

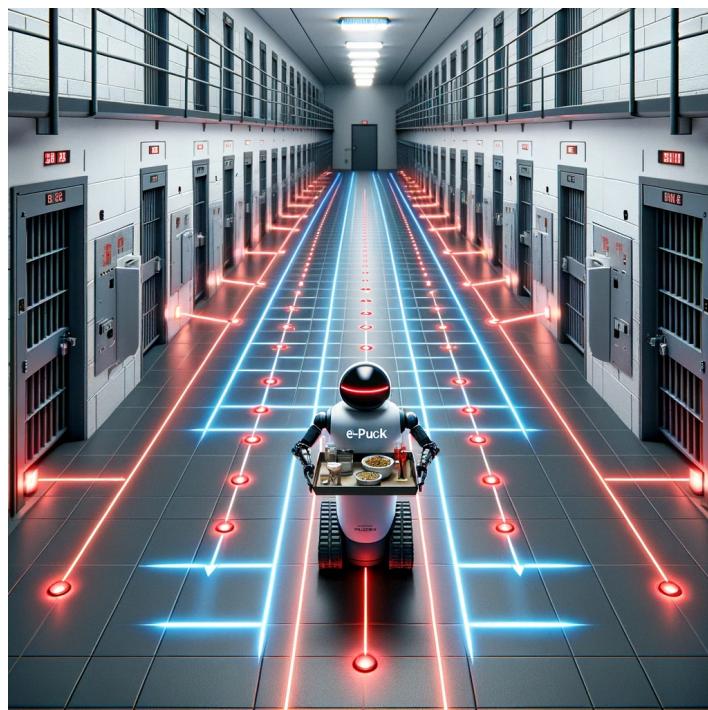


Table des Matières

1	Description générale de l'application	1
2	Utilisation des périphériques	2
2.1	Caméra	2
2.2	Capteurs Infrarouge	2
2.3	Moteur	2
3	Fonctionnement détaillé	3
3.1	Répartition entre les fichiers	3
3.2	Thread <i>ColorDetection</i>	3
3.3	Thread <i>Movement</i>	4
3.4	Thread <i>ProxDetection</i>	5
4	Mécanisme du Scheduler	6
5	Gestion de la mémoire	7
6	Communication entre les threads	8
6.1	<i>ProxDetection</i> vers <i>Movement</i>	8
6.2	<i>ColorDetection</i> vers <i>Movement</i>	8
7	Conclusion	9
8	Annexes	10
8.1	Conception et Imprimerie 3D d'extensions pour le robot	10
8.2	Références	10

1 Description générale de l'application

Notre projet consiste à implémenter un programme permettant au robot E-PUCK2 la distribution de plateaux-repas dans les prisons. Il est ainsi capable de se déplacer à travers les couloirs du pénitencier, détecter si une cellule doit être servie ou non, lui apporter le plateau jusqu'à la porte cas échéant, repartir, et changer de rangée lorsqu'il a fini celle en cours. Le programme est aussi doté d'une fonction *Emergency*, permettant au robot de se mettre en sécurité lors d'alertes ou d'émeutes.

2 Utilisation des périphériques

Pour fonctionner, l'E-PUCK utilise la caméra, les détecteurs IR4 et IR5 ainsi que les moteurs. Un support comportant un miroir est fixé sur le dessus de l'E-PUCK. Ce miroir est incliné à $\frac{\pi}{4}$ vers le bas, ce qui permet à la caméra de scanner directement le sol.

2.1 Caméra

La caméra permet de transmettre les instructions de déplacements grâce aux couleurs captées. Ainsi selon les différentes bandes de couleurs scannées, le robot effectuera différentes tâches.

Bande Rouge : Indique qu'une cellule doit être servie.

Bande Bleu : Indique que le robot est revenu dans sur sa ligne de trajectoire principale et est prêt à repartir.

Bande Verte : Indique au robot qu'il a fini la rangée et qu'il peut passer à la suivante.

Bande Blanche : Signal une alerte dans la prison et indique au robot de se coller au mur et de ne plus bouger.

Ainsi, une fois les bandes détectées, l'information de la couleur en question est transmise à une machine d'état finis.

2.2 Capteurs Infrarouge

Les capteurs vont permettre d'indiquer au robot lorsqu'il est à la bonne distance de la porte pour donner le plateau à travers l'ouverture. Nous avons choisi d'utiliser les 2, IR4 et IR5 ensemble, plutôt que le capteur TOF évitant tout phénomène redondant , car ce dernier pourrait ne pas s'arrêter suffisamment tôt et percuter la porte s'il se trouvait déphasé d'un certaine angle. Grâce à l'utilisation simultanée des deux capteurs, nous effectuons implicitement deux tests de conditions afin de pallier à un déphasage éventuel.

2.3 Moteur

Les moteurs vont executer les mouvements reçus par les capteurs. Les mouvements sont soit en ligne droite, soit en rotation. Les rotations du robot peuvent être $\frac{\pi}{2}$ et π sur lui-même, dans le sens horaire ou anti-horaire. À noter que le robot avance avec la caméra dans le dos et les IR4-5 devant, c'est-à-dire que les moteurs fonctionnent avec des vitesses négatives.

3 Fonctionnement détaillé

Pour faire fonctionner les moteurs, la caméra et les capteurs IR4-5 en même temps, nous utilisons le multi-threading. Nous avons donc 3 threads qui tournent en continue et de façon collaborative.

3.1 Répartition entre les fichiers

Il convient également de donner un bref tableau récapitulatif qui nous permet d'indiquer au travers de quels fichiers .c les différents périphériques sont utilisés. Par ailleurs, nous rajouterons que la communication entre les différents périphériques (threads) grâce au bus-message.

Périphérique	Caméra	Capteurs Infrarouges	Moteur
Fichier de gestion	color_detection.c	prox_detection.c	move.c

TABLE 3.1 – Tableau résumant la gestion des périphériques par les différents fichiers .c

3.2 Thread *ColorDetection*

Ce thread permet la capture ainsi que l'analyse d'une image. Une fois l'image capturée par la caméra, les pixels RGB sont chacun récupérés et placés dans 3 variables respectives. Une moyenne d'intensité (sur 255) est ensuite effectuée sur l'ensemble des pixels de chacune des 3 couleurs différentes. Ce qui nous donne au final une moyenne d'intensité pour les 3 couleurs élémentaires de l'image capturée. Ces moyennes sont ensuite comparées dans des boucles conditionnelles afin de déterminer quelle couleur prédomine sur l'image prise. Des seuils de "noir" et "blanc" sont utilisés lors des comparaisons, et doivent possiblement être recalibrés lorsque la luminosité ambiante change.

Pour obtenir le bon fonctionnement de la caméra, un calibrage doit être fait lorsqu'on change de milieu afin de changer les seuils. De plus il est important de noter que l'Auto_White_Balance est initialisé à 0 afin d'avoir des différences d'intensité plus grandes entre les couleurs, ce qui permet une confiance plus large et accrue dans l'exécution du programme.

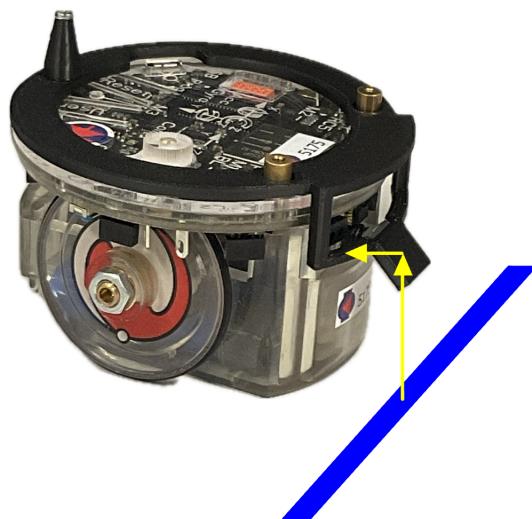
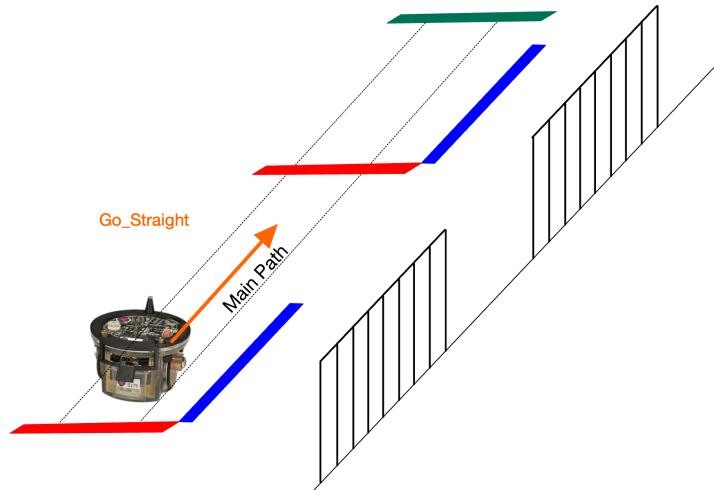


FIGURE 3.1 – Schéma simplifié du parcours moyen de la lumière (en jaune) à analyser vers le miroir puis vers la caméra

3.3 Thread Movement

Ce thread comporte la machine d'états finis permettant une succession logique des états de l'exécution. La machine comporte alors 4 états :



GO_STRAIGHT : Correspond à l'état de "croisière". Le robot roule à une vitesse constante en attendant un signal de la part d'un capteur (Caméra ou IR).

FIGURE 3.2 – Schéma de l'exécution de Go_Straight.

DELIVERY : Correspond à la distribution d'un plateau dans la cellule. L'opération consiste en une rotation de $\frac{\pi}{2}$ puis un allé en ligne droite vers la cellule. Le robot roule et ne s'arrête que lorsque le sémaphore `wall_sem` est libéré par le thread `ProxDetection`.

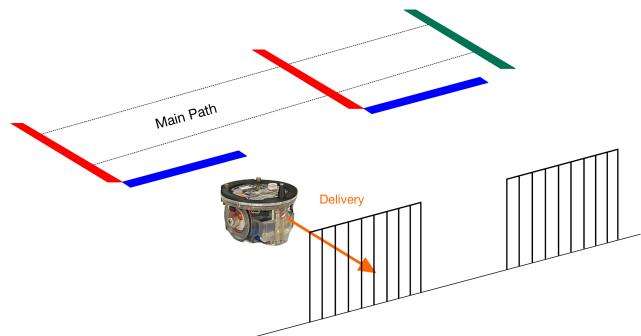


FIGURE 3.3 – Schéma de l'exécution de Delivery.

RETURN : Correspond au retour du robot sur sa trajectoire de distribution après avoir déposé le plateau. L'opération consiste en une rotation de π puis un retour en ligne droite et une rotation de $\frac{\pi}{2}$, après la détection d'une ligne bleue, afin de se remettre dans sa trajectoire. Pour finir, il avance de quelques centimètres dans sa trajectoire, afin de passer la ligne rouge de la cellule et ne pas la redétecter.

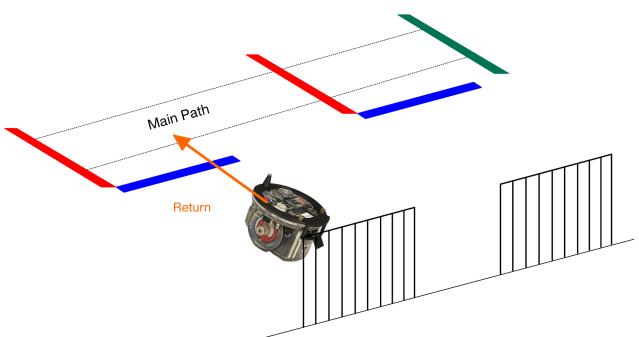


FIGURE 3.4 – Schéma de l'exécution de Return.

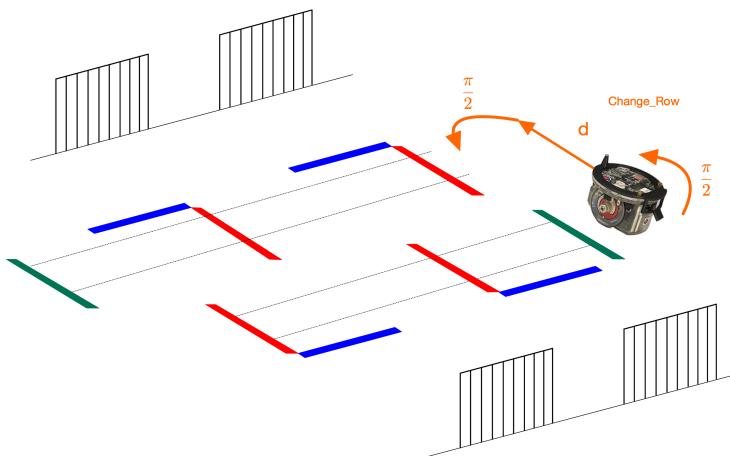


FIGURE 3.5 – Schéma de l'exécution de Change_Row

EMERGENCY : Correspond à la mise en sécurité du robot lors d'alerte. Il procède par rotation à $\frac{\pi}{2}$ puis en ligne droite jusqu'au mur (le robot sera arrêté par le signal des capteurs IR4 et IR5). Cela lui permet de libérer le passage pour l'afflux de personnes sans s'endommager. De plus l'entrée du robot dans cet état est faite par la détection de lumière blanche, ce qui est très facile à faire, et ce à n'importe quel moment (allumage d'un gros spot). Après ce cas, il est possible de rallumer le robot afin qu'il reprenne sa course. Il peut repartir de n'importe où tant qu'il est dans sa trajectoire.

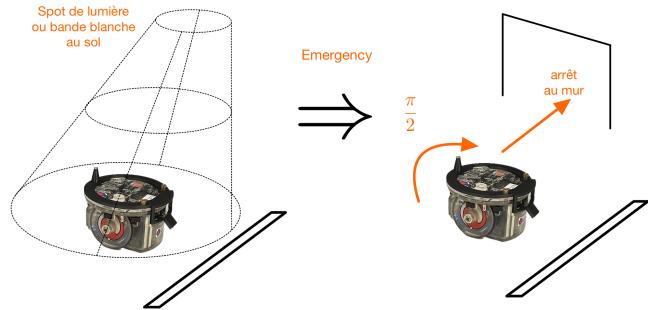


FIGURE 3.6 – Schéma de l'exécution de Emergency.

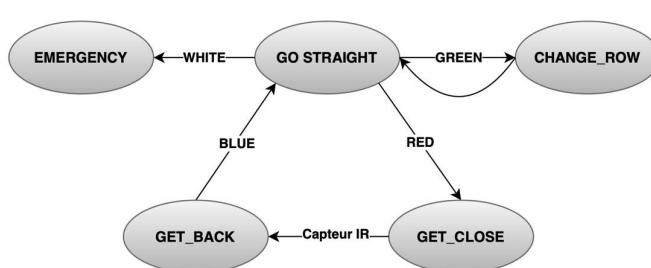


FIGURE 3.7 – Machine d'état fini de l'exécution du programme

3.4 Thread *ProxDetection*

Ce thread utilise les capteurs IR4 et IR5, en récupérant dans chacun la proximité qu'ils renvoient. Ces proximités sont sommées et ensuite comparées à un seuil fixé. Ce seuil doit donc être ajusté selon usage, en se rappelant que c'est une proximité, suivant donc une relation inversement proportionnelle à la distance à des facteurs près.

CHANGE_ROW : Correspond au changement de rangée, une fois que toutes les cellules de la précédente ont été servies. Le processus commence après avoir scanné une ligne verte. Il consiste en une rotation de $\frac{\pi}{2}$, une ligne droite d'une distance d et une autre rotation de $\frac{\pi}{2}$ pour se remettre dans la trajectoire de ligne principale.

Afin de mieux visualiser l'exécution, il nous a paru judicieux de fournir la machine d'états finis sous forme de diagramme bloc comme ci-contre :

4 Mécanisme du Scheduler

Il a été décidé de mettre *Movement* à `NormalPrio+1` et tous les autres threads à `NormalPrio`, étant donné que *Movement* est le thread principal par lequel converge les données des différents capteurs, permettant ainsi au programme d'avoir une meilleure réactivité.

Thread	ColorDetection	Movement	ProxDetection	Main
Priorité	NormalPrio	NormalPrio+1	NormalPrio	NormalPrio

TABLE 4.1 – Ordre de priorité des threads.

5 Gestion de la mémoire

Une fois la commande `Make` appliquée et la compilation terminée, nous utilisons la mémoire de la manière suivante :

text	data	bss	dec	hex	filename
88036	3764	21500	113300	1ba94	build/CamReg.elf

FIGURE 5.1 – Utilisation de la mémoire.

Avec ces valeurs, nous sommes maintenant en capacité de connaître le pourcentage de la mémoire Flash et RAM que nous utilisons. D'abord, il convient de rappeler que *text* correspond au segment pour le programme en mémoire Flash. En ce qui concerne *data*, il s'agit du segment pour les variables globales et variables statiques initialisées non-nulles et se situent en RAM. *bss* constitue lui aussi en un segment en RAM pour les variables globales et variables statiques non-initialisées ou initialisées nulles.

$$\text{flash}_{\text{used}} = 100 \cdot \frac{\text{text}}{10^6} = 100 \cdot \frac{88036}{10^6} = 8.80\% \quad (5.1)$$

$$\text{RAM}_{\text{used}} = 100 \cdot \frac{(\text{data} + \text{bss})}{192000} = 100 \cdot \frac{(3764 + 21500)}{192000} = 13.16\% \quad (5.2)$$

Comme notre programme n'utilise qu'une partie restreinte de la mémoire flash et RAM de l'e-puck, il serait possible d'envisager de faire fonctionner l'exécution en combinaison avec d'autres applications. Il est aussi important de donner la taille que nous avons décidé d'allouer à nos threads afin de garantir une exécution correcte et optimisée en terme de mémoire, notamment au niveau de la pile :

Thread	ColorDetection	Movement	ProxDetection
Taille en Byte	1024	512	128

TABLE 5.1 – Mémoire allouée aux threads

6 Communication entre les threads

6.1 *ProxDetection* vers *Movement*

Ce que l'on souhaite transmettre entre les deux threads est un message binaire qui nous dit si le robot est assez près du mur. Dans ce cas, l'utilisation d'un sémaphore *wall_sem* est la plus adaptée. De plus ce sémaphore est libéré à travers une fonction *wait_until_wall()* permettant d'éviter la libération prématurée du sémaphore lorsque ce dernier n'est pas sur le chemin du retour (lorsqu'un gardien passe devant le robot par exemple).

6.2 *ColorDetection* vers *Movement*

Le message que l'on souhaite transmettre n'est plus binaire, mais multiple, car il doit pouvoir représenter le rouge/bleu/vert/noir/blanc. Ainsi l'utilisation des messagebus est plus adapté dans ce cas là. À chaque image prise et analysée, la couleur détectée est alors envoyée au thread *Movement* sous le topic *color_topic*.

7 Conclusion

Pour conclure, la réalisation d'un projet tel que celui-ci aurait une application réel. Elle permettrait de soulager le corps pénitentier ainsi qu'éviter les problèmes d'agression ou bien même de corruption des agents. De plus, le principe d'utilisation reste très simple. Une fois les leds installées au sol, le robot effectue son programme comme le veut son utilisation standard. Mais les leds peuvent aussi s'allumer de différentes couleurs au même emplacement, ce qui permet d'executer des tâches ou parcours personnalisés, sans avoir besoin de toucher au robot et en contrôlant simplement l'allumage et l'extinction des leds (les fonctionnalités étant préenregistrées selon les detections).

8 Annexes

8.1 Conception et Imprimerie 3D d'extensions pour le robot

Afin de rendre possible la détection d'image, il fut nécessaire d'imprimer en *PLA* (*polylactic acid*) qui permet de soutenir le miroir. Cela nous permettra de détecter les bandes de couleurs situées au sol qui ne sont pas détectables par la caméra normalement. Afin de maintenir l'anneau sur le robot, un cône de maintien a été usiné et est fixé grâce à la vis rendu disponible de base sur le E-Puck.

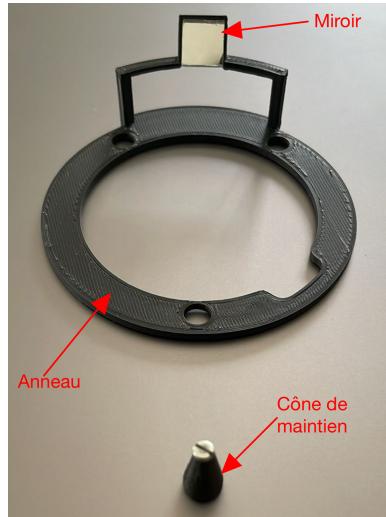


FIGURE 8.1 – Extensions Anneau, Miroir et Cône de maintien

8.2 Références

L'image de garde a été générée par assistance IA avec ChatGPT. Toutes les autres images ont été créées par nous-même. Sur certaines images de la section 3, le robot E-PUCK apparaît grâce à un découpage d'une photo de ce dernier. Finalement, la [Figure 5.1](#) est une capture d'écran tirée du terminal VisualStudio Code obtenue suite à la compilation de notre programme.