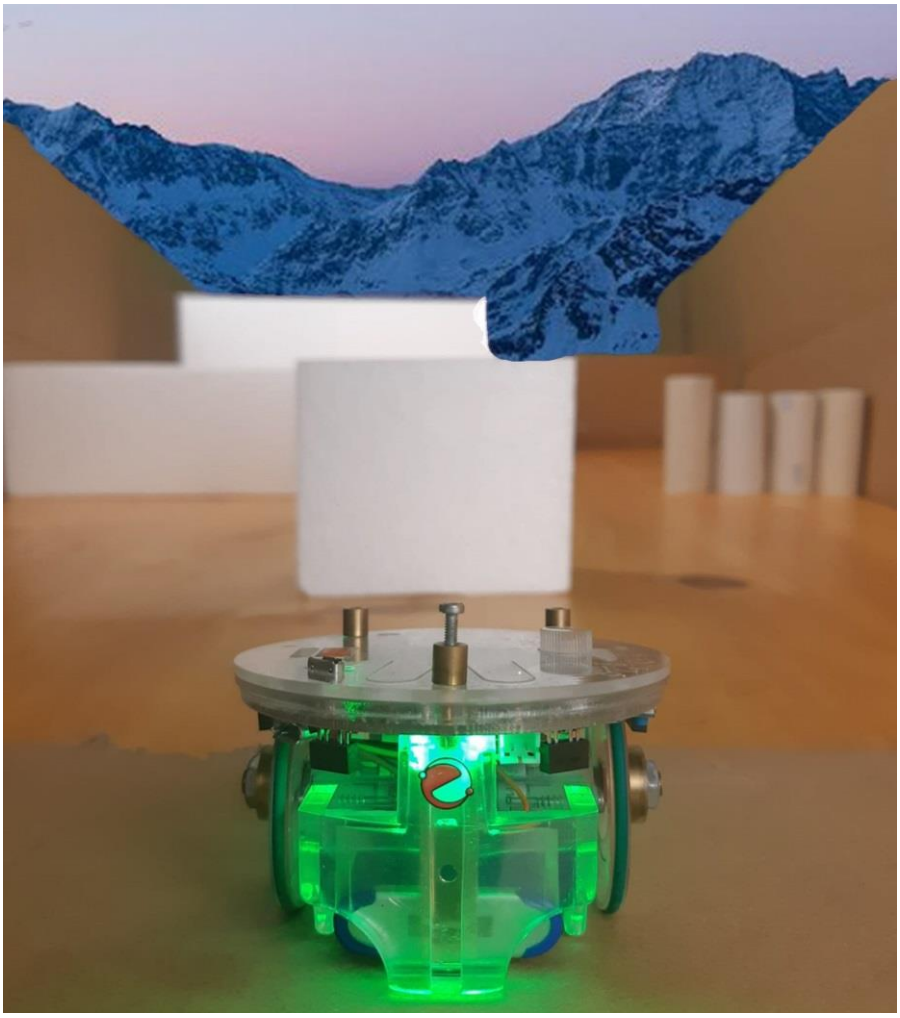
Rapport de projet : *L'Alpiniste*



Réalisé par le Groupe 19 :

Imstepf Laetitia - *SCIPER 296029*

Broccard Brendan - *SCIPER 301452*

Section MT-BA6

Table des matières

1. Introduction 2
2. Principe de fonctionnement 2
2.1 Trouver la pente 2
2.2 Contourner les obstacles 4
2.3 Autres fonctionnalités (moteurs pas à pas et LEDs) 5
3. Organisation du code 5
4. Résultats 7
5. Conclusion 8
Bibliographie 8

1. Introduction

Dans le cadre de ce projet, notre robot e-puck2 doit, tel un alpiniste, gravir une pente tout en évitant les obstacles sur son chemin. Pour ce faire, nous avons exploité les données des capteurs de proximité infrarouges ainsi que l'IMU, plus précisément l'accéléromètre. Les moteurs pas à pas sont donc utilisés afin de contourner les obstacles et monter la rampe.

2. Principe de fonctionnement

Les deux principales tâches de l'e-puck sont donc de se tourner dans la direction du sommet de la pente et de contourner les obstacles qui se présentent sur son chemin.

2.1 Trouver la pente

Afin de trouver la direction du sommet de la pente, le code dédié au traitement des données, plus précisément la fonction move_towards_up(), analyse les valeurs mesurées par l'accéléromètre (sur les axes x et y). Si ces mesures dépassent la valeur du threshold (déterminée expérimentalement), le robot tourne sur lui-même vers la gauche ou vers la droite jusqu'à ce qu'il se positionne dans la bonne direction. Une fois dans cette direction, il avancera tout droit, jusqu'à ce qu'il ne soit plus positionné en direction du sommet de la pente. Il est à noter que le thread principal, qui appelle la fonction move_towards_up(), focalise le robot sur l'appel répété de cette fonction et l'acquisition des données mesurées par l'accéléromètre de l'IMU, et cela jusqu'à ce que la bonne direction aie été trouvée. Durant ce temps, étant donné que le robot tourne sur lui-même et ne risque donc aucune collision, le thread ne démarre pas de processus de détection/esquive d'obstacle, décrit au point 2.2.

Nous avons donc dû déterminer une valeur de seuil telle qu'elle ne soit pas si grande que le robot ne détecte aucune pente et si petite que le robot détecte le moindre bruit. Pour ce faire, nous avons étudié les valeurs de l'accélération mesurées sur les axes x et y. Les *figures 1,2 et 3* sont les valeurs reçues par Hterm en temps réel de l'accélération mesurée sur l'axe x (get_acc_filtered avec un filter_size de 50). Afin de nous aider à trouver un bon threshold, nous avons fait des mesures lorsque le robot e-puck est penché une fois dans la bonne direction (*figure 1*) c'est-à-dire le minimum d'accélération et à 90° de la position vertical (*figure 2*) c'est-à-dire le maximum d'accélération. Les valeurs se situent sur une plage de -100 à 3'000. Une fois ces valeurs en tête, nous avons placé le robot dans une position que nous jugions comme ayant une marge suffisante. La *figure 3* montre bien que la valeur de l'accélération à cette position est d'environ 500, c'est pour cette raison que nous avons choisi un tel seuil.

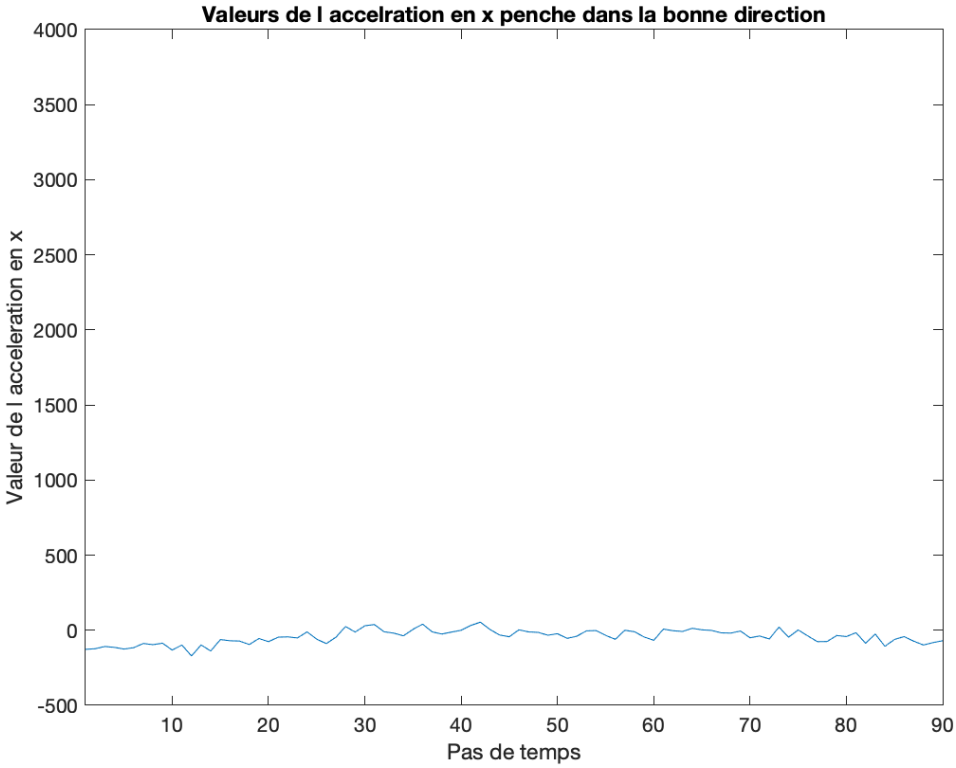


Figure 1 : échantillons de 90 valeurs de l'accélération en x (avec filter_size de 50) lorsque l'e-puck est penché dans le sens de la pente, avec le numéro d'échantillon sur l'axe des abscisses

Figure 2 : échantillons de 90 valeurs de l'accélération en x (avec filter_size de 50) lorsque le robot est tourné à 90° par rapport à la pente, avec le numéro d'échantillon sur l'axe des abscisses

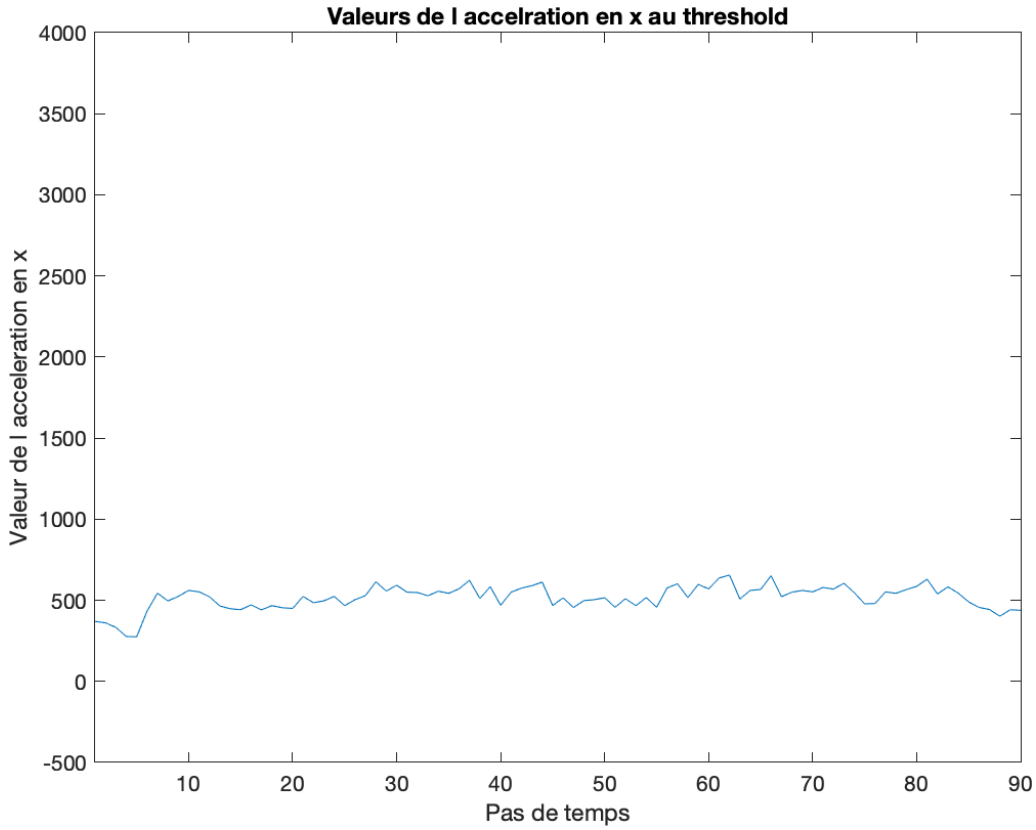


Figure 3 : échantillons de 90 valeurs de l'accélération en x (avec filter_size de 50) lorsque le robot est plus ou moins positionné dans la même direction que la pente, avec le numéro d'échantillon sur l'axe des abscisses

2.2 Contourner les obstacles

Afin de contourner un obstacle, le code traitant les données mesurées par les capteurs infrarouges compare les valeurs reçues à un seuil (déterminé expérimentalement). Cette étape se divise en deux phases dans notre code, la phase de détection d'obstacle, gérée par la fonction obstacle_check(), puis la phase d'évitement, gérée par la fonction dodge_obstacle().

La première phase consiste donc à utiliser les 4 capteurs situés à l'avant du robot, afin de déceler la présence d'un éventuel obstacle. La première analyse se fait sur les capteurs IR0 et IR7[1], ceux de devant. Si les valeurs mesurées par ces 2 capteurs dépassent un seuil, le robot tourne d'un quart de tour à droite (capteur IR0) ou à gauche (capteur IR7) et passe à la phase d'esquive. Dans le cas où ces deux capteurs ne détectent pas d'obstacle, les valeurs mesurées par les capteurs IR1 et IR6 sont à leur tour comparées à un seuil beaucoup plus élevé. Si un obstacle est détecté, le robot tourne de 45 degrés dans la direction opposée à l'obstacle et entre également dans la phase d'évitement. Le choix d'un seuil plus élevé pour ces deux capteurs est simplement que cette seconde analyse est une sécurité, qui permet d'éviter un choc imminent

[1]https://moodle.epfl.ch/pluginfile.php/2002535/mod_resource/content/9/e-puck2_F4-SCH-BOM-IMP.PDF (Dossier électronique du robot e-puck2, 2021)

avec un obstacle diagonal qui n'aurait pas été détecté par les deux capteurs situés à l'avant du robot.

Démarre ensuite la phase d'évitement, durant laquelle la fonction move_towards_up(), mentionnée au point 2.1 n'est plus appelée. Le robot avance tout droit, latéralement à l'obstacle détecté précédemment, jusqu'à le dépasser entièrement. Pour se faire, les valeurs mesurées par les deux capteurs de proximité latéraux (IR2 et IR5) sont comparées à un seuil plus faible, ce qui permet de s'assurer que l'obstacle est toujours sur le côté du robot. Lorsque l'obstacle a été dépassé, le robot fait un virage large de 90 degrés, afin d'éviter une collision avec le coin de cet obstacle et continuer sa route. Durant la phase d'esquive, le robot utilise la fonction front_obstacle_analysis() qui lui permet de détecter une paroi latérale qui pourrait l'empêcher de contourner l'obstacle. Le cas échéant, il comprend qu'il se trouve dans un coin du labyrinthe, fait demi-tour et entâme une esquive de l'obstacle par l'autre côté. Cette fois-ci, plusieurs tests physiques ont été effectués afin de trouver des valeurs de seuils qui faisaient sens pour les différents capteurs de proximité, de sorte que le robot ne le contourne ni trop tôt ni trop tard.

2.3 Autres fonctionnalités (moteurs pas à pas et LEDs)

Différentes fonctions régissant les moteurs sont appelées au fur et à mesure du programme, afin de déplacer le robot. Nous avons, pour éviter les magic numbers, créé des #define, spécifiques à notre utilisation et déterminés expérimentalement, qui répertorient le nombre de steps nécessaires, à une vitesse précise, pour que le robot fasse un quart de tour, un demi-tour ou un huitième de tour. Les fonctions spécifiques à l'utilisation des moteurs sont toutes définies dans le fichier deplacement.c.

Différentes LEDs ont été paramétrées de façon à permettre l'utilisateur de visualiser les actions entreprises par l'e-puck2. Typiquement, lors de l'initialisation, 2 LEDs rouges (LED3 et LED7) s'allument jusqu'à ce que les calibrations de l'IMU et des capteurs IR soient terminées et que les valeurs d'offset soient stables. Ensuite, lorsque l'e-puck se trouve sur un plat, c'est-à-dire qu'il a atteint le sommet de la pente, il s'immobilise et allume la Body LED. Lorsqu'il cherche la bonne direction à prendre, le robot allume la Front LED, et il l'éteint lorsqu'il l'a trouvée. Enfin, le robot allume de façon précise les LEDs 1, 3 et 7 lors de la phase d'esquive, décrite ua point 2.2. En effet, pour une esquive à droite, il enclenche la LED7, pour une esquive à gauche, il enclenche la LED3. La LED1 s'allume si l'esquive se fait à 90 degrés, mais reste éteinte pour une simple esquive à 45 degrés, déterminées en fonction des cas de figures décrits au point 2.2.

3. Organisation du code

Notre code est composé d'un thread principal traitant les données des deux capteurs (accéléromètre de l'IMU et capteurs IR de proximités) et de threads spécifiques à l'acquisition des données de chaque périphériques (IMU, moteurs, capteurs IR, …), initialisés dans le main grâce aux fonctions imu_start(), proximity_start(), motors_init(). Le programme fonctionne donc avec les threads suivants :

- Le thread qui met à jour les valeurs de l'IMU
- Le thread qui met à jour les valeurs du capteur de proximité
- robotControlThd, qui traitent les données de l'IMU et du capteur de proximité

Les deux threads de mise à jour des valeurs des capteurs ont la même priorité, c'est-à-dire NORMALPRIO, comme défini dans la librairie de l'epuck2_main-processor. Notre thread robotControlThd a cependant une priorité supérieure donc NORMALPRIO+1. Durant la phase d'esquive d'obstacle décrite au point 2.2, la priorité du thread principale est réduite à NORMALPRIO, grâce à la fonction chThdSetPriority(NORMALPRIO)[2], que nous avons découvert lors du TP3. Cela permet de focaliser le robot sur la phase d'esquive et ainsi de fluidifier l'acquisition des données des capteurs IR de proximité. La *figure 4* ci-dessous donne plus de précision sur les étapes du code dans le thread robotControlThd. Dans un premier temps, l'e-puck fait un traitement de données de l'IMU afin de déterminer s'il est dans la bonne direction et ensuite il traite les données des capteurs IR de proximité afin de vérifier qu'il n'y a pas d'obstacle.

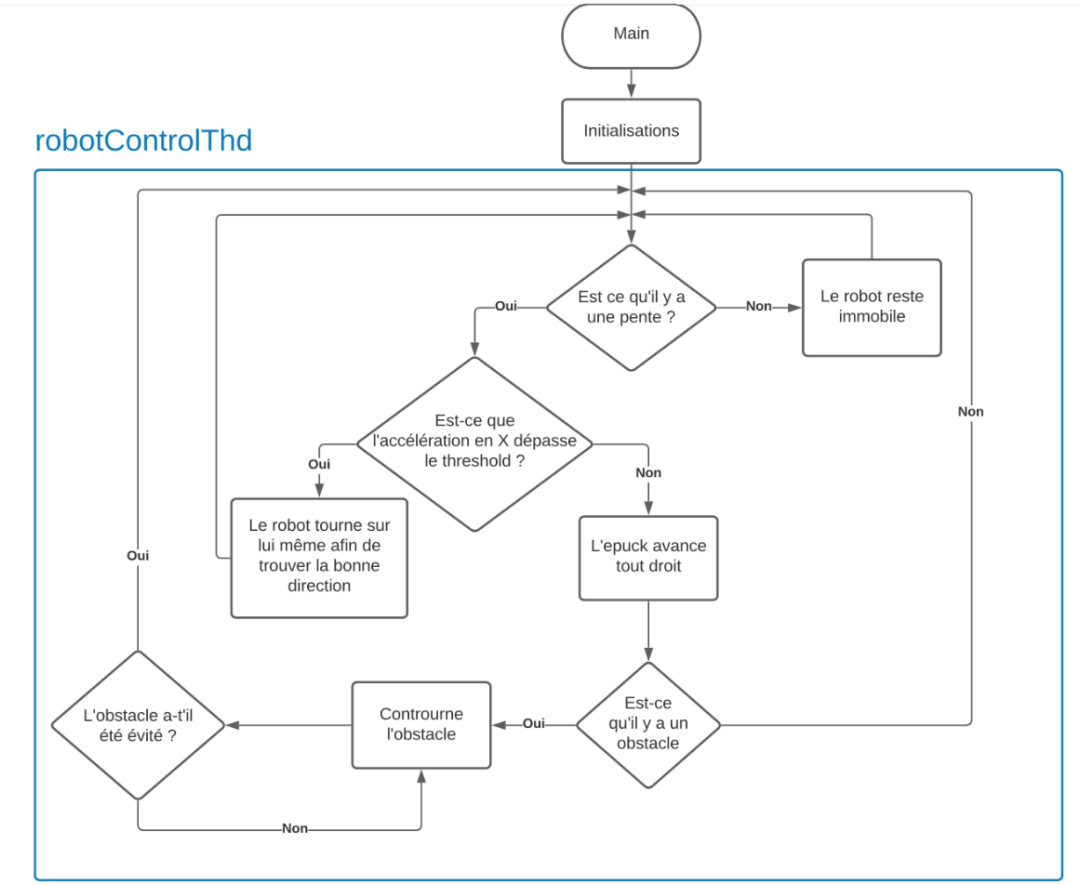


Figure 4 : voici le flow chart de notre code qui explique bien les étapes par lesquelles le code passe. Les étapes en losange représentent les questions, les étapes en rectangle sont les actions entreprises et les étapes ovales indiquent dans quelle partie du code nous nous trouvons

[2] (Prof. Mondada, TP3 corrections, 2021)

4. Résultats

Afin d'être le plus précis possible lorsque l'e-puck cherche la direction de la pente, nous avons réduit la vitesse de ses moteurs durant l'appel de la fonction move_towards_up(), décrite au point 2.1. Sans cette réduction de vitesse, nous prenions le risque que le robot « rate le coche » et tourne trop.

De plus, nous nous sommes rendu compte que la surface mise en pente sur laquelle le robot devait rouler avait une grande importance sur son déplacement. Après plusieurs tentatives, nous avons déterminé qu'une surface lisse non glissante était la plus appropriée aux roues de l'e-puck.

Lors de nos essais durant la conception projet, nous avons toujours travaillé avec une pente entre 18% et 45% nous ne garantissons donc pas que le robot soit aussi performant sous d'autres conditions.

Concernant l'optimisation de nos ressources mémoire, nous avons des valeurs de l'accélération initiée en int16_t, car plus de précision n'aurait pas apporté plus à notre projet au vu des valeurs communiquées par l'IMU.

5. Conclusion

Ce projet est le premier système physique sur lequel nous travaillons et codons. Afin de respecter le cahier des charges nous avons décidé d'utiliser les capteurs de proximités, l'IMU ainsi que les deux moteurs pas-à-pas. Ce mini-projet nous a permis de comprendre les interactions et les problèmes que l'on peut rencontrer avec un système physique ainsi que de mettre en pratique la théorie vue en cours (threads[3], ressources mémoires, …). De plus, le fait de devoir utiliser GitHub nous a pris beaucoup de temps au début du projet mais cela nous sera certainement bénéfique pour la suite de nos études.

Bibliographie

(2021, Mai). Récupéré sur Dossier électronique du robot e-puck2: https://moodle.epfl.ch/pluginfile.php/2002535/mod_resource/content/9/e-puck2_F4-SCH-BOM-IMP.PDF

Prof. Mondada, F. (2021, Mai). *Slide cours 4*. Récupéré sur https://moodle.epfl.ch/pluginfile.php/2026577/mod_resource/content/6/TP3Corrections.pdf

Prof. Mondada, F. (2021, Mai). *TP3 corrections*. Récupéré sur https://moodle.epfl.ch/pluginfile.php/2026577/mod_resource/content/6/TP3Corrections.pdf

[3] (Prof. Mondada, Slide cours 4, 2021)