PROJET DE MICROINFORMATIQUE

E-PUCK 2 DELIVERY

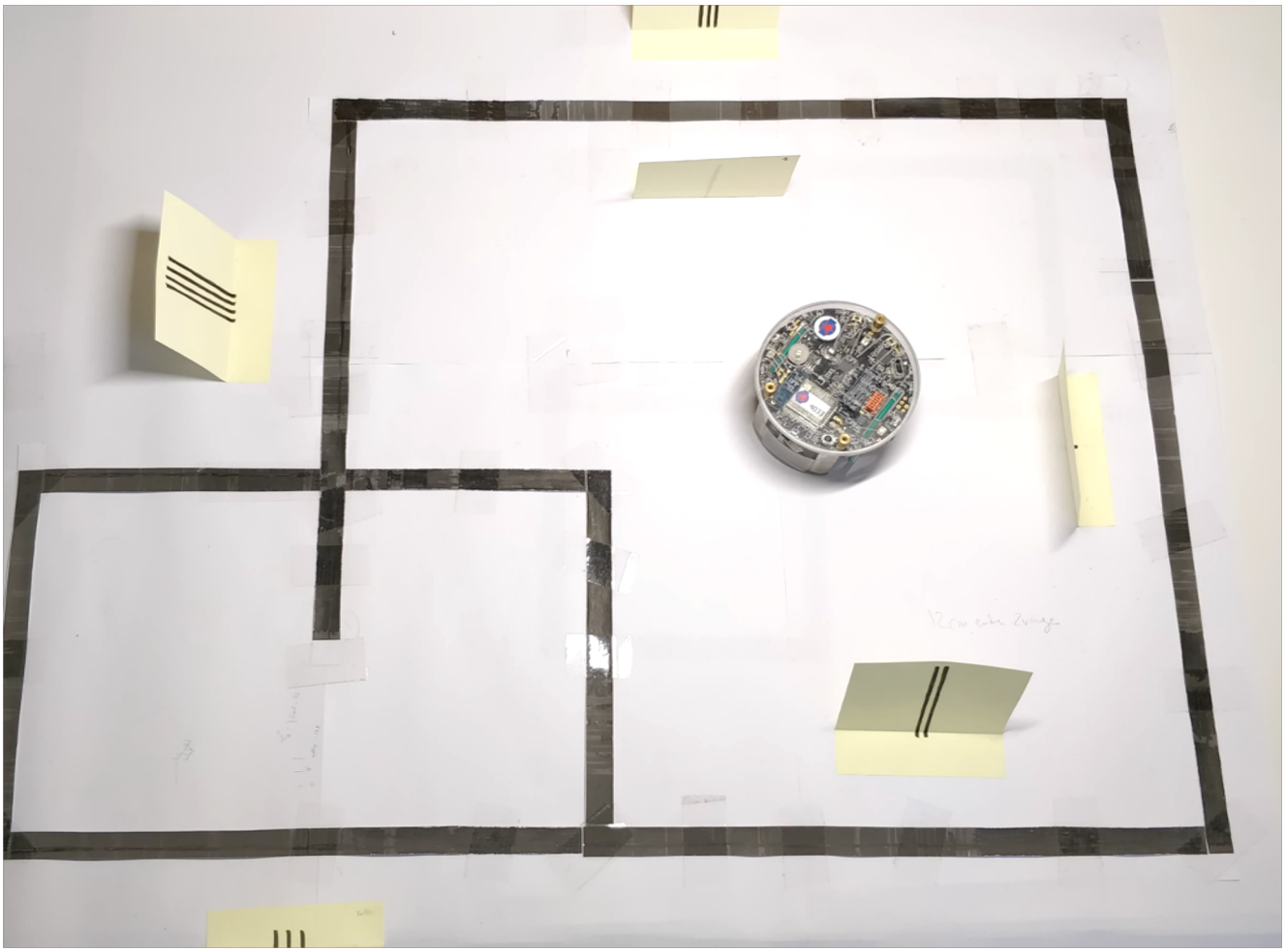

Figure 1 : Photographie de l'e-puck2 et du montage effectué dans le cadre du projet : la ligne correspond au chemin à suivre, les post-its représentent les différentes maisons

Table des matières

1 Introduction

Ce projet est réalisé dans le cadre du cours de microinformatique de 3ème année en microtechnique. Notre projet a pour but de simuler une livraison de colis complètement automatisée grâce au l'e-puck. Après avoir donné les ordres aux robots dans un hangar, le robot quitte le hangar et effectue entièrement sa mission. Cette simulation simplifiée, implémentée dans la vie réelle, permettrait d'avoir un mode de livraison adapté à la situation de crise actuelle en évitant ainsi les contacts humains.

2 Principes de fonctionnement

2.1 Fonctionnement général

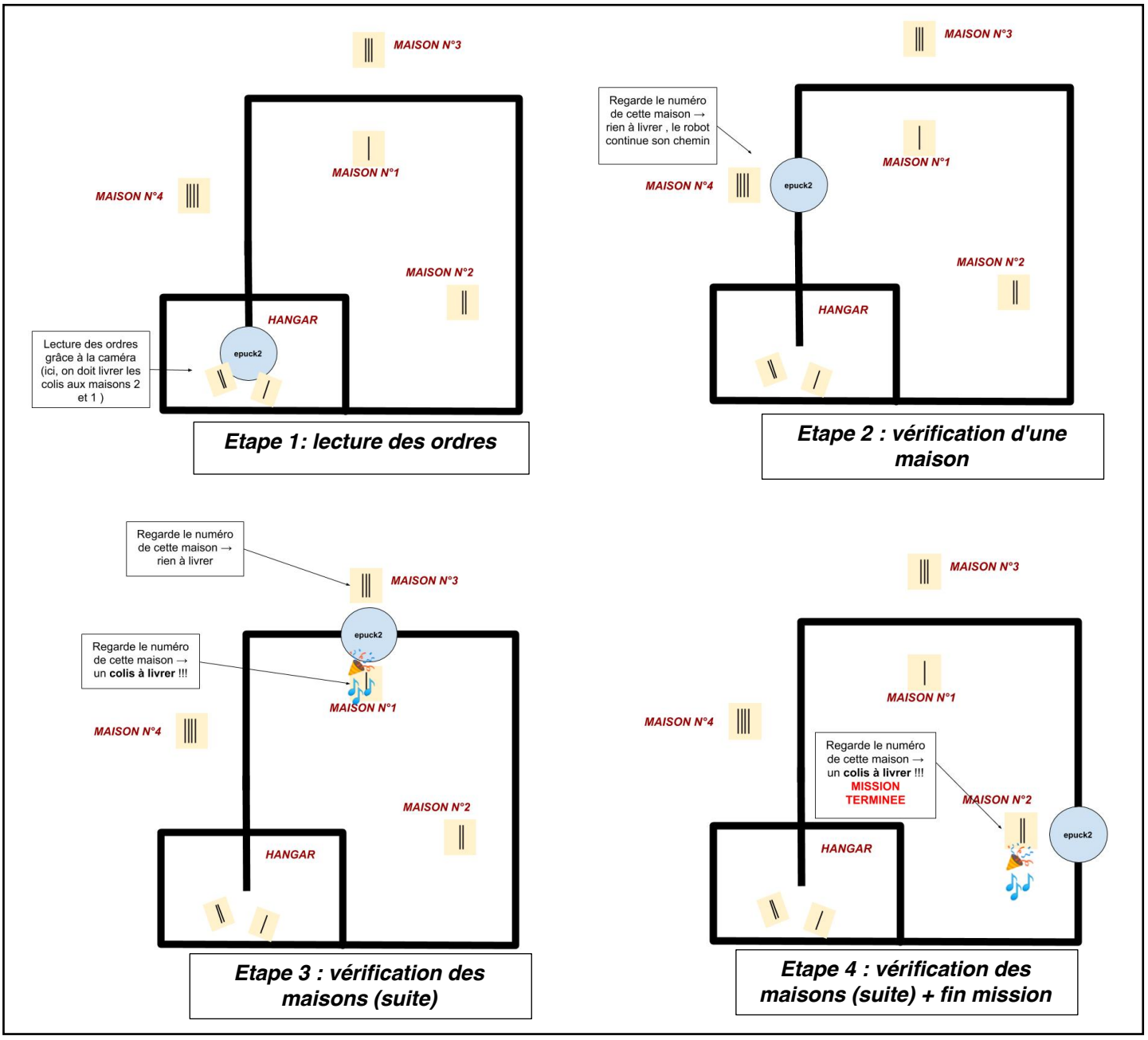

Figure 2 : Schéma expliquant brièvement le fonctionnement de notre projet en plusieurs étapes (les lignes noires en gras représentent le chemin que le robot doit suivre)

Tout d'abord, nous plaçons le robot à son point de départ, le hangar (cf . *Figure 1*). Nous lui donnons ensuite une série d'ordre, en lui montrant différents codes-barres écrits sur des post-it. Pour chaque colis devant être livré, une led bleue va s'allumer. Le robot peut recevoir au maximum 4 colis. Si on veut lui donner moins de 4 colis à livrer, il faut lui montrer une feuille blanche pour qu'il quitte le hangar et qu'il commence la navigation.

Ensuite, l'e-puck va alors suivre le chemin en noir jusqu'à ce qu'il détecte une maison dans ses alentours. Une fois détectée, il va lire le code barre présent sur la maison (aussi un post-it) et vérifier s'il doit livrer un colis à cette maison. Si oui, il va alors livrer le colis (leds clignotent + musique), sinon il continue son chemin jusqu'à ce qu'il ait livré tous les colis initiaux.

2.2 Utilisation des capteurs

2.2.1 Caméra

Dans notre projet, la caméra a deux utilités différentes. Dans les deux cas, nous cherchons à différencier le noir du blanc. Nous avons choisi d'extraire seulement la couleur verte (qui contient plus d'information que le bleue et le rouge). Après quelques tests, nous avons établi un seuil pour différencier ces deux couleurs.

2.2.1.1 Navigation

D'une part, nous utilisons la caméra pour permettre au robot de suivre une ligne noire tracée au sol, et de gérer les virages. Nous utilisons une ligne en bas de la caméra pour analyser la route. Nous calculons ensuite l'erreur d'alignement par rapport au centre de la route, et nous utilisons un régulateur proportionnel avec un Kp=2, trouvé empiriquement (le régulateur PI ne nous apportait pas une grande amélioration et causait trop d'instabilités lors des virages). Pour gérer les virages à 90°, la caméra détecte du noir soit à droite, soit à gauche de la ligne de pixel. Nous faisons donc en sorte d'avancer le robot de la bonne distance, et de le faire tourner pour prendre correctement le virage. Lors d'un croisement, le robot prend toujours à gauche.

2.2.1.2 Lecture de code-barres

D'autre part, elle nous permet de lire les codes-barres (c'est-à-dire compter le nombre de barres sur chaque post-it). Si la bande noire sur le post-it est assez large et que la couleur claire à sa droite est considérée comme un espace, alors l'e-puck détecte 1 bande. Sinon, cela est considéré comme du bruit (cf. *Figure 3*). Le nombre de bande sur le post-it détermine le code-barres. Nous utilisons la même ligne de la caméra que pour la navigation (le code barre est visible par cette ligne de la caméra). Ces valeurs de codes sont soit mémorisées pour la livraison des colis, soit comparées avec les valeurs mémorisées lors de la vérification des maisons.

Figure 3 : Affichage d'une ligne de caméra dans le terminal après tous les filtrages, lors de la lecture d'un code 4 (post-it avec 4 bandes). En rouge, nous distinguons un trait noir et en orange, un trait noir, mais qui déborde sur 2 lignes dans cet affichage. En vert, nous voyons du bruit qui est négligé par notre algorithme par la suite.

2.2.2 Capteurs de proximité IR pour la détection des maisons

Les capteurs de proximité nous permettent de détecter lors de la navigation, s'il y a une maison (un post-it) autour du robot. Nous avons utilisé les capteurs de proximité droit et gauche. Le capteur de proximité IR, nous permet de distinguer de très faibles distances. En effectuant quelques tests avec le robot, nous avons établi une valeur de threshold pour le capteur qui nous permettait de bien distinguer la présence des post-its sur le bord du chemin parcouru.

Pour éviter les erreurs d'alignement avec le post-it (on veut que la caméra se retrouve bien au centre du post-it), dès que l'on détecte un post-it à proximité (la détection peut être un peu lente), nous avançons millimètre par millimètre jusqu'à ce le capteur ne détecte plus rien, le robot recule alors d'une distance connue pour que la caméra soit centrée lorsque le robot se tournera.

3 Organisation du code

3.1 Relation entre les modules et threads

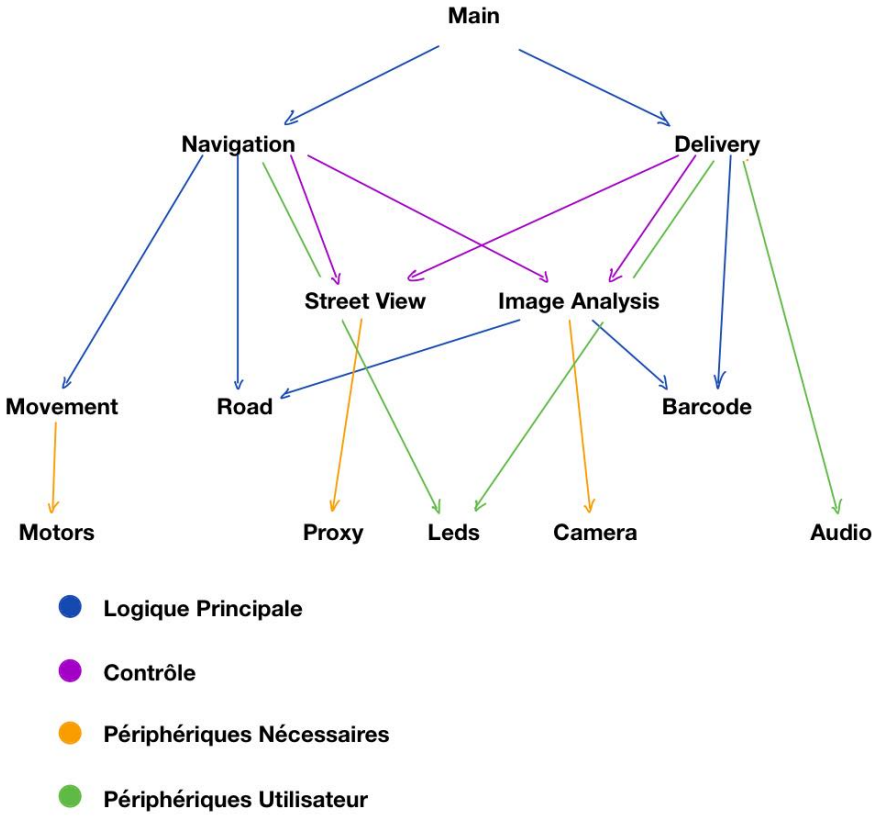

Figure 4 : Arbre des dépendances simplifié entre les modules de notre projet e-puck2_delivery, seulement les modules important pour la simulation (pas utilitaires, defines, chprintf etc.)

Le thread principal main utilise les fonctions des deux modules : navigation et delivery. Navigation s'occupe de la gestion du mouvement pour suivre la route et bien se positionner par rapport aux maisons, et delivery de la gestion de la livraison. Navigation contient un thread pour suivre constamment la route (si le thread est activé).

Plusieurs modules utilisent les mêmes périphériques, des modules de contrôle de ces ressources ont alors été créés dans l'esprit de "separation of concerns" et modularité (cf. *Figure 4*).

Les threads CaptureImage et ProcessImage permettent également de gérer respectivement l'acquisition des images et le traitement de l'image (diffère si analyse la route ou la maison).

3.2 Contrôle de versions – git

Accès public: https://github.com/W4li8/e-puck2-delivery, le code est disponible sous le répertoire code/src/

Le travail, au départ, a été divisé en deux branches axées sur les branches principales du projet. Une approche top-down a été utilisée pour la décomposition et puis bottom-up pour la construction des éléments individuels un à un. Par la suite, l'organisation générale du code a changé à plusieurs reprises (organisation des modules, threads ou pas, scope des variables et autres). A chaque nouvelle étape qui paraissait dévier de la ligne de développement ou lors de tests particuliers, une nouvelle branche a été créée. Quelques soucis de mise en route sont survenus par moment, d'où des commit un peu répétitifs ou inutiles par moments.

4 Tests et résultats

4.1 Problèmes rencontrés et tests

Nous avons effectué plusieurs tests pour la calibration des capteurs et pour le fonctionnement de notre algorithme :

Tout d'abord, les variations de lumières venant de l'extérieur par la fenêtre pouvaient poser problème pour la détection de la route et des code-barres par la caméra. Il a donc fallu tout renfermer dans un environnement artificiel (stores fermés, et lumière par une lampe) pour finalement calibrer le seuil d'intensité de la ligne de caméra pour la couleur verte (cf. *Figure 5*). Nous avons également fixé le temps d'exposition pour être plus précis.

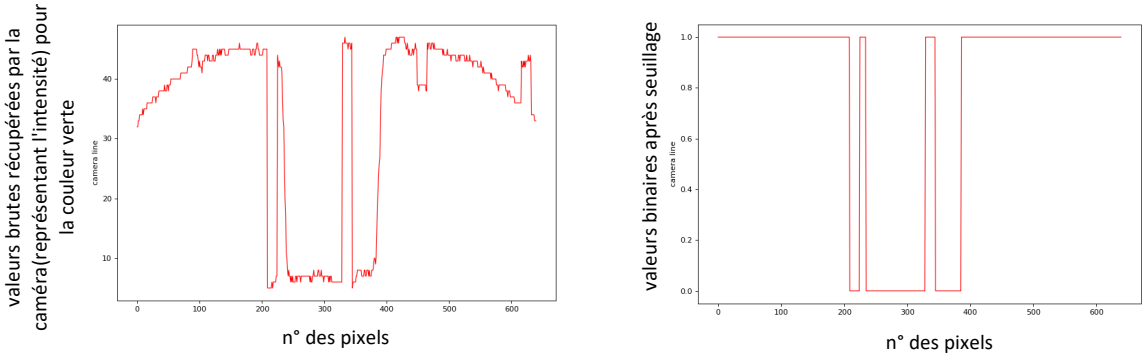

Figure 5 : Ces deux courbes représentent la récupération (a) et le traitement (b) des données d'une ligne de caméra lorsque l'e-puck est face à la route (une ligne noire). Le graphe (a) nous a permis de choisir un seuil de détection pour distinguer le blanc du noir (un seuil de 15 nous a paru approprié). Dans le graphe (b), nous pouvons voir que les valeurs au-dessus de ce threshold prennent la valeur 1 (blanc) et les valeurs au-dessous prennent la valeur 0 (noir). Ce résultat est ensuite utilisé par les algorithmes de détection de route et lecture de code barre, comme le montre la Figure 3 qui sont, entre autres, chargés d'éliminer les pics parasites.

Ensuite, la calibration des capteurs de proximité a été faite par des tests via le terminal (CoolTerm). Le robot a été placé à la distance de fonctionnement désirée et un seuil a été choisi pour déterminer la présence/absence d'une structure.

Le bruit de ces deux capteurs a été un problème et des seuils/erreurs maximales ont été définies afin de remédier à cela.

Nous avons également fait face à des problèmes pour l'alignement de la caméra au centre des post-it, le code-barres détecté était parfois faux à cause du mauvais alignement. La solution expliquée en section 2.2.2 a été adoptée.

4.2 Simplifications apportées

Après une série de tests, plusieurs simplifications ont été apportées au projet.

Au niveau de la route, nous avons choisi d'avoir une largeur de route uniforme (1cm). Les virages sont également tous à 90 degrés. Il n'y a pas de route courbée (cela aurait pris trop de place pour le montage). Cependant, plusieurs cartes ont été développées et le robot et également capable de suivre des routes courbées si le rayon de courbure est assez grand (pas quantifié).

Pour la détection des maisons, nous avons également fait en sorte que les maisons soient à une distance précise (4,5cm) de la route et aient des tailles identiques. On ne détecte pas de maisons dans les virages, et il n'y a pas d'obstacle sur la route.

Différents types de post-its ont été testés. A cause du champ de vue réduit de la camera, la version centrée et homogène (à peu près la même distance entre chaque bande noire et les bandes noires ont toujours la même épaisseur) du code-barres a été retenue. C'est aussi la plus facile à décoder et la moins susceptible aux erreurs.

Au niveau logique, nous avons décidé que le robot aurait aucune notion du chemin, le robot est constamment "perdu", il n'a aucune idée de la carte. Il suit juste la route. Il prend toujours à gauche lors des croisements. A la fin de la mission, il ne retourne pas au point de départ, une mission peut recommencer tout de suite.

4.3 Extensions possibles et limitations

Le robot serait peut-être plus adapté pour des espaces clos, comme des centres de stockage, où les conditions sur le fonctionnement de notre prototype et l'environnement sont plus facilement mises en place.

Il est facile de recalibrer le robot pour un autre environnement ou modifier complètement le fonctionnement en changeant simplement quelques macros et appels de fonction.

Les limitations principales sont liées aux simplifications faites qui rendent le robot inefficace parfois. Par l'algorithme "tourner à gauche aux croisements" il peut se retrouver coincé sur une boucle de la carte et ne jamais trouver une maison (cela ne pose pas problème avec notre montage actuel si on fait toujours partir le robot du hangar). Il ne peut pas continuer tout droit à un croisement : dans l'état actuel des choses, il ne le saurait même pas s'il y a une route ou pas (lecture de ligne horizontale). Il perd également la route de vue dans les virages de par l'orientation de la caméra, il en faudrait une orientée vers le bas. Et enfin, le robot peut seulement détecter un virage ou un croisement s'il arrive de loin; s'il est trop proche, il ne verra pas le croisement (mais cela nous permet également de sortir du hangar en allant tout droit et pas à gauche).

Le robot peut être gêné à l'interface entre deux feuilles de papier. Si nous installons correctement le montage et que la route est bien plate, il n'y aura pas de souci.

Les maisons/post-its sont très standardisés et il n'y a aucune adaptation qui est faite quant au positionnement du robot vis-à-vis de ceux-ci, c'est de la calibration de mesures pure.

5 Conclusion

Nous avons à travers ce mini projet pu recréer une version simplifiée de robots de livraison, chose que beaucoup de compagnies essaient actuellement de mettre en place. Certes notre livraison reste symbolique de par la construction de l'e-puck qui ne permet pas de faire de vraies livraisons mais cela nous a permis de toucher à certaines problématiques auxquelles ces compagnies font face tous les jours.

Ce projet nous a permis d'utiliser pour la première fois nos connaissances de programmation pour le contrôle d'un robot. Nous avons pris conscience de la difficulté à implémenter une logique sur un système physique. Les limitations des différents capteurs réduisent les possibilités et nous ont poussé à faire plusieurs simplifications et beaucoup de tests pour arriver au résultat souhaité. Ce projet nous a également permis d'optimiser nos méthodes de travail grâce à l'utilisation de GIT et GITHUB qui nous ont permis d'être plus efficace dans la gestion du code.

6 Références

GCtronic, e-puck2, https://www.gctronic.com/doc/index.php/e-puck2

Cours et TPs de Microinformatique du Professeur Mondada en 3ème année de microtechnique

Documentation sur les threads de ChibiOS sur chibios.org (site actuellement en maintenance)