



L'e-puck explorateur

Maxime Marchionno & Nicolas Peslerbe
supervisé par Prof. Francesco Mondada & Eliot Ferragni

Avril 2018

1	Introduction	2
1.1	Donnée	2
1.2	Notre projet	2
2	Fonctionnement général	2
2.1	Télécommande audio	2
2.1.1	Fonctionnement	2
2.1.2	Discussion	2
2.2	Découverte	2
2.2.1	Fonctionnement	2
2.2.2	Discussion	3
2.3	Exploration	3
2.3.1	Fonctionnement	3
2.3.2	Discussion	3
3	Les modules	3
3.1	Module exploration	4
3.2	Module cartographie	4
3.3	Module communication	4
3.4	Module audio	4
4	Les compléments nécessaires à la démonstration	5
4.1	L'application mobile	5
4.2	Le programme python	5
4.3	Le décor	5
5	Conclusion & Discussion	6
5.1	Choix modelant notre projet	6
5.2	Conclusion Personnelle	6
6	Annexes	7

1 Introduction

1.1 Donnée

Nous réalisons ce projet dans le cadre du cours de microinformatique de 3^{ème} année. Ce dernier se base quasiment exclusivement sur le robot pédagogique e-puck qui est passé, dernièrement, en version 2. Nous avons l'honneur d'être la première volée à l'utiliser.

La donnée du projet fut ouverte et les contraintes réduites. Nous devions principalement utiliser la librairie de base, les moteurs, un capteur de distance et un autre capteur vu durant les travaux pratiques. Le tout devait respecter les conventions de programmation et s'intégrer à la librairie ChibiOS.

1.2 Notre projet

Profitant de la liberté donnée, nous avons souhaité exploiter un maximum de fonctionnalités de l'e-puck2. Dans une approche robotique, nous souhaitions également travailler sur la précision de ses capteurs et de ses moteurs. L'objectif était tout donné, un robot explorateur.

Commandé par une télécommande audio (sous forme d'application mobile), notre e-puck est capable de repérer un terrain de dimension inconnue et d'y cartographier des objets. Lors de la détection d'un objet, une photo est prise de celui-ci et est enregistrée sur un ordinateur via un programme en Python.

Notre objectif était de créer un projet qui ne se contentait pas uniquement d'exploiter les notions abordées durant le cours mais qui allait plus loin. Nous désirions mettre nos connaissances générales au profit de ce projet. Une grande partie du projet a été la résolution de problèmes complexes de géométrie. Nous avons également traité des signaux audio et exploité la caméra.

2 Fonctionnement général

2.1 Télécommande audio

2.1.1 Fonctionnement

La télécommande audio est le seul moyen de donner des ordres au robot. Lorsque l'e-puck est de couleur verte, cela signifie qu'il est prêt à répondre à un ordre. Dans cette situation, le robot écoute avec le micro frontal et retourne l'information qu'un ordre a été donné au thread principal, qui lance l'action si celle-ci est possible.

Dans le cadre de notre démonstration, nous utilisons une application mobile qui émet les fréquences désirées (code swift inclus avec le projet).

2.1.2 Discussion

Cette télécommande remplace le sélecteur de mode. Ce qui nous permet de ne pas toucher le robot lors des changements de "mode" et donc de ne pas perdre en précision.

2.2 Découverte

2.2.1 Fonctionnement

L'étape de découverte, bien que peu impressionnante en apparence, est une étape qui fait appel à beaucoup de calculs et nécessite une haute précision des mesures.

Dans un premier temps, le robot effectue une rotation complète dans le sens horaire et capte la distance qui le sépare des murs avec le TOF. Ensuite, ces mesures sont transformées en points avec comme origine la position du robot au début de la découverte. Se référer à la De ces points sont calculés les équations des droites de chacun des murs, puis les intersections de ces droites. Une fois la position des murs déterminées, l'origine est déplacée à l'intersection des deux murs que le robot a mesuré en premier. À partir de ce moment, toutes les mesures du robot sont référencées dans ce nouveau repère.

Vous trouverez plus de détail dans les annexes du rapport.

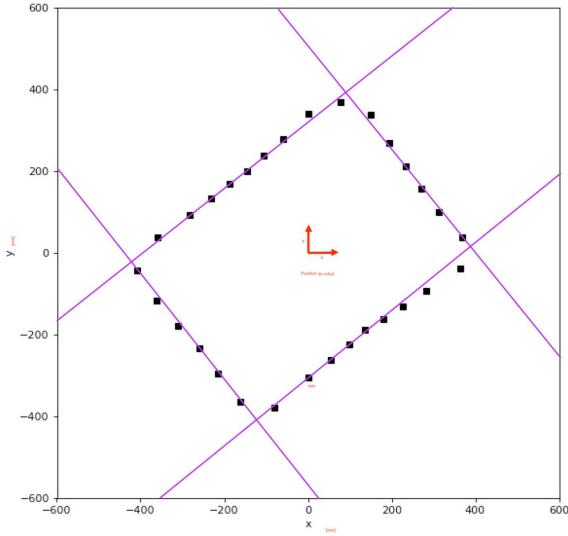


FIGURE 1 – Ce que reçoit et affiche le programme python



FIGURE 2 – Notre terrain modulable

2.2.2 Discussion

Nous avons remarqué que la précision des mouvements des moteurs avec une batterie faible est moindre. Il nous a fallu faire une longue étape de calibration pour arriver à une précision de ± 2 [cm]. De plus les valeurs du TOF variaient également.

2.3 Exploration

2.3.1 Fonctionnement

Lors de l'exploration, le robot effectue une rotation générale et lorsqu'il trouve un objet, il scanne la zone d'intérêt pour identifier la position exacte de l'objet. Une fois la position trouvée, le robot se place à une distance donnée et lance la capture de l'image. Cette image est envoyé, via bluetooth, à un programme python sur notre ordinateur. Ce dernier l'enregistre et la place sur la carte, définie lors de la découverte. Une fois la rotation complète effectuée, l'exploration prend fin.

2.3.2 Discussion

Notre idée de base était de parcourir tout le terrain, y répertorier les objets et les éviter. Mais nous n'avons pas pu pour des raisons d'imprécision des moteurs. Cette imprécision est particulièrement présente lors de (nombreuses) rotations. Le robot se met à confondre les murs et les objets. Le principe d'exploration est resté, cependant, présent.

3 Les modules

Pour une facilité de lecture des différentes librairies, celles-ci ont été séparées en modules, chaque module gérant un lot de tâches en rapport avec le nom du module. Les modules les plus intéressants par leur construction sont ceux de l'exploration, la cartographie, la gestion audio et la communication.

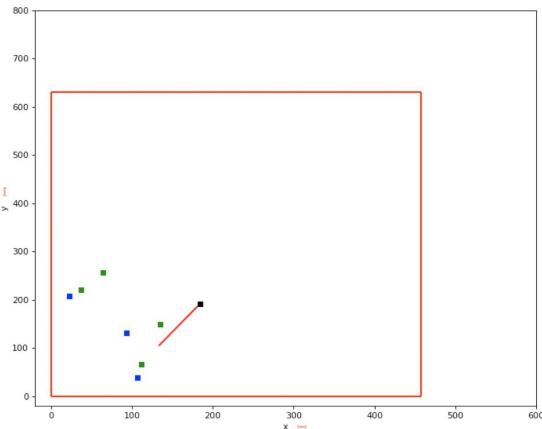


FIGURE 3 – Ce que reçoit et affiche le programme python



FIGURE 4 – Notre terrain modulable avec les PLAYMOBILs

3.1 Module exploration

L'exploration, s'occupe de lier les informations provenant des capteurs, de donner des ordres aux moteurs, tout ça en consultant / actualisant les données de cartographie. Lorsqu'une action est lancée depuis le main, celle-ci est allouée à un thread, cependant, elle bloque le programme principal qui ne reprend qu'à la fin de la tâche. Les ordres de modifications des moteurs sont gérés sur un thread et toute la cartographie se fait à partir des informations données aux moteurs et non celle souhaitées par l'utilisateur. Ce qui donne plus de précision. Enfin des changement de positions peuvent être donnés en absolu (par rapport au terrain) ou en relatif (par rapport au robot).

3.2 Module cartographie

Ce module s'occupe de tous les calculs de coordonnées et de points. Par exemple, lorsque le robot souhaite aller à un point précis, la position de ce dernier est récupérer par ce module et celui-ci retourne le chemin à parcourir. Aussi la position des murs, des différents objets et du robot sont enregistrés dans ce module. Toutes les vérifications de l'environnement (présence d'un mur, d'un objet etc.) y sont effectuées.

3.3 Module communication

La communication se fait selon un modèle qui a été établi pour ce projet. Ce qui nous permet d'envoyer tout type d'information et de les identifier avec le programme python. La structure est la suivante : taille du body || type de l'information || body. Si c'est un mur, le programme python trace un mur. Si il s'agit d'une image, il l'enregistre. Le module de communication est uniquement sortant.

3.4 Module audio

Pour les communications entrantes, ce module est utilisé. Lorsque cela est nécessaire, le détection audio est activée. Si un message cohérent est reçu, un sémaphore prévient le programme principal. Par exemple, une fréquence de 400Hz est nécessaire pour la découverte et une de 500Hz pour l'exploration.

4 Les compléments nécessaires à la démonstration

4.1 L'application mobile

Pour la télécommande, nous avons écrit une application qui émet les fréquences nécessaires aux différents modes. Il s'agit d'une application iOS simple codée en swift. Le code se trouve en annexe.

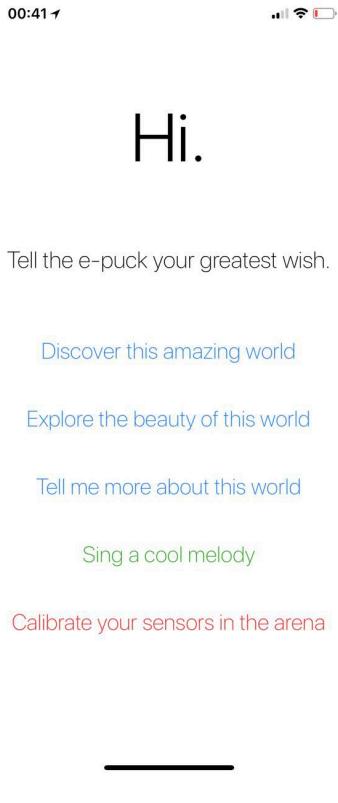


FIGURE 5 – Application iOS

4.2 Le programme python

Basé sur le code trouvé sur moodle, le programme ouvre un graphique qui cartographie points et murs. Aussi il affiche dans le terminal les messages envoyés par le robot, affiche en temps réel la position, l'angle du robot et enregistre les images.

4.3 Le décor

Nous souhaitions avoir une arène modulable, nous avons donc fait une arène en bois qui peut changer de taille afin de prouver l'adaptabilité de notre programme. Le système a été pensé sans vis et pour se transporter facilement.

Des PLAYMOBIL® sont utilisés comme objets à détecter.

Un tapis antidérapant, normalement fait pour des tiroirs de cuisine fait office de sol, afin de maximiser la précision du robot.

5 Conclusion & Discussion

5.1 Choix modelant notre projet

Malheureusement, en raison des limites de précision que nous avons obtenu avec le robot nous avons dû revoir nos espérances à la baisse : Une grande précision angulaire après plusieurs rotations n'ayant pu être obtenue, nous avons limité au maximum les déplacement du robot pour la cartographie. Nous avions créé un évitement d'obstacle se basant sur les véhicules de Braitenberg. Pour les même raison, nous n'avons pas pu exploiter cette fonctionnalité.

5.2 Conclusion Personnelle

Si nous devions retenir une seule chose de ce projet, c'est qu'en robotique il est très difficile de faire confiance au matériel. Nous avons malheureusement été beaucoup trop ambitieux. Malgré le fait que la théorie fonctionne et que nous soyons capable de faire ce que nous souhaitions. Nous avons vite été freiné par la calibration puis par les erreurs dépendantes de l'environnement.

Nous avons donc appris à travailler plus simplement.

Annexe - Trigonométrie robot e-puck

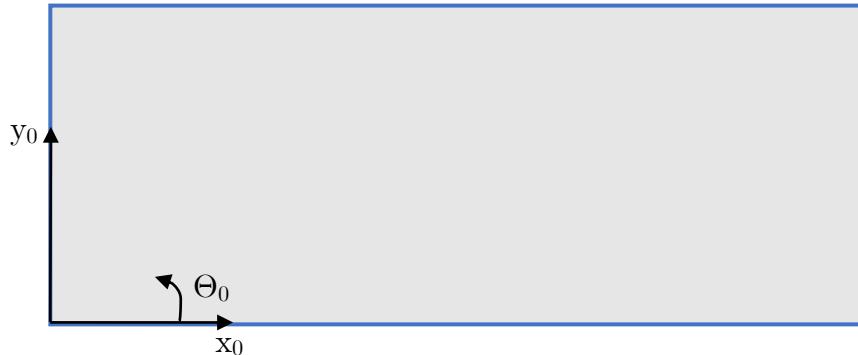
Type d'élément	Variables	Schéma
Référenciels ou lignes infines robotPosition_t	<pre>typedef struct { int x; // in mm int y; // in mm float theta; // in rad } robotPosition_t;</pre>	
Point point_t	<pre>typedef struct { int x; // in mm int y; // in mm } point_t;</pre>	
Vitesse des roues du robot wheelSpeed_t	<pre>typedef struct { int left; // in mm/s int right; // in mm/s } wheelSpeed_t;</pre>	
Vitesse du robot robotSpeed_t	<pre>typedef struct { int mainSpeed; // in mm/s float angle; // in rad/s } robotSpeed_t;</pre>	

Calcul des vitesses des roues (x_1, y_1, θ_1)

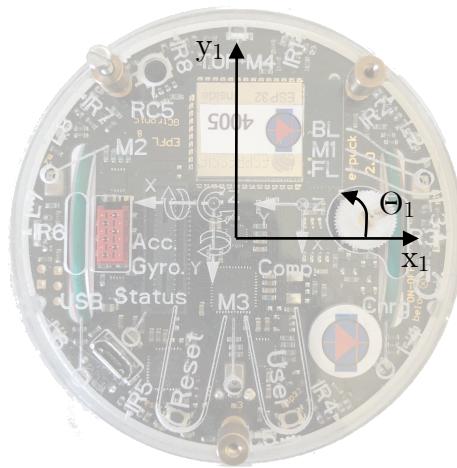
robotSpeed_t -> wheelSpeed_t
$left = mainSpeed + \frac{angle * Rayon du robot}{2}$
$right = mainSpeed - \frac{angle * Rayon du robot}{2}$
wheelSpeed_t -> robotSpeed_t
$mainSpeed = \frac{left + right}{2}$
$angle = \frac{right - left}{Rayon du robot}$
wheelSpeed_t -> motor_set_speed
$left_{motor} = left * Constante de calibration$
$right_{motor} = right * Constante de calibration$
avec : $Constante de calibration = \frac{motorSpeed}{realSpeed (in mm/s)}$

Référenciels

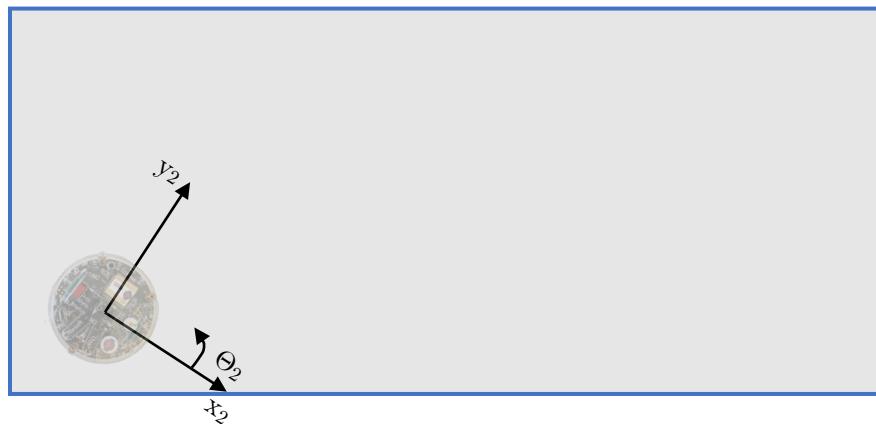
Terrain (x_0, y_0, θ_0) - `robotPosition_t`



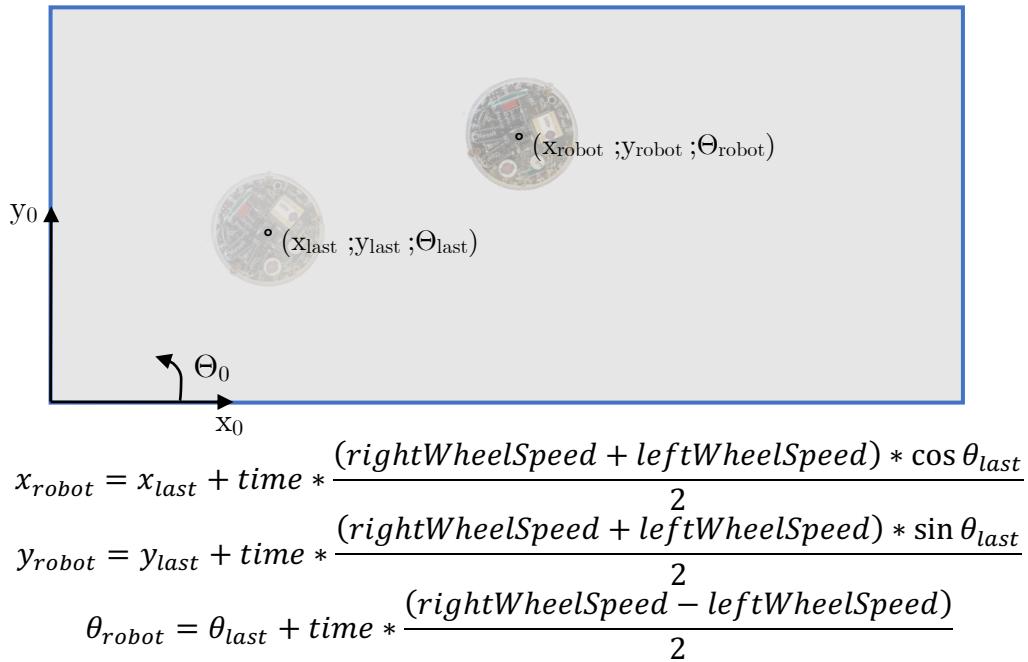
Robot (x_1, y_1, θ_1) - `robotPosition_t`



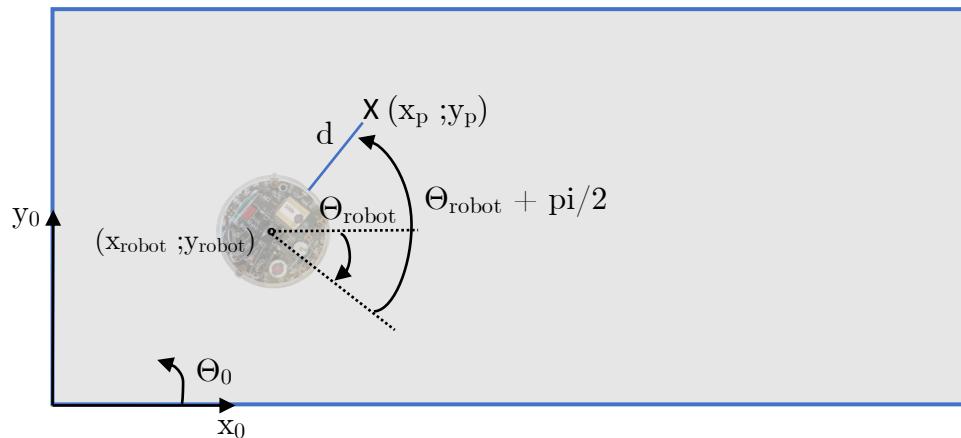
Initial (x_2, y_2, θ_2) (avant découverte des murs) - `robotPosition_t`



Nouvelle position du robot



Position sur le terrain d'un point mesuré avec le TOF



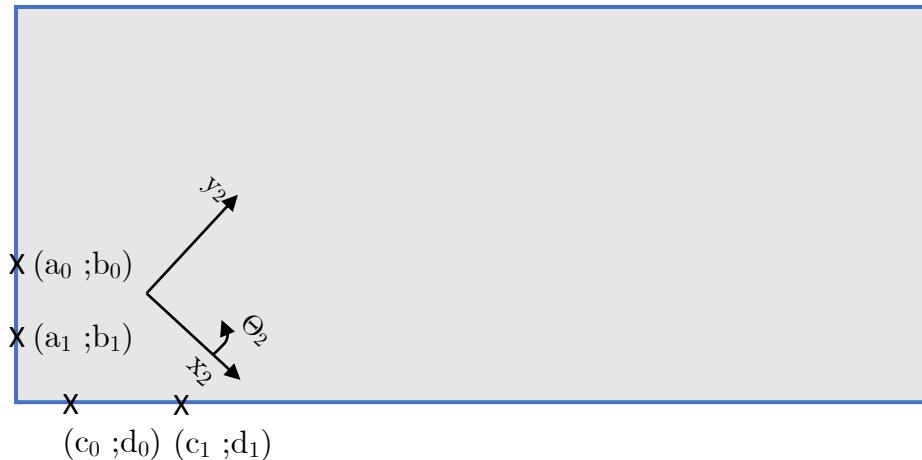
$$x_p = (d + \text{Rayon du robot}) * \cos\left(\theta_{robot} + \frac{\pi}{2}\right) + x_{robot}$$

$$x_p = -(d + \text{Rayon du robot}) * \sin\theta_{robot} + x_{robot}$$

$$y_p = (d + \text{Rayon du robot}) * \sin\left(\theta_{robot} + \frac{\pi}{2}\right) + y_{robot}$$

$$y_p = (d + \text{Rayon du robot}) * \cos\theta_{robot} + y_{robot}$$

Nouvelle position de l'origine



Équation de la droite du mur dans le repère initial (2)

$$m = \frac{y_0 - y_1}{x_0 - x_1}$$

$$p = y_1 - m * x_1$$

Point d'intersection des deux murs

$$x_{intersection} = \frac{p_2 - p_1}{m_1 - m_2}$$

$$y_{intersection} = m_1 * x_{intersection} + p_1$$

Orientation du mur dans le repère initial (2)

$$\theta_{mur} = \text{atan} \left(\frac{y_0 - y_1}{x_0 - x_1} \right)$$

Changement position robot dans le repère terrain

$$(x; y; \theta)_{x_0, y_0, \theta_0} = (x_{robot} - x_{intersection}; y_{robot} - y_{intersection}; \theta_{robot} - \theta_{mur})_{x_2, y_2, \theta_2}$$