Robotics Project I

Vincent Carrel, Group 9

IN.2022 Robotics 2019, BSc Course, 2nd Sem.
University of Fribourg
vincent.carrel@unifr.ch

Résumé

Brief description of the content (5-10 lines). Helps people decide whether the report is relevant for them or not. Usually written at the end.

Keywords: add, keywords, for, indexing

The use of LaTeX is mandatory for the Project I report. Apart from the examples in the appendix below, this template may not be modified. A good introduction to scientific writing is given by [1]

Table des matières

Chapitre 1

Introduction

Objectives of this project, and brief description of the structure of the report.

Chapitre 2

Sensors

Les robots utilisées durant les cours de Robotique sont des e-puck2, développés par l'EPFL et GCtronic. Ils sont équipés de nombreux capteurs, dont voici les plus communs.

2.1 Proximity infra-red sensors

L'e-puck2 possède 8 capteurs infrarouge de distance, nommés [Prox0] à [Prox7]. Un capteur infra-rouge fonctionne grâce à deux diodes : un émetteur infra-rouge et un capteur (phototransistor). Le premier émet une lumière infra-rouge, et le deuxième, équipé d'un filtre coupant la lumière ambiante, va capter la réflexion du rayon lumineux. En fonction de l'intensité du rayon capté, le capteur peut ensuite évaluer la distance de l'objet éclairé. La puissance lumineuse sortante étant faible, cela fonctionne uniquement sur de très petites distance, les capteurs utilisés ici ne captent rien au delà d'environ 6-8 cm.

Voici le graphique représentant les valeurs utilisées pour les capteurs infra-rouge lors des simulations Webots :

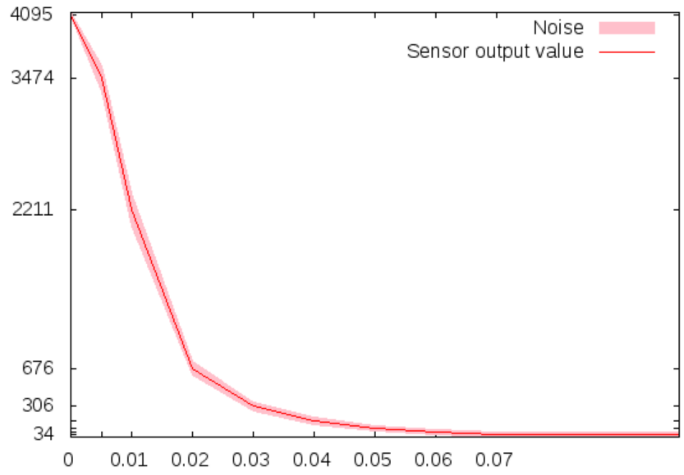

FIGURE 2.1 – Réponse des capteurs infra-rouges utilisée dans Webots.

Afin de mieux comprendre les possibles différences de comportement entre une simulation Webots et l'application réelle sur le robot e-puck2, nous avons reproduit et analysé ce graphique

Le robot (numéro 204) fut placé sur une table vide, à l'écart de tout objet durant la phase de calibration. Ensuite, un objet est placé devant le robot, et celui-ci va s'en éloigner progressivement en mesurant régulièrement la distance.

La phase de calibration est le moment où le robot va, à chaque initialisation, calibrer ses capteurs, pour réduire au maximum le bruit, les interférences inhérentes à chaque capteurs, qui pourrait venir tromper les mesures. Le robot prend plusieurs mesures, en fait une moyenne, puis enregistre cette valeur comme "valeur par défaut", la valeur du point neutre, qu'il soustraira à chaque nouvelle valeur mesurée ensuite. Cette phase est nécessaire dans la réalité, car la plupart des capteurs ont des défauts de fabrication, et envoie des valeurs différentes pour une situation identique. En particulier, on peut étudier les capteurs [ps2] à [ps5] dans la figure suivante. Les 4 capteurs sont éloignés de tout obstacles durant l'entièreté de l'expérience, et grâce à la calibration effectuée au début, ils sont tous relativement bien réguliers et proches du 0.

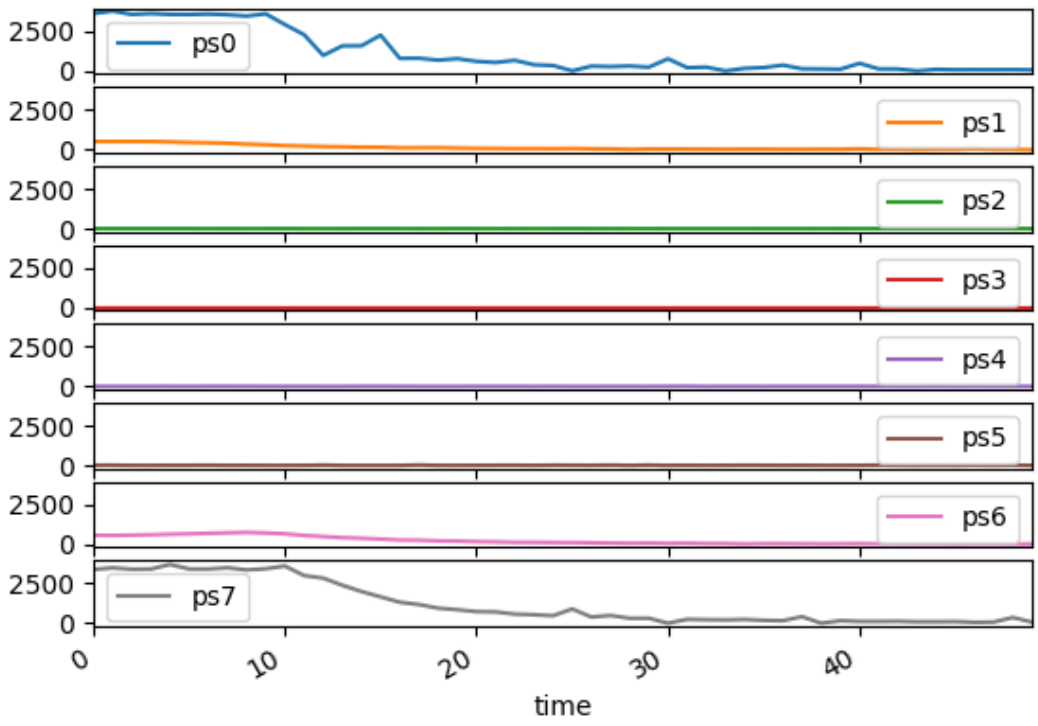

Figure 2.2 – Réponse de chaque capteur individuellement

En regroupant l'ensemble des données rassemblée, nous obtenons dans la Figure 2.3 un graphique ressemblant approximativement à la Figure 2.1. Malgré de nombreux creux et pointes dus à l'imprécision des capteurs, au bruit beaucoup plus présent en présence d'un obstacle et à l'unicité de l'expérience, nous reconnaissons la courbure générale des graphes [ps0] et [ps7]. Les résultats auraient pu être notablement améliorés en multipliant l'expérience, et en faisant une moyenne des mesures, ce qui nous aurait rapproché de la courbe régulière.

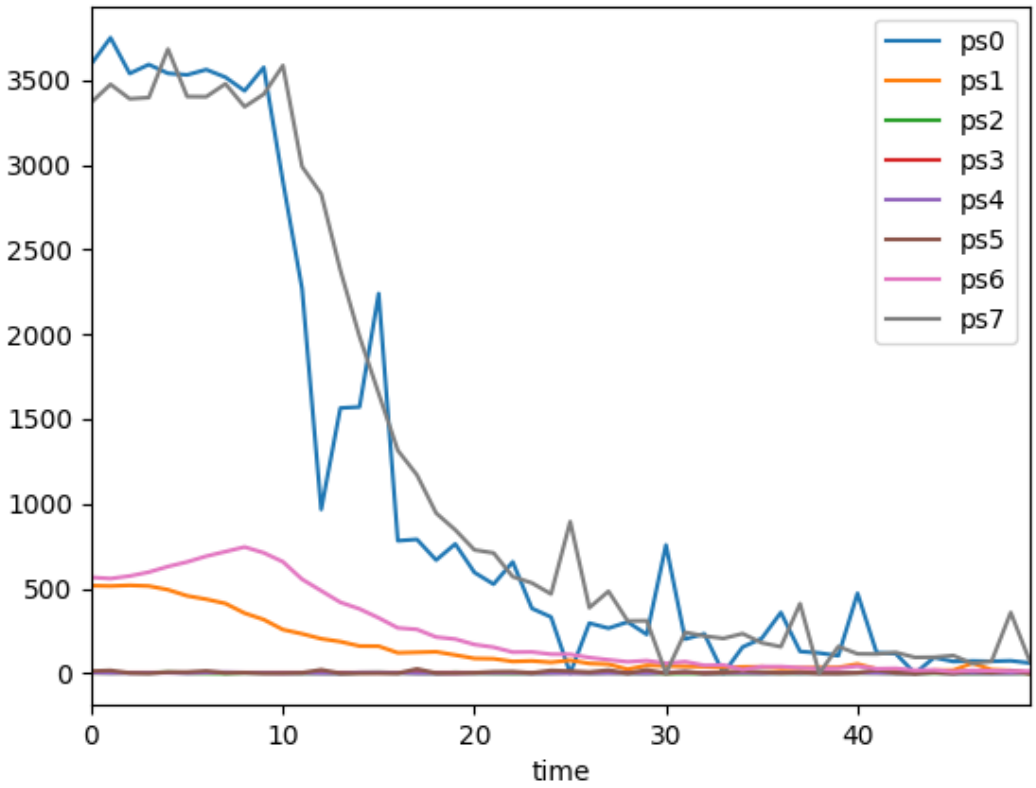

Figure 2.3 – Réponse des capteurs du robot 204

2.2 Infra-red ground sensor

Description of the sensor and graphs of measurements

2.3 Camera

Description of the camera and graphs of measurements

Chapitre 3

Behaviours

Chapter about the behaviours that will be implemented for the assignments

3.1 Braitenberg vehicle

Les machines de Braitenberg sont un concept d'étude du comportement, dans notre cas d'un robot. Valentino Braitenberg, à l'origine de cette théorie, analysait notamment les comportements animaliers, avec des modélisations simples. La machine (ou les animaux dans la réalité) vont réagir de façon proportionnelle à un stimulus, chaque moteur étant, en théorie, relié à un capteur. Voici un schéma représentant le comportement de deux simples machines de Braitenberg.

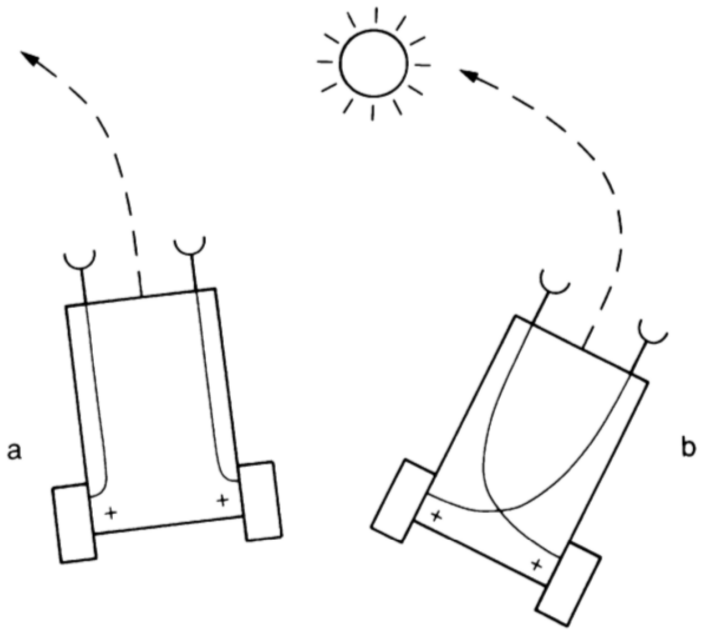

FIGURE 3.1 – 2 machines de Braitenberg

Dans cet exemple, on voit que chaque moteur (les roues) est relié à un seul capteur (la sorte de fourchette). Le signe "+" indique que la puissance du moteur augmentera proportionnellement à l'augmentation du stimulus (le soleil, une source lumineuse dans notre cas). La roue

droite du véhicule a accélérera plus que la roue gauche, ce qui explique la trajectoire s'éloignant de la source, tandis que la roue droite, éloignée de la source, du véhicule b, sera plus rapide, ce qui donne cette trajectoire se rapprochant de la source de stimuli.

Dans les exemples précédents, le comportement des machines est toujours identique, soit elle s'éloigne, soit elle s'approche d'un stimulus. Nous allons donc fabriquer une machine de Braitenberg un peu plus développé, qui est capable de passer d'un état à l'autre. D'abord, le robot devra se rapprocher d'un mur (le stimulus), et trouver son point d'équilibre. Ensuite, le robot change de mode, et doit s'éloigner du mur, avant de pouvoir revenir à son état de base. Le système aura deux entrées, stimulis : les capteurs externes nous permettant de savoir si le robot rentre un objet (mur) ou pas, et un compteur interne qui nous permettra de décider si nous sommes en position d'équilibre ou pas. Voici le schéma FSM représentant ce comportement :

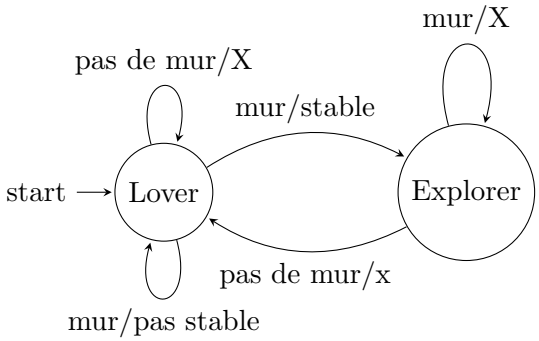

Figure 3.2 – Schéma FSM représentant le comportement à implémenter

3.1.1 LOVER

En mode LOVER, le but est que le robot avance en ligne droite, à vitesse constante, jusqu'à ce qu'il rencontre un obstacle. À ce moment, il devra ralentir progressivement sa vitesse, jusqu'à atteindre une vitesse nulle, en position d'équilibre (arbitraire). Une fois en équilibre, le robot doit allumer ses LED pour annoncer qu'il passe en mode EXPLORER.

En pratique, due aux fluctuations des données recueillies par les capteurs infra-rouge, la vitesse ne se stabilise jamais à 0, mais le robot oscille, effectuant de très courts et rapides aller-retours autour de sa position d'équilibre. Nous avons donc mis en place un compteur, qui regardera combien d'aller-retours sont effectués. Lorsqu'un nombre arbitraire est atteint, on peut déclarer que le robot est effectivement en position d'équilibre.

Si dessous, nous pouvons voir l'implémentation de ce compteur. *avant* est une variable permettant de suivre l'état actuel de la machine (un mini schéma FSM pourrait être réalisé pour représenter cette situation). Si nous étions actuellement en train d'avancer et la vitesse devient négative, alors nous comptons un aller. Dans que la vitesse négative, rien ne se passe. Puis dès que la vitesse devient positive, alors que nous étions actuellement en train de reculer, le compteur ajoute un retour, et ainsi de suite.

```
    if(avant == 0 && speed < 0){

        counter++;
        avant = 1;

    } else if(avant == 1 && speed > 0){

        counter++;
        avant = 0;

    }
```

Figure 3.3 – code permettant de décider de l'état d'équilibre

3.1.2 EXPLORER

Une fois la position d'équilibre trouvée, le robot passe en mode EXPLORER. Le robot doit évaluer les alentours et s'éloigner de la source du stimuli, façon proportionnelle : Plus un coté est proche, plus il veut s'en éloigner rapidement.

```
    double proxL = (prox_values[7] + prox_values[6] + 0.7 * prox_values[5] + 0.2 * prox_values[4]) / 2.9;
    double proxR = (prox_values[0] + prox_values[1] + 0.7 * prox_values[2] + 0.2 * prox_values[3]) / 2.9;
    double dsL = (NORM_SPEED * proxL) / MAX_PROX;
    double dsR = (NORM_SPEED * proxR) / MAX_PROX;
    double speedL = bounded_speed(NORM_SPEED - 550 + dsL);
    double speedR = bounded_speed(NORM_SPEED - 550 + dsR);

    set_speed(speedL,speedR);
```

Figure 3.4 – code EXPLORER pour s'éloigner de la source

Afin de pouvoir s'éloigner du mur "fluidement", le robot doit évaluer la proximité de la source (le mur) par rapport à sa gauche et sa droite. L'important, est d'avoir la voie libre devant lui, car le but final est de pouvoir repartir en ligne droite, on voit donc, dans le code ci-dessus, que les valeurs captées par les capteurs latéraux ont une importances réduites. On les garde, afin de ne pas être totalement aveugle, au cas ou se trouverait face à un mur arrondi par exemple, et que le coté du robot touche alors que l'avant n'a rien repéré, mais leurs coefficients (0.7 pour les capteurs latéraux et 0.2 pour les capteurs arrières) permettent de mettre surtout l'importance sur ce qui se passe devant.

Une autre subtilité de notre code se trouve dans la formule de la vitesse, le *-550*. Le comportement de base consiste simplement à ajouter *dsL* ou *dsR*, un nombre positif, en fonction du rapprochement du mur. Or une formule de base *NORM SPEED + dsL* possède 2 problèmes.

D'abord, si le mur est trop proche, *dsR* et *dsL* peuvent être les deux très élevés, et donc potentiellement supérieurs à la vitesse max. Dans un cas où les deux cotés seraient supérieurs à la vitesse max, ils seraient les deux bornés à la même valeur, propulsant le robot tout droit, même si une des roues est sensée tourner plus vite que l'autre, permettant un virage.

Ensuite, les manœuvres à grande vitesse sont propices aux dérapages et perte d'adhérence, en réduisant arbitrairement à la vitesse lors de ces manœuvres, on s'assure un risque réduit de trajectoires faussées par un patinage. Comme *550* est supérieur aux 400 contenue dans *NORM SPEED*, la roue a une vitesse négative lorsque le *ds* est nul, ou très bas. Donc le robot va pivoter sur lui-même afin de se réorienter plutôt que de suivre une trajectoire courbe, qui peut régulièrement entraîner une collision avec le mur.

```
1    if((prox_values[7] + prox_values[6] + 0.01 * prox_values[5] + prox_values[0] + prox_values[1] + 0.01 * prox_values
         [2]) < 50){

3        status = 0;
         toggle_led(0);
5        toggle_led(1);
         toggle_led(2);
7        toggle_led(3);
```

Figure 3.5 – code EXPLORER pour retourner en LOVER

Voici ci-dessus le code permettant à notre robot de passer de mode EXPLORER à LOVER. Le coefficient "0.01" accolés aux capteurs latéraux est là pour éviter une situation où le robot serait collé latéralement au mur. C'est un très petit coefficient car si la voie est libre (ce qui se vérifie avec les 4 capteurs avants), le robot peut longer le mur, mais, s'il est trop proche, il risque de toucher le mur. Nous avons vu avec la figure 2.1 que les valeurs renvoyés sont très élevées, puis chutent rapidement. Ce coefficient permet de faire la différence, une valeur haute, dans les 3'000, aura une grande influence, mais 2 cm plus loin, à moins de 300, le capteur n'aura déjà presque plus aucune influence, car le robot peut considérer cette position comme acceptable, il peut longer le mur du moment qu'il y a quelques centimètres d'espace.

Voici une vidéo présentant notre robot appliquant ce comportement. Les LED allumés annoncent que le robot est en mode EXPLORER : `https://youtu.be/rDlxz8RVZrM`

3.2 Line-following

Description of the line following behaviour using a braitenberg (reactive) controller

3.3 Wall-following

Description of the wall-following behaviour using a PID controller (including a description of the PID in general)

3.4 Color recognition

Description of the color recognition behaviour

3.5 Multi-robot coordination

Description of the Multi-robot coordination using communication between robots

Chapitre 4

Conclusion

Synthesis of the report and outlook for further work.

Bibliographie

[1] Justin Zobel. *Writing for Computer Science*, 2nd edition. Springer-Verlag, London, 2004, 275 pages.

[2] Valentino Braitenberg. *Vehicles : Experiments in Synthetic Psychology*. MIT Press, 1986.

[3] *Webots Reference Manual*. `https://www.cyberbotics.com/reference.pdf` version 2019a Last visited : 11.02.2019.

Appendix

Appendix A Experimental Results

Place to list the gathered data.

Appendix B Source Code

Place to list source code.

B.1 IR sensors calibration procedure

The code below shows the IR sensor calibration procedure.

```
// get the correction values for prox sensors
void get_prox_corr_vals() {
  int i, j;

  // init array for calibration values
  for (i=0; i<PROX_SENSORS_NUMBER; i++) {
    prox_corr_vals[i] = 0;
  }

  // get multiple readings for each sensor
  for (j=0; j<NBR_CALIB && wb_robot_step(TIME_STEP)!=-1; j++) {
    for (i=0; i<PROX_SENSORS_NUMBER; i++) {
      prox_corr_vals[i] += wb_distance_sensor_get_value(prox_sensor_tags[i]);
    }
  }

  // calculate average for each sensor
  for (i=0; i<PROX_SENSORS_NUMBER; i++) {
    prox_corr_vals[i] = prox_corr_vals[i] / NBR_CALIB;
  }
}
```