Rapport du projet d'ingénierie robotique

Maxence Ahlouche Maxime Arthaud Korantin Auguste
Martin Carton Thomas Forgione Thomas Wagner

9 décembre 2013

Table des matières

1 Présentation de l'équipe

Cette équipe a été menée par Maxime Arthaud, assisté de son Responsable Qualité Thomas Forgione. Les autres membres de l'équipe sont Martin Carton, Maxence Ahlouche, Korantin Auguste et Thomas Wagner.

2 Méthodes de programmation du robot

2.1 Programmation graphique

Au début du projet, nous avons commencé à programmer le robot comme indiqué dans le sujet, c'est à dire en utilisant le logiciel de programmation graphique disponible sur Windows. Seulement, ce logiciel, bien qu'accessible à tout le monde, nous a semblé très peu pratique à utiliser, si bien que nous n'avons pas été très productif lors des premières séances.

En effet, dès que nous souhaitions faire autre chose que des instructions de base (par exemple, faire tourner le moteur à une vitesse dépendant de la valeur d'un capteur), nous passions beaucoup trop de temps à essayer de comprendre comment fonctionnait l'interface, et à nous battre avec le programme, qui plantait souvent.

De plus, nous avons rencontré un problème que nous n'avons pas réussi à résoudre : notre robot avançait de manière saccadée. Bien que nous ayons quelques hypothèses sur l'origine de ce problème (bug ou documentation non à jour), nous n'avons pas réussi à le résoudre avec le logiciel fourni.

À cause de ces désagréments, nous avons décidé d'essayer d'autres méthodes pour programmer le robot.

2.2 Librairie Python

Un des membres de notre équipe a trouvé sur Internet une librairie en Python, qui permet de contrôler un robot branché en USB. Cette librairie (qui s'appelle `nxt-python`) nous a permis de régler notre problème d'avancement saccadé.

Elle nous a également permis de programmer notre robot de manière très simple, et de gagner beaucoup de temps par rapport au logiciel de programmation graphique.

Elle présente toutefois un inconvénient majeur : elle ne permet pas de télécharger le programme sur le robot (le python étant interprété sur l'ordinateur) ; par conséquent, le robot devait rester branché à l'ordinateur lors de l'exécution du programme, et nous devions le suivre avec l'ordinateur. Ce qui n'est évidemment pas pratique.

2.3 Langage NXC

Finalement, nous avons décidé de programmer le robot en utilisant le langage NXC (*Not Exactly C*), développé spécifiquement pour les robots NXT.

Le compilateur que nous avons trouvé nous permet également de télécharger le programme sur le robot, donc nous avons résolu le problème posé par la librairie en Python, ainsi que ceux posés par le logiciel de programmation graphique.

3 Suivi de murs

Nous avons réalisé un programme qui permet au robot de longer un mur, grâce à NXC.

3.1 Suivi d'un mur

Afin de longer un mur, nous avons posé un capteur à ultrasons sur le côté gauche de notre robot. Lorsque nous détectons que nous sommes trop éloignés du mur, nous tournons légèrement à gauche.

Cette méthode est plutôt efficace, mais pose un problème majeur : lorsqu'il y a un angle droit, le robot ne peut pas savoir qu'il y a un mur devant droit lui, et le percute. Nous avons donc décidé d'ajouter un autre capteur à ultrasons à l'avant du robot. Ainsi, notre robot ne fonce plus dans les murs, et tourne à droite quand le capteur de devant détecte quelque chose.

Ainsi, nous pouvons suivre un mur de manière efficace.

3.2 Lissage du mouvement du robot

Pour avoir un déplacement plus fluide de notre robot, avec moins d'à-coups, nous avons utilisé la technique de *régulateur PID*, avec seulement l'action proportionnelle (P).

Le PID est une technique de correction d'erreur utilisé couramment en robotique. Il s'agit d'un algorithme calculant la correction a effectuer en fonction des mesures. Connaissant la mesure et la valeur voulue, on en déduit l'erreur. La correction de cette erreur se fait par 3 actions :

— une action proportionnelle P : on multiplie l'erreur par une constante
— une action intégrale I : on intègre l'erreur, et on la divise par une constante
— une action D : on dérive l'erreur, et on la multiplie par une constante

Finalement, on somme les 3 actions pour obtenir la variation à appliquer à notre commande.

Concrètement, dans notre cas, il nous a suffit de faire la différence entre la distance mesurée par le capteur de gauche et la distance voulue, puis de multiplier ceci par une constante. On ajoute cette valeur à la puissance initiale des moteurs pour avoir un déplacement proportionnel à la distance au mur. Avec cette méthode, nous avons un robot se déplaçant de manière très fluide.

Pour de vrai robot, les constantes multiplicatives dans le PID demandent des réglages fins, car ils influent sur l'efficacité de l'algorithme. Dans notre cas, il nous a suffit de tester avec moins d'une dizaine de valeur pour trouver quelque chose de satisfaisant.

3.3 Recherche d'un mur

Dans le cas où le robot est placé loin d'un mur, il va se mettre à tourner sur lui même. Nous avons donc eu l'idée de faire en sorte qu'il fasse plutôt une spirale pour rechercher le mur le plus proche.

4 Labyrinthe

Pour résoudre le problème du labyrinthe, nous avons simplement utilisé le suivi de mur gauche. En effet cela permet de sortir d'un labyrinthe sans ilots, même si ce n'est pas forcement efficace.

Nous avons aussi réfléchi à deux algorithmes plus efficaces : Dijkstra (que nous avions déjà implémenté dans l'UA1) et A*. Nous n'avons pas implémenté ces algorithmes sur le robot, cas nous ne savions pas comment avec un robot Lego nous aurions pu réaliser le parcours du labyrinthe en enregistrant une carte de celui-ci. Or ces algorithmes nécessitent que le labyrinthe soit connu.

Le problème du positionnement du robot est aussi très délicat : en se basant uniquement sur les ordres qu'on envoie aux moteurs et sur la position initiale, on ne peut pas connaître avec précision notre position, car les roues dérapent souvent sur le sol, et plus on se déplace plus les imprécisions vont s'accumuler, ce qui donne une mesure de plus en plus erronée.

5 Annexes

Listings

Listing 1 – Robot suiveur de murs

```
#define LEFT_MOTOR OUT_B
#define RIGHT_MOTOR OUT_A
#define MOTOR_POWER 100

#define ULTRASONIC_FRONT IN_1
#define FRONT_MIN_DISTANCE 20

#define ULTRASONIC_LEFT IN_4
#define LEFT_OBJECTIF 12
#define LEFT_C 4

#define FRONT_WAIT 700

#define SPIRALE_K 50

task main()
{
    float c, d, cmd_left, cmd_right, infinite_left = 0;
    SetSensorLowspeed(ULTRASONIC_FRONT);
    SetSensorLowspeed(ULTRASONIC_LEFT);

    while(1)
    {
        d = SensorUS(ULTRASONIC_LEFT);

        if(d > 2 * LEFT_OBJECTIF)
        {
            d = 2 * LEFT_OBJECTIF;
            infinite_left++;
```

```
        }
        else if(d < LEFT_OBJECTIF)
        {
            infinite_left = 0;
        }

        if(SensorUS(ULTRASONIC_FRONT) < FRONT_MIN_DISTANCE)
        {
            OnRev(LEFT_MOTOR, MOTOR_POWER / 2);
            OnFwd(RIGHT_MOTOR, MOTOR_POWER / 2);
            Wait(FRONT_WAIT);
        }
        else
        {
            c = (d - LEFT_OBJECTIF) * LEFT_C - infinite_left / SPIRALE_K;

            cmd_left = MOTOR_POWER - c;
            if(cmd_left > 120)
                cmd_left = 120;
            if(cmd_left < 0)
                cmd_left = 0;

            cmd_right = MOTOR_POWER + c;
            if(cmd_right > 120)
                cmd_right = 120;
            if(cmd_right < 0)
                cmd_right = 0;

            OnRev(LEFT_MOTOR, cmd_left);
            OnRev(RIGHT_MOTOR, cmd_right);
        }
    }
}
```