Heitor ABRE DE ANDRADE
Malaurie BERNARD

TP 1 automatique - Recherche de chemins dans un graphe : LegoRover

Introduction

Dans ce tp, l'objectif est de trouver le chemin optimal selon différents critères (temps, plus court). Dans un premier temps il faudra configurer le robot pour qu'il suive une ligne courbe, ensuite nous étudierons et observerons différentes méthodes pour trouver le meilleur chemin selon le critère choisi.
Pour cela nous allons utiliser la carte ci-dessous, constituée d'une ligne courbe pour tester si le robot suit bien la ligne, et de plusieurs points reliés par des segments. Ainsi nous pourrons chercher le meilleur chemin pour aller d'un point à un autre.

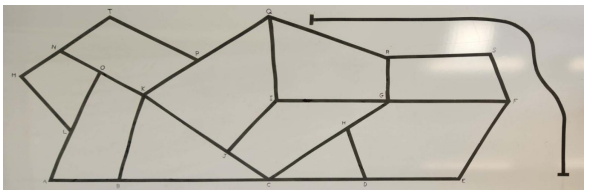

Figure 1 - Carte des trajets

I - Mise en route du robot

On a commencé par allumer et connecter le robot en bluetooth à l'ordinateur. Pour vérifier que la connexion est bien établie, nous avons fait une commande ping avec succès.

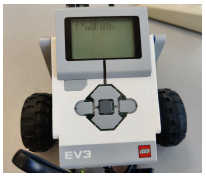

Figure 1 - Maquette connecté à notre ordinateur

```
U:\>ping 10.0.1.1

Envoi d'une requête 'Ping'  10.0.1.1 avec 32 octets de données :
Réponse de 10.0.1.1 : octets=32 temps=59 ms TTL=64
Réponse de 10.0.1.1 : octets=32 temps=36 ms TTL=64
Réponse de 10.0.1.1 : octets=32 temps=29 ms TTL=64
Réponse de 10.0.1.1 : octets=32 temps=39 ms TTL=64

Statistiques Ping pour 10.0.1.1:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
    Minimum = 29ms, Maximum = 59ms, Moyenne = 40ms
```

Figure 2 - Vérification de la connexion

Pour finir la préparation du robot, nous avons effectué un calibrage en plaçant les capteurs lumineux du robot sur une zone toute noire.

II - Mise en place de la partie guidage

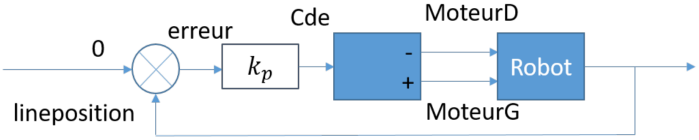

Figure 3 - Schéma de régulation

➢ Pour pouvoir vérifier que le robot suit correctement une ligne, on a besoin d'une chaîne de retour. Afin de pouvoir gérer la stabilité du système, on met en place un correcteur simple proportionnel. Les moteurs des roues reçoivent des commandes différentes pour permettre au robot de tourner lors d'une courbe.

```
Cde = linePosition*kp;
motorD.setSpeed( speed - Cde ); // speed - value in degrees/sec
motorG.setSpeed( speed + Cde ); // speed - value in degrees/sec
```

Figure 4 - Fichier pilotRoberto.java complété

➢ On remarque que les deux moteur reçoivent une valeur de vitesse qui vaut la vitesse que le robot doit avoir plus ou moins une valeur qui dépend de l'angle de rotation voulue (=0 si on est sur une ligne droite)

```
//test sur la ligne courbe
//listOfOrders.add(new Order(0, 2000, 300)); // tester le suivi de ligne
```

Figure 5 - Fichier test_BasNiveau.java complété pour suivre la courbe

➢ Notre premier test était avec une vitesse à 300°/s, une distance de 2 mètre et un Kp = 10. Le robot réussit à suivre la ligne courbe.
➢ Voici ci-dessous ensuite tous nos tests de différentes valeurs de Kp et vitesse sur la ligne courbe pour une distance de 2 mètres.

Kp = 10 V =100	Kp = 10 V =200	Kp = 10 V =300	Kp = 10 V =400

Kp = 5 V = 300	Kp = 10 V = 300	Kp = 20 V =300	Kp = 30 V =300	Kp = 40 V =300

Finalement, si la vitesse est trop grande alors on perd la ligne car la correction est trop lente par rapport à la vitesse. Si kp est trop grand il y a de fortes oscillations car la correction trop forte, on finit aussi par perdre la ligne.
Dans tous les cas, si on a trop d'oscillations on ne finit pas le trajet car on parcourt une distance plus grande.

III - Mise en place de la partie Gestion de la trajectoire

Nous avons mesuré les angles et les distances du trajet entre A et Q, et nous avons à l'aide de la fonction listOfOrders rentré nos valeurs pour que le robot parcourt le trajet voulu.

```
//test qui fait A, B, K, P, Q
listOfOrders.add(new Order(0, 420, 360));  // A to B
listOfOrders.add(new Order(-75, 530, 360)); // B to K
listOfOrders.add(new Order(35, 330, 360)); // K to P
listOfOrders.add(new Order(0, 430, 360)); // P to Q
```

Figure 6 - Fichier test_BasNiveau.java complété pour suivre aller du point A au point Q et QR code pour voir l'expérience associé

IV - Mise en place de la partie Gestion du plan

- ➢ Formalisation du Problème de Chemin
  - ○ ensemble de sommets = {A,B,C,D,E,....}
  - ○ ensemble d'arcs reliant ces sommets = {(A,B),.....}
  - ○ poid d'arcs W(A, B) = distance ou temp

Ainsi, la carte des trajets peut être modélisée par un graphe. Il est à noter que ce graphe n'est pas orienté car le robot doit pouvoir se déplacer dans n'importe quel sens.

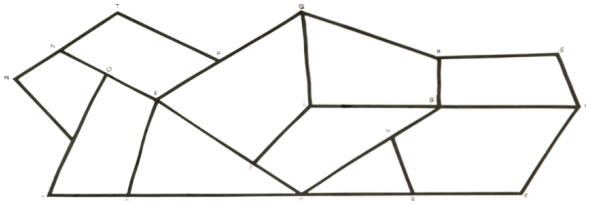

Figure 1 - Graphe des chemins

- ➢ Algorithme glouton : On calcul la distance de tous les chemins possibles, et on garde le plus court. Cet algorithme est lourd en calcul et donc pas très efficace.

```
// recherche du noeud de valeur minimale non encore exploré
for (int i = 0; i < carte[noeud_init].length; i++) {
        ////////////////////////////////////////////////////////
                ////////////////////////////////////////////////////////
      if ( tab_value[i]<valmin && tab_bool[i] == false ) {
                valmin = tab_value[i];  //On actuakise la valeur minimal
                noeud_retenu = i;  //On actualise l'indice du noeud qu'on retiens
                }
                }
```

Figure 8 - Fonction Touvmin du Fichier Djikstra.java complété

```
public void MajCcum(int noeud_retenu) {
    int i;
    double[][] map = this.map.getMap();

    for (i = 0; i < map[this.noeud_init].length; i++) {
        // si le crit□re du noeud i en passant par le noeud retenu est meilleur
        // alors on change de chemin
        if( (tab_value[noeud_retenu] + map[noeud_retenu][i]) < tab_value[i] ) {
            tab_value[i]=tab_value[noeud_retenu]+map[noeud_retenu][i];
            tab_noeuds[i]=noeud_retenu;
        }
    }
}
```

Figure 8 - Fonction MajCcum du Fichier Djikstra.java complété

```
// définition du critère du Dijkstra
this.Map[li][col] = distance;
this.Map[col][li] = distance;
```

Figure 7 - Fichier Map.java complété avec un critère de Dijkstra de distance et QR code pour voir l'expérience associé

```
// définition du critère du Dijkstra
this.Map[li][col] = distance/vitesse;
this.Map[col][li] = distance/vitesse;
```

Figure 8 - Fichier Map.java complété avec un critère de Dijkstra de temps et QR code pour voir l'expérience associé

Nous pouvons donc voir que l'algorithme de Djikstra nous donne un résultat différent et cohérent pour la vitesse et pour la distance.

V - Extension Possibles

- Une solution pour que le robot quitte un nœud de départ et arrive à un nœud final en passant par des points spécifiés et non ordonnés consiste à former tous les chemins possibles qui permettent d'accomplir la mission, à utiliser l'algorithme de Dijkstra pour calculer le coût entre chaque nœud, à additionner les coûts des nœuds pour obtenir le coût du chemin, et enfin à choisir le chemin ayant le coût le plus faible.
  Par exemple :
  nœud de départ : A
  nœud final : D
  nœuds intermédiaires : B et C
  Étapes
  1. Produire tous les chemins possibles
     chemins = {(A, B, C, D), (A, C, B, D)}
  2. Prenez chaque chemin et calculez son coût en utilisant Dijkstra.
     Exemple avec le premier chemin : coût total du chemin = coût(dijkstra(A, B) + coût(dijkstra(B, C)) + coût(dijkstra(C, D))
  3. Choisir le chemin dont le coût total est le plus bas

- Si le robot dispose de ressources limitées, il peut en laisser la moitié pour faire un trajet et l'autre moitié pour revenir. Après avoir fixé ce coût d'aller, il teste les nœuds d'extrémité possibles jusqu'à ce qu'il en trouve un dont le coût est inférieur à celui de la ressource.
- Dans le cas du Mars Rover, nous combinons les deux idées ci-dessus, générons tous les chemins avec les points souhaités, calculons les coûts et vérifions la possibilité d'aller et de revenir, et choisissons finalement le chemin qui maximise les récompenses.

VI - Conclusion

Finalement, à travers ce TP, nous avons pu observer une application pratique de la théorie des graphes. De plus, nous avons pu constater l'action du correcteur et de la vitesse sur la capacité du robot à suivre la ligne. Ensuite, nous avons utilisé l'algorithme de Djikstra et observé la différence de résultat en fonction du critère de Djikstra choisi.