RAPPORT MÉTHODE COMPUTATIONNELLE FUZZY ROBOT

Maxime Yonnet SIIA1

I. Démarche

Le problème auquel on s'intéresse est celui du pilotage d'un robot-mobile dans un environnement inconnu. L'objectif du robot est de rejoindre un ensemble de buts placés aléatoirement tout en évitant les obstacles présents.

Pour se faire nous utilisons la logique flou (Fuzzy Robot). Le principe est d'utiliser des nombres entiers au lieu d'utiliser des booléens, on ne fera plus des conditions booléennes mais on regardera le degré d'appartenance à un fait, c'est-à-dire à quel point quelque chose est vrai et selon ce résultat on donnera des ordres de puissances différentes. Les ordres données eux ne peuvent pas être en flou, nous devons les convertir en valeurs classiques. Les conditions sont marquées en français et sont complétées par des ensembles définissant les degrés d'appartenance. Des règles sont aussi appliquées pour définir comment choisir quelle règle est prioritaire et pouvoir lier des règles.

II. Contrôleur

Dans le projet nous avons des fichiers à l'extension "des" qui représente tous les obstacles de la carte, un fichier .fuz qui décrit tous les comportements que doit avoir le robot et enfin l'exécutable qui s'occupe de faire l'affichage et convertir les comportements en action réel.

C'est dans le fichier fuz que l'on décrit toutes les règles devant être exécuté et les ensembles. A chaque frame, le robot évalue les règles et définit des valeurs de sorties en fonctions.

Pour ce projet j'ai utilisé la configuration suivante :

```
configuration
{
  et := min;
  ou := max;
  decodage := centre;
  implication := larsen;
  ainsi_que := somme;
}
```

Ce sont les valeurs qui étaient par défaut dans le programme, j'ai modifié "ainsi_que" avec la valeur "max" lors de mes tests, cela me permettait de mieux comprendre quelle règle étaient utilisées et quelles valeurs étaient envoyées. J'ai ensuite remis la valeur de somme car cela permet d'avoir des transitions entre les règles plus fluides et donc avoir des mouvements moins brusque pour la voiture. Pour le reste je n'ai pas remarqué de différence.

III. Ensemble entrée et sortie

Dans ce projet nous avons donc des données d'entrées et de sorties. Parmi les données d'entrée nous avons la distance de l'obstacle le plus proche dans les 4 directions, mais aussi la distance de l'objectif et dans quelle direction il se trouve. Le programme peut seulement influer sur l'orientation du robot et sa vitesse.

Pour ce projet nous avons donc une multitude d'ensemble afin d'affiner les degrés d'appartenance, en voici les plus importants.

```
turnRight := triangle(-150.0,-75.0,0.0);
turnLeft := triangle(0.0,75.0,150.0);
```

Ces deux ensembles permettent au robot de tourner quand une règle le veut. J'ai décidé de ne pas centrer sur la vitesse de rotation maximale afin d'éviter que le robot ne fasse des virages trop brusques et finissent dans un mur. J'ai aussi les règles turnTinyRight et turnTinyLeft qui ont la même fonction mais de façon moins extrême (le triangle est plus petit et centré sur un nombre plus proche de 0).

```
  close :=  rampe_bas(250,400.0);
  mediumClose := rampe_bas(90,250);
  far :=  rampe_haut(90.0,800.0);
  veryClose := rampe_bas(90,125);
  veryCloseSide := rampe_bas(15,150);
```

Ici nous avons 4 ensembles permettant d'évaluer diverses distances. Pour le veryClose le minimum est 90 car c'est la distance minimum avant que l'avant de la voiture ne soit en contact avec l'objet. Le veryCloseSide permet d'évaluer une distance entre un côté de la voiture est un objet car la voiture sera en contact avec l'objet quand la distance sera égale à 0, j'ai fait le choix de ne pas mettre 0 mais 15 afin de laisser un peu de marge pour permettre au robot de manoeuvrer. Pour Close, mediumClose et far ce sont des valeurs choisies arbitrairement selon ma vision des distances.

```
  stop := triangle(-100.0,0.0,50.0);
  slow := triangle(-100,50.0,150.0);
  fast := triangle(125,150,151.0);
```

Enfin pour les vitesses nous avons trois vitesses. Le fast est utilisé quand la voie est libre, que tous les obstacles sont loins et donc que la voiture peut aller vite. Le slow est utilisé quand de la prudence est requise, notamment parce que des obstacles sont proches et que des manœuvres sont requises. Le stop lui est utilisé quand le robot est trop proche d'un obstacle et doit s'arrêter sinon un accident va arriver. J'ai utilisé des rampes lorsqu'il n'y avait pas de valeur minimale ou maximale et que ce ne sont pas des valeurs de sorties, sinon j'ai utilisé des triangles.

IV. Règles

Pour ce projet nous avons aussi une multitude de règles utilisant les ensembles décrits précédemment afin d'indiquer au robot comment se comporter.

```
  si DirecGoal est inFront et ObstFront est pas  veryClose alors Sang
est strainghton;                                          //R1

  si DirecGoal est onRight et ObstRight est pas veryCloseSide et
ObstFront est pas veryClose alors Sang est turnRight;        //R2

  si DirecGoal est onLeft et ObstLeft est pas veryCloseSide et ObstFront
est pas veryClose alors Sang est turnLeft;           //R3
```

Nous avons tout d'abord les règles permettant au robot de s'orienter par rapport à l'objectif. A savoir que le robot ne va se diriger vers l'objectif que s' il n'y a pas d'obstacle très proche en face de lui et dans la direction ou il doit s'orienter.

```
  si DistGoal est mediumClose alors Slin est slow;
  si DistGoal est far et ObstFront est far alors Slin est fast;
  si ObstFront est veryClose alors Slin est stop;
  si ObstFront est close alors Slin est slow;
```

Ici nous avons les règles décidant de la vitesse du robot.
Le robot va vite s'il n'y a pas d'obstacle et ralentit lorsqu'il s'approche d'un objet, il s'arrête lorsqu'il est trop proche d'un obstacle.

```
  si ObstFront est veryClose et ObstRight est pas veryCloseSide alors
Sang est turnRight;
  si ObstRight est veryCloseSide alors Sang est turnTinyLeft;
  si ObstLeft est veryCloseSide alors Sang est turnTinyRight;
```

Enfin nous avons les règles qui permettent d'esquiver les obstacles. Si un obstacle est devant, est trop proche et qu'il n'y a rien à droite, alors le robot va tourner à droite pour contourner l'obstacle. Si il y a un mur vraiment proche sur un des côtés, alors le robot va tourner un tout petit peu afin de longer ce mur.

V. Résultats

D’après les nombreux tests que j’ai fait, le robot marche à 100% sur la carte 0 et 1. Pour ce qui est de la carte 2, le robot fonctionne la plupart du temps. Cependant il peut arriver que le robot tourne trop brusquement et se prenne un mur. Si le robot longe un mur qui est à sa droite et rencontre un mur en face, alors il ne peut pas tourner et est contraint d’avoir un accident. Il arrive aussi que le robot se coince dans la carte 3 notamment, à chaque fois qu’il rencontre un mur, il tourne à 90 degrés, et rencontre un autre mur, et si il n’en rencontre pas, il veut s’orienter vers l’objectif le forçant à rencontrer un mur et à tourner en rond.

VI. Problèmes Rencontrés

Habituellement, lorsque je travaille sur un projet, j’essaie toujours de surenchérir sur ce que j’ai fait afin d’avoir un meilleur résultat, mais ici j’ai vite compris que si l’on met trop de règles, alors on perd la notion de ce qu’il se passe et si l’on met vraiment beaucoup de règles alors on peut perdre la notion de logique floue. Il m’a donc été compliqué d’essayer de simplifier au maximum mes règles afin d’avoir quelque chose de simple et fonctionnel. Avec plus de temps j’aurais pu ajouter des règles afin de faire marcher la carte 3.

Aussi pour faire fonctionner le programme, j’utilisais un SubSystem, sauf que lorsque j'édite le fichier sous windows et que je l'exécute sous ce subSystem linux, l’encodage n’étant pas le même, le fichier n’était pas reconnu. Des fois aussi, je mettais des commentaires qui faisaient crash le programme.