

# Méthodes Formelles

## TP Logique Floue

Valentin Guerlesquin  
Shulong Ling

1<sup>er</sup> février 2007

### 1 Objectif du TP

Il s'agit de se familiariser avec l'écriture de règles en logique floue. Dans une première partie, nous cherchons à appréhender l'influence des différents paramètres, puis nous tenterons d'écrire des règles destinées au contrôle d'un robot.

### 2 Approximation d'une fonction de contrôle

Dans cette première partie, on s'intéresse à l'écriture de règles floues permettant la représentation des surfaces floues présentées dans le sujet. Les surfaces en question sont assez simples, et deux d'entre elles sont des surfaces fonction d'un seul paramètre. Le but de cette partie est de mieux appréhender l'influence des règles, des types d'inférence, des types d'ensemble flous, sur le résultat obtenu.

### 3 Contrôle d'un comportement réactif

Il s'agit maintenant d'écrire des règles floues destinées au contrôle d'un robot dans un environnement comportant des obstacles.

#### 3.1 Le robot et son environnement

Le robot que l'on souhaite contrôler évolue sur une surface plane, carrée, comportant une cible et, éventuellement, des obstacles. Le robot doit rejoindre la cible sans heurter les obstacles. Une fois la cible atteinte par le robot, celle-ci ré-apparaît à un autre endroit.

Le robot dispose en permanence de 8 informations :

- Sa distance par rapport à la cible (distance “à vol d'oiseau”)
- Sa direction par rapport à la cible
- Sa vitesse linéaire actuelle

- Sa vitesse angulaire actuelle
  - Des informations sur la distance relatives des plus proches obstacles l'entourant, et cela à l'avant, arrière, gauche et droite.
- Le contrôleur du robot peut agir sur les choses suivantes :
- La vitesse linéaire
  - La vitesse angulaire

## 3.2 Règles de contrôle

La fonction de contrôle floue du robot est du type  $\mathbb{R}^8 \mapsto \mathbb{R}^2$ . Il est donc nécessaire de procéder par étapes pour l'élaboration de celles-ci.

### 3.2.1 Environnement sans obstacles

**Règles naïves** L'écriture des règles dans un environnement sans obstacle paraît assez simple. En effet, on peut facilement imaginer que des règles du type

```
si DistGoal est loinDuBut alors Slin est Vite ;
si DistGoal est presDuBut alors Slin est Lent ;
```

pour la vitesse linéaire et, pour la vitesse angulaire :

```
si DirectGoal est butADroite alors Sang est aDroite ;
si DirectGoal est butAGauche alors Sang est aGauche ;
si DirectGoal est butToutDroit alors Sang est toutDroit ;
```

suffiront pour le contrôle du robot.

**Mangeons de la brique** Les règles précédentes fonctionnent plutôt bien, cependant, si l'on s'approche trop près d'un mur, on a de grandes chances de s'y cogner. Et pour cause : on ne tient pas compte des obstacles, bien que les murs en soient. On se propose donc de modifier quelque peu les règles afin de tenir compte de cette situation.

On a alors :

```
si DistGoal est loinDuBut
et ObstFront est obstaclePres
alors Slin est Lent ;
si DistGoal est loinDuBut
et ObstFront est obstacleLoin
alors Slin est Vite ;
si DistGoal est presDuBut alors Slin est Lent ;
```

On s'en sort plutôt pas mal<sup>1</sup>.

---

<sup>1</sup>Remarquons cependant une chose : pour se sortir d'un cas particulier (rencontre inopinée avec un mur), il a été nécessaire d'ajouter une règle.

### 3.2.2 Des obstacles

Notre robot est censé évoluer dans un environnement jonché d’obstacles. Il est possible d’utiliser des cartes comportant un nombre croissant d’obstacles. On teste alors nos règles précédentes sur la carte la plus simple, et on est amené à écrire d’autres règles (ou à les modifier) afin que notre robot se comporte bien dans son nouvel environnement. Il faut notamment tenir compte de la présence d’obstacles à gauche, à droite, arrière et avant.

Nous ne ferons pas le détails de toutes les règles, car celles-ci sont amenées à évoluer. Nous remarquerons cependant que, pour chaque coté, il est nécessaire d’ajouter au moins une règle permettant de gérer les obstacles.

On obtient sans trop de peine des règles permettant de diriger le robot sans qu’il ne percute un mur. Cependant, on se retrouve avec un robot qui, assez souvent, est coincé : la présence d’obstacles autour de lui fait qu’il ne peut plus avancer. Pour résoudre ce problème (un cas particulier, probablement<sup>2</sup>), on se propose d’ajouter une règle l’autorisant à tourner à gauche ou à droite lorsqu’il se retrouve immobile, et ce même si la cible se trouve déjà droit devant lui (une règle de contournement, en quelques sortes).

La règle en question peut ressembler à ça :

```
si DistGoal est loinDuBut
et InSlin est aLArret
et ObstFront est obstaclePres
et InSang est toutDroit
alors Sang est aGauche ;
```

Cela permet de se sortir de certaines situations. Mais, encore une fois, d’autres cas particuliers apparaissent, et il est alors nécessaire d’écrire d’autres règles destinées à nous sortir de ce que l’on pense être des extremums locaux de la surface de contrôle (ou d’une quelconque surface, bref, des situations pas glop. . .)

## 4 Conclusions

**Le contrôle flou : un problème globalement simple. . .** La mise au point de règles permettant le contrôle d’un tel robot peut sembler, a priori, relativement simple. En effet, la logique floue s’appliquant particulièrement au comportement humain, il “suffit” de retranscrire le comportement que l’on adopterait pour piloter ce robot.

**. . .qui s’avère localement compliqué** Cependant, la mise en évidence de situations bloquantes nous conduisent à l’écriture de règles de “contournement”, qui certes peuvent nous sortir d’un cas particulier, ou d’un groupe de situations, mais qui ne nous garantissent ni un fonctionnement exhaustif, ni même l’absence d’effets de bords. Qui nous dit que la règle solutionnant une situation particulière de blocage ne va pas en générer une inexistante auparavant ?

---

<sup>2</sup>Mais parmi combien d’autres ?

**La génétique à notre secours ?** Paresseux que nous sommes<sup>3</sup>, il nous vient à l'esprit que, sans doute, l'utilisation d'un algorithme génétique pour la configuration du contrôleur flou (de sa linguistique notamment) nous permettrait d'arriver plus facilement et plus sûrement à nos fins. Il est en effet nécessaire de s'assurer que les règles de contrôle que nous soumettons au contrôleur sont les plus pertinentes, c'est à dire celles qui le meneront le moins possible à une situation de blocage.

**Le contrôle flou est-il fiable ?** Imaginons que l'on confie à un contrôleur flou le pilotage d'une application critique - au hasard<sup>4</sup>, un avion. Quelle confiance peut-on porter à ce contrôleur ? Plus, autant, ou moins que celle que l'on a envers un pilote. Il est probable que l'on puisse avoir autant confiance en un contrôleur flou qu'un pilote humain, à formation égale. En effet, la qualité de la formation du pilote, et ses heures de vols (sans crash) sont l'assurance de sa fiabilité. Pourquoi ne pas mettre en parallèle la qualité de l'implémentation du contrôleur flou, ainsi que ses heures de vols (sur simulateur) ? Un pilote, humain ou logiciel, capable de se sortir de plusieurs situations prédéfinies, et disposant d'un nombre d'heures de vol suffisant, ne seraient-ils pas presque équivalents ?

---

<sup>3</sup>Disons plutôt : économes de notre temps, ce qui devient une vertu

<sup>4</sup>Au hasard, ou presque.