C++ / Évaluation « Essaim de robots »

Robotique/UE 3.1 – Décembre 2019

Supports de cours disponibles sur
`www.simon-rohou.fr/cours/c++`

A. Les lois de Reynolds

On considère un *essaim*[1] de $n = 20$ robots décrits par les équations d'état suivantes :

$$\mathcal{R}_i \begin{cases} \dot{x}_i &= \cos(\theta_i) \\ \dot{y}_i &= \sin(\theta_i) \\ \dot{\theta}_i &= u_i \end{cases} \tag{1}$$

Le vecteur d'état de chaque robot est $\mathbf{x}_i = (x_i, y_i, \theta_i)^\intercal$, avec θ_i son cap. Chaque robot $\mathcal{R}_i$ peut voir l'ensemble de l'essaim, sans être capable de communiquer avec les autres robots. On cherche à simuler un comportement d'essaim, comme illustré sur la Figure 1.

La simulation repose sur les trois lois de Reynolds :

— la répulsion ;
— la cohésion ;
— l'alignement.

On implémente pour chaque robot un contrôleur modélisant ces trois lois par une méthode de potentiels [1].

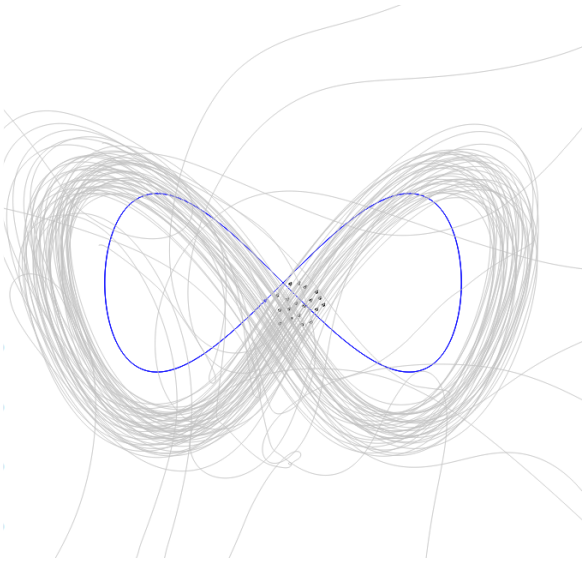

FIGURE 1 – Un essaim de 20 robots formant une courbe de Lissajous, dessinée en bleu.

B. La loi de contrôle (tirée de [1])

On définit le potentiel du robot $\mathcal{R}_i$ par :

$$V_i = \sum_{j \neq i} \left(\alpha ||\mathbf{p}(i) - \mathbf{p}(j)||^2 + \frac{\beta}{||\mathbf{p}(i) - \mathbf{p}(j)||} \right), \tag{2}$$

où $\mathbf{p}(i) = (x_i, y_i)$. La partie quadratique de la somme correspond à la cohésion. Le terme hyperbolique est lié à la répulsion. Les coefficients α et β détermineront l'intensité de l'une des lois dans le comportement de l'essaim. Par exemple, si β est élevé, la répulsion sera forte et l'essaim sera dispersé.

Le gradient de V_i est :

$$\frac{dV_i}{d\mathbf{p}(i)} = \sum_{j \neq i} \left(2\alpha\big(\mathbf{p}(i) - \mathbf{p}(j)\big) - \frac{\beta\big(\mathbf{p}(i) - \mathbf{p}(j)\big)}{||\mathbf{p}(i) - \mathbf{p}(j)||^3} \right). \tag{3}$$

On cherche à faire décroître V_i, il faut donc suivre la direction opposée au gradient. De plus, il nous reste à considérer la troisième loi de Reynolds, l'alignement : le robot $\mathcal{R}_i$ doit suivre le même cap que les autres éléments de l'essaim. On obtient la direction désirée exprimée par le vecteur suivant :

$$\mathbf{s} = \sum_{j \neq i} \left(-2\alpha\big(\mathbf{p}(i) - \mathbf{p}(j)\big) + \frac{\beta\big(\mathbf{p}(i) - \mathbf{p}(j)\big)}{||\mathbf{p}(i) - \mathbf{p}(j)||^3} + \gamma \cdot \begin{pmatrix} \cos\theta_j \\ \sin\theta_j \end{pmatrix} \right). \tag{4}$$

1. Essaim : rassemblement en nombre important, qui se déplace.

Lisez attentivement les questions suivantes, et implémentez-les dans l'ordre.

C. Commentaires

1. Toutes les fonctionnalités développées ci-après devront être commentées.

D. Les éléments du groupe

2. Récupérer le fichier `Robot.h` sur le site web du cours, et compléter les définitions des méthodes et attributs afin de restreindre les éléments en lecture et respecter la *const correctness*.
3. Ajouter dans le projet les fichiers `math.cpp` et `math.h`, à télécharger sur la page du cours. Ces fichiers n'auront pas besoin d'être complétés.
4. Implémenter le contenu de la classe `Robot`. Le constructeur
   `Robot(const std::vector<Robot>* v_swarm);`
   générera un robot à une position aléatoire dans une zone $[-200, 200] \times [-200, 200]$. S'aider des fonctions proposées dans le fichier `math.h` du projet.
5. Coder la méthode `f` implémentant les Équations d'état (1).

E. L'essaim

6. Dans le programme principal, ajouter une structure de données dynamique nommée `swarm` représentant l'essaim, et contenant déjà un robot.

F. La commande

7. On prendra les coefficients suivants :
   — $\alpha = 0.005$ (loi de cohésion)
   — $\beta = 5$ (loi de répulsion)
   — $\gamma = 0.5$ (loi d'alignement)
8. Ajouter la méthode `float Robot::u() const;` retournant la commande u_i du robot $\mathcal{R}_i$. La valeur de retour est donnée par :
   `return sawtooth(angle(sum) - m_theta); // final command`
   où `sum` est le vecteur de l'Équation (4), à définir dans le corps de la méthode `u()`.

G. Vue graphique

9. Initialiser la vue graphique avec

```
vibes::beginDrawing();
vibes::newFigure("Flocking");
vibes::setFigureProperties("Flocking",
 vibesParams("x", 100, "y", 100, "width", 700, "height", 700));
vibes::axisLimits(-100.,100.,-100.,100.);
```

10. Dans la boucle de simulation, on veillera à effacer la figure au début de chaque itération :

```
vibes::clearFigure("Flocking");
```

H. La simulation

11. La simulation s'étale sur 1500 unités de temps avec un pas de temps de 0.2. En fin de simulation, l'essaim sera constitué de $n = 20$ robots. Ajouter ces constantes dans le programme principal.
12. En tête de programme principal, initialiser la génération de nombres aléatoires avec une graine liée au temps `time(NULL)`.
13. Dans la boucle de simulation, ajouter une pause :

    ```
    usleep(1000.); // animation speed (microseconds)
    ```

14. En cours de simulation, tant que l'essaim n'a pas atteint sa taille maximale de vingt robots, ajouter un nouvel élément au groupe toutes les 10 unités de temps, et apparaissant à une position aléatoire.
15. Utiliser une méthode d'Euler pour intégrer les équations d'état de chaque robot.

I. La trajectoire des robots

16. Enrichir la classe `Robot` d'attributs gardant une trace des positions en x et y. Y ajouter les accesseurs associés.
17. Modifier la classe de façon à ce qu'un changement de position du robot soit systématiquement enregistré dans la trajectoire.
18. Dans le programme principal, afficher les trajectoires à la fin de la simulation à l'aide de `vibes::drawLine()`.

J. Influencer l'essaim

L'un des robots ($\mathcal{R}_{i=0}$) ne suit pas les lois de Reynolds, mais une courbe de Lissajous. Son comportement dissident va influencer le reste du groupe.

19. Dans la boucle de simulation, spécifier exactement l'état du robot en fonction du temps :

$$\begin{cases} x_0 &= 2a\sin(t/b) \\ y_0 &= a\sin(2t/b) \\ \theta_0 &= \operatorname{atan2}(2a\cos(2t/b)/b, 2a\cos(t/b)/b) \end{cases} \tag{5}$$

    avec $a = 30$ et $b = 80$.
20. Afficher en bleu le robot dissident, ainsi que sa trajectoire.
21. Visualiser son impact sur le reste du groupe.

Critères d'évaluation

— Développement des questions
— Organisation propre des sources (indentation, nom des variables, fonctions)
— Commentaires et lisibilité du code
— Respect de la *Const Correctness*
— Pas de fuite mémoire
— *Malus :* rendu de votre exécutable ou de fichiers temporaires ou compilés (`.o`, *etc.*)

Références

[1] Luc Jaulin. *Mobile robotics.* 2015. OCLC : 986557752.