Implémentation et Déploiement d'une famille d'algorithmes de Navigation pour les robots

Selma Kchir,Tewfik Ziadi,Mikal Ziane

5 novembre 2013

1 Introduction

Le but de ce projet est d'implémenter plusieurs variantes de la famille d'algorithmes de navigation (Bug [3]) et de refactorer le code pour l'intégrer à une ligne de produits. Le travail demandé consiste à implémenter ces algorithmes en C++ (en s'appuyant sur le framework OROCOS [1]. La démarche de développement sera une démarche agile et autant que possible dirigée par les tests. Le code produit devra être testé sous le robot Wifibot disponible au LIP6 et le simulateurs ROS Stage [2].

1.1 Contexte

Avant de présenter le travail demandé, voici quelques concepts importants en robotique :

- Une mission est un ensemble d'actions qui permettent d'atteindre un objectif bien déterminé.
- Chaque robot possède un ensemble de capteurs et d'effecteurs :
  - Les capteurs permettent au robot de percevoir l'état de son environnement à partir de sa position courante.
  - Les effecteurs permettent au robot d'agir et de se déplacer dans son environnement pour se rapprocher de l'objectif de sa mission.

Les robots mobiles autonomes sont capables d'effectuer plusieurs missions dont la navigation. Cette dernière permet au robot d'explorer son environnement tout en évitant les obstacles sur son chemin (en se basant sur les informations retournées par ses capteurs).
Dans ce projet, nous nous intéressons particulièrement aux algorithmes de navigation Bug[3]. Ces derniers forment une famille d'algorithmes partageant certaines propriétés et hypothèses concernant l'environnement et l'objectif du robot. Cependant, chaque algorithme définit une stratégie différente d'évitement d'obstacles. L'algorithme initial de cette famille était l'algorithme Bug1 [2] [3]. Depuis lors, plusieurs variantes ont été développées afin d'améliorer ce dernier.

1. `http://www.orocos.org/`
2. `http://www.ros.org/wiki/stage`

2 Spécification

En se basant sur la démarche décrite dans l'article [1] et sur 3 variantes des algorithmes de Bug qui vous seront fournies, il vous est demandé dans un premier temps de :

- Réutiliser l'algorithme générique abstrait Bug (voir algo 1) de l'article ainsi que les fonctionnalités de base qui vous seront fournies afin d'implémenter en OROCOS au moins 8 variantes de la famille Bug.
- Tester les algorithmes générés en simulation avec ROS Stage et sur le robot Wifibot.
- Implémenter la solution proposée dans l'article pour la gestion de la variabilité.

Tout au long de ce projet, il aura un travail important d'apprentissage pour maitriser OROCOC et aussi maitriser l'installation et la configuration de robot Wifibot.

Le travail à réaliser concerne donc trois lots principaux :

Lot 1 : Implémentation en OROCOS et test en simulation d'au moins 8 variantes de la famille d'algorithmes Bug.

Lot 2 : Implémentation d'une solution pour la gestion de la variabilité.

Lot 3 : Test du code des produits générés en simulation avec ROS Stage et le robot WifiBot

2.1 Minimum demandé et bonus

Pour obtenir la note de 15/20 il s'agira de réaliser les 3 lots principaux et évaluer le générateur sur une mission qui pourra être celle présentée dans ce document. Un bonus sera donné si plus de cinq variantes d'algorithmes sont implémentées. Un autre bonus sera donné en fonction de la qualité de tests unitaires définis étant donné qu'il n'est pas facile de tester automatiquement l'ensemble des éléments d'une mission robotique.

3 Démarche itérative dirigée par les tests

Le projet suivra une démarche itérative et autant que possible dirigée par les tests[3]. Il ne s'agit donc pas de faire complètement tel lot avant de passer au suivant mais au contraire de développer rapidement une version du compilateur qui permettent de voir une mission très simple tourner sur le simulateur. Il s'agira alors d'ajouter peu a peu des fonctionnalités.

3. `http://www.agiledata.org/essays/tdd.html`

Algorithm 1: Bug generic algorithm

Sensors : A perfect localization method.
An obstacle detection sensor
input : Position of Start (q_{start}), Position of Target (q_{goal})
Initialisation: robotPos ← `getPosition()` ; direction ← `getDirection()` ;

```
if goalReached(robotPos) then
    EXIT_SUCCESS;
end
else if obstacleInFrontOfTheRobot() == true then
    identifyLeavePoint (direction, robotPos, goalPos);
    if leavePointFound() && researchComplete(robotPos,
    getHitPoint(), goalPosition) then
        goToLeavePoint(getLeavePoint());
        faceGoal()();
    end
    else if completeCycleAroundObstacle(robotPos, getHitPoint())
    && !leavePointFound() then
        EXIT_FAILURE;
    end
end
else
    motionToGoal();
end
```

4 Les outils

Comme nous l'avons signalé plus haut, il est demandé d'implémenter les algorithmes de Bug en OROCOS. OROCOS [4] est un framework C++ pour le développement des applications robotiques. Il permet de spécifier une application robotique sous forme d'un ensemble de composants C++. OROCOS utilise ROS (Robot Operating System - http ://www.ros.org/wiki/) comme un moyen de communication entre ces composants. Les algorithmes implémentés doivent être déployés sur simulateurs ROS Stage et aussi sur le robot Wifibot.

4. `http://www.orocos.org/`

Références

[1] Selma Kchir, Tewfik Ziadi, Mikal Ziane, and Serge Stinckwich. A Top-Down Approach to Managing Variability in Robotics Algorithms. In *International Workshop on Domain-Specific Languages and models for ROBotic systems – DSLRob*, page To be published, November 2013.

[2] V. Lumelsky and A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *Automatic Control, IEEE Transactions on*, 31(11) :1058 – 1063, nov 1986.

[3] Vladimir J. Lumelsky and Alexander A. Stepanov. Effect of uncertainty on continuous path planning for an autonomous vehicle. In *Decision and Control, 1984. The 23rd IEEE Conference on*, volume 23, pages 1616 –1621, dec. 1984.