Contrôleurs dans l'espace

Julien Bouvet, Antoine Chenon, Mikaïl Demirdelen, Hoel Kervadec
Encadrants : Yann Ricquebourg, Loic Helouet

Juin 2014

Première partie

Avant-propos

0.1 Un simulateur en temps réel

Orbiter est un simulateur de vol spatial, créé par Dr. Martin Schweiger en 2000 et ce afin de combler le manque de simulateur de vol réaliste disponible sur internet. Gratuit mais non libre, un kit de développement a fait son apparition pour permettre à tout un chacun de développer son propre vaisseau. Fourni avec un lot de scénarios et de vaisseaux, Orbiter permet d'expérimenter et de mieux appréhender les lois de la physique.

C'est dans cette optique que nous proposons une première démarche d'automatisations des vaisseaux dans Orbiter.

Dans un premier temps ce projet est compatible avec un seul vaisseau : le Shuttle A.

0.2 Module lunaire

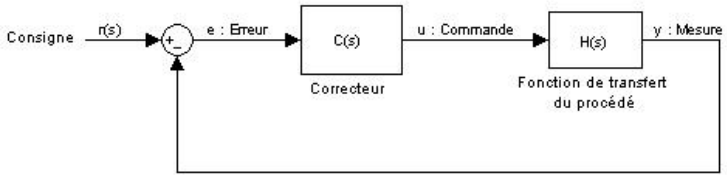
Le Shuttle A est un module lunaire : en effet il est conçu pour permettre aux spationautes de rentrer sur Terre une fois l'alunissage terminé. Ces caractéristiques sont donc plus adaptées à cet environnement. En effet le décollage depuis la Terre est plus diffcile (compte tenu de la gravité, plus forte que sur la Lune) et plus gourmand en carburant. Cependant le Shuttle A a deux avantages :

- Il dispose de nombreux propulseurs, ce qui en facilite la manoeuvre
- Il est codé en une seule classe C++ :, c'est transparent pour l'utilisateur, mais dans le cas de notre travail, cela simplifier grandement l'étude.

0.3 Théorie du contrôle

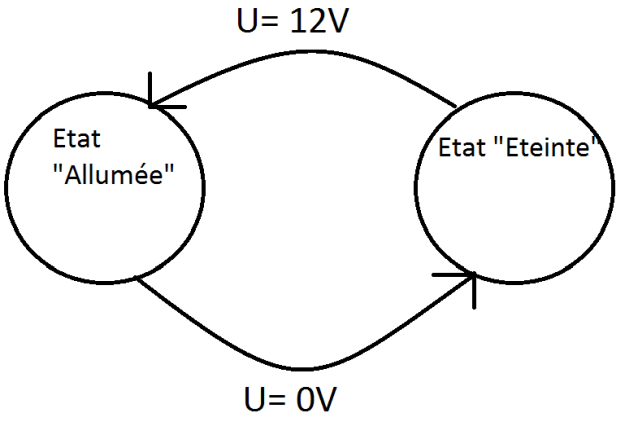
Pour pouvoir automatiser le pilotage des vaisseau, en particulier du ShuttleA nous disposons de plusieurs outils formels, dont la théorie du contrôle.

C'est un moyen de formaliser le fait qu'un système, pour se maintenir sur une trajectoire prend des mesures, les compare à des références et corrige si nécessaire. C'est une boucle qui va des capteurs vers le contrôleur, chargé de prendre les décisions pour suivre l'objectif.



0.4 Automates

Les automates sont un ensemble d'états et de transitions, qui correspondent à des états d'un système. Par exemple on peut imaginer qu'une lampe soit modélisée par un automate. Elle aurait deux états : Allumée, éteinte. Cette lampe peut passer d'un état à l'autre grâce à des transitions. Ces transitions définissent sous quelles conditions le système change d'état. Pour la lampe, la transition de l'état éteinte vers allumée se fait par exemple si elle est alimentée et vice-versa.



Pour le vaisseau on peut imaginer une infinité d'états différents, ce qui permet de contrôler le vaisseau à notre guise. Par exemple l'état "'Propulsion MAXI"' avec une transition sur l'état "'Propulsion Gauche seule"' en cas de détection d'obstacle.

Deuxième partie

Langage Mittlewerk

0.5 Introduction

Nous avons créé un langage simple permettant de piloter le vaisseau sans maîtriser le C++. En effet, ce langage permet de créer facilement des automates, et donc de créer un contrôleur personnalisé. Certains aspects du langage sont proches du C++ mais ce sont des concepts classiques en programmation (affectation, déclaration ...).

0.6 Instructions basiques

Toutes les instructions se terminent par un ';' pour indiquer au compilateur qu'une instruction est complète.

0.6.1 Variables

Une variable a un nom et un type, il faut donc les déclarer au compilateur avant de pouvoir les utiliser. Cela se fait très simplement :

```
type nom;
```

Par exemple, si vous souhaitez définir un double (nombre à virgule à valeur max très élevée) que vous appelez toto, il vous faudra avant de l'avoir utilisé l'avoir défini de cette façon :

```
DOUBLE toto;
```

Pour affecter une valeur, la démarche est identique à la plupart des langages :

```
toto = 42;
toto = toto + 1;
toto = max(1, toto);
```

On peut affecter à une variable la valeur de retour d'une fonction, d'une autre variable ... Il faut juste vérifier que le type que l'on donne à la variable correspond à celui qu'on lui a donné lors de sa déclaration.

0.6.2 Fonctions

Les fonctions peuvent être déclarées, hors du corps d'une fonction, et être appelées, dans le corps d'une fonction ou dans un etat de l'Automate.

Pour déclarer une fonction c'est très simple :

```
type nomDeFonction(type1 Param1, type2 Param2 ...) { corps de la
                                fonction }
```

Si la fonction a un type différent de VOID, elle doit retourner une valeur :

```
RETURN toto;
```

Pour l'appeler : `nomDeFonction(param1, param2, ...)`

0.6.3 Commentaires

Il est parfois très utile de commenter son code : ce qui est commenté n'est pas interpreté lors de la compilation, ce ne sont que des informations pour les humains. Pour encadrer du commentaire il faut mettre /* avant le commentaire, */ à la fin.

```
VOID fonctionInutile() { /* cette fonction ne fait rien*/ }
```

0.6.3.1 Exemple de synthèse :

```
DOUBLE megaBadAss(DOUBLE toto) {
 goToSchool(toto); /* Fonction fictive, rassurez vous ;) */
 RETURN 42;
}
```