

INF6600 Rapport TP4

Améliorations au système de contrôle d'un drone fermier

Daniel Lussier-Lévesque et Ian Gagnon

Abstract—L'implémentation d'un système de contrôle de drone fermier a été réalisé progressivement tout au long de la session en commençant par un module TrueTime avec un système continu déjà implémenté en Simulink. Par la suite, l'implémentation a continué sur une machine virtuelle QNX roulant sous Vmware, avec certaines fonctions déjà fournies. Ce rapport focus sur ce qui a été fait sur cette base de code pour améliorer les aspects les plus importants. La première étape a été d'implémenter un système de log qui nous permettent de voir comment se comporte l'ordonnancement des tâches et la performance du système. Par la suite, malgré le manque de droits d'accès qui nous permettraient de comprendre l'ensemble des problèmes de performances, nous avons toutefois été capable de les mesurer et de les documenter. Un système graphique vivant en dehors de la machine virtuelle nous permettant de valider le comportement du système a aussi été implémenté.

I. INTRODUCTION

LE système de contrôle considéré est celui d'un drone autonome capable de se déplacer sur trois axes et ayant une caméra fixe pour prendre en photo l'ensemble d'un champs. Le drone doit naviguer les champs et prendre des photos jusqu'à ce que la mémoire soit pleine, puis transmettre les photos à une station base. Lorsque la batterie est presque vide (10%), le drone doit retourner à la station base pour être rechargé.

Le système de contrôle implémente le contrôle de la navigation (où le drone va), le contrôle de la batterie (est-il temps d'aller recharger la batterie?) et le contrôle mission (quelle séquence d'étapes doit être exécutés pour opérer avec succès?).

Le système de contrôle de drone sur lequel nous avons apportés nos modifications est un système simple où la communication entre le système continu et discret est effectué par des variables atomiques et par l'enregistrement de *callbacks*. La communication en dehors du système s'effectue par des messages de log sur la console de debug où à intervalle régulier (chaque seconde) l'état du drone est envoyé. Toutes les tâches sont asynchrones les unes par rapport aux autres et roulent sur leur propre fil d'exécution. Nous avons choisi de mettre à jour le système de contrôle du drone toutes les 100 ms.

De plus, l'implémentation du système de contrôle doit être accompagné d'une plateforme de simulation d'un environnement dans lequel opérer, qui comprend le temps de réponse de la caméra, les variations d'orientation et de vitesses, le temps de transmission de photo à la station base et la charge et décharge de la batterie. Le système continu est mis à jour toutes les 20ms de sorte qu'il soit significativement plus rapide

que le système de contrôle. De cette manière, le système discret ne voit pas énormément de différence dépendamment s'il se fait juste avant ou juste après une mise-à-jour du système continu.

L'ensemble des photos prises peut être visualisé à travers une carte du monde qui est mis à jour à chaque transmission et montre quelle région a été prise en photo. La carte doit être accédée en allant chercher les fichiers sur le système de fichier de la machine virtuelle.

La politique d'ordonnancement utilisée est un systèmes de priorités fixes et un ordonnancement FIFO pour chaque niveau de priorité. Le choix a été fait de placer la priorité du contrôle caméra avant celle du contrôleur de navigation parce qu'il est plus facile de compenser pour une échéance manquée dans le cas de la navigation (une photo manquée implique de défaire les dernières mise-à-jour de navigation pour reprendre la photo).

La machine virtuelle roule sous QNX 7.0 sous VMware Workstation avec un hôte Windows 10. La machine hôte à un processeur d'Intel à quatre coeurs, le i7 3770, avec 16 GB de mémoire DDR3 et utilise un Seagate ST1000DM003 comme disque dur.

II. LACUNES DU SYSTÈME DE BASE

Le système décrit plus haut a plusieurs lacunes importantes. Nous allons ici nous focaliser sur les trois principales faiblesses qui nous ont empêcher de réaliser correctement plusieurs améliorations auxquels nous avons pensés. Il va s'en dire que nous avons penser plus important de régler ces lacunes au meilleur de nos capacités en priorité.

A. Peu de visibilité sur l'état du système

Le principal problème est qu'avec le système de base, nous ne savons pas si le système est *correct*, c'est à dire que le système effectue les actions que l'on s'attend à ce qu'il fasse. Les messages de logs qui sont envoyés sont répétitifs et il est facile de manquer un comportement anormal dans les 1800 messages que comportent une séquence d'exécution. De plus, il est impossible de savoir ce qui se passe à l'intérieur de cette période d'une seconde. une période plus courte n'est toutefois pas envisageable: il serait impossible d'analyser l'entièreté du message en temps réel et donc encore plus facile de manquer une information inattendue. Il n'est pas non plus envisageable d'analyser les données à la fin de la simulation: la console de debug ne peut contenir qu'un nombre limité de messages.

En conséquence, il n'est pas possible de qualifier le système d'aucune manière: un problème de comportement du système demanderait une correction importante au code qui invaliderait l'ensemble des mesures effectués. Comme une séance de simulation dure 30 minutes, il serait difficile d'arriver à un ensemble de résultat suffisant dans un temps raisonnables si nos mesures devaient être invalidés. Sans mesures fiables, il est impossible de déterminer si une tentative d'amélioration du système améliore réellement le système. Pour cette raison, cette lacune est considérée comme la plus importante du système implémenté.

B. Aucune visibilité sur la qualité de l'ordonnancement

C. Pics de latence

III. AMÉLIORATIONS APPORTÉES

IV. RÉSULTATS

V. CONCLUSION

The conclusion goes here.

REFERENCES

- [1] H. Kopka and P. W. Daly, *A Guide to L^AT_EX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.