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 se concentre 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 avions 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'ont 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

En assumant que le comportement du système est correct, nous ne pouvons toujours pas penser à améliorer les performances du système ou rajouter de nouvelles fonctions utiles parce que nous n'avons aucune métrique de performance de disponible. Nous ne savons pas si les tâches manquent leur échéance de manière régulière et arrivent tout juste à rester fonctionnels, ou au contraire si il y a largement de ressources disponibles pour de nouvelles tâches.

Dans le cas où le comportement du système n'est pas celui attendu, les métriques de qualité d'exécution nous sont aussi utile pour comprendre les problèmes. Si une tâche est en famine parce que les tâches plus prioritaires prennent toutes les ressources, ce sera facilement visible et évitera une longue séance de debug en aveugle. Nous considérons cette lacune comme fondamentale et bloquante pour des améliorations à d'autres aspect du système.

C. Pics de latence

Après avoir apportés les modifications qui seront détaillés à la section III, nous avons réalisés que le système tel que simulé avait encore une lacune qui bloquait des améliorations que nous avions planifiés: le système est affecté par des pics de latences non-déterministes. Ces pics de latences affectent toutes les tâches en même temps. Plus spécifiquement, ils affectent des tâches qui n'acquirent pas de mutex où autre verrou global et communiquent uniquement par des variables atomiques. On peut donc conclure que ces pics arrivent au niveau système et non au niveau de l'ordonnancement ou au niveau de la communication.

En plus d'affecter les performances du systèmes et de le rendre moins fiable, un impact important de ces pics de latences est que toutes les métriques de pire temps d'exécution ou de déviations standard sont affectés de manière aléatoire et qu'une comparaison avant et après d'une tentative d'amélioration est noyée dans le bruit de ces pics.

Les pics des latences durent plusieurs millisecondes. Plusieurs tentatives de contourner le problème ont été essayé: réduire le nombre de cœurs du système de contrôle, augmenter le nombre de cœurs du système et déplacer la machine virtuelle du disque réseau au disque dur local. Déplacer la machine virtuelle à permis de réduire significativement la durée de ces pics, mais aucune des ces tentatives n'a permis de ramener les variations de latences à des temps raisonnables.

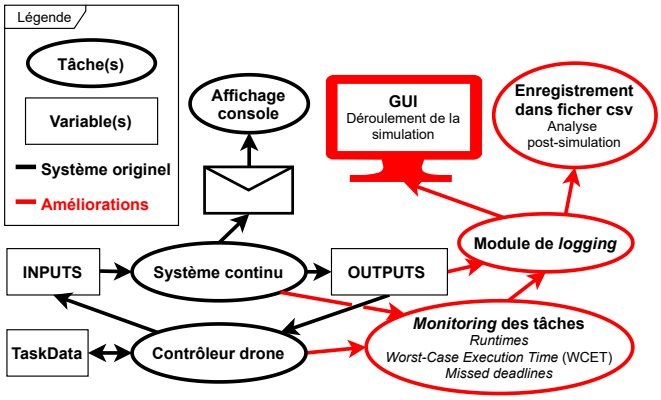

Fig. 1: Diagramme de tâches du système

III. Améliorations apportées

Comme il est possible de le voir à la figure 2, les améliorations ont été apportés au niveau du *logging* et du *monitoring* des tâches pour avoir des sorties de simulations faciles à interpréter et qui corrigent les deux premières lacunes décrites à la section II, le manque de lisibilité de ce qui se passe dans le système.

A. Monitoring des tâches

Pour chaque tâche qui s'exécute, une mesure du temps d'une exécution est faite à l'aide de `std::chrono::steady_clock::now()`. Deux variables (atomiques) sauvegarde la valeur de la dernière exécution et le pire cas. Dans le cas de notre système, nous n'avons pas d'horloge permettant une granularité plus grande qu'une milliseconde.

B. Logging

Une fois que nous avons implanté le code pour avoir l'information dans des variables, il faut penser à un moyen de la communiquer. Il a donc fallu implémenter une nouvelle fonction (avec le code de *monitoring* associé) pour lire l'ensemble des variable et les pousser vers les nouvelles sorties. Nous avons décider d'exécuter la tâche toutes les 100 millisecondes afin de ne pas surcharger le système. Cette fonction est appelée par la tâche de *monitoring* déjà existante qui auparavant ne poussait les donnée que vers la console une fois toute les secondes.

C. Sortie en fichier csv

La sortie qui permet de faire une analyse compréhensive du système est un simple fichier csv qui est sauvegardé en mémoire non-volatile au fur et à mesure que la simulation avance (de sorte qu'un *crash* peut être investigué en s'aidant du fichier). Le format csv étant supporté par pratiquement tous les outils d'analyse et de base de données, il est facile de s'en servir pour obtenir des statisques sur un grand nombre d'exécutions, de réaliser des analyses et des graphiques en Python, en R, en MATLAB ou tout simplement dans Microsoft Excel ou LibreOffice Calc.

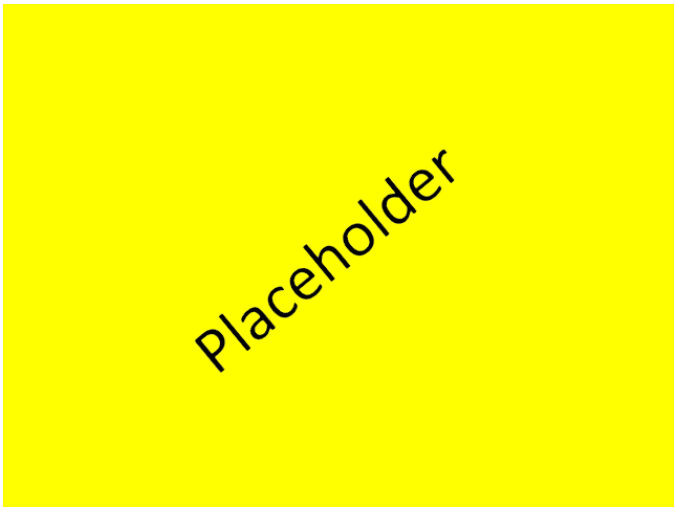

Fig. 2: Le gui en action

D. GUI

#TODO

IV. Résultats

V. Conclusion

The conclusion goes here.

References

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