Université de Cergy-Pontoise,
UFR Sciences & Techniques,
Département des sciences informatiques,
Master 1 I&ISC

Informatique Embarquée

Projet - Détéction de ligne en vue d'une conduite autonome - Covid19 Edition

Paul Valentin

20 mai 2020

Certification - Temps réel Kesako ?

Temps Réel

Un système dit temps réel est assujetti à une contrainte temporelle. Cette contrainte peut être :

1. Temps réel critique ou dur : Dans ce cas le non respect d'une échéance le système se voit fortement impacter voir détruit.
   Exemple : Système de contrôle d'une fusée, si le système ne prend pas en compte une variable à temps la fusée peut s'auto-détruire.
2. Temps réel mou : Dans le cas du non respect d'une échéance, aucune conséquence sur le système à part la perte des données en cours.
   Exemple : Système de vidéo-projection. Si une échéance est loupé (corruption des données) alors la frame est juste perdue. Système de drivers souris, loupé quelques échéances (ie : ne pas déplacer le curseur alors que la souris se déplace) n'aura pas beaucoup d'impact sur l'utilisation tant que ceux-ci reste rare.

Dans notre cas, uniquement le temps réel mou nous intéressera.

Certification

Pour chaque niveau de temps réel différents niveaux de certification peuvent être mis en place. En partant du temps réel dur, le système doit être capable de gérer tous les imprévus qu'ils soient hardware (variation possible de température qui peut venir gêner la bonne cadence du processeur, ...) ou software (système stressé, ...), afin que toutes les échéances soient respectées. Une mesure des performances en fonction des circonstances doit être effectué dans un panel de cas qui est exhaustif.

Pour le temps réel mou les certifications devront être adaptées à notre cas d'utilisation, et une moins grande exhaustivité des tests est demandée, le système devra réagir normalement dans la plupart des cas dans un contexte normal d'utilisation. Si une échéance est loupée le système n'étant pas mis en péril.

Exemple de Certification possible pour le cas d'une fusée

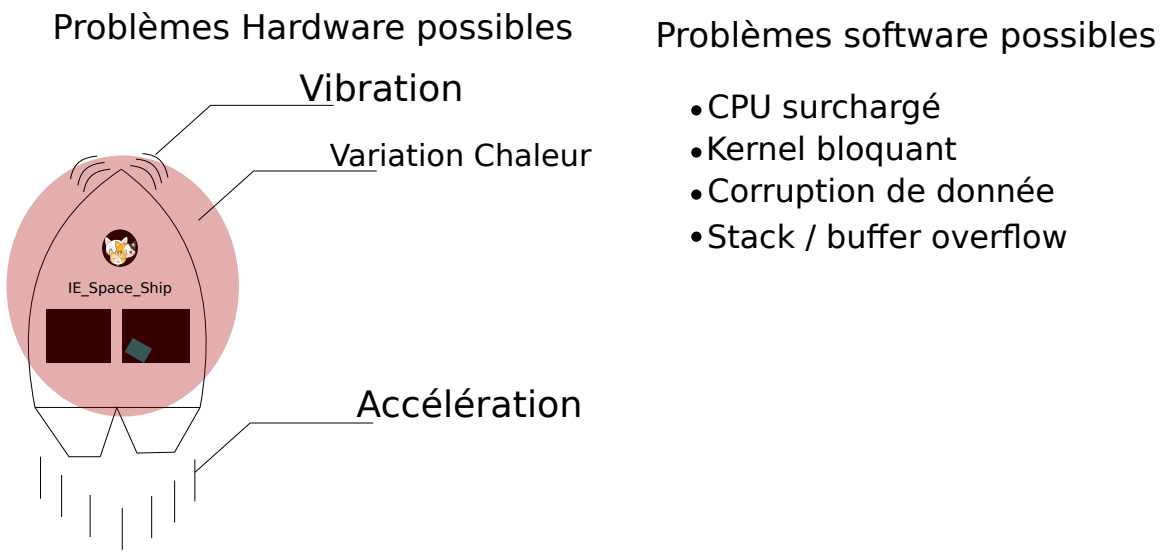

FIGURE 1 – Différents problèmes que peut rencontrer une fusée, lors de son décollage, les certifications doivent prendre en compte tous ces problèmes afin que le vol se passe dans les meilleurs conditions.

Dans ce cas nous pouvons imaginer plusieurs certifications :

1. Performances mini/maxi en fonction de la variance de température
2. Performances en fonction du stress du système
3. Le système supporte-t-il bien les accélérations ? Les vibrations ?
4. Le kernel peut-il bloquer des tâches critiques ?

Bien entendu cet exemple n'est pas réel, une fusée embarquerait des systèmes dédiés à la tâche qu'il doivent exécuter afin de limiter à la base les problèmes de kernel bloquant et autres problèmes de concurrences, d'accès mémoire, etc. D'autres solutions existent pour remédier à ces problèmes comme l'utilisation de kernel temps réel.

1 Introduction

On souhaite faire une voiture autonome simple. Vous êtes chargés de réaliser le système de traitement d'image de cette voiture. Ce système prend l'image de la caméra en entrée et donne l'angle avec lequel tourner en sortie. Les ingénieurs du système de contrôle ont besoin de connaître la vitesse à laquelle tourne votre système. Vous allez donc devoir certifier la fréquence de fonctionnement de votre voiture. Une meilleure fréquence de fonctionnement permettant à votre voiture d'aller plus vite, il vous faut également étudier quelles modifications apporter pour augmenter cette fréquence sans compromettre l'aspect temps-réel. La stabilité de cette fréquence de fonctionnement sera également à étudier. Utiliser l'ordonnanceur précédemment effectué peut être une bonne idée.

IMPORTANT

Merci de documentez tout choix, protocole de test, performance lors de ce projet par exemple :

— Avec une version A de l'algorithme quelles sont les performances ? Quel est le benchmark ?
— J'ai upgrade à une version B quel gain / perte ai-je obtenu ?

De plus soyez cohérent : garder les même unités de tests entre vos différentes versions (Hz / Fps), ainsi que les mêmes tables de tests, si vous mesurez le nombre de FPS à un endroit donné pour une version, gardez le plus de cohérence possible sur le test pour une version ultérieure.

2 Traitement d'image

Afin que le robot puisse naviguer de manière totalement autonome, il doit reconnaître le chemin qui sera indiqué par des lignes au sol.

Pour ce faire nous allons utiliser la transformée de Hough codé dans les précédents TPs afin de détecter les points de fuites formés par les lignes au sol.

En plus des tests liés aux performances, testez le système lorsque celui-ci est stressé, et non stressé ?

Rappel : La transformée de Hough permet de détecter les lignes orientées d'un certain angle en passant un filtre de détection de contours, puis en passant de l'espace cartésien à l'espace log polaire et enfin en associant, à chaque point de ce nouvel espace, une densité qui correspond au nombre de courbes qui y passe.

3 Optimisation du traitement d'image

Maintenant que nous avons la transformée de Hough prête, il faut que celle-ci puisse tourner assez rapidement. Plusieurs options d'optimisation s'offrent à vous :

- Threader le traitement d'image, Paralléliser les convolutions : chaque thread peut traiter une partie de l'image.
- Optimiser l'algorithme de la transformée de Hough : peut-on utiliser une convolution réduite pour économiser du temps de calcul ? Peut-on convoluer sur une partie de l'image seulement ? Doit-on binariser l'image ? Combien d'angle doit-on prendre en compte ?
- *Recompiler entièrement le noyau Linux afin de pouvoir préempter tous les autres processus (Attention cette option vous prendra certainement beaucoup plus de temps que ce que l'on accorde lors des séances d'IE, plein de bugs peuvent apparaîtrent, etc.).*

4 Rendu

Compte rendu sous la forme d'une note de 4 pages maximum. Les 4 pages doivent être expliquées, justifiées et illustrées par des figures, captures d'écrans, courbes de résultats. Si nécessaire, des annexes (code, photos, figures) peuvent aussi être ajoutées aux 4 pages.

Ces 4 pages doivent contenir :

- une description de l'architecture logicielle (et matérielle si appropriée)
- une certification du fonctionnement de votre application :
  - le protocole simple utilisé
  - la mise en place d'une métrique

  - les résultats obtenus
- une analyse de vos résultats, et les perspectives d'amélioration
- si vous avez travaillé en groupe, la répartition des taches.

On s'intéresse à la démarche, pas au résultat.
Si vous y passez plus de 6h, il y a un problème.

5 Compétences attendues

1. Écriture d'un protocole expérimental : élaboration de certifications
2. Connaissance des différents problèmes que peuvent rencontrer les systèmes embarqués.
3. Analyse des différentes problématiques de temps-réel : mise en place de métriques

6 Base fournie

Installez opencv-dev

1. Linux : `sudo apt install libopencv-dev`
2. Max OS : `brew install opencv` Si vous n'avez pas brew : `brew.sh`
3. Windows : prennez le tutoriel en fonction de votre IDE dans la documentation opencv : `https://docs.opencv.org/master/df/d65/tutorial_table_of_content_introduction.html`

La dernière version d'opencv introduit des changements de noms de constante. Un fichier patch vous est fourni. Sous linux vous pouvez l'appliquer avec : `patch < update_ubuntu_2004.patch`. Sinon il est lisible depuis votre éditeur texte.