Test de la navigation de systèmes autonomes dans des mondes virtuels

Thierry Sotiropoulos, Jérémie Guiochet, Félix Ingrand, Hélène Waeselynck

Test de la navigation de systèmes autonomes dans des mondes virtuels

Thierry Sotiropoulos, Jérémie Guiochet, Félix Ingrand and Hélène Waeselynck
CNRS, LAAS - CNRS, 7 av. du colonel Roche, F-31400, Toulouse, France
Université de Toulouse, France
Email : prénom.nom@laas.fr

***Résumé*—Un système autonome doit accomplir des tâches sans intervention humaine dans des environnements extrêmement variés. Les fautes de tels systèmes peuvent avoir des conséquences inacceptables vis à vis de la sûreté de fonctionnement, des performances ou de la mission. Le test en simulation de tels systèmes est efficace pour révéler et éliminer ces fautes tout en présentant l'avantage d'être moins coûteux, moins dangereux et potentiellement plus exhaustif qu'un test dans le monde réel. L'objectif de nos travaux est de trouver comment caractériser le domaine d'entrée du test, d'identifier les critères de sélection à retenir pour les tests, d'élucider comment satisfaire ces critères en combinant des procédés de génération de tests classiques et des procédés de génération de mondes et ainsi, de proposer une approche permettant de définir et de réaliser les tests dans le contexte des systèmes autonomes. Pour cela, nous avons tout d'abord identifié divers composants qui interviennent durant les phases de tests que sont la génération des entrées, l'exécution du test, ainsi que l'analyse des résultats de l'exécution. Nous utilisons le simulateur *Morse* qui est générique et basé sur le moteur de rendu 3D *Blender Game engine* et qui permet de simuler un ensemble de capteurs et d'actionneurs. Le système sous test comprend les modules robotiques nécessaires au service de navigation, générés à partir de modèles *GenoM*. Nous avons mené une première expérience dans le but d'observer l'indéterminisme des modules dédiés à la navigation.**

I. Introduction

Les systèmes autonomes mobiles développés aujourd'hui ont pour but d'accomplir des tâches sans supervision et sans intervention humaine dans des environnements variés. Le déploiement de tels systèmes dans des espaces non structurés et parfois partagés avec l'homme, pose de fortes contraintes vis-à-vis de la sûreté de fonctionnement, des performances ou de la réalisation de la mission. En effet les défaillances de tels systèmes peuvent avoir des conséquences inacceptables du point de vue de ces critères. Le test de tels systèmes est efficace pour révéler et éliminer ces fautes, mais dans le contexte des systèmes autonomes, il pose plusieurs problématiques. Tout d'abord, les incertitudes liées à l'environnement et aux algorithmes de perception, conjuguées aux possibilités de contexte d'exécution, font que le domaine d'entrée des tests est infini (par ex. le terrain, les conditions de visibilité, les obstacles, etc.). L'indéterminisme du comportement de ces systèmes font qu'il est extrêmement complexe de mettre en place des oracles de tests performants. Et enfin, l'exécution des tests dans le monde réel est longue, coûteuse et peut mener à des conséquences inacceptables (destruction du matériel). Une direction possible est de développer le test en simulation car il présente l'avantage d'être moins coûteux et potentiellement plus exhaustif qu'un test dans le monde réel. Bien que les simulateurs actuels ont atteint une maturité suffisante pour permettre de réaliser ces tests, il n'existe pas encore à notre connaissance d'approche systématique permettant de caractériser le domaine d'entrée, de guider la sélection des tests dans ce domaine d'entrée et de les implémenter automatiquement sur des plate-formes de simulation.

L'objectif est donc de proposer une telle approche permettant de définir et de réaliser des tests dans le contexte des systèmes autonomes. Nous prendrons comme cas d'étude les services de navigation d'un robot mobile, en utilisant les techniques de génération procédurale de monde.

La section II présente le contexte de l'étude, puis la section III détaille l'approche générale que nous souhaitons développer. La section IV présente les premiers résultats de la mise en place expérimentale de notre approche. Enfin, nous concluons sur les prochaines étapes de ce travail dans la section V.

II. Contexte

A. Navigation

La tâche de navigation concerne la perception de l'environnement, et la décision des mouvements d'un système autonome, que nous dissocions de la locomotion qui concerne l'exécution du mouvement. La manière dont le système autonome va planifier sa trajectoire sera liée à la manière dont il perçoit le monde par ses capteurs mais aussi par la nature de la représentation qu'il a de son environnement. En effet un robot peut représenter son environnement sous la forme d'une carte d'occupation, d'une carte d'élévation, d'un modèle qualitatif de l'espace libre ou encore d'une représentation de la topologie de l'environnement [1]. De nombreux travaux, notamment au LAAS-CNRS, traitent de ce domaine, et parmi eux nous utiliserons l'algorithme présenté dans [2] et intégré dans un module nommé *P3D*. Le principe général est de choisir le chemin qui optimise un critère *intérêt/coût* calculé le long d'arcs générés à partir de la position du robot dans son modèle numérique du terrain. Le calcul de ce critère se fait le long de chaque arc à différents points répartis régulièrement sur celui-ci, on calcule alors le coût en appelant une fonction de placement de la structure 3D du robot, ce qui permet d'évaluer la valeur du risque lié à la configuration et l'attitude qu'aurait le robot s'il était positionné à cet endroit. Ensuite *P3D* calcule le risque lié à l'enchaînement de ces diverses attitudes successives et ainsi donne la valeur du critère *intérêt/coût* de l'arc, l'intérêt dépendant du rapprochement vis à vis du but.

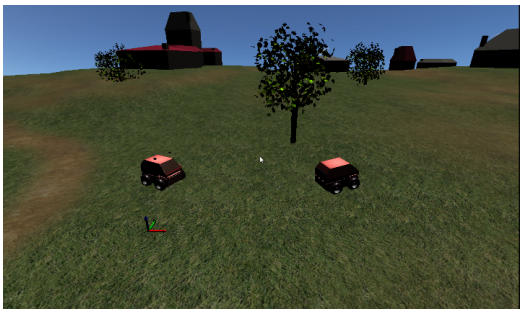

FIGURE 1: Exemple de vue 3D avec MORSE

Il est important de noter que ce type d'algorithme est sensible à des variations dans la perception mais aussi dans les ressources possibles pour l'exécution des tâches du contrôleur. Ainsi, pour une même configuration, deux exécutions pourront être différents. Nous mentionneront cette caractéristique comme l'indéterminisme de la navigation.

B. Simulation en robotique

Les simulateurs sont utilisés en robotique pour la conception ou lors de la phase de validation ou de tests. Pour la conception, la simulation permet un prototypage rapide de robot grâce à des ensembles d'actionneurs et de capteurs déjà implémentés, et la réalisation de mouvements. D'autres simulateurs permettent également de pratiquer des tests dans des environnements complets, où sont intégrés la gravité, les frottements, les forces externes, etc. Cela permet de réaliser ces simulations sans avoir à utiliser le robot réel et donc en évitant de l'endommager, de causer des dégâts ou bien des blessures le tout sans avoir besoin d'espace ou de matériel d'installation. Dans [3], une vision globale de ces catégories de simulateur est présentée, notamment en les regroupant selon les deux catégories précédentes. Le simulateur MORSE (*Modular OpenRobots Simulation Engine*) [3] développé au LAAS-CNRS, basé sur l'outil 3D Blender, permet de simuler le robot dans son monde (voir Figure 1), et de lancer les simulations avec le contrôleur de robot utilisant ses propres modules.

Un simulateur peut également permettre d'utiliser les techniques de génération procédurales de monde comme dans [4]. Ces techniques présentent un double avantage : d'une part elles permettent de réduire encore plus les coûts, d'autre part elles permettent d'insérer une dimension aléatoire dans le contenu ainsi obtenu.

C. Test en robotique

Le test peut être définit comme une « vérification dynamique effectuée avec des entrées valuées » [5]. Le système testé étant noté SUT pour *System Under Test* sur la figure 2 qui représente une vue générique du processus de test.

Dans le cas ou le système sous test est un système autonome, nous sommes confrontés à plusieurs problèmes. La diversité des situations auquel est soumis ce type de système, implique qu'en phase de test, il convient de soumettre le système à une importante variété d'entrées. Dans ce contexte

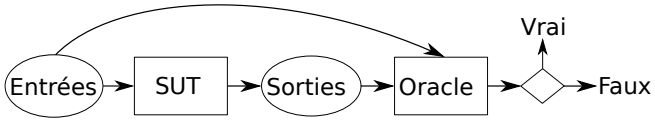

FIGURE 2: Vue générale du test

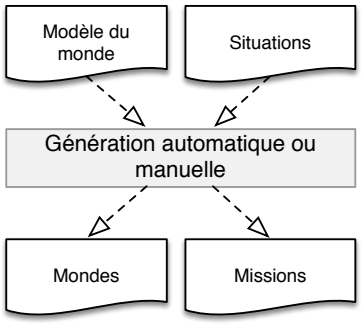

FIGURE 3: Vue sélection d'entrée de test de robustesse en robotique

la génération procédurale de mondes comporte l'avantage d'économiser du temps lors de la génération d'entrées de test. Elle permet par ailleurs d'obtenir des entrées plus diverses que si elles avaient été créées manuellement. Dans la littérature, on peut citer le papier [6] qui présente une telle approche. Le test de la robustesse des algorithmes de navigation d'un système autonome est effectué grâce à un monde généré à l'aide d'un bruit de Perlin 2D, ainsi qu'une mission de navigation et une population d'obstacles mobiles. Les entrées peuvent aussi être obtenues à partir d'un ensemble de situations nominales et critiques. C'est la démarche utilisée dans [7], qui traite du test d'algorithmes de vision par ordinateur. Les auteurs s'appuient sur deux modèles : un modèle de domaine qui décrit les objets possibles et l'environnement, et un modèle de situations critiques. Ces situations étant préalablement identifiées grâce à des techniques d'analyse de risque de type *HAZOP* (*HAZard OPerability*). Les entrées sont générées dans le but de couvrir les situations critiques et le domaine. Cette démarche se retrouve aussi dans [8]. La Figure 3 présente de façon générique les éléments que l'on retrouve dans ces différentes approches de la génération des entrées de test pour les systèmes mobiles.

Cette génération de mondes et de missions, mène en général à un nombre très important de possibilités qu'il convient de sélectionner. Dans le domaine du test de systèmes informatiques, deux approches peuvent être citées : la sélection déterministe (manuelle ou assistée d'outils comme des solvers de contraintes) ou non déterministe. Pour cette deuxième catégorie plusieurs techniques de génération automatique d'entrées de test ont été étudiées dans la littérature, par exemple en utilisant des méta-heuristiques [9], ou encore en échantillonnant l'espace des possibles puis en utilisant des techniques de *clustering* afin de supprimer les scènes trop similaires [7].

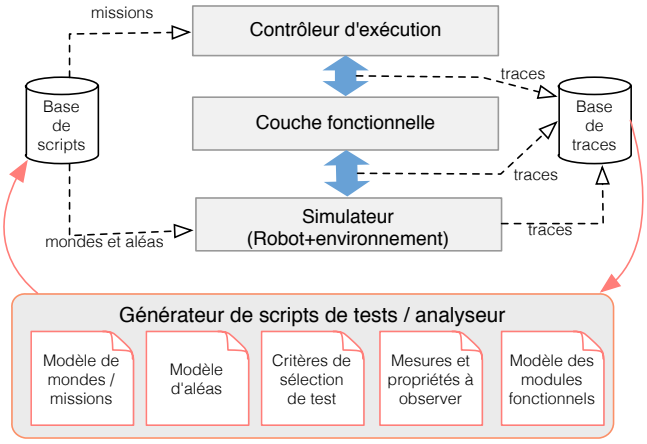

FIGURE 4: Objectif final

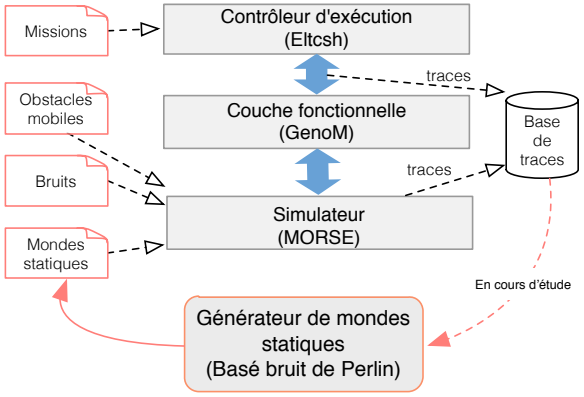

FIGURE 5: Expérimentation et architecture actuelle

Enfin, comme mentionné précédemment, la mise en place d'oracles de tests en robotique est une problématique complexe due à l'indéterminisme des algorithmes de navigation.

III. VERS LE TEST DANS DES MONDES VIRTUELS

A. Vue générale

La démarche que nous proposons est représentée figure4 sous forme d'une boucle entre la production de la base de scripts et la lecture des traces d'exécution. Nous avons identifié trois principales phases dans cette boucle :

1) Génération : durant cette phase on génère des mondes et des missions à partir du modèle de mondes/missions et le modèle d'aléas (situations critiques). Ces mondes sont sélectionnés afin de satisfaire des critères de sélection de test et ils doivent être adaptés au robot testé décrit dans le modèle des modules fonctionnels.
2) Simulation : les missions sont injectées dans le contrôleur d'exécution qui se chargera de piloter la couche fonctionnelle. Les mondes et aléas sont injectés dans le simulateur.
3) Analyse : les traces collectées durant la phase de simulation sont analysées et les mesures dérivés de ces traces serviront à calculer les nouveaux paramètres de la Génération (étape 1).

B. Le domaine d'entrée

Notre approche s'inspire de la méthode *FARM* [10] qui sert à définir les principaux attributs d'une campagne d'injection de fautes pour du test de robustesse. Lors de ce test il convient de définir les fautes injectées (*Fault*), l'activité du système (*Activity*), les logs (*Readouts*) ainsi que les mesures (*Measures*) qui en sont dérivées. Dans notre approche la notion de *Faute* concernera principalement l'introduction d'imprécisions (bruits par exemple) sur la perception. L'activité du système est ici la mission que le robot aura a réaliser.

C. La génération des entrées de tests

Le point le plus difficile de notre approche est la réalisation des méthodes permettant de relier les résultats des tests aux paramètres permettant de générer les entrées des tests. Comme présenté sur la figure, un ensemble de données vont servir pour ajuster ces paramètres (critères de test, modèle d'aléas, etc.). Par exemple dans la génération de monde avec un bruit de Perlin, le paramètre que l'on peut utiliser est nombre d'octaves (permettant de générer des terrains plus chaotiques). Ainsi, il serait possible de lancer des tests en augmentant progressivement la difficulté du terrain via ce paramètre.

IV. EXPÉRIMENTATION

Une vue générale des technologies utilisées pour l'expérimentation est présentée figure 5. Cette figure fait également apparaitre l'état actuel du projet, avec la prochaine étape (flèche intitulée ≪en cours d'étude≫). Comme présenté précédemment nous utilisons une architecture LAAS, avec des composants *GenoM* [11] et le simulateur *MORSE*.

A. Le système sous test

Les diverses expérimentations ont été effectués en simulation sur l'avatar du robot Mana qui est un robot mobile à quatre roues non holonome. Mana utilise le module *P3D* qui fait sa planification à partir d'une carte d'élévation fournit par un module nommé DTM. Nous considérons que ces deux modules sont les systèmes sous test ; nous n'incluons donc pas volontairement ce qui relève de la locomotion.

B. Architecture et composants de tests

Pour décrire l'architecture du test nous avons utilisé une adaptation d'un profil UML pour le test de robustesse [12]. D'un point de vue architectural, quatre composants ont été définis : l'ordonnanceur, le générateur, le moniteur ainsi que l'analyseur.

Ordonnanceur: il est le chef d'orchestre de la campagne de tests. Il gère le départ des diverses phases des tests et lance les divers composants associés en leur transmettant leur paramètres.

FIGURE 6: Trajets observés sur 20 simulations avec les mêmes entrées de test

Générateur: ce composant permet de générer les entrées de test, c'est à dire un monde et une mission. Il peut prendre divers paramètres - qui lui permettrons de construire la carte notamment - comme le nombre d'octaves de bruit de Perlin, ou encore le nombre d'obstacles présents sur la carte.

Moniteur: le moniteur surveille le comportement du robot pendant toute la phase de simulation. Il enregistre (*logs*) le trajet parcouru par celui-ci ainsi que des événements catastrophiques tels qu'une chute ou bien une collision entre le robot et un obstacle.

Analyseur: ce composant permet d'analyser les divers *logs*. Sa mission est double : d'une part il joue le rôle d'oracle et d'autre part il transmet à l'ordonnanceur des informations qui lui permettront de réorienter le prochain test.

C. Résultats préliminaires

Afin de valider notre architecture de test ainsi que notre approche, nous avons décidé d'observer comme première expérience, l'indéterminisme des modules dédiés à la navigation. Nous avons lancé plusieurs expériences sur divers types de cartes plus ou moins bruitées. Pour évaluer l'indéterminisme nous avons calculé la distance maximale entre des points deux à deux sur la même fenêtre temporelle entre toutes les trajectoires.

Afin de générer les cartes, nous avons utilisé le bruit de Perlin de la librairie python *noise* version 1.2.2. Nous avons couplés trois bruits de Perlin basés sur l'élévation, la difficulté, et le niveau de détail. Cela nous a permis d'obtenir un monde plus détaillé et réaliste qu'avec un bruit de Perlin simple. Puis, 20 exécutions de la même mission dans ce monde ont été réalisées. Cette expérience a été réalisée dans cinq mondes différents, ne comportant pour l'instant aucun obstacle statique ou dynamique (soit 100 exécutions au total). Une vue 3D est présentée sur la Figure 5, où sont représentées les 20 exécutions pour un monde. La mission consistait à aller d'un point A à un point B. Outre la validation de l'architecture de test, cela nous a permis d'observer le non déterminisme de la navigation, puisque pour ce monde, certaines exécutions sont un échec et d'autres un succès ! Sans détailler les raisons des échecs, car ce n'est pas l'objet de ce papier, nous avons établi dans un premier temps qu'ils provenaient du caractère accidenté du terrain (et du fait que notre robot n'est pas capable pour l'instant de replanifier).

V. CONCLUSION ET DIRECTIONS FUTURES

Ce travail préliminaire a permis d'identifier où se situaient les points durs de la mise en place d'une méthodologie de test dans des mondes virtuels pour des robots autonomes. Notamment, il est complexe d'établir des règles pour la transformation des résultats des tests (*logs*), en paramètres de génération des mondes. Une première étude expérimentale nous a permis d'explorer les limites et les possibilités des outils actuels de simulation, et d'effectuer un premier cas d'étude avec les outils développé au LAAS. Cette étude est bien sûr incomplète, et nous explorons actuellement la possibilité de reboucler les exécutions des tests.

RÉFÉRENCES

[1] T. Peynot, *Sélection et contrôle de modes de déplacement pour un robot mobile autonome en environnements naturels*. PhD thesis, Institut National Polytechnique de Toulouse-INPT, 2006.

[2] D. Bonnafous, S. Lacroix, and T. Siméon, "Motion generation for a rover on rough terrains," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 2, pp. 784–789, IEEE, 2001.

[3] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine : Morse," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 46–51, IEEE, 2011.

[4] A. Peytavie, *Génération procédurale de Monde*. PhD thesis, Université Claude Bernard-Lyon I, 2010.

[5] J.-C. Laprie, J. Arlat, J.-P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.-C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac, and P. Thévenod, *Guide de la sûreté de fonctionnement*. Toulouse, France : Cépaduès - Éditions, 1995.

[6] J. Arnold and R. Alexander, "Testing autonomous robot control software using procedural content generation," in *Computer Safety, Reliability, and Security*, pp. 33–44, Springer, 2013.

[7] O. Zendel, W. Herzner, and M. Murschitz, "Vitro-model based vision testing for robustness," in *Robotics (ISR), 2013 44th International Symposium on*, pp. 1–6, IEEE, 2013.

[8] Z. Micskei, Z. Szatmári, J. Oláh, and I. Majzik, "A concept for testing robustness and safety of the context-aware behaviour of autonomous systems," in *Agent and Multi-Agent Systems. Technologies and Applications*, pp. 504–513, Springer, 2012.

[9] X. Zou, R. Alexander, and J. McDermid, "Safety validation of sense and avoid algorithms using simulation and evolutionary search," in *Computer Safety, Reliability, and Security*, pp. 33–48, Springer, 2014.

[10] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault injection for dependability validation : A methodology and some applications," *Software Engineering, IEEE Transactions on*, vol. 16, no. 2, pp. 166–182, 1990.

[11] S. Fleury, M. Herrb, and R. Chatila, "GenoM : A tool for the specification and the implementation of operating modules in a distributed robot architecture," in *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, vol. 2, pp. 842–849, IEEE, 1997.

[12] R. Moraes, H. Waeselynck, and J. Guiochet, "UML-based modeling of robustness testing," in *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on*, pp. 168–175, 2014.