De l'utilité des jeux vidéo pour l'enseignement de l'informatique temps-réel

J. Sérot (1) , J. Laffont (2) , M. James (2)

(1) Polytech'Clermont-Ferrand / LASMEA UMR 6602 CNRS/UBP
(2) Polytech'Clermont-Ferrand

Résumé

On décrit une maquette pédagogique visant à faciliter l'apprentissage des principaux concepts de la programmation concurrente et temps-réel sur une plate-forme de type micro-contrôleur embarquée. La principale originalité de cette maquette est de s'appuyer sur un modèle virtuel de l'environnement de l'application étudiée suffisamment réaliste pour appréhender les problématiques posées dans la réalité par ce type d'applications. Cette maquette est utilisée depuis 2005 au sein de la filière *Systèmes Embarqués* du département Génie Electrique de Polytech'Clermont-Ferrand.

1. Introduction

Tous les enseignants confrontés à l'enseignement des concepts et techniques de l'informatique concurrente et temps-réel savent le challenge que représente cette mission.
La principale difficulté est d'abord conceptuelle : la décomposition d'une application en tâches concurrentes et coopérantes va à l'encontre des habitudes de programmation de la plupart des étudiants. Même pour ceux familiers des techniques de l'informatique dite industrielle (fonctionnement sous interruptions, ...), le mode de pensée et la vision des applications restent en effet essentiellement séquentiels.
A ceci s'ajoute souvent des difficultés d'ordre plus technique. D'abord, dès que l'application dépasse un certain niveau de complexité, le recours à un noyau / exécutif temps-réel est quasiment indispensable, ce qui, en pratique, suppose en bonne connaissance de l'API du noyau employé (nature et interface des fonctions de contrôle) et un minimum de familiarité avec la programmation système (gestion de la mémoire, interruptions, ...). Par ailleurs, certaines notions - réactivité, échéance temporelle, conflit d'accès, ... - ne peuvent être clairement illustrées que dans un contexte où l'application étudiée dialogue avec son environnement autrement que par un simple clavier/écran. Ceci suppose, en pratique, de développer l'application sur une cible de type micro-processeur/micro-contrôleur embarqué disposant des interfaces matérielles appropriées (capteurs, actionneurs) et donc d'utiliser des chaînes et des outils de développement dédiés.
Mais la principale difficulté consiste à plonger cette application au sein d'un environnement à même d'exercer et d'illustrer ses propriétés de concurrence et de temps-réel justement.

Les applications de type robotique mobile, embarquées sur des mini-véhicules comme les micro-robots PIONEER, PEKEE ou WIFIBOT [1,2,3] peuvent constituer, au premier abord, une réponse au cahier des charges esquissé ci-dessus. La diversité des missions que l'on peut confier à ce type de robot et des capteurs utilisables (télémétrie, odométrie, vision, ...) font qu'il est possible d'aborder, en développant une application sur un tel robot, la quasi-totalité des problématiques sus-citées, et ce dans un contexte*de facto* très réaliste. La disponibilité de noyaux Linux temps-réel pour certaines de ces plates-formes peut faciliter par ailleurs le développement et la migration de code. Ces solutions ont toutefois plusieurs inconvénients. D'abord, les exécutions n'étant pas par définition reproductibles, la mise au point des algorithmes peut s'avérer compliquée. D'autre part, pratiquement, elles posent de réels problèmes de fiabilité dès lors qu'elles sont utilisées de manière intensive, dans le cadre de séries de travaux pratiques par exemple (d'autant plus que, pendant la phase de mise au point, le matériel est en général mis à rude épreuve : commandes incohérentes sur les actionneurs, collisions, *etc.*). A ceci s'ajoute un problème de coût, qui peut devenir important lorsqu'il s'agit d'équiper un ensemble de postes

Article disponible sur le site http://www.j3ea.org ou http://dx.doi.org/10.1051/j3ea:2008066

Face à ce constat, nous avons donc cherché à développer une maquette permettant d'illustrer, dans le cadre favorable de la robotique mobile, les problématiques de la programmation concurrente et temps réel, mais sans avoir à subir les contingences liées à l'usage d'un robot matériel. C'est cette maquette, et le matériel pédagogique que nous avons développé autour, que nous présentons dans cet article.

L'idée consiste à remplacer le véhicule réel par un véhicule virtuel, simulé par un PC, évoluant dans un environnement lui aussi simulé. Le recours à un logiciel spécialisé[1] pour la modélisation du véhicule et de son environnement (RAYDIUM) confère un très grand réalisme à la simulation. L'application elle-même fait appel à un noyau temps-réel et s'exécute sur le micro-contrôleur d'une carte déportée dialoguant avec le simulateur via une liaison sérielle. Du point du vue du développeur, donc, tout se passe exactement comme si le micro-contrôleur était embarqué sur un véhicule réel[2]. La complexité des missions assignables au véhicule n'est limitée que par celle du modèle de l'environnement dans lequel on le fait évoluer et par celle de l'application implantée sur le micro-contrôleur. Toutes les problématiques temps-réel qui seraient illustrables avec un robot physique le sont avec cette maquette.

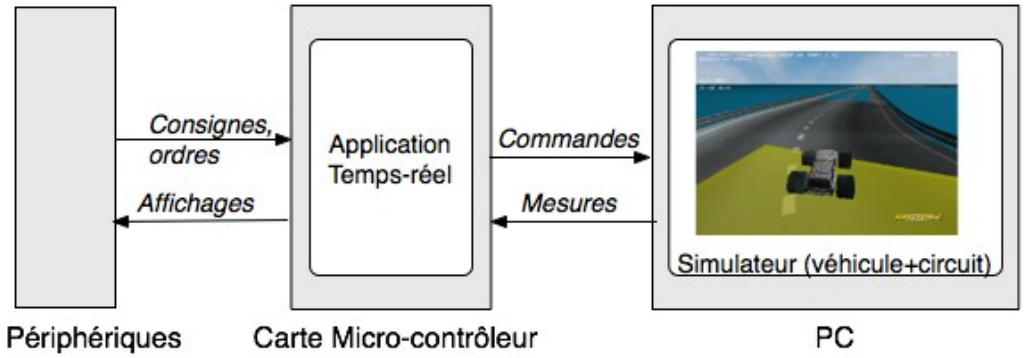

Figure 1 – Principe de la maquette

Le plan suivi pour la suite de cet article est le suivant. La section 2 décrit le matériel et le logiciel utilisés. Dans la section 3, on présente un exemple d'application développée sur cette plate-forme, et qui est utilisée actuellement au sein de notre filière dans la cadre de l'enseignement sur les systèmes réactifs et temps-réel. La section 4 présente les conclusions et enseignements que nous avons pu tirer de cette expérience.

2. Description de la maquette

Une vue générale de la maquette est donnée figure 2. Les deux fenêtres sur l'écran du PC sont celles de l'environnement de développement du micro-contrôleur (en arrière-plan) et du simulateur (premier plan).

Le simulateur lui-même est écrit en C à l'aide de la bibliothèque RAYDIUM [5]. Cette bibliothèque a été développée initialement pour la programmation de jeux vidéos. A ce titre elle propose tout un ensemble de fonctions pour la gestion des entrées-sortie (souris, clavier, joystick), le rendu 3D de scène (via OPENGL), le son et sa spatialisation, la gestion du réseau et des communication client/serveur. Mais surtout, elle s'appuie sur un moteur physique pour reproduire de manière réaliste le déplacement d'objets dans une scène. Un moteur physique gère l'évolution d'une scène composée de corps solides non déformables en se fondant directement sur lois de la physique. Il permet donc de simuler le comportement d'objets en fonction de leur masse, moment d'inertie, forces et couples appliqués. Les mouvement de chaque objet peuvent être contraints par des liaisons, qui définissent les degrés de libertés de mouvement de cet objet par rapport à un autre. Il est possible d'asservir les déplacements ou rotations selon certains axes des liaisons en utilisant des moteurs. Lors d'une itération le moteur physique réalise une détection de

1 Très utilisé par ailleurs dans le monde des jeux vidéos - d'où le titre de cet article

2 A ceci près que les collisions ne portent pas atteinte, ici, à l'intégrité physique du robot !

collisions permettant de définir des liaisons ponctuelles simulant les interactions d'objets libres entre eux. Le moteur physique trouve à chaque itération une nouvelle configuration de la scène respectant les équations physiques et les contraintes appliquées. La bibliothèque RAYDIUM utilise le moteur physique OPENDE [6].

Dans notre cas, le véhicule simulé est modélisé à l'aide de sphères et de parallélipèdes. Il est doté de quatre roues, dont deux sont motrices, d'une tourelle portant un télémètre ultrasons, et d'un capteur indiquant la couleur de la route. Les roues sont commandables séparément en direction et en vitesse. La commande de la tourelle permet de définir sa vitesse de rotation et de déclencher des mesures. En retour on peut observer l'angle effectif du télémètre par rapport à l'axe de la voiture et la distance mesurée.

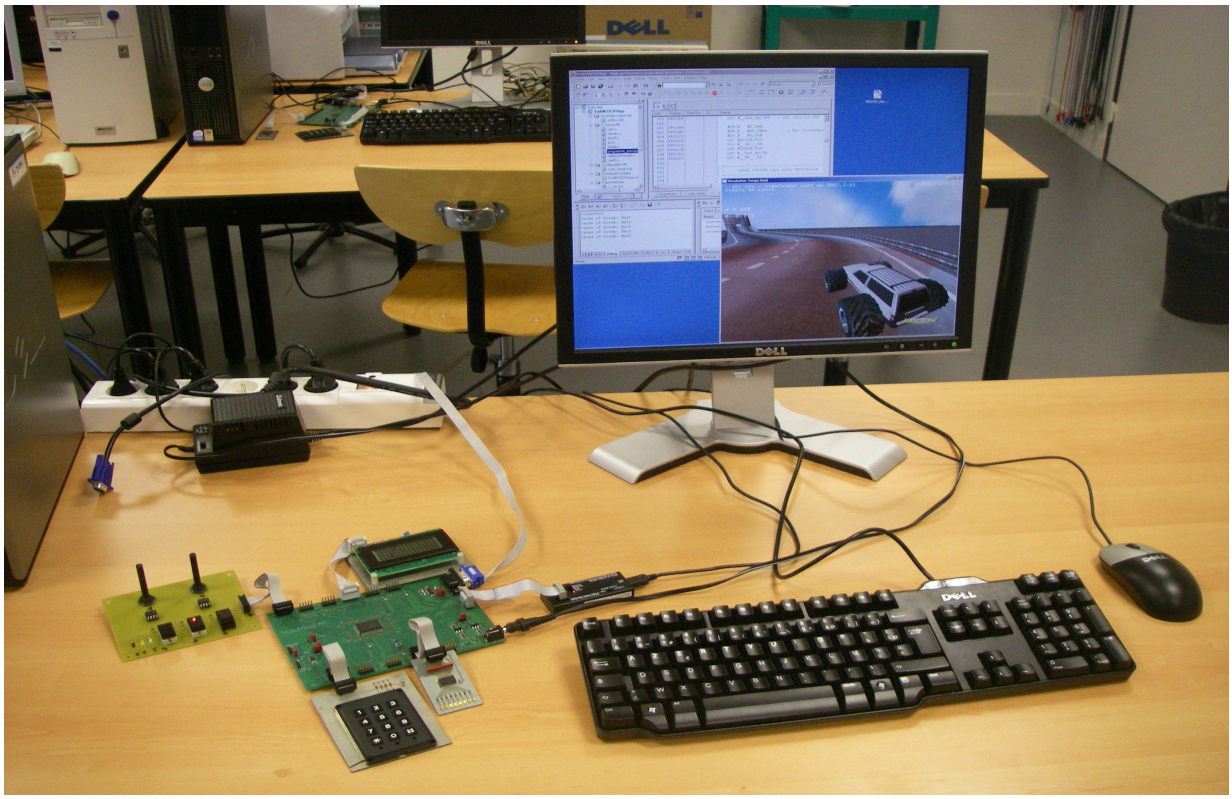

Figure 2 – Vue générale de la maquette en fonctionnement

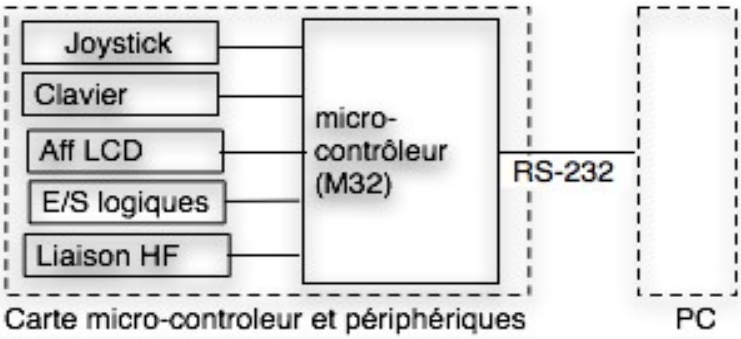

Figure 3 – Synoptique matériel

Le simulateur tourne sur un PC d'entrée de gamme (Pentium 4, 2Ghz, 512Mo) équipé d'une carte graphique 3D (ATI Radeon 9500).

Le micro-contrôleur, sur lequel s'exécute l'application contrôlant le véhicule est un M32C87 de Renesas (16 bits). Il est relié aux périphériques suivants (cf. figure 3) : un clavier matricé 12 touches, un afficheur alphanumérique à cristaux liquides 4 lignes de 16 caractères, une platine portant deux potentiomètres (jouant le rôle d'un *joystick*) , 3 boutons et 3 diodes électro-luminescentes, un récepteur multi-voies de radio-commande HF et une liaison série type RS232 pour communiquer avec le simulateur présent sur le PC. Les routines d'accès bas-niveau à ces périphériques ont été étudiées et développées en partie par les étudiants dans une série de TP précédente dédiée à la programmation des micro-contrôleurs. Un jeu de routine opérationnel est fourni en début de TP par l'équipe enseignante, l'objectif étant ici que les étudiants se concentrent sur les problématiques «temps réel ».

L'environnement de développement pour le micro-contrôleur et les outils associés (compilateur C, simulateur, *debugger*) ont été fournis gratuitement par la société Renesas. L'application est écrite en utilisant les services d'un exécutif temps-réel propriétaire, MR308/4. Ce noyau répond aux spécifications µTRON4.0, une norme industrielle japonaise concernant les noyaux temps réels [4]. Les possibilités offertes par ce noyau sont comparables à de nombreux produits équivalents. L'empreinte mémoire du noyau est corrélée au nombre de primitives instanciées dans l'application ce qui est bien adapté aux applications embarquées sur micro-contrôleurs.

Le simulateur et l'application s'exécutant sur le micro-contrôleur dialoguent via une liaison série (RS-232 dans notre cas, mais d'autres implantations sont aisément envisageables – USB, Ethernet, ...) grâce à un protocole *ad-hoc*. Les aspects bas-niveau de ce protocole (formatage des trames, synchronisation, ...) sont encapsulés et l'application dialogue avec le simulateur via un ensemble de fonctions permettant d'observer (resp. de commander) les différents organes du véhicule : roues (angle et vitesse), télémètre (angle et mesure), etc... Les mesures sont asynchrones : pour lire une valeur, l'application fait une requête de lecture auprès du périphérique concerné; le simulateur répond en écrivant la valeur dans un descripteur et prévient l'application soit en passant un drapeau à 1 (attente active) soit en générant un événement logiciel (attente passive).

3. Applications

Afin de favoriser une familiarisation avec les outils et une assimilation progressive des concepts sous-jacents il est demandé aux étudiants de développer, sur la maquette présentée ci-dessus, plusieurs applications de complexité croissante.

La **première application** consiste à prendre en main les périphériques de la carte et à maitriser la communication application-simulateur. Pour cela, on demande de réaliser un programme permettant de controler la tourelle du télémètre depuis le clavier. On ne s'intéresse donc pas encore, à ce niveau, au contrôle du véhicule (qui restera à l'arrêt). L'application doit permettre la définition de l'angle d'azimut du télémètre à partir d'une valeur en degré rentrée au clavier. La tourelle est asservie en position mais la consigne donnée est une vitesse angulaire permettant de positionner la tourelle en azimuth. Dans cette application, les étudiants doivent choisir les primitives du noyau les plus adaptées pour répondre au cahier des charges. Pour cela, ils doivent comprendre le fonctionnement des différents types de tâches et les différents moyens de faire communiquer ces tâches (variables partagées, événements, queue de messages, *etc.*).

La **seconde application** consiste à mettre en oeuvre la régulation de la direction du véhicule en s'appuyant sur la mesure de la distance au bord de la piste, effectuée par le télémètre ultra sons.A la fin de la séance, le véhicule doit être capable de réaliser un tour complet de circuit de façon autonome. La difficulté de cette application consiste à faire cohabiter deux régulations, celle concernant la tourelle du télémètre et celle de la direction du véhicule.

Dans la **troisième application,** le véhicule doit maintenant suivre plusieurs schémas de régulation en fonction de la portion du circuit abordée (à chaque portion, on peut associer un type de terrain et donc un réglage optimal des paramètres de régulation). Les portions du circuit sont identifiées par des zones de couleur sur la piste, repérées par un capteur dédié du véhicule (cf. figure 4). Il est souhaitable que chaque schéma soit sous la forme d'une tâche propre. Une tâche de supervision aura alors pour rôle d'activer ou de désactiver les tâches correspondantes aux différents schémas en fonction de la portion de piste sur laquelle se trouve le véhicule.

Figure 4 – Capture d'écran du simulateur

Dans la **quatrième et dernière application**, on finalise les différents modes de régulation en fonction de la section de piste sur laquelle se trouve le véhicule. Le véhicule doit exécuter un tour complet de circuit en un temps minimum. Dans cette application, aux contraintes liées au respect des échéances temporelles pour les différentes régulations, s'ajoute la nécessité de prendre en compte des événements asynchrones en provenance de l'opérateur. Par exemple, la vitesse de consigne et le gain des régulations sur les différentes zones du circuit doivent pouvoir être ajustés *pendant le fonctionnement de* l'application via les touches du clavier matricé et l'angle de consigne du télémètre est modifiable via le joystick. L'application doit par ailleurs afficher en permanence le taux d'occupation du processeur (0-100%) et la distance mesurée par le télémètre doivent être affichés sur l'écran LCD au moins une fois par seconde. C'est véritablement à ce niveau que les étudiants prennent conscience d'une part de la nécessité d'une décomposition de l'application en tâches indépendantes / coopérantes et d'autre part de l'intérêt d'utiliser un exécutif temps-réel pour implanter cette structure en assurant le respect des contraintes temporelles.

4. Résultats, bilan et perspectives

La plate-forme décrite ici a été utilisée depuis trois ans auprès de publics variés : étudiants en 2^{ième} année de cursus ingénieur au sein de la filière *Systèmes Embarqués* du département Génie Electrique de Polytech'Clermont-Ferrand, étudiants en Licence Professionnelle, mais aussi dans le cadre de plusieurs projets tutorés.

Le retour d'expérience est globalement très positif.

Comme indiqué plus haut, cette approche offre la possibilité de travailler sur des systèmes « réalistes » sans supporter les contingences liées à l'usage de matériel spécifique, difficile à mettre au point, à maintenir et à faire évoluer. Les manipulations proposées sont robustes, fiables, réactives et duplicables. L'usage de logiciels libres (au moins pour la partie simulation) favorise la diffusion et la maintenance des solutions.

Pour les étudiants, les objectifs fixés sont aisément compréhensibles et facilement visualisables. Le fait de pouvoir relier les performances finales du véhicule à des options d'implantation telles que l'usage de tel ou tel mécanisme de synchronisation ou l'assignation de tel ou tel niveau de priorité à une tâche donnée est très formateur. Ces facteurs favorisent par ailleurs l'émergence d'une émulation constructive entre les groupes et développe l'autonomie. Il faut toutefois veiller tout particulièrement à ce que les étudiants différencient bien la partie opérative de la partie commande, différentiation qui est ici « gommée » par la virtualisation, *a fortiori* si ces deux parties s'exécutent sur la même machine.

Pour l'équipe enseignante, l'usage d'une telle « plate-forme virtuelle » favorise un véritable travail transversal sur les différents aspects d'une même application (modélisation des systèmes physiques, informatique, informatique industrielle, automatique,robotique*, etc.*).

Les deux principaux freins à l'utilisation de cette approche sont la relative complexité de la tâche de modélisation de l'objet simulé et de son environnement et la puissance de calcul requise par le simulateur.

La modélisation physique de l'objet et de son environnement n'est pas triviale et suppose une analyse rigoureuse des phénomènes à simuler. Certains paramètres du moteur physique doivent parfois être ajustés de manière empirique afin d'obtenir une simulation réaliste. L'ajustement de ces paramètres doit se faire avec précaution car les modèles numériques sous-jacents peuvent diverger dans certains cas. Cette étape de modélisation physique se double d'une étape de modélisation graphique, avec des logiciels comme Blender ou 3DS Max. L'effort correspondant est difficilement quantifiable dans l'absolu car il dépend des compétences initiales de l'enseignant. A titre purement indicatif, la modélisation complète (physique et graphique) d'un bras robotique à 6 degrés de liberté a pu être menée en une cinquantaine d'heures par une personne sans compétences préalables dans ces domaines

Par .ailleurs, la précision et le réalisme sont limités en pratique par la puissance de calcul disponible, si l'on souhaite conserver un taux de rafraîchissement acceptable. Il faut donc trouver un compromis acceptable entre réalisme, vitesse et précision. L'utilisation d'une carte graphique correcte permet de simuler une dizaine de voitures sur un même circuit avec un moteur physique itérant à 400Hz et un rafraîchissement d'environs 60 images/seconde.

L'évolution des outils dans le domaine des moteurs physiques et du rendu graphique – liée en grande partie aux besoins sans cesse croissants des jeux vidéos – va probablement, et à très court terme, considérablement limiter les inconvénients sus-cités. On peut à ce titre citer l'émergence d'une nouvelle génération de moteurs physiques, comme Physsim [7] qui offrent un niveau de précision bien supérieur et la possibilité d'accélérer les calculs du moteur physique en les déportant sur les GPU équipant les cartes graphiques des ordinateurs via des bibliothèques comme cuda [8].

Nous estimons donc que le recours à des « plate-formes d'expérimentation virtuelles » -- telle que celle présentée ici – va constituer, à court terme, une solution pédagogiquement valide tant en termes d'effort de développement pour les enseignants qu'en termes de vecteur de compréhension pour les étudiants.

Références

[1] http://www.pekee.fr
[2] http://www.wifibot.com
[3] http://www.activrobots.com/ROBOTS/p2at.html
[4] http://www.assoc.tron.org/spec/itron/mitron-400e.pdf
[5] http://raydium.orgm :
[6] http://opende.sourceforge.net
[7] http://physsim.sourceforge.net
[8] http://www.nvidia.com/object/cuda_home.html