Réalisation d'un robot sous-marin autonome

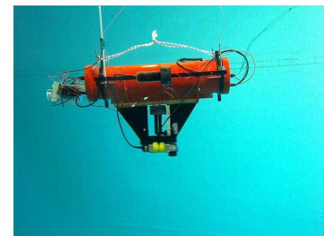
Fourniture 2 associée au contrat MRIS 2008-2009
Jan Sliwka, Fabrice Le Bars, Luc Jaulin.
ENSIETA
Avril 2009

Descriptif de l’étude

Comme tous les ans depuis 2007, un robot sous-marin réalisé par des élèves de l’ENSIETA sera présenté pour le concours SAUC’E (Student Autonomous Underwater Challenge – Europe) 2009. La compétition, organisée par le Ministery of Defense (MoD) Britannique et la Délégation Générale pour l’Armement (DGA) aura lieu cette année du 6 au 10 Juillet à Gosport en Angleterre, comme pour l’édition 2007. Les épreuves demandées aux robots sous-marins autonomes se déroulent dans une piscine et mettent en jeu des problématiques de détection, localisation et poursuite d’objets avec cartographie. Vu que pour la première fois, une équipe a réussi à effectuer toutes les épreuves du concours (l’équipe d’Heriot-Watt, au concours 2008), des missions ont été rajoutées et/ou compliquées. Pour plus de détails, vous pouvez consulter le site

Etat actuel du projet

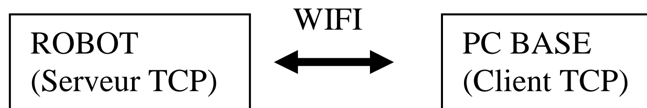
Le sous-marin actuel, qui a été présenté au concours 2008 est fonctionnel mais quelques problèmes, principalement des bugs informatiques l’ont empêchés de mener à bien les missions du concours. Il n’est donc pas à l’heure actuelle totalement autonome.



Développement de cette année

Cette année, nous avons décidé de nous focaliser sur la fiabilisation de la partie informatique du robot. Nous travaillons notamment sur le Simulateur, la réorganisation du code du robot autonome, la création d’une IHM et finalement la génération automatique de code à partir d’un modèle graphique. Nous essayons aussi de développer une communication acoustique nous permettant de faciliter le débogage du code pendant les tests dans la piscine.
Vous trouverez les détails sur ces travaux dans les sous sections ci-dessous.

L'IHM



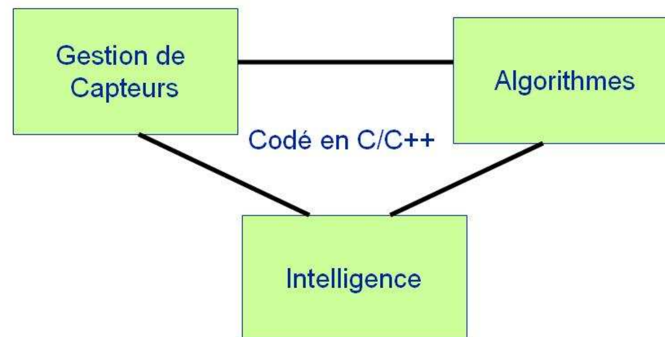
En mode télé opéré, Le PC de base communique avec le robot par WIFI. Nous aimerions avoir une IHM qui nous permettrais de

- Tester le bon fonctionnement du robot et le préparer au lancement de sa mission autonome et de se reconnecter au robot après son surfaçage.
- Rejouer la mission en se basant sur les données accumulées par le robot pendant sa mission
- Tester et déboguer tous les algorithmes

Jusqu'à maintenant nous utilisions une interface graphique exécutée sur le PC de base. Cette IHM était élaborée avec des fonctions graphiques de l'API WIN32 qui sont assez difficiles à utiliser. La taille du code est vite devenu assez importante ce qui a rendu difficile une évolution quelconque de celle-ci.
Nous avons décidé de faire une nouvelle interface graphique en utilisant des outils plus adaptés à un développement rapide comme par exemple *C++ Builder* ou *Qt*. Nous sommes actuellement en train d'étudier les avantages et inconvénients de l'un ou de l'autre : C++ *Builder* semble intéressant pour sa facilité d'utilisation. *Qt* a l'avantage d'être portable.
Nous nous sommes aussi posé la question de l'endroit d'exécution de l'interface graphique. Celle-ci peut s'exécuter comme l'ancienne IHM sur le PC de base. La nouvelle approche serait d'exécuter l'interface graphique sur le robot lui-même (vu que le PC104 est un PC comme les autres). Pour que l'utilisateur puisse voir l'interface, celui-ci devrait se connecter au robot avec une connexion bureau à distance (Windows) ou connexion SSH avec un serveur X11 lancé sur le PC de base (Linux), ou encore par connexion VNC.

La réorganisation du code

Pour rendre le code modulaire et facilement utilisable nous procédons à une réorganisation de code en divisant le code source en plusieurs groupes et modules et en spécifiant des interfaces entre les différents modules.
Voici un aperçu d'une division par groupe :



Voici un aperçu des modules fonctionnels du robot

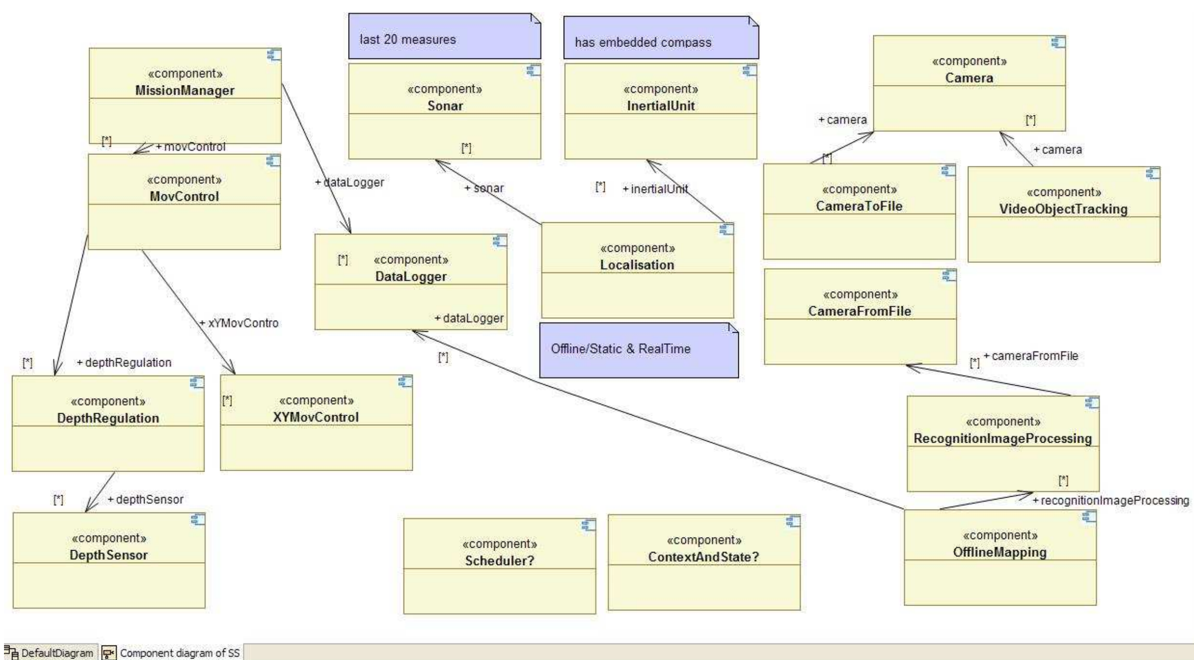


Figure 1 : modules fonctionnels du logiciel embarqué. Créé avec PapyrusUml

Le simulateur

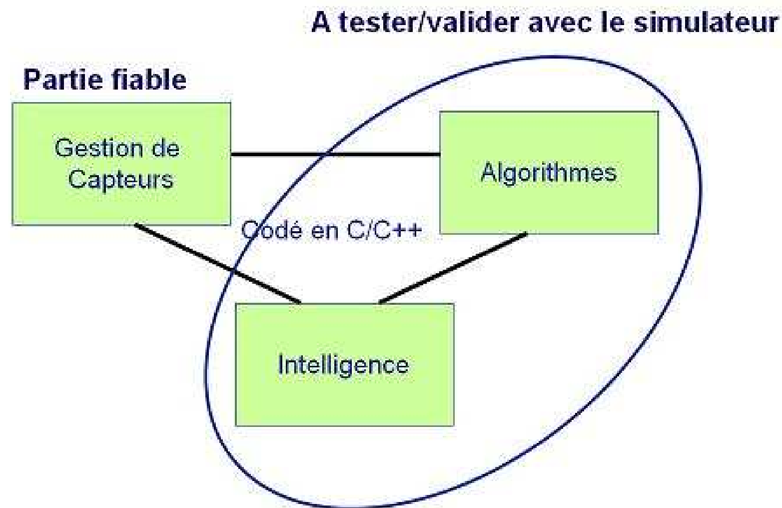
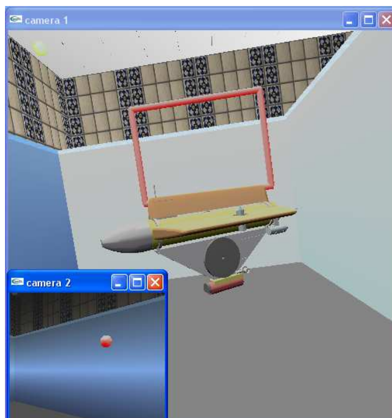


Figure 2 : Modules logiciels à tester avec le simulateur

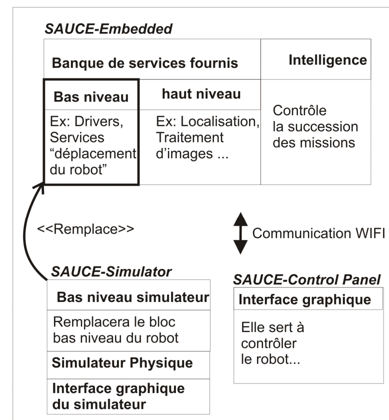
Cette année nous bénéficions de l'aide de deux étudiants de deuxième année qui travaillent sur le simulateur dans le cadre de leur projet industriel (13 séances de 1 journée répartis sur 4 mois). Le simulateur servirait à tester le code source du robot, par conséquent celui-ci devra être capable de se comporter comme si on avait un vrai robot dans la piscine. Il faudra donc que le simulateur :

- S'interface avec le code gérant « l'intelligence » du robot (i.e. pouvoir utiliser les fonctions de détection d'objets, de calcul de position,…).
- Dispose d'une interface graphique pour afficher le robot dans son environnement (qu'on voit à l'écran un robot se déplacer dans une piscine)
- Simule le comportement physique du robot (équations d'état avec résolution par méthode d'Euler et éventuellement simulation précise des différents capteurs du sous-marin)

Un simulateur avait déjà été réalisé avec Visual Studio 6 et la bibliothèque OpenGL en 2006-2007 par un élève de 3ème année mais n'avait pas été repris par la suite.



Simulateur existant



Architecture informatique

La génération automatique de code

Pendant le concours de 2008, le robot devait effectuer plusieurs missions, exemple, passer par la porte de validation, percuter une bouée, lancer un marqueur sur une cible… Chaque mission pourra être partagée en plusieurs taches. Par exemple dans le cas ou le robot doit percuter la bouée, il faut d'abords que ce dernier la trouve, ensuite il faut qu'il se positionne devant la bouée et finalement, il faut qu'il lance la phase d'approche fine … Il faudra bien sûr gérer plusieurs exceptions. Par exemple, s'il y a un obstacle entre la bouée et le robot, il faudra que le robot le contourne …Tout ça pour dire que la gestion de l'exécution de missions est quelque chose de complexe.
Pour pallier à cette complexité, nous avons pensé à faire de la génération automatique de code à partir de machines à état (voir figure ci-dessous) édités en mode graphique.

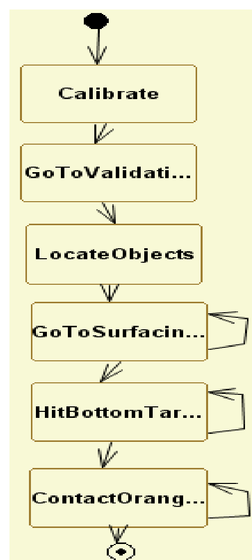


Figure 3 : Machine à état année 2008 éditée avec PapyrusUml

Si on regarde l'état *HitBottomTarget* correspondant au fait de percuter la bouée, celui-ci se décompose en plusieurs taches représentées aussi par des machines à états (voir figure suivante). Pour générer le code, nous disposerions

- d'un **éditeur graphique** qui nous permettrais de décrire l'évolution de notre système. On utiliserait un éditeur existant comme *Rhapsody*, *papyrusUml* … Ou bien nous ferions un éditeur par nous même en utilisant *Topcased* et *GMF* d'Eclipse, l'avantage serait d'avoir un éditeur plus adapté à nos besoins, l'inconvénient serait la non-conformité à la norme.
- d'un **générateur de code** qui transformerait le « model » crée avec notre éditeur en code C++. Pour ce faire, nous utiliserions un générateur déjà existant comme celui de du logiciel Rhapsody. Une autre alternative serait d'utiliser MDWorkbench ou Accéléo pour créer notre propre générateur

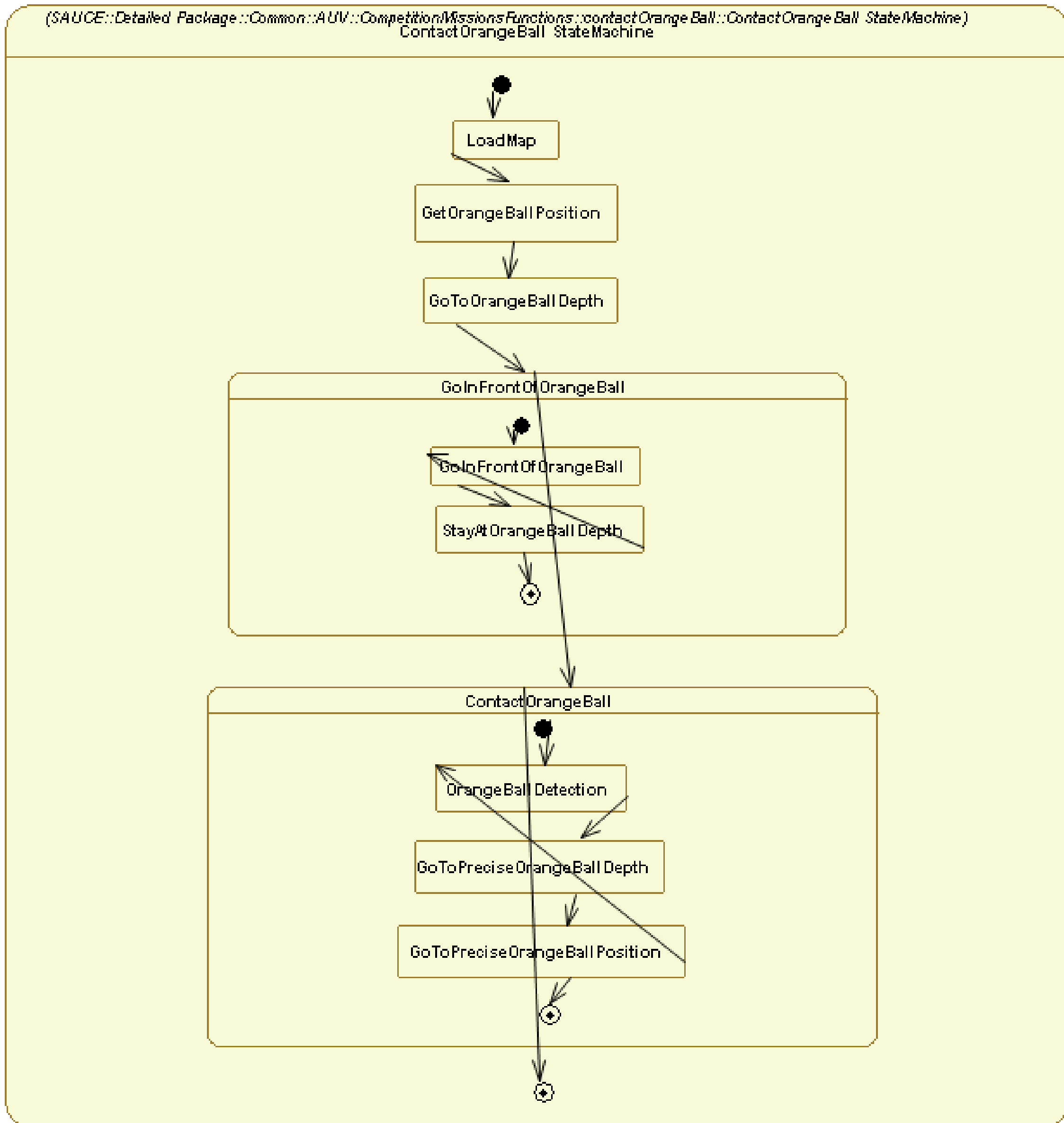


Figure 4 : exemple d'une sous-machine à état

Communication acoustique

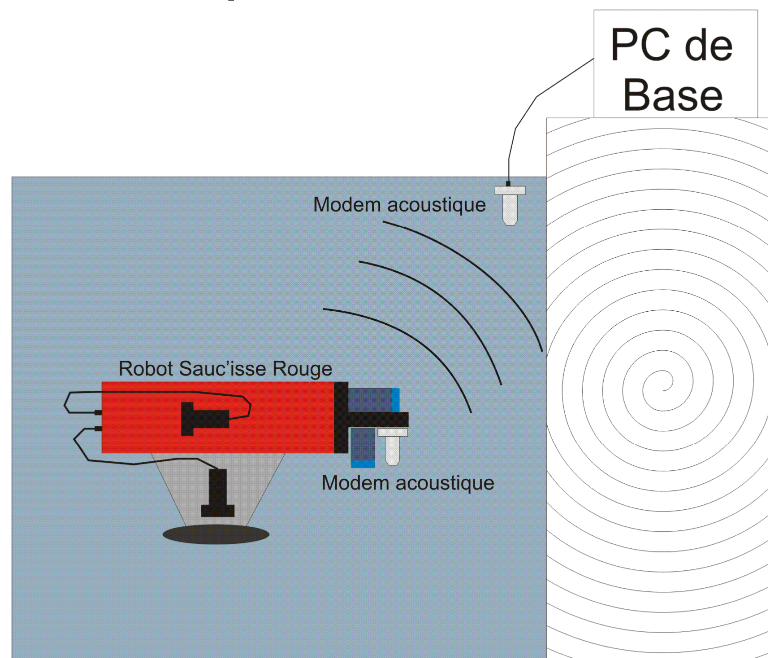


Figure 5 : Communication acoustique

Quand le robot se trouve dans le mode autonome, on perd la communication WIFI car les ondes électromagnétiques se propagent mal dans l'eau. Pour suivre l'évolution du robot en temps réel et pour pouvoir par exemple changer sa mission en cours de route, nous sommes en train de développer une liaison acoustique entre le robot et le PC de base. Deux stagiaires s'occupent de l'intégration de cette communication dans le robot.

Conclusion

La première année du concours le robot a été construit. L'année suivante nous avions fiabilisé tout ce qui est mécanique et électronique et nous nous sommes rendu compte que sans architecture informatique fiable nous ne pouvions pas tester nos algorithmes. Le but de cette année est donc de fiabiliser la partie informatique et ainsi obtenir une plateforme fiable nous permettant de tester nos différents algorithmes de traitement d'image, de localisation et de cartographie.