Université du Québec
École de technologie supérieure

RAPPORT TECHNIQUE

PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

DANS LE CADRE DU COURS LOG791- PROJET SPÉCIAUX

# VISION TEST TOOL

Jonathan Ducharme

DUCJ02108309

Jean-François Im

IMXJ20098206

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

**Professeur superviseur**

**Alain April**

MONTRÉAL, 20 AVRIL 2010
HIVERS 2010

# Table des matières

# Sommaire

Le contenu du présent document décrit l'analyse ainsi que les efforts requis et faits pour obtenir le Vision Test Tool, un outil de test de filtres de vision automatique pour le club S.O.N.I.A. Vous trouverez à l'intérieur du document le problème auquel le club étudiant S.O.N.I.A. à fait face ainsi que les solutions apportées pour corriger ce problème et comment l'équipe constituée de Jonathan Ducharme et Jean-François Im a fait pour arriver au logiciel final.

Il sera aussi question des améliorations qui devront être apportées dans le futur pour rendre le logiciel encore plus efficace pour traiter les vidéos, des risques quant au développement de cette application et des décisions de conception.

## Avant-propos et remerciements

Depuis notre entrée à l'École de technologie supérieure, nous nous sommes impliqués dans le club scientifique S.O.N.I.A. Nous avons à cœur les succès du club et nous sommes fiers d'avoir eu la chance de participer aux activités de S.O.N.I.A. Nous tenons à remercier particulièrement, M. François Coallier et Alain April qui nous ont permis de faire ce projet spécial dans le cadre d'une activité scolaire qui nous tient fortement à cœur.

Nous tenons aussi à remercier les membres du club S.O.N.I.A. qui nous ont aidés dans l'élaboration de ce projet, soit Martin Morissette et Marc-André Courtois. Nous voulons aussi remercier Vanessa Jean et Sébastien Martin pour leur support dans notre projet.

Jonathan Ducharme remercie sa compagne depuis les 3 dernières années, Gennifer Greiss, pour avoir composé avec les humeurs changeantes, la charge de travail et l'incapacité à être entièrement disponible pendant ces années consacrées au projet S.O.N.I.A.

# Glossaire

| | |
|---|---|
| **AUVSI** | **Association for Unmanned Vehicle Systems International** |
| **ETS** | **École de technologie supérieure** |
| **C++** | **Langage de programmation orienté objet** |
| **Chaîne de filtre** | **Ensemble de filtres de vision pour la détection d'objet** |
| **Filtre de vision** | **Traitement sur une image qui retourne l'image originale modifiée avec de nouvelles informations** |
| **GUI** | **Interface graphique d'un logiciel (Graphical User Interface)** |
| **Java** | **Langage de programmation orienté objet maintenu par Sun Microsystems (Oracle)** |
| **Movie Annotator** | **Outil logiciel du club S.O.N.I.A. pour annoter les vidéos avec les informations pertinentes pour les objets.** |
| **ONR** | **Office of Naval Research** |
| **OpenCV** | **Librairie d'Intel pour faire du traitement visuel** |
| **S.O.N.I.A.** | **Système d'Opération Nautique Intelligent et Autonome** |
| **SRS** | **Document de requis logiciel (Software Requirements Specification)** |
| **Vision Editor** | **Outil logiciel du club S.O.N.I.A. qui aide à la création de chaîne de filtres** |

# Introduction

Le club étudiant S.O.N.I.A. utilise de la vision robotique depuis plusieurs années. Dans la mission donnée au robot, il doit être en mesure de trouver des objets, et ce, de manière autonome et de les marquer, soit en fonçant dans l'objet, en le récupérant ou en laissant tomber quelque chose à l'intérieur de ce dernier.

Pour être en mesure d'identifier les objets, les membres du club S.O.N.I.A. doivent produire des filtres de vision capable de trouver ces objets. Les tests de ces filtres se font en général par un être humain et par l'instinct de ce dernier. Il n'y avait aucune automatisation de ces tests.

# Club étudiant S.O.N.I.A.

Le club S.O.N.I.A. est un club étudiant de l'É.T.S. fondé en 1999 dont le but est la construction d'un robot sous-marin capable d'accomplir un ensemble de tâches définies sans besoin d'intervention humaine. Le club regroupe présentement une vingtaine d'étudiants de diverses concentrations (génie logiciel, génie électrique, génie mécanique, génie de la production automatisée, génie des technologies de l'information) unissant leurs forces afin de produire une plateforme performante pour la compétition annuelle de l'AUVSI et de l'ONR tenues à San Diego.

Le club S.O.N.I.A. tente sans cesse d'être avant-gardiste dans son choix de solutions aux problèmes soulevés par la complexité du projet tout en maintenant des standards de qualité très élevés. C'est dans cette optique que ce nouveau système de test automatisé pour la vision a été réalisé.


**Figure 1 SONIA 2009**

## La vision robotique

Le système de vision robotique dans S.O.N.I.A. se compose de plusieurs détecteurs d'objets qui permettent au système d'intelligence artificielle de reconnaître ses environs. Ces détecteurs sont des chaînes de filtres, suivant le style architectural "pipe and filter" et sont composés de séquences de filtres dont le dernier donne une description de haut niveau de l'image analysée.

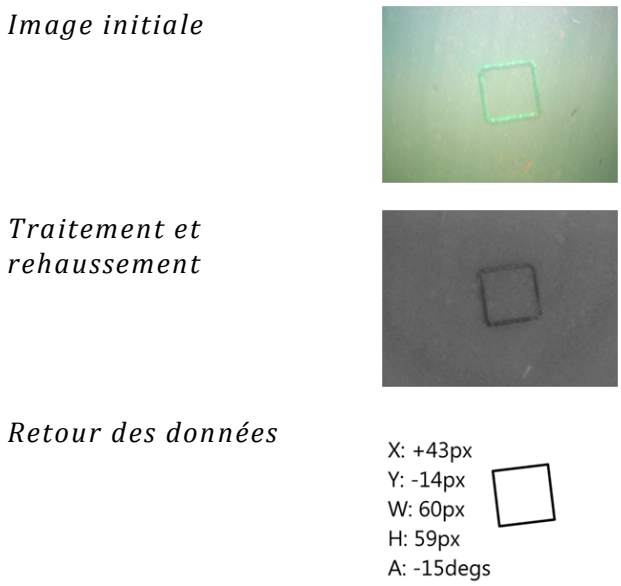


**Figure 2 Traitement d'image**

Ces chaînes de filtres sont représentées par un fichier décrivant les filtres présents dans la chaîne de filtres ainsi que leurs paramètres. Les fichiers de chaînes de filtres sont éditables par les divers outils vision et sont chargés dynamiquement dans le système de vision robotique de S.O.N.I.A. C'est donc pourquoi l'on veut être capable de tester ceux-ci.

Lorsque le véhicule est dans l'eau, celui-ci enregistre des données à l'aide des caméras présentes à bord du sous-marin de façon non compressée. Ceux-ci nous permettent d'avoir un ensemble de données sources pour les tests et de travailler avec exactement les mêmes données qu'une chaîne de filtres travaillerait lors de son exécution à bord du véhicule.

## Problématique

Le club S.O.N.I.A. à de nombreux filtres de vision pour identifier les objets qui sont présentés lors de la compétition auquel il participe. Malheureusement, aucun système n'est en place pour tester la fiabilité des filtres de vision.

En temps normal, la tâche se fait manuellement à l'aide de logiciel que les membres du club ont développé pour aider à la conception et calibration des filtres de vision.

Or, en compétition, cette manière de faire, qui sera détaillée dans l'analyse du système en cours, n'est pas optimale. Elle demande à un développeur qui pourrait travailler sur d'autre à améliorer les filtres, de ce pencher pendant des heures sur des vidéos que le sous-marin à pris lors de test dans précédent dans le basin de compétition. Comme chacune minute, compte, et

comme les membres du club doivent être le plus efficace possible lors de leurs séjour en Californie, il est important d'avoir un moyen fiable, rapide et automatique de tester les filtres de vision.

De plus, il est impossible de reproduire les tests en ce moment. Il n'y a aucun moyen de garder les résultats dans des fichiers. Tout est dans la tête du testeur ce qui fait que trouver des résultats ou revenir sur les résultats aux fins de comparaison entre deux chaînes de filtres, ou deux configurations différentes est impossible. Comme le jugement d'un humain est en cause dans le procédé, les résultats ne sont pas quantitatifs. Les métriques utilisées sont purement qualitatives parce que les résultats sont faits par le testeur. S'il pense que le filtre est bon, il est bon. Avoir un vraiment jugement quantitatif n'est pas faisable et donc la définition entre quels des deux filtres est meilleurs est très dur à réaliser.

## Analyse des besoins

Le club S.O.N.I.A. a besoin d'un logiciel qui aidera à résoudre les problèmes soulevés. Il devra être efficace, facile d'utilisation, les résultats devront être répétables et les métriques quantitatives.

En annexe vous retrouverez le SRS qui a été réalisé par l'équipe pour ce projet. Dans ce document, on retrouve les besoins et requis pour le logiciel. Les plus importants seront étudiés brièvement dans les prochaines lignes.

## Analyse du système en cours

Le processus actuel pour les tests des filtres dans le club S.O.N.I.A. est entièrement manuel et demande la concentration d'une personne pendant de longues minutes. Le processus est relativement simple :

- Chargement de la vidéo
- Application de la chaîne de filtre désirée
- Ajustement des paramètres des filtres
- Visionnement de la vidéo pour déceler les erreurs

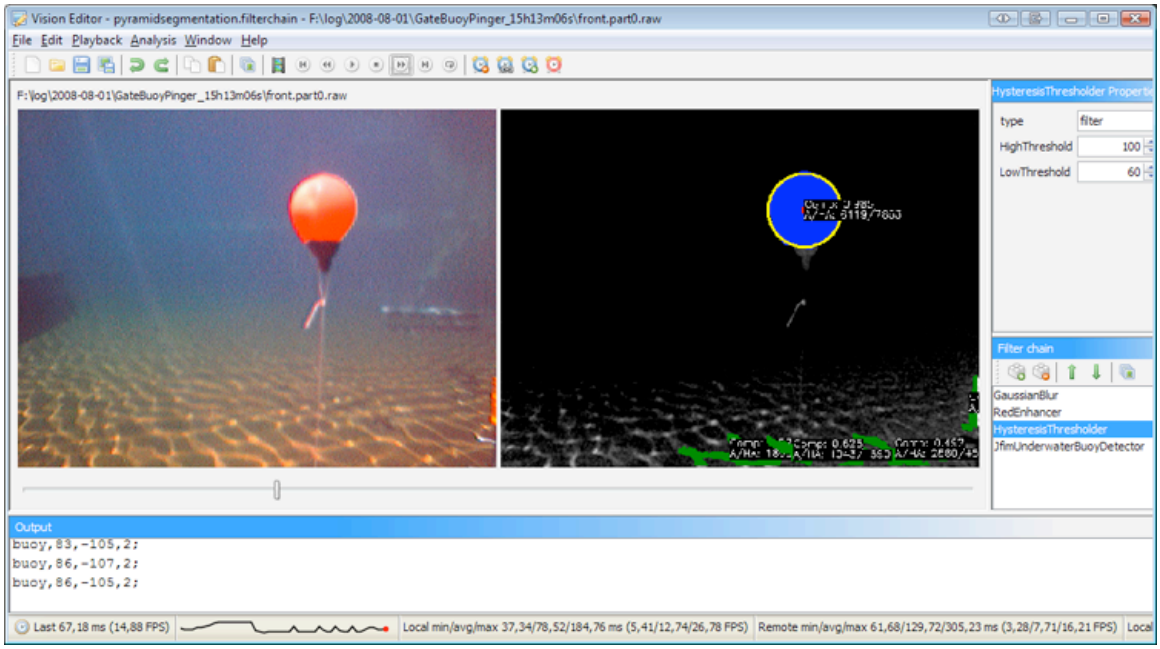


**Figure 3 Vision Editor**

Par contre, ce simple processus laisse place à beaucoup d'interprétation et est une tâche très fastidieuse. Il demande à la personne une attention totale à la vidéo et aux objets qui se trouvent dans la vidéo. Il doit être en mesure de voir les faux positifs et les vrais négatifs. Il doit noter manuellement les endroits où des erreurs ont été trouvées, pour ensuite les corriger et recommencer le processus pour s'assurer qu'il n'a pas commis d'erreurs à d'autres endroits où la chaîne de filtre peut être activée.

Une personne expérimentée peut facilement passer plus de 30 minutes à regarder le même vidéo de test qui ne dure pourtant que 15 minutes. Pendant ce temps, l'équipe perd un membre qui pourrait travailler à améliorer la mission, les filtres de visions, ou effectuer une autre tâche importante pour l'équipe. Ces tests sont d'une importance cruciale. Cependant, si l'équipe peut utiliser son temps plus efficacement, ce peut être la différence qui mènera l'équipe sur le podium.

# Revue de la documentation

L'équipe a dû faire quelques recherches pour mettre au point le Vision Test Tool. Voici la documentation que les membres de l'équipe ont étudiée.

## Librarie Hadoop

Hadoop est une libraire Java, faite pour aider les systèmes distribués. Hadoop remplirait un besoin de l'application qui est de fonctionner efficacement, et ce, sur plusieurs ordinateurs en même temps.

### Patron Stratégie, GoF
Le patron stratégie a été utilisé dans le projet pour la comparaison d'objet. Quelque recherche ont été faite pour s'assurer de la faisabilité de l'idée de l'équipe et de l'utilisation du patron stratégie.

### Documentation de Maven/Nexus
L'équipe a une bonne base de connaissance avec Maven. Cependant, un problème s'est posé avec Nexus. Une recherche a été faite pour déterminer comment régler un problème d'authentification entre un logiciel utilisant Maven et un serveur Nexus protégé. La réponse, dans le settings.xml.

### White
Le logiciel White est un logiciel pour tester les GUI. Nous ne voulons pas tester du GUI mais bien faire de l'automatisation de test pour les filtres de vision du club scientifique S.O.N.I.A.

### Architecture client-serveur
Le patron architectural client-serveur décrit un système en deux modules complètement distincts. D'un côté, on retrouve le client qui pourra traiter les vidéos sur les serveurs d'une ferme de serveur, et de l'autre le serveur qui distribuera les tâches au client.

### JasperReport
Librairie logicielle faite en java qui aide à la confection de rapport. Il est possible de faire des rapports PDF, Excel, Open Office, HTML et Word avec cette librairie.

## Structure du projet

### Échéancier
Comme dans tout projet logiciel, l'échéancier a beaucoup changé depuis la première version. Des tâches ont été déplacées, d'autres ont changé et certaines d'entre elles ont été enlevées.

L'annexe A présente l'échéancier le plus à jour pour le projet du Vision Test Tool. Les changements seront commentés dans cette section.

D'abord, il manquait une partie sur la conception du système. Comme le club S.O.N.I.A. a une rotation de membre dans l'équipe logiciel, il est de mise d'avoir un système qui est facilement maintenable. Donc si la conception n'est pas bien réussie, le logiciel pourrait devenir un enfer à maintenir dans le futur et ce n'est pas le but. Ce logiciel doit aider l'équiper à sauver du temps et non en perdre à essayer de le comprendre et de le maintenir.

Un autre important changement se situe au niveau de la ferme de serveurs qui a été complètement enlevée de l'échéancier parce que cette fonctionnalité ne pouvait pas être réalisable dans le cadre du projet. Par contre, pour remplacer cette tâche une nouvelle est apparue, soit la réalisation du Movie Annotator. Cet outil pourrait faire un excellent projet à lui seul, mais comme il avait déjà été entamé, il y avait moins de travail à réaliser. Nous avons cependant du le compléter et le tester pour s'assurer qu'il fonctionnait bien. Même si le Movie Annotator n'est pas dans la portée initiale du projet, il devait être complété pour assurer le

succès du Vision Test Tool à court, moyen et long terme. Sans cet outil, il est très difficile de faire l'annotation des vidéos.

## Risques

Les mêmes risques ont été présentés dans le rapport d'étape, cependant les priorités ont changé depuis ainsi que l'impact sur le projet. Une mise à jour des risques à donc été faite pour aligner les risques avec cette nouvelle réalité.

| Risque | Probabilité | Impact | Priorité | Mitigation |
|---|---|---|---|---|
| Ne pas avoir accès à une ferme de serveur ou un nuage | Élevé | Faible | Faible | Trouver un terrain d'entente avec le département de génie logiciel pour avoir accès à une ferme de serveur de l'école |
| Manque de temps possible du au stage en industrie d'un membre de l'équipe | Moyen | Moyen | Élevé | Avoir une meilleure organisation de l'horaire du membre en question |
| Logiciel préalable n'est pas terminé | Élevé | Élevé | Élevé | Dois le finir soit même |

### Ne pas avoir accès à une ferme de serveur

Ce risque, qui était au début du projet très important, est devenu secondaire. Faute de temps, une fonctionnalité qui ne sera pas développée dans le cadre du projet spécial est l'utilisation d'un nuage pour l'analyse des données. Par contre, cette fonctionnalité est nécessaire et sera implémentée dans le futur pour améliorer le temps de traitement du Vision Test Tool.

### Manque de temps possible du au stage en industrie d'un membre de l'équipe

Un stage demande évidemment du temps. Comme un membre de l'équipe est en stage pendant la session où le projet spécial est présenté, c'est un risque à considérer. L'impact est cependant moindre que l'estimation initiale. Avec une restructuration du projet, et une meilleure gestion du temps, l'étudiant en stage a été en mesure de faire ce qu'il avait à faire pour que le projet soit un succès.

### Logiciel préalable n'est pas terminé

Ce risque est le plus important de tous et il est celui qui prendra le plus d'importance avec l'avancement du projet. Malheureusement, le club S.O.N.I.A. n'a pas de ressources à assigner pour terminer le Movie Annotator et c'est un logiciel essentiel à la réussite du projet du Vision Test Tool. Sans ce logiciel, l'annotation de vidéo est réalisable, mais beaucoup plus fastidieuse. Il est donc nécessaire que l'équipe termine ce logiciel pour que le Vision Test Tool soit un logiciel utilisable par les membres du club S.O.N.I.A.

## Méthodologie de travail

L'équipe de développement est expérimentée ensemble et comme ils ont réalisé plusieurs projets ensemble, ils connaissent leurs forces et faiblesses. Cependant, pour structurer le projet l'équipe a utilisé des outils logiciels pour améliorer l'efficacité du développement. La présentation des outils utilisée se fera plus formellement dans les lignes suivantes de cette section.

De plus, les techniques de développement Agile ont été utilisées dans le projet avec de courtes itérations, soit entre 2 et 3 semaines. Le but d'avoir de petites itérations est de pouvoir corriger le tir rapidement si quelque chose n'est pas implémenté de la bonne façon ou ne rend pas les utilisateurs du projet heureux.

### Maven

Outil logiciel, similaire à ANT, pour aider à gérer les dépendances et le processus de construction du logiciel. Maven est devenu un outil essentiel dans le développement du Vision Test Tool pour sa simplicité d'utilisation et son efficacité à gérer les dépendances utilisées par l'équipe de développement.

### Team City

Team City est un logiciel d'intégration continue de JetBrains. Il permet à l'équipe de développement d'avoir une phase de construction du logiciel toutes les fois que du code est envoyé à SVN.

### Subversion (SVN)

SVN est l'outil que le club S.O.N.I.A. utilise pour gérer son code source. Comme pour tout projet qui se respecte, avoir une place centralisée ou envoyer du code, et qui est sauvegardé, est essentiel.

### JUnit

Cet outil de test bien connu des développeurs Java sert pour les tests de logique dans le code du Vision Test Tool. En ce moment, les tests faits avec l'aide de JUnit se résument au deux points critique du logiciel, soit les parseurs de données, et les comparateurs des données.

### TRAC

TRAC est un gestionnaire de tâche. S'il y a un défaut au logiciel, une amélioration ou une tâche à accomplir, c'est dans TRAC que ce sera ajouté, car c'est l'outil officiel du club S.O.N.I.A. pour ce genre de travail.

## Architecture du logiciel

L'outil de test étant un outil important dans le processus de développement vision, nous avons conçu une architecture flexible et modulaire afin de permettre des changements ultérieurs à la portée du projet sans nécessiter de refonte majeure de celle-ci.

## Contraintes influant l'architecture

Tel qu'énuméré dans le document SRS, l'architecture est contrainte par trois critères importants de conception:

- CC01 – L'application doit fonctionner sous Windows, Linux et Mac
- CC02 – L'application doit être capable d'utiliser la librairie de filtres existante
- CC03 – L'application doit être intégrable avec les outils de développement vision

Ces trois critères de conception impliquent plusieurs choix lors de l'implémentation. Afin de pouvoir intégrer l'application de test ainsi que d'assurer une portabilité entre Windows, Linux et Mac, Java a été choisi comme langage de programmation. Or, comme la librairie de filtres de vision est une librairie native écrite en C++, l'utilisation d'une couche intermédiaire entre les deux s'avère nécessaire.

Cependant, l'intégration d'une librairie native dans une application Java n'est pas sans problèmes; si lors de l'exécution du code natif celui-ci plante, l'application Java parent est arrêtée sans aucune possibilité de récupération d'erreur. Considérant que l'outil de test de vision est un outil de développement, il est impossible de s'assurer que le code natif sous-jacent est stable, ce qui implique qu'il faut construire le logiciel en fonction qu'il est possible que celui-ci s'arrête de façon subite et imprévue.

```
#
# An unexpected error has been detected by Java Runtime Environment:
#
#   EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x05e56bc3, pid=852,
tid=4308
#
# Java VM: Java HotSpot(TM) Client VM (10.0-b23 mixed mode, sharing
windows-x86)
# Problematic frame:
# C  [cxcore110.dll+0x46bc3]
#
# An error report file with more information is saved as:
# C:\projects\ServeurVision\javainterface\dist\hs_err_pid852.log
#
# If you would like to submit a bug report, please visit:
#   http://java.sun.com/webapps/bugreport/crash.jsp
# The crash happened outside the Java Virtual Machine in native code.
# See problematic frame for where to report the bug.
#
Process finished with exit code 1
```

**Figure 4 Plantage lors d'exécution du code natif**

De plus, l'interface entre la librairie native et Java n'a pas été conçue en fonction de l'utilisation par plusieurs fils d'exécution, ce qui pose un problème sérieux; comme l'exécution des tests

impliquent une charge de travail processeur considérable, il est désirable de vouloir utiliser toutes les ressources fournies par la machine en exploitant tous les cœurs de celle-ci.

Il existe donc deux contraintes techniques qui influent l'architecture:

- Robustesse aux pannes de la librairie vision
- Interface entre Java et la librairie vision n'est pas conçue pour être utilisée par plusieurs fils d'exécution

## Solutions aux contraintes techniques

Afin de résoudre les problèmes techniques, l'architecture de l'outil de test vision est conçue en fonction des limitations techniques impliquées par l'intégration d'une librairie native dans un logiciel Java. L'exécution de code natif est entièrement séparée du processus parent en exécutant les fonctions dans une autre machine virtuelle enfant et en communiquant avec la machine virtuelle enfant via un "pipe." Lorsque celui-ci est fermé, cela indique à la machine virtuelle parent que le processus enfant s'est terminé, que ce soit avec succès ou non. L'application parent envoie donc des commandes à un sous-processus afin d'isoler l'application contre les plantages de la librairie native, redémarrant le processus enfant si nécessaire.

Ces deux processus communiquent avec un protocole relativement simple afin de permettre l'échange d'informations en temps réel entre les deux processus, permettant ainsi au processus parent d'obtenir des informations de progression de façon immédiate, ce qui ne serait pas le cas dans une exécution par lots.

## Aperçu de l'architecture

Afin de faciliter l'implémentation de l'application ainsi que sa compréhension, celle-ci a été divisée en cinq modules: un module d'interface graphique, un module de traitement des images, un module de désérialisation des données, un module de comparaison ainsi qu'un module de gestion d'exécution de tests de vision.

## Module d'interface graphique

Le module d'interface graphique permet d'afficher à l'utilisateur la progression du test ainsi que de l'interrompre en cours d'exécution s'il s'aperçoit que le test sort des critères désirés.

L'interface est fortement inspirée de l'interface graphique de l'outil JUnit, affichant une barre de progression permettant de savoir l'état des tests ainsi qu'une console permettant de voir les erreurs lors de l'exécution des tests. Cependant, celle-ci diffère sur certains points vus les domaines d'application différents. Par exemple, les tests unitaires qui sont exécutés par JUnit ne prennent habituellement que quelques secondes tandis qu'un test avec l'outil de test vision peut prendre plusieurs heures.

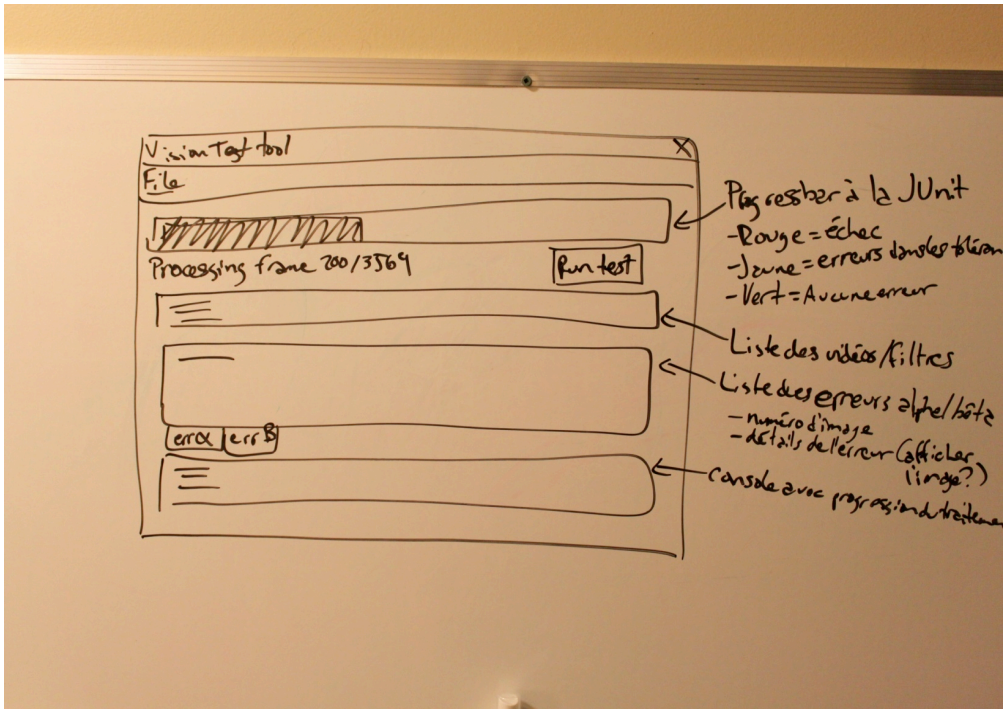


**Figure 5: Prototype Statique**

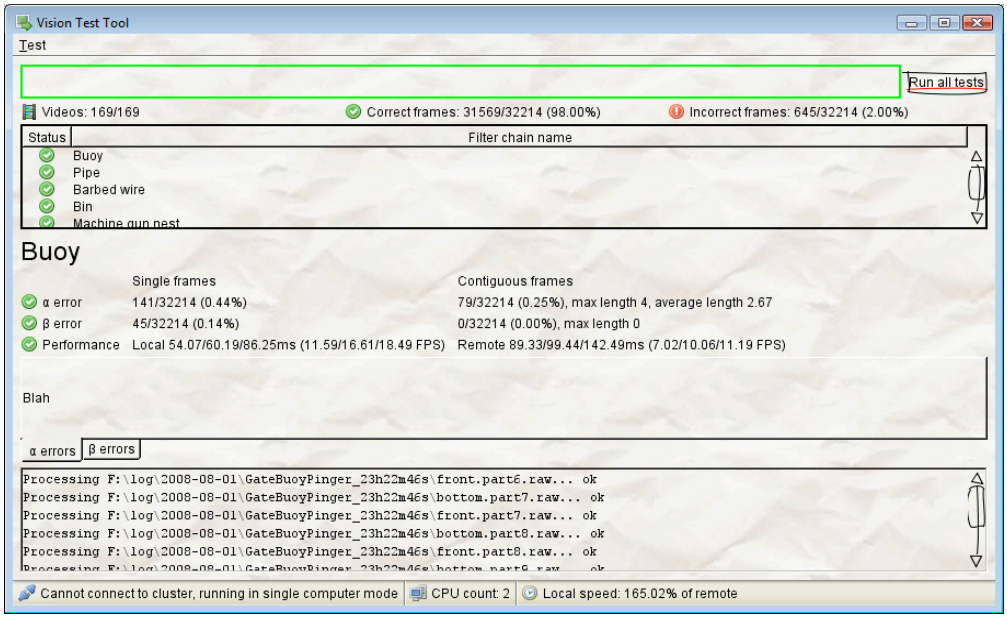


**Figure 6 Prototype 1**

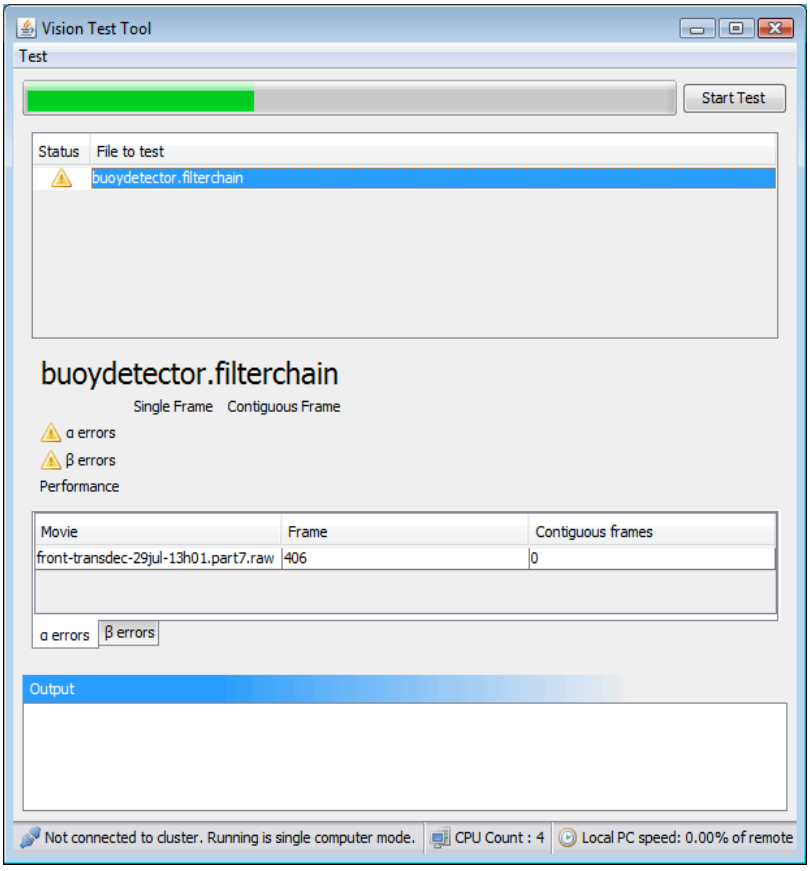


**Figure 7 Version 1.0**

## Module de traitement des images

Le module de traitement des images permet d'exécuter une chaîne de filtres sur un fichier vidéo et de retourner la sortie de la chaîne de filtres au système de test. Celui-ci s'occupe de charger la version appropriée de la librairie de filtres de vision, le ou les fichiers vidéo rattachés au test ainsi que la définition de la chaîne de filtres, exécute ladite chaîne de filtres sur une série d'images et retourne la sortie du filtre de vision pour chacune des images traitées.

Ce module s'occupe aussi de l'isolation entre le processus parent et le processus faisant l'exécution des filtres en offrant une interface qui encapsule la complexité sous-jacente à l'exécution et à la communication interprocessus.

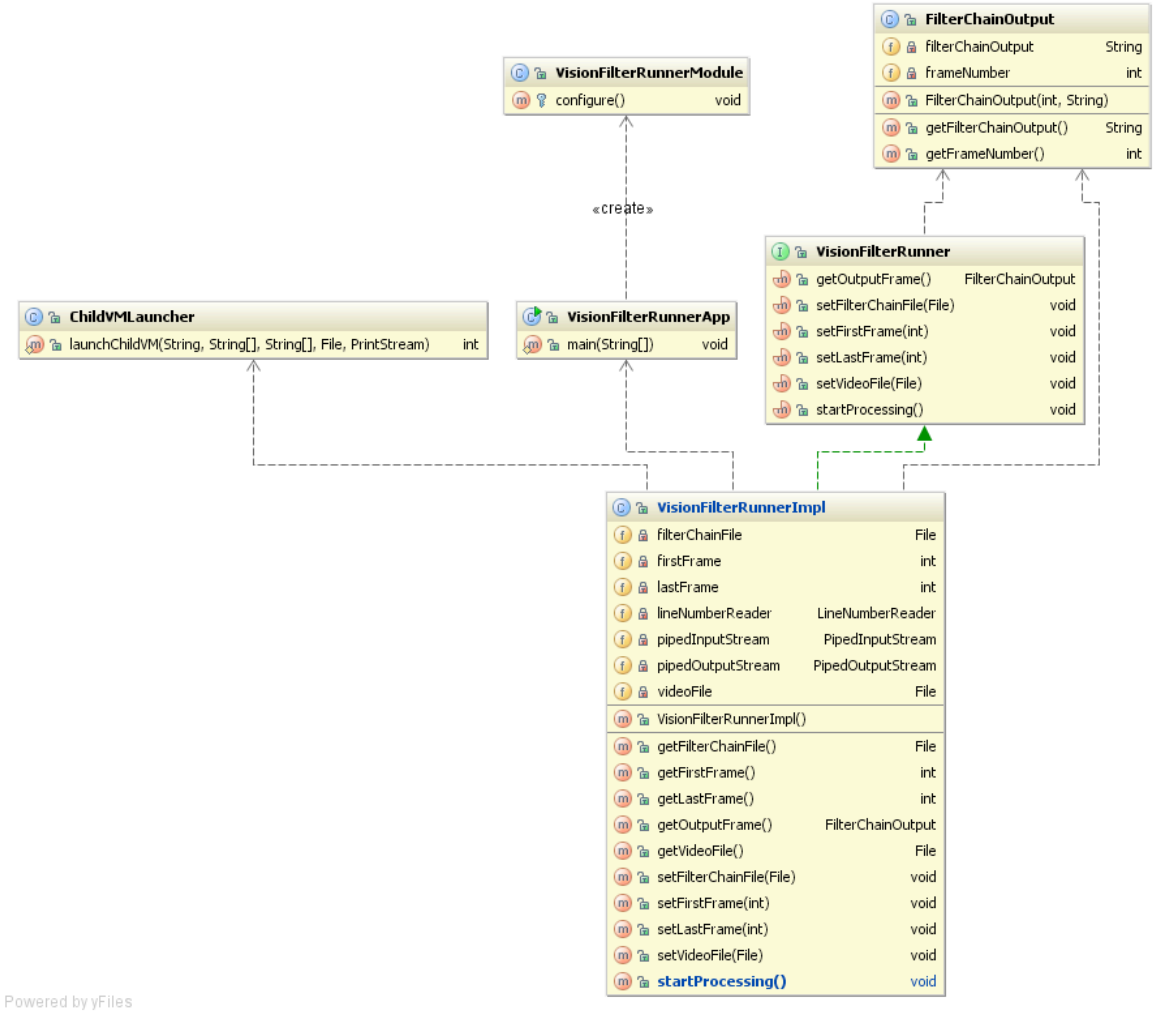


**Figure 8 Diagramme de classes du module**

## Module de désérialisation des données

Le module de désérialisation des données permet de convertir une description haut niveau d'une image en sa représentation Java associée. Celui-ci utilise le système d'annotations introduit dans la version 5 de la plateforme Java afin de permettre aux développeurs d'utiliser une représentation haut niveau du mécanisme de stockage des données sans se préoccuper de la conversion entre les deux formats.

```
@ParsableVisionObject(filterOutputName = "buoy")
public class Buoy implements VisionObject {
    @ParsableVisionField(unit = VisionUnit.UNIT_PIXELS)
    private int x;

    @ParsableVisionField(unit = VisionUnit.UNIT_PIXELS)
    private int y;

    @ParsableVisionField(unit = VisionUnit.UNIT_IMAGE_PERCENTAGE)
    private int size;

    // Accesseurs et autres méthodes sans complexité pertinente
}
```

**Figure 9 Fichier Java annoté**

Le module de désérialisation permet aussi via les annotations rattachées de spécifier les unités reliées aux champs. Ces métainformations sont utilisées par le module de comparaison, tel que spécifié ultérieurement dans ce document.

```
AnnotationVisionParser annotationVisionParser = new
AnnotationVisionParser();
annotationVisionParser.addAnnotatedVisionObjectClass(Buoy.class);
List<VisionObject> visionObjects =
annotationVisionParser.parse("buoy:1,2,3;");
assertThat(visionObjects, is(not(empty())));
assertThat(visionObjects.size(), is(1));
Buoy buoy = new Buoy(1, 2, 3);
assertThat((Buoy) visionObjects.get(0), is(equalTo(buoy)));
```

**Figure 10 Exemple de l'utilisation du module de désérialisation**

## Module de comparaison

Le module de comparaison compare des listes d'objets de vision et ressort les différences entre deux listes d'objets de vision. Comme le domaine de la vision artificielle comporte des marges d'erreur inhérentes au type de problème, une comparaison naïve des chaînes retournées par les filtres de vision serait absurde.

Le module de comparaison permet de spécifier des tolérances pour la comparaison des données retournées par les filtres de vision. Ainsi, il est possible de spécifier des tolérances par unité ainsi que par propriété de façon à définir les métriques d'erreur acceptable de façon précise.

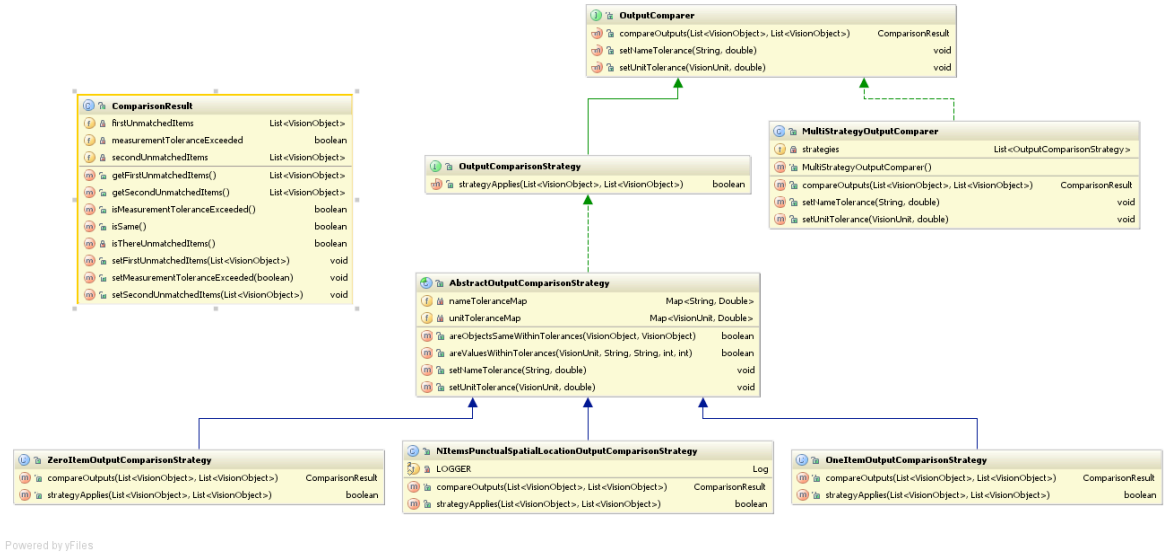


**Figure 11 Diagramme de classes du module de comparaison**

De plus, le module de comparaison possède plusieurs stratégies de comparaison afin de comparer les listes d'objets de vision. Les trois stratégies qui sont présentement dans le logiciel sont:

- Comparaison des listes lorsqu'au moins une des deux listes ne contient aucun objet
- Comparaison des listes lorsque les deux listes ne contiennent qu'un seul objet
- Comparaison des listes avec un nombre quelconque en utilisant leur relation spatiale pour la comparaison

La troisième stratégie de comparaison fonctionne en tentant d'associer les objets en paires et en tentant de minimiser la somme des distances euclidiennes entre les paires d'objets en utilisant un algorithme glouton. De cette façon, il est possible de comparer les objets, et ce, même si le nombre d'objets ne concorde pas à cause de faux positifs.

## Module de gestion de l'exécution des tests

Le module de gestion de l'exécution des tests gère l'exécution des tests, la distribution de la charge de travail sur plusieurs processus et plusieurs machines et offre un modèle à l'interface graphique afin qu'elle puisse afficher la progression du test et obtenir les erreurs en temps réel. Cela permet à l'interface utilisateur de suivre l'architecture MVC.

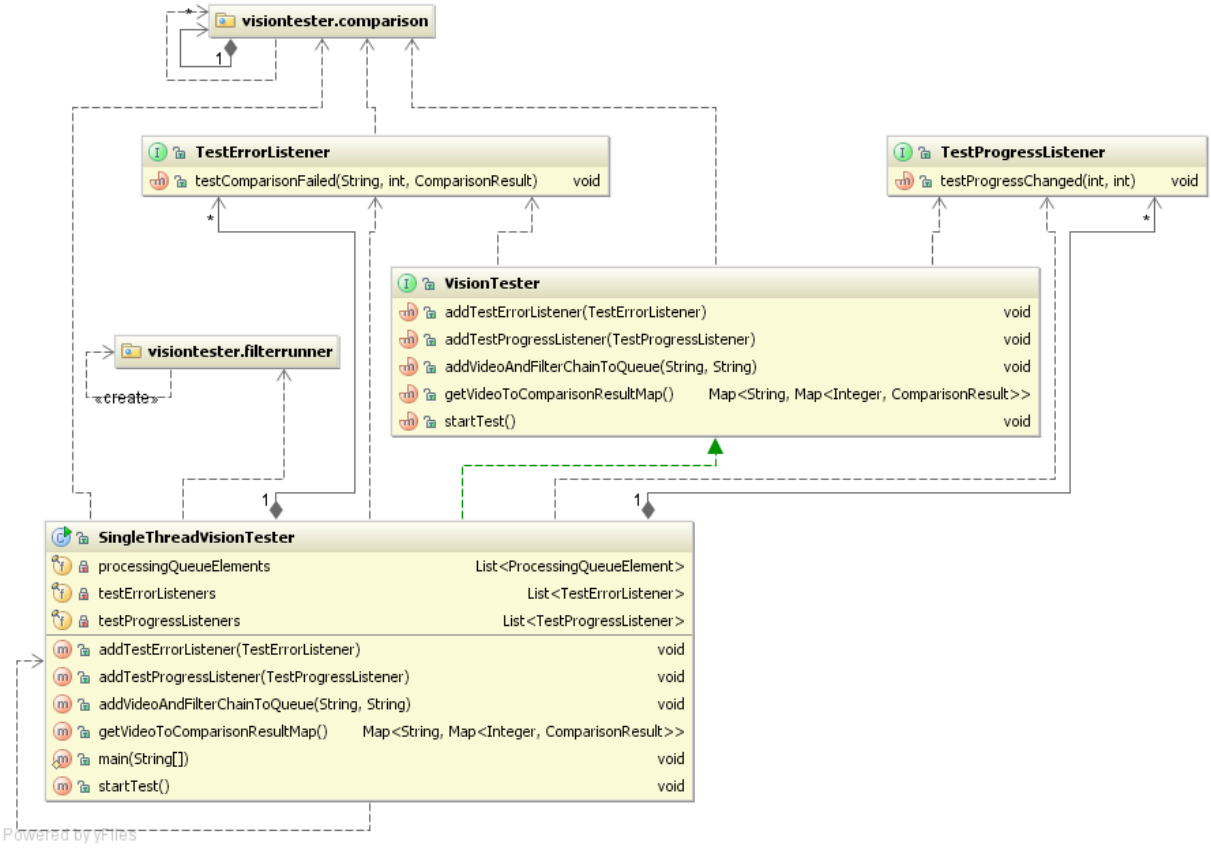


**Figure 12 Diagramme de classes du module de gestion de l'exécution des tests**

De plus, ce module sert de façade, ce qui permet d'abstraire les détails d'implémentation de l'exécution des tests derrière une interface simple, encapsulant ainsi la complexité inhérente à la gestion et la distribution de la charge de travail. De cette façon, il sera facile d'ajouter un mode distribué à l'application de test sans changer l'interface présentée aux clients.

# Solution

La solution évidente est le Vision Test Tool, mais celui-ci comporte plusieurs parties qui vont aider le club S.O.N.I.A. à réaliser ses objectifs. Il sera question de 3 importantes fonctionnalités du logiciel dans cette section.

## Automatisation des tests

Le processus actuel des tests sur les chaînes de filtres de vision du club S.O.N.I.A., comme il a été démontré plus haut dans le document, est entièrement manuel. Il n'y a donc aucune automatisation du processus. Un humain s'assoit devant un ordinateur et juge si la chaîne de filtres performe selon ses propres standards. Et cette démarche prend beaucoup de temps.

Or, avec l'arrivée du Vision Test Tool, il sera possible de non seulement tester un test sans qu'une personne soit présente pour analyser toutes les données, mais aussi de faire plusieurs tests consécutifs.

Quand le logiciel sera bien configuré, que le testeur aura ajouté les tests qu'il désire, le logiciel fera tous ces tests, et sera en mesure de donner les résultats au testeur à la fin de la séquence de test. Peu importe le temps que le test prend, le testeur va être en mesure de travailler sur autre chose et d'être plus efficace.

Il est aussi possible dans le futur d'ajouter des options pour rendre l'application encore plus dynamique. Par exemple, avec l'aide d'outil d'intégration continu, il serait possible de démarrer l'application en ligne de commande, lui donner un fichier de test et laisser l'application suivre son cours. Cette fonctionnalité pourrait aider à automatiser les tests de vision d'une nouvelle façon qui n'est pas possible en ce moment dans le club S.O.N.I.A.

## Ferme de serveurs

Faute de temps, cette fonctionnalité aura été repoussée à une itération future après le projet spécial. Mais elle demeure néanmoins essentielle pour augmenter l'efficacité du logiciel.

Le club S.O.N.I.A. prend beaucoup de vidéo. Toutes les fois que le sous-marin va à l'eau, et qu'une mission est activée, des vidéos sont enregistrés par la caméra avant et celle qui pointe vers le fond du bassin. Les deux caméras enregistrent environ un gigaoctet de données par minute et le club S.O.N.I.A. possède en ce moment à plus d'un téraoctet de vidéo, soit plus de 16 heures de vidéo. Avec l'été qui s'en vient et la période de tests qui commence, on peut s'attendre à au moins 16 autres heures de nouveau vidéo à analyser.

Or, c'est beaucoup demander pour une seule machine. L'idéal serait de pouvoir répartir le travail sur plusieurs ordinateurs, voir un nuage. Cette solution permettra au club S.O.N.I.A. d'être encore plus efficace pour l'analyse de ses vidéos, et les résultats des tests arriveraient beaucoup plus rapidement.

Il y a cependant quelques problèmes, le plus important est sans doute comment transférer toutes ces données quand le club est sur le site de compétition, ou à l'hôtel à San Diego. Transférer par internet sur des connexions instables et lentes quelques gigaoctets de données va prendre un certain temps. Il serait peut-être judicieux d'utiliser un « mini nuage » formé avec les ordinateurs portables des membres du club qui sont à San Diego.

Peu importe la solution employée pour résoudre le problème de transfert de données, cette fonctionnalité est importante pour être en mesure d'étudier les données et les résultats des chaînes de filtres rapidement.

## Rapport des tests

Comme tout outil de test, il est important d'avoir une forme de rétroaction avec le logiciel. Les rapports des tests seront utilisés à cette fin. Il est aussi important d'avoir une forme d'archives pour les tests. Ces archives peuvent servir à comparer les chaînes de filtres et déterminer lequel est le plus apte à faire une certaine tâche.

Les rapports de test doivent contenir quelques informations. Le nom du test, les vidéos, le nom de la chaîne de filtres testé et le type d'erreur, et des statistiques utiles pour identifier la performance de la chaîne de filtres. Ces statistiques peuvent être la moyenne d'erreur par vidéo, la performance de la chaîne de filtre, soit combien d'images par seconde il est capable de traiter, le type d'erreur, la tolérance aux pannes, entre autres.

En ce moment, le rapport est un simple fichier texte qui contient quelques informations, comme le nom de la vidéo, le nom du test et les erreurs. Rien sur les statistiques qui sont intéressantes à avoir, mais qui ne sont pas absolument nécessaires. Les statistiques vont venir avec les futures améliorations apportées au logiciel.

Un autre projet futur qui touche les rapports des tests, seraient de pouvoir les comparer ensemble et de rapidement déterminer lesquels des tests sont les plus performants. Bien entendu dans ce type de test, il est important de s'assurer que la comparaison se fait sur le même vidéo, et que les deux chaînes de filtres, ou les deux configurations différentes de propriété testent la même chose.

Dans l'avenir il sera important de faire un rapport un peu plus professionnel avec un outil tel JasperReport. Les rapports seront ainsi plus faciles à présenter qu'un simple fichier texte et l'information sera mieux répartie sur l'ensemble d'une page. De plus, même si l'aspect esthétique du rapport n'a pas une grande importance, il est toujours intéressant d'avoir quelque chose de beau à regarder qui représente bien le club S.O.N.I.A. et son professionnalisme.

# Discussion

## Communication

Une personne de l'équipe est en stage, l'autre à l'école, il est donc nécessaire d'avoir de bonne mécanique de communication. Jean-François et Jonathan sont habitués de travailler ensemble. Ils connaissent leurs forces et leurs faiblesses ce qui aide grandement au travail en équipe.

Cependant, il est important d'avoir une bonne méthode de travail et des bonnes pratiques de développement pour être en mesure d'arriver aux fins du projet. L'utilisation d'outils tels que SVN pour le partage du code et des documents est donc une nécessité pour l'équipe.

L'équipe s'est rencontrée à quelques reprises pour être en mesure de faire avancer les tâches plus rapidement. Le téléphone, la messagerie instantanée (Google Talk) et les courriels ont été des moyens de communication utilisée à profusion lors du développement. Ce sont des moyens rapides et efficaces pour consolider l'information pertinente au projet. De plus, ce sont des traces écrites si jamais une mésentente survient.

Dans un projet comme celui-ci, l'efficacité du travail à distance et de la communication est la clef du succès. Sans les bonnes méthodes, il est pratiquement impossible de réussir.

## Itération future

Il reste encore beaucoup de travail à accomplir sur le Vision Test Tool. Les améliorations proposées dans le cadre de futures itérations serviront à enrichir le logiciel et le rendre encore plus efficace est utile au club S.O.N.I.A.

### Hadoop, ou le nuage

Hadoop est une libraire pour aider la gestion d'application distribuée. Or, le point a été soulevé auparavant, il serait extrêmement pratique pour le Vision Test Tool de distribuer l'analyse des vidéos sur plusieurs ordinateurs. Cette avenue rendrait le logiciel plus rapide, ce qui donnerait de la rétroaction aux testeurs et développeurs plus rapidement. Ainsi, les améliorations aux chaînes de filtres pourraient être faites plus efficacement et du temps serait gagné.

### Intégration dans les outils de vision

Un autre aspect qui permettra d'être plus efficace lors du développement des chaînes de filtres serait d'avoir le Vision Test Tool intégré dans le Vision Editor. Ainsi, il serait possible de rouler les tests à partir d'un seul bouton. Un peu comme l'intégration de JUnit et d'Eclipse pour tout code Java. Cette amélioration au logiciel permettra de ne jamais quitter l'espace de développement, ce qui réduirait le changement de contexte du développeur, ce qui est dispendieux pour l'être humain.

### Graphique des métriques de performance

Ce qui est recherché dans cette amélioration est de pouvoir juger la performance des chaînes de filtres de vision dans le temps. L'utilité de cette fonctionnalité est de pouvoir juger, numériquement, quelle chaîne de filtres est la plus performante pour certaines conditions et certains obstacles. Comme plusieurs chaînes de filtres peuvent détecter les mêmes objets, il est intéressant de savoir lequel est le plus performant dans quel cas, surtout lors de la compétition.

# Conclusion

Le Vision Test Tool répond à un besoin criant du club S.O.N.I.A., soit celui d'avoir un outil logiciel capable de tester les chaînes de filtres de vision pour aider le sous-marin à détecter les objets et les identifier de quelques manières que ce soit. Le nouvel outil de la suite logicielle S.O.N.I.A. aidera aux tests et libérera du temps à un membre de l'équipe de compétition pour qu'il soit en mesure d'aider l'équipe avec des tâches aussi importantes, mais qui ne peuvent pas être automatisées.

Le Vision Test Tool devra voir beaucoup d'améliorations dans le futur pour le rendre encore plus efficace et plus agréable à travailler avec. Le support pour un nuage, l'amélioration pour les rapports, l'intégration avec les outils de vision déjà existants et le démarrage automatique ne sont que quelques exemples d'améliorations possibles qui seront destinées à rendre l'outil toujours plus efficace.

Avec les méthodologies employées lors du projet, il sera facile pour les nouveaux développeurs de rentrer dans le projet sans trop de difficultés et d'apporter ces améliorations.

Le Vision Test Tool devrait être utilisé au courant de l'été, et ce, pour plusieurs années à venir pour rendre les chaînes de filtres encore plus efficaces qu'à l'heure actuelle et amener les processus de test du club S.O.N.I.A. vers de nouveaux sommets.

# Références

Cours LOG 792 Projets de fin d'études en génie 2010. Site du projet de fin d'études en génie. ]
http://cours.logti.etsmtl.ca/log792.
Consulté le 8 avril 2010.

Site Wikipedia sur l'architecture client-serveur
http://en.wikipedia.org/wiki/Client-server
Consulté le 27 janvier 2010

Site de Hadoop
http://hadoop.apache.org/
Consulté le 27 janvier 2010

Site de White
http://www.codeplex.com/white
Consulté le 27 janvier 2010

Site de JasperReport
http://jasperforge.org/projects/jasperreports
Consulté le 6 avril 2010

Clements, Paul et al. Documenting Software Architectures: Views and Beyond. Addison-Wesley Professional, 2002, 560 pages.

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J. Desing Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995, 393 p.