Université du Québec
École de technologie supérieure

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS LOG792 PROJET DE FIN D'ÉTUDES EN GÉNIE LOGICIEL

SENTINELLE PAINTBALL
CONTRÔLEUR DU FUSIL PAINTBALL

SÉBASTIEN LUSSIER
LUSS03107904

JEAN-FRANÇOIS POMERLEAU
POMJ06078209

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur superviseur

Alain April

MONTRÉAL, 23 DÉCEMBRE 2011
AUTOMNE 2011

Remerciements

On aimerait remercier Christian Larose, pour le temps qu'il a bien voulu nous accorder lorsqu'on avait des problèmes d'ordre techniques avec le contrôleur de gâchette. Un gros merci aussi à Patrice Dion pour son support technique. Merci aussi à Alain April qui a parti ce super projet et sans qui il ne pourrait y avoir de continuité. Merci aussi de nous avoir permis de se joindre à ce magnifique projet et nous donner la possibilité de le continuer jusqu'au bout.

Table des matières

Remerciements 2

Table des matières 3

Liste des figures 6

Liste des abréviations, sigles et acronymes 7

Liste des symboles et unités de mesure 8

1. Introduction 9

2. Fonctionnalités 10

2.1 Contrôle des moteurs 10

2.2 Contrôle de gâchette 10

2.3 Réception des commandes 10

2.4 Gestion des commandes reçues 10

2.5 Renvois de confirmation 10

2.6 Priorisation des tâches 10

2.7 Système de diagnostic 10

2.8 Indépendance des tâches des contrôleurs 11

2.9 Récupération de panne 11

3. Technologies 12

3.1 Carte EVK1100 AVR32 12

3.2 Contrôleur de moteurs MD49 12

3.3 Moteurs EMG49 12

3.4 Contrôleur de gâchette Pololu 13

3.5 Librairie FreeRTOS 13

3.6 Programmation C 13
4. Architectures 14
4.1 Globale 14
4.2 Matérielle 15
4.3 Logicielle 16
5. Implémentation 18
5.1 Affichage LCD 18
5.1.1 Objectifs 18
5.1.2 Développement 19
5.1.3 Adaptation Multithread 20
5.1.4 Développement futur 20
5.2 Communication Réseau 21
5.2.1 Objectifs 21
5.2.2 Implémentation 21
5.2.3 Développement futur 22
5.3 Récupération de panne 23
5.3.1 Objectifs 23
5.3.2 Développement futur 24
5.4 Contrôle des moteurs 25
5.4.1 Objectifs 25
5.4.2 Problèmes rencontrés 25
5.4.3 Recommandations 27
5.5 Contrôle de la gâchette 28
5.5.1 Alimentation 28
5.5.2 Signaux de contrôle 28
5.5.3 Indicateurs LED 29

5.5.4 Protocole de communication disponible 29
5.5.5 Configuration 30
5.5.6 Utilisation 30
5.5.7 Développement futur 30
5.5.8 Problèmes rencontrés 31
6. Conclusion 33
7. Références 34

Liste des figures

Figure 1 - Carte de développement EVK1100 12
Figure 2 - Contrôleur de moteur MD49 12
Figure 3 - Moteurs EMG49 12
Figure 4 - Contrôleur de gâchette Pololu 13
Figure 5 - Schéma globale de l'architecture 14
Figure 6 - Schéma détaillé de notre architecture matérielle 15
Figure 7 - Architecture logicielle 16
Figure 8 - Présentation du LCD sur l'EVK1100 18
Figure 9 – Illustration de l'opération de défilement de la pile d'affichage 19
Figure 10 - Diagramme de séquence de récupération de panne 24
Figure 11 - Graphique sur l'accélération des moteurs 26
Figure 12 - Application « Serial Port Monitor » 28
Figure 13 - Application « Pololu Serial Transmitter » 29
Figure 14 - Interprétation du signal envoyé par le contrôleur de gâchette 30

Liste des abréviations, sigles et acronymes

SVN	*Subversion, un logiciel de gestion de versions*
LCD	*Liquid Crystal Display, soit « écran à cristaux liquides »*
ETS	*École de technologie supérieure*
LOG	*Génie Logiciel*
PC	*Personal Computer, soit « ordinateur personnel »*
GPL	*General Public License, soit « licence publique générale GNU »*
RISC	*Reduced Instruction-Set Computer, soit « microprocesseur à jeu d'instruction réduit »*
AVR	*Terme utilisé par Atmel pour désigner le cœur du processeur et la famille de microcontrôleurs les implémentant.*
PWM	*Pulse Width Modulation*

Liste des symboles et unités de mesure

V	Volt
A	Ampère
ms	milliseconde
rpm	rotation par minute

1. Introduction

Lors de la session d'été 2011 dans le cadre du cours LOG 550 – Programmation en temps réel, Christian Larose nous a approchés pour nous joindre au projet de tourelle de paintball. Le défi étant très intéressant nous a convaincus, Sébastien Lussier et moi-même, de nous joindre à l'équipe. L'objectif initial était de compléter le module de contrôle des moteurs ainsi que l'ajout du module de contrôle de la gâchette. Par contre, pour répondre à des besoins qui se sont vite manifestés, plusieurs autres objectifs se sont rajoutés par la suite. On nous a donc remis un premier prototype fonctionnel qui avait été réalisé par des étudiants de session précédente. Comme il s'agissait de modifier et d'ajouter du code dans une application temps réel impliquant de l'électronique, il fallait s'attendre à ce que la courbe d'apprentissage soit assez élevée.

Pour faciliter le développement et la maintenance du logiciel de contrôle de la tourelle, on a opté pour un développement itératif utilisant la méthode agile. De plus, une documentation pouvant être générée par le logiciel libre Doxygen a été ajoutée dans le code.

2. Fonctionnalités

2.1 Contrôle des moteurs

Le système doit s'assurer de bien envoyé des commandes aux moteurs et que celles-ci soient bel et bien exécutées.

2.2 Contrôle de gâchette

Lorsque le système reçoit la commande de tirer une ou plusieurs balles, celui-ci doit envoyer la bonne commande au contrôleur pour activer la gâchette.

2.3 Réception des commandes

Le système permet de recevoir des commandes de l'ordinateur exécutant le logiciel de vision et les achemine aux différentes parties connectées aux contrôleurs.

2.4 Gestion des commandes reçues

Le système permet une gestion de différents codes de commandes qui peuvent être effectuées. Des codes d'initialisations, pour tirer, pour déplacer le fusil, etc. Selon les commandes, le contrôleur peut les prioriser et les acheminer aux bonnes parties.

2.5 Renvois de confirmation

Les contrôleurs sont en mesure de répondre au système en confirmant la réception et le traitement des commandes pour assurer de son bon fonctionnement au système via une communication bidirectionnelle.

2.6 Priorisation des tâches

Le système est capable de prioriser des tâches. Par exemple, si une commande est reçue, son exécution devra être traitée de façon prioritaire sur l'affichage du contrôleur ou l'envoi de confirmation de la réception de cette commande.

2.7 Système de diagnostic

Via l'affichage sur l'écran LCD du contrôleur et/ou les envois de confirmations, le système est en mesure d'envoyer un « feedback » au système et à l'utilisateur dans

le cas où des problèmes surviendraient avec une des parties du contrôle des moteurs.

2.8 Indépendance des tâches des contrôleurs

Chaque contrôleur est capable d'être indépendant et autonome le plus possible pour éviter des cas de « dead lock » du système complet si la communication avec un des modules externes tels que celui des moteurs ou de la gâchette est perdue pour être capable d'en informer l'utilisateur.

2.9 Récupération de panne

Le système est en mesure de récupérer automatiquement des pannes causées par des mauvaises connexions des modules externes. Il doit aussi identifier rapidement la source du problème et permettre un diagnostic rapide des modules externes.[1]

[1] Une partie de la section « Fonctionnalités » a été reprise du rapport d'étape

3. Technologies

Figure 1 - Carte de développement EVK1100

3.1 Carte EVK1100 AVR32

Carte de développement EVK1100 avec le microcontrôleur AVR32 UC3A0512.

Fourni par l'ETS pour les laboratoires du cours LOG 550 et utilisé dans le cadre du projet de la tourelle de paintball.

Figure 2 - Contrôleur de moteur MD49

3.2 Contrôleur de moteurs MD49

Le Devantech MD49 24V 5A Double H-Bridge Motor Driver est conçu pour fonctionner avec les moteurs à engrenages EMG49 de Devantech. Il dispose de deux modes de fonctionnement, le contrôle individuel des moteurs ou la capacité d'envoyer une vitesse et une commande de tour.

Figure 3 - Moteurs EMG49

3.3 Moteurs EMG49

Il s'agit d'un moteur 24V à engrenages entièrement équipé avec des encodeurs et un réducteur 49:1.

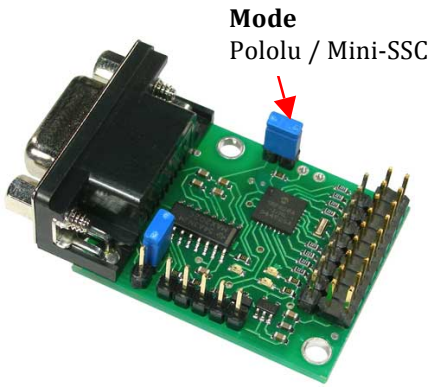

Figure 4 - Contrôleur de gâchette Pololu

3.4 Contrôleur de gâchette Pololu

Permets de contrôler jusqu'à huit servomoteurs avec presque n'importe quel contrôleur de robot ou ordinateur. Il possède deux protocoles de communication : Pololu et Mini-SSC.

3.5 Librairie FreeRTOS

FreeRTOS est un noyau de système d'exploitation temps réel pour microcontrôleur. Il est distribué sous la licence libre GPL avec une exception facultative. L'exception permet aux utilisateurs avec du code source propriétaire de garder leur code source fermé/secret tout en maintenant le noyau lui-même en logiciel libre (open source), ce qui facilite l'utilisation de FreeRTOS dans des applications propriétaires[2]

3.6 Programmation C

Le C est un langage de programmation impératif conçu pour la programmation système. Ces caractéristiques en font un langage privilégié quand on cherche à maîtriser les ressources utilisées, le langage machine généré par les compilateurs étant relativement prévisible et parfois même optimal sur les machines d'architecture RISC à grand nombre de registres. Ce langage est donc extrêmement utilisé dans des domaines comme la programmation embarquée sur microcontrôleurs, les calculs intensifs, l'écriture de systèmes d'exploitation et tous les modules où la rapidité de traitement est importante. Il constitue une bonne alternative au langage d'assemblage dans ces domaines, avec les avantages d'une syntaxe plus expressive et de la portabilité du code source.[3]

[2] http://fr.wikipedia.org/wiki/FreeRTOS
[3] http://fr.wikipedia.org/wiki/C_(langage)

4. Architectures

4.1 Globale

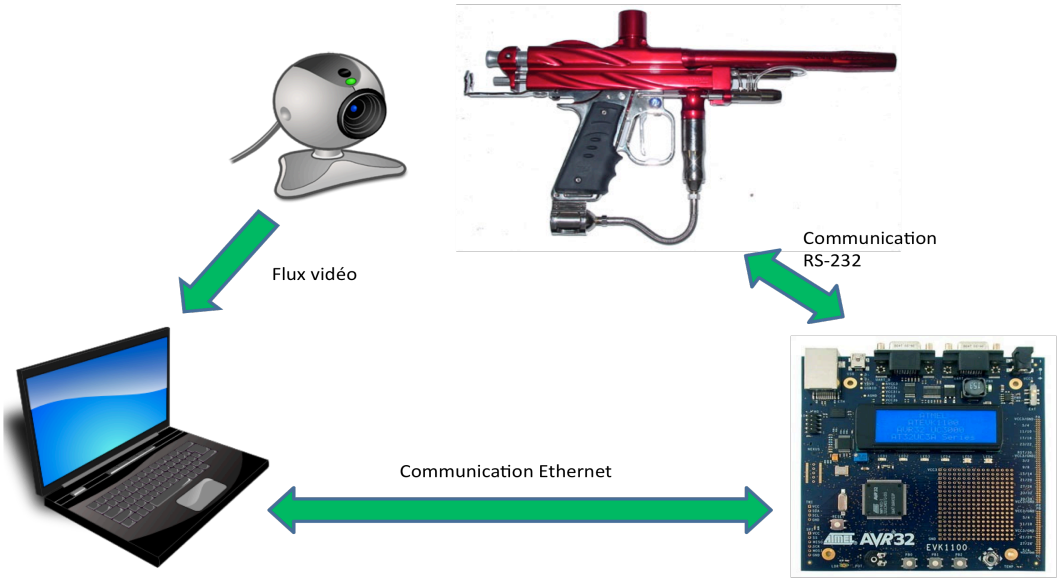

Figure 5 - Schéma global de l'architecture

L'architecture globale du système se divise en 4 sections principales qui effectuent chacune leur fonction spécifique.

1. La caméra est la vue du système, elle permet de fournir les images qui contiennent les cibles potentielles du système qui seront analysées par l'ordinateur.
2. L'ordinateur est la partie du système qui reçoit les images de la caméra et qui exécute le programme qui fait l'analyse des images. De cette analyse découlent des cibles à éliminer. L'application doit donc maintenant déterminer la position des cibles et envoyer des coordonnées au contrôleur.
3. Le contrôleur est responsable de recevoir les commandes générées par l'application sur l'ordinateur, de faire déplacer les moteurs pour cibler la position désirée par le programme et de tirer lorsque désiré. Pour ce faire, le contrôleur est connecté à des modules de contrôle externes qui active des moteurs et solénoïde installé sur le fusil paintball.
4. Le fusil paintball est installé sur un trépied qui possède un engrenage pour permettre les mouvements du fusil et le solénoïde est installé sur la gâchette du fusil paintball pour lui permettre de tirer lorsque commandé par le contrôleur.

4.2 Matérielle

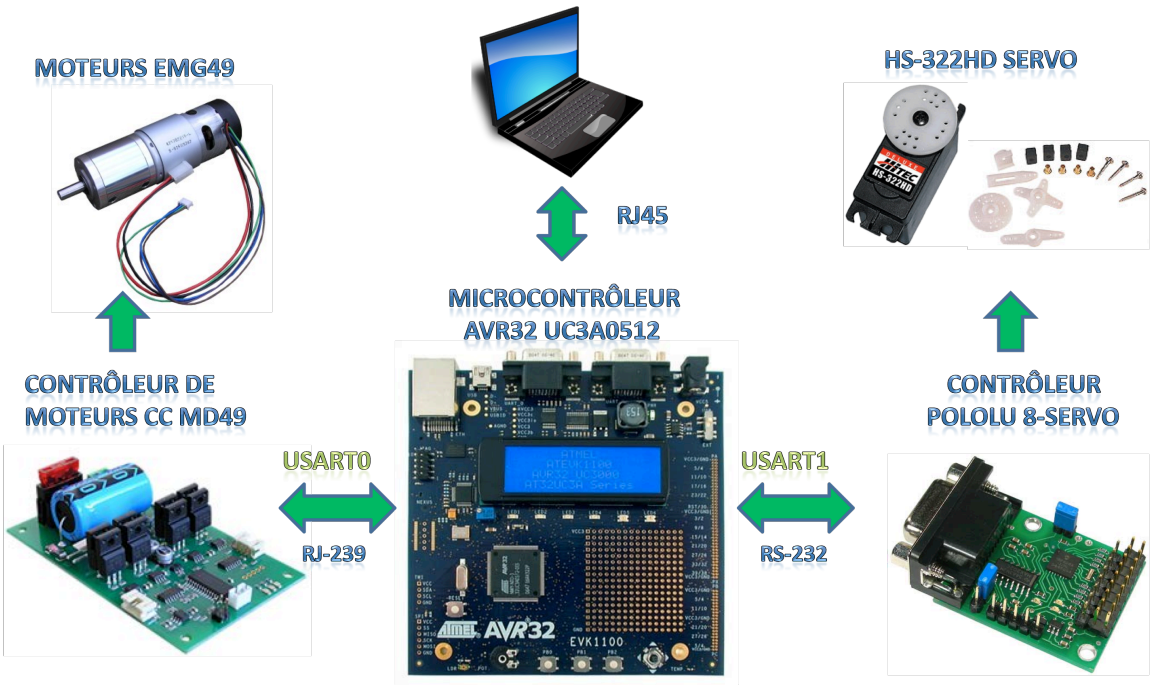

Figure 6 - Schéma détaillé de notre architecture matérielle

L'architecture matérielle du contrôleur est composée de plusieurs composants, car le microcontrôleur UC3A0512 ne contrôle pas lui-même les périphériques extérieurs tels que les moteurs et la gâchette. Il serait surement possible de le faire, mais vu que les gens qui développent le projet ont une formation en logiciel, des modules externes sont utilisés pour simplifier le développement.

- Microcontrôleur UC3A0512 : il est le cœur du système de contrôle du fusil. Ce microcontrôleur est responsable de l'exécution du programme développé avec la librairie FreeRTOS en programmation C. Ce programme est celui qui va permettre d'interpréter les commandes reçues du programme de vision et de distribuer les actions aux bons contrôleurs.
- Contrôleur MD49 : le contrôleur de moteur est responsable de lancer et d'arrêter les mouvements des moteurs EMG49 et qui lit le positionnement des arbres des moteurs.
- Contrôleur Pololu 8-servo : ce contrôleur de servo est utilisé pour envoyer un signal PWM qui permet l'activation du servo HS-322HD pour lui faire une rotation de 180 degrés.

4.3 Logicielle

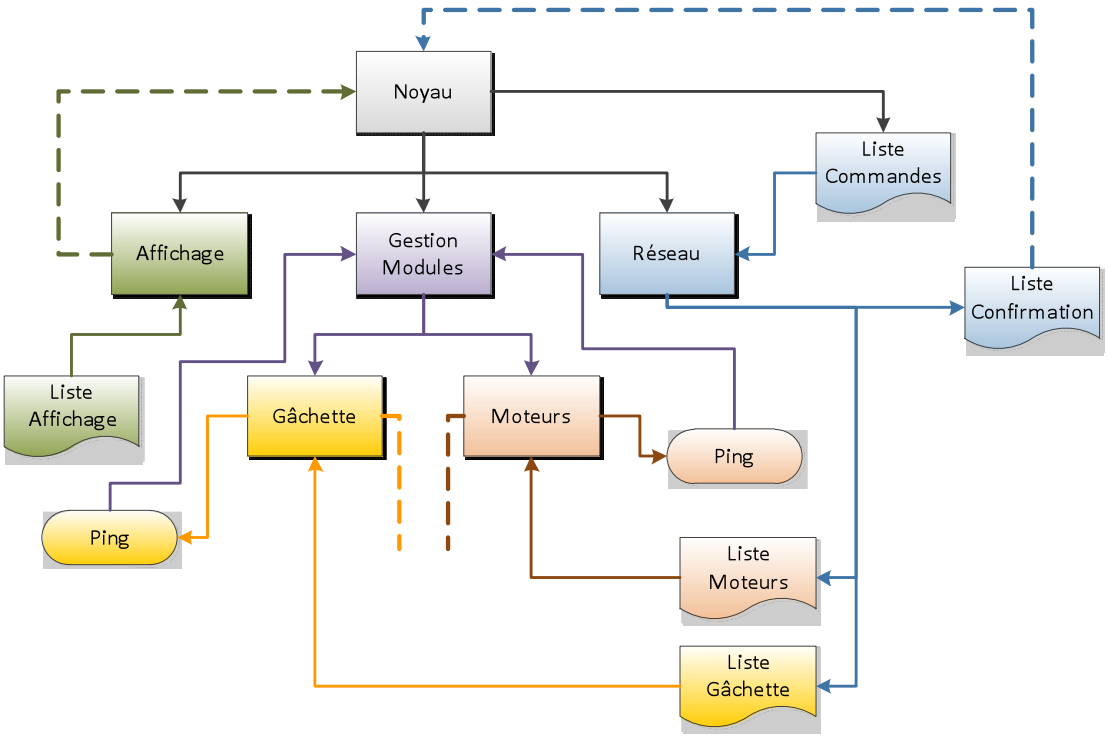

Figure 7 - Architecture logicielle

L'architecture logicielle est composée de plusieurs tâches qui roulent dans un environnement multithread, dans le but de bien mettre en valeur l'architecture, seules les relations importantes sont illustrées dans le schéma ci-haut.

Pour commencer, le noyau est la partie principale où tout se serait déroulé dans une architecture avec un seul thread. Cette partie est responsable de faire initialisation des connecteurs matériels, des interruptions et de créer les autres tâches qui vont être exécuté en parallèle.

Dans les autres tâches, la première est celle qui gère l'affichage sur l'écran LCD du microcontrôleur UC3A0512. Cette tâche est responsable de l'affichage de tous les messages désirés avec une certaine gestion, elle sera décrite plus en détail dans la section d'implémentation.

Une autre tâche très importante est celle de la communication réseau. Cette dernière est responsable de la réception et l'envoi de toutes les commandes entre le microcontrôleur et le logiciel de vision. Cette tâche est aussi responsable de décoder les commandes reçues et de les faire parvenir aux destinataires désirés.

La dernière tâche est divisée en 3 sous-tâches. C'est la tâche de gestion des modules. La première sous-tâche est responsable de créer et effacer au besoin les tâches responsables des moteurs et de la gâchette. Cette architecture non orthodoxe est justifiée par le module de gestion des pannes et le fonctionnement sera expliqué plus en détail dans la section d'implémentation.

5. Implémentation

5.1 Affichage LCD

5.1.1 Objectifs

Afin d'aider à plusieurs étapes, que ce soit le développement, la maintenance ou même l'apprentissage, l'affichage sur l'écran LCD c'est avéré primordial. L'affichage est le moyen le plus efficace et le plus complet d'obtenir des informations sur les réactions du microcontrôleur d'AVR.

De plus, l'utilisation de l'écran LCD a permis de libérer le 2e port série qui est nécessaire pour connecter le Pololu 8-servo qui actionne la gâchette. Le développement de cette partie n'est pas parti de zéro, car dans l'outil de développement AVR Studio, une librairie de base est disponible pour permettre l'affichage.

Figure 8 - Présentation du LCD sur l'EVK1100

5.1.2 Développement

Cette librairie ne répondait pas complètement aux besoins de l'application, donc certains développements ont été faits pour mieux encadrer les fonctionnalités disponibles. Pour commencer, la librairie elle-même a été modifiée pour faciliter l'affichage et éliminer les comportements erratiques dont elle faisait preuve. Cette modification est l'implémentation d'une limite qui tronque les chaines de caractères qui dépassent la limite de 20 caractères d'affichage de l'écran. Cette modification a été faite dans la librairie vu que le niveau de difficulté était plus grand sans offrir aucun gain.

Une autre modification qui a été faite est un « wrappeur » de la librairie d'affichage d'AVR pour permettre d'avoir des interactions simplifiées pour le reste de l'application. Le « wrappeur » comprend une pile interne qui contient les 4 lignes d'affichages de l'écran et qui gère le défilement des chaines. Lorsque le maximum de chaînes est atteint, la pile écrase la première chaîne pour faire de la place à la nouvelle chaîne qui arrive et ainsi simuler un défilement à l'écran.

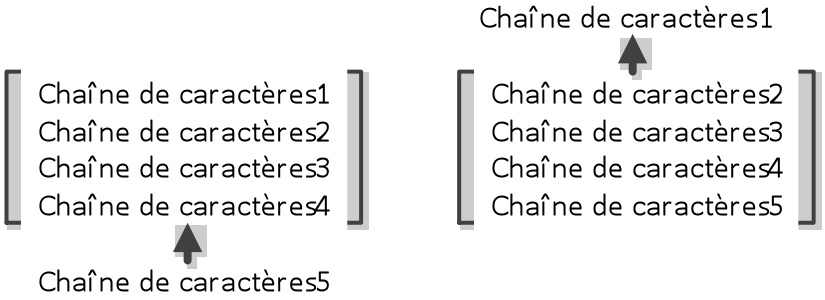

Figure 9 – Illustration de l'opération de défilement de la pile d'affichage

Une autre fonctionnalité qui a été implémentée est la possibilité de vider le contenu au complet de l'écran via une variable globale. Lorsque la variable est vraie, le contenu est effacé au moment où une nouvelle chaîne de caractère est affichée.

Une autre méthode a été ajoutée au « wrappeur » pour afficher des chaînes de caractères paramétrables. Cette méthode a pour but d'aider surtout au niveau du déverminage de

l'application pour permettre l'affichage rapide de plusieurs paramètres numériques tels que les coordonnées des moteurs ou l'adresse IP du réseau.

5.1.3 Adaptation Multithread

Le développement de l'affichage a été le premier développement fait sur le projet, il a donc fallu l'adapter pour l'architecture multithread et cette adaptation a été réalisée par l'implémentation d'une pile de type FreeRTOS. Cette pile globale permet aux autres tâches d'acheminer tous les messages qui doivent être affichés à l'écran par la tâche.

5.1.4 Développement futur

La section de l'affichage s'est avérée très pratique pour plusieurs parties du développement du projet, mais des améliorations très pratiques pourraient être envisageables pour bonifier l'offre de service de cette tâche.

Dans les améliorations, il serait intéressant d'avoir plusieurs piles de message avec des priorités différentes. Cette fonctionnalité pourrait permettre d'avoir une pile prioritaire pour les messages d'erreurs, une pile pour l'affichage régulier et une pile pour l'affichage de messages de déverminage spécialisés. Dans le cas des messages d'erreurs, ils seraient prioritaires sur tous les autres messages et permettre de retourner à l'affichage normale lorsque le problème résolu.

Une autre fonctionnalité qui pourrait être pratique, serait d'avoir la possibilité d'avoir des messages persistants qui résisteraient à l'écrasement actuellement observé lors du défilement du texte à l'écran.

5.2 Communication Réseau

5.2.1 Objectifs

Lors de la première ébauche du prototype par l'équipe précédente, la communication réseau était très éphémère et il fallait donc rendre l'ensemble plus solide, fiable et constant. Ce sont d'ailleurs les problèmes avec la communication réseau qui ont poussé le développement de l'architecture multithread.

En effet, la communication réseau ne pouvait pas fonctionner lors que le module des moteurs n'était pas connecté au microcontrôleur. De plus, avec l'ajout de code, le temps de réponse par réseau n'était pas constant et pouvais même créer des « timeout » de communication.

5.2.2 Implémentation

Pour permettre une communication plus stable, deux piles de messages ont été créées pour permettre une communication plus fiable et asynchrone par le fait même. Le protocole de communication n'a pas été changé, il s'agit toujours d'une communication de 3 octets via le protocole UDP et l'adresse IP est fixe au 192.168.0.2 sur le port 6625.

Par contre, vu que la communication multithread est maintenant asynchrone, la commande originale est attachée à la réponse pour s'assurer que le module de vision puisse faire un suivi précis de toutes les confirmations reçues par le contrôleur.

De plus, la tâche de réseau est responsable de l'interprétation des commandes et de les affecter au bon module. Dans le cas où la commande est destinée aux moteurs ou à la gâchette, les commandes sont mises dans les piles de messages des modules respectifs.

Voici la liste des commandes et des confirmations disponibles actuellement :

Commandes	Codes	Confirmations	Codes
Vérification système	0x41 0x41 0x41	Robot opérationnel	0x61 0x61 0x61
Tirer une balle	0x42 0x42 0x42	Balle tirée	0x62 0x62 0x62
Tirer trois balles	0x43 0x43 0x43	Trois balles tirées	0x63 0x63 0x63
Positionner le robot	0x44 0xX 0xY	Robot positionné	0x64 0x64 0x64
		Problème robot	0x65 0x65 0x65
		Problème commande	0x66 0x66 0x66

5.2.3 Développement futur

En ce qui concerne la planification du développement futur au niveau de la communication réseau, certaines fonctionnalités pourraient être très intéressantes tels que des commandes de configuration. Avoir certaines configurations modifiables pourrait permettre des changements de paramètres dans le système sans devoir recompiler un nouveau code. Ceci pourrait être très pratique lorsque le système sera testé ou encore lors d'un changement de configuration quelque part dans le système.

5.3 Récupération de panne

5.3.1 Objectifs

Dans le cas de panne, il est intéressant que le système soit capable de récupérer sans devoir redémarrer le contrôleur et réinitialiser l'ensemble des modules. Il est aussi difficile de savoir si un module externe est en erreur ou ne répond tout simplement pas.

Donc pour être en mesure d'identifier les sources de problème relié aux modules externes, un système de récupération de panne et de diagnostic des modules externes a été mis au point. Ce système consiste en une tâche mère qui peut créer et éliminer ses tâches enfants et les tâches enfants sont chacune responsable d'un module externe.

Les tâches enfants possède un bout de code spécifique qui leur permet de « pinger » leur module externe et de valider si le module répond ce qu'il est supposé. Dans le cas où le module répond correctement, un sémaphore est mis à jour dans la mémoire globale. La tâche mère, lorsque les tâches enfants sont créées, va vérifier de temps en temps que le sémaphore est disponible et donc par le fait même que ses tâches enfants ont des modules fonctionnels.

Par contre, si une tâche enfant n'est pas en mesure d'incrémenter le sémaphore parce que la réponse au ping n'est pas valide, ou que le module externe ne répond plus et a gelé la tâche enfant par le fait même, la tâche mère va effacer la tâche enfant, afficher un message d'erreur et recréer la tâche fautive. La tâche mère va ré-effectuer cette opération jusqu'à ce que la tâche enfant soit de nouveau fonctionnelle et ainsi récupérer d'une panne comme qu'un fil déconnecté.

Voici un diagramme de séquence pour bien illustrer les étapes expliquées ci-haut :

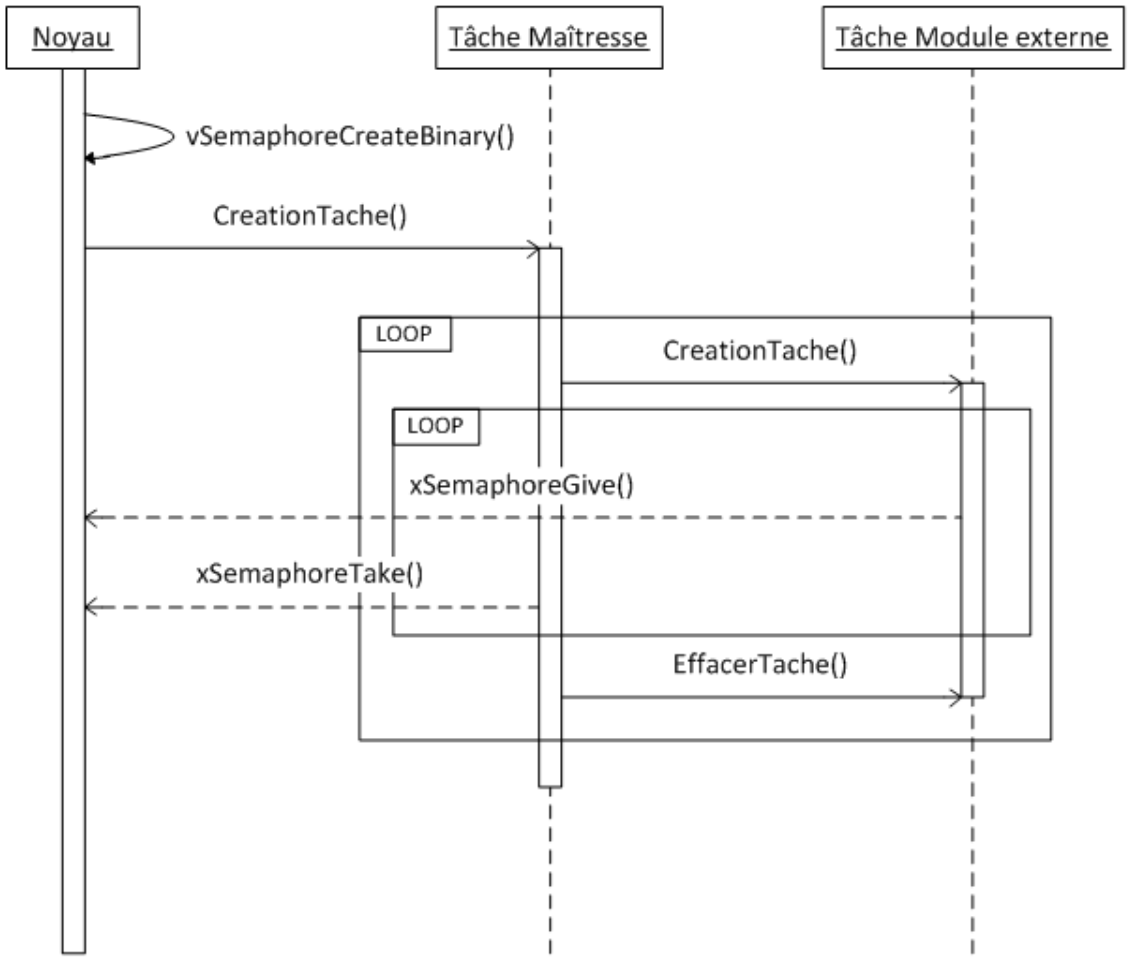

Figure 10 - Diagramme de séquence de récupération de panne

5.3.2 Développement futur

Que pourrait-il être amélioré dans cette partie de l'application? Pas grand choses. Mais une option qui serait intéressante serait que les tâches enfants possèdent un système de diagnostic plus évolué et ainsi être en mesure de fournir des messages d'erreurs encore plus précis concernant les modules externes.

5.4 Contrôle des moteurs

5.4.1 Objectifs

Le contrôleur des moteurs doit être en mesure de recevoir une position en X et Y et de positionner le robot à ces coordonnées le plus précisément et le plus rapidement possible. Initialement, cette partie du projet avait été donnée à un collaborateur international pour aider l'équipe de développement du projet, mais malheureusement aucune progression notable n'a été réalisée.

5.4.2 Problèmes rencontrés

L'équipe de projet a pris le relais du développement, mais le temps fut leur ennemi numéro un et donc impossible d'arriver à un développement suffisant pour faire des premiers tests sur le robot.

Durant la tentative de développement, un autre obstacle a fait surface. Le positionnement précis des moteurs c'est avéré très difficile, car le contrôleur des moteurs MD49 n'est pas spécifiquement conçu pour le positionnement, mais plus pour un système de propulsion. Lors de la prise en charge du projet, on avait laissé sous-entendre qu'il existait plusieurs modes d'opération des moteurs, mais que la documentation était floue et imprécise à cet égard.

Après plusieurs tentatives de déplacement, une recherche a été effectuée pour découvrir les autres modes de déplacement. Cette recherche a révélé que les autres modes de déplacements n'étaient en fait que des modes différents de propulsion où le mécanisme de direction pour faire pivoter un robot potentiel lors d'un déplacement. Donc le premier mode de sélection de vitesse s'est avéré le plus pratique et il sera conservé dans les tests futurs.

Toutefois, la documentation a révélé qu'il serait surement possible de déplacer les moteurs plus précisément que dans les premières tentatives effectuées sur le banc d'essai.

Non seulement la vitesse est variable, mais l'accélération/décélération aussi. Lors des essais préliminaires, il s'était avéré très difficile d'exécuter un déplacement vers une position limitrophe, car le déplacement était trop rapide et dépassait de beaucoup la cible.

Lors des rencontres initiales du projet, on avait laissé sous-entendre que la vitesse du déplacement devrait être calculée pour permettre une décélération brusque qui donnerait un mouvement non volontaire. Mais à la lecture de la documentation plus en détail, il semblerait que ce calcul ne sera pas nécessaire parce que les moteurs ne peuvent pas s'arrêter brusquement, ils suivent une courbe d'accélération/décélération définie que l'on peut ajuster au besoin par une commande au contrôleur MD49. Voici un tableau qui représente l'évolution de la vitesse des moteurs selon le facteur d'accélération sélectionné.

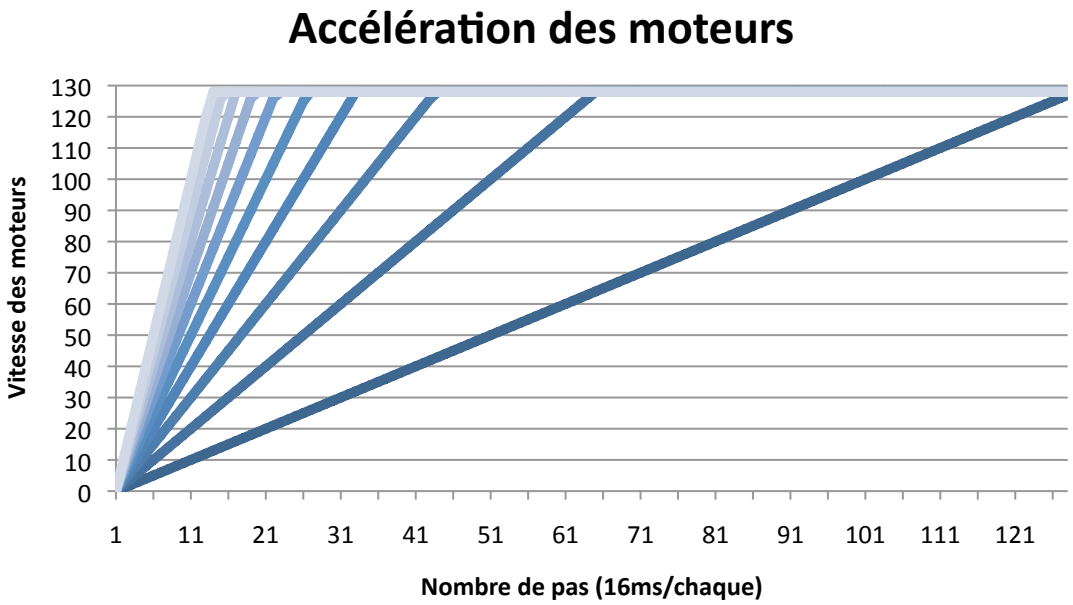

Figure 11 - Graphique sur l'accélération des moteurs

Le tableau ci-dessus permet de voir la flexibilité de l'accélération du moteur qui devra être une nouvelle variable dans le calcul optimal de l'équation. Un autre fait intéressant qui explique les difficultés de déplacements rencontrés est que l'accélération et la décélération fonctionnent à un intervalle fixe de 16ms. Quel est l'impact de cette caractéristique? Et bien dans les cas où le déplacement est très proche, il faudra quand

même calculer un temps certains temps pour effectuer le déplacement vu que l'on devra faire le déplacement plus lentement.
En ce qui concerne la vitesse de rotation des moteurs, les spécifications parlent d'une vitesse nominale de 122rpm alors que la vitesse à vide est estimée à 143rpm. Il y aurait donc un ajustement à faire entre le banc d'essai où les moteurs sont à vide et le trépied du robot qui offrira surement une charge inconnu en ce moment.

5.4.3 Recommandations

Bien que l'équipe de développement va tenter de livrer le meilleur mouvement possible du robot, il est possible que les efforts ne portent pas fruit soit parce que le déplacement reste trop imprécis ou que le déplacement soit trop lent pour avoir une bonne réaction pour atteindre les cibles.

Si c'est le cas, la recommandation serait de remplacer les moteurs EMG49 à engrenage ainsi que le contrôleur MD49 par un contrôleur et des moteurs électriques pas à pas aussi connus sous le nom de « stepper » en anglais, car dans certains cas, il est possible d'obtenir une précision de déplacement au degré prêt.

Toutefois, cette recommandation est faite sans avoir fait de recherche exhaustive sur les besoins que les moteurs du système doivent rencontrer, la vitesse et la force des moteurs pas à pas qui sont disponibles à des prix raisonnables ainsi que bien d'autres facteurs très importants qui devront être pris en considération.

Par contre, il est clair que déjà avec les tests préliminaires, il est plus facile de voir ce que les moteurs ont besoin de réaliser comme tâche et qu'avant de faire quelques modifications que ce soit au système de déplacement des moteurs, il faudra faire des tests complets sur le prototype du robot pour observer les déplacements du robot et identifier s'il y a des problèmes ou des sources d'améliorations possibles. Sans ses tests, tout changement serait effectué serait comme cette recommandation, c'est-à-dire pure spéculation.

5.5 Contrôle de la gâchette

5.5.1 Alimentation

Le contrôleur Pololu demande sa propre alimentation de 5-16V. Un mauvais branchement de l'alimentation peut engendrer des problèmes : bruler le contrôleur qui le rendrait inutilisable ou bruler un servomoteur qui le rendrait aussi inutilisable.

5.5.2 Signaux de contrôle

Le contrôleur émet et reçoit des signaux via une communication sérielle RS-232. Il faut que celui qui lui transmet un signal lui envoie du 5V. Pour le besoin de ce projet, nous avons utilisé plusieurs applications pour effectuer des tests, dont « Pololu Serial Transmitter » et « Serial Port Monitor ».

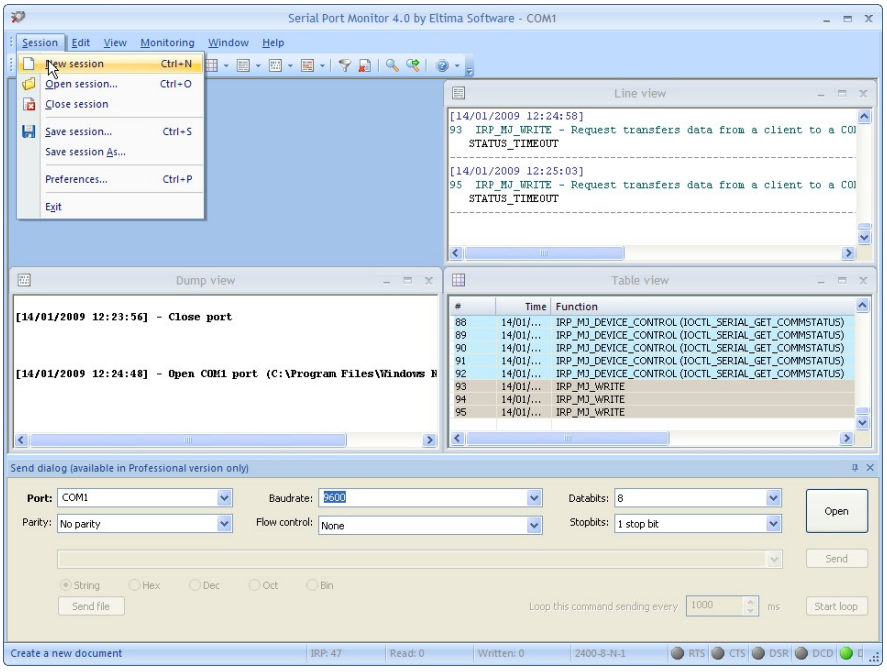

Figure 12 - Application « Serial Port Monitor »

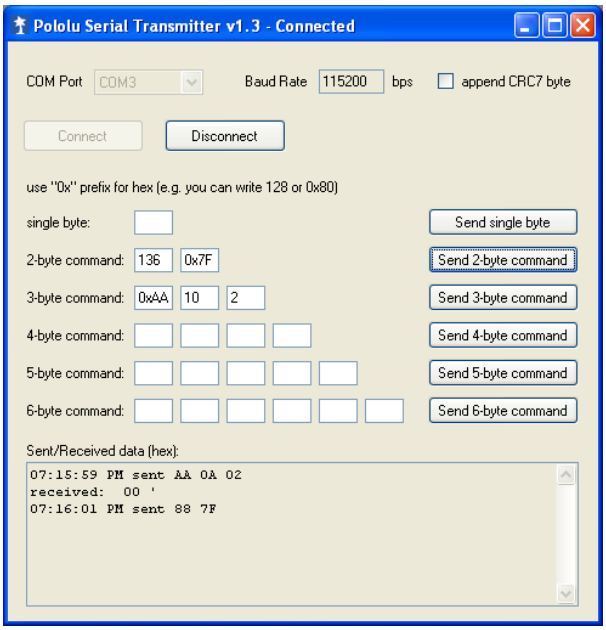

Figure 13 - Application « Pololu Serial Transmitter »

5.5.3 Indicateurs LED

Activité sur le port série

En attente d'une commande

Indique une erreur fatale

Vitesse de transmission trop rapide

Vitesse de transmission trop lente

5.5.4 Protocole de communication disponible

Pololu : mode de communication qui permet au contrôleur d'envoyer une position et vitesse au servomoteur. Le mode Pololu permet aussi de brancher en série plusieurs contrôleurs Pololu.

Mini-SSC : Plus simple, mais plus restreint. Ce mode ne permet que d'envoyer une position à un servomoteur spécifique.

Comme nous n'avons pas besoin de faire varier la vitesse et de brancher plusieurs contrôleurs en série. Notre choix s'est arrêté sur le mode Mini-SSC.

5.5.5 Configuration

Vitesse de transmission sur le port sériel : 2400

Pour définir le numéro du servomoteur utilisé, il faut suivre la démarche ci-dessous :

1. Mettre le contrôleur en mode Pololu en retirant le « jumper » attitré au mode Pololu / Mini SSC.

2. Envoyer la séquence de 3 octets appropriés :

[octet de départ = 0x80][change # servo = 0x02][numéro de servo = 0x00-0x10]

5.5.6 Utilisation

L'utilisation du contrôleur en mode Mini-SSC est assez simple une fois configuré. Il suffit d'envoyer une séquence de 3 octets :

Ex : [octet de départ = 0xFF][numéro de servo = 0x1F][position = 0xFE]

5.5.7 Développement futur

Afin d'être en mesure d'envoyer une commande de tire efficace permettant de tirer une à plusieurs balles, il faudra identifier le délai mécanique entre deux tires de la gâchette du fusil. Ce délai est nécessaire si on veut éviter de toujours tirer en rafale.

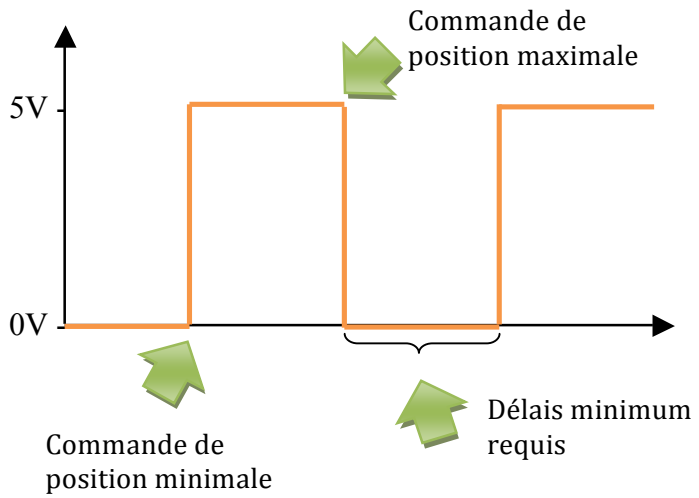

Figure 14 - Interprétation du signal envoyé par le contrôleur de gâchette

5.5.8 Problèmes rencontrés

Pour pouvoir développer le module de la gâchette, on a reçu un contrôleur Pololu 8-Servo qui se devait être fonctionnel. L'équipe qui avait travaillé sur le projet avant nous n'avait pas eu le temps d'implémenter le module de la gâchette et par conséquent ils n'ont pas testé s'il était fonctionnel. On a donc pris pour acquis qu'il l'était.

Le premier problème qu'on a rencontré lors de la session d'automne quand on a voulu débuter l'implémentation du module de contrôle de la gâchette était le manque d'alimentations. Le Pololu 8-Servo demande sa propre alimentation et sans ça, il ne peut fonctionner. Il a fallu plusieurs semaines avant d'obtenir quelque chose pour alimenter notre contrôleur. Du même coup, lorsqu'on on nous a fourni l'alimentation on nous aussi fournis un servomoteur pour pouvoir tester le tout. On avait donc le kit complet pour commencer le développement du module et ainsi être en mesure d'effectuer des tests rapidement.

Après plusieurs tentatives d'envoi de commandes de positionnement d'un PC vers le contrôleur par l'intermédiaire de l'application « Pololu Serial Transmitter utility for Windows » on n'avait rien de concluant. Le LED vert ne faisait que clignoter. Selon la guide d'utilisation du fabricant, on aurait dû avoir une certaine rétroaction. C'est-à-dire que le LED jaune devait être allumé lorsqu'on branchait le contrôleur à sa source d'alimentations. C'est à ce moment qu'il nous est passé par la tête qu'il serait possible d'une défectuosité matérielle. On a demandé l'aide de Christian Larose pour déterminer si la cause de l'absence de rétroaction du contrôleur était due à un problème matériel. Suite à ça, on a conclu que le problème était d'ordre matériel. On a donc demandé un nouveau contrôleur à Patrice. On lui a fait passer la même série de tests et heureusement celui était bien fonctionnel.

Suite à ça, on s'est rendu compte que lorsqu'on y branchait le servomoteur, on trouvait que celui-ci devait extrêmement chaud. On a tout de même essayé de faire fonctionner notre module de contrôle de la gâchette, mais sans résultat positif. On

s’est dit qu’il était possible que le servomoteur soit aussi défectueux. Nous avions fait face une fois au problème il était donc fort probable que cela se répète. En effet, il s’agissait bel et bien d’un servomoteur défectueux, car une fois que Patrice nous a remis un second servomoteur et qu’on lui a fait passer les mêmes tests, il s’est avéré que l’ancien était défectueux.

Pour terminer, une fois notre « setup » fonctionnel, on a essayé d’envoyer les commandes de l’EVK1100 vers le contrôleur. Malheureusement, rien de fonctionnait. Toujours aucune réponse, mais cette fois-ci le LED vert n’allumait pas. Le LED jaune demeurait toujours allumé comme s’il ne recevait rien du microcontrôleur. Le problème qui a été identifié par Patrice était le même que l’équipe avant nous a rencontré. C’est-à-dire que l’EVK1100 génère du 3.3V sur le port série, mais les contrôleurs demandent du 5.12 V. Il fallait donc, comme pour les moteurs, un convertisseur. Patrice a donc pris la relève pour nous en fabriquer un. C’est lui qui a présentement tout le kit de développement afin de pouvoir tester le convertisseur pour ensuite nous le remettre et continuer notre développement.

6. Conclusion

En résumé, l'équipe de développement croit avoir fait de grands progrès dans l'élaboration du projet de sentinelle paintball en surmontant plusieurs défis technologiques, que ce soit au niveau des outils utilisés, du matériel électronique en place ou encore de l'aspect R&D qui peut forcer la découverte de certaines réponses par des cas typiques d'essais-erreur vus l'aspect innovant de certaines réalisations.

Les technologies étaient relativement loin de la zone de confort de l'équipe de développement, plusieurs aspects comme les contrôleurs physiques étaient inconnus, mais plusieurs objectifs tels que la communication réseau fiable, un affichage sur l'écran LCD, un mécanisme de gâchette testé et une architecture multithread ont été atteints.

Toutefois, l'équipe ne serait pas honnête de ne pas vouloir admettre qu'ils avaient un objectif personnel et cet objectif était d'amener le prototype à un niveau opérationnel et d'être en mesure d'avoir des résultats tangibles comme une démonstration à présenter dans ce rapport. C'est d'ailleurs en partie ce désir qui motive l'équipe à continuer le développement du projet sentinelle au-delà du cadre du cours de LOG792 à la prochaine session pour tenter de compléter une première évolution de la sentinelle.

7. Références

Documentation FreeRTOS, *http://www.freertos.org*

Pololu, Robotics & Electronics, http://www.pololu.com/file/0J38/ssc04a_guide.pdf

Pololu Serial Transmitter utility for Windows, *http://www.pololu.com/docs/0J23*

Tutorial: Using the Pololu Servo Controller in .NET,

http://colinkarpfinger.com/blog/?s=pololu

Contrôleur MD49, *http://www.robot-electronics.co.uk/htm/md49tech.htm#acceleration*

Moteurs EMG49, *http://www.robot-electronics.co.uk/htm/emg49.htm*

Notes du cours de Log550

Rapport d'étape de notre équipe

Fiche de présentation de notre équipe