

ENSTA BRETAGNE

NON CONFIDENTIEL

Architecture Controller Area Network compatible Arduino pour un système d'assistance à la navigation



Auteur :
Kévin BRUGET ENSI 2013
Option Robotique

Ingénieurs pilotes :
Benoit CLEMENT
Bernt WEBER

Résumé Ce rapport de projet de fin d'études a pour but de présenter une architecture construite autour du bus de terrain Controller Area Network (CAN), compatible avec les cartes électroniques Arduino, afin de proposer un réseau de communication alternatif aux applications dans le domaine de la robotique. Celle-ci combine à la fois la robustesse du bus CAN et l'accessibilité du logiciel Arduino et de son environnement de développement. Dans le cadre de ce projet, nous avons décidé de l'utiliser afin d'améliorer le système d'assistance à la navigation handivoile développé par Splashelec. Ce système est composé de multiples nœuds que sont les CANinterfacers, cartes électroniques compatibles Arduino construites par Splashelec, chacun reliés avec différents capteurs et actionneurs. Tous peuvent communiquer avec une interface Homme-Machine (IHM) accessible directement depuis un smartphone ou une tablette utilisant le système d'exploitation open-source Android. Les informations sont partagées entre tous les nœuds (i.e les CANinterfacers) et ce grâce au bus. L'implémentation d'un bootloader CAN permet de reconfigurer chaque nœud directement à l'aide de messages CAN. Un protocole CAN de haut niveau, baptisé SimpleCAN, a été élaboré afin d'augmenter le niveau d'abstraction du système et d'apporter des fonctionnalités à nos noeuds (synchronisation, filtrage, ...). Le but étant de créer un système générique capable de fonctionner dans différents types de situations et adapté à tous types d'utilisateurs, qu'ils soient handicapés ou non. Le résultat de ce travail sera présenté dans un démonstrateur visible lors de la WRSC 2013 à Brest. Celui-ci sera construit autour du miniJI du club handivoile, un voilier monoplace contrôlé par joystick.

Abstract This final year project report presents a Controller Area Network (CAN) Bus architecture based on Arduino compatible boards, to be used as an alternative communication system for robotic applications. This combines both, the robustness of CAN and the accessibility of Arduino software. In the frame of the project, the architecture is developed here to improve the Splashelec Navigational Assistance System, which was initially created for disabled people. The system is composed of Arduino compatible boards build by Splashelec, the CANinterfacers, wired with various sensors and actuators, and communicating with an Human Machine Interface (HMI), directly accessible via a mobile phone or a tablet running on the open-source operating system Android. Information is transferred through the CAN bus architecture between multiple nodes (i.e. Arduino compatible boards) and the implementation of a CAN bootloader allows the reconfiguration of the nodes directly through the bus. A high layer CAN protocol was also developed in order to increase the abstraction level and bring new functionalities (synchronisation, filtering, ...). The aim is to create a generic system able to work in various kinds of situations, adaptable to all kinds of users, including persons with all sorts of disabilities. This work will result in a demonstrator on a MiniJI for the WRSC 2013, an entirely joystick controlled boat for single handed sailing.

Table des matières

1	Introduction	5
2	Présentation du contexte	8
2.1	ENSTA Bretagne	8
2.2	Splashelec	9
2.3	Objectifs du projet de fin d'études	10
2.4	Planning prévisionnel	11
3	Etat de l'art	13
3.1	Systèmes d'assistance à la navigation	13
3.2	Protocoles utilisés	15
4	Controller Area Network	17
4.1	Historique	17
4.2	Généralités sur le protocole	18
4.2.1	Modèle réseau	18
4.2.2	Fonctionnement pratique	20
4.3	Choix de l'architecture	22
5	Une architecture CAN compatible Arduino	24
5.1	Résultats préliminaires	24
5.1.1	Le système Splashelec	24
5.1.2	Les premiers démonstrateurs	27
5.2	Une architecture plus aboutie	29
5.3	Reprogrammation des nœuds	30
5.3.1	Présentation	30
5.3.2	Etapes de reconfiguration	31
6	SimpleCAN, une couche protocole CAN de haut niveau	34
6.1	Nécessité d'un tel protocole	34
6.2	Fonctionnalités apportées	35
6.2.1	Synchronisation	35
6.2.2	Election de leader	38
6.2.3	Filtrage sur le bus	39
7	WRSC 2013	42
7.1	Objectifs de l'évènement	42
7.2	Elaboration d'un démonstrateur	43
8	Conclusion	46
A	Annexes	50
A.1	Article IRSC 2013	50
A.2	Documentation librairies CAN (Anglais)	64

Table des figures

1.1	Exemple de nœud CAN : le joystick et le servo-contrôleur programmable dans sa première version puis dans une version intégrant directement un CANinterfacer dans la carte.	6
2.1	Planning prévisionnel de réalisation du projet de fin d'études.	11
4.1	Exemple de modèle OSI pour les protocole CANopen, CANKingdom, DeviceNet et SDS.	19
4.2	Représentation d'une trame de donnée selon la spécification du protocole CAN.	20
4.3	Représentation d'une trame de requête selon la spécification du protocole CAN. On remarque qu'elle ne contient pas de champ de données.	20
4.4	Arbitrage sur le bus entre deux nœuds. L'on remarque que le nœud 2 perd la priorité d'émission au profit du nœud 1 qui contient un bit dominant '0' contre un bit récessif '1' pour le deuxième nœud. Sa transmission est donc interrompue jusqu'à ce que le nœud 1 ait terminé.	21
4.5	L'architecture de notre système construite autour du bus CAN.	23
5.1	Le système est assimilé à une boucle fermée où l'utilisateur peut être retiré pour un comportement autonome.	25
5.2	Le servo-contrôleur programmable de Splashelec dans un boîtier étanche. Il contient plusieurs entrées pour le joystick, le compas, le capteur d'angle de barre, la batterie, etc...	26
5.3	Le système Splashelec est composé d'un vérin électrique (en rouge), d'un joystick et d'un clavier de contrôle (en bleu) et du gouvernail (en vert).	26
5.4	L'IHM affiche les différentes données capteurs circulant sur le bus et possède un clavier virtuel (en jaune) pour sélectionner les différents modes de fonctionnement (joystick, pilote automatique).	27
5.5	Le terminal série du logiciel Arduino IDE permet de visualiser le comportement de notre jeu de communication.	28
5.6	CANinterfacer construit par Splashelec. Il s'agit d'une carte électronique compatible Arduino Leonardo et contenant un contrôleur CAN intégré, le MCP2515.	30

5.7	La reprogrammation d'un noeud fait intervenir les trois éléments indispensables que sont le PC contenant le nouveau programme, l'interface CAN2USB et le noeud à reconfigurer.	31
5.8	Le script python lancé depuis un terminal linux témoigne de la reconfiguration réussie de la carte possédant l'identifiant 0xFF avec un sketch nommé "Blink".	32
5.9	La reconfiguration fait suite à une série de questions/réponses permettant de cibler puis transmettre le programme à la carte.	33
6.1	La synchronisation des noeuds se fait à l'aide d'un message spécifique directement envoyé par le maître à l'instant T_1 , contenant la mesure $T_{2\text{maître}}$ de la période précédente. L'objectif est de faire coïncider $T_{2\text{esclave}}$ avec à l'instant T_{sync} à l'aide d'un régulateur PID.	37
6.2	Représentation graphique de l'évolution du déphasage entre $T_{2\text{maître}}$ et $T_{2\text{esclave}}$ sur une consigne nulle. Sur la figure, un cycle représente une période de 100 ms, et la résolution des timers correspond au compteur interne de la carte.	38
6.3	La composition des différents filtres sur les buffers de réception du MCP2515.	39
6.4	Utilisation des filtres et masques sur les buffers de réception du MCP2515.	40
6.5	Filtrage sur un identifiant spécifique.	40
6.6	Filtrage sur une plage d'identifiants.	41
7.1	Le système sera implémenté sur un voilier miniJI pour servir de démonstrateur durant la WRSC 2013.	43
7.2	L'algorithme a été testé sous un simulateur réalisé grâce à QTCreator, représentant le voilier en 3 dimensions, permettant de choisir la direction du vent et de visualiser le comportement du bateau.	45

Introduction

La robotique autour du monde de la voile est aujourd’hui devenue une réalité. Que ce soit pour remplacer l’humain en course, pour réaliser des mesures scientifiques de manière autonome ou simplement assister le navigateur, la robotique est omniprésente dans ce genre de système [14, 17, 20, 27]. La démarche de Splashelec, auto-entreprise gérée par Bernt Weber, n’est pas de supprimer totalement l’action humaine comme on l’entend souvent en robotique mais de l’assister tant au niveau de l’acquisition de l’information, de l’aide à la décision que de l’action dont les aspects puissance sont pris en charge par le robot. Car, en effet, le monde de la voile est un bon exemple d’activité qui demeure relativement inaccessible aux personnes handicapées en raison de la perception de l’information qui se fait par la mobilité du skipper et par les efforts importants qui peuvent être demandés pour piloter un bateau. Destiné à promouvoir et à rendre accessible ce loisir à destination de ce public, Splashelec a alors développé un système d’assistance à la navigation afin de répondre aux contraintes imposées par les particularités inhérentes à la voile et la perte de mobilité de l’utilisateur, tout en faisant preuve d’innovation et en utilisant les technologies existantes.

La spécificité de ce projet réside dans l’adaptation d’un boîtier électronique couplé avec un joystick et un système de barre électrique couplé avec un vérin surdimensionné, qui permet à une personne handicapée de manœuvrer un voilier avec les sensations d’un barreur classique. Ce système est entièrement amovible et son installation peut se faire de manière non invasive sur différents modèles de voiliers. Contrairement aux autres systèmes existants, le mouvement du safran suit celui imposé par le joystick, et lorsque celui-ci retourne à la position neutre, la barre en fait de même.

C’est dans ce contexte que nous avons été amené à collaborer l’an passé, lors d’un projet de deuxième année, afin de développer ce système par l’ajout de capteurs (anémomètre, sondeur, centrale inertielle,...) et d’une interface Homme-Machine (IHM) afin de visualiser ces données sur une tablette Android. Cette collaboration entre Splashelec et l’ENSTA Bretagne s’est ainsi poursuivi tout naturellement lors du projet de fin d’études afin de proposer une architecture décentralisée autours de plusieurs noeuds, cha-

un responsable d'un capteur/actionneur spécifique, relié par le bus de terrain Controller Area Network (CAN). Le nouveau projet ainsi créé est alors basé sur un double objectif :

- Le premier est de bâtir une architecture basée sur le bus CAN compatible avec les cartes électroniques Arduino et de manière générale avec les microcontrôleurs AVR d'ATMEL qui les composent. Cette architecture a pour but d'être utilisée comme une alternative fiable aux systèmes de communication pour les applications relevant du domaine de la robotique, grâce à la robustesse procurée par le protocole CAN, tout en combinant l'accessibilité du logiciel Arduino afin d'élargir le public visé par ce type de système.
- Le deuxième objectif est d'utiliser les bases de cette architecture afin d'améliorer le système existant d'aide à la navigation handivoile développé par Splashelec. A cette fin, notre architecture sera implémentée sur le MiniJI que le club handivoile met à notre disposition afin de présenter un démonstrateur viable au cours de la World Robotic Sailing Championship (WRSC) 2013.

Le système résultant sera alors composé de cartes compatibles Arduino, les CANinterfacers, produites par Splashelec [30], câblées avec différents capteurs et actionneurs (Fig. 1.1), et communiquant par Bluetooth avec une interface homme-machine, directement accessible depuis un smartphone ou une tablette fonctionnant sous le système open-source Android. Les informations transiteront directement à travers l'architecture du bus CAN en bénéficiant d'un protocole CAN de haut niveau, SimpleCAN, développé en open-source au cours de ce projet. La mise en œuvre d'un bootloader CAN permet également une reprogrammation d'un nœud spécifique directement par le bus en complément du logiciel Arduino IDE. L'objectif d'une telle architecture est de créer un système générique capable d'évoluer dans toutes sortes de situations, mais également adaptable à tous types d'utilisateurs. Les résultats de ces travaux donneront lieu à la création d'un démonstrateur pour le WRSC 2013.

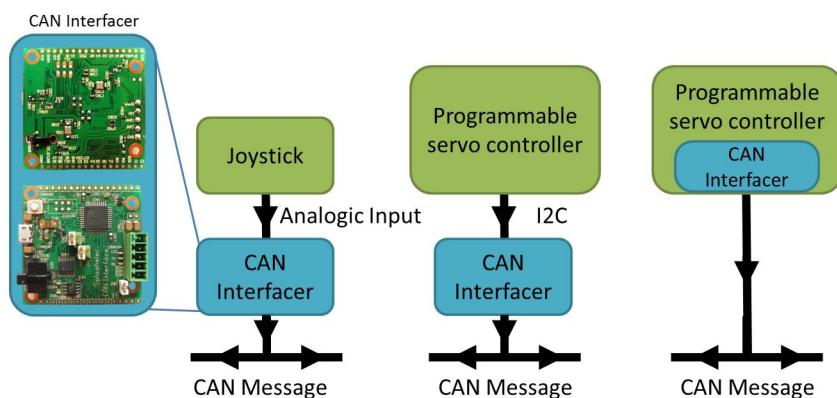


FIGURE 1.1 – Exemple de nœud CAN : le joystick et le servo-contrôleur programmable dans sa première version puis dans une version intégrant directement un CANinterfacer dans la carte.

Ce présent rapport commencera dans un premier temps par replacer le sujet dans son contexte et environnement d'étude en présentant l'organisme d'accueil, l'ENSTA Bretagne et son partenaire Splashelec, ainsi que les objectifs fixés par le sujet. Un planning prévisionnel des tâches à effectuer sera également détaillé. Dans un second temps, le bus de terrain CAN, son fonctionnement et ses généralités seront explicités afin de justifier notre choix d'un tel protocole mais également pour permettre au lecteur non initié de comprendre les bases de notre architecture. Nous détaillerons ensuite notre système depuis sa création, ses aspects et fonctionnalités ainsi que le mécanisme particulier de la reprogrammation à l'aide d'un bootloader (i.e le programme d'amorçage de la carte) compatible CAN. Une partie sera également dédiée à notre protocole de haut niveau, SimpleCAN, bâti au cours de ce projet sur une initiative open-source. Enfin nous évoquerons notre participation à la World Robotic Sailing Championship (WRSC) 2013 par le biais de notre démonstrateur et de la conférence mais aussi des objectifs que représentent un tel évènement.

Pour conclure, si dans ce rapport nous traitons l'implémentation de notre architecture dans le système d'assistance à la navigation handivoile de Splashelec, il ne faut pas perdre de vue que son objectif réside dans l'adaptabilité du système à d'autres types de robots, terrestres comme marins et veut se proposer comme une architecture fiable, robuste et polyvalente. Toujours dans la problématique du handicap, la portabilité de ce système sur des fauteuils roulants électriques est une piste sérieuse qu'envisage Splashelec dans l'optique de proposer des solutions open-source à des systèmes demeurant majoritairement fermés et propriétaires.

Chapitre 2

Présentation du contexte

Ce projet de fin d'études étant une collaboration étroite entre l'ENSTA Bretagne et Splashelec, il convient de décrire ces deux organismes plus en détail afin de replacer le contexte dans lequel ce projet a été réalisé. Nous reviendrons également sur les principaux objectifs spécifiés par le sujet, ainsi que sur le planning prévisionnel et réel des réalisations.

2.1 ENSTA Bretagne

L'ENSTA Bretagne est un établissement public d'enseignement supérieur et de recherche sous tutelle de la Direction Générale de l'Armement (DGA) du ministère de la Défense. Elle partage ce statut avec 3 autres écoles : l'Ecole Polytechnique, l'ISAE (Institut Supérieur de l'Aéronautique et de l'Espace) et l'ENSTA ParisTech. Situé à Brest, l'ENSTA Bretagne forme des ingénieurs (civils et militaires) dont les compétences répondent aux exigences des industries mécanique, électronique et informatique les plus innovantes.

Crée en 1971, l'ENSTA Bretagne (anciennement ENSIETA) était destinée, à l'origine, à former exclusivement des ingénieurs militaires pour les besoins du ministère de la Défense. L'école s'est ouverte aux civils à partir de 1990 et la montée en puissance des effectifs civils s'est faite rapidement en parallèle à une décroissance des effectifs militaires liée aux restructurations de l'appareil de défense. Au cours de la même période, l'ENSTA Bretagne faisait son entrée dans le monde de la R&D en accueillant, en 1992, son premier enseignant-chercheur. Aujourd'hui, les promotions sont dimensionnées autour de 200 élèves par an (80% de civils et 20% de militaires).

Outre la formation initiale d'ingénieurs qui y est dispensée (Cycle ingénieur classique et Cycle ingénieur par alternance), l'ENSTA Bretagne propose des formations de type Master et Mastère spécialisé. Elle dispose aussi d'une large gamme de stages destinés aux entreprises dans le cadre de la formation continue. Elle organise également des formations spécifiques au plan national pour la DGA et au plan international dans le

cadre d'accords de coopération. Enfin, l'ENSTA Bretagne mène des activités de R&D principalement axées sur les sciences et technologies de l'information et de la communication et la mécanique des structures navales. Elle est ainsi administrateur du « GIS Europôle Mer, des pôles de compétitivité « Mer Bretagne » et « ID4CAR » mais aussi du PRES « Université Européenne de Bretagne ». Elle est également membre des pôles de compétitivité « Image et réseaux » et « EMC2 ».

Etant élève en option Robotique avec une composante Génie Logiciel, mon stage s'est tout naturellement déroulé au sein du pôle STIC de l'école, me permettant de bénéficier du soutien des enseignants chercheurs et de mon tuteur Benoit Clément lors de difficultés techniques ou pratiques.

2.2 Splashelec

Depuis trois ans, Bernt Weber développe sous le nom Splashelec des systèmes qui assistent des personnes en situation de handicap souhaitant s'adonner à la voile. Les premiers systèmes commercialisés permettent aux personnes ayant relativement peu de force ou une amplitude de mouvements restreinte de pouvoir prendre le contrôle du voilier en maniant la barre à l'aide d'un joystick. Ainsi, il est désormais possible d'intégrer une personne handicapée dans un équipage de voilier et ceci de manière valorisante, car le poste de barreur est un poste à forte responsabilité.

Techniquement, ces systèmes de barre utilisent la mécanique des pilotes automatiques du commerce. Cependant, l'électronique a été développée sur mesure par Splashelec, car les systèmes des pilotes automatiques ne permettent pas de changer le logiciel interne, ce qui est pourtant nécessaire pour l'adaptation aux différents handicaps. Utilisés sur des bateaux d'une certaine taille pour pouvoir permettre l'activité en équipage, ces systèmes ont généré de l'intérêt auprès du public et ont débouché sur des demandes venant d'utilisateurs de petits bateaux monoplaces, utilisés en général par des associations spécialisées dans la promotion de la voile pour les personnes handicapées. Sur de tels bateaux, il n'est pas suffisant de pouvoir diriger la barre à l'aide du joystick, il convient de s'occuper également de l'ensemble des réglages, notamment ceux des voiles. Si sur le papier, un tel système peut paraître simple, il est techniquement beaucoup plus compliqué à réaliser. De plus un développement sur mesure s'impose concernant la partie mécanique car il n'existe pas de systèmes satisfaisant dans le commerce. Il faut également, au niveau électronique et logiciel, pouvoir connecter et gérer une multitude de capteurs et actionneurs, ce qui est difficile avec l'architecture centralisée du système de barre actuel de Splashelec.

Ainsi, depuis bientôt deux ans, Splashelec travaille avec le club Handivoile Brest et l'ENSTA Bretagne sur différents aspects permettant d'améliorer le système actuel, qui devraient désormais aboutir à un premier bateau prototype fonctionnel. Un mécanicien a conçu plusieurs générations de prototypes de petits winchs électriques, et Splashelec a développé des nouvelles cartes électroniques (CANinterfacer) pour un système construit autour d'un bus CAN.

2.3 Objectifs du projet de fin d'études

Comme nous l'avons évoqué précédemment, Splashelec développe depuis plusieurs années un pilote automatique pour voiliers basé sur une initiative « open source » autant au niveau matériel que logiciel, intégrant une compatibilité avec les cartes électroniques Arduino. Le firmware et le matériel de ce pilote peuvent facilement être adaptés à des situations non-standards et, jusqu'ici, ce système était utilisé pour ajouter au pilote un mode d'assistance pour les personnes atteintes de handicap moteur en leur permettant de barrer le voilier en finesse à l'aide d'un joystick.

Plusieurs projets ont vu le jour afin d'ajouter d'autres fonctionnalités au système existant et tous ont montré le besoin d'un câblage simplifié et plus standardisé que celui actuellement utilisé. Un système de bus permettrait une intégration à un niveau d'abstraction plus élevé ce qui faciliterait l'adaptation à des situations beaucoup plus complexes et surtout non prévues. Il a été fait le choix de travailler avec le bus de terrain CAN, une carte électronique spécifique (le CANinterfacer de Splashelec) et au cours de l'application système (voir 5.1.2 Les premiers démonstrateurs), une première implémentation de protocole a déjà été réalisée en utilisant une librairie standard Arduino relativement simplifiée. Ces détails seront abordés ultérieurement dans leurs parties respectives.

Le but de ce projet de fin d'études est de travailler sur l'implémentation de ce nouveau bus et de bâtir une architecture basée sur un réseau de noeuds. Les objectifs initiaux de ce projet sont détaillés ci-dessous :

- Finaliser l'implémentation d'un protocole CAN pour notre architecture
- Réaliser un bootloader CAN Arduino pour permettre la reprogrammation et la reconfiguration de tous les noeuds directement à travers le bus
- Permettre la transmission de messages textes des noeuds vers un PC pour le déverminage de l'application
- Intégrer à notre architecture un système ROS et/ou MOOS pour éléver le niveau d'abstraction.
- Préparer un démonstrateur pour la WRSC 2013 (World Robotic Sailing Championship) sur le miniJI du club Handivoile afin de montrer la faisabilité d'un tel système.

Dans le cadre particulier de cette démonstration à la WRSC 2013, le voilier monoplace sera équipé de deux winchs électriques et d'un système de barre, le tout interconnecté par le biais du bus CAN. En rajoutant les capteurs initiaux sur le bus (GPS, anémomètre, centrale inertielle,...) ce voilier deviendra alors capable d'exécuter des tâches de voile robotisée. Ces capacités correspondent à une demande exprimée par les moniteurs de voile handisport pour augmenter la sécurité des personnes. En effet on pourra alors par exemple limiter la gîte du bateau ou éviter de sortir d'une zone de navigation. Une autre demande également est celle de pouvoir prendre le contrôle du bateau par le biais d'une

télécommande, exigence qui est également préconisée par le règlement de la WRSC.

Dans l'ensemble, ces tâches ont tous été maintenues au cours de l'avancement du projet à l'exception de l'intégration d'un système ROS/MOOS dans notre architecture. En effet, au cours d'une discussion avec Luc Jaulin, enseignant chercheur au pôle robotique de l'ENSTA Bretagne, il est apparu qu'un tel système ne serait pas adapté à notre architecture et n'apporterait rien en matière de performance. Initialement choisi pour intégrer le traitement d'image par OpenCV à notre démonstrateur, nous avons fait le choix d'abandonner son intégration.

2.4 Planning prévisionnel

La planification de manière générale s'est organisée suivant le diagramme de Gantt suivant (Fig. 2.1) établi en début de projet :

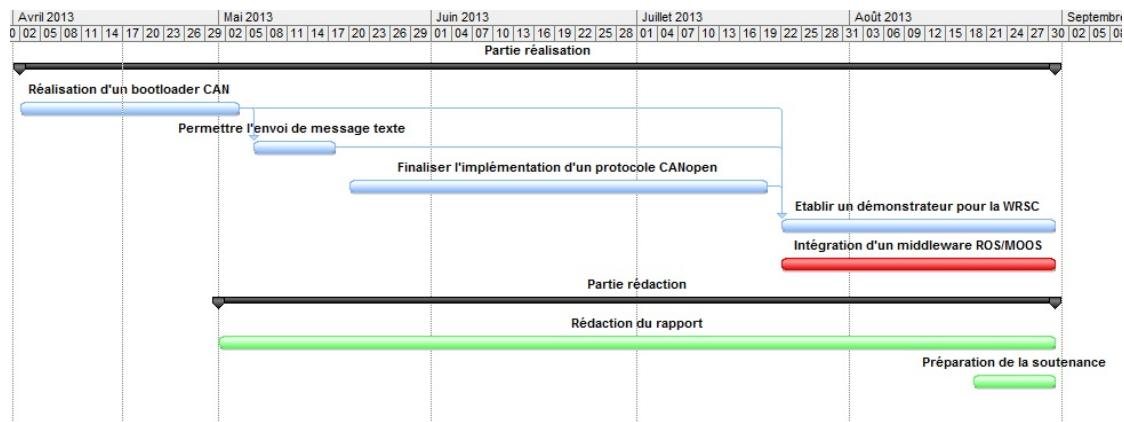


FIGURE 2.1 – Planning prévisionnel de réalisation du projet de fin d'études.

Les tâches étant plus ou moins indépendantes les unes des autres, elles se succèdent de manière linéaire dans le but d'obtenir au final un système construit autour d'une architecture CAN fonctionnelle. Ce système sera alors exploité afin de produire un démonstrateur sur le voilier MiniJI et de se présenter au challenge WRSC. Comme évoqué précédemment, l'utilisation d'un middleware de type ROS/MOOS n'était pas encore clairement défini à ce stade du projet, c'est pourquoi sa réalisation et son ordre de priorité ont été placés en fin de projet, pour finalement être abandonnés.

Globalement, les délais de réalisation estimés ont été tenus. En effet un léger retard a eu lieu en début de projet, car la complexité du bootloader CAN fut mal estimée au départ, mais celui-ci fut comblé par le développement rapide du nouveau protocole SimpleCAN.

A l'heure où ce présent rapport est écrit, la partie démonstrateur est partiellement entamée :

- Les algorithmes de suivi de ligne élaborés par Luc Jaulin [16], testés et utilisés sur le robot voilier VAIMOS sont en cours de portage vers une carte électronique Raspberry Pi [5].
- Matériellement, les CANinterfacers requis sont prêts ainsi que le matériel électronique et mécanique pour les winchs électriques.
- L'architecture du système reste à définir et à implémenter : intégration des CANinterfacers à chaque capteur/actionneur, algorithme de comportement de chaque nœuds.

Une fois ces tâches réalisées, le système devra être testé en laboratoire puis à terme sur le miniJI du club handivoile afin de réaliser des tests en mer et régler les derniers détails avant la WRSC. Ces résultats ne sont pas encore connus à l'écriture de ce rapport, ils seront abordés lors de la présentation orale.

Moyens humains et matériels :

L'essentiel du projet est à réaliser en autonomie. Je dispose du soutien technique de Bernt Weber pour toutes les parties relatives à la programmation sur Arduino et l'élaboration du système en général. La proximité avec les enseignants du pôle STIC est également un atout pour toutes questions sur le domaine de la robotique, de l'électronique et sur la WRSC plus globalement. Je dispose du club Robotique pour toutes les réalisations électroniques, câblages, soudures,... ainsi que pour le matériel. Bernt Weber met également à disposition certains éléments de matériel dont l'école ne dispose pas (carte électronique Splashelec, programmateur, carte de puissance, actionneurs...). En terme de logistique, nous pouvons disposer du zodiac de l'ENSTA Bretagne pour réaliser nos campagnes d'essai en mer avec le miniJI du club handivoile de Brest.

De manière générale, ce projet n'a amené aucun coût supplémentaire qui n'ait pas été initialement prévu. Les logiciels utilisés durant la réalisation des travaux étaient open-source et leur utilisation gratuite. La production de CANinterfacers et cartes de puissance a été intégrée dans le budget nécessaire à la réalisation du démonstrateur. Bien que la programmation du bootloader présente des risques sur le fonctionnement de la carte si les réglages ne sont pas les bons, aucune carte n'a été endommagée, ce qui n'a pas engendré de frais supplémentaires.

Chapitre 3

Etat de l'art

Avant de bâtir notre système, nous avons procédé à diverses recherches bibliographiques sur le sujet afin de réinvestir au maximum les travaux déjà existants ou relativement similaires à notre besoin. Fort des connaissances de Bernt Weber sur le sujet des pilotes automatiques existants, nous avons passé à la loupe leur principe de fonctionnement afin de pouvoir s'inspirer et réutiliser si possible certaines fonctionnalités déjà implémentées. Nous avons également fait le tour des différents protocoles utilisés dans ces systèmes afin de confirmer notre choix de l'usage de CAN comme une solution viable.

3.1 Systèmes d'assistance à la navigation

Le système d'assistance à la navigation handivoile produit par Splashelec, permettant à une personne handicapée de barrer un bateau à l'aide d'un joystick, est très proche des différents pilotes automatiques disponibles dans le commerce. A savoir que :

- Splashelec utilise dans la plupart des cas le système d'actionneurs vendu avec les pilotes automatiques commerciaux. Il existe cependant une différence dans le dimensionnement dans le cas de petits bateaux (i.e d'une taille jusqu'à 10m environ). En effet, une barre de pilote automatique utilise typiquement un moteur électrique avec une puissance de l'ordre de 10 à 20 W, ce qui permet seulement des mouvements de gouvernail lents (par comparaison avec ce qu'un barreur humain peut produire). Si l'on veut donner à une personne handicapée les mêmes sensations par manipulation d'un joystick, le moteur doit être surdimensionné par rapport à un vérin de pilote automatique commercial.
- L'électronique de puissance des calculateurs commerciaux ainsi que le système de Splashelec sont conçus autour d'un pont en H, ce qui permet une modulation de la vitesse pour réaliser des mouvements du gouvernail assez fins. Il n'y a pas de différences techniques entre les deux systèmes globalement.
- Le firmware des pilotes automatiques commerciaux est très spécialisé pour ce qui est de la conservation du cap à l'aide d'un compas, des données de l'anémomètre

ou des données GPS. Tous les pilotes automatiques dont nous avons connaissance utilisent, tout comme nous, un contrôle en boucle fermée à l'aide d'un correcteur PID. En revanche, certains pilotes automatiques du commerce qui utilisent aussi un joystick pour contrôler le gouvernail ont un comportement différent. En effet, et contrairement à notre système, en agissant sur le joystick nous allons initier un mouvement sur le gouvernail, mais lorsque le joystick revient dans sa position neutre le gouvernail reste en position. L'utilisation d'un mode proportionnel permet au gouvernail de suivre le joystick [24]. Un tel système offre une assistance beaucoup plus aboutie à un pilote handicapé [31].

La modification de ce firmware n'est malheureusement pas possible pour les pilotes automatiques disponibles dans le commerce pour des raisons évidentes de licence propriétaire. Cependant celle-ci pourrait permettre d'ajouter d'autres adaptations pour des handicaps différents comme :

- Un joystick avec amortissement. En effet de nombreux handicaps produisent des mouvements saccadés de la main. L'idée d'un contrôle à l'aide d'une simple position moyenne permettrait un contrôle précis d'un voilier pour ce type de personnes.
- Une correction automatique de la position du gouvernail en fonction du centre de la manette. En effet, un bateau à voile n'est généralement pas complètement équilibré et à tendance à se placer face au vent, le « Weather helm ». Le gouvernail doit garder un angle fixe afin de maintenir le bateau dans une direction, ce qui implique de maintenir en permanence le joystick dans une position excentrée ce qui peut rendre la tâche difficile, plus encore si le handicap implique une dextérité réduite.

L'expérience acquise lors de l'utilisation de notre système d'assistance à la navigation nous a également montré l'apport que pourrait apporter l'utilisation d'un capteur de position pour permettre un retour de force dans le joystick. En établissant une relation entre le gouvernail et la position du joystick, l'on pourrait faire bénéficier à l'utilisateur cette sensation lors du pilotage d'un bateau. En effet, dans cette configuration, un angle de gouvernail plus grand fait que l'eau sera enclin à pousser plus fort sur la barre. En utilisant le ressort de rappel du joystick, le barreur sentirait la force qu'exerce l'eau sur son gouvernail et aurait également une information sur sa position.

Systèmes d'aide existants conçus pour les handicapés

Les systèmes commerciaux de pilote automatique sont assez nombreux et très variés dans les fonctionnalités proposées. A travers l'exemple de deux systèmes d'assistance à la navigation existants, nous allons constater les problématiques relatives au câblage et les solutions qui en résultent.

L'entreprise « Hansa Sailing » [26] est un fabricant de plusieurs systèmes de commandes électriques destinés aux skippers handicapés mais aussi vendeur d'une gamme de bateaux. Leur câblage applique une topologie en étoile centrée autour d'un boîtier de commande [18]. Les actionneurs du winch et du gouvernail disposent chacun de 2

connecteurs métalliques pour les relier à des moteurs à courant continu. Les périphériques d'entrées, eux, ont besoin d'une connexion 9 broches, généralement utilisée pour un joystick et plusieurs boutons poussoirs associés à différentes actions comme la sélection de modes par exemple. En revanche aucun capteur n'est associé à ce système.

La société de Steve Alvey [19] vend, elle, des commandes électriques pour le Martin 16 et d'autres types de bateau à voile utilisés par des personnes handicapées [2]. Il conçoit également des systèmes spécialisés dans ce domaine, comme par exemple le bateau piloté par Hilary Lister [12]. Pour manipuler ce bateau, elle utilise un système « sip-and-puff » composé de trois pailles, et dispose d'un pilote automatique Raymarine. Le manuel de ce système [1] montre une prise 3 broches étiquetée « Sea Talk » pour les connexions d'interface de commande (joystick, sip-and-puff, etc...), et ce système de bus permet de simplifier le câblage. Une deuxième prise, « Helm Drive » s'enfiche dans le connecteur standard du vérin électrique Raymarine ST4000/SPX-5 (2 broches sont utilisés pour la connexion au moteur). Aucun capteur supplémentaire n'est relié à cet actionneur.

Les conclusions que nous pouvons tirer de tels systèmes d'aide à la navigation est que le nombre de connexion a atteint une limite où il devient impossible ou très coûteux d'ajouter d'autres périphériques, qui plus est dans un environnement où chaque connecteur doit être résistant à l'eau de mer et étanche. Ainsi, ajouter des capteurs aux actionneurs déjà existants compliquerait encore plus le câblage. C'est ici qu'entre en scène le système de bus, qui a déjà montré son potentiel pour des interfaces utilisateurs, et que nous pensons étendre à l'ensemble du système, y compris dans les actionneurs et les capteurs associés. Ceci simplifierait les installations et pourrait permettre des interactions plus avancées et complexes, une meilleure expérience de la part de l'utilisateur mais aussi rendrait le système plus modulaire, flexible et adaptable à d'autres types d'utilisations.

3.2 Protocoles utilisés

Les afficheurs marins multifonctions et les pilotes automatiques installés sur les bateaux d'aujourd'hui utilisent le plus souvent des bus série multi-drop avec un protocole propriétaire spécifique. Nous avons par exemple le système SeaTalk utilisé par Raymarine [23] et Topline de NKE [9]. Sur certains bateaux récents, nous retrouvons le bus NMEA200 [3], un dérivé du protocole CAN, qui remplace de plus en plus les protocoles propriétaires. Pour des besoins en bande passante importants engendrés par l'ajout de radars ou de sondeur, ces bus sont parfois complétés par un câble Ethernet supplémentaire (par exemple le système Furuno Navnet [10], ou SeaTalkHS de Raymarine [22]).

Il est important de noter que l'approche par Ethernet est avide en ressources, coûteux et difficile à adapter à des microcontrôleurs 8 bits simples comme l'Arduino. Cependant, s'inspirer des systèmes existant utilisant NMEA2000 pourrait être une bonne solution dans l'élaboration de notre système. Le seul problème, et il est de taille, est que NMEA2000 est basé sur le protocole J1939, un protocole propriétaire [33] qui ne convient pas à notre démarche open-source. La solution serait un protocole, open-source, adapté

aux Arduino, qui permettrait de concevoir et d'intégrer le nouveau matériel.

Si l'on s'abstrait du milieu marin, d'autres types d'architectures utilisant des bus de terrain existent déjà et certaines sont dans des stades assez avancés. Nous pouvons par exemple mentionner le travail de recherche de Akin Avcı lors de sa thèse à l'université de Bilkent en Turquie [4]. A travers l'étude des différents moyens de communications existants dans les microcontrôleurs (USB, FireWire, CAN, PCI, ...) il a été en mesure de développer un bus qui à son sens utilise les moyens de communications les plus simples et plus efficaces. Son architecture, appelée Bus Universel pour les Robots, a été implémentée sur des microcontrôleurs 8051 développés par les Laboratoires Silicon (Si-Labs) et utilise les communications UART (RS232) et SMBUS (I2C). Ses principaux critères de sélection concernent la facilité de mise en œuvre, la robustesse et l'adaptabilité à des microcontrôleurs de faible puissance. Selon lui, même si le protocole CAN est un des protocoles le plus souvent utilisé pour des applications en temps réel, la plupart des détails concernant la couche application du protocole est laissé au soin de l'utilisateur, ce qui le rend coûteux et délicat à implémenter. Il prend l'exemple d'une donnée supérieure à 8 octets qui doit engendrer un découpage afin de pouvoir être envoyée dans plusieurs trames différentes, ce qui exige un protocole de plus haut niveau.

Cependant le protocole CAN possède de sérieux atouts que nombre de ses concurrents ne disposent pas. En plus de pouvoir choisir la vitesse de transmission des données (jusqu'à 1Mbit/s), CAN intègre un système de détection d'erreur très perfectionné. Utilisant un champ CRC pour contrôler l'intégrité des données transmises, ce système de détection permet un taux d'erreur enregistré très faible (inférieur à $4,6.10^{-11}$). De plus, le réseau est capable de différencier les erreurs ponctuelles des erreurs redondantes. Ainsi, tout périphérique défaillant peut être déconnecté du réseau afin de limiter les perturbations. Le réseau entre alors en mode « dégradé ». Un champ d'acquittement, certes perfectible (voir 4.2.2 Fonctionnement pratique), est capable de signaler les erreurs de transmissions. Enfin, la communication multipoints permet d'envoyer un message à un ou plusieurs destinataires grâce à la notion d'identifiants, de filtres et de masques.

Tous ces avantages sont une des raisons qui font que nous avons privilégié l'utilisation du protocole CAN dans notre architecture (voir 4.3 Choix de l'architecture). Couplé à une couche protocole de plus haut niveau, SimpleCAN, cela nous permet de bénéficier d'un bus capable de fonctionner en synchronisation entre les différents nœuds, d'avoir une notion de « heartbeat » pour détecter les déconnexions, et d'une communication maître/esclave modulaire avec un système d'élection dynamique. Nous allons désormais voir en détail l'implémentation de ce bus de terrain au sein de notre architecture et les fonctionnalités apportées.

Chapitre 4

Controller Area Network

Cette présente partie est consacrée au protocole CAN. Son histoire sera brièvement résumée pour expliquer les raisons qui ont poussées Bosch à développer un tel bus de terrain et ce qui en a fait son succès. Nous verrons également son principe de fonctionnement dans les grandes lignes pour comprendre notre architecture et les termes utilisés par la suite. Enfin nous justifierons le choix de son utilisation plutôt qu'un autre type de système.

4.1 Historique

Afin de satisfaire les exigences de plus en plus importantes de la part des clients en matière de sécurité et de confort au sein d'un véhicule, l'industrie automobile a dû développer de nouvelles commandes électroniques : systèmes anti-patinage, contrôle électronique du moteur, de l'air climatisé, fermeture centralisée des portes, etc... Cependant, une telle complexité a un coût. En effet ces systèmes nécessitent d'échanger des données entre eux ce qui signifie une augmentation du nombre de connexions nécessaires pour rendre cet échange possible. Outre le prix d'un tel câblage, la place qui lui est nécessaire peut rendre la cohabitation de ces systèmes impossible. De plus, une telle complexité dans les branchements engendre à fortiori de sérieux problèmes de fiabilité et de réparation.

C'est cette raison qui a amené Bosch, un important équipementier automobile, à fournir des solutions à ce type de problème. Ainsi est né le bus CAN dans le milieu des années 80. Entièrement défini par l'entreprise Allemande, Bosch a cependant autorisé de nombreux autres fabricants à développer des composants compatibles CAN. De part son adoption massive par ces industries, le faible coût des microcontrôleurs possédant des interfaces CAN et leur accessibilité grandissante, cette technologie est rapidement devenue un protocole réseau courant. Utilisable à des débits très importants (jusqu'à 1 Mbits/s), disposant d'un haut niveau de fiabilité pour un coût très faible, nombreux sont les facteurs qui ont fait sa réussite. De plus, il n'a cessé de migrer dans de nom-

breux autres types d'applications, pas toujours liées au domaine de l'automobile, au cours de ces dernières années, ce qui a engendré l'exigence d'un protocole de plus haut niveau, standardisé, qui fournit un système fiable d'échange de messages avec un moyen de détecter, configurer et utiliser les nœuds.

Ainsi, plusieurs protocoles CAN de haut niveau ont émergé comme SAE J1939 utilisé dans NMEA 2000, CANKingdom, DeviceNet ou encore CANopen. Bien que chaque protocole fût développé dans un but particulier, CANopen est le protocole le plus populaire pour les applications de réseaux embarqués et est largement utilisé dans différents types d'applications. De cette initiative est également né un groupe international, CAN in Automation (CiA) [7] regroupant aussi bien les utilisateurs que les fabricants qui développent et soutiennent les architectures CAN de haut niveau. Ces membres initient et développent des spécifications qui sont ensuite publiées comme normes CiA. Elles portent aussi bien sur les définitions de la couche physique du protocole que sur la couche applicative CANopen et les profils de périphériques connexes.

4.2 Généralités sur le protocole

Cette partie présentera brièvement les savoirs acquis au cours de ce projet concernant le protocole CAN en lui-même, ses principes et son fonctionnement. Issue de la littérature, des recherches bibliographiques menées et des résultats obtenus au cours de ce projet, elle n'a pas pour but de se substituer aux nombreux manuels existants sur le protocole, mais de proposer des bases pour la compréhension de notre architecture.

4.2.1 Modèle réseau

Le CAN, comme tout autre protocole réseau, s'intègre à l'intérieur de la norme ISO/OSI (ISO pour International Standards Organization, OSI pour Open Systems Interconnection). Elle permet de définir 7 couches différentes qui couvrent l'ensemble d'un protocole : la couche physique, la couche liaison, la couche réseau, la couche transport, la couche session, la couche présentation ainsi que la couche application qui elle permet de définir les services du protocole. Dans le protocole CAN, seule les couche physique, liaison et application sont décrites (Fig. 4.1).

Couche physique

Il existe deux normes concernant la couche physique, l'ISO 11898-3 (anciennement ISO 11519-2) qui définit une vitesse de transmission de 0kbits/s à 125kbits/s inclus (Low-Speed) et l'ISO 11898-2 qui définit des vitesses de 125kbits/s à 1Mbits/s (High-Speed). Le protocole CAN à lui seul permet de décrire la représentation détaillée d'un bit, cependant tout ce qui concerne le moyen de transport des messages et le niveaux des signaux échangés ne sont pas décrits naturellement, ce qui permet à l'utilisateur de pouvoir les optimiser selon l'application et le besoin.

ISO/OSI layers		CAN system layers			
8	User application	CANOpen	CAN Kingdom	Device Net	SDS(Smart Distributed System)
7	Application	CAL:CAN Industrial applications	CAN Kingdom	Device Net specification	SDS specification
6	Presentation	Not defined			
5	Session				
4	Transport				
3	Network				
2	Data link	LLC(ang. Logical Link Control) MAC(ang. Medium Access Control) Bosch specification: CAN2.0A and CAN2.0B			
1	Physical	Standards: “Low-Speed CAN” ISO11519-2 “High-Speed CAN” ISO11898			

FIGURE 4.1 – Exemple de modèle OSI pour les protocoles CANopen, CANKingdom, DeviceNet et SDS.

Couche liaison de données

Cette couche du modèle ISO/OSI est divisé en deux sous-couches. D'une première part, nous avons la sous-couche LLC (Logical Link Control) qui effectue le filtrage des messages sur le bus, la notification des surcharges (overload) et la procédure de recouvrement des erreurs. D'une seconde part, nous avons la sous-couche MAC (Medium Access Control), au cœur du protocole CAN, qui effectue la mise en forme du message à l'intérieur d'une trame, l'arbitrage sur le bus, l'acquittement des messages ainsi que la détection des erreurs et leurs signalisations. Elle est en concordance avec les spécifications CAN2.0A et CAN2.0B qui définit le format standard ou étendu du champ identifiant.

Couche application

Elle donne aux applications le moyen d'accéder aux couches inférieures et sa spécification est laissée à l'utilisateur.

4.2.2 Fonctionnement pratique

Les messages CAN envoyés sur le bus sont construits suivant une forme bien spécifique. De plus, on peut distinguer deux types de trames que sont les trames de données (Data Frame, Fig. 4.2) et les trames de requête (Remote Frame, Fig. 4.3).

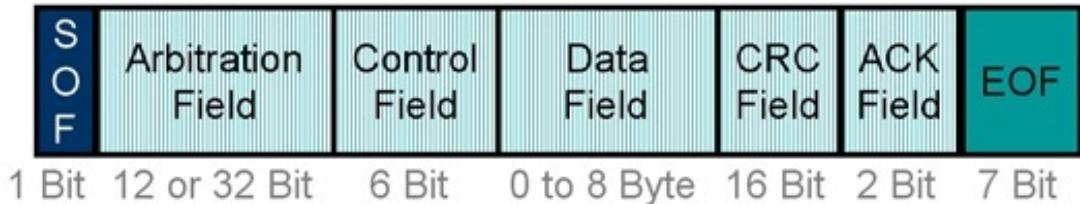


FIGURE 4.2 – Représentation d'une trame de donnée selon la spécification du protocole CAN.

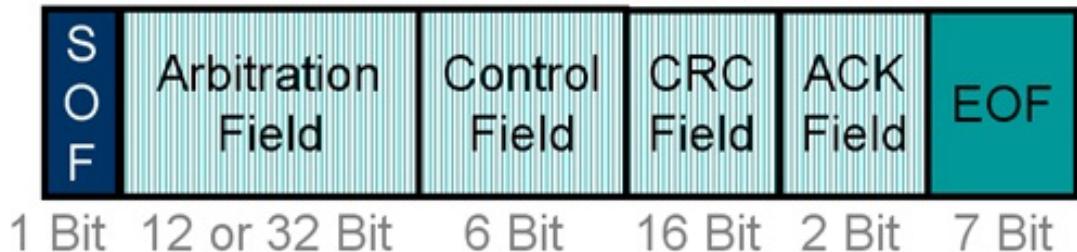


FIGURE 4.3 – Représentation d'une trame de requête selon la spécification du protocole CAN. On remarque qu'elle ne contient pas de champ de données.

Les trames de données sont produites par un nœud du bus lorsque celui souhaite transmettre des informations ou, si cela lui a été demandé par un autre nœud. A l'intérieur de celle-ci l'on peut véhiculer jusqu'à 8 octets de données. Les trames de requête sont émises lorsqu'un nœud souhaite accéder à des informations détenues par un autre nœud. Celui-ci émet alors une trame, comportant le même identifiant que la donnée auquel il souhaite accéder, à la source. Une trame de données sera émise en réponse à cette requête à distance.

Il y a deux différences fondamentales entre ces deux types de trame. Premièrement, le bit RTR (voir plus loin Arbitrage) est transmis comme un bit dominant ('0') dans une trame de données et comme un bit récessif ('1') dans le cadre d'une requête. De plus, cette dernière ne comporte pas de champ de données. La spécificité du bus CAN réside dans la priorité des messages caractérisée par un identifiant, supposé unique, des trames qui transitent. En effet, plus l'identifiant sera faible, plus sa priorité sera importante sur les autres messages. Cependant, dans le cas où une trame de données et une trame de requête avec le même identifiant sont émises au même moment il pourrait y avoir conflit d'arbitrage. Ce cas de figure est contourné par le bit RTR qui complète le champ de

l'identifiant. Dominant dans le cas de la trame de données, il lui permet de remporter l'arbitrage et de transmettre immédiatement les données souhaitées. La description des différents champs que comporte une trame CAN est détaillée ci-dessous :

Le bit de démarrage Il indique le début d'une trame de données ou une trame de requête par un bit dominant. Une transmission peut se produire uniquement lorsque le bus est libre, puis les autres nœuds se synchronisent sur le bit de démarrage du nœud qui commence la communication.

Le champ d'arbitrage Il est composé de deux parties : l'identifiant et le bit RTR. La longueur de l'identifiant dans le format standard est de 11 bits et correspond à l'ID de base du nœud. Il est suivi par le bit de RTR, dominant pour les trames de données et récessif pour les trames de requête. Dans le format étendu, l'identifiant comporte deux sections : l'ID de base avec 11 bits et l'ID étendue de 18 bits. Un exemple d'arbitrage sur le bus est donné figure 4.4.

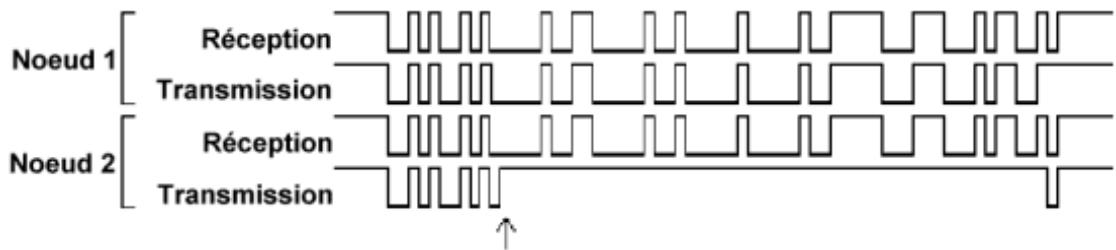


FIGURE 4.4 – Arbitrage sur le bus entre deux nœuds. L'on remarque que le nœud 2 perd la priorité d'émission au profit du nœud 1 qui contient un bit dominant '0' contre un bit récessif '1' pour le deuxième nœud. Sa transmission est donc interrompue jusqu'à ce que le nœud 1 ait terminé.

Le champ de contrôle Il est composé de deux parties. Les deux premiers bits sont réservés alors que les quatre autres indiquent la longueur du champ de données.

Le champ de données Il contient les données envoyées d'un nœud sur le bus.

Le champ CRC Le champ est composé de 15 bits de CRC (Cyclic Redundancy Check) et d'un bit délimiteur toujours récessif. Le CRC est calculé à partir de l'ensemble des champs afin de constituer le polynôme $f(x)$ lui-même multiplié par 2^{15} puis divisé par le polynôme $g(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$. Le reste de la division, modulo 2, constitue la séquence de CRC qui est transmise en même temps que le message. Une erreur CRC est détectée si le résultat du calcul n'est pas le même que celui reçu dans la séquence de CRC. Dans ce cas, le récepteur ignore le message et envoie une trame d'erreur pour demander la retransmission du message.

Le champ d’acquittement Le champ est composé d’un bit d’acquittement et d’un bit délimiteur toujours récessif. Un nœud récepteur qui n’a pas eu d’erreur de CRC précédemment émet un bit dominant pendant la durée du bit ACK, ce qui permet au nœud émetteur de savoir qu’au moins un des nœuds sur le bus a reçu le message. Dans le cadre d’une architecture multi-nœuds, tout autre nœud qui serait en désaccord avec ce jugement ne peut en aucun cas forcer ce bit à 1 (impossibilité électrique), donc elle détruit la trame en produisant une trame d’erreur. Il faut cependant nuancer la puissance de ce champ d’acquittement qui ne permet pas d’affirmer que le destinataire a bien reçu la trame car il peut être temporairement déconnecté du réseau. Il est simplement la preuve que la trame qui circule est valide, et qu’au moins un nœud a validé cette structure physique.

Le bit de fin de trame Chaque trame qui circule est délimitée par une séquence de sept bits récessifs.

Si ces champs peuvent paraître nombreux, la majorité d’entre eux sont directement gérés par le protocole bas niveau du CAN. L’utilisateur souhaitant transmettre des messages par le biais de ce bus se contentera de spécifier l’identifiant du message ainsi que le champs de donnée et sa taille si cela est nécessaire. Il devra également spécifier s’il s’agit d’une trame de donnée ou de requête, et s’il utilise des identifiants standards (11bits) ou étendus (29bits).

4.3 Choix de l’architecture

En raison de la complexité des tâches pour lesquelles les robots mobiles sont conçus, ceux-ci utilisent généralement des ordinateurs embarqués avec une puissance de calcul et de traitement beaucoup plus importante que de simples cartes électroniques et leurs architectures sont souvent centrées autour de l’un d’eux. La connexion des différents éléments matériel au robot (capteurs, actionneurs, ...) est faite en utilisant toutes sortes d’interfaces disponibles sur l’ordinateur : ports USB, port Ethernet, ports série (RS232, RS435, RS485), ou par l’utilisation de hubs et/ou de concentrateurs (par exemple, NASA ou Roboat [28]) le cas échéant. En utilisant des microcontrôleurs à la place de certains concentrateurs, on dispose également d’un accès à toutes sortes d’interfaces électroniques de niveau inférieur comme l’I2C ou le SPI (NAO) [25].

Dans sa configuration la plus simple, notre système se compose uniquement d’un joystick et d’un actionneur pour le gouvernail (vérin). Dans cette configuration, il n’est pas nécessaire d’utiliser un ordinateur embarqué car le système ne met en œuvre que des comportements réactifs de bas niveau (contrôle du cap à l’aide du compas ou de la direction à l’aide du joystick). Un simple microcontrôleur peut aisément exécuter ces tâches, à faible coût tout en ayant une consommation électrique très faible.

Pour ce qui concerne les architectures centrées autour de matériels informatiques embarqués, tout doit être relié à ce qu’on pourrait appeler une microcentrale, un or-

gane centralisant les ressources. Cependant les interfaces disponibles sont plus rares et le nombre de capteurs et d'actionneurs qu'il est possible de connecter est souvent très limité. Ce type d'architecture centralisée peut vite devenir irréalisable lorsque plusieurs composants doivent être reliés à un microcontrôleur simple de type microcontrôleur 8 bits utilisé dans les Arduino.

C'est alors qu'apparaît le choix d'un système de bus qui offre une solution souple et modulaire pour interconnecter différents microcontrôleurs, chacun responsable de capteurs et/ou d'actionneurs. Actuellement, la liaison RS-485 utilisée en liaison multipoints est de plus en plus remplacée par le bus CAN d'un niveau plus élevé, désormais intégré matériellement dans une majorité de microcontrôleurs. Non plus limité au milieu automobile et industriel, CAN est aujourd'hui utilisé dans les robots et permet de créer des architectures matérielles hautement évolutives, comme par exemple celle de Merten et Gross [21] qui utilisent le protocole CANopen. Différentes implémentations en open-source de ce protocole existent comme CANFestival, CANopen SlaveLib et CANopenNode.

De ce fait, nous avons retenu le bus CAN pour notre architecture (Fig. 4.5), mais nous avons également décidé de développer un système capable de travailler avec des noeuds très simples et d'un réseau capable de fonctionner sans ordinateur embarqué. D'après les recherches que nous avons pu mener, les trois implémentations de CANopen citées ne permettent pas de travailler sans un ordinateur embarqué ou du moins d'un microcontrôleur de plus hautes performances que nos cartes Arduino. La pièce manquante à notre architecture semble être un protocole CAN, open-source et assez simple pour fonctionner sur des microcontrôleurs 8 bits. Il devra également fournir les services nécessaires pour les capteurs et actionneurs répartis sur le bus, comme par exemple synchroniser certaines tâches comme la lecture de capteur et la réalisation des boucles de commande. Pour cela nous avons décidé de développer notre propre protocole, SimpleCAN, afin de répondre à ces exigences élémentaires. Son fonctionnement sera détaillé plus tard dans ce rapport.

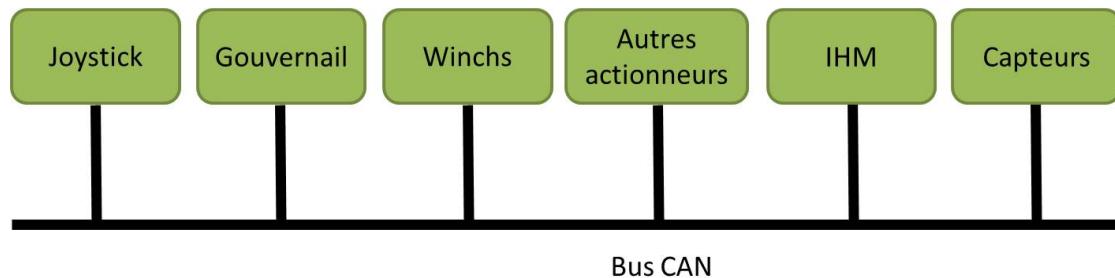


FIGURE 4.5 – L'architecture de notre système construite autour du bus CAN.

Chapitre 5

Une architecture CAN compatible Arduino

Cette présente partie du rapport est consacrée au développement d'une architecture CAN compatible avec nos cartes électroniques Arduino. Ce projet ayant commencé bien avant le début officiel du projet de fin d'études par le biais de « l'application système » (voir 5.1.2 Les premiers démonstrateurs), il apparaît important de relater en premier lieu les résultats préliminaires et leurs conclusions, tout en présentant également le système d'assistance à la navigation initial développé par Splashelec. Nous verrons ensuite comment cette première architecture fut améliorée pour mieux répondre à nos exigences, et la nouvelle fonctionnalité apportée par l'ajout d'un bootloader CAN permettant de reconfigurer directement les noeuds à travers le bus.

5.1 Résultats préliminaires

Développé par Splashelec, le système d'assistance à la navigation handivoile est apparu à l'ENSTA Bretagne pour la première fois lors d'un projet en collaboration au cours de ma deuxième année d'études. A l'époque, le principal objectif était d'y apporter différents capteurs et une interface Homme-Machine afin d'augmenter le degré d'autonomie du navigateur handicapé en lui fournissant des informations visuelles par le biais d'une tablette ou d'un smartphone. La problématique de câblage est rapidement apparue lorsqu'il a fallu interconnecter notre système avec la carte électronique initiale. Ainsi est née l'idée de développer un système de bus permettant d'interconnecter plusieurs noeuds afin de faciliter les connections et d'augmenter la fiabilité du système en cas de panne.

5.1.1 Le système Splashelec

Le système élaboré par Splashelec est en fait composé de deux parties principales : le servo-contrôleur programmable (PSC) et l'IHM, qui affiche les informations des cap-

teurs et permet au skipper d'activer le joystick ou d'utiliser le pilote automatique. La communication entre le bus et la tablette est réalisée par Bluetooth, tandis que la communication interne sera entièrement réalisée en utilisant le protocole CAN. Son comportement peut être assimilé à un système en boucle fermée (Fig. 5.1). Les informations qui proviennent de l'environnement, comme les interventions humaines, ont une influence sur les actionneurs et le système sera en mesure de réaliser des tâches automatiques. Ces tâches peuvent être par exemple le suivi de cap à l'aide du pilote automatique avec possibilité de limiter la gîte, ou le contrôle de la direction à l'aide du joystick.

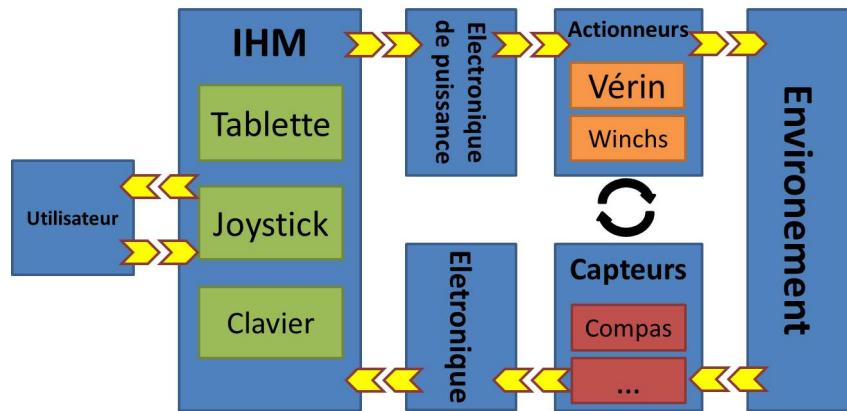


FIGURE 5.1 – Le système est assimilé à une boucle fermée où l'utilisateur peut être retiré pour un comportement autonome.

Le servo-contrôleur programmable (Fig. 5.2) est un composant clé du système et peut réaliser le travail d'un calculateur de pilote automatique (navigation au vent ou à l'aide du compas suivant le mode et les capteurs associés). La carte électronique, relativement compacte (6 x 7,5 pouces), contient un microcontrôleur, l'électronique de puissance pour un moteur électrique (vérin ou winch), une commande électrique et l'alimentation. Le système ainsi créé est visible figure 5.3. Différentes interfaces pour le capteur d'angle du gouvernail, le joystick et le clavier de contrôle existent déjà et dans les futures versions, le circuit imprimé principal intégrera directement une interface pour bus CAN. Pour ajouter les winchs électriques, nous utiliserons un servo-contrôleur pour chacun d'entre eux.

Entièrement open-source, il permet l'intégration facile de nouvelles fonctionnalités ou de modifier celles existantes. Le microcontrôleur est compatible avec les cartes Arduino et peut donc être programmé avec environnement de développement intégré, l'Arduino IDE, qui donne accès à l'interface de programmation, les bibliothèques existantes et divers exemples en ligne. Les résultats de ce projet (matériel et logiciel) sont accessibles directement depuis Internet [29].

Reliés à cette carte, plusieurs capteurs comme la centrale inertie, le capteur de vent et le sondeur bénéficient d'une architecture CAN pour communiquer. Lorsque plusieurs

actionneurs seront présents, chacun utilisera un servo-contrôleur avec son électronique de puissance associée, ses capteurs locaux, et une interface bus CAN. Ainsi, tout le système (y compris l'ensemble des pièces mécaniques), est transportable et peut s'adapter à différents bateaux.



FIGURE 5.2 – Le servo-contrôleur programmable de Splashelec dans un boîtier étanche. Il contient plusieurs entrées pour le joystick, le compas, le capteur d'angle de barre, la batterie, etc...

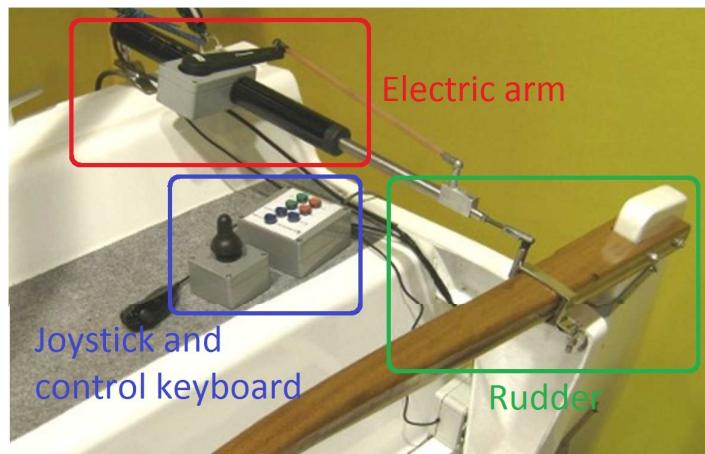


FIGURE 5.3 – Le système Splashelec est composé d'un vérin électrique (en rouge), d'un joystick et d'un clavier de contrôle (en bleu) et du gouvernail (en vert).

L'IHM, réalisée l'année précédente, est programmée en Android à l'aide du kit de développement logiciel Android (SDK) pour Eclipse [8]. Elle contient différentes zones d'information (Fig. 5.4), à la fois textuelles (vitesses du vent, du bateau, température,...) mais également graphiques pour certaines données (orientation du vent et sa vitesse, angle du compas et angle de la barre) afin de permettre une meilleure compréhension de l'environnement par le skipper. Enfin une zone agit comme un clavier virtuel pour que l'utilisateur puisse entrer le cap à suivre et basculer entre le pilote automatique et

le mode joystick. La réalisation graphique de l'IHM est réalisée en langage XML et elle peut être facilement modifiée pour s'adapter aux besoins spécifiques des utilisateurs. Il en résulte un système (joystick, tablette, clavier déporté, ...), compatible avec différents types de handicaps mais aussi utilisable par des skippers valides.

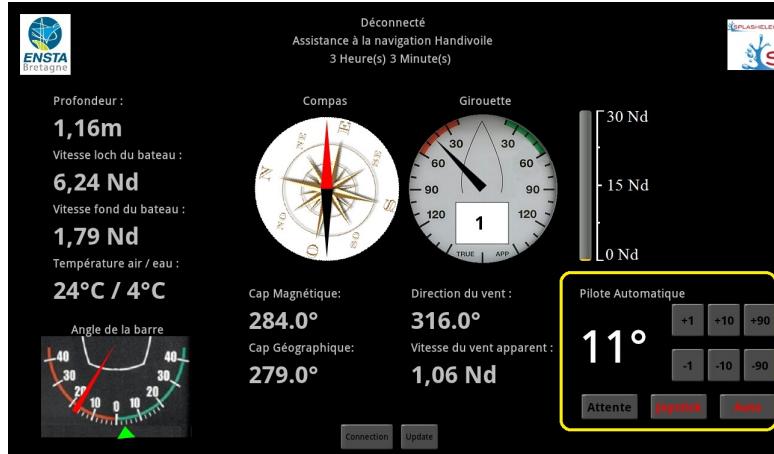


FIGURE 5.4 – L'IHM affiche les différentes données capteurs circulant sur le bus et possède un clavier virtuel (en jaune) pour sélectionner les différents modes de fonctionnement (joystick, pilote automatique).

5.1.2 Les premiers démonstrateurs

L'ENSTA Bretagne a la spécificité de proposer à ses étudiants de 3ème année de démarrer leur sujet de projet de fin d'études en avance par le biais d'une phase d'avant-projet. Appelée « Application système », elle a lieu en fin de semestre et permet de poser les premières bases afin de débuter sereinement le PFE. Dans le cadre de cette phase préliminaire, je me suis attelé à reprendre nos travaux précédents sur l'architecture de Splashelec afin de décentraliser l'unique carte électronique présente en de multiples nœuds, chacun responsable d'un ou de capteurs spécifiques, reliés par le biais d'un bus CAN. Les différentes recherches bibliographiques sur le sujet ainsi que les rencontres avec les enseignants de l'école familiers avec ce type de communication m'ont permis de mieux cerner le fonctionnement de cette architecture et d'en dresser les fonctionnalités générales (voir 4. Controller Area Network).

Cette étude a débouché sur la réalisation de deux démonstrateurs permettant de mettre en avant les connaissances acquises. Basés sur une architecture à deux nœuds, bâties à l'aide de deux cartes électroniques Arduino UNO ainsi que d'un shield CAN de chez Sparkfun utilisant le microcontrôleur CAN MCP2515, le premier démonstrateur a permis d'établir un « jeu » entre les deux cartes en se basant sur les fonctionnalités fournies par les filtres du MCP2515 (voir 6.2.3 Filtrage sur le bus) et le second de dialoguer entre la tablette Android et le vérin de barre par le biais du bus. Leur fonctionnement

est décrit en détail ci-dessous. La base du protocole CAN sur Arduino provient de la librairie CANduino permettant l'accès à ses fonctionnalités (envoi, réception de message, définition des filtres, changement de mode, ...). Relativement basique, cette librairie fut remplacée par la suite.

Le jeu de communication

Ce jeu de communication repose sur deux nœuds : un récepteur et un transmetteur. Le récepteur choisit aléatoirement un nombre entre une valeur minimum et une valeur maximum afin de régler le filtre de son premier buffer de réception sur celui-ci. L'expéditeur se doit de découvrir ce nombre afin de pouvoir communiquer avec lui. Pour cela il va au hasard faire des suppositions sur celui-ci et régler l'identifiant de son message avec. Lorsque la supposition s'avère juste, le récepteur est en mesure de recevoir le message et de lui répondre. Par la suite, les deux microcontrôleurs échangent leur rôle. Ce jeu relativement basique permet de manipuler la gestion des filtres, ce qui permet de soulager la charge du microcontrôleur de l'Arduino car le traitement se fait en interne par le MCP2515. Le résultat de ce démonstrateur est visible figure 5.5.

Figure 5.5 shows the Arduino IDE Serial Monitor window titled "COM5". The text in the window is annotated with red lines pointing to specific parts of the communication flow:

- "Define role..." and "I'm the sender" are annotated with a red line pointing to the first message from the sender.
- "1,Retry", "6,Retry", "5,Retry", "3,Retry", "8,I think I got it?:8", "I'm now the listennner", and "Guess has been set at: 10" are annotated with a red line pointing to the sequence of messages exchanged to determine the correct role.
- "Reading frame buffer 1", "[#1]:3, Reading frame buffer 1", "[#2]:4, Reading frame buffer 0", "[#2]:10!", and "Other party got it on filter: 0" are annotated with a red line pointing to the frames read by the receiver.
- "L'émetteur choisit aléatoirement un nombre entre une valeur MIN et MAX et l'envoie sur le bus" is a descriptive text annotation pointing to the first message.
- "Si l'estimation de ce nombre est correcte, il reçoit une confirmation du récepteur et ils échangent leur rôle." is a descriptive text annotation pointing to the role exchange messages.
- "Le récepteur n'est en mesure de recevoir le message que si le nombre estimé correspond à celui placé dans le filtre" is a descriptive text annotation pointing to the frame reading messages.

FIGURE 5.5 – Le terminal série du logiciel Arduino IDE permet de visualiser le comportement de notre jeu de communication.

Communication bidirectionnelle

Cette partie reprend les bases des travaux précédents concernant l'acquisition des données capteurs. Cependant quelques améliorations ont été faites afin de pouvoir transmettre ces données à travers une trame CAN. Par manque de temps durant cette phase d'avant-projet, seul le compas fut intégré dans le système. L'architecture utilise toujours deux nœuds distincts, un premier en charge de la transmission des données du compas et un second en charge de sa transmission par bluetooth à la tablette. Ce même nœud est également en communication I2C avec la carte principale du système Splashelec

afin de manipuler le vérin de la barre. Les résultats de ces démonstrateurs permettent de visualiser les informations directement sur la tablette (informations textuelles mais également compas interactif) et de manipuler le vérin à l'aide du clavier virtuelle de la tablette lorsque le pilote automatique est enclenché (modification du cap cible). Ce second démonstrateur, d'un intérêt plus abouti que le premier, a permis de bâtir une première architecture compatible avec le matériel de Splashelec et l'IHM de notre précédent projet. Ce fut également un grand pas franchi pour la suite du projet car les bases étaient alors maîtrisées.

5.2 Une architecture plus aboutie

Comme évoqué précédemment, la librairie CANduino utilisée durant la phase d'application système a rapidement montré ses limites et défauts. En effet nous nous sommes aperçus, après avoir finalisé le bootloader CAN, qu'il nous était impossible de faire communiquer les cartes ensemble à l'aide de cette librairie. Après de nombreux tests dans le but de comprendre cette défaillance, nous nous sommes rendu compte à l'oscilloscope que la vitesse du bus était erronée. La raison à ce problème était une absence d'écriture dans les registres du MCP2515, en charge de définir la vitesse, pour une raison restée inconnue. Ce problème n'était pas apparu auparavant car jusqu'alors les cartes utilisaient la même librairie défaillante et donc possédaient toutes la même vitesse erronée. L'ajout du bootloader CAN réalisé à l'aide d'une autre bibliothèque, celle de Fabian Greif [11] du club robotique d'Aachen en Allemagne, a mis à jour ce défaut. Afin de ne pas perdre trop de temps avec une bibliothèque trop simple et trop peu évoluée pour notre architecture future, il a été décidé de l'abandonner complètement et de ne pas consacrer plus de temps à corriger ses défauts. Tous les efforts se sont donc concentrés pour rendre compatible la bibliothèque universelle CAN de Fabian Greif avec notre matériel Arduino et disposer d'une base beaucoup plus solide et robuste pour notre futur protocole de haut niveau, SimpleCAN.

Relativement petite en taille, environ 1500 octets, elle permet de ne pas saturer la mémoire flash de l'Arduino lors de l'utilisation de cette bibliothèque dans notre code. Placée sous licence BSD [32], la moins restrictive dans le monde informatique, elle peut alors être modifiée, être incorporée dans tous types de codes déjà existants et même être utilisée à des fins commerciales ce qui cependant, rappelons-le, n'est pas l'objectif de ce projet.

Initialement conçue pour certains microcontrôleurs AVR, elle n'intégrait pas la compatibilité avec les microcontrôleurs Arduino dont celui de notre CANinterfacer (Fig. 5.6). Il a donc fallu apporter ces modifications pour qu'elle puisse être utilisée dans le logiciel Arduino IDE afin d'en utiliser les fonctionnalités CAN. Egalement bâtie sur une architecture de fichier différente (utilisation de makefile, plusieurs dépendances de fichiers,...), il a fallu unir les différents programmes pour ne former qu'un seul et même fichier (en réalité un fichier .cpp et son header associé). Pour cela, nous nous sommes dans un premier temps limités aux fonctionnalités touchant le microcontrôleur CAN

MCP2515 utilisé dans nos cartes et avons mis de côté la compatibilité avec l'AT90CAN et le SJA 1000, tous deux des microcontrôleurs AVR avec gestion du CAN intégré.



FIGURE 5.6 – CANinterfacer construit par Splashelec. Il s'agit d'une carte électronique compatible Arduino Leonardo et contenant un contrôleur CAN intégré, le MCP2515.

Une fois la nouvelle bibliothèque adaptée à nos cartes Arduino, l'étape suivante fut de reprendre nos travaux précédents concernant la reconfiguration des noeuds à travers le bus en bénéficiant cette fois ci d'une compatibilité entre le bootloader et la librairie utilisée.

5.3 Reprogrammation des noeuds

Une fonctionnalité majeure de ce nouveau système de bus est la possibilité de pouvoir se servir de messages CAN afin de pouvoir reprogrammer directement nos noeuds. Une fois en fonctionnement, les différentes cartes seront positionnées dans des boitiers étanches et toutes tentatives pour les retirer de leur emplacement comporte le risque de les endommager. Prévoir un port USB ou série directement accessible à travers le boitier est une solution envisageable, mais en multipliant les entrées nous apportons de nouvelles problématiques d'étanchéité. L'idée d'utiliser directement la liaison CAN apparait donc une solution crédible à ces contraintes.

5.3.1 Présentation

La mise en œuvre d'un protocole CAN ouvre la voie à une programmation simplifiée de chaque noeud directement par le bus, chose permise grâce à une connexion unique à celui-ci. Pour obtenir une telle fonctionnalité, nous avons besoin d'un nouveau bootloader Arduino qui remplacera celui existant et qui accepte la reprogrammation du CANinterfacer directement à l'aide de messages CAN. Fabian Greif du club de robotique d'Aachen, en Allemagne, a déjà travaillé sur ce sujet avec un matériel très similaire [11] et il a développé son propre bootloader CAN, qui permet la mise à jour du firmware de la carte ainsi que du code local de l'application. Afin de rendre compatible ce programme avec notre matériel, des modifications ont été apportées pour pouvoir ensuite l'adapter à nos CANinterfacers. Relativement petit en taille, 1010 octets, il est inférieur à 512 mots

sur une carte pouvant atteindre 2048 mots pour la section du bootloader. Cependant celui-ci n'intègre pas la gestion du l'USB, contrairement au bootloader original (i.e celui de l'Arduino Leonardo), ce qui fait qu'il est impossible de charger un sketch à l'aide du lien série, la reprogrammation se fait exclusivement par CAN ou à l'aide d'un programmeur ISP. Le défi pour faire cohabiter les deux fonctionnalités de ces bootloaders est de taille et ne sera pas abordé dans ce projet. Il reste cependant une piste pour des travaux futurs.

5.3.2 Etapes de reconfiguration

L'opération de reprogrammation à l'aide du bus CAN ne peut être réalisé sans la présence de trois éléments fondamentaux dans le processus : un script Python, un programme Arduino de conversion et un bootloader CAN. Il s'agit de trois chaînons indissociable de l'étape de programmation d'un noeud, si l'un deux venait à être défaillant, l'opération échouerai. De manière schématique, l'on peut représenter l'architecture comme ceci (Fig. 5.7).



FIGURE 5.7 – La reprogrammation d'un noeud fait intervenir les trois éléments indispensables que sont le PC contenant le nouveau programme, l'interface CAN2USB et le noeud à reconfigurer.

Nous allons maintenant revenir plus en détail sur chacune des parties.

Un programme CAN2USB Arduino

Une fois la liaison série hors service suite au remplacement du bootloader, il devient impossible de communiquer par ce moyen entre l'ordinateur et la carte. Il est donc important de disposer d'une carte intermédiaire qui se chargera de faire la liaison entre le lien série et le bus CAN. C'est ici qu'entre en jeu notre programme Arduino qui transformera une carte en convertisseur CAN vers USB et inversement afin de proposer un moyen de communication bidirectionnel. Relativement simple dans sa conception, il se chargera de recevoir des données reçues par le biais du script Python et les renverra sous forme de trames CAN à la carte cible. Un point important de cette reprogrammation est que cette dernière doit être dans sa <> section <> bootloader à ce moment précis. Il existe deux alternatives pour arriver à cet état lorsque les cartes seront en fonction et donc dans leur <> section <> de programme. La première est un <> power reset <> engendré par l'appui de l'utilisateur sur le bouton Reset de la carte afin de redémarrer celle-ci et la faire entrer dans son bootloader puis de la reprogrammer avec notre méthode. Si cette manière de procéder est relativement simple et n'exige pas une démarche complexe, elle reste néanmoins contraignante car une fois en service, nos cartes seront dans des boîtiers

étanches et l'idée de les retirer pour les redémarrer lors d'un essai en mer est simplement inenvisageable (mauvaise manipulation, chute, contact avec de l'eau de mer, . . .).

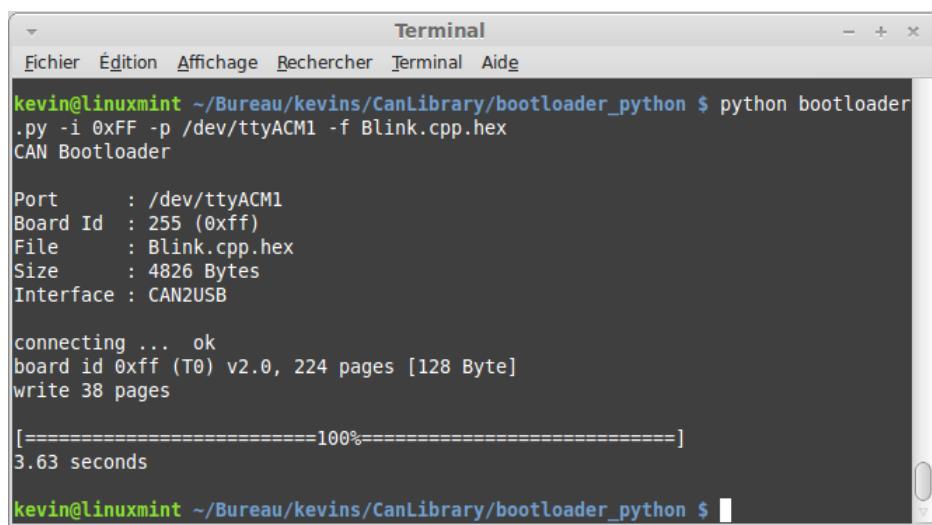
La seconde option, bien plus sûr, est ce que l'on appelle un « software reset », un redémarrage logiciel sur commande. Tout aussi efficace, il permet de s'affranchir de la manipulation de la carte et peut être intégré directement dans notre programme. Celui-ci est réalisé à l'aide d'un appel à une routine assembleur (5.1) dans laquelle la variable `BOOTLOADER_ADDRESS` doit être remplacée par l'adresse de démarrage du bootloader (voir la documentation correspondante à la carte). Pour le cas de notre CANinterfacer, nous aurons donc `BOOTLOADER_ADDRESS` = 0x3800 en hexadécimal.

asmvolatile("jmp BOOTLOADER_ADDRESS") (5.1)

Cette routine doit être présente dans tous les programmes de nos noeuds et être appelée à un moment bien spécifique afin de ne pas redémarrer abusivement la carte. Nous avons donc choisi d'émettre, avant chaque reprogrammation, un message CAN à destination de la cible comprenant tous les champs de données à 0xFF. A l'obtention de ce signal, la carte cible sait qu'une démarche de reprogrammation est enclenchée à son encontre et va redémarrer afin de pouvoir commencer le processus.

Un script Python

Le script python employé permet de palier à l'absence du logiciel IDE Arduino afin de transmettre un sketch. Nous allons en effet par ce biais transmettre le programme sous format « Intel Hex » [13] à notre carte de liaison CAN2USB en spécifiant l'identifiant de la carte à reprogrammer. La transmission se fait par lien série entre l'ordinateur et la carte reliée au bus. Un exemple de programmation réussie est montré figure 5.8.



```
Terminal
Fichier Édition Affichage Rechercher Terminal Aide
kevin@linuxmint ~/Bureau/kevins/CanLibrary/bootloader_python $ python bootloader.py -i 0xFF -p /dev/ttyACM1 -f Blink.cpp.hex
CAN Bootloader
Port      : /dev/ttyACM1
Board Id  : 255 (0xff)
File       : Blink.cpp.hex
Size       : 4826 Bytes
Interface  : CAN2USB

connecting ... ok
board id 0xff (T0) v2.0, 224 pages [128 Byte]
write 38 pages

[=====100%=====]
3.63 seconds

kevin@linuxmint ~/Bureau/kevins/CanLibrary/bootloader_python $
```

FIGURE 5.8 – Le script python lancé depuis un terminal linux témoigne de la reconfiguration réussie de la carte possédant l'identifiant 0xFF avec un sketch nommé "Blink".

Un bootloader CAN

La communication à l'aide de messages CAN est réalisée en utilisant deux identifiants standards (11 bits) : 0x7FF (pour la liaison PC-Carte) et 0x7FE (pour la liaison Carte-PC). Ces deux identifiants ont la particularité d'avoir la priorité la plus basse sur le réseau et donc perturbent peu le fonctionnement sur le bus entre les autres noeuds. Chaque carte reçoit un identifiant unique de 8 bits attribué lors de l'écriture initiale du bootloader. La raison de cette unicité est que, si l'on peut donc avoir deux cartes simultanément en mode bootloader, une seule peut être programmée à la fois, car un envoi simultané avec le même identifiant provoquerait des collisions sur le bus. L'envoi de données n'est pas automatique, il faut que la carte réponde correctement aux premières questions pour s'assurer que le noeud est bien présent et actif (demande d'identification). De manière générale, la carte n'émet aucune requête, elle se contente de répondre aux questions du PC (taille disponible pour l'application, nombre de pages inscriptibles,...). L'ensemble du processus est détaillé figure 5.9 en faisant abstraction de l'interface CAN2USB qui ne fait que convertir les messages dans un format compréhensible par les deux parties.

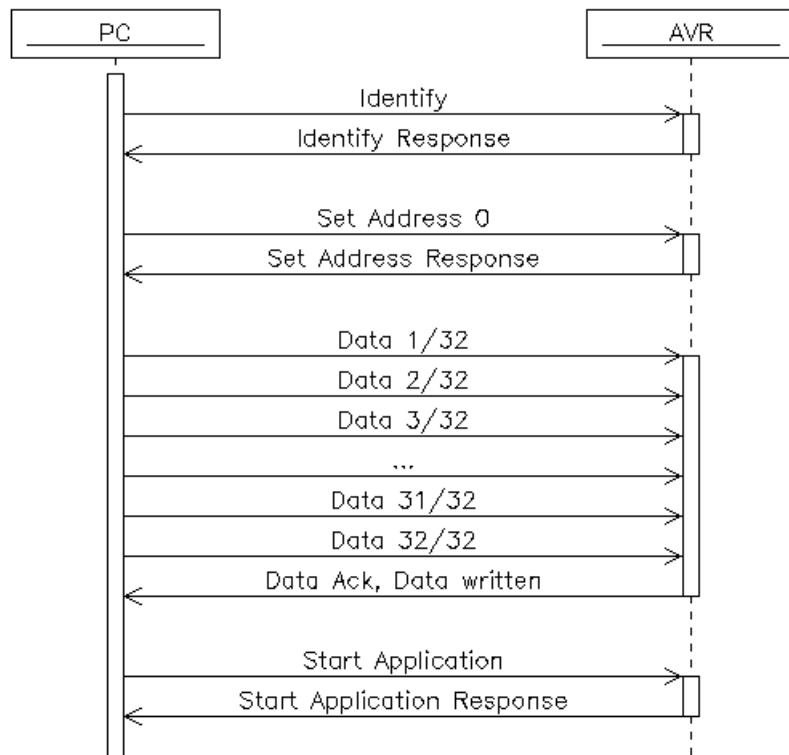


FIGURE 5.9 – La reconfiguration fait suite à une série de questions/réponses permettant de cibler puis transmettre le programme à la carte.

SimpleCAN, une couche protocole CAN de haut niveau

A cette étape du projet, nous disposons d'une librairie CAN fonctionnelle et d'un bootloader permettant de reconfigurer nos nœuds sur demande, sans utilisation du lien série. Afin de pouvoir spécifier un comportement à notre architecture, il apparaît essentiel de disposer d'un protocole de haut niveau afin de décrire des actions que le protocole bas niveau n'est pas en mesure de réaliser. Cette partie décrit l'élaboration de notre protocole CAN de haut niveau, SimpleCAN et les raisons qui nous ont amenées à développer notre propre couche réseau. En effet, nous avons vu précédemment qu'il en existait de multiples et que nous souhaitions réutiliser au maximum l'existant afin de ne pas réinventer des fonctionnalités déjà existantes, pourtant nous avons tout repris depuis la base en ce qui concerne SimpleCAN. Pour cela nous détaillerons dans un premier temps ce qui a engendré ce choix technique, puis nous verrons en détail les fonctions apportées par SimpleCAN et les raisons de leur implémentation.

6.1 Nécessité d'un tel protocole

La norme CAN définit le matériel , « la couche physique », et la communication sur un niveau de base, « la couche de liaison de données». Le protocole CAN en lui-même permet de spécifier combien de paquets de données peuvent être transportés d'un point A vers un point B en utilisant un moyen de communication partagé. Cependant, il ne spécifie rien sur le contrôle de flux, le transport de données plus importantes que ne peut contenir un message de 8 octets, les adresses réseaux, l'initialisation de la communication, etc...

Afin de gérer ces spécificités, un protocole de couche supérieure (« Higher Layer Protocol ») est nécessaire. Ce terme est dérivé du modèle OSI et de ses sept couches et spécifie des comportements à notre système (comportement au démarrage, comment distribuer les identifiants de messages entre les différents nœuds d'un système, comment

traduire le contenu de trames de données, établir le rapport d'état du système, . . .).

Il existe différents protocoles de couche supérieure : CANopen, CanKingdom, DeviceNet, CCP/XCP, J1939, OSEK/VDX, SDS et bien d'autres. Seul CANopen présente un réel intérêt dans notre démarche car il est le seul à posséder une version open-source, CanFestival [6], qui est largement utilisé à travers le monde et qui bénéficie du soutien d'une communauté solide, la CiA (CAN in Automation).

Cependant, si une implémentation d'une version « esclave » sur processeur AVR (i.e les processeurs utilisés entre autre dans les Arduino) existe, cette version est plus tournée vers des microcontrôleurs avec CAN intégré comme l'AT90CAN par exemple. De plus, CANopen est très dépendant d'un système embarqué disposant d'un OS de type ordinateur ou carte électronique Armadeus, Raspberry Pi, . . . ce dont nous souhaitons nous affranchir au maximum pour ce type de système (voir 4.3 Choix de l'architecture).

Après de longues recherches et en s'appuyant sur les avis rencontrés auprès de la communauté Arduino et des utilisateurs de CAN, il est apparu qu'il n'existe pas de démarches comme la nôtre proposant un protocole CAN de haut niveau, à la fois simple et open-source. Ainsi nous sommes alors partis dans l'idée de développer notre propre protocole, SimpleCAN, basé sur nos besoins et facilement modifiable pour y ajouter d'autres fonctionnalités. Si celui-ci n'a pas la prétention d'être au niveau des autres ténors du marché, il se veut à la fois simple et fonctionnel et propose une base pour d'autres évolutions possibles. Les différentes fonctions apportées par ce protocole sont décrites dans la section suivante de ce rapport.

6.2 Fonctionnalités apportées

Les différentes fonctionnalités que nous avons décidé d'implémenter dans notre protocole de plus haut niveau résultent des besoins précédemment établis par l'étude de notre architecture et de son comportement désiré. Du fait de la réalisation de plusieurs boucles de contrôle distribuées entre plusieurs nœuds, et de l'emplacement différent des capteurs et des actionneurs, il apparaît important de disposer d'une synchronisation sur le système afin de se prévenir des accès concurrents (« Race condition ») qui pourrait survenir. Lorsqu'on parle de temps de référence, il faut également penser à un organe maître, un nœud leader qui agira en chef d'orchestre afin de synchroniser tout le réseau sur sa propre mesure. Enfin, il est également judicieux d'exploiter le microcontrôleur MCP2515 à l'aide du filtrage de message pour alléger la charge sur le microcontrôleur principal de l'Arduino. Nous allons donc voir plus en détail la réalisation de ces tâches au sein de notre architecture.

6.2.1 Synchronisation

La synchronisation puise son intérêt dans le fait que la complexité des programmes est non uniforme (par exemple entre le calculateur et le nœud du compas), et l'on risque régulièrement d'arriver au point où l'un des nœud à fait deux tours de boucle pendant

que le premier n'en a fait qu'un seul. En synchronisant certaines actions essentielles (envoi et lecture de messages par exemple) on s'assure qu'un nœud ne prendra pas d'avance sur ces concurrents. Cela permet également de compenser la dérive naturelle des micro-contrôleurs mais aussi d'affecter une mesure temporelle aux données transitant sur le bus, de détecter les absences, etc...

Dans notre système, nous avons choisi de travailler à une fréquence de 10Hz, soit une période de synchronisation de 100ms. Bien que complexe, notre architecture reste relativement simple dans ses tâches de traitement, ce qui nous permet de travailler à des fréquences relativement faibles tout en étant certain que tous les nœuds ont fini leur traitement dans le délai imparti. Le nœud leader (voir Election de leader) comportera une tâche supplémentaire dans sa routine d'interruption puisqu'il sera en charge de l'envoi du message de synchronisation à tous les nœuds.

Pour générer cette interruption, nous nous sommes servis des timers internes à l'Arduino afin de générer une interruption synchrone appelant une routine spécifique pour chaque nœud. Cette interruption est possible en s'appuyant sur les timers internes de chaque carte. Disposant de trois timers (Timer0 et Timer2 étant des compteurs 8 bits, et Timer1 un compteur 16 bits) nous avons utilisé le Timer1 en mode CTC (Clear Timer on Compare match). Afin de pouvoir « compter » jusqu'à 100 ms, la carte utilise son cristal de fréquence 16 Mhz pour incrémenter un compteur interne. L'interruption est générée lorsque celui-ci atteint la valeur cible correspondante à 100 ms. Pour calculer cette valeur, on utilise la formule :

$$\text{Target Timer Count} = \frac{\text{Timer Clock Frequency}}{\text{Target Frequency}} - 1 \quad (6.1)$$

$$\text{Target Timer Count} = \frac{16000000}{10} - 1 = 1599999$$

Cependant le compteur a une résolution de 16 bits ce qui implique pour la valeur cible $\text{Target Timer Count} \leq 2^{16} = 65536$, ce qui n'est pas le cas ici. Pour cela, on utilise un « Prescaler » (1, 2, 8, 64, 256 ou 1024) permettant de diminuer la fréquence de notre cristal et ainsi réduire le nombre de « Ticks » nécessaire pour atteindre notre valeur cible. Afin de rendre la synchronisation la plus précise possible, nous avons conservé une résolution assez grande pour notre compteur et nous avons donc choisi le Prescaler le plus petit possible qui satisfait le critère 6.1, soit 64. Ainsi notre équation devient :

$$\text{Target Timer Count} = \frac{1}{\text{Prescaler}} \cdot \frac{\text{Timer Clock Frequency}}{\text{Target Frequency}} - 1 \quad (6.2)$$

$$\text{Target Timer Count} = \frac{1}{64} \frac{16000000}{10} - 1 = 24999$$

Ainsi, lorsque nous parlerons plus bas des temps internes $T1$, $T2_{maitre}$, $T2_{esclave}$ et T_{sync} , il s'agit de la valeur du compteur interne de la carte comprise entre 0 et 24999, et remis à zéro à chaque nouvelle période de 100 ms.

Afin de bâtir l'algorithme de synchronisation, nous nous sommes appuyés sur une

des spécificités du bus CAN qui nous assure qu'un message transmis est réceptionné en même temps par toutes les stations destinataires (sauf cas exceptionnels, si déconnection d'un noeud par exemple, mais cela n'affecte pas les autres). Ainsi, dans la figure 6.1, un seul esclave est représenté par mesure de lisibilité, mais les autres noeuds auront le même comportement. Nous allons donc influencer, à l'aide d'un correcteur PID, le temps de latence entre l'émission réussie d'un message par le leader et la réception du message par les autres stations. En effet, le leader enregistrera son temps local lorsque son message de synchronisation aura correctement quitté son buffer d'émission, valeur qu'il émettra aux autres stations à la période suivante. Les stations réceptrices enregistreront leur temps local à la réception de celui-ci et appliqueront le correcteur afin de faire coïncider ces deux mesures.

Ainsi, si l'on note T_{max} le temps maximum entre deux messages de synchronisation valides, $T2_{maître}$ le temps mesuré par le maître à l'envoi, $T2_{esclave}$ le temps mesuré par l'esclave à réception et δ la dérive temporelle de l'esclave (environ 10 us/s), le protocole garantit, dans le cas d'un système idéal, que

$$|T2_{esclave} - T2_{maître}| < 2.T_{max} * \delta = 2us \quad (6.3)$$

Dans la réalité, nous avons des incertitudes de mesures sur notre système. Pour cela, nous allons noter δ_m et δ_s l'erreur temporelle maximum de mesure sur émission pour le maître et de réception pour l'esclave. La formule précédente devient :

$$|T2_{esclave} - T2_{maître}| < 2.T_{max}.\delta + \delta_m + \delta_s \quad (6.4)$$

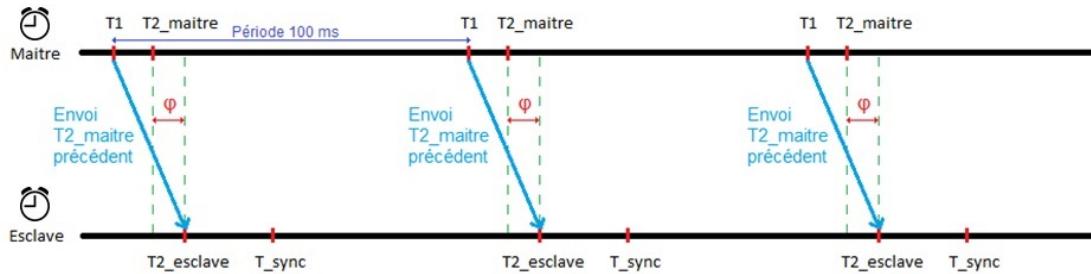


FIGURE 6.1 – La synchronisation des noeuds se fait à l'aide d'un message spécifique directement envoyé par le maître à l'instant $T1$, contenant la mesure $T2_{maître}$ de la période précédente. L'objectif est de faire coïncider $T2_{esclave}$ avec à l'instant T_{sync} à l'aide d'un régulateur PID.

A l'aide de cette synchronisation, nous avons obtenu une précision de l'ordre de la vingtaine de microsecondes (Fig. 6.2). Pour se faire nous avons utilisé deux correcteurs PID dont les coefficients ont été déterminés à l'aide de la règle de Zeigler-Nichols. Il en résulte que la principale source d'imprécision provient des termes $\delta_m + \delta_s$ (retard dans l'exécution des interruptions, horloge imprécise,...) et est donc très dépendante du matériel utilisé.

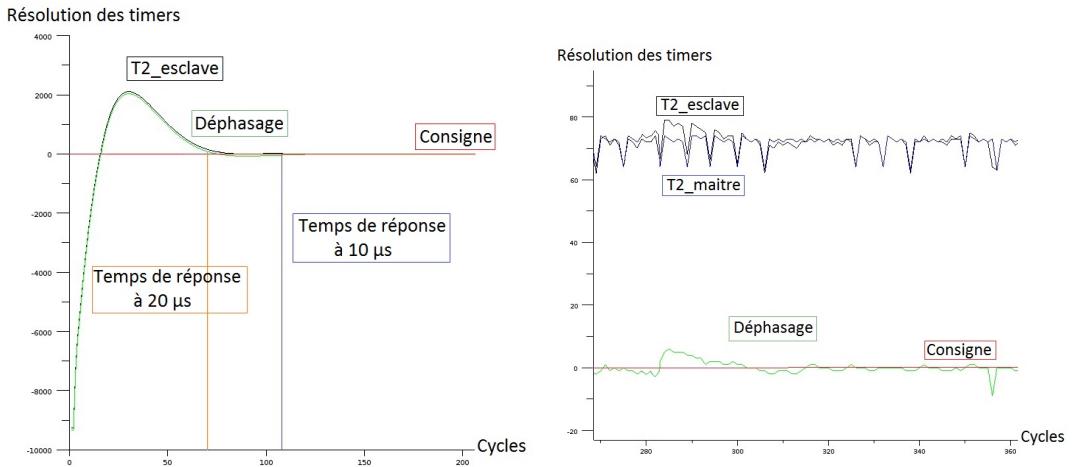


FIGURE 6.2 – Représentation graphique de l'évolution du déphasage entre $T2_{maître}$ et $T2_{esclave}$ sur une consigne nulle. Sur la figure, un cycle représente une période de 100 ms, et la résolution des timers correspond au compteur interne de la carte.

Afin de proposer un système fiable et de ne pas laisser la moindre erreur fausser nos données de mesures, nous avons alloué un des trois buffers d'émission disponible sur le MCP2515 uniquement à l'émission des messages de synchronisation. De plus la priorité de ces messages est bien plus haute que n'importe quel autre message de données, ce qui garantit que les noeuds esclaves reçoivent le message le plus tôt possible après son émission.

Si ce type de système supporte très bien une défaillance sur l'un des noeuds esclave, le mécanisme tout entier s'effondre si le noeud maître n'est plus opérationnel. Pour cela nous nous sommes appuyés sur un algorithme d'élection de leader persistant qui est en mesure de garantir une réélection en cas de perte du maître. Son fonctionnement est détaillé plus bas.

6.2.2 Election de leader

Si nous avons démontré qu'un leader est un élément indispensable à notre architecture, il apparaît également que de son fonctionnement dépend celui de l'architecture tout entière. Ainsi l'on est en droit de se demander si le choix irrévocable du maître est une solution judicieuse dans un environnement décentralisé. A cette fin nous avons donc décidé d'instaurer un mécanisme d'élection dont l'issu dépendrait de l'identifiant, unique, de chaque carte. Dans ce cas de figure, une première élection est réalisée à l'initialisation du système, et un leader est élu sur la base de l'identifiant le plus faible du réseau.

En charge de la synchronisation sur le réseau, il sera également le « fournisseur officiel » d'un temps de mesure sur le bus, le timestamp, qui accompagnera les données qui

transitent. Ainsi, si le maître venait à avoir une défaillance qui l'empêche d'émettre les messages de synchronisation, les esclaves remarqueraient une absence de rafraîchissement de ce timestamp et procéderaient à une nouvelle élection afin de remplacer le leader défaillant. Si ce fonctionnement permet de maintenir le système en vie, même après une panne sur un noeud crucial, le processus d'élection est d'autant plus gourmand en ressources que le nombre de noeuds présents est important, et le bus sera le plus souvent saturé jusqu'à la fin de l'élection. Il convient également, comme nous allons le voir dans le paragraphe suivant, à configurer les filtres de manière à ce que toutes les cartes soient toujours en mesure de recevoir ces messages sous peine de surprises (plus d'un leader, voir aucun).

6.2.3 Filtrage sur le bus

Nous opérons sur des microcontrôleurs 8 bits, suffisant pour les tâches que nous avons à réaliser mais tout de même relativement petits en terme de puissance de traitement. Il faut donc veiller à limiter autant que possible les traitements inutiles pour améliorer les performances du système : c'est là qu'intervient le filtrage des messages.

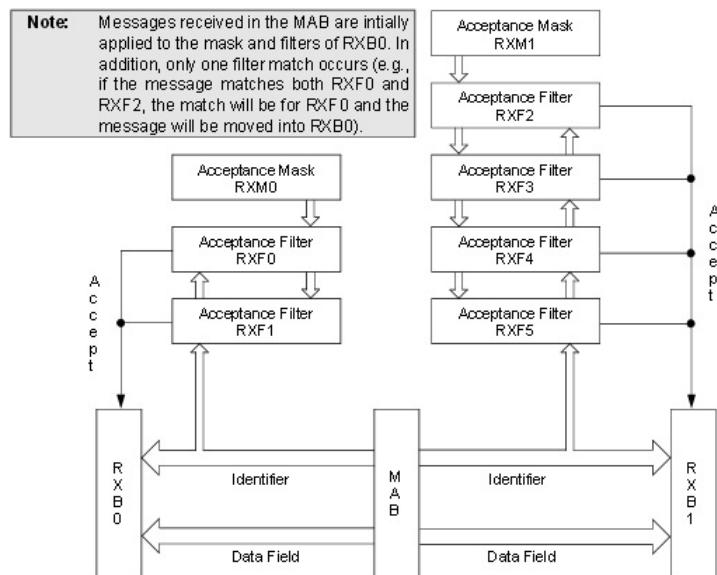


FIGURE 6.3 – La composition des différents filtres sur les buffers de réception du MCP2515.

Le MCP2515 dispose de deux buffers de réception, chacun disposant d'un masque ainsi que d'un certain nombre de filtres, deux pour le buffer RX0 et quatre pour le buffer RX1 (Fig. 6.3). Ainsi, afin de ne pas encombrer inutilement les buffers de réception par des messages dont le noeud n'aurait aucune utilité, le microcontrôleur MCP2515 réalise le traitement en interne afin de ne sélectionner que les messages qui satisfont les critères

des filtres mis en place. Comme le nombre de filtres est au maximum de 6, nous pouvons très vite être limité si un nœud doit recevoir plus de messages.

Afin de prendre en compte ce type de situation, nous avons également implémenté une méthode permettant de spécifier directement une plage d'identifiants d'une longueur définie. Avant de montrer un exemple d'implémentation, un bref rappel issu de la documentation du MCP2515 sur l'utilisation des filtres est visible figure 6.4.

Mask Bit n	Filter Bit n	Message Identifier bit	Accept or Reject bit n
0	X	X	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Note: X = don't care

FIGURE 6.4 – Utilisation des filtres et masques sur les buffers de réception du MCP2515.

Exemples

Pour rappel, les identifiants standards sont codés sur 11 bits.

Exemple 1 Filtrage sur l'identifiant 80 : On définit le filtre suivant la valeur de l'identifiant que l'on souhaite obtenir, ici 80.

ID 80	0 0 0 0 1 0 1 0 0 0 0	ID 150	0 0 0 1 0 0 1 0 1 1 0
Masque	1 1 1 1 1 1 1 1 1 1 1	Masque	1 1 1 1 1 1 1 1 1 1 1
AND	0 0 0 0 1 0 1 0 0 0 0	AND	0 0 0 1 0 0 1 0 1 1 0
Filtre	0 0 0 0 1 0 1 0 0 0 0	Filtre	0 0 0 0 1 0 1 0 0 0 0
Match ?	Accepté	Match ?	Rejeté

FIGURE 6.5 – Filtrage sur un identifiant spécifique.

Si l'on reprend la figure 6.3, on constate que, pour l'identifiant 80 qui satisfait le critère du filtre, le message sera placé dans un des buffers de réception du MCP2515, alors que celui comportant l'identifiant 150 sera rejeté. Le microcontrôleur sera donc allégé de la charge de calcul nécessaire au tri des messages.

Nous avons cependant vu qu'il pouvait être restrictif de se limiter à un filtrage sur un nombre seul, et qu'il pourrait être plus avantageux d'accorder des plages d'identifiants pour augmenter le nombre de nœuds possibles et donc la taille de l'architecture.

Exemple 2 Filtrage sur la plage 48-55 : On définit le filtre suivant la plus petite valeur de notre plage que l'on souhaite obtenir, ici 48. Le fait de placer les 3 bits de poids faible du masque à 0 permet de considérer une plage de 7 bits ($2^2 + 2^1 + 2^0$)

ID 49	0 0 0 0 0 1 1 0 0 0 1		ID 56	0 0 0 0 0 1 1 1 0 0 0	
Masque	1 1 1 1 1 1 1 1 0 0 0		Masque	1 1 1 1 1 1 1 1 0 0 0	
AND	0 0 0 0 0 1 1 0 0 0 0		AND	0 0 0 0 0 1 1 1 0 0 0	
Filtre	0 0 0 0 0 1 1 0 0 0 0		Filtre	0 0 0 0 0 1 1 0 0 0 0	
Match ?	Accepté		Match ?	Rejeté	

FIGURE 6.6 – Filtrage sur une plage d’identifiants.

La définition des filtres présentée ici n'a qu'une valeur démonstrative, et peut être modifiée selon les besoins, mais permet de proposer une méthode pour alléger la charge processeur de nos cartes à l'aide un traitement interne au contrôleur CAN.

Ainsi se conclut la partie relative à notre protocole SimpleCAN, relativement simple mais qui propose quelques fonctionnalités utiles à notre architecture. Cela n'a pas été mentionné, mais une fonction d'envoi de message texte à travers le bus a également été implémentée pour répondre aux objectifs du projet et permet le déverminage de l'application. Une fonction permettant de découper un message de plus de 8 octets en plusieurs trames a également été réalisée, cependant elle est très peu utilisée pour ne pas saturer le bus inutilement. Par ailleurs, plus son identifiant sera faible, plus le « recollement » du message sera long. De plus, les données transitant sur le bus sont bâties suivant un format spécifique afin d'obtenir des variables réseaux dont le cadre est connu à l'avance par tous les nœuds (identifiant, longueur, offset si nécessaire, ...). Ce format sera enregistré directement dans la mémoire flash afin de ne pas saturer la RAM. Pour ce qui est des données capteurs, elles seront conservées localement par chaque carte si celle-ci y est abonnée (à l'aide des filtres). D'autres fonctionnalités ont été abordées, comme une meilleure gestion de l'acquittement des messages par exemple, mais n'ont pas été réalisées car elles ne correspondaient pas à une priorité immédiate pour notre architecture.

L'ensemble du protocole ainsi que les librairies associées seront prochainement mis en ligne sur le GitHub de Splashelec [29] afin de profiter des compétences de la communauté Arduino pour améliorer l'existant et apporter de nouvelles fonctionnalités.

WRSC 2013

Du 2 au 6 septembre 2013 aura lieu la coupe du monde de voilier robot (WRSC), organisée cette année par l'ENSTA Bretagne, conjointement avec l'école Navale et l'IFREMER (Institut Français de Recherche pour l'Exploration de la Mer). Profitant de cette opportunité nous avons décidé d'exposer notre architecture par le biais d'un démonstrateur, mais nous avons également prévu de participer à certaines épreuves afin de placer le système dans des conditions proches de ce pour quoi il a été construit. Nous serons également présent à la conférence qui a lieu parallèlement à cet évènement (IRSC) afin d'y présenter notre projet devant un public. Enfin, je participerai aussi à l'organisation de l'évènement dans ses aspects logistiques, mise en place des épreuves, etc... J'ai également effectué des « reviews » d'articles soumis à examen pour la conférence IRSC.

7.1 Objectifs de l'évènement

La WRSC, pour World Robotic Sailing Championship, est une compétition de voilier autonome qui vise à promouvoir le développement de la robotique marine et plus particulièrement des voiliers robots autonomes. C'est une compétition qui puise son inspiration du défi Microtransat, une course transatlantique à la voile pour les robots autonomes. La promotion des robots voiliers autonomes propulsés par la seule force du vent se fait à travers une série de courses de courtes distances, des épreuves de navigation et des défis visant à tester leur autonomie.

Cet évènement est également accompagné de l'IRSC, pour International Robotic Sailing Conference, qui offre aux chercheurs, travaillant sur les problèmes liés à la navigation autonome et la robotique marine en générale, la chance d'échanger des idées lors de ce colloque scientifique.

La première édition a été organisée par l'INNOC, l'Association autrichienne pour l'innovation en informatique, et a eu lieu en 2008 à Breitenbrunn/Neusiedlersee, en Au-

triche. Depuis, elle a lieu chaque année dans un pays différent :

- En 2009 elle a eu lieu dans la ville de Matosinhos, au nord du Portugal et organisée par la faculté d'ingénierie de l'université de Porto en coopération avec le « Clube Naval de Leca » (un club de voile local).
- En 2010 à Kingston, Ontario, Canada.
- En 2011 à Lübeck en Allemagne
- En 2012 au Cardiff Bay Yacht Club de Cardiff, Pays de Galles. Le concours était organisé par l'Université d'Aberystwyth avec l'aide de l'Université de Cardiff.
- En 2013 à Brest, France. Cette édition est organisée par l'ENSTA Bretagne en coopération avec l'Ecole Navale et l'IFREMER.

7.2 Elaboration d'un démonstrateur

Une fois notre architecture élaborée et le protocole finalisé, l'objectif final du projet serait de tester celle-ci à l'aide d'un démonstrateur afin de montrer la flexibilité d'une telle architecture. Le choix du bateau s'est porté sur un MiniJI (Fig. 7.1) basé sur une réplique à l'échelle 1/7 des 12 Metre JI de la Coupe de l'America. et mis à disposition par l'association «Handivoile Brest ». Il s'agit d'un bateau pour une seule personne, habituellement dirigé par des pédales ou avec un volant de direction. Dans une position confortable, il offre de belles sensations pour le pilote logé dans un siège baquet. Peu coûteux et pourtant très technique, ce type de bateau a été adopté par de nombreux skippers handicapés. Il est également utilisé comme base pour le robot voilier VAIMOS, projet de l'IFREMER et de l'ENSTA Bretagne [17, 14, 20].



FIGURE 7.1 – Le système sera implémenté sur un voilier miniJI pour servir de démonstrateur durant la WRSC 2013.

Pour se diriger, notre voilier robot sera muni de différents capteurs pour l'aider dans sa route : un compas, un GPS et un capteur de vent. S'il existe des méthodes fiables pour

se passer des informations de l'anémomètre [15], nous avons fait le choix de nous en servir pour faciliter l'utilisation. Pour ce qui est des méthodes permettant de réguler un bateau à voile, elles sont nombreuses, mais certaines demandent une connaissance précise des équations d'état du voilier, ce qui en soit n'est pas très réaliste pour un comportement dans des situations pas toujours connues à l'avance (poids des passagers, courant, hauteur des vagues, etc...). L'algorithme de suivi de ligne que nous allons employer sur notre démonstrateur est celui développé par Luc Jaulin [16]. Il s'agit d'une méthode robuste qui cherche à imiter le comportement d'un navigateur. A la fois facile à implémenter, elle possède des paramètres facilement réglables possédant tous une signification physique.

La ligne suivie et sa marge de tolérance forment ce qu'on appellera une route. Dans cet algorithme, le bateau devra rester au plus près de sa ligne tout en restant sur sa route afin de nous permettre de gérer la situation en évitant d'éventuelles collisions (autres voiliers, rochers...). Cette stratégie de suivi de route peut ainsi s'apparenter à ce que l'on retrouve pour la circulation routière. Cet algorithme est actuellement déjà implémenté dans le robot voilier VAIMOS de l'IFREMER et a démontré sa fiabilité à de nombreuses reprises.

Les différents paramètres sont décrits ici :

- Données capteurs : Le cap θ du robot est mesuré à l'aide d'un compas, l'angle du vent ψ est lui acquis à l'aide de l'anémomètre, et la position \mathbf{m} est donnée par le GPS
- Actionneurs : Les entrées du robot sont l'angle de barre δ_r et l'angle maximal d'écoute δ_s^{max} .
- Paramètres : δ_r^{max} est l'angle maximal que peut prendre le gouvernail, r est la largeur de la route suivie, γ_∞ est l'angle d'incidence du vent, ζ est l'angle au près
- Références : Deux points \mathbf{a} , \mathbf{b} définissent la ligne à suivre.
- Variable d'État : La variable discrète $q \in [-1, 1]$ correspond au bord privilégié.

Algorithme

$In : m, \theta, \psi, a, b; Out : \delta_r, \delta_{rmax}; InOut : q$
$e = \det\left(\frac{b-a}{\ b-a\ }, m-a\right)$
if $ e > \frac{r}{2}$ then $q = sign(e)$
$\phi = atan2(b-a)$
$\theta^* = \phi - \frac{2 \cdot \gamma_\infty}{\pi} \cdot atan\left(\frac{e}{r}\right)$
if $\cos(\psi - \theta^*) + \cos(\zeta) < 0$
or ($ e < r$ and $(\cos(\psi - \phi) + \cos(\zeta) < 0)$)
then $\bar{\theta} = \pi + \psi - q \cdot \zeta$
else $\bar{\theta} = \theta^*$
end
if $\cos(\theta - \bar{\theta}) \geq 0$ then $\delta_r = \delta_r^{max} \cdot \sin(\theta - \bar{\theta})$
else $\delta_r = \delta_r^{max} \cdot sign(\sin(\theta - \bar{\theta}))$
$\delta_s^{max} = \frac{\pi}{2} \cdot \left(\frac{\cos(\psi - \bar{\theta}) + 1}{2}\right)$

Le fonctionnement de cet algorithme n'est pas détaillé dans ce rapport, celui-ci étant clairement énoncé dans l'article « A simple controller for line following of sailboats » [16] de Luc Jaulin.

Cet algorithme a pu être testé à l'aide d'un simulateur sous Qt Creator afin de vérifier son fonctionnement (Fig. 7.2).

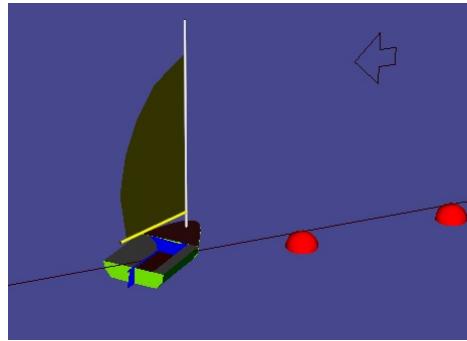


FIGURE 7.2 – L'algorithme a été testé sous un simulateur réalisé grâce à QTCreator, représentant le voilier en 3 dimensions, permettant de choisir la direction du vent et de visualiser le comportement du bateau.

A l'heure actuelle, il est difficile d'évoquer plus en détail les différentes réalisations effectuées sur le voilier miniJI car elles n'ont pas encore été abordées. Si matériellement, nous disposons de plusieurs CANinterfacers ainsi que des cartes de puissances pour les actionneurs, il reste à programmer ces différents noeuds pour obtenir le comportement désiré, à réaliser des tests en laboratoires avant de procéder à des essais en mer pour valider notre architecture. Ces étapes seront réalisées au cours du mois d'août et les résultats seront présentés lors de la soutenance orale .

Conférence

En addition avec la présentation d'un démonstrateur, nous avons également rédigé un article scientifique afin de présenter notre architecture au cours de cette conférence. Intitulé « An Arduino compatible CAN Bus architecture for sailing applications », il présente brièvement nos travaux réalisés et en cours de réalisation. L'ensemble de cet article peut être trouvé en annexe de ce rapport.

Chapitre 8

Conclusion

Dans ce rapport, nous avons décrit un nouveau protocole de communication à la fois modulaire, robuste, temps réel, et adapté à des cartes électroniques de faibles puissance de traitement (8 bits) basé sur le bus de terrain CAN. Il nous permet ainsi de relier capteurs et actionneurs au sein d'un réseau décentralisé de noeuds, communiquant de manière synchrone grâce à un organe leader, élu à l'aide d'un système d'élection adaptatif, et capable de palier à la défaillance d'un ou plusieurs noeuds sans réelle incidence sur le fonctionnement général. Si notre domaine d'application s'est limité dans le cadre de ce projet à une application d'aide à la navigation handivoile, la portabilité d'une telle architecture à d'autres systèmes est bien entendue possible et même fortement encouragée. Le fait d'avoir développé ce système en open-source a pour but de proposer une solution fiable, comportant une base solide permettant de développer des fonctionnalités adaptées aux systèmes ciblés tout en bénéficiant de la robustesse du CAN et de l'appui de la communauté Arduino pour la continuité de son développement.

En ajoutant la possibilité de reconfigurer les noeuds directement à l'aide de messages CAN grâce à un bootloader spécifique, nous avons supprimé la dépendance du lien série et donc diminué les câblages nécessaires. Cependant, nous aurions tort de nous priver totalement de la gestion USB présente nativement dans les bootloaders d'origines et une amélioration possible serait de faire cohabiter les deux dans un espace mémoire relativement restreint. N'étant pas une priorité absolue pour notre système à l'heure actuelle, cet ajout sera réalisé plus tard.

En ce qui concerne la gestion des capteurs par le système, le comportement est pour le moment réalisé à la main et au cas par cas. Ainsi l'architecture est encore, pour le moment, extrêmement tributaire du choix de ces derniers et lorsqu'un utilisateur souhaitera utiliser les librairies que nous avons développé, il aura la charge de définir le comportement de ses noeuds en fonction des capteurs qu'il aura à sa disposition, suivant les types de communications qu'ils utilisent (I2C, NMEA2000, signal analogique,...). Pour la suite, il faudra s'inspirer du système de profils d'applications proposé par CANopen qui permet de définir des comportements génériques préétablis en fonction des types

d'entrées/sorties dont on dispose dans le but d'améliorer son accessibilité à tous types de publics.

S'il est encore trop tôt pour pouvoir parler des résultats du démonstrateur qui est encore dans une phase de développement, d'autres idées d'adaptations sont venues à l'esprit de Bernt Weber qui envisage une implémentation de ce protocole sur les fauteuils roulants électriques. Toujours dans la problématique du handicap, nous avons constaté au cours de nos recherches que la demande pour un système ouvert et modifiable des protocoles CAN des fauteuils roulants était présente, tout en conservant la sécurité et fiabilité des systèmes actuels, dont le comportement reste figé par la définition d'usine. L'adaptation de notre système à ce type d'architecture permettrait aux utilisateurs et développeurs de pouvoir modifier le comportement de la chaise en fonction du type de handicap, ce qui n'est pas toujours le cas avec les systèmes commerciaux.

De plus, l'ENSTA Bretagne qui participe chaque année au trophée SIA (Société des Ingénieurs de l'Automobile) trouverait également un intérêt indéniable dans ce protocole CAN, initialement conçu pour l'industrie automobile rappelons le, qui résoudrait la problématique de branchement des capteurs à l'aide d'un bus unique.

De toute évidence, ce projet restera un moment fort dans ma formation d'ingénieur à l'ENSTA Bretagne, autant par sa dimension humaine que par son cadre grâce à la collaboration avec Splashelec et notre participation à la conférence IRSC 2013. Il m'a permis d'étendre mes compétences acquises au cours de ma formation à l'ENSTA Bretagne dans le domaine de la robotique avec l'élaboration d'un démonstrateur robotisé pour la WRSC 2013, mais également d'en apprendre et d'en perfectionner de nouvelles, comme la programmation en Arduino, la manipulation de protocoles réseaux, la réalisation mécanique du robot voilier, etc... La rédaction d'un article pour la conférence IRSC 2013 m'a permis d'améliorer mes capacités de synthèse et de projection de ce projet à long terme, et le fait d'avoir participé au jury des « reviewers » m'a permis de confronter ma méthode à celles d'autres candidats et d'en découvrir un peu plus sur le monde de la voile robotisée. Pour cela, je tenais à remercier Bernt Weber pour m'avoir donné l'occasion de participer à l'amélioration de son système, mon encadrant Benoit Clement pour son soutien de tous les instants et sa grande disponibilité, ainsi que les différents professeurs du labo STIC, Olivier Reynet, Luc Jaulin, Fabrice Le Bars, pour leur aide apportée à ce projet.

Bibliographie

- [1] Steve Alvey. *Martin 16 Power-Assist System - Mk IV, Operator Manual* http://www.martin16.com/uploads/auto_mkiv.pdf, 2005.
- [2] Steve Alvey. *Martin 16 Power-Assist System - Mk IV, Brochure* <http://www.martin16.com/uploads/autobrochure.pdf>, Cited 23 June 2013.
- [3] National Marine Electronics Association. *NMEA2000(TM) standard*, In : National Marine Electronics Association website http://www.nmea.org/content/nmea_standards/nmea_2000_ed3_00.asp, 2012.
- [4] Akin Avci. *The Universal Robot Bus : A Local Communication Infrastructure For Small Robots*. PhD thesis, Bilkent University, 2008.
- [5] David Braben. *Official website of Raspberry Pi* <http://www.raspberrypi.org/>, Cited 15 July 2013.
- [6] CanFestival. *CanFestival, a CANopen framework*. <http://www.canfestival.org/>, Cited 13 May 2013.
- [7] CiA. *CAN specifications*, In : CAN in Automation <http://www.can-cia.org/index.php?id=can>, Cited 13 May 2013.
- [8] Eclipse. *Eclipse (software)*. <http://www.eclipse.org/>, 2013.
- [9] Nke Marine Electronics. *nke marine electronics* <http://www.nke-marine-electronics.com/home.html>, Cited 23 June 2013.
- [10] Furuno. *Furuno Navnet* <http://www.navnet.com/>, 2012.
- [11] Fabian Greif. *CAN Bootloader*, In : Universal CAN library. Roboterclub Aachen e.V. <http://www.kreatives-chaos.com/artikel/can-bootloader>, 2010.
- [12] hilarylister.com. *Sip & Puff* http://www.hilarylister.com/d5483/hilary_s_boat/sip_amp_puff.aspx, 2012.
- [13] Intel. *Intel Hex format*, <http://www.datelec.fr/uniprof/format%20intel.htm>, Cited 15 July 2013.
- [14] L. Jaulin. Modélisation et commande d'un bateau à voile. In *Proceedings of 3rd Conference Internationale Francophone d'Automatique*, Douz, Tunisie, 2004.
- [15] L. Jaulin. Représentation d'état pour la modélisation et la commande des systèmes. In *Coll. Automatique de base*, Hermès, London, 2005.
- [16] L. Jaulin and F. Le Bars. A simple controller for line following of sailboats. In *Proceedings of the 5th International Robotic Sailing Conference*, Cardiff, England, 2012.
- [17] L. Jaulin, B. Clement, Y. Gallou, F. Le Bars, O. Menage, O. Reynet, and J. Sliwka. Suivi de route pour un robot voilier. In *Proceedings of 7th Conference Internationale Francophone d'Automatique*, Grenoble, France, 2012.

- [18] Access Dinghy Sailing Systems Pty Ltd. *Access Dinghies OPERATIONS & SAFETY MANUAL - LIBERTY*
http://www.sailingforall.com/imgs/74liberty_operation_&_safety_manual.pdf.pdf, Cited 25 June 2013.
- [19] Martin16. *M16 sailboat for able and disabled sailors*
<http://www.martin16.com/>, Cited 23 June 2013.
- [20] O. Menage, F. Gaillard, T. Gorgues, T. Terre, P. Rousseaux, S. Prigent, Y. Auffret, L. Dussud, B. Forest, M. Repecaud, L. Jaulin, B. Clement, Y. Gallou, and F. Le Bars. VAIMOS : Voilier autonome instrumente pour mesures oceanographiques de surface. In *Symposium on Vulnerability of coastal ecosystems to global change and extreme events*, Biarritz, France, 2011.
- [21] Matthias Merten and Horst-Michael Gross. Highly adaptable hardware architecture for scientific and industrial mobile robots. In *RAM*, pages 1130–1135. IEEE, 2008.
- [22] Raymarine. *Ethernet*, In : Raymarine website
<http://www.raymarine.eu/view/?id=5537>.
- [23] Raymarine. *SeaTalk/SeaTalk1*
<http://www.raymarine.eu/view/?id=5535&collectionid=26&col=5557>, Cited 23 June 2013.
- [24] Raymarine. *Autopilot Joystick*, In : Raymarine website
<http://www.raymarine.com/view/?id=2705>, Cited 24 June 2013.
- [25] Aldebaran Robotics. *NAO Software 1.14.3 documentation : low level architecture* http://www.aldebaran-robotics.com/documentation/naoqi/sensors/dcm/low_level_architecture.html, Cited 23 June 2013.
- [26] Hansa Sailing. *Hansa Sailing* <http://hansasailing.com/>, Cited 23 June 2013.
- [27] J. Sliwka, J. Nicola, R. Coquelin, F. Becket De Megille, B. Clement, and L. Jaulin. Sailing without wind sensor and other hardware and software innovations. In *Proceedings of the 4th International Robotic Sailing Conference (Springer Eds.)*, Lübeck, Germany, 2011.
- [28] Roland Stelzer and Karim Jafarmadar. The robotic sailing boat asv roboat as a maritime research platform. In *Proceedings of 22nd International HISWA Symposium*, 2012.
- [29] B. Weber. *Splashelec autopilot documentation*, In : The Splashelec Wiki
<http://wiki.splashelec.com/index.php/autopilot>, Cited 13 May 2013.
- [30] B. Weber. *The CAN Interfacer*, In : The Splashelec Wiki.
<http://wiki.splashelec.com/index.php/caninterfacer>, Cited 13 May 2013.
- [31] wetwheels.co.uk. *About the boat* <http://www.wetwheels.co.uk/about-us/about-the-boat/>, Cited 24 June 2013.
- [32] Wikipedia. *BSD Licenses*, In : English Wikipedia
http://en.wikipedia.org/wiki/bsd_licenses, Cited 15 July 2013.
- [33] Wikipedia. *NMEA 2000*, In : English Wikipedia
http://en.wikipedia.org/wiki/nmea_2000, Cited 23 June 2013.

Annexe **A**

Annexes

A.1 Article IRSC 2013

An Arduino compatible CAN Bus architecture for sailing applications

Kévin Bruget, Benoît Clement, Olivier Reynet and Bernt Weber

Abstract This paper describes a Controller Area Network (CAN) Bus architecture based on Arduino compatible boards, to be used as an alternative communication system for robotic applications. This combines both, the robustness of CAN and the accessibility of Arduino software. The architecture is developed here to improve a Navigational Assistance System, which was initially created for disabled people. The system is composed of Arduino compatible boards, wired with various sensors and actuators, and communicating with an Human Machine Interface (HMI), directly accessible via a mobile phone or a tablet running on the open-source operating system Android. Information is transferred through the CAN bus architecture between multiple nodes (i.e. Arduino compatible boards) and the implementation of a CAN bootloader allows the reconfiguration of the nodes directly through the bus. The aim is to create a generic system able to work in various kinds of situations, adaptable to all kinds of users, including persons with all sorts of disabilities. This work will result in a demonstrator on a Miniji for the WRSC 2013 and an entirely joystick controlled boat for single handed sailing.

Kévin Bruget
ENSTA Bretagne, 2, Rue Francois Verny, 29200 Brest, France e-mail: kevin.bruget@ensta-bretagne.fr

Benoît Clement
ENSTA Bretagne Lab-STICC UMR CNRS 6285, 2, Rue Francois Verny, 29200 Brest, France e-mail: benoit.clement@ensta-bretagne.fr

Olivier Reynet
ENSTA Bretagne Lab-STICC UMR CNRS 6285, 2, Rue Francois Verny, 29200 Brest, France e-mail: oliver.reynet@ensta-bretagne.fr

Bernt Weber
Splashelec, 18, rue de Pont Louët, 29200 Brest, France e-mail: bernt.weber@splashelec.com

1 Introduction

Robotics in the sailing field is now a reality. Since 2008 with the first World Robotic Sailing Championship (WRSC) and International Robotic Sailing Conference (IRSC) [9, 10, 11, 31, 5], studies have shown that sailing robots can overcome the embryonic stage [18, 19, 23, 32]. Able to replace humans during competition or to realise autonomous measurements, its field of action becomes larger than before. Our approach is not to totally remove human action, but to assist it during information acquisition, decision process and execution (steering and trimming). Sailing globally has remained a field inaccessible for disabled people, because of the extreme mobility the sailor needs to acquire information and to manipulate commands of a boat. Sailing requires significant efforts that the disabled cannot afford. Splash-elec aims to give disabled sailors access to those kinds of activities. The assistance system was initially composed of an electronic board and a joystick which allow a person to steer a boat manually as helmsman, forming part of a crew. When in need to free his hands, the helmsman can activate compass guided PID steering (autopilot).

Working with ENSTA Bretagne, an Android based Human Machine Interface (HMI) has been developed, in order to complete the system with visual navigational aid for disabled people [4]. To provide a more complete view of its environment for the skipper and to counterbalance his lack of mobility, some sensors, like wind sensor, compass and GPS were added to the system. Linked to an Arduino board, information is sent through Bluetooth to a tablet in charge of the information display.

At that point interfacing and cabling had already become complex and interfaces to connect extensions became a rare resource. Users then asked for joystick steering on smaller single handed boats, with need for supplementary actuators for the sails and more sensors to control the extra actuators. It was time to think about a new system architecture, with less connections for diminished cost and better reliability.

The present part of the project is the follow-up, which aims to develop a Controller Area Network (CAN [8]) bus interface board and the necessary software to allow communication using a more modular and flexible bus system, that is easily adaptable to all sorts of situations and disabilities. That is to say, developers can plug their own sensors to the system and show data directly through the tablet. On the software side, an Arduino bootloader compatible with CAN allows the programming of nodes directly through the bus. There is also a support for debugging messages relayed by the bus.

The main objective of this work is to provide to sailors, disabled or not, a navigational assistance system able to perform automated tasks (heeling limitation, autopilot, ...). It integrates the needed adaptability into the development process and aims to offer a solution for various kinds of disabilities. For example steering with a joystick compensates the lack of strength in the arm, the HMI centralises sensor information to compensate the lack of mobility and sensitivity (wind, boat speed,...). Furthermore, due to its open-source nature, the software system is entirely and easily modifiable and developers can add functionality or change the HMI according to users specific needs. Finally the system should be adaptable to every kind of boat.

This paper describes a complete system and its features, its architecture and the communication process between nodes. The demonstrator, an adapted Miniji boat will be presented at the WRSC 2013.

2 Related work

2.1 Low-level architecture of existing robot systems

Due to the complexity of tasks mobile robot are designed for, robots use generally embedded computers with substantial processing power and are often centered around one of them. Connecting to the robot's hardware is done using all sorts of interfaces available to the computer: USB, Ethernet, serial (RS232, RS435, RS485), and using hubs and port concentrators (e.g. NASA's [25] or Roboat [33]) where necessary. Employing microcontrollers in place of some port concentrators gives access to all the lower level electronic interfaces as I2C, SPI (NAO [29]).

In its simplest configuration, our system consists only of a joystick and a rudder actuator. In this configuration, there is no need for any embedded computer, the system implements only low-level reactive behaviours (compass controlled or joystick steering). A very simple microcontroller can execute these tasks, at low cost and with low current consumption.

As for the embedded computer centered architectures, everything must be wired to the one central microcontroller, but resources and available interfaces are more sparse on the lower abstraction level and the number of connected sensors and actuators is very limited. Centralized architectures become impractical when more components have to be connected to one simple microcontroller.

Bus systems offer a flexible and modular solution to interconnect microcontrollers, each having sensors and/or actuators attached. Historically used RS-485 multi-point serial cabling is more and more replaced with higher level CAN, with help of hardware implementations of the CAN protocol itself now integrated to microcontrollers. As for automotive and industrial field buses, CAN is today used in robots and results there in highly adaptable hardware architectures (Merten and Gross [24]).

Merten and Gross' robot architecture uses the CANopen [7] protocol. Open-source implementations of this protocol exist (CANFestival [6], CANopen SlaveLib [38] and CANopenNode [17]).

So we retained the CAN bus, but decided to go with an architecture able to work with very simple nodes and a network that is able to run without an embedded computer. Being able to run without embedded computer means being able to power-down the computer regularly for energy savings, or running entirely without it in simple configurations. And as we understand it, the three cited CANopen implementations do not allow to work without an embedded computer or high performance microcontroller. The missing piece seems to be a suitable simpler protocol, we are

not aware of a high-level (on-top of CAN) open-source protocol, simple enough for 8-bit microcontrollers, providing services necessary for sensors and actuators distributed over the bus, as for example synchronised sensor reading and distributed control loops.

2.2 *Commercial marine electronics*

2.2.1 **Communication networks**

Displays and autopilots installed on today's sailing yachts use mostly multi-drop serial buses with vendor specific proprietary protocols. Examples are Raymarine's SeaTalk [27] and NKE's Topline [13]. On some recent boats, CAN based, industry standard NMEA2000 [3] buses replace the proprietary protocols. For higher bandwidth requirements such as radar or echo sounder images, these buses are sometimes completed by extra Ethernet cabling (e.g. Furuno Navnet [14], Raymarine SeaTalkHS [26]).

Note that the Ethernet approach is power-hungry, costly and difficult to adapt to simple 8-bit microcontrollers, but industry choice CAN based NMEA2000 could be a good solution. The problem is that NMEA2000, and the J1939 protocol it is based on, are proprietary protocols [37]. Once again, as with CAN based robot architectures, CAN looks promising, but the missing piece is an adapted open source protocol, that would allow to design and integrate new hardware for different situations of handicap.

2.2.2 **Autopilots**

Commercial available autopilots are very close to joystick assisted steering for disabled people:

- Commercial autopilot actuators are used for steering by disabled. Splashelec uses for most boats actuators sold with commercial autopilots. A difference exists though in dimensioning for small vessels up to about 10 m length: a typical tiller autopilot uses an electrical motor with a power rating of about 10 to 20 W, which allows only for slow rudder movements compared to what a human helmsman can do. To give a disabled person using a joystick the same performance, the actuator has to be oversized compared to a commercial autopilot.
- Power electronics of commercial course computers as well Splashelec's model are designed around a H-bridge, allowing speed modulation for smooth rudder movements. There is no technical difference.
- Firmware of commercial autopilots is very specialized to course keeping following compass, wind or GPS data. All the autopilots known to us use PID closed loop control, calling the PID coefficients "rudder", "counter-rudder" and "auto-trim" in the manuals.

Some commercial autopilots allow joysticks to control the rudder: pushing the joystick moves the rudder, when the joystick returns to the center, the rudder stays in position. A subset of these systems allows also for a proportional mode: the rudder follows the joystick (i.e. [28]). Such a system offers assistance to a disabled helmsman and some are used in this way [36].

Modifying firmware, which is not possible with commercially available autopilots, would allow for further adaptation to individual disabilities:

- joystick damping: many disabilities produce jerky hand movements. Simple position-averaging allows fine control of a sailing yacht.
- automatic adaptive correction of the rudder position corresponding to the joystick center: under sail, a boat is generally not completely balanced (i.e. weather helm), the rudder must keep an angle to steer straight. A simple proportional joystick must then stay continuously in an off-center position, against the spring, which is uncomfortable, and becomes impossible if the handicap implies reduced dexterity.

2.3 Existing assistance systems designed for disabled sailors

The company "Hansa Sailing" [30] is a manufacturer selling a range of boats and assorted electric control systems destined for disabled sailors. Wiring applies star topology around a control box (Access Liberty Manual: [20]). Winch and rudder actuators have 2 wire connectors for DC-motors. The input devices need a 9 wire connection, typically a joystick with associated push buttons for mode selection. No sensors are associated to the DC-motors.

Steve Alvey's company [22] sells electric controls for the Martin 16 and other sailing boat types used by disabled people [2]. He also designs specialized systems on purpose. An example is the boat steered by Hilary Lister around Britain; She uses a three straw sip-and-puff interface [16], and has access to a Raymarine autopilot.

The manual of the Martin 16 system [1] shows a Raymarine 3 pin socket labelled "Sea Talk" for command interface connections (joystick, sip and puff, etc.), the bus system simplifying cabling. A second socket labelled "Helm Drive" mates the standard connector of the used Raymarine ST4000/SPX-5 electric actuator (2 used pins for direct DC-motor connection). No additional sensors are added to the actuator.

Experience using our rudder only steering system showed us that using a position sensor enables tactile feedback, by establishing a relation between rudder and joystick position. And, in this setup, a bigger rudder angle makes water push harder on the rudder, as does the center return spring in the joystick. The helmsman feels the rudder, its position and the water force.

Our conclusion is that connection count of existing sailing assistance systems for disabled has reached a limit, where it becomes impractical and expensive, in the environment where every connector has to be waterproof and salt water resistant. Joystick box connections have up to 9 wires, adding sensors to individual

actuators would complicate the cabling even more. Bus systems have already been employed to user-interfaces, we think that a bus system should be extended to the entire system, including the actuators and the associated sensors. This would simplify installations and allow more advanced interactions, better user experience and make the systems more modular, flexible and adaptable.

3 System description

The system is actually composed of two major parts, the Programmable Servo Controller (PSC) and the HMI, which displays sensor information and allows the skipper to activate the joystick or to use the autopilot. The communication between the bus and the HMI tablet uses Bluetooth, whereas internal communication is made entirely using the CAN protocol. Its behaviour can be assimilated to a closed-loop system (Fig. 1). Information which comes from the environment as any human input has influence on the actuators, that is to say the system will be able to perform automated tasks. An example is a standard course keeping autopilot, or joystick steering mode, with the autopilot taking over when necessary to limit heeling.

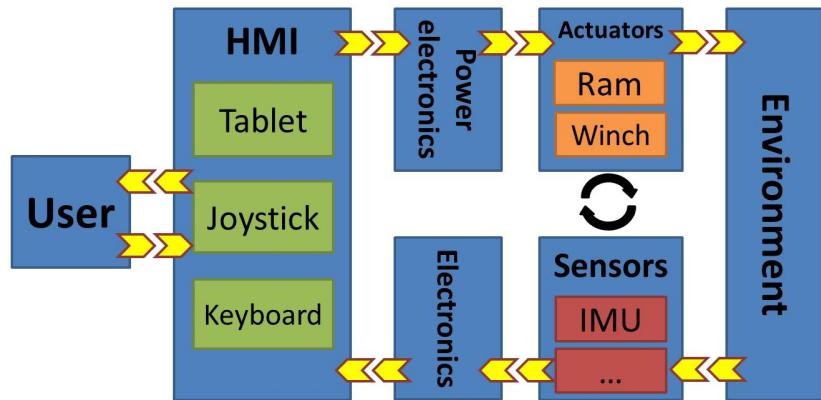


Fig. 1 The system is based on a closed-loop where the user can be removed to implement autonomous behaviour.

The programmable servo controller PSC (Fig. 2) is a key component of the system and can do the work of an autopilot course computer, which can steer to wind or compass when associated to the corresponding sensors. The PSC board (6 x 7.5 inches) contains a microcontroller, power electronics for an electric motor (ram or winch), an electric clutch and the power supply including filter circuits (see Fig. 3). Various interfaces for rudder angle sensor, joystick and control keyboard already exist and, in future versions, a CAN bus interface will be integrated directly to the

main circuit board. To add electric winches, we use one instance of the PSC for each.



Fig. 2 PSC of the Splashelec system in a waterproof case. It contains several inputs for the keyboard, the compass, the rudder angle sensor, the battery...

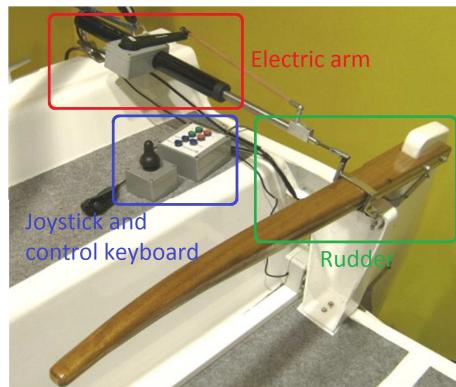


Fig. 3 The Splashelec system composed of an electric ram (in red), a joystick and a control keyboard (in blue) and the rudder (in green).

This PSC is based on open-source technology in order to allow the easy integration of new functionalities or to modify actual ones. The microcontroller is compatible to Arduino boards and can so be programmed with Arduino's Integrated Development Environment (IDE) software, which gives an access to the programming interface, existing libraries and various on-line examples. Results of this project (material and software) are publicly available on the Internet [34]. Wired to this box, multiple sensors such as an Inertial Measurement Unit (IMU), a wind sensor or loch-speedo are linked into a CAN architecture. When using more than one actuator, each one uses its own dedicated PSC, with instances of the power electronics,

local sensors, and a CAN bus interface. All the system, including mechanical parts, is transportable and can adapt to different boats.

The HMI, programmed for Android by using the Android Software Development Kit (SDK) tool for Eclipse [12], contains different areas (Fig. 4): in addition to textual information, some data, such as wind orientation and speed, compass and rudder angle, is also displayed in graphic form to allow a better understanding of the environment. A last area acts as a virtual keyboard to switch between autopilot and joystick mode and the user can modify the course to steer. The graphical layout of the HMI is realised in XML language and could be easily modified to fit needs of specific users. This results in a system (joystick, tablet display, extra keyboard,...), compatible with different disabilities, while still being usable by sailors without any disabilities.

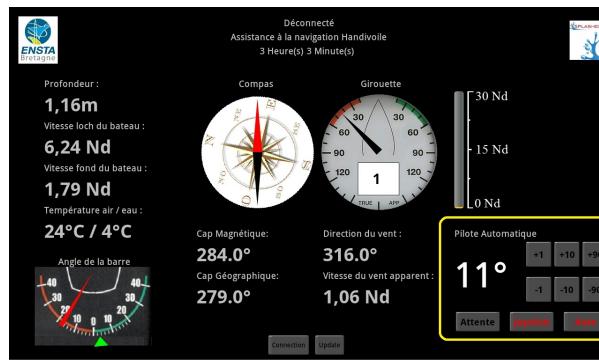


Fig. 4 The Human Machine Interface displays sensor information and contains a virtual keyboard (in yellow) to replace the physical one.

4 CAN Bus architecture

The need for a bus architecture came from the multi sensor and actuator problematic and the cabling and communication complexing with each added element. Used first in automotive applications, the CAN protocol is now widely available and starts migrating into many non-automotive applications. This open standardised high layer protocol provides a reliable message exchange system between various nodes.

The next section describes the protocol used in our system and its architecture. Finally the following section discusses the implementation of a Arduino CAN bootloader able to reconfigure nodes through the bus.

4.1 CAN protocol

The choice of the CAN protocol for our architecture arose from the need of a broadcast communication mechanism, that had to be easy to use, able to work with multiple nodes and which provides a reliable communication protocol to exchange information from sensors and actuators. Indeed CAN is able to detect errors with no less than three mechanisms: Cyclic Redundancy Check (CRC) to verify message integrity, Frame check to verify that data is sent in the correct shape and acknowledgments to guarantee reception. Besides, adding nodes to an existing CAN network can easily be done, which meets our needs for a modular architecture.

Adding new nodes to the network is straight forward: adding a new sensor with interfaces as for example NMEA 183 (National Marine Electronics Association) or I2C (Inter Integrated Circuit), implies reading the data by the connecting nodes microcontroller and to put the data in a CAN frame. Arduino allows to do this in very few lines of code, using libraries for CAN and a plethora of choice of sensor interfaces.

4.1.1 Higher-layer protocol

At this point appears the need of a higher level CAN protocol to organize data in the CAN frames. Various higher-layer CAN protocols already exist such as SAE J1939 used in NMEA 2000, CAN Kingdom, or CANopen but it appears that no one fits our need of a simple CAN protocol. Too complex or not adapted for 8-bit microcontrollers, CAN protocols are not widespread and their code is not always open-source. It has been decided to develop our own protocol based on our needs, called SimpleCAN, into an Arduino library containing the essential functions to support our architecture. The protocol is designed in a way permitting to add new features afterwards.

4.1.2 System architecture

Our architecture (Fig. 5) is based on Arduino compatibility and uses a CAN bus interface card we call the CANinterfacer. The CANinterfacer is compatible to an Arduino with a CAN shield on top. This on purpose designed board [35] uses an ATMEGA32U4 microcontroller as do the Arduino Leonardo and Micro. It is small in dimension (1.95 x 1.95 inch, slightly smaller than 5 x 5 cm) and can be programmed by USB using the Leonardo bootloader and the Arduino IDE. It can be powered from the CAN bus by an on-board switching power supply accepting from 7 to 32 Volts, most of the I/O pins are available for local connections.

In the system, a group of elements (actuators, sensors, HMI elements,...), wired to a CANinterfacer, becomes a CAN node (Fig. 6). That is to say, each CANinterfacer typically uses Arduino libraries to convert input from various sources, for example analog inputs, NMEA 183 or I2C connected sensors, into a CAN messages. Putting

a CANinterfacer between the new hardware and the bus to integrate it as standard node gives great flexibility.

In the same way, the upcoming version the PSC will contain an CANinterfacer and be a native node in our bus system. This type of node allows to integrate servo controlled actuators such as rams and winches with their associated sensors.

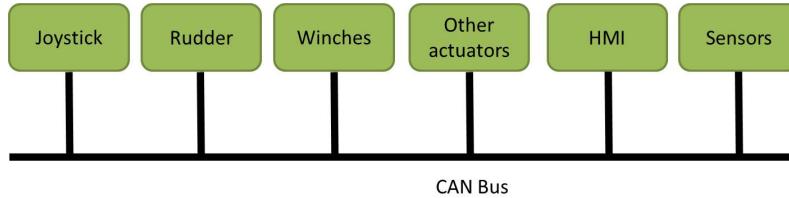


Fig. 5 CAN architecture of the system

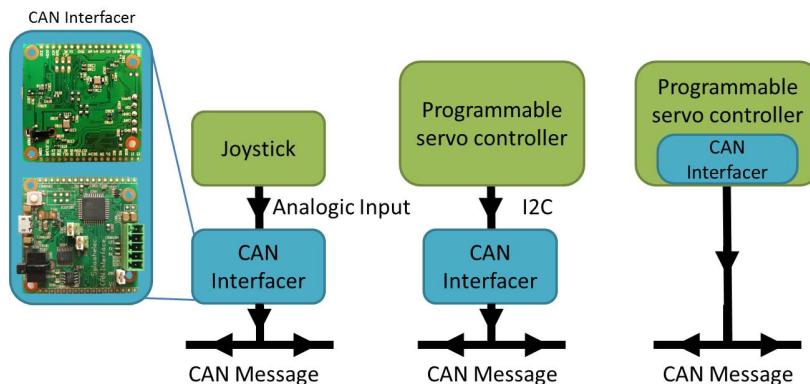


Fig. 6 Examples of a CAN node: The joystick and the Programmable Servo Controller (PSC) in its first version. The next version will have a CANinterfacer integrated into the board.

4.2 CAN Bootloader

The implementation of the CAN protocol opens the way to simplified programming of every node directly through the bus, one unique connection to the bus permits this. To obtain that, we need a new Arduino bootloader that accepts CAN programming commands for our CANinterfacer. Fabian Greif from the Robotics Club of Aachen,

Germany, has already worked on the subject with very close hardware [15] and built its own CAN bootloader, that allows updating firmware and local code from CAN messages. Small modifications have been made, in order suit connections of our CANinterfacer. [35].

The programming operation is initiated by a Python script that initiates the communication process between the PC connected programming node and the node which needs to be reconfigured. More concretely, the programming node will act as an In-System Programming (ISP) interface: it receives the new program by USB or serial port and sends it encapsulated in CAN messages to reprogram the specified node. The whole process is detailed in [35].

4.3 Demonstrator

The final objective of the current project is to build a demonstrator to show robotic functionality during WRSC 2013 [21] that proves the flexibility of such an architecture. The choice of the boat fell on a Miniji (Fig. 7) made available by the association "Handivoile Brest", which is based on a small scale replica of a historic America's Cup hull. It is a single-handed sailing boat, ordinarily steered by foot pedals or with a steering wheel. In a comfortable position, it offers vivid sensations to the sailor housed in a bucket seat. Inexpensive and very technical, the boat type was adopted by many French disabled sailors. But it is also used as sailing robot for the VAIMOS project of Institut Français de Recherche pour l'Exploration de la Mer (IFREMER) and ENSTA Bretagne [18, 19, 23].



Fig. 7 The system will be implemented on a Miniji sailboat to act as a demonstrator.

Furthermore, there is demand to increase the autonomy and safety, coming from instructors working around disabled sailing. The boat will be equipped with two electric winches and a rudder system, all interconnected by the CAN bus. By adding sensors to the bus (e.g GPS and IMU), this boat will be able to perform automated tasks, as does a sailing robot. Indeed, to increase security, we can limit the heeling

or restrict the navigational area. Instructors will also be able to take control of the boat with a remote controller for safety reasons or even to activate the autopilot if the skipper becomes unable to manipulate commands.

5 Conclusion

In conclusion we have a system based on a CAN bus architecture which allows developers to connect new sensors or actuators, and to reprogram the system easily using Arduino technology. Such a system assists the sailor during navigation and can automate complex tasks. It helps disabled by easing the access to the sailing activity, or gives any other people navigational assistance. Based on the open-source approach, electronics and software can be modified according to specific requirements, numerous opportunities for development exist. During WRSC 2013, a prototype will demonstrate robotic functionality. The final aim is to obtain products with new features derived from robotic sailing, encouraging people to develop their own system modifications.

References

1. Steve Alvey. *Martin 16 Power-Assist System - Mk IV, Operator Manual* http://www.martin16.com/uploads/auto_mkiv.pdf, 2005.
2. Steve Alvey. *Martin 16 Power-Assist System - Mk IV, Brochure* <http://www.martin16.com/uploads/autobrochure.pdf>, Cited 23 June 2013.
3. National Marine Electronics Association. *NMEA2000(TM) standard*, In: National Marine Electronics Association website http://www.nmea.org/content/nmea_standards/nmea_2000_ed3_00.asp, 2012.
4. N. Brocheton, K. Bruget, A. Wibaux, O. Reynet, B. Clement, and B. Weber. Système d'assistance à la navigation handivoile. In *Proceedings of Handicap 2012 : 7th congress on technical assistances for disabled people*, Paris, France, 2012.
5. J. Finnis C. Sauze. *Proceedings of the 5th International Robotic Sailing Conference*. Springer, 2012.
6. CanFestival. *CanFestival, a CANopen framework*. <http://www.canfestival.org/>, Cited 13 May 2013.
7. CiA. *CANopen*, In : CAN in Automation <http://www.can-cia.org/index.php?id=systemdesign-canopen>, 2013.
8. CiA. *CAN specifications*, In : CAN in Automation <http://www.can-cia.org/index.php?id=can>, Cited 13 May 2013.
9. Collective. *Proceedings of the 1st International Robotic Sailing Conference*. 2008.
10. Collective. *Proceedings of the 2nd International Robotic Sailing Conference*. 2009.
11. Collective. *Proceedings of the 3rd International Robotic Sailing Conference*. Universidade do Porto, 2010.
12. Eclipse. *Eclipse (software)*. <http://www.eclipse.org/>, 2013.
13. Nke Marine Electronics. *nke marine electronics* <http://www.nke-marine-electronics.com/home.html>, Cited 23 June 2013.
14. Furuno. *Furuno Navnet* <http://www.navnet.com/>, 2012.

15. Fabian Greif. *CAN Bootloader*, In : Universal CAN library. Roboterclub Aachen e.V. <http://www.kreatives-chaos.com/artikel/can-bootloader>, 2010.
16. hilarylister.com. *Sip & Puff* http://www.hilarylister.com/d5483/hilary_s_boat/sip_amp_puff.aspx, 2012.
17. Janez. *CANopenNode* <http://sourceforge.net/projects/canopennode/>, Cited 23 June 2013.
18. L. Jaulin. Modelisation et commande dun bateau a voile. In *Proceedings of 3rd Conference Internationale Francophone d'Automatique*, Douz,Tunisie, 2004.
19. L. Jaulin, B. Clement, Y. Gallou, F. Le Bars, O. Menage, O. Reynet, and J. Sliwka. Suivi de route pour un robot voilier. In *Proceedings of 7th Conference Internationale Francophone d'Automatique*, Grenoble, France, 2012.
20. Access Dinghy Sailing Systems Pty Ltd. *Access Dinghies OPERATIONS & SAFETY MANUAL - LIBERTY* http://www.sailingforall.com/imgs/74liberty_operation_&_safety_manual.pdf.pdf, Cited 25 June 2013.
21. F. Le Bars M. Melguen. *Official WRSC 2013 Website*. <http://www.ensta-bretagne.eu/wrsc13/>, 2013.
22. Martin16. *M16 sailboat for able and disabled sailors* <http://www.martin16.com/>, Cited 23 June 2013.
23. O. Menage, F. Gaillard, T. Gorges, T. Terre, P. Rousseaux, S. Prigent, Y. Auffret, L. Dussud, B. Forest, M. Repecaud, L. Jaulin, B. Clement, Y. Gallou, and F. Le Bars. VAIMOS: Voilier autonome instrumente pour mesures oceanographiques de surface. In *Symposium on Vulnerability of coastal ecosystems to global change and extreme events*, Biarritz, France, 2011.
24. Matthias Merten and Horst-Michael Gross. Highly adaptable hardware architecture for scientific and industrial mobile robots. In *RAM*, pages 1130–1135. IEEE, 2008.
25. Eric Park, Linda Kobayashi, and Susan Y. Lee. Extensible hardware architecture for mobile robots. In *ICRA*, pages 3084–3089. IEEE, 2005.
26. Raymarine. *Ethernet*, In: Raymarine website <http://www.raymarine.eu/view/?id=5537>.
27. Raymarine. *SeaTalk/SeaTalk1* <http://www.raymarine.eu/view/?id=5535&collectionid=26&col=5557>, Cited 23 June 2013.
28. Raymarine. *Autopilot Joystick*, In: Raymarine website <http://www.raymarine.com/view/?id=2705>, Cited 24 June 2013.
29. Aldebaran Robotics. *NAO Software 1.14.3 documentation: low level architecture*, In: website of Aldebaran Robotics http://www.aldebaran-robotics.com/documentation/naoqi/sensors/dcm/low_level_architecture.html, Cited 23 June 2013.
30. Hansa Sailing. *Hansa Sailing* <http://hansasailing.com/>, Cited 23 June 2013.
31. A. Schlaefer and O. Blaurock. *Proceedings of the 4th International Robotic Sailing Conference*. Springer, 2011.
32. J. Sliwka, J. Nicola, R. Coquelin, F. Becket De Megille, B. Clement, and L. Jaulin. Sailing without wind sensor and other hardware and software innovations. In *Proceedings of the 4th International Robotic Sailing Conference (Springer Eds.)*, Lubeck, Germany, 2011.
33. Roland Stelzer and Karim Jafarmadar. The robotic sailing boat asv roboat as a maritime research platform. In *Proceedings of 22nd International HISWA Symposium*, 2012.
34. B. Weber. *Splashelec autopilot documentation*, In: The Splashelec Wiki <http://wiki.splashelec.com/index.php/autopilot>, Cited 13 May 2013.
35. B. Weber. *The CAN Interfacer*, In: The Splashelec Wiki. <http://wiki.splashelec.com/index.php/caninterfacer>, Cited 13 May 2013.
36. wetwheels.co.uk. *About the boat* <http://www.wetwheels.co.uk/about-us/about-the-boat/>, Cited 24 June 2013.
37. Wikipedia. *NMEA 2000*, In: English Wikipedia http://en.wikipedia.org/wiki/nmea_2000, Cited 23 June 2013.
38. Raphael Zulliger and Edouard TISSERANT. *CANopen SlaveLib* <http://canopen.sourceforge.net/projects/slavelib/slavelib.html>, Cited 23 June 2013.

A.2 Documentation librairies CAN (Anglais)

SimpleCAN

Documentation guide

I. The protocol

The SimpleCAN library for Arduino contains the C++ file, its header and a configuration header. The last could be modify by the user to fulfill his need. It contains the number of sensor/actuator entries, the synchronisation period and the definition of network variables.

The *SimpleCAN.cpp* file contains all the functions of the protocol:

- The initialisation function set the bit-rate of the bus and also reset masks and filters from the previous use.
- The election function is in charge to set a leader on the bus responsible of the synchronisation task. This node will periodically send its time and other nodes will synchronise them on it with a PID controller.
- The synchronisation set the Timer 1 of the Arduino in CTC (Clear Timer on Compare match) mode according to the period defines in the configuration file. This period must be the same for each node.
- Two functions to set MCP2515 filters:

A function which set filter according to a specific ID. In this configuration, the first filter on reception buffer 0 is set automatically to receive synchronisation ID. It takes into parameters a table of ID. Because the MCP2515 has only 6 filters (2 on buffer 0 and 4 on buffer 1), with this configuration each nodes can only have 6 subscriptions to other data (including synchronisation messages).

A function which set filter according to a range of ID. Actually the range is 7 long (bits <3:0>) but could be modify. Only 6 ranges have been defined:

Range 1 : IDs 48-55	Range 4 : IDs 72-79
Range 2 : IDs 56-63	Range 5 : IDs 80-87
Range 3 : IDs 64-71	Range 6 : IDs 88-95

Other ranges could be added easily with the same method. Both functions use unitary

functions *setFilterRXFn* which set each filter according to an ID.

- A debugging function which allows the transmission of text messages into one or more CAN frame. For better results, use only short message (< 8 characters) to use only one CAN frame. Emitting too much frames for debugging messages will saturate the bus.

II. The Universal CAN library

This CAN library is based on Fabian Greif work (<http://www.kreatives-chaos.com/>). The main modification of this library was to make it compatible with Arduino boards and usable with the Arduino IDE. Originally it contains compatibility for multiple CAN controllers such as MCP2515, AT90CAN and SJA1000. Because only MCP2515 are integrated into the CAN bus shield for Arduino and into our CANinterfacer, our library only contains references to the MCP2515. Further modifications to add the others will be done in the near future.

This library contains several functions to interact with the CAN, for example:

- Send and receive message. Because MCP2515 has 3 transmission buffers, we allocated the buffer TX2 for synchronisation messages. We let TX0 and TX1 free for standard CAN message.
- Change mode of the controller (NORMAL, LISTEN ONLY, CONFIGURATION...)
- Activate different kind of interruption. For example interruption on buffer TX2 empty for synchronisation.

To use it just copy/paste the directory into Arduino library folder.

III. CAN Bootloader and reprogramming

To reprogram nodes through the CAN bus we need at least three parts:

- An Arduino sketch which will act as a CAN2USB interfaucer between the PC and the node which need to be reprogrammed.
- A python script to send the new sketch by serial port to the CAN2USB board.
- A CAN bootloader inside the targeted board to receive the new sketch and reprogrammed itself.

1. Arduino sketch

The board which will run this sketch will act as a CAN2USB interface: it receives USB data from the python script and sends CAN message to the node to reprogram it.

Important point:

The board which needs to be reconfigured must be into its bootloader section, that is to say a reset has to be done.

Two ways to do that:

- A power-up reset occurs when the board is switch OFF then ON.
- A reset could be done into the software section by calling the routine:

```
asm volatile(" jmp BOOTLOADER_ADDRESS")
```

BOOTLOADER_ADDRESS corresponds to the specific address of the board.

Example:

For ATmega32U4 with a 2048 words bootloader (see Datasheet of the microcontroller), the bootloader starts at address 0x3800 (14336 in decimal).

The instruction will be:

```
asm volatile(" jmp 14336")
```

In our program, we have chosen to work with a software reset called by a specific CAN message (all data at 0xFF).

That is why we send a CAN message into the

setup function of our sketch in order to reset the board before starting the python script. The message identifier must be changed to match the target.

With this implementation, each node must contain a verification part to check if a reset CAN message is received.

Example:

```
if(can.mcp2515_get_message(&msg))  
{  
    for(int i=0;i<msg.length;i++){  
        if(msg.data[i]==0xFF)  
            restart=true;  
        else  
        {  
            restart=false;  
            break;  
        }  
    }  
}  
  
if(restart){  
    asm volatile (" jmp 14336");  
}
```

2. Python script

The python script is used to send an Arduino sketch by using the CAN bus. It will transfer the data through USB to a node in charge of the programming aspect. This node will transform USB data into CAN messages and send them to the target.

How to use it:

- Open a terminal on the directory of the python script.
- Simply type:

```
python bootloader.py -i BOARD_ID -p COM_PORT -f FILE.hex
```

Arguments:

- BOARD_ID corresponds to the ID of the targeted board.
- COM_PORT corresponds to the port where the CAN2USB is wired to the PC.
- FILE.hex is the new sketch which will be uploaded to the target.

3. CAN Bootloader

The CAN Bootloader should be implemented on each nodes of the bus. For the moment this bootloader is not compatible with USB, so after this step it will be impossible to use the Arduino IDE to upload a new sketch unless you put back the original bootloader of your Arduino board.

How to use it:

config.h

This file has to be modified in order to fit your configuration

1. #define BOOT_LED PORT,NUMBER
2. #define MCP2515_CS PORT,NUMBER
3. #define MCP2515_INT PORT,NUMBER

Makefile

The makefile contains three areas that need to be modify in order to fit your project

1. MCU = your board (e.g ATmega32U4)
2. F_CPU = your frequency clock
3. BOOTLOADER_BOARD_ID = the identifier you want for this board (**MUST be unique**)

Uploading

1. Open a terminal and go to the directory of the bootloader.
2. Type "make clean" and "make all" to generate the .hex file of the bootloader
3. Type:

```
avrdude -p BOARD_TYPE (e.g m32u4 for Leonardo) -c PROGRAMMER (e.g usbtiny) -U flash:w:bootloader.hex:i
```