Projet M2 LSE 2016

Pilotage automatique d'un voilier via un bus CAN Arduino

Auteur:
Romain Le Forestier
M2LSE

Encadrant:
Goulven Guillou

Sommaire

1 Présentation du projet 3
1.1 Sujet 3
1.2 Déroulement du projet 3
1.3 Projet similaire 4
2 Présentation du matériel utilisé 4
2.1 Carte Arduino 4
2.1.1 Présentation de la carte 4
2.1.2 Présentation du fonctionnement du protocole CAN 5
2.2 Interface SeaTalk 6
2.2.1 Le protocole SeaTalk 6
2.2.2 La carte d'interface SeaTalk 9
3 Présentation des programmes 10
3.1 La communication Seatalk - Arduino 10
3.2 La communication Arduino – ordinateur 12
4 Problème rencontrer 14
5 Conclusion 15
6 Annexe 16
6.1 Référence 16
6.2 Programmation sur le bus CAN 16

1 Présentation du projet

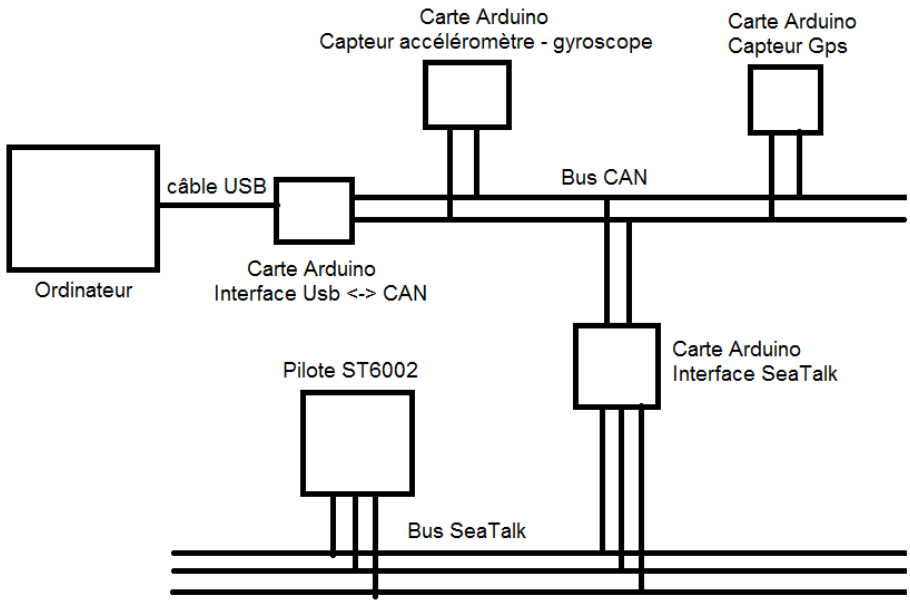
1.1 Sujet

Aujourd'hui nous disposons d'un ensemble de noeuds Arduino associés en un bus CAN permettant de récupérer un ensemble de données issues de capteurs positionnés sur un voilier et en particulier de décoder des données au format propriétaire SeaTalk. Le projet a pour objectif d'utiliser ces données pour commander via le bus SeaTalk (en se faisant passer pour une télécommande) un vérin de pilote automatique. Le projet comporte le développement d'une interface simple pour PC (PC qui sera raccordé à un noeud) pour d'une part visualiser certaines données (type tableau de bord) et pour d'autre part contrôler le pilote via un algorithme de contrôle/commande (qui pourra être un simple PID dans un premier temps) le noeud ne servant que d'interface entre le PC et le bus SeaTalk. Ce même noeud devra pouvoir également se substituer au PC, c'est-à-dire faire tourner un algorithme de pilotage de manière autonome.

1.2 Déroulement du projet

Le projet devait se dérouler de la manière suivante, premièrement réaliser l'interfaçage entre les cartes Arduino et le matériel Raymarine du pilote automatique existant sur le voilier. Les sources du projet sont disponibles sur un dépôt [GithHub].

Les cartes Arduino proviennent d'un projet précédent et devront être réutilisées. Ensuite, implémenter une interface pour permettre la communication entre un PC et une carte Arduino qui servira d'interface entre le réseau CAN et le pc. L'interface devra permettre une communication bidirectionnelle afin d'implanter un pilote sur l'ordinateur basé sur les données disponibles sur le réseau de carte Arduino et Raymarine et ensuite envoyer des données au pilote du bateau basé sur ces données.



Représentation du circuit globale mettant en oeuvre le matériel utilisé

1.3 Projet similaire

Ce projet s’est inspiré d'autre réalisation de pilote déjà existante, le projet [Freeboard] apportait une approche intéressante, car il impliquait la réalisation d'une interface entre une carte Arduino et du matériel Raymarine avec un ordinateur pour contrôler le pilote automatique.

Un second projet plus simple présenter sur le site [Arribasail] présent l'interfaçage entre une carte Arduino Uno et un metteur RF Raymarine pour tester un afficheur, ce qui met en oeuvre une transition de donnée Seatalk émise par une carte Arduino et afficher sur du matériel Raymarine en ayant transité sur un bus SeaTalk.

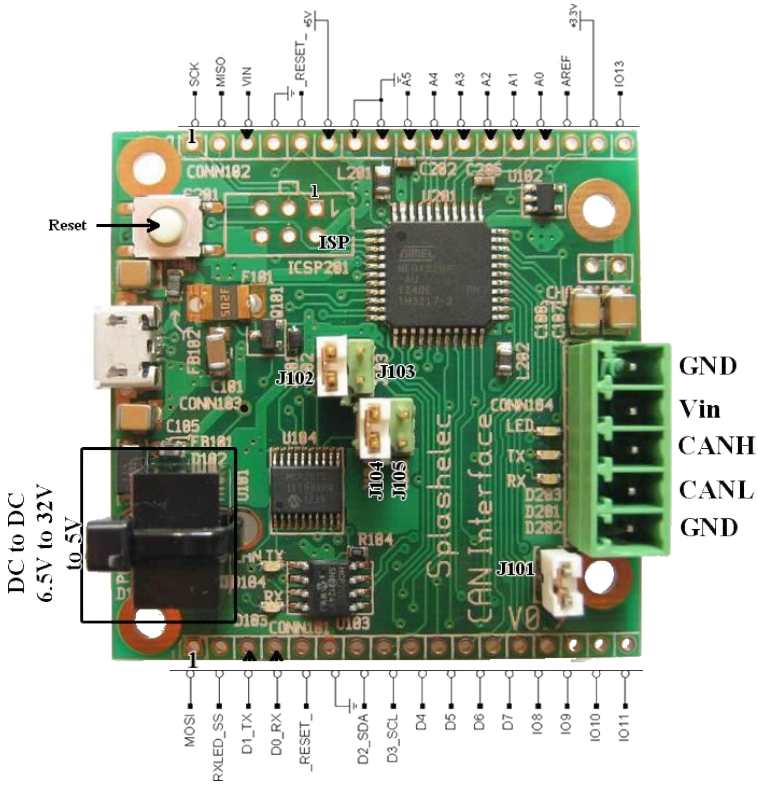
Un troisième projet présenter sur [sympatico.ca] présente le remplacement de la télécommande Raymarine ST4000+ qui permet de modifier le cap du pilote automatique par un émetteur RF avec quatre boutons qui transmet les commandes à un module RF en 433MHz relier à une carte Arduino, qui convertie les commandes en SeaTalk pour les envoyées au pilote automatique.

2 Présentation du matériel utilisé

2.1 Carte Arduino

2.1.1 Présentation de la carte

Les cartes Arduino utilisées pour ce projet sont celle utilisée lors du projet de TER effectuer précédemment, le projet de TER consistait a réaliser un réseau CAN entre les cartes Arduino le projet et les sources du TER sont disponible sur un [dépôt Github TER].

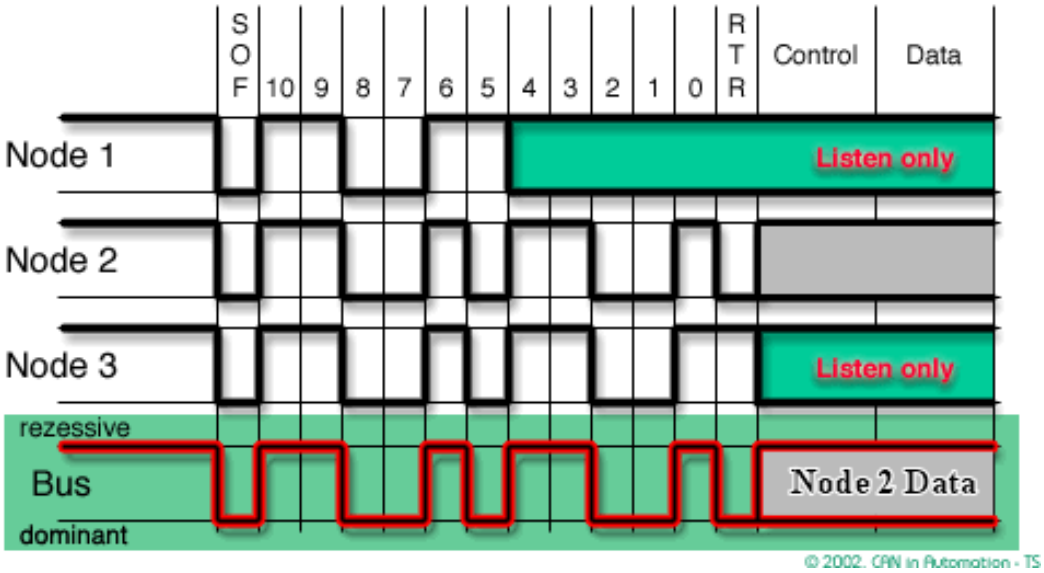


Carte Arduino utilisé pour le projet

Les cartes Arduino sont basées sur un microcontrôleur Atmel mega32u4, correspondant au processeur utilisé sur les cartes Arduino Leonardo. Un contrôleur de bus CAN, le MCP2515, est installé sur la carte et permet la réalisation du réseau de carte Arduino via le bus CAN. Pour la réalisation du réseau ont utilisera la librairie CAN utilisé lors du projet de TER.

2.1.2 Présentation du fonctionnement du protocole CAN

Le protocole CAN est un bus de terrain développer pour le secteur automobile par Bosch. Le bus utilise 2 fils pour communiqué, c'est un bus bidirectionnel sans maître avec une vitesse maximale de 1 Mb/s . Un seul noeud du réseau peut émettre à la fois ce qui correspond à un principe de broadcast, un noeud qui parle les autres écoutes et récupère les données s'ils en ont besoin. Les messages sur le bus sont différentiés par un identifiant de tram dans le champ d'arbitrage, car le noeud qui émet avec la plus petite valeur d'identifiant sera prioritaire, chaque noeud écoutant ce qui est émis quand il écrit, les noeuds avec une valeur d'identifiant supérieur à celui qui émet passeront en mode "écoute" et attendront que le bus soit de nouveau libre pour émettre.



Arbitage sur le bus CAN

Le 0 sur le bus CAN est dominant ce qui permet de différentier les identifiants des messages lors d'une phase d'arbitrage. Sur cette image représentant une phase d'arbitrage entre 3 noeuds sur un bus CAN, on peut remarquer que le noeud 2 remporte l'arbitrage, car il a la valeur d'identifiant la plus faible. On peut également constater que la valeur des autres noeuds est écrasée par la valeur prioritaire et que chaque noeud passe en mode écoute dès qu'il détecte qu'il a perdu la phase d'arbitrage sur le bus.

Pour mettre en oeuvre une communication sur le bus on constate donc que chaque message doit avoir un identifiant unique afin de différentier chaque noeud et grandir que 2 noeuds n'utilise pas le même identifiant en même temps se qui invaliderait la phase d'arbitrage. Pour ce faire lors du projet de TER nous avions réaliser un fichier header pour la communication CAN qui était commun a tous les noeuds ce qui garantissait que 2 noeuds n'utilisent pas le même identifiant et permettait également de garantir que chaque noeud puisse récupère la bonne donné en lecture. Pour le projet nous allons donc ajouter de nouvel identifiant. On utilisait le mode standard de CAN ce qui permettait de codé jusqu'a 2^{11} identifiant de message différent, la librairie utilisait permettait également d'utiliser le mode étendu, qui permettait de codé 2^{29} identifiant, mais cette quantité n'est pour le moment pas nécessaire pour le réseau que l'on souhaite réaliser, car il ne comporte que 5 noeuds.

2.2 Interface SeaTalk

Pour rendre notre réseau de carte Arduino compatible avec le matériel Raymarine embarqué sur un voilier, il faut réaliser une carte d'interface quipu convertie la sortie série de la carte Arduino en un signal compatible avec le langage des réseaux Raymarine le SeaTalk.

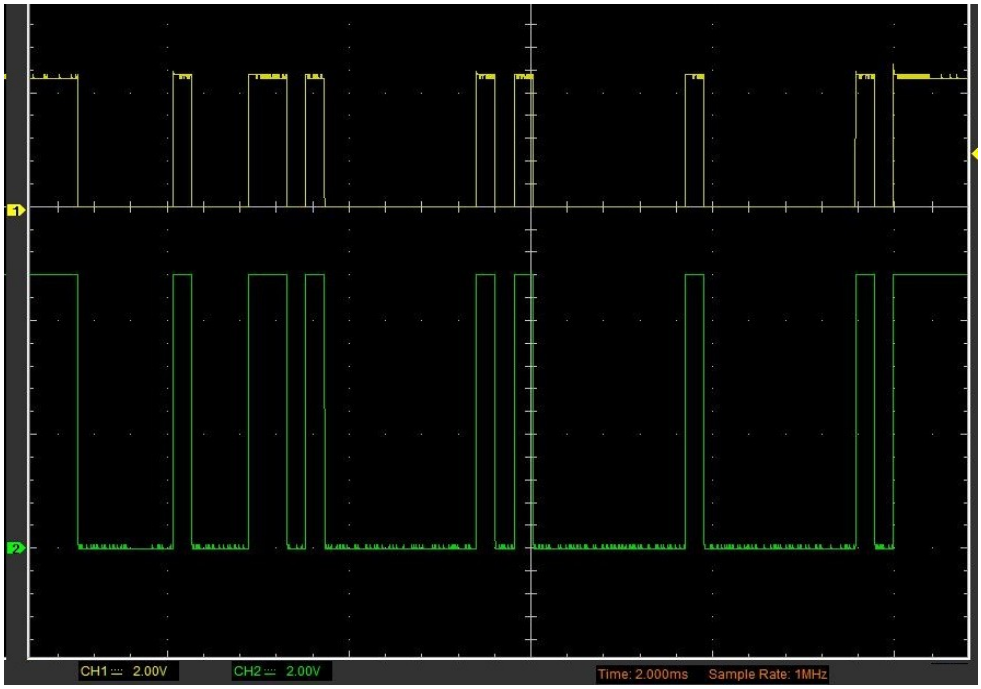
2.2.1 Le protocole SeaTalk

Le protocole SeaTalk est un langage de communication série inventée par Raymarine vers 1989 pour réaliser une communication entre un pilote de voilier et les instruments de bords. C'est un protocole série propriétaire qui a été en partie documenté par [Thomas

Knauf] sur son site. Cette documentation nous a servi de référence pour interpréter les trames émises par le pilote et formata les trames que l'on envoyait sur le pilote pour tester le fonctionnement du réseau en essayant d'afficher un cap sur le pilote en transmettant l'information via le réseau.

L'interface matérielle SeaTalk est basée sur une connexion avec 3 fils, 2 fils pour l'alimentation en 12V afin de transmettre l'alimentation aux appareils reliés au réseau et un fil pour transmettre les données. Les données sont transmises en mode série avec une tension entre 12V qui représente l'état libre et le 1 logique, et le 0V qui marque les espaces et le 0 logique. Les données sont transmises à une vitesse de 4800 Bauds. Le réseau est dit sans mettre, il n'y a donc pas d'ordre de priorité pour l'ordre de transmission des données par le matériel. Chaque appareil qui veut émettre et qui détecte que le bus est libre peut envoyer ses données, le 0 étant prioritaire sur le bus, chaque matériel écoutant ce qui se passe sur le réseau, si l'émetteur détecte que ce qui circule sur le réseau ne correspond pas a e qu'il émet, il arrête la transmission immédiatement et attend que le bus soit de nouveau libre pour émettre. Les autres appareils reliés au réseau détecteront l'erreur, car la trame sera trop courte pour être valide. Le bus est considéré comme inactif quand aucune donné n'a circuler depuis 10/4800 secondes soit envieront 2 ms. Chaque caractère émis est composé de 11bits, 1 bit de début, 8 bits de données, le bit de donnée le moins significatif est émis en premier, 1 bit de commande qui marque le premier caractère d'un datagramme et un bit d'arrêt.

Les



Une trame de message en SeaTalk

datagrammes de message comprennent entre 3 et 18 caractères, le premier caractère représente le type de commande du message. Le deuxième caractère représente l'attribut du message, les 4 bits les plus significatifs sont mis à 0 où sont utilisées une partie des données, les 4 bits les moins significatifs sont utilisés pour indiquer le nombre de

caractères additionnel n, la taille de la trame du message correspond donc a 3+n. Les autres cratères de la trame correspondent à des données.

Pour transcrire la signification des trames des messages, nous avons utilisé les valeurs des [datagrammes reconnus] par Thomas Knauf.

```
184 6 22 0 0 0 FE 0 0
186 41 83 7C
190 0 3
186 41 84 7B
186 41 81 7E
```

Différente trame de donnée reçue lors d'un échange entre une carte Arduino et le pilote ST6002

Les commandes si dessus ont été récupérer lors d'un échange entre une carte Arduino méga2560 qui émettait un cap et un pilote automatique Raymarine ST6002. Avec les données disponibles sur le site de Thomas Knauf, on a formé une trame pour envoyer au pilote automatique un cap à afficher, cette commande correspond à la première ligne.

La trame "184 6 22 0 0 0 FE 0 0" qui est donc émise par la carte Arduino a été formatée par rapport à l'entrée suivante des références du site de Thomas Knauf :

"84 U6 VW XY 0Z 0M RR SS TT Compass heading Autopilot course and Rudder position"

Le 184 correspond a la trame 84 avec son bit de commande mis à 1 pour identifier le début du message, cette trame permet de transmettre la valeur du compas qui correspond aux lettres UVW, la position position du gouvernail qui correspond aux lettres RR (0xFE correspond a 2° vers la gauche) calculer avec le complément a 2, la direction dans laquelle le bateau tourne représenter par le bit le plus significatif de U, si le bit est à 1 le bateau tourne vers la droite sinon s'il est a 0 vers la gauche. La lettre M permet de configurer des alarmes sonores. Les lettres SS et TT permettent d'afficher des messages et indiquer s'il provient d'un ordinateur de course.

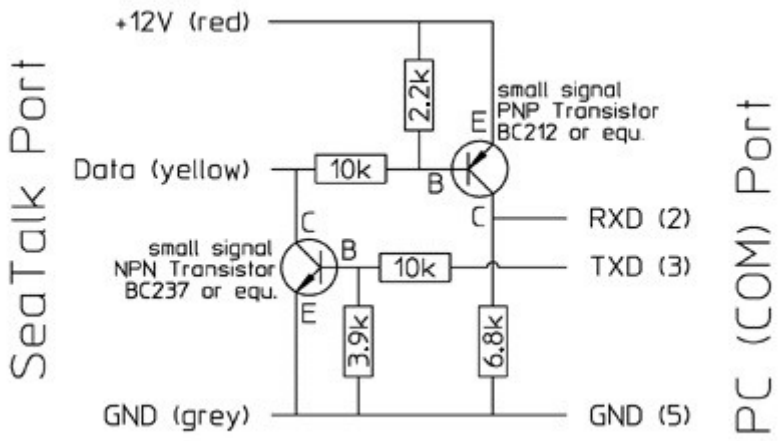
Les trames émises par le pilote sont plus simples à interprété, la trame "190 0 3" est émise toutes les 2 secondes et correspond a l'identification du matériel, donc identifie qu'un pilote ST6002 est relié aux bus SeaTalk.

Les trames "186 41 83 7C", "186 41 84 7B" et "186 41 81 7E" correspond a des appuis sur les touches du pilote automatique, "83 7C" correspond à un appui sur la touche +10 pendant 1s, "81 7E" correspond a un appui sur +1 pendant 1s et la trame avec "84 7B" est émise par le pilote lorsque les touches sont relâchées.

2.2.2 La carte d'interface SeaTalk

Pour permettre la communication entre la carte Arduino et le bus SeaTalk il fallait réaliser une carte qui permettrait de servir d'interface entre les signaux série qui utilise 2 fils sur la carte Arduino et le fil de communication du bus SeaTalk, ainsi que de permettre une transmission entre les signaux en 5V de la carte Arduino et les signaux en 12V du bus SeaTalk.

Le site de Thomas Knauf proposait un schéma pour une carte d'interface entre une interface RS232 et le bus SeaTalk.



Interface RS232 <-> SeaTalk proposé par Thomas Knauf

Les cartes Arduino ne sont cependant pas directement compatibles avec le RS232, en effet en mode attente le port émetteur des cartes Arduino émet du 5V en continu alors que le RS232 est à 0V en mode attente. Les signaux émis par la communication série des cartes Arduino est donc inversé par rapport aux formats RS232, de ce fait il a fallut apporté des modifications u schéma proposé par Thomas Knauff.

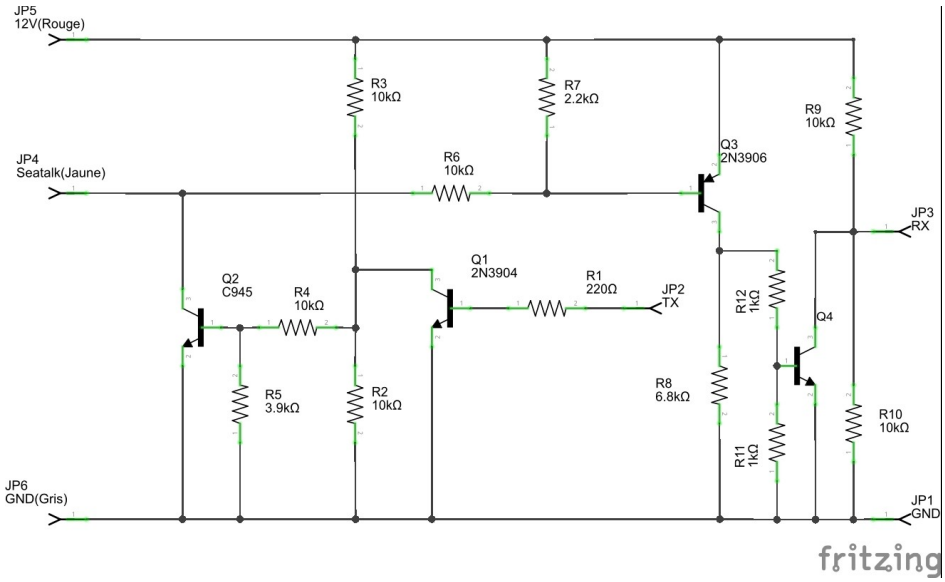


Schéma de la carte d'interface entre la carte Arduino et le bus SeaTalk

Ce schéma a été tester lors de la communication qui a permis de récupérer les trames de message présenté dans le point 2.2.1 lors de la communication entre une carte Arduino mega2560 et un pilote ST6002.

Les transistors ajoutés par rapport au schéma proposé par le site de Thomas Knauf permettent d'inverser les signaux de la carte Arduino pour les rendre compatibles à un format RS232 ce qui correspond au format pour lequel la carte de Thomas Knauf est prévue. La modification du schéma a été apportée lorsque l'on à détecter un problème de compatibilité entre le format des trames Arduino et les trames qui était émise par le pilote ST6002 lors l'affichage des trames des massages sur un oscilloscope ce qui a permis de détecter un problème matériel et non logiciel.

3 Présentation des programmes

3.1 La communication Seatalk - Arduino

Pour permettre aux cartes Arduino d'émettre des messages en SeaTalk et de les interpréter il fallait modifier la librairie qui gérait les ports série des cartes afin qu'elles puissent émettre le bits de commandes soit émettre 9bits à la place des 8bits qu'elles émettent normalement.

On m'a fournis au début du projet, une modification de la librairie HardwareSerial réalisés par [François Gautrais] permettant la lecture des trames SeaTalk. Cette librairie fut pratique pour lire les données émises par le pilote ST6002, mais n'était pas prévue pour réaliser l'écriture de trame SeaTalk.

Pour pouvoir réaliser la lecture et l'écriture, j'ai trouvé la source des modifications apportées par François Gautrais qui provenait d'un sujet du [forum Arduino] à propos de la communication 9bits SeaTalk. Cette communication en 9 bits est possible sur les ports

série des microcontrôleurs ATMEL qui sont [compatible 9bits]. Cependant la librairie qui gère les communications dans le compilateur Arduino AVR n'est pas prévu pour être configuré pour utilisé la communication avec 9bits. La librairie responsable de la gestion des liaisons série en Hardware est HardwareSerial et se trouve au coeur du compilateur dans le logiciel Arduino et ce situe a l'adresse "\hardware\Arduino\avr\cores\Arduino\".

La modification de cette librairie doit donc conserver la compatibilité avec une communication en 8bits afin par exemple de conserver la communication série USB pour un des noeuds du réseau de carte sur le bus CAN. Cette librairie doit être modifié à cet emplacement, car l'utilisation de la liaison série matérielle doit être transmise par exemple lors de la compilation a la librairie "Arduino.h" afin de garantir lors de la compilation que le port n'est pas configurer pour 2 utilisations différentes, par exemple sortie série et entré de données. La librairie Arduino mettant en pul-up par défaut les entrées qui ne lui sont pas signalées comme utiliser lors de la compilation.

Un autre problème est survenu lors de la compilation de la librairie qui datait de 2012, le compilateur AVR utilisé par Arduino marquait les interruptions qui permette d'intercepter l'arrivé de caractère dans l'UART série était marqué comme obsolète. Le site [Atmel] présente les interruptions du compilateur prises en charge par le compilateur AVR., sur cette page un tableau récapitule les différents vecteurs d'interruption ainsi que l'équivalence entre les anciens vecteurs d'interruption et ceux actuellement utilisés.

Une fois la librairie modifiée et la carte d'interface fonctionnelle, on a pu réaliser un programme qui émettait caractère par caractère une trame de message SeaTalk.

La programmation de l'émission d'une trame SeaTalk par une carte Arduino se programme de façon identique à l'émission d'une trame sur un port série de façon standard pour la carte Arduino. La modification de la librairie apporte cependant 2 fonctions supplémentaires qui sont spécifiques à la communication en SeaTalk.

```
Serial1.begin(4800, SERIAL_9N1);
```

La fonction d'initialisation prend en charge un second argument qui permet d'indiquer au compilateur que l'on souhaite avoir une communication sur 9bits.

```
Serial1.write9(c, true);
```

La fonction d'écriture en 9bits, le premier paramètre correspond au caractère émis qui est un uint16_t, le second paramètre est un booléen qui indique à la fonction si le bit de Comanche doit être mis à 1.

Il faut ensuite vérifier que le port est libre avant d'émettre un message SeaTalk en vérifiant que le buffer est vide. Le port écoutant l'émission qui est faite permet également de vérifier que le caractère émis est identique au caractère reçu, ce qui implique de prévoir une seconde émission du message si le message est corrompu afin de correspondre a une trame de message correcte. Lors des tests je n'ai pas réussi a vérifié les trames des messages lors de leur émission, mais après qu'elle a été transmisse, une latence dans le circuit qui sert d'interface entre la carte Arduino et le bus SeaTalk semblait être le problème, mais il s'agissait d'une latence minime. Il faut également pensé lors de la

comparaison du premier caractère émis à ajouté "0x100" a la valeur que l'on compare, car le bit de commande a été ajouté lors de l'émission du message.

La réception d'un message SeaTalk correspond a lire le buffer des messages et de commencer a interprété les trames pour toutes les valeurs de caractère reçu supérieur a 0x100, car ces caractères marques le début d'une trame de message, l'interprétation du message ce fait ensuite avec la correspondance aux références des trames du site de Thomas Knauf qui indique les formules pour retrouvé les valeurs des informations transmises dans les trames SeaTalk référencé.

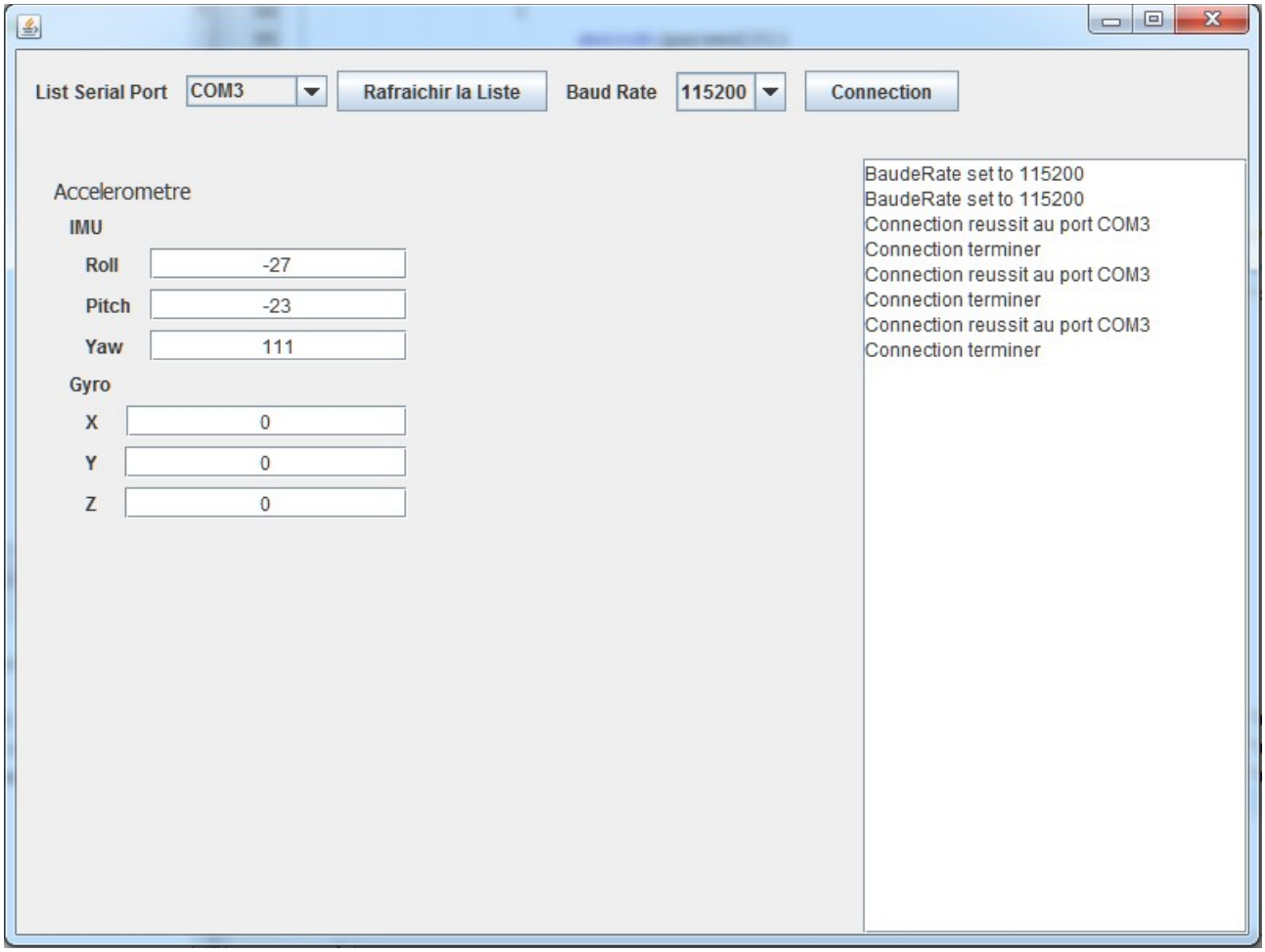
3.2 La communication Arduino – ordinateur

Pour établir une communication entre la carte Arduino et un ordinateur il faut pouvoir se connecter au port série de l'ordinateur, ces ports sont identifiés sous la référence ports "COMx" sous Windows où x est un nombre attribué par Windows et "ttyx" sous Linux où x correspond à un appareil communicant comme un terminal. Deux langages de programmation proposent des librairies compatibles avec plusieurs systèmes d'exploitation pour exploiter les ports série.

Le langage python propose la librairie [pyserial] qui permet d'établir une liaison série avec un port série, il suffira ensuite d'envoyer ou d'émettre des trames de message de type string préformaté en ligne par ligne pour établir une communication entre le pc et la carte Arduino.

Le langage Java disponible sur plusieurs systèmes d'exploitation propose plusieurs librairies pour la communication série. La plus commune est [RXTX] qui est relativement standard pour la communication série. Elle ne nécessite cependant d'installer un libraire natif dans le dossier java de l'ordinateur désirant exécuter un programme qui utilise cette librairie ce qui réduit sa portabilité. Une autre librairie est [Serial communication manager] qui à pour avantage de proposer des librairies natives pour Linux, Mac, Solaris et Windows directement embraquées dans le fichier jar ce qui rend la portabilité de la librairie plus simple. Cette librairie propose également des listener de message préconfiguré ce qui facilité la programmation.

Plus habitué au langage Java, j'ai préférer utiliser ce langage pour développer une interface simple de communication entre la carte Arduino USB et un ordinateur.



Interface de communication entre la carte Arduino et java

Cette interface été développée pour tester la communication entre le réseau CAN et l'ordinateur via un bus USB. L'interface affiche les données envoyer par un accéléromètre qui est relier a une des cartes du bus CAN. Les données sont donc émises par une carte Arduino sur le bus CAN et récupérer par la carte qui fait sert de connections entre le bus CAN et l'ordinateur, la carte Arduino transcrit l'information qui transitait sur le bus CAN en valeur et les émet via USB sur le port série du pc sous forme de chaine de caractère se terminant par un passage à la ligne, le programme Java, une fois la connections avec le port série établie n'as plus qu'as formaté les donné textuel pour les exploités. Les chaines de caractère émises par la carte Arduino sur port série sont les suivantes:

MSG_GYRO_X_Y_Z

GYRO x:0 y:-8 z:-21

MSG_IMU_PHI_THETA_PSI

IMU phi:7 theta:2 psi:56

L'information que l'on souhaite afficher dans l'interface sont donc les lignes qui commence par GYRO et IMU. En séparant le texte reçu par ligne et en séparant chaque ligne par les espaces, ont peux savoir l'information que contient la ligne et si l'on souhaite

l'exploiter en comparant le premier mot de la ligne, il faut ensuite séparer dans les lignes qui nous intéresse la valeur de chaque groupe de caractères l'identifiant de la valeur, car l'on souhaite recouper que la valeur des champs.

Le format des chaines de caractère émises par la carte Arduino a été choisi pour être lisible dans un terminal, même si l'on peut extraire la valeur pour 'afficher dans une interface on peut imaginer un format plus simple avec chaque information transmise par la carte Arduino commençant par un identifiant de trame, connaissant ensuite le format de la trame l'exploitation des informations contenues dans la trame est relativement simple.

La communication entre le pc et la carte Arduino fonctionnent sur le même principe, chaque ligne permettant de séparer un type d'information, et un identifiant de trame permettant de différencier le contenu des messages. La fonction [readStringUntil(terminator)] permet de récupérer une String du buffer de message jusqu'au caractère de terminaison configuré.

On peut également utiliser fonction de base en C pour réaliser cette opération comme indiqué pour cette question posée sur le site [arduino.stackexchange.com] pour récupérer des informations provenant d'une entré série en donné exploitable par une carte Arduino.

4 Problème rencontrer

Plusieurs problèmes se sont posés lors du déroulement du projet. Le projet était inspiré par des projets existants qui semblaient similaires à celui que nous essayons de résoudre. Le plus gros problème rencontré a été l'interfaçage entre la carte Arduino et le bus SeaTalk qui malgré les différents projets similaires ne fonctionnait pas exactement comme présenté dans ces projets. En effet le problème de compatibilité de signaux entre la carte Arduino et ceux du pilote ST6002 n'était pas résolu par la carte proposée par Thomas Knauf malgré le fait que la solution était utilisée dans d'autres projets avec suces. Ce problème a été résolu par l'analyse des signaux émis et des signaux attendus avec l'aide d'un oscilloscope ce qui a permis de réalisé l'interface Arduino SeaTalk qui a finalement fonctionné avec une carte Arduino mega en fin de projet.

Un autre problème a été cause par la carte Arduino utilisée, en effet les microcontrôleur utilisé corresponde aux cartes Leonardo, les microcontrôleurs atmega32u4 qui sont connus pour posé des problèmes avec leur port série. Après la lecture de la documentation et de différents forums il apparait que le problème est dû à la configuration du port série et de la liaison USB qui sont matérielle dans le microcontrôleur alors que les autres microcontrôleurs utilisés pour Arduino non que des entré série, l'entré série une étant utilisé alors pour la connections USB et convertie avec un circuit externe. Le port USB embarqué dans le microcontrôleur utilisé entre en conflit avec le port série si l'on essaye de configuré le microcontrôleur pour utilisé la liaison série comme la liaison 0. La libraire gérant le port série matérielle pour la carte Léonardo est différente de

HardwareSérial utilisé par les autres cartes. Le problème de compatibilité avec le microcontrôleur n'a pas été résolu.

Un autre problème provenait de la librairie utilisée, essayant au début d'utiliser la librairie de François Gautrai qui lisait correctement les trame SeaTalk, cette dernière ne prenait pas en charge l'écriture sur le bus SeaTalk et ne mettait jamais le bit de commande. Ce problème a été résolu par la récupération d'une librairie qui était proposée sur le forum Arduino spécifiquement pour permettre l'écriture sur le bus SeaTalk. Cette dernière a nécessité des modifications au niveau des interruptions qui était marqué comme déprécié par le compilateur pour pouvoir être compilé. Cette librairie semble fonctionner correctement avec les cartes Uno, Mega2560 et nano (message émis et reçu correctement avec la bonne forme de signal pour les trames SeaTalk testées).

Un dernier problème c'est posé lors de l'interfaçage avec l'ordinateur lors de l'utilisation de la bibliothèque RXTX de java qui a mis en avant sont problème de portabilité, la librairie native que j'avais installée n'était pas placée dans la bonne installation de java ce qui empêchait la librairie de fonctionné et ensuite l'utilisation d'une librairie qui ne correspondait pas au type de java installation 32bits sur un système 64bits a planté mon système. J'ai donc préféré opté pour un libraire série qui embarquait les librairies natives afin de simplifié l'utilisation et garantir un minimum de portabilité.

5 Conclusion

Ce projet fut malgré tout intéressant, car il impliquait la mise en oeuvre de réseaux, de système embarqué et de logiciel de plus haut nivaux sur l'ordinateur. La complexité du système dans son ensemble étant diminué lorsque l'on prend chaque élément du système séparément ce qui permet d'apporter une approche interface par interface et ensuite de mettre en relation les différents systèmes qui fonctionne.

Ce projet est intéressant, car il approche les pilotes automatiques embarqués sur des voiliers ce qui représentait un système que je connaissais peut. Ce projet m'a permis de découvrir les différentes solutions proposées et de voir comment elle fonctionnait.

Ce projet propose également une ouverture avec la possibilité de crée une interface sur l'ordinateur plus complexe avec par exemple la mise en place d'une carte pour aidé a la navigation pour exploité les donné du GPS.

Le but d'implémenter un algorithme de pilote sur le pc n'a pu être atteint à la fin du projet par manque de temps dû aux différents problèmes, mais devrait être désormais possible avec les différents modules du système qui semble fonctionner.

6 Annexe

6.1 Référence

[GithHub] https://github.com/surpriserom/Pilotage-automatique-d-un-voilier-via-un-bus-CAN-Arduino

[Freeboard] http://www.42.co.nz/freeboard/

[Arribasail] http://blog.arribasail.com/2015/08/seatalk-using-arduino.html

[sympatico.ca] http://www3.sympatico.ca/ericn/

[dépôt Github TER] https://github.com/surpriserom/Protocole_SimpleCan_Arduino

[Thomas Knauf] http://www.thomasknauf.de/seatalk.htm

[datagrammes reconnus] http://www.thomasknauf.de/rap/seatalk2.htm

[François Gautrais] https://github.com/FrancoisGautrais/SeaSerial

[forum Arduino] http://forum.arduino.cc/index.php?topic=91377.0

[compatible 9bits] http://www.atmel.com/images/atmel-7766-8-bit-avr-atmega16u4-32u4_datasheet.pdf page 198

[Atmel] http://www.atmel.com/webdoc/AVRLibcReferenceManual/group__avr__interrupts.html

[pyserial] https://github.com/pyserial/pyserial

[Arduino python communication par USB] http://www.instructables.com/id/Arduino-Python-Communication-via-USB/?ALLSTEPS

[RXTX] http://rxtx.qbang.org/wiki/index.php/Main_Page

[Serial communication manager] http://www.embeddedunveiled.com

[readStringUntil(terminator)] https://www.arduino.cc/en/Serial/ReadStringUntil

[arduino.stackexchange.com] http://arduino.stackexchange.com/questions/1013/how-do-i-split-an-incoming-string

6.2 Programmation sur le bus CAN

La programmation sur le bus CAN a été abordé lors du projet de TER précédent, pour programmer des nouveaux messages, il faut définir des identifiants dans le fichier "parseCan.h" dont on peut récupérer une version dans le dossier test dans le dépôt [GithHub] du projet.

Si l'on veut par exemple définir un nouveau identifiant pour émettre la direction du compas et la position de la barre qui est composée de 2 entiers, la direction du compas étant comprise entre 0° et 360° et la position de la barre peut être codé entre -180° et 180° leur valeur peut être codé sur 2 entiers soit 4 octets. Une fonction de la librairie permet de convertir un entier pour qu'il puisse être mis dans le buffer qui sera transmis sur le bus CAN.

```
void intToUChar(unsigned char buff[], int offset, int val);
```

La fonction intToUChar prend en paramètre le buffer d'envoi, l'offset qui permet de savoir à partir de quelle case il faut enregistrer la donnée et la valeur de l'entier à convertir. La conversion conserve le signe de la valeur ce qui permet d'émettre les entiers signés.

```
#define MSG_HEADING_RUDDER          0x31
```

L'identifiant pourra servir pour transmettre la donné, ainsi formé on conviendra que le premier entier du message correspondra a la valeur de direction du compas et le second entier la valeur de la direction de la barre

```
void set_seatalk_heading_rudder(unsigned char buff[], int heading, int rudder);
void get_seatalk_heading_rudder(unsigned char buff[], int* heading, int* rudder);
```

Ces deux fonctions permettront de créer le buffeur qui sera émis à partir de la direction du compas et de la barre fournis en argument et de les récupérer sur un autre noeud.

```
void ParseCan::set_seatalk_heading_rudder(unsigned char buff[], int heading, int rudder)
    {
        intToUChar(buff,0,heading);
        intToUChar(buff,2,rudder);
    }


void ParseCan::get_seatalk_heading_rudder(unsigned char buff[], int* heading, int* rudder)
    {
        *heading = ucharToInt(buff, 0);
        *rudder = ucharToInt(buff, 2);
    }
```

Le code impliqué pour l'émission et la réception de ces deux entiers est relativement simple, lors de la mise en oeuvre dans le programme Arduino, l'envoi sera réalisé de la manière suivante:

```
CAN.sendMsgBuf(MSG_HEADING_RUDDER, 0, 4, buff);
```

Ce qui correspond à émettre l'identifiant de message correspondant à la direction du compas et de la barre, en mode standard, avec 4 octets et les deux valeurs sous forme du tableau de unsigned char buff.