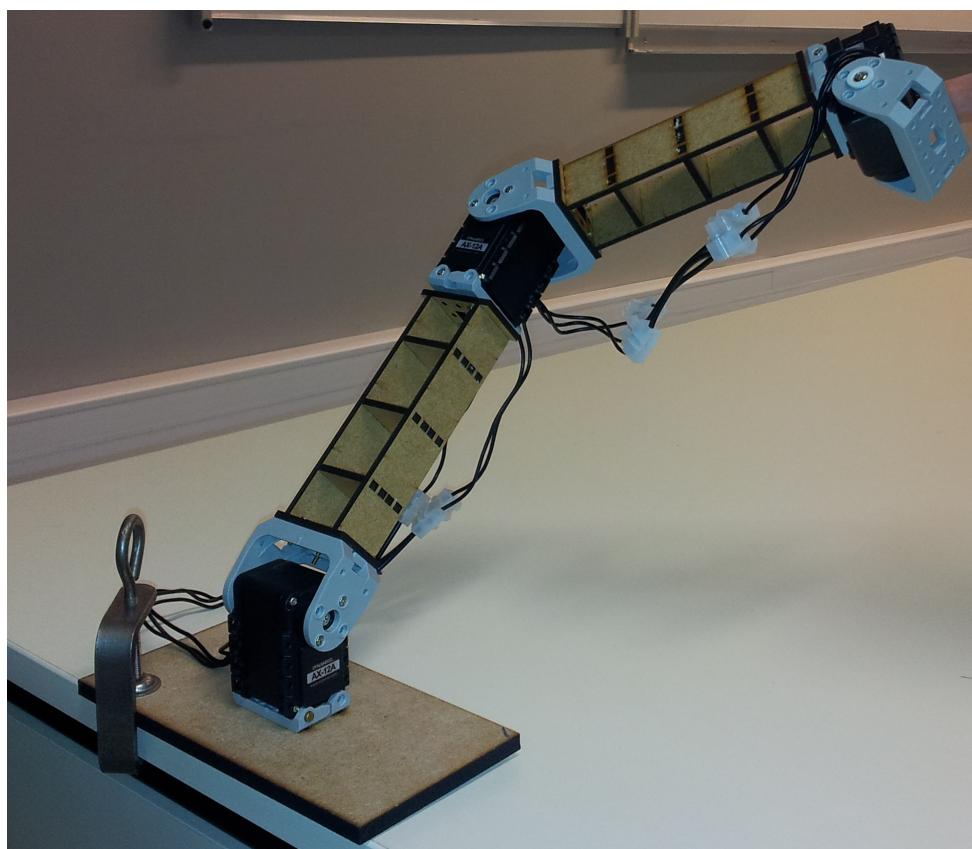


ÉCOLE NATIONALE SUPÉRIEURE DE L'ÉLECTRONIQUE ET DE SES APPLICATIONS
PROJET DE DEUXIÈME ANNÉE



Contrôle d'un bras de robot

FOURTINON Luc & BLAS François
5 juin 2013



ENCADRANT DE PROJET : M. MONCHAL

Table des matières

1 Remerciements	3
2 Naissance d'un projet	3
3 Conception du bras de robot	3
4 Développement du calcul des angles pour situer le robot	4
4.1 Le modèle de Denavit-Artenberg (D.H.)	5
4.2 Une grande simplification	7
Les équations du modèle directe	7
4.3 Le modèle inverse	7
Les équations du modèle inverse	8
4.4 Volume de travail des servo-moteurs	8
5 Présentation des servo-moteurs AX-12	8
5.1 Agencement de la mémoire	8
Liaison entre servo-moteurs	9
5.2 Caractéristiques techniques	10
5.3 Présentation du protocole de communication : USART	10
5.4 La communication côté AX-12	11
6 La carte STM32F100RB	11
6.1 Petit rappel de code bit à bit	12
6.2 Configuration de l'USART sur la carte	12
6.2.1 Activation de l'horloge	12
6.2.2 Activation du transmitter/receiver	12
6.2.3 Activation de l'half-duplex	13
6.2.4 Activation du baudrate	13
6.3 Description de la trame USART	13
6.3.1 Configuration du bit de stop	13
6.3.2 Configuration de la taille du paquet	14
6.3.3 Configuration de la parité	14
6.3.4 Activation de l'USART	14
6.4 Initialisation des GPIO de la carte	14
6.4.1 Activation des horloges	14
6.4.2 Activation des modes	15
6.5 Envoi des données	15
6.6 Les fonctions principales pour l'USART sur la carte	16
6.6.1 Initialisation de l'USART	16
6.6.2 Activation de l'USART	17
6.6.3 Envoi des données	17
6.7 Séquence de communication AX-12	18
6.7.1 Théorie	18
6.7.2 Exemple	18
6.7.3 Fonctions principales	19
7 Débuter un projet sur Keil	20
7.1 Prérequis	20
7.2 Démarrer le projet	20
8 La carte Arduino Mega 2560	22
8.1 Spécificité du code Arduino	22
8.2 Pourquoi déboguer les AX-12 ?	22
8.2.1 Débogage 1 : réinitialisation des valeurs de baud et de l'ID	22
8.2.2 Débogage 2 : Modification des valeurs du baudrate et de l'ID	22

9 Utilisation du projet sous Arduino	23
9.1 Prérequis	23
9.2 Démarrer le projet	23
9.3 Importer des bibliothéques	24
9.4 importation du code	25
10 Fonction d'automatisation des actions sur le bras	25
Description	25
11 La manette	25
11.1 Configuration des ports	26
11.2 Les registres d'Input	26
11.3 Les fonctions liées à la manette	26
11.3.1 Fonction finale d'automatisation	26
12 Organisation du code	27
Appendices	28

1 Remerciements

Nous remercions M. Monchal pour son encadrement, son suivie et ses conseils. Nous remercions également M. Chapron pour sa disponibilité, alors même qu'il n'était pas notre professeur encadrant il a accepté de nous donner un peu de son temps pour suivre l'avancée du projet et nous conseiller.

Enfin nous remercions Ares pour l'aide qu'ils nous ont apporté, en particulier Vianney Ville pour nous avoir aidé à concevoir notre manette et Brieuc du Maugouër pour son aide au moment de la découpe des liaisons entre servo-moteurs.

Plan de la présentation

La présentation va se dérouler de la façon suivante, nous allons d'abord parler de l'achat du matériel nécessaire, dans notre cas ce fut les servo-moteurs. Puis il a fallu bâtir les liaisons entre les servo-moteurs.

Ensuite nous avons développé le code qui se décompose en deux parties, calculer les équations nécessaires au placement de l'organe terminal du bras de robot et en parallèle développer le code nécessaire à la transmission numérique des ordres aux servo-moteurs.

Enfin faire le PCB de la manette qui contrôlera le robot.

2 Naissance d'un projet

Idée initiale

En première année, j'avais assisté au cours de robotique dispensé par M. Chapron mais je n'avais pas pour autant vu de robot fonctionner autrement qu'en vidéo. J'ai donc proposé à mon binôme que nous nous lancions dans la création d'un tel robot.

Réaliser un robot gardien de but capable de bloquer et renvoyer une balle sous le contrôle d'un utilisateur.

Plus concrètement l'idée revenait à cela :

Définition du sujet

Créer et contrôler un bras articulé

3 Conception du bras de robot

Choix et achat des servo-moteurs Notre choix s'est rapidement porté sur les servo-moteurs AX-12, étant déjà utilisés par l'association de robotique de l'école et ce depuis plusieurs années, cela signifiait a priori qu'ils présentaient de bonnes caractéristiques.

En effet les servo-moteurs de type AX-12 ont une précision de l'ordre du cinquième de degré puisque qu'ils ont une amplitude de plus de 200° pour 1024 valeurs. De plus ils sont très robustes et efficaces. Par contre ils sont chers, 50 euros pièces, relativement à la somme attribué au projet de chaque binôme. Ainsi nous ne pouvions pas en acheter plus de trois. Ce prix nous a même amené à nous poser la question, à savoir si nous n'allions pas acheter de simples moteurs, le problème étant qu'il aurait fallu les asservir entièrement. Or ceci n'est pas simple, cela peut même faire l'objet d'un projet entier si cet asservissement est poussé. Nous en sommes alors restés au AX-12.



FIGURE 1 – Un servo-moteur AX-12A

La particularité des servo-moteurs est qu'ils sont déjà asservis (d'où le 'servo'), ainsi le contrôle se fait entièrement en numérique. Il est possible de contrôler le servo-moteur en position(angle), au niveau du couple et de la vitesse.

Conception des bras de robot La création des liaisons entre chaque servo-moteur se fait en commençant par dessiner un patron de ces liaisons avec le logiciel de CAO : Solidworks.

Ces patrons vont ensuite servir de modèle pour être découpé dans des plaques de bois (MDF, ce sont des panneaux de fibres) au faclab de Gennevilliers, tout ceci gratuitement.

Cela permet d'avoir un résultat particulièrement propre et réussi.

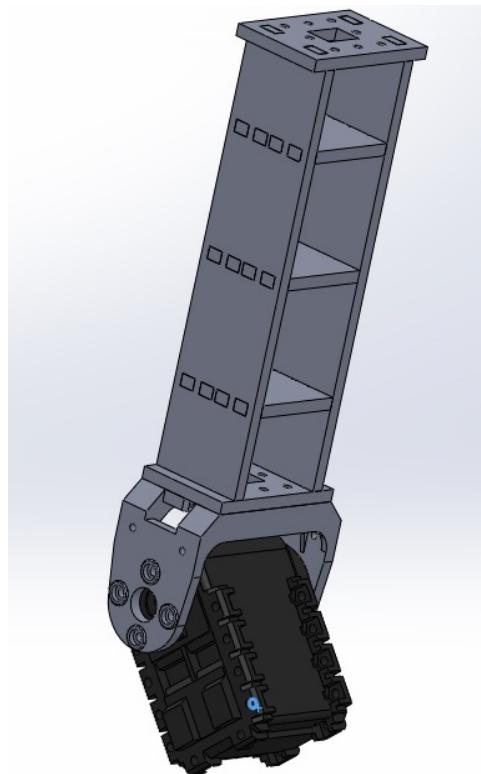


FIGURE 2 – Vue des supports designés sous solidworks

4 Développement du calcul des angles pour situer le robot

Note : tous les calculs ont été faits sous mathematica et le script permettant un calcul direct et inverse rapide a été donné en annexe du code complet.

Quelques remarques Ce calcul ne s'est pas fait en une seule fois. D'abord nous sommes partis sur chacun des servo-moteurs selon un axe différent afin d'avoir les trois directions de l'espace. Cela compliquait le projet inutilement et l'élaboration des formules, d'apparences très simples, ne s'est pas manifesté du premier coup et a été source de tatonnement. Maintenant que le projet est plus mûri, cette simplicité est particulièrement appréciable pour la lisibilité des calculs, et tendrait dans un second temps à être complété en ajoutant d'avantage de servo-moteurs.

Le placement des servo-moteurs Un premier servo-moteur sert à déplacer entièrement le bras selon une direction de l'espace. Ensuite dans la continuité de l'idée de bloquer une balle venant d'un lanceur, il fallait que l'organe terminal soit face au lanceur et vertical. Ceci intégrant une contrainte sur les angles. Le dernier servo-moteur doit compenser l'angle du précédent.

4.1 Le modèle de Denavit-Hartenberg (D.H.)

Lors des cours de robotique de l'année précédente, j'ai appris à utiliser le modèle de D.H., l'idée générale de cette méthode est de répartir les mouvements de moteurs en deux mouvements simples : liaison prismatique(élongation d'un bras selon un axe) qui est très peu utilisée et la liaison rotatoire(rotation d'un axe par rapport à l'autre) bien plus utilisée, à relier à l'être humain qui n'a que ces liaisons par exemple. Ainsi ces mouvements peuvent être paramétrés par des matrices, c'est très pratique, une composition d'angle revient à faire un produit de matrice. Dans chaque matrice on projette le repère du servo-moteur N dans le repère du servo-moteur N-1.

Le travail du roboticien D.H. se dénote en particulier sur l'optimisation de la mise en place des paramètres de servo-moteurs dans les matrices, avec des choix de repères bien précis.

Vous pouvez ici voir comment on place le repère pour chacune des articulations. Par liaison on entendra ici le prolongement des servo-moteurs ou l'axe qui le traverse.

Nous avons pour O_{i+1} : le pied de la perpendiculaire commune aux liaisons (axes) l_i et l_{i+1} . Quand il y a ambiguïté, choisir la solution la plus simple(symétrique).

Pour x_{i+1} le vecteur unitaire de la perpendiculaire commune aux liaisons l_i et l_{i+1} . Quand il y a ambiguïté, choisir la solution la plus simple et la plus physique.

Pour z_{i+1} le vecteur unitaire de la liaison l_{i+1} . y_{i+1} n'est pas placé sauf sur les premiers et derniers repères.

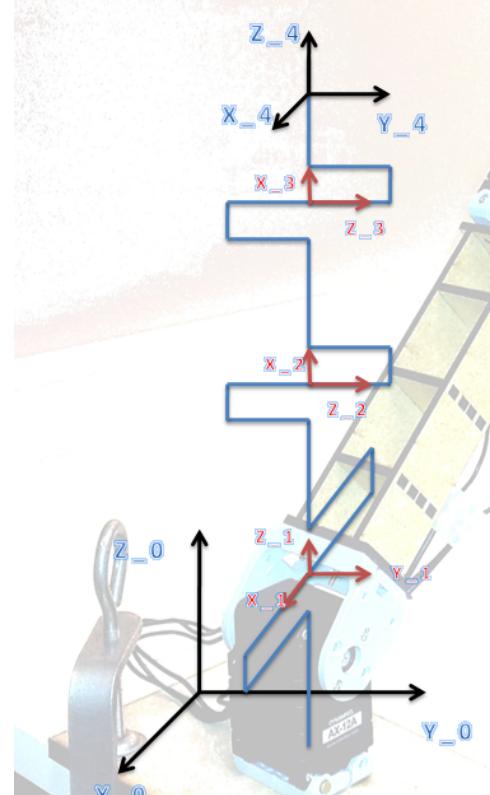


FIGURE 3 – Placement de nos repères

Coefficient/liaison	1→2	2→3	3→4
σ	0	0	0
a_i	17.5	17.5	2.6
r_i	0	0	0
α_i	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$
θ_i	θ_1	θ_2	θ_3
$(q)_i$	$\frac{\pi}{2}$	0	0

TABLE 1 – Coefficient de D.H.

Signification des coefficients :

$\sigma = 0$ si c'est une liaison rototoïde sinon 1 pour une liaison prismatique.

a_i correspond à la longueur de la liaison(autrement dit du bras).

Remarque : dans la suite pour les longueurs de bras on utilisera a, b et c

r_i correspond à la longueur du bras projetée sur un autre axe

α_i angle entre l'axe de la profondeur(côte) des repères consécutifs autour de x_{i+1} . Soit angle entre z_i et z_{i+1} pris autour de x_{i+1} .

θ_i angle entre les axes des abscisses des deux repères consécutifs pris autour de la côte du premier repère, soit angle entre x_i et x_{i+1} autour de z_i

$(q)_i$ angle ou longueur du paramètre vivant dans l'état initial.

Ces coefficients nous permettent de rapidement remplir la matrice de projection d'un repère sur le précédent.

La matrice $T_{1 \rightarrow 2}$:

$$\begin{bmatrix} \cos \theta_1 & 0 & \sin \theta_1 & a \cdot \cos \theta_1 \\ \sin \theta_1 & 0 & -\cos \theta_1 & a \cdot \sin \theta_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice $T_{2 \rightarrow 3}$:

$$\begin{bmatrix} \cos \theta_2 & -\sin \theta_2 & 0 & b \cdot \cos \theta_2 \\ \sin \theta_2 & \cos \theta_2 & 0 & b \cdot \sin \theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice $T_{3 \rightarrow 4}$:

$$\begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 & c \cdot \cos \theta_3 \\ \sin \theta_3 & \cos \theta_3 & 0 & c \cdot \sin \theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Il faut aussi ajouter une matrice pour translater le point d'origine des calculs du centre du premier

servo-moteur au centre de la base. $T_{0 \rightarrow 1} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Pour enfin obtenir la matrice du modèle direct il faut faire le produit $T_{0 \rightarrow 4} = T_{0 \rightarrow 1} \cdot T_{1 \rightarrow 2} \cdot T_{2 \rightarrow 3} \cdot T_{3 \rightarrow 4}$. Nous allons à présent introduire une contrainte qui va nous permettre de simplifier fortement les calculs.

4.2 Une grande simplification

Cette simplification vient du fait que le lanceur va envoyer la balle face au gardien de but. De plus la balle doit être renvoyée de face ce qui amène l'organe terminal à rester vertical en toute circonstance.

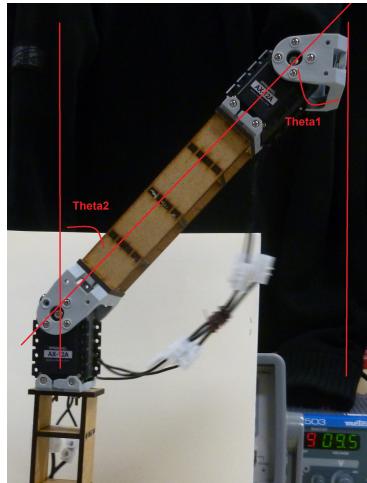


FIGURE 4 – Vue des supports dessinés sous solid-works

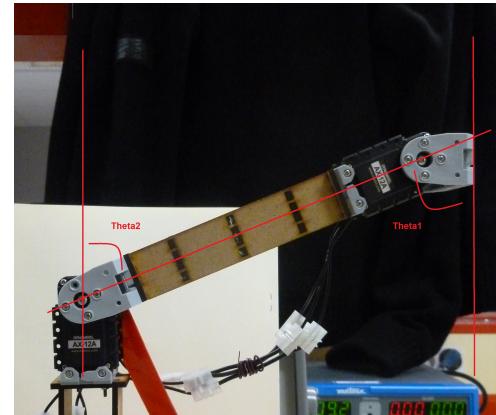


FIGURE 5 – Vue des supports dessinés sous solid-works

Nous pouvons voir sur l'image ci-dessus qu'il existe une relation entre les deux angles car ce sont des angles alternes-internes. Les angles sont orientés, l'angle 0° pour le premier servo-moteur est valable quand il est vertical, pour le second quand la partie mobile est dans le prolongement de la liaison. Nous voulons donc en toute circonstance $\theta_1 + \theta_2 = \frac{\pi}{2}$.

Pour bien le voir regardons les images, sur celle de droite, θ_1 est à 5° et θ_2 est à 85° . Nous avons donc bien $\theta_1 + \theta_2 = \frac{\pi}{2}$.

Une fois que l'on connaît l'angle θ_2 , l'obtention du troisième angle est directe.

La matrice se simplifie alors pour ressembler à cela,

$$T_{0 \rightarrow 4} = \begin{bmatrix} 1 & 0 & 0 & c + b \cdot \sin \theta_2 \\ 0 & -\cos \theta_1 & \sin \theta_1 & \cos \theta_1 \cdot (a + b \cdot \cos \theta_2) \\ 0 & -\sin \theta_1 & -\cos \theta_1 & 4 + (a + b \cos \theta_2) \cdot \sin \theta_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Les équations du modèle direct

Grâce au modèle nous avons les coordonnées de l'organe terminal à partir des angles, ces coordonnées se situent parmi les trois premières lignes de la dernière colonne de la matrice $T_{0 \rightarrow 4}$.

$$(S) \begin{cases} x = c + b \sin \theta_2 \\ y = \cos \theta_1 \cdot (a + b \cdot \cos \theta_2) \\ z = 4 + \sin \theta_1 \cdot (a + b \cdot \cos \theta_2) \end{cases}$$

4.3 Le modèle inverse

Le modèle inverse permet cette fois, à partir des coordonnées de l'organe terminal, c'est à dire l'endroit où se situe l'extrémité du robot, de calculer les angles des servo-moteurs. La méthode est donc tirée en grande partie de son nom, il faut inverser des matrices.

La position à atteindre est écrite dans une matrice de paramètre que l'on appelle U_0 . Cette matrice est constituée pour ses trois premières colonnes des projections du repère de l'organe terminal dans le repère de base, néanmoins nous ne tiendrons pas compte de cela car la position seule et non l'orientation de l'organe terminal nous importe, nous mettrons alors seulement la matrice identité. De plus il est notable que ces projections ne changent pas car comme dit précédemment nous voulons que l'organe terminal reste vertical et face au lanceur. Le repère reste donc inchangé. La dernière colonne est constituée des positions x, y et z à atteindre. Nous obtenons alors la matrice suivante :

$$T_{0 \rightarrow 4} param = \begin{bmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & e \\ 0 & 0 & 1 & f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Puis nous égalisons cette matrice paramètre, à la matrice du modèle direct $T_{0 \rightarrow 4}$:

$$T_{0 \rightarrow 4} param = \begin{bmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & e \\ 0 & 0 & 1 & f \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & c + b \cdot \sin \theta_2 \\ 0 & -\cos \theta_1 & \sin \theta_1 & \cos \theta_1 \cdot (a + b \cdot \cos \theta_2) \\ 0 & -\sin \theta_1 & -\cos \theta_1 & 4 + (a + b \cos \theta_2) \cdot \sin \theta_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_{0 \rightarrow 4}$$

Nous voulons de là obtenir (S_1)

$$\begin{cases} \theta_1 = f(d, e, f) \\ \theta_2 = f(d, e, f) \\ \theta_3 = f(\theta_2) \text{ cf. simplification} \end{cases}$$

avec d, e et f qui sont les coordonnées à atteindre et qui sont imposées par l'utilisateur.

Il faut faire le produit à gauche pour atteindre notre objectif :

$T_{1 \rightarrow 0} \cdot U_0 = T_{1 \rightarrow 2}(\theta_1) \cdot T_{2 \rightarrow 3}(\theta_2) \cdot T_{3 \rightarrow 4}(\theta_3)$ avec $T_{1 \rightarrow 0}$ qui est la matrice inverse de $T_{0 \rightarrow 1}$
puis $T_{2 \rightarrow 1} \cdot T_{1 \rightarrow 0} \cdot U_0(\theta_1) = T_{2 \rightarrow 3}(\theta_2) \cdot T_{3 \rightarrow 4}$

de cette expression nous tirons une relation entre θ_1 et θ_2 . Or en faisant le calcul nous avons remarqué que le calcul de θ_2 était directe, nous obtenons alors ensuite θ_1 et θ_3 .

Les équations du modèle inverse

Nous obtenons alors les équations suivantes : (S_2)

$$\begin{cases} \sin \theta_2 = \frac{d-c}{b} \\ \tan \theta_1 = \frac{f-4}{e} \\ \theta_3 = \frac{\pi}{2} - \theta_2 \end{cases}$$

4.4 Volume de travail des servo-moteurs

Nous avons fait un script sous matlab qui permet de calculer et afficher le volume de travail de l'organe terminal. Le script est donné avec le reste du code.

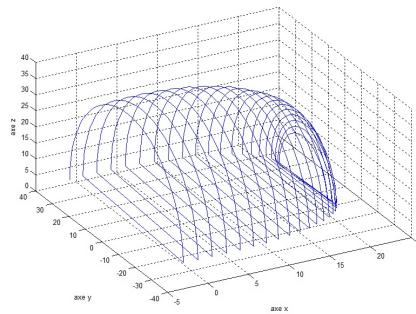


FIGURE 6 – Surface atteinte par l'organe terminal

Nous remarquons que l'organe terminal en se déplaçant forme une surface de travail, il ne peut pas atteindre une position de plusieurs façons.

5 Présentation des servo-moteurs AX-12

5.1 Agencement de la mémoire

Les AX12 sont des servomoteurs numériques. Ce sont des moteurs 'intelligents' asservis en position (angle) comme il en existe beaucoup, cependant ils sont numériques et à ce titre possèdent une mémoire ROM et RAM.

Cela change la façon de communiquer avec ce type de moteur, contrairement à un servomoteur «analogique» contrôlé en PWM(Pulse Width Modulation), les AX12 communiquent avec l'extérieur grâce au protocole de communication USART. Nous allons inscrire en mémoire des valeurs qui permettront le contrôle de l'AX12 : vitesse, position... Le mappage de la mémoire est décrit dans la datasheet donnée par le fabricant, vous pouvez en voir un extrait ici..

Address	Item	Access	Initial Value
0(0X00)	Model Number(L)	RD	12(0x0C)
1(0X01)	Model Number(H)	RD	0(0x00)
2(0X02)	Version of Firmware	RD	?
3(0X03)	ID	RD,WR	1(0x01)
4(0X04)	Baud Rate	RD,WR	1(0x01)

FIGURE 7 – description partielle des registres

Chaque case mémoire, repérée par son adresse, est définie par une valeur particulière. Ces cases mémoires définissent une donnée précise de l'AX12 (identifiant, position, vitesse, baudrate, LED allumée/éteinte, température...), elles sont sur 8 bits et forment un registre. Il faut noter que certaines données de ce servomoteur prennent plus de place que d'autres et sont donc, non pas sur un seul emplacement mémoire, mais sur 2 emplacements consécutifs ce qui permet d'avoir des valeurs supérieures à 8 bits. En théorie l'utilisation de 2 registres de 8 bits permet de manipuler des valeurs de 16 bits, cependant ici seuls 12 bits suffiront (0 à 1024 en décimal). Par exemple, sur cette image, la position est définie sur 2 registres alors qu'en orange le baudrate n'est défini que sur 1 registre.

4(0X04)	Baud Rate	RD,WR	1(0x01)
37(0X25)	Present Position(H)	RD	?
38(0X26)	Present Speed(L)	RD	?

FIGURE 8 – description des registres

Liaison entre servo-moteurs

Il est possible de les connecter les uns aux autres en série et n'envoyer qu'un bus de données à travers l'ensemble des AX12.

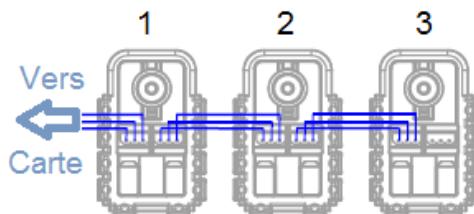


FIGURE 9 – liaison entre servo-moteurs

Chaque AX12 possède en mémoire un identifiant qui caractérise le servomoteur. Ainsi, lors d'un envoi de bus de données (soit plusieurs paquets de données), chaque AX12 recevra ses propres instructions liées à son identifiant.

5.2 Caractéristiques techniques

Poids	55g	
Tension d'entrée	7 à 10V (tension recommandée : 9.6V)	
	7V	10V
Couplage de serrage maximum	12 kg·cm	16.5 kg·cm
temps pour effectuer un angle de 60°	0.269 s	0.196 s
Précision angulaire	0.35°	
Possibilité angulaire	200°	
Courant maximum	900 mA	
Température acceptable	-5°C +85°C	
Signal de commande	Paquet de données	
Type de protocol	Half-duplex sous USART	
Lien(Physique)	TTL Level Multi Drop	
Identifiant ID	254 ID(0 253)	
Vitesse de communication	7453 bps - 1Mbps	

TABLE 2 – Caractéristiques diverses

5.3 Présentation du protocole de communication : USART

USART est le nom générique pour Universal Asynchronous Receiver Transmitter. C'est un module électronique permettant de relier l'ordinateur au port série. L'ordinateur va envoyer les données en parallèle (autant de fils que de bits de données), le module USART va permettre d'envoyer ces données sur le même fil.

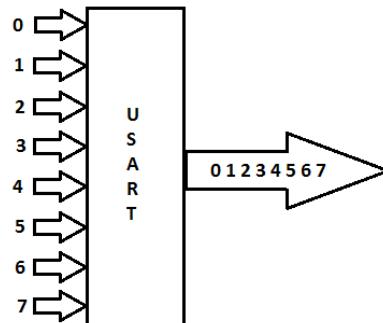


FIGURE 10 – Transposition parallèle/série des données par l'USART

Généralement une trame USART est constituée d'un bit de départ (start), puis des bits de données (8 bits de façon classique = 1 octet), d'un bit de parité éventuellement, et enfin d'un bit de stop.

Bit number	1	2	3	4	5	6	7	8	9	10	11
	Start bit	5-8 data bits							Parity bit	Stop bit	
	Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Parity	Stop

FIGURE 11 – Une trame USART

Ces données sont envoyées à une certaine vitesse, c'est la valeur définie par le baudrate. Le baud est le nombre de symboles envoyés par seconde. Un USART classique (entre 2 cartes par exemple) communique à une vitesse de 9600 baud (soit 9600 symboles par seconde). Afin de faciliter la communication entre les différents périphériques certaines valeurs ont été définies comme standards, en voici quelques unes : 110, 1200, 9600, 38400, 115200, 460800, 921600, 3686400.

L'USART présente 2 sous modules sur une carte : le TX et le RX. Le module TX est celui qui permet l'envoi de données (Transmitter) et le RX celui qui permet leur réception (Receiver). Il y a 2 configurations

qui en découlent : l'une dite Full-duplex (un fil pour TX et un fil pour RX) et l'autre dite Halfduplex (un seul fil pour TX et RX).

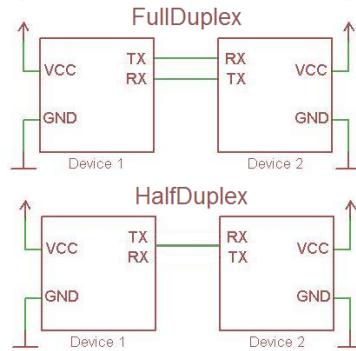


FIGURE 12 – Half-duplex vs Full-duplex

5.4 La communication côté AX-12

Les AX12 communiquent en USART avec les caractéristiques suivantes : 1 bit de stop, pas de parité, 8 bits de données (octet par octet), communication en halfduplex, avec un baudrate de 1Mega (par défaut). En effet un AX12 ne comporte que 3 connexion : une pour l'alimentation, la 2nde pour la masse et enfin la dernière pour la communication, il est donc impossible de communiquer en Full-duplex dans ce cas.

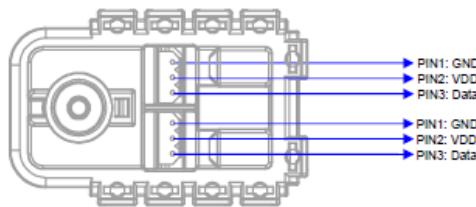


FIGURE 13 – Les ports d'entrée de l'USART

6 La carte STM32F100RB

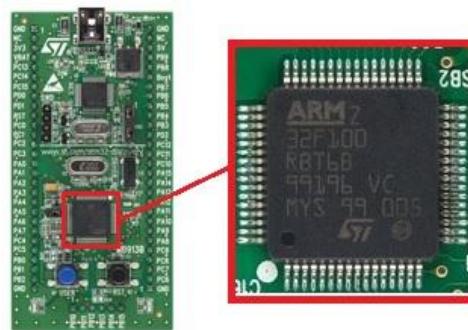


FIGURE 14 – La carte STM32F100RB et son microprocesseur

La STM32F100RB est une carte électronique développée et distribuée par STMicroelectronics. Elle comporte des quartz afin de générer des fréquences utilisables, des ports afin de communiquer avec l'extérieur et surtout un microprocesseur programmable : le Cortex M3 de la société ARM. Ce dernier constitue le cœur de notre carte, il est formé d'un ensemble de modules configurables par des registres. Parmi ces modules nous trouvons : les entrées/sorties (GPIO), la communication USART, les compteurs (TIMER), les convertisseurs analogique-numérique (CAN) et bien d'autres.

6.1 Petit rappel de code bit à bit

Le langage C permet de déplacer des bits dans les registres à l'aide principalement de 2 opérateurs :
Le décalage d'un bit de gauche à droite : «

Le décalage d'un bit de droite à gauche : »

Il permet également d'effectuer des opérations logiques entre bits :

Le NON logique : ~

Le OU logique : |

Le ET logique : &

Quand on combine ces différents éléments on peut alors manipuler à volonté les registres.

Exemples d'utilisation sur un registre 32 bits nommé REG->CONF :

REG->CONF															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

FIGURE 15 – Registre d'exemple

Écriture dans le registre d'un 1 en 4ème position : $\text{REG} \rightarrow \text{CONF} |= (1 \ll 4)$;
 Écriture dans le registre d'un 1 en 4ème position (version 2) : $\text{REG} \rightarrow \text{CONF} |= 0x00000010$;
 Écriture dans le registre d'un 0 en 26ème position : $\text{REG} \rightarrow \text{CONF} \&= (1 \ll 26)$;
 Attention ici , il est essentiel d'écrire de cette manière, la commande $\text{REG} \rightarrow \text{CONF} \&= (0 \ll 26)$; ne mettra pas que le bit 26 à 0 mais tout le registre !
 Écriture dans le registre d'un 0 en 26ème position (version 2) : $\text{REG} \&= 0x1B111111$;

6.2 Configuration de l'USART sur la carte

6.2.1 Activation de l'horloge

Pour obtenir la configuration nécessaire au bon fonctionnement d'un AX12 il faut aller modifier les registres concernant l'USART ainsi que ceux concernant les ports (GPIO) sur la STM32. Commençons par les registres de l'USART (USART1 pour l'exemple) : Il faut tout d'abord activer l'horloge de l'USART1 : $\text{RCC} \rightarrow \text{APB2ENR} |= 0x00004000$;

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Bit 14 USART1EN: USART1 clock enable Set and cleared by software. 0: USART1 clock disabled 1: USART1 clock enabled															
15	14	13	12	SP	EN										
Res.	USART1EN	Res.	Res.	RW											

FIGURE 16 – Activation de l'horloge

Remarque : Sans fréquence de travail, le module ne fonctionne pas. On désactive ensuite l'USART afin de ne pas générer de problèmes. Il est conseillé (voir spécifié) dans la datasheet de toujours désactiver le module USART quand on effectue une configuration de celui-ci. $\text{USART1} \rightarrow \text{CR1} |= (1 \ll 13)$;

6.2.2 Activation du transmitter/receiver

Il faut ensuite activer le transmitter et le receiver :

$\text{USART1} \rightarrow \text{CR1} |= (1 \ll 2)$;

$\text{USART1} \rightarrow \text{CR1} |= (1 \ll 3)$;

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	UE	M	WAKE	PCE	PS	PPE	TXEIE	TCEIE	RXNEIE	IDLEIE	TE	RE	RWU	BSK	
	<small>rw</small>														

Bit 13 **UE**: USART enable
This bit is set and cleared by software.
0: USART prescaler and outputs disabled
1: USART enabled

Bit 3 **TE**: Transmitter enable
This bit enables the transmitter. It is set and cleared by software.
0: Transmitter is disabled
1: Transmitter is enabled
Note: 1: During transmission, a '0' pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word, except in smartcard mode.
2: When TE is set there is a 1 bit-time delay before the transmission starts.

Bit 2 **RE**: Receiver enable
This bit enables the receiver. It is set and cleared by software.
0: Receiver is disabled
1: Receiver is enabled and begins searching for a start bit

FIGURE 17 – Activer la transmission et la réception

Si ces registres ne sont pas activés, pas de communication possible.

6.2.3 Activation de l'half-duplex

La STM32 est par défaut en configuration Full-duplex. Pour changer cette configuration et mettre du Halfduplex il faut aller activer le bit de sélection halfduplex prévu à cet effet : USART1→CR3 |= (1<<3) ;

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13													
Reserved	Bit 3 HDSEL: Half-duplex selection Selection of Single-wire Half-duplex mode														
1 4 3 2 1 0															
EN	NAV	HDSEL	IRLP	IREN	EIE										
<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>										

FIGURE 18 – Activation du half-duplex

6.2.4 Activation du baudrate

Dernier registre à régler, celui qui définit le baudrate. Plus haut il est écrit que l'AX12 communique par défaut avec un baudrate de 1Mega, cependant nous avons préféré limiter la vitesse à 500 000 baud , on va donc devoir abaisser le baudrate de l'AX12. (cette manipulation a été réalisé avec l'arduino pour des raisons de simplicité et de rapidité). Notre choix n'est pas anodin, car il permet d'obtenir une valeur de registre extrêmement simple. Le registre de baudrate est donc à régler sur 500k. Le calcul pour trouver la valeur du registre est assez simple, il faut diviser la valeur de la HSI (High Speed Internal clock,8MHz pour la STM32F100) par le baudrate voulu . On obtient une valeur décimale de 16 ce qui donne en hexadécimal : USART1→BRR=0x0010 ;

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]															
<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>	<small>rw</small>

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV

These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV

These 4 bits define the fraction of the USART Divider (USARTDIV)

FIGURE 19 – activation du baudrate

6.3 Description de la trame USART

6.3.1 Configuration du bit de stop

Réglons à présent le bit de stop à 1 :

USART1→CR2 &= (1<<12) ;

USART1→CR2 &= (1<<13) ;



FIGURE 20 – Configuration sur le registre du bit de stop

6.3.2 Configuration de la taille du paquet

De même, réglons la taille du paquet de données situé entre le bit de stop et celui de parité :
USART1→CR1 &= (1<12) ;

6.3.3 Configuration de la parité

La non parité est imposée par les AX12 :
USART1→CR1 &= (1<10) ;

Remarque : ces 4 dernières désactivations de bits ne sont pas obligatoires car ce sont les valeurs par défaut des registres.

6.3.4 Activation de l'USART

Il ne reste plus qu'à activer l'USART :
USART1→CR1 |= (1<13) ;

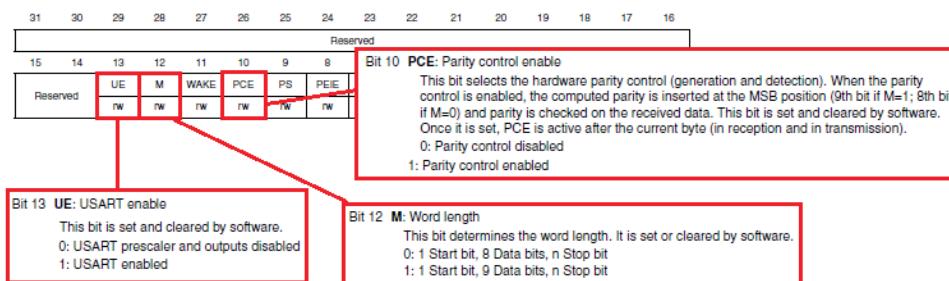


FIGURE 21 – Registre à configurer pour activer l'USART

L'USART est à présent configuré pour un fonctionnement adéquate avec l'AX12.

6.4 Initialisation des GPIO de la carte

6.4.1 Activation des horloges

Les ports de la STM32 ont un fonctionnement principal et un fonctionnement alternatif. L'USART fait partie des fonctions alternatives de la carte, il va donc falloir configurer les ports comme tels. Le mappage réalisé sur la STM32F10x a placé l'USART1 sur les ports A9 (TX) et A10 (RX).

Pin number (P2)	Pin number (chip)	Pin name	Type	Main function	Alternate function	Remap
7	42	PA9	I/O	Port A9	USART1_TX / TIM1_CH2	TIM15_BKIN
8	43	PA10	I/O	Port A10	USART1_RX / TIM1_CH3	TIM17_BKIN

FIGURE 22 – Les ports à configurer

Cependant l'AX12 impose une configuration Halfduplex ce qui donne :
Activation de l'horloge du port A :
RCC→APB2ENR |= 0x00000004 ;
Activation de l'horloge de la fonction alternative :
RCC→APB2ENR |= 0x00000001 ;

Bit 0 AFIOEN: Alternate function I/O clock enable
Set and cleared by software.
0: Alternate Function I/O clock disabled
1:Alternate Function I/O clock enabled

Bit 2 IOPAEN: I/O port A clock enable
Set and cleared by software.
0: I/O port A clock disabled
1:I/O port A clock enabled

FIGURE 23 – Activation de l'horloge alternative

6.4.2 Activation des modes

Activation du port A9 en alternate function opendrain à 10 MHz : GPIOA→CRH |= 0x0A000000 ;

CNFy[1:0]: Port x configuration bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to [Table 16: Port bit configuration table on page 101](#).

In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved

In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

MODEy[1:0]: Port x mode bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to [Table 16: Port bit configuration table on page 101](#).

00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

FIGURE 24 – Configuration de la fréquence de l'alternate function

6.5 Envoi des données

L'USART ainsi que les ports de la carte sont maintenant configurés, il ne reste plus qu'à transmettre les données. Comment faire ? On va utiliser 2 registres de l'USART1 qui sont les registres d'envoi et de réception de données USART1→DR et le registre permettant de vérifier l'état de USART1→DR (savoir s'il est vide ou non) USART1→SR. Cela donne :

Tant que le registre de données est plein on attend

while ((USART1→SR & (1<<7)) != (1<<7));

Une fois vide on écrit dans le Data register

USART1→DR = data;

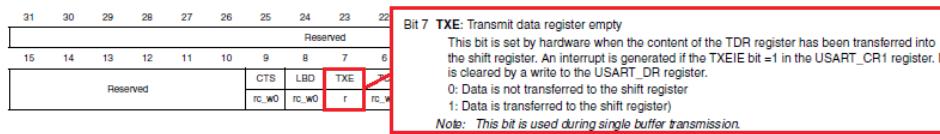


FIGURE 25 – Vérification de l'envoi des données

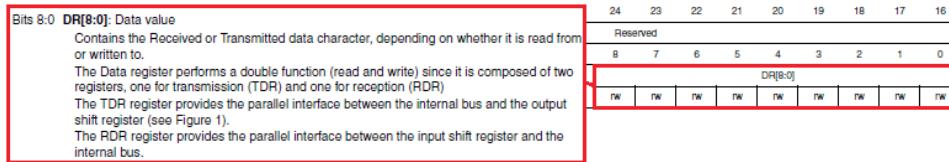


FIGURE 26 – Registre d'envoi des données

Une fois cette donnée inscrite dans le Data Register, elle sera automatiquement envoyée à travers le port configuré auparavant.

Remarque : la configuration pour l'USART 2 et 3 se déroule de la même manière, outre le numéro qui change dans les registres de l'USART (USART2->...), il faut penser à changer également les ports qui ne sont pas les mêmes. (Par exemple USART2 est sur PA2 et PA3)

6.6 Les fonctions principales pour l'USART sur la carte

6.6.1 Initialisation de l'USART

```
/* Configure USART1 */
void usart1_init(Debit p_debit, Direction p_dir)
{
/* AFIO Clock Enable */
RCC→APB2ENR |= 0x00000001 ;
/* USART1 Clock Enable */
RCC→APB2ENR |= 0x00004000 ;
/* PORTB Clock Enable */
RCC→APB2ENR |= 0x00000008 ;

/* Remappage vers PB6 et PB7 */
AFIO→MAPR |= 0x0004 ;

/* Sélection de la direction des pins */
if(p_dir==HALF_DUPLEX)
{
/* HALF DUPLEX */
/* PB6 en Alternate output open-drain */
GPIOB→CRL &= 0xF0FFFFFF ;
GPIOB→CRL |= 0xFD000000 ;
}
else
{
/* FULL DUPLEX */
/* PB6 en Alternate output push-pull et PB7 floating input */
GPIOB→CRL &= 0x00FFFFFF ;
GPIOB→CRL |= 0x49000000 ;
}

/*Usart1 disable*/
USART1→CR1 &= (1<<13) ;
```

```

/* Usart1 Enable, Transmitter Enable, Receiver Enable */
USART1→CR1 |= 0x0000000C;

/* Sélection Half/Full duplex */
if (p_dir == HALF_DUPLEX) { USART1->CR3 |= 0x00000008; }

/* Sélection baudrate */
switch (p_debit)
{
case BAUD_9600 :
USART1→BRR=0x341;
break;

case BAUD_115200 :
USART1→BRR=0x45;
break;

case BAUD_500000 :
USART1→BRR=0x0010;
break;
}
}

```

6.6.2 Activation de l'USART

```

//Activate USARTx
void enable_usart(int usartx)
{
switch(usartx)
{
case USART_1 :
USART1→CR1 |= (1«13);
break;

case USART_2 :
USART2→CR1 |= (1«13);
break;

case USART_3 :
USART3→CR1 |= (1«13);
break;
}
}

```

6.6.3 Envoi des données

```

//Send data (1 byte) with USART
void send_byte(int usartx,unsigned char byte)
{
if (usartx == USART_1)
{
while ((USART1→SR & (1«7)) != (1«7)); // Wait for the TX gets ready to send a byte
USART1→DR = byte;
}

else if (usartx == USART_2)
{
}
}

```

```

while ((USART2→SR & (1<<7)) != (1<<7)); // Wait for the TX gets ready to send a byte
USART2→DR = byte;
}
else if (usartx == USART_3)
{
while ((USART3→SR & (1<<7)) != (1<<7)); // Wait for the TX gets ready to send a byte
USART3→DR = byte;
}
else { }
}

```

6.7 Séquence de communication AX-12

6.7.1 Théorie

Une fois la communication configurées pour les AX12 il faut s'intéresser à ce que l'on va envoyer. Pas question ici d'envoyer n'importe quoi, la datasheet des AX12 impose une certaine séquence de données : 0xFF

0xFF

Identifiant de l'AX12

Longueur du paquet de données

Écriture/Lecture dans la RAM/ROM

Emplacement (Adresse) dans la mémoire où il faut lire ou écrire

Instructions à écrire ou lecture de la donnée

Vérification

Quelques remarques :

Les 2 0xFF de début de séquence initialisent la communication et sont indispensables. La longueur du paquet de données (4ème instruction de la séquence) est égale au nombre d'instructions déclarées qui la suivent. La vérification suit un algorithme particulier donné par le fabricant, il s'agit de faire la somme de tous les paquets de la séquence (exceptés les 2 '0xFF' et la vérification elle-même) et d'inverser logiquement le résultat (les 0 deviennent 1 et inversement). Le résultat ainsi envoyé permet de valider la séquence ou non. (Si la séquence n'est pas validé, les instructions ne sont pas écrites/lues en mémoire)

6.7.2 Exemple

Positionner l'AX12 de façon précise

**Initialisation de la communication

0xFF

0xFF

**identifiant de l'AX12

id = 0x03

**5 instructions déclarées après sa propre déclaration

0x05

**valeur (donnée par datasheet) permettant d'imposer l'écriture dans mémoire AX12

0x03

**la position se situe à l'adresse 0x1e dans la mémoire de l'AX12 (0x1e = 30)

0x1e

**La position est une donnée sur 2 registres (consécutifs en adresse) il faut donc écrire sur ces derniers grâce à un simple décalage de 8 bits (rappel : 1 registre = 8 bits)

position & 0x00ff

(position & 0xff00)»8

**La vérification

vérification = ((id + 5 + 3 + 30 + (position& 0xFF) + ((position& 0xFF00)»8)) % 256)

6.7.3 Fonctions principales

Initialisation de l'AX-12

```
void ax12_init(int usartx)
{
switch(usartx)
{
case USART_1 :
usart1_init(BAUD_ 500000, HALF_DUPLEX);
enable_usart(USART_1);
break;

case USART_2 :
usart2_init(BAUD_ 500000, HALF_DUPLEX);
enable_usart(USART_2);
break;

case USART_3 :
usart3_init(BAUD_ 500000, HALF_DUPLEX);
enable_usart(USART_3);
break;
}
}
```

Positionner un AX-12

```
void ax12_position(int usartx, int id, int pos) {
int checksum1 = ((id + 5 + 3 + 30 + (pos& 0xFF) + ((pos& 0xFF00)»8)) % 256);

if(pos<160){ pos=161;} //Range of position forbidden
else if(pos>871){ pos=871;}
else{ pos=pos;}

send_byte(usartx,0xff); //Initialize communication
send_byte(usartx,0xff);
send_byte(usartx,id); //ID of the AX12
send_byte(usartx,0x05); //Length
send_byte(usartx,0x03); //Write in the RAM of the AX12
send_byte(usartx,0x1e); //Address AX12 in the RAM
send_byte(usartx,(pos & 0x00FF)); //Send the position
send_byte(usartx,((pos & 0xFF00)»8));
send_byte(usartx,checksum1); //Verify

/* Have to send instructions 2 times */
send_byte(usartx,0xff);
send_byte(usartx,0xff);
send_byte(usartx,id);
send_byte(usartx,0x05);
send_byte(usartx,0x03);
send_byte(usartx,0x1e);
send_byte(usartx,(pos & 0x00FF));
send_byte(usartx,((pos & 0xFF00)»8));
send_byte(usartx, checksum1);
}
```

Remarque Dans la dernière fonction, la séquence est envoyée 2 fois au lieu d'une. Il s'est avéré que par expérience une seule séquence n'a jamais permis, avec la STM32F100RB, de faire fonctionner les AX12, cependant en effectuant une redondance du code ça marche ! La cause exacte reste inconnue. Une conséquence non négligeable de ce phénomène va être que le temps d'exécution du code sera fortement entaché. Pour pallier à ce problème il pourrait être intéressant d'utiliser les interruptions et le DMA (Direct Memory Access).

7 Débuter un projet sur Keil

7.1 Prérequis

Posséder une STM32F100RB et avoir téléchargé et installé µVision4 sur son PC
<http://www.keil.com/download/product/> puis MDK-ARM. Remplir le formulaire et le tour est joué.

7.2 Démarrer le projet

Lancer µVision cliquer sur le menu (en haut) Project puis New µVision Project

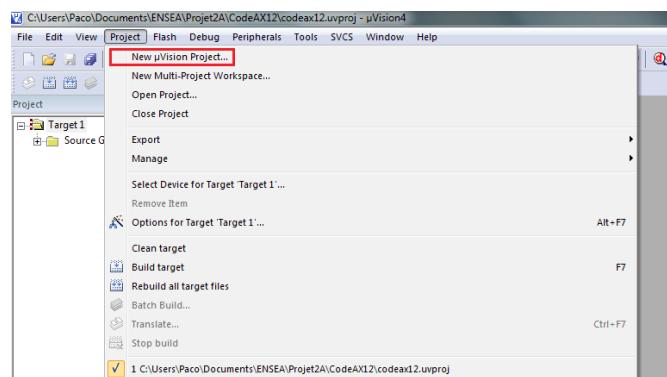


FIGURE 27 – Nouveau projet sous Keil

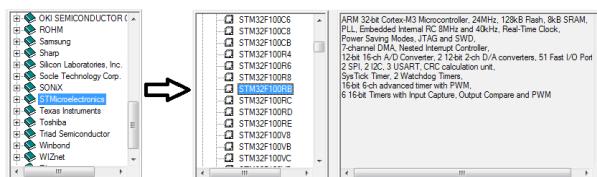


FIGURE 28 – Choix de la carte

Un message apparaît lors du lancement du projet demandant l'ajout du fichier de démarrage ou non de la carte, cliquer OUI.

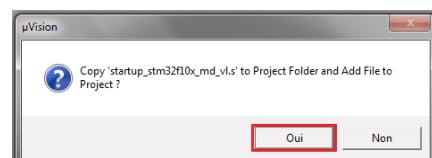


FIGURE 29 – Copier le fichier de startup

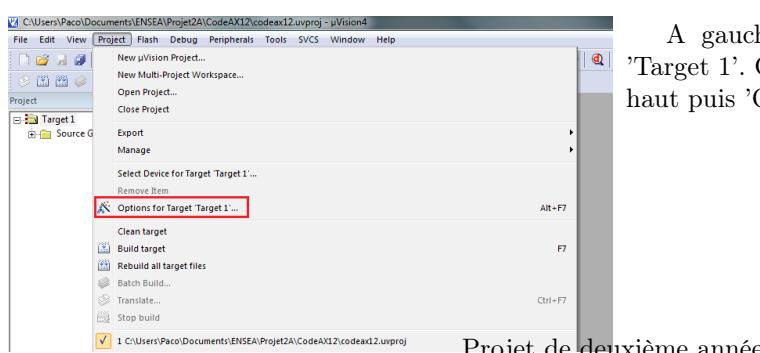


FIGURE 30 – Trouver l'option de débogage

A gauche dans l'onglet 'Project' sélectionner 'Target 1'. Cliquer sur l'onglet Project du menu du haut puis 'Option for Target' (ou faire Alt+F7).

Dans l'onglet C/C++, dans le champ 'Define' de 'Preprocessor Symbols' écrire STM32F10X_LD_VL.

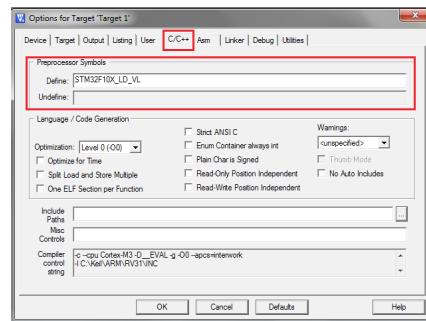


FIGURE 31 – Option de débogage

Dans l'onglet 'Debug', cocher 'Use' (à droite) à la place de 'Use Simulator' (à gauche), puis dans le menu déroulant sélectionner ST-Link (Deprecated Version). Enfin dans 'Settings' cocher SWD.

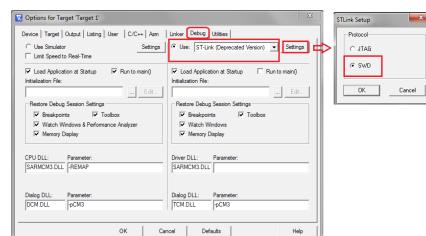


FIGURE 32 – Débogage sur la sonde

Dans l'onglet 'Utilities' cocher Use Target Driver for Flash Programming et sélectionner ST-Link (Deprecated Version) dans le menu déroulant.

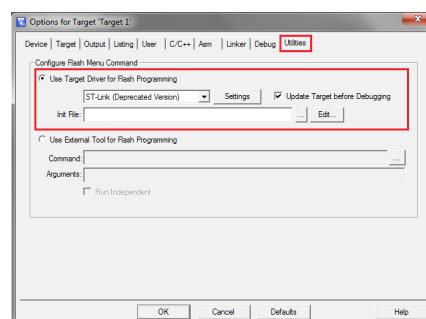


FIGURE 33 – Type de sonde

Cliquer sur 'OK' pour enregistrer les modifications faites dans le menu.

***Ajouter les fichiers au projet : A gauche, faire un clique droit sur 'Source Group 1' puis 'Add files to Source Group 1'. Sélectionner l'ensemble des fichiers de la librairie (.c et .h) puis faire 'Add'.

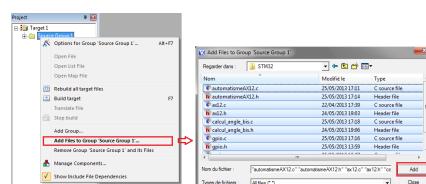


FIGURE 34 – Ajouter ses fichiers au projet

8 La carte Arduino Mega 2560

L'Arduino Mega 2560 est une carte électronique développée par Arduino, projet opensource destiné à promouvoir l'électronique et sa programmation à travers le monde. Facile d'utilisation et soutenue par une très large communauté, les cartes Arduino sont très populaires et permettent de monter des projets de grande envergure sans 's'arracher les cheveux'.

De façon classique, la carte dispose d'une batterie de ports et de modules en tout genre comme l'USART ou les TIMERs et naturellement d'un microprocesseur : l'ATmega 2560 de la société Atmel. (Architecture RISK(Reduced Instruction Set Computer))

Ce dernier, comme pour la STM32F100, constitue le cœur de notre carte et permet de contrôler les différents modules et ports de la carte.



FIGURE 35 – Une carte Arduino de type Mega 2560

8.1 Spécificité du code Arduino

Globalement le code arduino est similaire à du C classique, cependant il comporte quelques particularités, parfois surprenantes. Pour plus d'informations concernant le code sous Arduino nous vous invitons à consulter les tutoriels proposés sur le site officiel : <http://arduino.cc/en/Tutorial/HomePage>

8.2 Pourquoi déboguer les AX-12 ?

Au lancement du projet il était question d'utiliser la carte Arduino afin de contrôler les AX12. En effet, nous avions trouvés une bibliothèque permettant d'écrire et lire dans les registres des servomoteurs. Cette bibliothèque s'inspirait largement du code proposé dans la documentation des AX12 (code pour un microprocesseur Atmega128) et après quelques tests sur nos moteurs s'est avéré être très efficace. Cependant, nous avons décidé de repartir de zéro en écrivant notre propre code sous STM32 comme vu précédemment. Mais nous n'avons pas jeté pour autant cette précieuse bibliothèque et nous l'avons utilisé à des fins de débogage. De façon plus générale, la carte Arduino a permis de tester et déboguer non seulement les moteurs mais également le module USART de la STM32.

8.2.1 Débogage 1 : réinitialisation des valeurs de baud et de l'ID

Nous avons mis au point un code sur arduino qui va tester toutes les possibilités de baudrates (standards) et d'identifiant pour chaque AX12 et pour chaque combinaison tenter d'écrire la valeur de 1Mega, pour le baudrate, et de 1, pour l'identifiant. Une instruction de position lui est envoyée juste après avec les valeurs de baudrate et d'identifiant précédentes, ainsi si l'AX12 bouge c'est qu'il est réinitialisé.

Remarque : grâce à la propriété des branchements série des AX12 on peut déboguer plus d'un AX12 à la fois !

En pratique cela se traduit par une double boucle : la première pour le baudrate et la seconde pour l'identifiant. De plus, pour être sûr que chaque instruction est bien prise en compte, on incorpore des délais entre elles. Il faut remarquer que déboguer du matériel peut être long voir très long dans ce cas. En effet, on teste des valeurs du baudrate de plus en plus faibles ce qui fatallement baisse la vitesse de communication entre la carte et les moteurs et donc allonge de plus en plus le temps d'exécution du code. Si on rajoute à cela la double boucle et les délais on ne parle plus de minutes d'exécution mais d'heures. On estime qu'il faut au maximum 5h pour déboguer un AX12, si ce dernier possède un baudrate faible.

Le code est en annexe du code complet de l'Arduino.

8.2.2 Débogage 2 : Modification des valeurs du baudrate et de l'ID

Après avoir réinitialisé l'ensemble des AX12, il faut pouvoir les différencier par leurs identifiants et de plus abaisser leurs baudrates respectifs car notre STM32 communique à 500kbps et non pas à 1Mbps.

Nous avons donc mis au point un code simple qui utilise les fonctions de la bibliothèque AX-12 pour arduino permettant d'écrire en mémoire l'identifiant voulu ainsi que le baudrate. Les étapes à réaliser sont notées au début du code.

Le code est en annexe du code complet de l'Arduino.

9 Utilisation du projet sous Arduino

9.1 Prérequis

Posséder une Arduino mega 2560. Avoir téléchargé Arduino sur son PC à cette adresse : <http://arduino.cc/en/Main/Software>.

9.2 Démarrer le projet

Lancer 'arduino.exe' puis connecter la carte en USB au PC.

Dans le menu du haut, cliquer sur 'Outils' puis 'Type de carte' et sélectionner 'Arduino Mega 2560 or Mega ADK'.

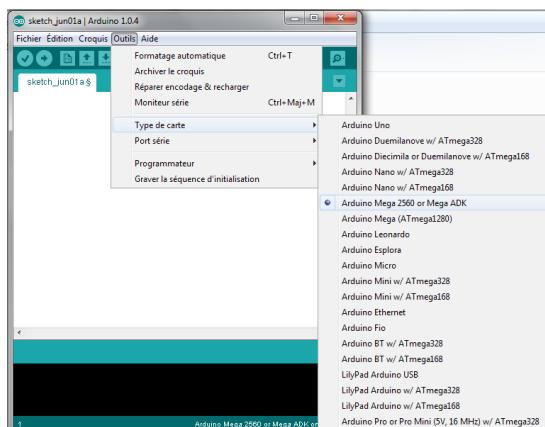


FIGURE 36 – Choisir sa carte arduino

Toujours dans le menu du haut dans 'Outils', cliquer sur 'Port Série' et sélectionner (si ce n'est pas fait) le 'COM' en question.

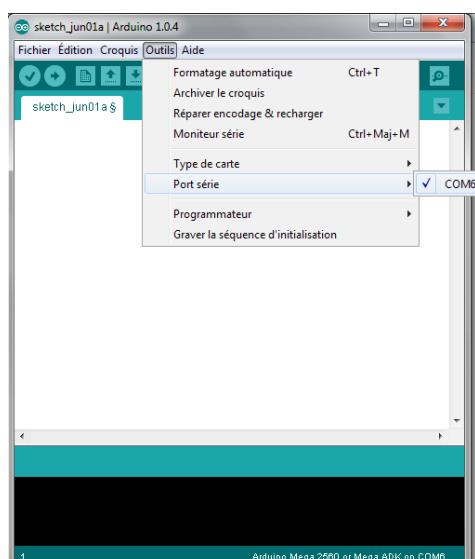


FIGURE 37 – Choix du port de sortie

9.3 Importer des bibliothéques

Sélectionner le dossier 'Bioloid' se trouvant dans le dossier librariesAX12/Arduino fourni. Copier-coller ce dossier dans le dossier 'libraries' se trouvant près de votre exécutable 'arduino.exe'.

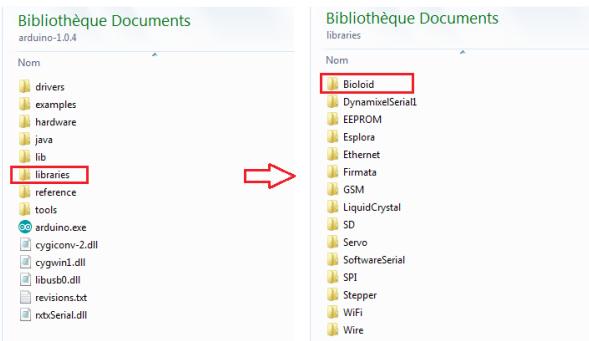


FIGURE 38 – Où importer ses bibliothéques ?

Pour tester le bon fonctionnement de la manipulation précédente on va tenter d'importer les bibliothèques. Dans le menu en haut cliquer sur 'Croquis' puis 'Importer bibliothèque...' et sélectionner 'Bioloid'.

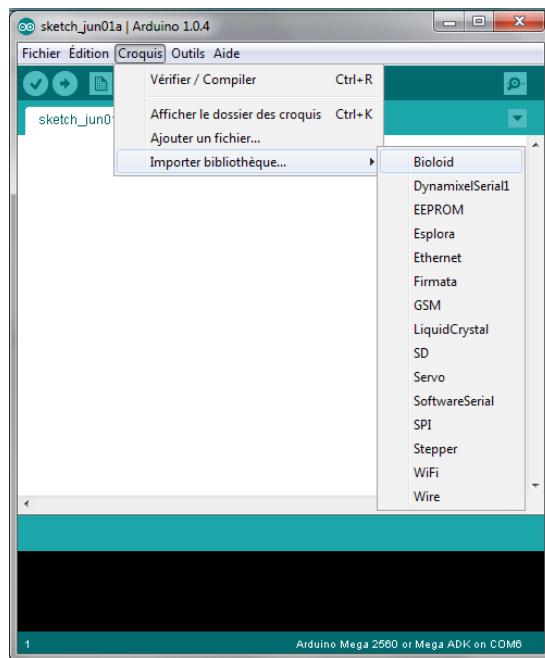


FIGURE 39 – Importer ses bibliothéques dans le code

Si tout se passe bien, 2 lignes de code devraient apparaître.

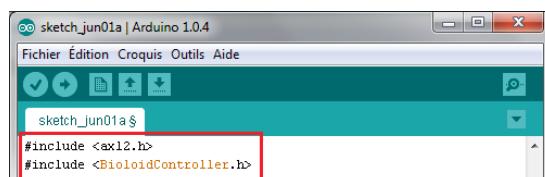


FIGURE 40 – Comment vérifier que les bibliothéques sont bien présentes

9.4 importation du code

Les codes de débogages sont situés dans le dossier librairiesAX12/Arduino/Codes. Leurs extensions sont des '.ino', extension propre à Arduino. Lancer ces fichiers directement avec Arduino (association d'extension de fichier) ou bien importer depuis le menu du logiciel : 'Fichier' puis 'Ouvrir...' et sélectionner le fichier.

10 Fonction d'automatisation des actions sur le bras

Description

Rappel : le but du modèle inverse de Denavit-Hartenberg est de pouvoir calculer les positions à imposer à nos moteurs afin de placer l'organe terminal du bras à une position donnée dans l'espace (x,y,z).

La fonction précédemment développée sur STM32 'ax12_position(int usartx, int id, int pos)' et les fonctions de calcul de position sont pour le moment séparées. En les agençant de façon adéquate on crée une fonction qui prend comme paramètres les coordonnées de l'organe terminal, qui calcule à partir de ces arguments les angles à donner aux servomoteurs puis transmet l'information aux AX12. Ainsi on automatise le processus de calcul et la transmission des angles et on le réduit à une unique fonction. Pour modéliser les coordonnées de l'organe terminal on va utiliser 3 variables globales représentant X, Y et Z. Bien évidemment ces variables évoluent selon la position de cet organe et vont donc prendre des valeurs particulières.

La seconde automatisation concerne la manette (présentée un peu plus bas). Chacun des 6 boutons va définir une direction particulière et donc une action spécifique à émettre selon l'appui. On utilise pour cela la fonction d'automatisation précédente dans laquelle on va rentrer les variables globales de position de l'organe terminal. Selon la direction sélectionnée on incrémentera/décrémentera l'une des variables.

Si on se déplace vers la droite, on aura automatisation(X,Y+1,Z). Si en revanche on se déplace sur la gauche cela donnera automatisation(X,Y-1,Z).

Remarque : un bouton supplémentaire a été ajouté au code. Il s'agit du bouton bleu 'USER' de la STM32F100. Il permet, lors de son appui, de mettre le bras à la verticale et réinitialise les variables globales de ce fait.

Le code est en annexe du code complet de la STM32 pour mettre le bras à la verticale lors d'un reset et un autre pour déplacer le robot sur plusieurs positions.

11 La manette

Il était au départ prévu de créer un système de vision pour le robot, mais étant donné la complexité de ce genre d'automatisation, il n'a pas été possible de la mettre en place. Pour pallier à cela et pouvoir malgré tout bouger le bras de façon semi-automatique, nous avons créé une manette. Elle comporte 6 boutons simples et un bus parallèle de 8 fils permettant de relier chaque bouton à un port d'une carte ainsi que de l'alimenter (3,3V ~ 5V).

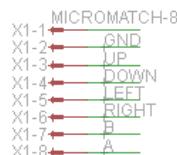


FIGURE 41 – Les entrées/sorties de la manette

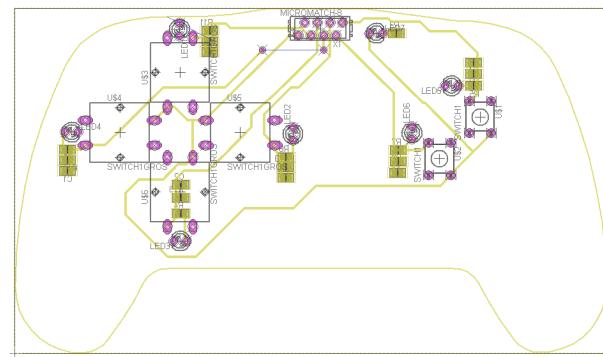


FIGURE 42 – Le PCB de la carte

11.1 Configuration des ports

Chaque port de la carte est configuré afin de pouvoir récupérer le signal, s'il existe, provenant de la manette. Pour simplifier les branchements nous avons présélectionné les ports afin que ceux-ci soient toujours les mêmes pour la manette et proche les uns des autres, de plus nous avons fait en sorte de ne pas entraver les ports pouvant servir à générer l'USART : PA4, PA5, PA6, PA7, PC4, PC5 ont été retenus.

Remarque Nous avons également configuré PA0 qui correspond au bouton bleu 'USER' présent sur la STM32F100RB.

11.2 Les registres d'Input

Ces ports sont configurés en floating input. Le signal réceptionné à l'entrée de ces ports est ensuite transformé en signal créneau interprétable par le microprocesseur grâce à un Trigger de Schmitt (intégré dans la carte). Pour savoir si un signal est présent en entrée (ou non), on utilise le registre IDR (Input Data Register), composé de 15bits (1 pour chaque port), il est à 1 quand un signal est présent sur le port et à 0 le reste du temps. Nous avons créé une fonction spécifique à ce registre qui va renvoyer 1 quand IDR est à 1 et 0 quand IDR est à 0.

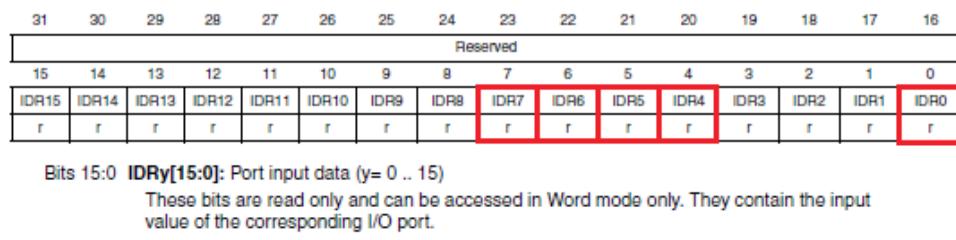


FIGURE 43 – Le registre IDR

L'automatisme présenté plus haut avec la manette a été codé de la manière suivante : on attend l'appui d'un bouton (boucle while) quand un bouton est pressé on sélectionne une action à effectuer suivant le bouton pressé (conditions if..) puis on attend le relâchement du bouton (boucle while de nouveau). Afin d'effectuer cet algorithme de façon permanente (sans avoir à remettre du code dans la carte) on incorpore le tout dans une boucle infinie (while(1)).

11.3 Les fonctions liées à la manette

Le code est en annexe.

11.3.1 Fonction finale d'automatisation

Après avoir configuré la communication avec les servomoteurs, les ports et la manette et après avoir générée l'automatisation des calculs d'angles et liés le tout avec la manette il ne reste plus qu'à écrire le tout dans une seule fonction, fonction qui sera appelée dans le main(). Cette fonction va mettre le bras à la verticale, initialiser les ports puis attendre en permanence l'appui d'un bouton de la manette qui générera un mouvement du bras dans l'espace.

```
void moveARM(int usartx)
{
    resetarm(usartx); //Remet le bras a la verticale
    joystick_button_config(); //Initialisation des ports d'inputs de la manette
    while(1)
    {
        joystick(usartx); //Driver manette/bras
    }
}
```

12 Organisation du code

L'organisation du code est un diagramme mettant en jeu l'ensemble des fonctions codées sous STM32, leurs relations ainsi que leur niveau d'action. Le niveau se scinde en trois parties : bas (LOW), intermédiaire (MEDIUM) et élevé (HIGH). Plus une fonction est bas niveau et plus elle effectue une action simple et proche des registres du microprocesseur, à l'inverse, plus une fonction est élevée et plus elle effectue une action complexe en utilisant d'autres fonctions.

De plus, sur le diagramme, chaque fonction comporte une couleur correspondant au fichier auquel elle est liée. Le code proposé comporte 4 fichiers sources distincts : `gpio.c`; `ax12.c`; `calcul_angle_bis.c`; `automatismeAX12.c`.

Ce diagramme permet de montrer l'importance du codage modulaire et la mise en œuvre pas à pas de celui-ci afin de parvenir à une unique fonction.

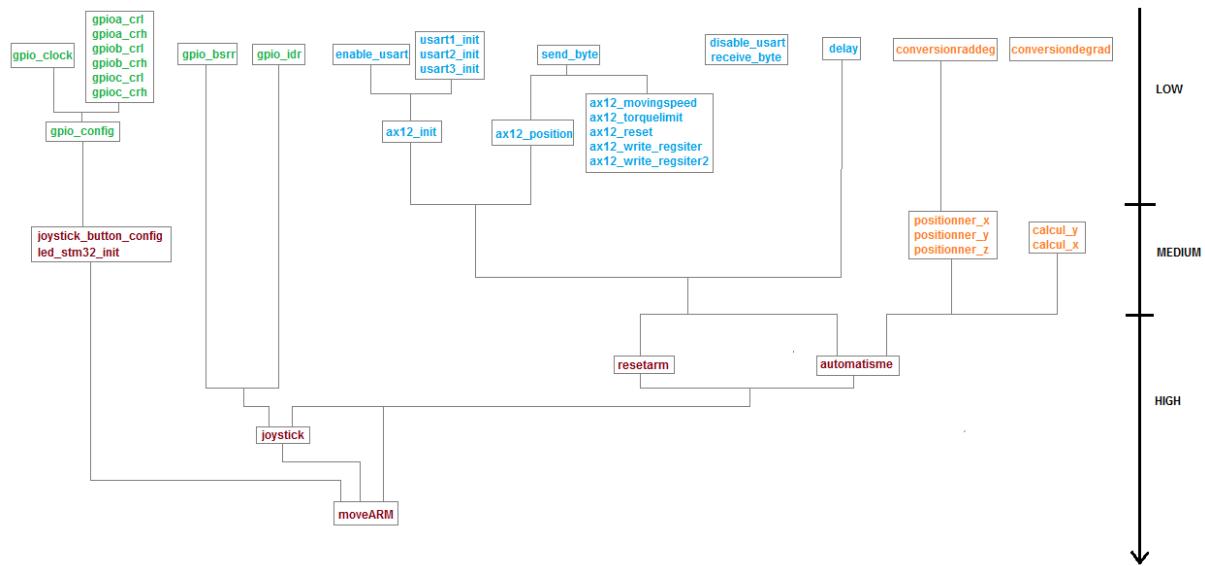


FIGURE 44 – Organisation du code

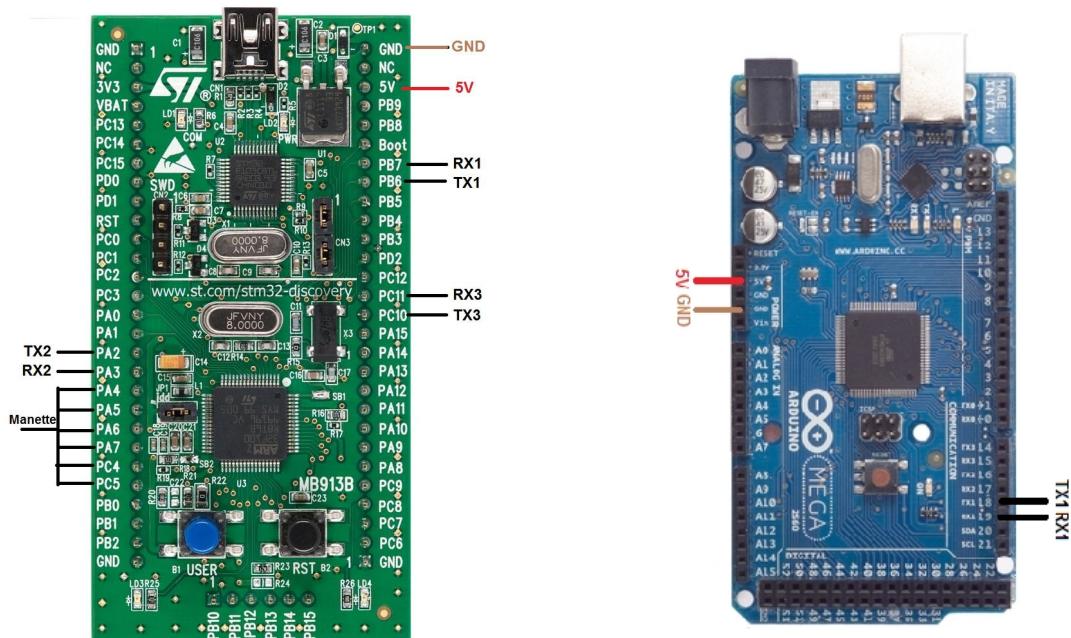
Appendices

Description des registres

Address	Item	Access	Initial Value
0(0X00)	Model Number(L)	RD	12(0x0C)
1(0X01)	Model Number(H)	RD	0(0x00)
2(0X02)	Version of Firmware	RD	?
3(0X03)	ID	RD,WR	1(0x01)
4(0X04)	Baud Rate	RD,WR	1(0x01)
5(0X05)	Return Delay Time	RD,WR	250(0xFA)
6(0X06)	CW Angle Limit(L)	RD,WR	0(0x00)
7(0X07)	CW Angle Limit(H)	RD,WR	0(0x00)
8(0X08)	CCW Angle Limit(L)	RD,WR	255(0xFF)
9(0X09)	CCW Angle Limit(H)	RD,WR	3(0x03)
10(0X0A)	(Reserved)	-	0(0x00)
11(0X0B)	the Highest Limit Temperature	RD,WR	85(0x55)
12(0X0C)	the Lowest Limit Voltage	RD,WR	60(0x3C)
13(0X0D)	the Highest Limit Voltage	RD,WR	190(0xBE)
14(0X0E)	Max Torque(L)	RD,WR	255(0xFF)
15(0X0F)	Max Torque(H)	RD,WR	3(0x03)
16(0X10)	Status Return Level	RD,WR	2(0x02)
17(0X11)	Alarm LED	RD,WR	4(0x04)
18(0X12)	Alarm Shutdown	RD,WR	4(0x04)
19(0X13)	(Reserved)	RD,WR	0(0x00)
20(0X14)	Down Calibration(L)	RD	?
21(0X15)	Down Calibration(H)	RD	?
22(0X16)	Up Calibration(L)	RD	?
23(0X17)	Up Calibration(H)	RD	?
24(0X18)	Torque Enable	RD,WR	0(0x00)
25(0X19)	LED	RD,WR	0(0x00)
26(0X1A)	CW Compliance Margin	RD,WR	0(0x00)
27(0X1B)	CCW Compliance Margin	RD,WR	0(0x00)
28(0X1C)	CW Compliance Slope	RD,WR	32(0x20)
29(0X1D)	CCW Compliance Slope	RD,WR	32(0x20)
30(0X1E)	Goal Position(L)	RD,WR	[Addr36]value
31(0X1F)	Goal Position(H)	RD,WR	[Addr37]value
32(0X20)	Moving Speed(L)	RD,WR	0
33(0X21)	Moving Speed(H)	RD,WR	0
34(0X22)	Torque Limit(L)	RD,WR	[Addr14] value
35(0X23)	Torque Limit(H)	RD,WR	[Addr15] value
36(0X24)	Present Position(L)	RD	?
37(0X25)	Present Position(H)	RD	?
38(0X26)	Present Speed(L)	RD	?
39(0X27)	Present Speed(H)	RD	?
40(0X28)	Present Load(L)	RD	?
41(0X29)	Present Load(H)	RD	?
42(0X2A)	Present Voltage	RD	?
43(0X2B)	Present Temperature	RD	?
44(0X2C)	Registered Instruction	RD,WR	0(0x00)
45(0X2D)	(Reserved)	-	0(0x00)
46(0X2E)	Moving	RD	0(0x00)
47(0X2F)	Lock	RD,WR	0(0x00)
48[0x30]	Punch(L)	RD,WR	32(0x20)
49[0x31]	Punch(H)	RD,WR	0(0x00)

FIGURE 45 – description des registres

Repérage de la connectique sur les cartes



Subtilités de connectiques liées au mode halfduplex : sur l'arduino, TX1 et RX1 sont reliés entre eux ; sur STM32F100 c'est le port où se situe le TX qui fait office de TX/RX, de plus il faut rajouter une résistance de pull-up en sortie de port (on conseille 10kOhms sur du 5V).

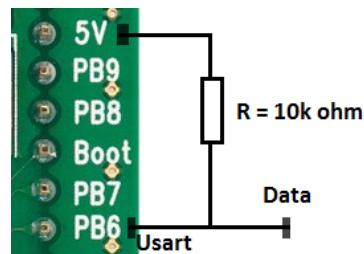


FIGURE 46 – Zoom sur la résistance de pull-up