BoucharD luka, renaud émile

Technique de systèmes ordinés, TSO 2020

247-616-LI , gr.1

Bras robotisé :

**Six axes de liberté**

Travail présenté à

M. Yves Roy

M. Kevin Cotton

M. Stéphane Deschenes

M. François Bouchard

Département de technologie du génie électrique

Cégep Limoilou

Le 5 juin 2020

# Démarrage (1)

## Architecture Matérielle :

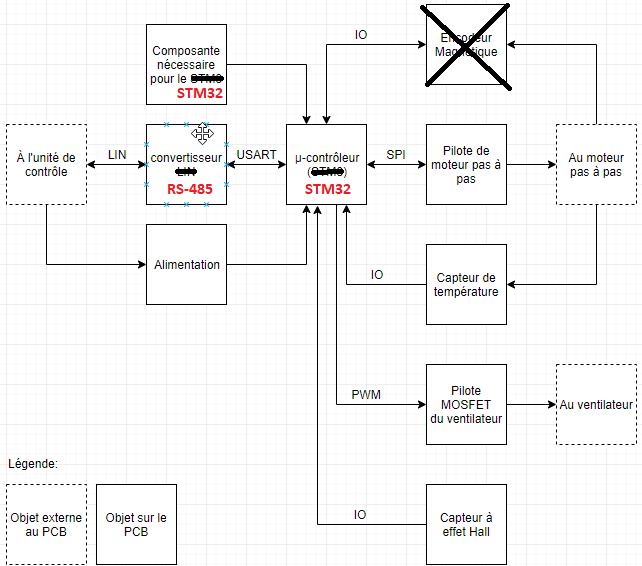
L’architecture matérielle de notre contrôleur de moteur est la plus importante. C’est sur ce PCB qu’on a travaillé la majorité de notre session. Les autres architectures matérielles se trouvent dans le répertoire [Paperwork\Schémas\Architecture Matérielle](../Schémas/Architecture%20Matérielle)

Figure 1: Architecture matérielle contrôleur de moteur

## Planification :

La planification du projet a été refaite lors de la première semaine pour s’adapter à la fin du monde. Malgré les complications que nous avons eues, nous avons été capables de suivre notre planification. Le seul changement que nous avons fait est de commencer l’interface graphique à la semaine 4.

### Semaine 1 et 2 :

* Révision des objectifs
* Planification
* Révision des PCB
* Setup IDE
* Setup gestion fichiers
* Code des pilotes

### Semaine 3 :

* Dessouder les Blue Pills
* Assemblage 2 PCB
* Debug des PCB
* Test des pilotes

### Semaine 4 :

* Debug des PCB
* Test des pilotes

### Semaine 5 :

* Assemblage de tous les PCB (Semaine 3 & 4)
* Montage des PCB sur le bras
* Mécanique du bras final

### Semaine 6, 7 et 8 :

* Programmations-objets

### Semaine 9 :

* Interface GUI python
* Intelligence du bras? (Python / C++)
* Intégration

### Semaine 10 :

* Intégration
* Rapport

## Livrables :

Nos livrables ont changé un peu vers la fin de la session. Comme nous avions pu beaucoup de temps restant, nous avons décidé de mettre un peu plus d’effort sur un logiciel plus complet sur l’ordinateur que de faire fonctionner le BeagleBone. Il était plus pertinent pour notre projet d’avoir un logiciel pour contrôler le bras qu’un BeagleBone qui fonctionne, mais pas de logiciel.

* 6 PCB de pilotes des moteurs.
* ~~1 PCB d’interfaçage des périphériques du BeagleBone Black.~~
* 6 axes du bras robotisé.
* ~~1 interface graphique pour l’écran LCD.~~
* 1 programme de contrôle des moteurs.
* 1 programme de gestion des capteurs.
* 1 programme de gestions des erreurs.
* 1 programme de communication ~~LIN~~ RS485.
* 1 programme de contrôle intégral du bras robotisé.
* 1 schéma des pilotes des moteurs
* ~~1 schéma de l’interfaçage des périphériques du BeagleBone Black.~~
* 1 circuit d’alimentation fonctionnel dans le système.
* 1 interface graphique pour le PC (bonus)
* 1 rapport complet
* Acheter le matériel nécessaire à la mise en marche du bras.

## Microcontrôleur :

Le choix de notre microcontrôleur s’est arrêté sur le STM32F103C8T6. Il s’agit d’un microcontrôleur très populaire de la famille STMicroelectronics. Il est connu, car il a un kit de développement (Bluepill) compatible avec Arduino en plus de son coût relativement faible coût. Nous avons réussi à acheter nos microcontrôleurs pour 4$ le kit complet, contrairement à 7$ pour la chip sur Digikey. Il nous permet d’interfacer tous nos périphériques sans problème. Tant bien, que nous avons une dizaine de GPIO de libre pour de futures améliorations. Le package est standard (Quad Flat Package) et assez gros, cela nous permet, en autre, de le souder facilement.

# Environnement de travail (2)

## Organisation :

Notre structure de projet était d’abord située dans un folder sur Google Drive. L’idée n’était pas mauvaise au départ, puisqu’il n’y avait que de la paperasse à faire. Ainsi, il était plus simple de simplement créer des fichiers .gdoc que de rendre les choses plus complexes dans un projet git. Une fois le projet entamé on a vite réaliser qu’un projet avec de la programmation sur Google drive peut vite devenir chaotique. C’est pourquoi on a migré vers git. La gestion de sources s’est vue beaucoup simplifiée sur le long terme et beaucoup plus sécuritaire. Il est arrivé à plusieurs reprises d’aller sur le git chercher des vieux *commit* de code pour les comparer à nos nouveaux codes. Le répertoire Github est disponible à l’adresse suivante : <https://github.com/mimil2014/6DOF-Robotic-Arm>

## IDE :

Notre choix d’IDE s’est arrêté sur VSCode pour la programmation en C/C++ des stm32. Nous voulions utiliser un environnement «user friendly» et ne pas perdre trop de temps à paramétrer notre environnement de travail. Ainsi, VSCode avec l’ajout de l’extension platform.io permettent de coder rapidement évitant de cette manière de passer trop de temps à faire fonctionner l’environnement et non le code. De plus, avec platform.io, nous pouvons utiliser les librairies Arduino. Dans nos cas, elles n’ont pas beaucoup servi. Elles ont servi seulement pour les pilotes des GPIO.

Pour la programmation de l’interface graphique en python, Visual Studio a été utilisé.

# Conception électronique (3)

## Choix des pièces :

Il y avait beaucoup de pièces à trouver. Nous avons fait plusieurs itérations du projet selon les pièces que nous trouvions. Pour les grosses composantes, il s’agissait de trouver ce qui coûtait le moins cher, mais qui allait être de qualité. Donc, pour le TMC2130 et le STM32, il n’y avait pas vraiment de compromis à faire. Les composantes passives ont été choisies en fonction des datasheets des composantes principales (ce qui était recommandé). Il y a plusieurs composantes qu’on a simplement pu récupérer de ce qu’on avait déjà. Nous possédions déjà en autre des MAX485 et l’alimentation. Ceci est le fichier pour le choix des composantes et non le BOM! Ce fichier ce trouve dans [Paperwork\Listes](../Listes) sous le nom de Choix pièces.

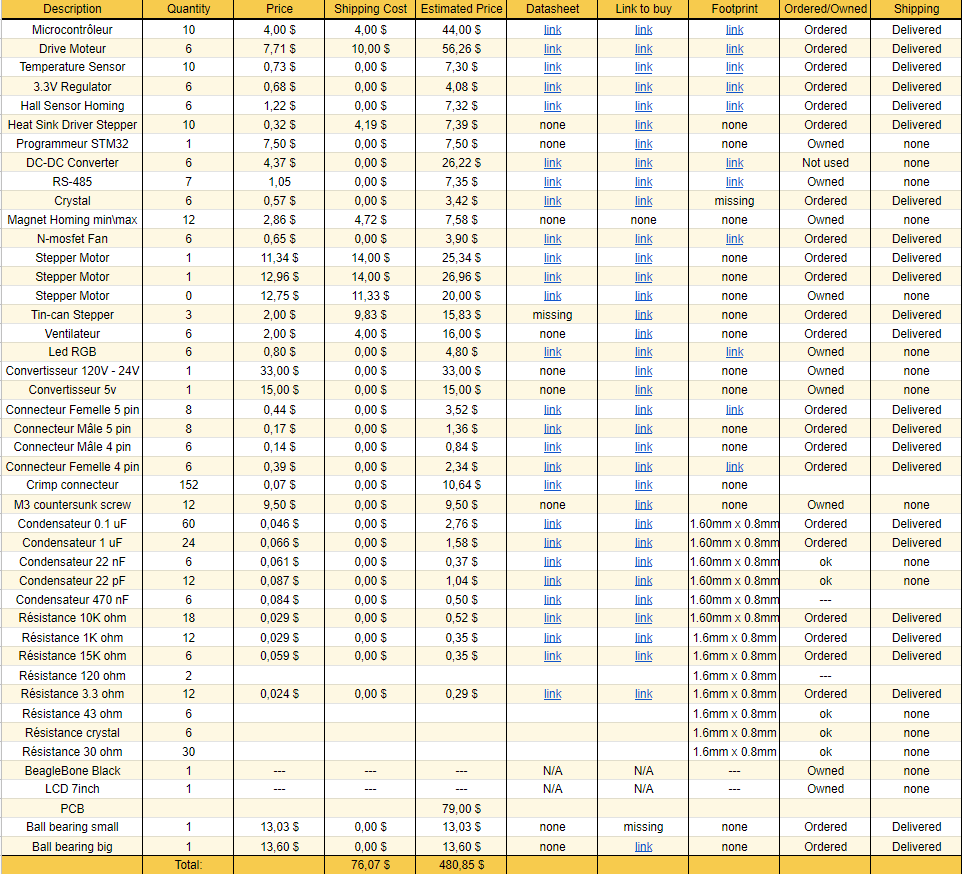


Figure 2: Choix des pièces

## Architecture logicielle :

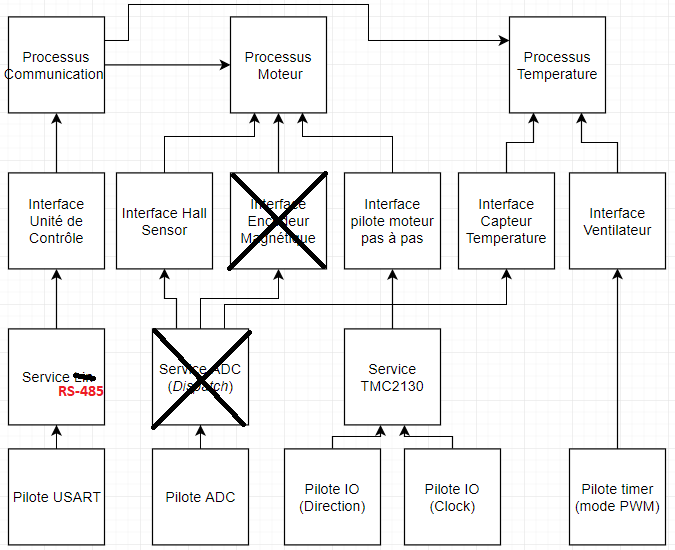
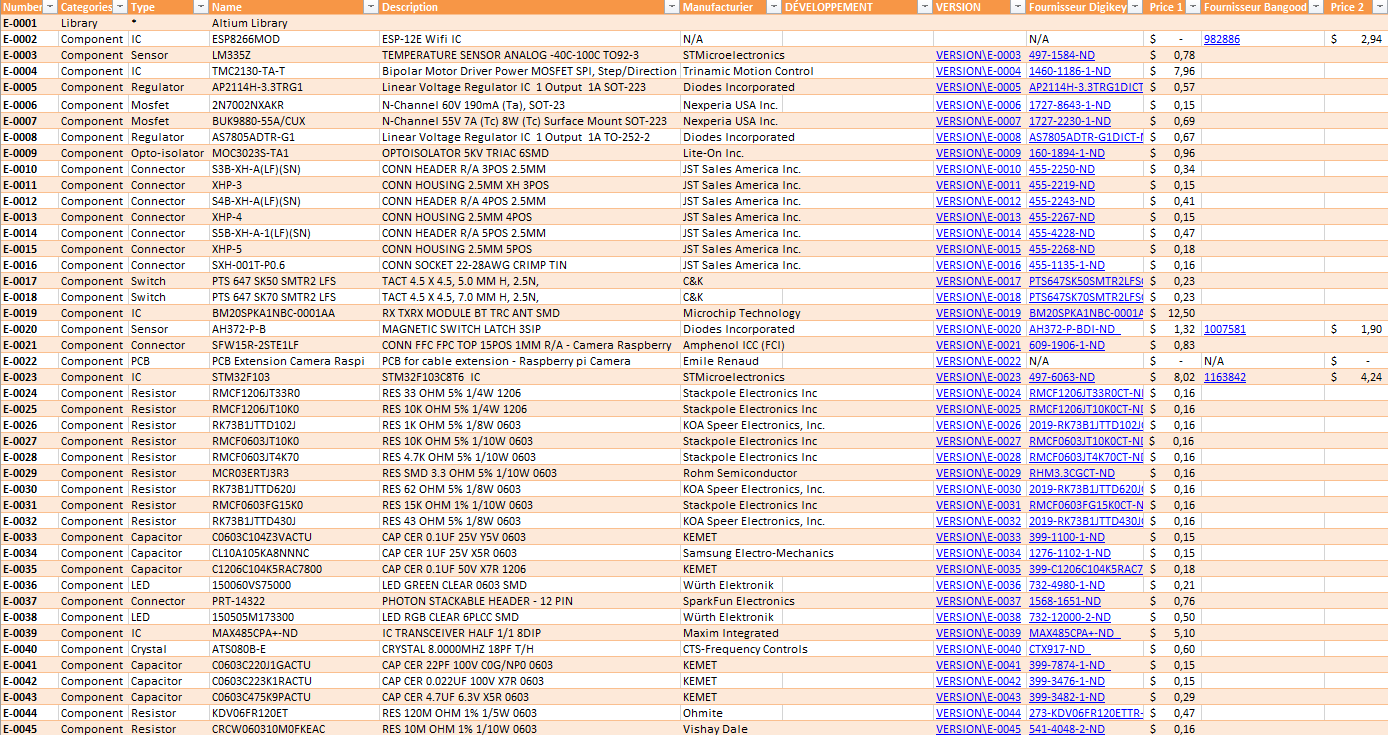
L’architecture logicielle du programme sur PC pour contrôler le bras est relativement simple. Celle de notre contrôleur de moteur est plus complexe et intéressante. Toutes les architectures logicielles se retrouvent dans le répertorie [Paperwork\Schémas\Architecture Logicielle.](../Schémas/Architecture%20Logicielle)

Figure 3 : Architecture Logicielle Contrôleur de moteur*.*

# Préparation (4)

## Librairies de pièces :

La librairie de pièces comprend toutes les pièces du projet. Elle est liée à un document Excel, permettant ainsi d’identifier chaque composante à un identifiant unique. Le document se trouve dans [Electronic\Components](../../Electronic/Components). Tous les dossiers pour les composantes que l’on retrouve dans le fichier Excel sont dans le dossier VERSION du même répertoire. Voici un aperçu :



# Dessin et programmation (5 & 7)

## Schémas électriques & PCB :

Tous les fichiers Altium pour le contrôleur de moteur se trouvent dans le répertoire [Electronic\PCB\Motor Driver](../../Electronic/PCB/Motor%20Driver). Les fichiers pour le LCD Cape du BeagleBone sont aussi disponibles, mais moins importants. Ils sont dans le répertoire [Electronic\PCB\LCD Cape BBB](../../Electronic/PCB/LCD%20Cape%20BBB)

Les fichiers Gerber pour commander le PCB Driver Motor ainsi que les plans en PDF se trouvent dans le dossier [E-0054](../../Electronic/Components/VERSION/E-0054).

## Fonctionnement des pilotes :

Les fichiers sources pour les pilotes se trouvent dans le répertoire [Code\Driver Motor\Sources\drivers](../../Code/Driver%20Motor/Sources/drivers). Les codes pour les tester sont dans [Code\Driver Motor\TestCode](../../Code/Driver%20Motor/TestCode).

Tous les pilotes sont faits en C++. Il utilise les fonctions Arduino pour le contrôle des GPIO. Nous avons fait un pilote GPIO même si nous utilisons celui de Arduino, car nous voulions le mettre dans une classe pour faciliter son utilisation dans les interfaces.

Les pilotes disponibles sont les suivants :

* + Analog\_driver
  + GPIO\_driver
  + RS485\_driver
  + SPI\_driver
  + Timer\_driver

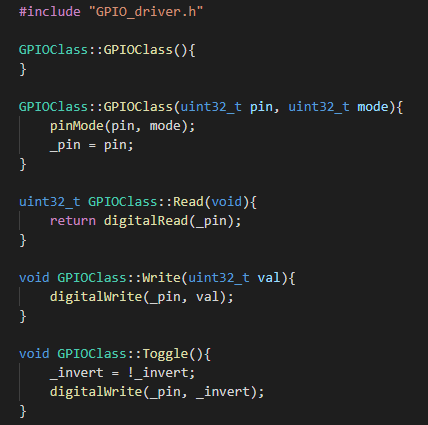


Figure 4: GPIO\_driver

# Conception mécanique (6)

## PCB & Placement des composantes :

Le PCB du pilote de moteur représentait un défi en soi. Il y avait une contrainte d’espace sur les axes du bras assez limitant. Notre PCB devait faire au maximum 42mm de diamètre et en plus devait être circulaire. Le placement des pièces ainsi que le «routing» s’est avéré plutôt difficile, mais nous avons réussi à faire un placement des composantes logique et fonctionnel.

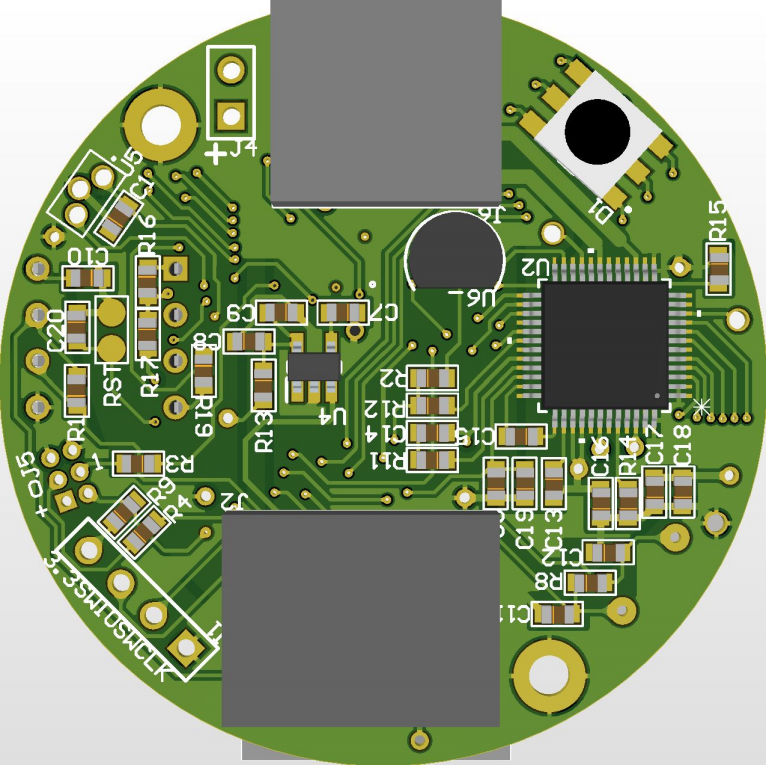
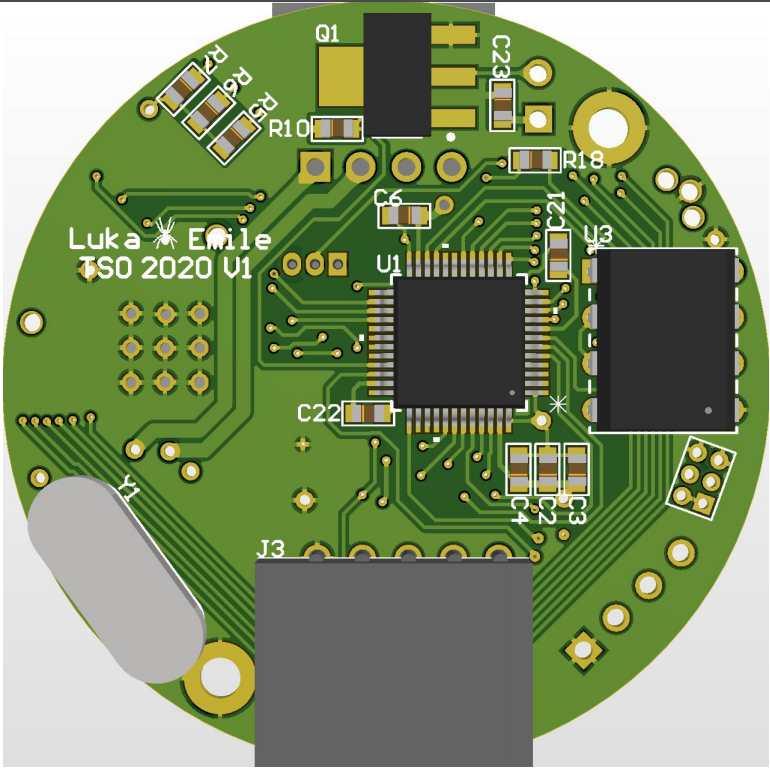


Figure 5: PCB 3D

# Évaluation et orientation (8)

## Entrevue technique :

Les documents en lien avec les entrevues techniques se trouvent dans le répertoire [Paperwork\Rencontre individuelle](../Rencontre%20individuelle)

# Assemblage et programmation (9 & 10)

## Assemblage des PCB :

Le répertoire pour l’assemblage du PCB est le [E-0056](../../Electronic/Components/VERSION/E-0056). Des instructions sont fournies pour faire les correctifs nécessaires.

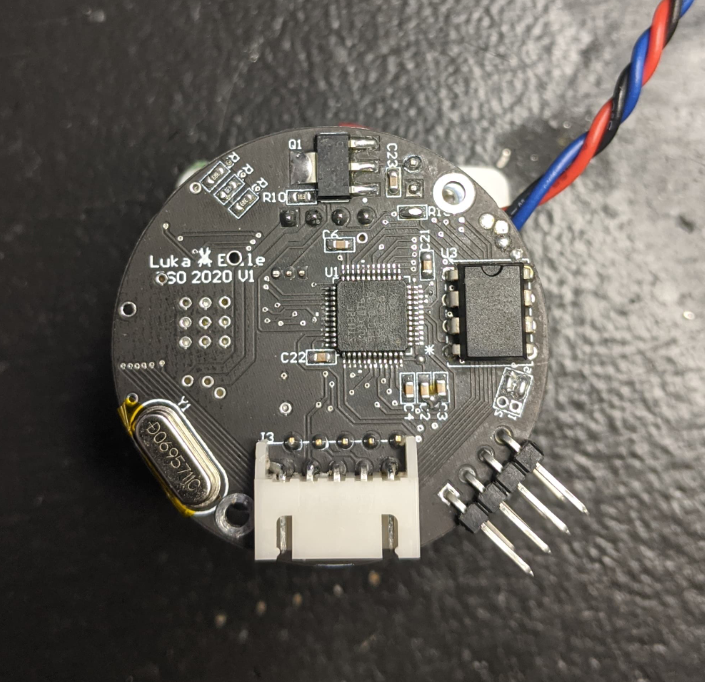
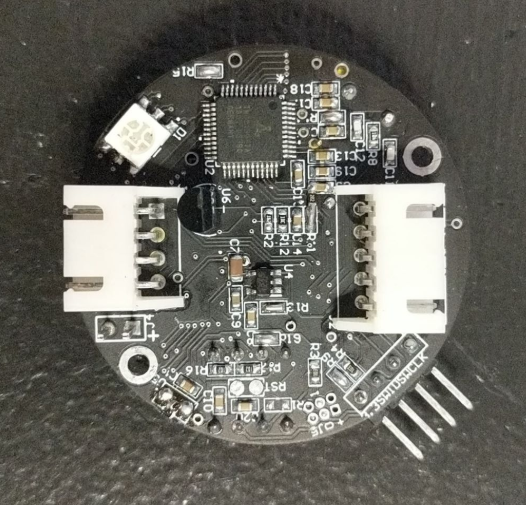


Figure 6: PCB Driver Motor Bottom & Top

## Programmation Interfaces:

Basé sur nos pilotes, les interfaces sont utilisées, en autre, pour faciliter l’utilisation des capteurs. Comme les pilotes, leurs fichiers sources sont dans le répertoire [Code\Driver Motor\Sources\interfaces](../../Code/Driver%20Motor/Sources/interfaces) et les codes test dans [Code\Driver Motor\TestCode](file:///D:\SDD%20Files\Documents\6DOF-Robotic-Arm\Code\Driver%20Motor\TestCode).

Les interfaces disponibles sont les suivants :

* + Communication\_interface
  + Debug\_interface
  + Fan\_interface
  + HallSensor\_interface
  + TempSensor\_interface
  + TMC2130\_interface

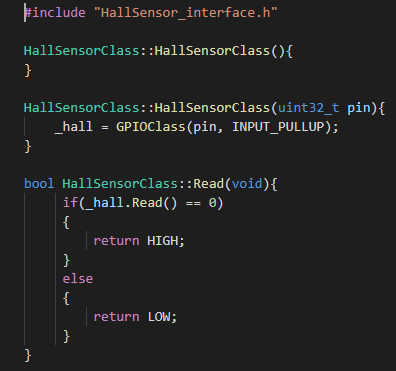


Figure : Interface Hall Sensor



Figure 8: Interface Capteur Température

# Vérification et validation

## Test des logiciels :

Chaque section de code est testée individuellement avant d’être intégrée dans le vrai code. Cela nous permet de passer moins de temps à déboguer un gros code. Les codes de test sont dans le répertoire [Code\Driver Motor\TestCode](file:///D:\SDD%20Files\Documents\6DOF-Robotic-Arm\Code\Driver%20Motor\TestCode).

Nous avons aussi fait une section de débug dans l’interface graphique sur PC pour nous aider à déboguer le code principal. Nous pouvons faire des trames de communication sur mesure pour tester chaque composante du PCB. Nous utilisons un adaptateur RS-232 à RS-485 que nous avons fait pour communiquer avec le bras.

Comme le développement de l’interface graphique c’est fait dans Visual Studio, nous avons testé des fonctionnalités de VS et avons créer un nouveau répertoire Github pour le logiciel. Il se trouve à l’adresse suivante : <https://github.com/mimil2014/GUI_Robotic_Arm>.

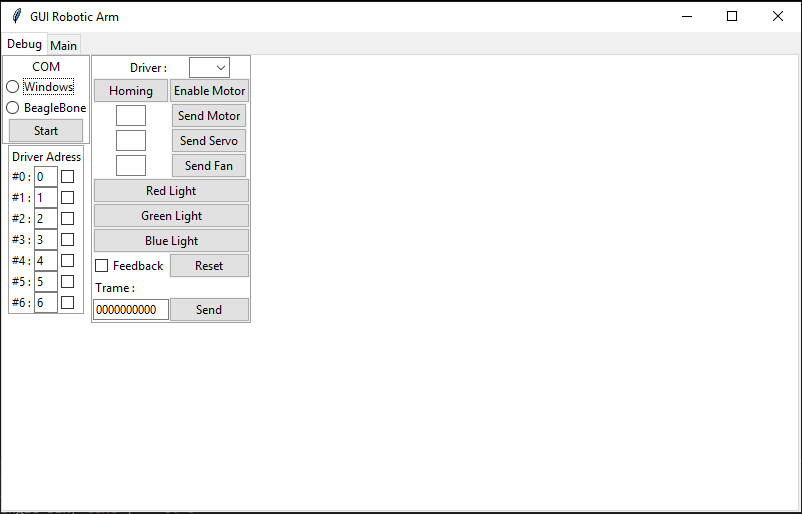


Figure : Mode debug, interface graphique

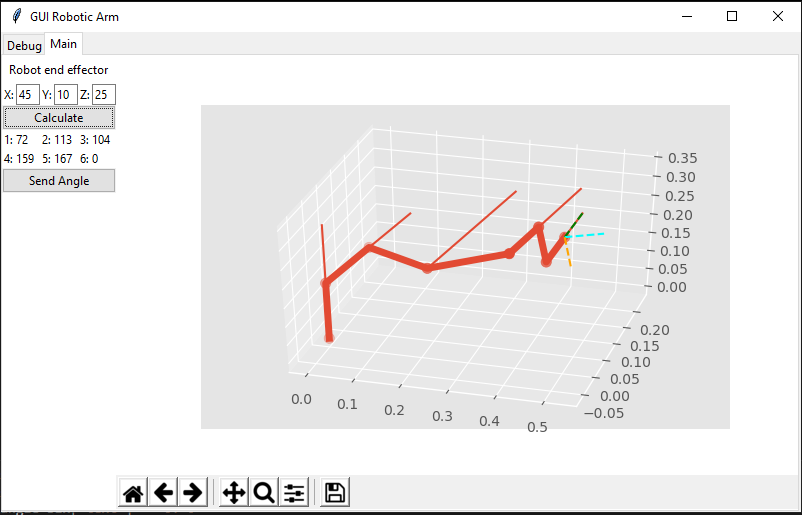


Figure : Cinématique inverse, interface graphique

## Processus :

Les deux processus importants sont le celui pour contrôler le moteur et celui pour gérer la température. Celui de la communication n’est pas beaucoup utilisé. Le processus regarde seulement si l’information est pour lui et l’envois dans de nouvelles variables. Nous avons gardé le « main » (while(1) ou loop()) le plus court possible. Nous avons alors fait quelques fonctions qui font appel au processus pour faire bouger le moteur et contrôler la température. Le processus de température n’a pas de fonction dans le main. On utilise l’objet dans la fonction process\_motor et process\_debug. Les fichiers sources sont dans le répertoire [Code\ Driver Motor\Sources\Processes](../../Code/Driver%20Motor/Sources) et les codes test dans [Code\Driver Motor\TestCode](file:///D:\SDD%20Files\Documents\6DOF-Robotic-Arm\Code\Driver%20Motor\TestCode).

Les processus disponibles sont les suivants :

* + Temperature\_process
  + Gripper\_process (pour une future pince)
  + Communication\_process
  + TMC2130\_process

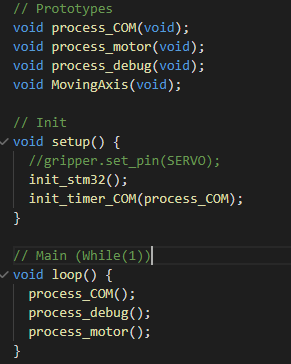


Figure 11:Fonction Main

## Configuration :

Du code de configuration pour le STM32 est dans le répertoire [Code\Driver Motor\Sources\Config](../../Code/Driver%20Motor/Sources/Config). Il s’agit d’un code pour initialiser toutes les GPIO. Surtout ceux qui ne sont pas utilisés.

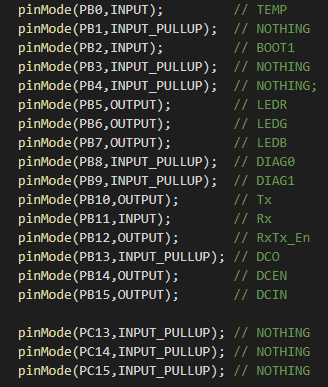
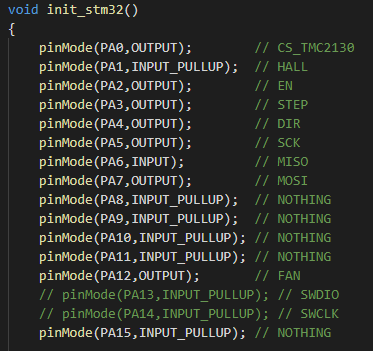


Figure 12:Fichier init\_stm32.cpp

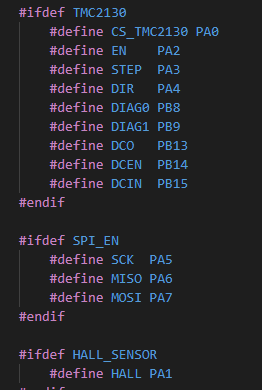
D’autres fichiers de configuration sont présents en fichier .h dans le répertoire [Code\Driver Motor\Includes\Config](../../Code/Driver%20Motor/Includes/Config). Dans le fichier configuration.h, nous avons la définition des pins du microcontrôleur : 

Figure 13:Fichier Configuration.h

Nous avons aussi l’adresse de la drive. Nous pouvons donc choisir rapidement à quelle adresse la drive va répondre :

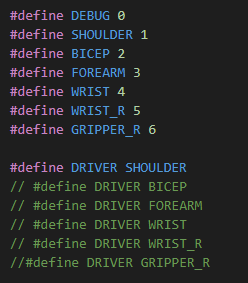


Figure 14:Fichier configuration.h

## Code actuel :

Le code à jour qui est sur le bras est encore en test. Il se trouve dans le répertoire [Code\Driver Motor\TestCode\TMC2130\_process](../../Code/Driver%20Motor/TestCode/TMC2130_process).

## Sauvegarde de l’information :

Le projet est disponible pour tous sur Github au lien suivant : <https://github.com/mimil2014/6DOF-Robotic-Arm>. La structure du projet est simple. La première série de dossiers fait la différence entre la mécanique, l’électronique, le code et le reste de la paperasse.

Dans la section électronique, le dossier Components contient la documentation des pièces électroniques ainsi que leur organisation. L’autre dossier « PCB » contient tous les projets Altium.

Dans la section mécanique nous avons un dossier STEP et STL pour les pièces que nous avons imprimées en 3D. Un dossier LCD contient le fichier source d’un support pour l’écran du Beaglebone. Le dernier dossier, Cura Settings, contient les paramètres à utiliser pour imprimer les plus petits engrenages.

Dans le dossier Code, nous avons 3 grosses sections du projet. La drive de moteur, la cinématique inverse (IK) et le four à SMD.

Dans le dossier Paperwork, les documents sont triés, en majorité, selon leur type.