

**Rapport Final**

**Pilotage d’un système dans les 3 dimensions**

M1. Master en Systèmes Embarqués et Systèmes Intégrés

Projet Master 1

Étudiant : JAVIER LÓPEZ José Antonio

No. Etudiant : 21804920

Date de livraison : 17 mars 2019

Professeur : Johann Laurent

# Introduction

L’objectif de ce document est d’expliquer comment la conception et la mise en œuvre du projet ont été réalisés.

# Développement

## Analyse du système

Le flux qui le projet fait est le suivant :

Schéma 1. Proposition d’algorithme de fonctionnement

**Initialisation**

**Tâche 1**

**Tâche 2**

**Matériel**

Initialisation des PWM

Obtention des données du capteur

Sélection du rapport cyclique

Fonctionnement des servo-moteurs

Initialisation du capteur

Le système est composé par la carte STM32 L053, le capteur LIS3DH et le Kit Servo leg VR013. Afin d'intégrer un système avec une étape d’acquisition des données et control des servo-moteurs. La fiche technique de chacun des composants a été lue pour déterminer le schéma de connexion entre les composants. En prenant en compte cette restriction, les connexions suivantes ont été effectués :

* STM32 L053 : 8 pins seront utilisés, 3 pour l'alimentation, 2 pour la communication et 3 pour le pilotage des servo-moteurs. Les pins destinés à commander les servo-moteurs seront reliés à TIM2, puisqu'il offre 4 canaux de sortie.

Tableau 1. Connexion carte STM32 L053

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **GND** | **Vcc** | **Vin** | **SDA/SDI** | **SCL** | **PWM1** | **PWM2** | **PWM3** |
| Masse général du système | Vcc-LIS3DH  SDA Pull-up LIS3DH  SCL Pull-up LIS3DH  SDO LIS3DH  CS LIS3DH | Switch | SDA LIS3DH | SCL LIS3DH | PWM servo-moteur 1 | PWM servo-moteur 2 | PWM servo-moteur 3 |

* Capteur LIS3DH : Comme le protocole I2C a été choisi pour le développement du projet. Le pin CS doit être alimenté à 3.3V (3.3V→ I2C, GND→SPI). Le pin SD0 est également alimenté à 3.3V car lui nous permet de gérer l’adresse SAD (GND→0011000b, 3.3V→0011001b). Finalement deux résistances seront nécessaires pour avoir un effet pull-up sur les pins SDA et SCL (recommandation de la fiche technique).

Tableau 2. Connexion du capteur LIS3DH

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **GND** | **Vcc** | **SDA/SDI** | **SCL** | **SDO** | **CS** |
| Masse général du système | VCC-3.3VSTM32 | Pull-up (Résistance 10kΩ).  PB9-STM32 | Pull-up (Résistance 10kΩ).  PB8-STM32 | VCC-3.3V STM32 | VCC-3.3V STM32 |

* Servo-moteurs 9g : Les composant doivent être alimentés avec une tension entre 4,8V à 6V, une alimentation externe sera donc nécessaire. La source externe pourra alimenter les servo-moteurs et pourra également alimenter la carte. STM32. Un switch sera utilisé afin de choisir si la carte électronique sera alimentée par l'ordinateur ou par l'alimentation externe (Jumper EV5→ Source externe, Jumper UV5→l’ordinateur).

Tableau 3. Connexion servo-moteur

|  |  |  |
| --- | --- | --- |
| **GND (Marron)** | **Vcc (Jaune)** | **PWM (Orange)** |
| Masse général du système | VCC-Source d’alimentation externe | PWM1 – PA0 STM32  PWM2 – PA1 STM32  PWM3- PB11 STM32 |

## Génération de taches

FreeRTOS est un système d'exploitation multitâche en temps réel qui permet de gérer les ressources matérielles et les temps d'exécution des différentes tâches mises en œuvre (Cruz, 2016). Pour le développement de ce projet, deux tâches principales ont été identifiées :

* Acquisition de données. L'étape d'acquisition des données est une activité qui s'effectue en continu. Freertos peut donc gérer la tâche d’une façon périodique. Tout d'abord, il est nécessaire d'initialiser une variable qui peut stocker le nombre total d'interruptions de tick qui se sont produites depuis le lancement de l'ordonnanceur. La utilisation de la fonction xTaskGetTickCount() a donc été utilisée.

Ensuite, la fonction vTaskDelayUntil a été utilisée pour bloquer la tâche jusqu'à ce que le nombre de tiques soit atteint. Une façon simple de gérer cette tâche est d'utiliser la fonction "pdMS\_TO\_TICKS ", qui permet de convertir les millisecondes en ticks.

* Contrôle des servo-moteurs. Cette tâche permet de gérer le rapport cyclique de tous les moteurs, en utilisant les données récupérées dans la tache précédant. L'utilisation d'un sémaphore est proposée afin de synchroniser les données. La tâche sera exécutée jusqu'à ce que l'acquisition des données soit terminée.

Un fois le sémaphore a été créé, sur la tâche à contrôler, un conditionnel a été place afin de comparer si le signal peut être exécuté (à l'aide de la fonction xSemaphoreTake). La tâche sera dans un état bloqué, quand elle reçoit les permissions correspondant, elle sera exécutée qu'une seule. La tâche de contrôle "Acquisition de données" définira quand déverrouiller la tâche "Contrôle des servo-moteurs" à l'aide de la fonction xSemaphoreGive.

La définition des tâches dépendra des besoins de notre système. Les deux tâches ont été sélectionnées, puisqu’il est intéressant de gérer plus d'une tâche à la fois.

## Acquisition de donnés

Le capteur peut se communiquer en utilisant les protocoles SPI et I2C. La communication I2C a été choisie parce qu'il s'agit d'un protocole de communication par rapport au SPI. Ce type de protocole est aussi recommandé quand on veut avoir un système avec plusieurs capteurs, car il est possible d'avoir différentes configurations (Maitre / Esclave, Maitre / Multiple esclaves, Multiple Maitres / Multiples esclaves) (Gao, 2014). L’acquisition de données est divisée en trois étapes :

### Configuration du capteur

L'adresse SAD (Slave ADdrees) nous permet d'accéder au capteur pour demander et écrire des informations dans des enregistrements spécifiques. Il existe deux types d'adresses SAD, en fonction de la configuration du pin CS (GND→0011000b, 3.3V→0011001b. Un décalage de bits doit être effectuée puisque la dernière position déterminera si le capteur est utilisé pour écrire ou lire des informations (Table 12. SAD+Read/Write paterns de la fiche technique). L’addresse SAD utilisé sur le projet est 0X19 avec un décalage à gauche d'une position. Deux configurations sont nécessaires pour utiliser le capteur :

* La première configuration est de sélectionner l'échelle de gravité à utiliser. Cette configuration se fait dans le registre CTRL\_REG24(0x23h). Pour utiliser l’échelé ±2G, les bits FS1→FS0 doivent être zéro, la valeur à écrire dans le registre est 0x00.
* La deuxième configuration à effectuer est l'activation des axes pour leur fonctionnement ainsi que la configuration du débit de données. Cette configuration se fait dans le registre CTRL\_REG1 (0x20h). Pour que les trois axes soient activés, les bits Zen, Yen, Xen ont été mis à '1'. La configuration de débit de données est 50 Hz, pour cette raison les bits ODR3→ODR0 ont été modifiés avec la valeur binaire '0100'. En effectuant l'intégration des bits, la valeur à écrire dans le registre est 0x47.

Avec cette configuration, nous pourrons alors établir une communication avec le capteur. Cette configuration a été réalisée en tant que fonction, de sorte qu'il puisse être appelé dans le code du programme.

### Réception des données

Puisque l'information de chaque axe est exprimée en deux compléments. La distribution suivante représente les registres par axe :

Tableau 4. Registre des données

|  |  |  |
| --- | --- | --- |
| **Axe** | **Registre [OUT\_H]** | **Registre [OUT \_L]** |
| X | 0x29 | 0x28 |
| Y | 0x2B | 0x2A |
| Z | 0x2D | 0x2C |

Afin d’obtenir la valeur de chaque axe, la fonction "HAL\_I2C\_Mem\_Read" a été utilisé. Trois paramètres sont importants pour l’utilisation de la fonction. Premièrement il faudra indiquer l'adresse SAD, puis le registre ou nous volons accéder et finalement la variable où l’information sera stockée. Cette procédure doit être effectuée pour chacun des compléments

Pour joindre les deux valeurs, il est nécessaire de déplacer la valeur " OUT\_H ". Le décalage doit être de huit positions à gauche, de cette façon l'information sera enchainée. Pour que cette procédure se déroule efficacement, un format int16 doit être utilisé.

## Génération d’un PWM

### Configuration du signal

Dans le but de contrôler la position du servo-moteur à travers les données reçues. La carte électronique devrait de moduler le rapport cyclique du PWM afin de positionner le servo-moteur sous un angle différent. La fiche technique du servo-moteur conseille d'utiliser une fréquence de 50 Hz, afin d’avoir une période de 20ms. Pour programmer telle fréquence nous devons déterminer le ‘‘Prescaler’’ et le ‘‘Period’’, la formule suivante a donc été utilisée :

Si et nous proposons un ‘‘Period’’ égal à 1,999. La valeur de ‘‘Prescaler’’ est :

L'horloge sélectionnée pour la configuration du PWM est TIM2. Puisqu'il dispose de 4 canaux de sortie, ce qui signifie que chaque sortie aura la même fréquence mais le rapport cyclique pourra être modifié indépendamment. Seulement 3 canaux ont été choisis puisque nous avons que 3 servo-moteurs. Les canaux suivants ont été choisis :

Tableau 5. Canaux à utiliser

|  |  |  |
| --- | --- | --- |
| **Canal 1** | **Canal 2** | **Canal 4** |
| PA0 | PA1 | PB11 |

La configuration ci-dessus doit être effectuée à l'aide du logiciel Cubemx.

### Détermination du rapport cyclique

En utilisant le programme Keil, nous pouvons modifier le rapport cyclique de notre signal PWM. Selon la fiche technique, le servo-moteur présente le comportement suivant : ‘‘La position "0°" (impulsion de 1,5 ms) est centrale, "90°" (impulsion ~2ms) est tout à droite, "-90°" (impulsion ~1ms) est tout à gauche’’. Cependant, après différents tests, il a été constaté que chaque servo-moteur avait un aspect de fonctionnement différent. Les résultats suivants ont été obtenus :

Tableau 6. Relation Servo-moteur- Rapport Cyclique

|  |  |  |
| --- | --- | --- |
| **Servo-moteur** | **Impulsion [ms]** | **Rapport Cyclique [%]** |
| 1 | 0.50-2.50 | 2.5-12.5 |
| 2 | 0.45-2.15 | 2.25-10.75 |
| 3 | 0.50-2.20 | 2.5-11 |

Tableau 7. Relation Position - Rapport Cyclique

| **Position [°]** | **Servo-moteur 1**  **Axe X [ms]** | **Servo-moteur 2**  **Axe Y [ms]** | **Servo-moteur 3**  **Axe Z[ms]** |
| --- | --- | --- | --- |
| -90 | 0.50 | 0.45 | 0.50 |
| 0 | 0.150 | 1.30 | 1.30 |
| 90 | 2.50 | 2.15 | 2.20 |
|  | Écarte de 100 unités d’une position à autre | Écarte de 85 unités d’une position à autre | Écarte de 90 unités d’une position à autre |

Tableau 8. Valeur maximale par Axe

|  |  |  |
| --- | --- | --- |
| **|Axe X|** | **|Axe Y|** | **|Axe Z|** |
| 16921 | 16977 | 19157 |

En prenant comme référence le rapport cyclique de fonctionnement de chaque servo-moteur, les données reçues par le capteur et la valeur maximale de chaque axe, une formule peut être utilisée pour faire varier le rapport cyclique :

De cette façon, les servo-moteurs pourront fonctionner en fonction des valeurs reçues par le capteur. Il est important de mentionner que la valeur maximale du capteur est obtenue expérimentalement.

## Transmission de données

Afin de valider le fonctionnement du capteur, l'analyse des données est nécessaire. L'utilisation du logiciel MATLAB est donc proposée. Les étapes suivantes permettent d'envoyer les données du capteur via la communication UART :

* En utilisant le logiciel Cubemx, le port UART2 a été activé car il est relié à la communication STM32↔ PC. La vitesse de communication a été réglée à 115200 bauds. Puisque la carte ne transmettra que des informations, il ne sera pas nécessaire de gérer les interruptions.
* Comme il s'agit d'une tâche secondaire, l'envoi d'informations a été placé dans la tâche de "Acquisition des données". La fonction HAL\_UART\_Transmit, a été utilisée afin de transmettre les données via UART. L’information à transmettre corresponde à une variable de 30 caractères qui contient les valeurs de X, Y et Z, séparés par des virgules.

## Traitement de données

Le logiciel Matlab est utilisé pour le traitement de données en temps réel. Afin d’établir une communication propre, il faut tout d’abord effacer toutes les traces des anciennes communications, puis il est possible de définir le port de communication à utiliser et la vitesse de communication.

Un fois la vitesse et le port ont été choisis, le port de communication est ouvert et les données commencent à arriver. Au lieu d'utiliser une boucle infinie, on utilise un boucle while. Le boucle s’arrêtera quand le temps sélectionné par l'utilisateur soit atteint. Dans le boucle, les données sont récupérées et analysées. L'algorithme suivant représente le schéma pour le traitement de données :

Schéma 2. Algorithme de communication UART (MATLAB)

Initialisation du port de communication

Temps=0

Temps\_arret=Choisi par l’utilisateur

Récupération de donnés

Traitement de données

Initialisation des variables

Fermeture du port de communication

Temp<= Temp\_arret

Temps++

**Oui**

**Non**

Le traitement des données est intégré par deux étapes essentielles :

* Transformation des données reçues sur une échelle de -1 à 1. La valeur reçue est divisée par la valeur maximale du capteur.

Calcul de position en degrés. La nouvelle échelle permet de déterminer la position du capteur à l'aide d'une fonction qui a pour entrer la valeur à calculer et l'axe de référence. En comparant le signe des valeurs, il est possible de déterminer dans quel quadrant se trouve le capteur. Le tableau suivant représente la logique qui suit la fonction pour déterminer la position du capteur en degrés :

Tableau 9. Relation des axes pour calculer la position en dégrées

| **Axe X ou Y [À calculer]** | **Axe Z [Reference]** | **Formule** |
| --- | --- | --- |
| **+** | **+** | accélération\_xy\*90 |
| **+** | **-** | 90+(90-90\*accélération\_xy) |
| **-** | **-** | 180-(90\* accélération\_xy) |
| **-** | **+** | 270+(90+90\* accélération\_xy) |
| **Axe Z [À calculer]** | **Axe X ou Y [Reference]** | **Formule** |
| **+** | **+** | 90-( accélération\_z\*90) |
| **+** | **-** | 90-(90\* accélération\_z) |
| **-** | **-** | 180+(90+90\* accélération\_z) |
| **-** | **+** | 270+(90\* accélération\_z) |

Avec la relation ci-dessus, la position en degrés du capteur est obtenue. Cette analyse est simple, cependant, elle nous permet de comparer si notre calcul pour le positionnement du système correspond à celui exécuté dans la vie réelle.

# Résultats

## Construction du circuit électronique

Afin de faciliter la manipulation du capteur, le circuit électronique suivant a été réalisé :

## Ensemble du système

Avant d'ensemble le kit VR013 il est nécessaire de positionner l'axe de chaque servo-moteur à 0°, ce qui nous permettra de nous assurer que son angle de fonctionnement est de -90 à 90°. Comme le support de bras est petit, un support plus grand a été conçu pour avoir une meilleure stabilité dans son fonctionnement.

## Réponse sortie PWM

## Traitement des données

En raison des ressources limitées de l'ordinateur, seuls deux axes de fonctionnement peuvent être affichés. Le premier test est effectué en tournant uniquement l'axe X et ensuite seulement en tournant l'axe Y.

# Conclusion

Le développement de ce projet a été un défi, car la communication I2C était la plus difficile à configurer, car il fallait faire un décalage vers la gauche a fin d'ajouter le bit de lecture ou écriture. L'inconvénient de l'utilisation d'un capteur d'accéléromètre est que les valeurs dépendent de l'accélération appliquée dans le système. Ce qui signifie que les valeurs établies pour le contrôle du système en trois dimensions peuvent devenir obsolètes. L'utilisation d'un gyroscope est donc proposée au lieu de l'accéléromètre.

Le système a été testé comme s'il s'agissait d'un bras mécanique, mais le servo-moteur de support n'a pas le couple nécessaire pour manipuler le mécanisme de façon optimale. Cela semble logique, car le kit est souvent utilisé comme une jambe au lieu d'un bras.

Finalement il faut bien choisir l'ordre d'importance des tâches parce que selon la configuration est le résultat qui donne. Par exemple, si la tâche "Acquisition données" à un niveau d'importance plus élevé, le résultat est : "Acquisition données", "Contrôle des servo-moteurs". Par contre, si la tâche "Contrôle des servo-moteurs" est plus importante, le résultat est le suivant : "Acquisition données", "Contrôle des servo-moteurs","Acquisition données". La gestion et le traitement des tâches est très intéressant, mais quand ils commencent à augmenter le nombre de tâches, sa gestion commence à être plus difficile.

# Références

Amazon. (s.d.). *Amazon*. Récupéré sur vTaskDelayUntil(): https://docs.aws.amazon.com/freertos-kernel/latest/ref/reference9.html

Amazon. (s.d.). *Amazon*. Récupéré sur xTaskGetTickCount(): https://docs.aws.amazon.com/freertos-kernel/latest/ref/reference30.html

Cruz, E. A. (2016, 08 06). *FREERTOS*. Récupéré sur Coffe Brain: http://www.coffeebrain.org/wiki/index.php?title=FreeRTOS

Gao, Y. (2014, 02 16). *What are the pros and cons of an I²C versus an SPI interface?* Récupéré sur Quora : https://www.quora.com/What-are-the-pros-and-cons-of-an-I%C2%B2C-versus-an-SPI-interface