Correction de trajectoire (partie 1/3) : Communication entre microcontrôleur et centrale inertielle

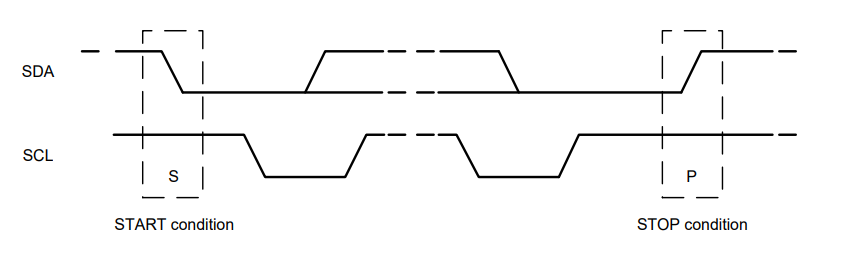
Dans la vidéo précédente avec le Smart Robot Car de Elegoo, je vous ai montré un code simple pour faire se déplacer le robot mais, vous l’avez sans doute remarqué, lorsque le robot est censé se déplacer en ligne droite, il dévie. Ainsi, le but des 2 vidéos qui vont suivre sera de vous expliquer comment faire en sorte que le robot se déplace vraiment en ligne droite. Pour cela il va falloir récupérer des données permettant de corriger la trajectoire du robot lors de son déplacement. Si vous chercher un peu sur internet, il y a différentes méthodes proposées comme l’odométrie ou le SLAM mais nous on va juste se baser sur l’angle de déviation entre la trajectoire réelle et la trajectoire désirée. L’obtention de cet angle dans notre cas repose sur un composant : Le MPU6050 qui est en faite une centrale inertielle fourni dans le kit Smart Robot Car d’Elegoo, c’est-à-dire qu’elle contient un gyroscope capable de fournir la vitesse de rotation angulaire selon les axes X, Y, Z mais aussi un accéléromètre capable de fournir l’accélération du robot selon ces mêmes axes. Il s’agit d’un composant vraiment intéressant capable d’autres fonctions que je vous invite à explorer via sa datasheet mais dans notre cas nous allons juste nous limiter à l’utilisation de son gyroscope.

Ceci dit, on se trouve face à un premier problème car comparé au driver de moteur de la dernière fois qui nécessitait juste l’envoi de pulses électriques, le MPU6050 comprend différents registres avec lesquels notre microcontrôleur doit communiquer s’il veut le contrôler. Pour faire communiquer des composants, on utilise des protocoles de communication dont les plus courant sont le protocole UART, SPI ou I2C. Pour ce qui est du MPU6050, il utilise le protocole I2C qu’il faudra donc aussi utiliser du côté du microcontrôleur pour communiquer avec lui.

Nous allons voir plus tard la section de la datasheet qui décrit comment fonctionne le protocole I2C au niveau du MPU mais le principe est globalement le même avec tous protocoles I2C. En effet on retrouve toujours 2 fils :

* SDA pour Serial DAta. Il permet de transmettre les données du dispositif maître vers le dispositif esclave ou dans l’autre sens.
* SCL pour Serial CLock. Il permet au maître de transmettre l’horloge qui agit comme un signal de synchronisation pour savoir quand lire une donnée.

Dans ce protocole, la transmission commence par un signal de départ et se termine par un signal de fin.



En dehors de ces signaux, les données sont transmises par mot de 8 bits.

La première chose à savoir qu’on on veut faire communiquer 2 dispositifs c’est le statut de chaque dispositif :

* maître : il met l’adresse de l’esclave avec lequel il veut communiquer sur le bus de même que les signaux de start et de stop.
* esclave : il répond au mettre qui lui parle

Dans la communication entre le microcontrôleur et le MPU6050, le MPU6050 agit toujours comme un esclave.

La 2ème chose à savoir est l’opération à effectuer :

* lecture : le maître envoi sur la ligne SDA, l’adresse du registre de l’esclave qu’il veut lire puis il laisse la main à l’esclave pour qu’il puisse lui transmettre la donnée demandée
* écriture : le maître envoie sur la ligne SDA, l’adresse du registre dans lequel il veut écrire puis un fois que l’esclave a confirmé qu’il est prêt à recevoir la donnée, il reprend possession de la ligne en envoyant la donnée qu’il doit écrire dans le registre esclave.

Il faut aussi savoir que plusieurs octets peuvent être transférés à la suite. Pour cela, il suffit que le maître n’envoie pas de signal de stop à la fin du premier octet. Dans ce cas, les octets suivants sont écrits au niveau des adresses suivant celle indiquée pour le premier octet.

Voici un résumé des éléments qui sont échangés lors d’un transfert :

* Pour une écriture :

Une image contenant texte, ligne, capture d’écran, Police

Description générée automatiquement

* Pour une lecture :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

On remarque que les opérations se déroulent en 2 phases :

* Phase 1 : le maître écrit à l’esclave pour lui indiquer qu’il veut effectuer une opération avec un de ses registres
* Phase 2 : Le maître indique l’opération qu’il veut effectuer avec ce registre et l’opération se produit

Maintenant qu’on a dit tout ça, je pense que vous êtes convaincu qu’il est assez complexe de programmer un protocole I2C en partant de 0. Heureusement, Arduino fournit la bibliothèque « Wire » qui implémente déjà ce protocole. Le plus simple est donc d’utiliser les fonctions qu’elle contient (<https://www.arduino.cc/reference/en/language/functions/communication/wire/>):

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Pour rappel, notre objectif est d’obtenir un l’angle de rotation autour de l’axe Z pour savoir si le robot dévie de sa trajectoire. Pour cette vidéo, nous allons juste nous contenter de voir si on peut établir une communication entre le microcontrôleur et le MPU6050. Cela nécessite 4 étapes :

1. Initialisation de la communication I2C
2. Configuration du MPU
3. Configuration du gyroscope
4. Récupération de la vitesse de rotation angulaire autour de l’axe Z.

L’initialisation de la communication I2C se fait à l’aide de la fonction begin() de la bibliothèque Wire.

La configuration du MPU nécessite la connaissance de ses registres (on peut facilement les trouver sur internet). Dans notre cas c’est le registre 107 qui nous intéresse :

Une image contenant texte, capture d’écran, ligne, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Ce registre nous permet :

* de sélectionner l’horloge qui servira à cadencer le MPU (bit0 à 2). Il est recommandé d’utiliser une horloge autre que l’horloge interne (CLKSEL=0) pour plus de stabilité
* d’activer ou non le capteur de température (bit3).
* D’activer ou non le mode CYCLE (bit5) et SLEEP (bit6)
* D’activer ou non le reset

Dans notre cas comme on s’intéresse à l’axe Z du gyroscope j’ai choisi de sélectionner la PLL ayant cet axe comme référence. J’ai désactivé tout le reste. Ainsi, la valeur à écrire dans ce registre est 0X09.

Pour la suite il est bon de noter que l’adresse du registre est 0X6B.

La configuration du gyroscope nécessite la manipulation du registre 27 du MPU :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Une image contenant texte, Police, capture d’écran, nombre

Description générée automatiquement

Ce registre permet de :

* sélectionner l’amplitude des valeurs que peut prendre le gyroscope. Dans notre cas +/- 250°/s devrait suffire donc on peut choisir FS\_SEL=0.
* Déclencher des auto-test selon les axes X (XG\_ST), Y (YG\_ST) et Z (ZG\_ST), le but de ces tests étant d’aider à choisir la valeur à mettre pour FS\_SEL.

Dans notre cas on va se contenter de sélectionner FS\_SEL=0 mais pour être plus rigoureux pour pourrait au préalable faire les auto-tests. La valeur à écrire dans ce registre est donc 0X00. Son adresse est 0X1B.

Les derniers registres qui nous intéressent sont les registre 71 et 72. En effet, les données fournies par le MPU sont sur 16 bits et donc nécessite 2 registres de 8 bits. Les registres 71 et 72 contiennent la valeur de la vitesse de rotation angulaire autour de l’axe Z (le registre 71 pour les 8 bits de poids fort de la valeur et 72 pour les 8 bits de poids faible). Ces registres seront donc à lire pour récupérer la vitesse de rotation angulaire autour de l’axe Z et leurs adresses sont 0X47 pour le registre 71 et 0X48 pour le registre 72.

Avec toutes ces informations on peut passer au code. J’ai choisi d’écrire 2 fonctions pour éviter de répéter les mêmes lignes de code :

* « lire\_byte\_i2c » : elle permet de lire un octet et nécessite pour cela l’adresse de l’esclave i2c et l’adresse du registre dans lequel on veut écrire
* « ecrire\_byte\_i2c » : elle permet d’écrire un octet et nécessite la donnée à écrire en plus de l’adresse de l’esclave i2c et du registre dans lequel on veut écrire

Ces fonctions ont été écrites en se basant sur les séquences d’écriture et de lecture extraites de la documentation technique :

Une image contenant texte, ligne, capture d’écran, Police

Description générée automatiquement

Ainsi pour la fonction « lire\_byte\_i2c », la séquence allant de S au 2ème ACK est traduite par les fonctions beginTransmission(addr), write(RA) et endTransmission(). beginTransmission(addr) va permettre de faire une requête en écriture auprès de l’esclave I2C dont l’adresse est précisée en paramètre, write(RA) permet d’indiquer la donné à écrire et endTransmission() enverra cette donnée.

La séquence allant du 2ème S à P est traduite par les fonction requestFrom(addr,1), available() et read(). requestFrom(addr,1) permet de faire une requête en lecture auprès de l’esclave I2C dont l’adresse est précisée en paramètre. Le 1 correspond au nombre de bytes à lire. Cette dernière fonction est utilisée en conjonction avec la fonction available() qui permet de savoir le nombre de bytes disponibles pour une lecture avec la fonction read() qui permettra donc de récupérer un octet transmis par l’esclave auprès duquel la requête a été faite.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Pour la fonction « ecrire\_byte\_i2c », toute la séquence est traduite par les fonctions beginTransmission(addr), write(RA), write(data) et endTransmission(). A la différence de ce qu’on faisait tout à l’heure, ici comme il y a plusieurs octets à écrire, on fait appel plusieurs fois à la fonction write() pour chaque octet. Il n’y a pas de nécessité de les envoyer immédiatement donc la fonction endTransmission() est appelée seulement à la fin.

Maintenant qu’on a ces 2 fonctions on peut réaliser les 4 étapes décrites tout à l’heure :

* Initialisation de la communication I2C : Wire.begin().
* Configuration du MPU : ecrire\_byte\_i2c(MPU\_addr,0X6B,0X09). MPU\_addr correspond à l’adresse de l’I2C esclave qui est celle du côté du MPU. On la trouve dans la datasheet du MPU. Sa valeur est 0b1101000. 0X6B est l’adresse du registre 107 vu précédemment et 0X09 est la valeur que j’ai dit qu’il fallait y écrire.
* Configuration du Gyroscope : ecrire\_byte\_i2c(MPU\_addr,0X1B,0b00<<3). Encore une fois on retrouve les informations données précédemment. 0b00<<3 permet de ne modifier que les bits 3 et 4 du registre 27, bits permettant de sélectionner FS\_SEL.
* Récupération de la vitesse de rotation angulaire autour de l’axe Z :

gz = (lire\_byte\_i2c(MPU\_addr,gyroz\_out\_h)<<8) | (lire\_byte\_i2c(MPU\_addr,gyroz\_out\_l)). Les informations récupérées dans les 2 registres sont concaténées de façon à avoir une valeur signée de sur 16bits.

La bibliothèque Serial est utilisé pour l’affichage.