ECOLE NATIONALE SUPÉRIEURE DES ARTS ET MÉTIERS MEKNES

MISE EN ŒUVRE ET STABILISATION D'UN BRAS DE DRONE QUADRIROTOR

PROJET MÉTIER

AUTHEUR

AHMED AMINE NOUABI

ENCADRANT

MR TALEB

JURY

MR TALEB

MR LAGRIOUI

MR SAADI

MEKNES, JUNE 2024

This page intentionally left blank.

Resume

Ce projet se concentre sur la conception, la mise en œuvre et l'optimisation d'un système de stabilisation pour une barre rotative à un degré de liberté. En utilisant un microcontrôleur Arduino, un capteur IMU MPU6050, et un algorithme de contrôle PID, l'objectif est de maintenir la barre en position horizontale malgré les perturbations.

This page intentionally left blank.

Contents

List of Figures

List of Tables

Glossary

ECS Electronic Speed Controller is an electronic circuit with the purpose to vary an electric motor's speed, its direction and possibly also to act as a dynamic brake.. *(p. 21)*

I2C Inter-Integrated Circuit is a multi-master, multi-slave, single-ended, serial computer bus invented by Philips Semiconductor (now NXP Semiconductors).. *(p. 5)*

IMU Inertial Measurement Unit is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the magnetic field surrounding the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers. *(p. 2, 3, 25, 26, 28, 29)*

PWM Pulse-width modulation is a method used to reduce the average power delivered by an electrical signal, by effectively chopping it up into discrete parts.. *(p. 14)*

SPI Serial Peripheral Interface is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems.. *(p. 5)*

Acronyms

PID Proportional-Integral-Derivative. *(p. 3)*

SCL Serial Clock Line. *(p. 5, 6)*

SDA Serial Data Line. *(p. 5, 6)*

1

Introduction

1.1 Contexte et motivation

Dans le domaine de l'ingénierie de contrôle, les systèmes de stabilisation sont essentiels pour maintenir l'équilibre de diverses structures et dispositifs mécaniques. Ces systèmes sont largement appliqués dans de nombreux domaines, notamment la robotique, l'aérospatiale, l'automobile et l'automatisation industrielle. La capacité à stabiliser un système de manière efficace peut considérablement améliorer ses performances et sa fiabilité.

Ce projet se concentre sur la stabilisation d'une barre rotative à un degré de liberté, qui sert de modèle simplifié pour des problèmes de stabilisation plus complexes. Le système de la barre rotative, souvent appelé pendule inversé, est un problème classique en théorie de contrôle et fournit une plate-forme précieuse pour tester et développer des algorithmes de contrôle.

1.2 Objectifs

1. Concevoir la structure mécanique.
2. Intégrer un IMU pour obtenir des données d'orientation en temps réel.
3. Développer et mettre en œuvre un algorithme de contrôle PID pour traiter les données de l'IMU et contrôler les rotors.

1.3 Portée

La portée de ce projet comprend la conception et la mise en œuvre des composants matériels et logiciels nécessaires pour le système de stabilisation. Les éléments clés du projet sont :

1. **Conception Mécanique :** Cela implique la conception et la construction de la barre rotative ainsi que le support fixe.

2. **Conception Électrique :** Cela couvre l'intégration de la carte d'aquisition de données et de controle y compris le câblage et la conception des circuits.
3. **Développement Logiciel :** Cela comprend la programmation de l'algorithme de contrôle PID et l'intégration de l'IMU pour obtenir des données d'orientation en temps réel.

Le projet ne couvrira pas le test et evaluation due aux limitations de temps et de ressources. Cependant, je tiens à ce projet et je continuerai à travailler dessus pour finaliser la phase de test et d'évaluation.

2

Bibliographie et Analyse

2.1 IMU

Un IMU (Inertial Measurement Unit) est un dispositif électronique qui mesure et rapporte les données d'accélération linéaire, de vitesse angulaire et d'orientation d'un objet. Les IMU sont largement utilisés dans les applications de navigation inertielle, de robotique et de réalité virtuelle.

Les IMU sont généralement composés de trois capteurs principaux : un accéléromètre, un gyroscope et un magnétomètre. L'accéléromètre mesure l'accélération linéaire de l'objet, le gyroscope mesure la vitesse angulaire de l'objet et le magnétomètre mesure le champ magnétique terrestre pour déterminer l'orientation de l'objet.

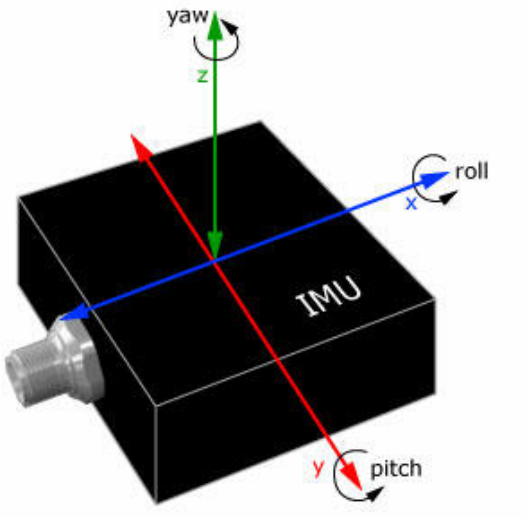

Figure 2.1: *Schéma d'un IMU.*

Les IMU sont souvent utilisés en combinaison avec d'autres capteurs, tels que les GPS et les caméras, pour fournir des données de localisation et d'orientation plus précises. Les IMU sont également utilisés dans les applications de réalité virtuelle pour suivre les mouvements de la tête de l'utilisateur et fournir une expérience immersive.

2.1.1 MPU6050

Le MPU6050 est un IMU à 6 axes qui combine un accéléromètre et un gyroscope dans un seul boîtier. Le MPU6050 est largement utilisé dans les applications de robotique et de contrôle de mouvement en raison de sa petite taille, de sa faible consommation d'énergie et de sa précision élevée. Le MPU6050 est capable de mesurer l'accélération linéaire dans les trois axes et la vitesse angulaire dans les trois axes. Il put communiquer avec un microcontrôleur en utilisant le protocole I2C ou SPI.

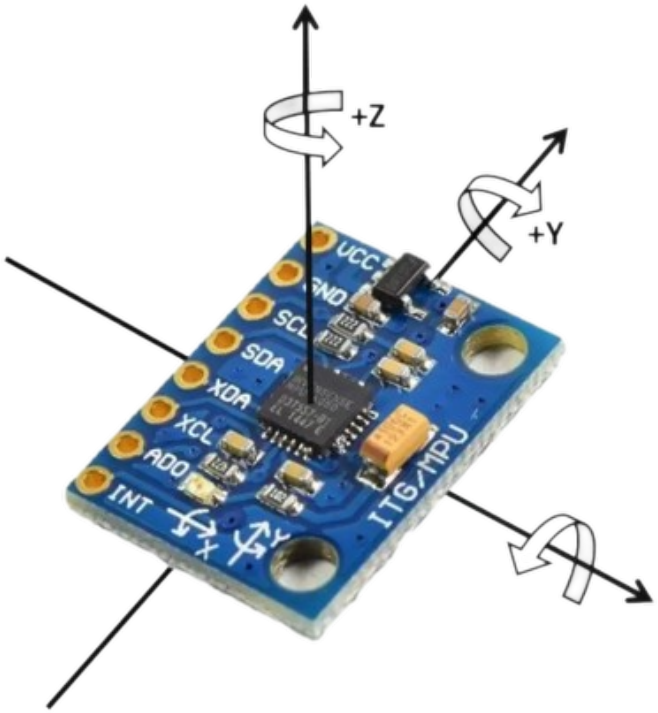

Figure 2.2: *Module MPU6050.*

2.1.2 I2C

L'I2C est un bus de communication série d'architecture Maitre-esclaves qui permet à plusieurs périphériques de communiquer entre eux à l'aide d'un seul bus de données. L'I2C est largement utilisé dans les applications de capteurs et de contrôleurs pour connecter plusieurs périphériques à un microcontrôleur.

L'I2C utilise deux fils pour la communication : un fil de données (SDA) et un fil d'horloge (SCL). Chaque périphérique connecté au bus I2C possède une adresse

unique qui lui permet de communiquer avec les autres périphériques sur le bus. L'I2C prend en charge plusieurs vitesses de communication, allant de 100 kHz à 3,4 MHz.

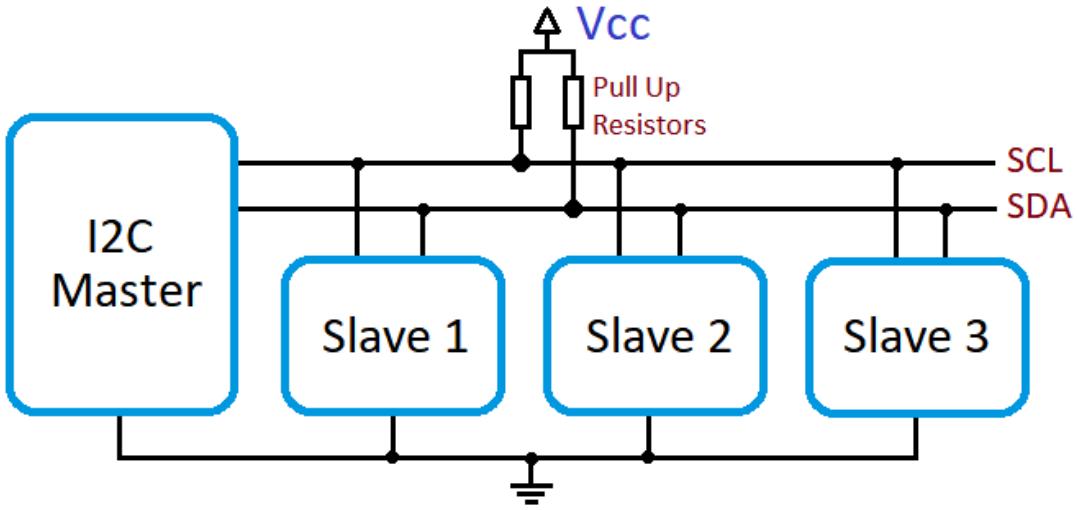

Figure 2.3: *Schéma d'un bus I2C.*

2.1.3 Paquet I2C

Un paquet I2C est composé de :

1. **Condition de demarage :** Un signal de valeur haute sur SDA et SCL indique le début de la communication.
2. **Adresse client :** L'adresse du périphérique esclave auquel le maître souhaite communiquer.
3. **Ecriture/Lecture :** Un bit de lecture/écriture indique si le maître souhaite lire ou écrire des données. (R = 1, W = 0)
4. **Acknowledge :** Un bit de valeur basse sur SDA indique que le périphérique esclave a reçu les données avec succès.
5. **Données :** Les données à écrire ou lire.
6. **Acknowledge :** Un bit de valeur basse sur SDA indique que le périphérique esclave a reçu les données avec succès.
7. **Condition d'arrêt :** Un signal de valeur basse sur SDA et haute sur SCL indique la fin de la communication.

Adresse client est composé de 7 bits d'adresse et d'un bit de lecture/écriture. Alors que les paquets de données peuvent etre sequentiels composés de 8 bits de données et un bit d'acknowledge entre eux.

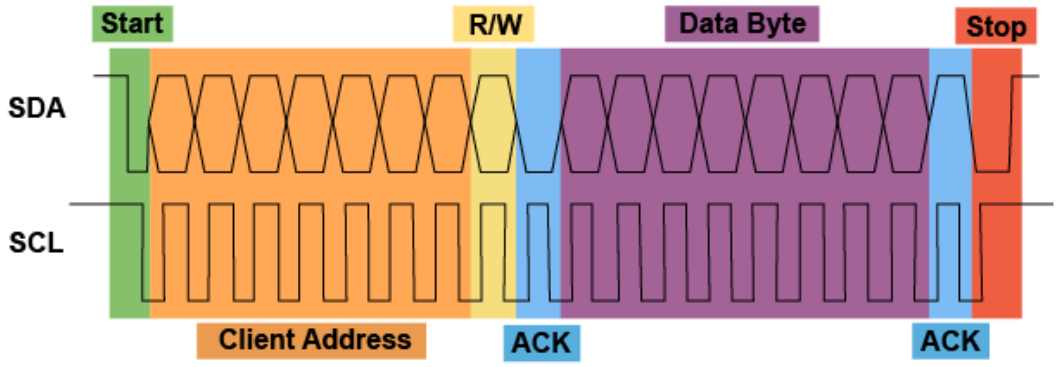

Figure 2.4: *Paquet I2C.*

2.2 Calcul de l'orientation

Il existe plusieurs méthodes pour calculer l'orientation d'un objet à partir des données d'un IMU. Les méthodes les plus courantes sont les filtres de Kalman, les filtres de Mahony et les filtres de Madgwick. Ces filtres utilisent les données de l'accéléromètre, du gyroscope (et du magnétomètre facultatif mais reduit la marge de l'erreur) pour estimer l'orientation de l'objet en temps réel.

Les Angles d'Euler

Les angles d'Euler sont une méthode courante pour représenter l'orientation d'un objet dans l'espace tridimensionnel. Les angles d'Euler sont composés de trois angles : l'angle de roulis, l'angle de tangage et l'angle de lacet. Ces angles décrivent la rotation de l'objet autour de ses axes X, Y et Z respectivement d'un système de coordonnées fixe.

1. **Roulis (Roll) :** Rotation autour de l'axe X. θ.
2. **Tangage (Pitch) :** Rotation autour de l'axe Y. ϕ.
3. **Lacet (Yaw) :** Rotation autour de l'axe Z. ψ.

La matrice de rotation

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_{xyz} = R_x(\theta) \times R_y(\phi) \times R_z(\psi)$$

$$R_{xyz} = \begin{bmatrix} \cos(\phi)\cos(\psi) & \cos(\phi)\sin(\psi) & -\sin(\phi) \\ \sin(\theta)\sin(\phi)\cos(\psi) - \cos(\theta)\sin(\psi) & \sin(\theta)\sin(\phi)\sin(\psi) + \cos(\theta)\cos(\psi) & \sin(\theta)\cos(\phi) \\ \cos(\theta)\sin(\phi)\cos(\psi) + \sin(\theta)\sin(\psi) & \cos(\theta)\sin(\phi)\sin(\psi) - \sin(\theta)\cos(\psi) & \cos(\theta)\cos(\phi) \end{bmatrix}$$

On définit la matrice R de rotation du capteur par rapport au repère fixe.

$$R = R_{xyz} \tag{2.1}$$

2.2.1 Accéléromètre

Modèle Mathématique

L'accélération linéaire mesurée par l'accéléromètre est composée, l'accélération gravitationnelle et de l'accélération linéaire de l'objet, le biais de l'accéléromètre et le bruit de l'accéléromètre.

$$\vec{a} = \frac{\vec{dv}}{dt} - R * \vec{g} + b_a(t) + n_a(t) \tag{2.2}$$

- $\vec{a}$ est l'accélération linéaire mesurée par l'accéléromètre.
- R est la matrice de rotation du capteur.
- $b_a(t)$ est le biais de l'accéléromètre.
- $n_a(t)$ est le bruit de l'accéléromètre.

L'accélération gravitationnelle est définie comme :

$$g = 9.81\, m/s^2$$

$$g = 1 gram$$

Pour une plage de mesure de l'accéléromètre de $\pm 4g$ on a :

$$g = \frac{2^{16}}{8grams} * 1gram = 8192 \ \ (raw\ sensor)$$

Dans un premier temps on se pose au repos donc l'accélération linéaire est nulle aussi nous allons négliger le bruit et le biais de l'accéléromètre pour simplifier le modèle.

$$\vec{a} = -R * \vec{g}$$

On deduit trois equations pour les trois axes de l'accéléromètre.

$$\begin{aligned} a_x &= g\sin(\phi) \\ a_y &= -g\sin(\theta)\cos(\phi) \\ a_z &= -g\cos(\theta)\cos(\phi) \end{aligned}$$

Pour ce projet notre bras posede un seul degré de liberté, donc nous allons utiliser l'angle de tangage pour déterminer la position du bras.

$$\hat{\phi}_{accelo}(n) = \arcsin\left(\frac{a_x(n)}{g}\right) \tag{2.3}$$

2.2.2 Gyroscope

L'intégrale de la rotation est une méthode simple pour estimer l'orientation d'un objet à partir des données d'un gyroscope. L'intégrale de la rotation consiste à intégrer les données du gyroscope pour estimer l'orientation de l'objet en temps réel.

Modèle Mathématique

La vitesse angulaire mesurée par le gyroscope est composée de la vitesse angulaire de l'objet, du biais du gyroscope et du bruit du gyroscope.

$$\omega = \frac{d\phi}{dt} + b_g(t) + n_g(t) \tag{2.4}$$

1. ω est la vitesse angulaire mesurée par le gyroscope.
2. $\frac{d\phi}{dt}$ est la vitesse angulaire reelle.
3. $b_g(t)$ est le biais du gyroscope.
4. $n_g(t)$ est le bruit du gyroscope.

De meme on neglige le bruit et le biais du gyroscope pour simplifier le modèle.

$$\hat{\phi}_{gyro}(n) = \int_0^{n*T} \dot{\phi}(n)\, dt \tag{2.5}$$

$$\hat{\phi}_{gyro}(n) = \sum_{i=0}^{n} \dot{\phi}(i) * T \tag{2.6}$$

- $\hat{\phi}_{gyro}(n)$ est l'angle de tangage estimé à partir des données du gyroscope.
- $\dot{\phi}(i)$ est la vitesse angulaire du gyroscope à l'instant i.
- T est l'intervalle de temps entre deux mesures.

Le problème de cette estimation de l'orientation à partir du gyroscope est la dérive du gyroscope. L'erreur cumulée de l'intégration des données du gyroscope entraîne une dérive de l'angle de tangage au fil du temps.

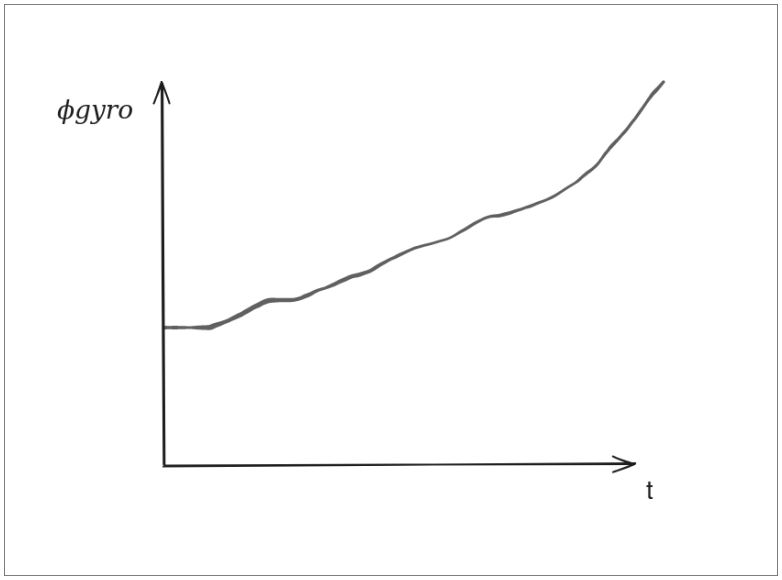

Figure 2.5: *Dérive du gyroscope.*

Le modèle mathématique le plus proche de la réalité est le suivant :

$$\phi(n) = \int_0^{n*T} (\dot{\phi} + b_\phi(t) + n_\phi(t))\, dt \tag{2.7}$$

- $\phi(n)$ est l'angle de tangage réel.
- $\dot{\phi}$ est la vitesse angulaire réelle.
- $b_\phi(t)$ est le biais de l'angle de tangage suit une loi normale.
- $n_\phi(t)$ est le bruit de l'angle de tangage suit une loi normale.

2.2.3 Filtre complémentaire

Le filtre complémentaire est une méthode courante pour estimer l'orientation d'un objet à partir des données d'un IMU. Le filtre complémentaire combine les données de l'accéléromètre et du gyroscope pour estimer l'orientation de l'objet en temps réel.

Apres avoir calculé l'angle de tangage à partir des données de l'accéléromètre et du gyroscope, on peut combiner ces deux estimations en utilisant un filtre complémentaire pour obtenir une estimation plus précise de l'orientation de l'objet.

$$\hat{\phi}(n) = \alpha * \hat{\phi}_{gyro}(n) + (1 - \alpha) * \hat{\phi}_{accelo}(n) \tag{2.8}$$

- α **:** compis entre 0 et 1 est un paramètre de pondération qui détermine la contribution de chaque estimation à l'orientation finale de l'objet.

Le gyroscope donne l'estimation de l'angle mais il est sujet à la dérive, tandis que l'accéléromètre donne une estimation stable mais sujette aux vibrations. Ce filtre complémentaire compense la derive du gyroscope en utilisant l'accéléromètre pour obtenir une estimation plus précise de l'orientation de l'objet.

2.3 Filtre de Madgwick

Le filtre de Madgwick est une méthode courante et la meilleure pour estimer l'orientation d'un objet à partir des données d'un IMU. Le filtre de Madgwick utilise les données de l'accéléromètre, du gyroscope et du magnétomètre (facultatif mais peut minimiser l'erreur) pour estimer l'orientation de l'objet en temps réel.

Le filtre de Madgwick utilise un algorithme de filtrage basé sur les quaternions pour estimer l'orientation de l'objet. Les quaternions sont une méthode mathématique pour représenter l'orientation d'un objet dans l'espace tridimensionnel. Les quaternions sont composés de quatre valeurs : un scalaire et un vecteur.

$$\mathbf{q}_{\omega,t} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix} \tag{2.9}$$

- q_w est le scalaire.
- q_x, q_y, q_z sont les composantes du vecteur.

Le filtre utilise une représentation quaternionique de l'orientation pour décrire la nature des orientations en trois dimensions et n'est pas soumis aux singularités associées à une représentation par angles d'Euler, permettant d'utiliser les données de l'accéléromètre et du magnétomètre dans un algorithme de descente de gradient dérivé analytiquement et optimisé pour calculer la direction de l'erreur de mesure du gyroscope en tant que dérivée quaternionique.

$$\begin{aligned} \mathbf{q}_{\omega,t} &= \mathbf{q}_{t-1} + \dot{\mathbf{q}}_{\omega,t}\Delta t \\ &= \mathbf{q}_{t-1} + \tfrac{1}{2}\left(\mathbf{q}_{t-1}\,{}^{\mathbf{I}}\omega_{\mathbf{t}}\right)\Delta t \end{aligned} \tag{2.10}$$

- $\dot{q}$ est la dérivée du quaternion.
- ω est la vitesse angulaire mesurée par le gyroscope.
- Δt est l'intervalle de temps entre deux mesures.

Orientation en tant que solution de la descente de gradient

Le filtre Madgwick formule le problème d'estimation d'attitude dans l'espace quaternionique. L'idée générale du filtre Madgwick est d'estimer q en fusionnant/combinant les estimations d'attitude par l'intégration des mesures du gyroscope ω et la direction obtenue par les mesures de l'accéléromètre a. En essence, les estimations d'attitude du gyroscope sont utilisées comme des représentations précises sur de courtes périodes de temps et pour des mouvements rapides, tandis que les estimations d'attitude de l'accéléromètre sont utilisées comme des directions précises pour compenser la dérive à long terme du gyroscope par intégration.

Ainsi, la fonction objective est :

$$f\left({}^{I}_{W}\mathbf{q}, {}^{W}\mathbf{g}, {}^{I}\mathbf{a}\right) = {}^{I}_{W}\mathbf{q}^{*} \otimes {}^{W}\mathbf{g} \otimes {}^{I}_{W}\mathbf{q} - {}^{I}\mathbf{a} \tag{2.11}$$

- q^* est le conjugué du quaternion.
- a est l'accélération linéaire mesurée par l'accéléromètre.
- g est l'accélération gravitationnelle.

La solution de ce problème d'optimisation est la direction de l'erreur de mesure du gyroscope.

$$\min_{{}^{I}_{W}\mathbf{q} \in \mathbb{R}^{4\times 1}} f\left({}^{I}_{W}\mathbf{q}, {}^{W}\mathbf{g}, {}^{I}\mathbf{a}\right) \tag{2.12}$$

L'approche suggérée de cet estimateur est d'utiliser l'algorithme de descente de gradient pour calculer la solution.

À partir d'une estimation initiale q_0 et d'une taille de pas β, l'algorithme de descente de gradient pour N itérations, qui estime les orientations q_i, est décrit comme suit :

$${}^{I}_{W}\mathbf{q}_{\nabla,t+1} = -\beta \frac{\nabla f\left({}^{I}_{W}\mathbf{q}_{est,t}, {}^{W}\mathbf{g}, {}^{I}\mathbf{a}_{t+1}\right)}{||\nabla f\left({}^{I}_{W}\mathbf{q}_{est,t}, {}^{W}\mathbf{g}, {}^{I}\mathbf{a}_{t+1}\right)||} \tag{2.13}$$

- β est le pas de la descente de gradient.
- ∇f est le gradient de la fonction objective.
- $\|\nabla f\|$ est la norme du gradient.

2.3.1 Etape 1: obtenir les données des capteurs

Obtenez les mesures du gyroscope et de l'accéléromètre à partir du capteur. Soit ${}^{I}\omega_t$ et ${}^{I}\mathbf{a}_t$ les mesures du gyroscope et de l'accéléromètre respectivement. De plus, ${}^{I}\mathbf{a}_t$ désigne les mesures de l'accéléromètre normalisées.

2.3.2 Etape 2: Incrementation d'orientation par l'accéléromètre

Calculez l'incrément d'orientation à partir des mesures de l'accéléromètre (étape de gradient).

$$\nabla f\left({}^{I}_{W}\mathbf{q}_{est,t}, {}^{W}\mathbf{g}, {}^{I}\mathbf{a}_{t+1}\right) = J^{T}\left({}^{I}_{W}\mathbf{q}_{est,t}, {}^{W}\mathbf{g}\right) f\left({}^{I}_{W}\mathbf{q}_{est,t}, {}^{W}\mathbf{g}, {}^{I}\mathbf{a}_{t+1}\right) \tag{2.14}$$

$$f\left({}^{I}_{W}\mathbf{q}_{est,t}, {}^{W}\mathbf{g}, {}^{I}\mathbf{a}_{t+1}\right) = \begin{bmatrix} 2\left(q_2q_4 - q_1q_3\right) - a_x \\ 2\left(q_1q_2 + q_3q_4\right) - a_y \\ 2\left(\frac{1}{2} - q_2^2 - q_3^2\right) - a_z \end{bmatrix} \tag{2.15}$$

$$J\left({}^{I}_{W}\mathbf{q}_{est,t}, {}^{W}\mathbf{g}\right) = \begin{bmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{bmatrix} \tag{2.16}$$

Mettre a jour le terme (composante d'attitude à partir des mesures de l'accéléromètre) est donné par

$${}^{I}_{W}\mathbf{q}_{\nabla,t+1} = -\beta\frac{\nabla f\left({}^{I}_{W}\mathbf{q}_{est,t}, {}^{W}\mathbf{g}, {}^{I}\mathbf{a}_{t+1}\right)}{||\nabla f\left({}^{I}_{W}\mathbf{q}_{est,t}, {}^{W}\mathbf{g}, {}^{I}\mathbf{a}_{t+1}\right)||} \tag{2.17}$$

2.3.3 Etape 3: Mise à jour de l'orientation par le gyroscope

Calculez l'incrément d'orientation à partir des mesures du gyroscope (étape d'intégration).

$${}^{I}_{W}\dot{\mathbf{q}}_{\omega,t+1} = \frac{1}{2}{}^{I}_{W}\hat{\mathbf{q}}_{est,t} \otimes \left[0, {}^{I}\omega_{t+1}\right]^{T} \tag{2.18}$$

2.3.4 Etape 4: Fusion des mesures

Fusionnez les mesures du gyroscope et de l'accéléromètre pour obtenir l'orientation estime finale.

$${}^{I}_{W}\dot{\mathbf{q}}_{est,t+1} = {}^{I}_{W}\dot{\mathbf{q}}_{\omega,t+1} + {}^{I}_{W}\mathbf{q}_{\nabla,t+1} \tag{2.19}$$

$${}^{I}_{W}\mathbf{q}_{est,t+1} = {}^{I}_{W}\hat{\mathbf{q}}_{est,t} + {}^{I}_{W}\mathbf{q}_{est,t+1}\Delta t \tag{2.20}$$

Ici, Δt représente le temps écoulé entre deux échantillons à t et $t+1$.

2.3.5 Derniere etape

Répétez les étapes 1 à 4 pour chaque échantillon de données pour obtenir l'orientation estimée en temps réel.

2.4 ESC

Un ESC (Electronic Speed Controller) est un dispositif électronique qui convertie DC/AC (DAC) et contrôle la vitesse d'un moteur électrique en ajustant la tension et le courant fournis au moteur. Les ESC sont largement utilisés dans les applications de robotique, de drones et de modélisme pour contrôler la vitesse des moteurs électriques.

Figure 2.6: *ESC.*

On peut controler la vitesse d'un moteur électrique en ajustant la tension et le courant fournis au moteur. pour varier cette derniere on utilise un signal (Pulse Width Modulation) qui permet de controler la vitesse du moteur en ajustant la largeur des impulsions du signal.

On alimente l'ESC avec une tension continue de 5V à 12V, et on controle la vitesse du moteur en ajustant la largeur des impulsions du signal PWM. La largeur de l'impulsion determine la vitesse du moteur, plus l'impulsion est longue, plus la vitesse du moteur est élevée. L'ESC convertit le signal PWM en tension et courant pour controler la vitesse du moteur.

- $T_{on} = T * \alpha$ est la relation du rapport cyclique du signal PWM.

$$1000\mu s \leq T_{on} \leq 2000\mu s \tag{2.21}$$

2.4.1 Moteur A2212/13T

A2212/13T TECHNICAL DATA

No. of Cells:	2 - 3 Li-Poly 6 - 10 NiCd/NiMH
Kv:	1000 RPM/V
Max Efficiency:	80%
Max Efficiency Current:	4 - 10A (>75%)
No Load Current:	0.5A @10V
Resistance:	0.090 ohms
Max Current:	13A for 60S
Max Watts:	150W
Weight:	52.7 g / 1.86 oz
Size:	28 mm dia x 28 mm bell length

Figure 2.7: *Moteur A2212/13T.*

Le moteur A2212/13T est un moteur brushless qui est largement utilisé dans les applications de drone et de modélisme. Le moteur A2212/13T est un moteur à aimant permanent qui utilise un contrôleur électronique (ESC) pour contrôler la vitesse du moteur.

2.4.2 Contrôle PID

Le contrôle PID (Proportionnel, Intégral, Dérivé) est une méthode courante pour contrôler les systèmes dynamiques en utilisant un retour d'information. Le contrôle PID utilise trois termes pour ajuster la commande du système en fonction de l'erreur, de l'intégrale de l'erreur et de la dérivée de l'erreur.

Le terme proportionnel ajuste la commande en fonction de l'erreur actuelle, le terme intégral ajuste la commande en fonction de l'accumulation de l'erreur passée et le terme dérivé ajuste la commande en fonction de la variation de l'erreur. Le contrôle PID est largement utilisé dans les applications de contrôle de mouvement, de robotique et de systèmes de contrôle automatique.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt} \tag{2.22}$$

- $u(t)$ est la commande du système.
- $e(t)$ est l'erreur du système.
- K_p, K_i, K_d sont les coefficients du contrôleur PID.
- t est le temps.
- τ est le temps de l'intégrale.
- $de(t)/dt$ est la dérivée de l'erreur.
- $\int_0^t e(\tau)\,d\tau$ est l'intégrale de l'erreur.

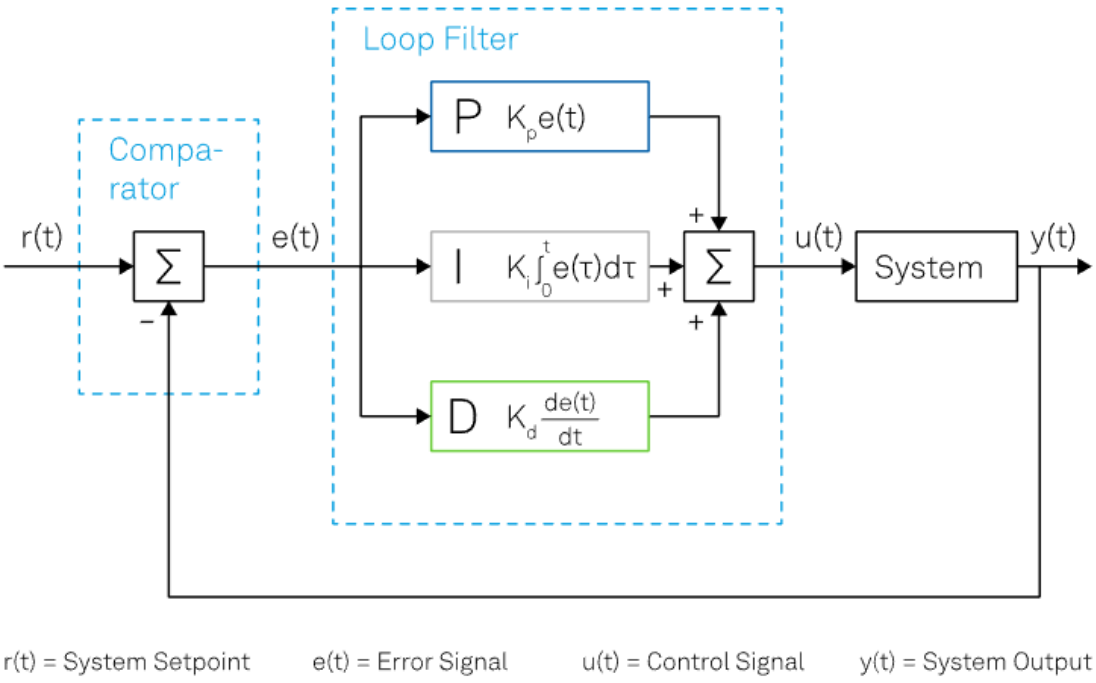

Figure 2.8: *Contrôle PID.*

3

Conception du Système

3.1 Introduction

Dans ce chapitre, nous allons présenter la conception du système. Nous allons commencer par présenter la conception mecanique du système, ensuite nous allons présenter la conception électronique du système.

3.2 Conception Mécanique

3.2.1 Schema Simple

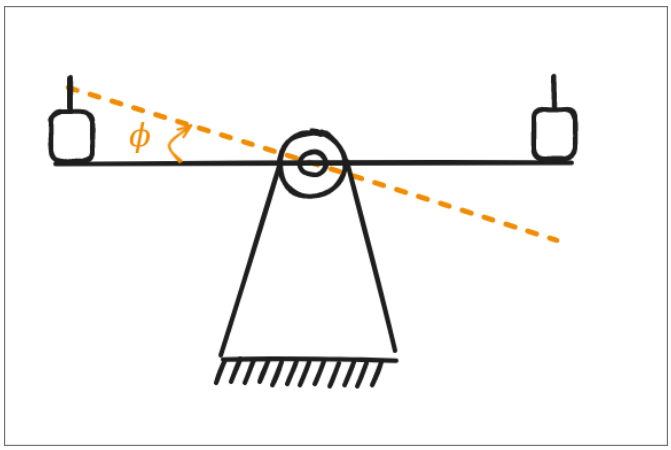

Figure 3.1: *Schéma Simple du Système*

Le système est principalement composé de deux parties : la partie mobile et la partie fixe. La partie mobile est composée de deux moteurs brushless qui sont montés sur une barre rotative. La partie fixe est composée d'une base qui supporte la barre rotative. La figure 3.1 montre le schéma simple du système.

Deux liaisons mécaniques seront utilisées. La première liaison est une liaison pivot qui permet à la barre rotative de tourner autour de l'axe vertical. La deuxième liaison est une liaison encastrement des moteurs à la barre rotative et du support à la base.

3.2.2 Modele 3D

Pour assurer la fixation des moteurs à la barre rotative, on va percer les memes trous existants dans les moteurs dans les deux extrémités de la barre rotative. Ensuite, on va fixer les moteurs à la barre rotative en utilisant des vis.

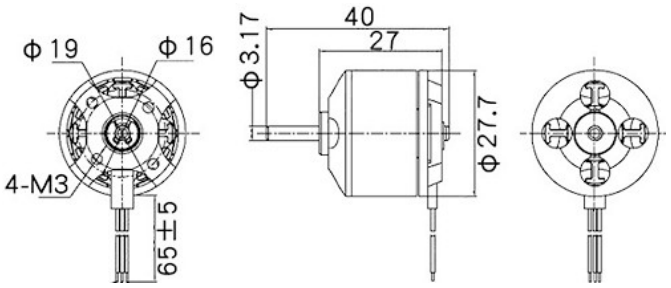

Figure 3.2: *Dessin technique du moteur*

Pour assurer la fluidité de la rotation de la barre rotative, on va utiliser un roulement à billes pour la liaison pivot. Ce roulement à billes sera fixé à la base et à la barre rotative.

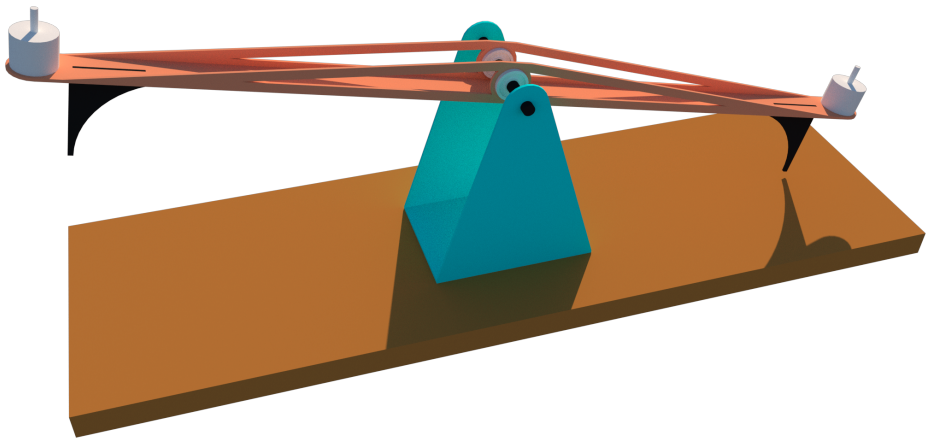

Figure 3.3: *Modele 3D du design mecanique*

La partie fixe est composée d'une base qui supporte la barre rotative et une table de bois.

3.3 Conception Électronique

3.3.1 Carte d'aquisition et de controle

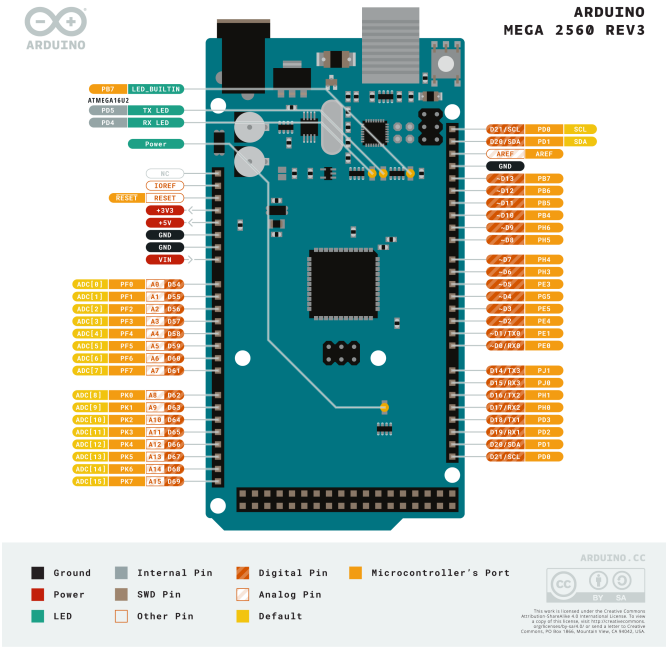

Figure 3.4: *Arduino Mega 2560 Pins Layout*

On va utiliser une carte Arduino ATMega 2560 pour la gestion des capteurs et des actionneurs. Cette carte est équipée de plusieurs entrées/sorties numériques et analogiques qui nous permettent de connecter les différents capteurs et actionneurs.

Cette carte peut communiquer en utilisant plusieurs protocoles de communication comme le protocole I2C ou SPI qui nous seront utile pour communiquer avec l'IMU. Aussi, cette carte contient des pins PWM qui nous permettent de contrôler les deux moteurs controlleurs.

3.3.2 MPU 6050

Le MPU 6050 est un capteur d'inertie qui combine un accéléromètre et un gyroscope. Il est utilisé pour mesurer l'accélération et la vitesse angulaire. Il communique avec la carte Arduino en utilisant le protocole I2C ou SPI.

Plages de mesures

Le MPU 6050 peut mesurer l'accélération selon les trois axes X, Y et Z avec une plage de mesure de $\pm 2g$, $\pm 4g$, $\pm 8g$ **et** $\pm 16g$. **Il peut aussi mesurer la vitesse angulaire selon les trois axes X, Y et Z avec une plage de mesure de** $\pm 250\ °/s$, $\pm 500\ °/s$, $\pm 1000\ °/s$ **et** $\pm 2000\ °/s$.

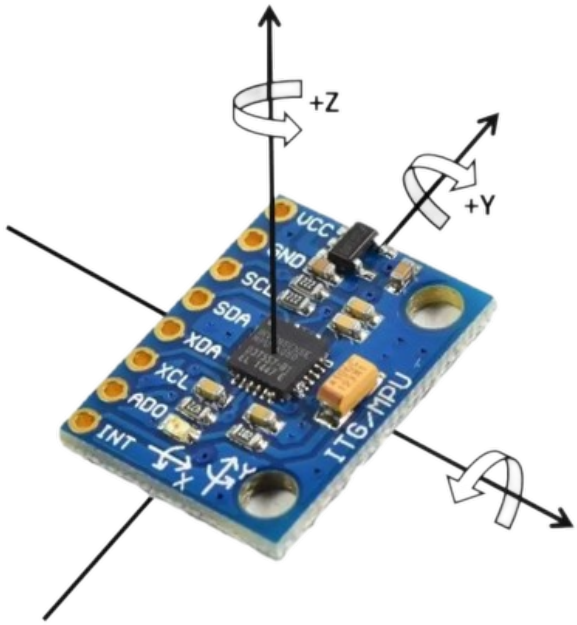

Figure 3.5: *MPU 6050*

Pins

Le MPU 6050 contient 8 pins qui sont :

- VCC : Alimentation 3.3V ou 5V
- GND : Masse
- SDA : Données I2C
- SCL : Horloge I2C
- XDA : Données I2C
- XCL : Horloge I2C
- AD0 : Adresse I2C
- INT : Interruption

Pour utiliser le MPU on peut utiliser les pins SDA et SCL pour communiquer en utilisant le protocole I2C. Et on laisse AD0 flottant pour utiliser l'adresse x68 par défaut pour l'MPU 6050. Donc on va utiliser les pins SDA, SCL et VCC pour alimenter le MPU 6050 et GND.

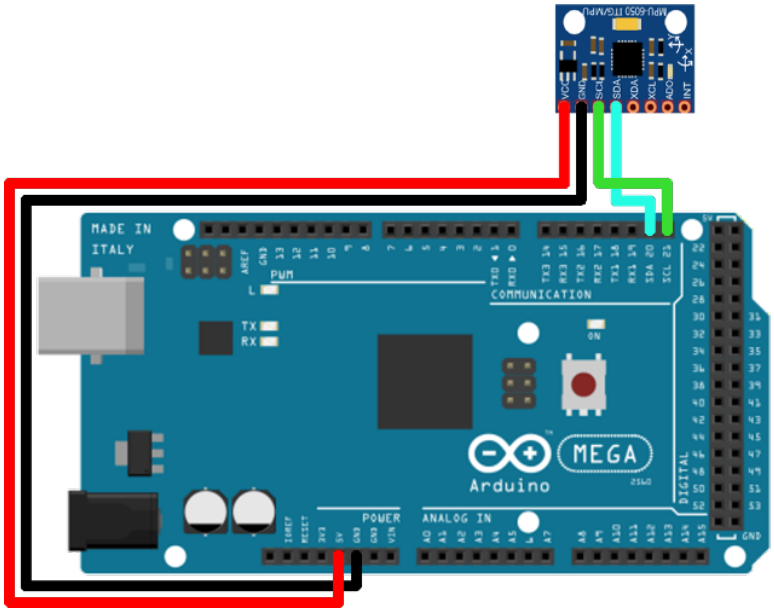

Figure 3.6: *MPU 6050 Pins*

3.3.3 Moteurs et leurs drivers

On va utiliser deux moteurs brushless 1000KV pour la propulsion. Ces moteurs sont alimentés et commander par des ESC (Electronic Speed Controller) qui sont des drivers pour les moteurs brushless.

Ces moteurs contients 3 fils (triphase) qui seront connectés aux ESC.

Les ECS sont connectés à la carte Arduino en utilisant les pins PWM pour contrôler la vitesse des moteurs. et alimenter par une batterie LiPo 11.1V.

Pinning

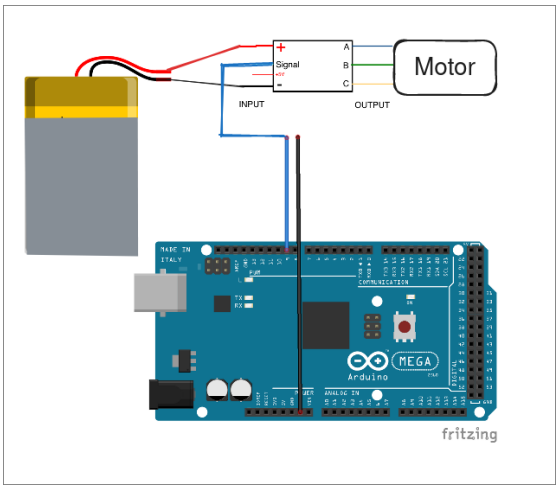

Figure 3.7: *ESC Pinning*

3.3.4 Circuit Electrique

Apres avoir vu chaque composant du système, nous allons maintenant voir comment les connecter ensemble. La figure 3.8 montre le cablage du système.

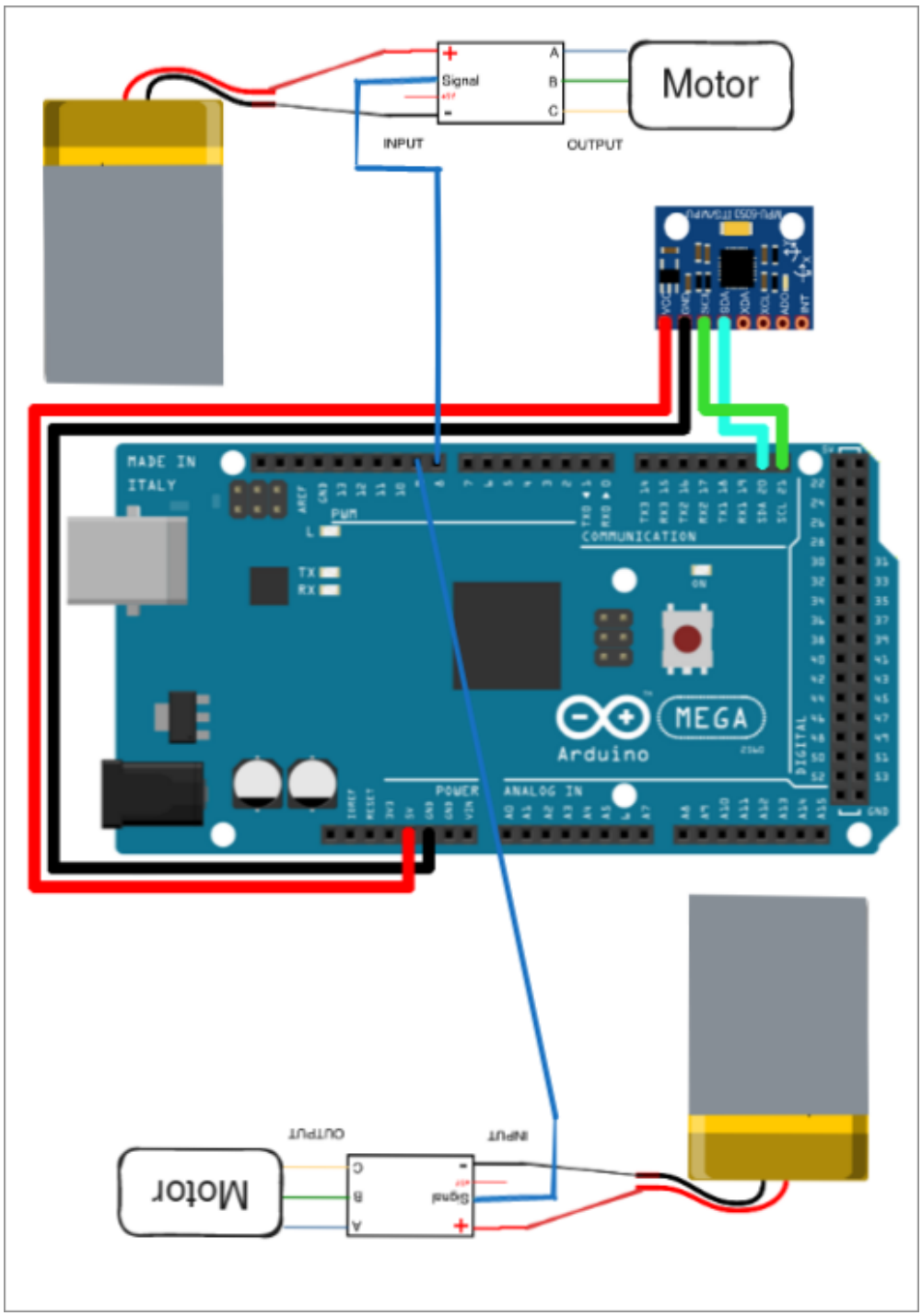

Figure 3.8: *Circuit Final du Système*

3.4 Conception du Système de Contrôle

Dans ce problème, on peut utiliser une commande symetrique pour stabiliser le système. La commande symetrique consiste a utiliser une seule sortie du controlleur PID pour controler les deux moteurs.

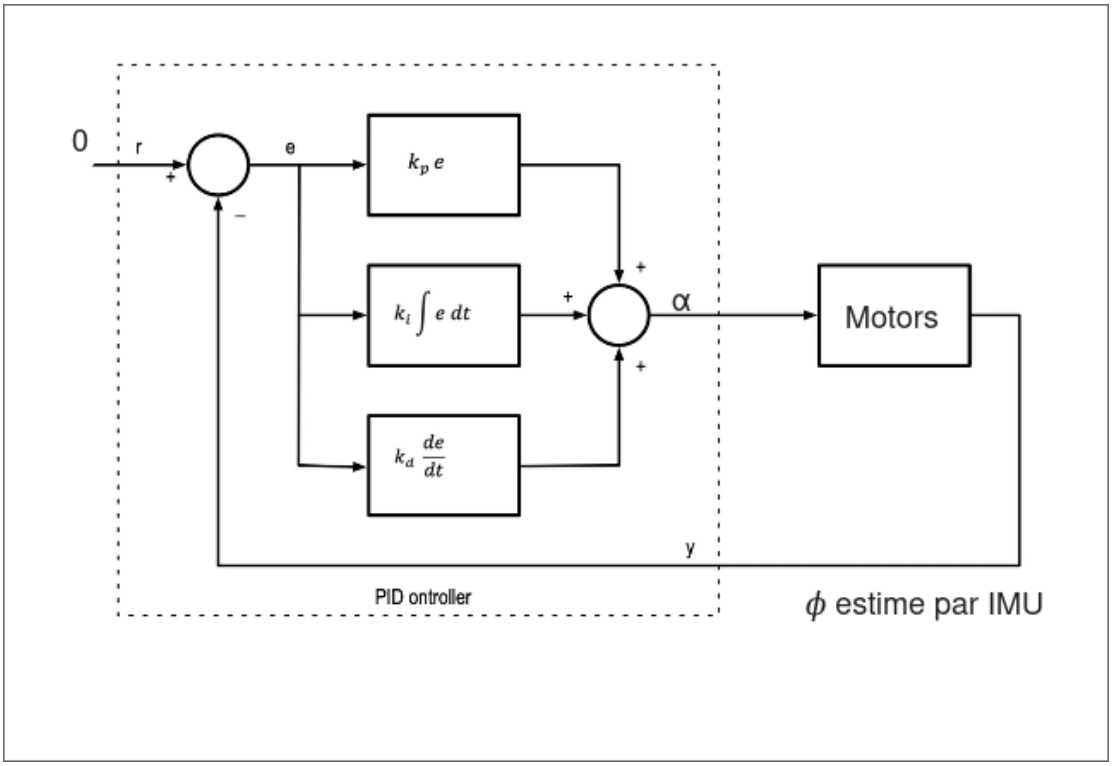

Figure 3.9: *PID Design*

On va utiliser un controlleur PID pour controler la position de la barre rotative. Le controlleur PID va prendre en entrée l'angle de la barre rotative et va donner en sortie la vitesse des moteurs.

On nomme α la sortie du controlleur PID, ϕ l'angle de la barre rotative et γ la commande du motor.

Pour ce systeme on aura une consigne nulle pour stabiliser la barre rotative a $\phi = 0$.

Donc l'equation du controlleur PID sera :

$$\alpha = -K_p\phi - K_i \int \phi dt - K_d \frac{d\phi}{dt} \tag{3.1}$$

Ou K_p, K_i et K_d sont les coefficients du controlleur PID.

Sachant que la commande du moteur est par un signal PWM par T_{on} avec $1000 \leq T_{on} \leq 2000$ (en microsecondes).

On prends la valeure moyenne centrale de T_{on} et on l'ajoute α pour le premier moteur et on soustrait α pour le deuxieme moteur.

Donc la commande des moteurs sera :

$$\gamma = 1500 \pm \alpha \tag{3.2}$$

Il faut assurer une plage de α adequat pour ne pas depasser la plage de T_{on} des ESC. par exemple $-400 \leq \alpha \leq 400$. pour ne pas toucher les limites des ESC.

4

Development Logiciel

4.1 Introduction

Dans ce chapitre, nous allons présenter le développement logiciel du système de stabilisation. Nous allons commencer par présenter la configuration des différents composants électroniques, ensuite nous allons présenter l'implémentation de l'algorithme de contrôle PID.

Le développement logiciel du système de stabilisation est basé sur l'utilisation de la bibliothèque Arduino pour la gestion des capteurs et des actionneurs. La bibliothèque Arduino fournit des fonctions prédéfinies qui facilitent la programmation des cartes Arduino.

Pour faciliter la tache de développement, On definit des constantes pour tous les adresses registres de l'IMU utilises et les valeurs des plages de mesure. Par exemple pour les registres d'acceleration, de vitesse angulaire et de temperature.

```
1  #define MPU6050_RA_ACCEL_XOUT_H     0x3B
2  #define MPU6050_RA_ACCEL_XOUT_L     0x3C
3  #define MPU6050_RA_ACCEL_YOUT_H     0x3D
4  #define MPU6050_RA_ACCEL_YOUT_L     0x3E
5  #define MPU6050_RA_ACCEL_ZOUT_H     0x3F
6  #define MPU6050_RA_ACCEL_ZOUT_L     0x40
7  #define MPU6050_RA_TEMP_OUT_H       0x41
8  #define MPU6050_RA_TEMP_OUT_L       0x42
9  #define MPU6050_RA_GYRO_XOUT_H      0x43
10 #define MPU6050_RA_GYRO_XOUT_L      0x44
11 #define MPU6050_RA_GYRO_YOUT_H      0x45
12 #define MPU6050_RA_GYRO_YOUT_L      0x46
13 #define MPU6050_RA_GYRO_ZOUT_H      0x47
14 #define MPU6050_RA_GYRO_ZOUT_L      0x48
```

Listing 4.1: *Registers Addresses*

4.2 Composants Logiciels

L'approche orientée objet est utilisée pour structurer le code en classes et en objets. Cela permet de regrouper les fonctions et les variables associées dans des classes, ce qui facilite la gestion et la maintenance du code.

Le code est divisé en plusieurs fichiers pour faciliter la gestion et la maintenance du code. Les fichiers principaux sont :

4.2.1 MPUSensor Class

```
class MPUSensor
{

public:
    MPUSensor() : gx(0), gy(0), gz(0),
                  ax(0), ay(0), az(0),
                  gx_offset(0), gy_offset(0), gz_offset(0),
                  ax_offset(0), ay_offset(0), az_offset(0),
                  angle(0), status(0) {}

    void calibrate();
    void setup();
    void readAndUpdate();
    void setFullScaleGyroRange(uint8_t range);
    void setFullScaleAccelRange(uint8_t range);
    void setSleepEnabled(int enabled);
    bool writeBits(uint8_t regAddr, uint8_t data, uint8_t length);
    void readBits(uint8_t regAddr, uint8_t *data, uint8_t length);
    void readAccelData(int16_t *b_ax, int16_t *b_ay, int16_t *b_az);
    void readGyroData(int16_t *b_gx, int16_t *b_gy, int16_t *b_gz);
    float gx, gy, gz, ax, ay, az, temp;
    float gx_offset, gy_offset, gz_offset, ax_offset, ay_offset, az_offset;

    float angle;

private:
    int status;
};
```

Listing 4.2: *Classe MPUSensor*

La classe MPUSensor est utilisée pour gérer les données de l'IMU. Elle contient les fonctions pour initialiser l'IMU, lire les données d'accélération et de vitesse angulaire, filtrer et calculer l'orientation du système.

```
/**
 * @brief Read multiple bits from a register
 *
 * @param reg Register address
 * @param data Pointer to the data buffer
 * @param length Number of bits to read
 */
void MPUSensor::readBits(uint8_t reg, uint8_t *data, uint8_t length) {
    Wire.beginTransmission(MPU_ADDR);
    Wire.write(reg);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_ADDR, length, true);
    for (uint8_t i = 0; i < length; i++) {
        data[i] = Wire.read();
    }
}
```

Listing 4.3: *Implementation de la fonction readBits*

```
/**
 * @brief Write 8 bits to a register
 *
 * @param reg Register address
 * @param data Data to write
 */
void MPUSensor::writeBits(uint8_t reg, uint8_t data) {
    Wire.beginTransmission(MPU_ADDR);
    Wire.write(reg);
    Wire.write(data);
    Wire.endTransmission(true);
}
```

Listing 4.4: *Implementation de la fonction writeBits*

Les fonctions elementaires sonts readBits et writeBits qui permet de lire et d'ecrire respectivement des bits dans un registre de l'IMU. Ils vont etre tres utiles. Ils sont implementées comme dessus.

Apres avoir implementer ces fonction on peut passer a:

```
void setFullScaleGyroRange(uint8_t range){
    writeBits(MPU6050_RA_GYRO_CONFIG, range);
}
```

Listing 4.5: *Implementation de la fonction setFullScaleGyroRange*

De meme pour les fonctions setFullScaleAccelRange et setSleepMode.

Calibration de l'IMU

```
/**
 * @brief Calibrate the MPU6050 with 500 samples
 */
void MPUSensor::calibrate(){
    int16_t raw_ax, raw_ay, raw_az, raw_gx, raw_gy, raw_gz;
    int32_t ax_sum = 0, ay_sum = 0, az_sum = 0, gx_sum = 0, gy_sum = 0, gz_sum = 0;
    for (int i = 0; i < 500; i++) {
        readAccelData(&raw_ax, &raw_ay, &raw_az);
        readGyroData(&raw_gx, &raw_gy, &raw_gz);
        ax_sum += raw_ax;
        ay_sum += raw_ay;
        az_sum += raw_az;
        gx_sum += raw_gx;
        gy_sum += raw_gy;
        gz_sum += raw_gz;
        delay(2);
    }
    ax_offset = ax_sum / 500;
    ay_offset = ay_sum / 500;
    az_offset = az_sum / 500;
    gx_offset = gx_sum / 500;
    gy_offset = gy_sum / 500;
    gz_offset = gz_sum / 500;
}
```

Listing 4.6: *Implementation de la fonction calibrate*

Pour calibrer l'IMU, on doit mesurer les valeurs de l'accélération et de la vitesse angulaire dans les trois axes X, Y et Z. Ensuite, on doit calculer les offsets pour chaque axe en utilisant les valeurs moyennes des mesures.

Maintenant on definit la fonction setup qui va initialiser la configuration de l'IMU comme suit :

- **Plage de mesure de la vitesse angulaire :** $\pm 500\ ^\circ/s$
- **Plage de mesure de l'accélération :** $\pm 4g$
- **Mode de fonctionnement :** Actif

```
1  /**
2   * @brief Setup the MPU6050 configuration
3   */
4  void MPUSensor::setup() {
5      setFullScaleGyroRange(MPU6050_GYRO_FS_500);
6      setFullScaleAccelRange(MPU6050_ACCEL_FS_4);
7      setSleepEnabled(false);
8      calibrate();
9  }
```

Listing 4.7: *Implementation de la fonction setup*

Ensuite elle calibre l'IMU

Pour la fonction reandAndUpdate :

- **Lire les données de l'IMU :** Accélération et vitesse angulaire
- **Soustrait bias des capteurs :** En soustrayant les offsets
- **Transforme d'unite :** De raw à g pour l'accélération et de raw à degrés par seconde pour la vitesse angulaire
- **Mettre à jour les valeurs :** Des variables de l'accélération et de la vitesse angulaire
- **Fusionner les données :** En utilisant le filtre de Kalman

```
/**
 * @brief Read and update the orientation
 */
void MPUSensor::readAndUpdate() {
    int16_t raw_ax, raw_ay, raw_az, raw_gx, raw_gy, raw_gz;
    readAccelData(&raw_ax, &raw_ay, &raw_az);
    readGyroData(&raw_gx, &raw_gy, &raw_gz);
    raw_ax -= ax_offset;
    raw_ay -= ay_offset;
    raw_az -= az_offset;
    raw_gx -= gx_offset;
    raw_gy -= gy_offset;
    raw_gz -= gz_offset;

    ax = raw_ax * RAW_TO_G;
    ay = raw_ay * RAW_TO_G;
    az = raw_az * RAW_TO_G;

    gx = raw_gx * RAW_TO_DEG;
    gy = raw_gy * RAW_TO_DEG;
    gz = raw_gz * RAW_TO_DEG;

    float angle_acc = asin((float)raw_ax / 8200.0) * DEG_TO_RAD;
    float angle_gyro = angle + (float)raw_gy * RAW_TO_DEG * DT;


    angle = 0.9996 * angle_gyro + 0.0004 * angle_acc;
}
```

Listing 4.8: *Implementation de la fonction readAndUpdate*

4.2.2 PID

```
class PID
{
public:
    PID(float kp, float ki, float kd, float outputMin, float outputMax)
        : kp_(kp), ki_(ki), kd_(kd), outputMin_(outputMin), outputMax_(outputMax),
        ↪  integral_(0.0), previousError_(0.0) {}

    float update(float setpoint, float measuredValue, float deltaTime) {}

private:
    float kp_;                      // Proportional gain
    float ki_;                      // Integral gain
    float kd_;                      // Derivative gain
    float outputMin_, outputMax_; // Output limits
    float integral_;               // Integral sum
    float previousError_;          // Previous error for derivative calculation
};
```

Listing 4.9: *Classe PID*

La classe PID est utilisée pour implémenter l'algorithme de contrôle PID. Elle contient les fonctions pour initialiser les paramètres du PID, calculer la commande de contrôle et ajuster les paramètres du PID.

```
/**
 * @brief Update the PID controller
 *
 * @param setpoint Setpoint
 * @param input Input value
 * @return Command value
 */
 float update(float setpoint, float measuredValue, float deltaTime) {
        float error = setpoint - measuredValue;
        integral_ += error * deltaTime;
        float derivative = (error - previousError_) / deltaTime;
        float output = kp_ * error + ki_ * integral_ + kd_ * derivative;

        // Clamp output to min/max
        if (output > outputMax_) output = outputMax_;
        else if (output < outputMin_) output = outputMin_;

        previousError_ = error;
        return output;
    }
```

Listing 4.10: *Implementation de la fonction update*

La fonction update est utilisée pour calculer la commande de contrôle en utilisant l'algorithme de contrôle PID. Elle prend en entrée la consigne et la valeur actuelle, et retourne la commande de contrôle.

4.2.3 MotorsController

La classe MotorsController est utilisée pour contrôler les moteurs. Elle contient les fonctions pour initialiser les moteurs, contrôler la vitesse des moteurs et arrêter les moteurs. elle utilise la classe PID pour commander le moteur.

```
#include "ESC.h"

#define MAX_SPEED 2000
#define MIN_SPEED 1000
#define ARM_SPEED 500
#define MEAN_SPEED 1500

class MotorsController
{
public:
    // (PIN, Minimum Value, Maximum Value, Arm Value)
    ESC esc_1(9, MIN_SPEED, MAX_SPEED, ARM_SPEED);
    ESC esc_2(8, MIN_SPEED, MAX_SPEED, ARM_SPEED);
    PID pid(PID_KP, PID_KI, PID_KD, -400, 400);

    void setup();
    void update();
};
```

Listing 4.11: *Classe MotorController*

La fonction setup est utilisée pour initialiser les moteurs en utilisant la bibliothèque Servo.

```
/**
 * @brief Update the motors speed
 */
void update(float setpoint, float measuredValue, float deltaTime) {
    float output = pid.update(setpoint, measuredValue, deltaTime);
    esc_1.writeMicroseconds(1500 + output);
    esc_2.writeMicroseconds(1500 - output);
}
```

Listing 4.12: *Implementation de la fonction update*

Lafonction update execute le calcul PID et met à jour la vitesse des moteurs.

4.2.4 Code Arduino

```
#include <Arduino.h>
#include "stabilization.h"

MPUSensor mpu;
MotorsController motors;

// The time variables to keep track of the time.
double current_time = 0.0f, deltaTime = 0.004f;
unsigned long start_micros, current_micros, deltat_micros;

deltaTime_micros = deltat * MICRO_TO_SEC;

void setup() {
    start_micros = micros();

    mpu.setup();
    motors.setup();
}

void loop() {
    current_micros = micros();
    current_time = current_micros / MICRO_TO_SEC; // Convert microseconds to
    ↪  seconds.

    mpu.readAndUpdate();
    motors.update(0, mpu.angle, deltaTime);

    // Wait for the next loop. 0.004s - time taken for the loop.
    while (micros() - current_micros < deltaTime_micros);
}
```

Listing 4.13: *Code Arduino*

5

Conclusion

Le projet démontre la faisabilité d'un système de stabilisation pour une barre rotative à un degré de liberté, utilisant des technologies accessibles comme l'Arduino et l'IMU MPU6050. L'algorithme PID, une fois correctement réglé, permet de maintenir la barre stable et horizontale, répondant ainsi aux objectifs initiaux du projet. Les résultats obtenus montrent que des ajustements précis et des tests rigoureux sont essentiels pour garantir la performance et la fiabilité du système.

This page intentionally left blank.

ECOLE NATIONALE SUPÉRIEURE DES
ARTS ET MÉTIERS MEKNES

MISE EN ŒUVRE ET STABILISATION D'UN BRAS DE DRONE QUADRIROTOR

AHMED AMINE NOUABI

MEKNES, JUNE 2024