Pegase

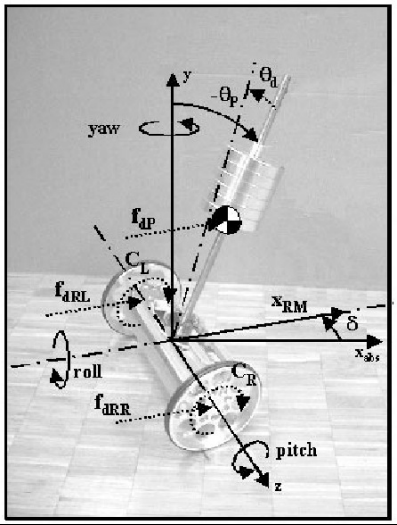
Mettre en œuvre un robot équilibriste

_ en comprenant sa programmation _

Introduction

Une des première réalisation de robot équilibriste semble dater de 2001 avec Joe, développé à l'EPFL et bien documenté par F.Gasser et A D'Arrigo,. Dès 2012, les capteurs intégrés ont facilité la mise en œuvre, et le nombre de projets a explosé, mais les descriptions sont du type "règle de cuisine". Peu comprennent ce qu'il font et pourquoi cela marche, si cel marche. Les problèmes de réglage sont toujours délicat, et les mathématiques sont peu utiles si les données ne valent rien.

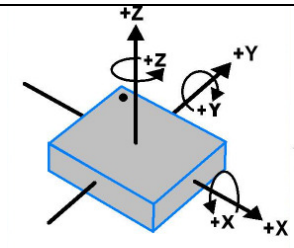
Figure 1. JOE



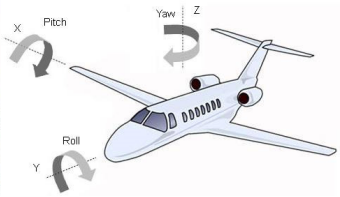
Il y a de la théorie avec des jolies formules, il y a des librairies qui appliquent ces formules en virgule flottante. Le problème est qu'un calcul ne peut donner des résultats précis que si les données sont précises et si on peut agir de façon précise.
Etudions donc en détails les composants d'un robot équilibriste et leurs comportements.
Notre projet utilise un Gy521 et des motoréducteurs 6mm, le but était de construire petit. Nos explications s'efforceront d'être générales.

Accéléromètre

Un accéléromètre mesure la gravité selon trois axes. Donc si un axe est exactement vertical, la gravité est de 1 et varie peu avec l'angle. Sur les deux axes perpendiculaires, la gravité est nulle et varie très rapidement. Avec ces trois signaux on peut donc connaitre la position du capteur dans l'espace. Mais ils sont terriblement bruités et il faut les filtrer, ce qui introduit des retards.
De plus, l'accéléromètre mesure les accélérations à la fois statique et dynamique. Il faut utiliser un filtre passe-bas pour avoir la gravité et un filtre passe-haut si on a besoin de l'accélération dynamique.



MPU-6050
Orientation & Polarity of Rotation

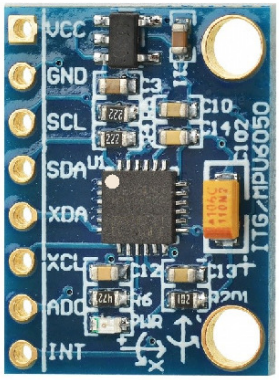


Gyroscope

Un gyroscope mesure la variation d'angle selon les trois axes. En connaissant la position initiale et en intégrant les variations, on obtient la position actuelle. Il y a de nouveau le problème du bruit, et pour chaque capteur la valeur donnée quand le capteur est

immobile n'est pas nulle (moyenne nulle puisqu'elle est bruitée). Cette dérive dépend de la tension, de la température, du circuit.	

Le capteur MPU6050 sur le module Gy521

Le module Gy521 accepte une tension de 3 à 5V, ce qu'il faut pour développer à 5V et ensuite naviguer à 3.7V en ayant éventuellement retouché les paramètres. La doc du MPU6050 est effrayante avec 120 registres de commande à première vue. En fait tous ces registres ont une valeur par défaut correcte pour notre utilisation, sauf une, nécessaire pour réveiller le circuit. Deux autres registres seront commentés plus loin. Le module est I2C, il faut ajouter les résistance pull-up (4k7). On peut ignorer les 4 signaux supplémentaires	

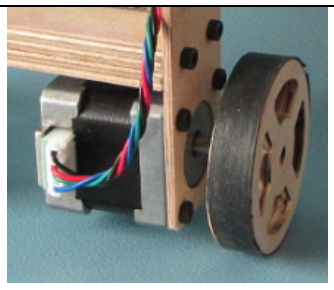
Lecture du capteur MPU6050

Les 6 données des capteurs, plus la température que l'on va ignorer, sont préparées dans un registre 16 bits, qu'I2C permet de lire par bouffée. Il faut donner l'adresse et le numéro du premier registre et ensuite on peut tout lire et stocker en mémoire. La machine interne remplit ces registres tous les 4ms (260Hz) ou moins – registre xx. Il faudra rediscuter de cela.
Vous avez un Gy521. Pour se familiariser, chargez le programme documenté sous https://playground.arduino.cc/Main/MPU-6050. Il affiche sur le terminal les valeurs des registres, et montre bien le bruit du capteur.

Banc de test

On ne comprend bien un capteur que si on on voit bien les valeurs qu'il donne dans les conditions d'utilisation, quel est son bruit et comment le réduire.
Le capteur est placé dans un cube qui permet de vérifier la documentation, constater les écarts de mesure dans les 6 positions du cube. Trois mesures consécutives sont listées, avec moyenne glissante.
Que remarquer dans ces mesures?
Le bruit semble très important pour les valeurs qui nous intéressent.
 	
Afficher sur Oled prend le même temps que sur le terminal, avec plusieurs avantages.
La librairie Adafruit, inutilement lourde et trop lente, n'est pas conseillée.
Détails plus loin.
	MPU6050ShortSerNum.ino sans filtrage
	`Horizontal  z=1G  1g is given as 16384`
	`  AcX= +01508  AcY= -00468  AcZ= +17400`
	`  AcX= +01448  AcY= -00396  AcZ= +17404`
	`  AcX= +01592  AcY= -00532  AcZ= +17376`
	`Vertical  y =1g`
	`  AcX= +01520  AcY= +16132  AcZ= +01096`
	`  AcX= +01596  AcY= +16096  AcZ= +00876`
	`  AcX= +01592  AcY= +16080  AcZ= +00888`
	`Basculé l x=1g`
	`  AcX= +17696  AcY= -00540  AcZ= +00508`
	`  AcX= +17676  AcY= -00532  AcZ= +00684`
	`  AcX= +17728  AcY= -00600  AcZ= +00652`

Moteur

Le moteur est pas à pas ou continu. Certains conseillent le moteur pas-à-pas qui peut réagir rapidement (si on met la puissance nécessaire) et a une grande gamme de vitesse. Un moteur jouet bon marché type "mabuchi" ne convient pas. Un moteur à cage tournante type Faulhaber/Maxon a un meilleur rendement, mais il ne faut pas l'utiliser en PWM, mais en PFM, voir		
https://www.didel.com/PFMversusPWMforRobots.pdf		

Une routine PFM est très simple, et s'appelle par interruption ou directement pour les tests.

Vous devez être familier avec la commande en PWM. Prenez votre montage PWM qui fonctionne et déclarez le câblage avant d'exécuter le programme ci-contre. Un potentiomètre câblé en pin A0 permettra de varier la vitesse plus simplement qu'en recompilant le programme avec de nouveaux paramètres.
Il faut un moteur avec un réducteur de 30 à 50.

```
// Definitions
#define pinMot1 4
#define pinMot2 5
MotorOn digitalWrite (pinMot1,HIGH); digitalWrite
(pinMot2,LOW)
MotorOffdigitalWrite (pinMot1,HIGH); digitalWrite
(pinMot2,LOW);
```

```
#define MaxPFM 64
void DoPfm (bytepp) {
  pfmCnt+=pp ;
  if (pfmCnt>20){
    pfmCnt-=20; MotorOn;
  } else {
    MotorOff;
  }
}
void setup () {
PinMode (pinMot1,OUTPUT);
PinMode (pinMot2,OUTPUT);
// Varianbles
Byte pfm; int ana;
// Test program
void loop () {
 ana = analogRead(A0);  // max 1023→ 32
 pfm = ana/16; // < MaxPFM
 DoPfm (pfm);
delay (2);
}
```

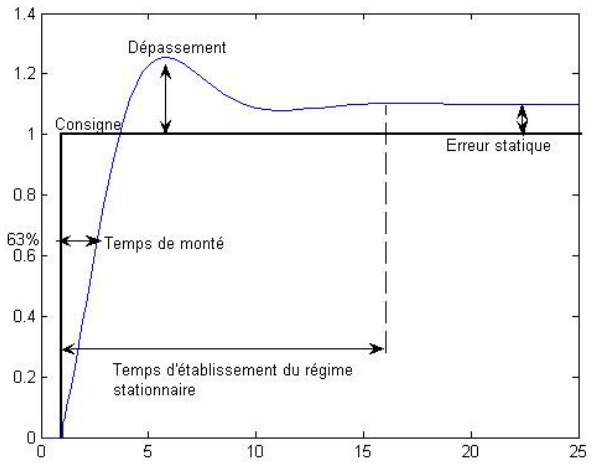
Moteurs du Pegase

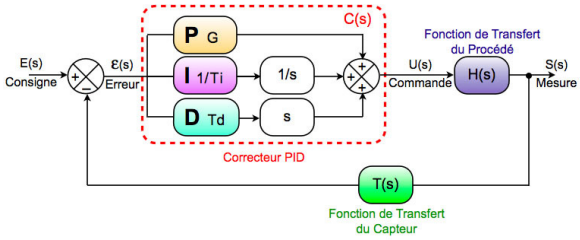
Les moteurs sont des Gm06/24 , 6mm de diamètre avec un réducteur planétaire 1:24. Environ 6000/24=200t/min= 4 tours/s. Avec des roues de 13mm, cela fait 16cm/s, ce qui est bien assez rapide pour la dimension du robot.

.

Réglage

On veut régler notre robot pour qu'il reste vertical. S'il penche en avant, il faut le faire avancer. Si la vitesse est proportionnelle à l'erreur (l'angle) on a un réglage P pour proportionnel.
Un effort, du frottement, le fait que Pégase n'est pas équilibré au départ, peut empêcher d'atteindre la position voulue ; c'est l'erreur statique. On ajoute pour corriger une fraction de l'intégrale de l'erreur, donc la somme des différences entre la position voulue et les positions successives. C'est la composante I, intégration.
On peut encore se préoccuper des oscillations, et tenir compte de la variation de l'écart. C'est la composante D, dérivée.
On a alors un réglage PID complet, mais on va laisser tomber le D parce que le capteur fluctue trop et trop vite, la variation du déplacement est noyée dans ce bruit.
La formule simplifiée pour un réglage PI est donnée ci-contre, avec
U(t) la correction à effectuer – PFM moteur
e(t) erreur de position au temps t
Int(e(t)) intégrale des erreurs depuis le début.
Kp et Ki sont des constantes qu'il faudra déterminer expérimentalement, faute d'une modélisation précise.

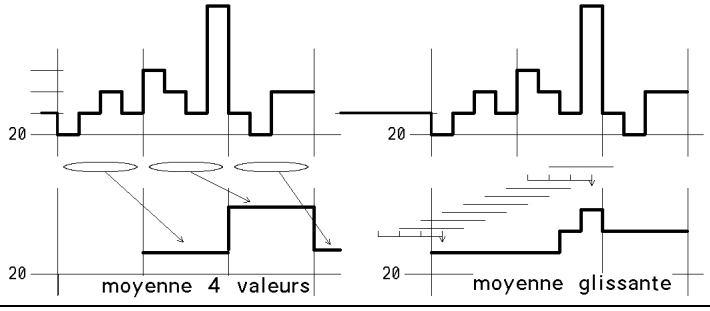




$$U(t) = Kp*e(t) + Ki*int(e(t))$$

Bruit des capteurs

Pour réduire le bruit des capteurs, il faut moyenner et il y a deux façons simples pour moyenner, et des techniques très complexes. La moyenne simple regroupe par exemple 4 mesures et donne un résultat 4 fois moins souvent, ce qui n'est évidemment pas favorable. La moyenne glissante est décrite dans https://www.didel.com/Moyennes.pdf



Dériver et intégrer

Pour dériver, on calcule la différence entre deux valeurs consécutives. Il faut mémoriser la valeur actuelle après avoir calculé la différence.
A noter que la différence est signée et que sa valeur absolue a la même taille que les valeurs soustraites.
Pour intégrer, on somme tout simplement, On intègre la différence entre la position voulue et la position actuelle. Il faut éviter un dépassement et saturer la valeur intégrée, éventuellement signaler si elle prend une valeur excessive.
Il faut initialiser correctement à l'enclenchement.

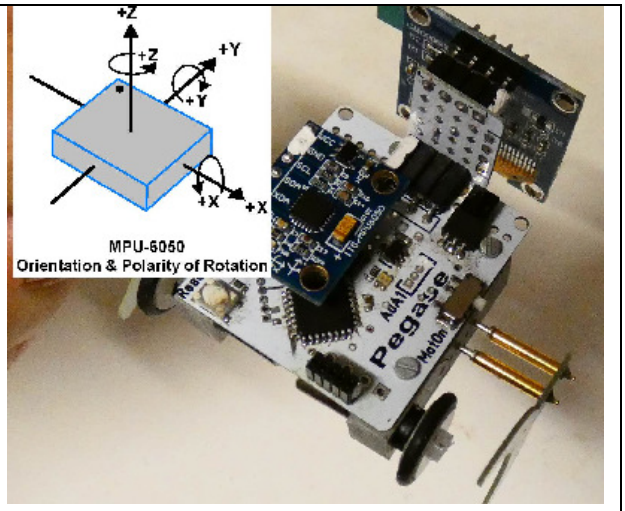
Principes de programmation

```
//Variables
int  v8, prevV8;
inu diff, itgr

A chaque lecture de v8

//Pour dériver
diff =  v8 - prevV8;
prevV8; = v8;

//Pour intégrer
itgr += v8;
```

Pegase: Environnement de test et fonctions utiles

La figure montre comment le capteur est positionné sur le robot Pegase.
La composante AcZ est horizontale et va jouer le rôle principal dans l'algorithme.
La valeur GyX peut être intégrée pour donner l'angle ou sirectement utilisée comme dérivée de l'accélération. Elle est très parasitée et elle n'est pas utilisée.
Il faut mesurer le plus précisément possible la position initiale verticale en équilibre et travailler par différence.
Nom des variables principales :

```
int16_t AcZ;      // lu sur le capteur
int16_t mAcZ;     // moyenne glissante
int16_t corAcZ;   // mAcZ- AcZini
int16_t dAcZ      // valeur intégrée
int8_t   pfmL pfmR  // pfm des 2 moteurs
```



On simplifie le problème pour ne prendre que AcZ. Au reset, le robot est immobile dans la position souhaitée et les valeurs initiales après filtrage sont mémorisées.
On donne la même valeur de pfm pour les deux moteurs. La conséquence est que Pegase va tourner lentement sur lui-même. Il faudra faire intervenir GyY, mais sans boussole, il y aura toujours de la dérive.
Visualiser les données et calculs est essentiel à chaque étape.
Un réglage P permettra de vérifier la qualité des signaux le paramètre Kp. On ajoutera ensuite l'effet de l'intégrale.

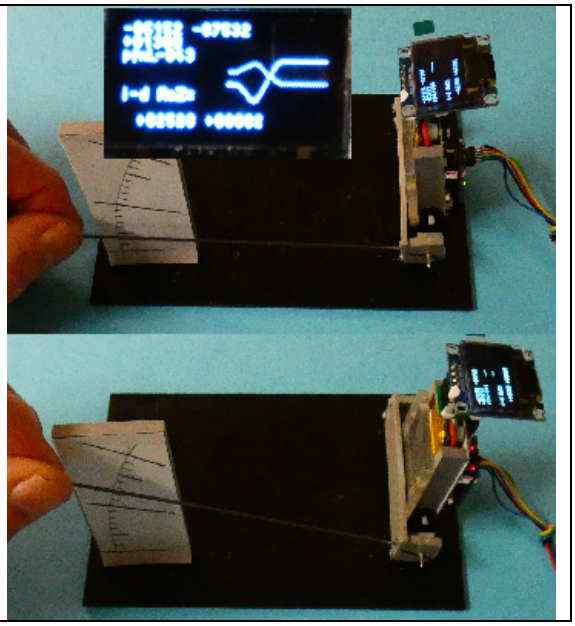
Aide à la mise au point

La librairie TerSer.h est utilisée à la place de la librairie Serial. La place mémoire est fortement réduite et l'affichage des zéros non significatifs évite les affichages en ziz-zag qui empêchent de bien voir l'évolution des variables.
Le programme est dans Pegase.zip.

Valeurs affichées par G521AcXYZSerNum.ino

```
AcX= 004D   AcY= 2091   AcZ= 0227
AcX= 0039   AcY= 2073   AcZ= 0220
AcX= 0046   AcY= 2079   AcZ= 0225
AcX= 004B   AcY= 2074   AcZ= 0226
AcX= 0055   AcY= 2069   AcZ= 0216
AcX= 0044   AcY= 2080   AcZ= 0213
```

L'affichage Oled offre les mêmes possibilités d'affichage que le terminal, non déroulantes et avec des graphiques en plus. Le temps d'écriture est similaire, env 50 ms, ce qui correspond au cycle de 50ms qu'il semble bon d'atteindre.

Le programme Gy521ZOledPfm.ino affiche les paramètres mesurés et calculés. Il en fait la moyenne glissante et calcule l'intégrale de AcZ. Et sa dérivée. On a la place pour afficher bien assez de variables, et 2 ou trois graphiques dont la position et l'amplitude nécessite un peu de tâtonnement. A droite on voir la valeur de l'accéléromètre AcZ et la valeur de PFM de correction estimées. Le support de test permet de pencher Pégase et voir la réaction des roues. Les paramètres peuvent ainsi être ajustés avant de poser le robot sur le sol	

Calcul du PFM

Commençons par un réglage P. Le Pfm est proportionnel à AcZ corrigé par l'offset initial. On vérifie sur le support que le moteur tourne dans le bon sens. Le facteur multiplicatif Kp doit être traîté différemment, utiliser la virgule flottante n'a pas de sens. Définissons une vitesse logique. Kp est une constante de type $x/2^n$. On multiplie par un nombre entier et on décale.
La table des vitesses fait correspondre à 16 vitesses logiques un PFM entre 0 et 80.

Optimisation

Le registre 25=0x19 Sample Rate Divider a une valeur liée à celle du registre 26=0x1A Configuration. La valeur par défaut est 0 et les application connues ne cherchent pas à la modifier. Un spécialiste conseille R25 valeur 22 R26 valeur 4 (passer à 6 devrait ralentir la réaction puisque les délai de filtrage passe à 19ms au lieu de 20ms).	