Rapport intermédiaire
Projet S6
Conception et commande d'un Segway

Etudiants : Ayman MOUMMADI, Papa Diatta WADE, Logan PAQUIN, Omar SIFA, Maxence NEUS

TUTEUR : Mr Komi Midzodzi PEKPE

Promotion 2023

Résumé :

Notre sujet se nomme “Conception et commande d’un Segway”. Nous devons réaliser un Segway miniature et le contrôler à distance. Autrement dit, identifier les composants nécessaires pour la fabrication du Segway et le concevoir au Fabricarium de Polytech Lille.

Ce rapport intermédiaire présente les différentes tâches qu’on a pu réaliser jusqu’à présent, la deuxième partie du projet qui consiste à finaliser la partie conception, commander notre Segway et le doter d’une caméra pour sa navigation autonome sera réalisée au semestre suivant (S7).

Abstract:

Our subject is called "Design and control of a Segway". We have to make a miniature Segway and remote it. In other words, identify the necessary components for the manufacture of the Segway and conceive it at the Fabricarium of Polytech Lille.

This interim report presents the various tasks that we have been able to carry out so far, the second part of the project which consists in finalizing the design part, ordering our Segway and providing it with a camera for its autonomous navigation will be carried out in the following semester (S7).

Plan :

I. Introduction :

a) Présentation du projet :

Notre projet consiste à réaliser un Segway miniature et à le contrôler à distance. Un Segway est un gyropode, soit un véhicule électrique monoplace, constitué d'une plateforme munie de deux roues parallèles sur laquelle l'utilisateur se tient debout, d'un système de stabilisation gyroscopique.

Nous allons créer les schémas électriques et identifier les composants nécessaires à la fabrication du Segway puis le concevoir au Fabricarium de Polytech. Ensuite nous allons programmer différentes commandes pour gérer le Segway. Cela inclut un contrôle à distance ainsi qu'un programme de pilotage autonome à l'aide d'une caméra inclus sur le système.

b) Planning :

Organisation et prévisions :

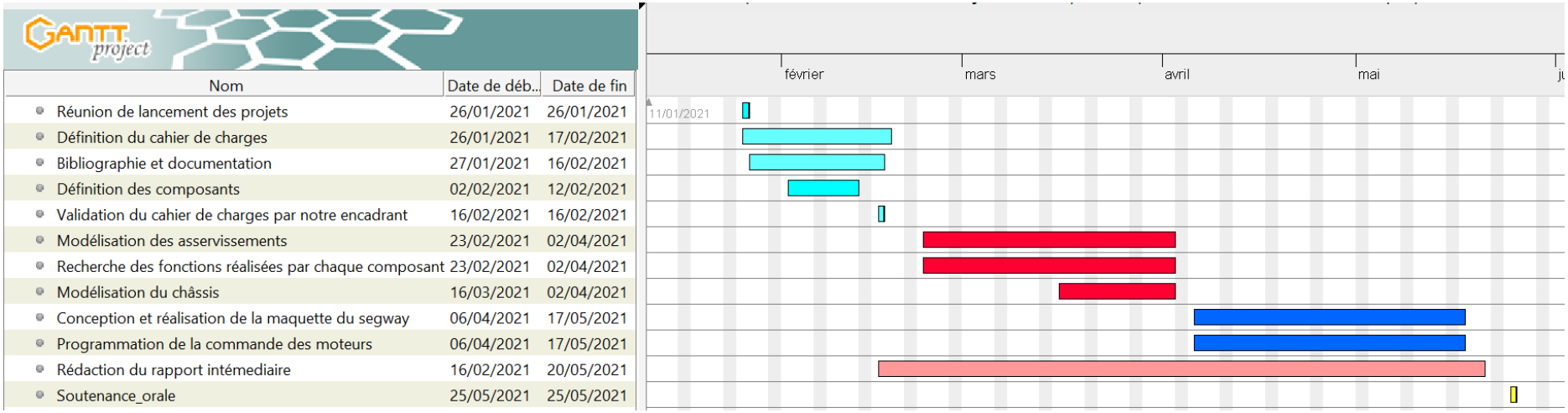
Résultats attendus au S6 :

- Établir un planning projet.
-Créer les différents schémas électriques.
-Proposer et déterminer les différents composants pour la réalisation de la maquette.
-Réalisation de la maquette.
-Rédaction d'un rapport intermédiaire incluant notre choix des composants, les différentes étapes pour la réalisation de la maquette et une estimation des coûts. (Voir ci-dessous notre diagramme de GANTT)

Résultats attendus au S7 :

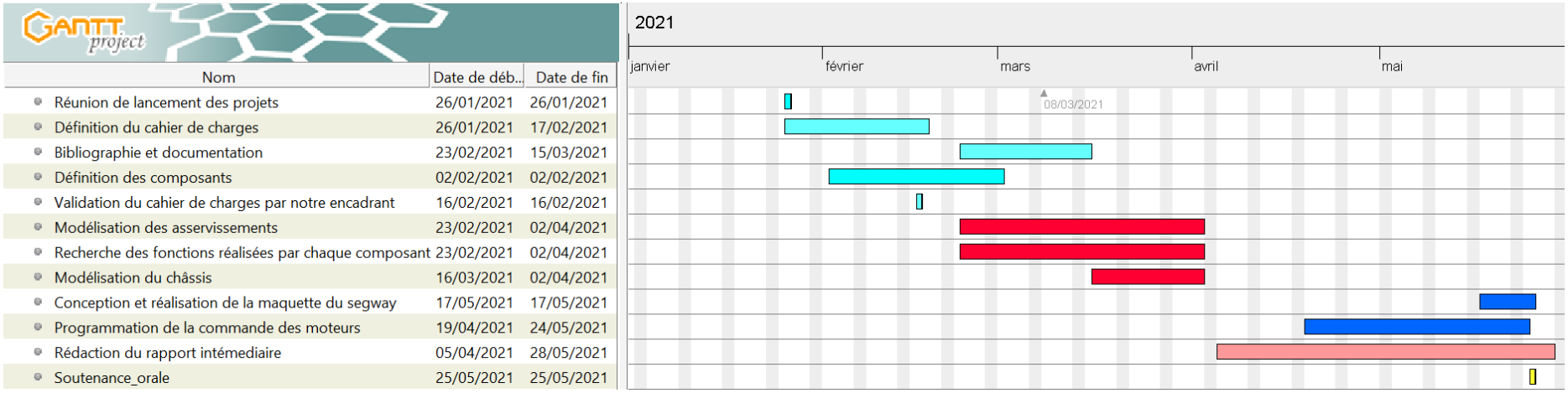
-Programmer le reste des lois de commande du Segway.
-Réaliser un programme de pilotage autonome du Segway à l'aide d'une caméra.
-Rédaction du rapport final incluant les résultats des essais et les conclusions.

Diagramme de GANTT du planning prévisionnel :



Nom	Date de déb...	Date de fin
Réunion de lancement des projets	26/01/2021	26/01/2021
Définition du cahier de charges	26/01/2021	17/02/2021
Bibliographie et documentation	27/01/2021	16/02/2021
Définition des composants	02/02/2021	12/02/2021
Validation du cahier de charges par notre encadrant	16/02/2021	16/02/2021
Modélisation des asservissements	23/02/2021	02/04/2021
Recherche des fonctions réalisées par chaque composant	23/02/2021	02/04/2021
Modélisation du châssis	16/03/2021	02/04/2021
Conception et réalisation de la maquette du segway	06/04/2021	17/05/2021
Programmation de la commande des moteurs	06/04/2021	17/05/2021
Rédaction du rapport intémediaire	16/02/2021	20/05/2021
Soutenance_orale	25/05/2021	25/05/2021

Planning réel :



Nom	Date de déb...	Date de fin
Réunion de lancement des projets	26/01/2021	26/01/2021
Définition du cahier de charges	26/01/2021	17/02/2021
Bibliographie et documentation	23/02/2021	15/03/2021
Définition des composants	02/02/2021	02/02/2021
Validation du cahier de charges par notre encadrant	16/02/2021	16/02/2021
Modélisation des asservissements	23/02/2021	02/04/2021
Recherche des fonctions réalisées par chaque composant	23/02/2021	02/04/2021
Modélisation du châssis	16/03/2021	02/04/2021
Conception et réalisation de la maquette du segway	17/05/2021	17/05/2021
Programmation de la commande des moteurs	19/04/2021	24/05/2021
Rédaction du rapport intémediaire	05/04/2021	28/05/2021
Soutenance_orale	25/05/2021	25/05/2021

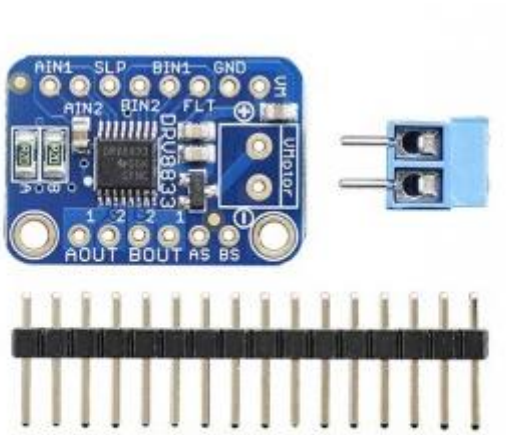
II. Définition des composants :

Après avoir réalisé une recherche bibliographique en se basant sur ce que notre tuteur nous a envoyé ainsi que sur notre propre recherche, nous avons défini nos composants. (Voir ci-dessous)

2 x moteurs pas à pas : Moteur 11HS18-0674S - Moteurs pas-à-pas | GO TRONIC

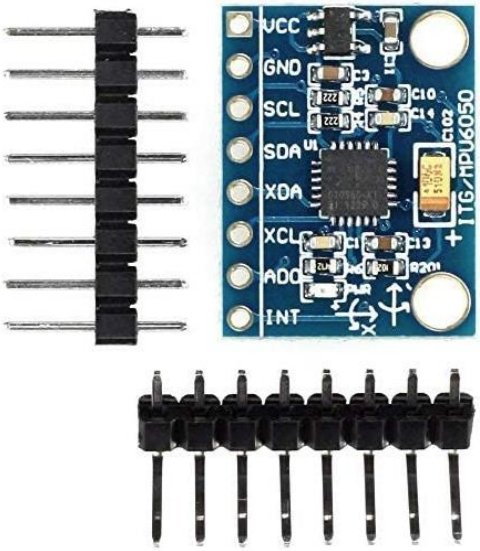
Kit Raspberry pi 0 w : PI-ZERO-U-K108 | Canakit Kit de démarrage Raspberry Pi Zero avec carte de 16 Go, 512 Mo

2 x drivers moteurs pas à pas : [Driver moteur pas-à-pas ADA3297 Adafruit - Commandes 2 moteurs cc | GO TRONIC]



2 x Roues : [PVO080-12LM44 | Roue de chariot Caoutchouc Noir 80mm x 35mm, alésage 12mm, longueur 44.4mm, Alésage lisse, Capacité de 70kg]

Inclinomètre : [Paradisetronic.com Module de Capteur MPU-6050, Accéléromètre à 3 Axes et Gyroscope à 3 Axes, Carte d'éclatement 6DOF, I2C, SPI, pour Arduino/Genuino, Raspberry Pi]



Nous avons fait le choix d'une raspberry pi pour notre microcontrôleur pour des raisons de familiarité et de grande capacité du côté des entrées/sorties, en effet une raspberry n'est rien d'autre qu'un ordinateur miniature avec des portes d'entrées/sorties programmables. Nous avons choisi spécifiquement le modèle de raspberry pi zero w pour son faible encombrement et sa faible consommation d'énergie.

En deuxième lieu, nous nous sommes penchés sur la motorisation du Segway. Nous avons décidé d'utiliser deux moteurs pas à pas car ils sont commandables en position comme en vitesse. Cette technologie nous permet d'incorporer plusieurs méthodes de stabilisation qui peuvent utiliser ces deux méthodes de commande.

Afin de réaliser la commande de ces moteurs, nous avons besoin d'utiliser des drivers spécifiques à la technologie pas à pas, car leur commande requiert des temporisations précises pour fonctionner correctement. Notre choix d'un module de chez Adafruit afin de pouvoir utiliser leurs multiples modules python qui nous faciliteront la tâche par la suite (cf VI. Programmation).

Ensuite, pour pouvoir stabiliser le Segway, nous avons besoin de connaître en tout temps l'angle qu'il fait avec la verticale (cf IV. Modélisation et analyse du Segway), pour ce faire nous avons choisis un module inclinomètre de chez Adafruit encore une fois pour les mêmes raisons que les drivers.

Finalement, pour ce qui est de la transmission de mouvement, nous entraîneront les deux roues en prise directe avec l'arbre du moteur via un adaptateur imprimé en 3D.

Pour ce qui est du châssis qui va accueillir tous ces composants, il sera également réalisé en impression 3D afin de pouvoir accommoder la fixation directe des composants qui seront alors à la fois bien positionnés et suffisamment robustes.

La conception et la réalisation des pièces 3D ont dû en revanche attendre la décision finale qui a suivi la réception des pièces pour s'assurer du bon montage de celles-ci.
Nous laissons donc cette étape pour le S7.

III. Schémas Électriques :

Il est à noter que le schéma électrique représente un choix antérieur pour ce qui est de l'inclinomètre, en effet à la réception de celui-ci, nous nous sommes rendu compte que son format était trop petit pour être incorporé dans le circuit à moins de réaliser un PCB par nous-même autour. Ce qui n'est pas, à l'heure d'aujourd'hui, à notre portée. Cela explique notre choix de passer sur le module présenté plus haut.

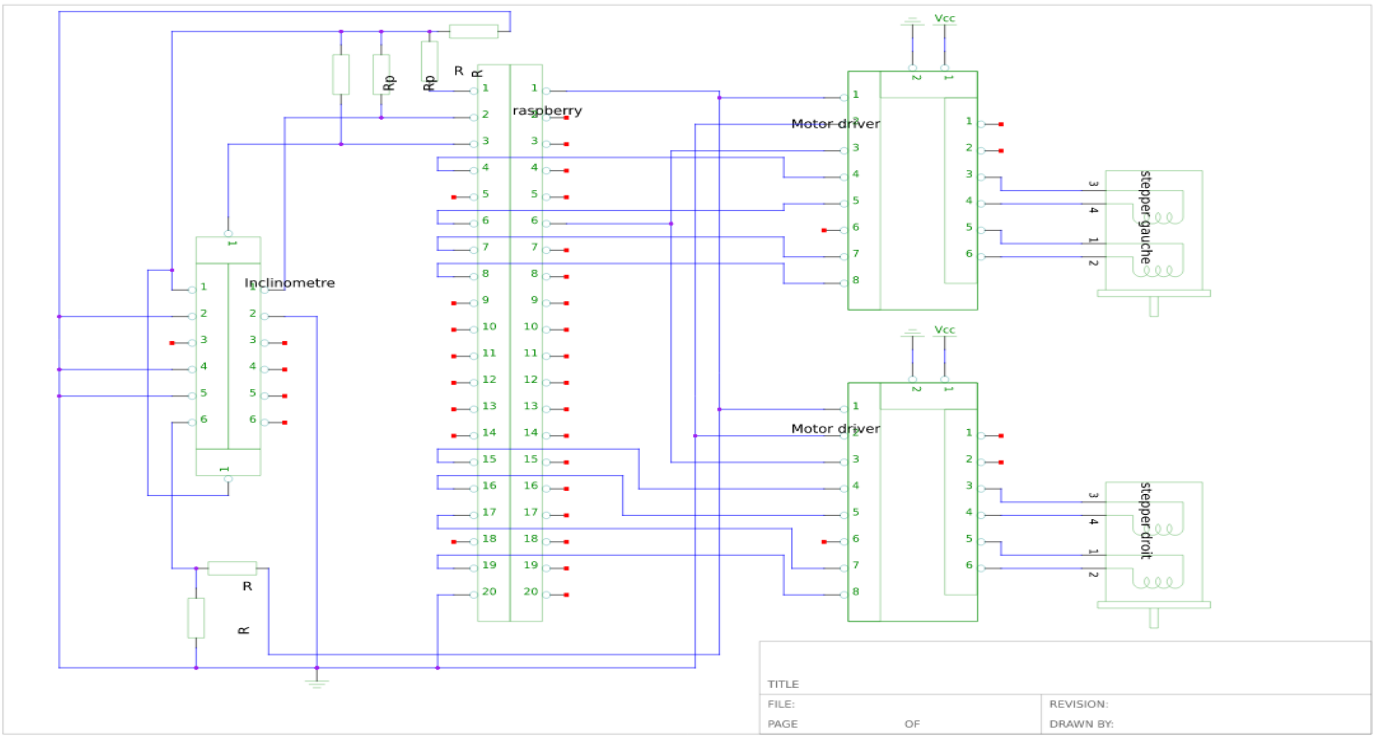
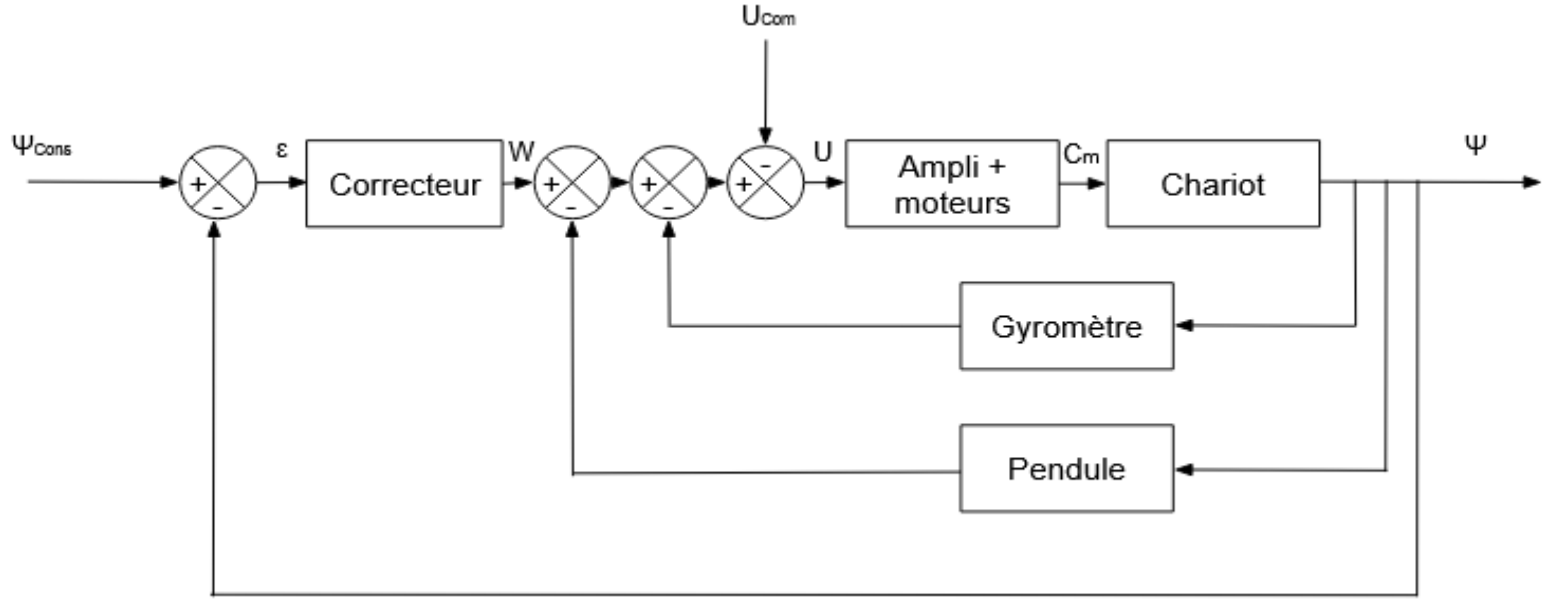


Schéma électrique

IV. Analyse et modélisation du Segway :

La régulation d'inclinaison du Segway consiste à maintenir la consigne$\psi_c(t)$ nulle. Cette régulation est réalisée si, quelle que soit l'inclinaison $\alpha(t)$ du conducteur, la sortie $\psi \ \ (t)$converge vers $\psi_c(t)$, valeur nulle ici. Le conducteur agit directement sur la valeur $\alpha(t)$ de pour accélérer ou décélérer. Pour le système Segway, conducteur exclu, le paramètre peut être considéré comme une perturbation.

Le schéma-bloc fonctionnel du Segway est donc le suivant :



Il est composé de différents éléments :

Un **correcteur**, pour réguler la commande

L'ensemble **ampli + moteurs** qui délivre un couple $C_m(p) = K.U(p)$

L'ensemble **chariot + conducteur** (le système mécanique) modélisé par l'équation $\chi(p) = \psi(p) - \alpha(p) = F(p)U(p) - \alpha(p)$ avec $F(p) = \frac{2(B+DR)/RDC}{((DA-B^2)/DC)*p^2-1}$, A=90kg.m², B=75kg.m, C=750kg.m²/s², D=125kg et R=240mm (ces valeurs seront expliquées dans la partie juste après)

Le **gyromètre** : $u_g(t) = k_g \frac{d\,\psi(t)}{dt}$ soit $U_g(p) = pk_g\psi(p)$

Le **pendule** : $u_p(t) = k_p\psi(t)$ soit $U_p(p) = k_p\psi(p)$

Nous avons choisi de prendre comme valeurs initiales pour le segway (l'ensemble conducteur + chariot) les valeurs fournies dans ces tableaux, accompagnées des matrices d'inerties correspondantes au système étudié:

Conducteur : *H*	
Centre de gravité G avec $\overrightarrow{AG} = h\vec{z}_4$ $h = 0,95$ m $m_H = 80$ kg	$I_G(H) = \begin{pmatrix} A_H & 0 & 0 \\ 0 & B_H & 0 \\ 0 & 0 & C_H \end{pmatrix}_{\vec{x}_4, \vec{y}_4, \vec{z}_4}$ $A_H = 18\ \text{kg} \cdot \text{m}^2$ $B_H = 20\ \text{kg} \cdot \text{m}^2$ $C_H = 1,2\ \text{kg} \cdot \text{m}^2$

Châssis : S	
Centre de gravité A $m_s = 25$ kg	$I_A(S) = \begin{pmatrix} A_S & 0 & 0 \\ 0 & B_S & 0 \\ 0 & 0 & C_S \end{pmatrix}_{\vec{x}_1, \vec{y}_2, \vec{z}_2}$ $A_S = 0,8 \text{ kg} \cdot \text{m}^2$ $B_S = 1,2 \text{ kg} \cdot \text{m}^2$ $C_S = 1,3 \text{ kg} \cdot \text{m}^2$
Roue gauche : R_G	
Centre de gravité O_G, avec $\overrightarrow{AO_G} = \frac{L}{2}\vec{x}_1$ Rayon $R = 240$ mm $L = 650$ mm $m_R = 5$ kg	$I_{O_G}(R_G) = \begin{pmatrix} A_R & 0 & 0 \\ 0 & B_R & 0 \\ 0 & 0 & C_R \end{pmatrix}_{\vec{x}_1, \vec{y}_G, \vec{z}_G}$ $A_R = 0,28 \text{ kg} \cdot \text{m}^2$ $B_R = 0,05 \text{ kg} \cdot \text{m}^2$ $C_R = 0,05 \text{ kg} \cdot \text{m}^2$
Roue droite : R_D	
Centre de gravité O_D, avec $\overrightarrow{O_DA} = \frac{L}{2}\vec{x}_1$ Rayon $R = 240$ mm $L = 650$ mm $m_R = 5$ kg	$I_{O_D}(R_D) = \begin{pmatrix} A_R & 0 & 0 \\ 0 & B_R & 0 \\ 0 & 0 & C_R \end{pmatrix}_{\vec{x}_1, \vec{y}_D, \vec{z}_D}$ $A_R = 0,28 \text{ kg} \cdot \text{m}^2$ $B_R = 0,05 \text{ kg} \cdot \text{m}^2$ $C_R = 0,05 \text{ kg} \cdot \text{m}^2$

La masse et l'inertie de la motorisation et de son réducteur associé sont supposées négligeables. Le moto-réducteur fait partie du châssis.

C’est ainsi que l’on trouve les valeurs de A, B, C, D et R (rayon d’une roue) pour l’ensemble du système mécanique modélisé par l’équation $\chi(p) = \psi(p) - \alpha(p) = F(p)U(p) - \alpha(p)$ avec $F(p) = \frac{2(B+DR)/RDC}{((DA-B^2)/DC)*p^2-1}$

On a en effet les équations suivantes:

$$\begin{cases} A\ddot{\Psi}(t) - B\dot{V}(t) = -2C_S(t) + C(\alpha(t) + \Psi(t)) \\ B\ddot{\Psi}(t) - D\dot{V}(t) = \dfrac{2C_S(t)}{R} \end{cases}$$

En prenant les hypothèses suivantes

$\sin(\alpha + \psi) = \alpha + \psi$

$\cos(\alpha + \psi) = 1$

et les termes en $\dot{\psi}^2$ négligeable devant ceux en $\dot{\psi}$

..

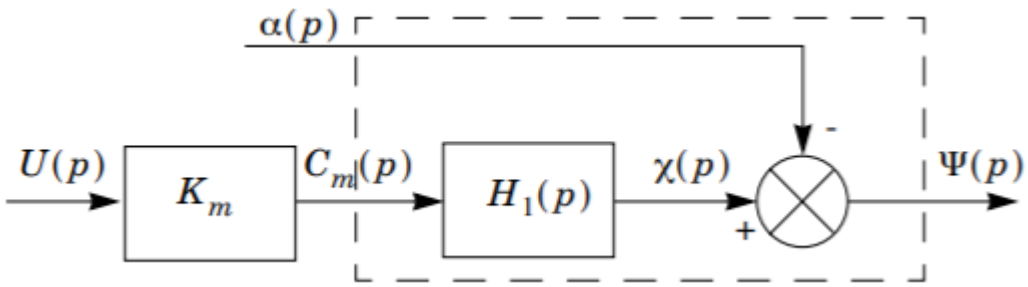
Puis, on en déduit :

$$A = A_H + A_S + m_H h^2 \quad B = m_H h \quad C = m_H hg$$

$$D = m_H + m_s + 2m_R + 2\frac{A_R}{R^2}$$

A.N. : $A = 18 + 0,8 + 80.0,95^2 = 91 kg.m^2 \quad B = 80.0,95 = 76 kg.m$

$C = 80.0,95.9,81 = 745,5\ N.m \quad D = 80 + 25 + 10 + 2.0,28 / 0,24^2 = 124,7 kg$



Nous étudions la fonction de transfert de : $F(p) = \frac{¥(p)}{U(p)}$

La régulation d'inclinaison du Segway® est réalisée par :

- un moto-réducteur qui permet de délivrer un couple $C_m(t) = K_m u(t)$ où $u(t)$ est une grandeur de commande et $K_m = 24\ \text{N} \cdot \text{m} \cdot V^{-1}$
- le système mécanique dont les équations ont été déterminées et qui peuvent, dans le cas où l'angle $\alpha(t)$ n'est pas supposé constant, se mettre sous la forme :

$$\begin{cases} \dot{V}(t) = \frac{1}{D}\left(B\ddot{\chi}(t) + 2\frac{C_m(t)}{R}\right) \\ (DA - B^2)\ddot{\chi}(t) = 2\left(\frac{B}{R} + D\right)C_m(t) + DC\chi(t) \end{cases} \quad \text{ou} \quad \begin{cases} A = 90\ \text{kg} \cdot \text{m}^2 \\ B = 75\ \text{kg} \cdot \text{m} \\ C = 750\ \text{kg} \cdot \text{m}^2\text{s}^{-2} \\ D = 125\ \text{kg} \\ R = 240\ \text{mm} \\ \chi(t) = \alpha(t) + \Psi(t) \end{cases}$$

Par commodité de signe, la notation $C_m(t) = -C_S(t)$ est utilisée dans les équations ci-dessus. Les conditions initiales sont toutes nulles.

Les conditions sont toutes nulles, nous obtenons à partir la seconde équation différentielle dans Laplace(ci-dessus), l'équation suivante :

$$[(DA-B^2)p^2-DC]\chi(p)=2(\frac{B}{R}+D)C_m(p)$$

Nous avons $H_1(p) =\frac{\chi(p)}{C_m(p)}$ d'après le schéma bloc ci-dessous.

Nous en déduisons : $$H_1(p)=\frac{2\left(\frac{B}{R}+D\right)C}{(DA-B^2)p^2-DC}=\frac{\frac{2\left(\frac{B}{R}+D\right)C}{DC}}{\frac{DA-B^2}{DC}p^2-1}=\frac{2\left(\frac{B+DR}{RDC}\right)}{\frac{DA-B^2}{DC}p^2-1}$$

Le schéma bloc se justifie alors en écrivant :

$$\psi(p)= \chi(p)= K_mH_1(p)U(p)$$

Le système d'entrée u (t) et de sortie ψ(t) est instable car le dénominateur de la fonction de transfert possède un pôle réel positif.

$$p=\pm\sqrt{\frac{DC}{DA-B^2}} \quad \text{avec } \frac{DA-B^2}{DC} = 0{,}06 > 0$$

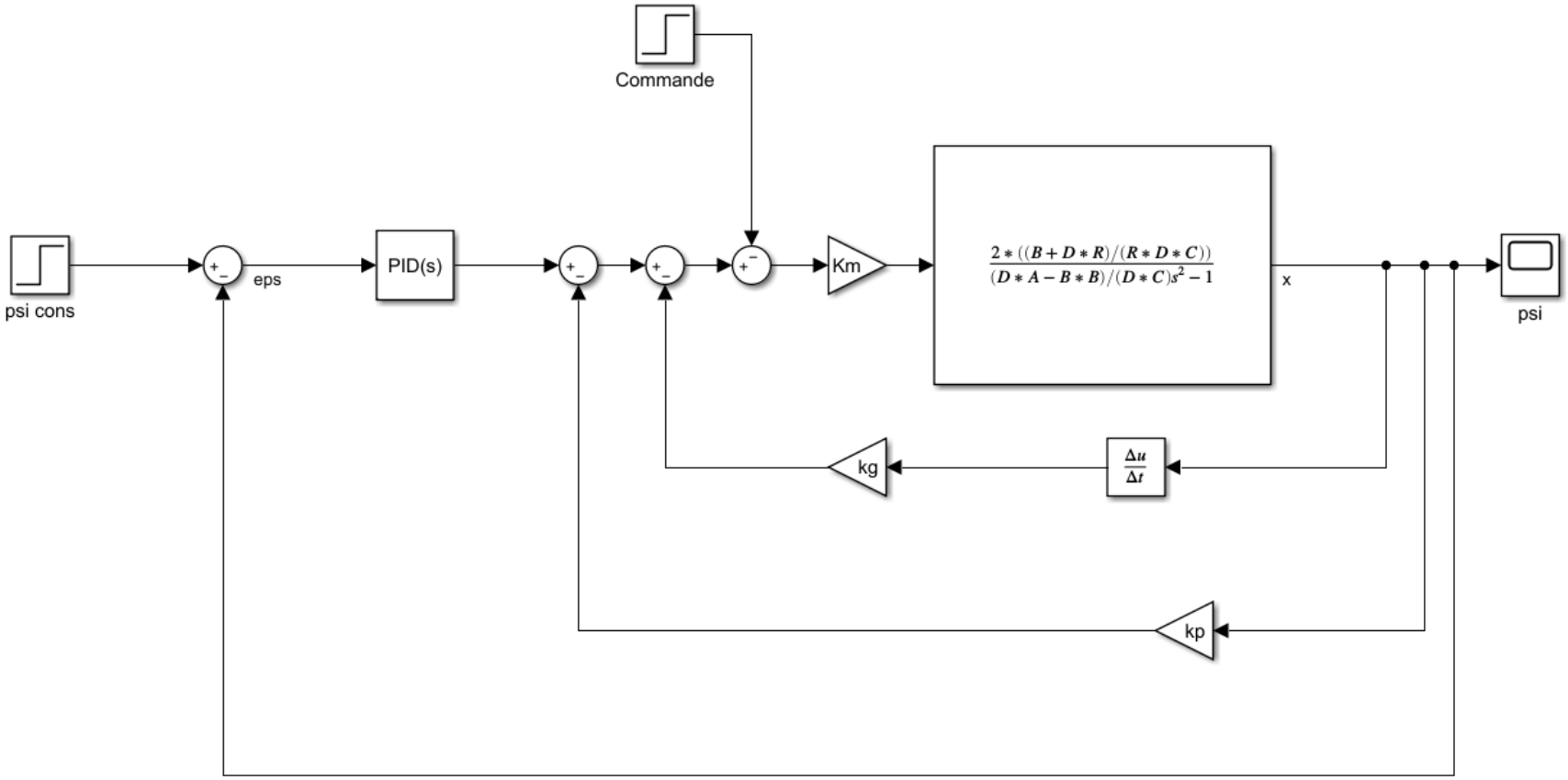
Nous déterminons les conditions sur K_v et K_p pour que le système soit stable cette fois-ci avec α=0 (Voir schéma ci-dessous)

$$\frac{\psi(p)}{W(p)} = \frac{\frac{K_mH_1(p)}{1+pK_vK_mH_1(p)}}{1+K_p\frac{K_mH_1(p)}{1+pK_vK_mH_1(p)}} = \frac{K_mH_1(p)}{1+pK_vK_mH_1(p)+K_pK_mH_1(p)} = \frac{K_s}{\frac{p^2}{\omega^2}+pK_vK_s+K_pK_s-1}$$

C'est un système de degré 2 qui est stable si et seulement si les coefficients des puissances de p sont différents de 0 et sont de même signe.

Ceci implique : $K_p > \frac{1}{K_s}$ et $k_v > 0$ pour que le système soit stable.

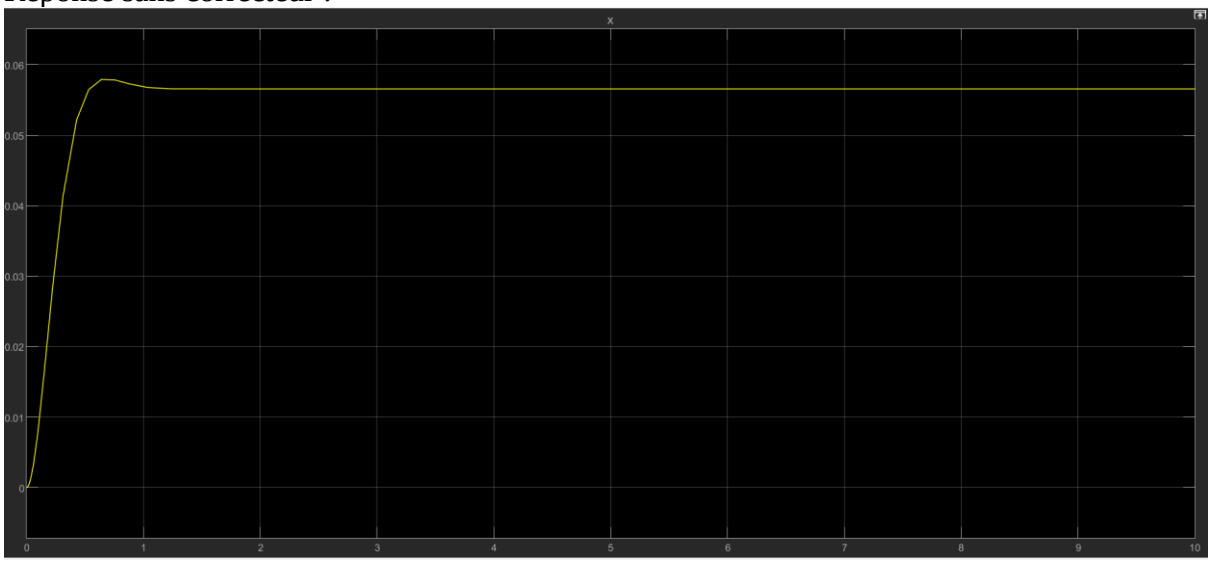
On a alors la modélisation matlab suivante:



V. Simulation :

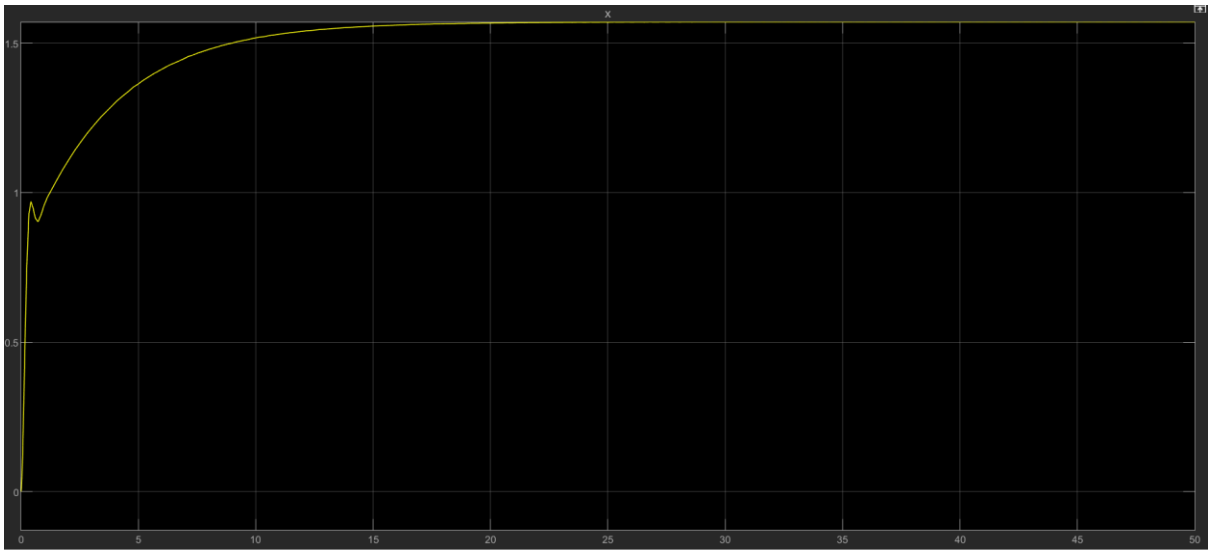
Nous allons maintenant vérifier le comportement de notre Segway virtuel afin de nous assurer de son bon fonctionnement. Nous allons pour cela analyser sa réponse pour une consigne d'angle de 1.57 rad (soit un angle de 90°), puis essayer de la corriger avec un régulateur.

Réponse sans correcteur :



Nous pouvons voir que l'asservissement via les différentes boucles de retour permet de stabiliser le système, cependant nous pouvons observer une erreur statique conséquente ($1.57 - 0.06 = 1.51$). Nous avons donc besoin d'un régulateur comportant au moins une composante intégratrice.

P=10 et I= 5

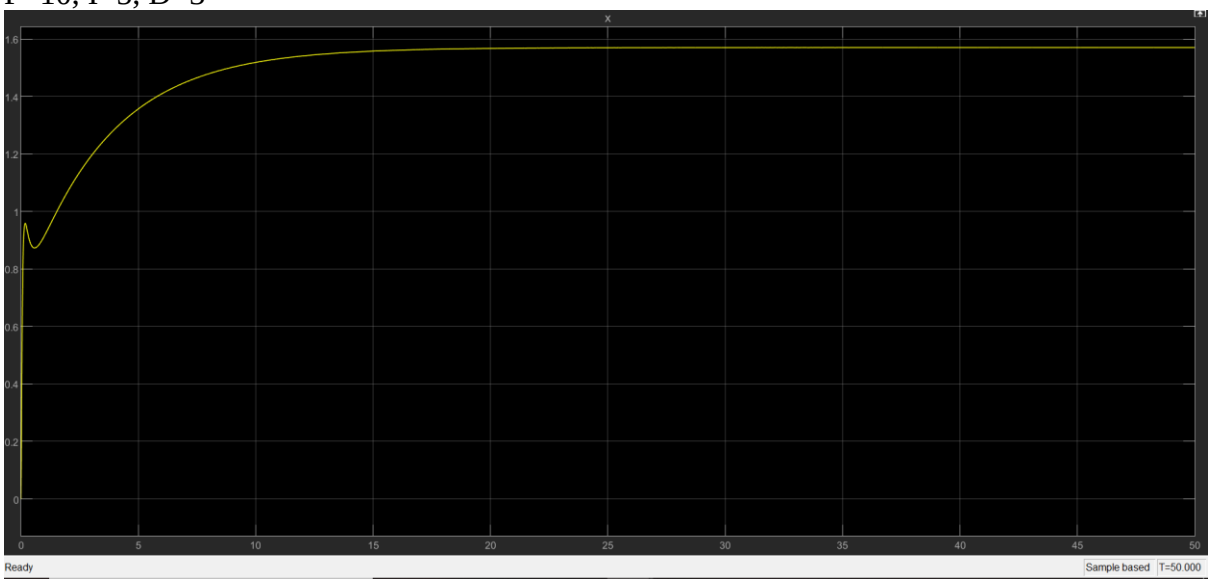


Ainsi, en ajoutant un correcteur PI, nous pouvons constater que l'erreur est devenue nulle. Le régulateur a donc permis de corriger la précision de notre asservissement, mais a grandement augmenté son temps de réponse ($t_{r,5\%} \simeq 10\ s$).

P=0 et I=18.15 (optimisé par Matlab)

En utilisant l'outil *tune* sur Matlab Simulink, permettant de calculer les valeurs optimales pour les différentes actions du régulateur, on obtient un résultat beaucoup plus rapide ($t_{r,5\%} \simeq 2\ s$) et qui possède un léger dépassement (1.63 rad), ce qui n'est pas gênant dans notre application.

P=10, I=5, D=3



Nous allons maintenant observer l'influence de l'action Dérivée en utilisant un régulateur PID. Nous pouvons voir que celle-ci ne semble pas modifier la stabilité ni la précision.

P=125, I=323 et D=10 (optimisé par Matlab)

En utilisant l'outil *tune* sur Matlab Simulink, on observe toujours une hausse de la rapidité ainsi que l'apparition d'un léger dépassement. Le dépassement est plus faible que pour le correcteur PI (1.6 rad), et le temps de réponse également ($t_{r,5\%} \simeq 1\ s$). Les performances de notre système sont donc meilleures avec ce dernier régulateur.

VI. Programmation :

Notre choix d'utiliser une raspberry nous amène ici à choisir le langage python pour réaliser le code, de plus les librairies fournies pour les différents modules Adafruit que l'on a choisis vont s'avérer utiles.

a) Structure du code :

Pour aider à la lecture et à l'utilisation du code, nous avons organisé le programme en classes et méthodes, mettant ainsi le python à notre avantage en tant que langage orienté objet.
La réalisation spécifique de ces classes est disponible dans l'annexe 1.

Par ailleurs, nous avons rassemblé les paramètres physiques et algorithmiques dans un fichier pinout.py (annexe 2) ce qui nous permet d'accéder simplement aux paramètres qui sont susceptibles de changer au cours de la conception.

Finalement, la réalisation de l'algorithme de commande en lui-même se fera lui aussi dans un fichier à part qui n'as pas encore été réalisé, étant donné la nature encore incertaine du reste du programme. Mais des perspectives quant à son implémentation sont discutées en **d)**.

Il est à noter que la classe qui gère l'inclinomètre est ici manquante, en effet le changement de composant décrit en **III**. ne nous a pas laissé le temps de faire les modifications, le code pour le modèle précédent a été laissé de côté, ayant été rendu inutile.

Il est également à noter qu'au moment de la fin du S6, seule la commande des moteurs a pu être testée et prouvée opérationnelle.

b) Commande des moteurs :

Comme dit en **I.** nous utilisons ici la bibliothèque "Stepper" distribuée par Adafruit et qui nous apporte la méthode "onestep" qui va nous permettre d'avoir une boucle dans laquelle on fait avancer les deux moteurs l'un après l'autre à chaque tour de boucle et nous permettent aussi de les faire avancer ou reculer chacun d'un nombre indépendant de pas. Il sera par la suite aisé de passer d'une consigne d'angle à une commande en pas en utilisant le nombre connu de pas par tour des moteurs.

c) Mesures de sécurité :

Les drivers pas à pas que nous avons choisis ont une sortie nommée *FLT* qui correspond à une sécurité qui déclenche un front lorsque le courant devient trop grand. Nous avons donc pu utiliser ce pin afin de réaliser une procédure de sécurité qui désactive tous les pins GPIO de la raspberry, la protégeant ainsi de potentielles dommages.

Cette procédure utilise le concept natif de python d'Exception et la méthode *add_event_detect* de GPIO qui nous permet de récupérer une interruption lorsqu'il y a front sur le pin sélectionné. La procédure renvoie alors une erreur via *exception* afin de prévenir l'utilisateur du problème.

d) Méthodes de commande :

Dans un premier temps, la priorité est la stabilisation autonome du Segway afin qu'il puisse rester vertical. Mais il est prévu une fois cet objectif atteint, de permettre à l'utilisateur de contrôler le Segway via une manette qui lui permettrait de le faire se déplacer en plus de garder la stabilisation.

Pour ce qui est de la réalisation de la stabilisation, nous avons à notre disposition plusieurs méthodes :

Une première approche naïve serait de mesurer à chaque pas de temps la position de l'angle de balancement et d'envoyer à ce moment le même angle sur les moteurs mais dans le sens opposé pour contrer l'action de la gravité.

Une seconde approche plus avancée serait d'utiliser un PID pour réaliser la régulation d'angle, nous avons pu d'ores et déjà récupérer un module de PID sur github (annexe 3) qui nous aidera grandement dans la réalisation de cette approche.

VII. Conclusion:

Ce projet de conception et commande d'un Segway nous a permis de comprendre le fonctionnement d'un gyropode à savoir la partie électrique et mécanique de celui-ci. Il nous a fallu faire beaucoup de recherches afin de trouver les outils nécessaires pour entamer notre projet qui consistait à faire la conception du Segway en définissant les composants, à le modéliser et le programmer.

Pour ce semestre 6, nous avons réussi à faire tourner les deux moteurs pas à pas. Et pour le semestre 7, notre objectif sera de le réaliser avec toutes ses fonctionnalités.

Pour finir ce projet nous a permis d'évoluer et de nous donner un aperçu de ce qui pourrait être un travail au sein d'une entreprise. Les compétences, l'expérience et les connaissances acquises vont nous être utiles dans vos futures vies professionnelles.

Annexes :

Annexe 1: Classes

```
import board
import digitalio
import smbus
from adafruit_motor import stepper
#import PID.py
import RPi.GPIO as GPIO
import time

GPIO_edge = {"r": GPIO.RISING, "f": GPIO.FALLING}

class moteur():
        delay = 0.001
        def __init__(self, coils, n_steps, delay):
                for coil in coils:
                        coil.direction = digitalio.Direction.OUTPUT
                        self.controleur = stepper.StepperMotor(     coils[0], coils[1], coils[2], coils[3],
                                                                    microsteps=None)
                self.n_steps = n_steps

        @staticmethod
        def spinDual(moteurL, moteurR, stepL, stepR) :
                l, r = 0, 0
                dirL = stepper.FORWARD if stepL > 0 else stepper.BACKWARD
                dirR = stepper.FORWARD if stepR > 0 else stepper.BACKWARD
                stepL, stepR = abs(stepL), abs(stepR)
                while (l <= stepL) and (r <= stepR) :
                        if r <= stepR :
                                moteurR.controleur.onestep(direction = dirR)
                                r+=1
                        if l <= stepL :
                                moteurL.controleur.onestep(direction = dirL)
                                l+=1
                        time.sleep(moteur.delay)

        def deg_to_steps(self, alpha) :
                return int(alpha/360*self.n_steps)

class security_exception(Exception):
        def __init__(self, message="Surtension du controler\n"):
                self.message = message
                super().__init__(self.message)

class security_checks():
        def __init__(self, pin, edgeType, exceptionMessage):
                self.pin = pin
                self.exceptionMessage = exceptionMessage
                GPIO.setup(self.pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
                GPIO.add_event_detect(self.pin, edgeType, callback = self.exception_handler,
```

```
                                                bouncetime=50)

        def exception_handler(self):
                GPIO.cleanup()
                raise security_exception(self.exceptionMessage)

class segway():
    def __init__(self, moteurL, moteurR, inclinometre, timestep, safety):
        self.moteurL = moteurL
        self.moteruR = moteruR
        self.inclinometre = inclinometre
        self.timestep = timestep
#       self.PID = PID()
        segway.PID.setSampleTime(timestep)
        self.safeties = [security_checks(test[0], GPIO_edge[test[1]], test[2]) for test in safety]
```

Annexe 2: pinout

```
import board
import digitalio

coilsL =    (
                digitalio.DigitalInOut(board.D22), #A1
                digitalio.DigitalInOut(board.D27), #A2
                digitalio.DigitalInOut(board.D4), #B1
                digitalio.DigitalInOut(board.D17), #B2
                )

coilsR =    (
                digitalio.DigitalInOut(board.D26), #A1
                digitalio.DigitalInOut(board.D13), #A2
                digitalio.DigitalInOut(board.D5), #B1
                digitalio.DigitalInOut(board.D6), #B2
                )

fault_IRQ = [(18, "r", "Sur tension des moteurs")]

NSTEPS = 200
DELAY = 0.0001
timestep = 0.1

P = 0.2
I = 0
D = 0
offset_inclinometre = 0
```

Annexe 3: PID

```
#!/usr/bin/python
#
# This file is part of IvPID.
# Copyright (C) 2015 Ivmech Mechatronics Ltd. <bilgi@ivmech.com>
#
# IvPID is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# IvPID is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

# title             :PID.py
# description       :python pid controller
# author            :Caner Durmusoglu
# date              :20151218
# version           :0.1
# notes             :
# python_version    :2.7

# ==============================================================================

"""Ivmech PID Controller is simple implementation of a Proportional-Integral-Derivative (PID) Controller in the Python Programming Language.
More information about PID Controller: http://en.wikipedia.org/wiki/PID_controller
"""
import time

class PID:
        """PID Controller
        """

        def __init__(self, P=0.2, I=0.0, D=0.0, current_time=None):

        self.Kp = P
        self.Ki = I
        self.Kd = D
```

```
self.sample_time = 0.00
self.current_time = current_time if current_time is not None else time.time()
self.last_time = self.current_time

self.clear()

def clear(self):
"""Clears PID computations and coefficients"""
self.SetPoint = 0.0

self.PTerm = 0.0
self.ITerm = 0.0
self.DTerm = 0.0
self.last_error = 0.0

# Windup Guard
self.int_error = 0.0
self.windup_guard = 20.0

self.output = 0.0

def update(self, feedback_value, current_time=None):
"""Calculates PID value for given reference feedback

.. math::
u(t) = K_p e(t) + K_i \int_{0}^{t} e(t)dt + K_d {de}/{dt}

.. figure:: images/pid_1.png
:align:   center

Test PID with Kp=1.2, Ki=1, Kd=0.001 (test_pid.py)

"""
error = self.SetPoint - feedback_value

self.current_time = current_time if current_time is not None else time.time()
delta_time = self.current_time - self.last_time
delta_error = error - self.last_error

if (delta_time >= self.sample_time):
self.PTerm = self.Kp * error
self.ITerm += error * delta_time

if (self.ITerm < -self.windup_guard):
self.ITerm = -self.windup_guard
elif (self.ITerm > self.windup_guard):
self.ITerm = self.windup_guard
```

```
self.DTerm = 0.0
if delta_time > 0:
self.DTerm = delta_error / delta_time

# Remember last time and last error for next calculation
self.last_time = self.current_time
self.last_error = error

self.output = self.PTerm + (self.Ki * self.ITerm) + (self.Kd * self.DTerm)

def setKp(self, proportional_gain):
"""Determines how aggressively the PID reacts to the current error with setting Proportional Gain"""
self.Kp = proportional_gain

def setKi(self, integral_gain):
"""Determines how aggressively the PID reacts to the current error with setting Integral Gain"""
self.Ki = integral_gain

def setKd(self, derivative_gain):
"""Determines how aggressively the PID reacts to the current error with setting Derivative Gain"""
self.Kd = derivative_gain

def setWindup(self, windup):
"""Integral windup, also known as integrator windup or reset windup,
refers to the situation in a PID feedback controller where
a large change in setpoint occurs (say a positive change)
and the integral terms accumulates a significant error
during the rise (windup), thus overshooting and continuing
to increase as this accumunlated error is unwound
(offset by errors in the other direction).
The specific problem is the excess overshooting.
"""
self.windup_guard = windup

def setSampleTime(self, sample_time):
"""PID that should be updated at a regular interval.
Based on a pre-determined sampe time, the PID decides if it should compute or return immediately.
"""
self.sample_time = sample_time
```