Be chaîne d'acquisition et commande numérique

Projet : Asservissement de courant d'une trottinette.

Rapport fait par : Anka Soubaai Abdelmajid et Housni Lamiaa

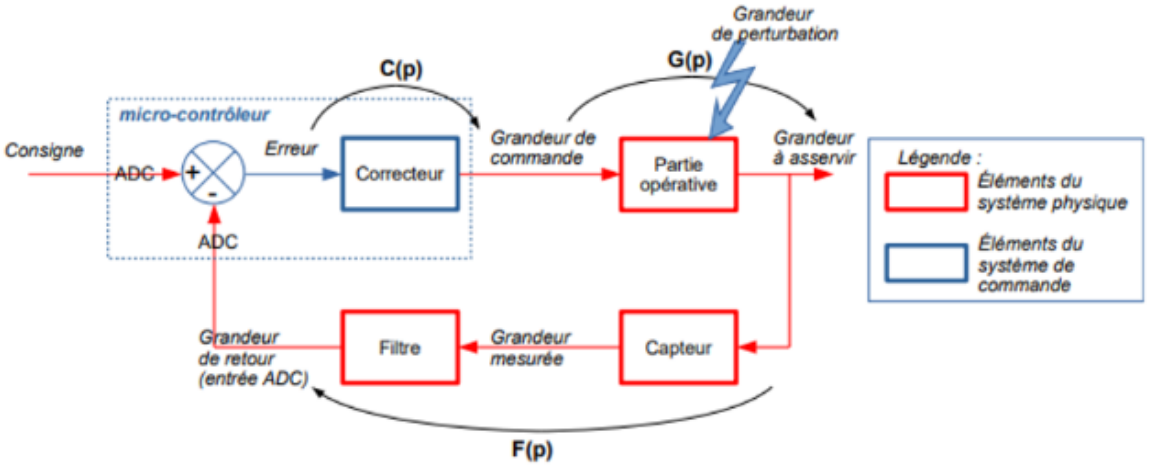
Sommaire :

I. Introduction :

Dans le cadre de ce bureau d'étude, nous allons travailler sur la commande d'une trottinette électrique commandée par deux commandes, soit par asservissement en vitesse ou bien asservissement en courant.

En lisant les documents fournis, nous comprenons de suite qu'on s'intéresse ici à l'asservissement en courant. Nous comprenons aussi que le but final est de mettre en place un correcteur pour commander le moteur de la trottinette à partir de l'erreur entre la consigne fournis et le résultat mesurée par le capteur de courant.

Pour ne pas parler dans l'abstrait, la figure ci-dessous montre le schéma fonctionnel de notre asservissement (les blocs détaillés seront présentés dans la partie II de ce compte rendu).



N.B :

Le correcteur qu'on va étudier ici est un correcteur numérique qui sera implanté directement dans le STM32 de la carte de puissance.

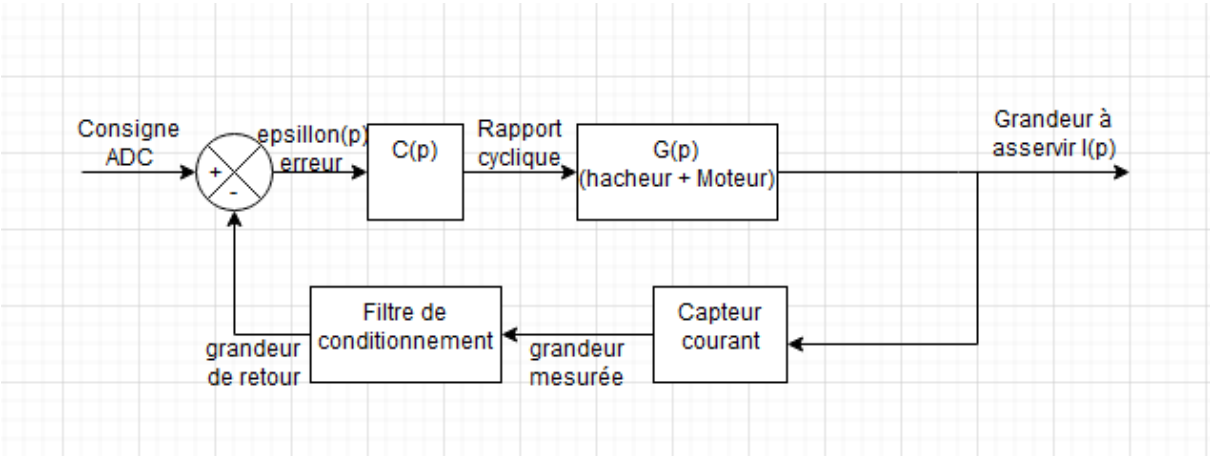
Guide de la méthode utilisé pour l'asservissement :

1. Identification de la fonction de transfert qui donne la grandeur à asservir à partir de la grandeur de commande.
2. Identification de la fonction de transfert qui donne la grandeur mesurée et filtrée entrée dans l'ADC en fonction de la grandeur à asservir.
3. Synthèse du correcteur à la main en respectant les spécifications du système C(p), et détermination de C(z) par l'approximation de la transformée bilinéaire.
4. Simulation Matlab afin de confirmer les calculs et les affiner.
5. Test du correcteur PI par simulation sous KEIL
6. Implémentation de ce qu'on a fait sur la trottinette et test

Les différentes grandeurs physiques de la trottinette vont être rappelées au fur et à mesure dans les différentes sous parties. Pour plus d'informations, veuillez se référer au dossier de la trottinette ainsi que les autres ressources présentes sur Moodle.

II. Schéma blocs

Le schéma bloc du système détaillé tracé sur draw.io est le suivant :



Légende :

- **G(p) :** est la fonction de transfert qui donne la grandeur à asservir à partir de la grandeur de commande.
- **F(p) :** est la fonction de transfert du filtre de conditionnement.
- **C(p) :** Il s'agit du correcteur qu'on réalisera. Il s'insère entre la consigne, la grandeur de retour, et la commande du système.

N.B : On retracera le schéma blocs après chaque modification faite, notamment pour réduire la chaine de retour.

III. Fonctions de transfert des différents blocs

1. Moteur + hacheur :

Le modèle électrique du moteur à courant continu est le suivant :

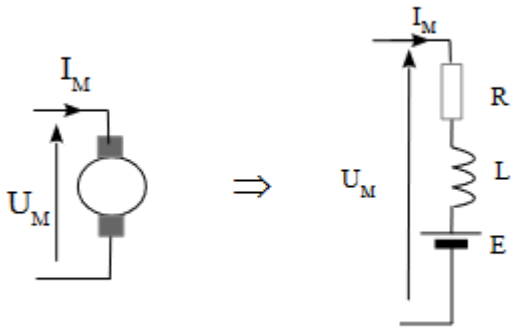
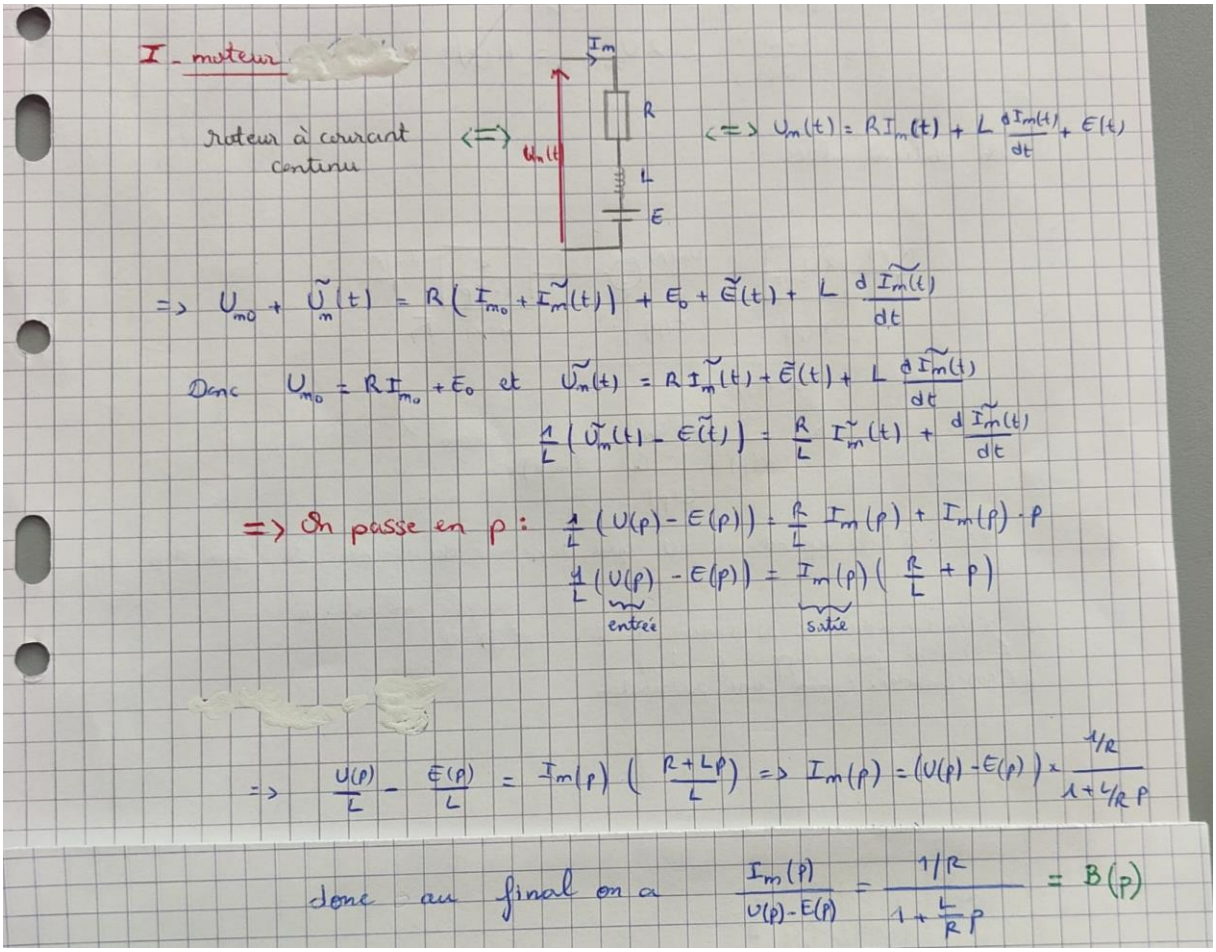
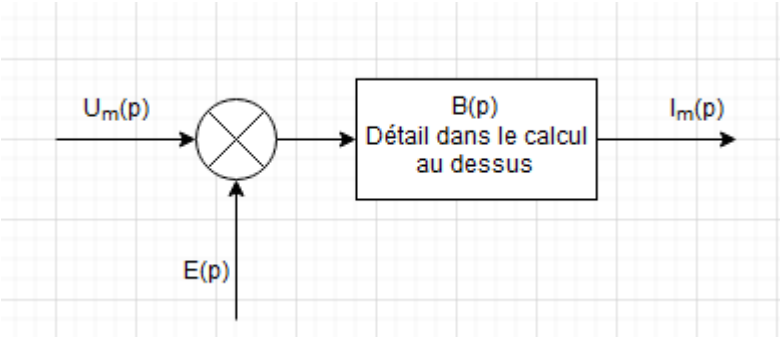


***Figure 6** : Modèle de la MCC*

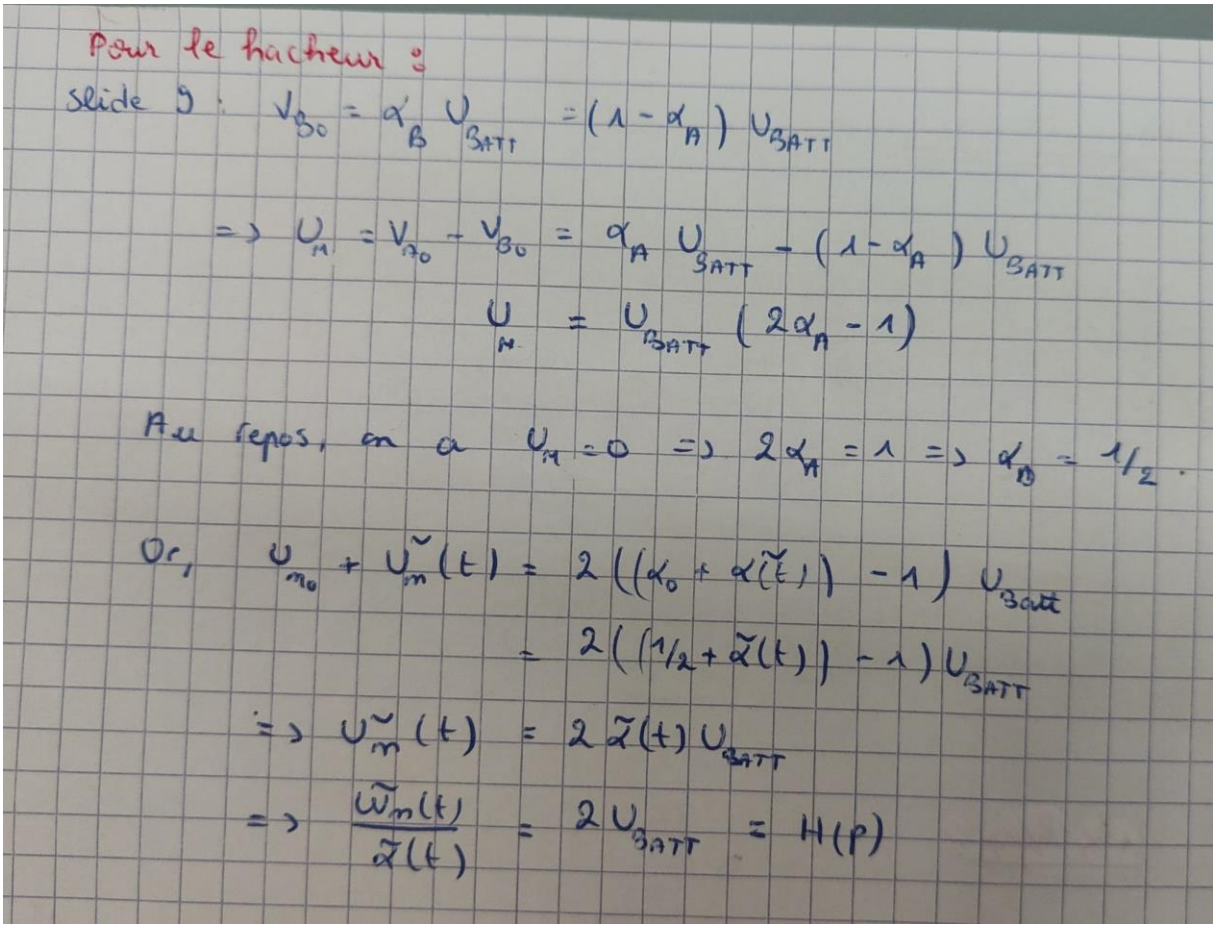
Calcul pour trouver la fonction du transfert du moteur :



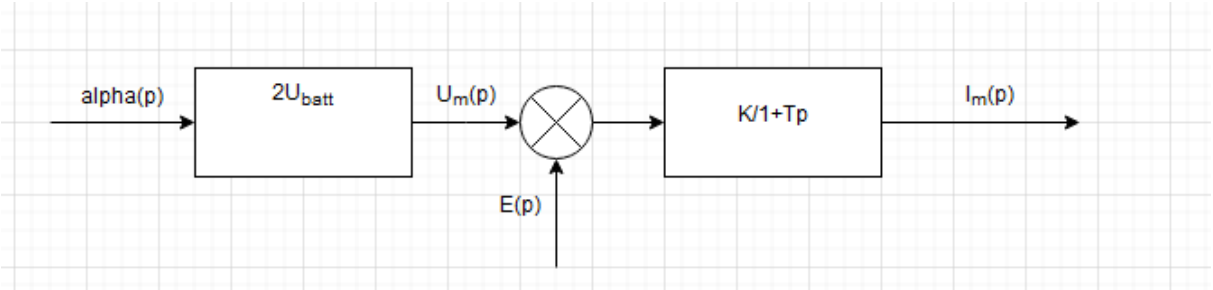
Le schéma bloc qui caractérise la fonction de transfert du moteur est le suivant :



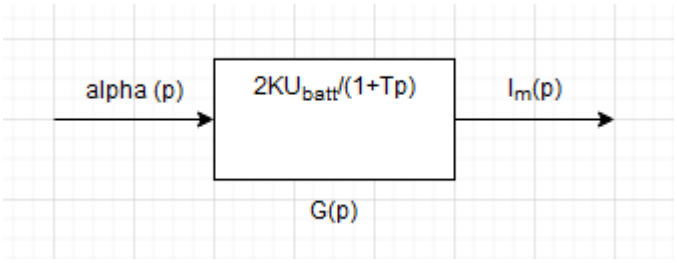
En ce qui concerne le hacheur, en s'appuyant sur la slide 9 du PDF sur l'introduction des hacheurs on a :



Le schéma bloc du hacheur est le suivant :



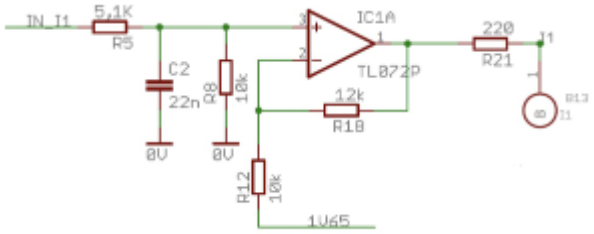
Qui peut se réduire en :



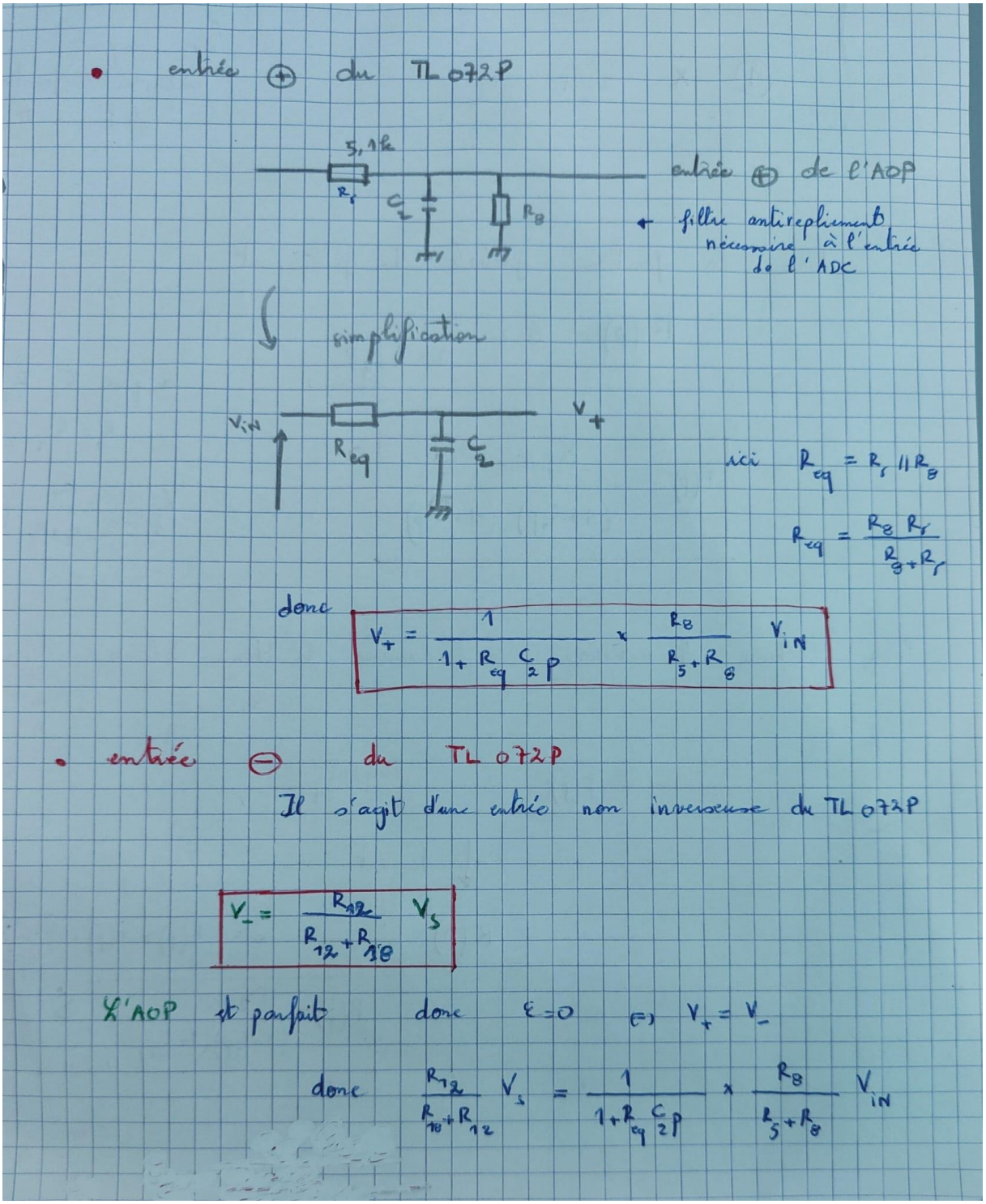
On trouve donc que la fonction de transfert du moteur + hacheur est : $$G(p) = \frac{2U_{batt} \cdot K_0}{1 + \tau_0 \cdot p}$$

2. Filtre :

En ce qui concerne le filtre on se confie au schéma suivant :



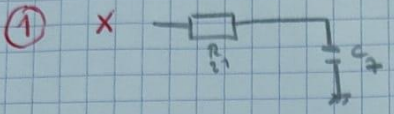
Le calcul de la fonction de transfert de ce filtre est comme suite :



donc ① $\frac{V_s}{V_{in}} = \frac{1}{(1 + R_{eq} C_2 p)} \times \frac{R_g}{R_f + R_g} \times \left(1 + \frac{R_{18}}{R_{12}}\right)$

$R_{eq} = \frac{R_g R_f}{R_g + R_f}$

Après l'AOP on a un filtre cascadé avec adaptation de l'ennemi donc on peut multiplier les fonctions de transfert

① × 

donc la fonction globale du filtre est

$F(p) = ① \times \frac{1}{1 + R_{21} C_1 p}$

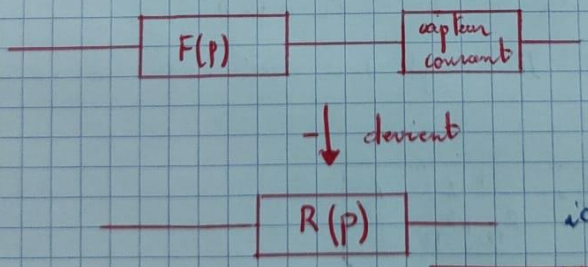
on pose $\tau_1 = R_{21} C_1$

$\tau_2 = R_{eq} C_2$

$K_{filtre} = \frac{R_g}{R_f + R_g}\left(1 + \frac{R_{18}}{R_{12}}\right)$

$$F(p) = \frac{K_{filtre}}{(1 + \tau_1 p)(1 + \tau_2 p)}$$

Pour réduire le retour on associe les deux blocs



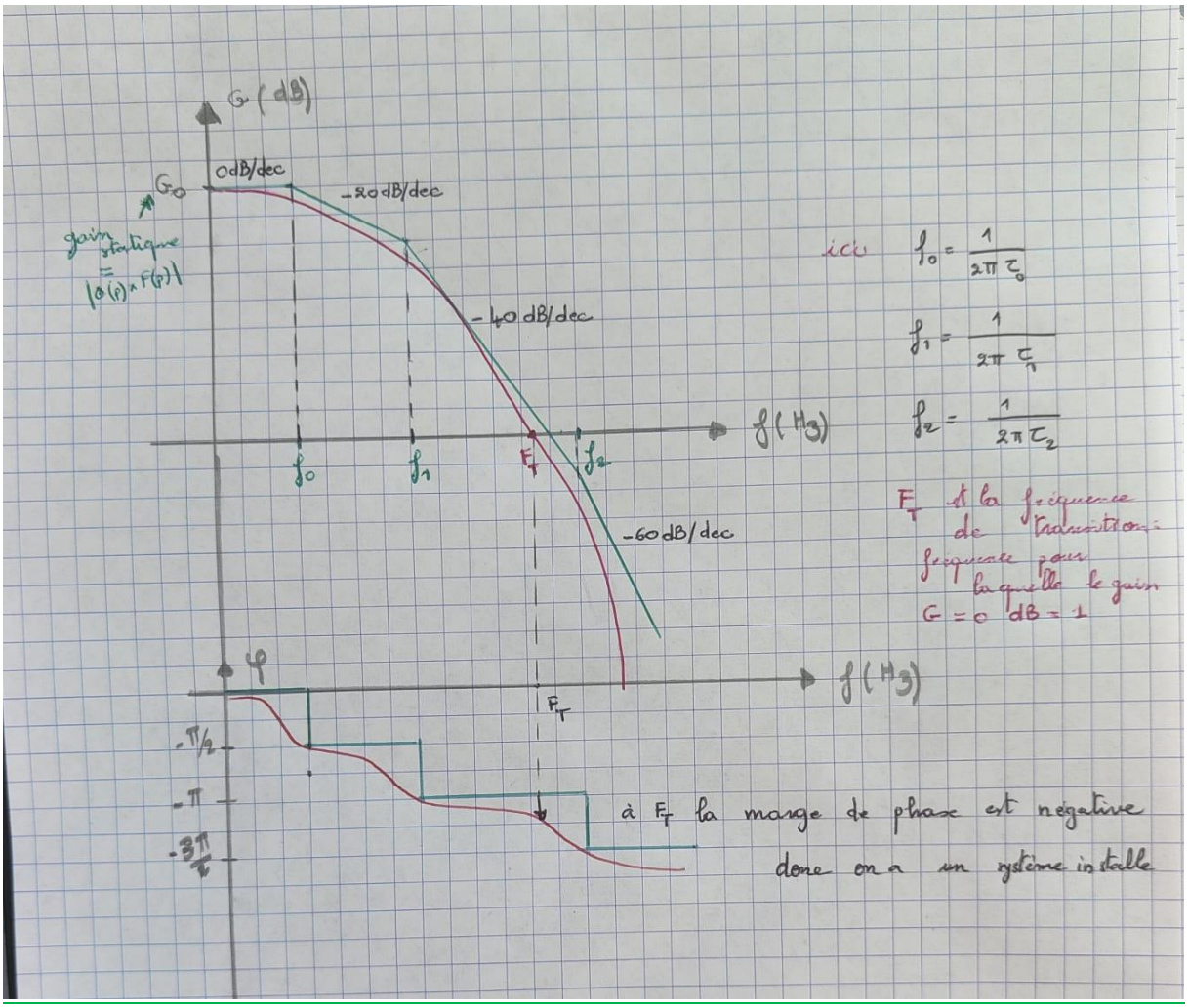
devient

ici $R(p) = F(p) \times$ sensibilité capteur

$$R(p) = \frac{K_{filtre}}{(1 + \tau_1 p)(1 + \tau_2 p)} \times S$$

IV. Correcteur :

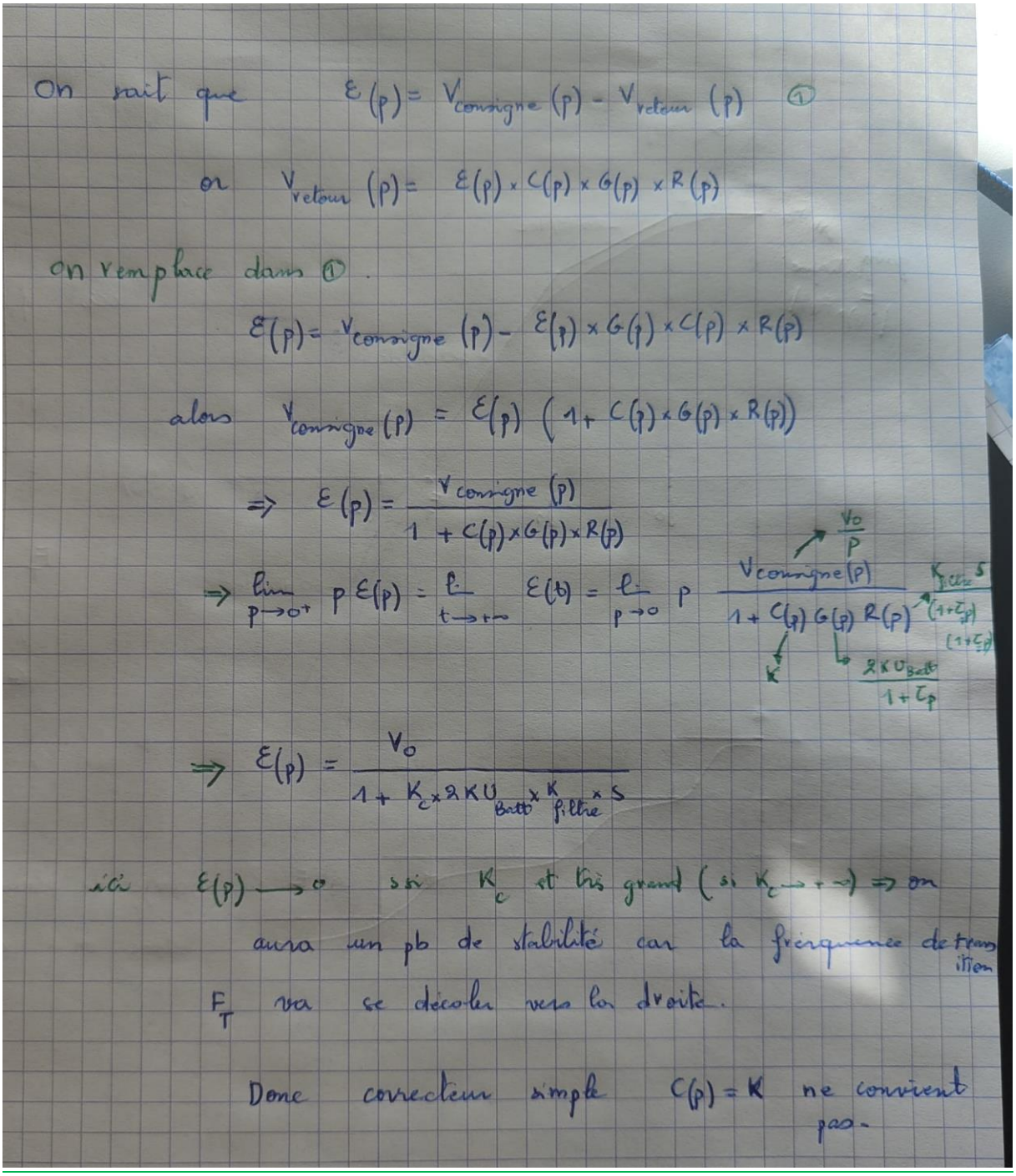
Pour étudier la stabilité de notre système on trace le diagramme de bode de la fonction de transfert du système complet sans correcteur (c'est-à-dire **G(p)*F(p)**).



Vu que le système est instable on décide d'utiliser un correcteur.

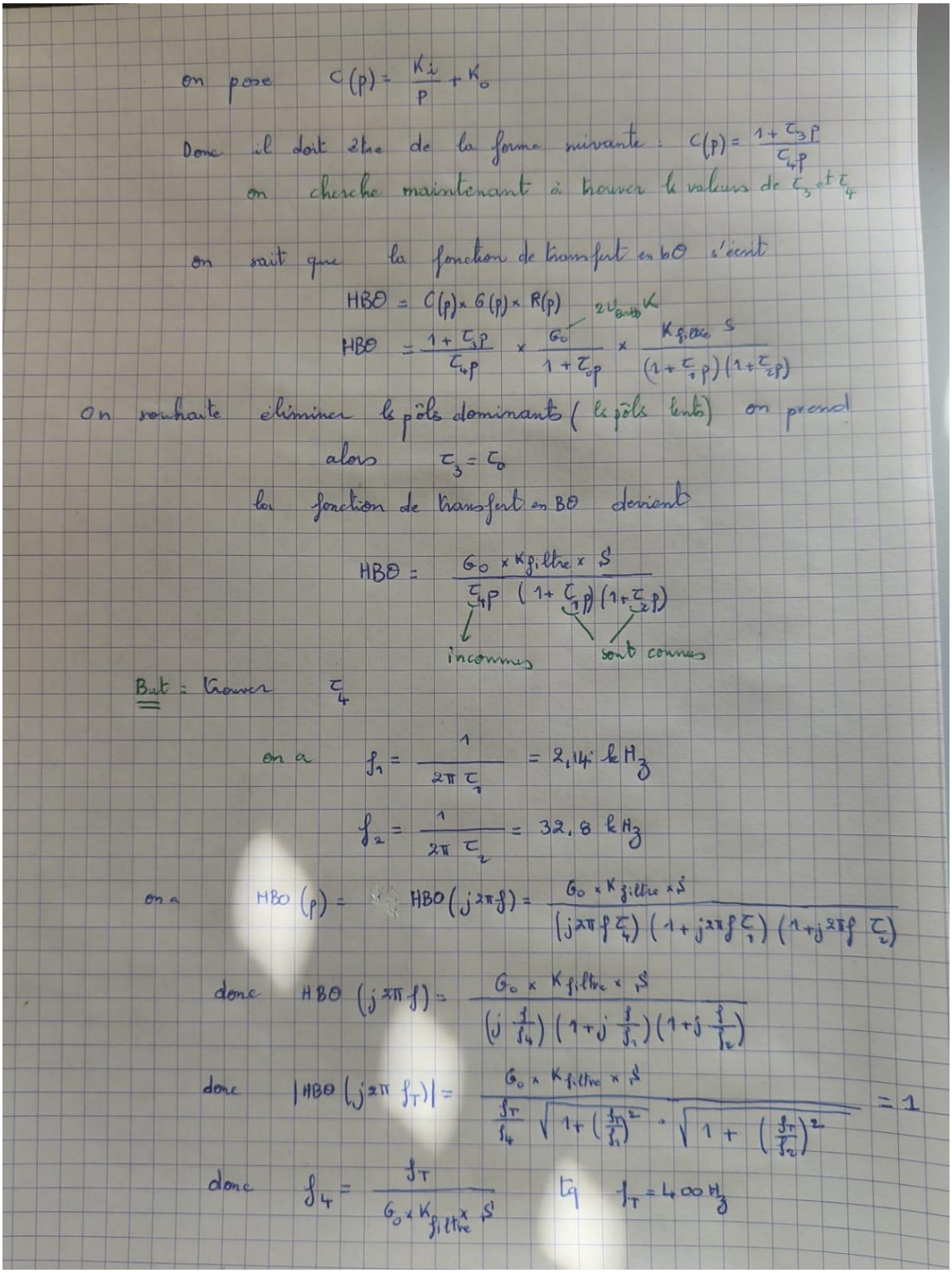
Question : Quel correcteur ?

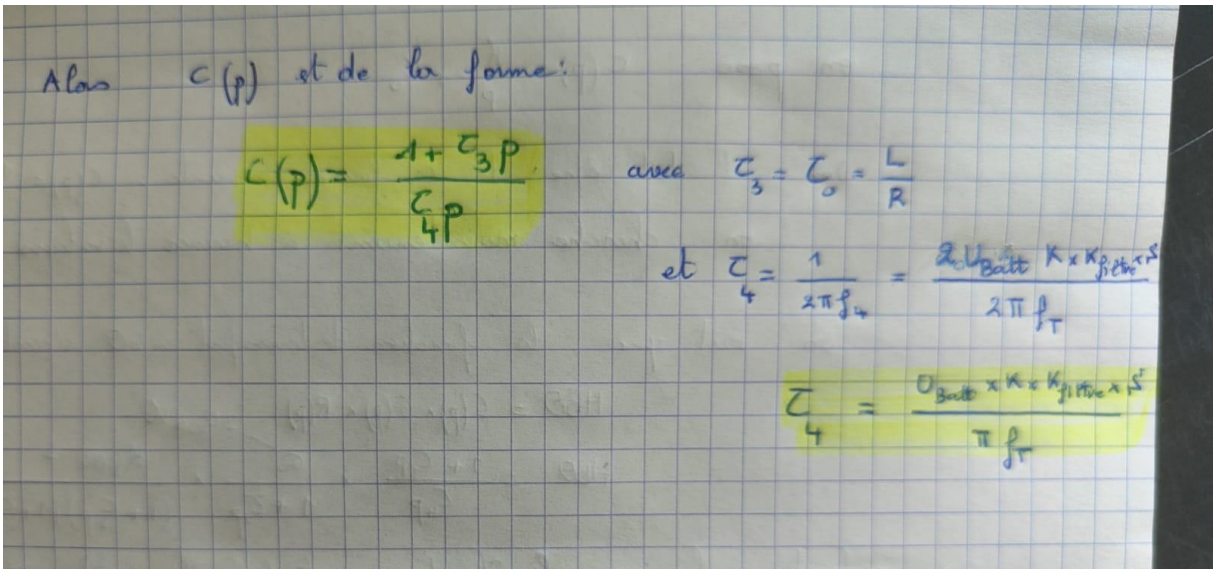
1ère approche : On décide dans un premier lieu d'utiliser un simple gain K comme correcteur tel que **C(p)=K.** On calcule alors l'erreur à l'infini et on utilise le théorème des valeurs finales.



2ème approche : Au lieu de choisir un correcteur simple de type gain on prend un PI

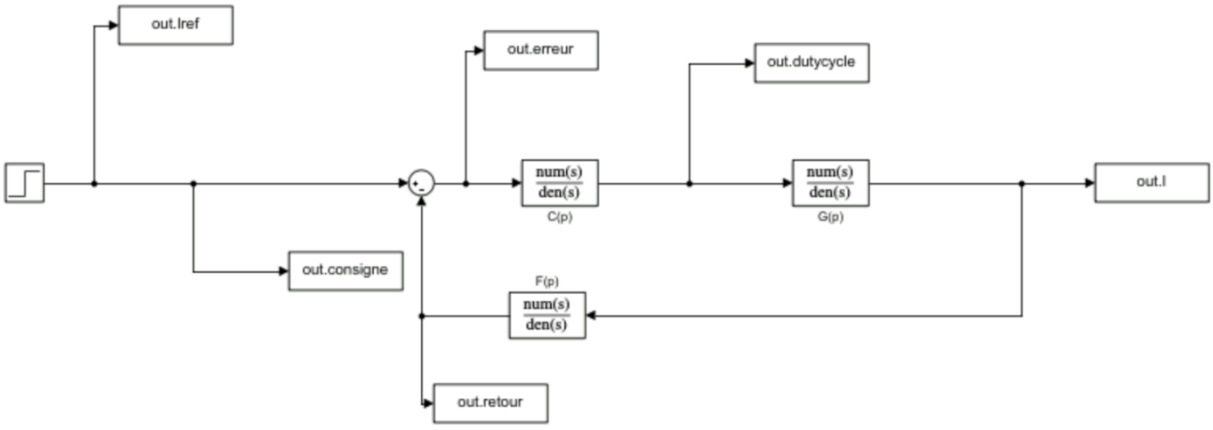
On pose alors $$C(p) = \left(\frac{Ki}{p} + K_0\right)$$




Alors C(p) est de la forme :
$C(p) = \frac{1+\tau_3 p}{\tau_4 p}$
avec $\tau_3 = \tau_0 = \frac{L}{R}$
et $\tau_4 = \frac{1}{2\pi f_4} = \frac{2 U_{Batt} K \times K_{\beta thc} \times S}{2\pi f_T}$
$\tau_4 = \frac{U_{Batt} \times K \times K_{\beta thc} \times S}{\pi f_T}$

V. Simulation sous Matlab du système:

Le schéma Simulink qu'on va utiliser pour une première approche est le suivant :



1. Tracé de bode en boucle ouverte :

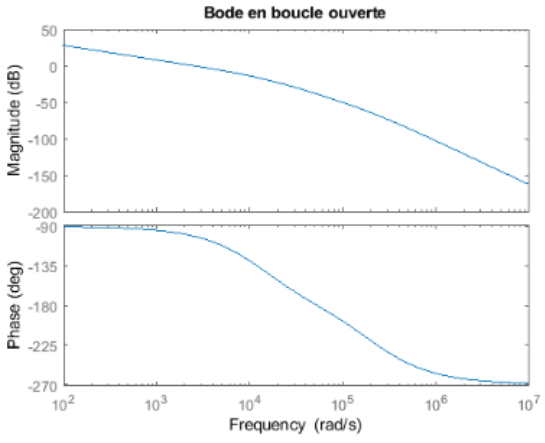
On trace notre système en boucle ouverte à l'aide des instructions suivantes :

```
HBO =

                    0.01456 s + 7.28
  ------------------------------------------------------------
  2.083e-15 s^4 + 4.596e-10 s^3 + 6.023e-06 s^2 + 0.002897 s

Continuous-time transfer function.
```

Diagramme de bode :

```
bode(HBO)
title("Bode en boucle ouverte")
```

Le bode qu'on obtient est :



Ce qui est conforme à notre fonction de transfert (1er ordre / ordre 4), on a bien -60 db/dec et -270 deg à haute fréquence.

Interprétations :

On trouve une fréquence de transition égale à 400 HZ et une marge de phase de 78 degrés.

2. Tracé de bode en boucle fermée :

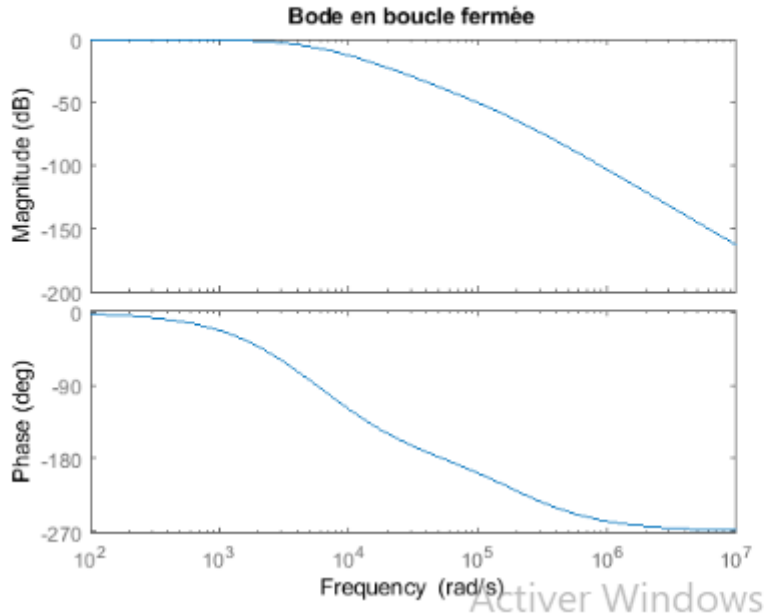
La fonction de transfert du système en boucle fermée est :

```
HBF = HBO / ( 1 + HBO)
```

```
HBF =

                3.034e-17 s^5 + 6.706e-12 s^4 + 9.104e-08 s^3 + 8.602e-05 s^2 + 0.02109 s
  ------------------------------------------------------------------------------------------------------------------
  4.341e-30 s^8 + 1.915e-24 s^7 + 2.363e-19 s^6 + 5.578e-15 s^5 + 4.564e-11 s^4 + 1.259e-07 s^3 + 9.441e-05 s^2
                                                                                                + 0.02109 s
```

Il s'agit d'un ordre 5 / ordre 8 ⇔ pente à -60 db/dec et -270 deg à haute fréquence.

```
bode(HBF)
title("Bode en boucle fermée")
```



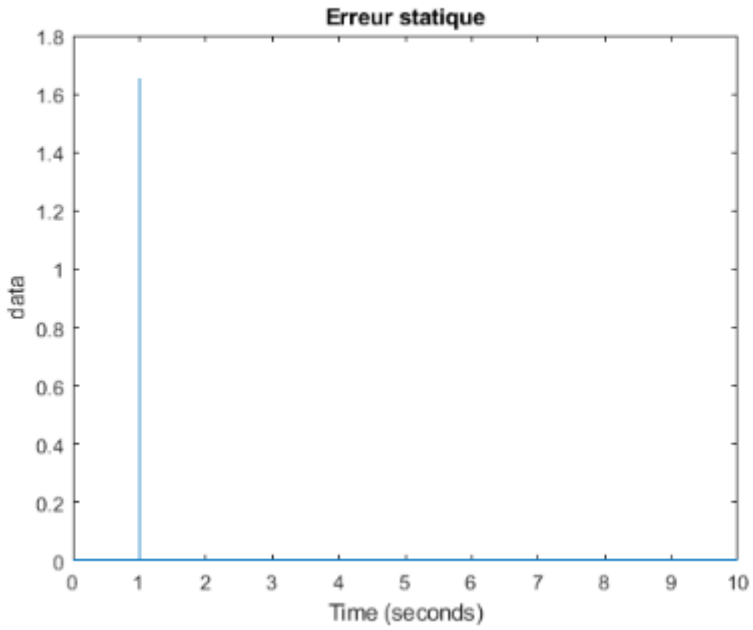
Interprétations :

On a bien une pente de -60 db/dec pour la courbe du gain ainsi que -270 deg pour le diagramme de phase ce qui confirme ce qu'il a était dits auparavant.

3. Validation du correcteur C(p)

On trace l'erreur statique sous Matlab pour bien vérifier que notre système est stable.

```
plot(out.erreur)
title("Erreur statique")
```



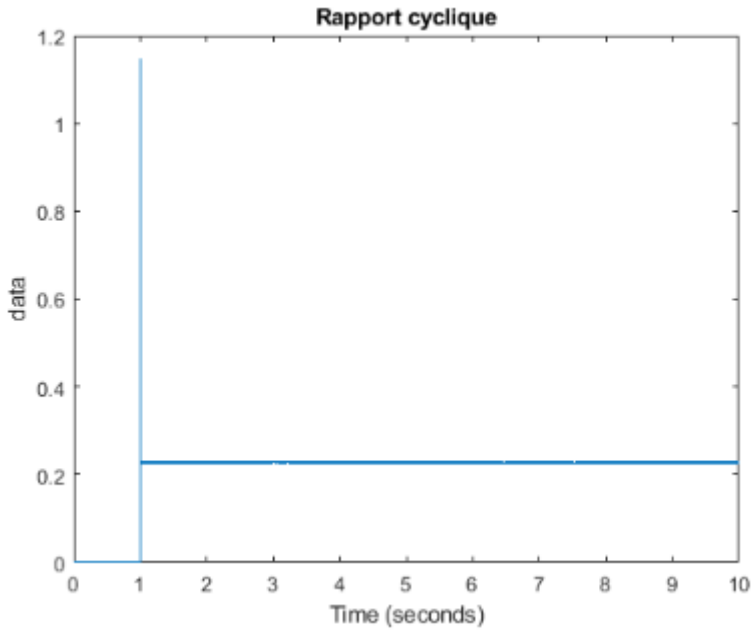
On trouve bien une erreur nulle, ceci confirme bien que le correcteur qu'on a choisis nous garantit la stabilité.

L'aberration en 1s est due à l'injection brusque de 1.65 V ce qui fait erreur =1.65-0 donc un pic de 1.65 V.

4. Rapport cyclique :

Afin de confirmer de manière définitive le choix du correcteur on décide de regarder le comportement du rapport cyclique.

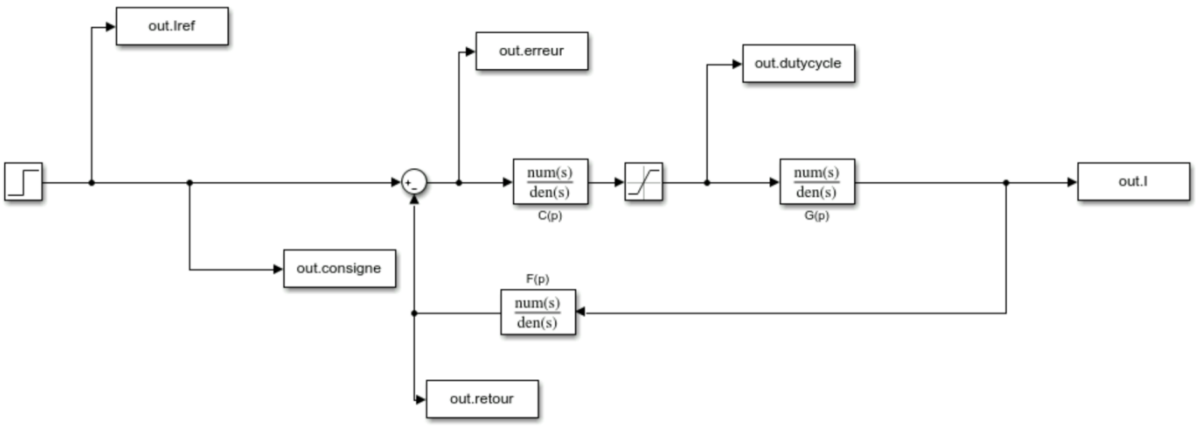
```
plot(out.dutycycle)
title("Rapport cyclique")
```



Nous on veut un rapport cyclique qui est autour de 0.5 (-0.5 à 0.5), ceci n'est pas vérifié par la simulation, on trouve un dutycycle autour de 1.18.

Pour remédier à ce problème on ajoute un saturateur qui va nous permettre de limité notre dutycycle entre -0.5 et 0.5.

Le Simulink devient comme suite :



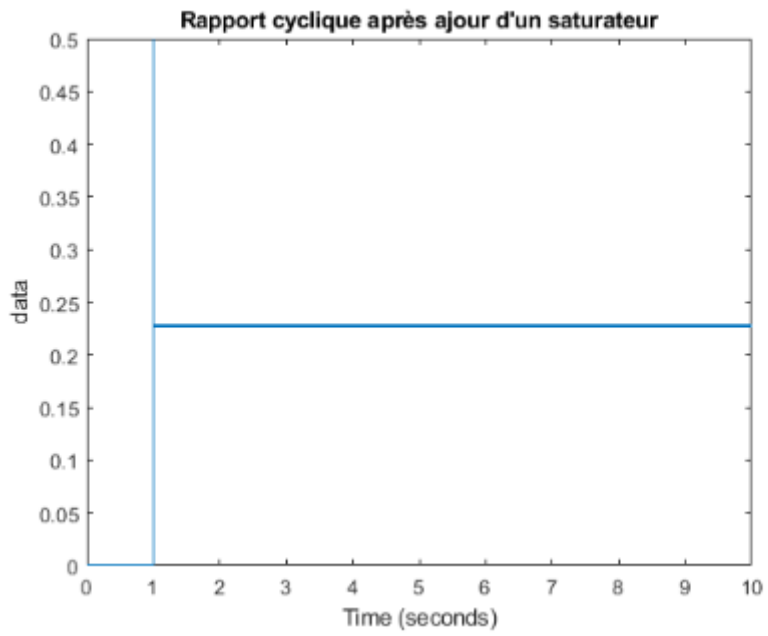
On refait alors la même simulation pour vérifier si le problème est résolu.

La simulation est la suivante :

```
out=sim("saturateur")
```

```
out =
  Simulink.SimulationOutput:

                    I: [1x1 timeseries]
                 Iref: [1x1 timeseries]
             consigne: [1x1 timeseries]
            dutycycle: [1x1 timeseries]
               erreur: [1x1 timeseries]
               retour: [1x1 timeseries]
                 tout: [560763x1 double]

    SimulationMetadata: [1x1 Simulink.SimulationMetadata]
          ErrorMessage: [0x0 char]
```

```
plot(out.dutycycle)
title("Rapport cyclique après ajour d'un saturateur")
```



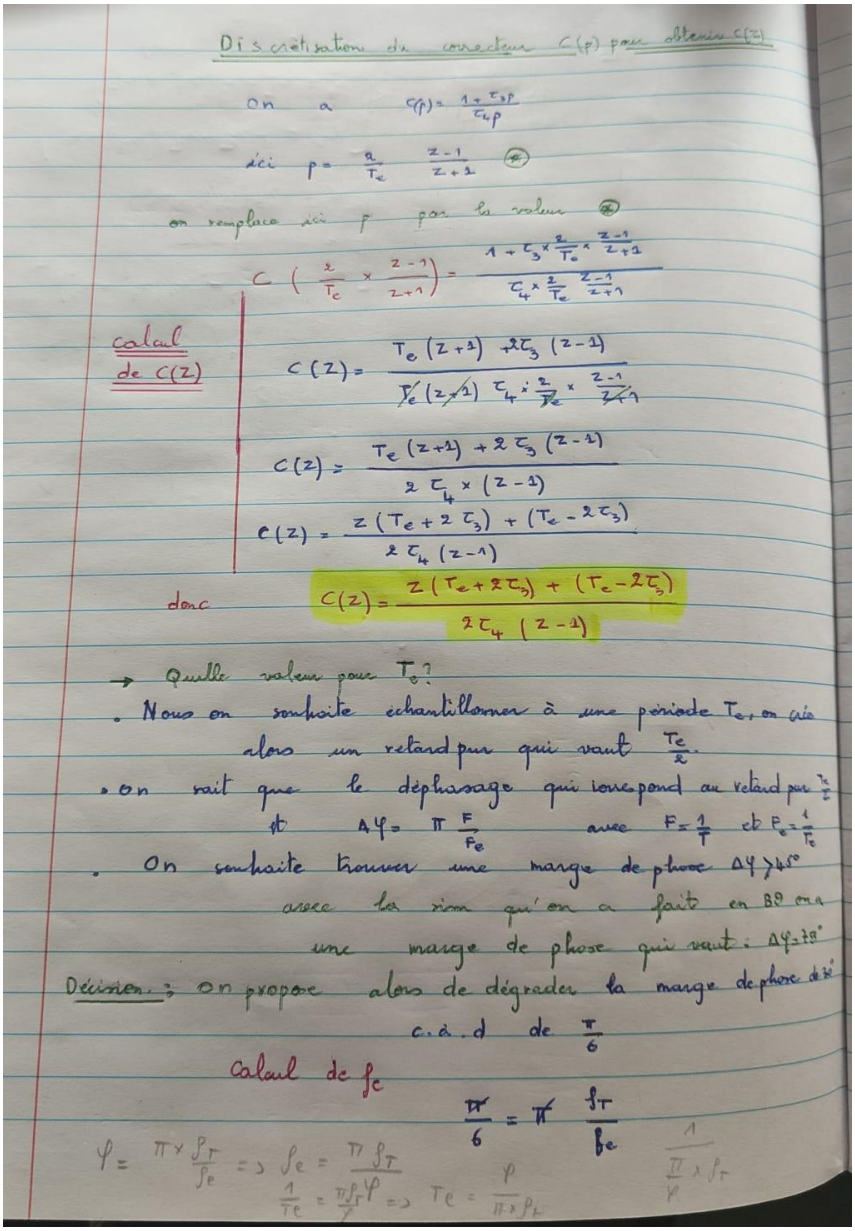
On remarque bien que notre saturateur a bien fait le boulot, on trouve bien un rapport cyclique qui varie entre -0.5 et 0.5.

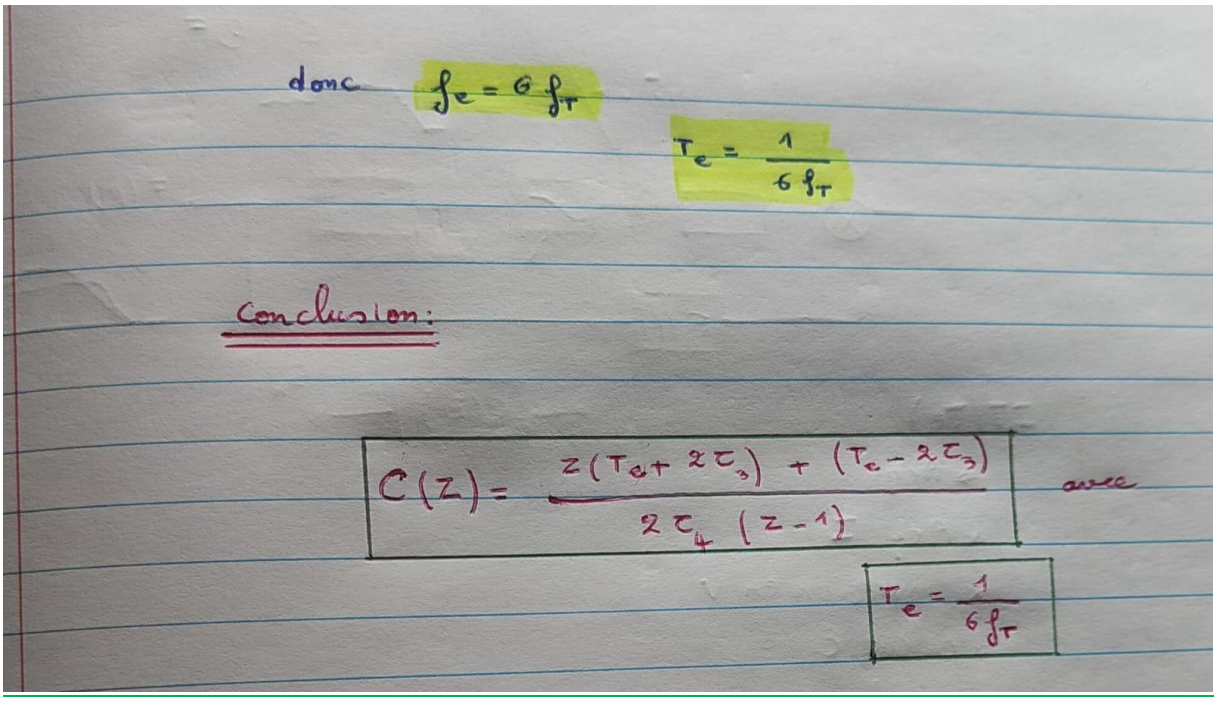
VI. Discrétisation de C(p)

La discrétisation est la transposition d'un état continu (fonction, modèle, variable, équation) en un équivalent discret.

Ce procédé constitue en général une étape préliminaire à la résolution numérique d'un problème ou sa programmation sur machine.

1. Calcul de C(z)





2. Validation de la simulation par Matlab

On écrit la fonction de transfert du correcteur en discret sous Matlab, puis on trace l'erreur en discret.

C(z) :

```
Cz = (z*(Te + 2*Tc3)+(Te -2*Tc3)) / ( z*2*Tc4 - 2*Tc4)
```

```
Cz =

  0.004417 z - 0.003583
  ---------------------
  0.005793 z - 0.005793

Sample time: 0.00041667 seconds
Discrete-time transfer function.
```

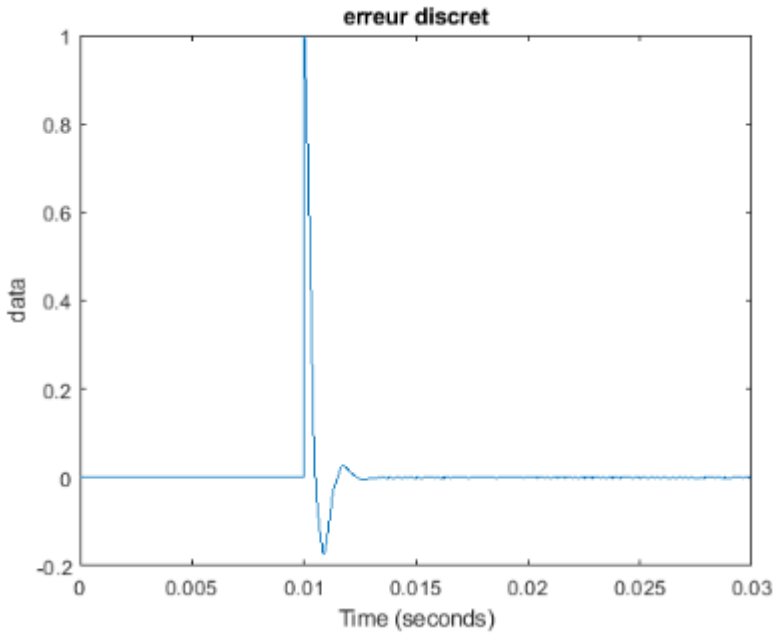
On simule l'erreur dans le domaine discret:

```
out=sim("discret")
```

```
out =
  Simulink.SimulationOutput:

                    I: [1x1 timeseries]
                 Iref: [1x1 timeseries]
             consigne: [1x1 timeseries]
            dutycycle: [1x1 timeseries]
              erreur: [1x1 timeseries]
              retour: [1x1 timeseries]
                tout: [1276x1 double]

    SimulationMetadata: [1x1 Simulink.SimulationMetadata]
          ErrorMessage: [0x0 char]
```

```
plot(out.erreur)
title("erreur discret")
```

Tracé de l'erreur :



Interprétations

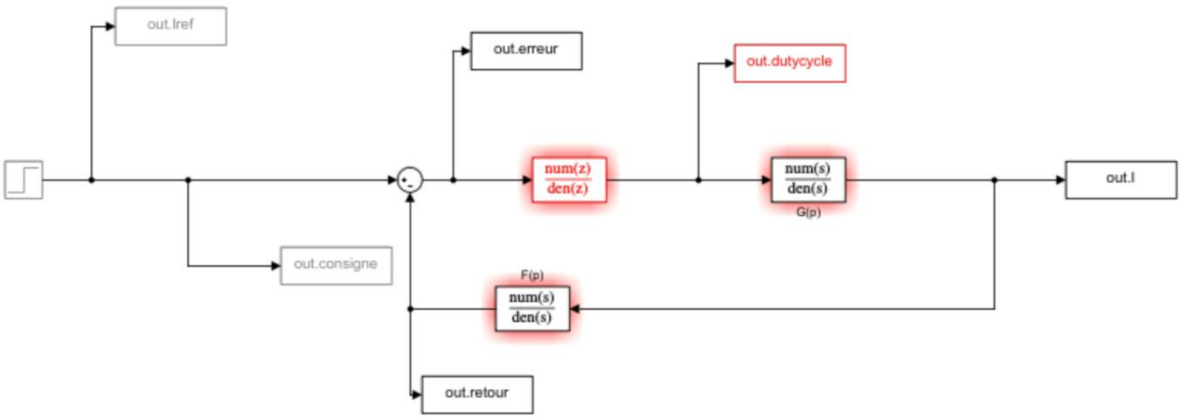
Même remarque que celle du tracé de l'erreur pour C(p).

On trouve bien une erreur nulle, ceci confirme bien que le correcteur discret C(z) nous garantit la stabilité.

L'aberration en 0.01s est due à l'injection brusque de 1.65 V ce qui fait erreur =1.65-0 donc un pic de 1.65 V.

Simulink du C(z)

Le simulink qu'on a utilisé pour le correcteur C(z) est le suivant :

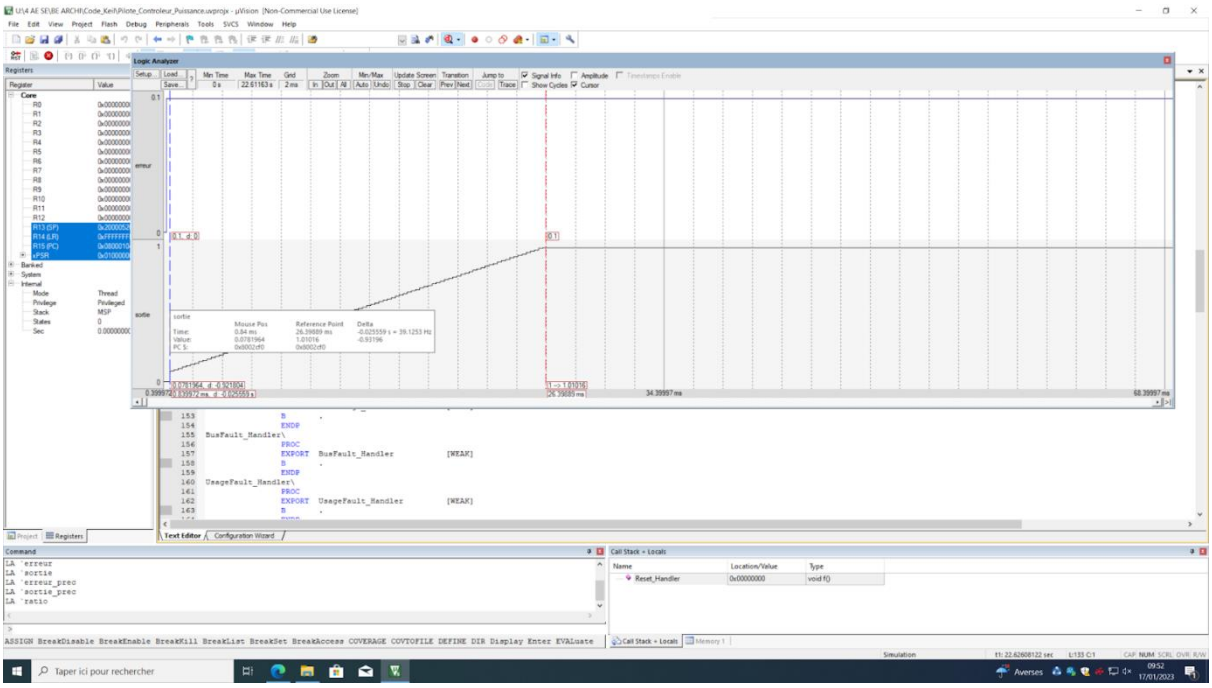


Le simulink nous permet de voir si les blocs sont ont discret (ceux qui sont en rouge sont ceux qui sont en discret)

VII. Implémentation sous KEIL :

Afin de compiler tout ce qu'on a fait sous le STM32 on utilise Keil, logiciel dans lequel on va remporter tout notre étude. Le code sera mis sous git.

Après avoir codé cela sous KEIL,on utilise le LOGIC-ANALYSER afin de voir l'allure de la sortie on trouve ceci :



On obtient une rampe de valeur : **a= (1-0,078)/(26,39-0,83)=0.036 V/s=36V/ms**

Validation avec Matlab

Pour valider le code implémenté sous KEIL on trace aussi la sortie sous MATLAB et on la compare avec celle obtenue avec KEIL.

Le code Matlab est le suivant :

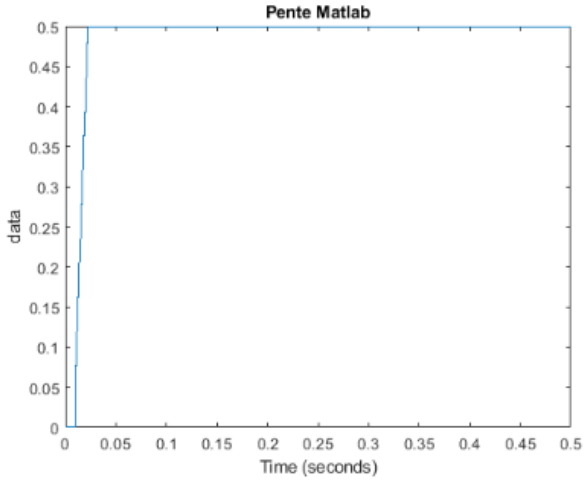
Pente à comparer avec la pente sur keil :

```
out=sim("validation")
```

```
out =
  Simulink.SimulationOutput:

                erreur: [1x1 timeseries]
                sortie: [1x1 timeseries]
                  tout: [1203x1 double]

    SimulationMetadata: [1x1 Simulink.SimulationMetadata]
          ErrorMessage: [0x0 char]
```

```
plot(out.sortie)
title("Pente Matlab")
```

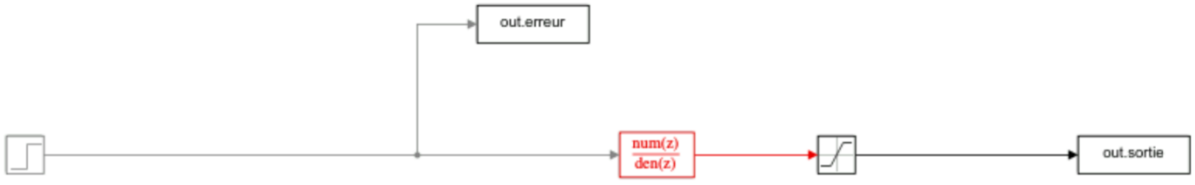
On Obtient alors cette pente :



La pente obtenue en utilisant Matlab et la suivante : **a1=0.037 V/s=37V/ms**

On remarque que **a1 ≈ a**

Le simulink qu'on utilise pour valider la pente sous Matlab est le suivant :



Conclusion :

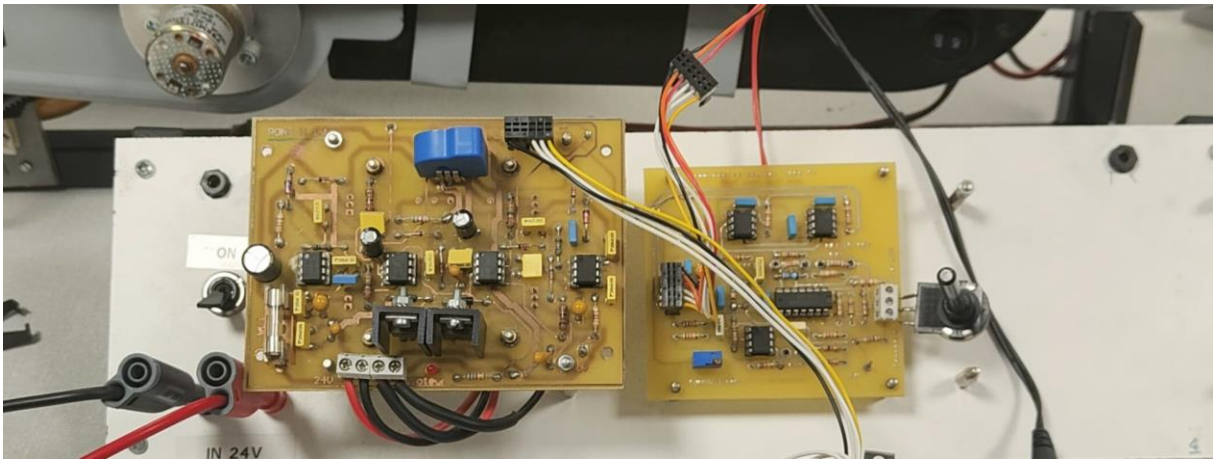
Le code utilisé sous Keil rejoint ce qu'on a fait sous Matlab donc on peut l'implémenter sous notre carte STM32 et faire nos études en réel.

VII. Validation en réel :

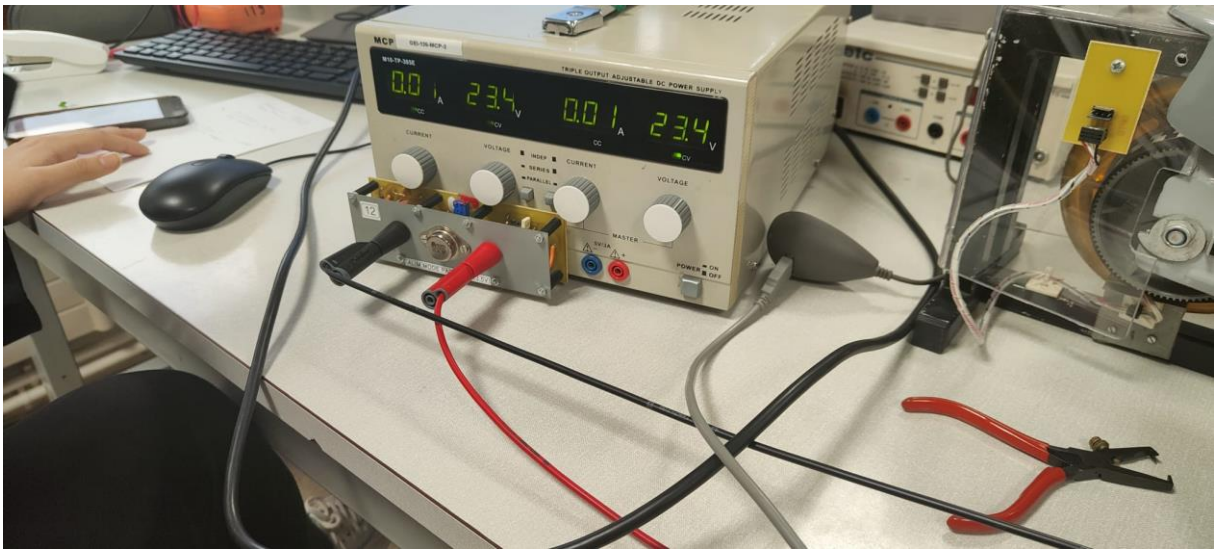
Test de la maquette

Nous commençons tout d'abord par tester notre maquette.

Le montage est le suivant :



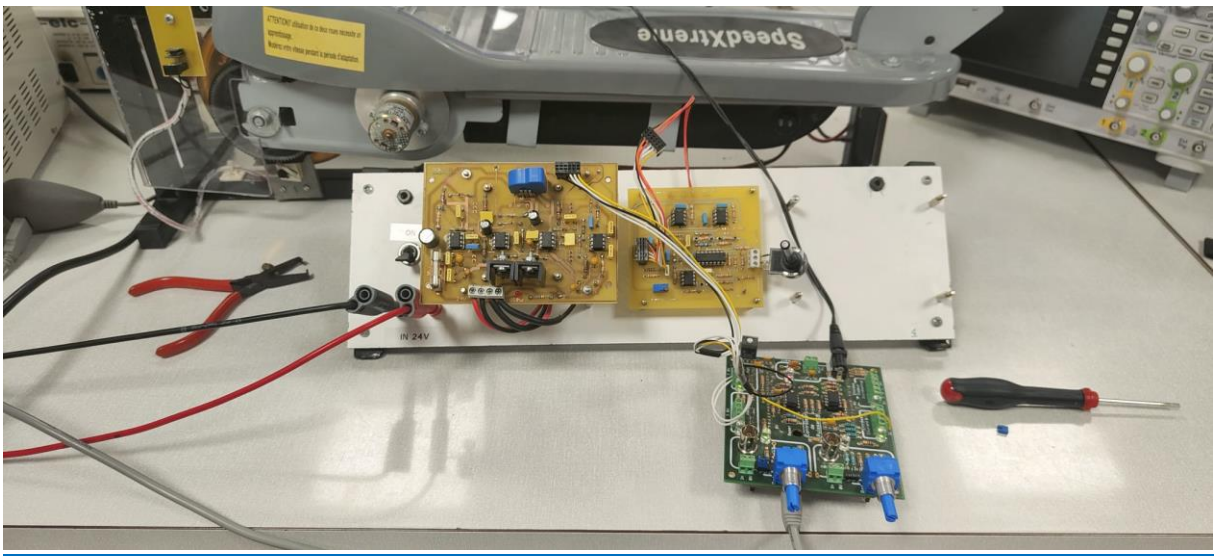
On alimente le système par une alimentation de 24 V on prend (23.5 par sécurité) et on commence par régler l'alimentation (on met le plus petit courant possible) on met l'alimentation en mode parallèle, puis à la sortie de cette dernière on met un transistor qui permettra d'arrêter l'alimentation si on injecte beaucoup de tension.

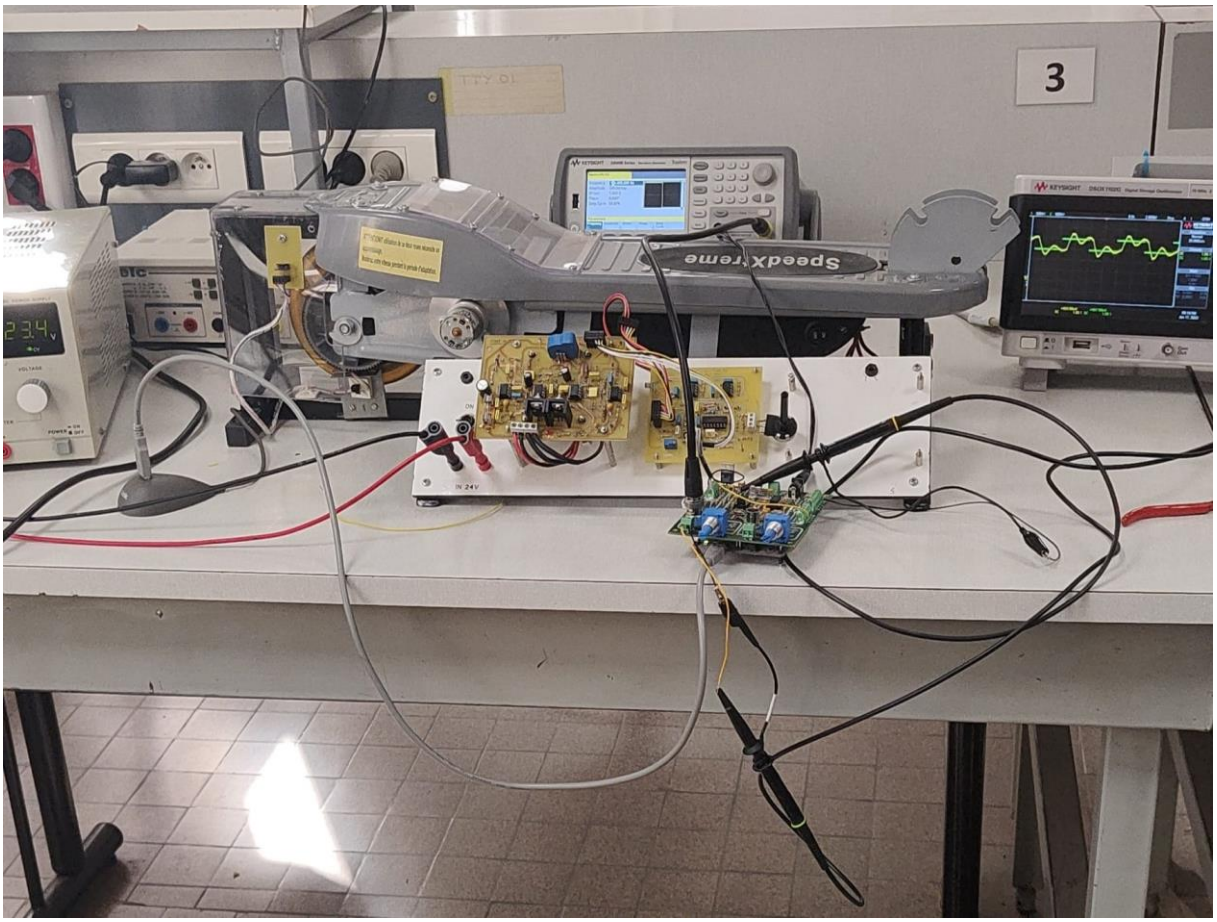


Ensuite, il suffit de faire tourner le potentiomètre dans un sens pour faire tourner la roue de la trottinette à droite et dans l'autre sens pour la faire tourner à gauche.

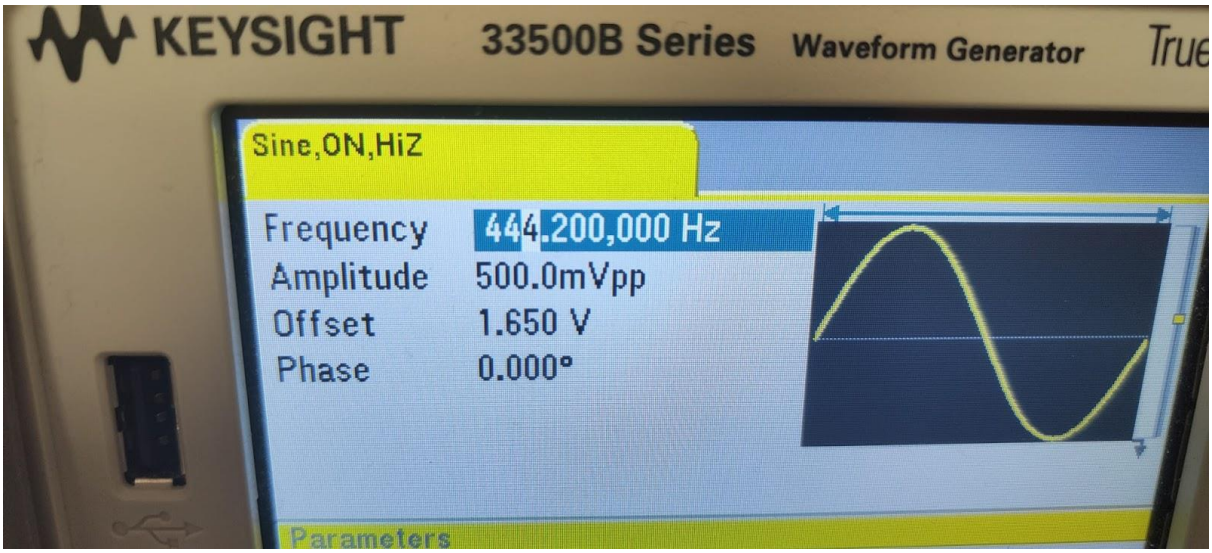
Compilation sur la carte STM32 :

On passe ensuite à l'implémentation sous la carte STM32 le montage est le suivant :



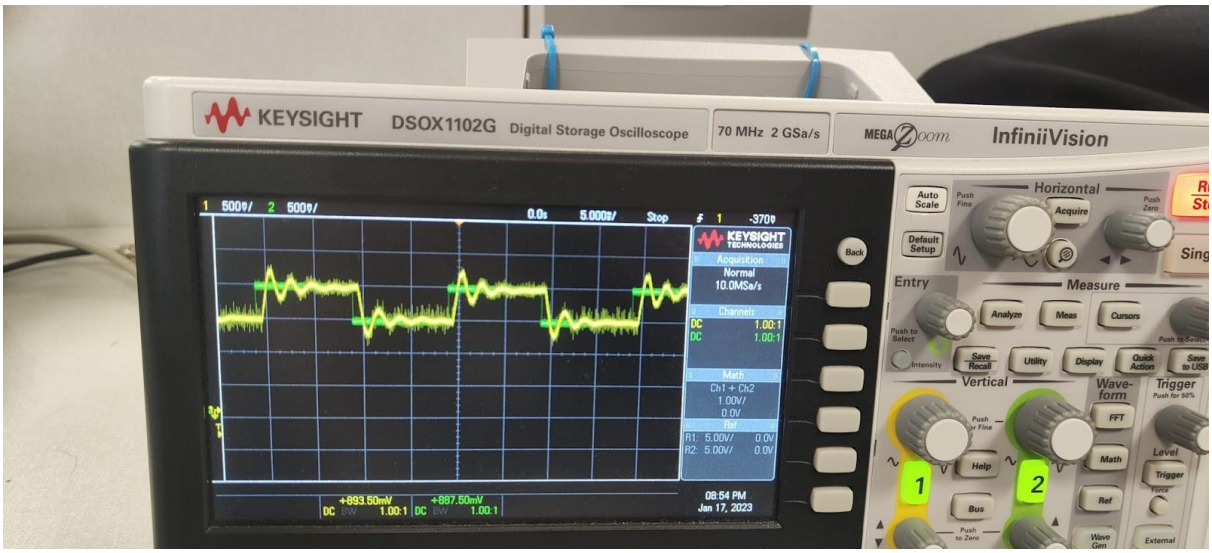


On règle le GBF comme suite :



Afin de bien vérifier notre montage on utilise un oscilloscope pour visualiser I1 et l'entrée de notre système.

On obtient ceci



Interprétations :

L'erreur est plus ou moins nulle, mais il y a des résonances.

Question : Source de cette résonnance ? (Fréquence de transition OK mais marge de phase pas bonne)

En premier lieu, on s'est dits que cette résonnance est due à la fréquence d'échantillonnage (peut-être qu'elle est élevé). Notre fréquence d'échantillonnage est de 5khz qui est de le bonne ordre donc la source de résonnance n'est pas la fréquence d'échantillonnage.

Dans un deuxième lieu, on s'est dits que notre fréquence de coupure est un peu élevé, on trouvait 400 hz qui est dans le bonne ordre donc la source de résonnance n'est pas la fréquence de transition.

Malheureusement, on n'a pas pu trouver la source de résonnance malgré plusieurs approche et ceci par faute de temps.

Conclusion :

Lors de ce BE, nous avons appris dans un premier temps une grande quantité d'informations et méthodes techniques de calcul grâce au questionnaire « Analyse de l'objet trottinette électrique ». En effet, on a pu revoir des notions vues dans les années précédentes et les affiner dans un cas concret ainsi que des nouvelles méthodes de calcul jamais vues avant.

Le « meilleur » aspect de ce BE est la nécessité d'utiliser en parallèle et simultanément de l'informatique, de l'automatique ainsi que de l'électronique sur un seul système. Pour nous c'était la première fois et on espère de pouvoir travailler sur d'autres projets de ce genre. En plus, ce bureau d'étude aura été un bon moyen pour intégrer certaines des contraintes qu'impose l'électronique de puissance à la mise en place d'une commande analogique et numérique. Sachant que la partie d'identification des fonctions de transfert, la création des schémas blocs ainsi que la modélisation sous Matlab et simulation sous Simulink, ont déjà été traités dans d'autres travaux pratiques de l'Automatique, ce qui nous suscité plus d'intérêt était la modélisation du correcteur. Ayant déjà vu comment un correcteur fonctionne, nous n'avons pourtant jamais été amené à en réaliser un.

Nous avons appris dans ce BE notamment comment choisir un correcteur, comment le modéliser ainsi que comment tester son fonctionnement. En plus, chose pas très évidente, nous avons appris qu'il existe des correcteurs analogiques et aussi numérique. Bien que le domaine analogique semble être dépassé dans nos jours, il a été intéressant de comprendre son fonctionnement afin de pouvoir comprendre celui numérique. La méthode d'étudier d'abord celui analogique puis, à travers une transformation bilinéaire, trouver celui numérique, a été très intéressante à mettre en œuvre et nous a simplifiés énormément le travail.