PROJET EIS - PX222

Guillemot MOUSSU et Rémi MAZZONE

2° semestre - 2022-2023

Table des matières

Table des figures

1 Séance 0 - 21/03/2023

1.1 Notes prises au tableau

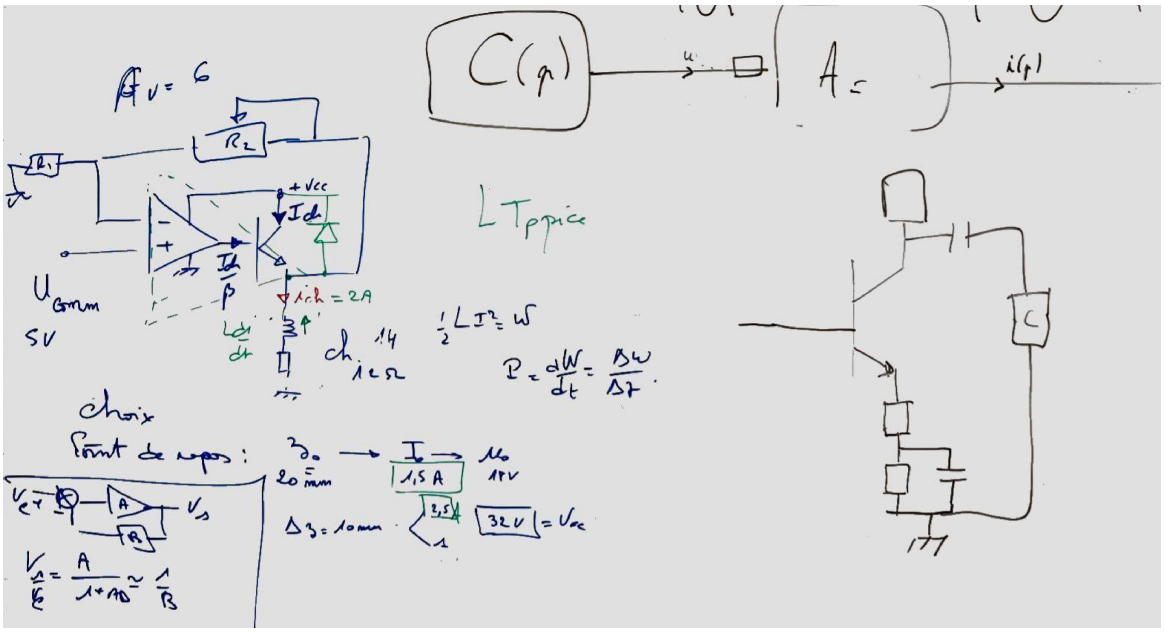

FIGURE 1 – Notes séance 0

1.2 Idées de plan de recherches

- **Caractérisation de la bobine** : mesure de la résistance et l'inductance de la bobine. Cela permettra de modéliser la bobine et de calculer le champ magnétique qu'elle produit
- **Modélisation de la bobine** : utilisation des valeurs mesurées pour construire un modèle électrique de la bobine, par exemple en utilisant l'équation de l'inductance d'une bobine. Pour une précision plus élevée, il peut être nécessaire d'inclure d'autres effets tels que la saturation du noyau de la bobine ou la résistance série équivalente
- **Étude du champ magnétique de la bobine** : calcul de la distribution du champ magnétique à l'aide d'un logiciel de simulation. Validation des résultats expérimentalement
- **Choix d'un capteur de distance** : choix d'un capteur de distance infrarouge adapté à la plage de mesure et à la précision requises. La fréquence de mesure et la plage de mesure peuvent influencer le résultat
- **Calibration du capteur de distance** : déterminer la relation entre la tension de sortie du capteur et la distance de la bille par rapport à la bobine
- **Conception d'un correcteur** : utiliser les données obtenues précédemment pour concevoir un correcteur proportionnel-intégral-dérivé (PID) ou un correcteur à avance de phase pour contrôler la position de la bille. C'est la partie clé de ce projet. Le correcteur PID est une méthode de contrôle classique qui est souvent utilisée pour les systèmes de positionnement. Un correcteur à avance de phase peut également être utilisé pour améliorer les performances.
- **Implémentation du correcteur** : test du correcteur via Matlab et Simulink
- **Étalonnage du système** : ajuster les paramètres du correcteur pour atteindre la précision et la stabilité requises
- **Validation du système** : tester le système dans différentes conditions et valider ses performances
- **Optimisation du système** : améliorer le système en optimisant les paramètres du correcteur et en utilisant des techniques avancées de contrôle, telles que le contrôle prédictif et le contrôle adaptatif

Il est important de noter que certains des points ont déjà été réalisés lors du premier semestre

2 Séance 1 - 05/04/2023

2.1 Reprise des éléments du semestre précédent

Avec Hopkinson, nous avions obtenu un schéma équivalent de la bobine :

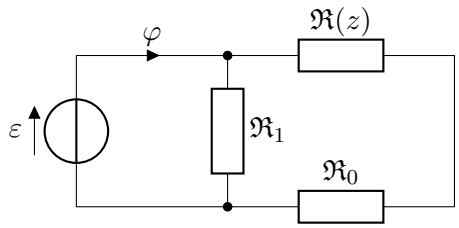

FIGURE 2 – Modélisation de la bobine

Nous avions trouvé que la fonction de transfert de la bobine est : $\boxed{T_{BO} = \frac{Z}{I} = \frac{k_i}{k_z - mp^2}}$

Avec les valeurs suivantes :
$I_0 = 2A$, $Z_0 = 22mm$, $L_1 = 6.73H$, $\alpha = 2.06$, $m = 35.8g$

$k_i = \frac{I_0 \cdot L_1 \cdot \alpha}{(1 + \alpha \cdot Z_0)^2}$, $k_z = \frac{I_0^2 \cdot L_1 \cdot \alpha^2}{(1 + \alpha \cdot Z_0)^3}$

2.2 Utilisation de Matlab

Nous avons donc notre fonction de transfert. Une idée qui nous est venue est d'adapter les scripts pour utiliser l'outil de M. Mendes que nous avons découvert en TP. Cela devrait nous permettre de calculer notre correcteur. Pour commencer, on a placé nos valeurs dans Matlab et tracé le bode obtenu. Cette étape est présente seulement pour nous faire une idée du système à l'heure actuelle

Code Matlab que nous avons utilisé :

```
I0 = 2;
Z0 = 22 * 10^-3;
L1 = 6.73;
alpha = 2.06;
ki = (I0 * L1 * alpha) / (1 + alpha * Z0) ^2;
kz = -(I0 ^2 * L1 * alpha ^2) / (1 + alpha * Z0) ^3;
m = 35.8;

num = ki ;
den = [(-m) 0 kz];
sys = tf(num,den);

figure;
bode(sys)
grid on
figure;
nyquist(sys)
grid on
```

On obtient donc le bode suivant :

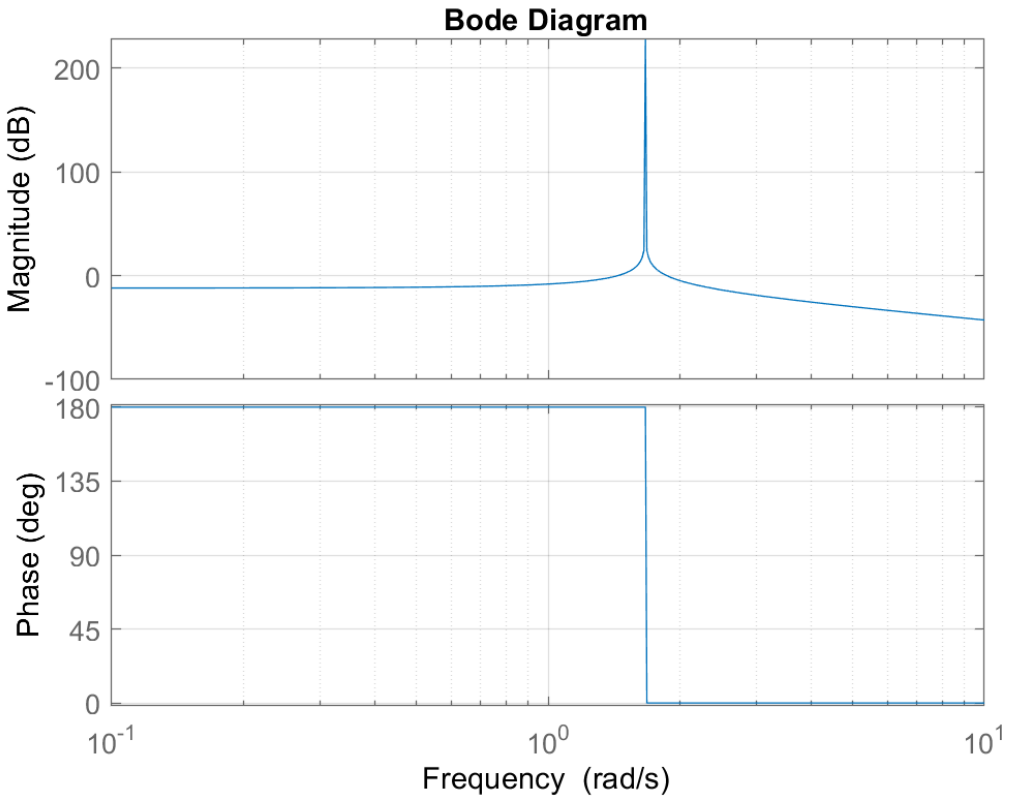

FIGURE 3 – Bode obtenu

On trace ensuite le diagramme de Nyquist :

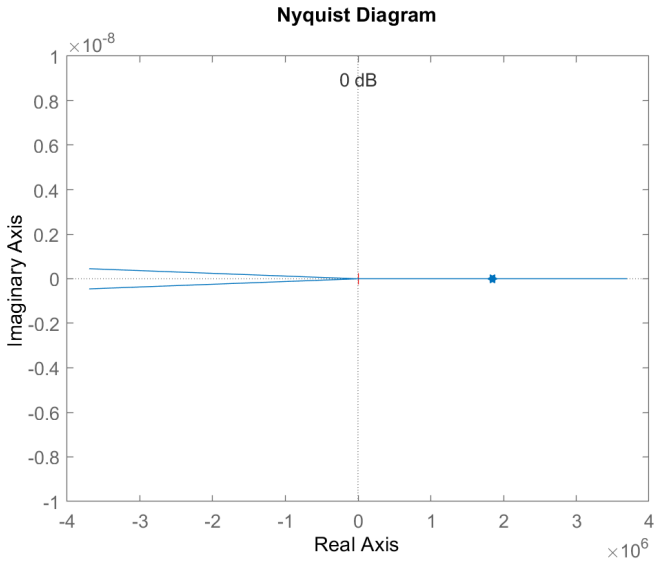

FIGURE 4 – Nyquist obtenu

On remarque donc qu'il reste du travail avant d'avoir un résultat satisfaisant, mais nous reviendrons sur ce point ultérieurement

2.3 Modélisation point par point du système

L'objectif est de réaliser un circuit correspondant à ce schéma bloc :

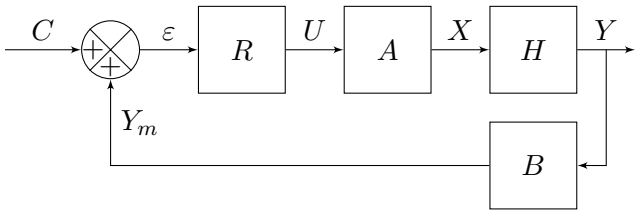

FIGURE 5 – Schéma bloc du système

2.3.1 Modélisation de l'amplificateur

Nous avons en sortie d'un STM8 un signal de 5V, nous voulons que la bobine soit alimentée en 30V. Pour cela on utilise un amplificateur à AOP, en série avec un transistor bipolaire. On a le schéma classique d'un amplificateur à AOP :

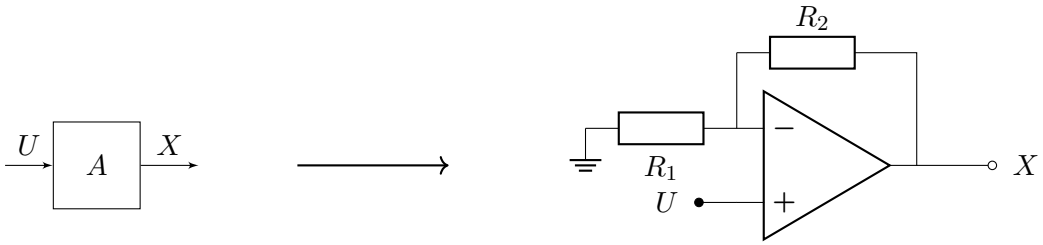

FIGURE 6 – Amplificateur à AOP

Si on ajoute le transistor et la bobine au schéma, cela nous donne :

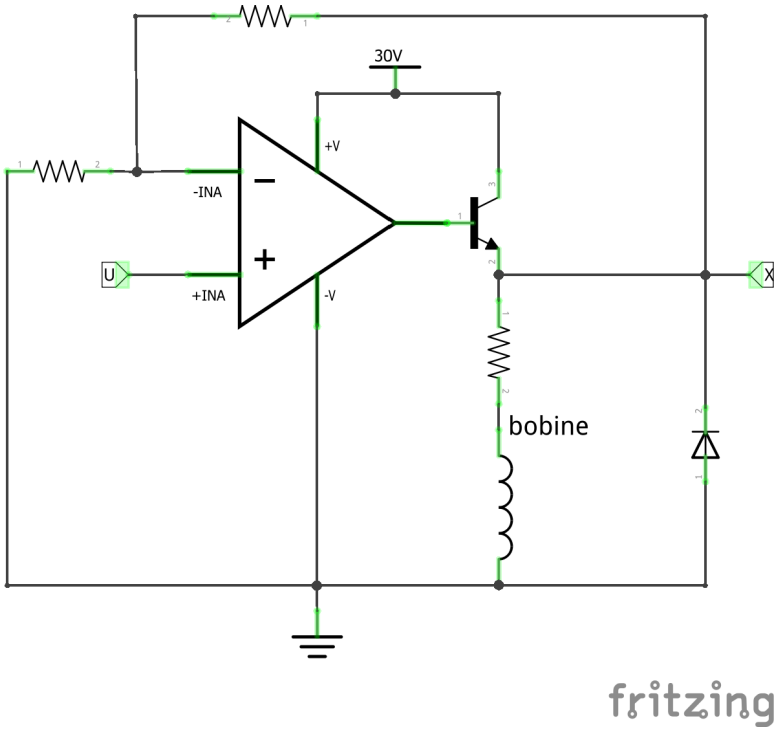

FIGURE 7 – Modélisation du bloc amplificateur

2.3.2 Modélisation d'un régulateur PD

On a le schéma classique d'un régulateur PD :

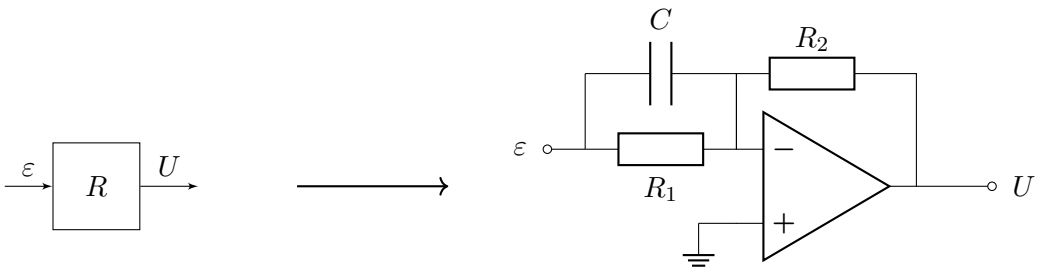

FIGURE 8 – Régulateur PD

Si on donne le schéma complet de ce bloc, cela nous donne :

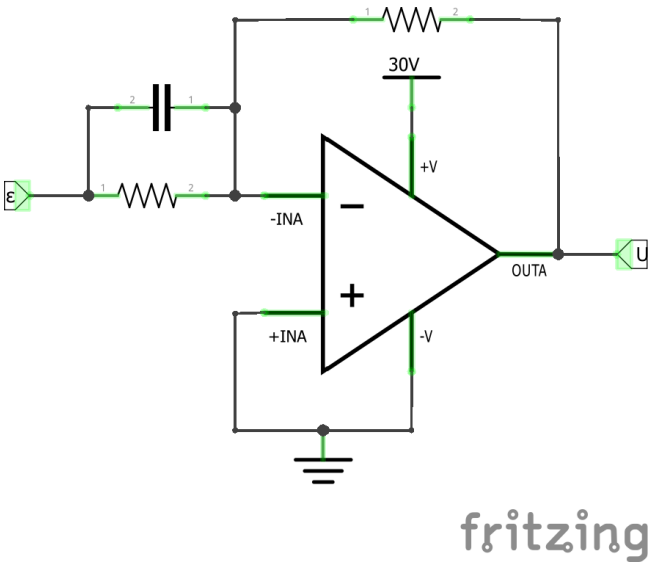

FIGURE 9 – Modélisation du bloc régulateur (PD)

2.3.3 Modélisation d'un régulateur PID

Dans le cas où l'on souhaite finalement modéliser un régulateur PID, on doit ajouter un condensateur en parallèle de notre résistance :

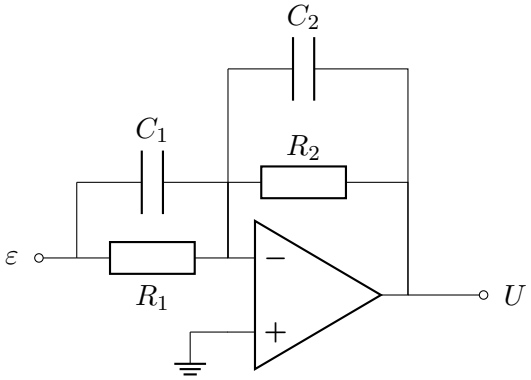

FIGURE 10 – Régulateur PID

2.3.4 Modélisation du système complet (PD)

On choisit le régulateur PD pour représenter le système complet. On a vu qu'il suffira d'ajouter une capacité pour passer à un régulateur PID au besoin. On repère bien les blocs comparateur, amplificateur (A), régulateur (R) et capteur (B)

a

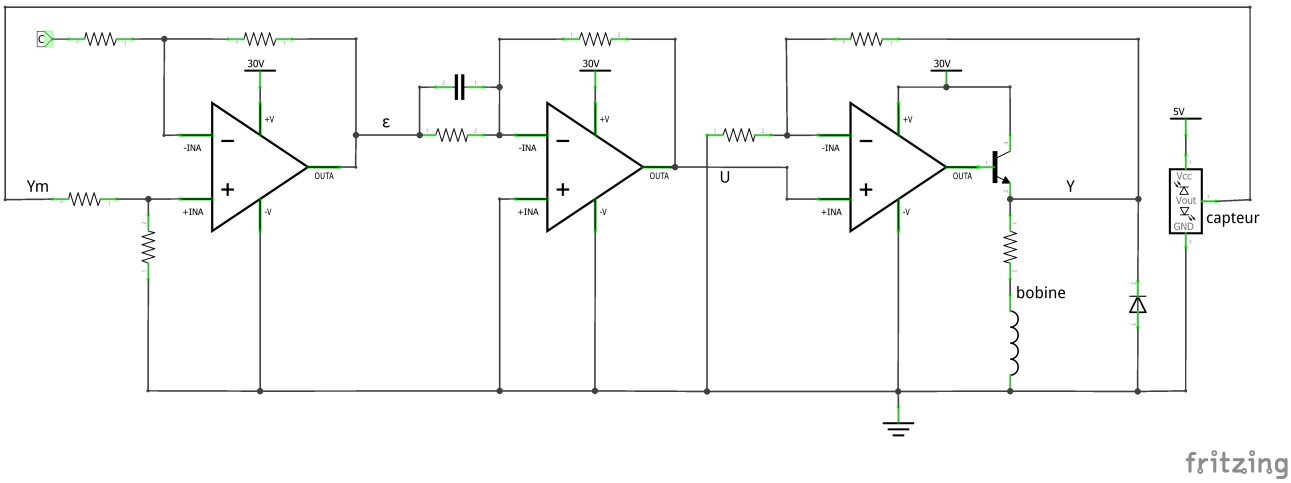

FIGURE 11 – Modélisation complète avec retour (PD)

2.4 Conclusions pour cette séance

On commence à avoir des pistes intéressantes. Les schémas que nous avons réalisés vont nous permettre de se faire une idée du système à concevoir, et nous avons perçu l'importance d'utiliser Matlab pour nous aider dans nos calculs

3 Travail entre les séance 1 et 2

3.1 Correction des éléments de la séance précédente

Nous avions manqué de temps pour terminer comme nous le souhaitions certains points de la séance précédente. Nous avons travaillé sur les points suivants :

3.1.1 Matlab

Pour plus de détails se référer à la section 2.2

Les schémas avaient étés rendus avec Octave, mais après comparaison des rendus, ceux de Matbab donnent d'autres résultats. Les figures 3 et 4 ont donc été mises à jour

3.1.2 Modélisation point par point du système

Pour plus de détails se référer à la section 2.3

La figure 5 est maintenant rendue en LaTeX

Pour plus de clarté, les points 2.3.1, 2.3.2 et 2.3.3 ont été repris. Les figures 6, 8 et 10 ont été reprises en LaTeX. Les figures 7 et 9 ont été reprises sur Fritzing, avec un bloc AOP mis à jour

Le point 2.3.4 a été séparé plus plus de clarté. La figure 11 y a été ajoutée

3.2 Simulation de l'amplificateur

On veut donc simuler notre bloc amplificateur. Dans l'idéal on devrait avoir le graphique suivant :

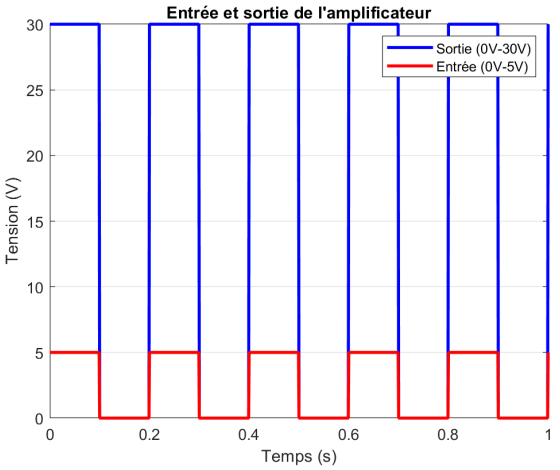

FIGURE 12 – Résulat idéal de la simulation

On rentre donc le modèle suivant dans LTSPICE :

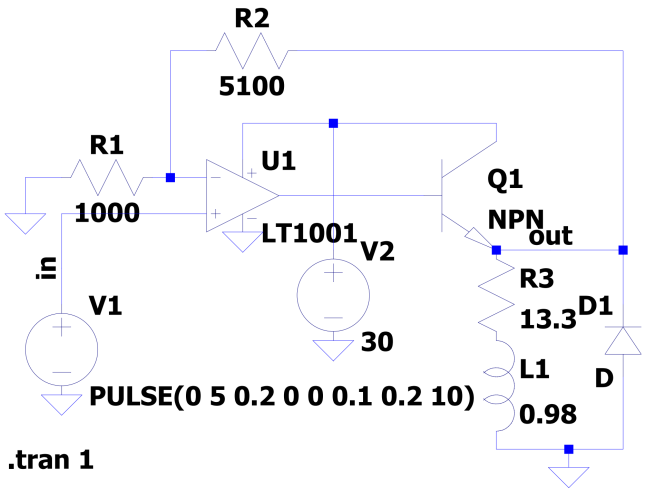

FIGURE 13 – Schéma LTSPICE de l'amplificateur

Et on obtient le résultat suivant :

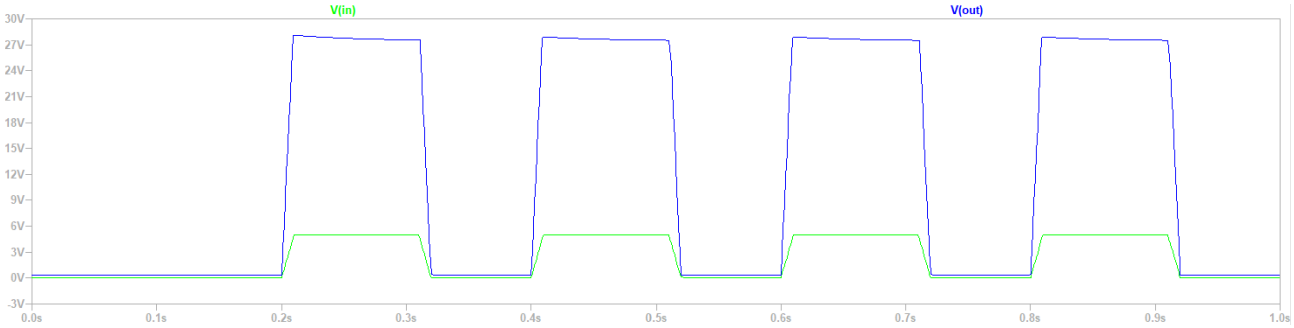

FIGURE 14 – Simulation de l'amplificateur

Remarques :

- On a utilisé un AOP quelconque (le LT1001 en l'occurence)
- On a utilisé les mesures de la bobine du S1 ($R = 13.3\Omega$ et $L = 0.98H$)
- On a utilisé un transistor quelconque

On remarque que la simulation correspond au modèle théorique voulu. C'est bon signe pour la suite, il reste à le tester avec des composants réels

4 Séance 2 - 18/04/2023

4.1 Réflexion sur la numérisation du correcteur

On va donc chercher à numériser la partie correcteur. Cela correspond aux éléments suivants sur notre schéma bloc :

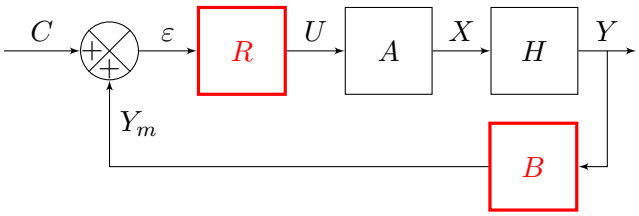

FIGURE 15 – Schéma bloc du système, version électronique numérique

On utilisera un STM8s pour la numérisation. On compte utiliser un port analog input pour l'entrée venant du capteur, et un port I/O piloté en PWM pour la sortie U

4.2 Prise en main du STM8s

On a le brochage du STM8 :

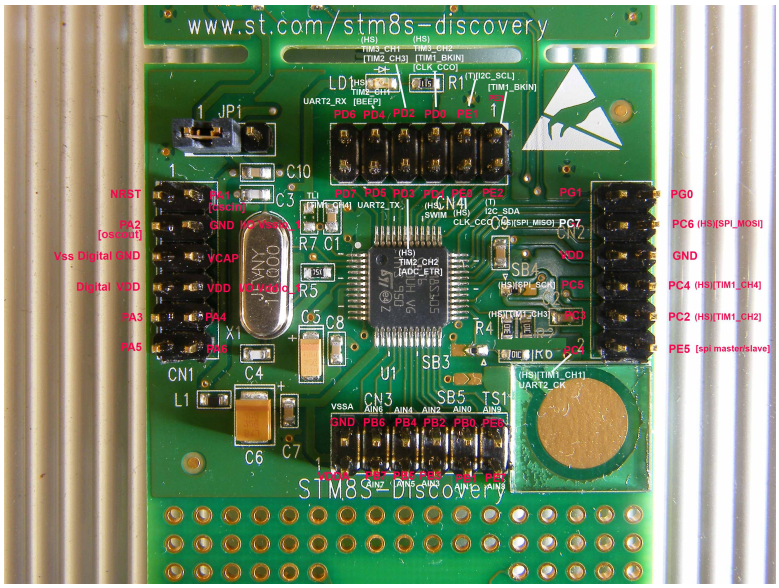

FIGURE 16 – Brochage du STM8s discovery

On définit pour le moment qu'on utilise le port $B0$ (Analog 0) pour l'entrée et le port $G0$ pour la sortie (I/O classique)

4.3 Premier essai de code

Par la suite, nous avons cherché à faire fonctionner la carte, en vue de pouvoir faire nos mesures. On se base sur l'exemple fourni par ST permettant de faire clignoter la LED installée sur la carte : fichiers du projet. Ce projet est écrit en C, compilé à l'aide de Cosmic. On confirme que cela fonctionne

4.4 Essais avec Arduino

Afin de pouvoir communiquer, notemment via la communication série avec la carte, on a essayé d'installer les librairies nécessaires sur l'Arduino IDE. On se base sur le travail de Michael Mayer pour le

site CircuitDigest. Malgré de nombreuses tentatives, nous n'avons pas réussi à compiler un projet et à l'envoyer sur la carte

Sur nos PC, nous obtenons l'erreur suivante :

```
Installing platform sduino:stm8@0.5-pre2
Failed to install platform: sduino:stm8.
Error: 13 INTERNAL: Cannot install platform: installing platform sduino:stm8@0.5-pre2:
    searching package root dir: no unique root dir in archive, found
    'C:\Users\guill\AppData\Local\Arduino15\tmp\package-518500940\STM8S_StdPeriph_Driver'
    and 'C:\Users\guill\AppData\Local\Arduino15\tmp\package-518500940\cores'
```

Et sur les PC de l'école, malgré l'installation des drivers, Arduino ne détecte pas la carte dans la liste des ports disponibles

4.5 Essais avec STVD

Nous avons en parallèle repris le code d'un précédent TP, où nous devions implémenter la génération de signaux PWM :

```
stm8/
    #include 'mapping.inc'
    #include 'stm8s105c_s.inc'

    segment 'rom'
debut
    bset PD_DDR,#3
    bset PD_CR1,#3
    bres PD_ODR,#3
    mov TIM4_PSCR,#6
    mov TIM4_ARR,#155
    bset TIM4_IER,#0
    bset TIM4_CR1,#0
    rim
    bset ADC_CR1,#0
    mov ADC_CSR,#$00
    bres ADC_CR2,#3

boucle_infinie
    jra boucle_infinie

    interrupt Timer4_ISR
Timer4_ISR
    bres TIM4_SR,#0
    bcpl PD_ODR,#3
    bset ADC_CR1,#0
attente_fin_conversion
    btjf ADC_CSR,#7,attente_fin_conversion
    bres ADC_CSR,#7
    ld A,#1
    or A,ADC_DRH
    ld TIM4_ARR,A
    iret

    segment 'vectit'
    dc.l {$82000000+debut}
    segment at 8064 'vectit'
    dc.l {$82000000+Timer4_ISR}

    end
```

Le but de ce code est de réaliser les actions suivantes :

- Toutes les 10ms, une interruption se produit et déclenche la conversion ADC, et attend la fin de la conversion
- La valeur de la conversion est stockée dans le registre TIM4_ARR, qui est utilisé pour définir la période du timer 4

En tant que tel, on peut ensuite utiliser un oscilloscope pour afficher la tension mesurée

Nous n'avons pas eu le temps de réfléchir au code, et nous ne savons pas encore si nous préfèrerons le faire en assembleur ou en C

4.6 Conclusions pour cette séance

Si nous avons pu prendre en main la carte et réfléchir à l'implémentation du correcteur numériquement, les nombreux freins (problèmes logiciels, logiciels peu ergonomiques...) ont été un frein et nous n'avons pas eu le temps de faire beaucoup de choses. Il sera donc important de travailler entre les deux prochaines séances pour avancer, tant sur la partie analogique, que la partie numérique

5 Travail entre les séances 2 et 3

5.1 Lecture d'une entrée analogique

Des recherches sur internet nous ont permis de trouver le tutoriel suivant : Tutoriel de CircuitDigest. Le code fourni permet de lire des entrées analogiques et de les afficher sur un écran LCD. On adapte le code pour que la valeur soit renvoyée dans le moniteur série :

```
#include "STM8S.h"
#include "stm8s103_adc.h"
#include "stm8s_uart1.h"

void UART1_Send_Char(char data)
{
    // Attendre que le buffer de transmission soit vide
    while(!(UART1->SR & UART1_SR_TXE));
    // Envoyer le caractere
    UART1->DR = data;
}

void UART1_Send_String(char *str)
{
    while(*str)
    {
        UART1_Send_Char(*str);
        str++;
    }
}

void UART1_Print_Var(int var)
{
    char buffer[6];
    sprintf(buffer, "%d", var);
    UART1_Send_String(buffer);
}

void main()
{
    // Declarations de variables
    unsigned int ADC_value_1 = 0;
    int ADC_voltage_1 = 0;
```

```
34
35      CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, ENABLE); // Activer l'horloge
            peripherique pour l'ADC
36      GPIO_Init (GPIOC, GPIO_PIN_4, GPIO_MODE_IN_FL_IT);
37
38      // Configuration de l'UART
39      CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, ENABLE);
40      UART1_DeInit();
41      UART1_Init((uint32_t)9600, UART1_WORDLENGTH_8D, UART1_STOPBITS_1, UART1_PARITY_NO,
            UART1_SYNCMODE_CLOCK_DISABLE, UART1_MODE_TX_ENABLE);
42
43      delay_ms(5000);
44
45      while (1)
46      {
47          ADC_value_1 = ADC_Read (AIN2);
48          ADC_voltage_1 = ADC_value_1 * (3.226); //(3300/1023 =~ 3.226)convert ADC value 1 to
                0 to 3300mV
49
50          UART1_Send_String("D1: ");
51          UART1_Print_Var(ADC_value_1);
52          UART1_Send_String(" -> ");
53          UART1_Print_Var(ADC_voltage_1);
54          UART1_Send_String(" mV\r\n");
55
56          UART1_Send_String("\r\n");
57          delay_ms(500);
58      }
59  }
```

5.2 Génération d'un signal PWM

Des recherches sur internet nous ont permis de trouver le tutoriel suivant : Tutoriel de CircuitDigest. On en retire le code suivant :

```
1   #include "STM8S.h"
2
3   void delay_ms(int ms) // Function Definition
4   {
5       for (int i = 0; i <= ms; i++)
6           for (int j = 0; j < 120; j++) // Nop = Fosc/4
7               _asm("nop");            // Perform no operation //assembly code
8   }
9
10  void main(void)
11  {
12      GPIO_DeInit(GPIOD);
13      TIM2_DeInit();
14      GPIO_Init(GPIOD, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST);
15      TIM2_OC1Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000, TIM2_OCPOLARITY_HIGH);
16      TIM2_TimeBaseInit(TIM2_PRESCALER_1, 500);
17      TIM2_Cmd(ENABLE);
18      while (TRUE)
19      {
20          signed int pwm_duty = 500;
21          TIM2_SetCompare1(pwm_duty);
22      }
23  }
```

Dans la fonction main, les fonctions de configuration et d'initialisation suivantes sont appelées :

- `GPIO_DeInit(GPIOD)` : réinitialisation des paramètres du port D
- `TIM2_DeInit()` : réinitialisation du Timer 2
- `{GPIO_Init(GPIOD, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST)` : configuration de la broche PD4 en sortie push-pull avec une vitesse élevée
- `TIM2_OC1Init(...)` : configuration du Timer 2 pour générer un signal PWM en mode 1 avec un rapport cyclique initial de 1000 (50%)
- `TIM2_TimeBaseInit(TIM2_PRESCALER_1, 500)` : configuration de la base de temps du Timer 2 avec un prédiviseur de 1 et une valeur d'autorechargement de 500
- `TIM2_Cmd(ENABLE)` : activation du Timer 2

Enfin, la partie :

```
while (TRUE)
    {
        signed int pwm_duty = 500;
        TIM2_SetCompare1(pwm_duty);
    }
```

Génère un signal PWM de rapport cyclique `pwm_duty`/100

6 Bibliographie

Liste des sites utilisés pour la réalisation du projet :

— Overleaf, guide d'utilisation du LaTeX
— ChatGPT, modèle de langage automatisé
— Manuel de Circuitikz
— Manuel des Schéma-blocs avec PGF/TIKZ