

PROJET EIS - PX222

Guillemot MOUSSU et Rémi MAZZONE

2° semestre - 2022-2023

Table des matières

1 Séance 0 - 21/03/2023	3
1.1 Notes prises au tableau	3
1.2 Idées de plan de recherches	3
2 Séance 1 - 05/04/2023	4
2.1 Reprise des éléments du semestre précédent	4
2.2 Utilisation de Matlab	4
2.3 Modélisation point par point du système	6
2.3.1 Modélisation de l'amplificateur	6
2.3.2 Modélisation d'un régulateur PD	7
2.3.3 Modélisation d'un régulateur PID	7
2.3.4 Modélisation du système complet (PD)	8
2.4 Conclusions pour cette séance	8
3 Travail entre les séance 1 et 2	8
3.1 Correction des éléments de la séance précédente	8
3.1.1 Matlab	8
3.1.2 Modélisation point par point du système	8
3.2 Simulation de l'amplificateur	9
4 Séance 2 - 18/04/2023	10
4.1 Réflexion sur la numérisation du correcteur	10
4.2 Prise en main du STM8s	10
4.3 Premier essai de code	10
4.4 Essais avec Arduino	10
4.5 Essais avec STVD	11
4.6 Conclusions pour cette séance	12
5 Travail entre les séances 2 et 3	12
5.1 Lecture d'une entrée analogique	12
5.2 Génération d'un signal PWM	13
5.2.1 Code	13
5.2.2 Description du code	13
5.2.3 Fichiers nécessaires	14
6 Séance 3 - 03/05/2023	14
6.1 Partie électronique numérique	14
6.1.1 Récupération d'une entrée	14
6.1.2 Génération de signaux PWM	14
6.1.3 Brouillon pour un régulateur P	14
6.2 Partie électronique analogique	16
6.2.1 Matériel utilisé	16

6.2.2	Calcul du point de fonctionnement	16
6.2.3	Résistance de la bobine	17
6.2.4	Inductance de la bobine	17
6.3	Conclusions	18
7	Séance 4 - 02/06/2023	18
7.1	Choix des composants	18
7.2	Réalisation	18
7.3	Pistes pour la suite	19
8	Bibliographie	20

Table des figures

1	Notes séance 0	3
2	Modélisation de la bobine	4
3	Bode obtenu	5
4	Nyquist obtenu	5
5	Schéma bloc du système	6
6	Amplificateur à AOP	6
7	Modélisation du bloc amplificateur	6
8	Régulateur PD	7
9	Modélisation du bloc régulateur (PD)	7
10	Régulateur PID	7
11	Modélisation complète avec retour (PD)	8
12	Résultat idéal de la simulation	9
13	Schéma LTSPICE de l'amplificateur	9
14	Simulation de l'amplificateur	9
15	Schéma bloc du système, version électronique numérique	10
16	Brochage du STM8s discovery	10
17	Bobine alimentée	16
18	Courbe de l'intensité mesurée en fonction de la distance	16
19	Mesure de la résistance	17
20	Mesure de l'inductance	17
21	Réponse de la bobine	17
22	Mesure du montage amplificateur	18

1 Séance 0 - 21/03/2023

1.1 Notes prises au tableau

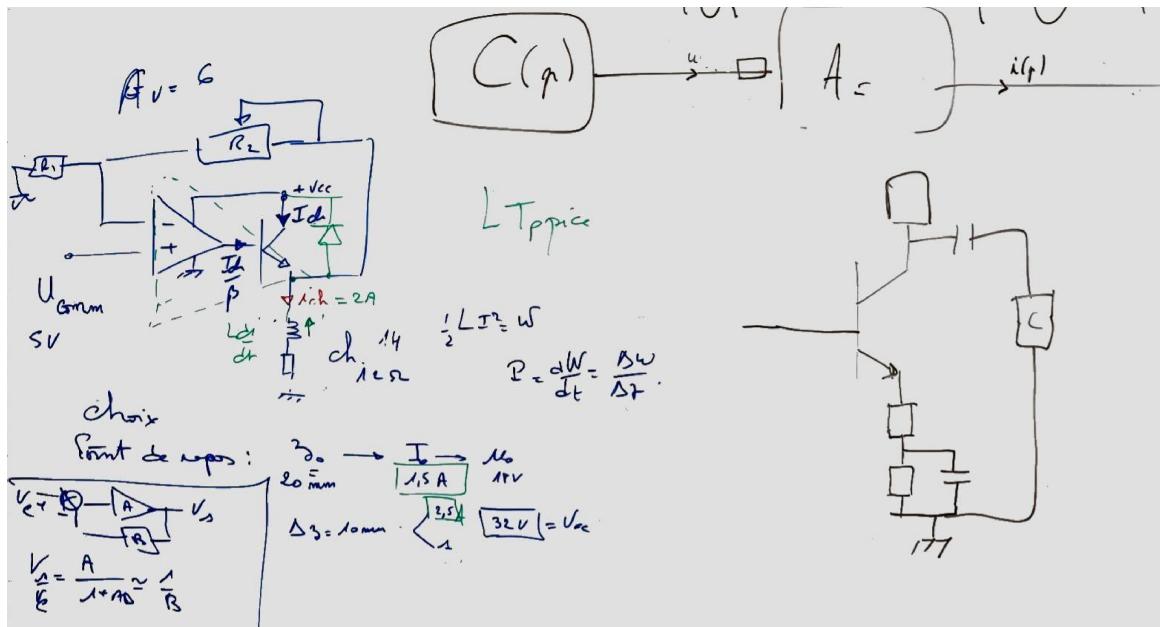


FIGURE 1 – Notes séance 0

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 :

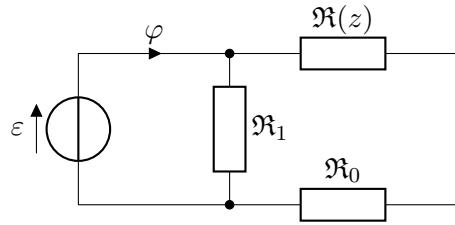


FIGURE 2 – Modélisation de la bobine

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

Avec les valeurs suivantes :

$$I_0 = 2A, Z_0 = 22\text{mm}, L_1 = 6.73H, \alpha = 2.06, m = 35.8g$$

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

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 :

```

1 IO = 2;
2 Z0 = 22 * 10^-3;
3 L1 = 6.73;
4 alpha = 2.06;
5 ki = (IO * L1 * alpha) / (1 + alpha * Z0) ^2;
6 kz = -(IO ^2 * L1 * alpha ^2) / (1 + alpha * Z0) ^3;
7 m = 35.8;
8
9 num = ki ;
10 den = [(-m) 0 kz];
11 sys = tf(num,den);
12
13 figure;
14 bode(sys)
15 grid on
16 figure;
17 nyquist(sys)
18 grid on

```

On obtient donc le bode suivant :

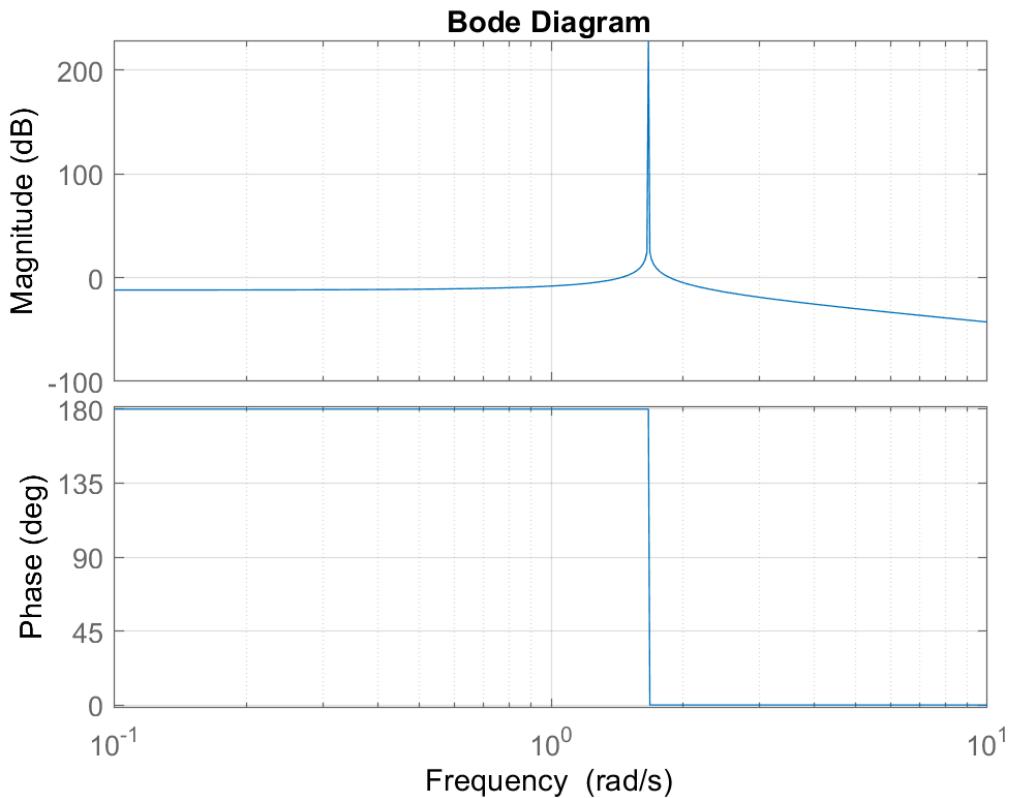


FIGURE 3 – Bode obtenu

On trace ensuite le diagramme de Nyquist :

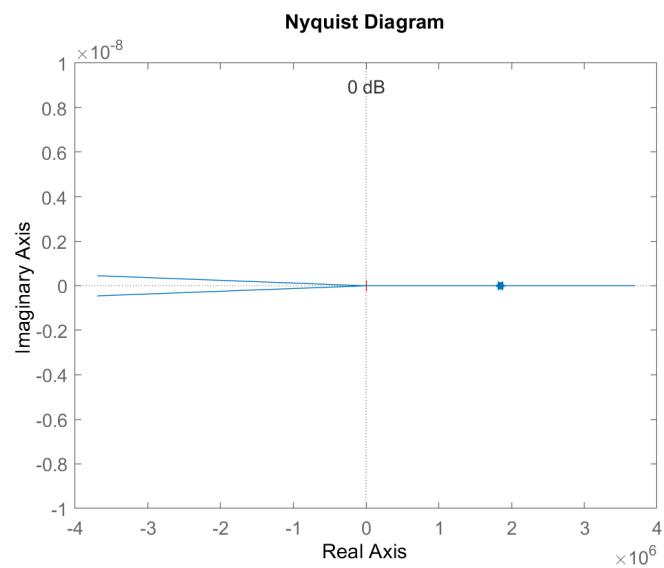


FIGURE 4 – Nyquist obtenu

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 :

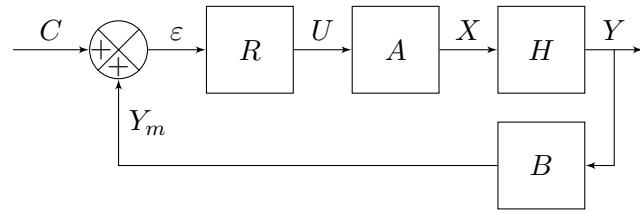


FIGURE 5 – Schéma bloc du système

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 :

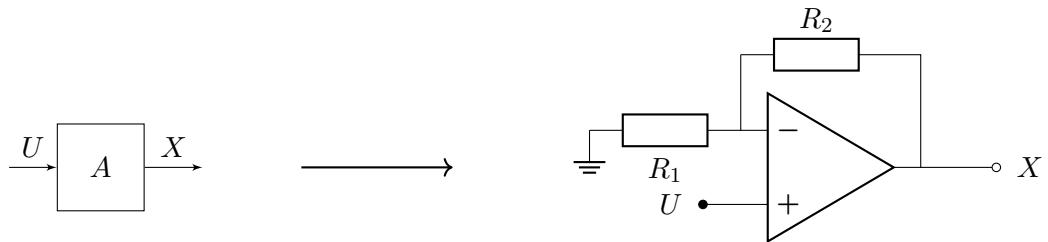
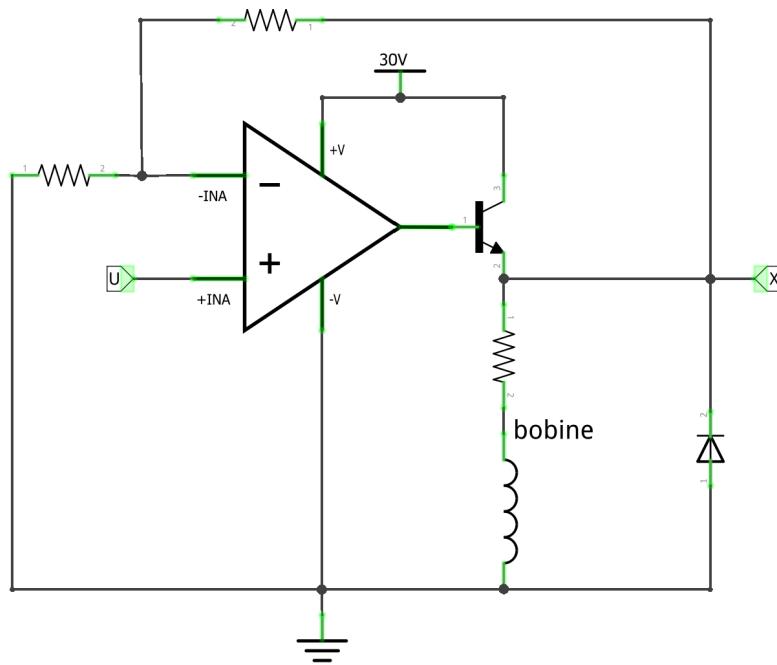


FIGURE 6 – Amplificateur à AOP

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



fritzing

FIGURE 7 – Modélisation du bloc amplificateur

2.3.2 Modélisation d'un régulateur PD

On a le schéma classique d'un régulateur PD :

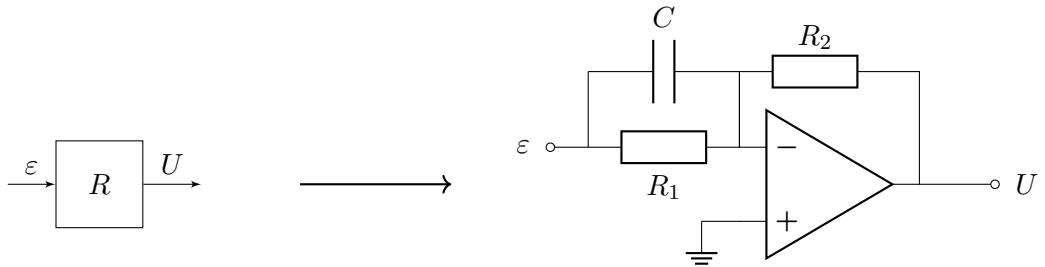
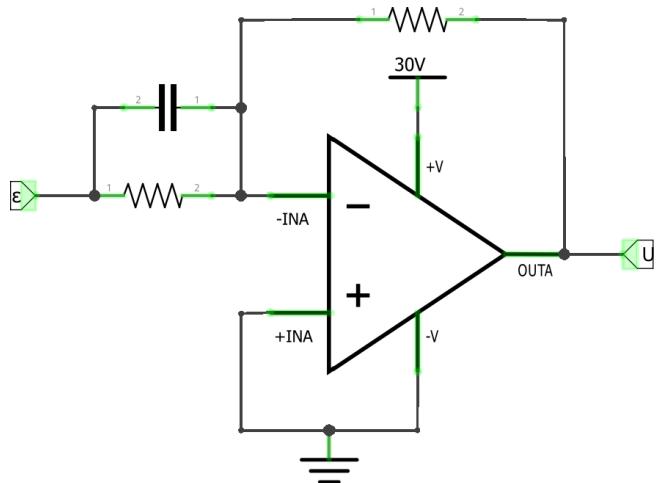


FIGURE 8 – Régulateur PD

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



fritzing

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 :

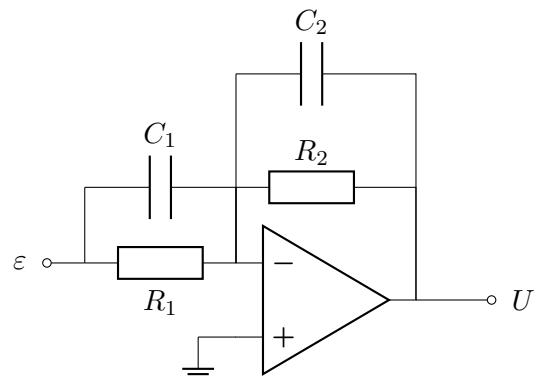


FIGURE 10 – Régulateur PID

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

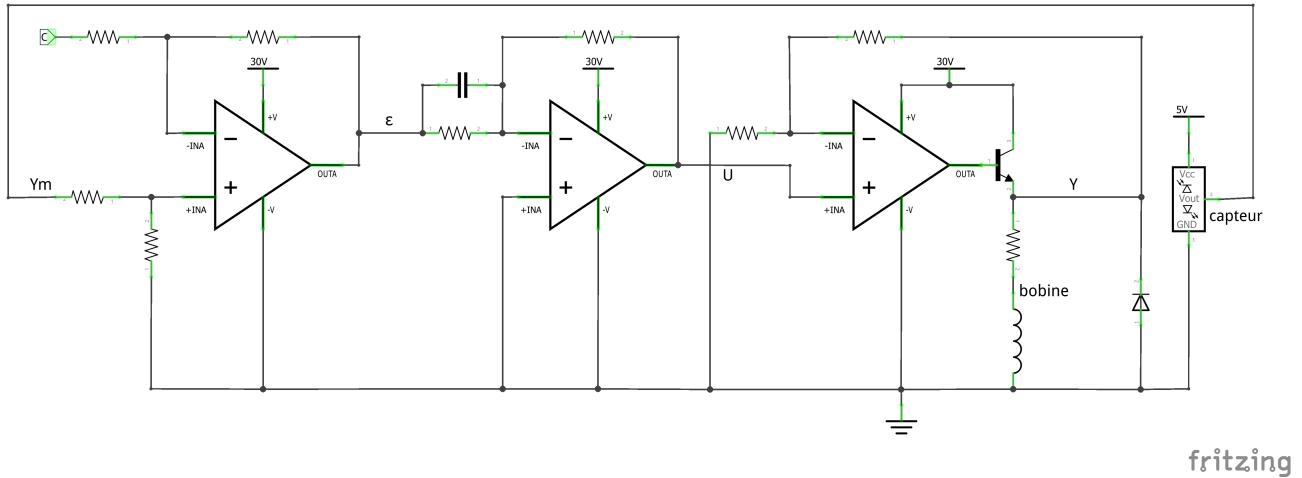


FIGURE 11 – Modélisation complète avec retour (PD)

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 été 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

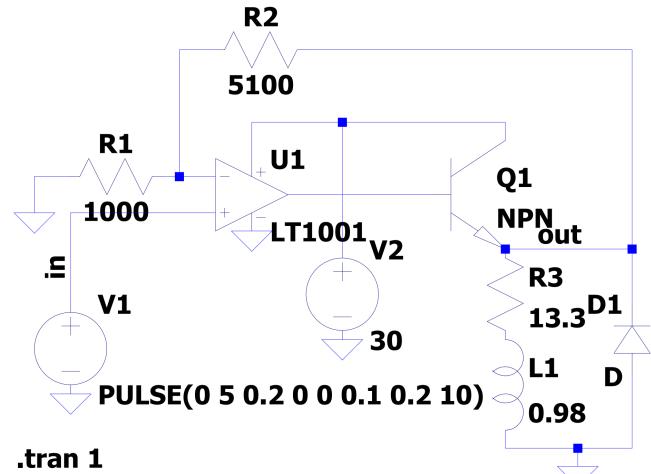
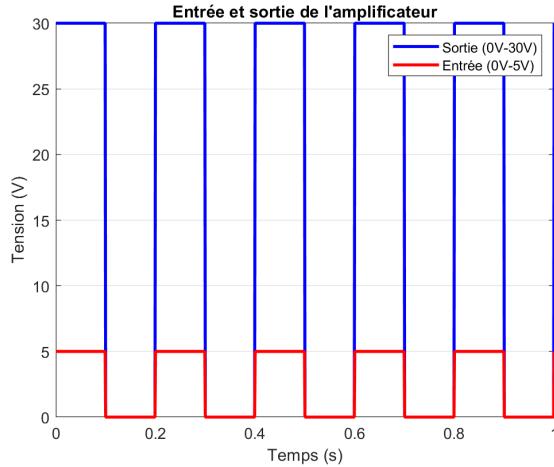
La figure 5 est maintenant rendue en \LaTeX

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

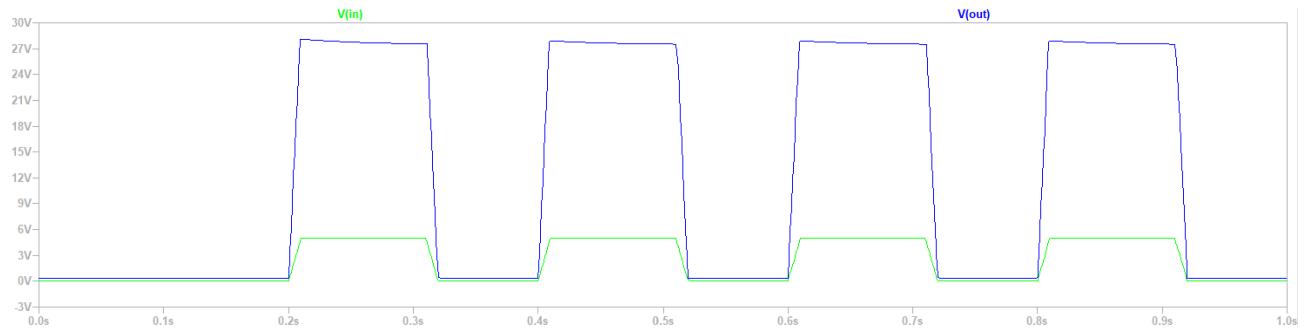
Le point 2.3.4 a été séparé plus plus de clarté. La figure 11 y a été ajoutée

3.2 Simulation de l'amplificateur

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



Et on obtient le résultat suivant :



Remarques :

- On a utilisé un AOP quelconque (le LT1001 en l'occurrence)
- 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 :

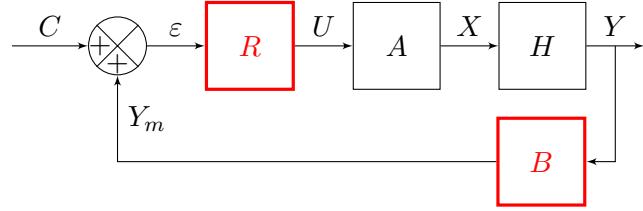


FIGURE 15 – Schéma bloc du système, version électronique numérique

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 :

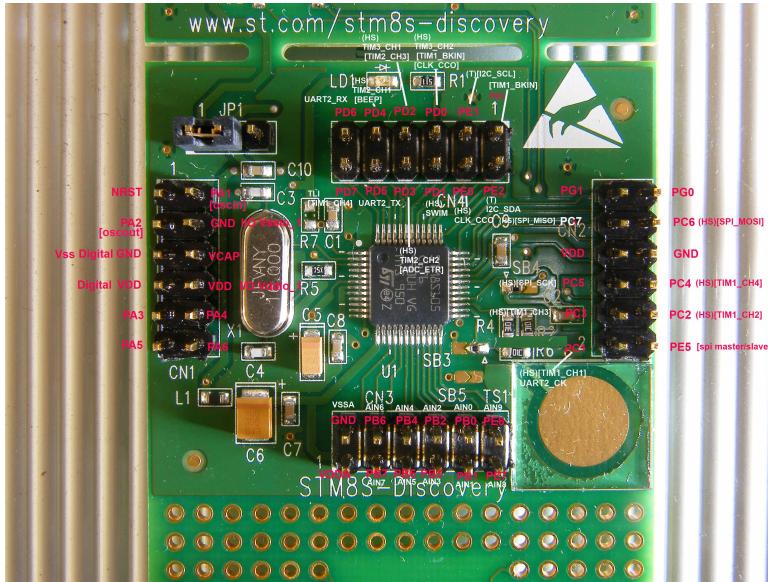


FIGURE 16 – Brochage du STM8s discovery

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, notamment 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 :

```
1 Installing platform sduino:stm8@0.5-pre2
2 Failed to install platform: sduino:stm8.
3 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 :

```
1 stm8/
2     #include 'mapping.inc'
3     #include 'stm8s105c_s.inc'
4
5     segment 'rom'
6 debut
7         bset PD_DDR,#3
8         bset PD_CR1,#3
9         bres PD_ODR,#3
10        mov TIM4_PSCR,#6
11        mov TIM4_ARR,#155
12        bset TIM4_IER,#0
13        bset TIM4_CR1,#0
14        rim
15        bset ADC_CR1,#0
16        mov ADC_CSR,#$00
17        bres ADC_CR2,#3
18
19 boucle_infinie
20     jra boucle_infinie
21
22     interrupt Timer4_ISR
23 Timer4_ISR
24     bres TIM4_SR,#0
25     bcpl PD_ODR,#3
26     bset ADC_CR1,#0
27 attente_fin_conversion
28     btjf ADC_CSR,#7,attente_fin_conversion
29     bres ADC_CSR,#7
30     ld A,#1
31     or A,ADC_DRH
32     ld TIM4_ARR,A
33     iret
34
35     segment 'vectit'
36     dc.l {$82000000+debut}
37     segment at 8064 'vectit'
38     dc.l {$82000000+Timer4_ISR}
39
40 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 les tutoriels suivants : Tutoriel de CircuitDigest pour ADC et Tutoriel de CircuitDigest pour la lecture série. On utilise le code de ces deux tutoriels pour produire le programme suivant :

```
1 #include "STM8S.h"
2 #include "stm8s103_adc.h"
3 #include "stm8s103_serial.h"
4
5 void main()
6 {
7     // Déclarations de variables
8     unsigned int ADC_value_1 = 0;
9     int ADC_voltage_1 = 0;
10
11    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, ENABLE); // Activer l'horloge
12        périphérique pour l'ADC
13    GPIO_Init(GPIOB, GPIO_PIN_0, GPIO_MODE_IN_FL_IT); // Initialiser le pin PB0 en entrée
14
15    Serial_begin(9600);
16    Serial_print_string("Bonjour !");
17    Serial_newline();
18    delay_ms(5000);
19
20    while (1)
21    {
22        ADC_value_1 = ADC_Read(AIN2);
23        ADC_voltage_1 = ADC_value_1 * (2.933); //(3000/1023 =~ 2.933)
24        Serial_print_string("BO: ");
25        Serial_print_string(ADC_value_1);
26        Serial_print_string(" -> ");
27        Serial_print_string(ADC_voltage_1);
28        Serial_print_string(" mV");
29        Serial_newline();
30        delay_ms(500);
31    }
}
```

On aura donc à priori besoin des fichiers suivants :

- Fichiers de base
- Fichier Serial.h
- Fichier ADC.h

5.2 Génération d'un signal PWM

5.2.1 Code

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)
4 {
5     int i;
6     int j;
7     for (i = 0; i <= ms; i++)
8     {
9         for (j = 0; j < 120; j++) {_asm("nop");}
10    }
11 }
12
13 void main(void)
14 {
15     GPIO_DeInit(GPIOD);
16     TIM2_DeInit();
17     GPIO_Init(GPIOD, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST);
18     TIM2_OC1Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000, TIM2_OCPOLARITY_HIGH);
19     TIM2_TimeBaseInit(TIM2_PRESCALER_1, 500);
20     TIM2_Cmd(ENABLE);
21     while (TRUE)
22     {
23         signed int pwm_duty = 500;
24         TIM2_SetCompare1(pwm_duty);
25     }
26 }
```

5.2.2 Description du code

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'autorecharge de 500
- TIM2_Cmd(ENABLE) : activation du Timer 2

Enfin, la partie :

```
1 while (TRUE)
2 {
3     signed int pwm_duty = 500;
4     TIM2_SetCompare1(pwm_duty);
5 }
```

Génère un signal PWM de rapport cyclique (`pwm_duty/10`)%

5.2.3 Fichiers nécessaires

Comme précédemment, on aura besoin des fichiers des Fichiers de base

6 Séance 3 - 03/05/2023

6.1 Partie électronique numérique

6.1.1 Récupération d'une entrée

Nous auront donc un capteur renvoyant des valeurs entre $0V$ et $3V$, correspondant à la distance avec la bille, dont il faudra récupérer les valeurs à l'aide d'un port analogique du STM8. Nous souhaitions récupérer les valeurs et les renvoyer vers un port série. Cependant, malgré l'installation de drivers et des bibliothèques nécessaires, que ce soit avec le moniteur Arduino ou avec Putty, nous n'avons pas réussi à récupérer les valeurs renvoyées par la carte. Le code que nous avions écrit plus haut compile cependant, et la carte arrive à l'exécuter. Nous pourrons vérifier son fonctionnement lorsque nous auront réussi à générer des signaux PWM.

6.1.2 Génération de signaux PWM

Nous avons donc tenté de générer des signaux PWM en utilisant notre code. Lorsque le programme est exécuté, on constate bien une tension entre PD4 et la masse, mais cette tension est constante. Nous avons tenté de nombreuses modifications sans succès. Nous avons donc voulu essayer des exemples présents en ligne, afin d'au moins se faire une idée sur la faisabilité de la génération de ces signaux.

Voici une liste des sites que nous avons utilisé, ainsi que le résultat que nous avons pu obtenir au final :

- Site de ST Microelectronics : ne contient pas un programme complet, ne compile pas, pas de résultats concluants
- Site de CircuitDigest : code dont nous nous étions inspiré, génère une tension positive constante mais pas de PWM
- Site de Mark-Stevens : ne contient pas les fichiers nécessaire à la compilation, pas de résultats concluants
- Site d'Embedded lab : compile avec quelques modifications, mais pas de tension en sortie des ports de la carte
- Github de stecman : nous n'avons pas eu le temps de regarder, mais semble intéressant

Au final nous n'avons pas encore de résultats. Il faudra par la suite soit trouver pourquoi nous n'arrivons pas à générer ces signaux avec un programme en C, soit essayer en assembleur. Le problème de l'assembleur est qu'il sera probablement plus difficile d'implémenter nos calculs de PID.

6.1.3 Brouillon pour un régulateur P

En parallèle, nous avons commencé à réfléchir à un programme permettant de récupérer l'entrée, d'appliquer un calcul (ici $Kp = 1$), et de produire une sortie avec un signal PWM. Ce code n'a pas encore été testé.

```
1 #include "stm8s.h"
2 #include "stm8s_adc1.h"
3 #include "stm8s_tim1.h"
4
5 // Parametre du regulateur proportionnel
6 #define KP 1.0
7
8 // Variables globales
9 volatile uint16_t consigne = 0;
10 volatile uint16_t adc_value = 0;
11
12 void main(void) {
13     // Initialisation des peripheriques (ADC, TIM1, etc.)
14     System_Init();
15
16     while (1) {
17         adc_value = ADC1_GetConversionValue(); // Lecture de la valeur ADC
18         regulation_P(); // Appel de la fonction de regulation proportionnelle
19     }
20 }
21
22 void System_Init() {
23     // Configuration ADC
24     ADC1_Init(ADC1_CONVERSIONMODE_SINGLE, ADC1_CHANNEL_0, ADC1_PRESSEL_FCPU_D2,
25             ADC1_EXTTRIG_TIM, DISABLE, ADC1_ALIGN_RIGHT, ADC1_SCHMITTTRIG_CHANNELO, DISABLE);
26     ADC1_StartConversion();
27
28     // Configuration TIM1 pour PWM
29     TIM1_TimeBaseInit(0xFFFF, TIM1_COUNTERMODE_UP, 0, 0);
30     TIM1_OC1Init(TIM1_OCMODE_PWM1, TIM1_OUTPUTSTATE_ENABLE, 0, TIM1_OCPOLARITY_HIGH);
31     TIM1_Cmd(ENABLE);
32     TIM1_CtrlPWMOutputs(ENABLE);
33
34     // Configuration du GPIO pour la sortie PWM
35     GPIO_Init(GPIOB, GPIO_PIN_0, GPIO_MODE_OUT_PP_HIGH_FAST);
36
37     // Configuration du GPIO pour l'entree analogique (ADC)
38     GPIO_Init(GPIOA, GPIO_PIN_0, GPIO_MODE_IN_FL_NO_IT);
39 }
40
41 // Fonction de mise a jour de la consigne
42 void set_consigne(uint16_t value) {
43     consigne = value;
44 }
45
46 // Fonction de regulation proportionnelle
47 void regulation_P() {
48     float erreur = (float)consigne - (float)adc_value;
49     float sortie = KP * erreur;
50
51     // Limiter la sortie entre 0 et la valeur maximale du compteur (ici 0xFFFF)
52     if (sortie < 0) {
53         sortie = 0;
54     } else if (sortie > 0xFFFF) {
55         sortie = 0xFFFF;
56     }
57
58     // Mise a jour de la sortie PWM
59     TIM1_SetCompare1((uint16_t)sortie);
}
```

6.2 Partie électronique analogique

6.2.1 Matériel utilisé

On utilise la bobine n°5

On utilise la boule "38" pesant 37.6g et mesurant 49.74mm de diamètre

6.2.2 Calcul du point de fonctionnement

On commence par relier la bobine à l'alim :

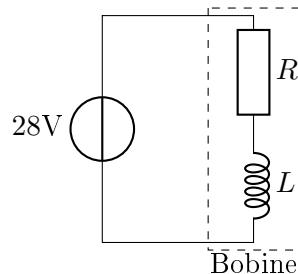


FIGURE 17 – Bobine alimentée

On a donc $V = 28V$, et on souhaite mesurer I en fonction z (distance entre le bas de la bobine et le centre de la boule)

On relève les valeurs suivantes :

Hauteur z	Courant I
0.0330m	0.542A
0.0375m	0.702A
0.0434m	0.922A
0.0462m	1.112A
0.0527m	1.342A
0.0555m	1.431A
0.0588m	1.601A
0.0613m	1.822A
0.0623m	1.851A
0.0626m	1.952A

On souhaite obtenir une interpolation affine de ces valeurs. On rentre dans MATLAB la commande `polyfit(z,I,1)`; et on obtient que :

$$I = 45.8215z - 1.0243$$

On produit donc la courbe suivante :

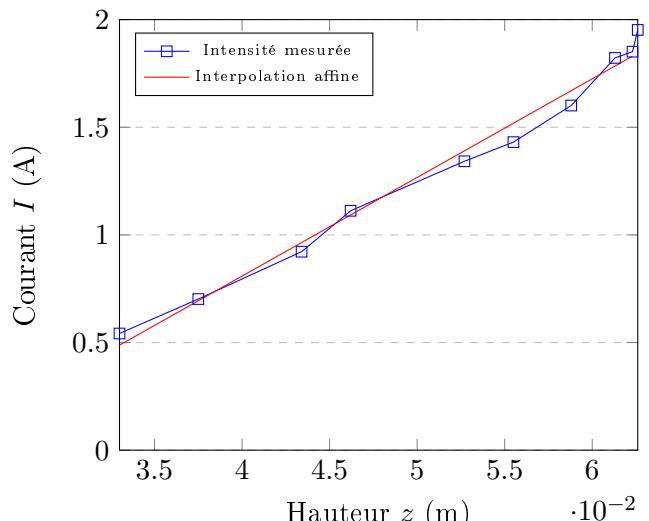


FIGURE 18 – Courbe de l'intensité mesurée en fonction de la distance

Au final, on choisit $I_0 = 1.3A$, ce qui nous donne $z_0 = 5.1cm$

6.2.3 Résistance de la bobine

On réalise le montage suivant :

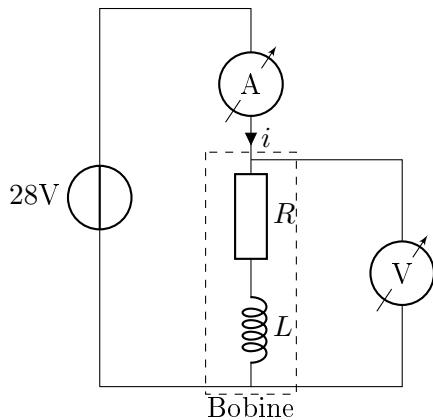


FIGURE 19 – Mesure de la résistance

Après avoir patienté un moment, pour laisser le temps aux caractéristiques de la bobine d'être stables, on mesure $U = 27.57V$ et $I = 1.95A$

On obtient donc la résistance de la bobine $R = 14.1\Omega$

6.2.4 Inductance de la bobine

On réalise le montage suivant :

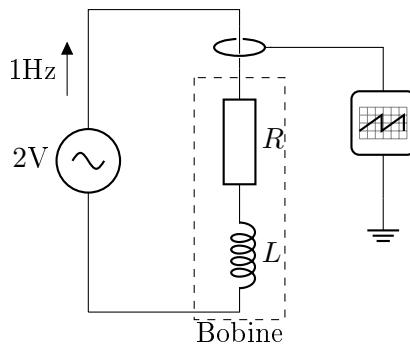


FIGURE 20 – Mesure de l'inductance

Après avoir patienté un moment, pour laisser le temps aux caractéristiques de la bobine d'être stables, on mesure un déphasage $\varphi = 150^\circ$

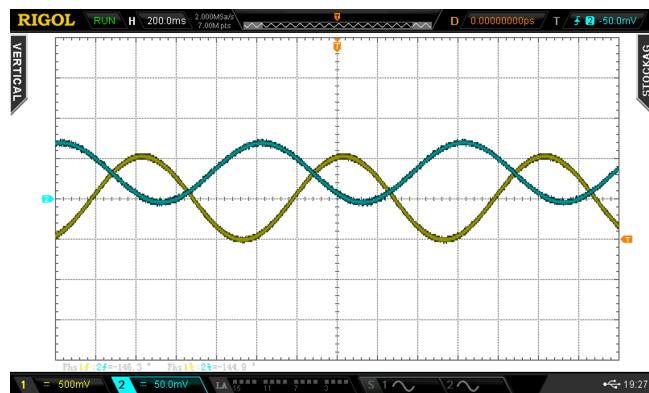


FIGURE 21 – Réponse de la bobine

$$\text{On calcule donc que l'inductance de la bobine vaut } -\frac{R \cdot \tan \varphi}{2\pi f} = -\frac{14.1 \cdot \tan 150}{2\pi} = 1.296H$$

6.3 Conclusions

Si nous avons peu avancé sur la partie numérique, nous avons pu caractériser la bobine, ce qui nous servira pour la suite dans la modélisation précise de notre système

7 Séance 4 - 02/06/2023

Lors de cette séance, nous nous sommes donnés l'objectif de réaliser l'ampli sur platine d'essai, afin de confirmer notre simulation

7.1 Choix des composants

- Transistor : TIP122
- AOP : TL081
- Diode : 1N4007

7.2 Réalisation

Nous avons réalisé le montage présenté dans la partie 3.2. Pour le moment le meilleur résultat que nous ayons obtenu est le suivant :

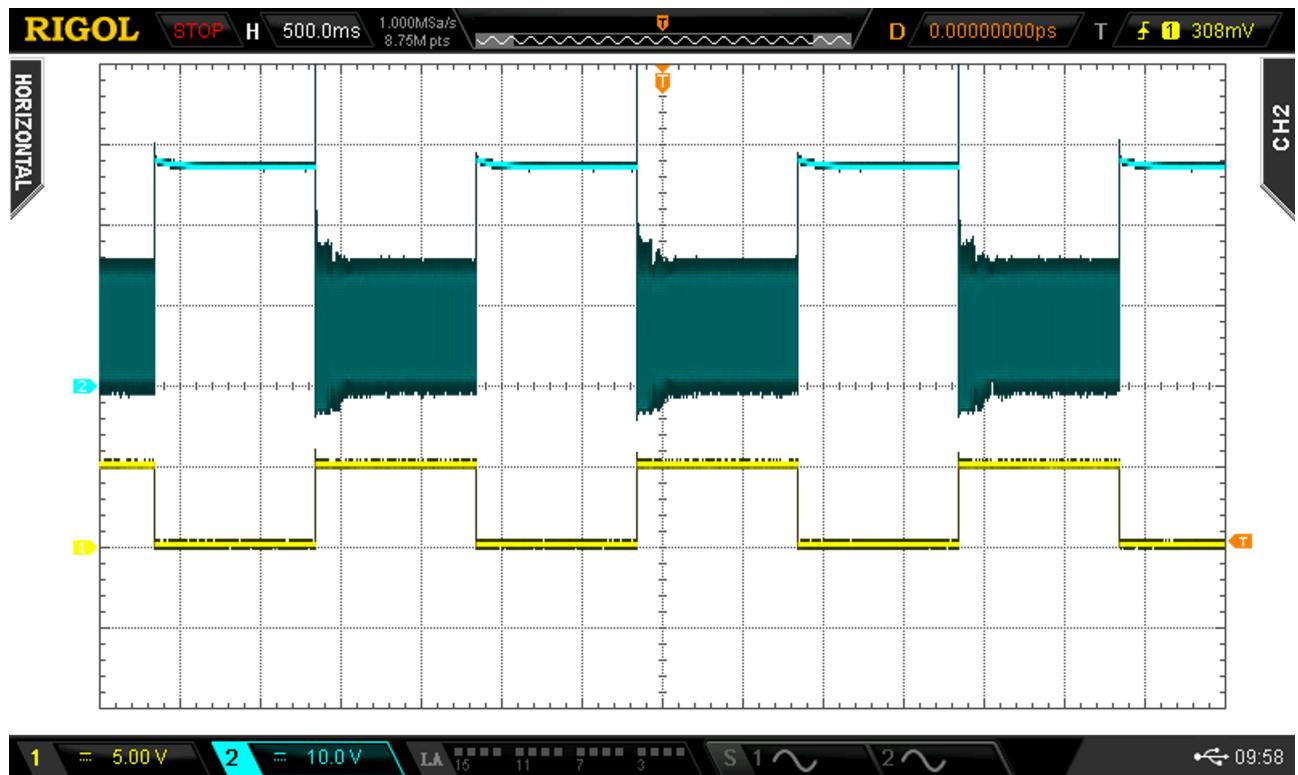


FIGURE 22 – Mesure du montage amplificateur

On a en jaune la commande et en bleu la sortie. Au final, on sent bien que la bille est plus ou moins attirée, mais nous n'avons pas l'effet "on-off" que nous souhaitions obtenir. Un autre problème auquel nous avons été confronté est que le courant délivré par le montage est beaucoup trop faible : entre 0.1A et 0.4A alors qu'avec la distance qu'on avait mise pour tester le montage, la bille ne pouvait décoller qu'à partir d'environ 1,6A

7.3 Pistes pour la suite

Il semblerait que le TL081 ne soit pas adapté à notre montage. On essaiera à la prochaine séance d'utiliser un LM342 à la place

8 Bibliographie

Liste des sites utilisés pour la réalisation du projet :

- Overleaf, guide d'utilisation du L^AT_EX
- ChatGPT, modèle de langage automatisé
- Manuel de Circuitikz
- Manuel des Schéma-blocs avec PGF/TIKZ
- Manuel des Schéma-blocs avec PGF/TIKZ
- Tutoriels de CircuitDigest pour STM8s