

ZG Masse Ressort

Coefficients :

Sécurité câblée et adaptation électronique - 1

Mise à jour de la conception mécanique - 1.5

Contrôle commande et acquisition - 1

IHM - 1

Post traitement - 1

Année scolaire : 2022-2023

Au cours du module Conception et Mécanique Vibratoire nous avons eu pour mission d'améliorer la maquette utilisée en zone généraliste 2. Ces améliorations ont pour but de faire faire plus de choses aux étudiants et ainsi leur permettre d'avoir une meilleure compréhension du sujet. Cette maquette est un système masse ressort piloté par un moteur, un capteur lidar est présent au niveau du moteur pour récupérer la distance jusqu'à la masse. Ce capteur est connecté à une carte électronique qui va elle même fonctionner avec une Interface homme Machine. L'objectif étant d'afficher les oscillations du système.

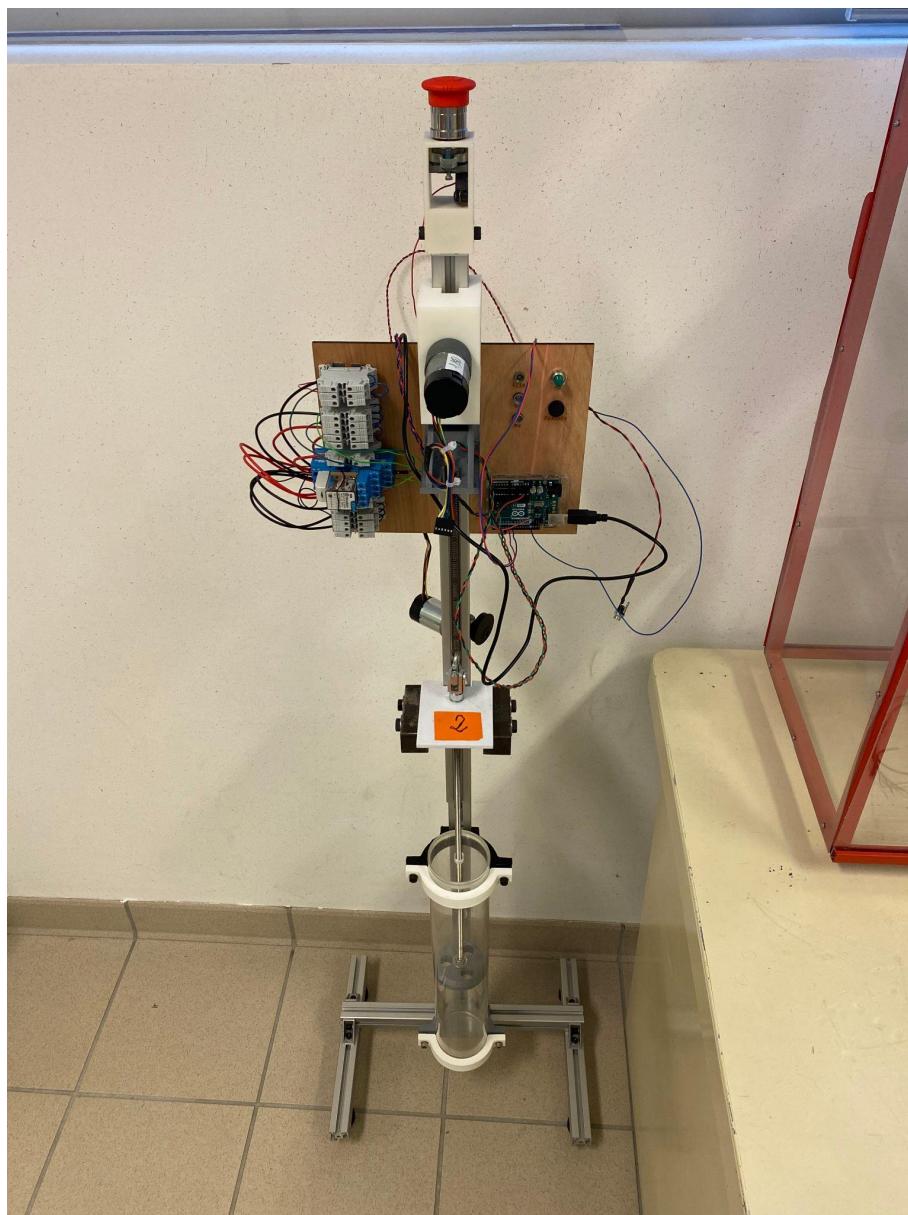


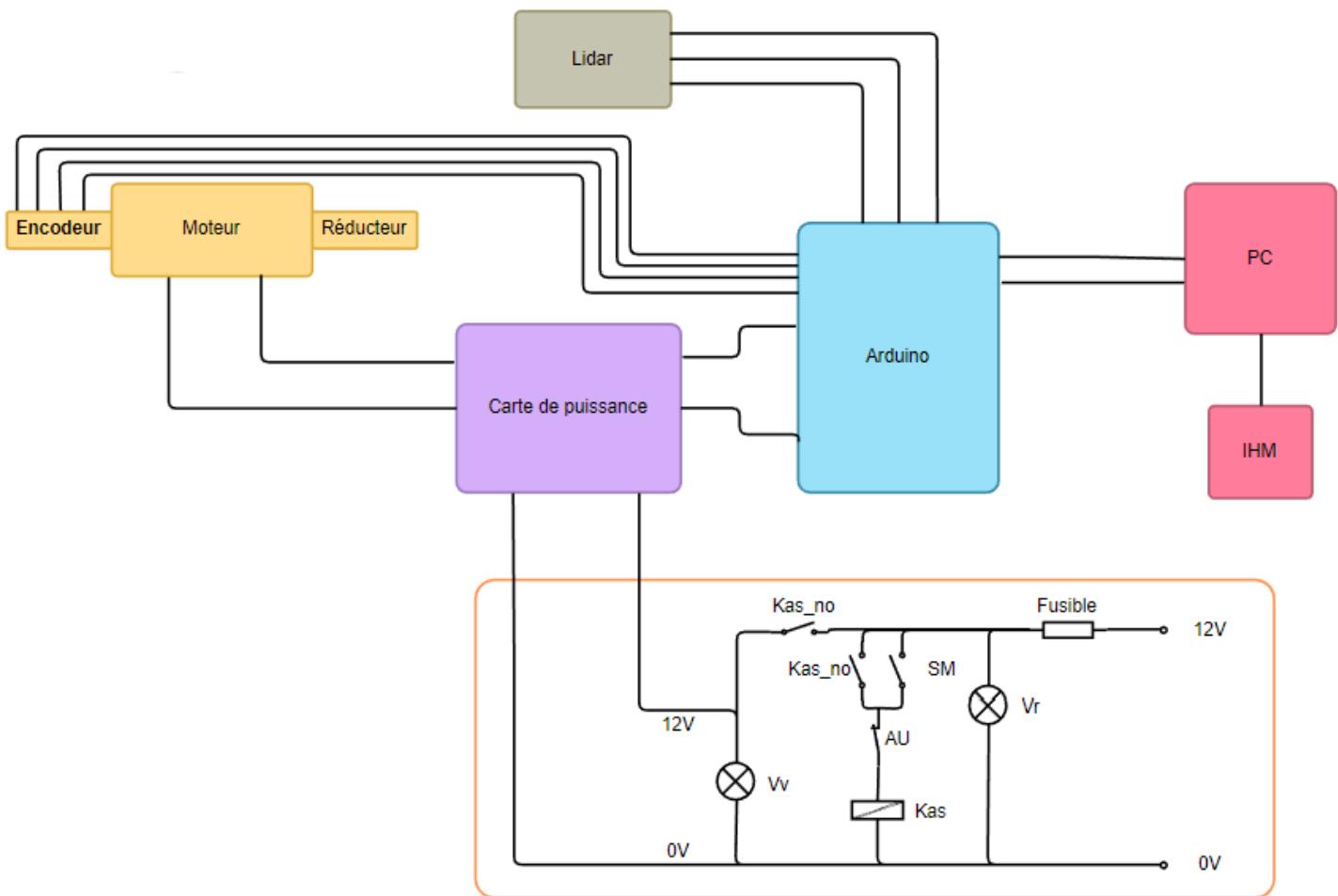
Table des matières

I. Sécurité câblée et adaptation électronique.....	5
I.1. Objectifs fixés.....	5
I.2. Etat initial de la tâche.....	6
I.3. Planification.....	6
I.4. Choix techniques.....	6
I.5. Bilan comparatif avec les objectifs initiaux.....	7
II. Mise à jour de la conception mécanique.....	8
I.1. Objectifs fixés.....	8
I.2. Etat initial de la tâche.....	8
I.3. Planification.....	9
I.4. Choix techniques.....	9
I.5. Bilan comparatif avec les objectifs initiaux.....	12
III. Simulation comportement vibratoire.....	13
I. Excitation libre.....	13
I.1. Paramétrage libre.....	13
I.2. Mise en équation libre et réponse temporelle.....	13
II. Excitation forcée.....	16
II.1. Paramétrage.....	16
II.2. Mise en équation.....	16
II.3. Réponse fréquentielle.....	17
III. Dimensionnement du moteur.....	18
III.1. Objectif fixé :	18
III.2. Etat initial de la tâche :	18
III.3. Choix techniques :	18
IV. Contrôle commande et acquisition.....	20
I.1. Objectifs fixés :	20
I.2. Etat initial de la tâche.....	20
I.3. Planification.....	20
I.4. Choix techniques.....	21
I.5. Travail réalisé.....	21
I.6. Perspective d'évolution.....	25
V. IHM.....	26
I.1. Objectifs fixés.....	26
I.2. Etat initial de la tâche.....	26
I.3. Planification.....	27
I.4. Choix techniques.....	28

I.5. Travail réalisé.....	29
I.6. Perspective d'évolution.....	31
VI. Post traitement.....	32
I.1. Objectifs fixés.....	32
I.2. Etat initial de la tâche.....	32
I.3. Planification.....	32
I.4. Choix techniques.....	32
I.5. Perspective d'évolution.....	41

I. Sécurité câblée et adaptation électronique

Schéma de câblage du projet



I.1. Objectifs fixés

L'objectif du semestre est de réaliser une maquette possédant un circuit de sécurité câblée externe à la carte électronique, en utilisant un relais industriel et un bouton d'arrêt d'urgence. Le câblage doit être réalisé en utilisant des borniers “industriels”.

I.2. Etat initial de la tâche

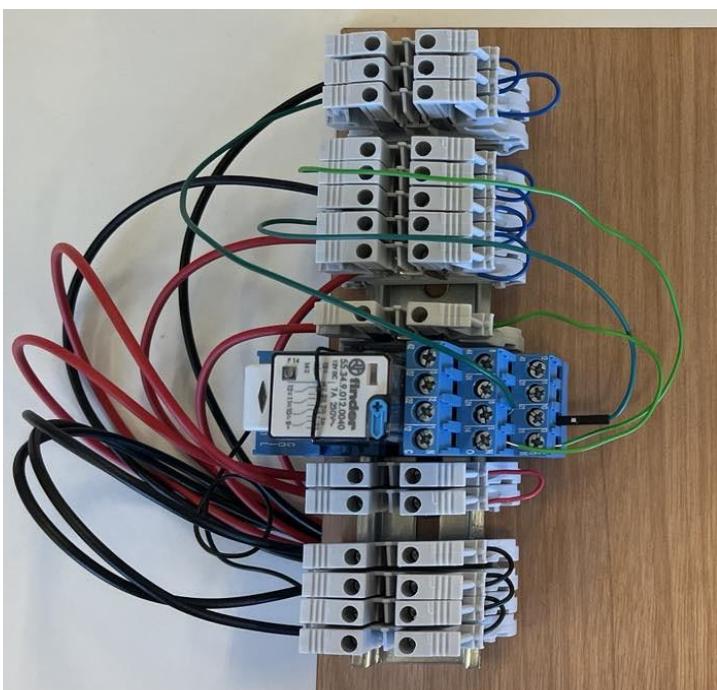
Initialement le projet fonctionnait avec une carte stm32 qui venait directement se placer sur une carte électronique réalisée au cours du semestre précédent. Cette carte électronique permettait de faire la connexion entre la carte stm32, la carte de puissance et la sécurité câblée. Etant donné que nous avions pour mission d'utiliser une carte arduino pour remplacer la carte stm32 et de réaliser un câblage externe, la carte électronique ne nous a pas servi.

I.3. Planification

TITRE DE LA TÂCHE	Noms	SEMAINE 9	SEMAINE 10	SEMAINE 11	SEMAINE 12	SEMAINE 13	SEMAINE 14	SEMAINE 15	SEMAINE 18	SEMAINE 19	SEMAINE 20	SEMAINE 21	SEMAINE 22	SEMAINE 23	SEMAINE 24
4 Sécurité câblée		27/02-03/03	06/03-10/03	13/03-17/03	20/03-24/03	27/03-31/03	03/04-07/04	10/04-14/04	01/05-05/05	08/05-12/05	15/05-19/05	22/05-26/05	29/05-02/06	05/06-09/06	12/06-16/06
Circuit de sécurité câblée	Jules/Maelys														
Mise en place relais industriel	Jules/Maelys														
Mise en place arrêt d'urgence	Jules/Maelys														
Modification plaque	Jules/Maelys														
Contact relais sécurité	Jules/Maelys														
Coupe alimentation avec relais sécurité	Jules/Maelys														

I.4. Choix techniques

Nous avons choisi d'utiliser des borniers afin que les étudiants puissent câbler eux même la sécurité câblée. Certains de ces borniers sont préalablement connectés entre eux afin que les branchements à effectuer ne se fassent que d'un côté des borniers, de façon à rendre le câblage plus simple et concis.



Il est possible de voir sur la photo que les câbles passent à l'arrière de la plaque et reviennent se câbler directement sur un seul côté de bornier.

Les câbles du relais sont verts afin de bien différencier cette partie du circuit.

Tous les borniers sont disposés sur un rail afin de les fixer directement à la plaque.

L'intégration d'un bouton d'arrêt d'urgence a aussi été réalisé de façon à couper entièrement l'alimentation du système en cas de problème. L'intégration électronique du bouton se fait directement sur les borniers, l'intégration physique du bouton sera expliquée dans la partie II Mise à jour de la conception mécanique.

Pour ce qui est de la carte électronique utilisée, on utilise une carte arduino UNO et pour commander le moteur, nous voulions se passer d'un circuit électronique externe et avoir tous sur la carte arduino. Nous avons trouvé un shield moteur qui vient se brancher directement sur la carte. Malheureusement, nous avons rencontré des problèmes lors de son utilisation, et nous n'avons pas réussi à résoudre ces problèmes dans le temps imparti; nous sommes donc retourné vers une carte de puissance utilisée précédemment. Comme cela s'est fait sur la fin, nous n'avons pas eu le temps d'intégrer proprement la carte puissance au pupitre.

I.5. Bilan comparatif avec les objectifs initiaux

Les objectifs initiaux ont tous pu être réalisés. Nous avons pu réaliser la sécurité câblée sur borniers industriels ainsi que la mise en place de l'arrêt d'urgence. Nous avons également relié les borniers entre eux afin de permettre aux étudiants de câbler sur un seul côté des borniers, avec un seul câble par bornier.

II. Mise à jour de la conception mécanique

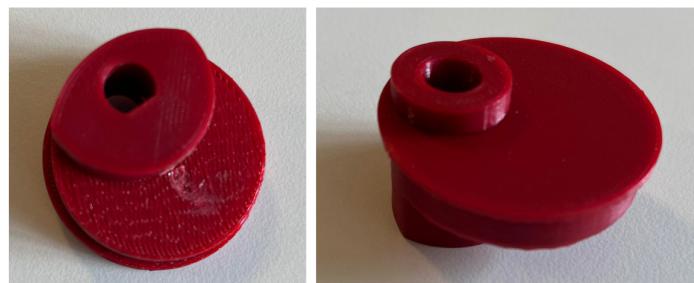
I.1. Objectifs fixés

Les objectifs de conception mécanique concernent les mises à jour de pièces ou d'éléments de pièces. Dans un premier temps, on retrouve l'intégration d'un bouton d'arrêt d'urgence sur le haut de la maquette. Ensuite il y a la mise en jour du système d'excitation, qui comprend de nombreuses pièces comme l'excentrique, pour passer d'un moteur 24V à 12V. Enfin la création d'une nouvelle interface de commande étant donné la nouvelle contrainte d'utiliser une carte arduino.

Par la suite nous avons vu d'autres objectifs liés tels que la création d'une fiche sur la mise en place d'un insert ainsi que la modification du roulement.

I.2. Etat initial de la tâche

Initialement, la maquette ne possédait pas de bouton d'arrêt d'urgence. La maquette fonctionnait avec un moteur 24V. De plus, dans le système d'excitation permettant de faire osciller le système, il y avait un problème au niveau de l'une des pièces, l'excentrique, qui ne convenait pas (figure ci-dessous), il était donc à refaire.



Excentrique d'origine

Le projet fonctionnait initialement avec une carte stm32, et nous avions pour objectif de la remplacer par un arduino. La plaque métallique ne correspondait donc pas à ce dont nous avions besoin.

I.3. Planification

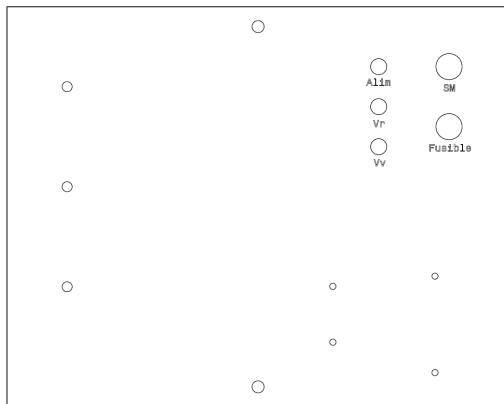
TITRE DE LA TÂCHE	Noms	SEMAINE 9	SEMAINE 10	SEMAINE 11	SEMAINE 12	SEMAINE 13	SEMAINE 14	SEMAINE 15	SEMAINE 16	SEMAINE 17	SEMAINE 20	SEMAINE 21	SEMAINE 22	SEMAINE 23	SEMAINE 24
		27/02-03/03	06/03-10/03	13/03-17/03	20/03-24/03	27/03-31/03	03/04-07/04	10/04-14/04	01/05-05/05	08/05-12/05	15/05-19/05	22/05-26/05	29/05-02/06	05/06-09/06	12/06-16/06
2 CAO et intégration physique															
Mise en place des meplats	Jules/Maelys														
Modification du roulement	Jules/Maelys														
Fiche mise en place d'un insert	Jules/Maelys														
Modification de la pièce défectueuse	Jules/Maelys														
Changement moteur	Jules/Maelys														
Intégration physique	Jules/Maelys														

I.4. Choix techniques

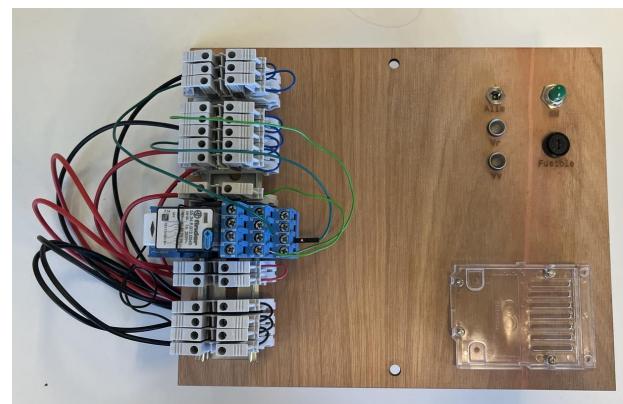
Pour faire ces changements, nous avons donc fait plusieurs choix. Le moteur a été modifié, nous passons d'un moteur 24V à un moteur de 12V, mais nous n'avons pas eu besoin de modifier son support étant donné qu'il était déjà imprimé.

La plaque qui fait office de poste de commande à dû être refaite, nous avons ici opté pour une plaque de contreplaqué en bois de 5mm d'épaisseur, découpée à l'aide de la découpeuse laser.

Il est possible de voir sur la plaque la disposition des différents éléments tels que le bouton, les voyants, le fusible, et l'alimentation sur la droite. On peut également voir une plaque en plastique représentant l'endroit de fixation de l'arduino, et le câblage avec les borniers sur le rail à gauche.

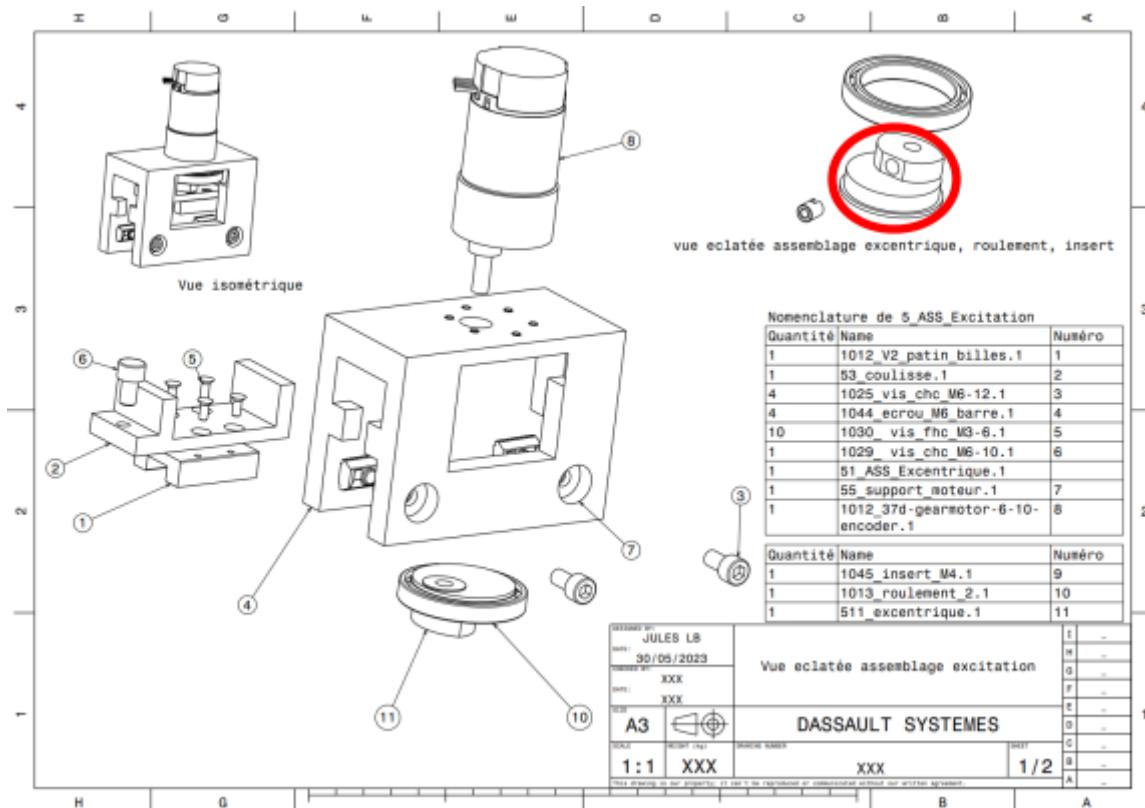


Mise en plan plaque



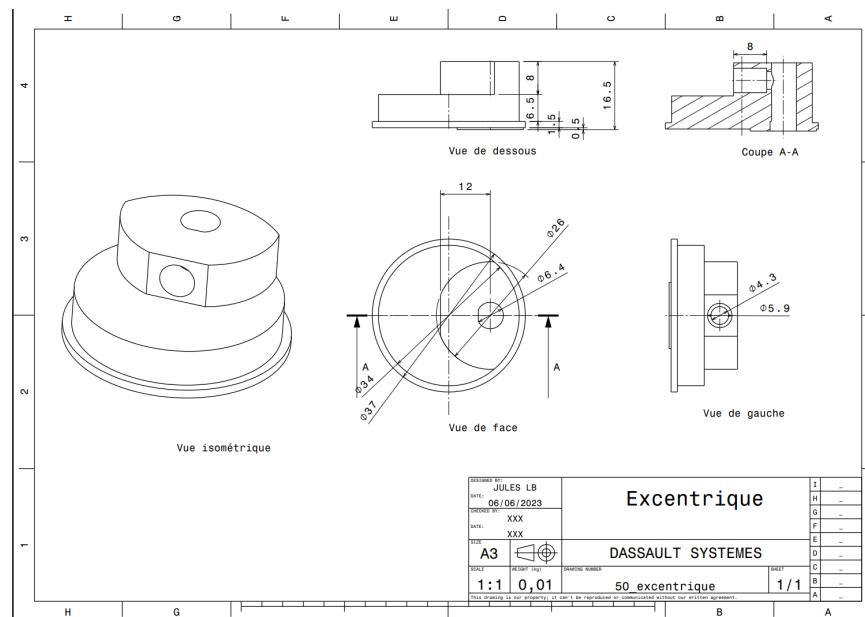
Plaque

La maquette fonctionne avec un système d'excitation composé de plusieurs pièces permettant de faire osciller le système. L'une de ces pièces a été sujet à des modifications, il s'agit de l'excentrique (entouré en rouge ci-dessous).



Nous avions un excentrique dans le projet mais qui nécessitait des modifications. Il nous a donc fallu plusieurs essais afin de trouver des dimensions pour lesquelles le montage était serré mais démontable.

Plusieurs essais d'impression ont donc été effectués. Les principaux paramètres à modifier étaient les diamètres des trous permettant la mise en place de l'insert et de l'axe moteur.



L'excentrique a été imprimé à plusieurs reprises afin de trouver les dimensions qui conviennent le mieux.



Excentrique final

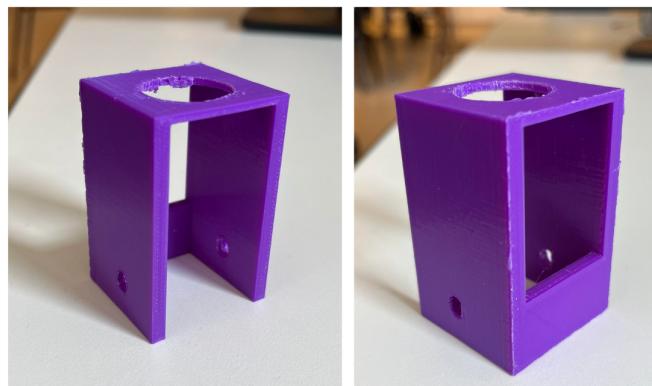
Lors des différentes impressions, nous avons gardé les mêmes paramètres afin de modifier uniquement les dimensions. Un changement de paramètres et de dimensions au même moment pouvant se montrer contre productif.

Les paramètres utilisés pour l'impression de l'excentrique et du support sont les suivants :

Paramètres	Imprimante	Filament	Remplissage	Pattern	Support
Excentrique	Prusa	PLA	90	Grid	Partout
Support bouton d'arrêt d'urgence	Prusa	PLA	90	Grid	Partout

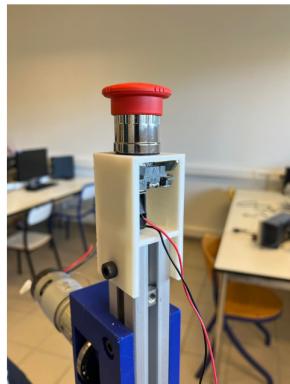
Et enfin, un support pour maintenir le bouton d'arrêt d'urgence a été modélisé et imprimé pour être fixé sur le dessus de la maquette.

Concernant ce support, nous avons fait un premier essai valide (cf figure ci-dessous) mais que nous avons souhaité améliorer afin d'éviter d'abîmer le bouton d'arrêt d'urgence. Nous avons donc rajouté un fond à ce support de façon à ce que la barre qui maintient le système soit en butée sur le support. Nous avons également renforcé la pièce en augmentant certaines épaisseurs.



1er essai support bouton arrêt d'urgence

Après ces modifications nous avons donc pu mettre en place l'arrêt d'urgence sur la maquette. Il est fixé sur le dessus, à l'aide de 2 vis et 2 écrous qui permettent de le fixer à la barre. Il est possible de voir sur la figure ci-dessous le fond rajouté sur le support afin que le support soit en butée sur la barre. Les dimensions du support permettent un montage serré sur la barre.



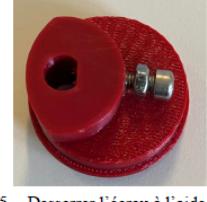
Support bouton arrêt d'urgence

I.5. Bilan comparatif avec les objectifs initiaux

Les objectifs initiaux ont tous pu être réalisés. Nous avons pu réaliser le support afin de positionner l'arrêt d'urgence sur la maquette, modifier l'excentrique pour qu'il réponde aux attentes du système, changer le moteur et faire une nouvelle interface de commande.

Nous avons également créé une fiche sur la mise en place d'un insert (cf figure ci-dessous) afin de rendre la tâche plus facile pour les prochains groupes.

Mise en place d'un insert

			
1. Placement de l'insert sur une vis avec un écrou comme ci-dessus :	2. Verrouiller l'écrou contre l'insert à l'aide d'une pince	3. Placer la pièce dans un étau	4. Visser l'insert à l'endroit souhaité le plus droit possible
		7. L'insert est en place.	
5. Desserrer l'écrou à l'aide d'une pince	6. Retirer la vis et l'écrou		

III. Simulation comportement vibratoire

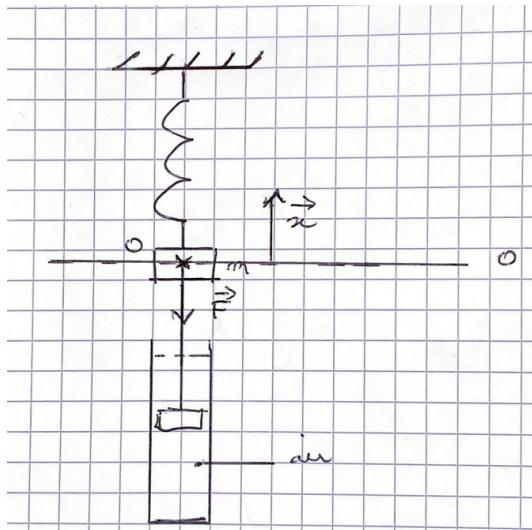
Notre simulation a été simulée en Python sur jupyter Notebook et est basée sur les données suivantes :

- masse m : 1.6 [Kg]
- raideur du ressort k : 100 [N/m]
- la pulsation propre “omega_zéro” : $\sqrt{k/m}$ [rad/s]

I. Excitation libre

I.1. Paramétrage libre

Laissons vibrer notre système à vitesse initiale nulle et position de lancer à 20 cm de l’origine, on obtient le paramétrage suivant :



I.2. Mise en équation libre et réponse temporelle

Par la suite, en isolant la masse m , le bilan des forces extérieures se présente comme suit :

- Force extérieure : $\vec{F}(t)\vec{x}$
- Force dûe à la raideur du ressort : $\vec{F}_{ressort} = -kx\vec{x}$
- Force de frottement visqueux due à l'amortisseur : $\vec{F}_{amoortisseur} = -b\dot{x}\vec{x}$

Ainsi, on obtient une équation dynamique de cette forme :

$$\ddot{x} + \frac{b}{m}\dot{x} + \omega_0^2 x = 0$$

En résolvant cette équation différentielle, on constate que le discriminant est négatif. Les racines complexes de l'équation sont alors les suivantes :

$$\delta < 0$$

$$\begin{cases} x(t) = e^{-\xi\omega_0 t} \left(x_0 \cos(\omega_0 \sqrt{1-\xi^2} t) + \frac{\dot{x}_0}{\omega_0 \sqrt{1-\xi^2}} \sin(\omega_0 \sqrt{1-\xi^2} t) \right) \\ r_{1,2} = (-\xi \pm i\sqrt{1-\xi^2}) \omega_0 \end{cases}$$

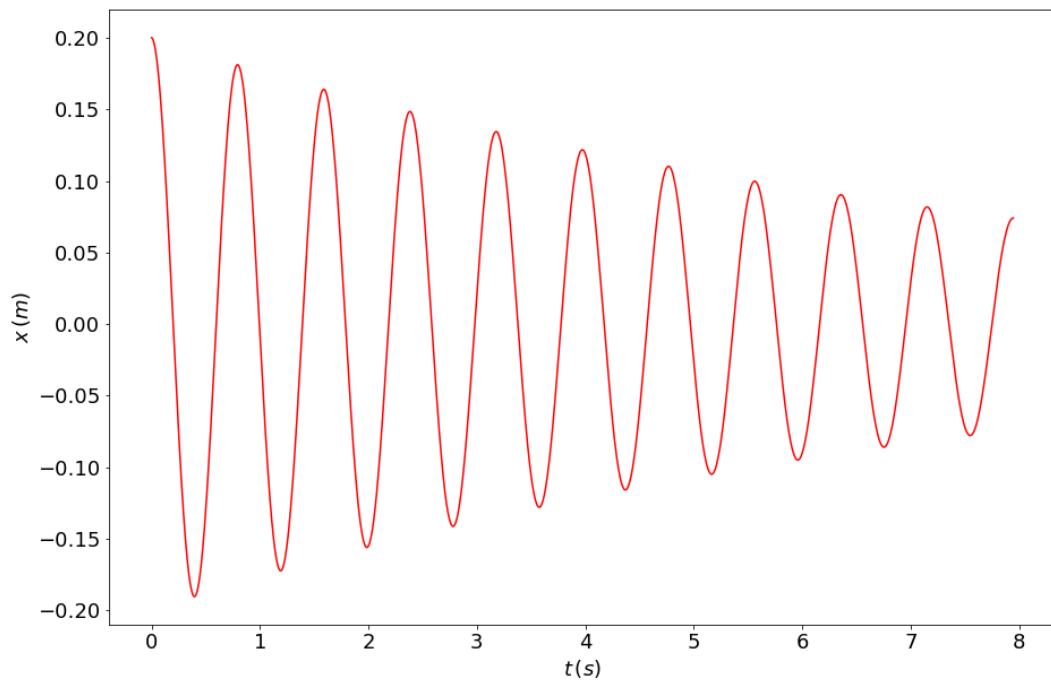
Avec $\xi = \frac{b}{2m\omega_0}$

Par conséquent, la solution de l'équation différentielle est donnée par :

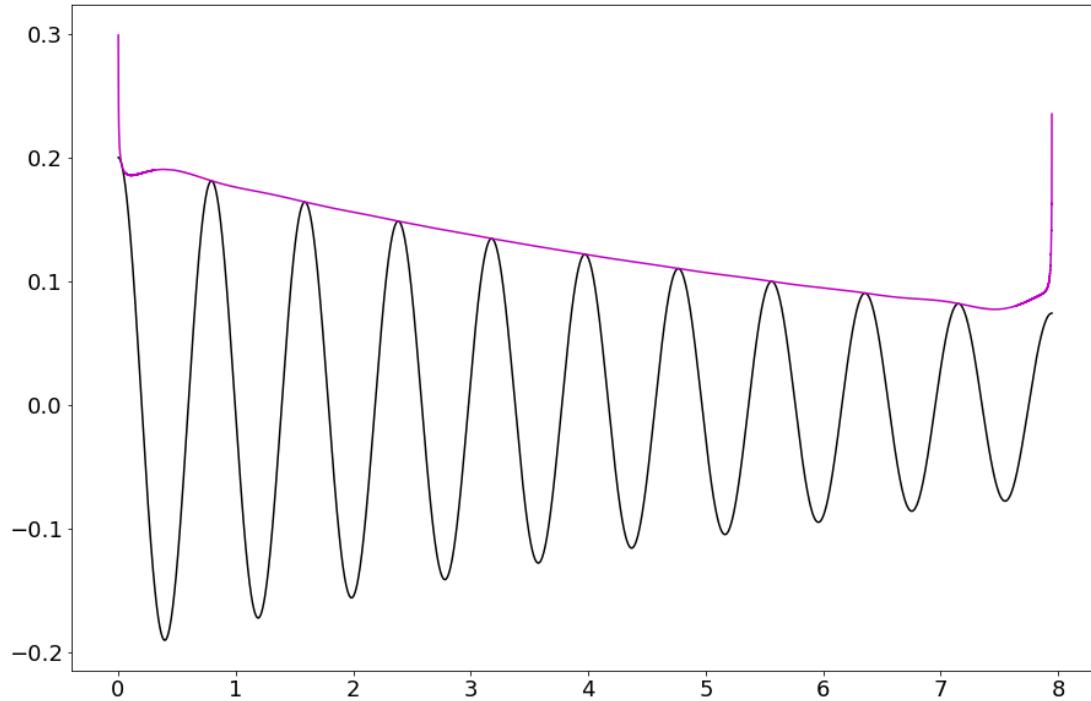
$$\begin{cases} x(t) = e^{-\xi\omega_0 t} \left(x_0 \cos(\omega_0 \sqrt{1-\xi^2} t) + \frac{\dot{x}_0}{\omega_0 \sqrt{1-\xi^2}} \sin(\omega_0 \sqrt{1-\xi^2} t) \right) \\ r_{1,2} = (-\xi \pm i\sqrt{1-\xi^2}) \omega_0 \end{cases}$$

Il s'agit alors d'un système dissipatif soumis à un régime d'oscillations libres.
La simulation temporelle est alors donnée par :

```
xi est= 0.015811388300841896
Out[21]: Text(0, 0.5, '$x \\\, (m)$')
```



Avec la fonction de Hilbert, on peut visualiser le décrément logarithmique sur python :



II. Excitation forcée

II.1. Paramétrage

Le paramétrage en oscillations forcées est le même qu'en libre mise à part une excitation sinusoïdale effectuée en amont du ressort. Le système vibre alors linéairement à 1 degré de liberté.

II.2. Mise en équation

Au lieu d'avoir un 0 dans le second membre de l'équation différentielle après une analyse dynamique, on l'équation donnée par :

$$m\ddot{x} + b\dot{x} + kx = F_0 \cos(\omega t)$$

La fonction de transfert obtenue est ainsi donnée par :

Fonction de transfert :

$$kH(\omega) = \frac{X_0(\omega)}{F_0/k} = \frac{1}{1 - \frac{\omega^2}{\omega_0^2} + i2\xi\frac{\omega}{\omega_0}} e^{-i\phi}$$

Facteur d'amplification dynamique :

$$|kH(\omega)| = \frac{1}{\sqrt{\left(1 - \frac{\omega^2}{\omega_0^2}\right)^2 + 4\xi^2 \frac{\omega^2}{\omega_0^2}}}$$

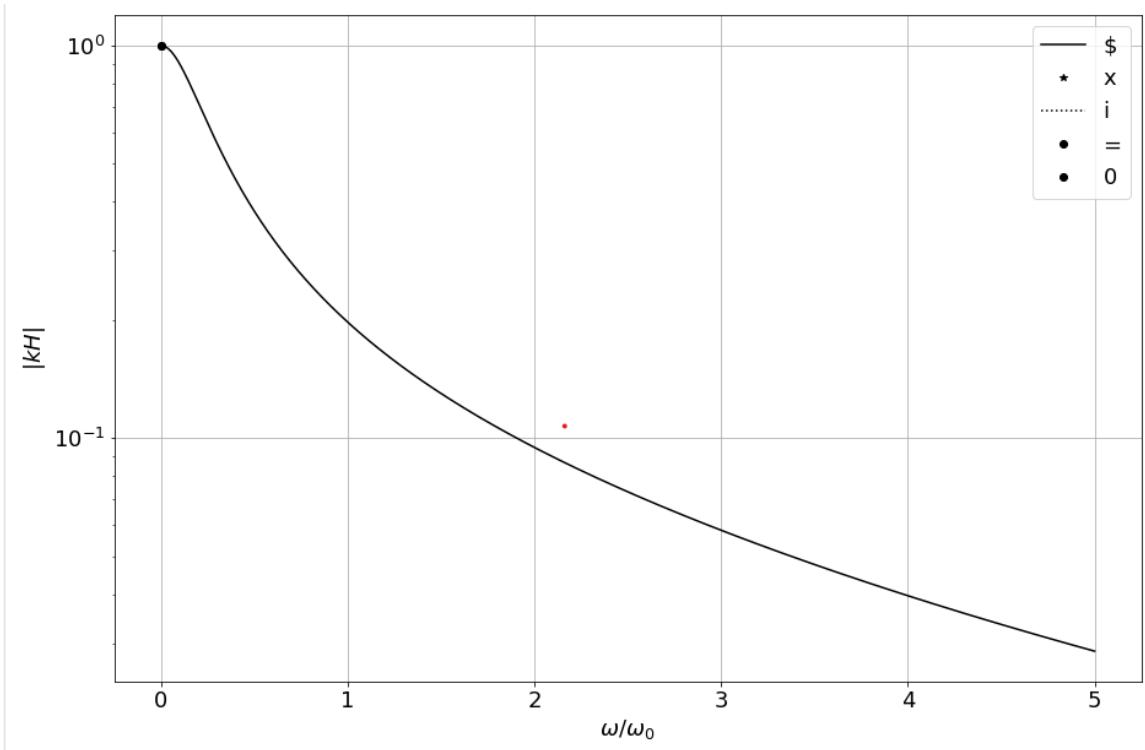
Pulsion de résonnance en déplacement :

$$\omega_d = \omega_0 \sqrt{1 - 2\xi^2}$$

Amplification dynamique maximale :

$$|kH(\omega_d)| = \frac{1}{2\xi\sqrt{1 - \xi^2}}$$

II.3. Réponse fréquentielle



III. Dimensionnement du moteur

III.1. Objectif fixé :

Cette partie a pour but de valider l'utilisation du moteur 12V pour savoir si ce dernier est apte à effectuer et soutenir la charge qu'on lui demande lors de l'oscillation forcée du système.

III.2. Etat initial de la tâche :

Le document technique du moteur nous a été fourni en amont.

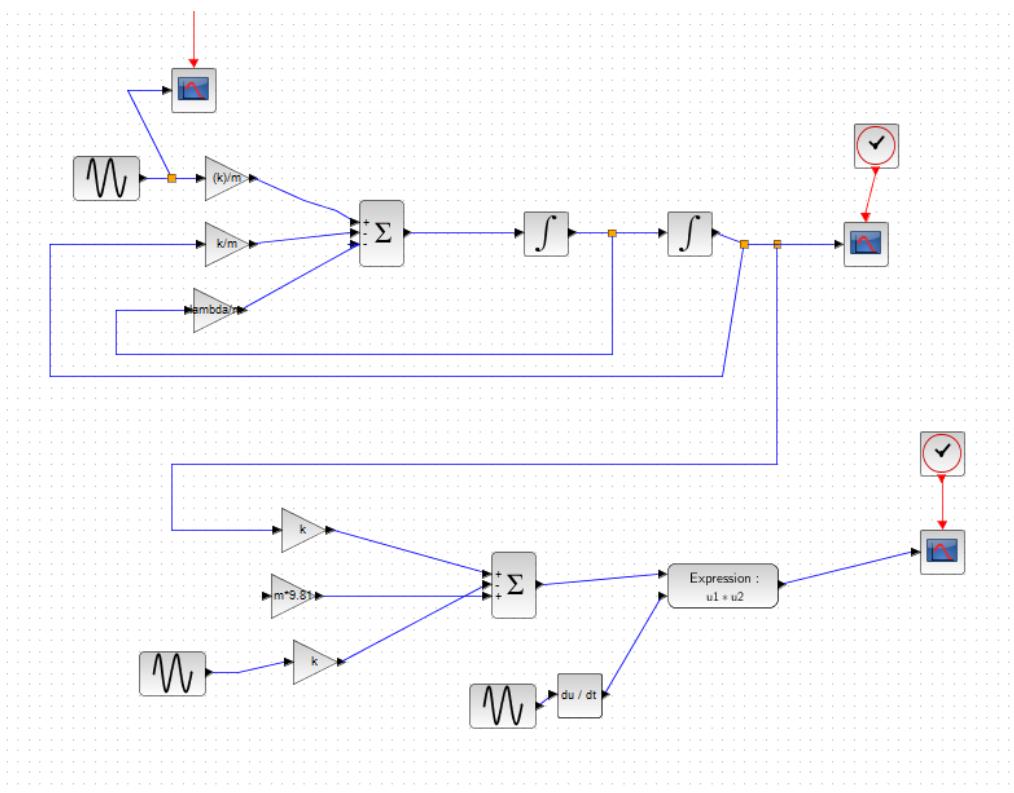
III.3. Choix techniques :

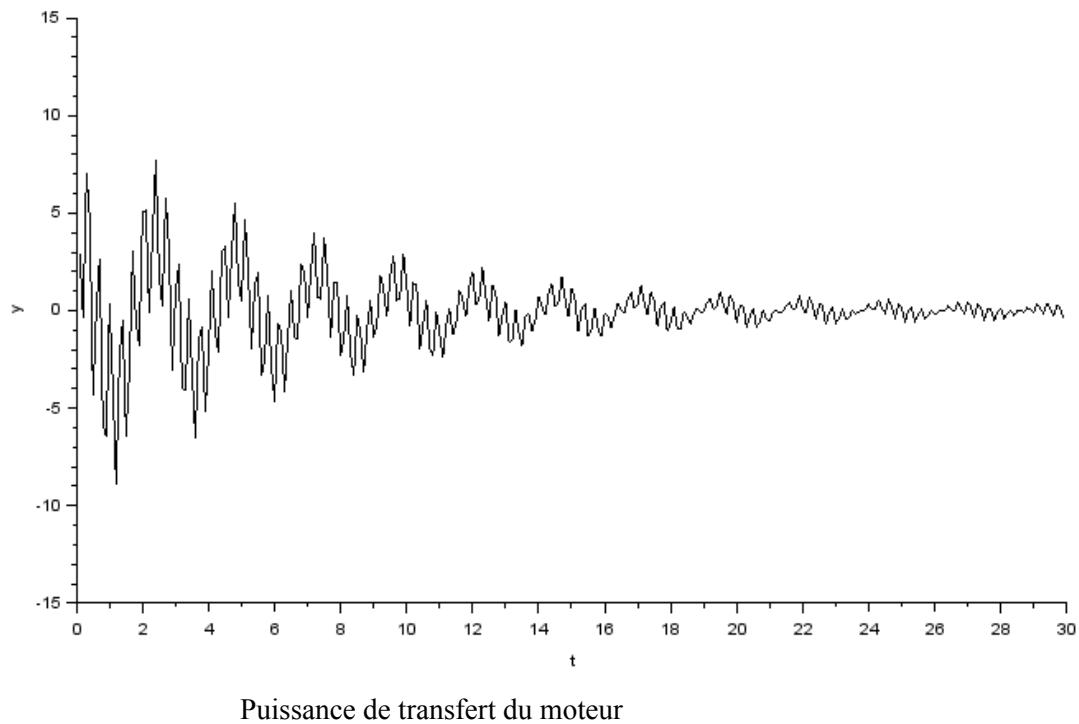
On a utilisé l'extension xcos simulink de scilab afin de simuler la puissance maximale que devrait supporter le moteur. Pour se faire on a utilisé l'équation de transfert de puissance qui est la suivante :

$$\mathfrak{P} = (k(x - e) + mg) \dot{e}$$

Avec "e" la position du moteur et donc "e point" sa vitesse.

Sur simulation, on obtient le résultat suivant :





Ainsi, on peut constater que le maximum de puissance de 12W que peut endurer le moteur est n'est pas dépassé pour notre système donc le moteur qu'on utilise est adéquat.

IV. Contrôle commande et acquisition

I.1. Objectifs fixés :

Dans cette section, notre attention s'est portée sur plusieurs points :

- La mise en place d'un PID fonctionnel sur une plateforme Arduino pour le contrôle du moteur.
- L'utilisation d'une seule carte Arduino pour la commande du moteur et la récupération des données provenant du capteur.
- La mise en place de la communication avec l'interface utilisateur (IHM).

En ce qui concerne l'acquisition des données, notre objectif principal était de comprendre comment exploiter les informations envoyées par l'encodeur du moteur, ainsi que de recevoir et traiter les données provenant du capteur Lidar pour obtenir la position du moteur.

I.2. Etat initial de la tâche

Lors du démarrage du projet, certains scripts Arduino étaient déjà en place. Voici ce que nous avions initialement :

Un programme permettant de contrôler un moteur 12V en boucle fermée à l'aide d'un correcteur PI. Ce programme nécessite un câblage spécifique car il utilise la valeur d'un potentiomètre pour générer la commande. Un réglage PI était appliqué pour ajuster la consigne envoyée au moteur en fonction de l'erreur de vitesse de rotation, obtenue à partir de l'encodeur rotatif. Cependant, ce programme ne pouvait pas être lancé à partir d'une interface utilisateur (IHM) et il utilisait un correcteur PI plutôt qu'un correcteur PID.

Un programme permettant de lire, de convertir et d'envoyer en continu les données provenant d'un capteur Lidar sur le port série. Ce programme permettait à l'IHM de connaître la position du système afin de l'afficher graphiquement. Cependant, la communication entre l'IHM et le programme était unidirectionnelle, ce qui signifie que l'IHM ne pouvait pas communiquer avec le programme.

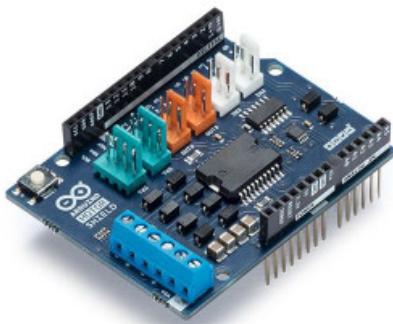
Bien que ces points de départ aient été bénéfiques, ils présentaient quelques problèmes qui devaient être résolus. Le programme de contrôle du moteur nécessite un câblage spécifique avec un potentiomètre, et il utilise un correcteur PI au lieu d'un correcteur PID. De plus, la communication entre l'IHM et le programme du capteur était unidirectionnelle, empêchant l'IHM d'interagir avec le programme.

I.3. Planification

5 Contrôle cmd et acquisition (Arduino)	
Commande du moteur	Yassine/Mans/Simon 80 %
Amélioration Acquisition	Yassine/Mans/Simon 100 %
Fusion en une seule carte	Yassine/Mans/Simon 100 %

I.4. Choix techniques

Contrôleur moteur:



Au cours de ce semestre, nous avons remplacé le moteur du système. Nous sommes passés d'un moteur 24VDC avec encodeur à un moteur 12VDC avec encodeur. Bien que le nouveau moteur puisse être contrôlé avec la carte de puissance de l'ancien moteur, nous avons décidé d'utiliser une carte Arduino Motor Shield 2 x 2 A A000079. Cette carte se branche directement sur l'Arduino et est conçue pour contrôler des moteurs de 5 à 12 VDC.

Mode d'acquisition :

Pour l'acquisition des données, nous avons choisi de tout gérer au niveau de la carte Arduino en fonction des informations transmises par l'IHM. Cela signifie que le démarrage de l'acquisition, la gestion du temps et l'arrêt de l'acquisition sont tous effectués sur la carte Arduino. De plus, nous avons introduit une attente passive dans la boucle loop() afin de définir la période de rafraîchissement des mesures.

Filtrage des valeurs du capteur Lidar :

Les données fournies par le capteur Lidar, une fois converties en distances, peuvent être assez bruitées. Par exemple, nous avons observé des variations de distance de l'ordre de 8 à 10 mm lors des mesures au repos. Pour améliorer la qualité des mesures, nous avons choisi d'implémenter un filtrage approprié pour réduire le bruit et obtenir des données plus exploitables.

I.5. Travail réalisé

Filtrage du capteur de position:

Dans un premier temps, nous avons pris en charge le code existant et supprimé les instructions 'delay()' afin d'éviter les arrêts complets de la carte Arduino. Notre objectif était de réaliser simultanément l'acquisition des données du capteur et le contrôle du moteur. Ensuite, nous avons mis en place un filtrage en utilisant un filtre de Kalman simplifié pour réduire les erreurs de distance causées par le bruit lors des mesures. Ce filtrage permet d'obtenir des données plus précises et moins sujettes aux variations indésirables.

```
float Q = 0.1; // Bruit du modèle
float R = 3.0; // Bruit de mesure |

float x = 275; // État estimé
float P = 0; // Erreur de covariance estimée
```

```

int16_t t = pulseIn(sensorPin, HIGH);
if (t != 0 && t <= 1850)
{
    int16_t d = (t - 1000) * 2; // Convertir la largeur d'impulsion en microsecondes en distance en millimètres.
    if (d < 0) { d = 0; } // Limiter la distance minimale à 0.
    //Filtrage de la valeur
    float x_pred = x;// Prédiction de l'état suivant
    float P_pred = P + Q;
    float K = P_pred / (P_pred + R); // Calcul du gain de Kalman
    x = x_pred + K * (d - x_pred); // Mise à jour de l'état estimé et de l'erreur de covariance estimée
    P = (1 - K) * P_pred;
    pos = x; //Position filtrée
}
    
```

Dans la déclaration des variables, nous avons initialisé la variable x à 275 car c'est la distance que nous mesurons lorsque le système est au repos. Dans la boucle 'setup()', nous effectuons 40 mesures de position pour initialiser le filtre et obtenir une valeur correcte dès la première acquisition de données.

De plus, nous avons regroupé les conditions 'if' en une seule boucle pour simplifier le code. Nous avons spécifié que seules les trames de données définies seront traitées, afin d'éviter toute confusion ou perte de communication avec l'IHM.

Control du moteur

La première étape a consisté à trouver et commander une carte d'extension compatible avec l'Arduino UNO, capable de contrôler un moteur 12VDC. Une fois la carte reçue, nous avons effectué des tests pour contrôler le moteur en boucle ouverte afin de nous familiariser avec le matériel.

Ensuite, nous nous sommes concentrés sur la partie boucle fermée et l'asservissement. Nous avons utilisé le programme de l'année précédente qui permettait de réaliser un asservissement en vitesse avec un correcteur PI. Nous avons adapté ce programme à nos besoins, en prenant en compte les broches PWM et le moteur spécifiés par la carte d'extension. Nous avons réussi à faire fonctionner le programme correctement, ce qui nous a donné une base fonctionnelle.

Ensuite, nous avons modifié le programme afin de pouvoir piloter le moteur avec une consigne envoyée via le port série de la carte Arduino. Parallèlement, nous avons également modifié la partie de temporisation passive en remplaçant une instruction 'if()' par une boucle 'while()', car nous utilisons une boucle 'while()' dans le programme du capteur.

Une fois que tout fonctionnait correctement, nous avons tenté d'implémenter un régulateur PID sur la base du régulateur PI existant. Cependant, nous n'avons pas pu terminer cette partie du projet en raison de problèmes rencontrés avec la carte d'extension. Lorsque nous branchions l'alimentation externe, la connexion série entre l'ordinateur et la carte Arduino se coupait, ce qui nous empêchait de communiquer et d'envoyer les consignes de vitesse. De plus, la carte Arduino chauffait beaucoup. Par conséquent, nous avons décidé de revenir à l'ancienne carte de puissance avec laquelle nous n'avions pas rencontré ces problèmes, afin de pouvoir continuer à tester nos programmes.

Pour le programme final, nous avons décidé d'utiliser le correcteur PI, car nous avons réussi à le tester et à le valider, contrairement au correcteur PID pour lequel nous avons rencontré des problèmes et n'avons pas pu le mettre en œuvre dans le temps imparti. Le correcteur PI s'est avéré être une solution satisfaisante pour atteindre nos objectifs de contrôle de la vitesse du moteur.

Fusion des programmes sous une seule carte

Pour fusionner les programmes, nous avons suivi une approche par étapes. Tout d'abord, nous avons réalisé l'implémentation en mode "local", c'est-à-dire sans utiliser l'IHM. Nous avons créé un programme qui gère passivement la période d'acquisition à l'aide d'une boucle "while()".

```

// Mise en place du temps
unsigned long currentMillis = millis();
unsigned long refreshMillis = currentMillis;

void loop() {

    // Mise en place d'un rafraichissement
    while((currentMillis - refreshMillis) < periodeLoop)
    {
        currentMillis = millis();
    }
    refreshMillis = currentMillis;
}
    
```

Ensuite, nous avons créé deux fonctions génériques qui génèrent des valeurs en les incrémentant automatiquement.

Ensuite, nous avons remplacé l'une de ces fonctions par la fonction "capteur()" qui lit, traite et envoie les données de position du système. De même, nous avons ajouté une fonction "moteur" qui contrôle le moteur en boucle ouverte pour faciliter la programmation et les tests.

Une fois que ces étapes ont été réalisées avec succès, nous avons intégré l'IHM pour vérifier si nous pouvions toujours récupérer les données du capteur de position tout en maintenant la communication avec l'IHM.

Cette approche progressive nous a permis de vérifier chaque étape et de résoudre les problèmes éventuels au fur et à mesure de l'intégration des différentes fonctionnalités.

Finalement, nous avons réussi à fusionner l'acquisition et la commande dans un même programme sur la même carte Arduino, en utilisant une période définie. Cette fusion s'est déroulée sans trop de difficultés.

Programme général

Pour cette partie, nous avons commencé à partir de zéro par rapport aux années précédentes, car aucun travail préalable n'avait été réalisé. Notre objectif était de mettre en place une véritable communication bidirectionnelle entre l'IHM et l'Arduino, afin que l'IHM puisse donner l'ordre à l'Arduino de commencer l'acquisition, que l'Arduino effectue l'acquisition uniquement pendant la plage de temps définie dans l'IHM, et renvoie les données de temps, de rotation et de position, ainsi que l'état de l'acquisition (terminée ou non). Nous avons procédé étape par étape pour atteindre cet objectif.

Dans un premier temps, nous avons travaillé sur l'envoi d'une trame prédéfinie de l'IHM vers l'Arduino, qui décide cette trame et stocke les valeurs correspondantes. Ensuite, nous avons mis en place l'envoi d'une trame de l'Arduino vers l'IHM, où l'IHM identifie les valeurs et les stocke pour les afficher.

Par la suite, nous avons ajouté la fonctionnalité de l'acquisition à partir du capteur, en permettant à l'Arduino de lire, traiter et envoyer les données de position. Enfin, nous avons intégré la fonctionnalité de contrôle du moteur.

À la fin de notre travail, nous disposons d'un programme presque terminé. Le mode d'oscillation libre fonctionne, ainsi que le mode d'oscillation forcée manuelle (envoi d'une seule consigne de vitesse). Cependant, nous n'avons pas eu le temps de mettre en place le mode d'oscillation

forcée automatique (envoi d'un nombre prédéfini de paliers de consigne à effectuer pendant la durée de la simulation).

Malgré cela, nous sommes parvenus à établir une communication fonctionnelle entre l'IHM et l'Arduino, permettant une acquisition contrôlée des données et une commande du moteur selon les consignes fournies par l'IHM.

Voici la trame reçue par l'arduino: **50,1,0,0,0.05,0.03,0.01,150,30,330,2,10/**

Dans notre système, la trame de communication contient plusieurs paramètres qui sont envoyés lors du démarrage ou de l'arrêt de l'acquisition depuis l'IHM. Tous les paramètres ne sont pas utilisés simultanément, c'est le rôle de l'Arduino de sélectionner les paramètres appropriés en fonction du mode d'oscillation choisi. Cette approche nous permet d'avoir une trame fixe pour simplifier la programmation et réduire la complexité de la communication.

```
if (Serial.available()) //Lecture du port série
{
    String message = Serial.readStringUntil('/'); // Lit la chaîne de caractères jusqu'au caractère '/'

    int numValues = toFloatArray(message, values, 12);

    //Mise à jour des paramètres en fonction des valeurs reçues
    freq_acq = int(values[0]); //fréquence aquisition en Hz
    state_acq = int(values[1]); //état aquisition, 0 inconnu, 1 départ, 2 stop
    state_mode_oscil = int(values[2]); //0 Mode libre, 1 mode manuel
    state_ctrl_mot = int(values[3]); //0 mode manuel, 2 mode auto
    kp = values[4];
    ki = values[5];
    kd = values[6];
    speed_order = int(values[7]); //consigne de vitesse mode manuel
    min_speed_order = int(values[8]); //consigne de vitesse min mode auto
    max_speed_order = int(values[9]); //consigne de vitesse max mode auto
    nb_pas = int(values[10]); //nb de palier
    tps_simu = int(values[11]); //temps de simulation en s

    //Mise à jour de la fréquence de rafraîchissement
    freqLoop = (unsigned long)freq_acq;
    periodeLoop = 1000/freqLoop;
}
```

En d'autres termes, l'IHM envoie une trame contenant tous les paramètres possibles, mais l'Arduino sélectionne les paramètres pertinents en fonction du mode d'oscillation sélectionné. Cela permet de garder une trame fixe pour la communication, ce qui facilite la programmation et assure une meilleure gestion des différents modes d'oscillation.

Ainsi, l'Arduino est responsable de décrypter la trame reçue, de choisir les paramètres appropriés en fonction du mode d'oscillation et d'effectuer les actions correspondantes, que ce soit le lancement de l'acquisition, l'arrêt de l'acquisition ou le réglage des paramètres du contrôleur moteur.

Voici la trame émise par l'arduino: **TxxRxxPxxCx**

Dans notre système, la carte Arduino envoie les valeurs de position et de vitesse sur le port série, en utilisant des balises pour faciliter le décryptage des informations dans l'IHM. Ces valeurs sont obtenues à partir des fonctions de contrôle et d'acquisition, en fonction du temps écoulé depuis le début de l'acquisition.

La communication entre l'Arduino et l'IHM est structurée à l'aide de balises qui délimitent les différentes informations. Cela permet à l'IHM de décrypter facilement les données reçues et d'associer les valeurs aux paramètres correspondants.

```
void envoiDonnees(void){  
    Serial.print("T");  
    Serial.print(tps);  
    Serial.print("R");  
    Serial.print(rpm);  
    Serial.print("P");  
    Serial.print(pos);  
    Serial.print("C");  
    Serial.println(completSimu);  
}
```

En plus des valeurs de position et de vitesse, la carte Arduino envoie également l'état de l'acquisition, indiquant si elle est en cours (0) ou terminée (1). Ces informations sont transmises dans la trame via les balises appropriées, ce qui permet à l'IHM de suivre l'état de l'acquisition en temps réel.

I.6. Perspective d'évolution

Nous avons réussi à faire une bonne partie des objectifs visés. Nous avons un programme de commande/acquisition sur la même carte, une communication bi directionnelle avec l'IHM. Pour faire évoluer le projet, il est possible de prendre en compte les perspectives suivantes :

Implémentation d'un PID : Cela permettrait d'ajuster plus finement la consigne de vitesse et d'obtenir un contrôle plus précis du moteur.

Simplification du programme : Il faudrait revoir et simplifier le programme afin de le rendre plus clair et plus efficace. Éviter les duplications de code et rechercher des moyens d'améliorer la lisibilité. Cela permettra aussi d'augmenter la fréquence d'acquisition et d'identifier et résoudre le problème périodique rencontré dans la prise des mesures en mode oscillation forcée peut être nécessaire.

Amélioration de la gestion des erreurs : Il est recommandé d'ajouter des mécanismes de gestion des erreurs et de récupération en cas de problèmes de communication ou de dysfonctionnements. Cela peut inclure la mise en place de messages d'erreur explicites.

V. IHM

I.1. Objectifs fixés

Les objectifs initiaux de la tâche étaient les suivants :

Fiabilisation de l'IHM pour les oscillations libres : L'objectif était d'améliorer la fiabilité de l'IHM en ajustant le délai d'acquisition des données.

Intégration des paramètres nécessaires pour les oscillations forcées : ajout des paramètres essentiels tels que le PID et la consigne de vitesse pour les oscillations forcées.

Les objectifs se sont vite ajustés au fur et à mesure de la compréhension du programme et en fonction des problèmes rencontrés. Les nouveaux objectifs sont les suivants :

Simplification de l'IHM : L'objectif était de simplifier l'IHM en supprimant les éléments superflus, afin de rendre l'interface plus claire et plus intuitive pour les utilisateurs. Cela inclut l'élimination de l'affichage de la courbe théorique en superposition de la courbe expérimentale.

Communication bidirectionnelle avec l'Arduino : L'objectif final était d'établir une communication bidirectionnelle entre l'IHM et l'Arduino, permettant ainsi à l'IHM de transmettre les commandes et les paramètres de configuration à l'Arduino, et à l'Arduino de renvoyer les données et l'état de l'acquisition à l'IHM.

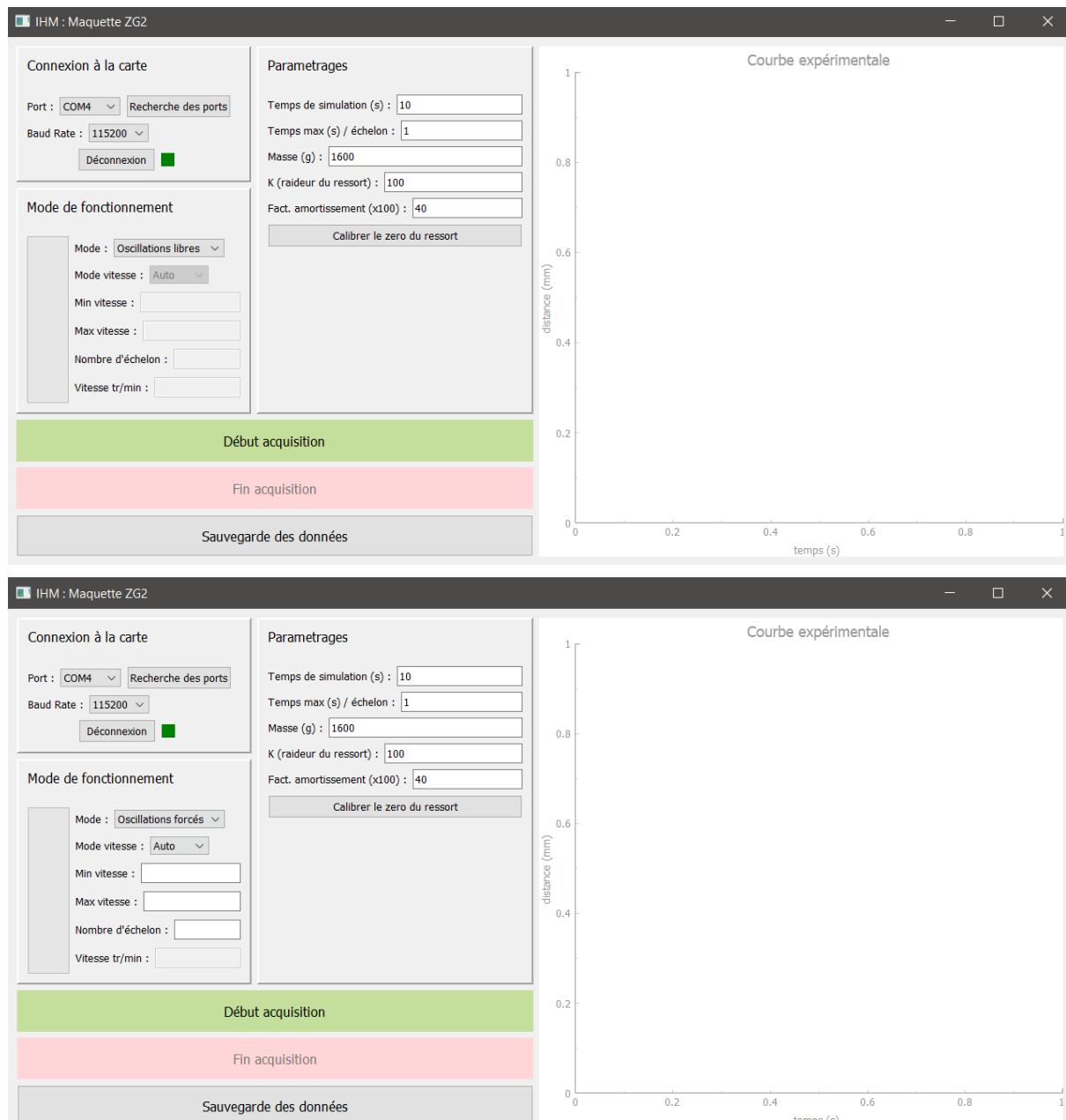
I.2. Etat initial de la tâche

L'IHM est présentée sous la forme d'un fichier exécutable. Lors de son lancement, elle permet de rechercher les ports série disponibles et propose à l'utilisateur de se connecter à la carte Arduino en choisissant parmi ces ports. Une fois la connexion établie, il est possible de se déconnecter de la carte Arduino, par exemple pour envoyer un programme vers celle-ci. L'IHM permet également de choisir le mode d'oscillation (libre ou forcé), ce qui donne accès à différents menus où l'utilisateur peut modifier les valeurs des paramètres pertinents. Une fois la configuration terminée, il est possible de lancer une acquisition.

À partir de ce moment, la position du système est affichée en temps réel en fonction du temps et l'utilisateur peut arrêter l'acquisition à l'aide d'un bouton "Stop" ou attendre la fin de celle-ci. Enfin, il est possible de récupérer le fichier .csv correspondant à l'essai effectué ou de modifier les paramètres et relancer un nouvel essai.

Au début du projet, l'IHM semblait communiquer avec l'Arduino car elle affichait les valeurs de la position du système en temps réel lorsqu'une acquisition était lancée. Cependant, en réalité, l'IHM ne communique pas directement avec l'Arduino. Elle se contente de lire les données envoyées en continu sur le port série par l'Arduino. La gestion du temps était effectuée exclusivement dans l'IHM, ce qui alourdissait et ralentissait son fonctionnement. Néanmoins, l'IHM était capable d'identifier et de décrypter les données de position reçues grâce à des balises. En revanche, elle ne permettait pas de contrôler le moteur. À la fin de l'acquisition, l'IHM fournissait un fichier .csv contenant les données enregistrées pour le post-traitement. Il est à noter que de nombreux paramètres étaient disponibles dans l'IHM, mais tous ne semblaient pas être utilisés.

Visuels de l'ancienne ihm



I.3. Planification

6 IHM											
Intégration paramètres	Simon	100 %									
Fiableisation IHM	Simon	100 %									

I.4. Choix techniques

Choix des paramètres

Pour optimiser les paramètres et ne conserver que ceux nécessaires à la simulation, nous avons effectué une sélection. Certains paramètres ont été supprimés tandis que de nouveaux ont été ajoutés, tels que la période d'échantillonnage. Cette approche a permis de simplifier l'affichage et de ne conserver que les paramètres essentiels à l'acquisition. Par exemple, les paramètres de masse et de raideur du ressort ont été retirés car ils sont utilisés uniquement lors du post-traitement pour générer la courbe théorique, une étape qui se déroule en dehors de l'IHM.

Gestion du temps

Nous avons pris la décision de supprimer la gestion du temps de simulation dans l'IHM, car cela ne relève pas de sa fonction principale. En effet, cela surchargeait et ralentissait le fonctionnement de l'IHM. La gestion du temps est désormais effectuée par l'Arduino, qui est programmé à cet effet. Cela permet de déléguer cette tâche à la carte Arduino, c'est son rôle de gérer la simulation.

Choix de la trame d'envoi

Pour assurer une communication efficace avec la carte Arduino, nous avons établi une liste de paramètres à transmettre. Cette liste devait obligatoirement inclure tous les paramètres modifiables, car ils sont essentiels à la simulation. En plus de ces paramètres, nous avons ajouté certains éléments qui varient en fonction de l'état des boutons ou des menus, tels que le mode d'oscillation ou le déclenchement de l'acquisition. Ce choix a été effectué en tenant compte des besoins spécifiques de la carte Arduino pour assurer une communication fluide et pertinente entre l'IHM et la carte.

Choix de la trame de sortie

Pour la deuxième partie de la communication, nous avons utilisé le système de balises pour organiser les informations reçues sur le port série. En collaboration avec l'Arduino, nous avons établi une structure de trame spécifique pour faciliter l'échange de données entre l'IHM et la carte. Cette trame permet de décrypter et d'identifier rapidement les informations transmises, en utilisant des balises spécifiques pour chaque type de donnée. Cela assure une communication fluide et cohérente entre les deux systèmes, en garantissant que les informations sont correctement interprétées et utilisées par l'IHM.

I.5. Travail réalisé

Choix des paramètres et des trames:

Voici la liste des paramètres utilisés :

```
self.freq_acq = 0 # fréquence d'aquisition
self.state_acq = 0 # 0 pas défini, 1 click lancement acq, 2 click arret acq
self.state_mode_oscil = 0 # 0 oscillation libre, 1 oscillation forcée
self.state_ctrl_mot = 0 # 0 Auto, 1 manuel
self.kp = 0 # Kp PID
self.ki = 0 # Ki PID
self.kd = 0 # Kd PID
self.speed_order = 0 # Consigne de vitesse du moteur mode manuel
self.min_speed_order = 0 # Consigne min de vitesse du moteur mode auto
self.max_speed_order = 0 # Consigne max de vitesse du moteur mode auto
self.nb_pas = 0 # Nombre de pallier de vitesse mode auto
self.tps_simu = 0 # Temps de simulation
```

Voici la trame d'envoie:

```
def writeData(self):
    # Freq acq | Etat acq | Mode oscillation | Mode ctr moteur | coeffM(Kp/Ki/Kd) | ParamMoteur (consigne/min/max/nb_pas) | Temps simu
    msg ="{}{},{}{},{}{},{}{},{}{},{}{},{}{}".format(int(self.freq_acq),int(self.state_acq),int(self.state_mode_oscil),
                                                    int(self.state_ctrl_mot),float(self.kp),float(self.ki),float(self.kd),
                                                    int(self.speed_order),int(self.min_speed_order),int(self.max_speed_order),
                                                    int(self.nb_pas),int(self.tps_simu))
    print(msg)
    self.serial.write(msg.encode())
```

Voici la trame reçue:

```
def readData(self): #Quand l'arduino écrit une trame sur le port série (b'T20R100P200\r\n'), on stock le temps, la rpm et la pos
    print("read data function")
    #read_data = self.serial.readLine()
    #print(read_data)
    #print(" --- ")
    if self.serial.canReadLine():
        read_data = self.serial.readLine()
        print(read_data)
        read_data_temps = str(read_data)[str(read_data).find("T")+1:str(read_data).find("R")]
        read_data_rpm = str(read_data)[str(read_data).find("R")+1:str(read_data).find("P")]
        read_data_pos = str(read_data)[str(read_data).find("P")+1:str(read_data).find("C")]
        read_data_simuCplt = str(read_data)[str(read_data).find("C")+1:-5]

        print('temps = ',read_data_temps)
        print('rpm = ',read_data_rpm)
        print('pos = ',read_data_pos)
        print('simuCplt = ',read_data_simuCplt)
        print(" ")

        if (read_data_simuCplt == "1" or int(read_data_temps) >= int(self.tps_simu)*1000 ):
            self.bouton_acquisition.setDisabled(False)
            self.bouton_stop.setDisabled(True)
            self.boutonCsv.setDisabled(False)
        if (read_data_simuCplt == "0"):
            try:
                temps = float(read_data_temps)
                pos = int(read_data_pos)
                rpm = int(read_data_rpm)
                self.setDataCourbeWindow(pos,rpm,temps)
            except:
                pass
```

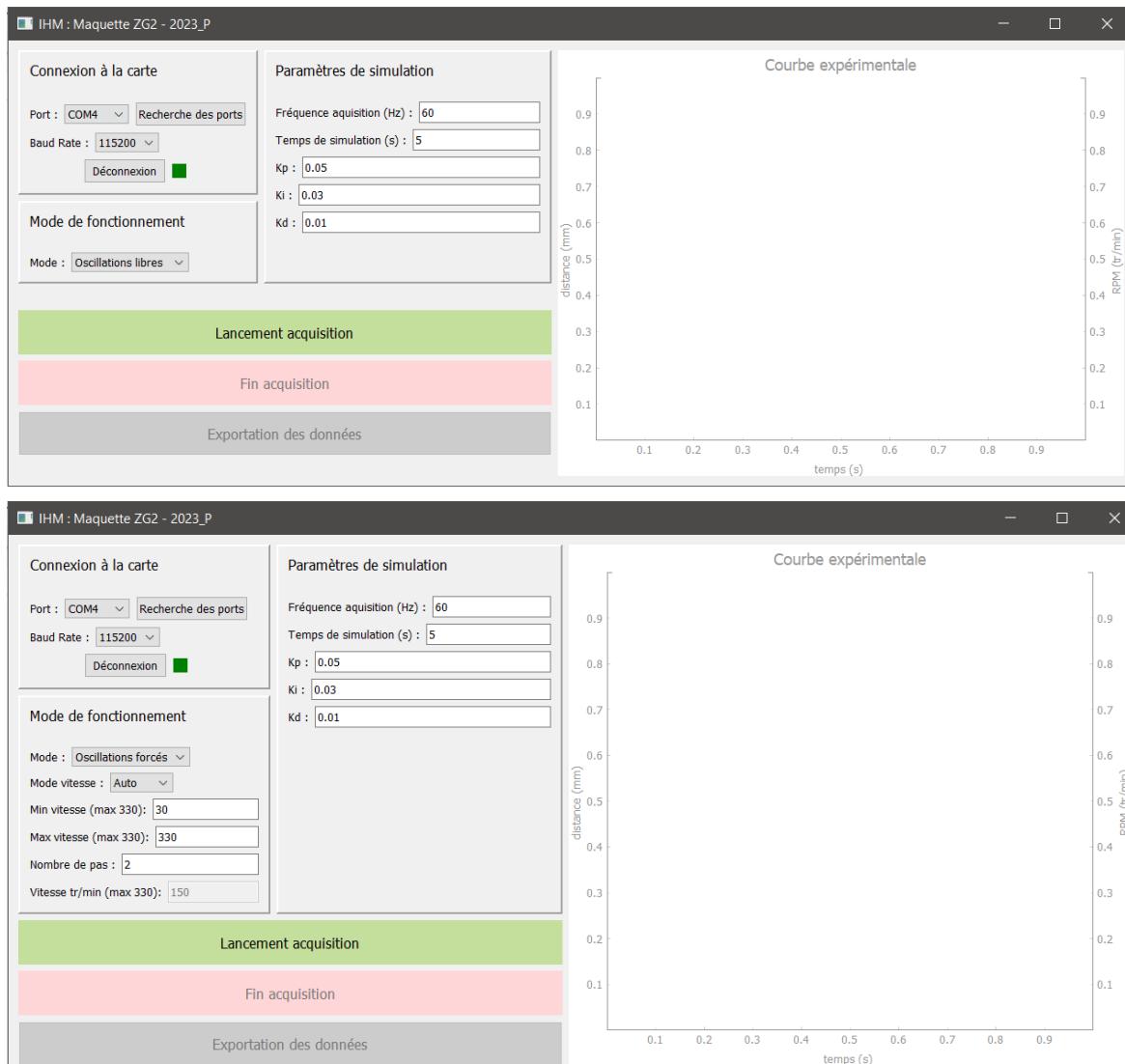
Gestion du temps

Dans le système mis en place, l'IHM transmet à l'Arduino uniquement la durée de simulation et le pas de temps souhaité. Ensuite, l'Arduino effectue l'acquisition des données et envoie sa trame de données sur le port série. À l'intérieur de cette trame, la dernière position est réservée à une valeur binaire (0 ou 1). Cette valeur spécifique permet à l'IHM de savoir quand la transmission des données est terminée en évaluant simplement cette valeur. Ainsi, l'IHM peut détecter la fin de la transmission sans avoir à analyser l'intégralité de la trame reçue, ce qui simplifie le processus de communication et accélère le traitement des données.

Amélioration du code

En enlevant les paramètres inutiles dans l'IHM, on en a profité pour enlever les lignes de code pour alléger sa compréhension. Dans la trame de données reçue depuis l'Arduino, nous avons ajouté les valeurs de vitesse de rotation pour pouvoir les afficher simultanément avec les valeurs de position. Cela permet de contrôler le moteur et d'observer à la fois la position et la vitesse de rotation du système en temps réel. Cela vous permet d'analyser plus facilement les relations entre la position, la vitesse de rotation.

Voici à quoi ressemble l'ihm maintenant:



I.6. Perspective d'évolution

Le nouvel IHM que nous avons développé communique efficacement avec l'Arduino, ce qui nous permet de réaliser des essais en oscillation libre ou forcée en mode manuel. Nous avons la possibilité de configurer la durée de simulation, la période d'acquisition, les gains du PID et les paramètres de vitesse du moteur. Et nous avons en temps réel la position du système et la vitesse de rotation.

Cependant, il existe des opportunités d'amélioration. Tout d'abord, il serait possible d'optimiser davantage le programme en supprimant les paramètres et les fonctions qui ne sont pas utilisés. De plus, en travaillant en collaboration avec la carte Arduino, nous pourrions envisager l'ajout de mécanismes de remontée d'informations pour détecter et gérer les erreurs éventuelles.

VI. Post traitement

I.1. Objectifs fixés

L'objectif, au départ, était de déterminer la pulsation propre et du taux d'amortissement à partir du signal temporel en oscillation libre.

I.2. Etat initial de la tâche

Au départ, un code d'acquisition pour la carte arduino nous a été fournis ainsi qu'un code en Python pour traiter le signal acquis. Ce code que nous avons trouvé un peu tard ce qui nous a forcés à effectuer les calculs à la main.

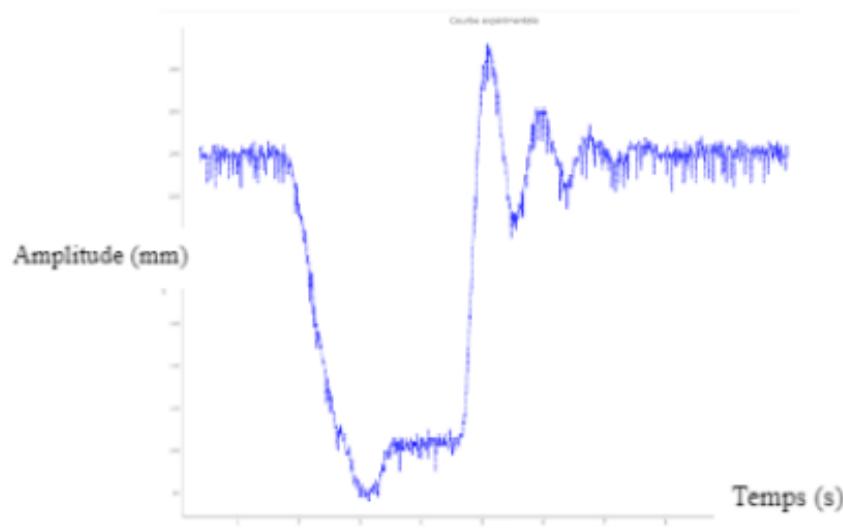
I.3. Planification

Nous avons décidé de commencer par le post-traitement en oscillation libre qui pouvait être effectué immédiatement car nous n'avions pas besoin d'asservir le moteur qui n'était pas encore monté sur la maquette. Puis nous continuerons avec le post traitement en oscillations forcées une fois le moteur monté sur la maquette et asservis.

I.4. Choix techniques

1) Oscillations libres

Pour la partie oscillation libre, le choix de l'amortissement à était un choix très important, en effet durant cette partie du mouvement si le coefficient d'amortissement était trop élevé. Avec un coefficient d'amortissement trop élevé le décrément logarithmique est très élevé et rend la mesure du mouvement tout d'abord plus compliquée et ensuite moins intéressante.



Comme on peut le voir sur la figure ci-dessus, l'amortissement se fait très rapidement avec une décroissance des amplitudes au fil du temps très forte.

C'est pour cela que nous avons décidé de mettre en place un amortissement à air uniquement. La nouvelle mesure en oscillation libre nous donne des résultats plutôt satisfaisants et nous permet de visionner un amortissement plus lent mais ce qui rend la mesure intéressante.

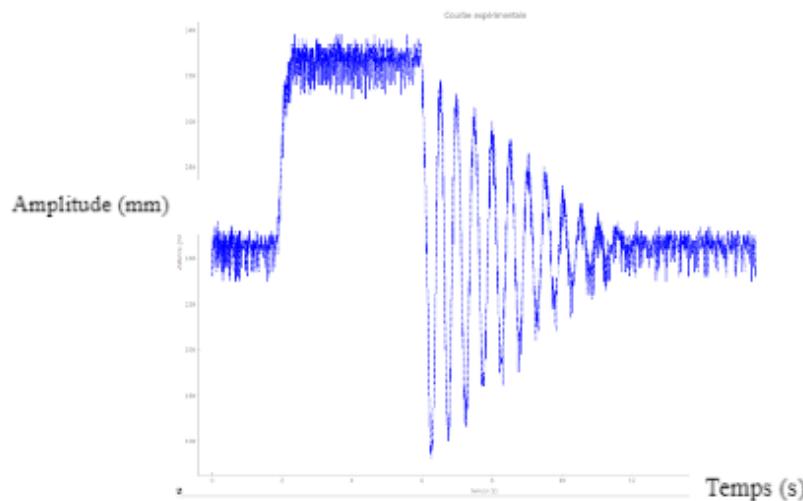


Figure 2 Oscillation libre avec un amortissement à air avec le PAD numéro 1

Une fois les mesures effectuées en oscillations libres nous essayons d'effectuer les mesures en oscillation forcée, afin de relever les différents propres, nous augmentons la tension aux bornes du moteurs et nous relevons une première fréquence propre qui correspond à environ 8 volts.

Le gros problème arrivé à cette tension qui correspondait à notre fréquence propre est que le système se met à prendre beaucoup d'amplitude et se met à arriver à la butée située en bas du système, dans ce cas aucune mesure n'est possible en oscillation forcée.

Plusieurs solutions s'offraient à nous à ce moment :

- a) Redimensionner le système
- b) Trouver un liquide qui amortit moins que l'eau
- c) Modifier certaines pièces du système

a) Redimensionnement du système

Pour ce qui est de redimensionner le système, cela reviendrait à modifier le rail qui guide la masse tout au long de l'axe vertical. Le problème de cette solution est que le ressort pour subir des déformations serait irréversible. En effet, le ressort possède une plage d'utilisation recommandée par le constructeur, en dehors de cette zone le ressort entre dans le domaine plastique où il subit des déformations irréversibles.

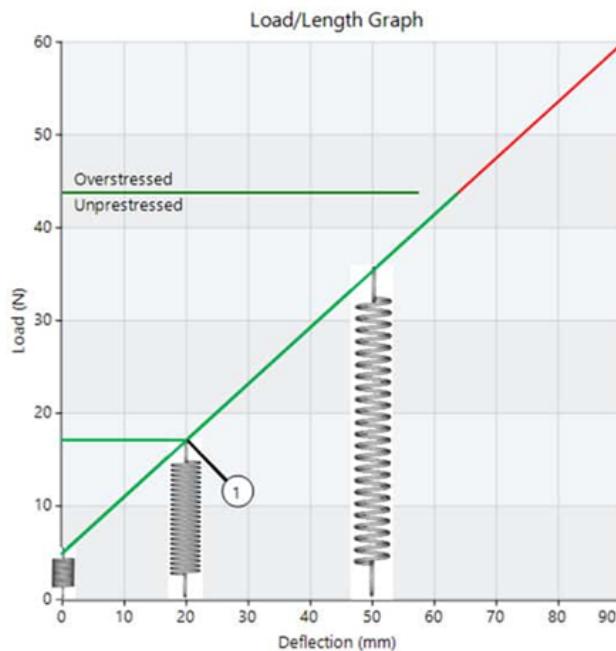


Figure 3 Courbe contrainte déformation d'après REM-RESSORTS.com

D'après la courbe ci-dessus, on peut très clairement comprendre que l'amplitude doit respecter un intervalle avant que le ressort n'entre dans sa plage de domaine plastique comme vu précédemment.

Cette solution demanderait donc beaucoup de changement voire un redimensionnement complet de la maquette ce qui nous pousse à voir donc la seconde option qui est de trouver un liquide qui amortit moins que l'eau.

b) Un liquide qui amortiront moins que l'eau

D'après une étude très sérieuse, l'eau savonneuse pourrait permettre de servir en tant que amortisseur, et ainsi permettre d'avoir un coefficient d'amortissement plus faible que l'eau dû à la présence d'air dans le liquide. Cependant pour mettre en place cette solution dans notre cas il faudrait mettre en place un réservoir qui permettrait de contenir ce liquide sans que cela ne déborde. Donc avant de concevoir ce jacuzzi, nous allons essayer la dernière solution.

c) Modification du PAD

Notre système possède un PAD qui se balade dans un cylindre qui est maintenant rempli d'air, notre PAD actuel quand bien même fermer (aucune ouverture entre les deux pièces) amortis encore de manière trop faible dû à l'espace entre les parois du cylindre et le système mobile.

La mise en place de segment tout autour de notre PAD en s'inspirant des systèmes utilisés dans les moteurs thermiques aurait pu être une bonne idée.

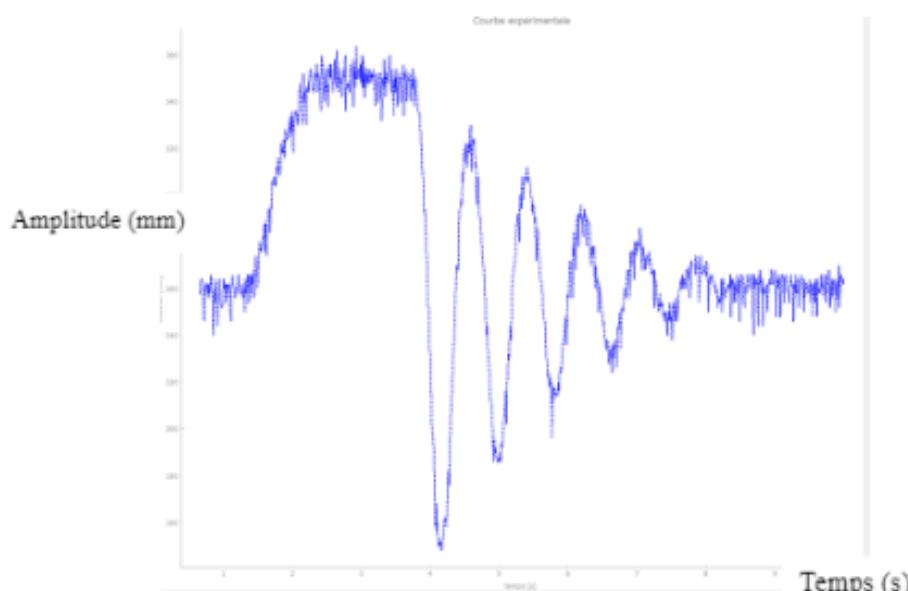


Segments d'un piston d'après auto-innovations.com

Toutefois nous possédions des PAD qui n'avaient jamais servis et après une mesure de ces nouveaux PAD, ils sont de quelques millimètres un peu plus larges.

Et cela nous permet de gagner un temps considérable en ne recommençant pas la conception d'une nouvelle pièce soit dit en passant plus complexe que les précédentes.

La mesure de l'amortissement libre avec le second PAD est plutôt satisfaisante avec un amortissement supérieur au PAD précédent toutefois inférieur à celui de l'eau.



Pour justifier ce choix d'amortissement on parlera des valeurs suivantes :

PAD :	PAD numéro 1		PAD numéro 2	
Amortissement :	EAU	AIR	EAU	AIR
Oscillation libre	Amortis très rapidement, inintéressant	Très intéressant, de belles oscillations, des temps d'amortissements intéressants.	Amortis extrêmement rapidement, ne correspond pas aux efforts exercés ici.	Des temps intéressants et ajustables grâce au PAD amovible qui permet de faire varier le temps d'amortissement.
Oscillation forcée	Difficile de trouver les différentes fréquences propres	Arriver en butée trop rapidement, risque d'endommager le système.	Ne correspond pas aux efforts exercés par le moteur l'amplitude possible avec l'excentrique.	A l'aide de l'ajustement possible avec l'ouverture entre les deux disques on peut avoir les fréquences propres sans arriver en butée.*

*Après une discussion avec Monsieur Pelt, le fait que le système tape la butée de fin de course est voulu, cela pourrait servir lors de différentes démonstrations. C'est pourquoi nous garderons un système qui une fois en oscillations forcées viennent rencontrer cette butée.

C'est donc avec ces différents éléments que nous allons commencer le post-traitement en oscillation libre.

Les premières études post-traitement ont été faites à la main avec un résultat plutôt long à obtenir mais ce qui nous a permis une compréhension du calcul afin d'obtenir à la fin une période propre et ainsi une fréquence propre.

Différents calculs de l'étude post traitement :

Pseudo-périodique : $d^2x/dt^2 + 2\lambda dx/dt + W_0^2 = 0$

Avec :

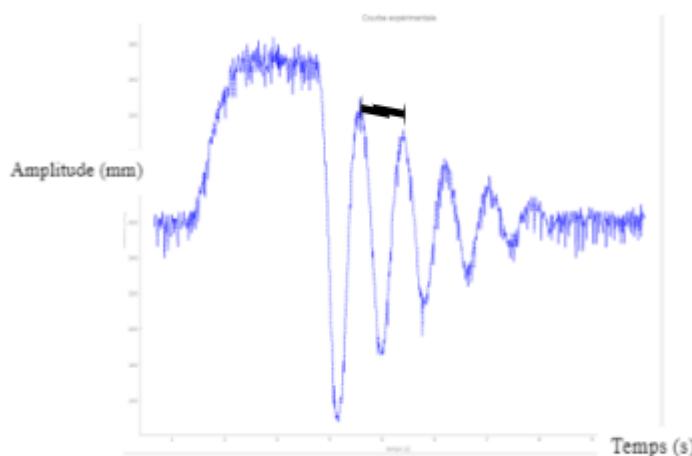
λ = Coefficient d'amortissement

W_0 = Pulsation propre

$x(t) = x_m e^{-\lambda t} \cos(\omega_1 t + \varphi)$, avec $\omega_1 = \sqrt{w_0 - \lambda^2}$

φ = phase à l'origine

x_m = constante



Pseudo période = 5,03-4,56

= 0,47 secondes

Décrément logarithmique = $\ln(348/332)$

= 0,047

4,56 secondes > amplitude = 348 mm

5,03 seconde > amplitude = 332mm

Finalement :

Décrément logarithmique est également égale à λT_1

$\Rightarrow \lambda = 0,1$ seconde

Ce qui nous donne :

$$T_0 = 0,2211 \text{ s}$$

$$F_0 = 4,522 \text{ Hz}$$

Ce calcul étant automatisé dans un code python nous pourrons ainsi gagner du temps si nous effectuons à nouveau des modifications sur le système à l'avenir, car un changement d'ouverture changerait le comportement de notre système et nous obligeraient à recommencer les calculs.

Résultat post traitement en oscillation libre :

Pseudo période	0,47 secondes
Décrément logarithmique	0,047
Coefficient d'amortissement	0,1
Période propre	0,2211 secondes
Fréquence propre	4,522 Hz

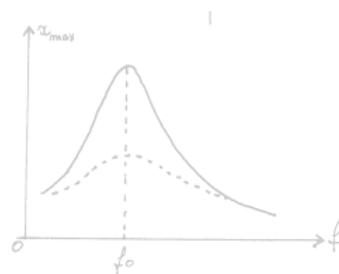
Pour ce qui est des oscillations forcées nous pourrons utiliser le code en python que nous avons trouvé à la fin du projet pour étudier ce signal.

2) Oscillations forcées

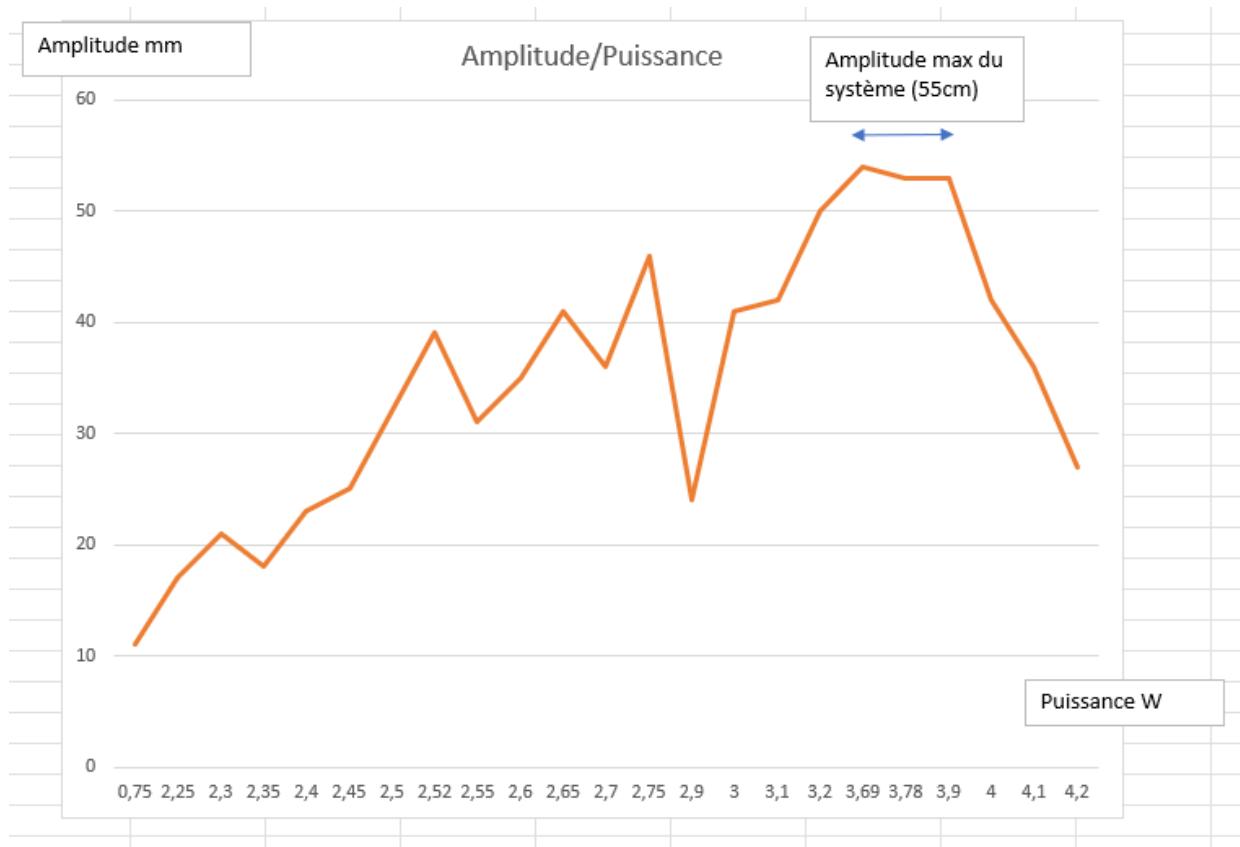
a) Temporel

Pour cette partie de l'étude post traitement en oscillations forcées de notre système, le principe de l'étude est de récolter un maximum de données et de trouver la fréquence propre de notre système.

Pour cela il suffit d'augmenter la vitesse et de trouver les différentes fréquences de rotation du moteur qui entraînent une importante amplitude de notre système.



Sur la figure ci-dessus nous pouvons voir la résonance d'élongation d'un système avec une fréquence propre f_0 qui entraîne une amplitude hors norme du système. Les différentes courbes (pointillées et trait plein) représentent la différence d'amortissement que nous imposons au système.



Dans notre cas, après avoir mis en rotation le moteur nous pouvons dire que la fréquence de résonance de notre système se trouve à une vitesse de rotation de 74 tours par minute.

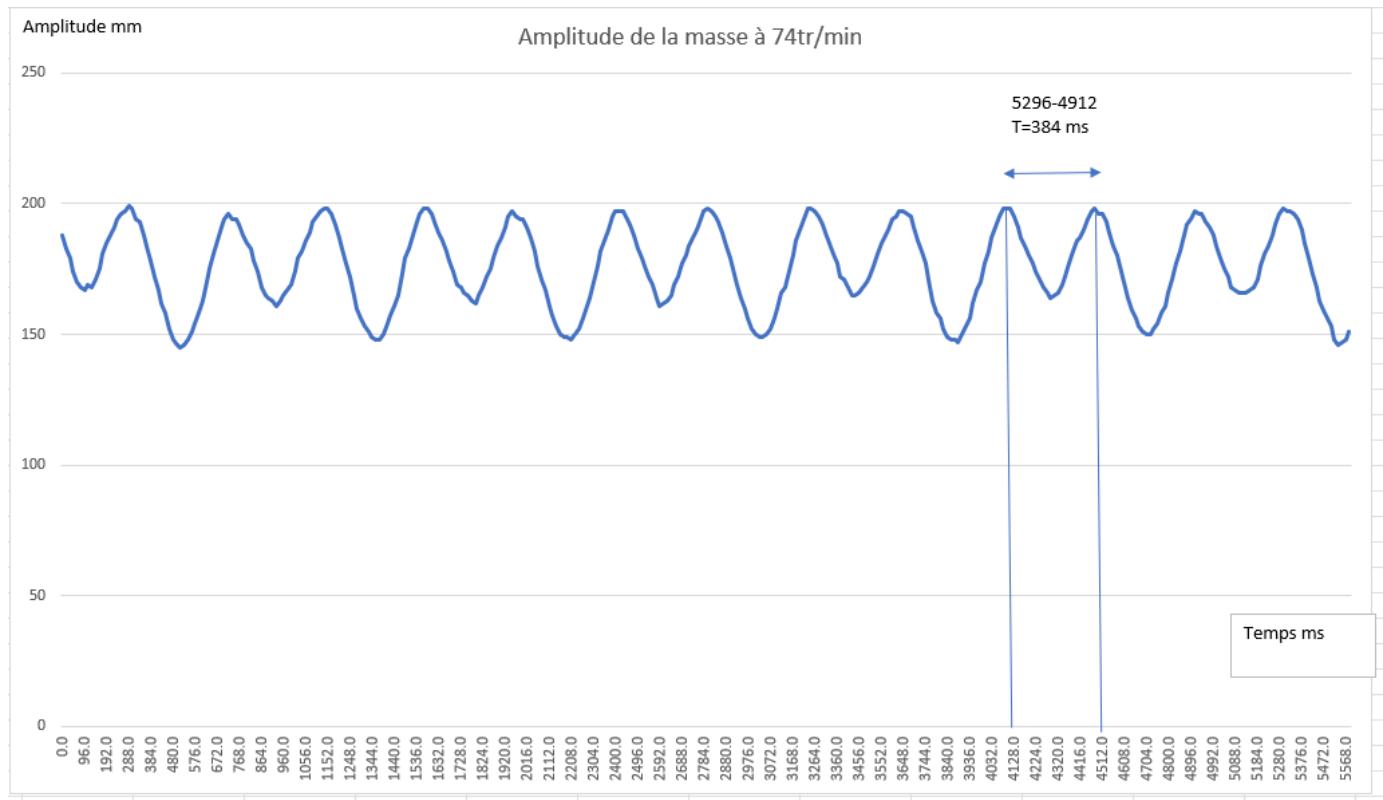
Cette fréquence de résonance à était plutôt simple à trouver grâce à la définition de la résonance qui veut que lorsque l'on atteint celle ci l'amplitude du système en question se met à avoir une amplitude hors norme. Après une longue mesure des amplitudes qu'adopte le système et avec les différentes variations de vitesse du moteur cette valeur à était simple à trouver.

Par manque de temps nous n'avons pas pu utiliser le programme en python pour étudier la fréquence propre du système, ce qui donne un gros problème dans notre étude c'est le manque cruel de points d'acquisition. Toutefois, ce graphique nous donne une idée de la fréquence propre du système.

On peut apercevoir plusieurs points qui pourraient correspondre aux différentes fréquences de résonance, mais le principal pic se trouve autour de 3,8 Watts. Il est impossible de donner une valeur précise du au dimensionnement de la maquette. Un redimensionnement du moteur pourrait être utile mais comme précisé précédemment le but de la maquette est d'afficher des amplitudes hors normes lorsque nous imposons une fréquence qui correspond à la fréquence de résonance.

b) Fréquentiel

Pour pouvoir mesurer le taux d'amortissement il faut passer dans le domaine fréquentiel afin d'avoir une fréquence et pouvoir comparer cette fréquence aux différentes études théoriques.



D'après ce graphique, la fréquence propre se trouverait donc à 2,6 Hz.

3) Cohérence des résultats

Une fois ces résultats obtenus, il convient de les comparer et de voir si les deux valeurs de fréquences propres obtenues sont cohérentes. Tout logiquement, les résultats sont bien différents et cela peut s'expliquer grâce à plusieurs raisons.

- La plus évidente est bien sûr que l'expérience ne peut être effectuée correctement au vu de la course possible limitée de la masse. Au cours de l'expérience les forts impacts de la masse sur la vis de fin de course sont explicites vis à vis de l'énergie que perd le système.
- Les calculs de fréquence propre ont été effectués à la main, il y a ici deux éléments qui ont pu engendrer des erreurs. Tout d'abord les arrondis qui ont été choisis au vu des calculs de logarithme népérien et autres calculs qui fournissent des résultats décimaux. Lors des calculs faits à la main, des paramètres ont dû être négligé également pour que le calcul reste possible.
- Des erreurs expérimentales qui ont été commises augmentent elles aussi la différence du modèle théorique et des résultats expérimentaux obtenus.

Pour conclure cette partie d'étude post traitement, une bonne évolution du projet serait de redimensionner le système pour permettre à la masse d'osciller librement tout en prenant garde à ne pas entrer dans la plage de déformation plastique du ressort. Le temps nous aura manqué également, pour permettre une étude plus approfondie de la partie post traitement le système doit être figé assez tôt.

I.5. Perspective d'évolution

Comme dit précédemment, une maquette avec un rail plus long permettrait de mieux voir l'amplitude de la masse une fois excitée à la résonance, tout en prenant en compte les différentes caractéristiques techniques du ressort et de la course possible de l'amortisseur. Une nouvelle maquette plus grande serait également une évolution bénéfique au projet de la Zg², en utilisant la maquette que nous utilisions comme V0, une V1 aux dimensions conséquentes et/ou avec une masse conséquente le résultat pourrait être impressionnant.

L'amortissement peut également faire l'objet d'une amélioration et permettre de voir l'impact de celui-ci sur le système.