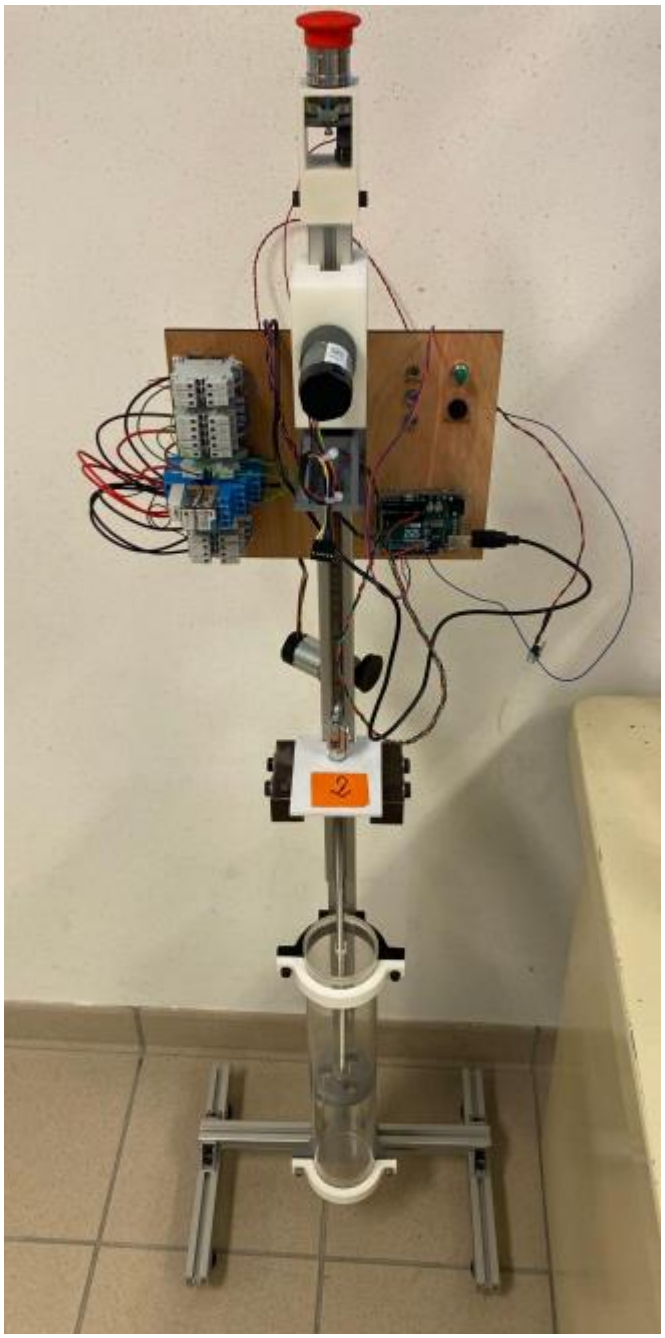
ZG Masse Ressort

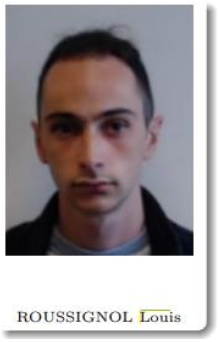
JAMBOU Clémence < c9jambou@enib.fr >
HARISTOY Clothilde <c9harist@enib.fr>
QJAOUJ El Ghalia < e1qjaouj@enib.fr >
THAUVEL Madeline <m0thauve@enib.fr>
ROUSSIGNOL Louis < l2roussi@enib.fr >

Expérimentation sur le système réel 8
CAO du système 5
Conception électrique CAO et cable management 4
Etude théorique du comportement vibratoire 1
Intégration physique de la carte de puissance 6

Pendant le module "Conception et Mécanique Vibratoire", notre mission consistait à améliorer la maquette utilisée dans la zone généraliste 2. Notre contribution visait à optimiser la préparation de cette zone afin d'assurer un déroulement optimal. Le système, qui repose sur un ensemble masse-ressort piloté par un moteur, comporte un capteur lidar positionné au niveau du moteur pour mesurer la distance jusqu'à la masse. Ce capteur est connecté à une carte électronique qui fonctionne en tandem avec une interface homme-machine (IHM). L'objectif principal était d'afficher les oscillations du système au cours de la phase de post-traitement.



I. Intégration physique de la carte de puissance



100%

I.1.Objectifs fixés

L'objectif de cette partie est d'utiliser la carte de puissance arduino motor shield, en testant les différents composants que l'on utilise et vérifier qu'on récupère bien toutes les informations des composants. Lors des années précédentes, un groupe avait utilisé la carte de puissance arduino et il y avait eu un problème avec cette dernière. C'est pourquoi une nouvelle carte de puissance a été intégrée Cette partie nous permettra de comprendre pourquoi est survenu ce problème

I.2. Etat initial de la tâche

En l'état, l'alimentation et la commande des différents composants fonctionnent, on peut piloter le moteur avec la carte arduino depuis l'IDE et récupérer les différentes informations. L'objectif initial du projet était d'alimenter le moteur grâce à la carte de puissance arduino motor shield. Actuellement, le moteur est alimenté et contrôlé avec la carte arduino R3 board et la carte de puissance L298N motor driver.

Afin de comprendre le problème survenu avec la première carte de puissance, j'ai décidé de repartir de zéro et de tester tous les composants afin de vérifier que nous pouvons tous les alimenter et que nous ne dépasserons pas les puissances admissibles par la carte de puissance motor shield.

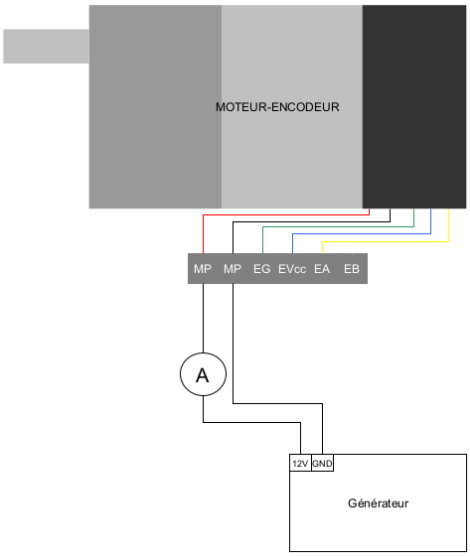
I.3. Test des différents composants

Afin de tester les différents composants, j'utiliserai un générateur CC, un multimètre et un oscilloscope avant de les câbler sur la carte.
Voici la liste des composants que nous utiliserons : carte arduino R3 board, carte de puissance arduino motor shield, capteur de distance et moteur-encodeur pololu.

I.3.a. Moteur-encodeur

Le moteur que nous utilisons actuellement est : 50:1 Metal Gearmotor 37Dx70L mm 12V with 64 CPR Encoder (Pololu). Généralement, le problème lorsqu'on alimente survient lors du

démarrage avec un pic d'intensité qui peut être 10 fois supérieur à l'intensité moyenne de ce qu'il consomme. Afin de vérifier ce pic, nous utiliserons la fonction peak du multimètre pour connaître l'intensité maximum.



Nous récupérons 2 informations : Imax= 2,13 A et Imoy= 0,16 A.

TECH SPECS

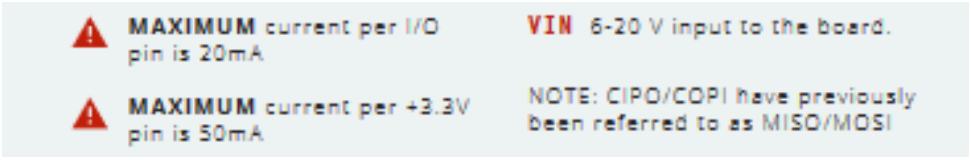
Operating Voltage	5V to 12V
Motor controller	L298P, Drives 2 DC motors or 1 stepper motor
Max current	2A per channel or 4A max (with external power supply)
Current sensing	1.65V/A
Free running stop and brake function	

Selon la datasheet de la carte de puissance arduino, le courant maximal admissible par la carte est 4A et celle utilisée actuellement est de 2A. Je peux donc conclure que si la carte que nous utilisons supporte ce pic, la carte arduino supportera ce pic.

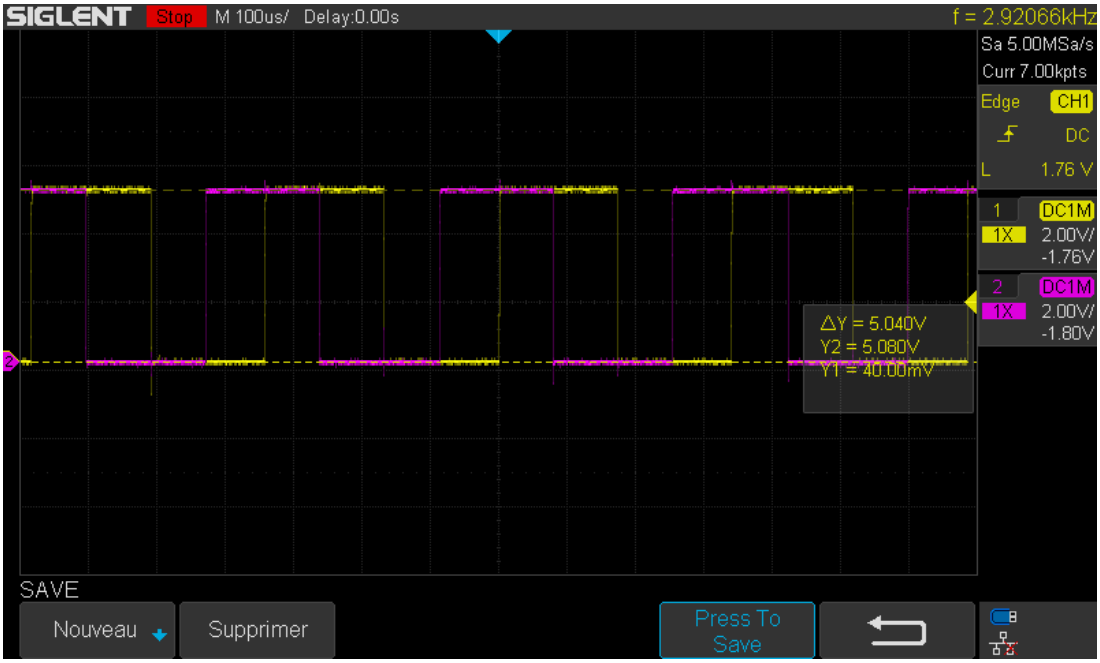
Cependant, le moteur a été testé sans charge, nous veillerons donc à tester le moteur avec charge pour être sûr que le courant maximal ne dépassera pas la limite de la carte. Le courant maximal du pic de démarrage indiqué par la data sheet est de 5,5A.

Quant à l'encodeur, la datasheet nous indique que l'on peut l'alimenter entre 3,5 et 20V. Les groupes précédents ont décidé d'utiliser le pin 5V de la carte.

La datasheet ne précise pas le courant maximum que peut fournir le pin, je pars donc du principe que c'est la valeur la plus petite que m'indique la datasheet par sécurité : 20mA.



Le multimètre m'indique Imoy= 8,15mA et nous récupérons bien les données de l'encodeur:



(En jaune l'encodeur A et mauve l'encodeur B)

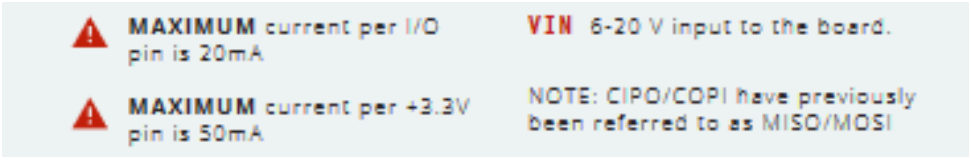
Le problème ne vient pas non plus de l'encodeur, donc la cause pourrait être le capteur.

I.3.b. Capteur de distance

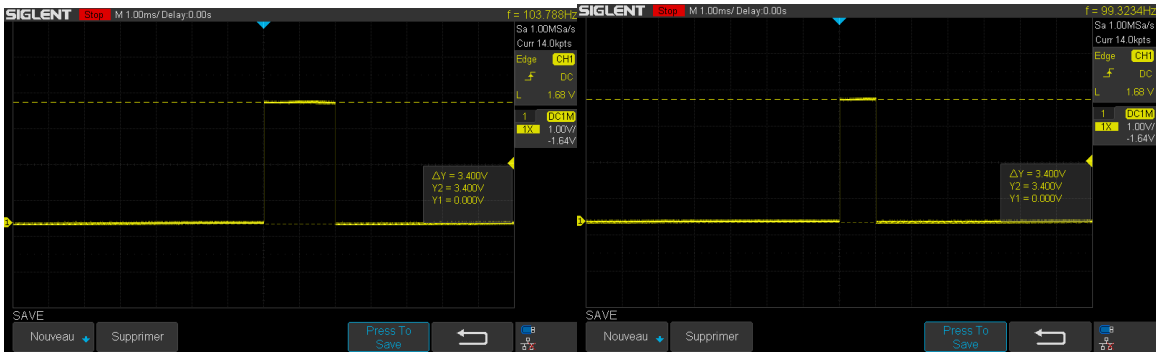
Actuellement, le capteur de distance est alimenté avec le pin 3,3V de la carte arduino r3 board. Le problème aurait pu survenir avec l'alimentation de ce capteur. Si nous dépassons le courant admissible par la carte sur le pin 3,3V, la carte pourrait être détériorée.

Selon la datasheet du capteur, le capteur consomme 30mA au maximum et la carte peut admettre un courant max de 50mA.

Maximum range:	200 cm
Sampling rate:	30 Hz[1]
Minimum operating voltage:	3.0 V
Maximum operating voltage:	5.5 V
Supply current:	30 mA[2]
Output type:	digital[3]



Après avoir vérifié au multimètre, le courant maximal indiqué est Imax= 32mA et Imoy= 24mA. On récupère bien la donnée :



(À gauche à distance max et à droite distance min)

J'en conclus que le problème ne vient pas du capteur. Après avoir testé tous les composants, la seule raison pour laquelle la carte a eu un problème est un mauvais câblage. En effet, les courants admissibles par la carte sont respectés, je peux donc alimenter les composants avec la carte de puissance arduino motor shield.

I.4. Intégration de la carte de puissance arduino

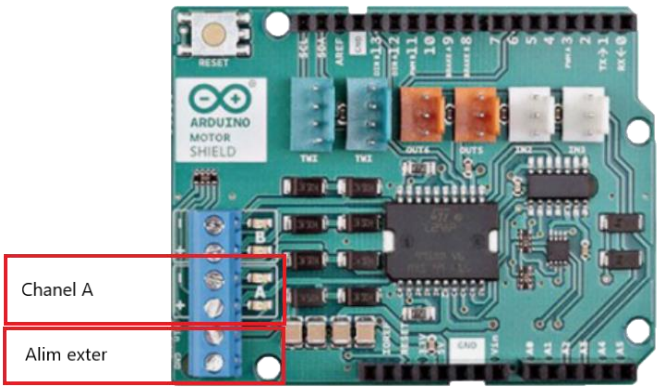
Après avoir vérifié les composants, je peux donc utiliser la carte arduino et sa carte de puissance pour piloter le moteur. J'alimenterai le moteur avec une alimentation extérieure de 12V, comme c'est le cas actuellement avec l'autre carte de puissance.

I.4.a. Montage de la carte de puissance

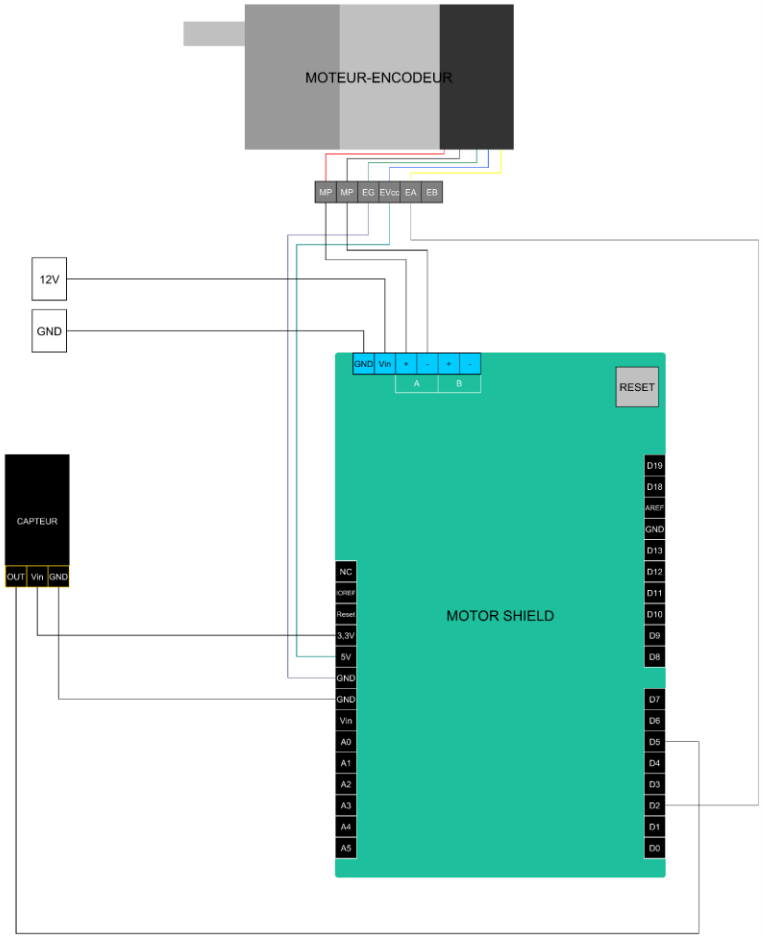
Pour monter la carte de puissance sur la carte arduino, il suffit juste de faire correspondre les pins de la carte arduino avec ceux de la carte de puissance tel que :

I.4.b. Câblage des composants et test du moteur

J’utiliserai un générateur pour alimenter le moteur, j’ai décidé d’alimenter le moteur sur le chanel A de la carte de puissance.



Avec les datasheets des différents composants le montage sera le suivant :



Afin de tester le pilotage du moteur, j’utiliserai le code suivant :

test_mot.ino

```
float Intensite=0;
void setup() {
  Serial.begin(9600);
  //Configuration du Canal A
 pinMode(12, OUTPUT); // Broche Arduino réservée pour le sens de rotation du moteur A
 pinMode(9, OUTPUT); // Broche Arduino réservée pour le freinage du moteur A
 Serial.print("Start");

}

void loop() {
 digitalWrite(12, LOW); // Le moteur A tourne dans le sens normal
 digitalWrite(9, LOW); // Désactivation du frein moteur A
 analogWrite(3, 255); // Demie vitesse pour le moteur A
 Intensite=analogRead(A0); // Lecture de l'entrée analogique A0
 Intensite=map(Intensite, 0, 255, 0, 2000); // Mappage de l'intensité en milliAmpère
 Intensite=Intensite/1000; // Convertion de l'intensité en Ampère
 Serial.print(" Intensité ="); // Affichage du texte
 Serial.println(Intensite); // Affichage de l'intensité conosmmée par le moteur connecté sur la voie A
}
```

Ce code permet de faire tourner le moteur en continu et il fonctionne. Cela conforte l'idée que les précédents groupes avaient mal câblés les composants à la carte de puissance.
Lors du 1er test, la carte arduino a grillé. La raison n'était pas un problème de câblage mais, pendant le test de la carte, je la tenais dans ma main et sans m'en rendre compte j'ai relié 2 pins et créé un court-circuit.
Le 2nd test a été un succès. Nous utilisons donc la carte de puissance arduino motor shield pour piloter le moteur.

I.5. Bilan comparatif avec les objectifs initiaux

En conclusion, l'ancienne carte de puissance a été remplacée par la carte de puissance arduino motor shield. Ce changement nous permettra de réduire le nombre de câbles, de gagner de la place sur la plaque ,où l'arduino est fixée, et d'encaisser plus de courant que la précédente.

II. Expérimentation sur le système réel

50% 50%

II.1. Objectifs fixés

Nous avons plusieurs objectifs :

- vérifier la sécurité câblé
- prendre en main la maquette
- création d'une base de données en boucle ouverte et boucle fermé
- Mettre en place un régulateur PID.

II.2. Etat initial de la tâche

Au commencement de notre travail sur ce sujet, nous avons examiné les documents disponibles. Il était évident qu'aucune base de données archivées n'était en place, et les informations concernant le post-traitement étaient extrêmement limitées. De plus, le tutoriel expliquant les bibliothèques nécessaires pour exécuter ou modifier le code de l'interface homme-machine (IHM) manquait de détails et présentait de nombreuses dépendances. La communication entre l'IHM et le programme du capteur était unilatérale, ce qui empêchait toute interaction de l'IHM avec le programme.

La maquette était déjà assemblée et câblée à notre arrivée. Dans le but de mieux comprendre son fonctionnement, nous avons tenté de reproduire les procédures des vidéos présentes dans les archives. Cependant, nous avons rapidement remarqué que certains câbles étaient débranchés ou mal connectés, ce qui a mis en évidence un ou plusieurs problèmes à résoudre.

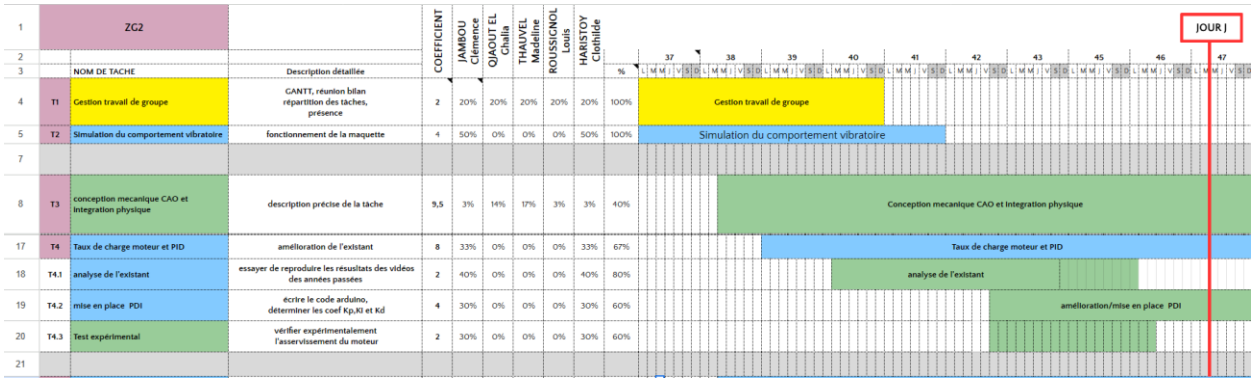
Nous avons découvert que certains scripts Arduino étaient déjà en place. Voici ce que nous avions initialement :

Un programme était en place pour la régulation en boucle fermée d'un moteur 12V à l'aide d'un correcteur PI. Ce programme exigeait un câblage spécifique car il utilisait 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.

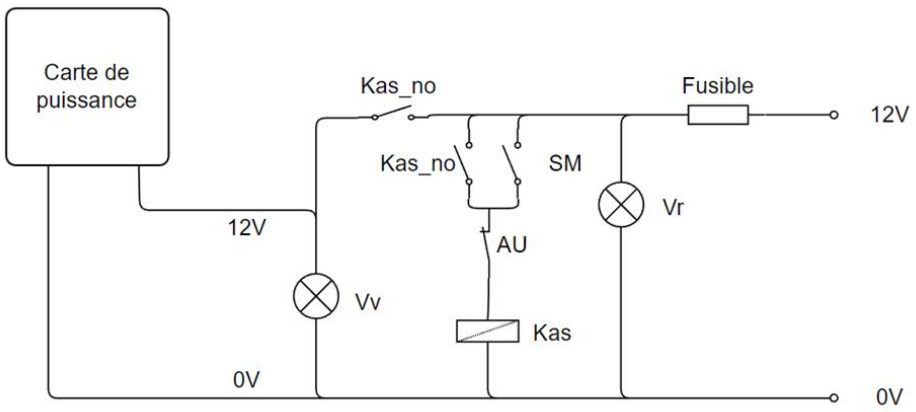
Un autre programme était destiné à lire, convertir et transmettre en continu les données provenant d'un capteur Lidar via le port série. Ce programme permettait à l'IHM de suivre la position du système pour une représentation graphique. Néanmoins, la communication entre

l'IHM et le programme semble unidirectionnelle, ce qui signifie que l'IHM ne peut pas interagir avec le programme.

II.3. Planification



II.4. Choix techniques



Grâce à la documentation existante, nous avons pu effectuer une vérification de la sécurité des câbles, conformément au schéma illustré ci-dessus. Nous pouvons confirmer que la sécurité câblée est opérationnelle, car nous l'avons soumise à des tests réussis. De plus, nous avons rétabli la connexion des câbles qui étaient débranchés. Nous avons suivi les étapes des tutoriels et des vidéos pour reproduire le même comportement.
Lorsque le système est en oscillation libre, l'acquisition de données fonctionne correctement. Cependant, en cas d'excitation forcée, le comportement diffère de ce qui était attendu. Nous observons un comportement aléatoire qui nous empêche d'obtenir des données fiables.

Notre démarche pour la création d'une base de données expérimentales, en ce qui concerne un système masse-ressort en oscillation libre et immergé dans de l'eau, s'est déroulée de la manière suivante :
Pour l'oscillation libre, nous avons utilisé des clés Allen comme gabarit pour ajuster l'ouverture de l'amortisseur. En appliquant une force constante à partir de la butée, nous avons utilisé des clés Allen de différentes tailles, notamment 10, 6, 4, 2, etc. afin de moduler l'amplitude des

oscillations. Nous avons ensuite répété l'expérience en remplissant le tube d'eau pour simuler des conditions immergées.

En poursuivant nos investigations, nous nous sommes penchés sur les aspects de communication entre les différents composants du système, notamment les temps de rafraîchissement. L'analyse du code écrit par quelqu'un d'autre s'est avérée être un exercice complexe et chronophage. Dans ce contexte, nous avons formulé plusieurs hypothèses pour tenter de comprendre le comportement aléatoire du système, cherchant ainsi à éclaircir les facteurs contribuant à ses variations inattendues.

Nous avons dans un premier temps pensé à un PID mal réglé, mais après plusieurs tests il nous était impossible de nous rapprocher d'un asservissement correct du moteur.

Le réglage d'un correcteur PID (Proportional-Integral-Derivative) est un processus courant en ingénierie de contrôle pour ajuster les paramètres Kp, Ki et Kd de manière à obtenir une réponse souhaitée d'un système de contrôle. Voici le processus général en trois étapes pour régler ces paramètres que nous avons suivi :

- Réglage de Kp (Proportionnel) :

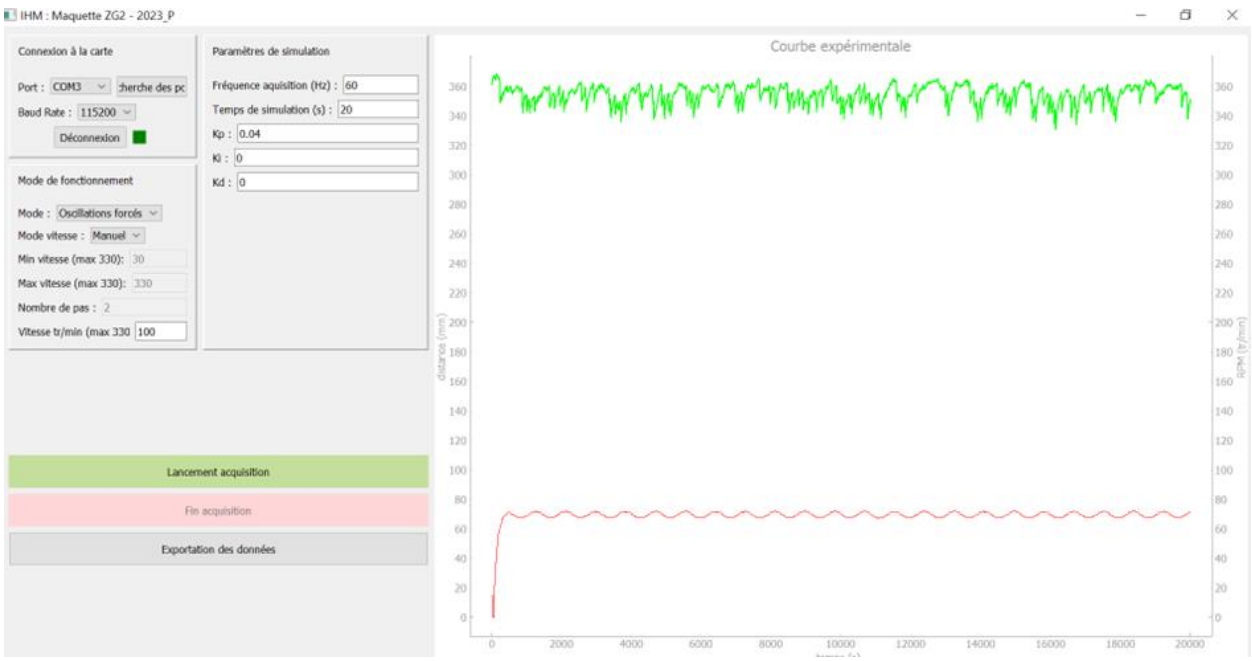
- Commencez par fixer les valeurs de Ki et Kd à zéro, de manière à avoir un contrôleur P (proportionnel) pur.
- Augmentez le Kp de manière progressive à partir de zéro. Observez la réponse du système à une entrée (par exemple, un échelon) et notez les changements.
- Augmentez le Kp jusqu'à ce que le système atteigne le point où il commence à osciller (oscillations incontrôlées) ou à être instable.

- Réglage de Ki (Intégral) :

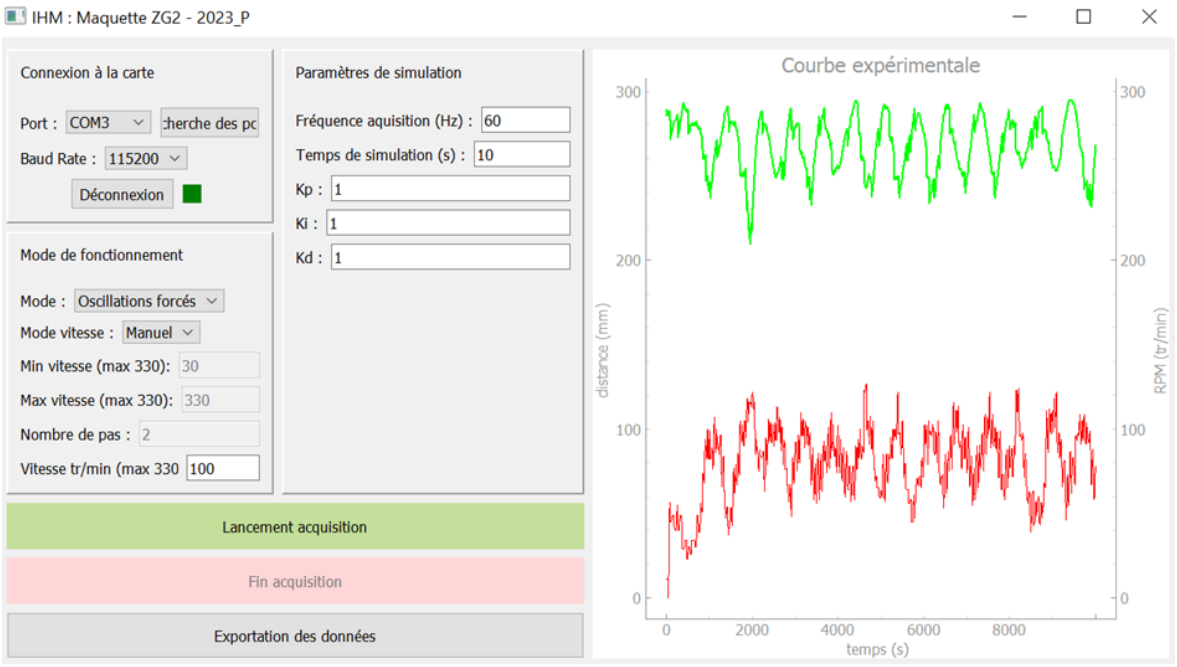
- Augmentez Ki de manière progressive (en laissant Kd à zéro) pour éliminer tout écart en régime permanent (erreur statique) causé par le contrôle proportionnel.
- Gardez à l'esprit que Ki doit être augmenté prudemment, car une augmentation excessive peut entraîner une réponse lente et des oscillations.

- Réglage de Kd (Dérivatif) :

- Réglez Kp et Ki à leurs valeurs trouvées précédemment.
- Ajoutez maintenant le terme dérivatif. Augmentez Kd progressivement pour améliorer la réponse transitoire, c'est-à-dire réduire le temps de montée, l'amortissement des oscillations, etc.
- Une valeur de Kd élevée peut réduire la rapidité de réponse du système et introduire des oscillations si elle est trop importante.



Notre principal objectif est de mettre en place un asservissement du moteur de manière à maintenir une vitesse constante, même en présence d'oscillations de la masse.
Dans l'image ci-dessus, nous nous trouvons dans une situation d'oscillation forcée, où nous avons défini une consigne de 100 tr/min. Cependant, malgré nos efforts, nous ne parvenons qu'à atteindre 70 tr/min. Nous avons même essayé de réduire la consigne à 50 tr/min après un redémarrage de l'interface homme-machine (IHM), mais cette fois-ci, nous nous retrouvons bloqués à une valeur bien en deçà de la consigne, sans jamais parvenir à l'atteindre.
Avec un système qui ne parvient pas à atteindre la vitesse demandée comme sous l'image ci-dessous, nous observons une instabilité dans la vitesse, et les coefficients du régulateur PID que nous utilisons ne permettent pas de résoudre ce problème de manière satisfaisante.



Nous avons alors étudié l'état des différentes variables du code pas à pas. Et avons remarqué que la valeur de la vitesse instantanée n'était pas mise à jour et restait proche de 0. Hors la plupart des calculs sont basés sur cette donnée.

Nous avons pris contact avec d'anciens membres du projet.

50% 50%

II.5. Attribution des pins

Le projet prenant du retard et l'intégration de la nouvelle carte de puissance ayant eu lieu, nous avons décidé de repartir de zéro.

II.5.a. Commande du moteur

L'intégration de la carte de puissance nécessite de modifier les pins pour piloter le moteur. Comme nous utilisons le channel A pour contrôler le moteur, la datasheet de la carte de puissance nous indique les pins que nous utiliserons :

Function	pins per Ch. A	pins per Ch. B
Direction	D12	D13
PWM	D3	D11
Brake	D9	D8
Current Sensing	A0	A1

D9 : Activer/désactiver le frein moteur
D3 : Signal PWM pour contrôler la vitesse du moteur
D12 : Choisir le sens de rotation du moteur

Dans le code de notre programme nous retrouvons l'initialisation de ces pins dans la fonction *setup()*.

II.5.b. Récupération des données de l'encodeur

L'encodeur nous permet de connaître la vitesse de rotation du moteur. Pour récupérer cette information, nous devons savoir comment fonctionne l'encodeur. C'est un encodeur magnétique en quadrature qui a une résolution de 64 comptes par tour. A chaque fois qu'un aimant passe

devant le capteur magnétique, une impulsion est visible en sortie. Nous utiliserons donc un pin qui, lorsqu'il y a un front montant, déclenche une interruption et appelle la fonction qui incrémente un compteur.

Avec la datasheet de la carte arduino, nous avons choisi d'utiliser le pin 2 pour récupérer les informations de l'encodeur. Etant donné que nous n'avons pas besoin de connaître le sens de rotation de notre moteur, nous utiliserons uniquement la sortie d'un des encodeurs (A ou B). La vitesse du moteur se déduira grâce à la formule suivante:

$$RPM = \frac{compteur \times R \times 4 \times 60 \times 1000}{\Delta t \times 64}$$

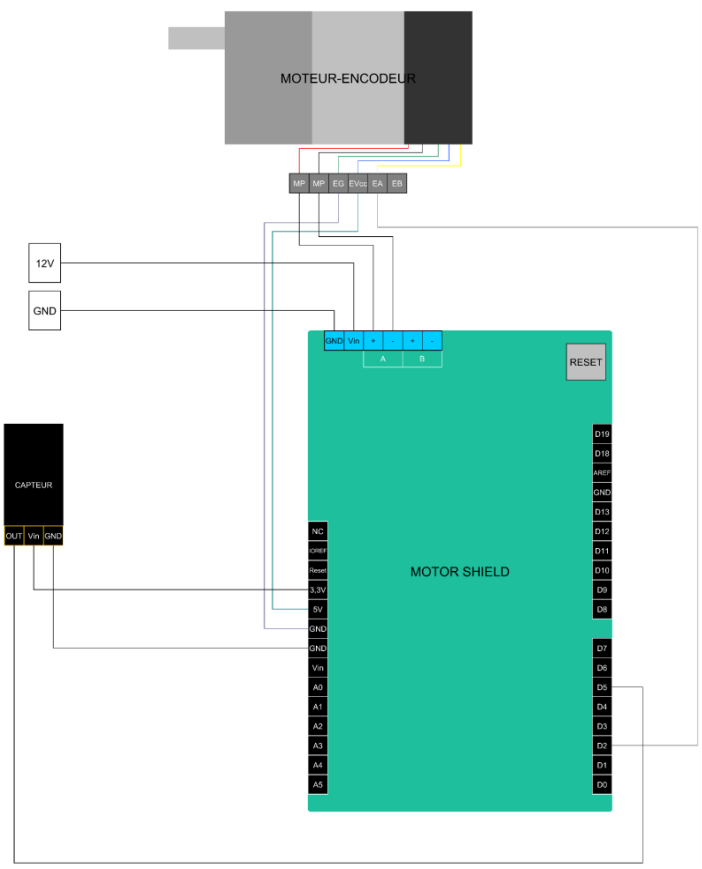
RPM : tour par minute
Δt : temps en milliseconde
R : Rapport de réduction (1/50)

II.5.c. Récupération des données du capteur

Le capteur IF émet une onde infrarouge qui, en fonction du temps mis par le faisceau pour atteindre l'objet et revenir au récepteur, envoie en sortie un signal carré plus ou moins large. Comme le capteur envoie émet dès qu'il a reçu le dernier signal, nous pourrons récupérer le temps avec la largeur d'impulsion grâce à la fonction *pulseIn(pin,mode)*.

II.5.d. Schéma électrique final

Le schéma électrique montre l'attribution des pins :



II.6. Modification du programme arduino

Même si nous sommes repartis de zéro, nous nous sommes tout de même inspirés du code des années précédentes. L'objectif était de conserver l'ihm et de supprimer les données inutiles, afin d'optimiser le programme.

II.6.a. Initialisation des pins

La fonctions setup permet d'initialiser les différents pins que nous utiliserons et un préréglage du capteur :

```
void setup() {
  Serial.begin(115200);//Fixe le débit de la liaison série
  pinMode(12, OUTPUT); // Broche Arduino réservée pour le sens de rotation du moteur A
  pinMode(9, OUTPUT); // Broche Arduino réservée pour le freinage du moteur A
  pinMode(3,OUTPUT);// PWM
  pinMode(2,INPUT_PULLUP);// Broche encodeur
  attachInterrupt(digitalPinToInterrupt(2),fencodeur, RISING);//Appel de la fonction fencodeur sur front-montant
  digitalWrite(12, HIGH); // Le moteur tourne dans le sens normal
  digitalWrite(9, HIGH); // Activation du frein moteur A
  analogWrite(3,0); // Pas de vitesse pour le moteur A (PWM)
```

```
  int i = 0;  //pre set capteur
  while( i <= 40){
    Capteur();
    delay(10);
    i += 1;
  }
  }
```

II.6.b. Réception des données

Les différents paramètres que nous voulons contrôler pour l'acquisition sont : la vitesse du moteur (tr/min), la période d'échantillonnage, le statut de l'acquisition, le mode d'acquisition, les coefficients du correcteur PI et la durée d'acquisition. Tous ces paramètres seront rentrés et envoyés depuis l'IHM en liaison série. Cette trame, uniquement envoyée avant le début de l'acquisition, mettra à jour tous les paramètres nécessaires à l'acquisition.

Si l'on reçoit une trame, on la lit comme cela :

```
if (Serial.available()){ //si on reçoit une tramme
      lectureIHM(); //lecture de la trame envoyée depuis l'ihm
    }
```

Le format de la trame reçu par l'arduino est : *"Te, state_acq, stat_mode_osil, moteur, kp, ki, kd, vitesse_tour_min, duree_acquisition"*. La fonction lecture met à jour ces paramètres :

```
void lectureIHM(void){
  String message = Serial.readStringUntil('/'); // Lit la chaîne de
caractères jusqu'au caractère '/'
  float numValues = toFloatArray(message, values, 9);//transforme les
chaines de caractères en float
  //Mise à jour des paramètres en fonction des valeurs reçues
  Te = (unsigned long)values[0];//période d'échantillonage
  state_acq = (int)values[1];//statut d'acquisition
  state_mode_oscil = (int)values[2];//Mode d'aquisition
  _moteur = (int)values[3];
  kp = values[4];
  ki = values[5];
  kd = values[6];
  vitesse_tour_min = (int)values[7];
  cde=(float)vitesse_tour_min;
  map((long)cde, 0, 200, 0, 255);
  cde=(float)cde;
  duree_acquisition = (unsigned long)values[8];
  duree_acquisition=duree_acquisition*1000;
}
```

II.6.c. Envoie des données depuis l'arduino

Les valeurs récupérées depuis l'IHM sont les suivantes : temps(ms), vitesse mesurée par l'encodeur (tr/min), distance (capteur) et si la simulation est finie ou non. L'envoie se fait avec la fonction *envoiDonnes()* où les données sont stockées dans un buffer :

```
void envoiDonnees(void){
    char buffer[20]; // Créer un tampon pour stocker la chaîne de
caractères formatée
    sprintf(buffer, "T%luR%dP%dC%d", tps,(int)nb_tour_min, pos,
completSimu); // Formater les données dans la chaîne de caractères
    Serial.println(buffer); // Envoyer la chaîne de caractères formatée
via Serial.print()
  }
```

Dans le but d'optimiser le programme, nous avons décidé de modifier la taille du buffer excessivement grande pour un buffer plus petit (1+4+1+2+1+2+1+2=14). Par sécurité, nous prendrons 20ms.

II.6.d. Gestion des états et modes d'acquisition

La variable *state_acq* peut prendre plusieurs valeurs :

- 1: état d'acquisition
- 0: état d'arrêt acquisition
- 2: état d'attente demande d'acquisition

```
if(state_acq == 1){//Demande d'acquisition···
else if(state_acq == 0){//Arrêt acquisition···
else if(state_acq == 2){//Attente···
```

La variable *stat_mode_osil*, servant uniquement dans le cas où l'on fait une acquisition, peut prendre deux valeurs :

- 0: Mode oscillation libre

```
    if(state_mode_oscil == 0){//Mode oscillation libre
        Capteur();//mesure de position
    }
```

- 1: Mode oscillation forcée

```
else if(state_mode_oscil == 1){//Mode oscillation forcé
    if (moteur==0){//Moteur à l'arrêt
      Capteur();//mesure position
      alu_mot();//Démarre le moteur
      encodeur();//mesure vitesse
      envoiDonnees();
      previous_time_corr=current_time;
      previous_time_acq=current_time;
    }
```

```
      else{//Moteur est déjà démarré
        if(tps_corr>5){
          Capteur();
          encodeur();//mesure vitesse rot
          Correcteur();
        }}}
```

II.6.e. Gestion du temps

- **Version 1**

Dans notre première version, nous utilisons 4 variables pour la gestion du temps : *current_time*, *tps*, *previous_time* et *previous_time_enc* ; et 2 fonctions : millis(), qui permet d'avoir le temps écoulé en millisecondes depuis la mise en tension de l'arduino et delay(), permettant de faire une pause pendant un temps donné dans le programme.

La variable *tps* sert à savoir si le temps écoulé depuis le début de l'acquisition est supérieur à la durée d'acquisition tel que :

```
      if(duree_acquisition<tps){
        state_acq = 0;}
```

Dans ce cas, nous changeons l'état d'acquisition et signalons à l'IHM que l'acquisition est terminée, en mettant à jour la valeur de *complet_Simu,*.

La variable *tps* est mise à jour au début de la boucle avec *current_time* et *previous_time* comme ceci :

```
current_time=millis();
tps=current_time-previous_time;
```

previous_time est quant à elle mise à jour lorsqu'on est en attente d'une acquisition.

La variable *previous_time_enc* nous sert pour le calcul de la vitesse de rotation du moteur. En effet, afin de gagner en précision sur cette vitesse, dont on se base pour l'asservissement du moteur, nous la mettons à jour lorsque nous appelons la fonction *encodeur()* tel que :

```
void encodeur(void){
  current_time=millis();//met à jour le temps courant pour plus de
précision
  nb_tour = compteur/(50*64/4);
  nb_tour_min=nb_tour*60*1000/((float)(current_time-previous_time_enc));
  previous_time_enc=millis();//variable spécifique pour précision
  compteur=0;}
```

Après avoir remarqué que la période n'était pas la même que la théorique et suite à une discussion avec monsieur Bourgeot nous avons décidé de supprimer la fonction delay() du programme, de corriger la vitesse du moteur périodiquement et d'utiliser d'autres variables pour la gestion des envois de données périodiques.

- **Version 2**

Dans cette seconde version nous utilisons 10 variables pour la gestion du temps et une fonction pour les mettre à jour.

La variable *tps_duree_acq* permet de savoir si la durée d'acquisition a été dépassée. Si c'est le cas, nous mettons fin à l'acquisition.

```
    if(duree_acquisition<tps_duree_acq){
      state_acq = 0;}
```

La variable *tps_corr* sert à connaître la vitesse du moteur et à corriger cette dernière toutes les 5ms.

```
    if(tps_corr>10){
         Capteur();
         encodeur();//mesure vitesse rot
         Correcteur();}
```

La variable *tps_acq* sert à envoyer les données dans le terminal série s'il est supérieur à Te.

```
    else if(tps_acq>Te){
     envoiDonnees();
     previous_time_acq=current_time;}
```

Afin de comparer les 2 versions, nous avons récupéré les données dans un tableau avec une période d'acquisition de 20 ms :

Nombre point théorique total à 20ms	750
Nombre de point réel total	497
Période d'échantillonnage réelle (ms)	30,1810865191147
Temps de calcul (ms)	10,1810865191147
Nombre de points théorique par période	37,5
Nombre de points réel par période	24

Version 1

Nombre point théorique total à 20ms	750
Nombre de point réel total	687
Période d'échantillonnage réelle (ms)	21,8340611353712
Temps de calcul (ms)	1,83406113537118
Nombre de points théorique par période	37,5
Nombre de points réel par période	34

Version 2

Nous voyons maintenant que le temps de calcul a drastiquement été réduit, nous avons désormais un échantillonnage pertinent. Mais il nous faut désormais déterminer la période minimale d'échantillonnage que nous pouvons utiliser pour chaque mode.

Pour connaître ces limites, nous avons testé différentes périodes d'acquisitions allant de 5ms à 20ms en les incrémentant de 5ms.

<u>Cas oscillations libres</u> :

Nombre de points théorique total à 5 ms	1000
Nombre de points réels total	607
Période d'échantillonnage réelle (ms)	8,23723228995058
Temps de calcul (ms)	3,23723228995058

Nombre de points théorique total à 10 ms	500
Nombre de points réels total	453
Période d'échantillonnage réelle (ms)	11,037527593819
Temps de calcul (ms)	1,03752759381898

Nombre de points théorique total à 15 ms	333,333333333333
Nombre de points réels total	272
Période d'échantillonnage réelle (ms)	18,3823529411765
Temps de calcul (ms)	3,38235294117647

Nombre de points théorique total à 20 ms	250
Nombre de points réels total	226
Période d'échantillonnage réelle (ms)	22,1238938053097
Temps de calcul (ms)	2,12389380530973

Grâce à ces tableaux, nous pouvons voir que la meilleure période d'échantillonnage et la plus petite est 10ms lorsque nous mettons à jour la position toutes les 10ms.

Cas oscillations forcées :

Nombre de points théoriques total à 5ms	1000
Nombre de points réels total	720
Période d'échantillonnage réelle (ms)	6,94444444444444
Temps de calcul (ms)	1,94444444444444

Nombre de points théoriques total à 10ms	500
Nombre de points réels total	487
Période d'échantillonnage réelle (ms)	10,2669404517454
Temps de calcul (ms)	0,266940451745381

Nombre de points théoriques total à 15ms	333,333333333333
Nombre de points réels total	380
Période d'échantillonnage réelle (ms)	13,1578947368421
Temps de calcul (ms)	-1,8421052631579

Nombre de points théoriques total à 20ms	250
Nombre de points réels total	256
Période d'échantillonnage réelle (ms)	19,53125
Temps de calcul (ms)	-0,46875

Les résultats nous permettent de dire que la période d'échantillonnage la plus petite que nous pouvons utiliser est 10ms lorsque nous corrigeons la vitesse et mettons à jour la position toutes les 5ms. Pour certains cas nous sommes même en sur échantillonnage, pour remédier à cela nous évoquerons des pistes d'améliorations.

II.6.f. Correcteur PID

La mise en place d'un correcteur est nécessaire pour que la vitesse réelle que l'on souhaite soit la plus proche possible de la consigne. Le système masse-ressort aura une influence sur le moteur faisant augmenter ou diminuer sa vitesse. Pour cela nous utilisons 6 variables : *kp* coefficient du correcteur proportionnel, *ki* coefficient du correcteur, *kd* coefficient du correcteur dérivé, *up* la réponse corrigée proportionnelle, *ui* la réponse corrigée intégrale, *ud* la réponse corrigée dérivée, *erreur_prec*, *cde* la commande envoyée au moteur et *erreur* la différence entre la consigne et la mesure. La fonction *Correcteur()* est appelée toutes les 5ms :

```
void Correcteur(void){
  erreur=(float)vitesse_tour_min-nb_tour_min;
  up=kp*erreur;
  ui=ui+ki*erreur*5;//corrige la vitesse toutes les t=10ms
  delta_erreur=erreur-erreur_prec;
  ud=delta_erreur*kd/5;
  erreur_prec=erreur;
  cde=ui+up+ud;
  if(cde>=0){
    digitalWrite(12, HIGH);//tourne dans le sens normal
```

```
    if(cde>255){
      cde=255;}}
  else if(cde<0){
    cde=abs(cde);//on prend la valeur absolue
    digitalWrite(12, LOW);//tourne dans l'autre sens
    if(cde>255){
      cde=255;}}
  analogWrite(3,(int)cde);//cde doit être entre 0 et 255
  previous_time_corr=millis();}
```

En BO, la vitesse du moteur oscillait beaucoup à cause des perturbations. La première hypothèse que nous avons retenue était que le système masse ressort avait une influence sur le moteur dû à sa masse importante. Pour confirmer cette hypothèse, nous avons fait un essai en BO à 80 tr/min avec une période d'échantillonnage de 20 ms, en BF avec correcteur proportionnel, correcteur PI et PID:

Consigne vitesse moteur (tr/min)	80
Fréquence de rotation moteur (Hz)	1,33333333333333
Fréquence perturbations (Hz)	1,34048257372654
Différence consigne-vitesse min (tr/min)	32
Différence consigne-vitesse max (tr/min)	47
% erreur max	58,75 %
Valeur moyenne (tr/min)	88,4093023255814
% erreur moyenne	10,51 %

BO

Consigne vitesse moteur (tr/min)	80
Fréquence de rotation moteur (Hz)	1,33333333333333
Fréquence perturbations (Hz)	difficile à identifier
Différence consigne-vitesse min (tr/min)	22
Différence consigne-vitesse max (tr/min)	10
% erreur max	27,50 %
Valeur moyenne (tr/min)	76,1283185840708
% erreur moyenne	4,84 %

BF, correcteur P

Consigne vitesse moteur (tr/min)	80
Fréquence de rotation moteur (Hz)	1,33333333333333
Fréquence perturbations (Hz)	difficile à identifier
Différence consigne-vitesse min (tr/min)	13
Différence consigne-vitesse max (tr/min)	15
% erreur max	18,75 %
Valeur moyenne (tr/min)	79,4690265486726
% erreur moyenne	0,66 %

BF, correcteur PI

Consigne vitesse moteur (tr/min)	80
Fréquence de rotation moteur (Hz)	1,33333333333333
Fréquence perturbations (Hz)	difficile à identifier
Différence consigne-vitesse min (tr/min)	19
Différence consigne-vitesse max (tr/min)	10
% erreur max	23,75 %
Valeur moyenne (tr/min)	80,1283185840708
% erreur moyenne	0,16 %

BF, correcteur PID

Nous voyons que l'intégration d'un PID réduit considérablement l'erreur moyenne de la consigne et l'amplitude des perturbations. Nous pouvons maintenant avoir des valeurs plus faibles en consigne, les limites sont maintenant : 15 tr/min et 150 tr/min.

Nous utilisons également 2 valeurs de correcteur, à cause de l'erreur statique qui change :

- 35 à 120 tr/min : kp=20, ki=0,001 et kd=10
- 120 à 150 tr/min: kp=20, ki=0,009 et kd=30

II.7. Modification de l'IHM

La modification du programme arduino entraîne forcément celui de l'IHM. L'ancien IHM envoyait, lorsque l'on demandait une simulation, des variables pas forcément nécessaires au fonctionnement du programme. Nous avons donc modifié ce dernier, toujours dans un but

d’optimisation. Nous avons également supprimé le mode auto, enlevé ou remplacé les variables envoyées par l’ihm et l’avons converti en exécutable.

II.8. Bilan comparatif avec les objectifs initiaux

Nous avons entrepris des tâches supplémentaires, qui n'étaient pas initialement prévues, notamment :

- Effectuer des tests sur les composants des cartes Arduino.
- Organiser et trier le matériel fourni.
- Consacrer du temps à la compréhension des notices de code.
- Tenter d'identifier la source du court-circuit.

Nous avons finalement refait le code arduino intégralement, le système est désormais fonctionnel. Les perturbations sont toujours présentes lorsque le moteur tourne vite mais il est impossible de les supprimer, elles sont réduites grâce au correcteur PID.

Il y a quelques pistes d’amélioration dans notre code arduino, notamment pour la gestion du temps. Pour effectuer les mesures et la correction à intervalle régulier, il serait plus intéressant d’utiliser des fonctions d’interruption ce qui permettrait d’uniquement faire les calculs et gérer l’envoie des données dans la boucle.

Nous avons remarqué que la mesure de position pouvait prendre beaucoup de temps, l’idéal serait de trouver une fréquence d’échantillonnage pour que cette dernière prenne le moins de temps possible.

III. CAO du système

100%

III.1. Objectif fixés

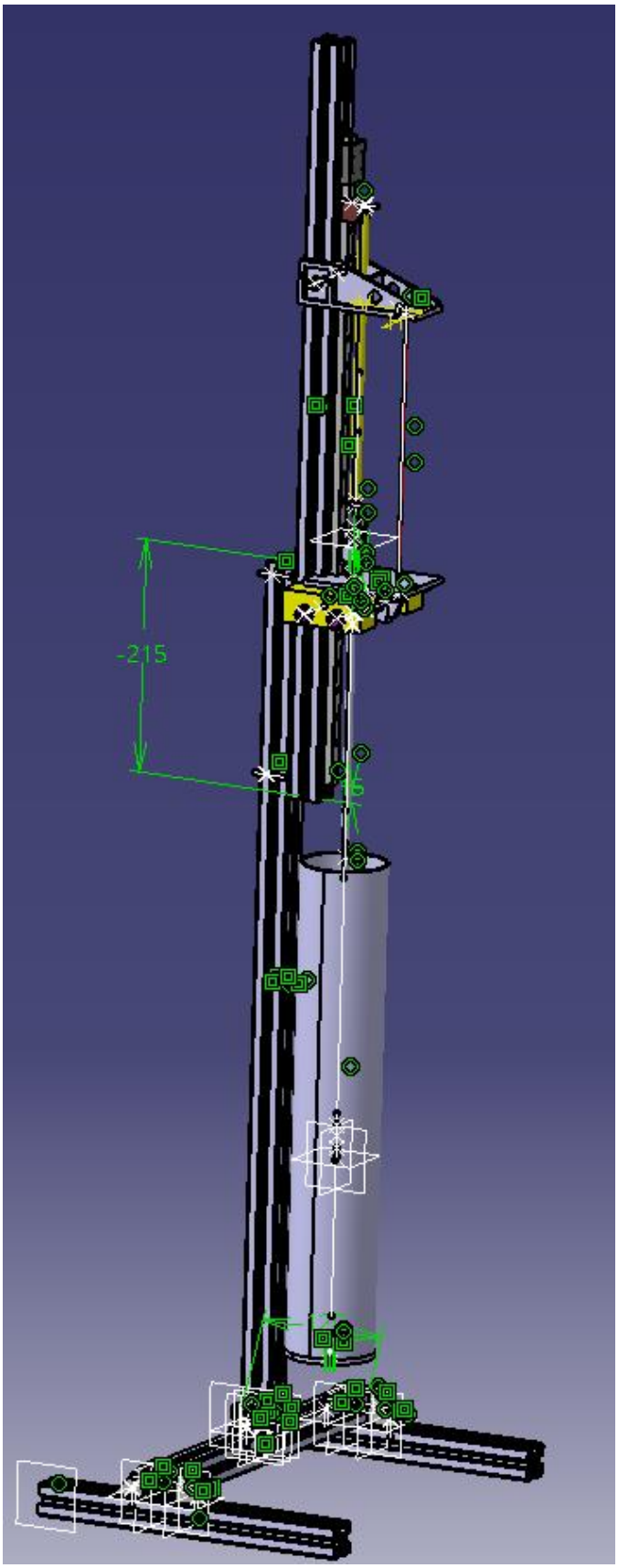
L’objectif pendant ce semestre est de mettre en place une CAO complète et fidèle, ainsi que des plans associés afin que les futurs élèves de la ZG2 (Zone Généraliste 2) puissent s’appuyer dessus pour construire et rendre opérationnel le système.
Il faut donc :

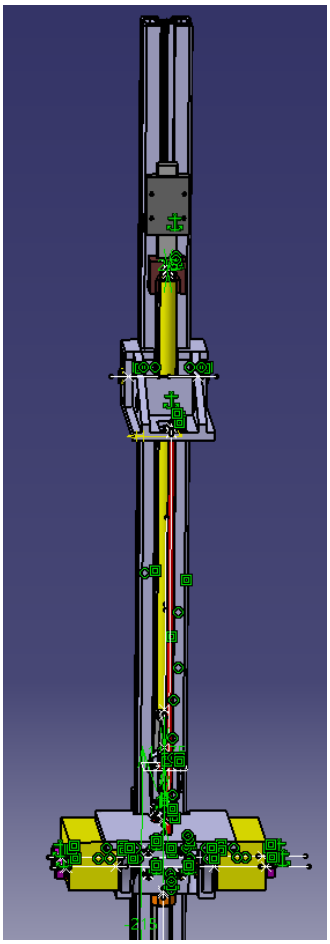
- ➔ Vérifier l’avancement qui a été fait durant les semestres précédents.
- ➔ Créer les pièces manquantes
- ➔ Rechercher dans les bases de données les composants achetés
- ➔ Modifié/adapté les pièces non conformes à notre modèle

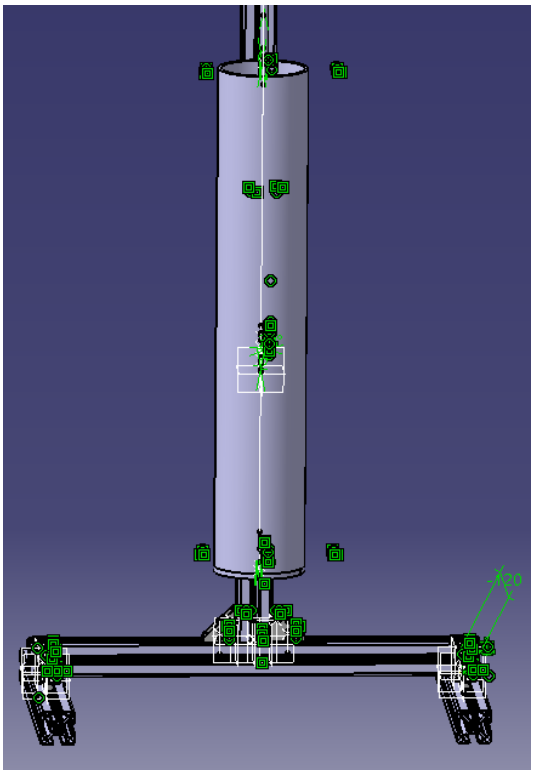
- ➔ Faire la mise en plan du système (système global et potentiellement des sous-ensembles)
- ➔ Faire du cable management afin de rendre le tout plus présentable

III.2. État initial de la tâche

Il y avait pas mal de choses dans les dossiers qui nous ont été fournis. Toute la base basse du système (structure, cylindre, ressort, lidar…). Il existait également dans des fichiers à part, tout l'assemblage de l'excitation ainsi que l'arrêt d'urgence et son support.


215

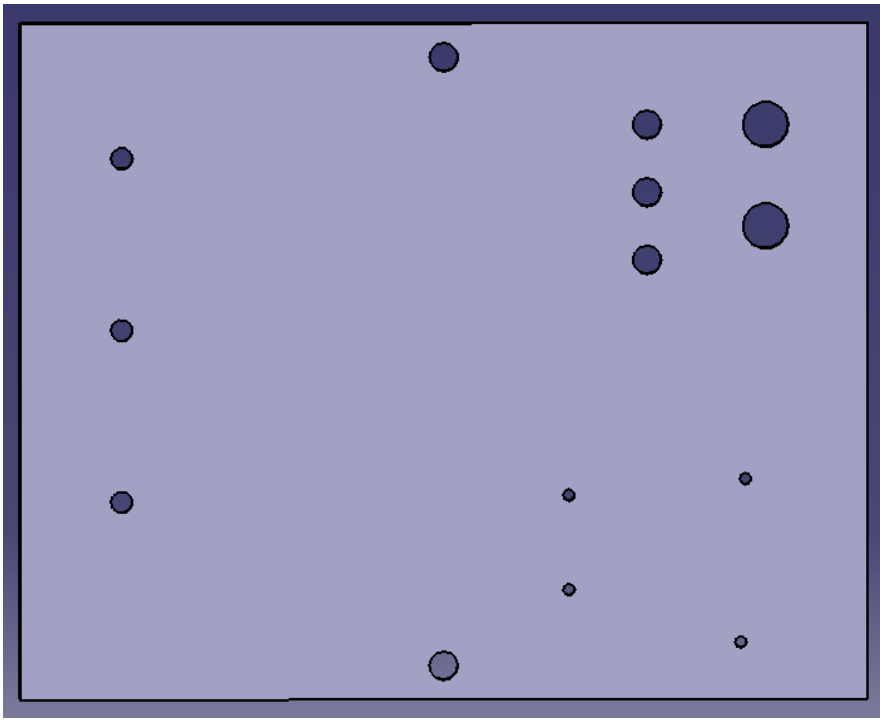




III.3. Ajout de la partie Électronique

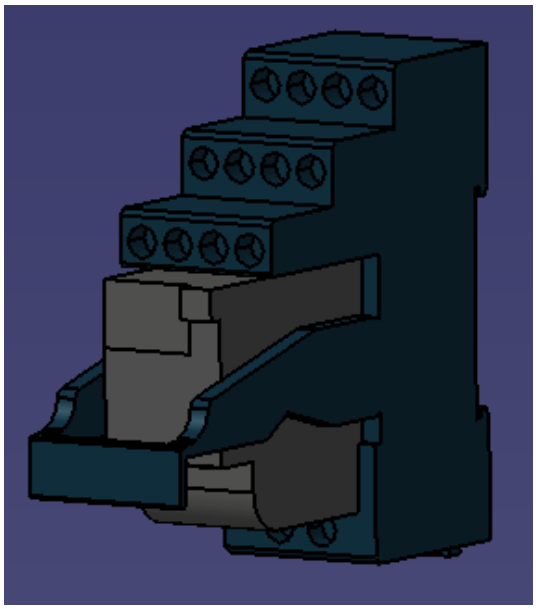
III.3.a. La plaque

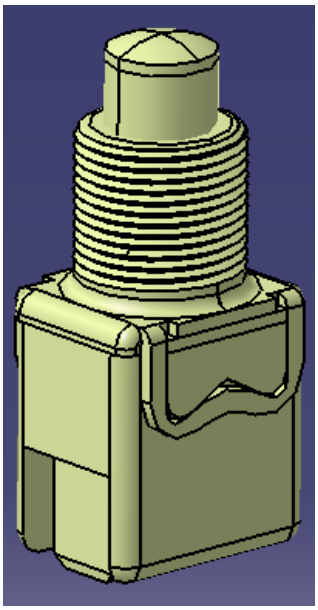
Les plans et la CAO de la plaque qui contient tous les éléments électroniques du système n'étaient pas dans les fichiers qui nous ont été transmis. Nous avons dû la refaire en prenant les mesures de celle faite au semestre précédent (S9P 2023).
La plaque fait 250 mm de longueur, 200 mm de largeur et 5 mm d'épaisseur.
Elle possède 3 trous sur sa gauche qui permettent l'accroche du rail (qui lui-même sera le support des borniers et du relais). Les 2 trous du milieu (en haut et en bas), permettent la fixation de la plaque sur le support métallique du système. Les 5 trous en haut à gauche accueilleront : Les voyants, le branchement de l'alimentation, les fusibles et le bouton poussoir qui permet l'allumage du système. Pour finir les 4 trous en bas à gauche serviront à viser la carte Arduino sur la plaque.

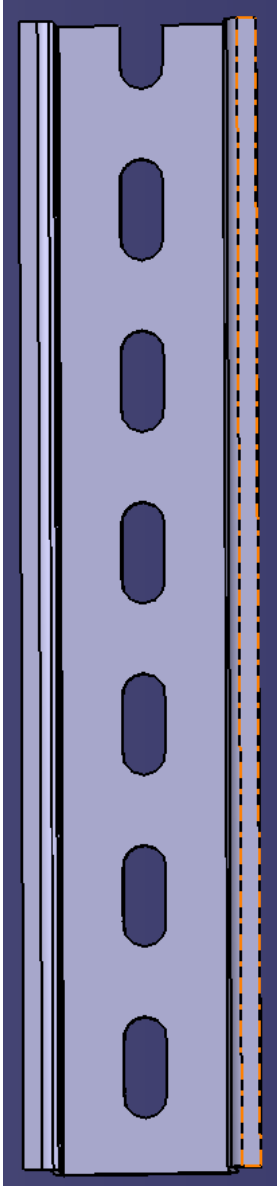


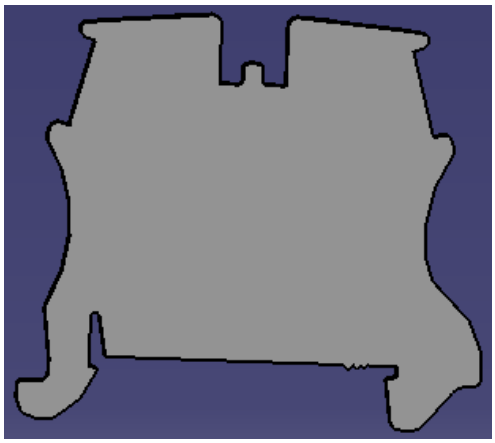
III.3.b. Les éléments mis en place sur la plaque

La plupart des éléments électronique mis en place sur la plaque sont des éléments standard, trouvable en magasin et donc en CAO dans les bases de données. Nous avons pris les références de toutes ces pièces afin de les chercher dans le logiciel "Tracepart". Nous en avons trouvé la moitié : le rail, les borniers, le relais et le bouton poussoir.

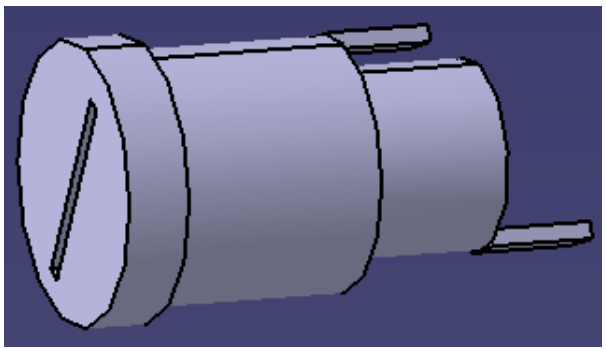


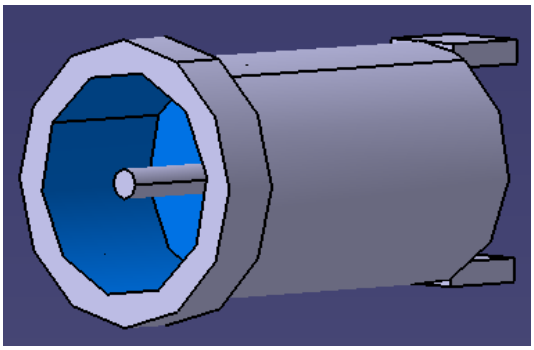


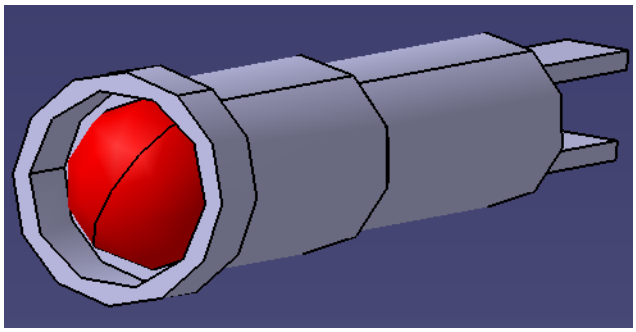


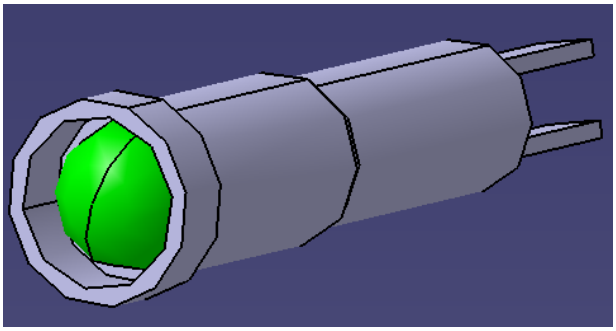


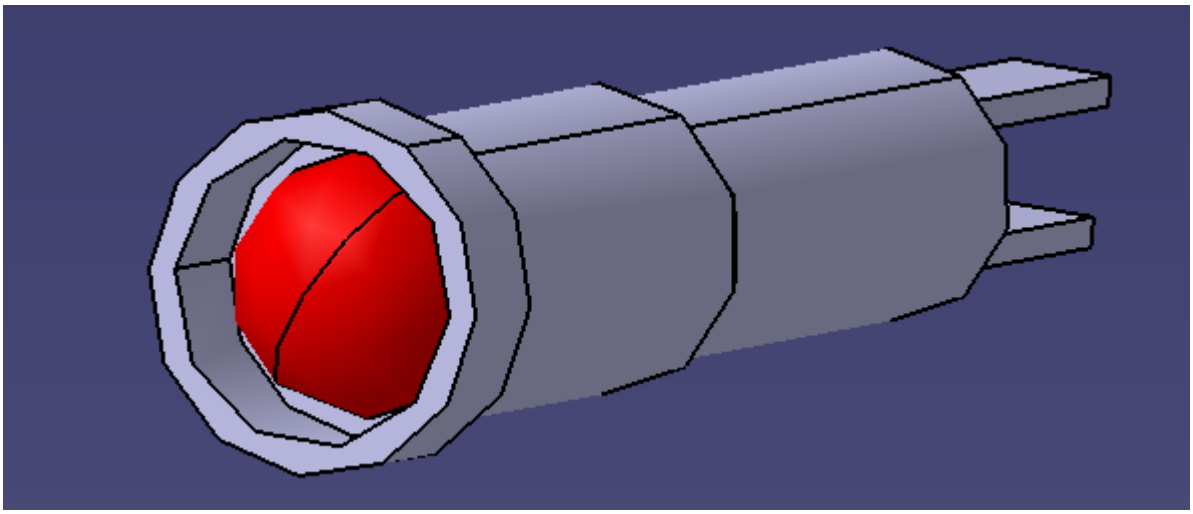
Pour les voyants, l'alimentation et les fusibles, nous ne les avons pas trouvés, nous les avons donc recréé à partir des côtes de ceux existant et que nous avions en notre possession.

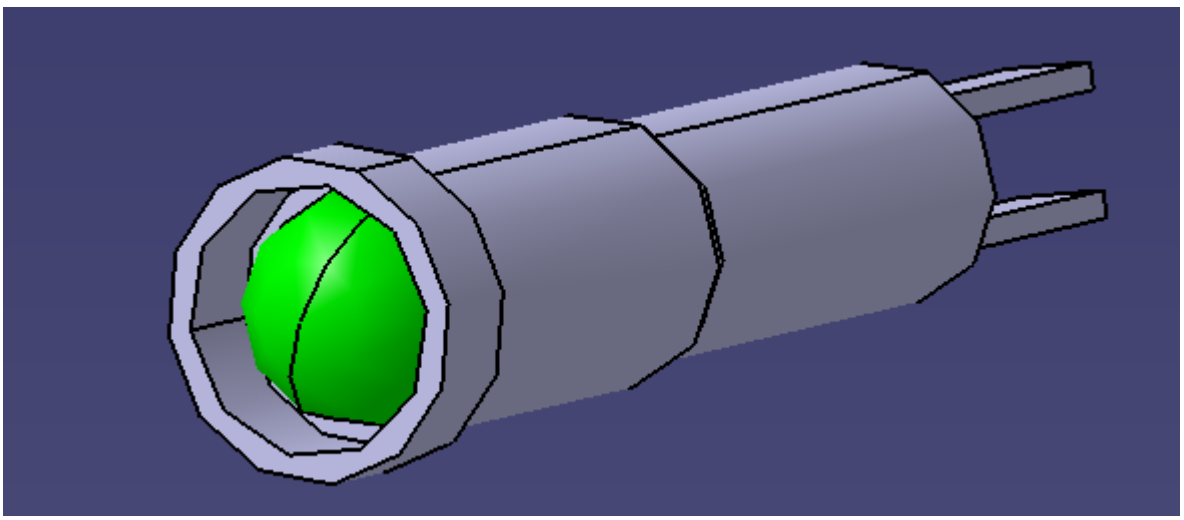




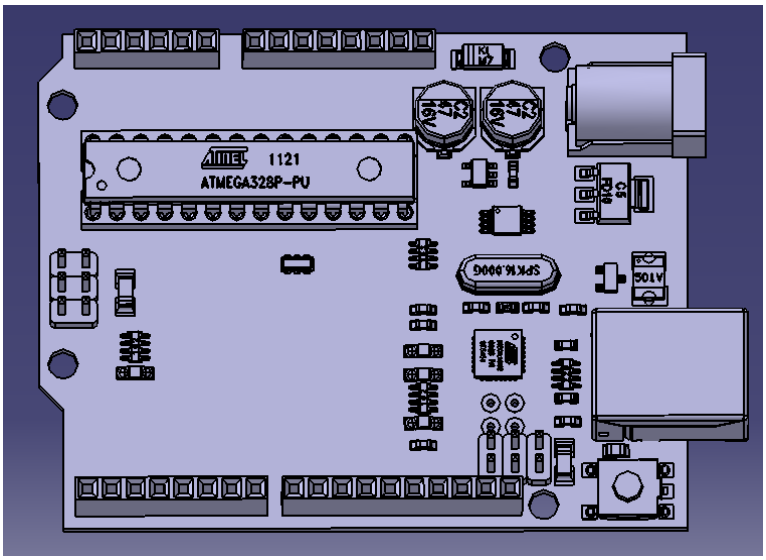




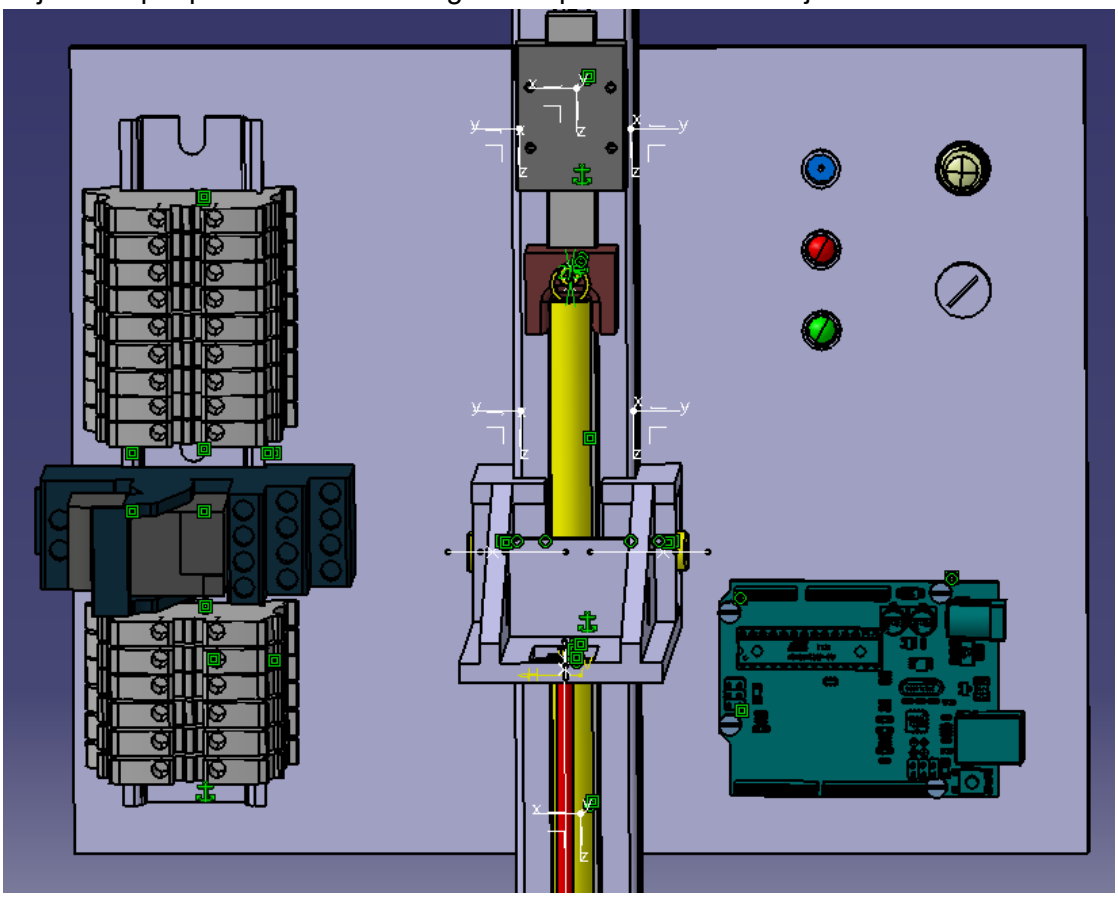




La carte Arduino à quant à elle était trouvée sur le site “GrabCAD”.

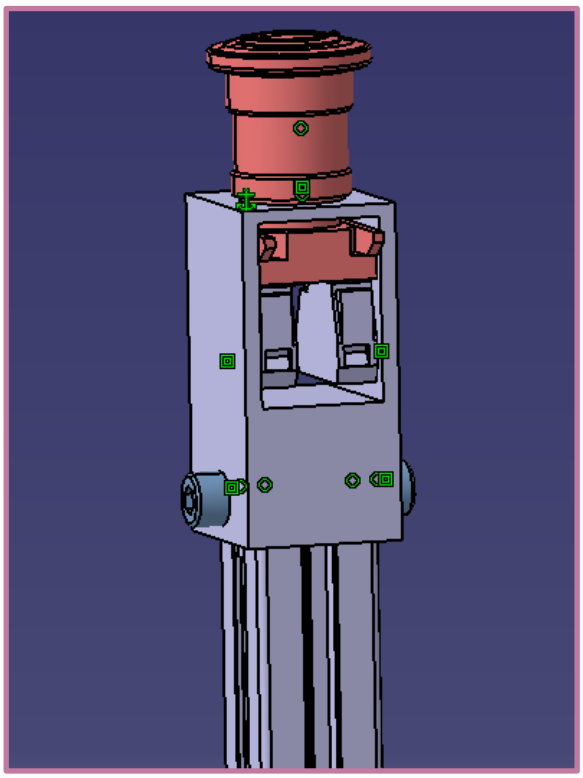


Finalement, nous avons réalisé l’assemblage de tous ces éléments sur la plaque, puis nous avons ajouté la plaque dans l’assemblage final que nous avions déjà.



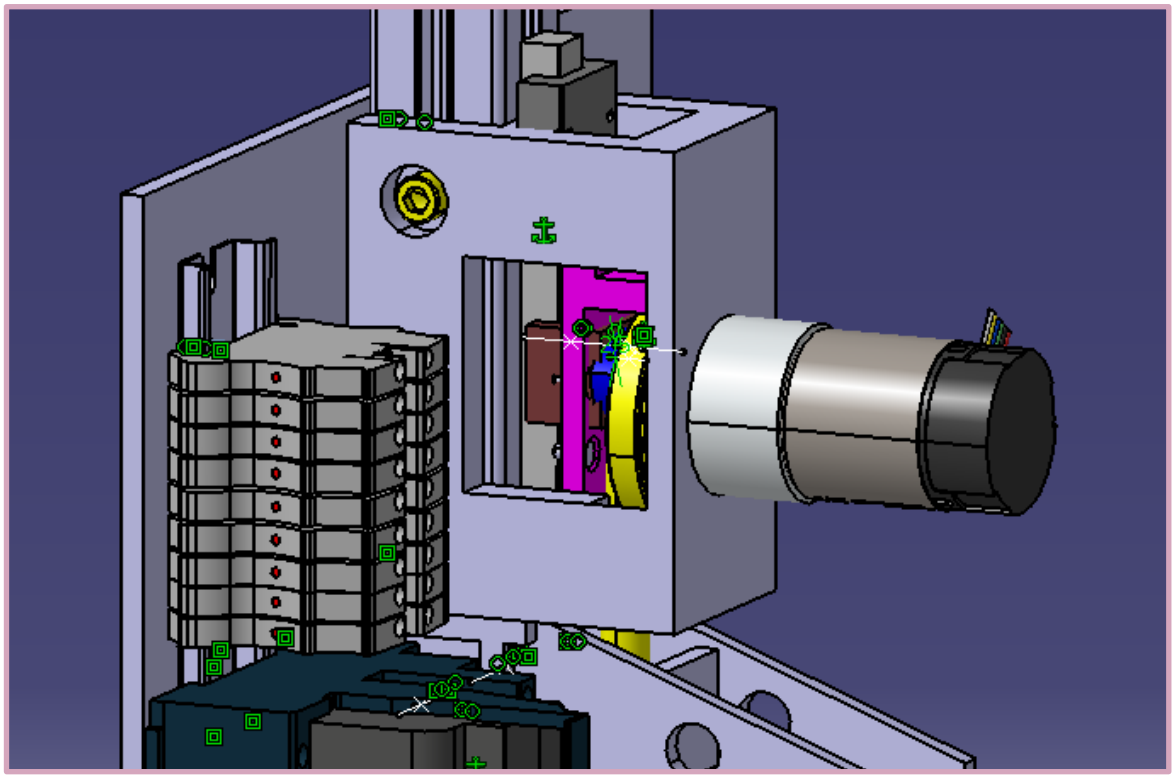
III.3.c. L’arrêt d’urgence et son support

Concernant l'arrêt d’urgence, les groupes précédents avaient déjà créé le support qui vient se fixer sur le dessus de l’ensemble.
Nous avons cherché le bouton d'arrêt d’urgence sur Trace Part, puis nous l'avons ajouté sur le support tel qu’il est sur le système existant.
Et nous avons placé le tout sur le système en 3D.



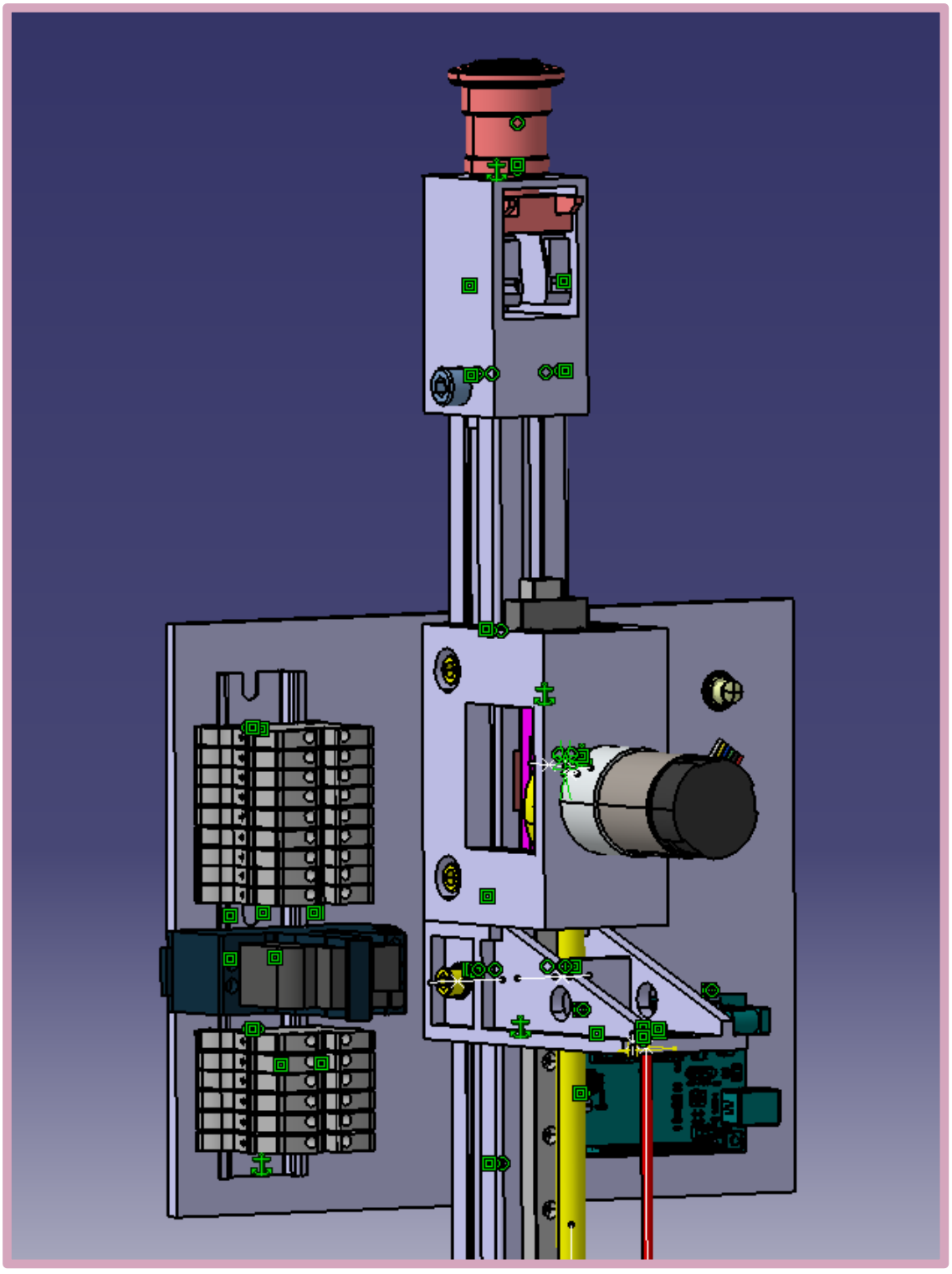
III.4.d. L’excitation

Pour l’excitation tout était déjà fait, nous avons juste eu à l’ajouter dans la CAO final.



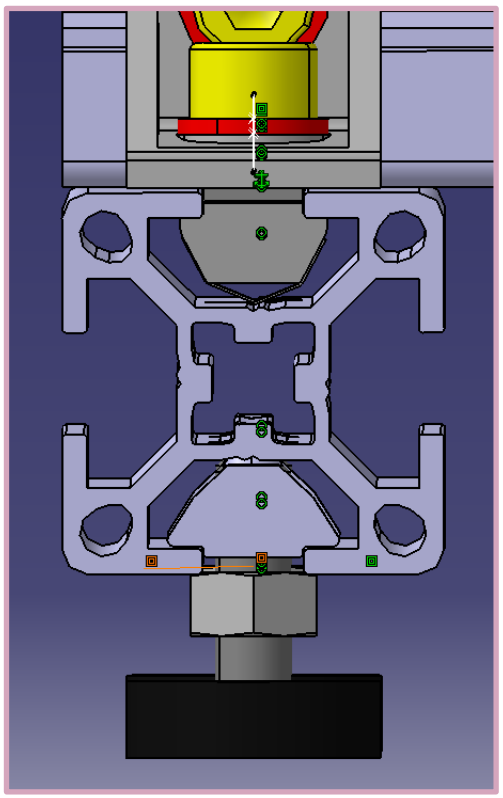
III.4.e. CAO Final

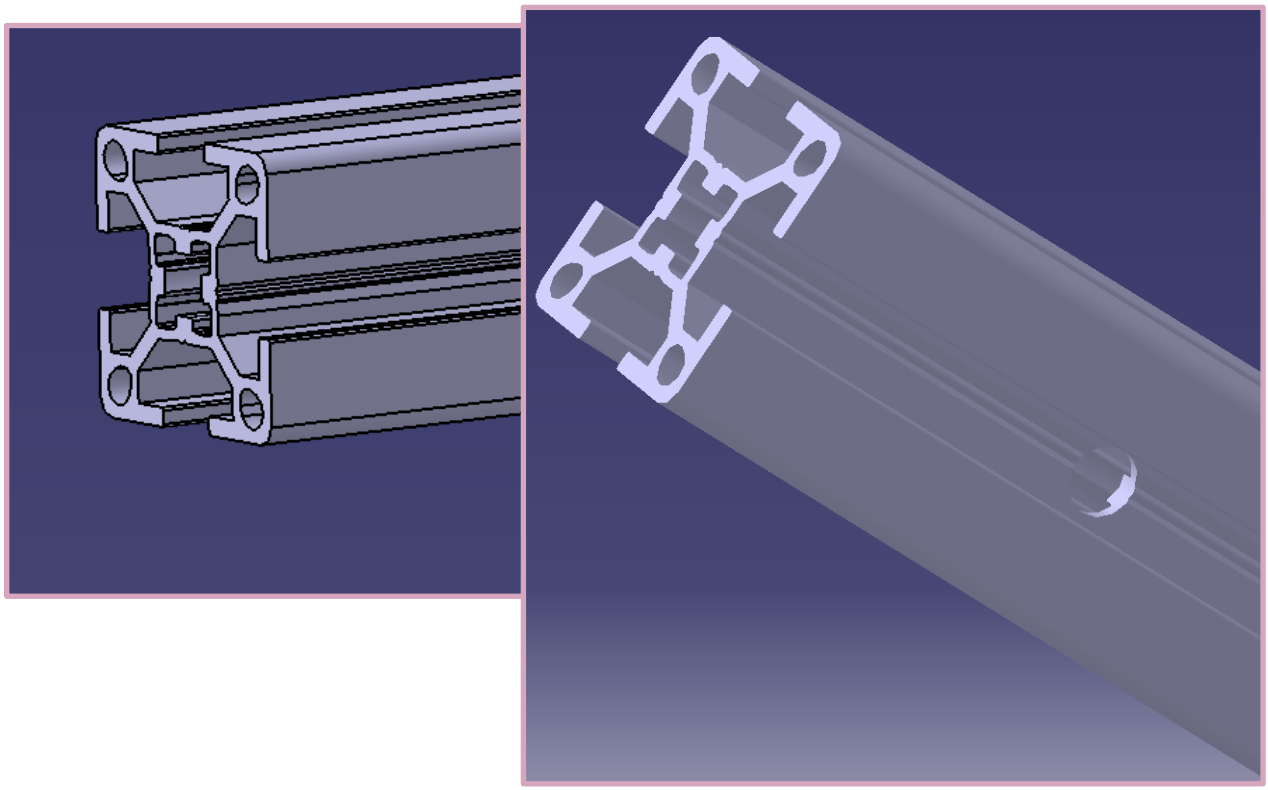
Voici donc L'assemblage final avec nos ajouts. Sachant que nous n'avons pas touché à la partie basse de la maquette 3D.



III.4.f. Ajout des pieds

Nous avons également ajouté les pieds sur la CAO, nous les avons trouvés sur Trace Part mais avant de les ajouter nous avons créé des trous sur les barres, à 65 mm du bords, qui accueilleront les pieds, afin de faciliter leurs mises en places et de permettre un meilleur réglage pour l'équilibre du système. Toujours dans cet objectif, nous avons mis en place, sur chaque pied, un contre écrou permettant de mieux fixer les pieds avec l'écrou barre.





III.5. Nomenclature

Ma dernière mission concerne la nomenclature, je l'ai refaite entièrement avec les nouvelles pièces ajoutées en prenant pour référence une nomenclature existante - mais non complète - afin de conserver les mêmes numérotations que sur les précédents bon de commande ou autre document de références. Nos nouvelles pièces sont donc numérotées à partir de 39.
De plus, j'ai également changé les noms de chaque pièce pour que le nom du fichier, le nom de la pièce dans l'assemblage et le nom dans la nomenclature soit indique pour faciliter la compréhension pour les semestres suivants.

	A	B	C	D	E	F	G	H	I	J
3	0a	1	Barre_800				18	15		22
4	0b	1	Barre_700				18	13	2 barres prises pour glissière motorisée	22
5	0c	1	Barre_400				18	14		22
6	0d	2	Barre_300				36	28		44
7	1	1	tube				18	14	2 tubes avec bouchon inf (à couper) en + 1 tube fissuré en +	22
8	2	1	couvercle_inf	collé au tube 1			18	14		22
9	3	1	couvercle_sup				18	17		22
10	4	1	tige filetée M8_INOX				18	14		22
13	5a_o	1	piston_a_orange	faible amortissement_1trait			18	26		22
14	5b_o	1	piston_b_orange	faible amortissement			18	28		22
15	6	1	chape_SG-M8				18	14		22
16	7	1	ressort	100N/m			18	35	1 sur maquette ED	22
17	8	1	support_masse				18	23		22
19	10	2	masse				36	29		44
20	11	10	rondelle_D8_INOX				180	25		220
21	12	8	ecrou_HM8_INOX				144	34	7 sur maquette ED	176
22	13	1	maillon_D4				18	14		22
23	14	7	rondelle_D3_INOX	moyenne ou large			126	43		154
24	15	4	vis chc_M3-20	support masse sur patin			72	44	4 sur maquette ED	88

Nomenclature de référence

	A	B	C	D	E	F	G	H
1	Assemblage	Sous assemblage	Numéro de l	nom de la pièce	quantité	présente	Ok sur CATIA	Ok sur fichier
2	Assemblage bati		0c	barre_ 400	1		☑	☑
3			0d	barre_ 300	2		☑	☑
4			0b	barre_ 700	1		☑	☑
5			0a	barre_ 800	1		☑	☑
6			28	rail glissière	1		☑	☑
7			39	Vis CHC M6x20	1		☑	☑
8		Assemblage vis CHC M6x38	24	Vis CHC M6x38	2		☑	☑
9			20	Rondelle D6	2		☑	☑
10			33	Ecrou M6 barre	3		☑	☑
11		Assemblage Equerre	32	Equerre	6		☑	☑
12			23	Vis CHC M6x12	12		☑	☑
13			33	Ecrou M6 barre	12		☑	☑
14			20	Rondelle D6	12		☑	☑
15		Assemblage pied	21	Ecrou HM6	4		☑	☑
16			33	Ecrou M6 barre	4		☑	☑
17			38	Pied prfm 20 M6x17	4		☑	☑
18	Assemblage Amortisseur		1	Tube	1		☐	☐
19			2	Couvercle inf	1		☐	☐

Nouvelle nomenclature

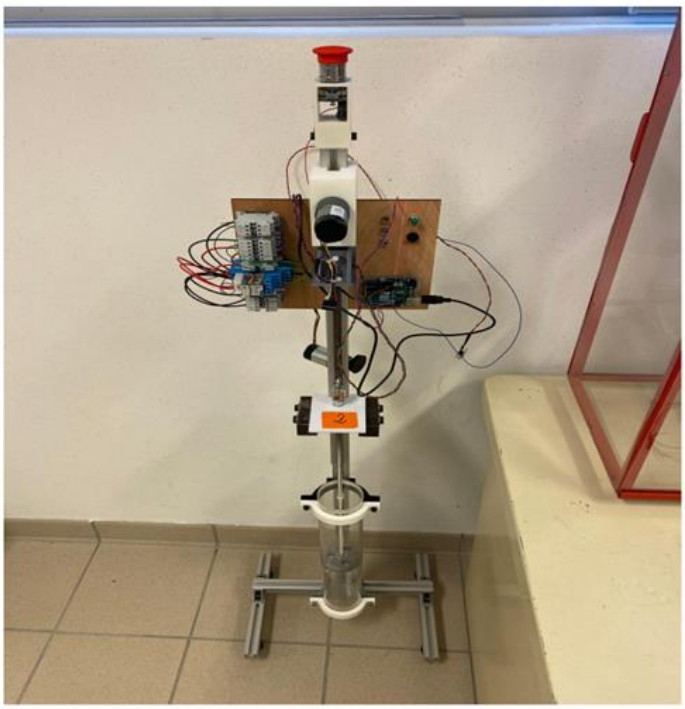
IV. Cable management et amélioration de la maquette

65% 35%

IV.1. Objectifs fixés

L'objectif principal de cette mission est d'optimiser le câblage et l'aspect global du système afin de le rendre visuellement attrayant pour l'article de M. DELALEAU, et également de le rendre reproductible pour les élèves qui vont travailler sur le projet ZG2.

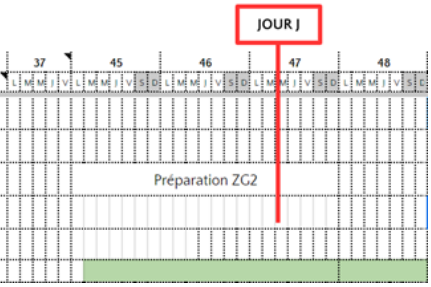
IV.2. Etat initial de la tâche



La première version prototype élaborée par les étudiants du dernier semestre présentait un agencement peu ordonné des câbles, lesquels traversaient les composants tels que les cartes, le moteur et le lidar. Notre objectif est d'optimiser la gestion des câbles en ajoutant des goulottes pour les guider à travers, améliorant ainsi l'esthétique générale de la maquette.

IV.3. Planification

ZG2			COEFFICIENT	JAMBOU Clémence	QJAOUT EL Challa	THAUVEL Madeline	ROUSSIGNOL Louis	HARISTOY Clothilde	
	NOM DE TACHE	Description détaillée							%
T3.3	Bilan de la maquette	durée de vie très important	2	0%	0%	0%	0%	0%	0%
T3.4	devis et chiffrage de la duplication du système d'excitation		1	0%	80%	0%	0%	0%	80%
T3.5	Préparation ZG2	cahier des charges?	1	0%	0%	0%	0%	0%	0%
T3.6	commande du matériel	remblissage BDC/ Est-ce que la partie excitation marche	1	0%	0%	0%	0%	0%	0%
T3.7	conception nouvelle pièce de fixation tube	CAO résolution de la tache/ les tubes vont être	0,5	0%	0%	0%	0%	0%	0%
	Cable management	Réalisation de la nouvelle maquette			60%	40%			



IV.4. Choix techniques

Afin de rendre le modèle plus propre visuellement, nous avons décidé de faire du câble management. Pour cela nous avons redessiné la plaque qui accueille toute la partie électronique la rendant plus grande afin de pouvoir accrocher des goulottes Legrand (ref : 030804) pour cacher les câbles.

La plaque mesure maintenant 314 mm de longueur et 264 mm de largeur mais conserve 5 mm d'épaisseur.

Nous avons découpé la plaque grâce à la découpeuse laser mise en place à la forge et nous avons fait le changement sur la maquette. Cependant nous avons oublié de graver à quoi sert chaque trou mais nous avons ajouté c'est écriture sur le modèle 3D, mais sur la maquette nous avons fait grâce à des post-it.

Mais une nouvelle plaque va être réalisée (et découpée) afin d'accueillir des goulottes plus industrielles (nous avions fait avec les goulottes électriques disponibles à l'enib mais pas pratique car on devait faire des trous pour passer les câbles). Il faudra donc faire le changement au semestre prochain et commander des goulottes industrielles.

Figure 1 : goulottes à utiliser pour la duplication

Figure 2 : goulottes disponibles au magasin ENIB

IV.5. Bilan comparatif avec les objectifs initiaux

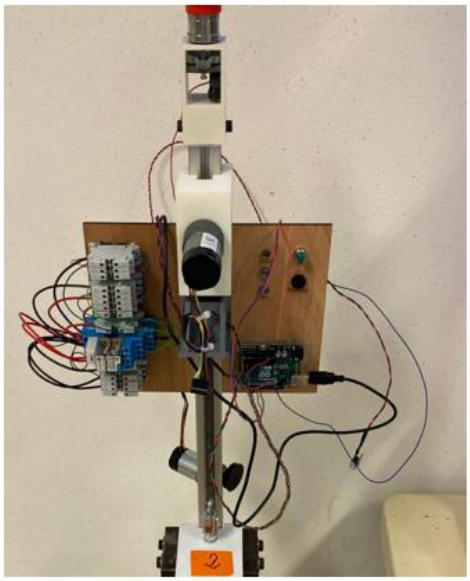


Figure 4 : Maquette au Début

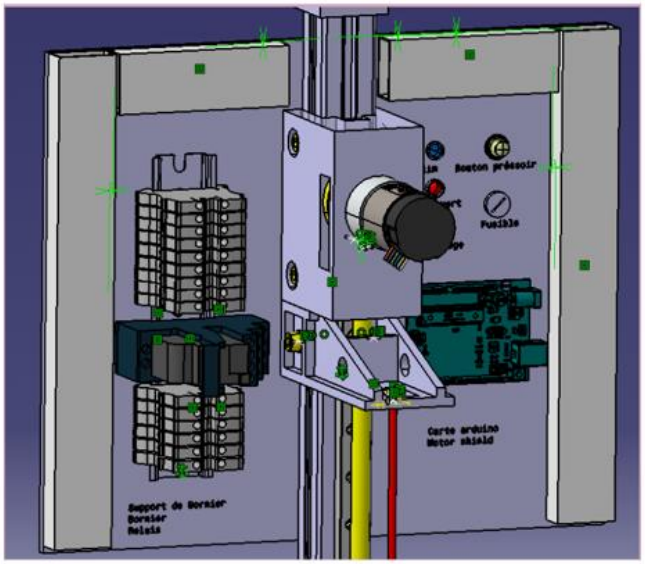


Figure 3 : Objectif Maquette finale

Après la réalisation de la nouvelle plaque, il fallait refaire le câblage de la sécurité, mais cette étape parait un peu compliquée à cause des soucis sur le relais et les borniers.

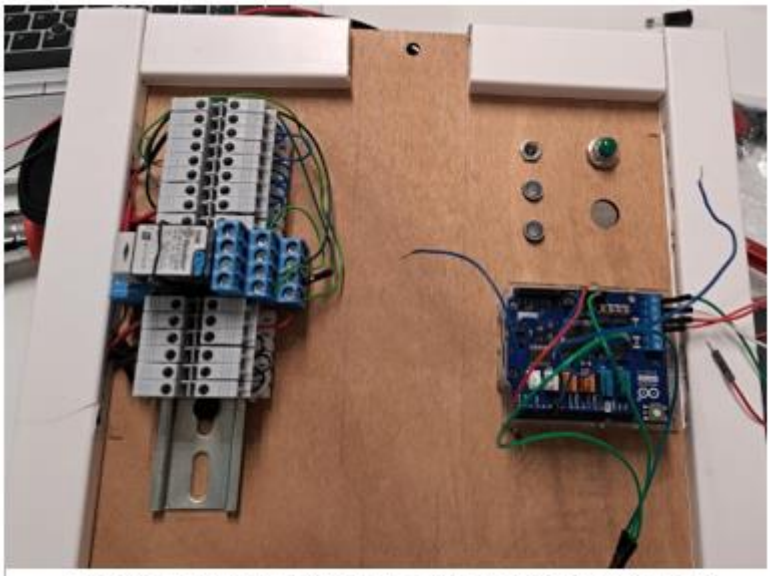


Figure 5 : Maquette en cours de finalisation

V. Etude théorique du comportement vibratoire

V.1.Objectifs fixés

Cette partie a pour but la simulation du système en Python, en exprimant les équations différentielles du mouvement. Afin d'obtenir les données théoriques du système et les comparer avec les données expérimentales dans la phase du post-traitement.

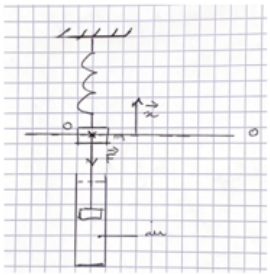
V.2. Etat initial de la tâche

Afin de bien saisir les instructions données, j'ai initié le processus en effectuant une lecture approfondie du rapport pluridisciplinaire créé par les étudiants du dernier semestre. Mon objectif était de comprendre leur étude et les méthodes qu'ils avaient utilisées. Cependant, la consultation du travail accompli par d'autres personnes peut parfois sembler abstraite en l'absence d'explications et de justifications claires. Pour remédier à cela, j'ai entrepris de recalculer les valeurs en me basant sur les informations fournies dans les documents du cours. Cependant, cette démarche s'est avérée complexe en raison de l'absence du coefficient de frottement nécessaire pour calculer le discriminant de l'équation caractéristique et ainsi identifier les solutions.

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

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

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

$$\delta < 0$$

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

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

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

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

V.3. Choix techniques

Une fois les calculs seront résolus et les éléments nécessaires à la modélisation mathématique du système identifiés, je vais recourir à l'outil Jupyter Notebook pour rédiger le code Python destiné à la simulation.
En suivant la méthodologie élaborée dans le document fourni par M. DELALEAU, j’ai réussi à obtenir le même résultat pour le cas des oscillations libres .
L’équation obtenue est $$m\ddot{x} + \lambda\dot{x} + kx = ke$$

Le coefficient de frottement reste inconnu.

V.4 Bilan comparatif avec les objectifs initiaux

Étant donné la quantité de temps que cette tâche m'a demandé sans que je puisse observer de progrès significatif, j'ai décidé de me consacrer en parallèle à l'organisation des câbles, afin de faire avancer la réalisation des tâches assignées pour ce projet.