2017-2018

1ère année du cycle ingénieur

Département : SAGI

Semestre 8

DEVELOPPEMENT D'UN BANC DE TEST POUR DRONE LOURD

NEVA AEROSPACE

QUENTIN CHOUTEAU
LEO COURSON
JAUNEAU VINCENT

Remerciements

En préambule à la rédaction de ce rapport de projet tuteuré « Banc de test pour drone lourd », nous souhaitons adresser nos plus sincère remerciements aux personnes qui nous ont apporté leur aide et qui ont contribué à la réussite de ce projet.

Nous tenons à remercier Monsieur Franck Mercier, notre tuteur, qui s'est toujours montré très disponible et à l'écoute tout au long de ce projet, ainsi que son aide et ses conseils lors de l'étude et le développement de notre solution.

Nous remercions également Monsieur Renaud-Pierre Blanpain, *Head of Engineering & Prototyping de Neva Aerospace*, commanditaire du projet, qui a su nous apporter les précisions dont nous avions besoin à la bonne réalisation de notre banc de test.

Nous remercions toute l'équipe de l'ISTIA pour leur aide au cours du projet ainsi que de l'IUT pour leur disponibilité.

Sommaire

Sommaire

Introduction

Les drones sont des outils modernes aux possibilités et capacités quasi illimitées. Leur développement rapide et leur omniprésence professionnelle, personnelle et médiatique en témoignent. Leur intérêt premier repose sur leur fonction de vecteur, c'est-à-dire leur capacité à soulever divers capteurs dans les airs, permettant des prises de mesures et des enregistrements. Actuellement, les systèmes les plus fréquemment embarqués sont les caméras et autres appareils photos aux dimensions compactes. Cependant le potentiel des drones est bien plus vaste. Dans un futur proche, en plus d'embarquer de multiples caméras certains seront en capacité de soulever des charges lourdes. Cela pourrait permettre de faciliter les transports de marchandises ou encore les interventions d'urgences délicates dans des zones à risques.

Neva Aerospace travaille depuis plusieurs années sur la conception de drone lourd. Ceux-ci pourraient être capable de déplacer une masse proche de la tonne. Pour atteindre cet objectif les turbines puissantes qui les composent devront fonctionner de manière optimale. C'est ici que démarre notre projet en collaboration avec Neva Aerospace. Notre objectif sera de concevoir un banc de test pour turbines de drones lourds.

1 - Contexte

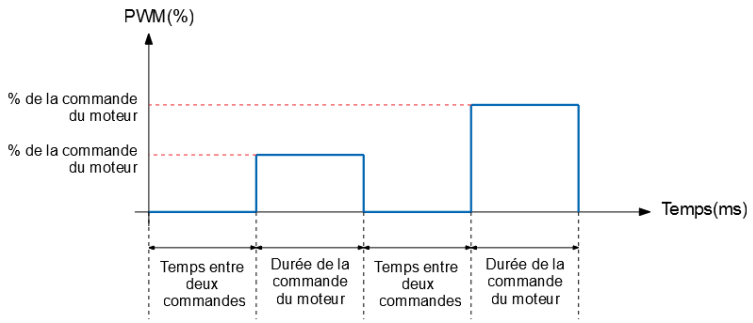
1.1 – Cahier des charges

Réaliser un banc de test, capable de mesurer la force de poussée, l'intensité et la tension d'une turbine de drone lourd. Ces mesures devront être affichées sur une interface graphique, simple d'utilisation et sauvegardées sous format texte ou Excel.

1.2 – Tests à réaliser

L'Arduino devra réaliser plusieurs tests en contrôlant les moteurs de la turbine de différentes manières afin de la caractériser le plus précisément possible.

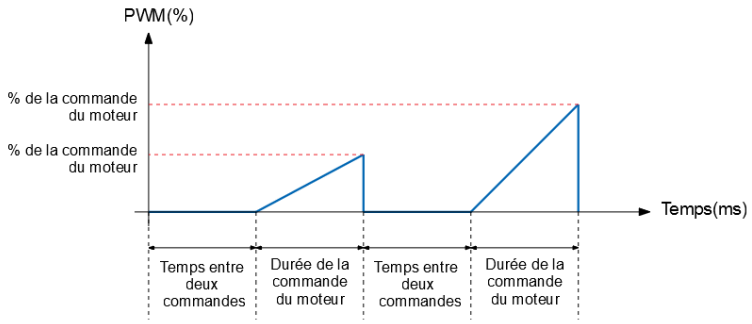
1.2.1 – Tests en échelon



Test en échelon

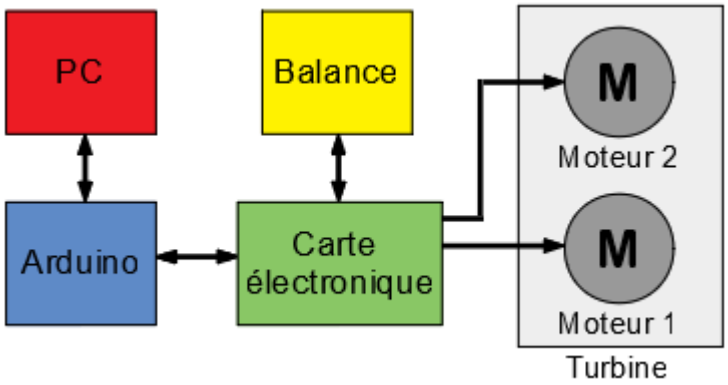
Lors de ce test, l'Arduino vient commander les moteurs de la turbine sous forme d'échelon le but étant d'analyser le comportement des moteurs de la turbine pour certain PWM. A chaque commande on vient augmenter le PWM de 10% entre chaque commande. Le temps entre deux commandes permet au flux d'air généré par la commande précédente de se dissiper.

1.2.2 – Tests en rampe



Lors de ce test, l'Arduino vient commander les moteurs de la turbine sous forme de rampe afin d'analyser la monté en tour des moteurs de la turbine. A chaque commande on vient augmenter l'accélération de la commande des moteurs entre chaque commande (PWM final augmenté de 10%). Le temps entre deux commandes permet au flux d'air généré par la commande précédente de se dissiper.

1.3 – Architecture du projet



- Carte électronique : Sert d'interface entre l'Arduino et le monde physique. Il va récupérer les valeurs à analyser (intensité/tension aux bornes des moteurs) et va adapter les niveaux de tension pour la liaison RS232 avec la balance.

- Arduino : Va effectuer la récupération des données, la commande des moteurs et l'envoi des données récupérée au PC.

- PC : Sert principalement d'interface au banc de test. Va permettre d'envoyer une procédure de test à l'Arduino puis analyser et afficher les valeurs récupérées lors du test.

- Balance : Va renvoyer la force de poussé sous forme de poids à l'Arduino.

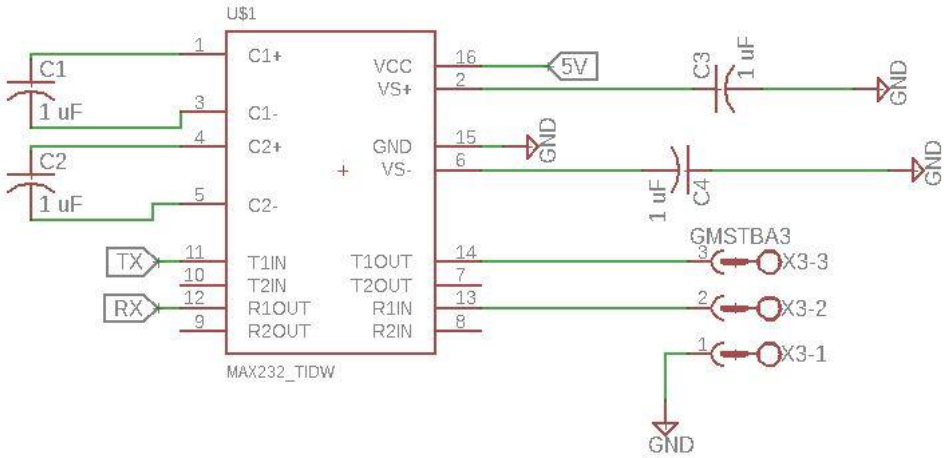
2 - Hardware

2.1 – Carte électronique

Pour établir la connexion entre l'Arduino et le PC il est nécessaire de concevoir une carte électronique. Celle-ci s'occupera de retransmettre les données des moteurs à l'Arduino. Celle-ci s'occupera de récupérer toutes les valeurs des moteurs.

2.1.1 – La force de poussée

Pour récupérer la force de poussée de la turbine nous utilisons une balance. Elle est connectée à notre carte électronique par un câble RS232. Il est donc nécessaire de prévoir un montage capable de convertir du RS232 en tension TTL (0-5V).
Pour ce faire nous utilisons un MAX232 qui est relié à notre connecteur DB9. Il est alimenté directement par l'Arduino en 5V.
Le Max 232 est un standard depuis longtemps, il permet de réaliser des liaisons RS232 et des interfaces de communications. Il amplifie et met en forme deux entrées et deux sorties TTL/MOS vers deux entrées et deux sorties RS232.

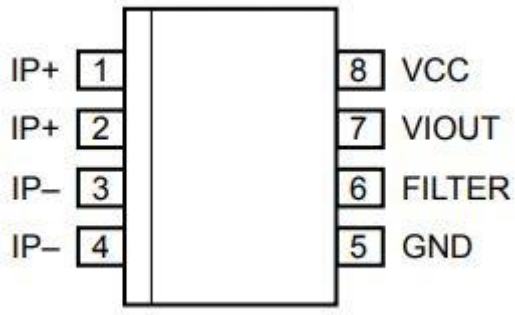


Connexion du MAX232

La communication avec la balance s'effectue directement au niveau des broches T1IN (14) et R1IN (13). Au travers des condensateur C1 et C3 il génère une tension de 10V à partir de sa tension d'alimentation (VCC = 5V). Sur le même principe, au travers des condensateurs polarisés C2 et C4 il génère une tension de -10V.
Les informations convertis sont retransmises à l'Arduino par les broches 11 (TX) et 12(RX).

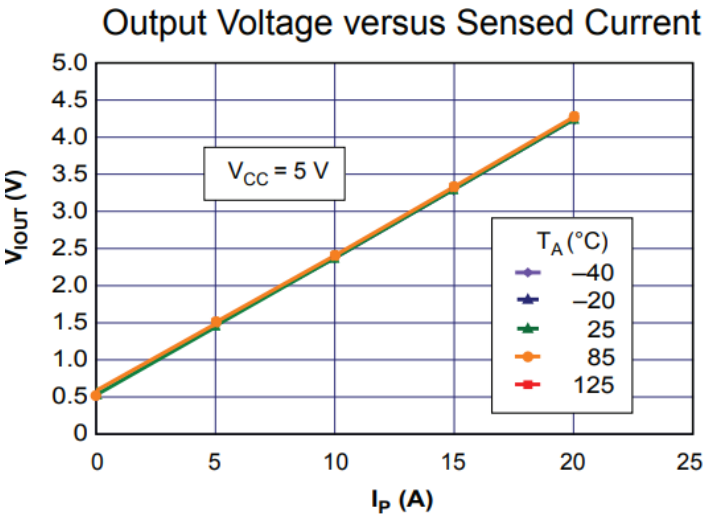
2.1.2 – L'intensité

Pour récupérer le courant d'alimentation des deux moteurs j'utilise une sonde de courant (ACS713).



Connexion de la sonde de courant

Le fonctionnement de ce type de composant est plutôt simple. Un courant circule entre les pins IP+ et IP-.
On obtient une image de ce courant, sous forme de tension, sur le pin 7 (VIOUT).
Ces sondes seront alimentées en 5V directement par le biais de l'Arduino.



Tension en sorti en fonction du courant en entrée

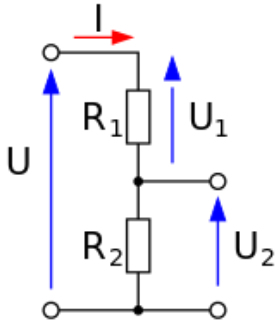
La courbe ci-contre représente la tension de sortie VIOUT en fonction de l'intensité du courant d'entrée IP pour un courant inférieur à 20A. Il est relativement simple de retrouver l'équation de cette courbe et de déterminer le calcul qui nous sera nécessaire pour obtenir la valeur du courant de chaque moteur.

On a $Viout = 0.1875\ IP + 0.5$

Donc $IP = \frac{Viout - 0.5}{0.1875}$

2.1.3 – La tension aux bornes des moteurs

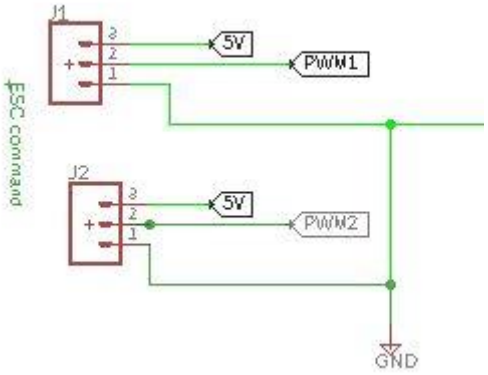
Le montage permettant de récupérer la tension d'alimentation est extrêmes simple puisqu'il s'agit d'un simple pont diviseur. Les moteurs fonctionnent en 6S, ce qui correspond à une tension d'alimentation de 25.2V.



La tension de sortie U2 = (R1/R1+R2). U1

Avec R1 = 300KΩ et R2 = 75KΩ on obtient une tension de sortie de 5.04V ce qui nous convient parfaitement. Cette tension sera lue directement sur une broche de l'Arduino.

2.1.4 – Commande et alimentation des moteurs

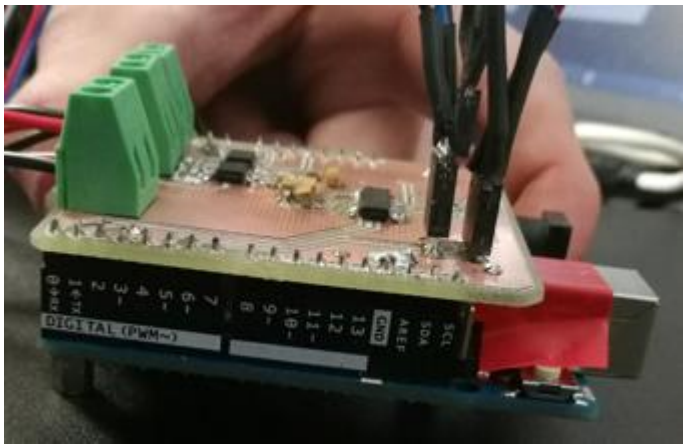


Borne d'envoi de la commande au moteur

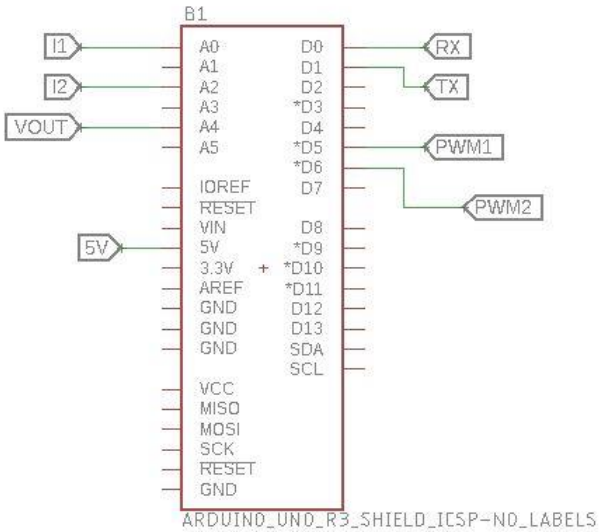
La commande des moteurs sera gérée via le PWM de l'Arduino et envoyée via les connecteurs J1 et J2 (connecteurs DIN 3pins).

En ce qui concerne la puissance, celle-ci sera délivrée par une alimentation stable +30V/-30V et renvoyée aux moteurs via des connecteurs DIN 4 pins.

2.1.5 – Connexion avec l'Arduino

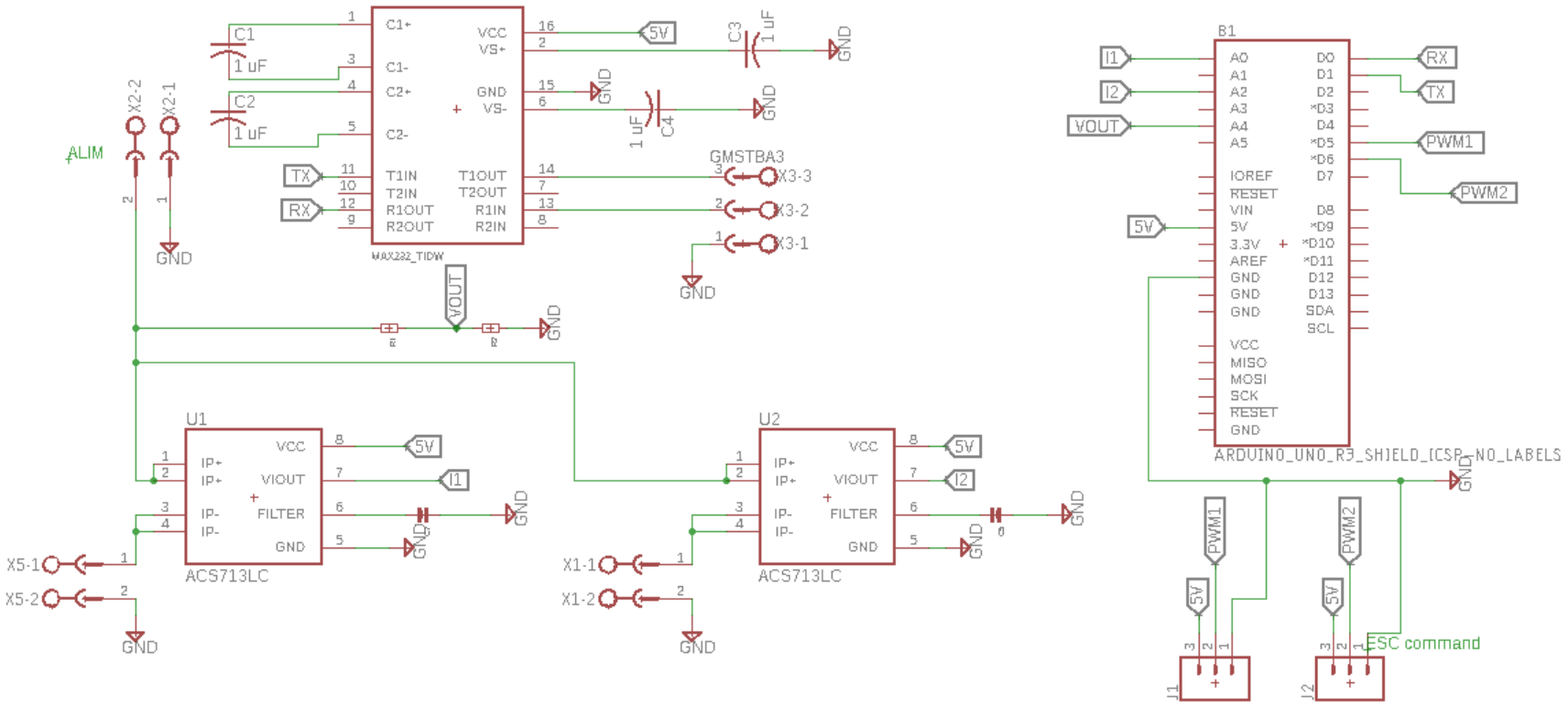


La connexion entre la carte électronique et l'Arduino se présente sous la forme de Shield. Les courants des moteurs sont récupérés sur les entrées analogique A0 et A2. En ce qui concerne la tension d'alimentation elle peut être lue sur l'entrée analogique A4. Les moteurs sont eux contrôlés par le biais des sorties PWM D5 et D6.

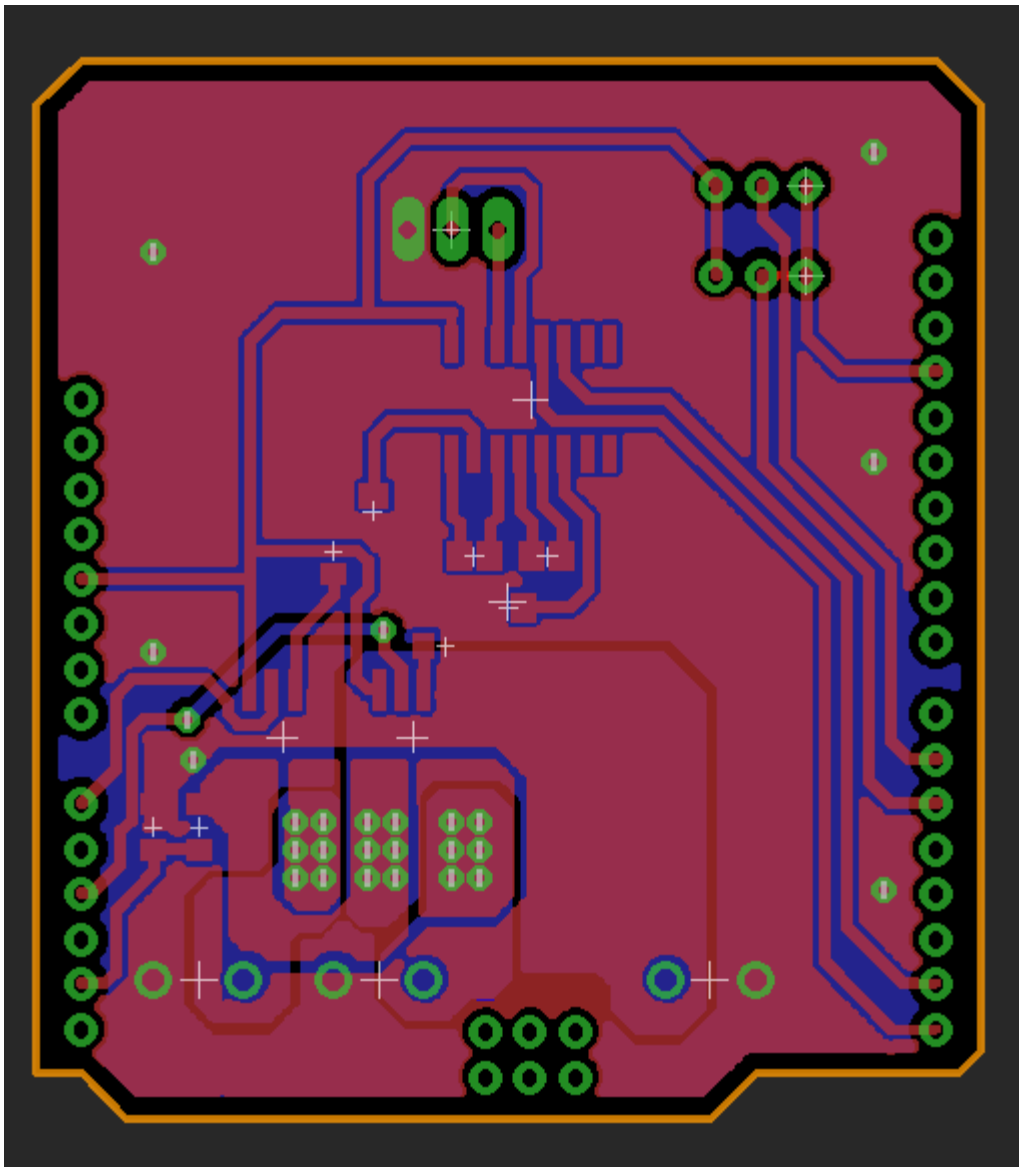


Connexion Shield/Arduino

2.1.6 – Schématique complète



2.1.7 – Carte routée



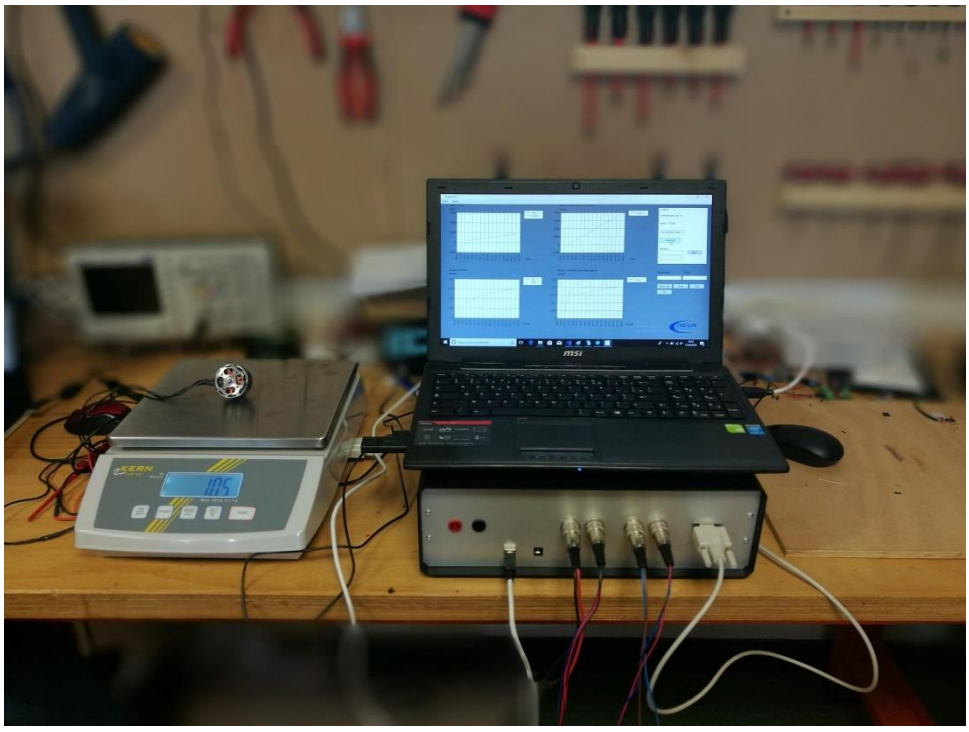
2.2 – Boîtier

Dans l'objectif de rendre un produit le plus professionnel possible nous avons customisé un boitier métallique avec plusieurs types de connecteur. Le boîtier contiendra l'Arduino, ainsi que des connecteurs industriels permettant de relier l'Arduino aux différents éléments du banc de test (moteurs, balance, PC…).

2.2.1 – Connectique utilisée

Rôle	Type de connecteur	Quantité	Image
Commande moteur	Connecteur DIN 3 pins	2 (1/moteur)	
Puissance moteur	Connecteur DIN 4 pins	2 (1/moteur)	
Connexion balance	Connecteur DB9	1	
Alimentation	Fiche Banane	2	

2.2.2 – Résultat final

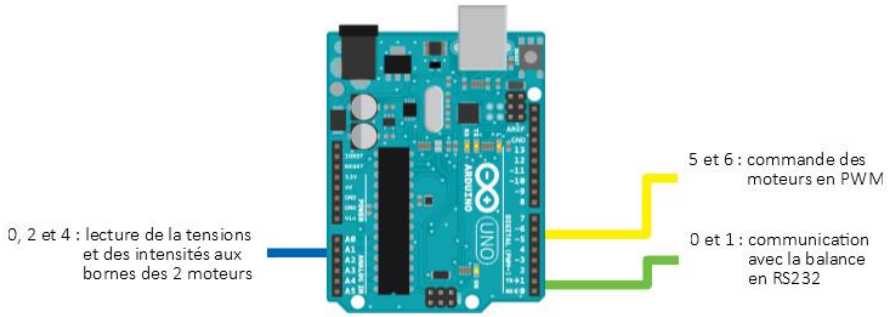


3 - L'Arduino

L'Arduino joue le rôle d'une passerelle entre le PC et le monde extérieur (capteurs, balance, moteurs…) à travers la carte électronique. L'Arduino devra donc attendre l'envoi d'une procédure de test de la part du PC qui lui indiquera comment commander les moteurs de la turbine. Une fois la procédure reçue l'Arduino effectuera le test en renvoyant au PC les valeurs des différents capteurs (tensions, intensités et force de poussée). L'Arduino devra donc être le plus rapide possible pour pouvoir avoir des courbes précises.

3.1 – Configuration de l'Arduino

3.1.1 – Broches de l'Arduino



Broches de l'Arduino utilisées

L'Arduino utilise donc 3 broches analogiques pour récupérer la valeur des différents capteurs, 2 broches PWM pour commander les 2 moteurs de la turbine, les 2 broches Rx et Tx pour communiquer avec la balance et le port série pour communiquer avec le PC.

3.1.2 – Fonctions du programme

```
void recoverDatas();                    //Récupère le procédure de test
void setTestValues();                   //Initialise les paramètres en fonction du numéro de test
void controlMotors(unsigned int pwm);   //Commande les moteurs
bool recoverWeight(float * weight);     //Récupère le poids de la balance
                                        //retourne false en ces d'erreur
void freeMemory();                      //Vide les vecteurs qui contiennent la procédure
void sendToPC();                        //Envoi les données au PC
```

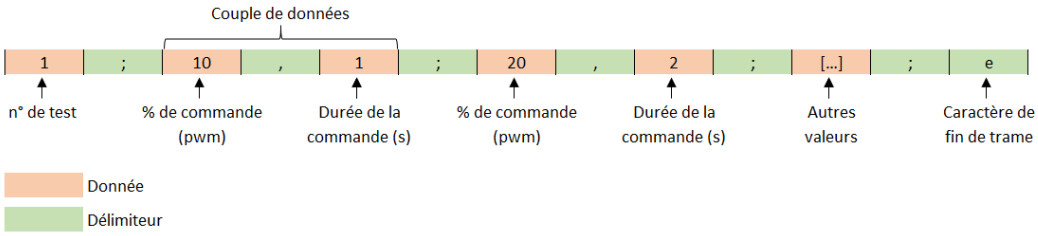
Fonctions du programme

3.2 – Communication avec le PC

Le PC devra donc communiquer avec l'Arduino via le port série, que ce soit pour ordonner l'exécution d'un test ou récupérer les données du test. Le nombre de données envoyés pouvant être important, la vitesse de transmission est réglée sur 115 200 bauds.

3.2.1 – Réception de la procédure de test : *recoverDatas()*

Pour effectuer un test le PC commence par envoyer la procédure à l'Arduino via le port série USB via cette trame.



Trame procédure de test

Les délimiteurs permettent de découper la trame et d'utiliser ses données.

Cette procédure de test contient :

- Le numéro du test : permet de définir la durée de commande de chaque moteur et la méthode de commande (échelon ou rampe).
- Plusieurs couples de données qui correspondent à un pourcentage de commande des moteurs pendant un temps donné.

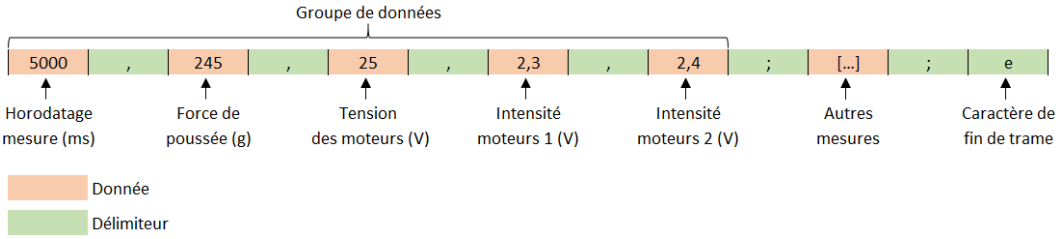
La récupération et le stockage des données de la procédure est effectué dans la fonction *recoverDatas()*.

```
void recoverDatas()
{
  bool data_received = false;
  while(!data_received) //While no data have been received
  {
    if(Serial.available())//Wait for datas on the Serial port
    {
      unsigned int data_length = 0;
      inchar = Serial.read();
      while(inchar != 'e')//'e' end of the data frame
      {
        data_frame[data_length] = inchar;//char read saved
        data_length++;
        inchar = Serial.read();//we read the next char on the frame
      }
      data_received = true;

      if(data_frame == NULL)exit(0);//if error, exit()

      token = strtok(data_frame,";");//We split data_frame with ","

      vec_data.push_back(token);//vec_data[0] = test number
      char str = vec_data[0][0];
      //nb_test = atoi(str);

      while(token != NULL)//On all the frame
      {
         token = strtok(NULL, ";");//We took the next token
         if(token!=NULL)
         {
           vec_data.push_back(token);//token store on vec_data
           nb_token++;
         }
      }

      for(int j=0; j<nb_token-1; j++)//For all token in vec_data
      {
        token = strtok(vec_data[j+1],",");//We split vec_data with ","
        vec_PWM.push_back(atoi(token));//First token transformed in unsigned and saved on vec_PWM

        token = strtok(NULL,",");//Go on the second token
        vec_Time.push_back(atoi(token)*1000);//Second token transformed in unsigned and saved on vec_Time
      }
    }
```

Pour récupérer ces données et les réutiliser, il va falloir les stocker dans des vecteurs. Pour ce faire la trame est tout d'abord parsée au niveau des « ; » à l'aide de la fonction *strtok()* pour obtenir le numéro du test (premier morceaux (= premier token) de la trame) et les couples de données (autres morceaux de la trame (exemple : « 10,2 »)). Les couples de données (*tableau de char*) sont ensuite parsés de nouveau au niveau des « , ». Les deux parties (*tableau de char*) sont transformées en *unsigned int* via la fonction *atoi()* puis enregistrées dans deux *<vectors> d'unsigned int* (un pour le pourcentage et un pour le moment de la commande) afin d'être utilisé plus tard lors du test. Pour finir le programme établi la méthode de commande (échelon ou rampe) et la durée de commande des moteurs en fonction du numéro de test.

3.2.2 – Envoi des données : *SendToPC()*

Une fois le test commencé, l'Arduino va à intervalles réguliers lire les valeurs de la balance et des capteurs de tension et courant afin de les envoyer au fur et à mesure au PC. La trame envoyée vers le PC est aussi organisée de manière à simplifier la récupération des données.



Trame de données

La trame renvoyée au PC contient ainsi plusieurs groupes de 5 valeurs séparés par des points virgules. Ces groupes contiennent donc le moment du test où les données ont été relevées (horodatage) ainsi que la force de poussée, la tension et les intensités aux bornes des deux moteurs de la turbine.

```
//SEND DATAS
if((millis() - previous_time_send) >= time_interval_send && in_command)
{
  sendToPC();
}
```

Appel de la fonction d'envoi

Afin d'effectué des envois vers le PC de manière régulière le programme utilise la fonction millis(). millis() renvoie le temps écoulé depuis le démarrage de l'Arduino ce qui permet de déduire le temps écoulé depuis le dernier envoi (la variable : *time_interval_send* définie le temps (ms) entre deux envoi de données).

```
void sendToPC()
{
  float weight;
  if(recoverWeight(&weight))//If a weight is recovered
  {
    //DATAS
    Serial.println(String(millis()-init_time,DEC)      //TimeStamp
    + "," + String(weight,DEC)                         //Weight
    + "," + String(analogRead(voltage_PIN),DEC)        //Voltage
    + "," + String(analogRead(current_PIN1),DEC)       //Intensity motor 1
    + "," + String(analogRead(current_PIN2),DEC)       //Intensity motor 2
    + ";");
    previous_time_send=millis();//Update of the time
  }
  //if the balance send a error trame, nothing is send to the PC*/
}
```

Fonction d'envoi au PC

Envoi d'effectuer l'envoi des données au PC *sendToPC()* appelle la fonction *recoverWeight()* afin de récupérer la force de poussée (poids de la balance). Si la fonction retourne *false* (trame reçue par la balance = trame d'erreur) rien n'est envoyé au PC sinon une trame contenant toutes les valeurs est envoyée.

3.3 – Communication avec la balance

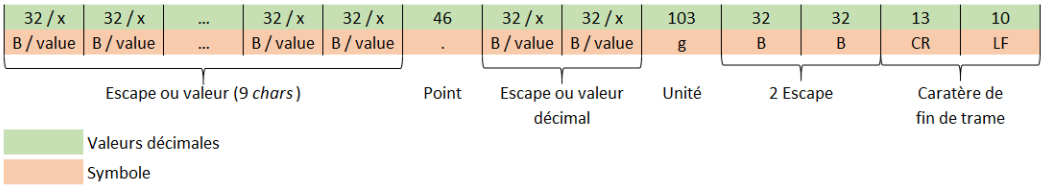
La force de poussée de la turbine est obtenue grâce à une balance qui renvoie ensuite sa pesé à l'Arduino via une liaison RS232. Les broches 0 (Tx) et 1 (Rx) de l'Arduino nous permettent de communiquer en RS232. Pour pouvoir effectuer la communication entre ces deux matériel la carte électronique est indispensable afin de transformer le niveau TTL de l'Arduino (0V/5V) au niveau du réseau RS232, dans notre cas (-10V/+10V).

3.3.1 – Configuration de la balance

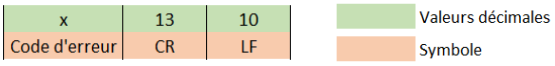
Pour que les deux matériels puissent communiquer, ils doivent être configurés avec les mêmes paramètres, soit une vitesse de communication réglée au maximum (19200 bauds) afin d'avoir le plus les données le plus rapidement possible. L'envoi des données par la balance est réglé sur ordre télécommande (la balance n'envoi le poids que si elle reçoit un 'W' de la part de l'Arduino) et la tare automatique est enlevée.

3.3.2 – Communication : *recoverWeigth()*

La balance renvoie à l'Arduino deux types de trame : les trames de poids et les trames d'erreur.



Trame de poids



Trame d'erreur

Les trames renvoyées sont composées de 18 *chars* pour la trame de poids et de 3 *chars* pour la trame d'erreur.

La fonction *recoverWeight()* se charge d'émettre une requête à la balance. Lorsque la balance retourne une trame sur le port RS232, l'Arduino stocke la trame envoyée dans un tableau de char (CR et LF étant les caractères de fin de trame). Une fois la trame récupérée, le programme vérifie son contenu.

- S'il s'agit d'une trame de poids (18 *chars*) on vient retourner la force de poussé pour l'envoyer au PC.
- S'il s'agit d'une trame d'erreur, rien n'est renvoyé à l'ordinateur.

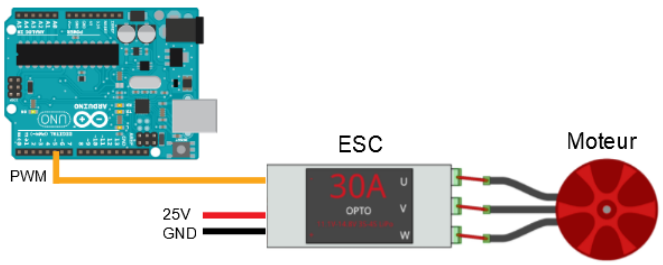
```
bool recoverWeight(float * weight)
{
  char trame[20];
  char weight_trame[20];
  RS232Serial.print('W');//Request to the balance
  if(RS232Serial.available())
  {
    unsigned int data_length = 0;
    inchar = RS232Serial.read();
    while(inchar != 10)//Final char of the trame
    {
      trame[data_length] = inchar;//Char saved on the trame
      data_length++;
      inchar = RS232Serial.read();
    }

    if(data_length!=18)return false;//If error trame

    else
    {
      data_length = 0;
      while(trame[data_length] != 'g')
      {
        weight_trame[data_length] = trame[data_length];//Recover the weight of the trame
        data_length++;
      }
      *weight = atof(weight_trame);//Parse string to float
      return true;
   }
  }
  return false;//If no answer
}
```

Fonction de communication avec la balance

Par manque de temps nous n'avons pas pu informer l'utilisateur du PC lorsque la balance renvoyait une trame d'erreur (ces trames d'erreur indiquent une mauvaise configuration de la balance).

3.4 – Commande des moteurs

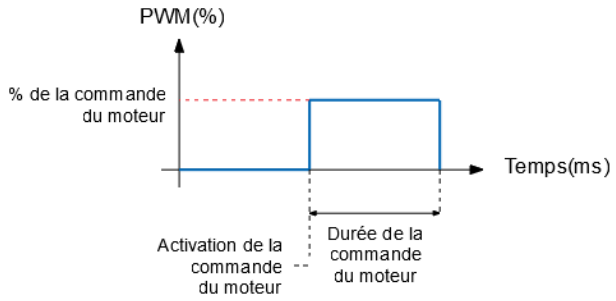
Les deux moteurs de la turbine sont contrôlés en PWM (modulation de largeur d'impulsion). A partir de ce signal PWM et d'une alimentation continue 25V, un ESC (Electronic Speed Control) se chargera ensuite de produire un signal triphasé pour contrôler les moteurs des turbines à la vitesse du PWM.



Connexion Arduino-ESC-Moteur

La procédure de test envoyée par l'ordinateur nous permet connaître comment commander les moteurs.

- Commande en échelon ou rampe via le numéro du test.
- Le temps entre deux commandes via le numéro du test.
- La durée et le pourcentage de commande des moteurs.



3.4.1 – Commande des moteurs : *controlMotors()*

```
void controlMotors(unsigned int pwm)
{
  int value =(pwm * 2.55);     //Control between 0 and 255
  analogWrite(PWM_PIN1,value);//Motor 1
  analogWrite(PWM_PIN2,value);//Motor 2
  previous_time_control = millis();
}
```

Commande des moteurs

Pour commander les moteurs on vient transformer le pourcentage et une commande entre 0 et 255 via pwm*2.55.

3.4.2 – Commande en échelon

```
//CONTROL MOTOR
if((millis() - init_time) >= time_until_next_control && !in_command && nb_test==0)
{
  controlMotors(vec_PWM[id_command]);//Control of the motors
  in_command = true;
}

//STOP MOTORS
if(((millis() - previous_time_control) >= vec_Time[id_command]) && in_command)
{
  controlMotors(0);
  time_until_next_control += time_interval_between2control + vec_Time[id_command];
  id_command++;       //Next command
  in_command = false;
}
```

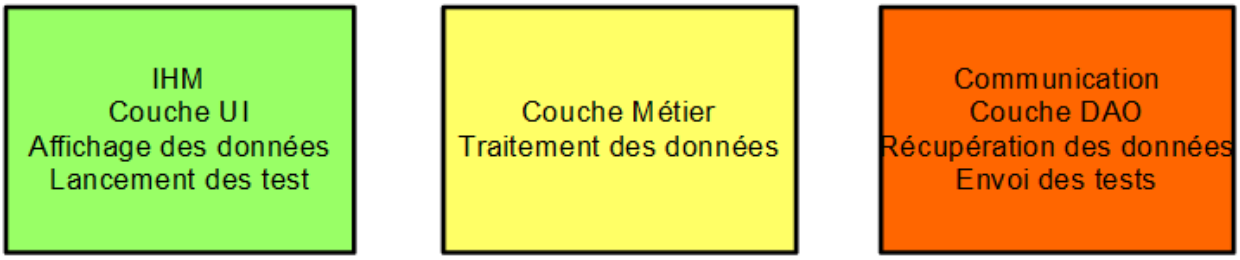
Appel de la fonction de control des moteurs

Commande pour l'envoi des données au PC, la fonction millis() nous permet de savoir le temps écoulé depuis le début du test et depuis le début de la commande du moteur. Ici on vient comparer le temps écoulé depuis le début du temps aux temps auquel doivent être commander les moteurs. Si le temps écoulé depuis le début de la commande est supérieur au temps durant lequel les moteurs doivent être contrôlés ils sont stoppés.

Par manque de temps nous n'avons pas pu commander les moteurs selon les tests en rampe.

4 - Application PC

Dans le cadre du projet nous devons créer une IHM qui permettra à l'utilisateur de lancer des tests sur les turbines ainsi que de visualiser les données. L'IHM sera réalisé en C# sous Visual Studio. J'ai choisi de réaliser l'application sous le modèle 3 couches (accès aux données, traitement des données, interface), ce modèle permet une amélioration future simple puisque l'on peut changer une des couches sans refaire tout le logiciel.



Architecture de l'application

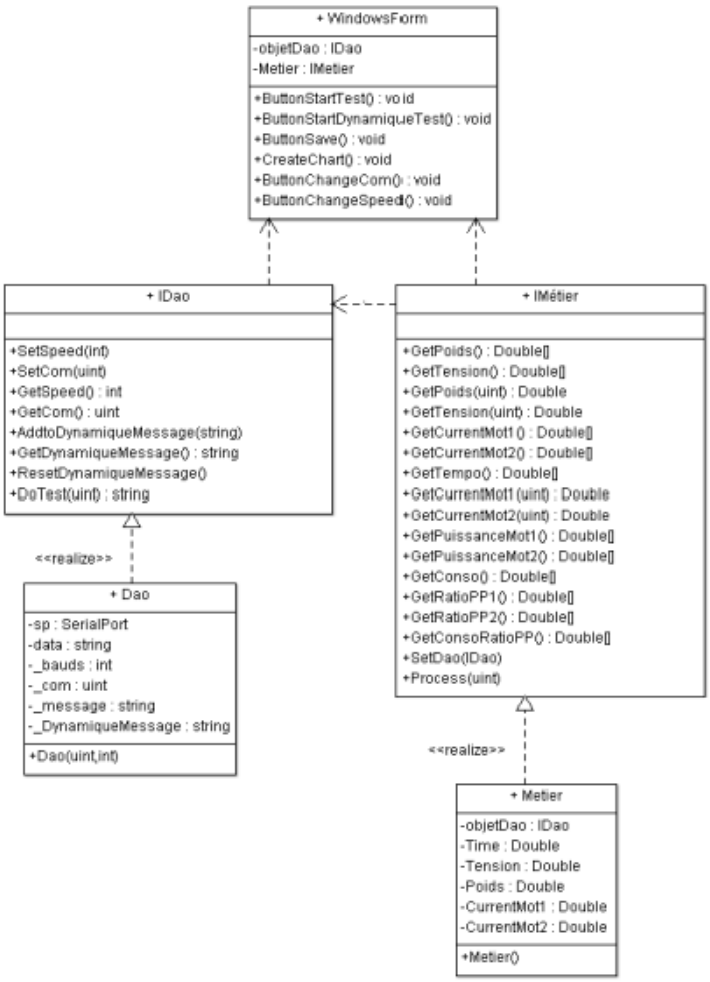


Diagramme UML de l'application

4.1 – Couche accès aux données (DAO)

La couche d'accès aux données doit permettre de communiquer avec l'arduino afin de réaliser les tests, elle doit aussi stocker les donner avant de les envoyer vers la couche métier.

Dans un premier temps nous devons communiquer avec l'arduino, pour cela j'utilise un objet du type SerialPort. En effet ces objets permettent une communication sur le bus USB et la librairie est très complète : Lecture / écriture / information sur le bus en temps réel.

```
/** Création de l'objet DAO; définition du port de communication et de la vitesse de transmission en bauds **/
/** Création d'un objet Serial port permettant la communication USB **/
public Dao(uint choixPort, int bauds )
{
        _com = choixPort;
        _bauds = bauds;
        sp = new SerialPort("COM"+choixPort, bauds);
}
```

Lors de la création de l'objet DAO dans la couche UI on doit déclarer un port de communication et une vitesse de base pour le port afin de créer cet objet mais j'ai des setters qui permettent à l'utilisateur de changer ces deux données lors de l'exécution du programme.

Si le port de communication est déjà utilisé le programme ne plante pas il affiche seulement une fenêtre d'erreur avec un message, ce qui est un avantage de Serial Port.

Une fois la communication avec l'Arduino réalisé il faut lancer des tests et récupérer les données, afin de lancer des tests j'envoie sur le bus USB un message à l'Arduino avec le type de test (échelons ou rampe), dans le premier cas j'envoie les échelons à atteindre et dans le deuxième des points à atteindre et l'Arduino les modifie en allure de courbe.

```
        // switch de choix du test à réaliser
        switch(ChoixTest)
        {

            case 0: _messages = "0;10,10;20,10;30,10;40,10;50,10;60,10;70,10;80,10;100,10;e";break;
            case 1: _messages = "1;10,10;20,10;30,10;40,10;50,10;60,10;70,10;80,10;100,10;e"; break;
            case 2: _messages = "1;10,10;20,10;30,10;40,10;50,10;60,10;70,10;80,10;100,10;e"; break;
            case 3: _messages = "1;10,10;20,10;30,10;40,10;50,10;60,10;70,10;80,10;100,10;e"; break;
            case 5: _messages = "0;"+_DynamicMessage+"e";break;
        }

        // ouverture du port de communication
sp.Open();
data = "";
sp.Write(_messages); // on envoit le test
```

Sur l'image ci-contre on peut apercevoir que le test comporte un type (0 ou 1) en fonction d'échelon ou rampe puis les points à atteindre ou les échelons et le caractère fin de chaîne, ensuite j'envoie le test avec la commande *SerialPort.Write()*.

```
// Ici on lit les données tant que nous ne recevons pas le caratère fin de chaine "e"
bool FlagStop = false;
do
{
    data += sp.ReadExisting();

    if(data.Contains("e"))
    {
        FlagStop = true;
    }
}
while (!FlagStop);
```

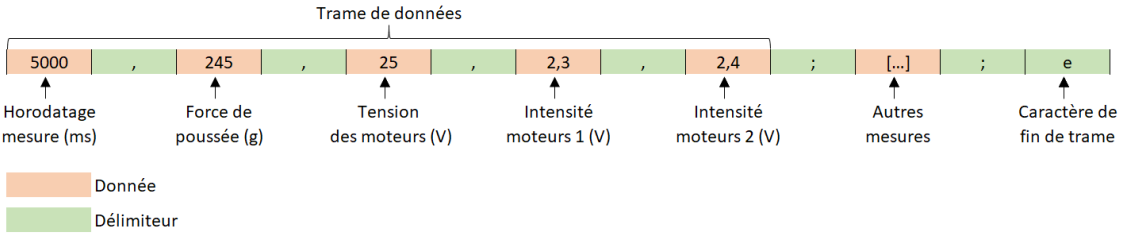
Ensuite j'attends la réponse de l'Arduino, c'est-à-dire que je lis le buffer du bus USB et j'enregistre les données tant que je ne reçois pas le caractère de fin de chaîne. Ensuite je ferme le port et je retourne les données reçues.

```
// on ferme le port de communication
sp.Close();
// on retourne les données reçues
if (data.Length != 0) return data;
else return null;
```

4.2 – Couche traitement des données (métier)

Pour notre application la couche métier va traiter les données reçues depuis la couche DAO, ces données doivent être utilisable par la couche UI qui les affichera sous forme de courbes. Pour cela on doit transformer la chaîne de caractères que nous a transmis l'Arduino en tableaux de doubles correspondant aux valeurs des mesures.

La chaîne de caractère est de la forme



Le but est de séparer chaque trame de mesures par le « ; » puis pour chacune de ces trames on redivise avec la virgule pour récupérer chaque mesure que l'on pourra « parser » en double et l'enregistrer dans un tableau.

```
// on divise une première fois la chaine de caractère pour récupérer les paquets de données
String[] substringPaquet = data.Split(delimiteurPaquet);

// pour chaque paquet de données
foreach (var sub in substringPaquet)
{
    // on divise le paquet pour récupérer les données une à une
    String[] substringdata = sub.Split(delimiteurData);
    idx = 0;
    foreach(var sub2 in substringdata)// pour chaque données
    {
        if (sub2 != "e") //on esquive le caractère de fin de chaine par précotion précaution
        {
            switch (idx) // en fonction de l'index on ajoute la valeur à la list correspondante
            {
                case 0:
                    // on utilise la fonction getdouble pour parser la chaine de caractère et arrondir
                    // pour garder seulement deux chiffres après la virgule
                    if (GetDouble(sub2, 7536) != 7536) ListTime.Add(arrondir(GetDouble(sub2, 0), 2));
                    break;
                case 1:
                    if (GetDouble(sub2, 7536) != 7536) ListPoid.Add(arrondir(GetDouble(sub2, 0)*9.81, 2));
                    break;
```

Ci-contre le code permettant de découper la chaîne de caractères pour la trier dans différentes listes que je transformerai en tableau à la fin de la fonction.

```
// on enregistre les données dans les tableaux de double
Time = ListTime.ToArray();

Poids = ListPoid.ToArray();
Tension = ListTension.ToArray();
CurrentMot1 = ListCurrent1.ToArray();
CurrentMot2 = ListCurrent2.ToArray();
```

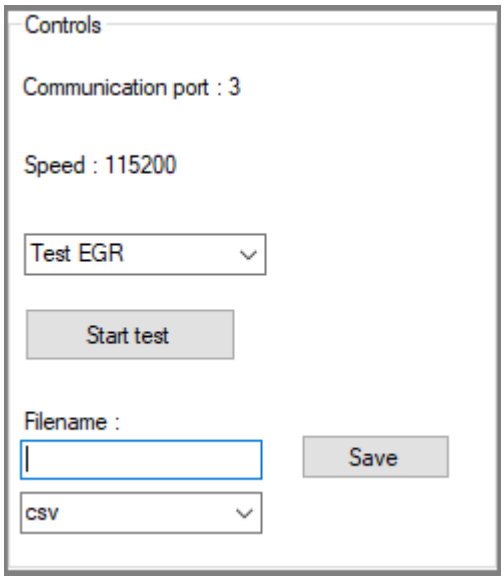
Ces mesures ne sont pas suffisantes pour notre application, en effet nous avons besoin de la puissance qui est un rapport entre la tension des moteurs et le courant consommé à chaque moment :

```
/** retourne le tableau de double avec les mesures de puissance consommée pour la turbine 1**/
public double[] getPuissanceMot1()
{
    try {
    List<double> PuissanceMot1 = new List<double>(); ;
    int idx = 0;
    foreach(double c1 in getCurrentMot1())
    {
        PuissanceMot1.Add(c1 * getTension(idx));
            idx++;
    }

    return PuissanceMot1.ToArray();
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

On répète la même fonction pour le moteur 2 et les ratios Poussée/Puissance.

4.3– Couche présentation des données (UI)

La couche UI a pour objectif de permettre à l'utilisateur de lancer un test simplement et de pouvoir visualiser les résultats reçus depuis l'Arduino. Pour réaliser la première tâche j'ai choisi un formulaire simple où l'utilisateur peut choisir le test qu'il veut effectuer dans un combobox et le lancer, ce formulaire affiche également les données importantes telles que le port de communication ou la vitesse de transmission.



On peut apercevoir que ce formulaire permet à l'utilisateur de sauvegarder le test dans un fichier du type csv ou txt.

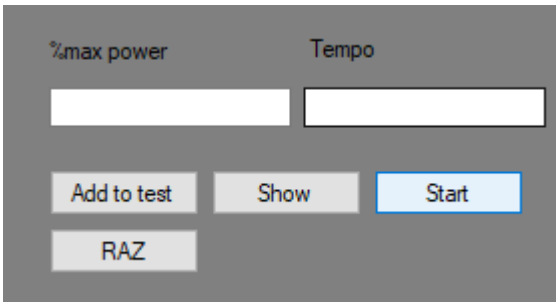
Pour cela j'utilise un objet streamwriter qui permet d'écrire dans un fichier texte, en effet un fichier csv n'est qu'un fichier texte délimitant les colonnes par des « ; ». Cette fonction va enregistrer les valeurs de tensions, poussée et courants des deux moteurs horodatés.

```
try
{
    using (StreamWriter writer = new StreamWriter(textBoxNameSave.Text + "." +
                                                  comboBoxFileSave.SelectedItem.ToString()  /*".csv"*/))
    {
        int count = 0;
        DateTime localDate = DateTime.Now;
        writer.WriteLine(localDate.ToString() + ";");
        writer.WriteLine("Horodatage;Poids;Tension;Courant Moteur 1;Courant Moteur 2");
        foreach (double t in metier.getTempo())
        {
            writer.WriteLine(t + ";" + metier.getTension(count) + ";" + metier.getPoids(count) + ";" +
                             metier.getCurrentMot1(count) + ";" + metier.getCurrentMot2(count) + ";");
            count++;
        }
    }
    MessageBox.Show("Saved");
}
```

Afin de lancer un test j'utilise le numéro d'index de la combobox qui me permet de savoir quel cas est validé dans le switch de la couche DAO.

```
//Lancement du test
private void buttonStartTest_Click(object sender, EventArgs e)
{
    try
    {
        metier.Process((uint)comboBoxChoixTest.SelectedIndex);
        Create_Chart();


    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Lors de discussions avec l'entreprise une idée a été mise en avant, la possibilité de faire des tests dynamiques. En effet notre panel de 4 tests était suffisant mais la possibilité de donner le droit à l'utilisateur de créer son propre test et de le réaliser sur les turbines est intéressante. Pour cela j'ai ajouté un formulaire permettant d'ajouter des échelons à un test dit dynamique.



Le message pour ce test dynamique est directement enregistré dans la couche DAO. Grâce à cela lorsque je lance un test j'ai juste ajouté une ligne à mon switch qui permet d'envoyer la chaîne de caractères correspondant au test.

Une fois le test envoyé et les données reçues via la couche Dao et traités par la couche métier, il faut les visualiser sur l'IHM. Visual studio dispose d'objet chart qui permettent de visualiser des séries de données sous forme de graphiques l'utilisation est simple puisqu'il suffit de passer deux tableaux de données en entrée et de choisir le type de graphique que l'on souhaite (courbe dans notre cas).

```
var series1 = new Series(" Inlet :" + idx);
series1.Points.DataBindXY(metier.getPoids(), metier.getPuissanceMot1());
series1.ChartType = SeriesChartType.Line;
chart1.Series.Add(series1);
```

Interface finale

Conclusion

La mise en œuvre de ce projet nous a permis de prendre en compte toutes les facettes d'une organisation de travail en petit groupe. Nous nous sommes réparti le travail équitablement en définissant 3 axes sur lequel le projet devait évoluer. La première partie était la réalisation d'un shield Arduino permettant une communication entre les moteurs, la balance et l'Arduino. La seconde partie consistait à programmer l'Arduino pour faire fonctionner le shield permettant de mesurer les informations utiles liées aux moteurs et de les communiquer à une IHM sur un PC. Enfin la dernière partie consistait en la réalisation d'un IHM sur Windows, cet IHM permet de lancer des tests sur les moteurs et de visualiser les résultats sous forme de courbes.

Nous nous sommes séparé le travail en fonction des envies et expériences de chacun, ce qui nous a permis d'avancer plus vite sur la réalisation de notre solution.

Nous avons rencontré des difficultés sur la programmation de l'Arduino, la mémoire de l'atmega32 étant faible l'utilisation des allocations dynamiques était de rigueur, ces mêmes allocations posaient problème lors de plusieurs tests à la suite. Nous avons trouvé une solution tard en l'utilisation d'une bibliothèque C++ permettant l'utilisation de List.

RESUME

En coopération avec Neva Aerospace l'ISTIA doit développer un banc de tests pour turbine de drone lourd. Ce présente la solution de mise en œuvre pour la réalisation de ce projet décomposé en 3 parties : Le Hardware, L'IHM et l'interface de contrôle et communication.

Mots clés : Drone lourd, turbine, banc de test, IHM, Neva Aerospace

ABSTRACT

In cooperation with Neva Aerospace, ISTIA has to develop a test bench for a heavy UAV turbine. This presents the implementation solution for the realization of this project divided into 3 parts: Hardware, GUI and control and communication interface.

Keywords : Heavy drone, turbine, benchmark, HUD, Neva Aerospace