2016-2017

Cycle Ingénieur AGI, 2ère année

Rapport de projet

RAPPORT DE PROJET NEVA AEROSPACE

Automatisation d'un banc de test pour turbines de drone

Julien Monnier
Elisa Rosse
Marius Runge

Sous la direction de M. Franck Mercier

REMERCIEMENTS

L’ensemble de l’équipe tient à remercier Franck Mercier le responsable du projet, pour ses conseils, son soutien et son aide.

Nous remercions, Monsieur Renaud-Pierre Blanpain, Responsable Ingénieur et Prototypage chez Neva Aerospace Ltd pour toutes les informations qu’il nous a fournies, et qui ont contribué à l’avancement de notre projet.

Nous tenons également à remercier l'entreprise Neva Aerospace pour nous avoir proposé cet intéressant projet de mise en place et automatisation de banc de test pour turbines de drones.

Nous remercions également l'ISTIA pour les contacts et partenariats qu'elle prend la peine d'établir avec des entreprises ayant la possibilité de nous proposer des projets enrichissants chaque année.

Sommaire

ANNEXES

Introduction

Dans le cadre de notre seconde année de cycle ingénieur à l'ISTIA (Institut des Sciences et Techniques d'Angers), les étudiants ont l'opportunité de réaliser un projet directement proposé par une entreprise extérieure à l'école. Ce projet s'effectue sous la responsabilité d'un ou plusieurs enseignants-chercheurs et poursuit les buts suivants :
L'évaluation des étudiants quant à leurs capacités à gérer et organiser un projet sur une durée de plusieurs mois, à utiliser et mettre en pratique les connaissances et outils vus en cours tout en tâchant de répondre au besoin d'une entreprise et à respecter un cahier des charges.
Ce projet permet également aux étudiants d'acquérir de nouvelles compétences et d'apprendre à utiliser de nouveaux outils relatifs à leur futur métier.

Notre équipe a donc réalisé un projet destiné à l'entreprise Neva Aerospace Ltd, consortium européen de conception et fabrication de drones destinés au transport possédant entre autre un site de développement basé à Angers. Le projet confié par cette entreprise visait à automatiser les cycles de mesure des bancs de test des turbines de ces drones, actuellement effectués manuellement. Ce système posant des problèmes de temps et précision, nous devions proposer une solution répondant à ce besoin. Nous avons donc choisi de proposer une application permettant de lancer, enregistrer, recommencer, etc.. les cycles de mesures.

Ce rapport présente les différentes parties de notre démarche afin de résoudre ce problème, depuis la façon dont nous nous sommes organisés et avons géré le projet jusqu'aux problèmes rencontrés en passant par les différentes parties de conception, les outils et technologies utilisées ainsi que les quelques points qu'il est possibles de compléter.

I – Présentation du projet

1 Contexte et Objectifs

1.1. L'entreprise : Neva Aerospace Ltd

Neva-Aeropace est un consortium européen de conception et fabrication de drones lourds destinés au transport d'équipement et aux réparations dans les zones à risques. Sur le site d'Angers sont notamment conçues et caractérisées des turbines nouvelle génération permettant le vol vertical. Afin de les caractériser, ces turbines sont testées sur un banc de mesures manuel.

Siège : Sussex Innovation Centre, Brighton BN1 9SB, United Kingdom

Site de Développement : Angers, France et Pease Pottage, United Kingdom

Centre d'essais en vol : Vilnius, Lithuania

1.2. Les Missions

Objectif : Automatiser ce banc de test autant dans son cycle de mesures que dans ses résultats :
- Cycle complet
- Cycle partiel

Nos missions consistaient donc à réaliser une application ou un logiciel permettant d'effectuer de manière automatique tout ce qui était fait manuellement auparavant.

Travail à faire dans le respect du cahier des charges :

- Mise en place d'un système d'automatisation sur base d'Arduino avec affichage et sauvegarde des données.
- Construction des caractéristiques des turbines (courbes et valeurs) sur écran d'ordinateur
- Définir l'interface homme/machine
- Migration sur système industriel (LabView) possible en fin de projet

1.3. L'Équipe

Notre équipe est composée de trois étudiants en deuxième année de cycle ingénieur à l'ISTIA en spécialité Génie des Systèmes Industriels. Nous sommes tous les trois en option AGI (Automatique et Génie Informatique)

Nous nous sommes dès le début, répartis les rôles ainsi :

Marius Runge : Développement application (python 3.4, javascript), interface utilisateur

Élisa Rosse : Développement interface graphique, organisation du projet

Julien Monnier : Programmation Arduino, électronique

2 Gestion du projet

2.1. Mise en place

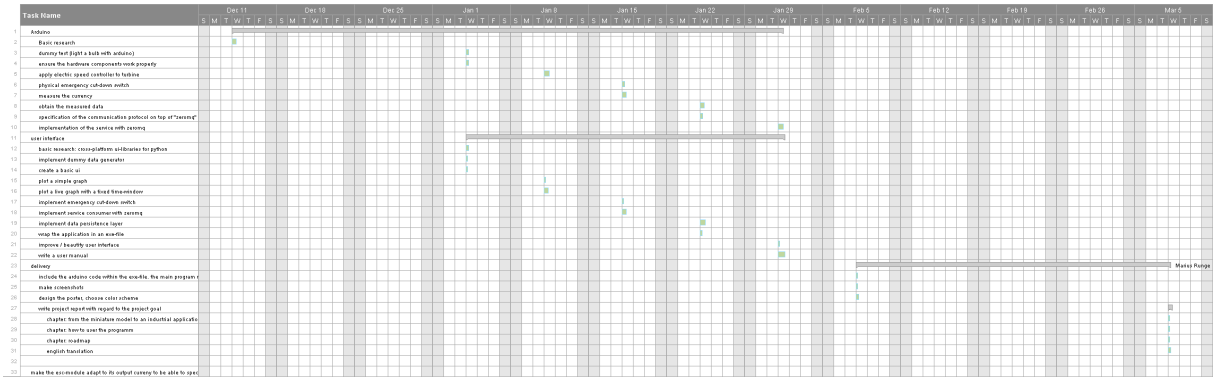
C'est à la gestion de notre projet que nous avons consacré la première demie journée du temps impartis à sa réalisation. Nous avons commencé par faire une liste des tâches[1], la plus complète possible , en détaillant et en classant les tâches en trois grandes parties : la partie arduino, l'interface utilisateur et la restitution.

Ceci fait nous avons fait quelques recherches pour savoir quel logiciel de gestion était le plus adapté. Une fois ce logiciel choisi nous avons généré un Diagramme de Gantt, un bon moyen de se mettre au clair sur les durées ainsi que sur qui devait réaliser quelle tâche et de s'en souvenir.

1 Cf Annexe 1

2.2. Planning

Diagramme de Gantt [Vue d'ensemble – détails en annexe]



II – Réalisation

1 Partie Électronique (Arduino - Hardware)

1.1. Synoptique général

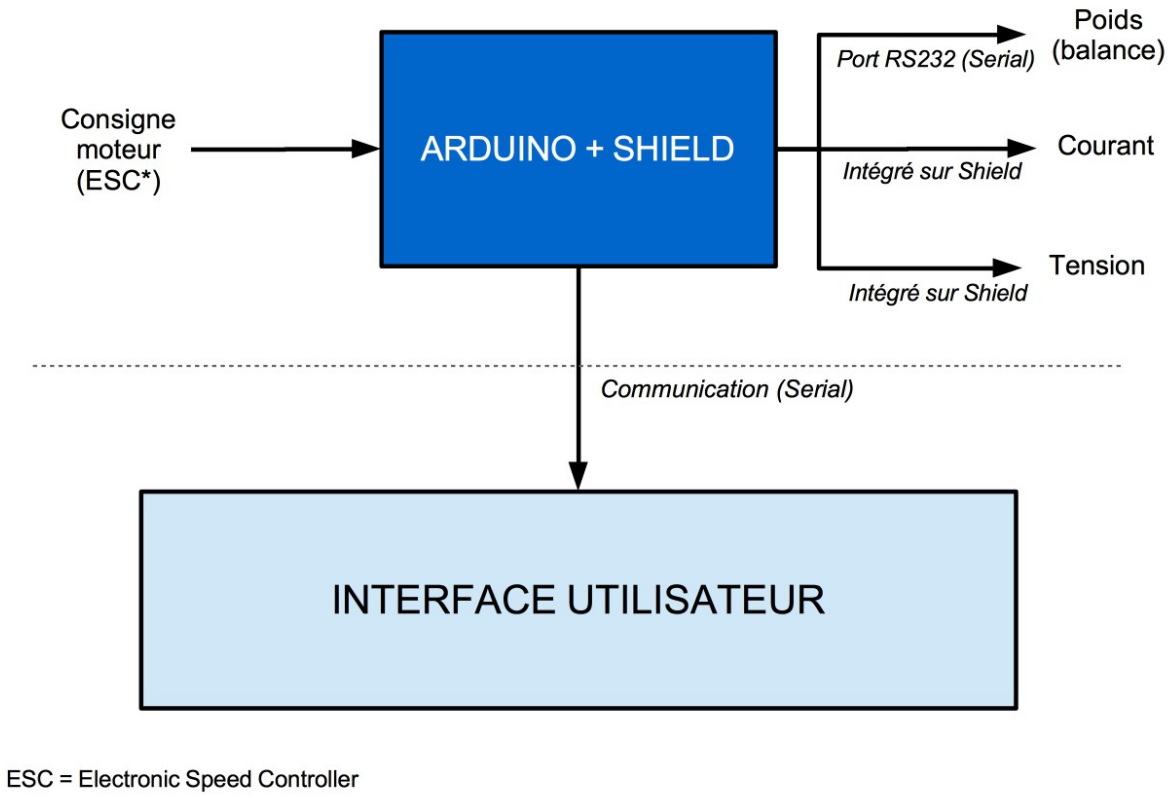


Figure 1 : Synoptique général

Dans l'optique d'exploiter les données issues du banc de test, notre projet doit contenir du matériel électronique. Celui-ci doit assurer le lien entre la mesure des grandeurs physiques qui nous intéressent et l'exploitation de ces mesures depuis l'interface utilisateur. La partie électronique du projet doit donc faire l'objet d'instruments de mesure et de commande.

1.2. Composition

1.2.1. Microcontrôleur Arduino UNO

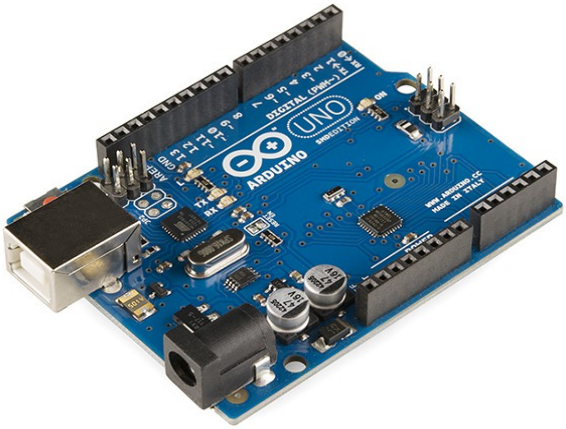


Figure 2 : Microcontrôleur Arduino UNO

Les cartes Arduino sont des microcontrôleurs dont la plus part des composants sont sous licence libre. Ces cartes conçues pour l'éducation et le prototypage ont la particularité d'être simple à utiliser car elles disposent de nombreux éléments par défaut (entrées/sorties analogiques et numériques). La programmation Arduino utilise le langage C++. Dans le contexte actuel, c'est la carte Arduino qui va établir de lien entre les grandeurs physiques issus du banc de test et l'exploitation de ces grandeurs par l'interface.

1.2.2. Shield Arduino

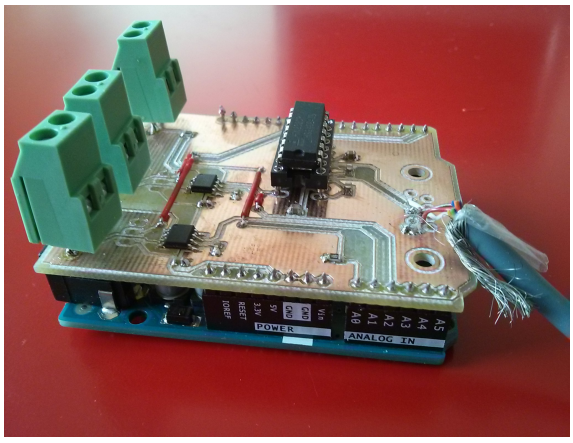


Figure 3 : Shield Arduino

Les connecteurs femelles de la carte Arduino offrent la possibilité d'intégrer une extension. La carte à elle seule ne peut pas convenir dans le contexte du projet. Le shield Arduino a pour rôle de mesurer le courant, la tension et établir le dialogue avec la balance (voir figure 6) grâce à son port série. Nous verrons plus en détails dans la section suivante les composants présents sur le shield.

1.2.3. Contrôleur électronique (Electronic Speed Controller)

Figure 4 : ESC

Les moteurs brushless faisant tourner les turbines sont commandable à l'aide d'un contrôleur. Ce dispositif permet donc de moduler la vitesse de rotation du moteur. La carte Arduino peut générer des signaux PWM (ou MLI = Modulation de Largeur d'Impulsion) à l'aide de librairies de programmation. Ils permettent de commander le moteur depuis les connecteurs de commandes (marron, orange et jaune sur la Figure 4).

1.3. Acquisition des grandeurs physiques

Le cahier des charges nous impose de relever 3 grandeurs physiques principales. Nous avons vu précédemment la procédure de test des turbines effectuée sur le banc qui met en jeu l'intensité du courant du moteur, la tension d'alimentation et la poussée de la turbine. Le courant et la tension peuvent être mesurés depuis l'entrée du système et la poussée peut être mesurée depuis la balance (voir le principe de base en introduction). Le microcontrôleur Arduino par défaut n'a pas les moyens d'exploiter les données physiques que nous cherchions à relever. Nous avons donc fabriqué un shield (carte d'extension pouvant être ajouté à la carte Arduino) composé de capteurs de tension et de courant ainsi qu'une communication série vers la balance afin d'obtenir la poussée.

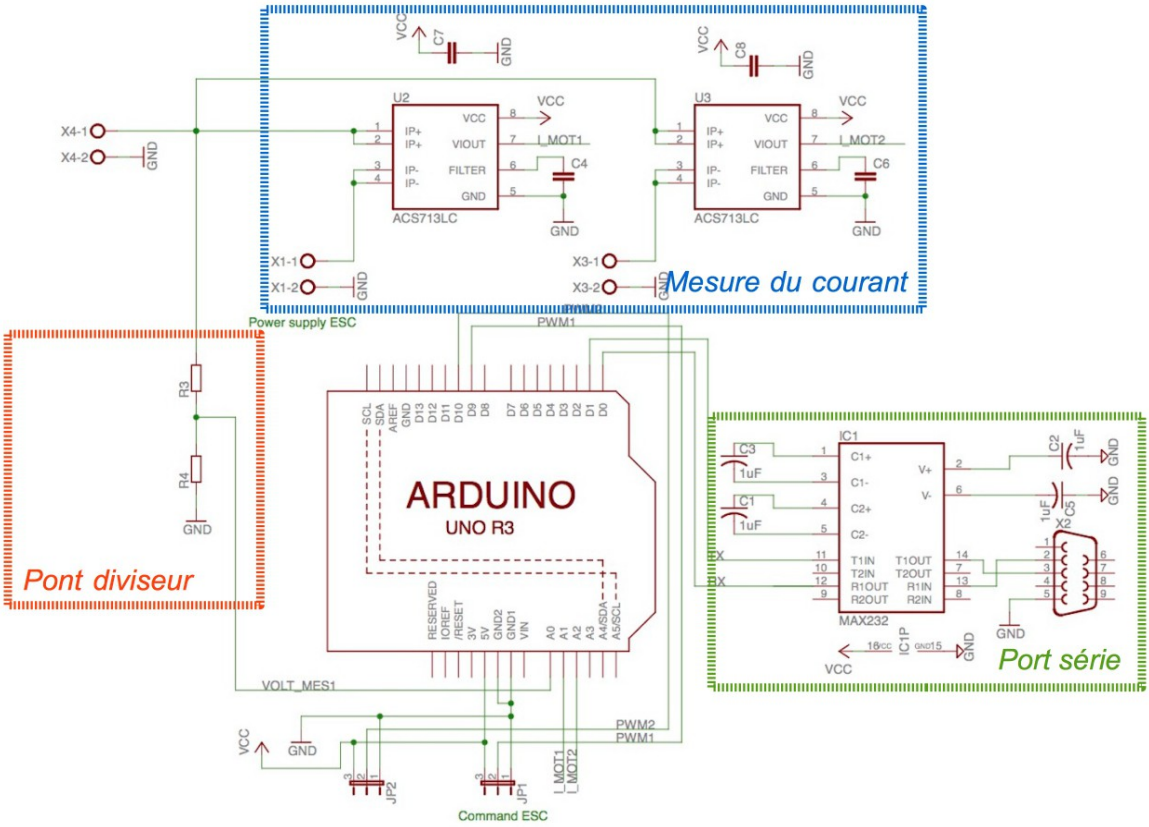


Figure 5 : Schéma électronique

1.3.1. Courant/Tension

Le shield est équipé de 2 capteurs de courant (ou ampèremètre) et d'un pont diviseur pour mesurer la tension (voir encadré bleu et orange figure 5). La carte Arduino disposant de plusieurs entrées analogiques, nous avons utilisé 3 trois entrées respectivement A0, A1 et A2 pour la tension, le courant du moteur 1 et le courant du moteur 2 (nous avons équipé le shield d'un deuxième ampèremètre dans le cas où un second moteur se présenterait lors des mesures). La programmation sur la carte Arduino permet ensuite de mesurer les données en temps réel.

1.3.2. Poussée

Figure 6 : Balance Kern FCB 12K1

Le châssis du banc de test est fait de sorte que la balance relève la poussée de la turbine (voir schéma de la présentation générale). La balance est équipée d'un port série via une prise RS232 pouvant faire la communication avec un système numérique telle qu'une carte Arduino, grâce aux entrées/sorties dont elle dispose. Nous avons donc équipé sur le shield une interface RS232 allant de la balance aux entrées/sorties de la carte Arduino (voir encadré vert figure 5). Le port série permet de commander et d'envoyer les données de la balance. Les trames envoyées par la balance sont de la forme suivante:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	B	B	B	B	B	B	B	B	.	.	.	B	g	B	B	CR	LF

Figure 7 : Trame type envoyée par la balance

B = espace ou blanc
. = valeur du poids
g = unité du poids

Afin de récupérer une trame de ce type, la carte Arduino a simplement à envoyer un caractère:
s ou w = récupère la valeur du poids
t = calibre la balance (t = "tare")

Dans le contexte actuel, seule la valeur du poids nous intéresse. Nous avons donc programmé la carte Arduino de sorte à ne récupérer que la valeur du poids.
Les données acquises doivent pouvoir être exploitées par l'interface utilisateur (voir figure 1). La carte Arduino doit être programmée de façon à transmettre les données vers une application web sur un PC.

2 Partie Application (UI – Software)

2.1. Travail demandé

L'interface utilisateur est un sujet délicat. Elle doit être à la fois esthétique et facile à comprendre. Comme les trois membres de notre groupe travaillent chacun sur leur propre ordinateur, chacun avec un système d'exploitation différent (Mac OS, Linux Debian et Windows 7), il nous semblait évident qu'il fallait choisir une technologie multi-plateforme afin de pouvoir collaborer, sans compter l'intérêt d'avoir une application flexible et adaptable à tous les systèmes d'exploitation. En outre il fallait emballer l'ensemble de toutes les librairies nécessaires dans un fichier exécutable, c'est à dire un *.exe pour Windows.

2.2. Analyse logiciel et première ébauche

Après avoir analysé les possibilités qui s'offraient à nous, nous avons finalement décidé de faire une application utilisant par défaut le navigateur, un logiciel déjà présent sur tous les systèmes d'exploitation. De plus utiliser les capacités de rendu d'un navigateur nous évite d'avoir à emballer un large panel de librairies pour gérer les composants graphiques. Ce choix fait, il fallait répartir les fonctionnalités de notre logiciel sur plusieurs couches.

Bien évidement il faut aussi un moyen de communiquer avec le navigateur afin d'afficher les valeurs récupérées auprès de l'arduino. Il faut pour cela disposer d'un serveur web, ce qui présente d'ailleurs l'avantage de pouvoir exposer toute fonctionnalité de l'application sur le réseau.

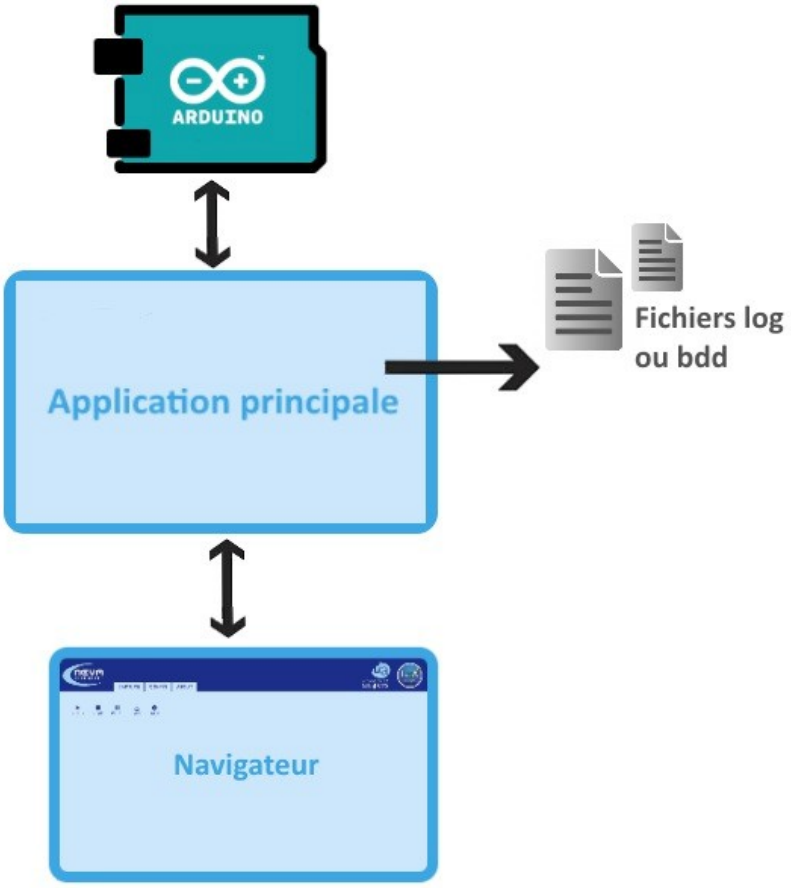


Figure 1 : Modèle de base des interactions de l'application

2.3. Modèle en couches

Comme l'indique le schéma ci-dessus, l'application principale joue un rôle central. Les appareils électroniques sont raccordés à l'arduino. Celui-ci ne fait rien de lui même. Il répond seulement aux requêtes de l'application principale.

Cependant, le navigateur montre une interface graphique avec laquelle l'utilisateur pourra interagir. Les événements déclenchés par celui-ci doivent être envoyés à l'application principale et, inversement, elle doit envoyer des données au navigateur pour qu'il puisse les afficher. Il faut donc une communication bidirectionnelle.

Le fichier log ou la base de données sert seulement à stocker les données récupérées auprès de l'arduino pour une éventuelle analyse avec par exemple Scilab.

2.4. Modèle non bloquant

Une fois ce modèle mis en place on voit facilement que le rôle de l'application principale est celui d'un routeur qui va perpétuellement relayer les messages d'un côté à l'autre.
L'attente ainsi que l'écriture d'un message bloque le processus de l'application. En d'autres termes, tout accès aux entrées ou sorties de l'application la rend non-réactive aux autres requêtes qui arrivent en même temps.
Par conséquent il nous faut plusieurs files d'exécution ou du code non-bloquant. Le langage le plus populaire pour ce type de code est bien JavaScript, soit dans le navigateur, soit dans l'environnement NodeJS. Depuis sa version 3.4 il est possible d'écrire un tel code avec python comme suit :

```
import time
def bloquant ():
   # cet appel bloque tout le fil d'exécution
   time.sleep(1)


import asyncio
@asyncio.coroutine
def coroutine (arg):
   # l'appel aux coroutines interrompre l'exécution de la fonction
   # mais ne bloque pas le processus entier.
   vield from asyncio.sleep(1)
```

Ce comportement est réalisé avec une boucle d'événement géré par le framework asyncio. Sur une couche inférieure le moteur python parcourt toutes les coroutines (fonctions ayant la capacité de céder l'exécution aux autres) enregistrées dans une boucle infinie et rend « la parole » à la première qui a quelque chose à répondre. Il existe plusieurs implémentations d'une telle boucle. Nous utiliserons l'implémentation standard.

2.5. Environnement de développement

Afin de travailler en collaboration nous avons utilisé git, ce qui nous a permis de réutiliser ce que nous avions appris en cours de génie logiciel.
Avant toutes choses nous avons essayé de faire marcher l'environnement sur un seul système d'exploitation afin d'envisager les éventuels problèmes de compatibilité au fur et à mesure du développement. Comme notre application est répartie entre un coté serveur et un côté client, il fallait mettre en place certains outils pour installer les dépendances et finalement exécuter notre logiciel, tout ceci représentant un temps non négligeable.

2.5.1. Python

La toute première étape consistait à mettre en place les librairies de python. Pour éviter de gêner d'autres projets en cours sur une même machine et tester différentes versions de python nous avons utilisé ce qu'on appelle des « environnements virtuels ». Nous nous sommes servi de pip pour définir les dépendances et pour les installer.

2.5.2. JavaScript

Pour développer le côté client nous avons installé l'environnement d'exécution *NodeJS* ainsi que son gestionnaire de librairies *npm*. Alors que l'application finale n'exécutera de code JavaScript que dans le navigateur, cet environnement permet de profiter des outils prévus pour le développement web, notamment pour le prétraitement des fichiers sources.

2.6. Les composants de l'interface

D'un point de vue technologique l'application est assez polyvalente. Elle doit être capable de communiquer sur des ports série et des websockets (ce qui implique l'usage d'un serveur web), ouvrir un navigateur installé, écrire dans un fichier log ou une base de données et accepter des paramètres. L'ensemble des librairies nécessaires doit ensuite être indépendant, c'est à dire ne pas avoir d'autres dépendances ou pré-requis à installer sur la machine d'exécution. Naturellement cette garantie était le tout premier critère à remplir.

2.6.1. L'exécutable

Il existe plusieurs approches pour emballer un logiciel écrit en python dans un fichier exécutable sous un système Windows. Après une recherche et des tests très chronophages nous avons choisi pyinstaller, une librairie qui elle-même est écrite en python. PyInstaller parcourt les fichiers sources indiqués dans un fichier de configuration de façon récursive et les pré-compile de façon à ce qu'on puisse exécuter le fichier exécutable avec un simple double-clic sans interpréteur python. De plus pyinstaller permet de relayer les paramètres de la ligne de commande au logiciel contenu. La liste des fichiers inclus peut d'ailleurs être étendue pour qu'elle contienne aussi les fichiers statiques à fournir au navigateur. Une fois emballés, ces fichiers se retrouve sur un chemin relatif au fichier exécutable comme l'indique l'exemple suivant :

```
import os, sys
# la variable _MEIPASS est fournie par pyinstaller
# pour retrouver les fichier statique emballés.
BASE_PATH = sys._MEIPASS
chemin_absolut = os.path.join(BASE_PATH, 'chemin/relatif/au/fichier/emballé')
```

2.6.2. Le serveur

Le défi de cette étape était de trouver un serveur qui : supporte le modèle non bloquant, offre une librairie supplémentaire sockjs (pour gérer des connexions socket web) et rende le tout dans un fichier

exécutable. Bien qu'il existe de nombreuses implémentations, il en existe peu qui soient écrites en python pure et donc n'exigent pas d'autres librairies individuelles par rapport au système d'exploitation. Au final nous avons trouvé CherryPy et aiohttp. Nous avons choisi le dernier. Voici un exemple :

```
import os, asyncio
from asyncio import web


def index(request):
  with open(os.path.join(BASE_PATH, 'index.html'), 'rb') as file:
     return web.Response(body=file.read(), content_type='text/html')


if __name__ == '__main__':
  loop = asyncio.get_event_loop()           # boucle d'événement
  app = web.Application(loop=loop)          # initialisation
  app.router.add_route('GET', '/', index)     # routage
  web.run_app(app, host='localhost', port=8090) # démarrage
```

2.6.3. Le navigateur

Afin d'améliorer le confort de l'utilisation du logiciel nous avons ajouté la librairie webbrowser qui permet tout simplement d'ouvrir un navigateur installé sur une URL quelconque. Dans notre cas, on veut bien-sûr naviguer sur notre propre page web qui fonctionnera désormais comme l'interface utilisateur. On étend l'exemple ci-dessus comme suit :

```
import os, asyncio, webbrowser
from asyncio import web


def index(request):
  with open(os.path.join(BASE_PATH, 'index.html'), 'rb') as file:
     return web.Response(body=file.read(), content_type='text/html')


@asyncio.coroutine
def open_browser:
  webbrowser.get().open_new_tab('http://localhost:8090/')


if __name__ == '__main__':
  loop = asyncio.get_event_loop()           # boucle d'événement
  app = web.Application(loop=loop)          # initialisation
  app.router.add_route('GET', '/', index)     # routage
  app.on_startup.append(open_browser)       # ajouter aux tâches de démarrage
  web.run_app(app, host='localhost', port=8090) # démarrage
```

2.6.4. L'arduino

Comme il a été déjà expliqué en 2.3 l'arduino n'est pas pro-actif, c'est à dire qu'il ne répond qu'aux requêtes. Afin d'interroger le système d'exploitation sur les appareils connectés et d'établir une connexion sur un port série nous avons utilisé la librairie pyserial. Cette librairie dispose déjà d'une bonne compatiblité avec le modèle non bloquant proposé par asyncio grâce à la spécification unix : « tout est fichier ». Malheureusement, Windows ne dispose pas d'une telle normalisation des entrées / sorties, ce qui nous a forcé à accepter quelques exceptions à ce paradigme. Pour établir une connexion sur port série il faut coder les deux côtés : l'arduino et le routeur.

L'arduino dispose déjà des capacités à communiquer sur port série. La convention de la programmation pour l'arduino exigent l'implémentation de deux fonctions principales : setup et loop. La première initialise la connexion, la deuxième définie le comportement de l'application.
L'initialisation de la connexion est d'une durée non-prévisible. Il est donc crucial de déclencher un événement pour signaler à l'interlocuteur que l'arduino est désormais à l'écoute. Selon nos expériences ce temps est d'environ 2-3 secondes.
Dans notre cas la fonction loop peut être vide car il existe une troisième fonction serialEvent qui, par convention, sera appelée lors de l'arrivée des données sur le port série. C'est cette fonction là qui accepte des requêtes, désérialise leurs trams et dirige le traitement des valeurs retenues aux fonctions prévues à cet effet. Si besoin, elle va sérialiser les résultats et répondre à l'interlocuteur en retour.

```
void setup() {
 // en place : initialiser la connexion avec une fréquence (baudrate) prédéfinie
 Serial.begin(19200);
 // prêts : attendre jusqu'à la connexion soit établie
 while (!Serial) { }
 // partez : avertir l'interlocuteur
 Serial.print("ready");
}

void loop () {}

void serialEvent () {
 // [...]
}
```

Afin d'établir un dialogue entre l'arduino est le routeur il fallait mettre en place un protocole. Celui-ci possède deux possibilités d'échange:

1. Suite à une demande de l'utilisateur, le routeur va envoyer une nouvelle valeur pour la consigne pour le composant ESC, décrit dans la partie d'arduino. La valeur sera codé sur 16 bits. L'ordre d'octets suit un « boutisme » grand-boutien (de l'anglais « big-endian ») c'est à dire un ordre des octets dans lequel l'octet de poids le plus fort est enregistré à l'adresse mémoire la plus petite . La trame d'une consigne de 10% est la suivante :

0x00 (code fonction)	0x00 (hi)	0xA0 (lo)

Cette instruction n'exige pas de réponse du côté arduino.

2. Lors d'une session de capture le routeur envoie des requêtes afin de récupérer les valeurs des différents composant raccordés à l'arduino, notamment le courant, la tension et les poids dans un premier temps, puis dans un second temps la vitesse de rotation ainsi que toutes les valeurs pour les deux moteurs de la turbine - nous n'avons pas eu le temps ici de faire cette deuxième partie, ce sera pour la suite si le projet est repris. Les deux premières valeurs seront codées sur 10 bit, la dernière sera la valeur directe en gramme. Pour la requête il suffit donc de transmettre le code fonction sous le format suivant :

0x01 (code fonction)

La réponse a le format suivant :

0x01 fonction	0x0A (hi) tension	0x0A (lo) tension	0x0B (hi) courant	0x0B (lo) courant	0x0C (hi) poids	0x0C (lo) poids

La transformation des valeurs codées sur 10 bit est faite à l'aide des ses contraintes (valeurs minimale et maximale) comme suit :

```
def dixBitsToInteger (valeur, min, max):
  interval = max - min
  return min + valeur * interval / 1023
```

Voici un exemple du code arduino qui suit ce protocole :

```
void serialEvent () {
 // ici on a la garantie qu'il y a au moins un octet dans le buffer du port série
 switch (Serial.peek()) {


  case 0x00:
   // lecture bloquante jusqu'à ce que le tram soit complet
   byte requete[3];
```

```
    Serial.readBytes(requete, sizeof(requete));

    // mettre la nouvelle consigne
    consigne = int(requete[1]) << 8;
    consigne += int(requete[2]);
    break;


  case 0x01:
    // lecture d'un seul octet
    Serial.read();


    // répondre avec les valeurs récupérées
    byte reponse[7];
    traitement(reponse);
    Serial.write(reponse, sizeof(reponse));
    break;
 }
}
```

La partie correspondante en python est la suivante :

```
import serial, struct


# supposant que l'arduino soit connecté au port 'COM3'
connexion = serial.Serial('COM3', 19200)


# lecture bloquante du mot 'ready'
connexion.read(size=5)


# mettre consigne à 10%
instruction = bytes([0x00]) + struct.pack('>H', 10)
connexion.write(instruction)


# demander les valeurs actuelles
requete = bytes([0x01])
connexion.write(requete)


# recevoir les valeurs actuelles
reponse = connexion.read(size=7)
```

```
tension, courant, poids = struct.unpack('>HHH', reponse[1:]) # ignorer le code fonction


# affichager les valeurs actuelles
print('tension [V] :', dixBitsToInteger(tension, 0, 16))
print('courant [mA]:', courant)
print('poids   [g] :', poids)
```

2.6.5. Le routage

Afin de relayer les valeurs récupérées auprès de l'arduino à l'interface utilisateur il fallait mettre en place une connexion socket web. Le protocole de sockjs permet de normaliser une telle connexion avec des navigateurs qui ne supporte pas encore ou partiellement la technologie websocket. Ce protocol doit être respecter par les deux côtés, l'application principale et navigateur. Pour définir un « endpoint » sockjs on étend l'exemple vu en 2.6.4 :

```
import sockjs, json
# [...]


@asyncio.coroutine
def sockjs_handler (msg, session):
  # ignorer les messages de type (dé)connexion
  if msg.tp != sockjs.MSG_MESSAGE:
     return


  # désérialiser le message reçu
  requete = json.loads(msg.data)


  # réagir à la requête, par exemple :
  print(requete['nom'])


  # éventuellement répondre au client
  reponse = { 'msg': 'Bonjour %s !' % requete['nom'] }


  # sérialiser et envoyer la réponse
  session.send(json.dumps(reponse))
```

```
  # on peut également répondre à tous clients connectés
  # voilà la base d'une application chat !
  session.manager.broadcast(json.dumps(reponse))


if __name__ == '__main__':
  # créer l'application web
  # [...]
  sockjs.add_endpoint(app, sockjs_handler, prefix='/sockjs/')
  web.run_app(app, host='localhost', port=8090)
```

La partie correspondante sur le côté navigateur est la suivante :

```
import SockJS from 'sockjs-client'


// initialiser la connexion
const connexion = new SockJS('http://localhost:8090/sockjs/')


// souscrire à l'événement d'ouverture de la connexion
connexion.on_open = function () {
 const requete = { nom: 'Toto' }
 // serialiser et envoyer la requête
 connexion.send( JSON.stringify(requete) )
}


// souscrire à l'événement de l'arrivée d'un message
connexion.on_message = function (evt) {
 // deserialiser le message reçu
 const reponse = JSON.parse(evt.data)
 // afficher 'Bonjour Toto !'
 console.log(reponse.msg)
}
```

D'ici on peut facilement s'imaginer comment mettre en place communication bidirectionnelle entre l'arduino et le navigateur. La notion des souscriptions est particulièrement intéressante. Nous avons implémenté un système de channels sur le serveur. A chaque fois qu'un client demande la connexion d'un arduino raccordé à la machine du serveur, un channel est créé. Le navigateur peut souscrire à ce channel pour être tenu au courant de toutes nouvelles, notamment des données récupérées lors d'une session de capture.

2.6.6. Session de capture

Une session de capture est définie par l'utilisateur. Elle consiste en plusieurs étapes, chacune exigeant la valeur de la consigne pour le composant ESC ainsi qu'une durée. On pourra aussi choisir si l'on veut récupérer des données de l'arduino pendant cette étape. Une étape qui n'enregistre pas de valeurs est prévue lors des phases de décélération ou accélération de la turbine. L'utilisateur pourra choisir le pas d'interrogation de l'arduino en millisecondes. Voici un exemple de configuration au format json pour démarrer une session de capture :

```
{
 "hwid": "identifiant arduino",
 "pace": 100,
 "states": [{
  "consigne": 10,
  "duration": 5000
 }, {
  "consigne": 20,
  "duration": 5000
 }]
}
```

L'envoie de la trame ci-dessus au serveur démarre une session de capture dont l'essentiel est décrit par la coroutine suivante :

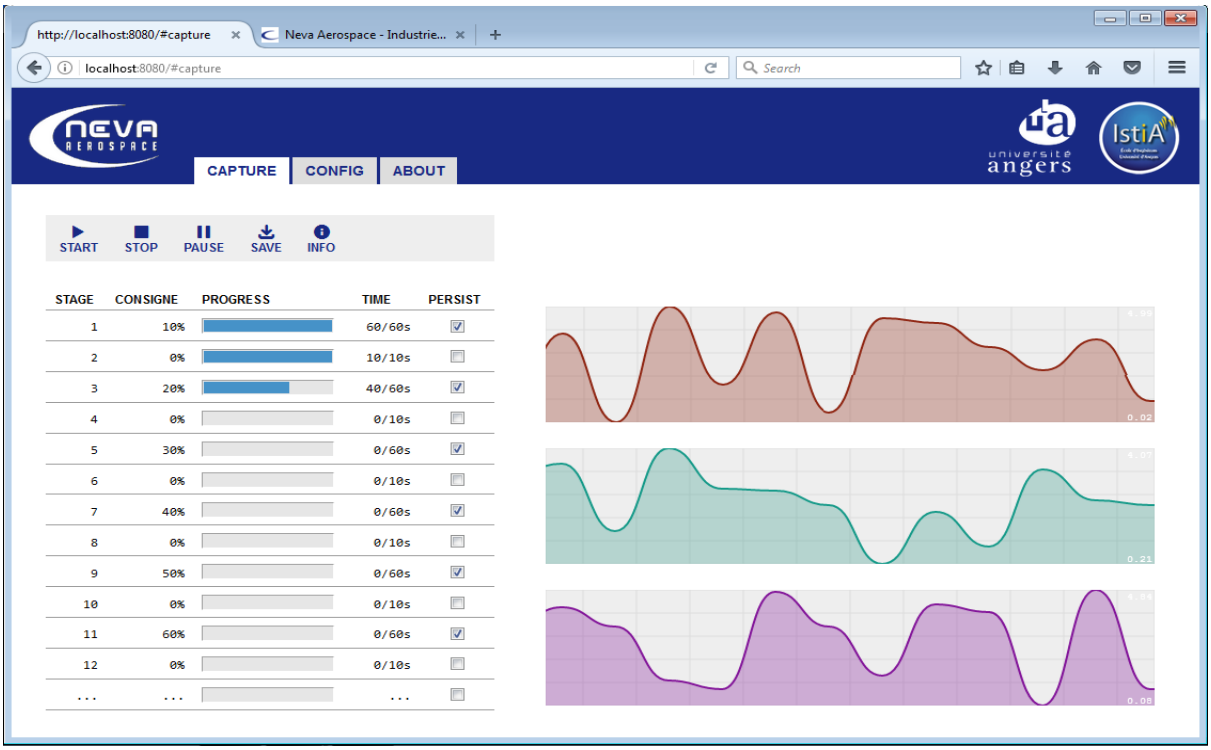
```
from datetime import datetime, timedelta
# [...]


@asyncio.coroutine
def capture_coro (service, config):
  # l'objet service dispose des connexion sur ports série et websocket
  hwid = config['hwid'] # identifiant arduino
  port = service.get_port(hwid)


  for state in config['states']:
     # envoyer la consigne de l'étape
     port.write(bytes([0x00]) + struct.pack('>H', state['consigne']))


     # initialiser la phase capture
     now = datetime.now()
     end = now + timedelta(milliseconds=state['duration'])
     while now < end:
        # demander, recevoir, deserialiser les valeurs
```

```
        port.write(bytes([0x01]))
        reponse = port.read(size=7)
        tension, courant, poids = struct.unpack('>HHH', reponse[:1])
        # publier les valeurs sur le channel de même nom
        # que l'identifiant de l'arduino associé.
        service.publish(hwid, {
            'datetime': datetime.now(),
            'tension': dixBitsToInteger(tension, 0, 16),
            'courant': courant,
            'poids': poids
        })
        yield from asyncio.sleep(config['pace'] / 1000)
        now = datetime.now()
```

Par manque de temps la partie graphique correspondante n'a pas été réalisée.

2.6.7. L'interface graphique

L'interface graphique est un site web et est donc réalisé avec les technologies standard du web : JavaScript, HTML et CSS. Pour normaliser le rendu par défaut des navigateurs nous nous sommes servi de la librairie normalizecss. Pour le montage des composants graphique, la récupération des valeurs d'entrées et la gestion d'événement utilisateur nous avons utilisé la librairie VueJS. Pour le rendu des graphes en temps réel nous avons utilisé la librairie smoothiejs. Même si la librairie webpack nous a permis d'accélérer notre travail (toute modification sauvegardée met à jour le navigateur immédiatement) nous n'avons malheureusement pas pu terminer ce travail par manque de temps. Voici une ébauche du résultat final attendu :



3 Technologies et outils

Pour le bon déroulement du projet nous nous sommes appuyés sur plusieurs types de technologies et d'outils, tant sur la partie électronique que la partie développement et organisation. Certains de ces outils étaient nouveaux pour nous, d'autres pas mais nous ne les avions utilisés qu'occasionnellement. Nous avons donc appris à les maîtriser tantôt grâce à l'aide disponible en ligne tantôt grâce à celle de notre responsable de projet.

3.1. Électronique

Figure 1 : logo de Eagle

Côté électronique nous avons eu besoin de concevoir un shield pour la carte Arduino. Le logiciel Eagle est destiné à la CAO (conception assistée par ordinateur) pour les circuit imprimés. Comprenant de nombreuses bibliothèques, Eagle permet de d'éditer les schéma électronique et d'effectuer les routages.

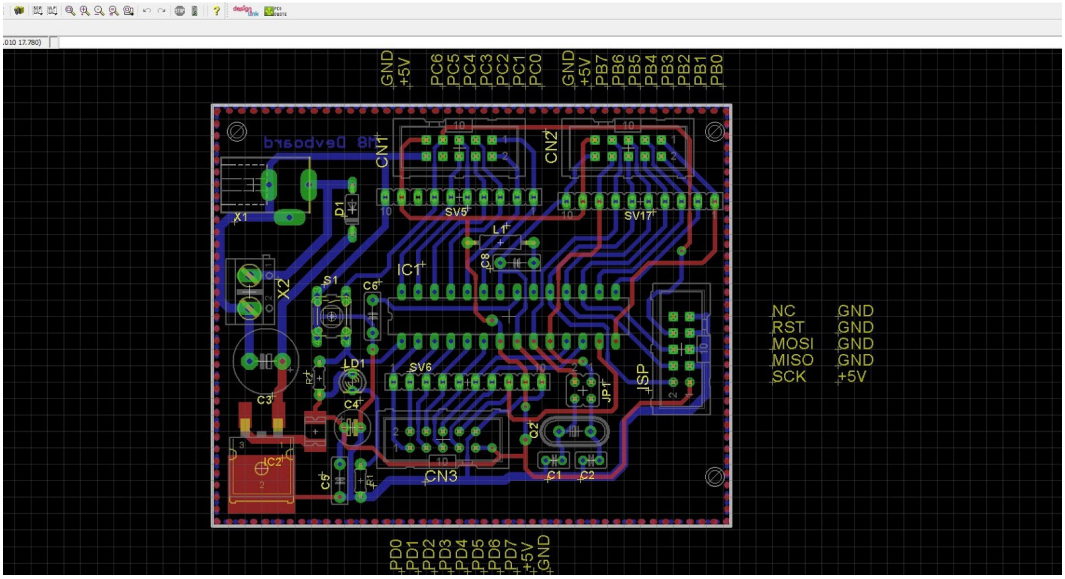


Figure 2 : interface routage Eagle

La figure 2 représente la partie routage de la conception d'un circuit imprimé. La principale difficulté se trouvait dans le choix de l'emplacement des composants. Une fois le routage terminé, les éléments du

circuit peuvent être transmis au graveur de circuits imprimés à l'aide du logiciel LPKF qui assure l'importation de n'importe quelle logiciel de conception de circuit imprimé.

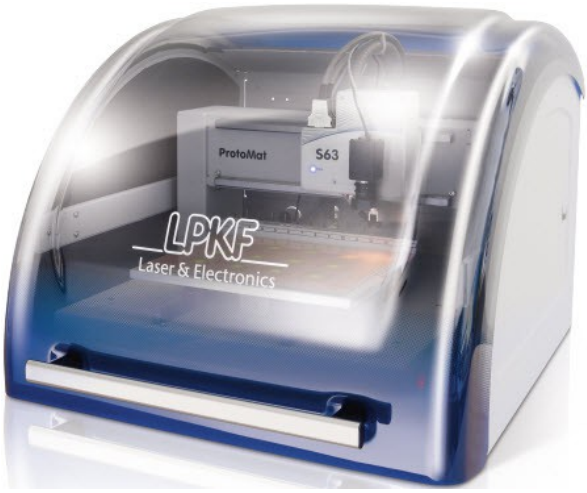


Figure 3 : Graveur LPKF ProtoMat S63

L'environnement de développement des cartes Arduino est une application permettant l'édition du code et la compilation de celui-ci. Le transfert du programme compilé se fait via la liaison série. Le langage de programmation s'écrit en C++ et la bibliothèque de développement est propre à l'architecture Arduino (gestion des entrées/sorties numérique et analogique).

Figure 4 : logo Arduino

Dans l'ensemble, la programmation sur Arduino est assez simple, de plus la communauté de développeurs autour d'Arduino est très grande, ce qui a grandement facilité nos travaux. La figure 5 représente un programme simple qui lit des trames numérique issues du port série.

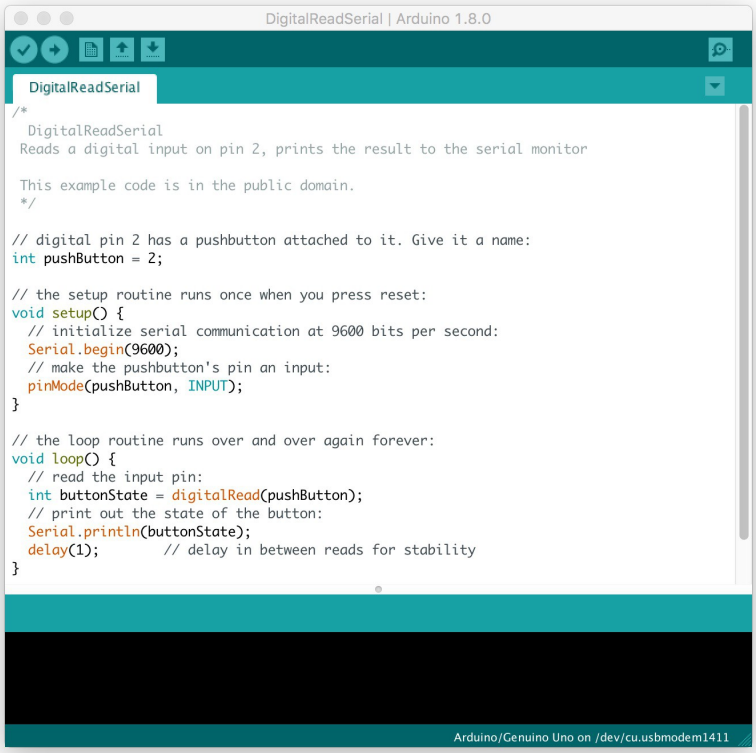


Figure 5: Exemple

3.2. Organisation et gestion de projet

Figure 6 : logo Smartsheet

Smartsheet : Logiciel de gestion de projet en ligne

Smartsheet est l'une des plateforme leader du SaaS (software as a service)pour la gestion et l'automation du travail collaboratif. Nous avons rapidement comparé plusieurs logiciels de gestion de projet avant de nous décider pour celui là en raison de sa facilité d'utilisation et de son rendu très professionnel.

Figure 7 : logo Github

Il s'agit d'un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. C'est le numéro un mondial. GitHub est centré vers l'aspect social du développement.

En plus d'offrir l'hébergement de projets avec Git, le site offre de nombreuses fonctionnalités habituellement retrouvées sur les réseaux sociaux comme les flux, la possibilité de suivre des personnes ou des projets ainsi que des graphes de réseaux pour les dépôts (en anglais repository).

GitHub offre aussi la possibilité de créer un wiki et une page web pour chaque dépôt. Le site offre aussi un logiciel de suivi de problèmes (de l'anglais issue tracking system).

GitHub propose aussi l'intégration d'un grand nombre de services externes, tels que l'intégration continue, la gestion de versions, badges, chat basés sur les projets, etc.

Figure 8 : logo de Git

Git : logiciel de gestion de versions, fondamental pour gérer un projet informatique.

3.3. Développement

Figure 9 : Pip

Pip : gestionnaire de paquets écrits en Python

L'un des avantages majeurs de pip est la facilité de son interface en ligne de commande, qui rend l'installation de paquets applicatifs Python aussi simple que de taper une commande.

III – Difficultés Rencontrées

Avec le recul, nous pouvons cibler plusieurs des difficultés rencontrées lors de ce projet :

Tout d'abord les problèmes techniques : (Problèmes qui auraient pu être résolus si l'on avait disposé de plus de temps pour se pencher dessus)

* Le fait que l'on ne récupère pas toutes les valeurs. On récupère bien le poids, mais pas la tension ni l'intensité de la turbine.

* Problème de connexion : Il n'y a qu'un seul port série côté interface tandis qu'on a deux appareils à connecter (la balance et l'arduino).

La simulation nous permet de résoudre d'un coup tout ces problèmes, c'est pourquoi c'est l'option que nous avons choisi pour donner une idée de l'application lors de la présentation.

Ensuite nous avons sans doute voulu faire trop bien, peut être un peu trop compliqué et du même coup un peu trop long pour les effectifs et délais dont nous disposions.

Enfin la principale difficulté de ce projet revient sur l'effectif. Le projet était réservé pour 4 ou 5 étudiants et nous étions seulement 3. Ce problème a donc eu un lourd impact sur le déroulement du planning où quelques tâches ont été difficiles à traiter. Malgré une bonne organisation du projet, de nombreuse tâches ont été plus longues à traiter que prévu.

IV – Perspectives

A la fin du temps imparti, notre projet comportait des éléments incomplets tels que la commande avec l'ESC permettant de commander le moteur avec la carte Arduino, la communication Arduino/Interface utilisateur encore perfectible. Avec tous les éléments terminés, notre projet aurait pu transiter vers une phase orientée industrielle, dans le sens que notre système serait porté vers une approche sous LabView de National Instrument.

Nous n'avons pas réussi à mettre la consigne à un pourcentage. On arrive à la mettre à 0, à 1 aussi mais pas entre les deux.

La communication série Interface-Arduino est encore incomplète.

Conclusion

En choisissant ce projet, après avoir parcouru le cahier des charges, nous savions qu'il demanderait beaucoup d'investissement. Au final, nous sommes un peu déçu de n'avoir pas pu terminer réellement le projet et de ne pas pouvoir présenter de résultats complets.

Notamment dans le sens où une fois la période de réalisation du projet finie, beaucoup d'éléments n'ont pas pu être développés et de problèmes résolus par manque de temps. On ne peut pas vraiment considérer notre système comme étant fonctionnel. C'est un peu frustrant.

En dépit des difficultés rencontrées, ce projet de six mois nous a beaucoup apporté sur différents plans, techniques bien sûr mais aussi professionnel et humain.

Ce projet était très riche au point de vue technique, non seulement parce qu'il nous a fait réutiliser toutes sortes de connaissances et méthodes vues en cours mais aussi parce qu'il nous en a fait découvrir et mettre en application plein d'autres. Autant dire que le bilan des apports personnels de ce type de projet est très positif.

Il était aussi intéressant de voir les contraintes du monde professionnel (bien qu'elles aient été ici légères) telles que le soucis d'avoir quelque chose de concret et visuel pour expliquer où l'on en est dans le projet, trouver des façons de rendre compte de l'avancement, faire des démonstrations, et s'adapter pour que le travail soit compréhensible par le plus de monde possible.

Nous avons également eu beaucoup de plaisir à travailler avec notre responsable de projet qui s'est montré très disponible.

Bibliographie

Documents

AutomatisationBancTestTurbines.docx

TravailDemande.pdf

docAduino6.pdf

Documentation ACS713.pdf

Sites Web

www.neva-aero.com

https://fr.smartsheet.com/

https://pypi.python.org/pypi/pip

https://git-scm.com/

https://github.com/

Annexes

Annexe 1 : Cahier des charges

Automatisation du banc de mesures de Turbines

Table des matières

1. Présentation du banc

Le banc est composé d'un structure en L permettant un renvoie de force.
La turbine est placé en haut du banc le sens des flux en horizontal, une balance est placé sous la partie horizontale telle que la distance entre le centre de poussée de la turbine et l'axe de rotation de la structure soit égale à la distance entre l'axe de rotation de la structure et le point d'appui de la structure sur la balance. (cf. image 1)
L'alimentation électrique est effectuée par une ou plusieurs alimentations stabilisées.
La commande des contrôleurs moteur (ESC) est assuré par un radio commande (RC)

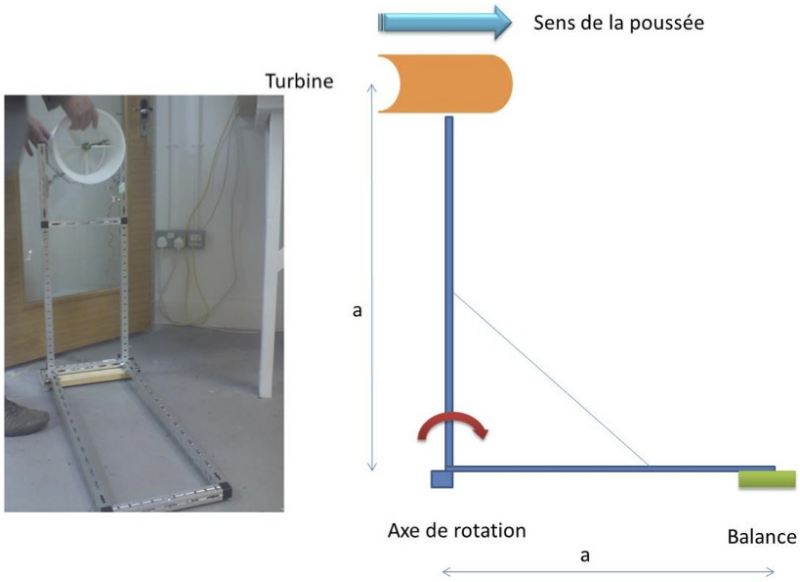


Figure 1 : schéma simplifier du banc

2. Méthode actuelle :

L'objectif dans chaque campagne de test est d'établir les courbes caractéristiques d'une turbine :

- Poussée /Puissance
- Poussée / Efficacité
- Poussée / Vitesse de rotation

La procédure actuelle est la suivante :

- Tension à 6.5volts
- Signal pwm à fond pendant 5secondes (consigne d'ESC à 100% de sa valeur maximale)
- Prise de la poussée, du couple tension/courant et de la vitesse de rotation de chaque moteur
- Attente de l'ordre de 1 minute
- Tension à 7 volts
- Signal pwm à fond pendant 5secondes
- Prise de la poussée, du couple tension/courant et de la vitesse de rotation de chaque moteur
- Attente de l'ordre de 1minute
- Idem en incrémentant de 1 volt jusqu'à 16 volts

Les mesures sont répétées entre 5 et 10 fois en fonction de l'instabilité

3. 1ère approche d'automatisation : cycle de prise de mesures + post traitement, tension fixe, consigne d'ESC variable

La première approche d'automatisation qui était pressentie était la suivante :

- Consigne d'ESC à 10% de sa valeur maximale
- Montée en puissance rapide puis maintient pendant 5 secondes, élément déclencheur : appui sur un bouton
- Prise de mesure avec affichage sur un écran de control + enregistrement des données de poussée, puissance, tension, courant, vitesse de rotation associées (sur carte SD), prise de mesure tous les 0.1 secondes + rafraichissement de l'écran
- Arrêt
- Reprise du cycle au bout d'une minute (à préciser) et ce 5 à 10 fois (à définir)
- Augmentation de la valeur de consigne de l'ESC de 10%
- Reprise du cycle et ce jusqu' à ce que la consigne d'ESC soit à 100% de sa valeur.
- A tout moment, arrêt des moteurs rapide par bouton d'arrêt d'urgence

Une fois les mesures terminées, les données sont injectées et traitées via un ordinateur pour en ressortir les courbes

Traitement :
Pour l'ensemble des mesures :

- Courbe poussée/puissance globale
- Courbe poussée globale/puissance par moteur (si 2 moteurs)
- Courbe d'efficacité
- Courbe poussée globale / vitesse de rotation par moteur (si 2 moteurs)

Ce premier niveau d'automatisation correspond au schéma figure 2 :

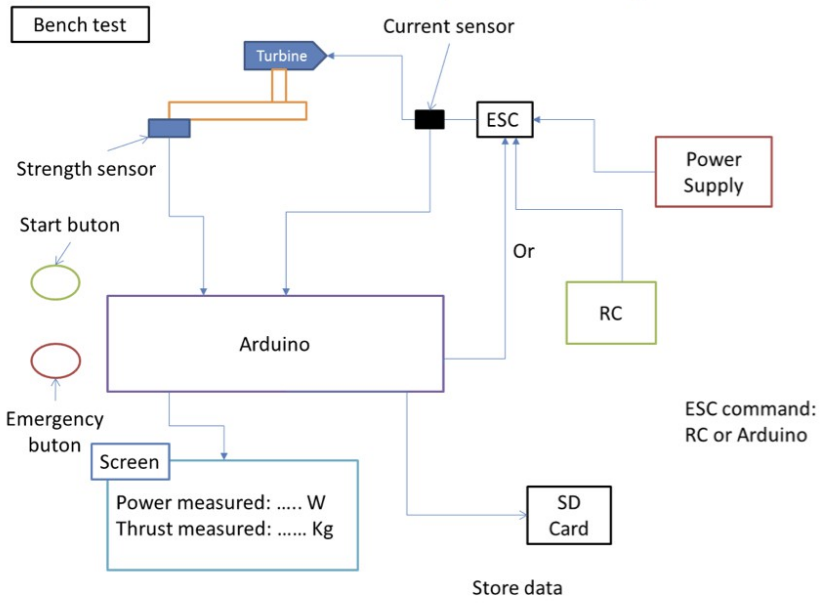


Figure 2 : 1ère approche d'automatisation

4. 2ème approche d'automatisation : cycle de prise de mesures + post traitement, caractérisation « rapide » d'une turbine, construction des courbes « à la volée », tension fixe, consigne d'ESC variable :

La 2ème approche d'automatisation est celle qui fera l'objet du projet d'automatisation du banc de test.

L'automatisation donnera le choix entre 2 modes :

- Cycle complet
- Caractérisation rapide

Cycle complet :

- Reprends entièrement la 1ère approche en liaison avec un ordinateur sur lequel est affiché l'ensemble des valeurs
- Commande marche / arrêt à l'écran
- Construction des courbes de mesure en cours en direct
- Construction des courbes de caractérisations en fin de cycle avec sauvegarde
- Bouton d'arrêt d'urgence physique

Caractérisation rapide :

- Consigne d'ESC à 0% de sa puissance maximale
- La consigne d'ESC répond à une rampe d'accélération de 0 à 100%
- Prise des valeurs de poussée, puissance, tension, courant et rotation des moteurs, la fréquence de prise de mesure dépends de la rampe d'accélération (nombre de mesures à définir)
- Affichage des valeurs à l'écran en direct
- Construction des courbes de caractérisation en direct avec sauvegarde
- Bouton d'arrêt d'urgence physique

Dans les 2 cas :

- Paramétrage des différentes variables liés aux essais via l'écran (tension, temps d'attente, rampe d'accélération, ...)
- Sauvegarde des paramètres dans les logs de données

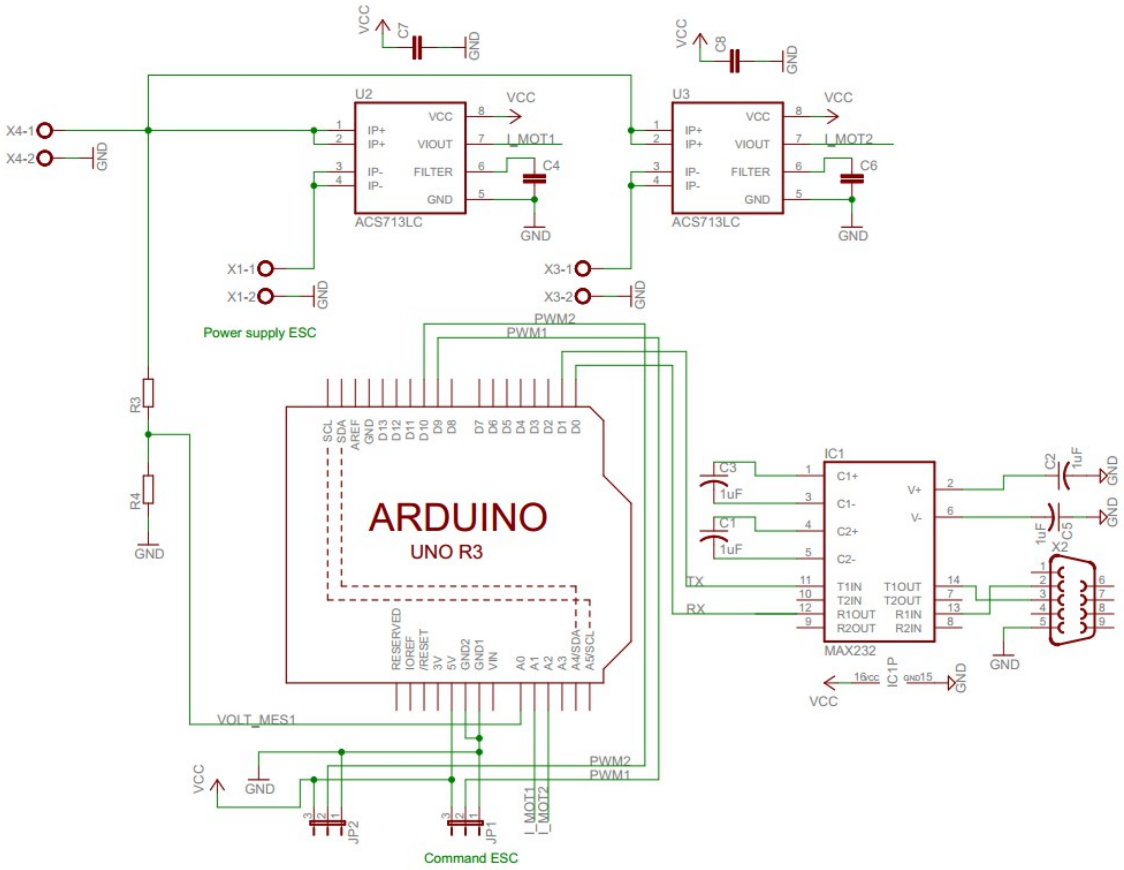
5. Portage du protypage arduino sur Labview

Reprise de la 2ème approche avec LabView

Annexe 2 : Task List (pour Diagramme de Gantt)

Project details
Product : Bench test for drone turbines
Team : Julien MONNIER - Elisa ROSSE - Marius RUNGE
Supervisor : Frank Mercier
Liste des tâches
Arduino
recherche basique
test basique(allumer une LED sur arduino)
evaluation du fonctonnement des périphériques
gestion ESC (controlleur) +turbines
ARU physique (réél pas le virtuel)
Mesurer le courant
Récupérer les donnéés d'échantillonage
Spécification du protocole (des trames) A-Commandes B-Données
Commande du module ESC en fonction du courant mesuré (adaptabilité)*
Implémentation du service avec Zeromq
Interface Utilisateur
Recherche basique : librairesIU cross platforme?
Générateur de données arbitraires temporaire
Créer un IU basique
Dessiner une courbe plot
Dessiner une courbe plot glissante(en temps réél)
Bouton ARU interface (virtuel)
Mettre en place la consommation du service (Zeromq)
Enregistrement des données pour un intervale de temps choisi
Packager l'application sous format executable (.exe)
Affiner/améliorer/faire claquer/ l'interface utilisateur
Faire le manuel utilisateur
Livrable
Inclure le programme arduino dans le .exe
Screen Shot pour le manuel/rapport/poster...
Dessin Poster
Rapport écrit en fonction du cahier des charges
chapitre modele miniature -> industrie réélle
chapitre explications/mode d'emploi
chapitre feuille de routes/ ouvertures / voie à approfondir
Traduction en anglais

annexe 3 : Arduino & shield Arduino



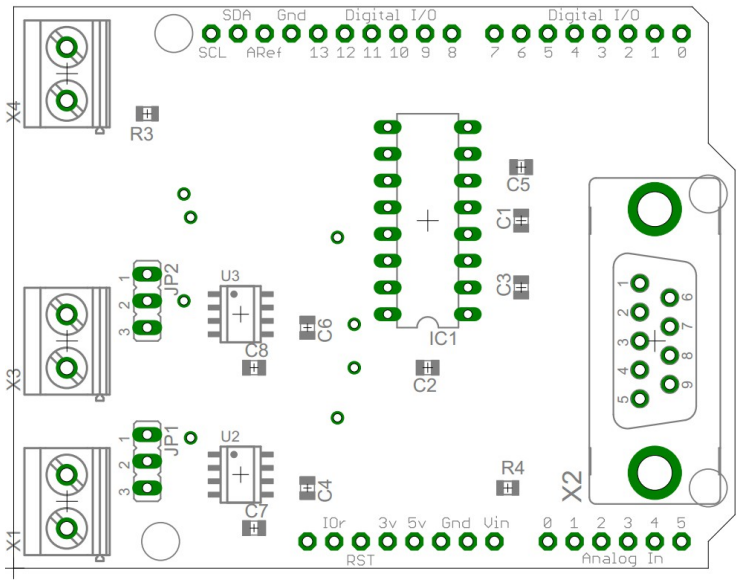


Table des matières

RÉSUMÉ

Ce rapport présente un projet d'automatisation du banc de test de turbines destinées au vol vertical réalisé par des étudiants de l'ISTIA (Institut des Sciences et Techniques de l'Ingénieur d'Angers) pour l'entreprise de conception de drones de transport Neva Aerospace.

Ce projet, son contexte, ses objectifs, missions, leur réalisation ainsi que ses résultats et livrables sont détaillés dans ce rapport à travers quatre grandes phases : la préparation du projet, sa réalisation, les difficultées rencontrées ainsi que le ressenti de chacun, et enfin les livrables et résultats.
La préparation du projet comprend notamment sa gestion ainsi que la répartition des tâches en fonction du temps et des disponibilités de chacun (Diagramme de Gantt).
La réalisation se constitue de principalement de la partie Arduino et de l'implémentation de l'interface utilisateur.

mots-clés : drone – turbines – vol vertical – Neva Aerospace – ISTIA – Arduino – interface utilisateur

ABSTRACT

This report presents a project to automate the vertical flight turbine's test bench carried out by ISTIA students (Institute of Sciences and Techniques of the Engineer of Angers) for the company of design of transport drones Neva Aerospace.

This project, its context, its objectives, its missions, its implementation and its results and deliverables are detailed in this report in four main phases: the preparation of the project, its realization, the difficulties encountered and the feelings of each one, and finally deliverables and results.
The preparation of the project includes the division of tasks according to the time and availability of each (Gantt chart).
The realization consists mainly of the Arduino part and the implementation of the user interface.

keywords : drone – turbine – Neva Aerospace – ISTIA – Arduino – User Interface – vertical flight

ISTIA
62 Avenue Notre-Dame du Lac
49000 Angers cedex
Tél. 02 44 68 75 00 | Fax 02 44 68 75 01

université angers

ENGAGEMENT DE NON PLAGIAT

Nous, soussignés Julien MONNIER, Élisa ROSSE, Marius RUNGE
déclarons être pleinement conscients que le plagiat de documents ou d'une partie d'un document publiée sur toutes formes de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.
En conséquence, nous nous engageons à citer toutes les sources que nous avons utilisées pour écrire ce rapport ou mémoire.

signé par les étudiants le 24 / 04 / 2017

Cet engagement de non plagiat doit être signé et joint à tous les rapports, dossiers, mémoires.

ISTIA
62 Avenue Notre-Dame du Lac
49000 Angers cedex
Tél. 02 44 68 75 00 | Fax 02 44 68 75 01