



Dossier technique du projet

Suivi-Personnel

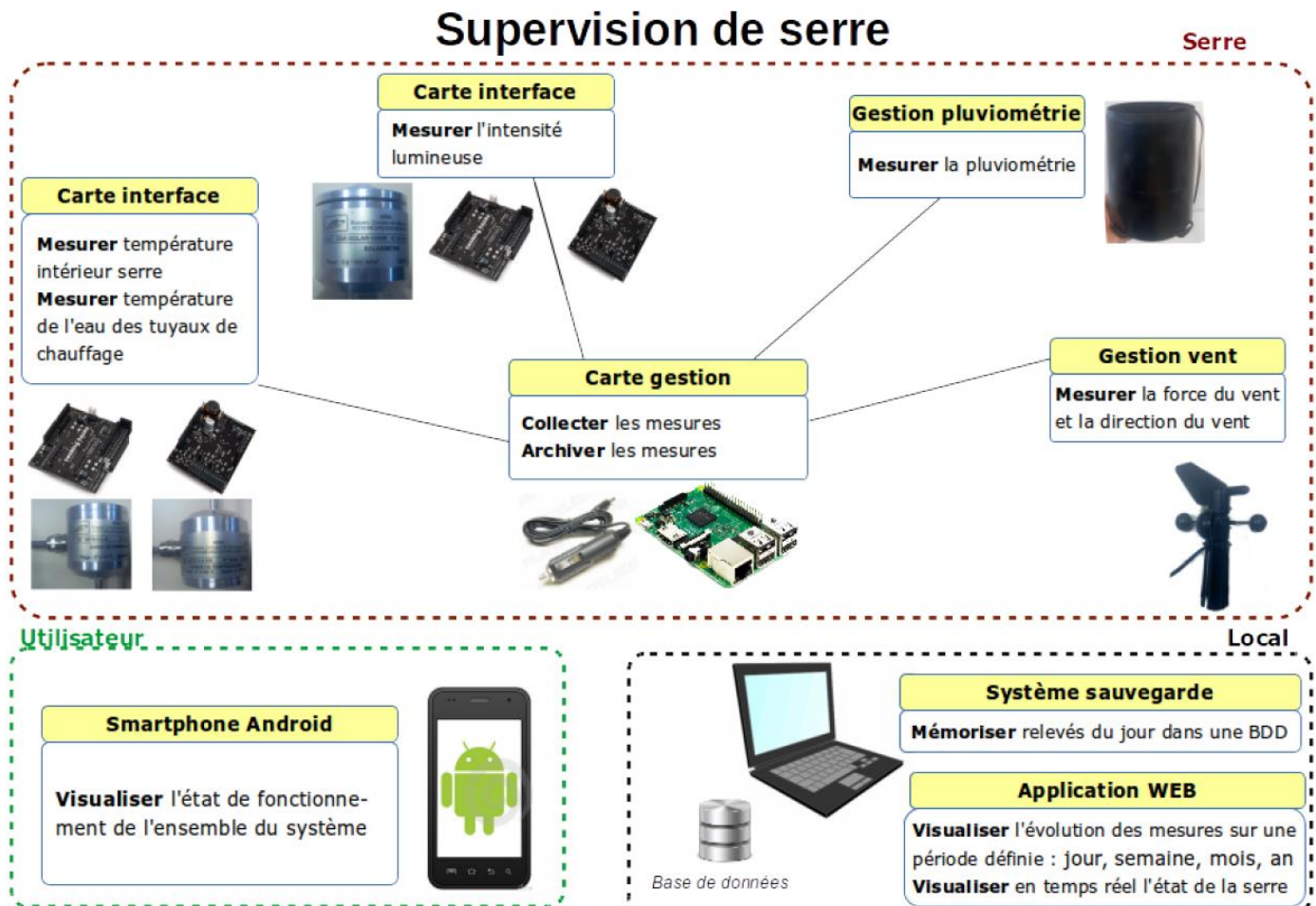
Etudiant 1

Table des matières

I.	Situation dans le projet.....	3
1.1)	Synoptique de la réalisation	3
1.2)	Rappel des tâches de l'étudiant	3
1.3)	Contraintes liées au développement	4
1.4)	Problème matériel.....	4
II.	Conception et mise en œuvre.....	5
2.1)	Fonctionnement de l'anémomètre	5
2.2)	Fonctionnement de l'Arduino.....	7
2.3)	Fonctionnement de la Raspberry	9
2.4)	Réalisation du diagramme de classe.....	10
III.	Récupération des mesures avec l'Arduino	11
3.1)	Programme mis en place sur l'Arduino.....	11
3.2)	Test du pluviomètre et résultat	13
3.3)	Test de l'anémomètre et résultat	16
3.4)	Création des classes avec Visual Studio.....	16
IV.	Utilisation de la carte Raspberry.....	18
4.1)	Envoie des données Arduino – Raspberry	18
4.2)	Programme pour la connexion à la base de données.....	20
4.3)	Requêtes Sql pour la base de données	Erreur ! Signet non défini.
4.4)	Création des classes en Python.....	21
4.4.1)	Etude des différentes classes du programme python	22
4.4.2)	Etude des différents cas d'utilisation	26
V.	Test Unitaire	27
V.	Conclusion.....	30

I. Situation dans le projet

1.1) Synoptique de la réalisation



1.2) Rappel des tâches de l'étudiant

Dans ce projet j'ai eu pour tâche de mettre en place l'anémomètre pour mesurer la vitesse et la direction du vent au niveau de la serre. Je devais aussi m'occuper de la partie Raspberry où je devais donc pouvoir collecter toutes les mesures pour pouvoir ensuite les utiliser ultérieurement.

Dans cette partie l'utilisateur pourra définir la période de relever des mesures et il aura également la possibilité de rajouter des nouveaux éléments de la base de données.

1.3) Contraintes liées au développement

Dans un premier temps, nous avons une **contrainte financière**. Nous avons donc un budget alloué de 100 euros. Et pour terminer, nous avons plusieurs **contraintes de qualité**.

La première est une contrainte d'évolutivité forte, ainsi, lorsque l'utilisateur voudra ajouter un capteur, ou une mesure, le travail à réaliser de son côté doit-être minime, voir automatique.

Une documentation complète sur le système doit être fournie au client, pour qu'une fois le projet terminer, une autre équipe que l'équipe d'étudiant puisse donner suite à ce projet.

1.4) Problème matériel

Lors de tests je me suis rendu compte que les valeurs retournées par l'anémomètre concernant la direction du vent étaient erronées.

Avec un multimètre on a vérifié les valeurs de tensions en faisant bouger l'anémomètre et on s'est aperçu qu'il n'était plus fonctionnel.

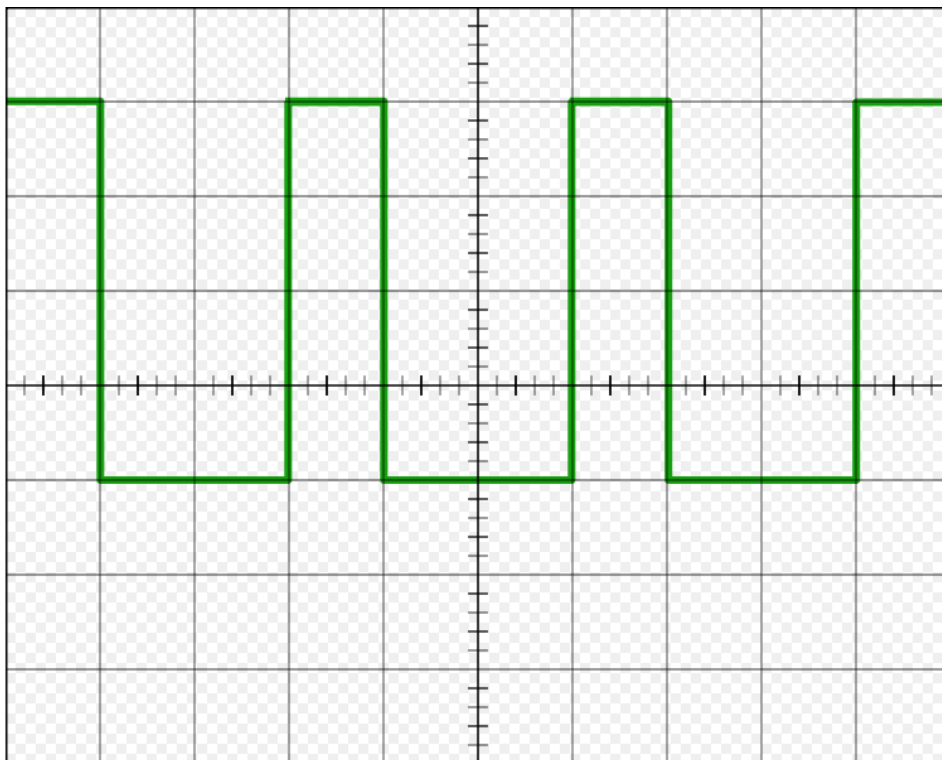
Pour résoudre ce problème j'ai donc dû utiliser un potentiomètre en remplacement car il a les mêmes fonctionnalités.

II. Conception et mise en œuvre

2.1) Fonctionnement de l'anémomètre

Pour l'anémomètre il y a donc deux capteurs, le premier sert à mesurer la vitesse du vent et le second sert à mesurer la direction du vent.

Pour la vitesse du vent c'est sous forme d'impulsion, il a un aimant au niveau de la girouette de l'anémomètre et à chaque tour il y a une impulsion. On regarde donc le nombre de tours que la girouette a effectué suivant la période donnée et cela nous donnera donc la vitesse du vent à ce moment-là.

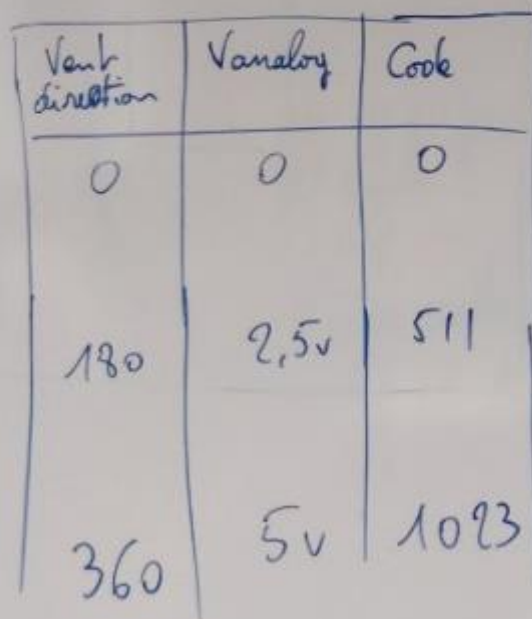


L'anémomètre fonctionnant de 0 à 5V la tension aura cette forme-là, c'est-à-dire qu'à chaque impulsion la tension passera à 5V comme dans l'image ci-dessus.

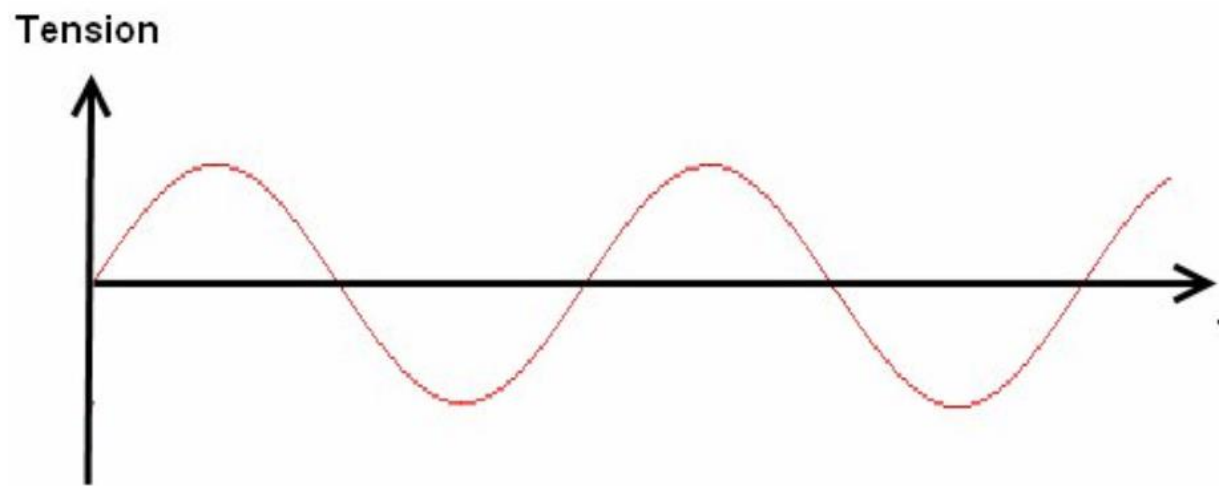
Pour le capteur de direction, on utilisera le CAN (Convertisseur analogique numérique).

Le capteur fonctionne avec une tension de 0 à 5V. Le CAN convertit cette tension en un code allant de 0 à 1023 car le CAN de l'Arduino est à 10 bits et ce sont donc ces valeurs qu'il faudra interpréter en tant que direction allant de 0 à 360 degrés.

Voilà donc un exemple du fonctionnement du CAN.



Vent direction	Vanalog	Code
0	0	0
180	2,5v	511
360	5v	1023



Pour la direction du vent la tension aura donc une forme sinusoïdale qui variera en fonction de la direction

2.2) Fonctionnement de l'Arduino

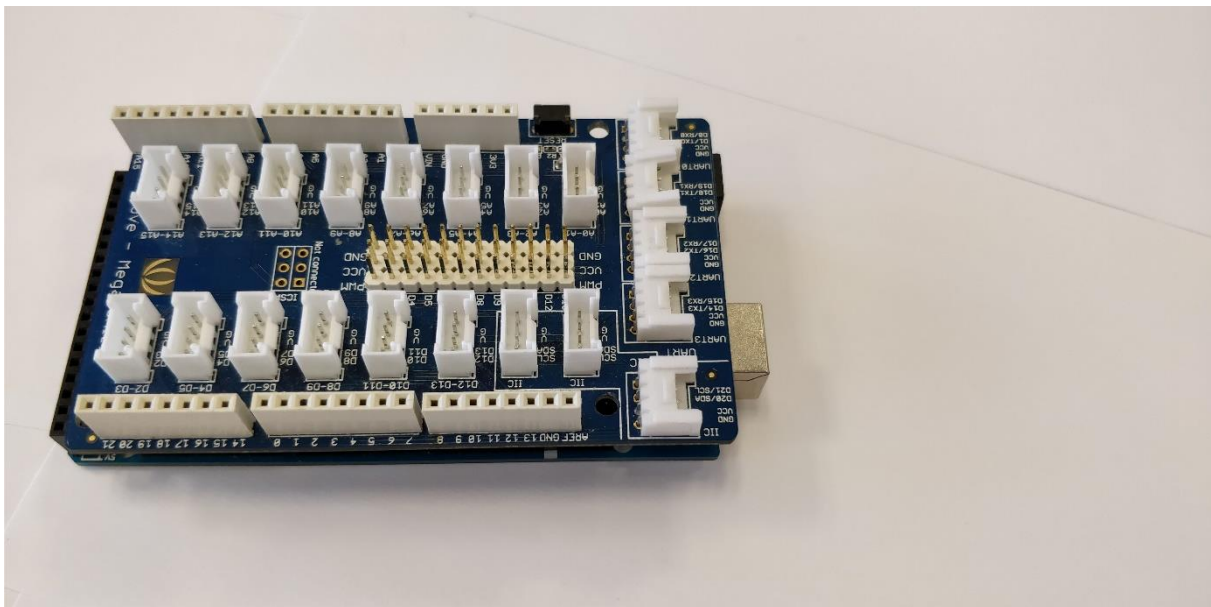
Pour la partie Arduino je devais donc écrire un programme commun pour les différents capteurs que je devais connecter sur la carte. Il y avait donc le pluviomètre et l'anémomètre qui compte pour deux capteurs.

Pour l'Arduino j'utilise donc deux ports :

- Le port analogique pour la direction du vent qui va donc utiliser le CAN
- Le port digital pour la vitesse du vent et pour le pluviomètre qui va donc utiliser des impulsions.

Ce programme devait donc tourner en boucle sur l'Arduino pour pouvoir ensuite le transférer sur la Raspberry et récupérer les valeurs attendues et les traiter pour les envoyer à la base de données sous forme de requêtes (détaillée dans la partie 4.3).

Pour se faire j'ai également dû créer des classes en python pour pouvoir me connecter à la base de données et aussi pouvoir faire mes requêtes correctement avec les mesures envoyées par la carte Arduino.

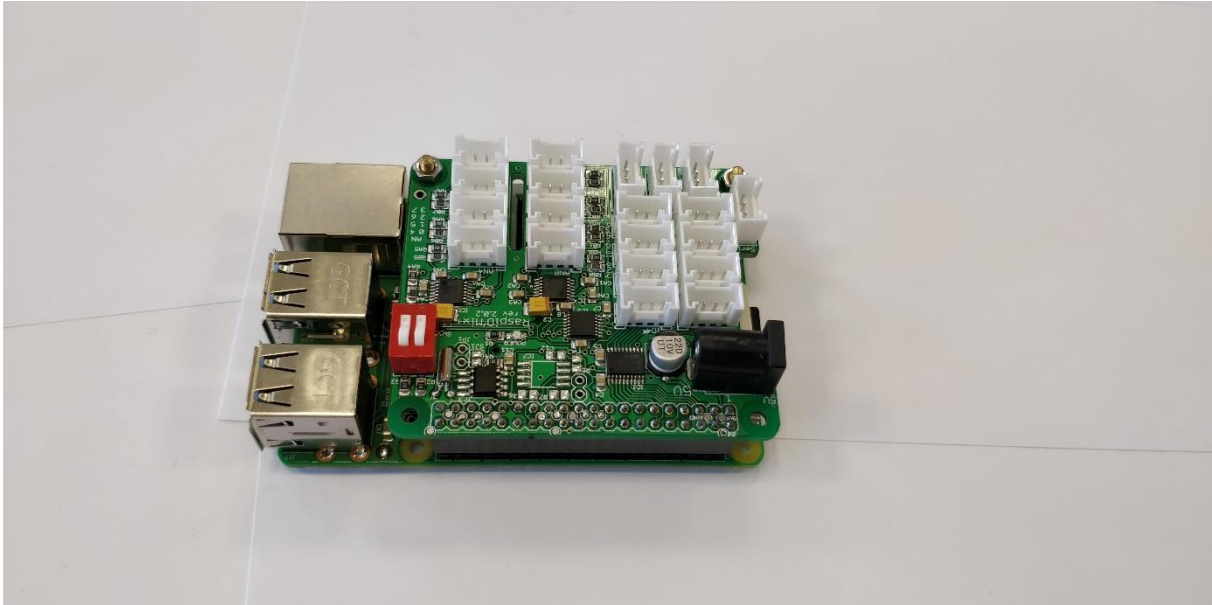


Pour me simplifier la tâche j'ai donc créé des classes en c++ avec Visual studio car je pouvais utiliser facilement les bibliothèques de l'Arduino. Je n'avais plus qu'à coder les classes en c++ avec ces bibliothèques et lancer un Main beaucoup plus propre et lisible pour l'utilisateur

2.3) Fonctionnement de la Raspberry

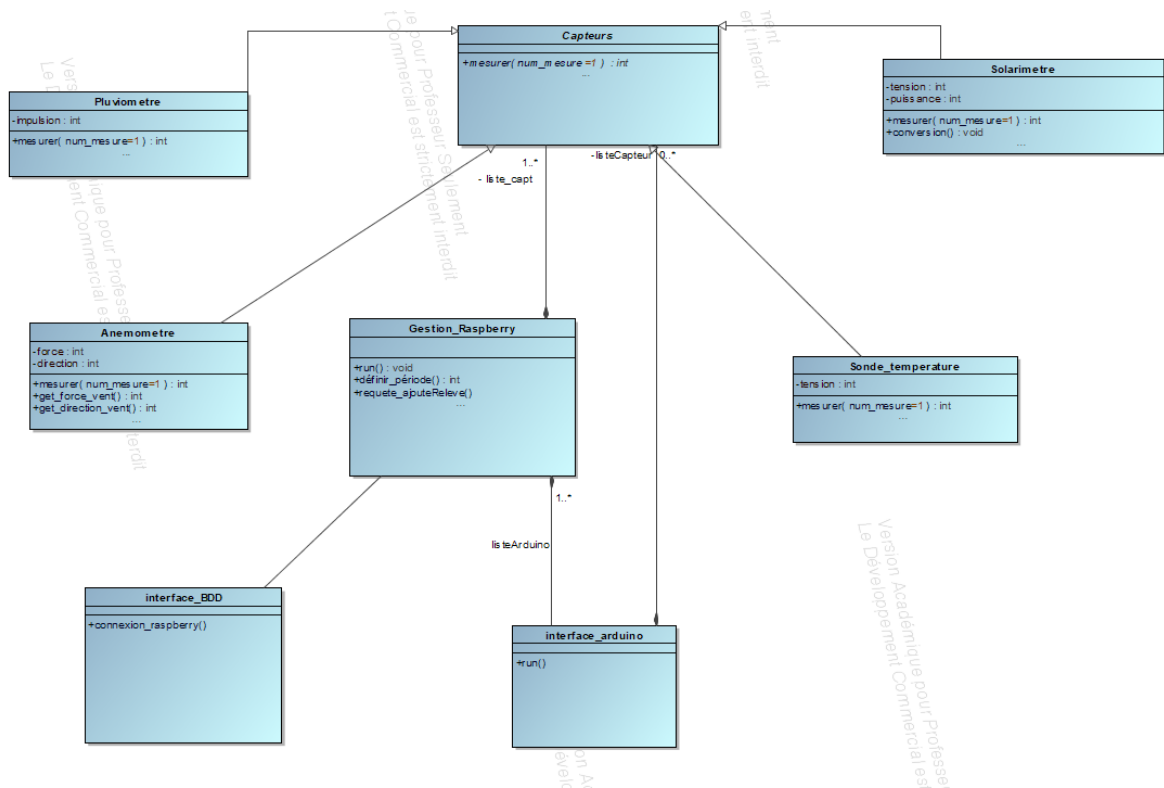
Cette partie Raspberry sera en quelques mots le cœur de ce projet car sans la Raspberry il n'y aura donc aucun moyen de rassembler toutes les parties du projet.

Pour cette partie la tâche était de récupérer les mesures envoyées depuis la carte Arduino et les envoyer à la base de données.



Voici donc la Raspberry utilisé pour notre projet.

2.4) Réalisation du diagramme de classe



III. Récupération des mesures avec l'Arduino

3.1) Test de l'anémomètre

Pour récupérer la mesure de la vitesse du vent j'ai donc écrit un code sur l'Arduino qui compte le nombre de tours effectuées sur une période de 3 sec et nous envoie donc l'équivalent en km/h. De plus j'ai fait un tableau avec les forces du vent associée aux vitesses correspondantes pour que l'utilisateur puisse avoir un aperçu de la force s'il est en intérieure. J'ai donc fait des tests avec mon programme et les résultats étaient conforme à ce que j'attendais.

Pour mesurer la direction du vent j'ai donc écrit un programme qui grâce à la librairie Arduino peut convertir le code reçu (allant de 0 à 1023) en degré (0 à 360). Par exemple si le code est de 0 cela correspond au degré 0 et a la direction Nord.

J'ai également fait un tableau avec tous les degrés et leurs correspondances avec les points cardinaux.

C'est donc en testant mon programme sur l'Arduino que j'ai pu déceler un défaut de fonctionnement de l'anémomètre pour la partie direction du vent.

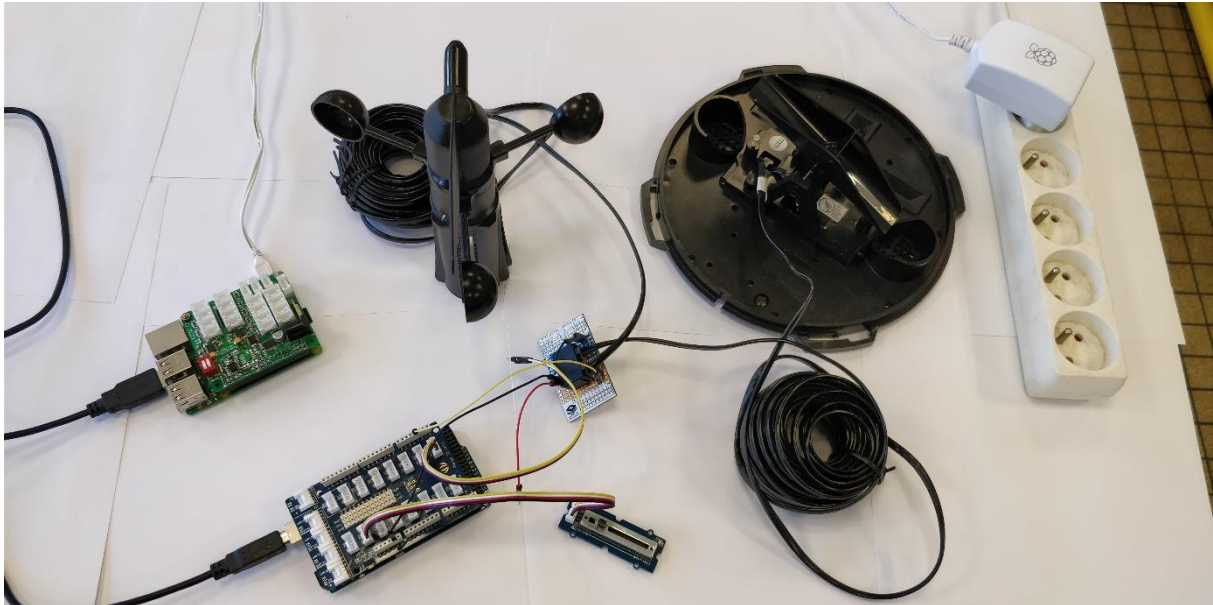
Direction du vent	Degré
Nord	0, 360°
Nord-Est	22,5°
Est	90
Sud-Est	135
Sud	180
Sud-Ouest	225
Ouest	270
Nord-Ouest	315

Force du vent	Km/h
Calme	Moins de 1
Très légère brise	1 à 5
Légère brise	6 à 11
Petite brise	12 à 19
Jolie brise	20 à 28
Bonne brise	29 à 38
Vent frais	39 à 49
Grand frais	50 à 61
Coup de vent	62 à 74
Fort coup de vent	75 à 88
Tempête	89 à 102
Violente tempête	103 à 117
Bombe météorologique	+ de 118

Voici donc les tableaux récapitulatifs pour aider l'utilisateur à mieux comprendre les données.

3.2) Test du code commun et résultat

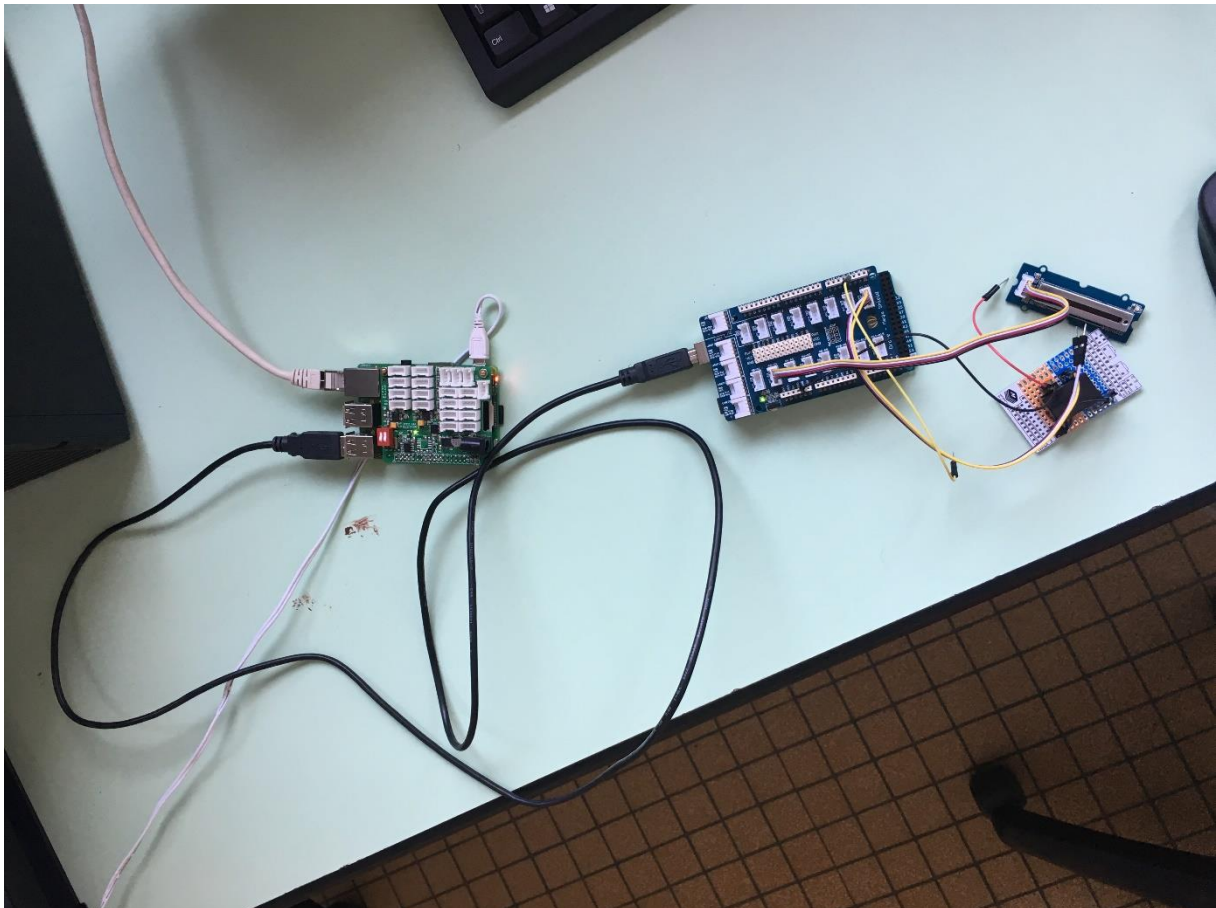
Après avoir tester les capteurs un par un je me suis penché sur le code commun qui me permettra de relever les mesures de chaque capteur en une seule fois.



J'ai donc dû définir une période pour effectuer ces relevés et éviter que les mesures arrivent en désordre ou bien que les mesures soient fausses.

J'ai aussi récupéré le code pour le pluviomètre et j'ai commencé à tout rassembler pour ensuite lancer le programme commun que voici ci-dessous.

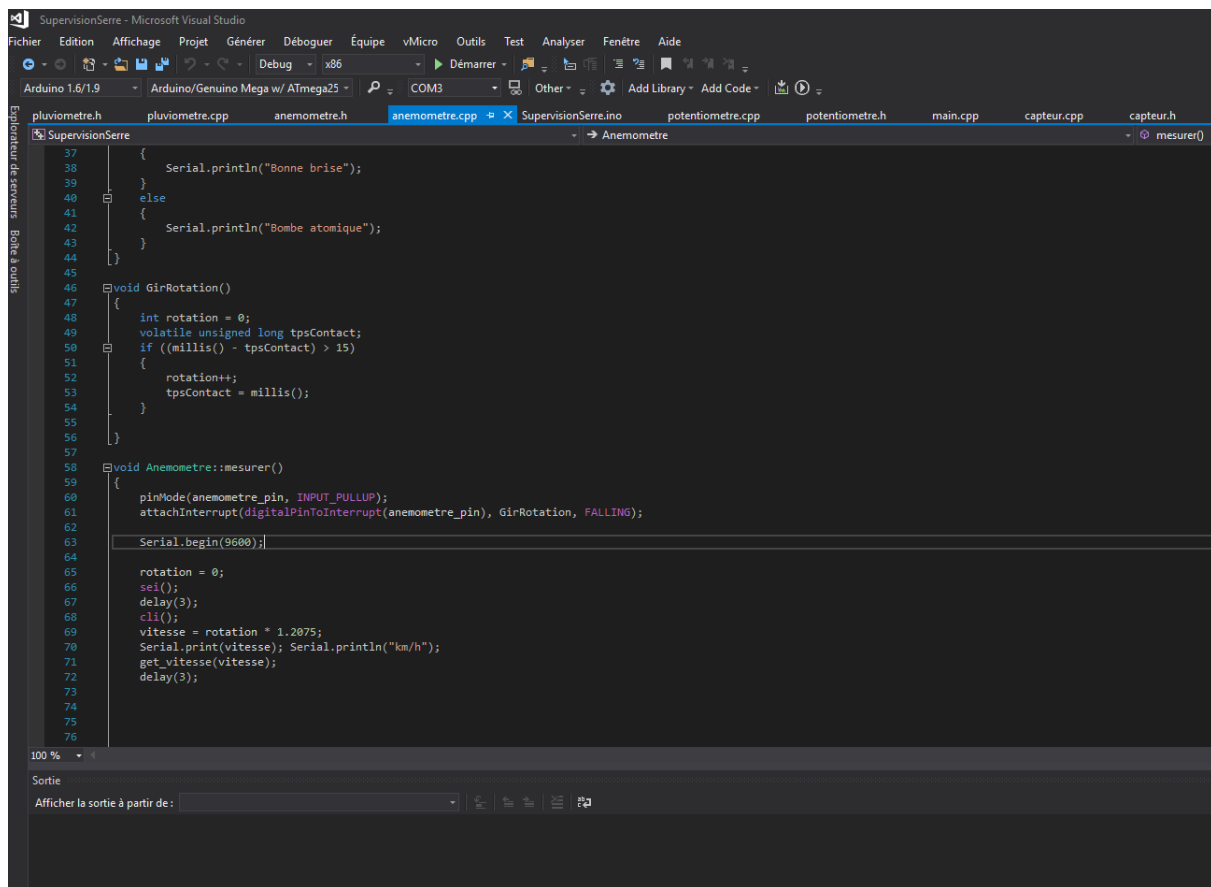
J'ai fait deux versions de ce programme, une pour être envoyé sur la Raspberry et ne relever que les mesures et une autre pour pouvoir exploiter les mesures collecter dire si le vent souffle fort ou bien indiquer sa direction (Nord, Sud...)



Nous pouvons donc voir ci-dessous les résultats obtenus après l'exécution du code commun.

Davis Wind Speed Test				
Km/h	Indication	Degre	Direction	Fluie
0.00	Vent Calme	68°	Est	
0.00	Vent Calme	68°	Est	
				goutte
				goutte
				goutte
0.00	Vent Calme	68°	Est	
				goutte
0.00	Vent Calme	68°	Est	
				goutte
0.00	Vent Calme	68°	Est	
0.00	Vent Calme	68°	Est	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	
0.00	Vent Calme	0°	Nord	

3.3) Création des classes avec Visual Studio

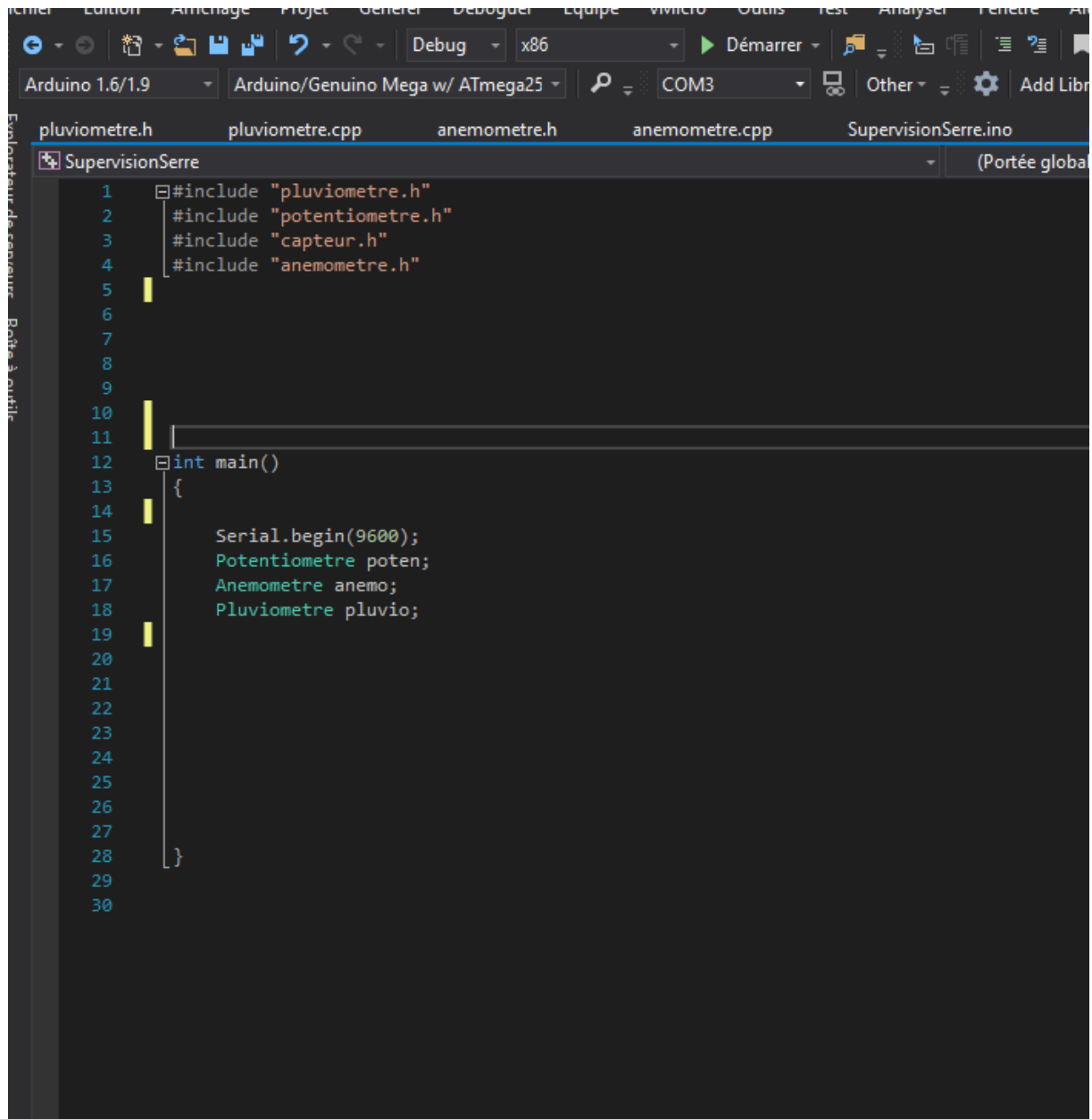


```
SupervisionSerre - Microsoft Visual Studio
Fichier Edition Affichage Projet Générer Débugger Équipe vMicro Outils Test Analyser Fenêtre Aide
Debug x86 Démarrer
Arduino 1.6/1.9 Arduino/Genuino Mega w/ ATmega256 COM3 Other Add Library Add Code
pluviometre.h pluviometre.cpp anemometre.h anemometre.cpp SupervisionSerre.ino potentiometre.cpp potentiometre.h main.cpp capteur.cpp capteur.h
SupervisionSerre -> Anemometre
- mesurer()

37 {
38   Serial.println("Bonne brise");
39 }
40 else
41 {
42   Serial.println("Bombe atomique");
43 }
44 }
45
46 void GirRotation()
47 {
48   int rotation = 0;
49   volatile unsigned long tpsContact;
50   if ((millis() - tpsContact) > 15)
51   {
52     rotation++;
53     tpsContact = millis();
54   }
55 }
56
57
58 void Anemometre::mesurer()
59 {
60   pinMode(anemometre_pin, INPUT_PULLUP);
61   attachInterrupt(digitalPinToInterrupt(anemometre_pin), GirRotation, FALLING);
62
63   Serial.begin(9600);
64
65   rotation = 0;
66   sei();
67   delay(3);
68   cli();
69   vitesse = rotation * 1.2075;
70   Serial.print(vitesse); Serial.println("km/h");
71   get_vitesse(vitesse);
72   delay(3);
73
74
75
76
100 %
Sortie
Afficher la sortie à partir de:
```

Comme nous pouvons le constater ci-dessus j'ai donc créé des classes sur Visual Studio pour simplifier le code et donc avoir un Main plus simple à exécuter

Voici donc un Main plus simple et plus compréhensible à l'aide des classes créées avec Visual Studio.

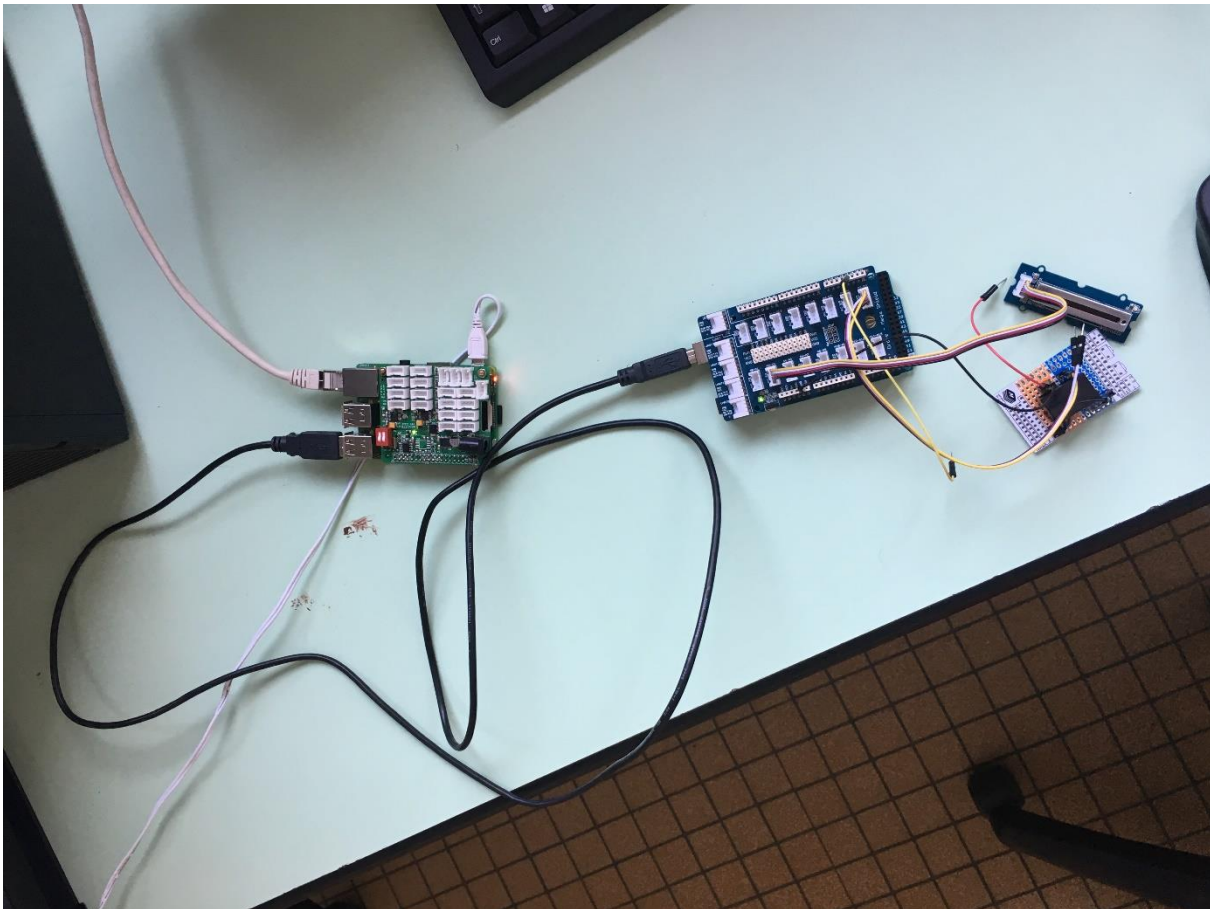


```
1  #include "pluviometre.h"
2  #include "potentiometre.h"
3  #include "capteur.h"
4  #include "anemometre.h"
5
6
7
8
9
10
11
12  int main()
13  {
14
15      Serial.begin(9600);
16      Potentiometre poten;
17      Anemometre anemo;
18      Pluviometre pluvio;
19
20
21
22
23
24
25
26
27
28  }
```

IV. Utilisation de la carte Raspberry

4.1) Envoie des données Arduino – Raspberry

Pour effectuer une connexion entre la Raspberry et l'Arduino on a donc opté pour une connexion USB car la mise en œuvre était simple et efficace.



Pour se faire, il fallait indiquer dans la Raspberry le port où était connecté l'Arduino puis la vitesse de connexion au système ici 9600 bauds.

Pour lire les informations de l'Arduino, il fallait donc utiliser la méthode `read()` intégrer.

```
connexion_arduino.py
1  #!/usr/bin/python3.4
2  # coding: utf-8
3
4  from serial import *
5  with Serial(port = "/dev/ttyACM0" , baudrate = 9600, timeout = 1, writeTimeout = 1) as ser:
6      if ser.isOpen():
7          while True:
8              ligne = ser.read()
9              print (ligne)
10
11
12  #nombre = input("Entrez un nombre : ")
13  #ser.write(nombre.encode('ascii'))
14
15
16
17 |
```

Pour la partie Arduino il fallait tout simplement utiliser la méthode `write()` de `Serial` pour envoyer des données à la Raspberry.

```
int main()
{
    int _a = 41;
    Serial.begin(9600);
    Potentiometre poten;
    Anemometre anemo;
    Pluviometre pluvio;
    while (true)
    {

        Serial.write(_a);

    }
}
```

4.2) Programme pour la connexion à la base de données

Pour pouvoir exécuter mon programme python et le rendre lisible et simple à comprendre j'ai donc décidé de créer des classes qui auraient chacune leur fonctionnalité.

En premier lieu pour tester la connexion à la base de données j'ai donc créé une classe BDD qui a pour fonction de créer un objet qui sera ma connexion à la base de données. Cet objet sera donc le facteur clé pour cette partie Raspberry car il aura pour rôle de faire le lien entre la Raspberry et la base de données.

Pour créer cet objet j'ai donc utilisé la librairie MySQL Connector puis je devais ensuite entrer en paramètre les informations sur le serveur et l'objet arrivait donc à se connecter à ma base de données.

```
connexion_arduino.py  BDD.py
1  #!/usr/bin/python3.4
2  # coding: utf-8
3  |
4
5  import mysql.connector
6
7
8  class BDD:
9      def __init__(self): #Constructeur pour tester la connexion a la bdd
10         try:
11             self.cnx = mysql.connector.connect(host="92.222.92.147", user="projetbts", \
12                                                password="Nantes44", \
13                                                database="supervision_serre")
14         except:
15             print("Connexion impossible")
```

4.3) Création de la classe gestion

```
1  #!/usr/bin/python3.4
2  # coding: utf-8
3
4
5  import sys
6  from datetime import datetime
7  import mysql.connector
8  import connexion_bdd
9  import time
10 from requete import requete
11 from BDD import BDD
12
13 class gestion:
14
15     typeMateriel = {"Capteur":"Capteur", "Microcontrôleur":"Microcontrôleur"}
16     unite = {"km/h":"km/h", "mm":"mm", "°":"°", "°C":"°C", "w/m²":"w/m²"}
17     abreviation = {"pluvio":"pluvio", "anemo":"anemo", "solari":"solari", \
18                  "sonde_tuyau":"sonde_tuyau", "sonde_serre": "sonde_serre"}
19
20     def __init__(self, bdd) : #Constructeur de la classe
21         self.__bdd = bdd
22         self.__codeResult = 0
23         self.__id_materiel = 0
24         self.__id_unite = ""
25         self.__id_type_materiel = 0
26
```

Voici la déclaration de la classe gestion qui va donc me permettre de gérer toute la communication avec la base de données.

4.3.1) Utilisation des requêtes

Pour communiquer avec la base de données il faut donc faire des requêtes et ces requêtes vont donc nous servir pour écrire sur la base de données.

Les requêtes que je vais utiliser seront principalement des INSERT et des SELECT.

Les INSERT permettent d'insérer des données dans la base de données et les SELECT permettent de sélectionner des données déjà présentes dans la base de données.

La fonction `__executerReqInsert` :

`sqlQuery` : nom du paramètre qui désigne une requête.

```
def __executerReqInsert(self, sqlQuery):
    try:
        if not self.__bdd.cnx.is_connected():
            self.__bdd.cnx.reconnect()
            """
            test si l'objet est connecté a la BDD si non elle tente une connection
            """
            if not self.__bdd.cnx.is_connected():
                raise ValueError
        cur = self.__bdd.cnx.cursor()
        print(sqlQuery)
        cur.execute(sqlQuery)
        """
        execute la requete entrer en parametre
        """
        self.__bdd.cnx.commit()
        """
        envoie l'exécution de la requete sur la base de données
        """
        cur.close()
    except ValueError: # verifie si il y a une erreurt de connexion a la base de données
        self.__codeResult = 10
        print("ERROR - échec connexion bdd")
    except: # verifie si il y a une autre erreur et nous indique de quel type il s'agit
        self.__codeResult = 11
        print("ERROR - \n{}".format(sys.exc_info()))

    return self.__codeResult
```

Voilà donc un exemple de fonction qui me permet d'exécuter des requêtes de type INSERT entrées en paramètre.

Dans cette fonction on utilise la fonction expliquée précédemment (`__executerReqInsert`) où l'utilisateur dans ce cas-ci ajoute un nouveau type de matériel à la base de données.

Pour se faire il a juste à entrer le nom du nouveau type en paramètre puis la requête s'exécutera automatiquement et avec la gestion des erreurs un problème sera détecté et indiqué à l'utilisateur.

Utilisation de la fonction `__executerReqInsert` dans la fonction `ajouterNouveauTypeMat` :

`nomTypeMat` : nom du nouveau type de matériel

```
def ajouterNouveauTypeMat(self, nomTypeMat): #Requete pour ajouter un nouveau type materiel
    self.__nomTypeMat = nomTypeMat

    query = "INSERT INTO type_materiel(nom) VALUES ('{}')" \
        .format(self.__nomTypeMat)

    self.__executerReqInsert(query)

    if self.__codeResult == 0:
        print("Succès exécution req INSERT")
    else :
        print("ERROR - \n{}".format(sys.exc_info()))
```

Voici donc un exemple de fonction pour exécuter une requête de type SELECT qui récupère l'ID du type de matériel demandé.

```
def __executerReqSelectIdTypeMat(self, sqlQuery):
    try:
        if not self.__bdd.cnx.is_connected():
            self.__bdd.cnx.reconnect()
        if not self.__bdd.cnx.is_connected():
            raise ValueError
        cur = self.__bdd.cnx.cursor()
        print(sqlQuery)
        """
        affiche la requete entrer en parametre
        """
        cur.execute(sqlQuery)
        """
        execute la requete entrer en parametre
        """
        self.__id_type_materiel = cur.fetchone()[0]
        """
        récupère seulement l'élément demandé à la base de données
        """
        cur.close()
    except ValueError: # verifie si il y a une erreurt de connexion a la base de données
        self.__codeResult = 10
        print("ERROR - échec connexion bdd")
    except: # verifie si il y a une autre erreur et nous indique de quel type il s'agit
        self.__codeResult = 11
        print("ERROR - \n{}".format(sys.exc_info()))

    return self.__codeResult, self.__id_type_materiel
```


Dans cette fonction on utilise la fonction expliquée précédemment (`__executerReqSelectIdTypeMat`) où l'utilisateur dans ce cas-ci ajoute un nouveau matériel à la base de données.

Utilisation de la fonction `__executerReqSelectIdTypeMat` dans la fonction `ajouterNouveauTypeMat` :

`nomMat` : nom du nouveau matériel.

`abreviation` : abréviation utilisé pour simplifier l'écriture du capteur (ex : pluvio pour pluviomètre).

`est_fonctionnel` : booléen (true ou false) pour dire si le matériel est en état de fonctionnement.

`TypeMat` : nom du type de matériel du nouveau matériel (ex : capteur pour pluvio).

```
def ajouterNouveauMat(self, nomMat, abreviation, est_fonctionnel, TypeMat):
    self.__nom = nomMat
    self.__abreviation = abreviation
    self.__est_fonctionnel = est_fonctionnel
    self.__TypeMat = TypeMat

    id_type_materiel = "SELECT id FROM type_materiel WHERE nom = '{}'" \
        .format(self.__TypeMat)

    self.__executerReqSelectIdTypeMat(id_type_materiel)

    if self.__codeResult == 0:
        print("Succès exécution req SELECT")
    else:
        print("ERROR - \n{}".format(sys.exc_info()))

    query = "INSERT INTO materiel(nom, abreviation, est_fonctionnel, id_type_materiel)" \
        "VALUES ('{}','{}',{},{})" \
        .format(self.__nom, self.__abreviation, \
            self.__est_fonctionnel, self.__id_type_materiel)

    self.__executerReqInsert(query)

    if self.__codeResult == 0:
        print("Succès exécution req INSERT")
    else:
        print("ERROR - \n{}".format(sys.exc_info()))
```

4.3.2) Utilisation du Main

```
1  #!/usr/bin/python3.4
2  # coding: utf-8
3
4
5
6  from gestion import gestion
7  from BDD import BDD
8  import mysql.connector
9  import connexion_bdd
10 #from classe_essai import classe_essai
11 #from requete import requete
12
13
14 bdd = BDD()
15 gest = gestion(bdd)
16 typeMat1 = "Micro"
17 typeMat2 = "Capteur"
18 typeMat3 = "Microcontrôleur"
19 est_fonctionnel = 1
20 abreviationCapt = "poterl"
21 nomCapt = "potentiometre"
22 unite = "mm"
23 valeur = 250
24 gest.ajouterNouveauTypeMat(typeMat1)
25 gest.ajouterNouveauMat(nomCapt, abreviationCapt , est_fonctionnel, typeMat1)
26 gest.enregistrerUnReleve(abreviationCapt , unite, valeur,)
```

V. Test Unitaire

```
def ajouterNouveauTypeMat(self, nomTypeMat): #Requete pour ajouter un nouveau type materiel
    self.__nomTypeMat = nomTypeMat

    query = "INSERT INTO type_materiel(nom) VALUES ('{}')" \
        .format(self.__nomTypeMat)

    self.__executerReqInsert(query)

    if self.__codeResult == 0:
        print("Succès exécution req INSERT")

    else :
        print("ERROR - \n{}".format(sys.exc_info()))

    return self.__codeResult
```

Pour le test unitaire j'ai décidé de le faire sur cette fonction avec la requête.

Le test doit me permettre de dire si la fonction s'est bien exécutée et pour cela j'ai donc vérifié si le codeResult qui est ma variable d'erreur est donc égale à 0 ce qui veut dire que tout c'est bien passé.

Voici le résultat du test unitaire j'ai donc appelé la fonction et vérifier que le codeResult que la fonction me renvoie est égal à 0.

Comme on peut le constater le test c'est bien effectué.

```
1  #!/usr/bin/python3.4
2  # coding: utf-8
3
4  import unittest
5  import random
6
7  from gestion import gestion
8  from BDD import BDD
9  import mysql.connector
10 import connexion_bdd
11
12 class RandomTest(unittest.TestCase):
13     def test_insert(self):
14         bdd = BDD()
15         gest = gestion(bdd)
16         typeMat1 = "Microme"
17         test = gest.ajouterNouveauTypeMat(typeMat1)
18
19         self.assertEqual(test, 0)
20
21
22
23
24
25
--
Shell
>>> %Run test_unitaire.py
INSERT INTO type_materiel(nom) VALUES ('Microme')
Succès exécution req INSERT
/usr/lib/python3.4/unittest/case.py:577: ResourceWarning: unclosed <socket.socket fd=6, family=AddressFamily.AF_INET,
6.200', 60774), raddr=('92.222.92.147', 3306)>
testMethod()
.
-----
Ran 1 test in 0.317s
OK
_ . . . .
```

Pour vérifier que mon test était véridique j'ai donc provoqué une erreur lors du test en vérifiant que le codeResult à la dernière ligne était égal à 1 et comme prévu le test s'est avéré faux.

```
1  #!/usr/bin/python3.4
2  # coding: utf-8
3
4  import unittest
5  import random
6
7  from gestion import gestion
8  from BDD import BDD
9  import mysql.connector
10 import connexion_bdd
11
12 class RandomTest(unittest.TestCase):
13     def test_insert(self):
14         bdd = BDD()
15         gest = gestion(bdd)
16         typeMat1 = "Microme"
17         test = gest.ajouterNouveauTypeMat(typeMat1)
18
19         self.assertEqual(test, 1)
20
21
22
23
24
25
--
```

Shell

```
=====
FAIL: test_insert (__main__.RandomTest)
-----
Traceback (most recent call last):
  File "/home/pi/Desktop/Supervision serre/test unitaire.py", line 19, in test_insert
    self.assertEqual(test, 1)
AssertionError: 11 != 1
-----

Ran 1 test in 0.292s

FAILED (failures=1)
```

VI. Conclusion

Pour conclure je vais dire que ce projet m'a beaucoup appris notamment sur le travail de groupe sur du très long terme la répartition des tâches.

Cela m'a aussi permis de m'améliorer en programmation python et d'être plus méthodique dans mes recherches et mon travail.

Ce projet de groupe m'a permis d'évoluer et d'utiliser des logiciels dont je n'avais pas l'habitude comme github ou bien l'utilisation de l'Arduino.

A l'écriture de ce dossier personnel le projet n'est pas encore terminé et je compte apprendre tout ce que je peux jusqu'à la fin et laisser un travail terminé pour ceux qui reprendront notre projet l'année prochaine.

