POLYTECH
LILLE

Université Lille1
Sciences et Technologies

Commande intelligente d'une VMC

- **DUTHOIT Simon**
- **HAMZAOUI Oussama**

2014

I) Introduction

Le présent rapport a pour but d'exposer le travail réalisé dans le cadre du projet du semestre 8. Les projets du semestre 8 se font tout seul ou par binôme sur un sujet choisi parmi une liste prédéfinie. 8 heures par semaine sont exclusivement allouées pour sa réalisation en plus des éventuelles heures personnelles. Promouvant le travail en totale autonomie (vis-à-vis des professeurs) et le travail en groupe si l'on est en binômes, il permet de mettre en pratique ses propres connaissances et bien sûr d'en apprendre de nouvelles.

Le projet que nous avons choisi est la commande de la VMC. Plusieurs raisons nous ont poussés à cela :

-ce projet était un l'un des projets obligatoires de la liste à choisir

-il est pluridisciplinaire, c'est-à-dire qu'il implique plusieurs domaines : l'automatique, l'électronique et l'électronique de puissance ainsi que la programmation.

-il s'inscrit parfaitement dans la problématique actuelle du développement durable : commander intelligemment une VMC peut permettre de faire des économies d'énergie.

Nous allons tout d'abord présenter le projet et décrire les objectifs que nous nous étions fixés. Nous parlerons plus en détail de chaque partie du projet et du travail réalisé. Nous exposerons aussi le résultat final et discuterons de possibilités d'améliorations.

II) Présentation du projet

Nous allons tout d'abord définir plus précisément notre projet suite à un entretien avec nos tuteurs. L'objectif est de réaliser une VMC (Ventilation Mécanique Contrôlée) simple flux dite "intelligente". Le principe de la VMC simple flux est d'extraire l'air de pièces dites "humides" (cuisine, salle de bains et toilettes) ce qui crée une dépression dans la maison. L'air extérieur rentre ensuite par des ouïes placées dans les "pièces à vivre" (chambre, séjour, etc ...) grâce à la dépression créée, ce qui permet d'avoir ainsi une maison ventilée avec un air sain. Elle peut, et c'est notre cas ici, avoir plusieurs vitesses.

Voici un schéma illustrant le principe de la VMC simple flux :

Le but de notre projet est de commander cette VMC à partir de paramètres qui sont:

- les températures intérieure et extérieure
- le pourcentage d'humidité des pièces dites "humides"
- le taux de CO2 dans la maison
- l'état du système de chauffage de la maison (ON/OFF)

Nous pourrons donc comparer tous ces paramètres, une fois acquis, à des "scénarios" que nous aurons prédéfinis et qui nous permettront donc d'établir une commande "intelligente" du système d'extraction. La raison d'être principale de ce projet est évidemment d'économiser l'énergie en ne laissant pas tourner la VMC constamment d'une part, et en limitant son utilisation le plus possible lors du chauffage d'une pièce d'autre part.

Après avoir compris le fonctionnement de la VMC et ciblé le travail à faire pour l'améliorer, nous nous sommes mis d'accord sur les solutions techniques à adopter pour mener à bien notre réalisation. Tous les paramètres évoqués précédemment (servant à commander la VMC) seront bien évidemment acquis par des capteurs, ou simulés dans le cas de capteurs trop chers et/ou rares. Ces données seront transmises au microcontrôleur (Arduino avec son ATMEGA328P), envoyées vers un autre microcontrôleur via le protocole Zigbee et traitées pour établir le signal de commande de la VMC. Ladite VMC sera simulée par un ventilateur. Nous réaliserons aussi une maquette d'une maison pour pouvoir simuler notre système. Enfin, des PCBs seront réalisés pour la carte capteurs et la carte de commande de la VMC, sur lesquelles seront implantés des Arduino Mini au lieu d'Arduino Uno car plus compacts.

Nous allons maintenant décrire le travail réalisé dans chacune des parties, à savoir la partie commande liée à la récupération des paramètres de qualité de l'air et de la commande de la VMC, ainsi que la partie électronique qui traitera de la conception des cartes.

III) Partie commande

1) Capteurs

Il a tout d'abord fallu définir comment nous allions procéder pour mettre en place la commande de la VMC. Nous avions déjà défini le microcontrôleur à utiliser (ATMEGA328P au sein d'un Arduino, Uno pour la phase de développement et Mini pour la phase finale d'implantation). Il ne nous restait donc plus qu'à définir quels capteurs nous devions utiliser et ensuite quel type de commande nous allions mettre en œuvre (méthode de régulation choisie incluant l'élaboration de scénarios possibles quant à la température et les paramètres de l'air dans les pièces humides).

Nous nous sommes d'abord penchés sur l'aspect récupération des données issues des capteurs sur l'Arduino Uno. Les capteurs que nous utilisons sont :

-le DHT11 : capteur de température/humidité disposant de sa propre librairie pour faciliter la récupération des données sur un Arduino. Il dispose de 3 broches qui sont l'alimentation, la masse et les données que transmet le capteursous forme de trames.

-le LM35 DZ : capteur de température qui lui a aussi 3 broches, les mêmes que celles du DHT11 sauf que la broche de données délivre une tension analogique (fonction de la température) et non une trame de bits.

-En ce qui concerne le capteur de CO2 il sera simulé à l'aide d'un potentiomètre car trop onéreux.

Nous avons donc réalisé sur breadboard les montages électriques permettant de relier les capteurs à l'Arduino pour l'exploitation des données. Il suffisait de câbler l'alimentation, la masse et de relier les broches de données aux broches de l'Arduino (une broche analogique pour le LM35 DZ et une broche numérique pour le DHT11). Le DHT11 nécessite une résistance de pull-up en plus (voir schéma ci-dessous). En ce qui concerne le capteur de CO2 simulé, nous avons réalisé un montage pont diviseur de tension où nous prélevons la tension aux bornes du potentiomètre (qui varie donc). Celle-ci attaque une broche analogique, à l'instar du LM35DZ, du microcontrôleur.

Voici ci-dessous le schéma de câblage de DHT :

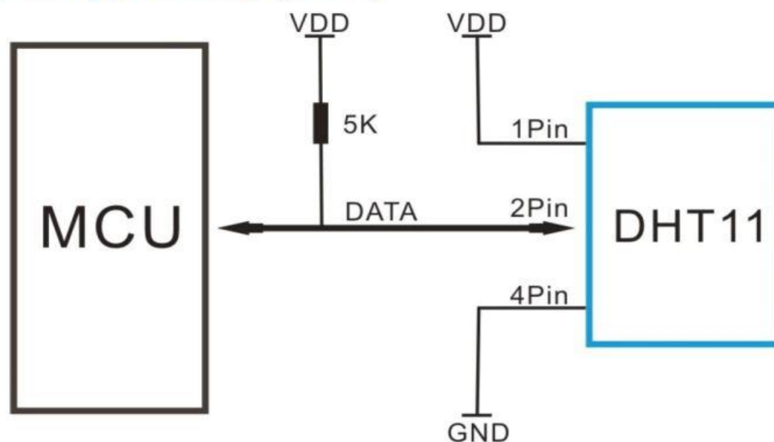


Figure 1 Typical Application

Les montages une fois faits, nous avons décidé d'utiliser l'environnement Arduino IDE, qui permet l'utilisation des nombreuses librairies Arduino, pour le développement. Comme dit, précédemment, le DHT11 dispose de sa propre librairie et nous n'avions donc plus qu'à utiliser les fonctions pour récupérer la température et l'humidité (il faut réveiller le capteur qui répond ensuite avec une trame dans laquelle sont inclus deux octets contenant ces données). Concernant le LM35 DZ et le capteur de CO2 simulé, il suffit d'utiliser l'ADC pour convertir la tension analogique en un nombre binaire que l'on remet à l'échelle (en fonction de la sensibilité du capteur, des tensions d'alimentation de l'ADC et du nombre de bits de la conversion) pour récupérer notre température, à l'aide de la fonction analogRead.

Nous avons aussi en première approche, simulé la VMC à l'aide d'une LED branchée sur une sortie PWM via une résistance. Ceci nous a permis de visualiser le comportement de la vitesse, ici représenté par l'intensité lumineuse de la LED, qui varie en fonction du rapport cyclique du signal PWM, que l'on règle par une simple valeur comprise entre 0 et 255 (0 à 1 pour le rapport cyclique).

Voici ci-dessous les bouts de code qui nous ont permis de récupérer tous les paramètres de qualité de l'air mesurés par les capteurs :

```
void temperatureHumidityDHT (float * temperature, float * humidity)
{
  int success = DHT11.read(pinDHT11);
  if (success==0)
  {
    Serial.print("temp:");
    *temperature = DHT11.temperature;
    Serial.print(*temperature);
    Serial.print(" humi:");
    *humidity = DHT11.humidity;
    Serial.print(*humidity);
    Serial.println();
  }
  else
  {
    Serial.print(success);
    *temperature = -1;
    *humidity = -1;
  }
}

void temperatureExtLM35(float * temperatureExt)
{
  *temperatureExt = analogRead(pinLM35);
  *temperatureExt = (*temperatureExt *5) / 1023;      // 0/5 V 10-bit conversion
  *temperatureExt = *temperatureExt * 100;            // Temperature conversion (10 mV/°C sensitivity)
  Serial.print("temp ext:");
  Serial.print(*temperatureExt);
  Serial.println();
}

void CO2Rate (float * CO2rate)
{
  *CO2rate = analogRead(pinCO2rate);
  Serial.print("CO2 rate :");
  Serial.print(*CO2rate);
  Serial.println();
}
```

2) Commande

Le choix de la commande est fondamental dans un système automatique, il en découlera toutes les performances et caractéristiques du système. C'est donc un choix réfléchi que nous nous devons de faire. Le nôtre s'est porté sur la logique floue.

Son père fondateur est Lofti Zadeh qui l'a rendue connue en 1965. Elle se base sur la théorie mathématique des ensembles flous : des ensembles différents de la logique simple qui n'inclut que vrai ou faux en terme d'appartenance à un ensemble. Par exemple, nous pouvons appartenir un peu ou moyennement à un ensemble. De plus, l'établissement de la commande se base sur des règles que nous définissons nous-mêmes et ne nécessite pas la connaissance du modèle mathématique du système. C'est donc le choix que nous avons jugé le plus judicieux de faire pour commander la VMC.

Le principe de la "logique floue" repose sur 3 étapes :

-la fuzzification : qui consiste à créer des fonctions d'appartenance pour chacune des entrées (froid ou chaud pour la température par exemple) et des sorties. Elles sont comprises entre 0 et 1 et dépendent de la grandeur de l'entrée. 2 fonctions d'appartenance consécutives, également, se chevauchent. La différence notable est que ces fonctions ne sont pas tout ou rien comme nous venons de le dire (on peut être "un peu froid" ou "moyennement froid", pas seulement "froid" ou "pas froid"). Nos entrées ici sont donc les températures intérieure et extérieure, l'humidité intérieure et le taux de CO_2 intérieur.

-l'inférence : qui consiste à créer les règles de commande sous la forme if "entrée1 = ... and entrée2 = ... and ... then sortie = ..."

-la défuzzification : qui consiste à partir des règles d'inférence définies précédemment et des fonctions d'appartenance créées durant la fuzzification, de commander la sortie.

2.1) Fuzzification

Nous avons établi le modèle de notre FLC (Fuzzy Logic Controller) sous Matlab Simulink qui dispose d'un outil permettant de faire de la logique floue. Nous avons tout d'abord commencé par la fuzzification et avons pu créer les MF (Membership Functions qui veut dire fonctions d'appartenance) grâce à des informations tirées d'Internet quant aux seuils acceptables/préférables et dangereux/inadaptés des différents paramètres. Ci-dessous, sont décrits ces seuils et se trouvent les fonctions d'appartenance définies.

Informations sur la qualité de l'air :

Taux de C02

-<400 ppm : excellente qualité d'air

-400-600 ppm : qualité moyenne

-600-1000 ppm : taux acceptable

->1000 ppm : mauvaise qualité d'air (danger)

Taux d'humidité

-entre 40 et 60 % : bon taux

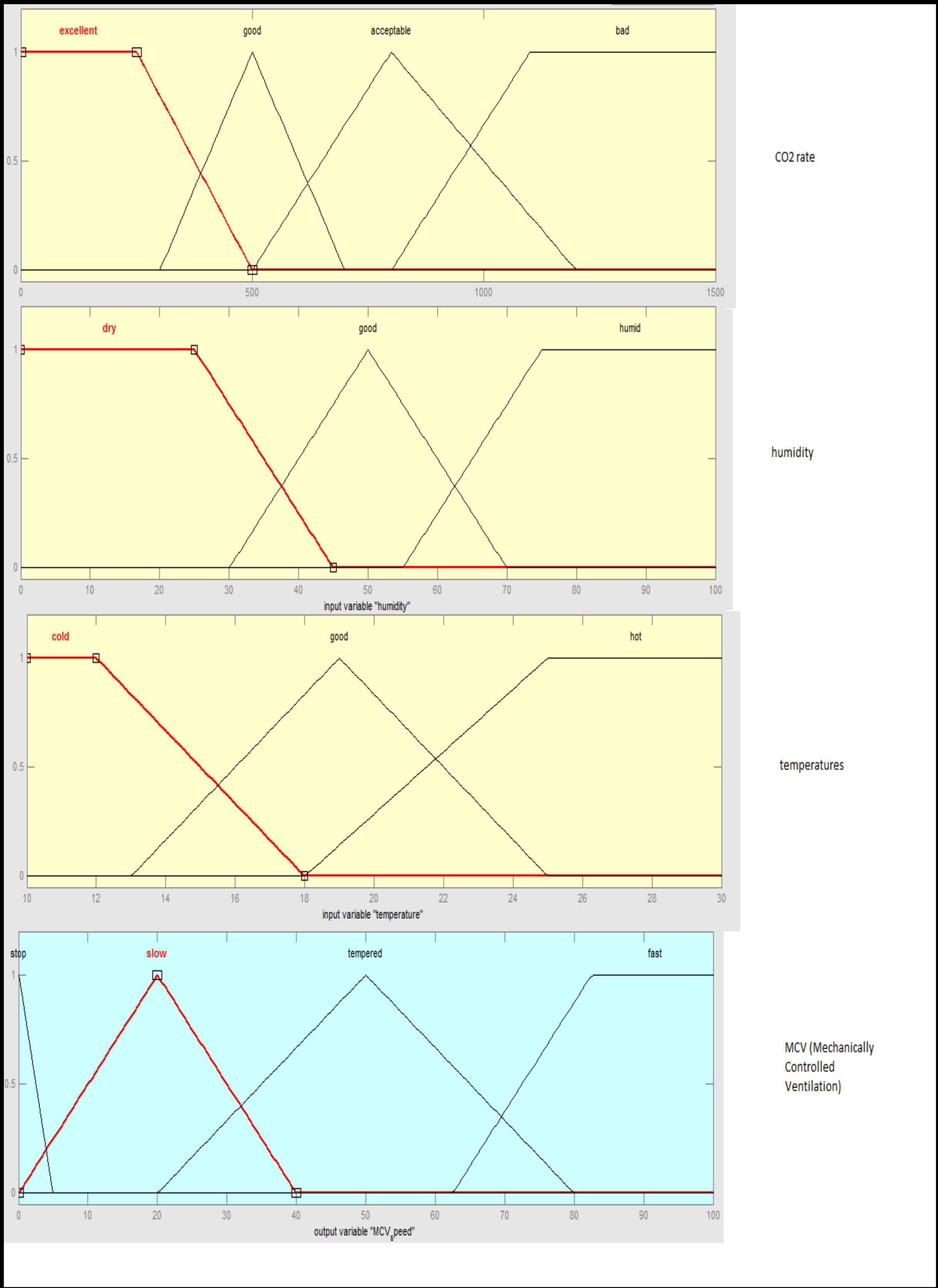
-<30 % : trop sec (alarme taux trop bas)

->70 % : trop humide (alarme taux trop haut)

Température

-19 degrés

Les fonctions d'appartenances sont :



2.2) Inférence

Comme dit ci-dessus l'inférence est la définition des règles de commande sous forme de if "condition then sortie". Détaillons un peu plus cette structure : en fait on a : if "entrée1 = MFentrée1 and entrée2 = MFentrée2 ..." then "sortie = MFsortie" où MFentrée1 est une fonction d'appartenance (Membership Function) de l'entrée 1 définie au préalable. Il en va de même pour MFentrée2 et MFsortie. Un exemple avec notre système donnerait : if "humidity = humid then sortie = fast".

Voici toutes les règles d'inférence que nous avons définies pour notre système :

1) If (CO2_rate is bad) then (MCV_speed is fast)

2 If (CO2_rate is excellent) and (humidity is humid) then (MCV_speed is fast)

3) If (CO2_rate is excellent) and (humidity is good) and (temperature is hot) and (ext_temperature is not hot) then (MCV_speed is slow)

4) If (CO2_rate is good) and (humidity is good) and (temperature is hot) and (ext_temperature is not hot) then (MCV_speed is tempered)

5) If (CO2_rate is good) and (humidity is humid) then (MCV_speed is fast)

6) If (CO2_rate is acceptable) and (humidity is good) and (temperature is not cold) then (MCV_speed is slow)

7) If (CO2_rate is acceptable) and (humidity is good) and (temperature is hot) and (ext_temperature is not hot) then (MCV_speed is tempered)

8) If (CO2_rate is acceptable) and (humidity is humid) then (MCV_speed is fast)

9) If (CO2_rate is not bad) and (humidity is dry) then (MCV_speed is stop)

10) If (CO2_rate is excellent) and (humidity is not humid) and (temperature is not hot) then (MCV_speed is stop)

11) If (CO2_rate is good) and (humidity is not humid) and (temperature is not hot) then (MCV_speed is stop)

12) If (CO2_rate is excellent) and (humidity is good) and (temperature is hot) and (ext_temperature is hot) then (MCV_speed is stop)

On a donc défini nos règles et à fortiori le comportement voulu de notre système. Le problème maintenant est de définir un degré d'appartenance (similaire au concept de MF), à chaque règle que l'on vient de définir car ce ne sont pas des conditions binaires au sein desdites règles. Il faut faire appel aux opérateurs flous pour pallier ce problème, en vue d'une future implantation au sein d'un microcontrôleur. Ici, seuls deux opérateurs nous sont nécessaires quant aux types de règles que l'on a :

-le **AND** est l'opérateur mathématique min

-le **NOT** (X) est remplacé par 1 – X

Prenons un exemple concret sur la règle simple suivante : **if "temperatureInt is hot and humidity is not humid then VMC = tempered"** où **hot**, **humid** et **tempered** sont les MF respectives de chaque entrée/sortie de la règle d'inférence. La fuzzification nous retourne donc une valeur entre 0 et 1 pour chaque entrée vis-à-vis de sa MF considérée. Admettons ici que l'on ait :

-temperatureInt is hot = 0.5

-**humidity is humid = 0.25**

On a **donc 0.5 AND 1 - 0.75 = 0.5 AND 0.25 = min (0.5,0.25) = 0.25**. Le degré d'appartenance de cette règle vaut donc 0.25. Cette méthode est donc appliquée à toutes les règles de notre système et va permettre de défuzzifier le système très simplement comme nous allons le voir dans le paragraphe suivant.

2.3) Défuzzification

La dernière étape est donc la défuzzification. Celle-ci concerne tout simplement à pouvoir appliquer une valeur de consigne à notre système, qui aura été déterminée par le comportement décrit par les règles d'inférence et les MF des E/S, toutes définies et explicitées précédemment.

Il existe plusieurs méthodes de défuzzification qui ont été développées par des chercheurs qui ont planché sur la logique floue. Ici, nous utiliserons une méthode simple et robuste nommée centroid, signifiant en anglais centre de gravité. Cette méthode lie notre sortie, par une relation très simple, aux degrés d'appartenance des règles d'inférence et à l'abscisse du max de la MF de sortie utilisée dans la règle considérée. Ces derniers paramètres seront respectivement notés A et B pour alléger la formule :

- Sortie :

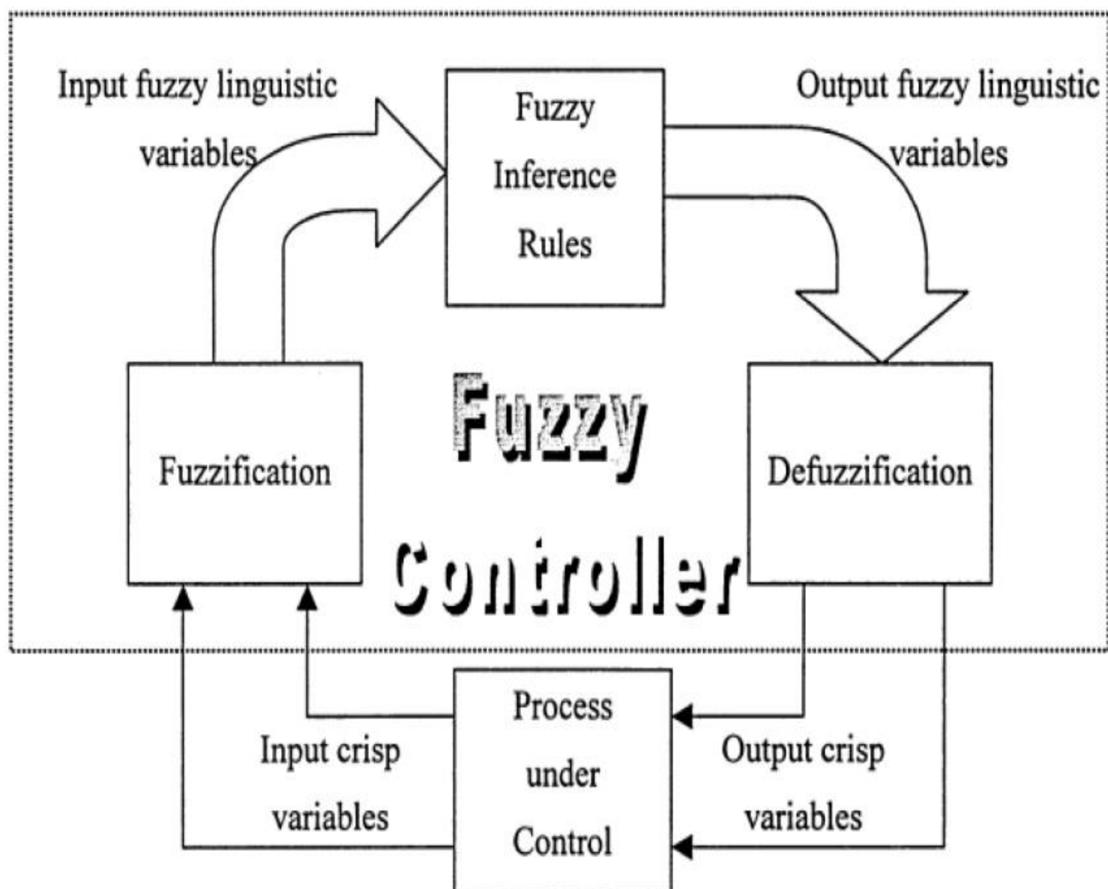
$$(\sum A * B)/(\sum A)$$

Prenons encore une fois un exemple concret pour éclaircir cette formule. Admettons que l'on ait deux règles ayant comme degré d'appartenance 0.75 et 0.25. Ces deux scalaires correspondent aux valeurs A dans notre formule. Pour la première règle admettons que la MF de sortie soit fast et pour le second **slow**. Conformément à l'explication précédente et à la MF définie précédemment (voir photo ci-dessus), on a donc B valant 100 % pour la première règle et 20 % pour la deuxième. Si on applique la formule on aura donc la sortie qui vaudra :

sortie = (0.75*100 + 0.25*20) / (0.75 + 0.25) = 80 %

Nous voilà arrivés au terme de l'établissement de notre modèle flou pour la commande de notre système. Il faut maintenant l'implanter au sein d'un microcontrôleur (ici l'Arduino Uno), donc en langage C. C'est l'objet des paragraphes suivants qui va traiter de l'adaptation de tout ce que nous avons expliqué à propos de la commande floue, en langage C.

Ci-dessous se trouve un schéma résumant les 3 étapes, qui constituent un FLC (Fuzzy Logic Controller), que nous avons expliquées.



3) Implantation au sein d'un microcontrôleur

C'est sans nul doute l'étape la plus intéressante du projet : transformer notre modèle théorique en quelque chose de compréhensible par le microcontrôleur pour commander le vrai système physique. Nous allons procéder de la même manière que précédemment pour expliquer le travail réalisé, c'est-à-dire que nous allons détailler l'implémentation de chacune des 3 parties du FLC (fuzzification, inférence et défuzzification).

3.1) Fuzzification

Le travail de récupération des différents paramètres à l'aide des capteurs avait déjà été réalisé donc nous avions déjà les données sous la main pour commencer la fuzzification de chaque entrée. Cette partie est la plus simpliste. En effet, il suffit de retourner l'image des données capteurs pour chacune de leurs MF qui sont triangulaires ou trapézoïdales (donc des fonctions linéaires / constantes). Voici la fonction permettant de retourner l'image d'une donnée capteur pour une MF donnée.

```
float fuzzify_MF(float x, float a, float b, float c, float d) //x=crisp input, a,b,c,d the marks of the membership function
{
float dom;
if(x>a && x<b) dom = (x-a)/(b-a);
else if (x>c && x<d) dom=(d-x)/(d-c);
else if (x>=b && x<=c) dom=1.0;
else dom=0;
return dom;
}
```

NB : b et c sont confondus dans le cas d'une MF triangulaire

La précédente fonction sert de base pour fuzzifier. Il a fallu généraliser cela à tous nos paramètres et stocker les résultats d'appartenance à chaque MF. Ces derniers sont stockés dans des tableaux de taille "nombre de MF pour le paramètre considéré". De plus, les arguments a,b,c et d de la fonction fuzzify_MF sont stockés dans des tableaux de taille "4*nombre de MF pour le paramètre considéré". Voici le bout de code qui a permis cela (tous les tableaux et les paramètres mesurés par les capteurs sont les paramètres de la fonction) :

```
void fuzzifyVariables(float tabMFCO2[], float resultMFCO2[],float tabMFHumidity[], float resultMFHumidity[],float tabMFTemp[], float resultMFTemp[],float tabMFTempExt[],
float resultMFTempExt[],float tabMFMCV[], float resultMFMCV[], float CO2rate, float temperature, float humidity, float temperatureExt)
{
 int i;

 //CO2 rate
 Serial.print("CO2 rate MF:");
 for(i=0;i<nbMFCO2;i++)
 {
   resultMFCO2[i] = fuzzify_MF(CO2rate,tabMFCO2[(4*i)],tabMFCO2[(4*i)+1],tabMFCO2[(4*i)+2],tabMFCO2[(4*i)+3]);
   Serial.print(resultMFCO2[i]);
   Serial.println();
 }
 Serial.println();

 //humidity
 Serial.print("Humidity MF:");
 for(i=0;i<nbMFHumidity;i++)
 {
   resultMFHumidity[i] = fuzzify_MF(humidity,tabMFHumidity[(4*i)],tabMFHumidity[(4*i)+1],tabMFHumidity[(4*i)+2],tabMFHumidity[(4*i)+3]);
   Serial.print(resultMFHumidity[i]);
   Serial.println();
 }
 Serial.println();
 //temperature
 Serial.print("Temperature MF:");
 for(i=0;i<nbMFTemp;i++)
 {
   resultMFTemp[i] = fuzzify_MF(temperature,tabMFTemp[(4*i)],tabMFTemp[(4*i)+1],tabMFTemp[(4*i)+2],tabMFTemp[(4*i)+3]);
   Serial.print(resultMFTemp[i]);
   Serial.println();
 }
 Serial.println();

 //exterior temperature
 Serial.print("Exterior Temperature MF:");
 for(i=0;i<nbMFTempExt;i++)
 {
   resultMFTempExt[i] = fuzzify_MF(temperatureExt,tabMFTempExt[(4*i)],tabMFTempExt[(4*i)+1],tabMFTempExt[(4*i)+2],tabMFTempExt[(4*i)+3]);
   Serial.print(resultMFTempExt[i]);
   Serial.println();
 }
 Serial.println();

 //MCV
 Serial.print("MCV MF:");
 for(i=0;i<nbMFMCV;i++)
 {
   resultMFMCV[i] = fuzzify_MF((CO2rate*100)/513,tabMFMCV[(4*i)],tabMFMCV[(4*i)+1],tabMFMCV[(4*i)+2],tabMFMCV[(4*i)+3]);
   Serial.print(resultMFMCV[i]);
   Serial.println();
 }
 Serial.println();

}
```

3.2) Inférence

Cette deuxième partie était intéressante à réaliser car il a fallu s'interroger quant à un moyen de stocker des règles de type if X and Y then Z. Le choix retenu est un tableau d'une structure contenant des tableaux pour chaque paramètre capteur. le tableau de structure contient autant de cases que de règles (i.e chaque case = une règle) et les tableaux des paramètres capteurs contiennent autant de cases que de MF liées au paramètre considéré.

La structure est remplie de la manière suivante : pour chaque case de cette dernière correspond donc une règle. Et chaque case des tableaux internes à la structure vaut 0,1 ou 2 selon que la case, qui correspond à une MF d'un paramètre capteur, fait intervenir cette dernière dans la règle d'inférence (1 ou 2 si la MF intervient en NOT) ou non (0 dans ce cas). Voici la définition en C de tout ce que l'on vient d'expliquer, c'est-à-dire de toutes les règles d'inférence. Ce bout de code sert d'exemple à la compréhension de ce paragraphe et on notera l'initialisation de la structure entière à 0.

```
struct rule                              //contains a rule
{
  int CO2Rule[nbMFCO2];                  //0 : MF not considered, 1 : MF considered, 2 : MF considered (not)
  int humidityRule[nbMFHumidity];
  int tempRule[nbMFTemp];
  int tempExtRule[nbMFTempExt];
  int MCVRule[nbMFMCV];                  //same without the 2
  float f;                               //value of the validity of the rule ([0;1])
};

int j,k;
for(j=0;j<nbRules;j++)
{
  for(k=0;k<nbMFCO2;k++) rules[j].CO2Rule[k] = 0;
  for(k=0;k<nbMFHumidity;k++) rules[j].humidityRule[k] = 0;
  for(k=0;k<nbMFTemp;k++) rules[j].tempRule[k] = 0;
  for(k=0;k<nbMFTempExt;k++) rules[j].tempExtRule[k] = 0;
  for(k=0;k<nbMFMCV;k++) rules[j].MCVRule[k] = 0;
  rules[j].f=0;
}

//Definition of rules
rules[0].CO2Rule[3] = 1;
rules[0].MCVRule[3] = 1;

rules[1].CO2Rule[0] = 1;
rules[1].humidityRule[2] = 1;
rules[1].MCVRule[3] = 1;

rules[2].CO2Rule[0] = 1;
rules[2].humidityRule[1] = 1;
rules[2].tempRule[2] = 1;
rules[2].tempExtRule[2] = 2;
rules[2].MCVRule[1] = 1;

rules[3].CO2Rule[1] = 1;
rules[3].humidityRule[1] = 1;
rules[3].tempRule[2] = 1;
```

```
rules[3].tempExtRule[2] = 2;
rules[3].MCVRule[2] = 1;

rules[4].CO2Rule[1] = 1;
rules[4].humidityRule[2] = 1;
rules[4].MCVRule[3] = 1;

rules[5].CO2Rule[2] = 1;
rules[5].humidityRule[1] = 1;
rules[5].tempRule[0] = 2;
rules[5].MCVRule[1] = 1;

rules[6].CO2Rule[2] = 1;
rules[6].humidityRule[1] = 1;
rules[6].tempRule[2] = 1;
rules[6].tempExtRule[2] = 2;
rules[6].MCVRule[2] = 1;

rules[7].CO2Rule[2] = 1;
rules[7].humidityRule[2] = 1;
rules[7].MCVRule[3] = 1;

rules[8].CO2Rule[3] = 2;
rules[8].humidityRule[0] = 1;
rules[8].MCVRule[0] = 1;

rules[9].CO2Rule[0] = 1;
rules[9].humidityRule[2] = 2;
rules[9].tempRule[2] = 2;
rules[9].MCVRule[0] = 1;

rules[10].CO2Rule[1] = 1;
rules[10].humidityRule[2] = 2;
rules[10].tempRule[2] = 2;
rules[10].MCVRule[0] = 1;

rules[11].CO2Rule[0] = 1;
rules[11].humidityRule[1] = 1;
rules[11].tempRule[2] = 1;
rules[11].tempExtRule[2] = 1;
rules[11].MCVRule[0] = 1;
```

La deuxième partie consiste à parcourir cette structure pour en déduire les degrés d'appartenance aux règles d'inférence. En se référant aux explications données précédemment à ce sujet (sur les opérateurs flous dans l'explication de l'inférence), on peut aisément comprendre le bout de code ci-dessous qui stocke dans le champ f de la structure rules le degré d'appartenance de la règle.

```
float mini=1;
for(j=0;j<nbRules;j++)
{
  for(k=0;k<nbMFCO2;k++)
  {
    if(rules[j].CO2Rule[k] == 1)
    {if(resultMFCO2[k] < mini) mini = resultMFCO2[k];}
    else if (rules[j].CO2Rule[k] == 2)
    {if((1-resultMFCO2[k]) < mini) mini = 1-resultMFCO2[k];}
  }


  for(k=0;k<nbMFHumidity;k++)
  {
    if(rules[j].humidityRule[k] == 1)
    {if(resultMFHumidity[k] < mini) mini = resultMFHumidity[k];}
    else if (rules[j].humidityRule[k] == 2)
    {if((1-resultMFHumidity[k]) < mini) mini = 1-resultMFHumidity[k];}
  }

  for(k=0;k<nbMFTemp;k++)
  {
    if(rules[j].tempRule[k] == 1)
    {if(resultMFTemp[k] < mini) mini = resultMFTemp[k];}
    else if (rules[j].tempRule[k] == 2)
    {if((1-resultMFTemp[k]) < mini) mini = 1-resultMFTemp[k];}
  }

  for(k=0;k<nbMFTempExt;k++)
  {
    if(rules[j].tempExtRule[k] == 1)
    {if(resultMFTempExt[k] < mini) mini = resultMFTempExt[k];}
    else if (rules[j].tempExtRule[k] == 2)
    {if((1-resultMFTempExt[k]) < mini) mini = 1-resultMFTempExt[k];}
  }

  rules[j].f = mini;
  Serial.print(rules[j].f);
  Serial.println();
  mini = 1;


 }
}
```

4) Communication entre la partie commande et la partie capteurs grâce au protocole Zigbee

Nous l'avons vu précédemment, nous avons réalisé en entier la partie commande de la VMC. Cette dernière se compose, pour résumer très rapidement, de la récupération des données capteurs et de l'algorithme de commande floue à partir de celles-ci. Jusqu'à présent, tout avait été fait et testé dans le même programme. Cependant, comme nous l'avons déjà exposé auparavant, deux cartes sont prévues : l'une récupérant les données capteurs et les transmettant à l'autre (via le protocole Zigbee) qui va s'en servir pour l'algorithme de commande. Il a donc fallu splitter notre programme en deux pour le mettre sur deux microcontrôleurs différents (Arduino Uno), programmer deux modules Xbee Series 1 pour la communication et sur le second microcontrôleur recevant les données capteurs, écrire un bout de code permettant de recomposer les chaînes de caractères reçues et en extraire les données utiles.

ZigBee est un protocole de haut niveau permettant la communication de petites radios, à consommation réduite, basée sur la norme IEEE 802.15.4 pour les réseaux à dimension personnelle (Wireless Personal Area Networks : WPAN).

La programmation des Xbee Series 1 s'avère être une tâche très facile. Il suffit de connecter son Xbee (branché sur un adaptateur USB fourni en salle de projet) sur le PC et d'ouvrir un terminal de communication. Les paramètres de communication choisis sont tout ce qu'il y a de plus standard : 9600 Bauds, pas de contrôle de flux , 8 bits de données. Il suffit ensuite de taper "+++" et attendre la réponse "OK" du Xbee, ce qui permet de rentrer en mode commande, pour commencer la configuration. Ce qu'on configure ici est très simple, c'est un réseau WPAN composé de deux stations. Les paramètres choisis sont les suivants :

-3000 est l'ID du réseau

-1 est l'adresse du Xbee émetteur dans ce réseau

-2 est l'adresse du Xbee récepteur

Lignes de commande utilisées pour l'émetteur :

-ATID 3000 (ID du réseau)

-ATMY 1 (adresse de l'émetteur)

-ATDH 0 et ATDL 2 (indique l'adresse du récepteur sur 16 bits, les 8 bits de poids fort sont indiqués par ATDH et les 8 faibles par ATDL)

-ATWR (sauvegarde de la configuration)

Lignes de commande utilisées pour le récepteur :

-ATID 3000

-ATMY 2

-ATWR

Cette très simple série de commandes nous permet donc de communiquer. Il faut cependant relier les Xbee à leurs Arduino respectifs pour établir la communication au sens propre du terme. Encore une fois, des adaptateurs sont disponibles à l'école où il suffit de brancher son Xbee. Un connecteur femelle sur cet adaptateur nous permet de relier les 4 broches dont nous avons besoin à l'arduino à savoir :

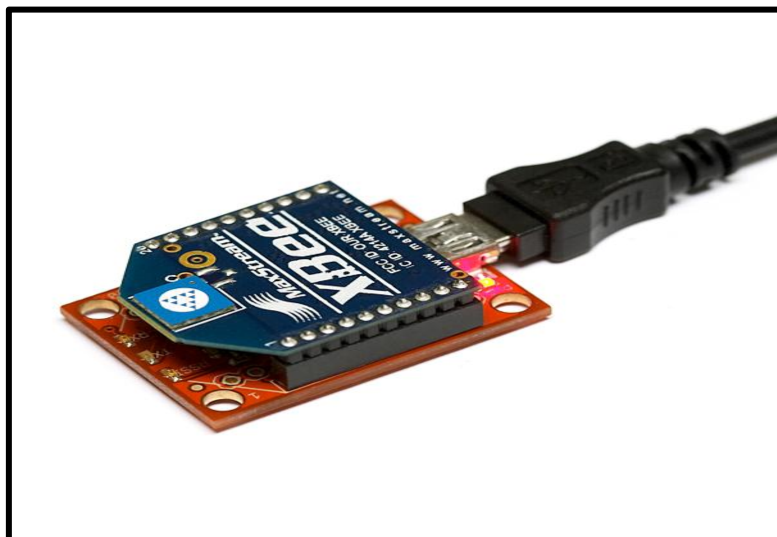
-Vcc

-Gnd

-Tx

-Rx

Adaptateur USB pour Xbee, identique à l'adaptateur pour relier à l'Arduino sans le connecteur 4 broches :



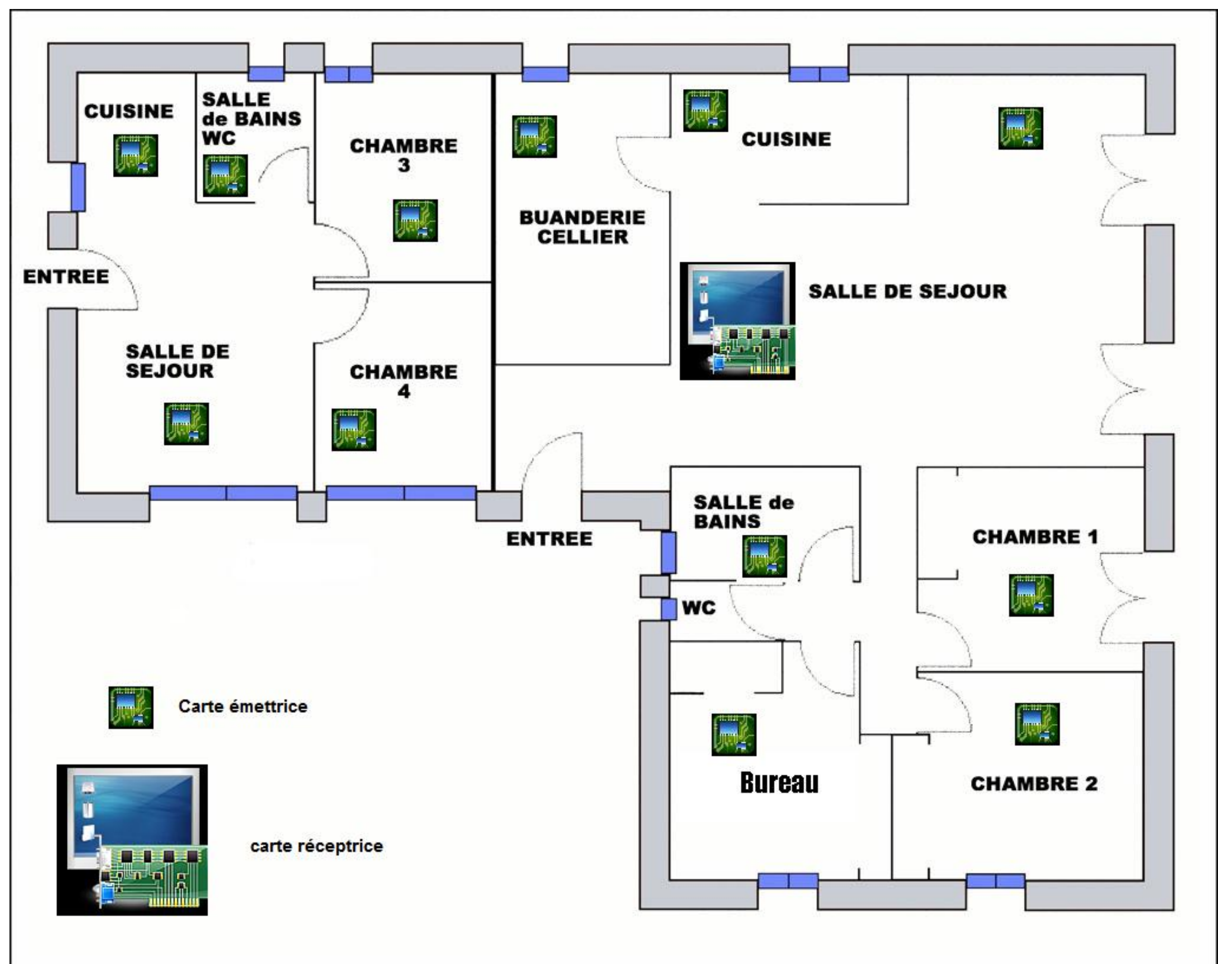
Nous entamons maintenant la dernière partie de la partie programmation qui est l'extraction des données reçues sur la carte qui va commander la VMC. Il faut tout d'abord reconstituer la chaîne de caractères reçue à intervalles réguliers et contenant toutes les données capteurs. Pour ce faire, on y a introduit le caractère 'Z' à la fin, pour pouvoir à la réception détecter la fin de la chaîne. Voici le bout de code qui reconstitue cette chaîne :

```
void receiveString(char chaine[500])
{

  while(Serial.available()>0)
  {
    if(i==500) i = 0;
    chaine[i]=Serial.read();
    i++;
    if(chaine[i-1] == 'Z')
    {
       chaine[i-1] = '\0';
       for(i=0;chaine[i]!='\0';i++) Serial.print(chaine[i]);
       i=0;
    }
  }

void extractData(char chaine[500],float*CO2rate,float*temperature,float*humidity,float*temperatureExt)
{

  int i,j;
  j=0;
  char numbers[20];
  *CO2rate = 0;
  *temperature = 0;
  *humidity = 0;
  *temperatureExt = 0;

  for(i=0;chaine[i]!='\0';i++)
  {
    if(chaine[i]>= 0x30 && chaine[i]<= 0x39)
    {
      numbers[j] = chaine[i]- 0x30;
      j++;
    }
  }

  j = j-1;

  *temperature = numbers[0]*10 + numbers[1] + numbers[2]/10 + numbers[3]/100;
  *humidity =  numbers[4]*10 + numbers[5] + numbers[6]/10 + numbers[7]/100;
  *temperatureExt =  numbers[8]*10 + numbers[9] + numbers[10]/10.0 + numbers[11]/100.0;

  if(j==18)
  {
    *CO2rate =  numbers[14]*100 + numbers[15]*10 + numbers[16] +numbers[17]/10 + numbers[18]/100;
  }
  else if(j==17)
  {
    *CO2rate =  numbers[14]*10 + numbers[15]*1 + numbers[16]/10 +numbers[17]/100;
  }

  else
  {
    if(j==16)
    {
      *CO2rate =  numbers[14] + numbers[15]/10 + numbers[16]/100;
    }
  }

  *CO2rate = (*CO2rate * 1500.0)/513.0;

}
```

IV) Partie électronique

Comme chaque pièce a ces propres caractéristiques atmosphériques. Il a fallu centraliser notre système. Et donc créer une carte de commande (centrale), ainsi qu'une carte émettrice qui sera constituée de capteurs. La carte émettrice sera installée dans chacune des pièces de la maison, et la carte de commande sera placée à un seul endroit. Il est à noter que les cartes communiquent entre elles via le réseau Zigbee.

Le schéma suivant permet de comprendre l'implémentation de notre système dans une maison :

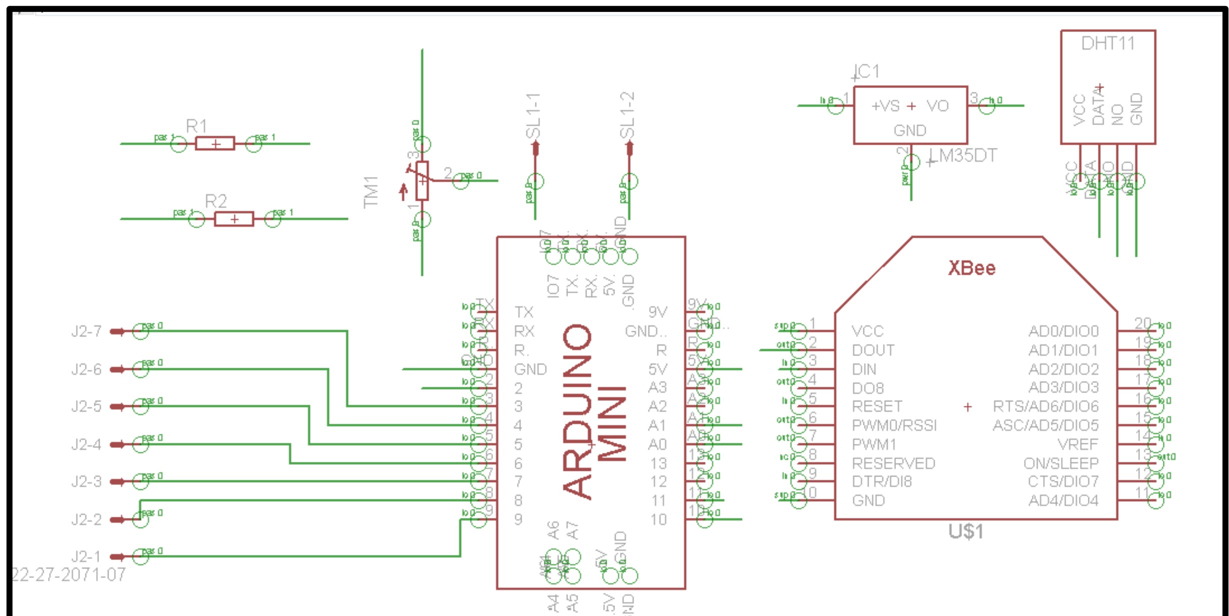


Pour faire le PCB, différents choix se sont présentés. Mais on a opté pour CadSoftEagle, qui est un logiciel de conception de cartes électroniques gratuit et Open-source donc très utilisé ce qui fait que le contenu est largement plus disponible sur internet comparé à ses concurrents.

La prise en main de ce logiciel a été néanmoins difficile, car nous utilisions pour nos précédents projets le logiciel Altium Designer, dont l'interface est assez intuitive contrairement à Eagle.

Nous avons commencé par établir l'architecture de la carte émettrice, qui est constituée comme on a cité avant, d'un : DHT11 (Humidité), LM35(Température), Potentiomètre(CO2), ainsi que de quelques résistances pour relier les capteurs à l'Arduino.

La carte émettrice est la suivante :

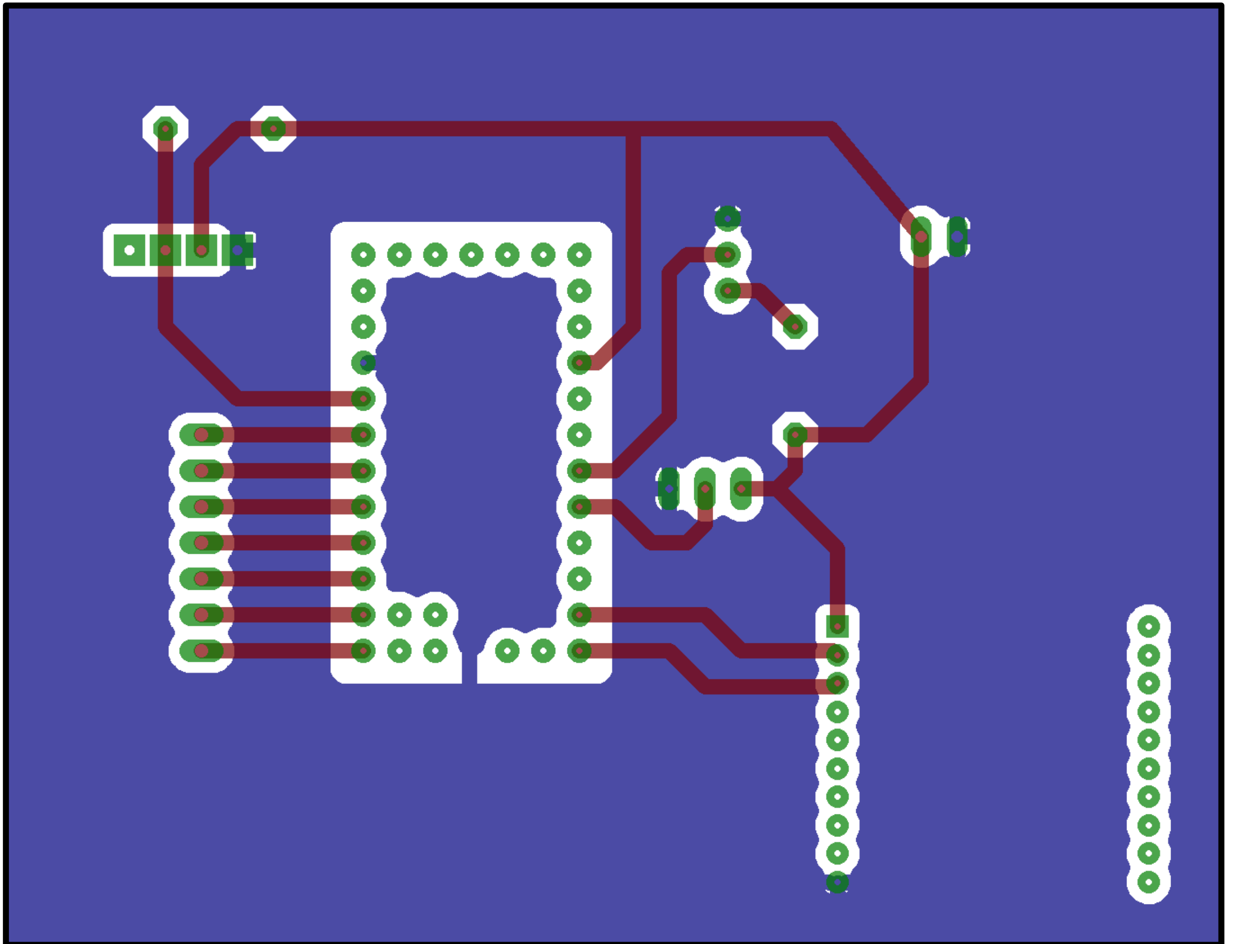


L'Arduino interroge les capteurs, un à un sur les valeurs qu'ils renvoient. Il les transmet par la suite au Xbee qui se charge de les transmettre à la carte de commande via son Xbee associé. La transmission est faite à une fréquence temporelle qu'on règle.

Une fois cette architecture fixée, on est passé à la réalisation du PCB. Cette étape est très importante, car il faut s'assurer que différentes normes sont respectées. Comme par exemple la taille des pastilles, qui a une valeur minimale, à ne pas dépasser, la taille des drills doit être minutieusement choisie afin de faciliter le perçage de la carte, la largeur des pistes …

Pour avoir une carte souple, et peu chargée, nous avons défini une face de la carte comme plan de masse pour alléger l'autre face qui ne recueillera que les pistes reliant nos composants.

Le PCB final de la carte émettrice est le suivant :

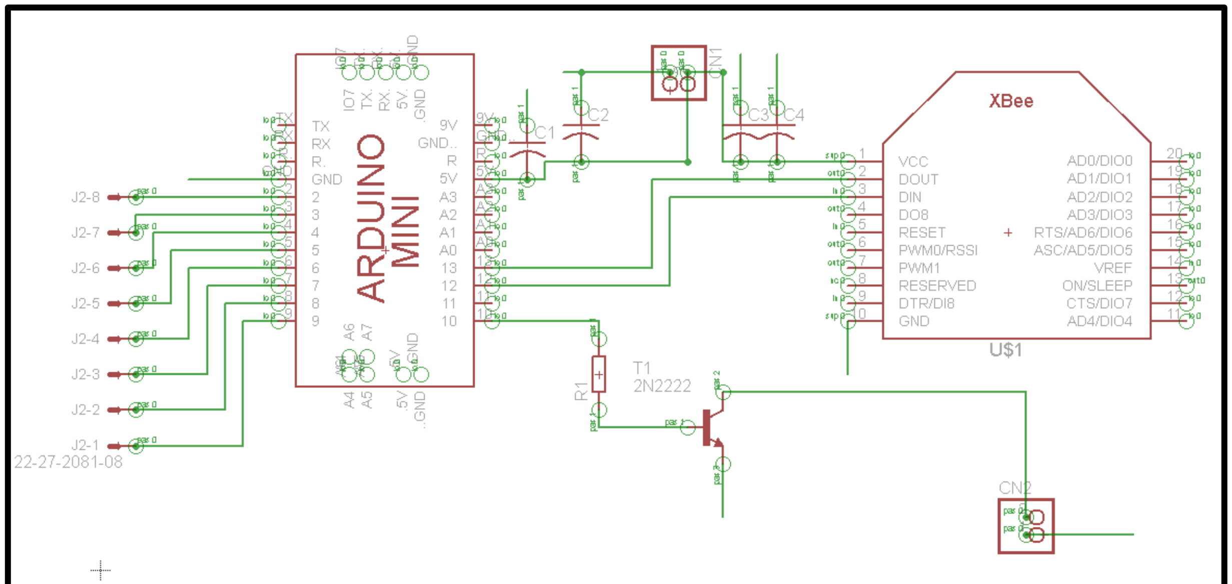


En rouge nous pouvons percevoir les liaisons entre les composants effectuées au top de la carte, alors qu'en bleu on peut voir le plan de masse qui relie les masses de chacun de nos composants.

Une fois le PCB de la carte émettrice terminé, nous avons commencé la carte centrale, qui servira de routeur aux différentes cartes émettrices.

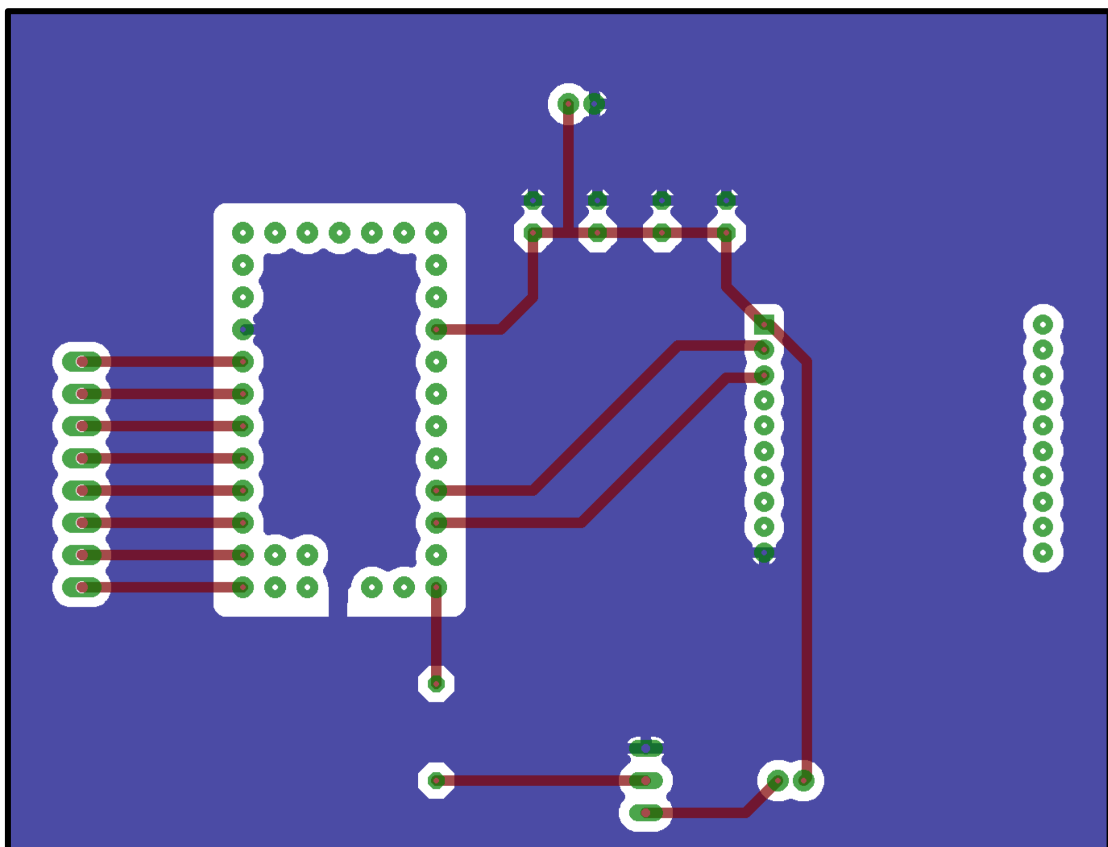
Cette carte se compose d'un Xbee qui se charge de récupérer les données via protocole Zigbee, d'un Arduino qui contient l'algorithme de commande décrit dans III), et de capacités de découplage qui servent à éliminer les parasites HF, augmentant donc l'immunité électromagnétique du circuit. Un transistor permet d'amplifier la puissance envoyée à la VMC.

L'architecture de la carte de commande est comme suit :

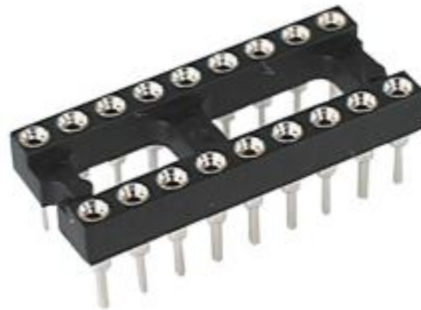


Une fois l'architecture définie sur Eagle, on passe au PCB. Et comme pour la carte émettrice, on utilisera le bottom comme plan de masse afin d'alléger la carte.

Le résultat est le suivant :



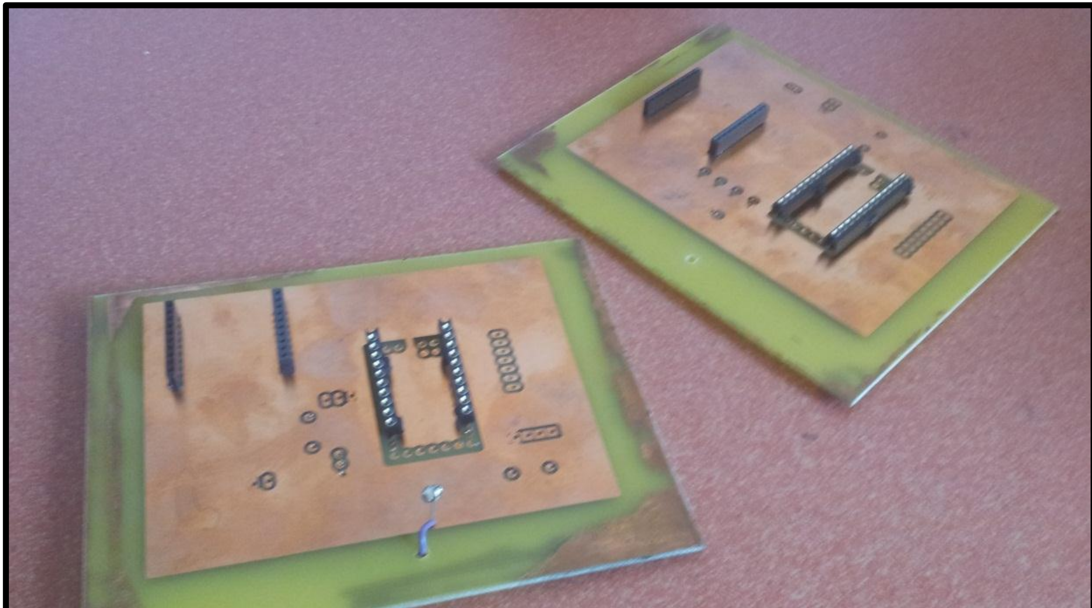
Les PCB finis, on les a envoyés à l'atelier électronique qui s'est chargé d'imprimer nos cartes. Une fois les cartes imprimées, on les a percés avec un foret de 0,8mm pour s'assurer de garder du cuivre sur les pastilles. Perçage fini, on a entamé le soudage des supports tulipes qui accueilleront nos composants.



(Support tulipe)

Une fois que nous avons fini le perçage et le soudage. A l'aide d'un multimètre on a vérifié que les liaisons électriques sont bien effectuées.

Le résultat est le suivant :

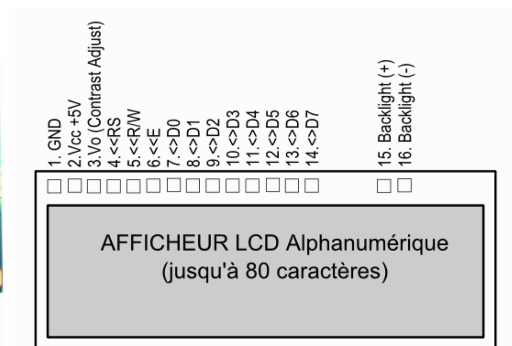


Cependant comme on le voit sur la photo, nous n'avons pas soudé tous les composants car on a été confrontés à l'impossibilité de téléverser notre code sur les Arduino Mini, même en utilisant un programmateur AVR MKII. Et donc il a été impossible d'utiliser les cartes, car on a opté finalement pour Arduino Uno, vu que le temps ne nous permettait pas d'en refaire des nouvelles.

On a décidé de relier notre système à un écran qui affichera à un intervalle donné, les valeurs du taux de CO_2, de la température extérieure, de la température intérieure, du taux d'humidité et de la vitesse de la VMC. Comme ça, un utilisateur lambda de notre système, pourra lire les valeurs des paramètres atmosphériques de l'environnement qui l'entoure. Et voir en temps réel l'évolution de la vitesse de sa VMC.

On a opté pour l'afficheur RGB d'Adafruit qui est facilement programmable, car le constructeur a mis en place des libraires en C qui réalisent l'initialisation de l'écran, l'envoi de chaînes de caractères, le changement de la résolution ...

(L'écran LCD d'Adafruit)



(Brochage de l'écran et Arduino)

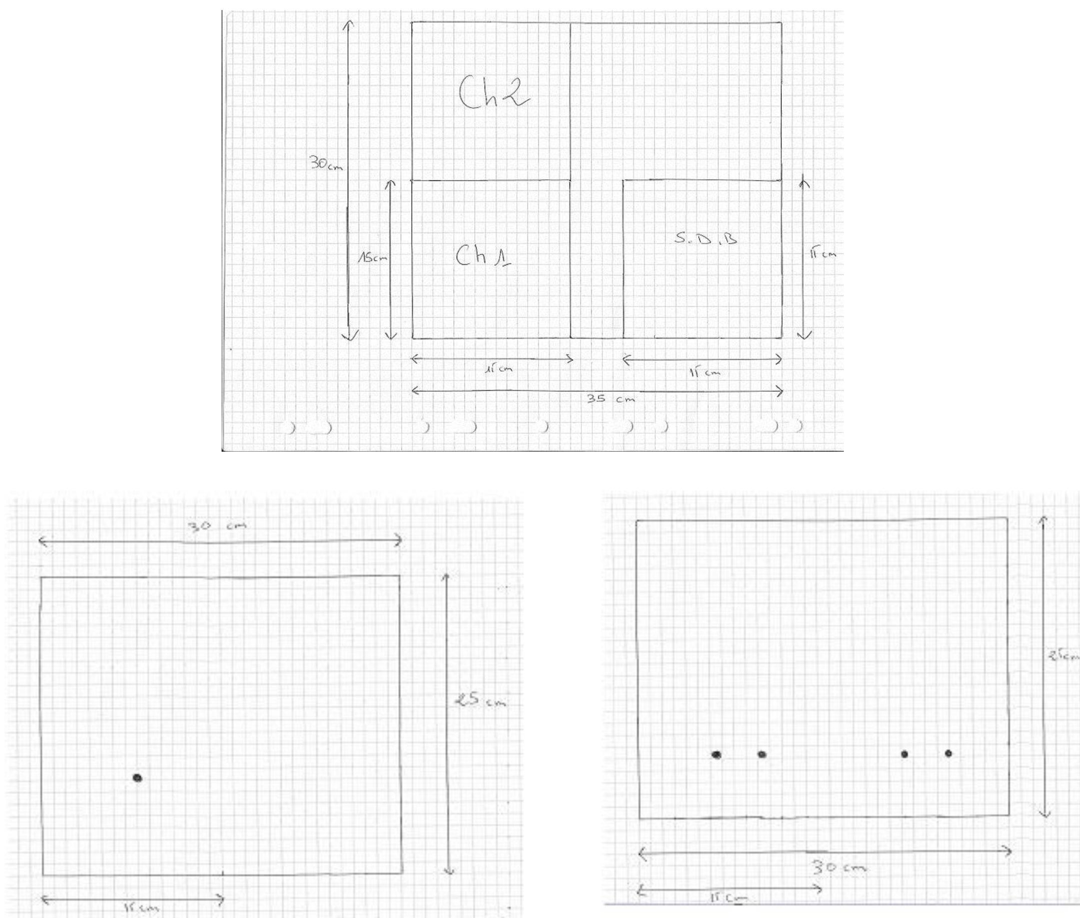
Pour afficher une chaîne de caractère donnée sur l'écran, il suffit d'utiliser le code suivant :

```
#include <Wire.h>
#include <EEPROM.h>
#include <LiquidCrystal.h>
#include <Bounce.h>
#include "DHT.h"
LiquidCrystal lcd(10, 9, 8, 7, 6, 5);
void setup() {
 lcd.begin(16,2);
 lcd.home ();

}
void loop() {
lcd.print("Bonjour !");
}
```

V) Partie maquette et tests

1) Partie maquette

Pour l’esthétique du projet, on a décidé de faire une maquette d’une maison 3 pièces. Pour ceci on a contacté l’atelier mécanique, un mois avant la date de la soutenance. Il nous fallait dessiner un plan de la maison, avec différentes vues. Les plans ont été transmis à l’atelier mécanique le 31 Mars 2014, et le responsable nous a assuré d’avoir la maquette au maximum 2 semaines après, à savoir le Vendredi 11 Avril. Cependant à cause des surcharges que connaissait l’atelier mécanique, on n’a pas pu avoir notre maquette en temps voulu, et une nouvelle date a été fixée qui était le 14 Avril.



(Plan de la maquette)

On a bien reçu la maquette de notre maison le 14/04, mais elle nous a été inutile car nous n'avions pas de PCB avec les composants dessus mais des montages « test » sur breadbord et donc peu adaptés pour être transvasés dedans.

2) Partie tests

Les tests ont été très concluants, car notre ventilateur (simulant la VMC) répondait exactement comme on avait prévu aux changements du climat extérieur qu'on lui infligeait (Chauffage, Augmentation/Réduction du taux de CO2, Changement de l'humidité). Selon ces facteurs, la vitesse de la VMC variant suivant le modèle qu'on a défini auparavant.

A titre d'exemple, une augmentation du taux de CO2 augmentait la vitesse de la VMC pour essayer de rendre l'air moins polluée, le dépassement d'un taux de 1000 PPM faisait tourner la VMC à sa vitesse maximale, car nous considérons qu'à partir de ce seuil l'air devenait extrêmement irrespirable. Ceci va de soit lorsqu'on augmente la température ou l'humidité.

VI) Conclusion

Ce projet nous a été très bénéfique, car il nous a confronté à la difficulté de définir soi-même un cahier des charges et de trouver par soi les solutions les plus efficaces pour réaliser le projet. De plus, on a pu appliquer la notion de commande floue que nous avons vu au cours de cette année, ainsi que les différentes notions acquises en réseau et en systèmes d'exploitation au cours du premier semestre.

De plus il nous a permis d'être autonomes, et d'apprendre à s'organiser et faire un travail d'équipe.