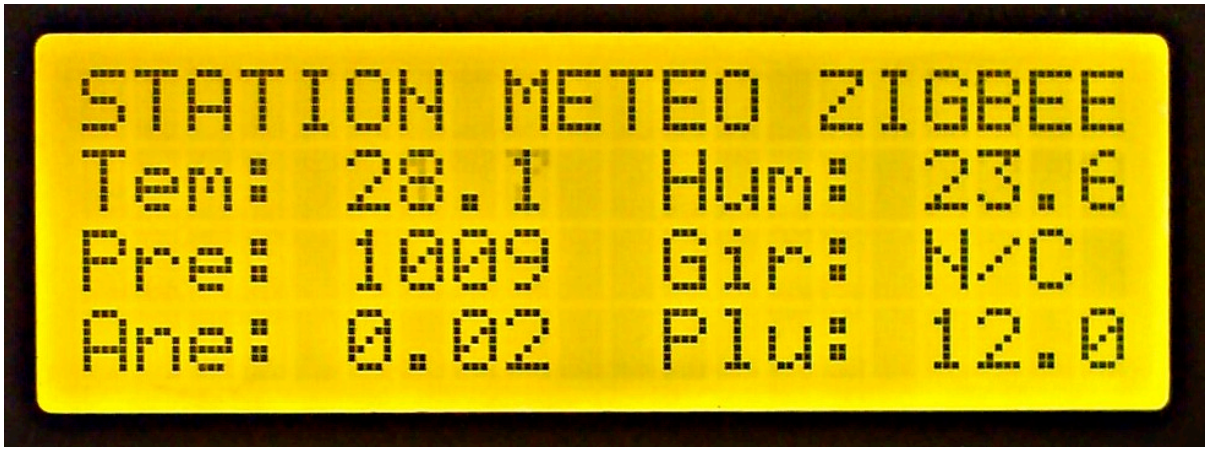
DOUROUX Thomas
MELLET Rémy
MONJANEL Benoît
THEVENET Emeric

Station Météo



Remerciements

A l'issue de ce projet, nous tenons à remercier notre tuteur et professeur de microcontrôleur M. Philippe Kauffmann, pour son aide et son encadrement durant ces deux périodes de projet.

Nous tenons également à remercier comme il se doit M. Christian Couderc, pour l'aide technique apportée.

Sommaire

Introduction

Pendant les deuxième et troisième périodes de notre deuxième année de DUT, nous avons dû réaliser un projet en groupes variables de trois à quatre personnes sur des sujets différents. Ce projet de fin d'année a pour but de nous amener à travailler en équipe et en autonomie sur une réalisation correspondant à nos compétences.

Nous avons choisi le sujet intitulé « station météo » pour appliquer nos acquis en matière de microcontrôleur à un projet complet et pour programmer sur un système matériel plutôt que de réaliser un logiciel. Enfin, parce que ce sujet correspondait bien à nos aspirations par son application très concrète et son aspect ludique.

Ce sujet nous mène à réaliser une station météo. Celle-ci est contrôlée par un microcontrôleur lié à une interface graphique, elle-même installée sur un ordinateur de bureau.

1. Etude du sujet

1.1 Elaboration du cahier des charges.

Le projet que nous avons choisi est donc la mise en place d'une station météorologique sans-fil autour d'un microcontrôleur nous permettant de traiter diverses données externes tels que la pluie, le vent, l'humidité, la pression barométrique et la température.

Le but était tout d'abord de gérer six capteurs (un capteur de température, un capteur de pression barométrique, un capteur d'humidité, un anémomètre, une girouette ainsi qu'un pluviomètre) par le biais du microcontrôleur Renesas de type M32C/80 et du module qui l'entoure. Nous devions programmer en langage C des pilotes pour chaque périphérique de la station (configuration, gestion des entrées / sorties…)

Le microcontrôleur devait ensuite pouvoir envoyer les données des capteurs vers un PC via une liaison RS232. Etant donné que la station se devait d'être sans-fil, nous avons donc utilisé un module ZigBee. La gestion de l'envoi des données du microcontrôleur vers le PC a également été codée en C.

Les informations venant des capteurs ainsi disponibles sur le PC, l'objectif était alors de les traiter en affichant des graphiques d'évolution de ces données en fonction du temps, le tout programmé en Tcl/Tk .

1.2 Choix des technologies utilisées

- L'atelier de génie logiciel HEW.

La contrainte principale qui nous était donnée dans le cahier des charges était d'utiliser le microcontrôleur 16bis Renesas M32C/80 à 32MHz. Nous avons par conséquent développé la partie logicielle à l'aide de High-performance Embedded Workshop (HEW).

HEW est un atelier de génie logiciel (AGL) fourni avec le microcontrôleur. Cet AGL est un environnement complet de développement qui intègre un débuggeur

simple, un débuggeur hardware (KD) et un simulateur (PD) très pratique pour travailler sans le microcontrôleur.

Nous avons donc développé et envoyé le code logiciel dans le microcontrôleur à l'aide de cette AGL.

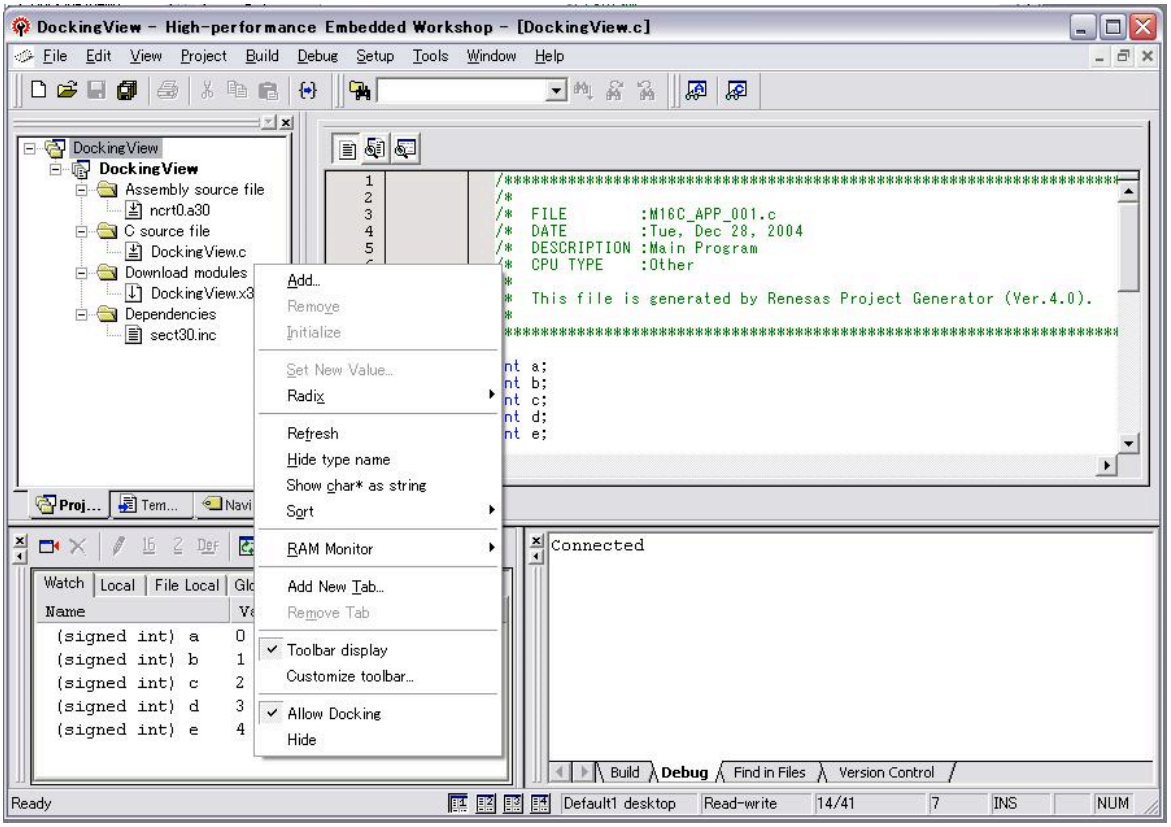


Figure 1 : Aperçu de l'interface de HEW

- Le langage de l'interface graphique

La deuxième contrainte était d'obtenir une interface graphique portable et facilement utilisable. Nous avons pour cela utilisé la bibliothèque Tk et le langage de script Tcl, via le logiciel Visual Tcl.
Nous reviendrons sur la partie Tcl/Tk un peu plus tard dans ce rapport.

1.3 Répartition des tâches.

Dans le but de terminer dans les délais, nous avons réalisé lors de la deuxième séance un WBS : Work Breakdown Structure. C'est un schéma qui représente les flux de travaux à réaliser au cours du projet.

Pour résumer notre WBS : dans les premiers temps nous avons ensemble recherché les composants, puis nous avons décidé que chacun s'occuperait d'un capteur. Une fois ce travail terminé, un groupe s'occuperait de la réalisation de l'interface graphique et l'autre de la liaison RS-232.

2. Les capteurs

2.1. Généralité

Pour pouvoir mettre en place les trois premiers capteurs (capteur température, humidité, pression), on a initialisé le convertisseur analogique numérique (CAN) en « sweep mode ». Ce mode permet de convertir plusieurs données analogiques successivement. Dans notre projet, nous avons seulement utilisé les entrées AN00 à AN02. Ce convertisseur analogique numérique transforme une tension comprise entre 0V et +5V en un nombre allant de 0 à 1023.

Ensuite, nous nous sommes aperçus que les données reçues des capteurs étaient bien souvent erronées. Ainsi pour minimiser ces erreurs, nous avons décidé de prendre dix fois plus de mesures et de faire la moyenne des dix dernières mesures.

Nous verrons ensuite le fonctionnement de chacun des capteurs.

Ci-dessous, le schéma global du câblage du microcontrôleur avec l'ensemble des capteurs.

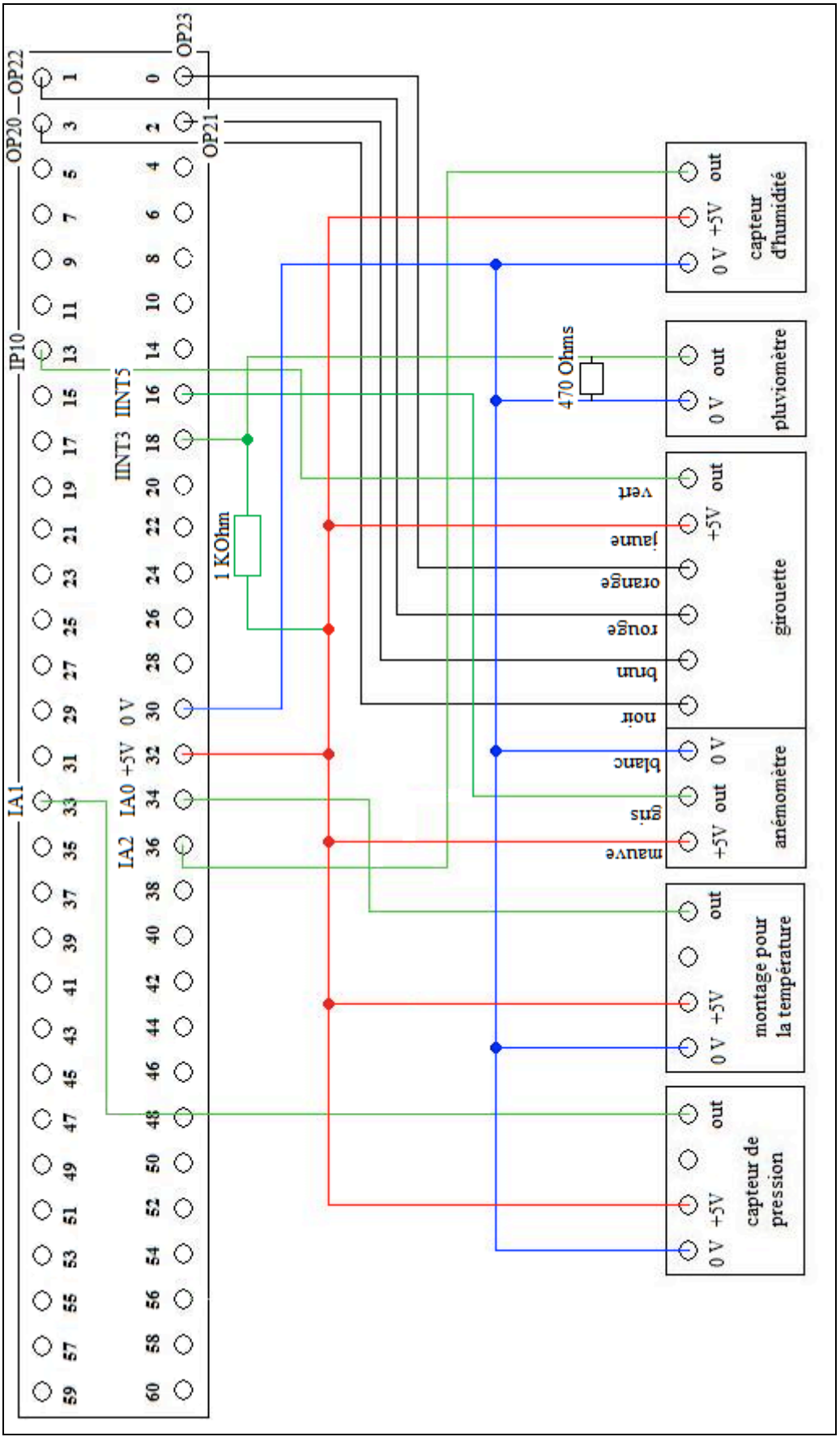


Figure 2 : Branchement des capteurs sur le microcontrôleur

2.2. *Capteur de température*

Pour obtenir la température, nous avons utilisé une sonde de type LM35 de chez National Semiconductor. Cette sonde est très pratique car elle est alimentée en +5V et est de type linéaire. De plus, en sortie de celle-ci, on obtient un signal linéaire défini par la caractéristique : 10.0 mV/°C.

Cette sonde peut fonctionner soit uniquement avec les températures positives, soit avec les températures positives et négatives (–55°C à +150°C). Evidemment, pour notre station météo, nous avons opté pour le mode positif/négatif et en choisissant une gamme de valeur, allant de –50°C à +50°C, ce qui correspond donc à des différences de potentiels allant de –500 mV à +500 mV.

Pour récupérer la température dans notre programme C, il suffit donc d'utiliser le CAN (Convertisseur Analogique Numérique) du microcontrôleur. Le CAN possède une gamme de valeur allant de 0 à 1023, ce qui correspond donc à des différences de potentiels allant de 0 à +5V.

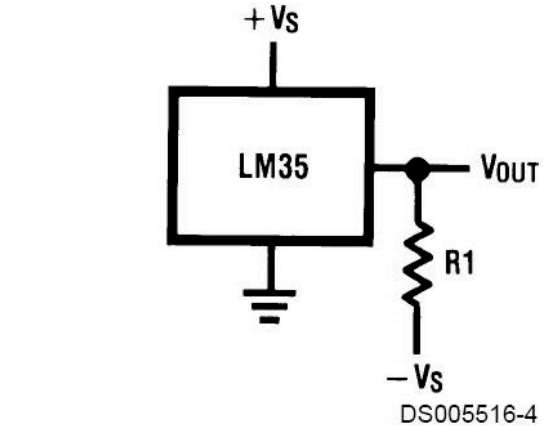
Ensuite on utilise le registre ad00 (0 à 1023), qui correspond au résultat de la conversion du CAN, puisque notre sonde de température est reliée à l'entrée IA00 du microcontrôleur.

Relation entre le résultat du CAN, la différence de potentiel et la température finale :

Température (°C) = (((ad00 * 5.000) / 1023) / 0.01)

On peut vérifier que cette relation fonctionne très bien avec un petit exemple :

Pour 0V en sortie de la sonde, donc pour 0°C :
En sortie du CAN on obtient 0 d'où Température (°C) = 0°C.



Choose $R_1 = -V_S/50\ \mu A$
V_{OUT}=+1,500 mV at +150°C
= +250 mV at +25°C
= −550 mV at −55°C

Figure 3 : Schéma de la sonde dans le mode températures positives/négatives.

Cela fonctionne très bien, mais il a fallu résoudre un petit problème qui nous a pris beaucoup plus de temps que prévu. Le problème est que le CAN où le capteur de température devait être relié, ne supporte pas les différences de potentiels négatives.

On a donc utilisé un montage à base d'amplificateur d'instrumentation, le INA 126 puis finalement le INA 128. Ce changement de composant est du au gain trop élevé de l' INA 126 et donc à un problème de saturation constante à la sortie du montage.

Avec l' INA 128, le montage fonctionne bien et le signal d'entrée (sortie du capteur de température) est amplifié avec un gain d'environ 4 :

Gain = 1 + (50K / (RG = 15K))

Bien sûr, en sortie de notre montage à base d'INA on obtient bien un nouveau signal décalé.
Ceci est vérifié par la relation de l'INA 128 :

VTemp = Gain * (Vin(+) – Vin(-)).

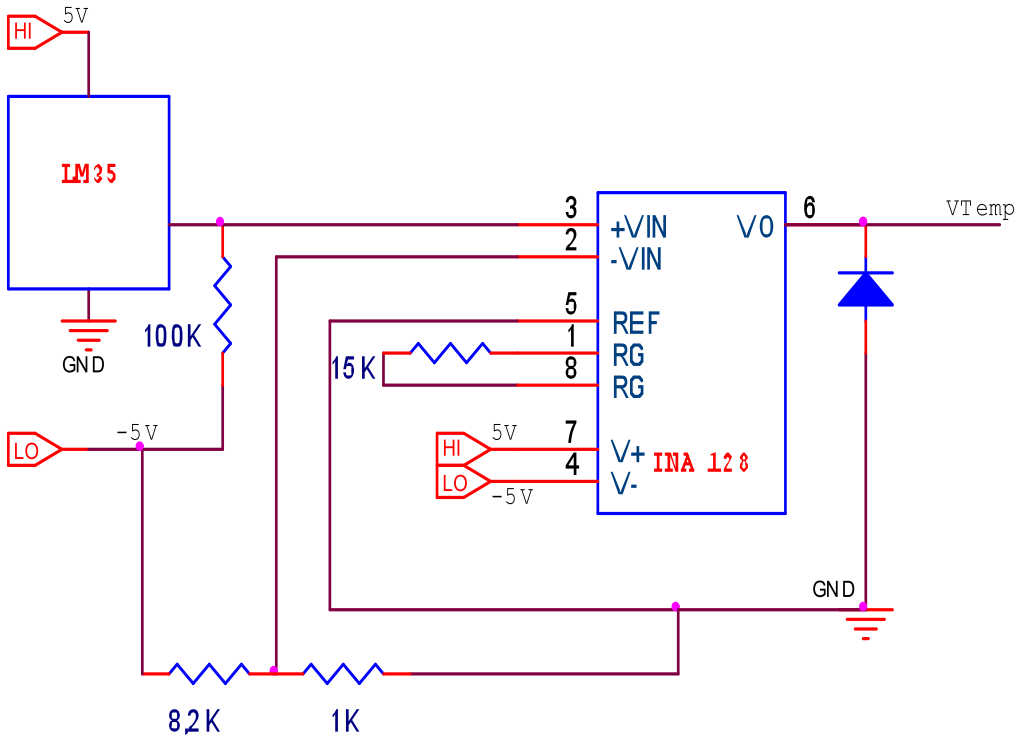


Figure 4 : Schéma complet du montage température à base d'INA 128

La diode en sortie du montage, se trouve ici uniquement pour s'assurer qu'aucune différence de potentiel négative ne puisse sortir du montage température et endommager le CAN. Effectivement, si la température devient inférieure à –50°C, il y aurait obligatoirement une différence de potentiel négative en sortie de l'INA 128 d'après la relation :

VTemp = Gain * (Vin(+) – Vin(-))
VTemp = 4.33 * (Vin(+) – (-0,5V))

Si la température devient inférieure à –50°C, on a donc Vin(+) qui devient inférieure à –0,5V.

VTemp = 4 * (-0,55V – (-0,5V))
VTemp = -0,2V !

A présent, on obtient en sortie de ce montage, les valeurs suivantes :

- Pour –50°C :
  On a –0.5V en sortie de la sonde d'où VTemp = 0V.
- Pour 0°C :
  On a 0V en sortie de la sonde d'où VTemp = 2,17V.
- Pour +50°C :
  On a +0.5V en sortie de la sonde d'où VTemp = 4,33V.

Maintenant, il ne reste plus qu'à déterminer la relation entre le résultat du CAN et la différence de potentiel, pour obtenir la température finale dans notre code C. En traçant sur un graphique, on obtient une droite linéaire d'équation :
f(VTemp) = (300 / 13) * VTemp – 50.

Relation entre le résultat du CAN, la différence de potentiel et la température finale :

Température (°C) = (((ad00 * 5.000)/1023)*(300/13)) – 50.

On peut vérifier que cette nouvelle relation fonctionne très bien avec un petit exemple :

Pour 0V en sortie du montage température, donc pour -50°C :
En sortie du CAN on obtient 0 d'où Température (°C) = -50°C.

A présent, le capteur de température ainsi que le montage à base d'INA 128, permettent d'obtenir une température avec une gamme de valeur allant de –50°C à +50°C et tout ceci sans endommager le CAN du microcontrôleur.

2.3 Capteur d'humidité

Comme pour chaque capteur, il a fallu nous plonger dans la documentation du capteur d'humidité alias SY-HS-230B. Après lecture de celle-ci, nous avons pu trouver le branchement du capteur ainsi que le schéma représentant le voltage retourné en fonction de l'humidité ambiante.

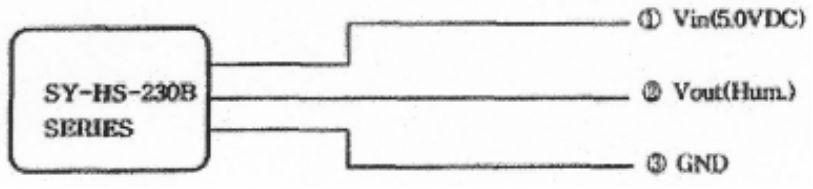


Figure 5 : Brochage du capteur d'humidité

Ce capteur utilise un branchement minimal. En effet il ne comporte que trois broches. Il s'alimente en 5 volts. Le résultat retourné par ce capteur sort de la broche centrale. Cette broche est connectée à l'entrée IA02 du microcontrôleur.

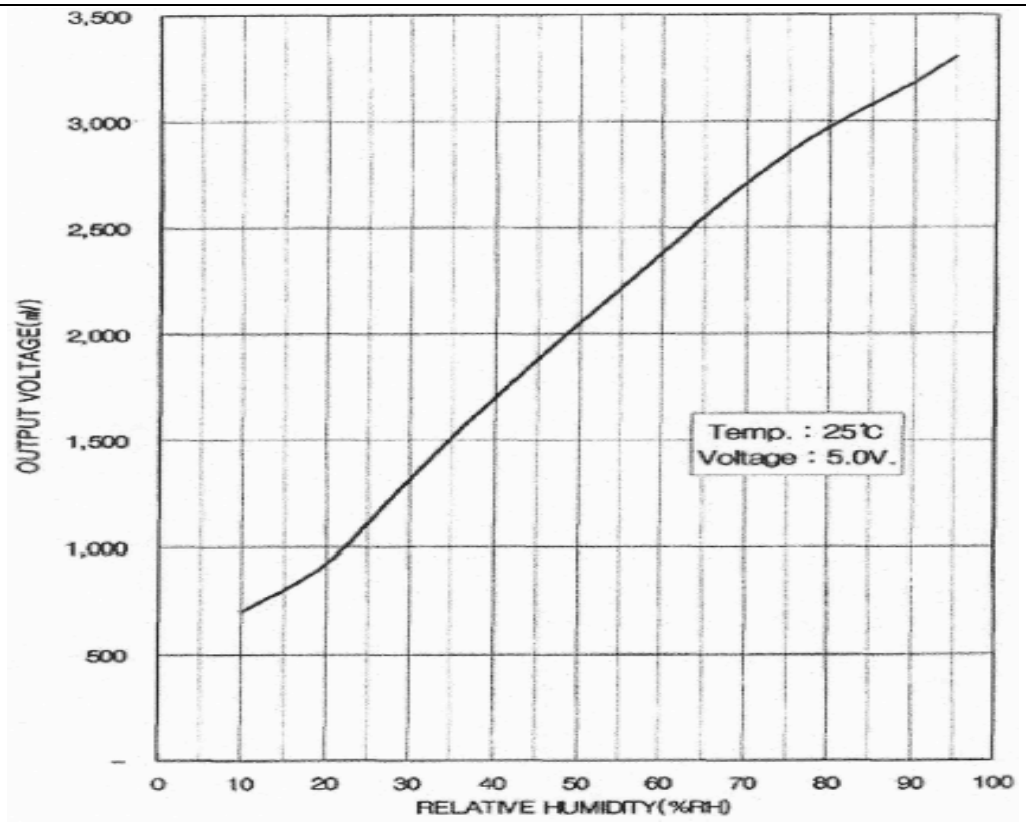


Figure 6 : Fonction de transfert du capteur d'humidité

Après étude de ce graphique, pour une humidité « normale », c'est-à-dire comprise entre 10% et 95%, on peut approximativement trouver cette formule :

Taux humidité = 30 * Vout – 10. (Résultat en pourcentage)

2.4 Capteur de pression

Nous avons utilisé un capteur barométrique MPX5100 amplifié. La documentation du composant nous a permis de comprendre que le branchement nécessaire à notre usage est le branchement minimal, c'est-à-dire utilisant uniquement la broche d'alimentation en 5V, la broche de masse et celle de sortie reliée a l'entrée IA01 du microcontrôleur.

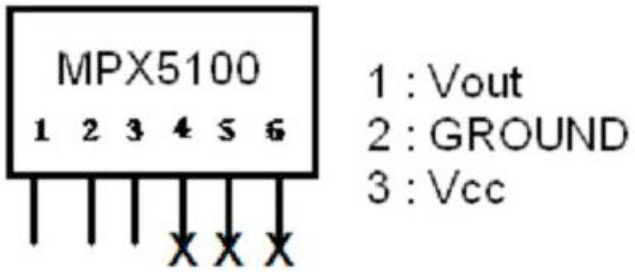


Figure 7 : Brochage du capteur de pression

La documentation nous a appris que la tension de sortie du capteur était fonction de la pression mesurée. Nous en avons déduit la fonction de transformation suivante :

Pression Mesurée (hPa) = (Vout / 5 + 0,095) / 0,0009.

2.5 Anémomètre

A chaque demi-tour, l'anémomètre émet une impulsion. La fréquence de cette impulsion est proportionnelle à la vitesse de rotation de l'axe de l'anémomètre. Dans un premier temps, nous avons voulu mesurer la fréquence de rotation de cet axe à l'aide d'un timer en mode « Pulse period measurement, pulse width measurement». Nous ne sommes pas arrivés à récupérer la valeur de la période, sans doute à cause d'un problème de configuration du timer. Les calculs de vitesse du vent associés à cette méthode auraient été ceux ci-dessous:

Pulsation : $\omega = 2.\pi.F$ avec $F = 1/T$

T est la période donc le temps que met l'anémomètre pour faire un tour. Comme on a deux impulsions par tour, la période que calcule le microcontrôleur est en fait la moitié du temps que met l'anémomètre pour faire un tour. On choisit donc de multiplier par deux la valeur de T. On prend maintenant pour T le temps que met l'anémomètre pour faire un tour. D'où $\omega = 2.\pi.(1/T)$.

La pulsation est proportionnelle à la vitesse de rotation, et inversement proportionnelle à la période. Nous souhaitons connaître la vitesse du vent. C'est la distance parcourue par unité de temps, cette distance est une longueur. Soit un point M de l'anémomètre qui fait un tour, il parcourt donc une distance : c'est la circonférence du cercle formé par la rotation de ce point M, et la mesure de cette distance est $2.\pi.Rayon$.

$V = distance / temps = (2.\pi.R) / T$

Sachant que $\omega = (2.\pi) / T$, alors : $T = (2 * \pi) / \omega$

Finalement $V = (2.\pi.R) / T$

$= (2.\pi.R) / ((2 * \pi) / \omega)$

$= (2.\pi.R.\omega) / (2.\pi)$

donc $V = R.\omega$ avec $\omega = 2.\pi.(1/T)$

Ensuite, nous avons voulu mettre le timer en mode comptage, mais nous ne sommes toujours pas parvenus à réaliser cette configuration.
Nous avons donc décidé de faire autrement sans gérer de timer en comptant nous même le nombre d'impulsions à l'aide de l'interruption INT5. Ainsi à chaque interruption nous incrémentons une variable nous servant de compteur. Puis, toutes les secondes, nous mémorisons ce nombre qui juste après est réinitialisé.

Le nombre mémorisé représente le nombre d'impulsions par seconde.
A chaque envoi d'information vers le PC, nous convertissons ce nombre d'impulsions par seconde en kilomètre par heure à l'aide de la formule ci-dessous :

Sachant que le périmètre d'un cercle est $P = 2.\pi.R$
On peut représenter l'anémomètre par un cercle de rayon R, R étant la distance entre l'axe de l'anémomètre et le milieu des palmes.
A chaque impulsion, le vent parcourt $\pi.R$ mètres, car il y a deux impulsions par tour.
La distance parcourue par le vent est :
Distance = $\pi.R.$(nombre d'impulsion)
Le nombre d'impulsion étant récupéré et réinitialisé toutes les secondes, on trouve :

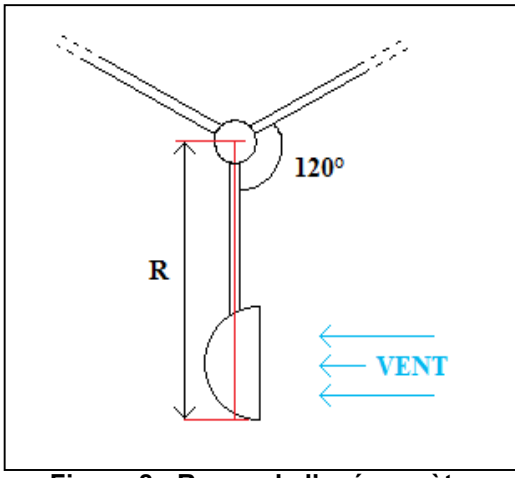


Figure 8 : Rayon de l'anémomètre

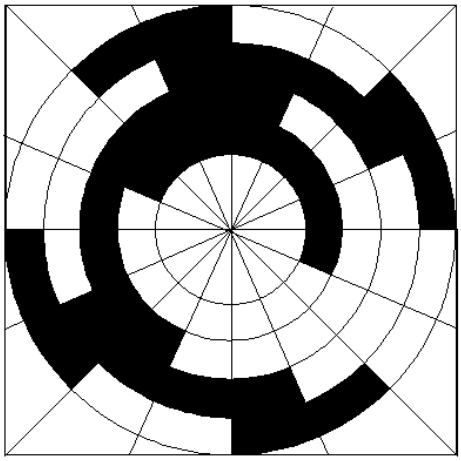
On veut dans un premier temps le résultat en mètre/seconde (m/s) :
V = distance / temps
V = distance / 1
V = $\pi.R.$(nombre d'impulsion) m/s

On veut maintenant transformer ce résultat en km/h :
Km / h = (m*1000) / (s * 60 * 60)
Ainsi, **v** m/s ⇔ 3,6***v** km/h

Donc :
Vitesse du vent = $\pi.R.$(nombre d'impulsion par seconde).3,6
= 3,14 * 0,035 * 3,6 * nb_impulsions
= 0,40* nb_impulsions

2.6 Girouette

La girouette est le capteur qui nous permet de connaître la direction du vent. En effet, un codeur optique est associé à l'axe pivotant de la girouette. Avec un codeur à 4 pistes comme le nôtre, on pourra différencier 2^4 soit 16 positions angulaires donc 16 directions de vent. En principe, le fonctionnement d'une telle girouette est le suivant : sur une plaque émettrice on dispose un alignement de 4 diodes infrarouge et, en face d'elles, un jeu de 4 récepteurs qui reçoivent le faisceau IR s'il n'est pas arrêté. On dispose donc entre émetteurs et récepteurs un disque solidaire de l'axe de la girouette comportant des zones opaques gravées en code gray.



Position n°	codage binaire	direction du vent
1	1 1 1 1	NORD
2	1 1 1 0	NNE
3	1 1 0 0	NE
4	1 1 0 1	ENE
5	1 0 0 1	EST
6	1 0 0 0	ESE
7	1 0 1 0	SE
8	1 0 1 1	SSE
9	0 0 1 1	SUD
10	0 0 1 0	SSO
11	0 0 0 0	SO
12	0 0 0 1	OSO
13	0 1 0 1	OUEST
14	0 1 0 0	ONO
15	0 1 1 0	NO
16	0 1 1 1	NNO

Figure 9 : Correspondance entre le code gray et la direction du vent

Le code Gray est fréquemment utilisé dans les capteurs angulaires ou de positionnement, mais aussi lorsque l'on désire une progression numérique binaire sans parasite transitoire.

Le code Gray est un code construit de telle façon qu'à partir du chiffre 0 chaque nombre consécutif diffère du précédent immédiat d'un seul digit. En l'exprimant autrement nous pouvons également dire que l'on change un seul bit à la fois quand un nombre est augmenté d'une unité. Si une erreur survient lors d'une transformation d'un nombre à un autre elle est ainsi minimisée.

Cependant, dans notre cas, on a bien quatre diodes émettrices mais seulement un récepteur comme le montre la figure ci-dessous.

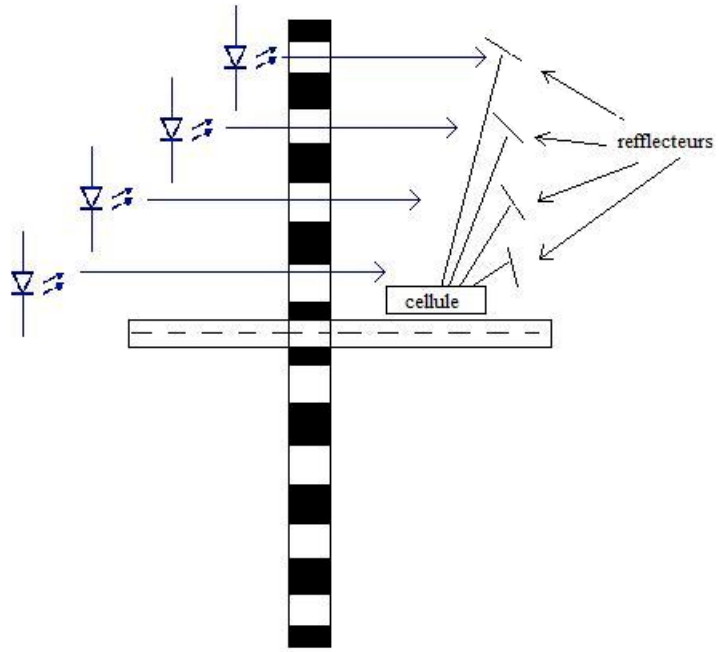


Figure 10 : Quatre diodes émettrices avec une seule cellule réceptrice

Au lieu d'alimenter les 4 diodes en même temps et de récupérer un code gray sur 4 bits directement en sortie de la girouette, il faut alimenter les diodes les unes après les autres et voir si le faisceau infrarouge passe pour chacune de ces diodes.

Il faut donc allumer une première diode, on vérifie si le faisceau passe : si oui, on met un 1 dans le bit correspondant au mot en binaire réfléchi sinon on met un 0. On éteint ensuite cette diode et on fait cela pour chacune des diodes du mécanisme.

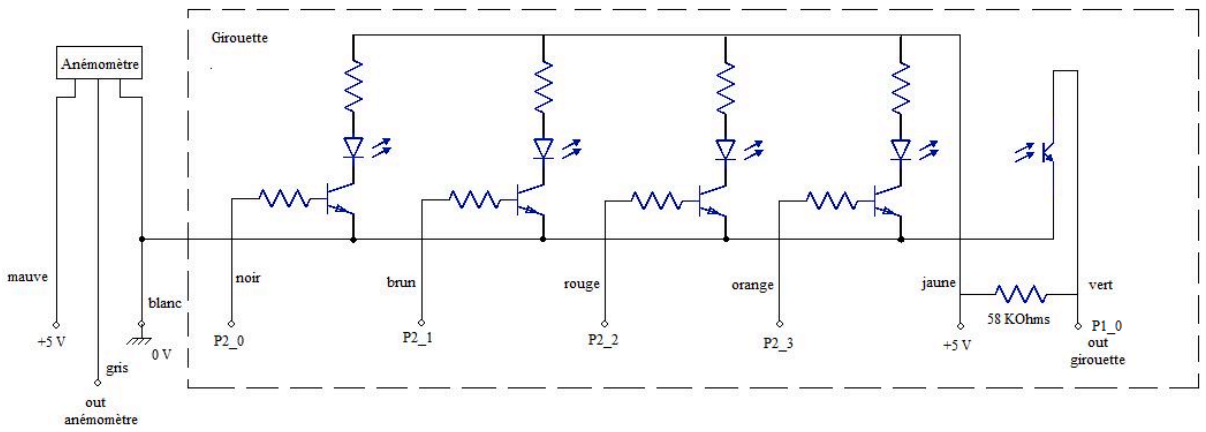


Figure 11 : Schéma interne de la girouette

On obtient ainsi notre code gray sur 4 bits qui est converti en code binaire naturel puis en un angle en degré par la fonction ci-dessous :

```
void direction_vent()
{
  char position=0;
  char pos;

  /* On alimente la diode correspondant au bit de poid fort du code gray. */
  p2_3=1;
  /* On attend 10 µs. */
  tempo_us(10);
  /* On met dans la variable position destinée a contenir les 4 bits du code gray
  la valeur de p1_0 qui correspond à la sortie de la girouette (1 ou 0 suivant
  si le faisceau passe ou non. */
  position|=p1_0;
  /* On fait maintenant un décalage à gauche pour qu'à la fin chaque bit recupéré
  sur p1_0 corresponde à sa position sur le code gray et pour ne pas ecraser la
  valeur mémorisée */
  position=position<<1;
  /* On éteint alors la diode  et on fait la même chose pour chaque diode */
  p2_3=0;

  p2_2=1;
  tempo_us(10);
  position|=p1_0;
  position=position<<1;
  p2_2=0;
  p2_1=1;
  tempo_us(10);
  position|=p1_0;
  position=position<<1;
  p2_1=0;
  p2_0=1;
  tempo_us(10);
  position|=p1_0;
  position=position<<1;
  p2_0=0;
  /* On traduit le code gray (binaire réfléchi)
  en binaire naturel */
  pos=traduire_BinRef_BinNat(position);
  /* on retourne la valeur recupérée en un angle en degré
  comme il y a 16 position, 360/16 = 22,5 */
  return (float)(pos+1)*22.5;
}
```

2.7 Pluviomètre

Le pluviomètre de la station est un pluviomètre à auget basculeur (ou à basculement), ce mécanisme évite d'avoir à vider le pluviomètre manuellement et est surtout obligatoire pour une station telle que la nôtre puisque tout est électronique et que les capteurs doivent se gérer de façon autonome. La pluie tombe dans un réceptacle puis s'écoule dans un des deux compartiments de l'auget basculeur. Lorsque l'eau de l'un des compartiments atteint un certain volume, l'auget bascule et le deuxième compartiment se met en place sous le réceptacle. Chaque inclinaison de l'auget est enregistrée. Une fois que les données sont consignées, l'eau se déverse dans la partie inférieure de l'auget et s'écoule.

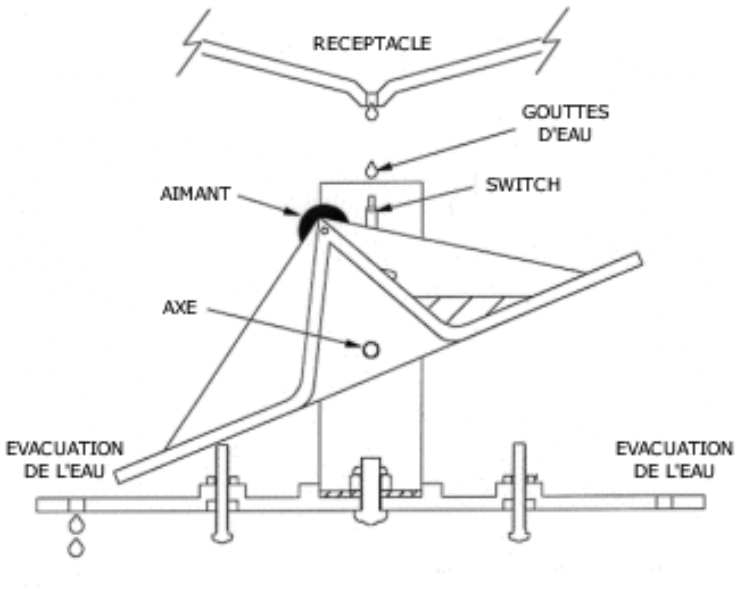


Figure 12 : Principe de fonctionnement (partie mécanique)

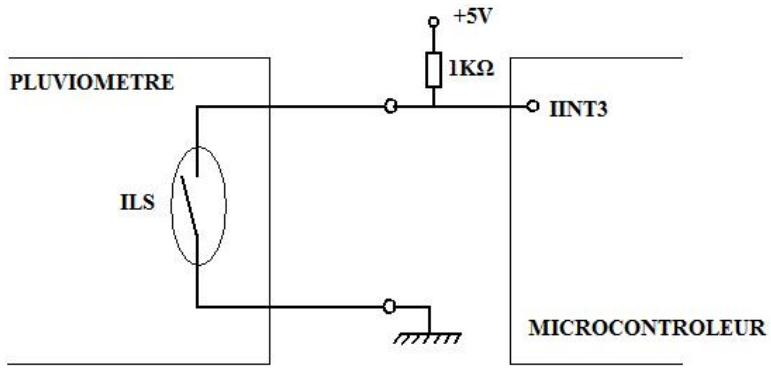


Figure 13 : Principe de fonctionnement (partie électrique)

Le contact est en fait un capteur ILS (Interrupteur à Lame Souple), voir schéma ci-dessus. C'est un capteur de proximité composé d'une lame souple et sensible à la présence d'un champ magnétique mobile.
Lorsque le champ se trouve sous la lame, il ferme le contact du circuit provoquant la commutation du capteur.

Il nous envoie donc une impulsion électrique à chaque basculement qui lance une interruption dans notre programme via la broche INT3 du microcontrôleur. Dans notre cas, une impulsion délivrée par ce capteur correspond à une précipitation de 0,8 mm d'eau.

3. Liaison entre la Station Météo et un PC

Pour relier la station météo à l'interface graphique, le cahier des charges nous imposait l'utilisation d'une liaison sans-fil ZigBee pour transmettre la donnée. Dans un premier temps, pour que nous puissions débuter au plus tôt l'interface graphique, nous avons mis en place la liaison RS-232.

3.1 Liaison RS-232

La liaison RS-232 est un protocole de communication datant des années 50. Il a été conçu pour relier un DTE (Data Terminal Equipment ou Element Terminal de Traitement de Données), plus simplement un ordinateur à un DCE (Data Communication Equipment ou Elément Terminal de Communication de Données), c'est-à-dire un modem. Cette liaison a l'atout d'être simple à utiliser contrairement à une liaison USB par exemple.

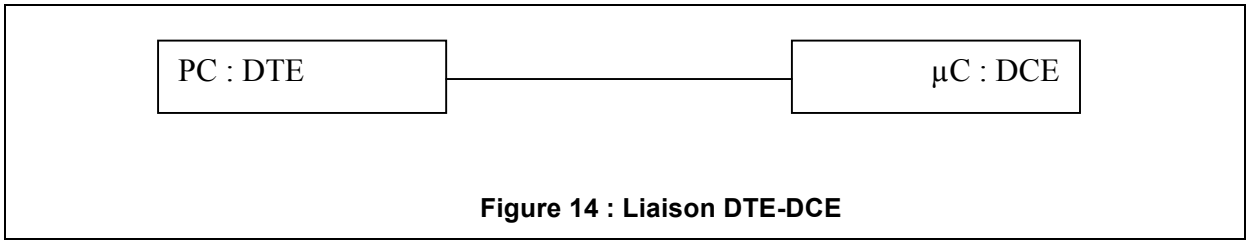


Figure 14 : Liaison DTE-DCE

- Comment initialiser la communication ?

Pour initialiser notre communication, nous devons indiquer la vitesse de transmission (en Bauds par seconde), la parité (paire, impaire ou sans), le nombre de bits de données par caractère (5, 6, 7 ou 8), le nombre minimal de bits d'arrêt et le mode de contrôle de flux (matériel, logiciel ou absent).

Cette configuration se fait via cinq registres :

- Le registre u0brg : permet de configurer la vitesse de transmission. Celle-ci est de 9600 bauds ce qui est dans notre cas amplement suffisant.
- Configuration du registre u0mr en mode 8 bits, avec l'horloge en interne, 1 bit d'arrêt et pas de parité.

- Configuration des registres de contrôle u0c0 et u0c1 en mode /CTS avec comme source d'horloge f1, le flag actif lorsque le buffer de transmission est vide, la transmission active, la réception inactive et le signal d'émission non inversé.

- Configuration de s0tic de manière à ce que l'interruption de transmission soit active.

- Comment envoyer un caractère ?

Le buffer de transmission est le registre u0tb. Donc il suffit de mettre une donnée (de 8 bits) dans u0tb pour qu'elle soit émise.

- Comment envoyer une chaîne de caractères ?

Lorsque que l'émission d'un caractère est terminée, l'interruption 17 est active. Ainsi, il suffit de déclarer la chaîne à envoyer globale ainsi qu'un entier représentant le numéro du caractère suivant à envoyer et à chaque nouvelle interruption, envoyer le caractère suivant si le caractère de fin de chaîne n'est pas trouvé.

On notera que pour que la chaîne de caractère soit considérée comme une ligne par l'interface graphique, il faudra qu'elle se termine par '\r\n'.

Voici un code simplifié permettant l'envoi de la chaîne 'Hello World !!! ' :

```
char chaineEnvoie[50] ;
int  carEnvoie ;

void envoiChaine(char* ch){
      strcpy(chaineEnvoie,ch);
      strcat(chaineEnvoie,"\r\n");
      carEnvoie=1;
      u0tb=chaineEnvoie[0] ;
}

// tant que le caractère de fin de la chaîne n'est pas
rencontré
// on envoie le caractère suivant
#pragma INTERRUPT 17 envoiCaractere

void envoiCaractere(){

      if(chaineEnvoie[carEnvoie]!=0){

          u0tb=chaineEnvoie[carEnvoie];

          carEnvoie++;

          adst_ad0con0=1;

      }

}


Dans la fonction main() :

envoiChaine('Hello World !!! ') ;
```

3.2 Liaison xBee

La liaison xBee est faite de deux modules ZigBee. Cette liaison permet de transmettre des données en utilisant les ondes. Le xbee est un peu comme le bluetooth du monde industriel mais avec une plus grande portée (environ 50m pour notre module zigBee).

Coté matériel, la mise en place du zigBee n'est pas compliquée – pour l'installer il faut avoir lu toute la documentation. - il suffit de brancher le premier module au PC sur le port USB et le deuxième module au microcontrôleur sur le port Com et s'assurer que la vitesse de transmission entre les deux modules est supérieure ou égale à celle entre le microcontrôleur et le deuxième module.

Coté PC (le module y est connecté via USB) l'installation d'un pilote émule un port Com ce qui permet de dialoguer avec le premier module (lié par USB) comme s'il s'agissait d'une liaison RS232. Le problème est que ce driver émule mal le port Com ainsi il est difficile de l'utiliser car d'après Windows, il est bien souvent « utilisé par une autre application ».

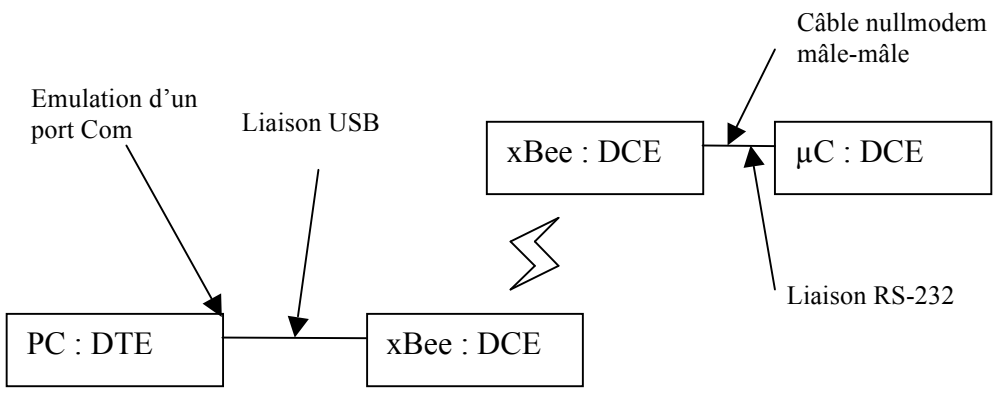


Figure 15 : Liaison ZigBee

4. L'interface graphique

Pour afficher les données reçues de la station météo, nous avons programmé une IHM (Interface Homme Machine) en TCL/TK. Ce langage permet de créer une interface graphique rapidement grâce à des fonctions de haut niveau.

Cette interface graphique ou GUI (Graphical User Interface), a pour objectif de représenter sous forme de courbes, les différentes variations météorologiques, pour une meilleure visualisation et cela grâce aux informations provenant des six capteurs.

Cette interface graphique réalisée en Tcl/Tk, est sobre, très simple d'utilisation donc peu gourmande en ressources.

4.1 Introduction à TCL / TK

Le langage Tcl est un langage de programmation conçu en 1989 pour le système d'exploitation UNIX. Un peu plus tard naît l'extension graphique Tk qui permet de développer des applications à fenêtrage indépendamment de la plate-forme.

Le Tcl est un langage de script, interprété, compilé à la volée, peu gourmand en ressources, multiplateforme, puissant et conçu pour être facilement étendu ou inclus dans une application (C, C++, etc.).

Sa syntaxe est tirée à la fois du SHELL, du C et du LISP, quelque peu inhabituelle mais basée sur quelques règles très simples :

- Tout n'est que commande, y compris les structures du langage et l'affectation.
- Tout peut être dynamiquement redéfini.
- Tout "type de données" peut être manipulé comme si c'était une chaîne de caractère, y compris le code.
- Gestion forte des événements.

- Portée des variables ou de commandes modifiable dynamiquement.

Pour que vos puissiez comprendre les exemples qui vont suivre, vous devez posséder quelques notions en TCL/TK. Attention, ces notions seront illustrées à l'aide de leur correspondance en langage C.

- ***Set*** permet d'attribuer une valeur à une variable.
  Par exemple : set toto 3 ⇔ int toto = 3 ; ou toto = 3 ;
- ***$*** permet de remplacer la variable par son contenu.
- **#** Permet d'ajouter un commentaire.
- **open** $numPort r+ ⇔ open(numPort,"r+");
- Les accolades ***{*** et ***}*** permettent d'indiquer à l'évaluateur jusqu'où il doit évaluer le code.
- ***eval*** est utilisé pour réévaluer la chaîne.
- ***CanvasTemperature cget -height*** permet de connaître la hauteur du canevas.
- En Tcl on peut utiliser les structures, voici un extrait d'une structure utilisée dans notre programme :

```
set Temperature(oldx) 0
set Temperature(oldy) [CanvasTemperature cget -height]
set Temperature(max)  50.0
set Temperature(min) -50.0
```

En langage C la structure complète de Température donne:

```
Struct CapteurGraphe{
   int oldx, oldy, x, y, min, max, val ;
} Temperature ;
```

4.2 Présentation de Visual TCL

Visual Tcl est un éditeur graphique libre et multi plateforme entièrement écrit en Tcl/Tk qui permet de concevoir facilement des interfaces graphiques.

Visual Tcl est le logiciel qui a été utilisé pour construire l'interface graphique. Il se présente sous la forme de huit fenêtres dont la fenêtre principale qu'on peut laisser ouverte ou fermée en fonction de nos besoins. Les fenêtres que nous avons utilisées sont :

- Attribute Editor : fenêtre permettant d'éditer nos widgets. On peut par exemple leur appliquer une couleur donnée, les ancrer à une certaine position par rapport à la fenêtre principale…
- Fonction List : fenêtre permettant d'éditer ou d'effacer une fonction existante, ou d'en créer de nouvelles. C'est le seul endroit dans Visual Tcl ou l'on peut insérer du code.
- Window List : fenêtre listant toutes les fenêtres de notre IHM.
- Widget Tree : fenêtre listant tous les widgets de notre IHM.
- Widget Toolbar : fenêtre listant tous les widgets tk.

Ce logiciel nous a permis de gagner beaucoup de temps car il nous a évité de coder manuellement notre interface graphique. En effet grâce à ce logiciel construire une IHM devient simple car il suffit de cliquer. Dans un premier temps, aucune notion en Tcl/Tk n'est demandée ; par exemple pour insérer un canevas, il suffit de cliquer sur « canevas » dans la fenêtre « Widget Toolbar » et de glisser ce widget dans la fenêtre de notre IHM.

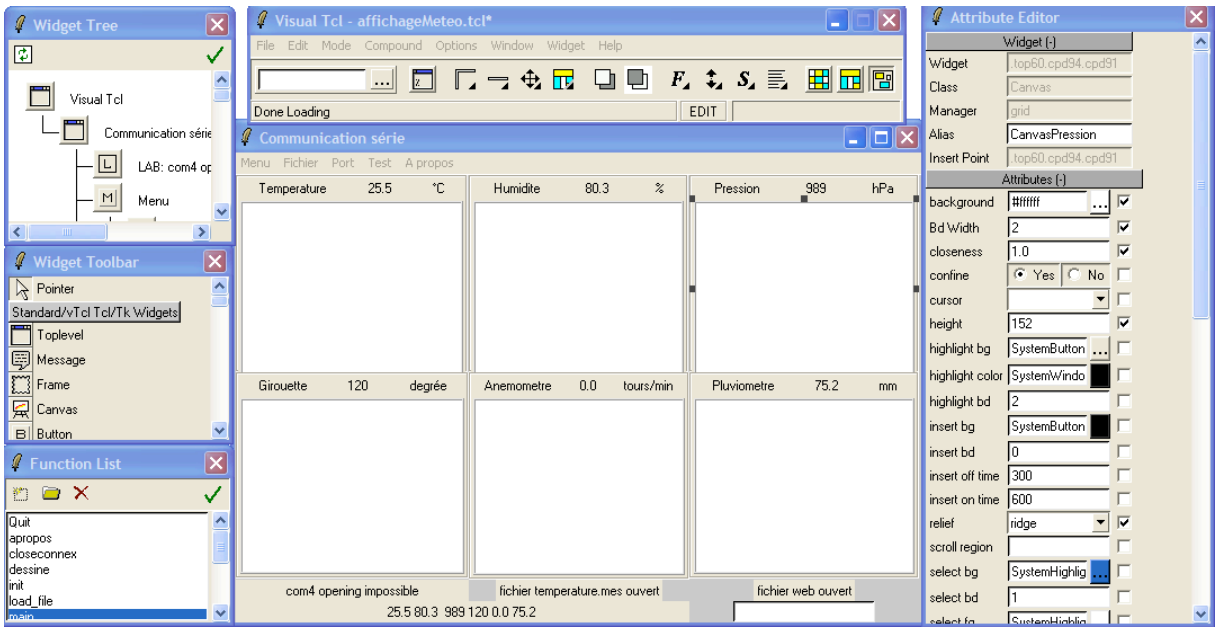


Figure 16: Aperçu de l'interface de Visual TCL

4.3 Notre interface graphique

Notre interface graphique possède plusieurs fonctionnalités, la principale est de représenter graphiquement les données émises par la base de la station météo. Elle peut aussi de stocker ces données dans un fichier pour pouvoir les recharger plus tard ou encore les exporter au format html pour qu'on puisse les visionner dans un navigateur Internet.

Nous allons maintenant voir étape par étape la réalisation de la fonction principale de notre IHM, c'est-à-dire la récupération des données ainsi que l'affichage des courbes représentant ces données (température, humidité, pression…).

- Réception de données

Dans un premier temps, nous avons seulement essayé de recevoir des données et de les afficher à l'écran dans un label.

Le code ci-dessous permet d'initialiser le port com2 de manière à ce que lorsque des données sont reçues, celles-ci soient mémorisées dans le buffer « ligne » et que la fonction read_line soit appelée. La fonction read_line ne fait qu'afficher le contenu de « ligne » dans un label.

```
   # Ouverture du port de communication et mémorisation du descripteur de
fichier dans portId
   set numPort 2    # port com2
   if [catch {open $numPort r+} portId] {
     # L'ouverture à echouée
     set status "impossible d'ouvrir le port $numPort"
     set estConnecte 0
   } else {
     # Ouverture réussi
     set status "$numPort ouvert"
     set estConnecte 1
     # Configure le port en mode 9600bps, pair, 8 bits par envoie et 1
bit d'arrêt. Utilise la variable ligne comme buffer
     fconfigure $portId -mode 9600,n,8,1 -blocking 0 -buffering ligne
     # Lorsque le port reçoit une donnée, appel la fonction read_line
avec l'argument « notest »
     fileevent $portId readable { read_line notest }
   }
```

- Extraction de l'information voulue

La donnée envoyée est une chaîne de caractère sous la forme :
<température><espace><humidité><espace><pression><espace><direction_du_vent><espace><vitesse_du_vent><espace><pluviométrie>

Exemple d'une chaîne de caractère que l'on peut recevoir de la station météo:
« 14 56 1100 270 10 3 »

Cette chaîne signifie qu'il fait 14°C, qu'il y a 56% d'humidité dans l'air, que la pression atmosphérique est de 1100HPa, que le vent souffle vers l'ouest (270° par rapport au Nord) à une vitesse de 10Km/h et qu'il a plu 3mm pendant les dernières 24h.

Etant donné que chaque information est séparée d'un espace, il est très facile en Tcl d'extraire l'information voulue.

Par exemple, si nous voulons avoir la pression atmosphérique, il suffit de rentrer la ligne de code ci-dessous :

```
lindex $ligne 2
```

- Affichage d'une courbe

Maintenant voyons comment afficher cette donnée dans un canevas afin de réaliser une courbe représentant l'évolution de l'une des informations reçue.

Le problème majeur de cette partie est que la gestion des canevas par Tk n'est pas très évoluée, en effet il est impossible d'appliquer une échelle à un canevas. Les coordonnées d'un point à afficher sur le canevas s'expriment seulement en pixel et de plus, la largeur du canevas ainsi que sa hauteur sont modifiées à chaque redimensionnent de l'IHM.

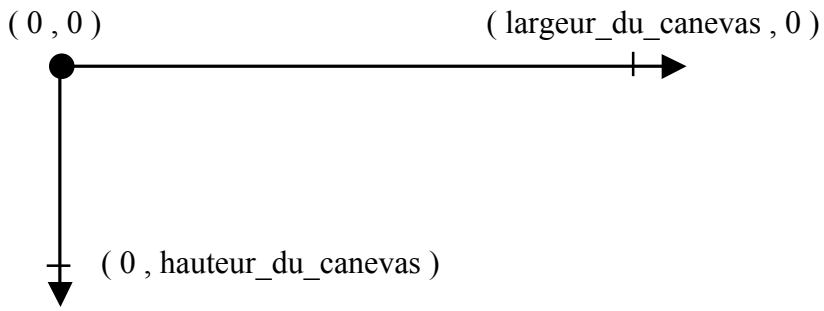


Figure 17 : coordonnées d'un point dans le canevas

Il a dont fallu trouver une fonction mathématique pour pouvoir afficher correctement les données reçues.

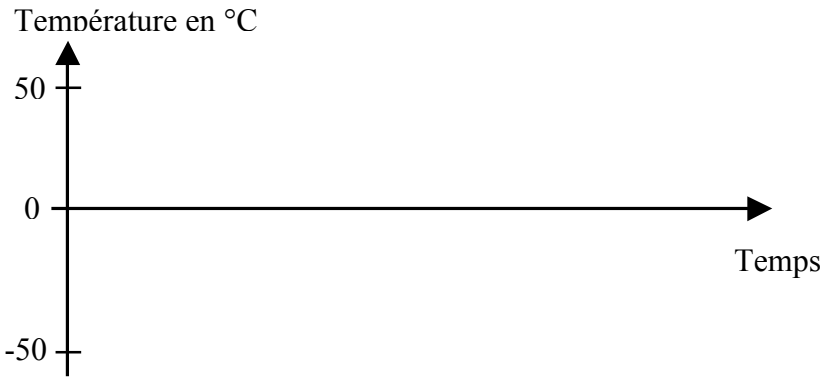


Figure 18 : Repère de la Température

Par exemple comment faire pour afficher une température allant de -50°C à +50°C sur un canevas dont la hauteur varie de 0 à hauteur_du_canevas ?

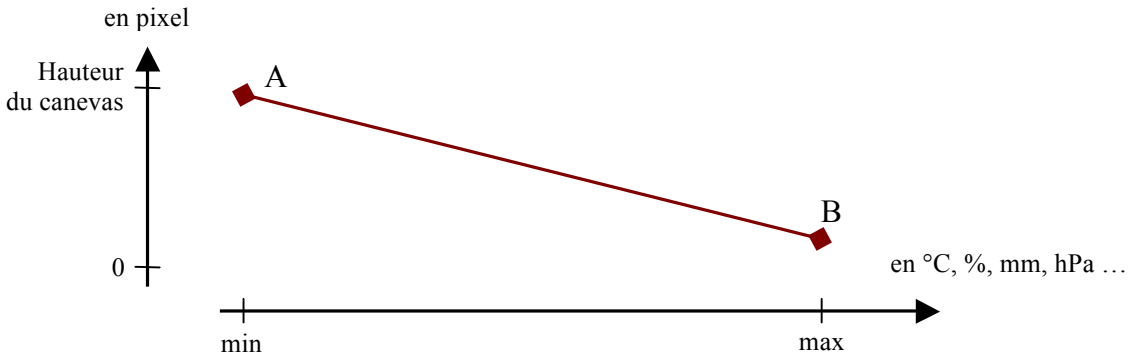


Figure 19 : Fonction permettant de trouver l'ordonnée correspondante à une donnée

f(donnee)=ordonnee_du_canevas_en_pixels.

Cette fonction se traduit par une fonction affine.

Essayons de trouver sa formule :

f(x)=ax+b

a = (yB-yA) / (xB-xA) = (0-hauteur_canevas) / (min – max)

On sait que f(max) = 0 on en déduit :

a * max + b = 0 ⇔ b = - a * max

Donc:

f(x) = a * x – a * max
= a (x – max)
= - (hauteur_canevas / (min – max)) (x – max)

Notre IHM doit pouvoir afficher les données provenant de six capteurs. Par souci de clarté nous avons choisi de ne mettre qu'une seule courbe par canevas. Nous avons donc six canevas à gérer, un pour chaque capteur.

Pour cela nous avons réalisé la fonction dessiner qui prend en argument le nom du capteur et la valeur à ajouter au canevas. Cette fonction trouve le canevas correspondant au capteur (ce canevas a comme nom : « Canevas » + nom_du_capteur, par exemple « CanevasTemperature ») et calcule les coordonnées du nouveau pixel à ajouter au canevas à l'aide de la fonction trouvée ci-dessus.

```
proc dessine {capteur val} {
      # les variables globale
      global Widget CanvasTemperature [...]

      # pour comprendre ce code, prenons par exemple :
      # capteur = 'Temperature' , val=14, hauteur du canvas = 200px

      set graphe "Canvas$capteur"
      # graphe vaut donc 'CanvasTemperature'

      eval set ${capteur}(val) $val
      # code équivalent à : set Temperature(val) $val
      # Temperature(val) = val = 14

      set a [expr - [$graphe cget -height] / [expr $${capteur}(max) - $${capteur}(min)] ]
      # a = - hauteur CanvasTemperature / ( Temperature(max) - Temperature (min) )
      # a = - 200 / (50 - -50) = -2

      set b [expr -1 * $a * $${capteur}(max)]
      # b = -a*Temperature(max) = -50a = 100

      eval set ${capteur}(y) [expr $a * $${capteur}(val) + $b]
      # Temperature(y) = a * Temperature(val) + b = -2 * 14 + 100 = -128

      eval set ${capteur}(x) $abscisse
      # Temperature(x) = abscisse = 1

      eval $graphe create line $${capteur}(oldx) $${capteur}(oldy) $${capteur}(x)
      $${capteur}(y)
      # CanvasTemperature create line $Temperature(oldx) $Temperature(oldy)
      $Temperature(x) $Temperature(y)
      # CanvasTemperature create line 0 0 1 -128

      eval set ${capteur}(oldx) $${capteur}(x)
      # Temperature(oldx) = Temperature(x) = 1
      eval set ${capteur}(oldy) $${capteur}(y)
      # Temperature(oldy) = Temperature(y) = -128
}
```

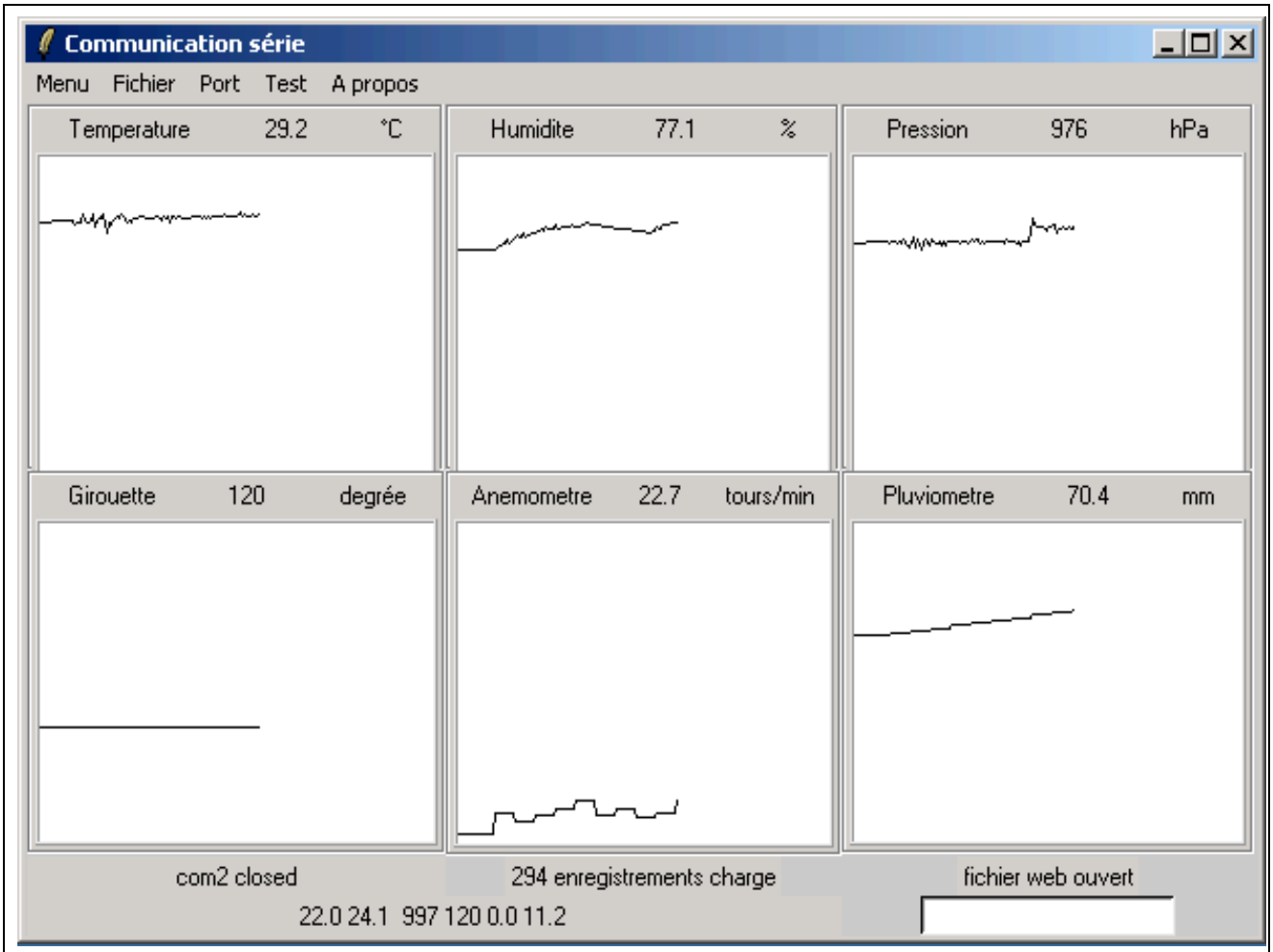


Figure 20 : Interface de la station météo

Les différents menus :

- Connexion : ouverture/fermeture de la connexion.
- Fichier : chargement / sauvegarde des données.
- Port : sélection du port com.
- Test : permet de tester l'interface sans la station (à l'aide du label situé en bas à droite)
- A propos

Bilan technique

Finalement notre station météo est fonctionnelle. Nous ne sommes cependant pas parvenus à faire fonctionner la girouette car le capteur de celle-ci semble ne pas fonctionner, en effet aucun signal ne parvient à l'oscilloscope lorsque celle-ci est convenablement branchée.

Ce projet nous a permis d'acquérir de grandes compétences techniques à la fois bas niveau comme la réalisation d'une liaison RS232, niveau intermédiaire avec la programmation d'un microcontrôleur, ainsi que haut niveau avec la programmation d'une l'interface graphique. Nous nous sommes rendus compte de la puissance du langage Tcl/Tk du fait de ses fonctions haut niveau, malgré sa syntaxe rébarbative.

Cependant l'interface graphique pourrait être améliorée. En effet les repères des différentes courbes ne sont pas tracés, ensuite il serait intéressant que l'application réalise des statistiques. Enfin, l'application de la logique floue, nous permettrait de réaliser des prévisions météorologiques à court terme de façon précise.

Conclusion

La réalisation de ce projet nous a apporté, sur un plan personnel, une vision différente du travail en équipe et une nouvelle façon de travailler. En effet, il a fallu diviser le travail en fonction des possibilités et implications de chacun. Nous avons tous été réactifs afin de trouver des créneaux horaires en commun. A l'inverse nous avons appris à travailler en autonomie sur une petite partie d'un projet plus grand. Nous avons su adapter notre comportement au travail réalisé par les autres membres de l'équipe.

Nous avons également fait face aux différents problèmes rencontrés avec calme et sérénité. Enfin nous avons pu appliquer nos acquis en matière de microcontrôleur. Et chacun a apporté aux autres un point de vue différent et une approche différente également dans sa démarche de résolution de problème.

Pour terminer, cette expérience a été plutôt ludique, en effet nous avons pris beaucoup de plaisir à réaliser ce projet fonctionnel et pratique.

Project Summary

We are a group of four students and for the final year course project of our second year of BSc in Informatics, we developed a weather station.

This weather station uses a RENESAS microcontroller and six sensors to transmit to a PC the temperature, the humidity, the atmospheric pressure, the speed and the direction of the wind and to finish with, the quantity of rain fallen during an interval of time.

The transmission to the PC from the microcontroller uses a wireless transmission with the ZIGBEE protocol.

To recover all the data concerning the weather and to post them, we developed a graphical user interface (GUI) too.

Moreover, to program the RENESAS microcontroller we used the C programming language and to write the GUI, we used the TCL/TK programming language.
Our weather station works but it is not completely finished. In fact, we did not have enough time to design the weather forecast part.

We divided all the work into several tasks to have at least one person per sensor. Then, to find and to solve all the bugs, we worked together on all the sensors, the GUI and the wireless transmission.
Several unexpected difficulties and bugs appeared such as bad electronic components or several crashes of the main program. Fortunately, we always found a solution for each problem by searching on the net or more simply, with more perseverance.

With this project, we improved our skills in the embedded industrial systems and we applied our management and programming skills on a real hardware system: the weather station.

To conclude, this project of fourteen weeks brought us a very good experience in microcontroller programming. We are proud to have developed this weather station even if we haven't completely achieved our goal.