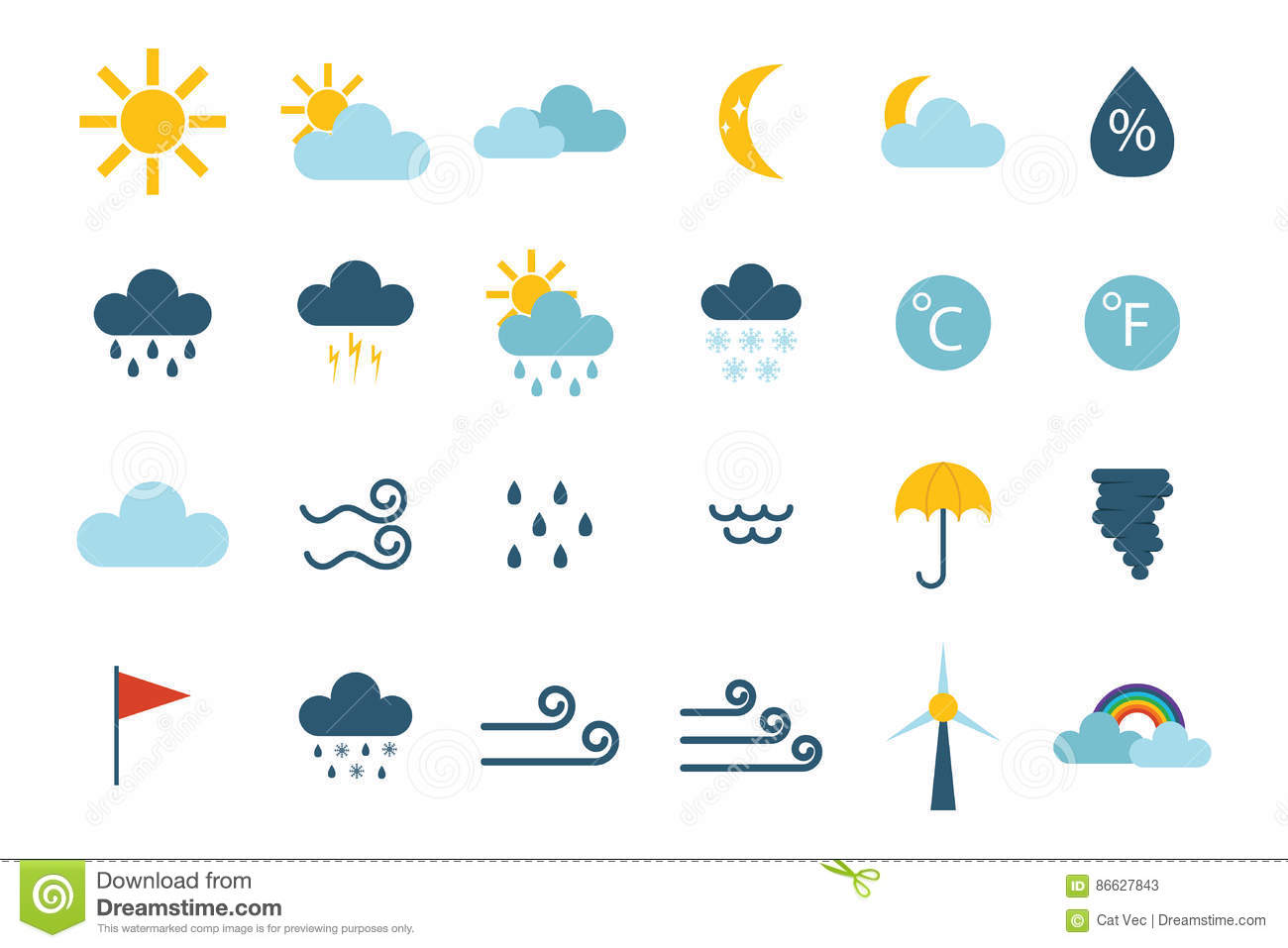
Soliveri Cedric

Faure Nicolas

Fahran mohand

Lemare romaric

A.W.T.I.S



**A**utomatic-**W**eather-**T**hermical-**I**ntelligent-**S**ysteme

# Introduction

Les cartes Arduino sont des cartes qui sont programmable à l’infini et leur microprocesseur permettent de réaliser des acquisitions de données issus de capteurs. Elles sont assez abordable compte tenu de leur prix et de leur facilité de programmation et peuvent gérer une communication sans fil type WiFi , Bluetooth ou LoraWan ce qui est très intéressant dans le cas d'une station météo.

# Sommaire

Introduction 1

Présentation de la station 3

La carte Arduino mini-pro 3

Les contraintes 4

Le module de communication sans fil 4

Alimentation exterieur de la carte Arduino et des capteurs 4

Les capteurs 7

Montage de l’anenometre 8

L'abri (exterieur) 8

Le sketch à téléverser dans la carte Arduino 9

Structure générale d'un programme Arduino 9

Le sketch Arduino 10

Le mode sommeil 10 Code arduino protocole de communication ABP 10

Code des capteurs 10

Initialisation de la station 11

Relevé des données enregistrées 11

Annexes 12

Le sketch Arduino 12

Remerciement 20

# Presentation de la station

La station météo se décline en deux version. (module interieur/exterieur)

## Station meteo interieur :

Dissimulé au sein d’un objet decoratif , la station météo envoie periodiquement les donnés recolté par les capteurs de la carte Arduino à un ordinateur en utilisant un protocole de communication de type LoraWan tres peu couteux en energie.

## Station meteo exterieur :

La station meteo exterieur repose sur le meme principe sauf que cette fois ci le dispositif est adaptéé à son environnement. L’objet decoratif devient un abri pour proteger les differents elements de la carte electronique. L’objectif etant de le rendre etanche à la pluie et d’augmenter la duree de vie de la batterie, ainsi un panneau solaire à été ajouter sur le dessus de l’abri. L’abri qui lui à été fabriqué par une imprimante 3d contient la carte arduino avec ses 3 capteur. Il est ventilé afin d’assurer des données fiable et contient donc :

-un capteur de température

- un capteur d'humidité

- Un anémomètre à ailettes qui surmonte l'abri

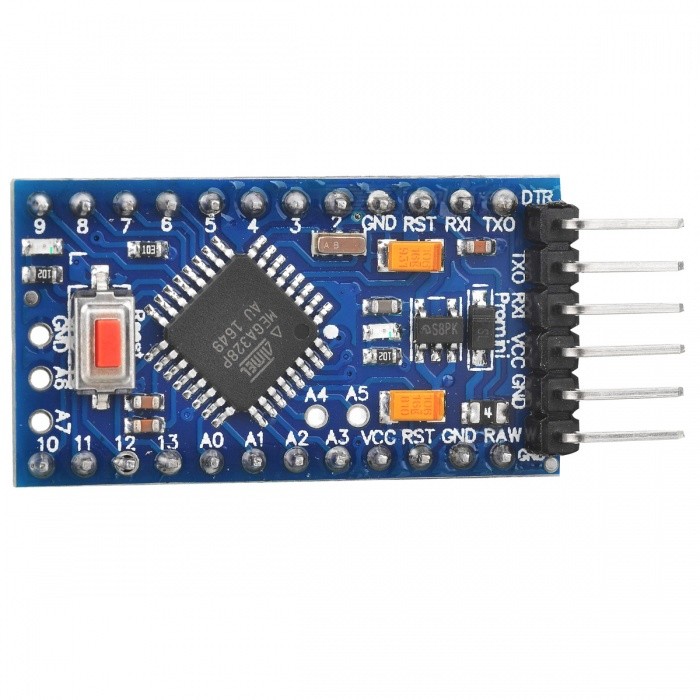
Le module exterieur est énergétiquement autonome grace à son panneau solaire qui recharge la batterie de 3,7 Volts et 400 mA. Cette batterie alimente la carte arduino et les capteurs, et un programme de communication OTAA permet d’envoyer périodiquement les données recoltées par les capteurs.

# La cartes Arduino Mini pro

Les cartes Arduino sont fabriquées autour d'un microcontrôleur ATméga . Il existe plusieurs modèle ayant chacun leurs spécificités. La carte **Mini Arduino Pro** est une carte microcontrôleur basé sur la  [ATmega328P](http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf) .

Il comporte 14 broches d'entrée / sortie numériques (dont 6 peuvent être utilisées en tant que sorties PWM), 6 entrées analogiques, un résonateur intégré, un bouton de réinitialisation et des trous pour les en-têtes de broche de montage

L'Arduino Pro Mini est destiné à une installation semi-permanente dans des objets. (Source :Arduino.cc)



Carte Arduino mini-pro

## Les contraintes

* L'autonomie en énergie impose une consommation de courant dans le circuit réduite au maximum
* Le panneau solaire photovoltaïque doit pouvoir recharger la batterie le jour pour qu’elle puisse fonctionner la nuit
* Les données captées doivent etre envoyées de manière periodique dans un laps de temps assez court pour un suivi précis de l'évolution de la météo
* L'Arduino doit comporter toutes les bibliothèques necessaires au bon fonctionnement des capteurs, du mode sommeil (LowPower.h) et afin de supporter la communication avec le site The Things Network et l’application Cayenne LPP

## Le module de communication sans fil

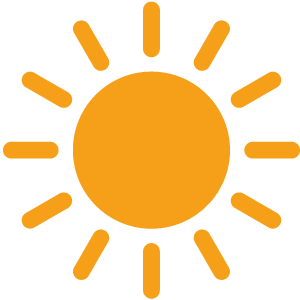
Le module est deja fixé sur notre carte arduino , c’est la partie UCA. Il contient une antenne permettant de faire communiquer la carte avec un PC à distance grace a des ondes electromagnetique.

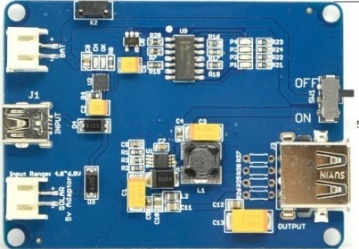
# Alimentation exterieur de la carte Arduino et des capteurs

## Matériel utilisé

* Lipo Rider Pro
* Panneau solaire
* Batterie lithium
* Carte Arduino mini

## Schéma l’alimentation extérieur



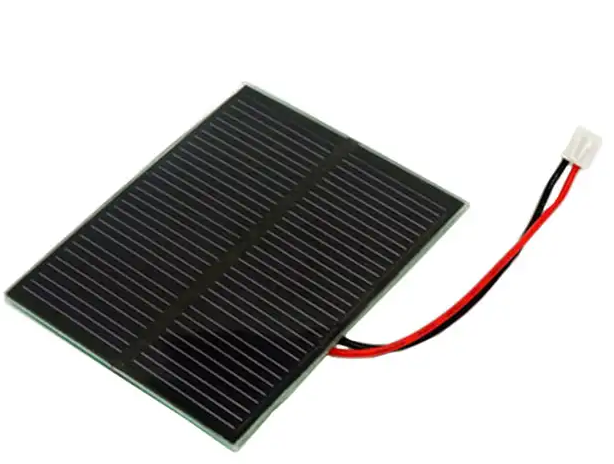




L’alimentation extérieur du projet, permettre au projet d’être autonome au niveau énergétique en utilisant un panneau solaire et une batterie lithium rechargeable

Pour l’alimentation extérieur nous avons utilisé la carte Lipo Rider Pro. Cette carte permet de charger la batterie et alimenté Arduino nano ainsi que les déférant capteurs pendent la journée puis au coucher de soleil la carte Lipo Rider Pro va utiliser la batterie comme source d’énergie.

Caractéristique de la batterie

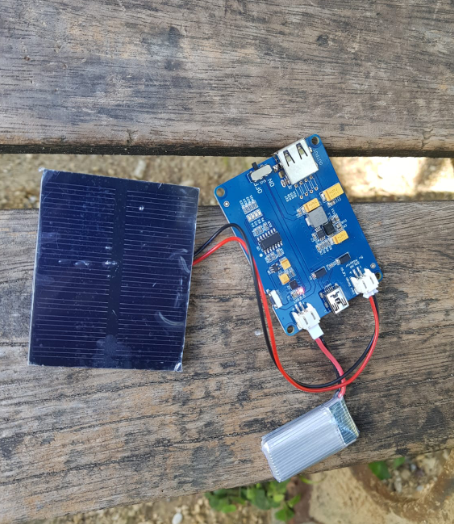
* Tension 3.7 Vcc
* Intensité 400 mAh
* Connecteur 2 broches
* Temps de charge : environ 45 mn
* Marque : JJRC

Caractéristique du panneau solaire

* Tension 5.5 Vcc
* Puissance Max 500mW
* Courant 100mA
* Description 55X70 cm



Mesure prélever en plein soleil en utilisant un multimètre.

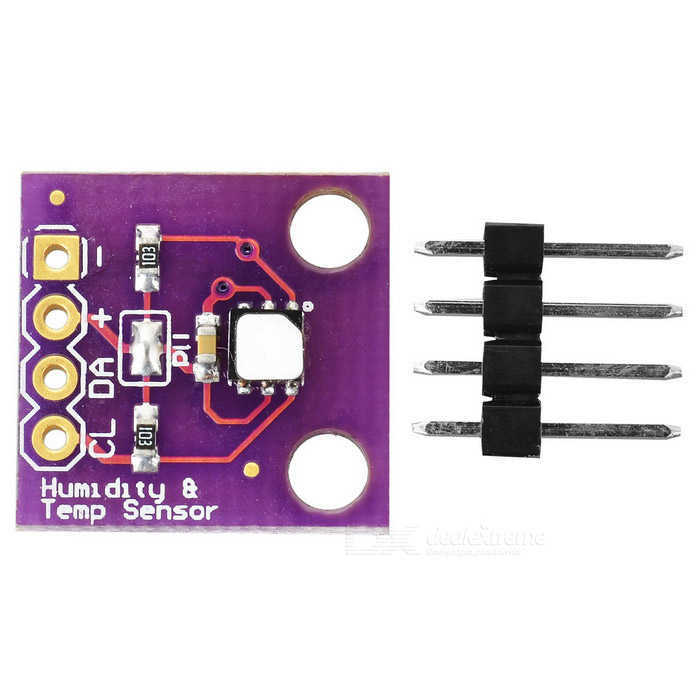


Cette photo nous montre que le panneau solaire est capable de recharger la batterie même si on est positionné à l’ombre



Pour l’alimentions intérieur nous allons utiliser une batterie lithium 1200m

## Les capteurs



Le capteur humiditer /temperature (SI7021)

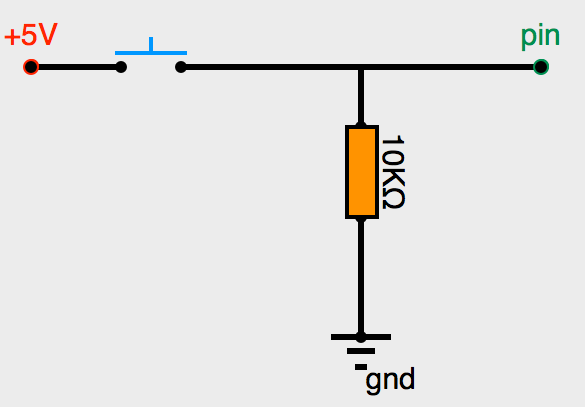
anémomètre

L’ anémomètre est un interrupteur de type ILS commandé avec aimant solidaire du moulinet, se ferme à chaque tour, la carte Arduino compte le nombre de tours par seconde (il faut utiliser un montage pull-down)

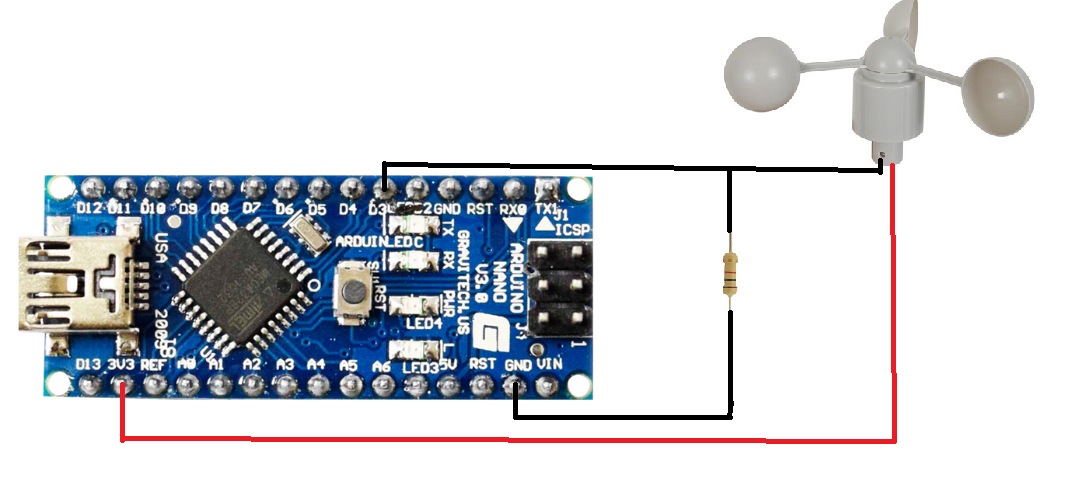
Lorsqu'un simple interrupteur mécanique est utilisé comme entrée à un circuit logique,

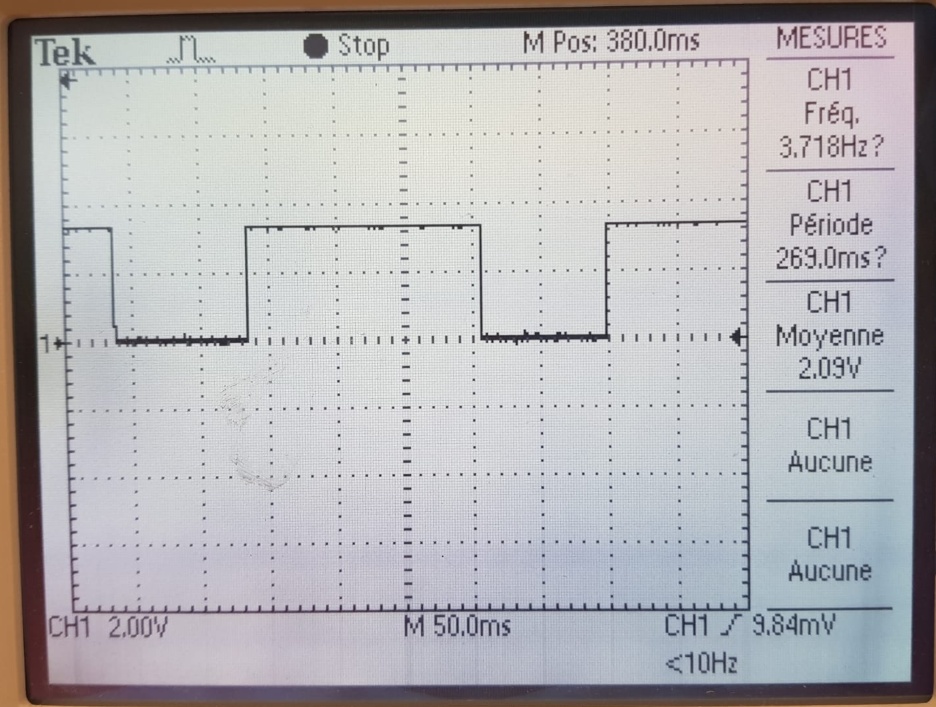
Les résistances pull-down permettent de fixer le pin (lorsque l’interrupteur est ouvert) à 0. Lorsque l’interrupteur est abaissé le courant va du +5V au pin car cela lui demanderait trop "d'efforts" de passer par la résistance de 10kΩ. Lorsque le bouton poussoir est ouvert, le courant résiduel ou le courant éventuel induit par d'éventuels perturbations environnantes est "aspiré" par le gnd.

Sans pull down, l'entrée serait flottante et ainsi donc à un niveau logique indéfini ce qui pourrait causer un fonctionnement erratique du circuit.



# Montage de L'anémomètre





En utilisant oscilloscope on peut voir la forme du signal, on bien un signal carré ou la période varie en fonction la vitesse de l'anémomètre.

# L'abri (exterieur)

Pour que les mesures soient valables, il faut que les capteurs soient en contact direct avec l'air, mais aussi protégés du soleil et de la pluie. L'abri doit donc être bien ventilé, mais aussi suffisament protecteur pour que les capteurs ne soient pas mouillés par le pluie. On a donc décider de créer cette abri avec une imprimante 3d pour qu’il puisse repondre parfaitement aux criteres recherché.

Il est possible de l’ouvrir via un des coter de l’abri afin d’avoir acces a la carte arduino pour une maintenance eventuelle.

L’abri qui contient les émetteur et la batterie doit être étanche, mais doit cependant être ventilé pour éviter une surchauffe de circuits électroniques, ainsi il y a un espace entre le toit de l’abri et sa strucutre permettant a l’air de passer mais pas la pluie.

# Le sketch à téléverser dans la carte Arduino

# Structure générale d'un programme Arduino

Un programme Arduino (sketch) comporte toujours au moins deux procédures :

1. La procédure "setup" dans laquelle on fixe les valeurs des variables globales, on intialise le port série utilisé et ses caractéristiques, etc ... Le setup est exécuté à chaque mise sous tension ou chaque reset de la carte Arduino et uniquement.

2. La procédure "loop" qui, comme son nom l'indique, tourne en permanence, sur elle-même tant que l'Arduino est alimenté.

3. Des fonctions peuvent être définies également, il faut les implémenter avec la procedure "loop".

# Le sketch arduino

Il est proposé dans les annexes et comporte essentiellement :

## Bibliothèques et variables

• L'importation des bibliothèques pour la gestion des capteurs, du mode sommeil et pour gerer la communication TTN via ABP.

## Le setup

Le setup contient l'ouverture du port série, et il fixe les paramètres des entrées et sorties..

## La boucle

On trouve dans la boucle

•Des instructions effectuées lors de chaque boucle :

La lecture du port série et l'exécution des instructions imposées par l'ordinateur

## Des fonctions

• lecture de la tempéraure, de l'hygrométrie, et de la vitesse du vent

# Le mode sommeil

Afin d’augmenter la duree de vie de notre batterie nous avons decidé d’inclure au sein des deux versions de notre station meteo, un code permettant d’eteindre certaine fonctionnaliter de la carte quand elle n’emet pas, ceci afin d’economiser de la batterie et ainsi de fonctionner plus longtemps.

## Code arduino basique du mode sommeil (code interieur/exterieur)

Ceci est le minimum de ligne de code necessaire à la mise en sommeil d’un module electronique. La bibiliotheque Low power Master disponible en telechargement sur <https://github.com/rocketscream/Low-Power> permet de n’avoir que tres peu de ligne de code a rentrer pour activer le mode sommeil .

// ajout de la bibliotheque low power.h de rocketstream

#include <LowPower.h>

Void setup

// Aucune action requise dans le setup

void loop() {

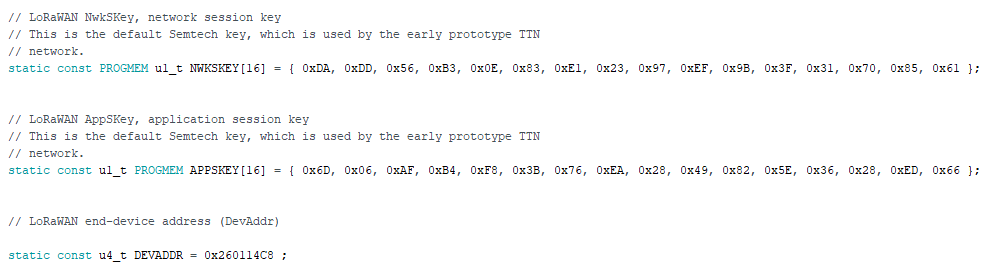
//Mode sommeil de 8seconde (max sans faire de boucle d’iteration)

LowPower.idle(SLEEP\_8S, ADC\_OFF, TIMER2\_OFF, TIMER1\_OFF, TIMER0\_OFF,

SPI\_OFF, USART0\_OFF, TWI\_OFF);

## Protocole de communication ABP (extrait du code interieur)

Afin que le site the things network puisse recevoir les donner des capteur il faut enregistrer les adresse des clef de session de l’application (NWSKEY , APPSKEY , DEVADDR) au format msb .



## Code pour les capteur (interieur/exterieur)

Le code permettant de faire fonctionner les capteur est le Code readSI7021 auquel on à ajouter le code de l’anenometre et le mode sommeil (voir annexe : code exterieur/interieur)

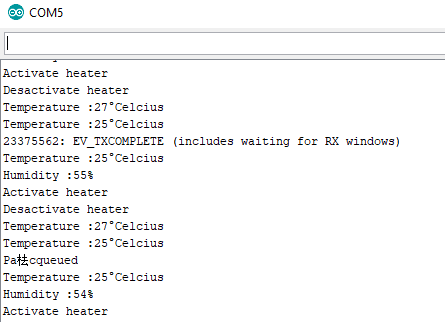
## Code anenometre (exterieur)

Voir annexe (code exterieur)

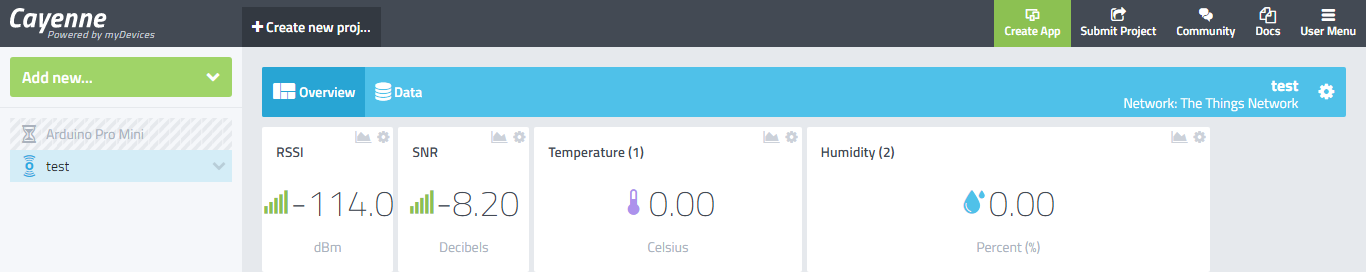
# Initialisation de la station

Il faut d'abord lancer le logiciel Arduino situé sur l'ordinateur qui permettera de recevoir les données.. Après la mise sous tension de la carte Arduino, appuyer sur le bouton "reset". Le sketch efffectue d'abord le "setup" puis la "loop". Le programme communique les informations recolté par les capteur à l’application cayenne qui reconnait notre module grace à TTN.

# Relevé des données enregistrées sur le moniteur serie Arduino ( code interieur)



## Resultats sur cayenne LPP



## Remarque :

Les données ne parviennent pas jusqu’à cayenne LPP car le code interieur n’est pas adapté au format cayenne LPP.

# Annexes

## Le sketch Arduino du code interieur

#include <lmic.h>

#include <hal/hal.h>

#include <SPI.h>

#include <Wire.h>

#include <SI7021.h>

#include <LowPower.h>

SI7021 sensor;

// LoRaWAN NwkSKey, network session key

// This is the default Semtech key, which is used by the early prototype TTN

// network.

static const PROGMEM u1\_t NWKSKEY[16] = { 0xDA, 0xDD, 0x56, 0xB3, 0x0E, 0x83, 0xE1, 0x23, 0x97, 0xEF, 0x9B, 0x3F, 0x31, 0x70, 0x85, 0x61 };

// LoRaWAN AppSKey, application session key

// This is the default Semtech key, which is used by the early prototype TTN

// network.

static const u1\_t PROGMEM APPSKEY[16] = { 0x6D, 0x06, 0xAF, 0xB4, 0xF8, 0x3B, 0x76, 0xEA, 0x28, 0x49, 0x82, 0x5E, 0x36, 0x28, 0xED, 0x66 };

// LoRaWAN end-device address (DevAddr)

static const u4\_t DEVADDR = 0x260114C8 ;

// These callbacks are only used in over-the-air activation, so they are

// left empty here (we cannot leave them out completely unless

// DISABLE\_JOIN is set in config.h, otherwise the linker will complain).

void os\_getArtEui (u1\_t\* buf) { }

void os\_getDevEui (u1\_t\* buf) { }

void os\_getDevKey (u1\_t\* buf) { }

static uint8\_t mydata[] = "Hello, world!";

static osjob\_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty

// cycle limitations).

const unsigned TX\_INTERVAL = 30;

// Pin mapping

const lmic\_pinmap lmic\_pins = {

.nss = 10,

.rxtx = LMIC\_UNUSED\_PIN,

.rst = 8,

.dio = {2, 7, 9},

};

void onEvent (ev\_t ev) {

Serial.print(os\_getTime());

Serial.print(": ");

switch(ev) {

case EV\_SCAN\_TIMEOUT:

Serial.println(F("EV\_SCAN\_TIMEOUT"));

break;

case EV\_BEACON\_FOUND:

Serial.println(F("EV\_BEACON\_FOUND"));

break;

case EV\_BEACON\_MISSED:

Serial.println(F("EV\_BEACON\_MISSED"));

break;

case EV\_BEACON\_TRACKED:

Serial.println(F("EV\_BEACON\_TRACKED"));

break;

case EV\_JOINING:

Serial.println(F("EV\_JOINING"));

break;

case EV\_JOINED:

Serial.println(F("EV\_JOINED"));

break;

case EV\_RFU1:

Serial.println(F("EV\_RFU1"));

break;

case EV\_JOIN\_FAILED:

Serial.println(F("EV\_JOIN\_FAILED"));

break;

case EV\_REJOIN\_FAILED:

Serial.println(F("EV\_REJOIN\_FAILED"));

break;

case EV\_TXCOMPLETE:

Serial.println(F("EV\_TXCOMPLETE (includes waiting for RX windows)"));

if (LMIC.txrxFlags & TXRX\_ACK)

Serial.println(F("Received ack"));

if (LMIC.dataLen) {

Serial.println(F("Received "));

Serial.println(LMIC.dataLen);

Serial.println(F(" bytes of payload"));

for (int i = 0; i < LMIC.dataLen; i++) {

if (LMIC.frame[LMIC.dataBeg + i] < 0x10) {

Serial.print(F("0"));

}

Serial.print(LMIC.frame[LMIC.dataBeg + i], HEX);

}

}

// Schedule next transmission

os\_setTimedCallback(&sendjob, os\_getTime()+sec2osticks(TX\_INTERVAL), do\_send);

break;

case EV\_LOST\_TSYNC:

Serial.println(F("EV\_LOST\_TSYNC"));

break;

case EV\_RESET:

Serial.println(F("EV\_RESET"));

break;

case EV\_RXCOMPLETE:

// data received in ping slot

Serial.println(F("EV\_RXCOMPLETE"));

break;

case EV\_LINK\_DEAD:

Serial.println(F("EV\_LINK\_DEAD"));

break;

case EV\_LINK\_ALIVE:

Serial.println(F("EV\_LINK\_ALIVE"));

break;

default:

Serial.println(F("Unknown event"));

break;

}

}

void do\_send(osjob\_t\* j){

// Check if there is not a current TX/RX job running

if (LMIC.opmode & OP\_TXRXPEND) {

Serial.println(F("OP\_TXRXPEND, not sending"));

} else {

// Prepare upstream data transmission at the next possible time.

LMIC\_setTxData2(1, mydata, sizeof(mydata)-1, 0);

Serial.println(F("Packet queued"));

}

// Next TX is scheduled after TX\_COMPLETE event.

}

void setup() {

Serial.begin(115200);

sensor.begin();

Serial.println(F("Starting"));

#ifdef VCC\_ENABLE

// For Pinoccio Scout boards

pinMode(VCC\_ENABLE, OUTPUT);

digitalWrite(VCC\_ENABLE, HIGH);

delay(1000);

#endif

// LMIC init

os\_init();

// Reset the MAC state. Session and pending data transfers will be discarded.

LMIC\_reset();

// Set static session parameters. Instead of dynamically establishing a session

// by joining the network, precomputed session parameters are be provided.

#ifdef PROGMEM

// On AVR, these values are stored in flash and only copied to RAM

// once. Copy them to a temporary buffer here, LMIC\_setSession will

// copy them into a buffer of its own again.

uint8\_t appskey[sizeof(APPSKEY)];

uint8\_t nwkskey[sizeof(NWKSKEY)];

memcpy\_P(appskey, APPSKEY, sizeof(APPSKEY));

memcpy\_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));

LMIC\_setSession (0x1, DEVADDR, nwkskey, appskey);

#else

// If not running an AVR with PROGMEM, just use the arrays directly

LMIC\_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);

#endif

// Disable link check validation

LMIC\_setLinkCheckMode(0);

// TTN uses SF9 for its RX2 window.

LMIC.dn2Dr = DR\_SF9;

// Set data rate and transmit power for uplink (note: txpow seems to be ignored by the library)

LMIC\_setDrTxpow(DR\_SF9,20);

// Start job

do\_send(&sendjob);

}

void loop() {

//Mode sommeil

LowPower.idle(SLEEP\_8S, ADC\_OFF, TIMER2\_OFF, TIMER1\_OFF, TIMER0\_OFF,

SPI\_OFF, USART0\_OFF, TWI\_OFF);

// temperature is an integer in hundredths

int temperature = sensor.getCelsiusHundredths();

temperature = temperature / 100;

Serial.print("Temperature :");

Serial.print(temperature);

Serial.println("°Celcius");

delay(500);

// humidity is an integer representing percent

int humidity = sensor.getHumidityPercent();

Serial.print("Humidity :");

Serial.print(humidity);

Serial.println("%");

delay(500);

// enable internal heater for testing

sensor.setHeater(true);

Serial.println("Activate heater");

delay(20000);

sensor.setHeater(false);

Serial.println("Desactivate heater");

// see if heater changed temperature

temperature = sensor.getCelsiusHundredths();

temperature = temperature / 100;

Serial.print("Temperature :");

Serial.print(temperature);

Serial.println("°Celcius");

//cool down

delay(20000);

// get humidity and temperature in one shot, saves power because sensor takes temperature when doing humidity anyway

si7021\_env data = sensor.getHumidityAndTemperature();

Serial.print("Temperature :");

Serial.print(data.celsiusHundredths/100);

Serial.println("°Celcius")

delay(500);

os\_runloop\_once();

}

## Code exterieur

#include <LowPower.h>

#include <Wire.h>

#include <SI7021.h>

SI7021 sensor;

const int m\_time = 5; //Meassuretime in Seconds

int wind\_ct = 0;

float wind = 0.0;

unsigned long time = 0;

void setup() {

Serial.begin(9600);

sensor.begin();

time = millis();

}

void countWind() {

wind\_ct ++;

}

void meassure() {

wind\_ct = 0;

time = millis();

attachInterrupt(1, countWind, RISING);

delay(1000 \* m\_time);

detachInterrupt(1);

wind = (float)wind\_ct / (float)m\_time \* 2.4;

}

void loop() {

//Mode sommeil

LowPower.idle(SLEEP\_8S, ADC\_OFF, TIMER2\_OFF, TIMER1\_OFF, TIMER0\_OFF,

SPI\_OFF, USART0\_OFF, TWI\_OFF);

meassure();

Serial.print("vitesse du vent: ");

Serial.print(wind); //Speed in Km/h

Serial.print(" km/h - ");

Serial.print(wind / 3.6); //Speed in m/s

Serial.println(" m/s");

delay(500);

// temperature is an integer in hundredths

int temperature = sensor.getCelsiusHundredths();

temperature = temperature / 100;

Serial.print("Temperature :");

Serial.print(temperature);

Serial.println("Â°Celcius");

delay(500);

// humidity is an integer representing percent

int humidity = sensor.getHumidityPercent();

Serial.print("Humidity :");

Serial.print(humidity);

Serial.println("%");

delay(500);

}

# Remerciements :

Jerome Lanterri

Fabien Ferrero