MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation: Information and Communication Technologies (ICT)

Capteurs IoT pour la mesure des effets des couvertures végétales sur la vigne

Auteur
Raoul Dupuis

Sous la direction de
Prof. Nabil Abdennadher

Expert externe
Prof. Fabien Vannel
Resp. du groupe CoRes HEPIA (Informatique matérielle)
Prof. Nicolas Delabays
Institut inTNE HEPIA (Agronomie)

Lausanne, HES-SO//Master, 4 Juin, 2021

Table des matières

Liste des figures

Abréviation et Traduction

HEPIA	Haute École du Paysage, d'Ingénierie et d'Architecture
GE	Genève
LoRa	Long range
LoRaWAN	Long Range Wide-Area Network
Wi-Fi	Wireless Fidelity
4G	quatrième génération (des standards pour la téléphonie mobile)
Hz	Hertz
USB	Universal Serial Bus
Soil	Sol
Moisture	Humidité
Sensor	Capteur
IDE	Integrated Development Environment
Setup	Mettre en place
Loop	Boucle
EUI	Extended Unique Identifier
Gateway	Passerelle
AWS	Amazon Web Services
GCP	Google Cloud Platform
CSV	Comma-Separated Values
SQL	Structured Query Language
IoT	Internet of Things
Trust Model	modèle de confiance
Trust Factor	facteur de confiance
MSE	Mean Squared Error
LLSE	Linear Least Squares Estimation
Availability	disponibilité
Regularity	régularité
dBm	rapport de puissance en décibels (dB) entre la puissance mesurée et un milliwatt (mW)

Introduction

Ce projet, en collaboration avec la filière agronomie de HEPIA à pour but d'analyser l'impact de différentes couvertures végétales sur la vigne. Ce dernier est évalué à partir des valeurs de température et d'humidité, dans l'air et dans le sol. Chaque type de couverture possède des avantages et des inconvénients connus de manière empirique, mais ceux-ci n'ont jamais été corroborés avec une expérience scientifique. Des photos de la parcelle de vigne sont disponibles en annexe.

La consommation de l'eau est l'un des points étudiés pour justifier l'utilisation d'un type donné de couverture végétale. En effet, une couverture rigoureuse consomme beaucoup d'eau et entre en concurrence avec la vigne qui n'aura pas assez d'eau pour se développer. Des capteurs d'humidité du sol sont donc installés à dix centimètres et à cinquante centimètres de profondeur, afin de pouvoir quantifier la différence entre chaque couverture.

Selon les agronomes, la couverture utilisée à une incidence sur le risque de gel. Certains vignerons préfèrent utiliser un sol nu afin que la chaleur emmagasinée la journée soit rayonnée pendant la nuit. Un capteur de température est donc installé au niveau de la vigne à environ cinquante centimètres au-dessus du sol, afin de pouvoir quantifier l'effet du rayonnement du sol.
D'autres points sont également étudiés par les agronomes, tels que la biodiversité, la richesse du sol et sa protection (limitation de l'érosion) ou encore la quantité d'herbicide utilisée. Mais ceux-ci ne sont pas mesurés par des capteurs dans le cadre de ce projet.

L'expérimentation est réalisée sur une parcelle du domaine cantonal (GE), situé sur le coteau de Bernex. Pour ce faire, six couvertures végétales, semées en octobre 2020, sont comparées dans le cadre d'un essai en blocs randomisés à quatre répétitions.

Pour réaliser ce projet, des capteurs à faible consommation d'énergie utilisant le protocole LoRa ont été installés. Les données sont récupérées et stockées dans une base de données. Les données sont ensuite représentées sous forme de graphiques.

Les capteurs étant déployés en extérieur, ils sont soumis à différentes contraintes d'ordre climatique, animale ou humain. Nous ne pouvons donc pas faire totalement confiance aux valeurs reçues. Leur intégrité doit être quantifiée. Pour cela, un algorithme générique de trust, proposé par le travail de master de Nizar Bouchedakh [1], est appliqué aux données récoltées. Cet algorithme a été utilisé dans le projet MEDInA [2], pour des données de capteurs sonores déployés à l'échelle d'une ville.

L'objectif de ce projet est double. Le premier est de fournir des mesures régulières aux agronomes pour les besoins de leur expérimentation. Ces données leur permettront de confirmer ou corriger leurs hypothèses en ce qui concerne l'impact des couvertures végétales sur les vignes. Le second est d'adapter l'algorithme de trust utilisé dans MEDInA, afin de vérifier l'intégrité des données, et reconnaître les couvertures végétales à partir des mesures collectées.

Ce document est organisé comme suit:
Le premier chapitre détaille l'installation des capteurs LoRa dans les vignes.
Le second chapitre présente l'architecture du système mis en place permettant de collecter les données et les présenter sous forme graphique aux utilisateurs finaux.
Enfin, le dernier chapitre traite du modèle de confiance et de son utilisation pour vérifier l'intégrité des données.
Une conclusion présente les perspectives de ce travail.

Chapitre 1 : Déploiement sur le terrain

1.1 Description du matériel

Afin de satisfaire les besoins des agronomes, plusieurs capteurs ont été étudiés tel que LM393 Soil Moisture Hygrometer [3] mais il aurait été nécessaire de le rendre étanche [4]. D'autres capteurs sont présentés sur ce site [5]. Finalement, après conseil d'un professeur en informatique materiel, les produits de la marque Libelium ont été choisis pour ce projet, car il propose du matériel étanche, et prêt à être installé. Libelium propose des capteurs pour des usages spécifiques tels que l'agriculture, l'eau, l'environnement, ou les parkings. Les capteurs sont connectés sur des cartes électroniques supportant différents types de protocoles de communication tels que LoRaWAN, ZigBee, Wi-Fi, ou 4G. Pour plus de détails, les solutions proposées par Libelium sont accessibles sur leur site [6].
Nous avons choisi, dans le cadre de ce projet, le boîtier Plug & Sense! Smart Agriculture Pro (Figure 1) [7][8]. Celui-ci est compatible avec les types de capteurs que nous souhaitons utiliser. Ce boîtier contient une carte électronique de type waspmote [9][10], ainsi qu'une batterie. A l'extérieur du boîtier, se trouve six connecteurs pour brancher des capteurs, un connecteur pour le panneau solaire, un connecteur pour l'antenne LoRa fonctionnant à 868MHz, une prise USB pour la programmation et un bouton de mise en tension. Tous ces connecteurs sont étanches.

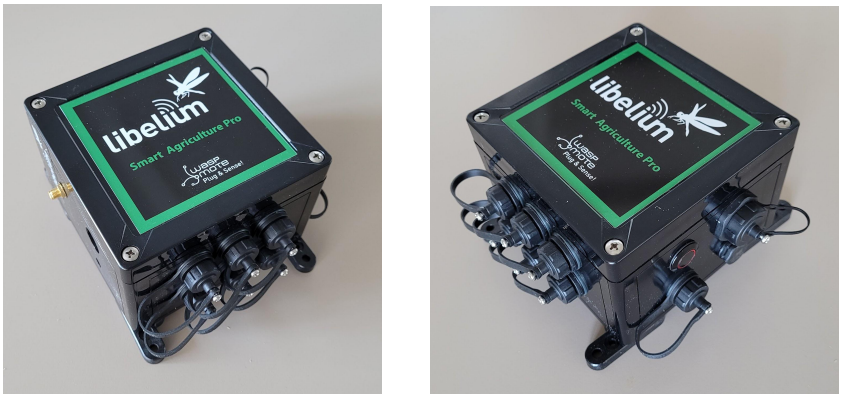


Figure 1. Smart Agriculture Pro Plug & Sense!

Afin de mesurer l'humidité du sol, deux capteurs "soil moisture sensor (watermark)" sont utilisés (Figure 2) [11].

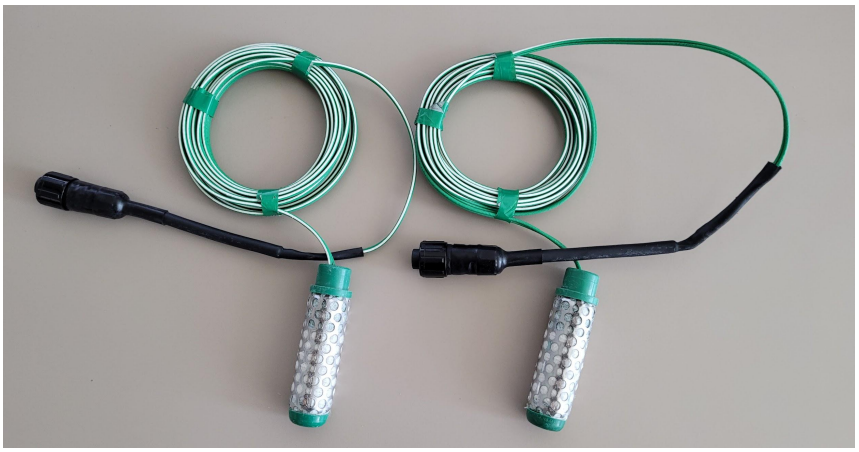


Figure 2. Soil moisture sensor

Contrairement aux capteurs d'humidité de l'air où l'unité est un pourcentage, les capteurs d'humidité du sol expriment leur valeur en Hertz.
Comme expliqué dans la documentation [12], une formule permet d'approximer le changement d'unité de mesures. La correspondance entre la valeur approximée et la valeur réelle n'est pas idéale, comme illustré dans la figure 3 ci-dessous. Les valeurs brut, en Hertz, sont enregistré dans la base de données

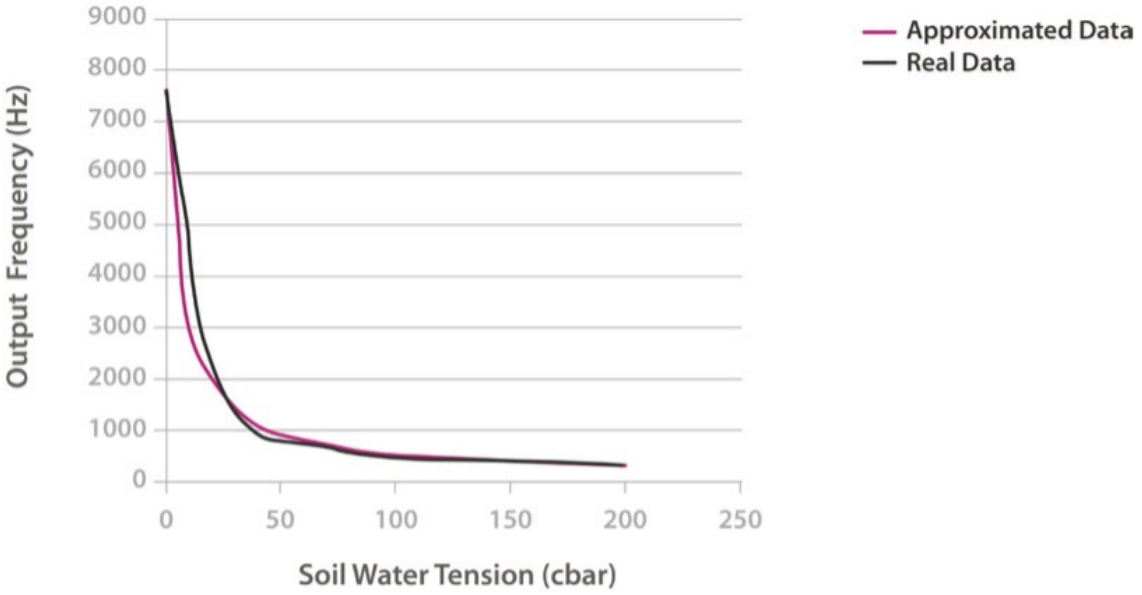


Figure 3. Correspondance fréquence / tension du capteur d'humidité du sol

Pour mesurer la température du sol, le capteur "soil temperature sensor (Pt-1000)" est utilisé (Figure 4) [13].

Figure 4. Soil temperature sensor

Enfin, le dernier type de capteur utilisé permet de mesurer la température de l'air (Figure 5). Ce capteur permet également de mesurer l'humidité et la pression de l'air. Il est de couleur blanche afin de limiter l'effet du soleil sur la température [14].

Figure 5. 9370P Temperature, Humidity and Pressure Probe

La batterie du boîtier est rechargée grâce à un panneau solaire de 3W (Figure 6). D'après les relevés réguliers du pourcentage de la batterie, ce panneau solaire est largement suffisant malgré une orientation sud-ouest et le risque d'une mauvaise météo sur plusieurs jours. En effet, pour les Plug & Sense! déployé, la tension ne descend jamais en dessous de 80%.

Figure 6. Panneau solaire

1.2 Création du support, et installation

La parcelle de vigne utilisée pour l'expérimentation est composée de six couvertures végétales différentes dupliquées quatre fois, ce qui donne vingt-quatre couvertures à mesurer. Pour le projet, est nommé “station” l’ensemble comprenant: le boîtier électronique avec batterie (Plug & Sense!), les capteurs de température et d’humidité, et le panneau solaire. Étant limité par le prix du matériel, une seule station est utilisée par couverture, il y a donc vingt-quatre stations sur la parcelle de vigne servant à l'expérimentation.
Pour chaque station, le panneau solaire et le boîtier Plug & Sense! ont été fixés sur un tube carré en aluminium grâce à un plat en aluminium. Ce tube carré est fixé sur les piquets servant de tuteur pour la vigne. Le tout est fixé à l’aide de rivets.
La liste qui suit détaille les étapes réalisées pour l’installation du matériel :

a. Installation des équerres sur les panneaux solaires a l’aide de boulons, fournis par libelium.
b. Découpe à moitié des tubes carrés pour passer de 2,5 mètres à 1,25 mètres
c. Découpe des plats en aluminium destinés à être placés entre le boîtier Plug & Sense! et le tube carré, et entre le panneau solaire et le tube carré. En tout, quatre morceaux de plats en aluminium sont nécessaires par stations.
d. Perçage des plats en aluminium permettant de placer les rivets.
e. Fixation des plats en aluminium sur les panneaux solaires à l’aide de rivets.
f. Perçage des tubes carrés permettant la fixation du panneau solaire et du boîtier Plug & Sense!
g. Fixation des panneaux solaires sur les tubes carré à l’aide de rivets
h. Sur la parcelle de vignes, fixation du tube carré sur le piquet de la vigne à l’aide de 3 rivets. Les piquets de la vigne sont déjà percés, mais les trous dans les tubes carrés ont été réalisés sur place.
i. Fixation du boitier Plug & Sense! sur le tube carré à l’aide de rivets
j. Branchement du panneau solaire, des quatre capteurs, et de l’antenne LoRa.
k. Enterrement des capteurs dans le sol à 50cm et 10cm à l’aide d’une barre à mine.
l. Programmation de chaque boîtier grâce à un câble USB, et changeant dans chaque programme la variable DeviceEUI, ainsi que le nom correspondant à l’emplacement, par exemple A1 pour la couverture A de la première rangée.

1.3 Programmation boitier sensor

Afin de programmer la carte se trouvant dans le boîtier Plug & Sense!, il faut télécharger l’IDE fourni par libelium. La version utilisée pour le projet est la suivante : “waspmote-pro-ide-v06.21”.
La structure du programme est la même que celle utilisée pour les cartes arduino. Le programme se compose de deux fonctions : setup() et loop(). Comme leur nom l’indique, la fonction setup est exécutée au démarrage et permet d’effectuer des initialisations. La fonction loop est exécutée dans une boucle infinie après la fin de la fonction setup.
Le programme exécuté par les boîtiers se trouvant dans les vignes est décrit ci-après.

La fonction setup permet de joindre le réseau LoRa par la méthode OTAA (Over The Air Activation) (Figure 8).

- Setup :

La fonction `LoRaWAN.factoryReset()` est exécutée au début afin de remettre à zéro les compteurs permettant d'éviter une incohérence avec la gateway lors d'un redémarrage.
Ensuite un data rate de 5 est appliqué, soit un débit physique de 5470 bit/s. Ce data rate permet d'avoir un débit relativement élevé afin de pouvoir émettre régulièrement sans dépasser la limite de 1% du temps d'émission [15]. Ce data rate est possible car le niveau de puissance du signal reçu par la gateway lors des tests avec le premier boîtier est bon (RSSI : environ -87dB).
Les trois variables DeviceEUI, AppEUI et AppKey sont ensuite appliquées. La variable DeviceEUI est unique à chaque périphérique (la valeur utilisée, choisie par le fabricant, apparaît sur une étiquette collée sur le boîtier). Cette variable permet d'identifier chaque périphérique lors de la configuration du réseau.
Les variables AppEUI et AppKey permettent de définir l'application et sa clé dans le cloud du réseau LoRa. Ces variables ont été générées aléatoirement et sont identiques pour les vingt-quatre périphériques installés dans les vignes.
Ensuite la fonction `LoRaWAN.joinOTAA()` permet d'échanger les clés de connexion avec la gateway.
Enfin, la configuration est sauvegardée grâce à la fonction `LoRaWAN.saveConfig()` et le réseau est coupé afin d'économiser de l'énergie.
Avant de sortir de la fonction setup, un nom familier est appliqué pour l'envoi du message, composé d'une lettre entre A et F, représentant le type de couverture, et d'un chiffre entre 1 et 4 représentant le rang de vigne, et la carte des capteurs est activée afin de permettre une lecture ultérieure.
La correspondance entre le nom de la parcelle élémentaire et sa position est représentée par la figure 7 ci-après.

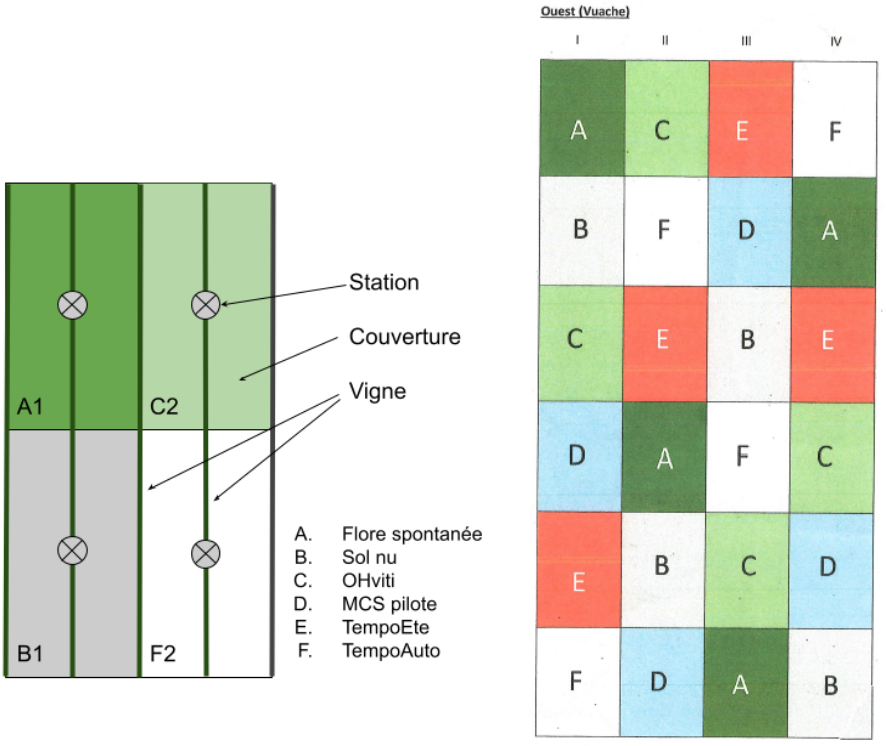


Figure 7. Schéma de la parcelle de vigne

```
// define the Waspmote ID
char moteID[] = "A2";

void setup() {
  USB.ON();

  // 1. Switch on
  error = LoRaWAN.ON(socket);

  // 2. Factory Reset
  error = LoRaWAN.factoryReset();

  // 3. Change data rate
  error = LoRaWAN.setDataRate(5);
  //error = LoRaWAN.setADR("on");

  // 4. Set Device EUI
  error = LoRaWAN.setDeviceEUI(DEVICE_EUI);

  // 5. Set Application EUI
  error = LoRaWAN.setAppEUI(APP_EUI);

  // 6. Set Application Session Key
  error = LoRaWAN.setAppKey(APP_KEY);

  // 7. Join OTAA to negotiate keys with the server
  error = LoRaWAN.joinOTAA();

  // 8. Save configuration
  error = LoRaWAN.saveConfig();

  // 9. Switch off
  error = LoRaWAN.OFF(socket);

  if (error_config == 0){
    USB.println(F("After joining through OTAA, the module and the network exchanged "));
    USB.println(F("the Network Session Key and the Application Session Key which "));
    USB.println(F("are needed to perform communications. After that, 'ABP mode' is used"));
    USB.println(F("to join the network and send messages after powering on the module"));
  }

  frame.setID(moteID);

  // Turn on the sensor board
  Agriculture.ON();
}
```

Figure 8. Plug & Sense! fonction setup

La fonction loop permet la lecture des capteurs et l'envoi d'une trame comprenant toutes les mesures (Figure 9).

- Loop :

Au début de la fonction, les valeurs des six capteurs (température, humidité et pression de l'air, température du sol et humidité du sol à 10cm et 50cm de profondeur) sont lues et affichées grâce au port série si un ordinateur est connecté.
Ensuite, la librairie fournit par libelium [16] permet de construire facilement le message qui sera envoyé. Pour chaque capteur, il suffit de renseigner le type de capteurs ainsi que sa valeur. Le message est alors construit selon un format prédéfini, séparant chaque valeur par un "#".
Afin de pouvoir envoyer le message, il faut activer le réseau avec cette fois ci un *join* ABP (Activation By Personalization) car les clés ont déjà été échangées au démarrage dans la fonction setup. Le message est ensuite envoyé sans confirmation avant de couper la connexion.

Le périphérique est ensuite en attente pendant six minutes avec la fonction delay. La fonction veille profonde, avec un timer pour le réveil, n'a pas été utilisée car le panneau solaire fournit assez d'énergie pour se passer de la veille profonde.

```
void loop()
{
  /////////////////////////////////////
  // 1. Read sensors value
  /////////////////////////////////////
  temp = Agriculture.getTemperature();
  humd  = Agriculture.getHumidity();
  pres = Agriculture.getPressure();
  soil_temp = pt1000Sensor.readPT1000();
  watermark1 = wmSensor1.readWatermark();
  watermark2 = wmSensor2.readWatermark();

  /////////////////////////////////////
  // 2. Print Sensors Values
  /////////////////////////////////////
  USB.print(F("Temperature:             "));
  USB.print(temp);
  USB.println(F(" Celsius"));
  USB.print(F("Humidity:                "));
  USB.print(humd);
  USB.println(F(" %"));
  USB.print(F("Pressure:                "));
  USB.print(pres);
  USB.println(F(" Pa"));
  USB.print(F("Soil temp:               "));
  USB.printFloat(soil_temp,3);
  USB.println(F(" Celsius"));
  USB.print(F("Watermark 1 - Frequency: "));
  USB.print(watermark1);
  USB.println(F(" Hz"));
  USB.print(F("Watermark 2 - Frequency: "));
  USB.print(watermark2);
  USB.println(F(" Hz"));

  ///////////////////////////////////////////
  // 1. Creating a new frame
  ///////////////////////////////////////////
  USB.println(F("Creating an ASCII frame"));

  // Create new frame (ASCII)
  frame.createFrame(ASCII);

  // set frame fields
  frame.addSensor(SENSOR_BAT, PWR.getBatteryLevel());
  frame.addSensor(SENSOR_AGR_SOIL1, watermark1);
  frame.addSensor(SENSOR_AGR_SOIL2, watermark2);
  frame.addSensor(SENSOR_AGR_SOILTC, soil_temp);
  frame.addSensor(SENSOR_AGR_TC, temp);
  frame.addSensor(SENSOR_AGR_HUM, humd);
  frame.addSensor(SENSOR_AGR_PRES, pres);

  // Prints frame
  frame.showFrame();
```

```
////////////////////////////////////////////
// 2. Switch on
////////////////////////////////////////////
error = LoRaWAN.ON(socket);

// Check status
if( error != 0 ) {
  USB.print(F("1. Switch ON error = "));
  USB.println(error, DEC);
  PWR.reboot();
}

////////////////////////////////////////////
// 3. Join network
////////////////////////////////////////////
error = LoRaWAN.joinABP();

// Check status
if( error == 0 )
{
  USB.println(F("2. Join network OK"));

  ////////////////////////////////////////////
  // 4. Send un-confirmed packet
  ////////////////////////////////////////////
  error = LoRaWAN.sendUnconfirmed( PORT, frame.buffer, frame.length);

  // Error messages:
  /*
   * '6' : Module hasn't joined a network
   * '5' : Sending error
   * '4' : Error with data length
   * '2' : Module didn't response
   * '1' : Module communication error
   */
  // Check status
  if( error == 0 )
  {
    USB.println(F("3. Send un-confirmed packet OK"));
    if (LoRaWAN._dataReceived == true)
    {
      USB.print(F("   There's data on port number "));
      USB.print(LoRaWAN._port,DEC);
      USB.print(F(".\r\n   Data: "));
      USB.println(LoRaWAN._data);
    }
  }
  else
  {
    USB.print(F("3. Send un-confirmed packet error = "));
    USB.println(error, DEC);
    PWR.reboot();
  }
}
else
{
  USB.print(F("2. Join network error = "));
  USB.println(error, DEC);
  PWR.reboot();
}
```

```
  ////////////////////////////////////////////
  // 5. Switch off
  ////////////////////////////////////////////
  error = LoRaWAN.OFF(socket);

  // Check status
  if( error != 0 ) {
    USB.print(F("4. Switch OFF error = "));
    USB.println(error, DEC);
    PWR.reboot();
  }

  ////////////////////////////////////////////
  // 6. wait for 6min
  ////////////////////////////////////////////
  delay(360000);
}
```

Figure 9. Plug & Sense! Fonction loop

Chapitre 2 : Déploiement cloud

La figure 10 présente l'architecture du système complet (hardware et software) mis en place.

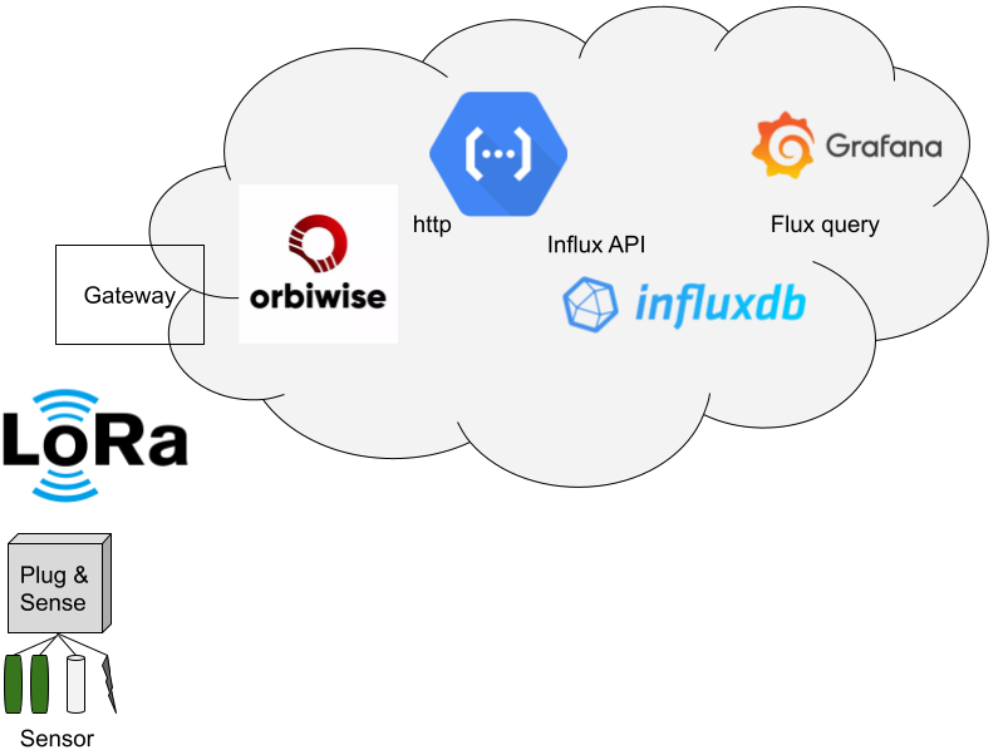


Figure 10. Architecture des données

La programmation du boitier Plug & Sense! Est décrit dans le chapitre précédent. Chaque élément se trouvant dans le cloud est détaillé dans ce chapitre.
La gateway permet de faire le lien entre le réseau LoRa sans fil, et internet (serveur orbiwise)
La plateforme orbiwise permet de configurer et recevoir les messages provenant du réseau LoRa. Ensuite, une fonction intermédiaire de type serverless permet d'enregistrer les nouveaux messages reçus dans la base de données InfluxDB. Enfin, Grafana permet d'afficher les données sous forme de graphiques en effectuant des requêtes à la base de données.

2.1 Configuration du réseau LoRa

L'infrastructure du réseau LoRa est gérée par les SIG (Services Industriels de Genève) via la plate-forme "orbiwise", qui est accessible par son interface Web. Le compte utilisé pour le projet ne permet pas la configuration de gateway; seule la gestion des périphériques (enregistrement, configuration, etc.) et la gestion des données sont possibles.
Lors de la connexion à l'adresse suivante : http://lora-ns.sig-ge.ch/, quatre menus sont disponibles.

L'onglet **Devices** permet d'afficher les devices enregistrés avec leur statut. Pour chaque device, il est possible de visualiser les messages reçus.
Pour enregistrer un nouveau device, il existe deux possibilités: la première est l'enregistrement manuel avec l'interface graphique, la deuxième est nommée "batch registration". Nous avons utilisé la seconde méthode qui permet d'enregistrer (et mettre à jour) les périphériques à l'aide d'un fichier *csv*.
L'onglet **Groups** permet de gérer les groupes de périphérique afin de leur envoyer des messages multicast.
L'onglet **Applications** permet la gestion des données reçues depuis les devices. L'application permet de configurer l'URL vers laquelle les données sont envoyées. Le bouton "start push" permet d'activer l'envoi des données.
Dans l'onglet **Profiles** se trouve la configuration de *device profiles*. Un device profile permet de configurer des paramètres du réseau LoRa tels que le type de connexion (OTAA ou ABP), ou la classe du périphérique (Supports class B et/ou supports class C). Les périphériques peuvent appartenir à l'une des trois classes permettant de définir le moment de réception des messages. La classe A permet de recevoir des messages uniquement après un envoi, pendant un délai défini. La classe B permet de recevoir des messages pendant une courte période à intervalle régulier, cette période est nommée *beacon*. La classe C permet de recevoir un message à tout moment. L'appartenance à une de ces trois classes à une incidence sur la consommation d'énergie. Le *device profile* permet également d'appliquer un *payload decoder* pour convertir le message brut reçu en message structuré.

2.2 Serverless function

Une fonction de type FaaS (Functions as a Service) hébergée par Google Cloud Platform (GCP) est utilisée pour interfacer les messages reçus par la plateforme orbiwise avec la base de données influxdb. Parmi les langages disponibles pour l'environnement d'exécution de fonction (Node.js, Python, Go, Java, .NET et Ruby), Python 3.8 est utilisé pour le projet. A la réception de chaque requête http, le code écrit par l'utilisateur est exécuté.

Dans le code écrit dans la fonction, les valeurs des différents capteurs sont récupérées depuis la requête envoyée par orbiwise, décodées, placées dans une structure de données, et enregistrée dans InfluxDB.

La fonction peut au choix être écrite sur l'interface web de GCP, ou sur l'ordinateur de développement en local, avant d'être déployée avec la CLI (Command Line Interface) de Google (gcloud). Lors du déploiement, le code source est compressé et importé, puis une compilation est envoyée à Cloud Build. La création du container et l'ajout du framework web (Flask pour le langage Python), qui gère la réception et la réponse des requêtes http, est totalement gérer par *Cloud Build*, elle est transparente à l'utilisateur.

2.3 InfluxDB cloud

Influxdb est une base de données optimisée pour des données de série chronologique. Elle peut être installée en local, ou de façon *serverless*, hébergée par AWS, Azure, ou Google Cloud Platform.

Pour le projet, GCP a été utilisé, mais, dans la pratique, il n'y a aucune différence entre les trois hébergeurs, hormis l'emplacement géographique du serveur, et le compte de facturation.
L'interface nous permet de charger des données et de les visualiser. Il est également possible de créer des tâches qui seront exécutées périodiquement ainsi que des alertes déclenchées lorsqu'une donnée dépasse un seuil, ou si un service arrête d'envoyer des données.
Trois méthodes sont possibles pour écrire des données. Des plugins nommés *telegraph* peuvent être utilisés pour charger les données automatiquement depuis des sources spécifiques. Des librairies client sont disponibles dans différents langages pour écrire des données depuis un programme. Un fichier au format *csv* ou *line protocole* peut également être directement uploader.
Afin de pouvoir enregistrer des données, il est important de comprendre comment celles-ci sont structurées à l'intérieur de influxdb.
Un *bucket* permet de regrouper plusieurs mesures différentes d'une même application. Pour chaque bucket, il est possible de définir la durée de rétention des données, entre 1h et 1 an, ou pour toujours.
Chaque mesure contient plusieurs champs prédéfinis décrit ci-après [17]:

- *_time*: permet de définir le timestamp de la mesure. Sur le disque, la valeur est enregistrée au format *epoch nanosecond*. Lorsque le timestamp est affiché, il est au format RFC3339. La précision du temps peut être spécifiée lors de l'écriture de la donnée.
- *_measurement*: cette valeur sous forme de chaîne de caractère permet de décrire le type de mesure. Pour le projet, le terme "temperature" a été utilisé pour les mesures de température dans le sol et dans l'air.
- *Field*:
    - *_field*: clé permettant de nommer la mesure. Par exemple "airtemp" pour la température de l'air.
    - *_value*: valeur effective de la mesure. Par exemple "15.2" pour une température de 15,2°
- *Tag*: les *tags* sont optionnels. Ils permettent une recherche de mesure plus facile car ils sont indexés. Par exemple, le tag *"exp"* = *"A"* est utilisé pour rechercher facilement les valeurs mesurées où la couverture vaut "A".

Dans le tableau de bord (*dashboard*), il est possible de composer des requêtes à la base de données avec la souris. Pour cela, il faut cliquer sur le bucket, puis le champ measurement choisi, puis field et enfin si nécessaire sur un tag. La fonctionnalité est pratique pour faire une recherche rapide sans connaître le langage ou les champs disponibles dans la base de données. Cette manipulation génère une requête au format *Flux* qui peut être ensuite visualisée et modifiée.

2.4 Grafana cloud

Grafana permet d'afficher des séries temporelles en provenance de différentes sources de données. Bien que InfluxDB permet d'afficher les données sous forme de graphique, Grafana y est souvent combiné car il permet un affichage étendu.

Après avoir connecté la base de données à grafana à l'aide d'un *token*, il est possible d'afficher les données dans un dashboard, en spécifiant une requête pour chaque graphique. Cette requête peut être effectuée avec le langage *Flux* ou *InfluxQL*.
Le langage *InfluxQL* ressemble à *SQL* et possède les mêmes instructions. En revanche, *Flux* est beaucoup plus complet: en plus du support de requêtes de base, il permet de manipuler les données avec un style fonctionnel [18].
Deux dashboard sont créés, et permettent respectivement d'afficher les données brutes des capteurs, et d'afficher les valeur des trust factors. Le dashboard consacré aux trust factors utilise le système de variable pour changer facilement le type de couverture et le type de capteur à observer. La variable permettant de définir la période d'observation est disponible par défaut.

Chapitre 3 : Trust Model

3.1 Trust Model Générique

L'un des obstacles les plus importants limitant l'adoption d'une infrastructure IoT est l'intégrité des données. En effet, chaque périphérique déployé sur le terrain est susceptible d'être altéré par un acte qui peut être intentionnel ou non. Pour l'IoT, les techniques d'intégrité de données conventionnelles sont difficiles à mettre en place car les périphériques utilisent peu d'énergie et ont des ressources de calcul limitées. Il n'est donc pas possible d'effectuer des calculs complexes de chiffrement. De plus, pour obtenir des prix de vente plus faibles, les fabricants utilisent parfois des composants de mauvaise qualité.

Étant donné que l'intégrité ne peut pas être appliquée aux périphériques, il est important de vérifier si on peut avoir confiance dans les données reçues. En revanche, ce mécanisme ne peut pas être appliqué à n'importe quel type de déploiement IoT, celui-ci ne doit pas contenir de données sensibles par exemple.
Pour cela, un algorithme de trust [19] se basant sur différents critères est appliqué aux données reçues.

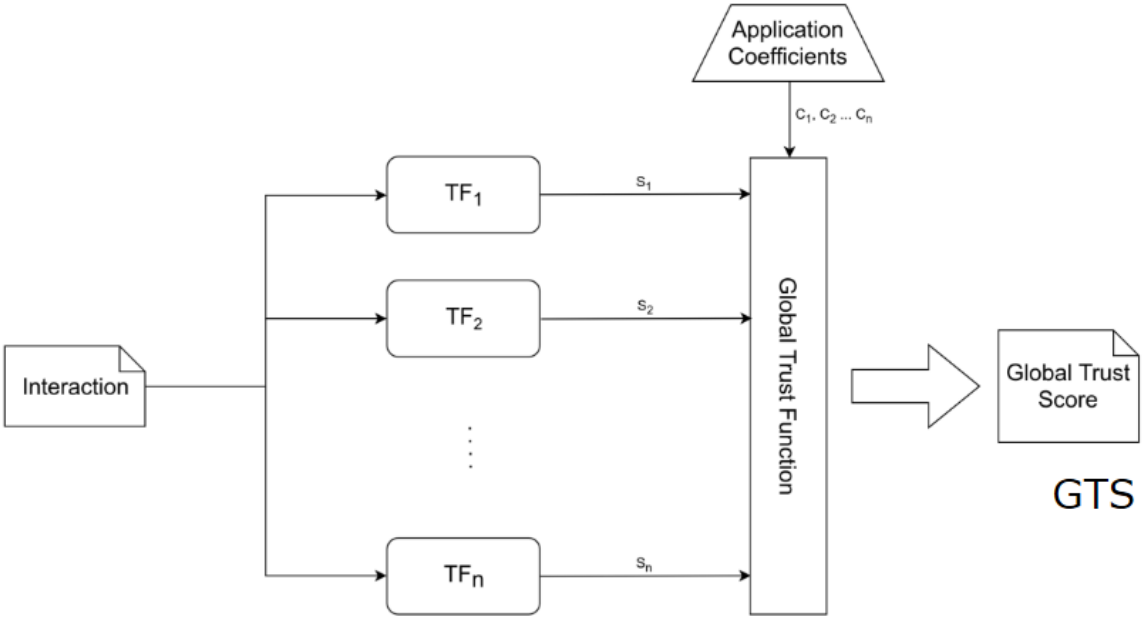


Figure 11. Générique Trust model

Comme le montre la figure 11, les données récoltées par les périphériques IoT passent par plusieurs algorithmes différents de *trust factor (TF)*. Chaque *trust factor* donne un score de confiance correspondant au type de confiance que l'on souhaite mesurer. Par exemple, le trust factor *availability* (disponibilité) donne une indication sur le nombre de messages reçu par rapport au nombre de messages attendus. Parmis les autres *trust factors*, on retrouve la *précision* ou la *signature*. Ces algorithmes produisent en sortie une valeur comprise entre 0 et 1. Ces valeurs sont ensuite agrégées au niveau d'une fonction globale dite *global trust function*. Cette fonction applique un coefficient sur chaque trust factor afin de lui donner plus ou moins d'importance, puis le résultat est sommé. La somme des coefficient vaut un, ce qui

permet d'avoir un *global trust score* compris entre 0 et 1. Ce *global trust score* permet de donner un indice de confiance sur la donnée reçue, et d'alerter l'utilisateur lorsque cette donnée n'est potentiellement pas fiable.

3.2 Trust Model pour les données récoltées dans les vignes

Le trust model utilisé pour les données des capteurs installés dans les vignes, se base sur un algorithme utilisé dans le cadre du projet MEDInA [2]. Étant donné que le type de capteur, et le contexte, est différent, le programme a été adapté afin de mieux correspondre au projet.
L'algorithme représenté sous forme de bloc dans la figure 12, utilise trois librairies spécifiques, quatre types de trust factor différents, et un script permettant d'exécuter les différents trust factors sur une période définie afin de remplir la base de données.

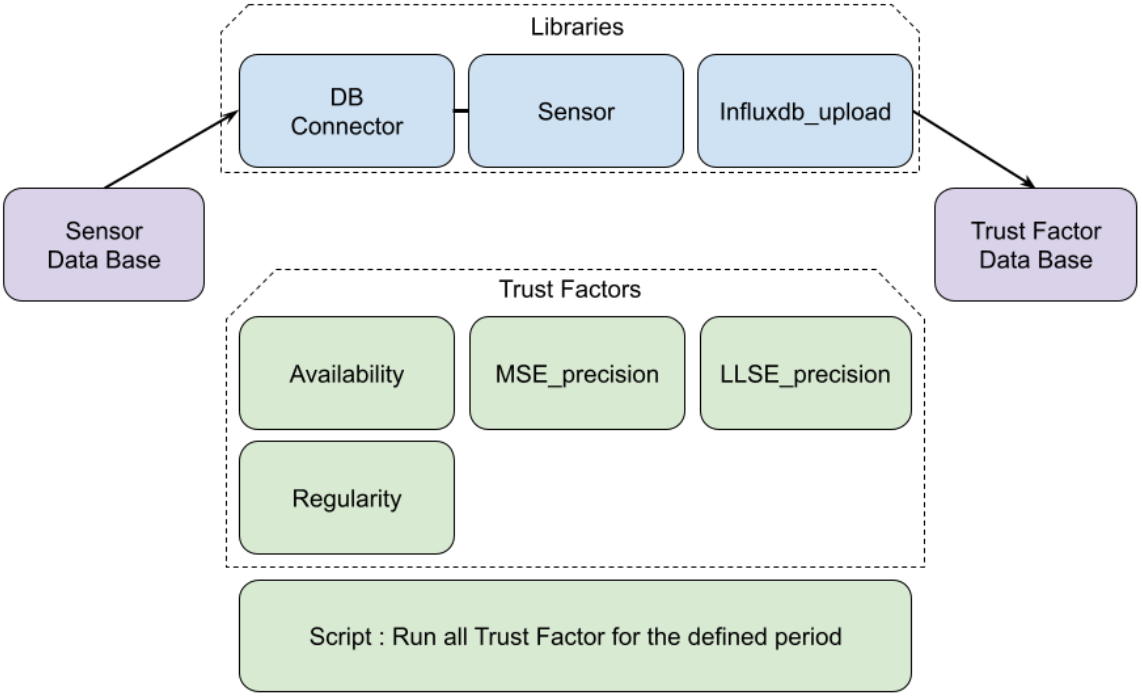


Figure 12. Trust Factor architecture

3.2.1 Librairie : DB_connector

La librairie nommée **DB_Connector** permet de faire une requête sur la base de données afin de récupérer les valeurs envoyées par les capteurs (Figure 13). Les données récupérées sont enregistrées en local dans un fichier csv afin de diminuer le nombre de requêtes, et accélérer l'acquisition.

```
def get_records(self, deveui, from_date=None, to_date=None, verbose=None):

    url = self.url + "/%s/payloads/ul?sort_by_timestamp=desc" % deveui
    file = path.join(self.dataFolder, "%s.csv" % deveui)

    # use csv file
    if path.exists(file):
        data = pd.read_csv(file, header=0, index_col=0, parse_dates=True)
        data['T'] = pd.to_datetime(data['timestamp'], utc=True)
        return data, None
    # send request to database
    else:
        r = requests.get(url, auth=(self.username, self.password), verify=False)
        if r.status_code == 200:
            result = pd.json_normalize(r.json())
            result['T'] = pd.to_datetime(result['timestamp'], utc=True)
            result.to_csv(file)
            return result, None
        else:
            return None, r.status_code
```

Figure 13. DB_connector

3.2.2 Librairie : Sensor

La librairie nommée **Sensor** permet de normaliser les valeurs reçues. La fonction `get_last_records()` (voir figure 14) permet de récupérer les données depuis la base de données (ou le fichier *csv*). Puis la fonction `normalize()` permet de définir un intervalle fixe entre chaque mesure. Le problème observé est que la fonction `round()` effectue un arrondi supérieur, et si deux mesures se trouvent dans le même intervalle de temps, une mesure est perdue, et comptabilisée comme manquante dans la mesure de l'*availability*. Pour le calcul de la précision, ce n'est pas un problème car la fonction `interpolate()` permet de combler les données manquantes par une interpolation. De plus, la normalisation permet d'avoir un nombre identique de mesures entre deux capteurs sur une période donnée. Ensuite, le *timestamp* est défini comme index.

```
SAMPLES_FREQ = 6.2

def normalize(dataframe):
    print(dataframe)
    dataframe['T'] = dataframe['T'].dt.round('%smin' % SAMPLES_FREQ)
    result = dataframe.set_index(['T'])
    result = result.sort_index()
    return result

def reindex(df, from_date=None, to_date=None):
    if from_date is None:
        from_date = df.index.min()
    if to_date is None:
        to_date = df.index.max()
    date_range = pd.date_range(start=from_date, end=to_date, freq='%smin' % SAMPLES_FREQ)
    df.loc[:,"missing"] = False
    df_with_missing = df.reindex(date_range)
    df_with_missing.loc[df_with_missing.isna().any(axis=1), "missing"] = True
    return df_with_missing

def interpolate(df):
    return df.interpolate()

 def get_last_records(self, nbRecords, from_date=None):
     """
     Returns the last known record from the DB.
     """
     df, _ = self.db.get_records(self.deveui)
     result = normalize(df)

     if from_date is not None:
         result = result.loc[:from_date]
     result = result.loc[~result.index.duplicated(keep='first')]\
         .tail(2*nbRecords).drop_duplicates().tail(nbRecords).copy()
     reindexed = reindex(result.tail(nbRecords).copy())

     interpolated = interpolate(reindexed).tail(nbRecords).copy()
     return interpolated
```

Figure 14. Librairie Sensor

3.2.3 Librairie : InfluxDB_upload

La librairie **InfluxDB_upload** permet d'enregistrer une mesure, nommée *Point* par InfluxDB, dans la base de données. Les valeurs nécessaires à la création du *point* sont données en paramètre au format json (Figure 15). Pour enregistrer plusieurs *Point* en même temps, il suffit de passer une liste en paramètre.

```
def write_point(self):
        data = json.loads(self.data)
        client = InfluxDBClient.from_config_file("config.ini")
        write_api = client.write_api(write_options=SYNCHRONOUS)

        if isinstance(data, list):
            mypoint = []
            for d in data:
                point_tmp = Point(d["measurement"])\
                    .field(d["field"], d["field_value"])\
                    .time(d['timestamp'])
                for t in d["tag"]:
                    point_tmp.tag(t, d["tag"][t])
                mypoint.append(point_tmp)
        else:
            point_tmp = Point(data["measurement"])\
                .field(data["field"], data["field_value"])\
                .time(data['timestamp'])
            for t in data["tag"]:
                point_tmp.tag(t, data["tag"][t])
            mypoint = point_tmp

        write_api.write(bucket=bucket, record=mypoint)
        client.close()
```

Figure 15. InfluxDB_upload

Pour rappel, chaque couverture correspond à une lettre entre “A” et “F”. Le numéro de la répétition est ajouté à la lettre pour avoir 24 stations entre “A1” et “F4” (voir figure 7). Un fichier *json* permet de faire la correspondance entre couverture végétale et devEUI. Dans le champ “deveui” se trouve un ensemble de clés-valeurs où la clé correspond au nom familier (exemple : `A1`) et la valeur le devEUI (exemple : `0004A30B00EFA335`). Dans le champ “cover” se trouve un ensemble de clé-valeur où la clé correspond au type de couverture (exemple : `A`) et la valeur la liste des stations correspondant à cette couverture (exemple : `["A1", "A2", "A3", "A4"]`).
Ce fichier permet de définir les voisins, mais également de remplacer le *devEUI,* en paramètre du programme, par le type de couverture.
Chaque *trust factors* utilisent les mêmes arguments, à savoir, la date de fin de l’analyse de confiance, le nombre de mesures avant cette date (exemple : 240 pour une journée), le type de couverture (exemple : A). Pour les deux *trust factors* “llse_precision” et “mse_precision”, le type de mesure analysée est également en paramètre (exemple : température de l’air, ou humidité du sol). Pour “llse_precision” le paramètre alpha est également ajouté, il s’agit de la sensibilité de l’algorithme.

3.2.4 Trust Factor : Availability

Le trust factor **availability** charge les données grâce aux librairies “Sensor” et “DB_Connector”. Après formatage des données avec un intervalle fixe entre chaque valeur, le nombre de données manquantes (avant interpolation) et compté (Figure 16). Ce formatage des données est nécessaire pour le calcul des autres *trust factors*. Ensuite la valeur de confiance est enregistrée dans la base de données InfluxDB.

```
def availability(records):
    missing = countMissingRecords(records)
    total = len(records)
    return 1 - missing/total

def countMissingRecords(records):
    return (records.missing == True).sum()
```

Figure 16. Availability

3.2.5 Trust Factor : Regularity

Le trust factor **regularity** permet de compléter availability en comptabilisant les messages arrivés avec un intervalle de temps trop court ou trop long. En effet, lorsque le périphérique LoRa détecte une erreur, il redémarre afin de se réinitialiser. Dans ce cas, on peut se retrouver avec des messages en provenance d’un même périphérique LoRa avec des intervalles de temps irréguliers. Ce phénomène peut indiquer un potentiel problème au niveau du périphérique.

Pour le calcul de ce trust factor deux méthodes ont été essayées, chacune produisant un résultat très semblable. La première consiste à compter uniquement les valeurs de différence de mesures qui sortent d’une tolérance de 10% autour de la valeur de référence (Figure 18). La deuxième consiste à appliquer la différence entre la valeur obtenue et la valeur de référence dans une fonction pseudo-gaussienne, et d’effectuer la moyenne sur la fenêtre d’observation (Figure 19). La figure 17 montre comment les intervalles de temps entre deux mesures sont obtenues avant d’appliquer une des deux méthodes.

La deuxième méthode produit une valeur légèrement inférieure, ce qui est normal car contrairement à la première méthode, l’erreur dans la marge de 10% est également prise en compte.

```
prev_date = None
delta_t = []
for t in result['T']:
    if prev_date != None:
        delta_t.append((t-prev_date).total_seconds())
    prev_date = t
```

Figure 17. Regularity, calcul de la différence en temps : delta t

```
cpt = 0
tolerance = IDEAL_DELTA / 20
for i in delta_t:
    if i > IDEAL_DELTA + tolerance or i < IDEAL_DELTA - tolerance:
        cpt += 1
tf = 1 - (cpt / nbRecords)
```

Figure 18. Regularity, première méthode de calcul de TF

```
tf = []
alpha = 5
for i in delta_t:
    tf.append(math.exp(-math.pow((IDEAL_DELTA - i)/alpha, 2)))
tf = np.mean(tf)
```

Figure 19. Regularity, deuxième méthode de calcul de TF

3.2.6 Trust Factor : Précision par calcul du MSE

Le trust factor **mse_precision** (Mean Squared Error) charge les données de la même façon que pour availability. Ensuite, la distance entre chaque courbe est calculée grâce à la fonction `mean_squared_error()` (Figure 20). Le résultat de ce calcul, Root Mean Square Error (RMSE) est un chiffre qui peut être inférieur à 1 si les deux courbes sont proches, ou un chiffre de l'ordre de 10^3, si les courbes sont éloignées, comme ça peut être le cas pour l'humidité du sol. Il est alors nécessaire de normaliser le résultat, pour cela, deux méthodes ont été utilisées (voir les deux fonctions dans la figure 20). La première consiste à utiliser la formule suivante, où y_{max} (respectivement min) est la valeur max (resp. min), des valeurs mesurées par les capteurs soumis à l'algorithme de trust pendant la période considérée.

$$NRMSE = RMSE / y_{max} - y_{min} \quad [20]$$

Cette méthode produit des résultats qui ne sont pas idéaux car les valeurs y_{max} et y_{min} varient en fonction de la période d'observation. Pour mieux faire, il aurait fallu prendre la valeur min et max sur une période beaucoup plus grande, d'une année par exemple. Malheureusement, à ce stade du projet, nous ne disposons pas d'assez de données pour cela.

Avec des données sur une année, la méthode suivante se basant sur la différence entre le $25^{ème}$ et le $75^{ème}$ percentile est plus adaptée pour éviter les *outliers*.

$$NRMSE = RMSE / Q1 - Q3 \quad [21]$$

La deuxième méthode est d'appliquer le résultat RMSE à une fonction pseudo-gaussienne, dont le résultat est compris entre 0 et 1.

$$NRMSE = \exp(- (RMSE / \alpha)^2) \ , \alpha > 0$$

Le facteur d'échelle αlpha peut être adapté afin de modifier la sensibilité (largeur de la courbe pseudo-gaussienne). Pour les calcul effectué, la valeur αlpha a été estimé de façon empirique à environ 10% de l'amplitude moyenne de la courbe. Ainsi, les valeurs suivantes ont été choisies pour le calcul du trust factor :

- Temperature de l'air : 2,5
- Humidité de l'air : 10
- Humidité du sol : 800
- Temperature du sol : 0,6

```python
def mse(recordsB_pred, recordsA_true):
    # squared=False for RMSE
    mse = mean_squared_error(recordsA_true, recordsB_pred, squared=False)
    return mse

def mse_compare(records, record, nbRecords=20, level="decoded.airtemp"):
    ret = []
    for (deveui, value) in records:
        mse_value = []
        for (deveui_, value_) in records:
            if deveui != deveui_:
                mse_value.append(mse(value.loc[:,level], value_.loc[:,level]))
        ret.append((deveui, np.mean(np.array(mse_value))))
    return ret

def mse_to_tf_normalize_1(mse_list, records, level):
    df = []
    for (_, value) in records:
        df.append(pd.DataFrame(value.loc[:,level]))
    max_value = pd.concat(df).max()[level] - pd.concat(df).min()[level]
    ret = []
    for deveui, mse in mse_list:
        norm = mse / max_value
        ret.append((deveui, 1 - (1 if norm > 1 else norm)))
    return ret

def mse_to_tf_normalize_2(mse_list, alpha):
    ret = []
    for deveui, mse in mse_list:
        ret.append((deveui, math.exp(-math.pow((mse - 0)/alpha, 2))))
    return ret
```

Figure 20. Trust factor précision MSE avec deux normalisation différentes

Ce calcul est basé sur le postulat que la majorité des capteurs se comportent bien. Ainsi, comme il y a quatre valeurs parmi les capteurs de la même couverture, on admet qu'un seul capteur se comporte mal, et que les trois autres se comportent bien. Si deux capteurs indiquent des valeurs semblables entre eux, et deux autres capteurs indiquent des valeurs différentes mais semblables entre eux, on ne peut pas savoir à quelle paire se fier, comme on peut le voir sur la figure 21 suivante.

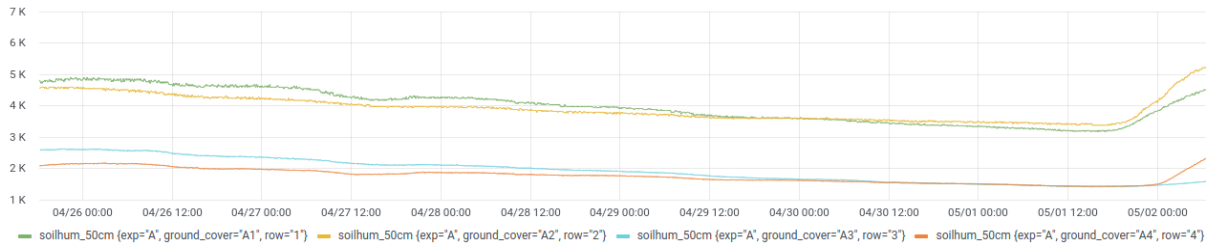


Figure 21. Humidité du sol à 50 cm, couverture A, du 26/04 au 02/05

3.2.7 Trust Factor : Précision par calcul de LLSE

Le trust factor **llse_precision** (Linear Least Squares Estimation) (Figure 22) a également été appliqué aux données, mais après observation des résultats, il ne s'est pas révélé concluant.
Le principe consiste à effectuer une estimation linéaire par rapport aux autres capteurs de la même couverture, selon la formule suivante :

$$\text{Estimation} = \text{moy}(A) + A.\text{cov}(B / \text{var}(B) * (\text{last}(B) - \text{moy}(B))$$

"A" représente la série de valeurs pour laquelle l'estimation est calculée, et "B" représente une série de valeurs, considérées de référence, appartenant à la même couverture végétale. Ensuite, afin de normaliser le résultat, celui-ci est appliqué à une fonction pseudo-gaussienne dont la formule est la suivante :

$$N_{Estimation} = \exp(- ((\text{Estimation} - \text{last}(A)) \ / \ \alpha)^2) \quad , \alpha > 0$$

Comme décrit précédemment, Le facteur d'échelle αlpha peut être adapté afin de modifier la sensibilité de la fonction. Les valeurs suivantes ont été choisies pour le calcul du trust factor:

- Température de l'air : 1
- Humidité de l'air : 5
- Humidité du sol : 50
- Température du sol : 1

Ces valeurs de alpha sont peut-être pas optimales, et une expérimentation supplémentaire sur d'autres données en faisant varier alpha permettrait d'affiner le choix.

```python
def llse(recordsA, recordsB):
    estimation = recordsA.mean() + recordsA.cov(recordsB)/recordsB.var() * (recordsB.iloc[-1] - recordsB.mean())
    return estimation

def compute_tf(records, level, alpha, record):
    estimations = []
    for (deveui, value) in records:
        esti = []
        for (deveui_, value_) in records:
            if deveui != deveui_:
                esti.append(llse(value.loc[:,level], value_.loc[:,level]))
        estimations.append((deveui, esti))

    tf_data = []
    cpt = 0
    for deveui, esti in estimations:
        avgEst = np.mean(np.array(esti))
        tf = math.exp(-math.pow((avgEst - records[cpt][1].loc[:,level].iloc[-1])/alpha, 2))
        tf_data.append((deveui, tf))
        cpt += 1
    return tf_data
```

Figure 22. Trust factor précision LLSE

3.3 Résultats du Trust Model

Afin d'avoir une vue plus globale sur les valeurs obtenues par le trust model, l'algorithme a été exécuté plusieurs fois, et les données de chaque trust factor ont été enregistrées dans la base de données influxDB. Les résultats peuvent être visualisés sous forme de graphiques.

Le trust score global n'a pas été calculé car il s'agit de la moyenne pondérée de chaque trust factor; Pour l'analyse, il est plus pertinent d'observer chaque trust factor indépendamment.
Entre chaque point du graphique, il y a un incrément de la date d'une heure.
Trois valeurs différentes de l'argument *nbRecords* ont été utilisées. Pour rappel, la date correspond à la date de la dernière mesure prise en compte, et le nombre d'échantillons (nbRecords) correspond au nombre de mesures antérieures à cette date.
Pour les test, les trust factors ont été calculé entre les dates suivante :

- Du 24 avril au 29 avril, avec *nbRecords* = 120 (soit 12 heures)
- Du 30 avril au 10 mai, avec *nbRecords* = 60 (soit 6 heures)
- Du 14 mai au 19 mai, avec *nbRecords* = 20 (soit 2 heures)

Pour l'**availability**, on remarque que seuls les capteurs de la rangée 4 hormis B4 ont des valeurs moins bonnes que les autres (Figure 23 et 24). En vérifiant la valeur de RSSI (Received Signal Strength Indication), on remarque qu'en fonction du périphérique ou de la date, la force du signal varie entre environ -85dBm et -110dBm. Une raison qui expliquerait que la rangée 4 se comporte différemment, est le fait que les personnes travaillant dans les vignes ont changé l'orientation de l'antenne LoRa.

Figure 23. Availability pour les stations de la rangée 1, 2 et 3 et la station B4

Figure 24. Availability pour les stations : A4 - C4 - D4 - E4 - F4

En observant l'évolution des résultats du trust factor **LLSE** pendant plusieurs jours, on se rend compte qu'il y a une grande oscillation entre 1 et 0, en particulier pour l'humidité de l'air ou du sol (Figure 28). La raison qui explique cela, est la forte variabilité entre deux mesures pour ces capteurs. Il est donc difficile de faire une estimation linéaire sur des valeurs qui oscillent beaucoup.
En revanche, en observant les résultats pour la température du sol (Figure 26), on remarque que le résultat du trust factor est souvent proche de 1. De plus, plus la fenêtre d'observation est petite (à droite sur le graphique), plus le résultat indique une bonne estimation de la méthode LLSE.
Ce trust factor est donc adapté uniquement pour une petite fenêtre d'observation mais avec un nombre d'échantillons suffisant, de plus, la variation entre deux mesures consécutives doit être faible.

Pour référence, la figure 25 montre les mesures en entrée du trust factor dont le résultat est affiché sur la figure 26.

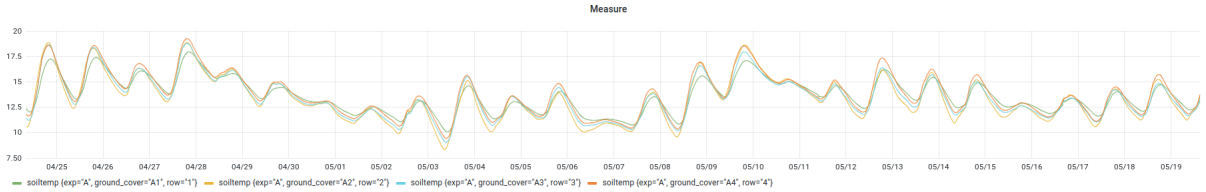


Figure 25. Couverture A, température du sol

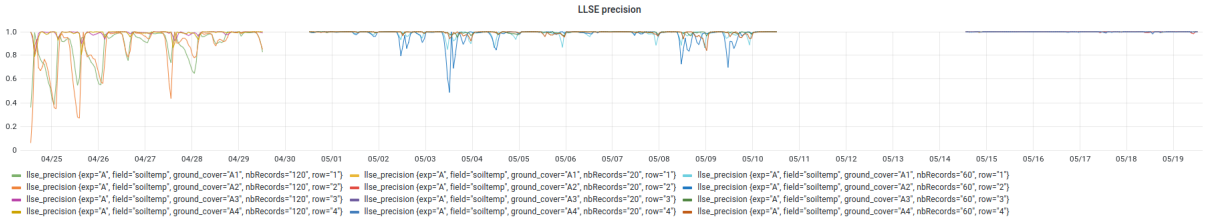


Figure 26. Couverture A, trust factor LLSE pour la température du sol

Pour référence, la figure 27 montre les mesures en entrée du trust factor dont le résultat est affiché sur la figure 28.

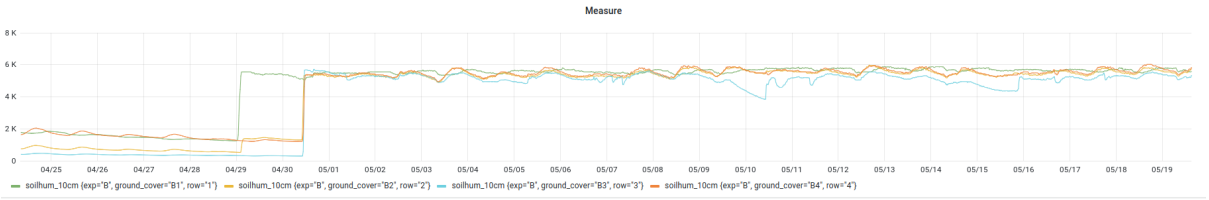


Figure 27. Couverture B, humidité du sol à 10cm

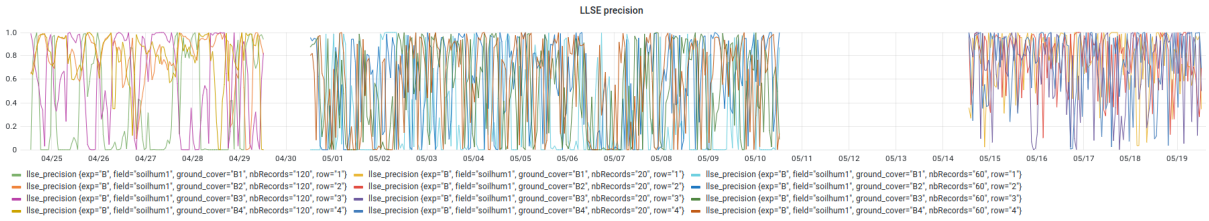


Figure 28. Couverture B, trust factor LLSE pour l'humidité du sol à 10cm

Pour le trust factor se basant sur l'erreur MSE entre chaque capteur d'une même couverture, les résultats sont différents en fonction de la normalisation choisie.
La première normalisation (Figure 30 en haut) correspond à la valeur divisée par (y_{max} - y_{min}), et la deuxième normalisation (Figure 30 en bas) correspond à l'utilisation de la fonction pseudo-gaussienne.
Pour référence, la figure 29 montre les mesures en entrée du trust factor dont le résultat est affiché sur la figure 30.

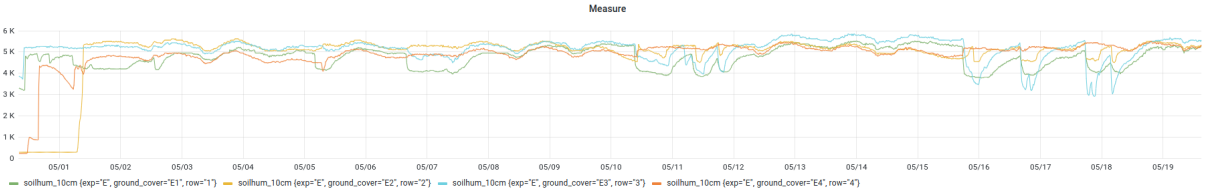


Figure 29. Couverture E, humidité du sol à 10cm

Figure 30. Couverture E, trust factor MSE pour l'humidité du sol à 10cm

Pour référence, la figure 31 montre les mesures en entrée du trust factor dont le résultat est affiché sur la figure 32.

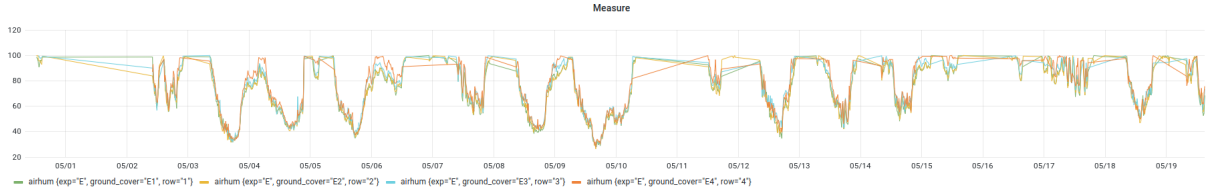


Figure 31. Couverture E, humidité de l'air

Figure 32. Couverture E, trust factor MSE pour l'humidité de l'air

Si on compare les deux trust factors availability et regularity, on constate qu'ils sont très proches (Figure 33). Ce qui n'est pas étonnant puisqu'ils vérifient tous les deux que les messages arrivent bien dans le temps attendu, mais la méthode est différente. Le trust factor regularity pourrait être amélioré en le partageant en deux trust factors différent : un qui

compte seulement les messages avec un intervalle trop petit (message en double), et un qui compte seulement les messages avec un intervalle trop grand (message manqué).
Sur la figure 33, le premier graphique représente la quantité de messages arrivés en dehors de la tolérance de 10%, et le deuxième graphique représente l'erreur moyenne de l'intervalle entre deux mesures par rapport à l'intervalle de référence.

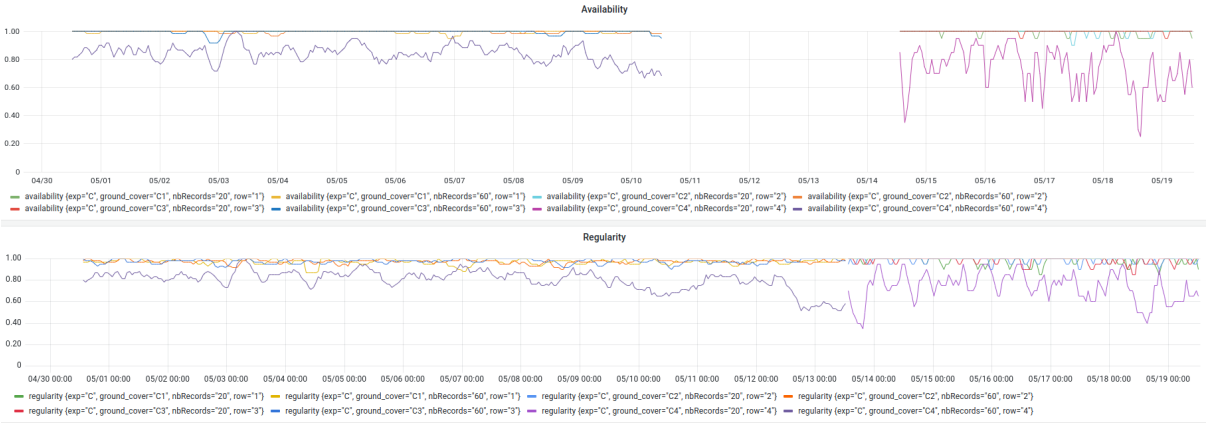


Figure 33. Couverture C, Availability et Regularity

Conclusion

Ce projet, soumis aux conditions climatiques et aux évolutions de la vigne ainsi que de sa couverture, est trop court pour émettre une conclusion définitive. Des données futures sont nécessaires pour continuer ce travail, et avoir plus de recul sur les données.
L'installation et la mise en place de la collecte des données devaient être effectuées rapidement afin d'enregistrer le plus tôt possible des données. La création des supports et l'installation du matériel dans les vignes ont mis plus de temps qu'espéré. En parallèle, le programme exécuté par les boîtes à été développé, avec les contraintes géographiques des gateways, contraignant les test de réseaux. Ensuite, il y a eu quelques problèmes avec l'enregistrement des mesures dans la base de données, réglé par la suite. Enfin, la création du dashboard était moins critique car l'enregistrement continu des données était opérationnel.
Finalement, tout est fonctionnel et exploitable. Le résultat est satisfaisant, et répond à nos attentes. Après observation des résultats intermédiaires, les agronomes ont pu adapter leur besoin et demander des données supplémentaires, tel que l'écart-type, ou la somme des températures (= degré jour).

Le trust factor étant générique, peu de transformation était envisagée au début. Finalement, les données et le contexte étant considérablement différents, certaines adaptations ont été nécessaires. Nous pouvons voir que dans les vignes les trust factors produisent des résultats très différents en fonction du type de mesure (humidité ou température). Nous ne pouvons donc pas utiliser les trust factors de manière identique sur les différents capteurs, même si ils sont physiquement au même endroit et dans le même contexte. D'autres trust factors peuvent encore être envisagés afin de diversifier le paramètre de confiance. La précision basée sur LLSE peut donner des résultats exploitables pour la température du sol, comme des résultats totalement inutilisables pour l'humidité du sol. Ceci est expliqué par la variabilité des données. De plus, tous ces trust factors doivent normaliser leur résultat, et pour cela, plusieurs méthodes sont possibles, il faut alors choisir la plus adaptée, avec les bon paramètres.
Le trust factor nous à permis de mettre en évidence les problèmes de réception des messages; en effet, il est fastidieux de vérifier sur tous les messages reçus, combien sont manquants ou en double, et de comparer les capteurs entre eux. Une possibilité d'amélioration de l'installation dans les vignes serait de déplacer l'antenne à l'aide d'une rallonge, et la placer en haut des mâts afin d'éviter les interférences avec les lignes en métal servant de tuteur. D'autre part l'affichage des niveaux de réception RSSI sous forme de graphique temporel pourrait donner un indication sur la qualité de la liaison radio.

Finalement, à l'heure actuelle, l'algorithme de confiance ne semble peu adapté à ce contexte pour les raisons suivante :

- Le nombre de voisins est trop faible pour une confiance se basant sur ceux-ci. En effet, sur quatre capteurs, si deux voisins se comportent mal, l'algorithme n'est plus pertinent.
- Nous n'avons pas de données sur l'évolution des plantes (vignes ou couverture du sol), en effet, après que les couvertures temporaires se sont bien développées, celles-ci sont fauchées. L'algorithme ne prend pas en compte cela.

- Les données météorologiques sont peut-être nécessaires pour justifier une variation imprévue des valeurs provoquée par la pluie ou le soleil.
- Considéré l'hypothèse qu'une fois les couverts temporaires fauchés, la différence entre deux couvertures est si faible qu'elles pourraient devenir voisine afin d'avoir plus de capteurs de référence pour le calcul des trust factors.
- La quantité de données n'est pas suffisante pour affiner et adopter les différents trust factors.

Finalement, malgré quelques incertitudes concernant la pertinence de l'algorithme de trust model pour ce contexte, ce travail a été très enrichissant sur l'analyse des données. De plus les données récoltées sont très utiles pour les agronomes, et le projet est intéressant car il vise à limiter l'usage des herbicides et promouvoir la biodiversité.

Sources

1. [No title]. [cited 14 Jun 2021]. Available: https://lsds.hesge.ch/wp-content/uploads/2018/05/Master-Report-Bouchedakh.pdf

2. Visuelle C-C. sMart EDge fabric for Iot Applications. [cited 7 Jun 2021]. Available: https://lsds.hesge.ch/smart-edge-fabric-for-iot-applications/

3. 0.44US $. [cited 8 Jun 2021]. Available: https://www.aliexpress.com/item/32345244850.html?spm=a2g0o.productlist.0.0.48465a77eM4I8X&algo_pvid=716d0820-c864-452d-8590-35cfddd36d1b&algo_expid=716d0820-c864-452d-8590-35cfddd36d1b-18&btsid=0bb0624416039659750012692ed46b&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_

4. micromet, » M. Waterproofing a Capacitance Soil Moisture Sensor. Instructables; 2 Nov 2019 [cited 8 Jun 2021]. Available: https://www.instructables.com/Waterproofing-a-Capacitance-Soil-Moisture-Sensor/

5. Soil Moisture Sensor: Which Soil Sensor is Perfect for You? 12 Sep 2017 [cited 8 Jun 2021]. Available: https://www.metergroup.com/meter_knowledgebase/which-soil-sensor-is-perfect-for-you/

6. Libelium » Connecting sensors to the Cloud. 10 Jul 2020 [cited 7 Jun 2021]. Available: https://www.libelium.com/

7. Plug&Sense! Waspmote Encapsulated version easyly deploy IoT networks. 25 Aug 2020 [cited 8 Jun 2021]. Available: https://www.libelium.com/iot-products/plug-sense/

8. Docs. [cited 8 Jun 2021]. Available: https://development.libelium.com/plug-and-sense/documentation

9. Waspmote. 11 Jul 2020 [cited 8 Jun 2021]. Available: https://www.libelium.com/iot-products/waspmote/

10. Docs. [cited 8 Jun 2021]. Available: https://development.libelium.com/waspmote/documentation

11. Irrometer Sensors. [cited 7 Jun 2021]. Available: https://www.irrometer.com/sensors.html

12. [No title]. [cited 13 Jun 2021]. Available: http://www.libelium.com/downloads/documentation/agriculture_sensor_board_3.0.pdf#page=78

13. [No title]. [cited 7 Jun 2021]. Available: http://www.libelium.com/downloads/documentation/agriculture_sensor_board_3.0.pdf#page=37

14. Temperature, Humidity and Pressure Sensor Probe. [cited 7 Jun 2021]. Available: https://www.mgsuperlabs.in/brands/libelium/smart-agriculture/4869/temperature-humidity-and-pressure-sensor-probe

15. Modulation & Data Rate. [cited 13 Jun 2021]. Available: https://www.thethingsnetwork.org/docs/lorawan/modulation-data-rate/

16. [No title]. [cited 17 Jun 2021]. Available:

https://www.libelium.com/wp-content/uploads/2013/02/data_frame_guide.pdf

17. InfluxDB data elements. [cited 13 Jun 2021]. Available: https://docs.influxdata.com/influxdb/cloud/reference/key-concepts/data-elements/

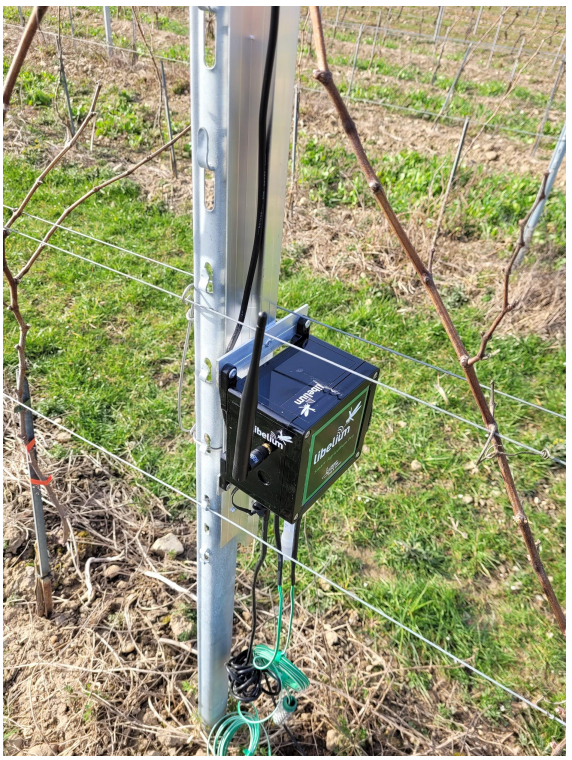
18. InfluxDB data source. [cited 7 Jun 2021]. Available: https://grafana.com/docs/grafana/latest/datasources/influxdb/#query-languages

19. [No title]. [cited 14 Jun 2021]. Available: https://lsds.hesge.ch/wp-content/uploads/2018/05/Master-Report-Bouchedakh.pdf

20. How to normalize the RMSE. [cited 9 Jun 2021]. Available: https://www.marinedatascience.co/blog/2019/01/07/normalizing-the-rmse/

21. How to normalize the RMSE. [cited 9 Jun 2021]. Available: https://www.marinedatascience.co/blog/2019/01/07/normalizing-the-rmse/

Annexes


libelium


libelium


libelium


libelium
Smart Agriculture Pro


libelium


libelium