

Station d'acquisition de données météorologiques

Octobre 26, 2022



Sommaire

- Contexte du projet et ses objectifs
- Equipe
- Organisation et étapes du projet
- Diagrammes fonctionnel et technique
- Matériels utilisés
- Réalisation
- Démonstration
- Pistes d'évolution
- Remerciements
- Conclusion
- Des questions ?



Contexte et objectifs

Projet : Récupération et intégration de données terrain.

- Balise:
 - Collection de données physiques par capteurs
 - Envoi au Maître à intervalle régulier
 - Connecté au Maître
- Maître:
 - Récupération des données de Balise
 - Envoi au serveur TCP
 - Gestion de feux
 - Connecté à la Balise
- Serveur Ubuntu 20.03 :
 - Réception des données
 - Stockage dans base de données
 - Accès aux données par application console



L'équipe

- Philippe RASOAMAHENINA
Diplôme d'ingénieur électronique
Développeur C/C++/Python /
Jeune diplômé

- Noella PINZI
Diplôme en génie électrique
expérience: en Automatismes
industriel

- Nizar REZAIGUI
Master II Ingénierie Logiciel /
Développeur C/C++/Python
devOps : Docker

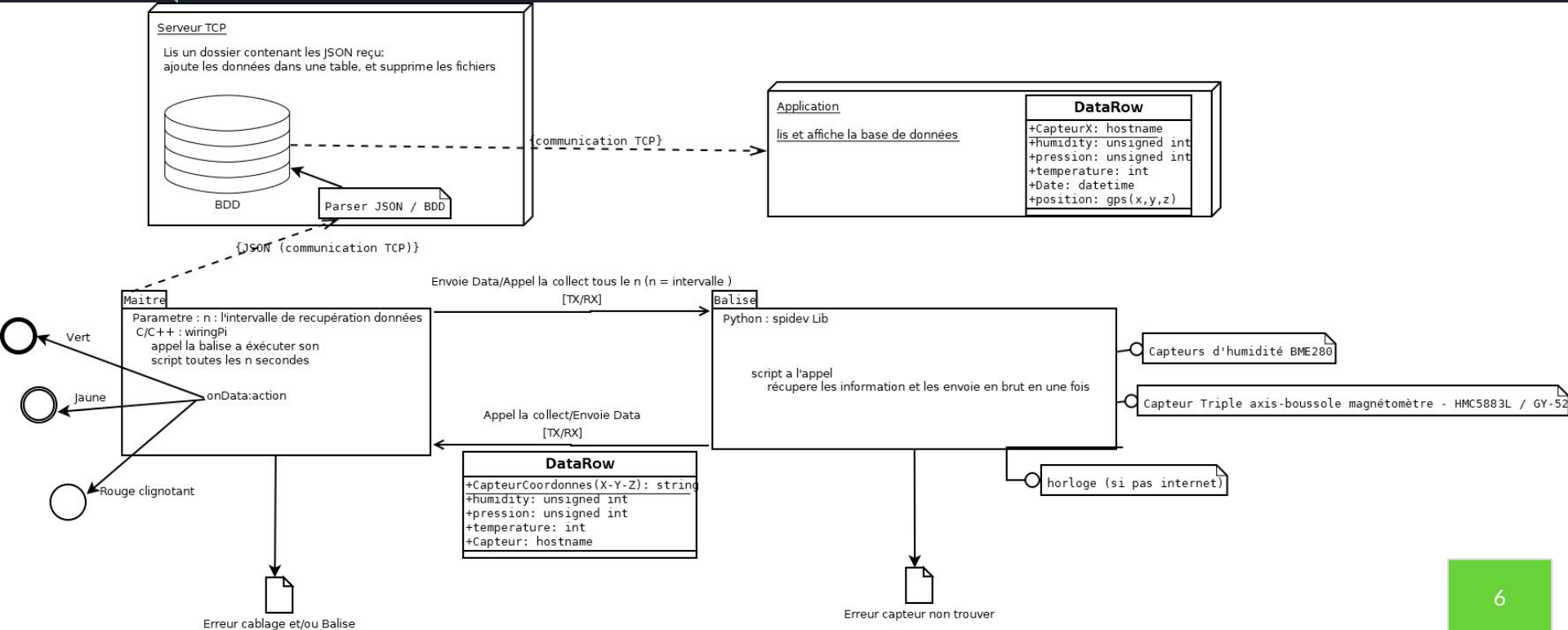
- Merzouk MENHOUR
Master II Informatique
Développeur C
Développeur Java Back



Etapes du projet

- Lecture et compréhension du sujet
- Réalisation du diagramme fonctionnel
- Mise en place du dépôt Git
- Répartition des tâches
- Réunions quotidiennes pour faire le point:
 - avancées
 - blocages
 - échanges de données entre différentes parties

Diagramme fonctionnel



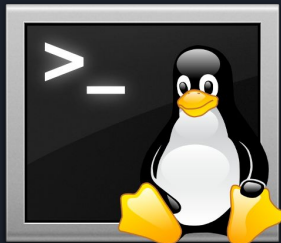


Organisation

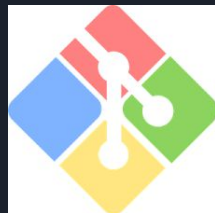
Personne	Tâche
Philippe RASOAMAHENINA	Capteurs, RPI Balise et Maître: communication série
Nizar REZAIGUI	RPI Maître: envoi/réception des données en TCP
Merzouk MENHOUR	Serveur TCP, BDD, application console
Noella PINZI	Feu tricolore RPI Maître, parse JSON

Les outils exploités

Linux (shell)



Git



{JSON}



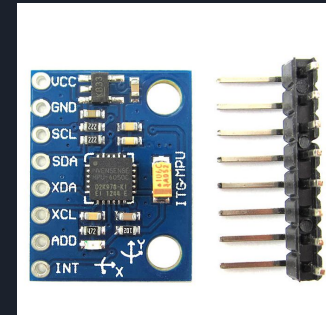
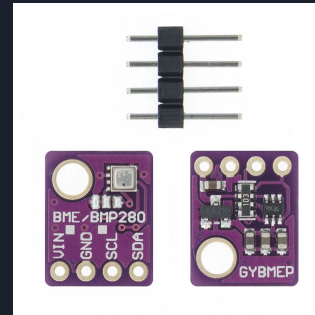
Github



<https://github.com/Krs-ceri/RaspberryPi/>

Raspberry Pi, BME280 et GY-521

- Raspberry Pi 3B+
 - Mini ordinateur
 - Système d'exploitation: Raspberry Pi OS (Linux)
 - CPU: 1.4 GHz
 - RAM: 1Go LPDDR2
 - Ports USB 2.0: 4
 - Ethernet
 - Wifi
- Capteur BME280
 - Température
 - Humidité
 - Pression
- Capteur GY-521
 - Boussole 3 axes
 - Magnétomètre
 - Accélération



Protocole UART

- UART : Universal Asynchronous Receiver/Transmitter
- Seulement 2 fils: RX (Réception) / TX (Transmission)
- Pour communication série
- Bidirectionnel (communication dans les 2 sens)
- Transmission sous forme de trames (bytes/octets)



Captures des grandeurs physiques

- Langage utilisé: C
- Modification pour écriture dans fichiers
- Envoi des données dans log_bme280.txt
- Envoi des données dans log_gyro.txt

```
16 #define MPU6050_PWR_MGMT_1      0x6B // R/W
17 #define MPU6050_I2C_ADDRESS      0x68 // I2C
18
19 Sensor::Sensor()
20 {
21     fd = wiringPiI2CSetup(MPU6050_I2C_ADDRESS);
22     if (fd == -1)
23         return;
24
25     wiringPiI2CReadReg8 (fd, MPU6050_PWR_MGMT_1);
26     wiringPiI2CWriteReg16(fd, MPU6050_PWR_MGMT_1, 0);
27 }
28
29 int8_t Sensor::getGyroX()
30 {
31     return (int8_t) wiringPiI2CReadReg8(fd, MPU6050_GYRO_XOUT_H);
32 }
```



Fichiers logs des capteurs

- Langage utilisé: Python
- Lecture des 2 fichiers de logs
- Récupération du texte dans les logs

```
20  # Open sensors log files in read mode
21  log_file = open("/home/pi/sensors-scripts/log_bme280.txt", "r")
22  log_file1 = open("/home/pi/sensors-scripts/log_gyro.txt", "r")
23
24  # Reading log files
25  data = log_file.read()
26  data = data + log_file1.read()
```

Communication Balise vers Maître

- Paramétrage liaison série

```
28 # Serial communication parameters between Master and Balise
29 ser = serial.Serial(
30     port='/dev/ttyS0',
31     baudrate = 9600,
32     parity=serial.PARITY_NONE,
33     stopbits=serial.STOPBITS_ONE,
34     bytesize=serial.EIGHTBITS,
35     timeout=None
36 )
```

- Envoi des données au Maître

```
28 # Encoding unicode string to byte strings
29 data = data.encode('utf-8')
30
31 # Sending data to Master through Serial connection
32 ser.write(data)
```

Réception et stockage côté Maître


- Réception des données
- Stockage des données dans un fichier log.json

```
53 while 1:
54     ##### red = gpio 26, yellow = 19, green = 13 => don't believe the sticker shown on camera #####
55
56     log_file = open(s, mode="w", encoding="utf-8")
57     for i in range(16):
58         # By default, yellow led is on when program is active
59         led_on(yellow)
60
61         # Reading serial text received from Balise Rpi
62         x=""
63         x=ser.readline()
```

```
80     log_file = open(s, mode="a", encoding="utf-8")
81     #log_file = open("/home/pi/logs/log_gyro.txt", "w")
82
83     if x=="":
84         print("rien")
85     else:
86         # Decode serial data from bytes strings into utf-8 unicode strings
87         x=x.decode('utf-8')
88         # Writing data in log file
89         log_file.write(x)
90
91     # Closing log file
92     log_file.close()
```

Transmission de données en format Json: Maître -> TCP



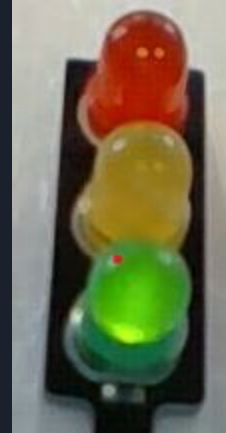


Transmission de données en format Json: Maître -> TCP

```
{
  capteur:{
    str_date: '23/10/2022 23:01:22',
    sensor_id: 'BME280',
    temperature: 24.894,
    humidity: 87.377,
    pressure: 689.989,
    gyro_x: -0.015,
    gyro_y: -0.008,
    gyro_z: 0.000,
    accel_x: -0.000,
    accel_y: -0.000,
    accel_z: 0.004
  }
}
```


Gestion de Feu Tricolore

- Rouge : Arrivée de nouvelles données
- Orange : Programme actif
- Vert: Transfert vers serveur TCP

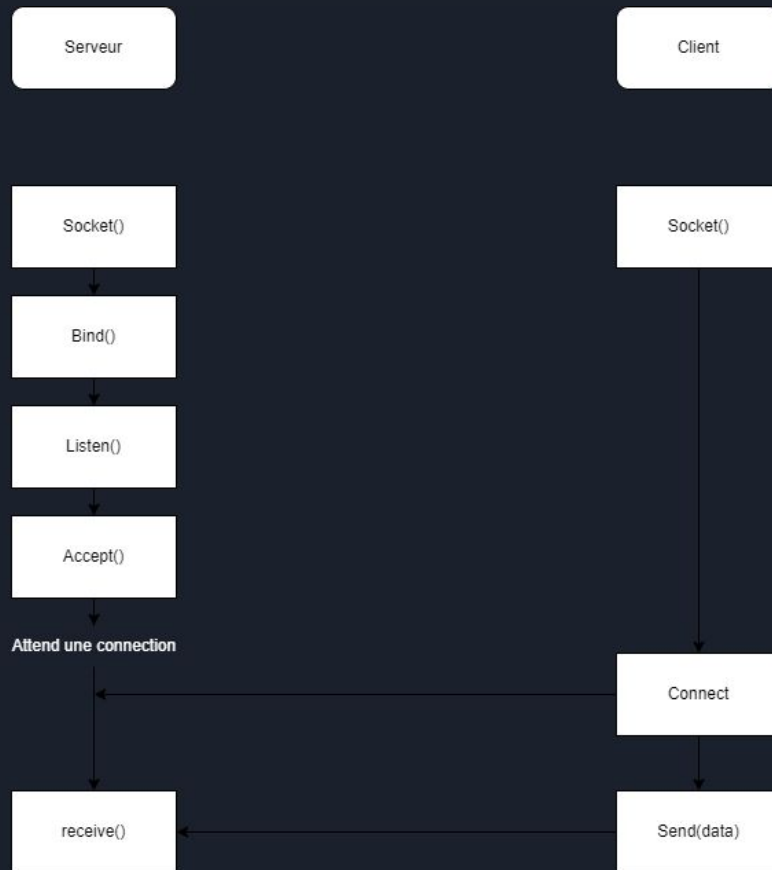


Récupération des données



Récupération des données

La congestion intervient lorsque trop de sources tentent d'envoyer trop de données trop vite pour que le réseau soit capable de les transmettre. Ceci entraîne la perte de nombreux paquets et de longs délais.





Serveur TCP: partie acquisition

- Données déposées par RPi Maître
- Vérification du dossier par script chaque minute
- Insertion dans la base de données
- Fichier déposé
 - Dossier succès, fichier au format JSON correct
 - Dossier échec, fichier au format JSON non correct



Serveur TCP: partie exposition

- Service d'exposition via BOOST ASIO sur serveur TCP
- Service BOOST ASIO en C++
- Méthodes pour récupérer données de BDD



Application Console

- Inclus client sous BOOST en C++
- Interrogation à distance du service BOOST sur serveur TCP
- Récupération données sous format JSON
- Affichage sur console



Démonstration



Pistes d'évolution

- Rendre le code plus générique :
 - Pouvoir ajouter de nouveaux capteurs
 - Adapter la récupération et gestion des données par la BDD
- Adapter le projet à une carte préprogrammée à la place d'une Raspberry Pi
- Ajouter Token JWT pour une authentification sécurisée
- Utiliser quelque chose de plus robuste que SQLite pour stocker plus de données
- Réaliser une suite de tests sur l'ensemble des modules
- Faire un seul programme "Maître" afin de centraliser



Remerciements

- Équipe pédagogique d'AJC formation
- Équipe administration d'AJC Formation
- Astek



Conclusion

- Familiarisation avec de nouvelles bibliothèques (BOOST ASIO, Wiringpi,...)
- Utilisation des notions acquises durant la formation
- Renforcement du travail en équipe
- Autonomie
- Projet concret
- Gain d'expérience

Merci pour votre attention!

Des questions ?