Université Grenoble Alpes

Master 2 - Génie Informatique

E-Service pour l'IOT

# Projet CarLoRA

*Auteurs :*
Paul Carretero
Trung Nguyen

*Responsable :*
Didier Donsez

9 avril 2018

# Table des matières

**1 Introduction** **2**

**2 Présentation du projet** **2**
2.1 Motivations . . . . . 2
2.2 Choix Techniques . . . . . 2
2.2.1 Appareil ELM327 ODB2 . . . . . 2
2.2.2 Carte Pycom LoPy et Pytrack . . . . . 3
2.2.3 Métriques de conduites retenues . . . . . 4

**3 Spécifications techniques** **5**
3.1 Appareil ELM327 et norme ODB2 . . . . . 5
3.2 Carte Pycom LoPy et Pytrack . . . . . 9

**4 Projet original (LoPy et ELM327)** **9**
4.1 Première réalisation . . . . . 9
4.2 Interface de visualisation Jyse . . . . . 15
4.3 Problème rencontré . . . . . 15

**5 Solution réalisée** **16**
5.1 Back-End JAVA . . . . . 16
5.1.1 Simulation de données de conduites . . . . . 17
5.1.2 Récupération de données de conduites . . . . . 18
5.2 Blockchain Ethereum . . . . . 18
5.2.1 Généralités . . . . . 18
5.2.2 Private Network . . . . . 19
5.2.3 Smart Contract . . . . . 20
5.2.4 Stockage des données : Bibliothèque Web3J . . . . . 20

**6 Conclusion** **23**

**7 Références** **23**

# 1 Introduction

Nous avons choisi de participer à la réalisation du projet CarLoRA. L'objectif de ce projet était de traiter des données de conduites.

Nous avons utilisé une board Pycom LoPy sur une carte d'extension Pytrack munie d'un GPS notamment. Les données de conduites ont été récupérées par un appareil ODB2 ELM 327.

Nous n'avons pas pu réaliser ce projet en raison de problèmes techniques. En effet les technologies Bluetooth de la carte LoPy et de l'ELM327 cont incompatible. Une solution a été de simuler les données retournées par la carte LoPy en LoRA et de les traiter, puis les stocker dans une blockchain Ethereum.

# 2 Présentation du projet

## 2.1 Motivations

L'idée du projet CarLoRA est de récupérer, analyser et stocker les données de conduite d'un utilisateur. L'usage principal de telles données est la personnalisation du contrat d'assurance d'un utilisateur. Si la conduite est jugée plus risquée alors sa prime d'assurance est augmentée et inversement dans le cas d'une "bonne" conduite. Il s'agit du concept de "Pay how you drive" utilisé dans les assurances outre-atlantique principalement.

La principale limite de cette méthode est le respect de la vie privée des utilisateurs dans la mesure où leurs déplacements sont enregistrés.

## 2.2 Choix Techniques

Le projet CarLoRA consiste donc à récupérer des informations de conduite par le biais d'un appareil ODB2 ELM327, de les envoyer par Bluetooth à une carte Pycom LoPy pour être envoyée en LoRA à un broker située hors de la voiture.

### 2.2.1 Appareil ELM327 ODB2

Cet appareil se branche à la prise ODB2 de la voiture. Il est principalement utilisé pour effectuer un diagnostic de la voiture. En effet le protocole OBD2 défini l'interface standard d'accès au système de diagnostics embarqués. Plus récemment la norme est utilisée pour contrôler les différentes données relatives à la pollution et à la consommation du véhicule.

Il existe de nombreuses applications Android permettant d'analyser les valeurs retournées par l'appareil ELM327. Ces applications permettent de réaliser un diagnostic ou de surveiller certaines valeurs

Dans le cas du projet certaines métriques retournées peuvent être utilisées pour analyser la conduite utilisateur, comme par exemple la vitesse ou l'accélération.

FIGURE 1 – Module ELM327 - ODB2

### 2.2.2 Carte Pycom LoPy et Pytrack

La carte PyCom Lopy dispose d'une connexion Bluetooth Low Energy et d'une antenne LoRA pour émettre des informations vers un broker LoRA. L'idée était ici de connecter la carte LoPy au module ELM327 en Bluetooth et d'envoyer les informations jugées utiles à l'analyse de la conduite en LoRA à un Broker.

Par ailleurs, la carte LoPy et sa board Pytrack offrent également d'autres fonctionnalités exploitable pour analyser la conduite. On pourra par exemple citer

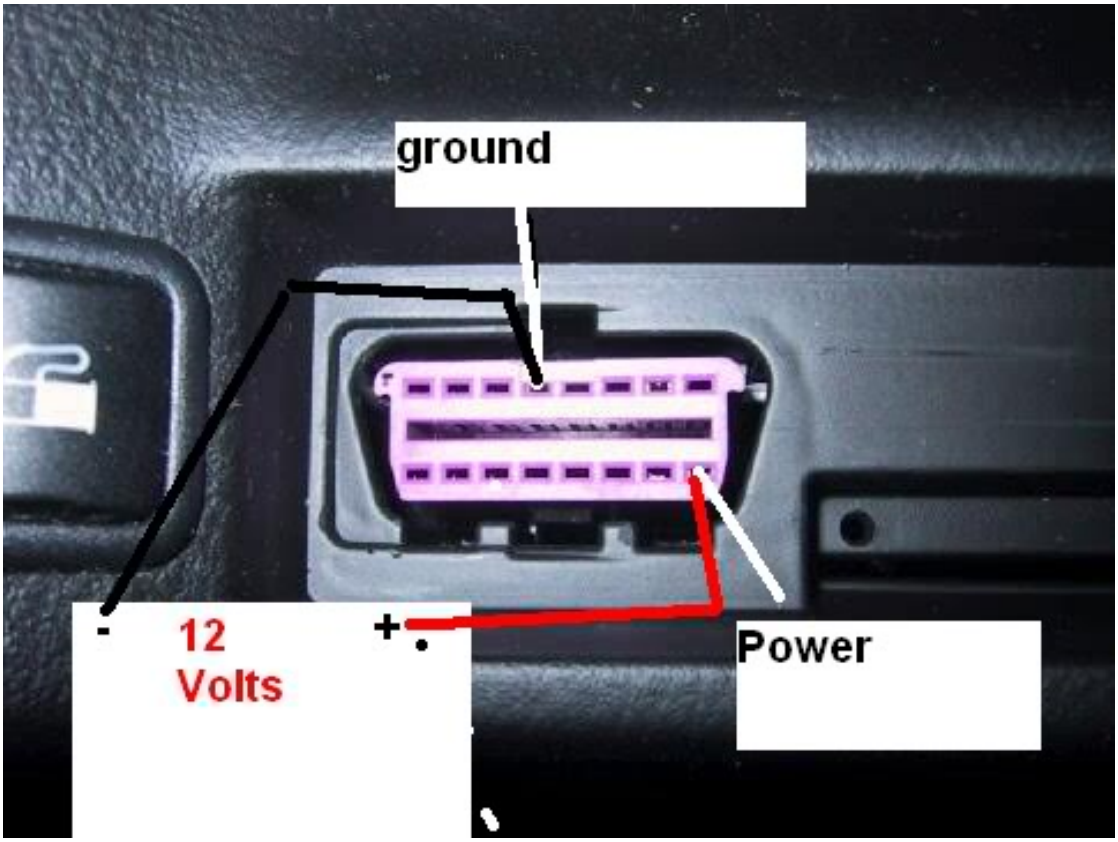


FIGURE 2 – Port ODB2

un module GPS ou un accéléromètre pour détecter les changements brutaux de trajectoire.

### 2.2.3 Métriques de conduites retenues

Nous avons sélectionné 3 métriques représentant la conduite d'un utilisateur :

**La vitesse** De manière générale, plus un véhicule vas vite, plus le risque d'accident augmente (Même si ceci est faux sur autoroute)

**La rotation du volant** De manière générale un grand nombre de coup de volant brusque ne sont pas signe d'une conduite souple et sécuritaire.

**Le nombre de rotation par minute du moteur** Le nombre de RPM est relativement lié à l'accélération du véhicule. Un nombre de RPM élevé signifie probablement une brusque accélération

Ces paramètres sont également analysables de manière simultanée par exemple accélérer en tournant le volant n'est pas recommandé et signe d'une conduite peu

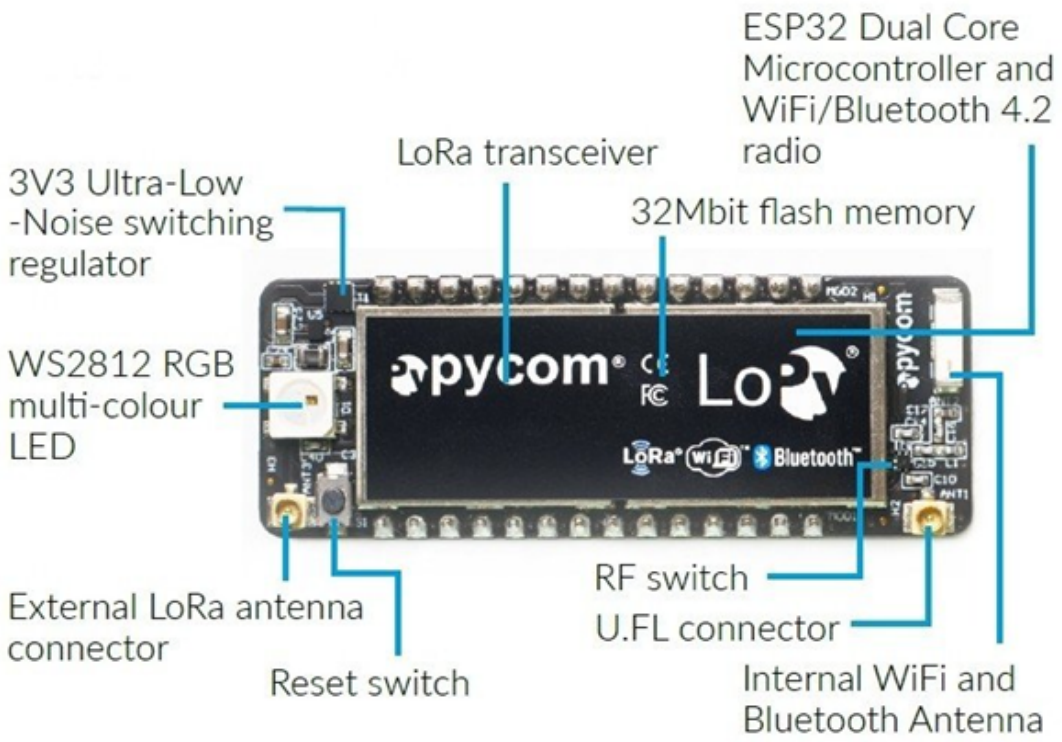


FIGURE 3 – Carte LoPy, ses connecteurs et caractéristiques principales

sûr.

Nous nous sommes limité à trois valeurs analysées en raison principalement des contraintes de temps. Il aurait été par exemple intéressant d'analyser le type de route sur lesquelles l'utilisateur conduit grâce à la position gps.

# 3 Spécifications techniques

## 3.1 Appareil ELM327 et norme ODB2

La norme ODB défini par défaut 10 modes de diagnostic utilisable au travers de plusieurs protocoles de transmission. Les véhicules plus récents (à partir de 2003) peuvent utiliser le protocole CAN plus rapide. Les principaux mode sont :

- Mode 1 : permet des lires les valeurs des sondes et capteurs du moteur (vitesse, température etc.)
- Mode 3 et 7 : permettent de lire les défauts moteur
- D'autres modes existe pour par exemple effacer des codes d'erreurs.

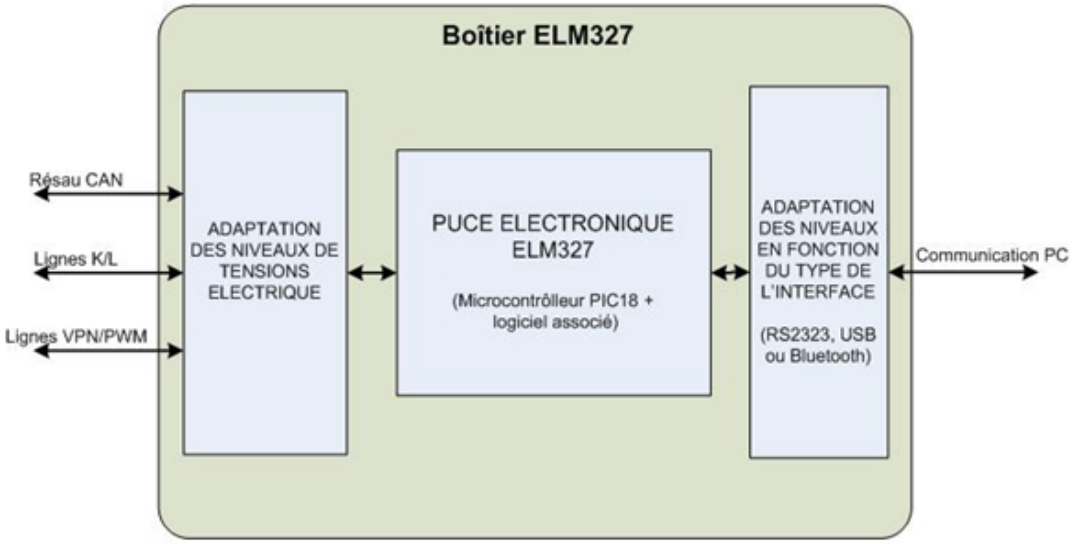


FIGURE 4 – Représentation schématique d'un boîtier ELM327

| Protocole de transmission | Norme ISO | Norme SAE | Vitesse de transmission |
|---|---|---|---|
| **KWP** (Key Word Protocol) | ISO 9141-2 | SAE J1979 | 10,4 kBit/s |
| **KWP 2000** (Key Word Protocol) | ISO 14230-4 | SAE J1979 | 10,4 kBit/s |
| **VPW** (Signaux à largeur d'impulsion variable) | - | SAEJ1850 | 10,4 kBit/s |
| **PWM** (Impulsions modulées en largeur) | - | SAEJ1850 | 41,6 kBit/s |
| **CAN** (Controller Area Network) | ISO 15765-4 | - | 500 kBit/s |

FIGURE 5 – Spécification des protocoles de la norme ODB2

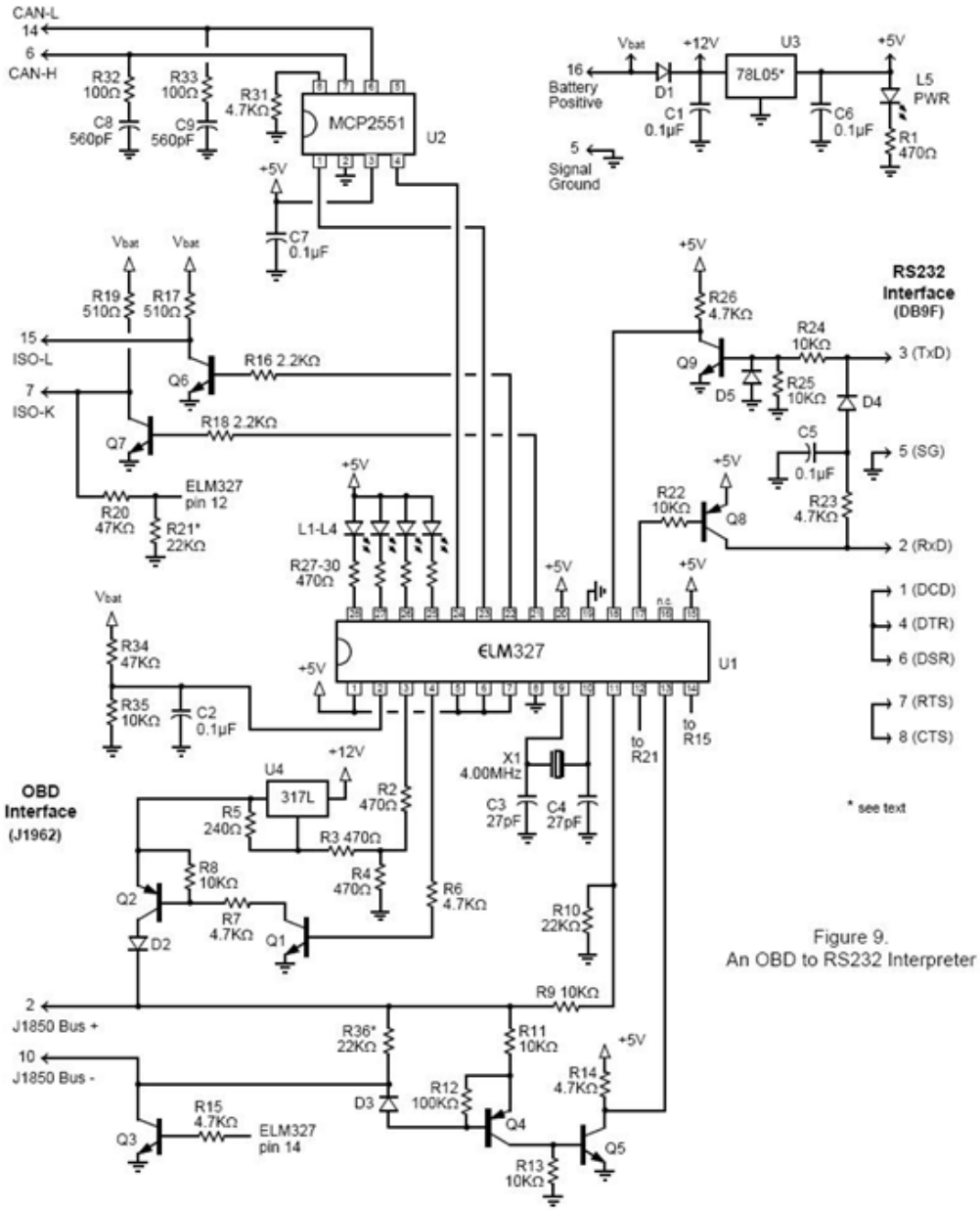


FIGURE 6 – Schéma de l'architecture physique d'un module ELM327

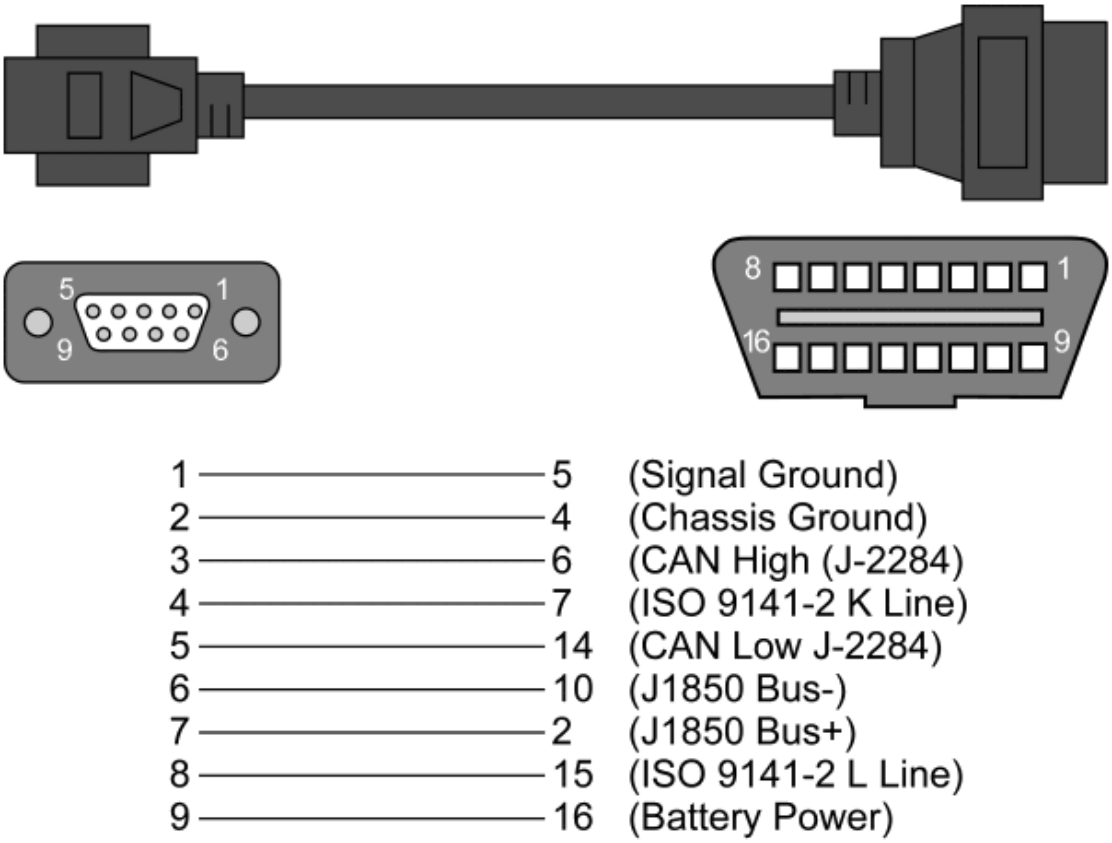


FIGURE 7 – Spécification d'un port ODB2

## 3.2 Carte Pycom LoPy et Pytrack

Pycom est une société hollandaise fabriquant des cartes de prototypage. La carte LoPy est particulièrement innovante dans le fait qu'elle intègre dans une seule carte 3 radios de communication (Wifi, Bluetooth et LoRA).

Les spécifications partielles de la carte LoPy sont :
- 512KB de RAM
- 4MB de Flash
- Accélération matériel pour les calculs en virgule flottante
- Support de MicroPython
- Multi-threading (sous MicroPython)
- WiFi 802.1b/g/n 16mbps
- Bluetooth Low Energy et classique (mais aucune bibliothèque n'est disponible pour la version classique)
- LoRA – Semtech LoRA transceiver SX1272
- Dual processor + WiFi radio System on chip
- 24 broches GPIO
- Power – Input : 3.3V – 5.5V

Les bibliothèques fournies pour MicroPython sont encore en cours de développement, Certaines fonctionnalités ne sont pas encore disponible pour la programmation d'applications utilisateur. Toutefois, ces fonctionnalités devraient être implémentées prochainement.

MicroPython est un langage dérivé de Python 3.5 conçu pour fonctionner sur des micro-controlleur. Il permet théoriquement de développer des applications plus rapidement et plus simplement en comparaison au langage C couramment utilisé dans ce domaine.

# 4 Projet original (LoPy et ELM327)

## 4.1 Première réalisation

Nous avons commencé par vérifier le fonctionnement de l'appareil ELM327 à l'aide d'une application Android. Ceci nous a notamment permis de nous familiariser avec la norme ODB2 et de choisir diverses métriques à retenir dans le cadre de notre projet.

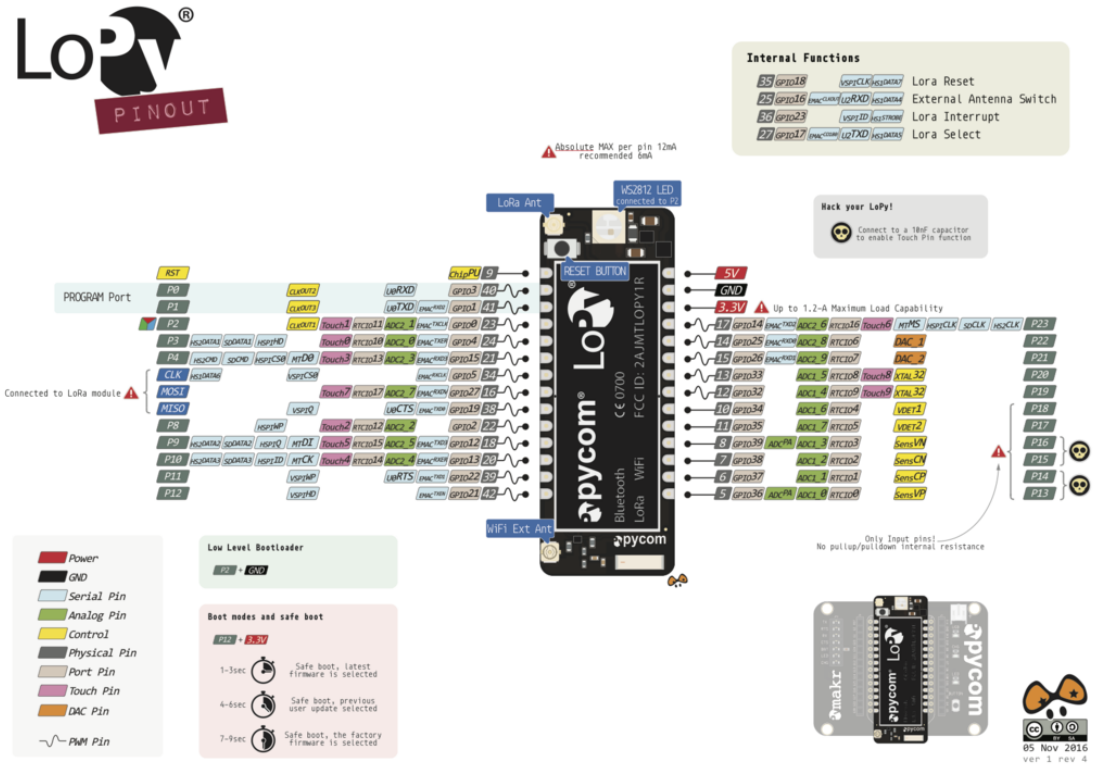


FIGURE 8 – Schéma de la carte LoPy et spécifications détaillées

Ayant reçu la carte LoPy neuve, nous avons dû mettre à jour le firmware de la carte LoPy et de PyTrack avec les firmwares récupérés sur le site de Pycom à l'aide de l'utilitaire DFUTools.

FIGURE 9 – Exemple d'interfaces Android pour visualiser des données ODB2

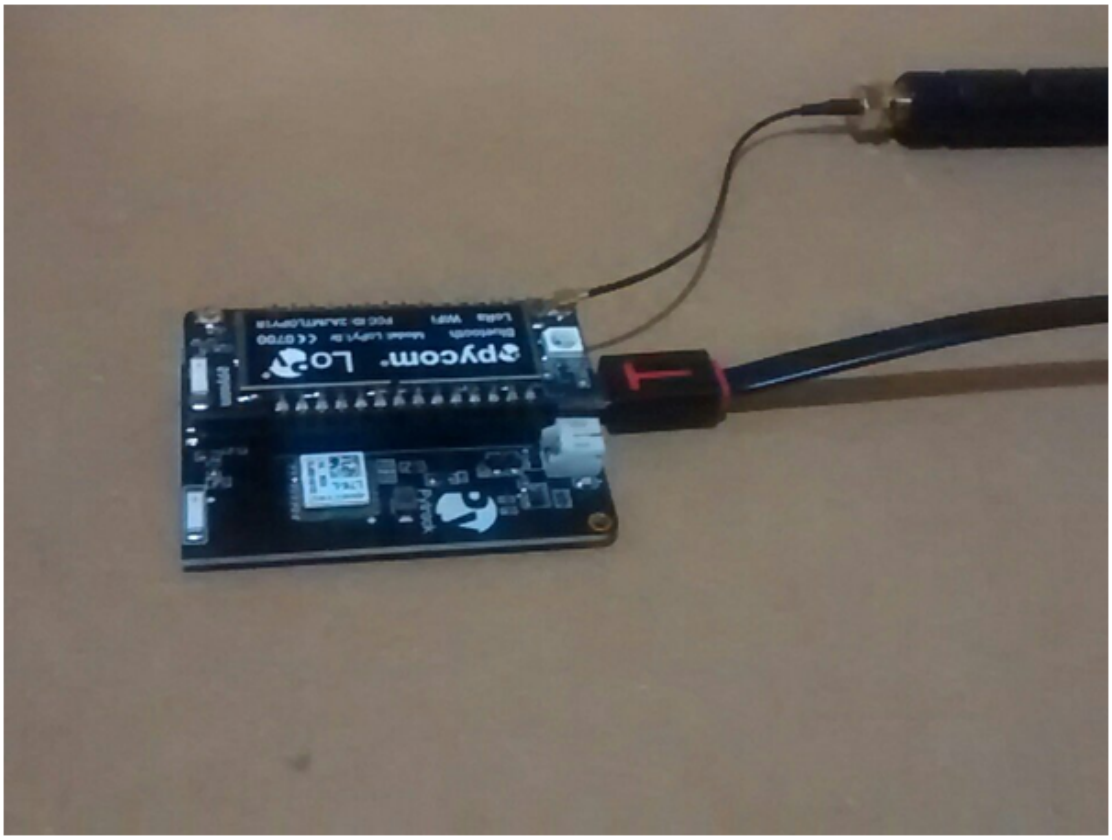


FIGURE 10 – Schéma de l'assemblage partiel d'une carte LoPy (il manque l'antenne WiFi)

FIGURE 11 – Carte LoPy sur le board Pytrack

```
sudo dfu-util -D pytrack_0.0.8.dfu

```

FIGURE 12 – La commande pour mettre à jour le firmware de Pytrack (en mode debug)

Nous avons ensuite installé et configureé le plugin “Pymakr Plugin” pour Visual Code. Le plugin nous a permis d’exporter nos programmes en python vers la carte LoPy.

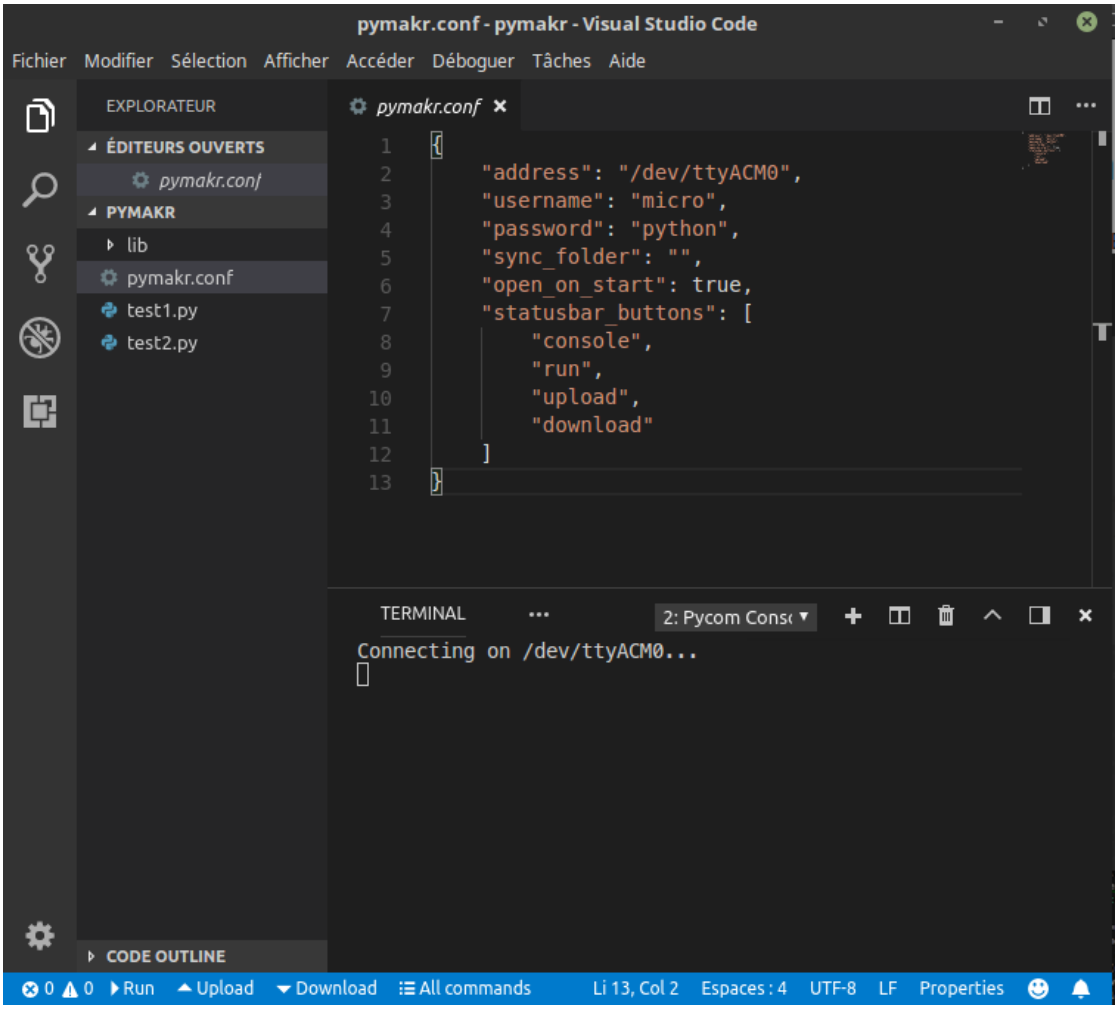


FIGURE 13 – Exemple de notre configuration du plugin PyMakr pour Visual Code

Il était nécessaire d’importer les diverse bibliothèques souhaitée dans l’espace de travail. Les bibliothèques sont disponible sur le GitHub de PyCom (`https://github.com/pycom/pycom-libraries`)

```
import pycom
from network import Bluetooth
import time
import binascii
bt = Bluetooth()
bt.start_scan(10)

for x in range(1, 10000):
    adv = bt.get_adv()
    if adv:
        print (binascii.hexlify(adv.mac))
        print("————————————————————————")
    time.sleep(0.01)

```

FIGURE 14 – Exemple de code pour scanner des périphériques BLE depuis la carte LoPy et récupérer leur addresse MAC

## 4.2 Interface de visualisation Jyse

Le logiciel Jyse nous a permis de créer un dashboard représentant les différentes mesures relevées en temps réel par l'utilisation de 3 sujets MQTT.

Nous avons représenté les valeurs en temps réel des 3 métriques de conduites que nous avions sélectionné (vitesse, rotation et RPM moteur). L'historique de la vitesse par rapport au braquage du volant a également été représenté.

## 4.3 Problème rencontré

Nous nous sommes aperçu tardivement, suite à des résultats incohérents, que les normes BLE et Bluetooth classique n'était pas compatible. Le problème étant que l'appareil ELM327 fourni ne fonctionnait qu'en Bluetooth classique avec "pairing" et la carte LoPy ne supportant pour l'instant que le BLE.

Le BLE ou Bluetooth à basse consommation a été développée par Nokia en 2006 pour devenir un standard ouvert. Il fonctionne dans sur la même plage de fréquence que le Bluetooth classique (2.4 - 2.5Ghz) et est davantage orienté vers l'IOT. Le BLE est moins adapté pour le transfert de fichiers de taille importante.

La modulation de la fréquence diffère entre le Bluetooth et de BLE de même que le nombre et type de canaux de communication. La taille des messages (les

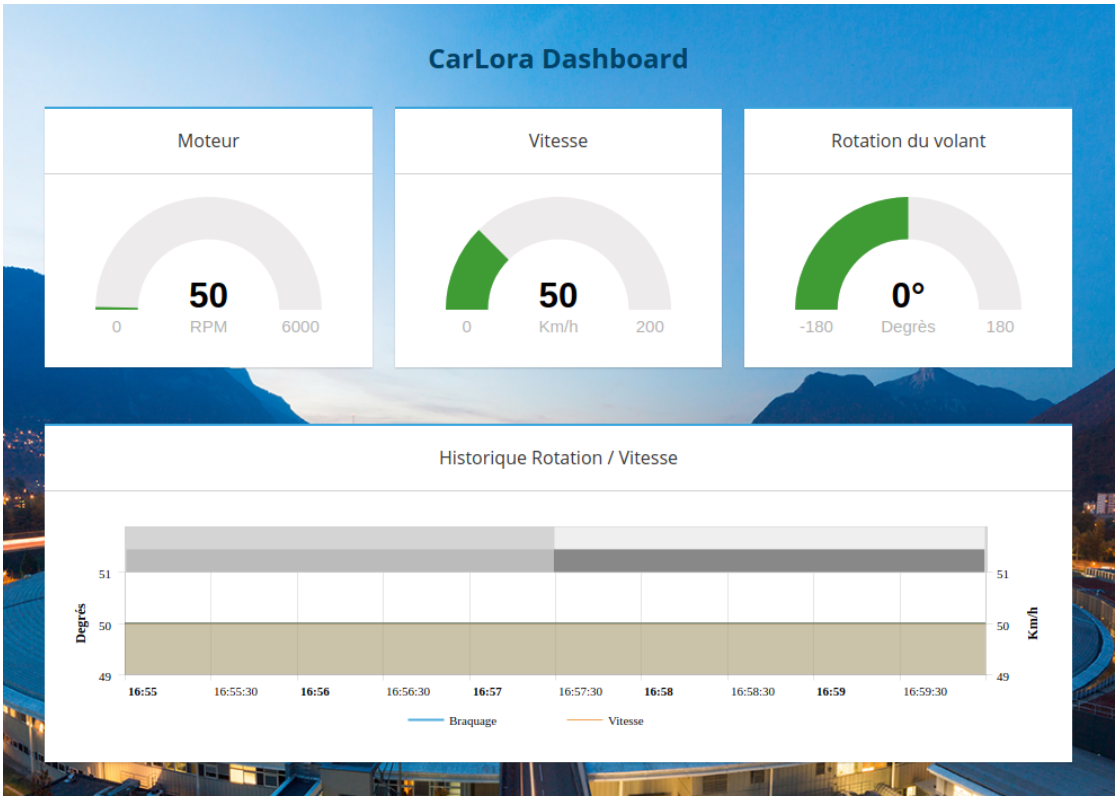


FIGURE 15 – Interface Jyse simple

messages sont plus volumineux pour le Bluetooth classique) varie également. De ce fait il n'est pas possible d'associer deux périphériques BLE et Bluetooth classique, il est nécessaire que les deux périphériques utilisent la même norme.

Nous n'avons donc pas pu poursuivre notre projet en utilisant la carte LoPy et le module ELM327 pour récupérer les informations de conduite.

# 5 Solution réalisée

Afin de continuer le projet malgré l'incompatibilité entre la carte LoPy et le module ELM327, nous avons choisi de simuler les données de conduite par le programme serveur en JAVA que nous réalisions.

## 5.1 Back-End JAVA

Notre Back-End a été réalisé en Java et dispose de 3 fonctionnalités principales : la simulations de données, le traitement des données et l'enregistrement des don-

| Specifications | Bluetooth | BLE(Bluetooth Low Energy) |
|---|---|---|
| Network/Topology | Scatternet | Star Bus |
| Power consumption | Low (less than 30 mA) | Very Low (less than 15 mA) |
| Speed | 700 Kbps | 1 Mbps |
| Range | <30 m | 50 meters( 150 meters in open field) |
| RF Frequency band | 2400 MHz | 2400 MHz |
| Frequency Channels | 79 channels from 2.400 GHz to 2.4835 GHz with 1 MHz spacing | 40 channels from 2402MHz to 2480 MHz (includes 3 advertising and 37 data channels) |
| Modulation | GFSK (modulation index 0.35) , π/4 DQPSK, 8DPSK | GFSK (modulation index 0.5) |
| Latency in data transfer between two devices | Approx. 100 ms | Approx. 3 ms |
| Spreading | FHSS (1MHz channel) | FHSS (2MHz channel) |
| Link layer | TDMA | TDMA |
| message size(bytes) | 358 (Max) | 8 to 47 |
| Error detection/correction | 8 bit CRC(header), 16 bit CRC, 2/3 FEC(payload), ACKs | 24 bit CRC, ACKs |
| Security | 64b/128b, user defined application layer | 128 bits AES, user defined application layer |
| Application throughput | 0.7 to 2.1 Mbps | less than 0.3 Mbps |
| Nodes/Active Slaves | 7 | Unlimited |

FIGURE 16 – Comparaison Bluetooth BLE vs Bluetooth classique

nées sur une blockchain Ethereum.

### 5.1.1 Simulation de données de conduites

Nous avons conservé le choix initial de travailler sur 3 données de conduites : La vitesse, le nombre de RPM du moteur ainsi que la rotation du volant. Nous avons choisi arbitrairement que la vitesse serait comprise entre 0 et 200 KM/h, le nombre de RPM entre 0 et 6000 et la rotation du volant serait comprise entre -180 et 180°.

La simulation garde en mémoire la valeur précédente et la fait varier de manière modéré dans une direction en respectant les valeurs minimum et maximum. La simulation n'est bien sûr pas conforme à un test en situation réel.

Un thread génère des valeurs à intervalle régulier et les publies sur le sujet de type /carlora/lopy/<var> sur le broker mqtt spécifié.

### 5.1.2 Récupération de données de conduites

```
<dependency>
      <groupId>org.eclipse.paho</groupId>
      <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
     <version>1.2.0</version>
</dependency>

```

FIGURE 17 – On utilise le client MQTT "eclipse paho"

Ce client MQTT nous permet de créer un objet de souscriptions aux sujets MQTT /carlora/lopy/+. Nous récupérons ainsi les valeurs des variables produites par le simulateur via le broker MQTT. nous les stockons dans un objet que nous stockerons dans la blockchain une fois complet (c'est-à-dire lorsque nous disposons des trois métriques de conduite considérées).

Pour le développement nous avons utilisé Mosquitto comme broker MQTT local.

## 5.2 Blockchain Ethereum

Nous avons utilisé la Blockchain Ethereum pour stocker les données de conduites recueillies.

### 5.2.1 Généralités

La blockchain Ethereum est un protocole d'échange décentralisé écrit en Go, s'exécutant sur plusieurs node ou pair. Les utilisateurs de ce réseau peuvent créer des smart contract (par exemple en solidity) pour stocker des contrats et des données. Ces smart contract sont par la suite consultable publiquement. Même si l'Ethereum est devenue une crypto-monnaie, à la base la blockchain Ethereum n'avait pas cette vocation.

Une interface en ligne de commande pour gérer un noeud du réseau Ethereum est disponible et se nomme Geth.

### 5.2.2 Private Network

Nous avons créé deux Node Ethereum grâce à un container Docker. Les instructions pour les lancer se trouvent dans le fichier Readme du repository GitHub `https://github.com/paul-carretero/carlora/tree/master/eth`.

On commence par lancer les deux conteneurs Docker contenant Ethereum et Geth, l'utilitaire de gestion en ligne de commande (CLI) d'Ethereum.

L'étape importante est de générer un utilisateur à l'aide du script fourni et d'associer les deux node grâce à leur adresse et leur IP dans le réseau Docker.

```
carretero@Mint-G751JY  ~/git/carlora/eth  master ●  sudo docker network inspect ETH
[sudo] Mot de passe de carretero :
[
    {
        "Name": "ETH",
        "Id": "3224db64916d5e21818cd4cf6e5a8c370a25e19f6781880bedfad59e91a81ead",
        "Created": "2018-04-07T23:14:05.128887704+02:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.21.0.0/16",
                    "Gateway": "172.21.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "Containers": {
            "5f1855faec7783774c10f9803184d77d1ebb43ba0ca95815b877543c75d16e63": {
                "Name": "elastic_galileo",
                "EndpointID": "d67c3b39957140f42e9325ff607e070a13c8129b749ef3b71aa3e37d7c120d28",
                "MacAddress": "02:42:ac:15:00:02",
                "IPv4Address": "172.21.0.2/16",
                "IPv6Address": ""
            },
            "e97c81692fc18d073a39c62c97075e73485e26a8fe7d72067eeab06d4528bc71": {
                "Name": "stupefied_banach",
                "EndpointID": "5529e81d2269ce5242dc7b08db92959778b573a1ac1f81dba29c6646d54a2bc2",
                "MacAddress": "02:42:ac:15:00:03",
                "IPv4Address": "172.21.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

FIGURE 18 – Réseau Docker contenant les deux images des Node Ethereum

```
carretero@Mint-G751JY  ~/git/carlora/eth  master ●  sudo docker run --rm -it -p 8545:8545 --net=ETH node_one
eth_user@5f1855faec77:~$ geth --identity="NODE_ONE" --networkid="500" --verbosity=1 --mine --minerthreads=1 --rpc --rpcaddr 0.0.0.0 console
WARN [04-08|10:21:29] Cannot start mining without etherbase    err="etherbase must be explicitly specified"
Fatal: Failed to start mining: etherbase missing: etherbase must be explicitly specified
eth_user@5f1855faec77:~$ ./eth_common/setup_account
INFO [04-08|10:21:47] Maximum peer count                       ETH=25 LES=0 total=25
Address: {d7fe33809568b26a36a9bc14b9c4e9334f2ae3fd}
eth_user@5f1855faec77:~$ geth --identity="NODE_ONE" --networkid="500" --verbosity=1 --mine --minerthreads=1 --rpc --rpcaddr 0.0.0.0 console
Welcome to the Geth JavaScript console!

instance: Geth/NODE_ONE/v1.8.2-stable-b8b9f7f4/linux-amd64/go1.9.4
coinbase: 0xd7fe33809568b26a36a9bc14b9c4e9334f2ae3fd
at block: 0 (Thu, 01 Jan 1970 00:00:00 UTC)
 datadir: /home/eth_user/.ethereum
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0
```

FIGURE 19 – Lancement du premier Node

```
> enode = "enode://31ea47f5b348fdc9f3e80a586383890d56c58a058216244fbf81c2ab6de5ee4c703522355650c054d0f2c2d38d68b72e70b8cf13b9aed9f8ce9b21269c44205e@172.21.0.2:30303"
"enode://31ea47f5b348fdc9f3e80a586383890d56c58a058216244fbf81c2ab6de5ee4c703522355650c054d0f2c2d38d68b72e70b8cf13b9aed9f8ce9b21269c44205e@172.21.0.2:30303"
> admin.addPeer(enode)
true
> admin.peers
[{
    caps: ["eth/63"],
    id: "31ea47f5b348fdc9f3e80a586383890d56c58a058216244fbf81c2ab6de5ee4c703522355650c054d0f2c2d38d68b72e70b8cf13b9aed9f8ce9b21269c44205e",
    name: "Geth/NODE_ONE/v1.8.2-stable-b8b9f7f4/linux-amd64/go1.9.4",
    network: {
      inbound: true,
      localAddress: "172.21.0.3:30303",
      remoteAddress: "172.21.0.2:41876",
      static: false,
      trusted: false
    },
    protocols: {
      eth: {
        difficulty: 44065350,
        head: "0x29b37ef7d326768b8b10978bae4a0adc1d63da99469d031b4defcb384dcb80f0",
        version: 63
      }
    }
}]
```

FIGURE 20 – Pairing du node 2 avec le node 1

### 5.2.3 Smart Contract

Le smart contract Solidity est très simple et permet de stocker les 3 informations de conduites ainsi que le timestamp associé.

Une fois le Smart Contract créé, nous avons pu le compiler à l'aide de solc et crééer sont wrapper Java grâce à l'utilitaire fourni par Web3J.

```
./web3j solidity generate ../../Record.bin ../../Record.abi -o ../../
    -p com.carlora

```

FIGURE 21 – Exemple de commande pour générer le wrapper Java d'un Smart Contract

### 5.2.4 Stockage des données : Bibliothèque Web3J

Une fois le wrapper du smart contract et le réseau ethereum privé déployé, nous avons pu associer notre back-end à la blockchain Ethereum privé de ce réseau. Pour ce faire, nous avons utilisé la bibliothèque Web3J.

```
pragma solidity ^0.4.0;
contract Record {

    uint timestamp;
    uint rpm;
    uint rot;
    uint spd;

    function Record(uint _timestamp, uint _rpm, uint _rot, uint _spd) public {
        timestamp = _timestamp;
        rpm = _rpm;
        rot = _rot;
        spd = _spd;
    }

    function getTimestamp() public constant returns (uint _timestamp) {
        _timestamp = timestamp;
    }

    function getRpm() public constant returns (uint _rpm) {
        _rpm = rpm;
    }

    function getRot() public constant returns (uint _rot) {
        _rot = rot;
    }

    function getSpd() public constant returns (uint _spd) {
        _spd = spd;
    }
}

```

FIGURE 22 – Smart Contract utilisé dans le projet poru stocker les données de navigation

Une file d'attente a été utilisée pour gérer les opérations parfois longues d'écriture sur le premier node de la blockchain Ethereum

De cette manière à chaque fois que les informations de conduites sont complétées, on créé un smart contract et on le publie dans la blockchain.

```
<dependency>
      <groupId>org.web3j</groupId>
      <artifactId>core</artifactId>
      <version>3.3.1</version>
</dependency>

```

FIGURE 23 – Ajout des dépendences vers Web3J

```
Record.deploy
     (this.web3, this.credentials, GAS_PRICE, GAS_LIMIT, ts, rpm, rot,
     spd)
     .send();

```

FIGURE 24 – Exemple d'instruction Java pour créer et enregistrer un smart contract

**Limites** Nous avons constater de grandes latences et une occupation CPU importante lorsque les deux node Ethereum étaient lancé (ce qui est cohérent avec le fait que nous les avions lancés avec l'argument -mine).

# 6 Conclusion

Ce projet nous a permis de découvrir certaines fonctionnalités de l'IOT. Nous avons notamment découvert les technologies de programmation et de communication des périphériques embarqués basses consommation. LoPy est une plateforme de développement récente et très riche en possibilité (de communication notamment) même si certaines ne sont pas encore totalement réalisées et documentées par la société éditrice.

Le logiciel Jyse est très puissant pour obtenir rapidement une visualisation des différentes données retournées par le système. Nous avons également pu découvrir les concepts de base de la blockchain Ethereum.

Toutefois, le problème d'incompatibilité entre le bluetooth classique et le BLE nous a empêché de poursuivre le projet dans les conditions prévues au début. Le réseau privé Ethereum est également très simple et aurait gagné à disposer d'outil de monitoring par exemple.

# 7 Références


— https://www.elm327.fr/norme-obd/modes-obd/
— https://www.outilsobdfacile.fr/interface-diagnostic-elm-327.php
— https://pycom.io/wp-content/uploads/2018/03/Pycom_Specsheet_LoPy.pdf
— https://docs.pycom.io/
— http://www.rfwireless-world.com/Terminology/Bluetooth-vs-BLE.html
— https://gist.github.com/fishbullet/04fcc4f7af90ee9fa6f9de0b0aa325ab
— https://docs.web3j.io/command_line.html
— https://dzone.com/articles/an-introduction-to-ethereum-and-smart-contracts-a
— https://github.com/JacqMT/web3j-maven-example/wiki/Smart-Contracts-in-Java
— http://solidity.readthedocs.io/en/v0.4.21/