ANNEE

2017

2018

INRIA Lille – Nord Europe
Equipe DEFROST
Parc scientifique de la Haute Borne
40, avenue Halley – A-206
59655 Villeneuve-d'Ascq CEDEX
FRANCE
Tél : +33 3 59 57 78 00

Nicky UNG & Hugo DELATTE
Département IMA – 5ème année

Polytech Lille
Secrétariat IMA – Bureau A107
Avenue Paul Langevin – Cité Scientifique
59655 Villeneuve-d'Ascq CEDEX
FRANCE
Tél: +33 3 28 76 73 40
Fax: +33 3 28 76 73 41

Tuteur école: M. Jérémie DEQUIDT
Tuteurs INRIA: M. Gang ZHENG &
M. Roudy DAGHER

INVENTEURS DU MONDE NUMÉRIQUE

P18 - Indoor localisation of quadrotors

[MEMOIRE DE PROJET DE FIN D'ETUDES]

Ce rapport présente le projet de fin d'études de Nicky UNG et Hugo DELATTE effectué entre Septembre 2017 et Février 2018 au sein du laboratoire de recherche INRIA Lille – Nord Europe.

Table des matières

1. Présentation générale du projet

Dans cette première partie, nous présenterons le contexte autour duquel s'articule le projet qui nous a été confié ainsi que la phase de réflexion qui intègre la planification des tâches et les choix technologiques.

CONTEXTE

Dans un monde de plus en plus connecté, de nombreux drones sont déployés pour diverses applications comme l'espionnage, la surveillance, la cartographie de lieux et la livraison de colis. Les technologies nécessaires pour contrôler ces drones sont des sujets de recherche actuels.

Le contrôle de ceux-ci passe tout d'abord par la maîtrise de leur position dans l'espace. Le développement des applications liées à l'utilisation de drones nécessite un travail important sur la localisation spatiale d'objets connectés.

De plus, dans un souci d'utilisation, il est essentiel de développer des interfaces logicielles permettant d'exploiter ces données.

OBJECTIF

L'objectif de ce projet est de développer une interface logicielle permettant de traiter en temps réel les informations de localisation d'un drone et de balises, dans le but de définir leur localisation dans l'espace, en intérieur.

DESCRIPTION

Pour réaliser le contrôle de drones, la première étape importante est de donner ses informations de position. Le but du projet est de développer une interface logicielle permettant de localiser le drone en temps réel en intérieur. Le drone se déplace dans un environnement dans lequel ont été installées de multiples balises. Il est équipé d'une carte, lui permettant de recevoir les distances relatives entre lui-même et les balises, sur laquelle différents algorithmes vont être implémentés pour effectuer la localisation en temps réel.

Le projet requiert des compétences en développement hardware (Crazyflie 2.0, voir https://wiki.bitcraze.io/) et software (Python/Javascript/HTML...).

La mise en place d'un tel dispositif permettrait grâce à l'utilisation d'un algorithme d'estimer avec une bonne précision les positions en intérieur d'une balise centrale (appelée Tag) et de balises (appelées Anchor). Le Tag scanne en temps réel la distance relative aux différentes balises, et il nous est retourné la position des balises et du Tag grâce à l'algorithme de localisation.

Sujet originel

To achieve the control of quadrotors, the first important task is to give its position information. This project is to develop a software(GUI) to localise the quadrotors in real-time. The quadrotor is flying in an environment where several transmitters have been installed. The quadrotor is equipped with a deck to receive the relative distances between the deck and all transmitters, based on which different algorithms will be implemented to realise the real-time localisation. The candidates need to have both experiences on hardware (Crazyflie 2.0, see https://wiki.bitcraze.io/) and software development (C/C, Python…).

CAHIER DES CHARGES

Le projet se compose de 3 parties:

- Collect & parse: collecter les données envoyées par les balises en communiquant avec un serveur et récupérer les données adéquates par un tri
- Localise: déterminer la matrice de position des balises et du Tag en traitant les données récupérées
- Display: développer l'interface homme machine pour afficher, étudier et utiliser les données traitées

Nous travaillerons dans un premier temps dans une optique temps différé avant de travailler en temps réel. Cela nous permettra d'acquérir des données utiles et réutilisables en aval.

A l'aboutissement du projet, nous devrions être capable via l'interface homme machine développée d'afficher précisément la position spatiale des balises ainsi que du Tag,

grâce à l'utilisation d'un algorithme de localisation et des données récupérées, et d'afficher d'autres données relatives au déplacement du Tag ou des balises dans l'espace.

Si le temps le permet, nous pourrons aussi être amenés à adapter et utiliser l'interface développée pour l'utilisation d'un drone.

CHOIX TECHNIQUES : MATERIEL ET LOGICIEL

Afin de développer ce dispositif, nous allons utiliser une Raspberry Pi 3 associée à une carte réceptrice, qui communique avec 4 balises composées chacune d'un microcontrôleur et d'une carte émettrice radio (UWB5C) et alimentées par batterie. La carte réceptrice joue le rôle du drone à localiser et communiquera les données émises par les balises vers l'ordinateur via Bluetooth.

CHOIX TECHNOLOGIQUES

Voici les technologies que nous utilisons pour chacune des parties de ce projet :

- Serveur de collecte des données : Scanne et parse les données récupérées pour fournir la matrice D des distances relatives du tag aux 4 balises
    - API
    - Python / Serveur Flask
- Serveur de localisation (fourni) : Calcule grâce à l'algorithme de localisation et la matrice D les positions spatiales des 4 nœuds et du tag
    - API
    - Javascript (JQuery)
    - Python
- Interface utilisateur : Affiche courbes, visualisation 3D, boutons ou données des fichiers envoyées par le serveur de localisation
    - HTML / Javascript
    - Fichiers JSON (serveurs de collecte et de localisation)

LISTE DES TACHES A EFFECTUER

Il se divise en 3 étapes principales, qui sont:

- Choisir les technologies utilisées ainsi que l'architecture (OS / APIs)
  - Se documenter sur le projet
  - Langages et technologies utilisées
  - Matériel utilisé
- Développer les différentes parties du projet
  - Mettre en place du travail sur le Git
  - Réaliser le serveur de collecte
  - Récupérer les trames envoyées par Bluetooth par le TAG via la Raspberry Pi
  - Intégrer le serveur de localisation pour interpréter les trames
  - Tester et optimiser
  - Réaliser l'interface Web utilisateur
- Intégrer et connecter les différentes parties du projet
  - Implémenter le serveur de collecte et de localisation
  - Afficher et traiter les données stockées dans les fichiers JSON via le serveur (temps différé + temps réel)
  - Tester et optimiser

2. Présentation du travail effectué

Cette seconde partie présente le déroulement de notre projet, c'est-à-dire toute la phase de recherche et développement mise en œuvre pour réaliser nos tâches. Nous présenterons la construction de notre projet en exposant nos phases de réflexion avant les phases de développement, ainsi qu'en expliquant comment nous avons implémenté toutes les fonctionnalités et comment fonctionne notre application.

STRUCTURE DE L'APPLICATION

Comme expliqué précédemment, le but de ce projet est de mettre en place une application qui permet d'un côté :

- Côté serveur, de récupérer les données de distance des balises collectées par la Raspberry Pi 3, et de les envoyer au serveur pour être traitées et parsées : ainsi, ces données sous format json peuvent être envoyées au serveur de localisation qui se charge de nous renvoyer des matrices X et A de données spatiales (x,y,z) pour localiser dans l'espace les balises, puis de les envoyer au client
- Côté client, de récupérer les données envoyées par le serveur Flask codé en Python, et de les afficher sur un site Web, de façon à ce que l'utilisateur puisse visualiser des données via une interface graphique

Lorsque l'utilisateur veut effectuer des actions via les boutons du site, il effectue des requêtes Ajax (via le Framework JQuery) qui vont être envoyées au serveur Flask. Ce dernier va lancer des commandes Python correspondants à chaque Web Service appelé par les requêtes Ajax.

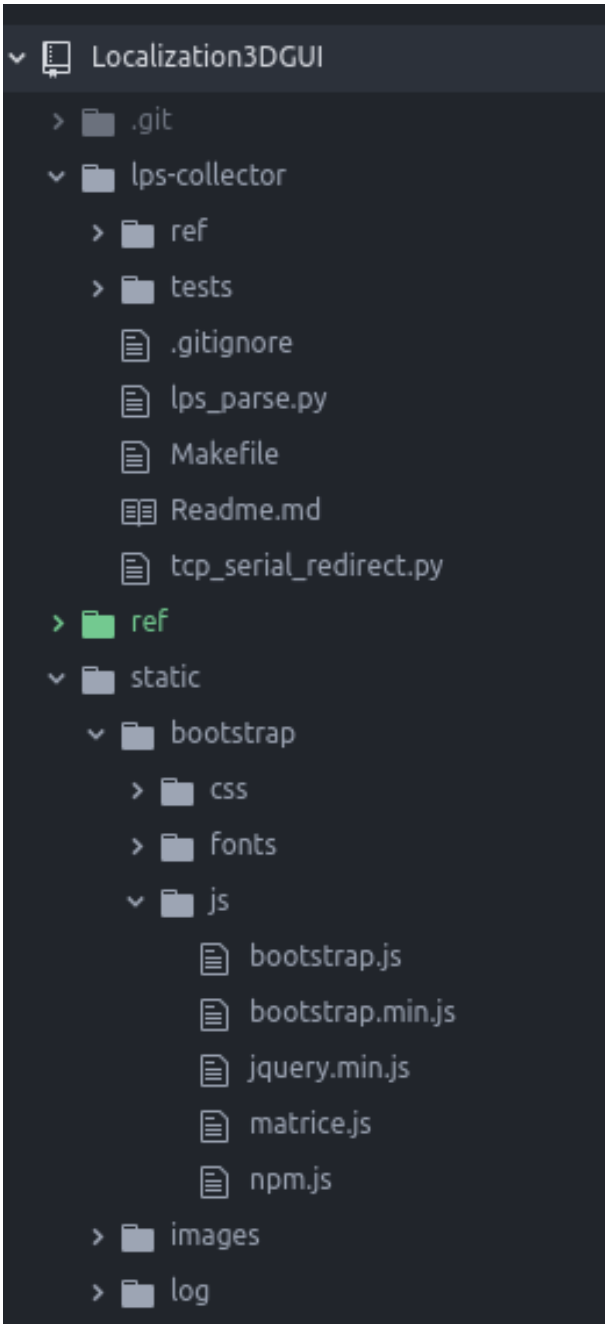
La collecte des données de distance est lancée manuellement par l'utilisateur via le site, et il peut choisir de lancer une collecte de durée définie, ou manuellement (et donc l'arrêter lui-même). Ces données sont stockées dans des fichiers .log et parsées par un programme *lps_parse.py* qui se charge de formater sous format .json et .csv les données de distance. Après avoir envoyé ce fichier json au serveur de localisation, il nous est retourné les coordonnées spatiales de chaque balise et du tag, et nous pouvons donc

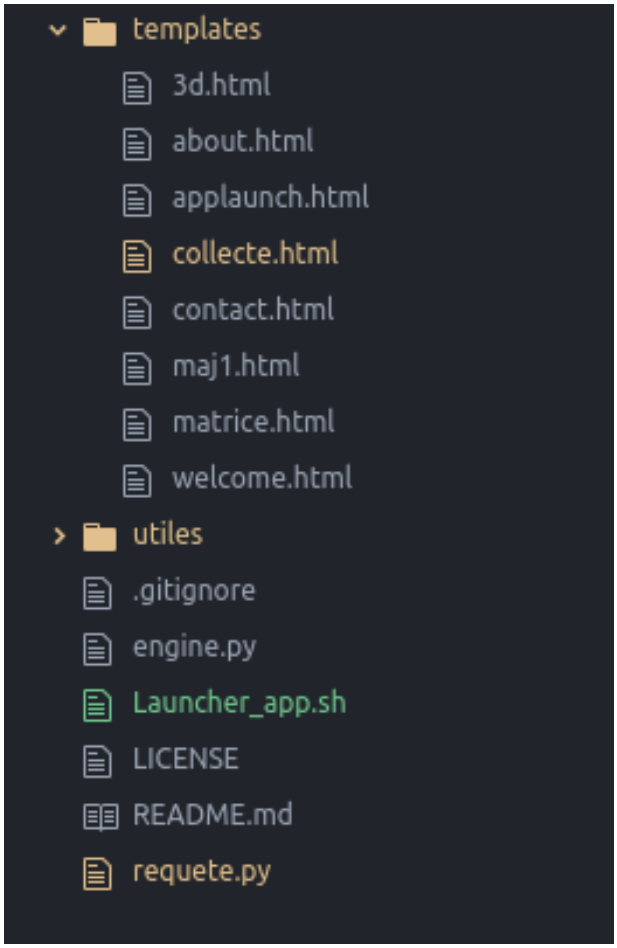
afficher ces données dans le site pour que l'utilisateur puisse visualiser graphiquement des courbes de distance, une visualisation 3D des positions des balises et du tag…

STRUCTURE DU DEPOT DU PROJET

Pour une meilleure compréhension de notre travail, il faut savoir comment se décline notre dépôt de projet:

- les scripts engine.py et requete.py qui regroupent les fonctions de l'application côté serveur
- un dossier /templates contenant les pages html du site (dont la plus importante, collecte.html)
- un dossier /static contenant les données de collecte dans /static/log au format log, json et csv, ainsi qu'un dossier /static/bootstrap avec les fonctions javascript (dont matrice.js) et les bibliothèques bootstrap
- un dossier /lps-collector contenant le script de redirection entre port TCP et port série tcp_serial_redirect.py ainsi que le script de parsage des données en format json et csv lps_parse.py





BACK-END : SERVEUR

Le serveur Python Flask se partage en deux parties:

- Le script requete.py représente le cœur du serveur et contient chaque fonction à réaliser et chaque page à afficher en fonction des URL entrées dans le navigateur. Il contient de plus nos services web nous permettant de réaliser certaines actions lors de requêtes AJAX.

- Le module engine.py regroupe l'ensemble des différentes fonctions Python appelées dans requete.py. Nous avons évidemment choisi cette conception modulaire pour ne pas surcharger le cœur du serveur de définition de fonctions.

Tout d'abord, nous détaillerons le contenu du fichier engine.py pour donner une vue d'ensemble des fonctions utilisées. Ensuite, nous détaillerons l'utilisation de ces différentes fonctions dans requete.py.

Le script engine.py

Le fichier engine.py n'est donc qu'une librairie contenant un ensemble de fonctions. Voici l'ensemble des fonctions codées à l'heure actuelle ainsi que leur définition. Il n'est pas essentiel de s'attarder sur cette partie, mais il sera utile d'y revenir régulièrement lors de l'explication de la seconde partie, pour se référer à l'action effectuée par certains appels de fonctions:

```
def connexion():
    print("CONNEXION")
    subprocess.Popen(['rfcomm','connect','/dev/rfcomm0','B8:27:EB:80:DC:89','1'])
    time.sleep(10)
    return 0
```

La fonction connexion() permet de créer un port série nommé "rfcomm0" qui sera relié via l'interface Bluetooth à l'adresse MAC de notre Raspberry Pi.

Nous lançons donc une commande shell dans un processus fils, et donc en tâche de fond grâce à la commande python "subprocess.Popen".

```
def redirection():
    print("REDIRECTION")
    subprocess.call("python lps-collector/tcp_serial_redirect.py /dev/rfcomm0 9600 &",shell=True)
    time.sleep(7)
    return 0
```

La fonction redirection() appelle le script python tcp_serial_redirect.py. Ce script nous a été donné par notre tuteur pour rediriger les données d'une connexion TCP/IP vers un port série et vice-versa. Cela nous permet de lancer des commandes :

```
def lecture():
    print("LANCEMENT CU")
    subprocess.Popen('echo "cu -s9600 -lttyACM0"| nc localhost 7777',shell=True)
    time.sleep(1)
    return 0
```

La fonction lecture() permet de lancer une nouvelle commande shell "call up" permettant l'affichage des informations sortant du port TTYACM0 de la Raspberry Pi. Grâce à un pipe, nous redirigeons cette sortie via le port 7777 de localhost.

Ces trois fonctions font partie de l'initialisation de la connexion. Elles permettent de préparer l'ensemble du système pour la collecte des informations. Nous irons collecter ces informations en local (localhost) via le port 7777 (TCP/UDP).

```
def collecter(temps):
    print("LECTURE CONNEXION")
    global pro
    global date
        date=time.strftime('%d%B%Y_%H-%M-%S')
    if temps == '0' :
        pro=subprocess.Popen('nc localhost 7777 | grep distance > static/log/dist-capture'+date+'.log',shell=True, preexec_fn=os.setsid)
        while True :
            time.sleep(100)
    else :
        pro=subprocess.Popen('nc localhost 7777 | grep distance > static/log/dist-capture'+date+'.log',shell=True, preexec_fn=os.setsid)
        float_time=float(temps)
        time.sleep(float_time)
        print("Coupure imminente")
        os.killpg(os.getpgid(pro.pid), signal.SIGTERM)
    return date
```

La fonction collecter(temps) permet de récupérer les informations via la commande "nc localhost 7777" pour les rediriger dans un fichier.log qui prendra comme nom la date de début de la collecte.

On peut distinguer deux cas dans cette fonction. Si le temps entré sur l'interface graphique est de "0", on laisse tourner cette fonction de collecte indéfiniment. Toutefois, la valeur de la fonction subprocess.Popen est stockée dans une variable globale "pro". Cela nous permettra de supprimer ce sous processus à l'appel d'une autre fonction (la fonction "kill"). Si le temps est différent de 0, alors nous exécutons la même fonction mais elle sera stoppée automatiquement grâce à la commande "os.killpg" a la fin du temps indiqué par l'utilisateur.

```
def kill():
    if pro != -1:
        os.killpg(os.getpgid(pro.pid), signal.SIGTERM)
    pro = -1
    return 0
```

La fonction kill() permet lors de stopper la collecte d'éléments manuellement. Elle utilise la même commande vue précédemment.

```
def parse():
    print("Parsage du fichier")
    global date
    subprocess.Popen("python lps-collector/lps_parse.py static/log/dist-capture"+date+".log",shell=True, preexec_fn=os.setsid)
    time.sleep(1)
    #json_data=open('static/dist-capture'+date+'.json')
    date = -1
    #return json_data
    return 0
```

La fonction parse() récupère le "fichier.log" venant d'être créé par la fonction log et le transforme en deux fichiers différents. un fichier.csv ainsi qu'un fichier.json. Ce sont ces fichiers dont nous nous servons ensuite pour l'exploitation.

```
def afficher(fichier):
    json_data=open(fichier)
    return json_data
```

La fonction afficher(fichier) permet d'afficher dans un tableau HTML un fichier json créé préalablement grâce aux fonctions collecter() et parse().

Le script requete.py

Les pages Web

Nous en avons enfin terminé avec l'explication des différentes fonctions mises en place. Nous pouvons donc nous intéresser à l'utilisation de ces fonctions. Comme nous avons vu auparavant, ce sera le fichier requete.py qui fera appel à toutes ces fonctions. Nous allons d'abord vous présenter ce fichier sans les API ajoutées, que nous détaillerons ensuite.

```
@app.route('/')
def dire_coucou():
    return render_template('applaunch.html')

@app.route('/welcome')
def welcome():
    return render_template('welcome.html')

@app.route('/matrice',methods=['GET', 'POST'])
def matrice():
    if request.method == 'POST':
        affiche=request.method == ['id']
        afficher(affiche)
        return render_template('matrice.html', titre="distances")
    if request.method == 'GET':
        return render_template('matrice.html', titre="distances")

@app.route('/3d')
def troisd():
    return render_template('3d.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/contact')
def contact():
    return render_template('contact.html')

@app.route('/collecte',methods=['GET', 'POST'])
def collecte():
    if request.method == 'GET':
        log=os.listdir("static/log")

        return render_template('collecte.html', titre="collecte",fichiers=log)

if __name__ == '__main__':
    app.run(debug=True)
```

Nous pouvons facilement voir un pattern se créer tout au long de ce fichier:

@app.route("/something")

def fonction():

#commandes ou fonctions

return render_template("page.html",data)

Cela représente une page de notre application. Par exemple, lorsque nous entrons l'URL "/collecte", la page affiché sera celle contenue dans la fonction "render_template()", c'est-à-dire ici la page "collecte.html". Les data données après sont des variables que nous pourrons utiliser directement dans la page "collecte.html" ensuite. Il est donc possible, à l'appel d'une page, d'exécuter des fonctions puis d'envoyer les valeurs de retour dans une page HTML.

Par exemple, nous récupérons un tableau "log" contenant l'ensemble des noms des fichiers contenus dans "static/log", que nous passons via la variable fichiers dans la page "collecte.html".

Nous avons volontairement basé nos exemples sur le service @app.route(/collecte) car cela représente la page principale du site. Les autres sont soit des pages d'information comme la page "/about" ou alors des pages qui nous seront utiles dans la suite du projet comme la page "/3d".

Nous pouvons maintenant ajouter les web services à notre serveur. Ce sont les APIs qui seront accessibles via nos requêtes ajax qui sont définies dans la suite du rapport.

Les Web Services

Les web services sont des chemins URL accessibles dans notre application. Ils ne font pas partie des pages à afficher dans l'interface web mais peuvent être appelés et effectuer des demandes faites par l'utilisateur.

```
@app.route('/affichage/<fichier>')
def affichage(fichier):
    data= json.dumps(json.load(open('static/log/'+fichier,'r')))
    return data
```

Cette première API permet de récupérer le contenu d'un fichier json dont nous connaissons le nom afin de le renvoyer pour que l'on puisse l'utiliser (l'afficher dans un tableau html dans notre cas). Le web service "/kill" permet tout simplement d'appeler une fonction python qui arrêtera nos processus de collecte.

```
@app.route('/kill')
def kill():
    kill()
    #var=parse()
    return 'Pro killed'
```

Ces deux services web permettent respectivement de lancer les fonctions python lecture() et collecte().

```
@app.route('/commandecu')
def commandecu():
    lecture()
    return "ajax cu ok"

@app.route('/commandecol')
def commandecol():
    temps=request.args['temps']
    collecter(temps)
    var=parse()
    return "ajax col ok, temps= "+temps
```

Tous les services web sont ressemblants et fonctionnent de la même manière mais il sont surtout séparés. En ce sens ils sont appelés par des requêtes ajax contenues dans des fonctions Javascript indépendantes aussi. C'est le principe d'un projet fait de la façon la plus modulaire possible. Il nous est très facile de travailler sur un feature en particulier sans empêcher le reste du projet de fonctionner le plus stablement possible.

FRONT-END : CLIENT

La page collecte.html

La page collecte.html est responsable de l'affichage de l'ensemble des ressources traitées jusqu'à présent. Cette page HTML étant assez consistante nous ne l'afficherons pas entier mais nous détaillerons son code au fur et à mesure.

Elle est tout d'abord composée d'une en-tête définie par la balise <head></head>.

```
<head>
    <meta charset="utf-8" />
    <link href="{{url_for('static', filename='bootstrap/css/bootstrap.min.css')}}" rel="stylesheet" type="text/css">
    <link href="{{url_for('static', filename='bootstrap/css/table.css')}}" rel="stylesheet" type="text/css">
    <link href="{{url_for('static', filename='bootstrap/css/starter-template.css')}}" rel="stylesheet">
    <script src="{{url_for('static', filename='bootstrap/js/jquery.min.js')}}"></script>
    <script src="{{url_for('static', filename='bootstrap/js/bootstrap.min.js')}}"></script>
    <script src="{{url_for('static', filename='bootstrap/js/matrice.js')}}"></script>

    <title>{{titre}}</title>
</head>
```

Nous pouvons ici retrouver les différentes commandes permettant de faire références à nos différentes feuilles de style ainsi que l'implémentation des frameworks "jquery" et "bootstrap".

La dernière ligne <script></script> fait quant à elle référence à un fichier javascript contenant toutes les fonctions javascript appelées dans notre application pour dynamiser la page. C'est dans ce fichier matrice.js que nous retrouverons les différents appels AJAX vers nos web services. Nous détaillerons la page matrice.js au chapitre suivant.

Nous voyons tout à la fin la balise title qui contient la variable "titre". Cette variable est envoyée par le serveur. Elle fait partie des data que nous avons fait passer dans la fonction "render_template()".

Nous retrouvons comme dans tout fichier HTML la balise <body>, qui regroupe comme son nom l'indique l'ensemble du corps de notre page web. cette balise contient l'attribut onload qui permet à l'initialisation de la page de lancer différentes fonctions javascript contenues dans matrice.js. Ici, nous ne voyons qu'une seule fonction, qui sera lancée après le chargement de la page. Cette fonction sera définie dans le détail du fichier javascript.

<body onload="chargement({{fichiers}});">

La première partie du "body" représente la barre de navigation. Nous n'entrerons pas plus profondément dans le sujet mais cette barre permet de naviguer entre les différentes pages de notre site.

```
    <nav class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <a class="navbar-brand" href="welcome">Localization3DGUI</a>
    </div>
      <ul class="nav navbar-nav">
        <li><a href="matrice">Données de distance</a></li>
        <li class="active"><a href="#">Collecte des données</a></li>
        <li><a href="3d">3D Positioning</a></li>
        <li><a href="about">About</a></li>
        <li><a href="contact">Contact</a></li>
      </ul>
  </div>
</nav>
```

Nous utilisons ici une classe "class" définie par le framework "bootstrap". De plus, on peut dès à présent observer la balise container qui permet de centraliser le contenu de cette balise. Ce concept vient du framework bootstrap qui nous permet de grillager notre page HTML afin de placer nos différents éléments dans des cases bien spécifiques.

La prochaine partie est celle regroupant l'ensemble des boutons permettant de lancer des fonctions javascript, et dans notre cas, lancer des appels ajax au serveur pour lui demander d'effectuer des fonctions Python et des commandes Shell coté serveur.

```
<h3><span class="label label-default">Outils</span></h3>
<p>
  <h4 style="text-decoration:underline;text-size:4;font-weight:bold;">Que voulez vous faire?</h4>
  <br/>
  <div id="init">
    <button type="button" class="btn btn-success btn-block" onclick="init();">Connexion bluetooth et Redirection port</button>
  </div>
  <br/>
  <button type="button" class="btn btn-success btn-block" onclick="commandecu();">lancer la commande cu</button>
  <br/>
  <label>Temps de collecte pour les mesures (en secondes):</label><input id="temps" type="number" value="60" name="temps"/><br>
  <label>(Il est recommandé de choisir au moins 60s)</label>
  <br/>
  <button type="button" class="btn btn-success btn-block" onclick="commandecol()">lancer collecte</button>
  <button type="button" class="btn btn-danger btn-block" onclick="alert('Mesure stoppée');">Stopper la mesure</button>
  <br/>

  <div class="dropdown">
    <button class="btn btn-info dropdown-toggle" type="button" id="dropdownMenu1" data-toggle="dropdown"
    aria-haspopup="true" aria-expanded="true">
      fichier de distance enregistrés
      <span class="caret"></span>
    </button>
    <ul id="droplog" class="dropdown-menu" aria-labelledby="dropdownMenu1">
    </ul>
  </div>
</p>
```

Les boutons suivent tous un fonctionnement identique. On définit le type, puis on choisit la classe qui ici définira principalement son aspect. Cette classe fait partie des composants bootstrap et nous permet de donner un style visuel plus attractif à notre bouton. On ne porte ici un intérêt qu'à l'attribut onclick qui renvoie à l'appui du bouton vers une fonction javascript définie dans matrice.js. Dans le cas ci-dessous, l'appui sur le bouton déclenche la fonction init().

<button type="button" class="btn btn-success btn-block" onclick="init();">something</button>

La classe dropdown est un menu déroulant qui ne contient rien comme on peut le voir à la balise . Ce menu déroulant est en fait chargé à l'initialisation de la page via l'attribut onload présenté précédemment. Pour rappel, cet attribut permet le lancement d'une fonction au chargement de la page.

Pour finir avec cette page collecte.html, on peut constater qu'elle contient un tableau ne contenant qu'une en-tête. Il sera lui aussi rempli grâce à une fonction javascript. Nous pouvons donc nous intéresser à présent au fichier matrice.js après avoir présenté les fonctions et les appels aux requêtes ajax.

Le script matrice.js

Ce fichier est sur le même principe que engine.py, un ensemble de fonctions qui sont appelées par collecte.html mais elles peuvent l'être par n'importe quelle autre page. Cela permet de prévoir l'extension du site ainsi que d'y voir plus clair dans le code. Nous allons donc vous présenter les 6 fonctions qui le composent pour l'instant.

```
function chargement(fichiers){
  for(var i=0; i<fichiers.length;i++){
    var reg = new RegExp(".json$");
    if(reg.test(fichiers[i]) != 0){
      $('#droplog').append('<li><button type="button" onclick="charger_json(\''+fichiers[i]+'\');">'+fichiers[i]+'</li>');

    }
  }
}
```

Cette fonction est la fonction lancée grâce à l'attribut onLoad dans collecte.html. Elle comporte la définition de l'expression régulière ".json$". Cette expression nous permet dans la suite de récupérer tous les noms de fichiers du tableaux "fichiers" terminant par ".json". On ajoute ensuite ces noms sous forme de bouton à la balise contenant l'id "droplog". Cette balise n'est autre que le menu déroulant de la page

collecte.html. Notre menu déroulant n'est donc plus vide mais contient les fichiers json créés auparavant. Nous ajoutons ces noms sous forme de bouton car à l'appui sur l'un de ces bouton, nous allons afficher dans notre tableau vide le contenu de notre fichier grâce à la fonction "charger_json(fichier)".

```
function charger_json(fichier){
  var lien="affichage/"+fichier;
  $.ajax({
    url : lien, // La ressource ciblée
    type : 'GET', // Le type de la requête HTTP
    dataType : 'json', // Le type de données à recevoir, ici, du json.
    success : function(code_json,statut){
      $('#r').text('');
      for(var i=0; i<code_json.ranges[1][0].length;i++){
        $('#r').append('<tr><td>distance: '+i+'</td><td>'+code_json.ranges[1][0][i]+'mm</td></tr>');
      }
      console.log(code_json.ranges[1][0]);
    },

    error : function(resultat, statut, erreur){
    alert(resultat + '////'+ erreur+ '////' + statut);
  }
 });
}
```

Cette fonction fait une requête ajax vers le web service affichage/<fichier>. Nous récupérons donc via la variable "code_json" le contenu du fichier json, que nous pouvons exploiter et mettre en forme, afin qu'il apparaisse dans la balise contenant l'ID "r". cette balise est notre tableau vide de base. Notre tableau affiche à présent les données récupérées sur notre serveur.

```
function init(){
  var chargement=`<FONT size="4"><strong>Veuillez patienter, initialisation connexion et redirection du port</strong></FONT></br>`
  $('#init').html(chargement);
  document.body.style.cursor="wait";

  $.ajax({
    url : '/init', // La ressource ciblée
    type : 'GET', // Le type de la requête HTTP

    success : function(result,statut){
      console.log(result);
      $('#init').html('<button type="button" class="btn btn-danger btn-block" onclick="init();">Vous avez deja initialisé une co, recommencer?</button>');
      document.body.style.cursor="";
    },

    error : function(resultat, statut, erreur){
      console.log(resultat + '////'+ erreur+ '////' + statut);
      $('#init').html('<button type="button" class="btn btn-danger btn-block" onclick="init();">Vous avez deja initialisé une co, recommencer?</button>');
      document.body.style.cursor="";
    }
  });
}

function commandecu(){
```

La fonction "init()" nous permet de faire une requête ajax vers le web service "/init" qui s'occupera de lancer les différentes fonctions d'initialisation de connexion avec la Raspberry Pi. Nous en profitons pour modifier la balise "#init" pour remplacer le premier bouton par un bouton précisant que l'initialisation a déjà été demandée mais rien n'empêche de recliquer dessus.

Les fonctions "commandecu" et "commandecol" sont elles aussi des fonctions faisant appel à des requêtes ajax. Elle permettent elles aussi de joindre deux web services qui lanceront respectivement la commande call up et la commande de collecte de données vers un fichier json.

La fonction "clicstop()" permet quant à elle de stopper une commande de collecte en cours via une requête ajax appelant un service web.

APPLICATION: FONCTIONNALITES ET FONCTIONNEMENT DU SITE

Après avoir vu les fonctions de l'application, nous allons maintenant voir l'application du côté client, c'est-à-dire comment se présente le site et comment utiliser les différentes fonctions que nous avons implémentées.

Nous allons nous intéresser principalement à la page collecte.html, que nous avons longuement mentionné jusque-là. On peut observer la barre de navigation en haut de la page ainsi que plusieurs éléments principaux:

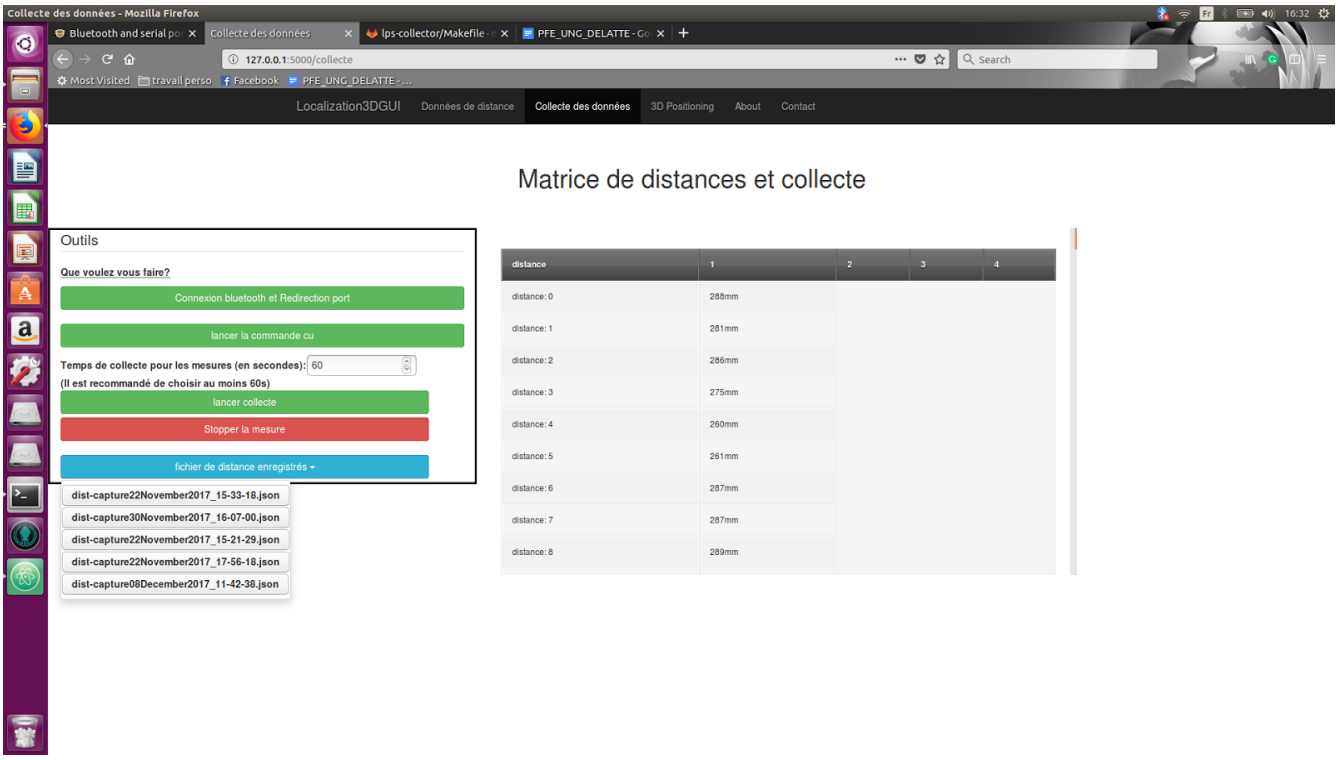
- un premier encart contenant plusieurs boutons ainsi qu'un champ modifiable: il contient les fonctions principales de l'application à réaliser par le serveur (via l'appel aux webservices par requêtes ajax) et la liste des noms de fichiers de données de collecte
- le tableau matriciel, dans lequel s'affichent les valeurs des distances collectées et parsées côté serveur

Le premier encart contient:

- un bouton pour réaliser la connexion bluetooth à la RPi et la redirection du port série vers un port TCP local (7777)
- un bouton pour lancer une commande cu

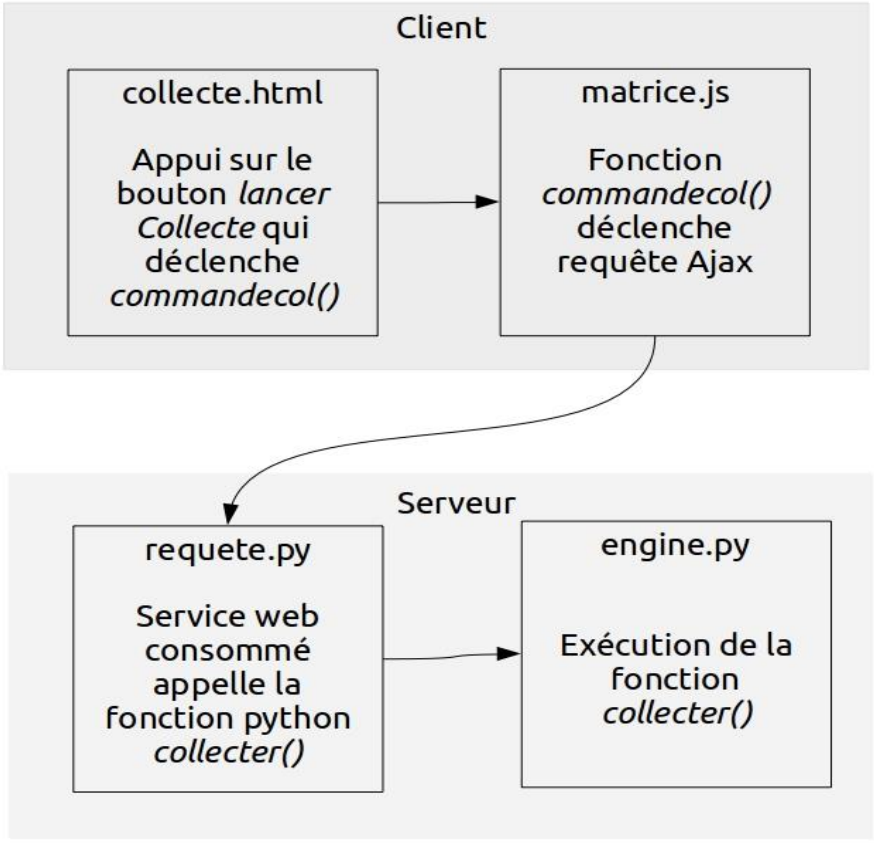
- un bouton pour lancer une collecte (prenant pour argument le champ "Temps)
- un bouton pour stopper la mesure manuellement
- un menu déroulant donnant accès à la liste des fichiers json stockés

Le second encart présente le tableau matriciel des données collectées correspondant aux données du fichier json sélectionné dans le dropdown du premier encart.



EXEMPLE DE CHEMIN DE FONCTION

Exemple avec le bouton *lancer collecte*



3. Démarche, organisation et travail restant

Cette troisième partie met en lumière les méthodologies que nous avons suivies, et plus particulièrement les méthodes mises en œuvre pour résoudre les problèmes auxquels nous avons été confrontés ainsi que les solutions apportées, les choix organisationnels effectués, et les résultats obtenus ainsi qu'une analyse de ces derniers.

CHOIX TECHNIQUES : MATERIEL ET FONCTIONNALITES

Réflexion sur les choix techniques et technologiques

Une partie des choix techniques matériels nous a été imposée par le contexte du projet. Nous avons donc dû partir de ce ces contraintes pour définir nos choix technologiques.

Nous sommes satisfaits des solutions que nous propose Flask. Nous aurions pu utiliser un serveur "django" mais nous ne sommes pas dans un contexte de production. Le serveur Flask reste tout à fait correct pour l'utilisation que nous en faisons.

Mettre sous forme d'API nos différentes fonctions permet de garder une conception modulaire de notre projet.

Nous avons découvert Bootstrap et son système de grille qui nous permet d'agencer plus facilement notre interface. Nous n'avons pas encore pu explorer ses réelles forces étant donné que nous travaillons plus sur le côté serveur pour l'instant.

JQuery nous facilite la vie et nous permet d'écrire beaucoup plus facilement nos requêtes Ajax.

RESULTATS OBTENUS ET ANALYSE

Fonctionnalités de l'application à ce jour

Nous avons préparé l'ensemble de la structure de notre application de façon à accueillir les différents features. Nous avons réussi à mettre en place, au sein de notre application, les différentes fonctions fournies par le tuteur.

Tout d'abord, nous pouvons récupérer les fichiers contenant les données de distance à traiter. Nous arrivons de plus à regrouper et afficher une liste de ces fichiers au sein d'une page Web. Ces fichiers sont sélectionnables afin d'afficher leur contenu dans cette même page Web.

Analyse de l'application: avantages et défauts, analyse des performances

Les temps de connexion sont encore un peu long pour pouvoir parler d'un système rapide à s'initialiser. Lors d'une collecte nous perdons pour l'instant la main sur notre interface Web et ne pouvons pas naviguer dessus. Nous devons attendre la fin de celle-ci.

L'initialisation de l'application n'est pas encore très automatisée. L'utilisateur est pour l'instant obligé de demander une connexion via un bouton au démarrage de son application. L'idéal serait que celle-ci se lance seule et que le client n'ait qu'à réaliser ses collectes de données.

Du côté matériel, il est a noté que si les ancres sont trop proches du TAG (moins de 20 cm) les données sont erronées.

Nous avons un accès facile aux fichiers de collecte déjà stockés. La lecture des fichiers json se fait aisément par l'interface Web par l'intermédiaire d'une transformation en un tableau HTML.

Travail restant

Il nous reste à créer les fonctions nécessaires pour envoyer les matrices de distance et récupérer les matrices de positions via le serveur de notre tuteur contenant l'algorithme de calcul. Il nous faudra ensuite afficher ces matrices de positions à travers une carte 3D.

Après avoir réalisé cet ensemble de fonctions en temps différé, il nous a été proposé de réussir à afficher cette carte 3D, en temps réel, pendant la collecte des données.

En parallèle, il nous faudra continuer à travailler sur l'apparence et l'ergonomie du site afin de le rendre attractif, et surtout facilement utilisable par un utilisateur.

ORGANISATION DU TRAVAIL ET DEMARCHE SUIVIE

Répartition du travail et organisation

En ce qui concerne la répartition du travail, nous avons dû mettre en place des outils pour travailler de façon efficace en binôme, étant donné que la structure de l'application fait que nous ne pouvions pas travailler sur deux parties indépendantes des codes.

Pour cela nous avons utilisé un dépôt GitLab de l'INRIA, dans lequel nous pouvions tour à tour mettre à jour certaines parties des codes en ayant un historique de notre avancée et pouvoir revenir à des versions précédentes. Nous nous sommes aussi servis de GitKraken, une interface graphique faite pour travailler sur des dépôts Git. Nous pouvons plus aisément voir l'arborescence de notre projet et "merge" ou "rebase" nos branches avec plus de précision.

Nous avons systématiquement pris des notes de nos avancées et du travail réalisé au cours de nos séances, qui nous servaient de base pour rédiger notre Wiki à chaque fin celles-ci. Nous prenions ensuite le temps de regarder le travail effectué et de planifier en conséquence celui de la séance suivante. Nous nous basions sur les échéances et le travail qu'il restait à exécuter.

Calendrier prévisionnel et replanification des échéances

Dates	Prévisionnel	Replanification
Avant le 29/09/17	Élaboration du Cahier des charges, de la liste des tâches et du calendrier prévisionnel Recherche et formation sur les technologies à utiliser	Élaboration du Cahier des charges, de la liste des tâches et du calendrier prévisionnel Recherche et formation sur les technologies à utiliser
Avant fin Octobre	Serveur de collecte et début du développement de l'interface logicielle	Travail en parallèle sur le serveur et l'interface logicielle
	Interface logicielle	

Avant fin Décembre		
Avant fin Février	Intégration finale des différentes parties et optimisation (Serveur de collecte des données & Serveur de localisation & Interface logicielle)	Avoir une application stable et fonctionnelle(débuggage) Ajout des fonctions de récupération et d'affichage des matrices de position
A la fin du PFE		Intégration finale des différentes parties et optimisation (Serveur de collecte des données & Serveur de localisation & Interface logicielle)

Le projet nous étant étranger au premier abord, nous avons donc mis en place un calendrier prévisionnel pour mettre une échéance sur chaque tâches à effectuer. Ce calendrier a cependant dû être revu au gré des problèmes rencontrés, mais nous avons pu atteindre la plupart de nos objectifs en terme d'avancée sur le projet. La mise en place des collectes de données a été plus longue que prévue car nous nous sommes rendus compte que le développement et le débuggage de celui-ci étaient plus dépendants de l'interface web que prévu. Nous avons donc travaillé en parallèle sur l'élaboration du front-end de façon à pouvoir tester les liens entre client et serveur. il a donc été nécessaire d'effectuer une replanification des échéances en se donnant pour objectif de terminer la base front-end / back-end avant la soutenance de mi-décembre.

Le serveur de localisation possédant une API, il devrait ne pas être trop fastidieux de récupérer les positions désirées. Nous n'avons pas encore commencé le travail d'affichage des données de position mais nous avons déjà réfléchi aux technologies pouvant représenter une solution. Par exemple, plotly.js est une librairie javascript permettant l'affichage de cartes 3D.

Conclusion

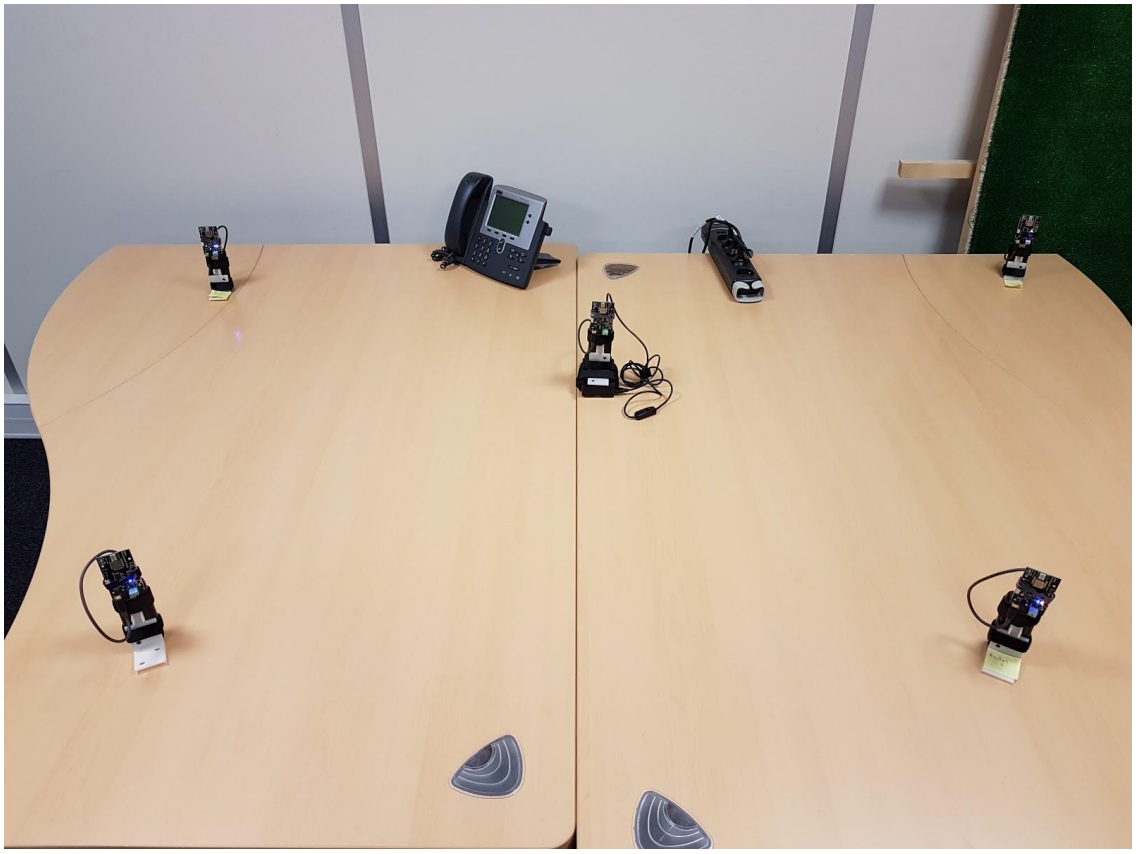
Nous avons pu, grâce cette première approche, nous familiariser avec les méthodes de gestion d'un projet complexe. En effet, nous avons eu l'opportunité de définir nous même les technologies à utiliser pour remplir au mieux les fonctions demandées tout en respectant les contraintes fixées. Il a été intéressant de devoir réfléchir au contexte et de pouvoir cibler les apports supplémentaires que nous pourrions fournir au cahier des charges initial.

Possédant une grande autonomie, nous avons dû nous confronter à plusieurs problématiques comme la gestion du temps et des échéances. Le travail étant en binôme nous avons donc dû apprendre à concevoir un logiciel stable en travaillant sur plusieurs features en parallèle.

Nous ne pouvons pas proposer de conclusion plus ouverte à ce jour car nous pensons que notre réflexion doit encore évoluer. Après avoir pu finaliser le projet, nous aurons donc une preuve que ce concept fonctionne en pratique. Cependant il nous est impossible à ce stade d'estimer les limites de celui-ci étant donné notre niveau de maturité sur le sujet. Il serait intéressant d'explorer les limites du prototype afin d'évaluer la possibilité d'une production.

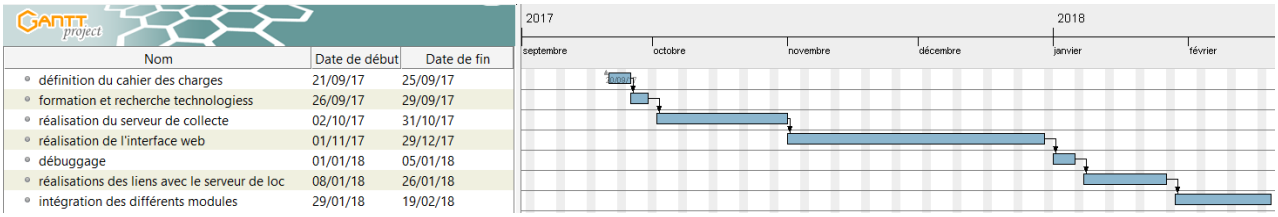
Annexes

MATERIEL ET PLANNING



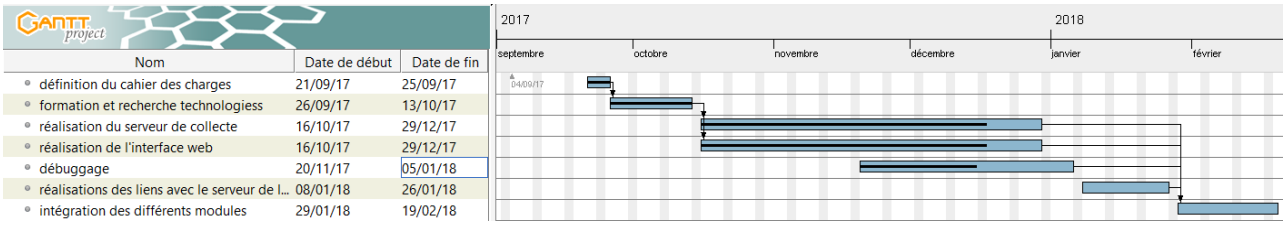
Système Loco Positioning

4 anchors et 1 tag connecté à une RPi3



Nom	Date de début	Date de fin
définition du cahier des charges	21/09/17	25/09/17
formation et recherche technologiess	26/09/17	29/09/17
réalisation du serveur de collecte	02/10/17	31/10/17
réalisation de l'interface web	01/11/17	29/12/17
débuggage	01/01/18	05/01/18
réalisations des liens avec le serveur de loc	08/01/18	26/01/18
intégration des différents modules	29/01/18	19/02/18

Calendrier prévisionnel



Nom	Date de début	Date de fin
définition du cahier des charges	21/09/17	25/09/17
formation et recherche technologiess	26/09/17	13/10/17
réalisation du serveur de collecte	16/10/17	29/12/17
réalisation de l'interface web	16/10/17	29/12/17
débuggage	20/11/17	05/01/18
réalisations des liens avec le serveur de l...	08/01/18	26/01/18
intégration des différents modules	29/01/18	19/02/18

Calendrier replanifié