ANNEE
2017
2018

INRIA Lille – Nord Europe
Equipe DEFROST
Parc scientifique de la Haute Borne
40, avenue Halley - Bât A 2ème étage
59655 Villeneuve-d'Ascq CEDEX
FRANCE
Tél : +33 3 59 57 78 00

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

Nicky UNG & Hugo DELATTE
Département IMA – 5ème année

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 mémoire 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.

Remerciements

Nous souhaitons tout d'abord exprimer notre gratitude envers les personnes qui ont contribué à rendre possible cette opportunité professionnelle et de quelque manière que ce soit, au bon déroulement de notre projet. Nous remercions notre tuteur M. Jérémie DEQUIDT, de nous avoir offert cette opportunité d'effectuer ce projet au sein de l'équipe de recherche Defrost de l'INRIA Lille. Nous tenons à remercier aussi Messieurs Gang ZHENG et Roudy DAGHER, pour leur bienveillance, leur patience et leur aide, qui nous ont permis de mener à bien notre projet, qu'ils ont initié et tutoré ; projet qui nous a aidé développer nos compétences, et mobiliser nos connaissances déjà acquises.

Nicky Ung, Hugo Delatte

Table des matières

Introduction

Dans le cadre de notre formation ingénieur a Polytech Lille, il nous est demandé de réaliser un projet de fin d'études, seul ou en binôme. Ce projet est la conclusion de notre master en ingénierie informatique et électronique. Il vise à mobiliser le socle de connaissances que nous avons acquis tout au long de notre cursus, et aussi à nous familiariser au rôle d'ingénieur. En effet, cela nous permet de développer un sens analytique sur un projet et de nous mettre en condition d'entreprise tout au long de celui-ci.

De nos jours, de plus en plus d'entreprises s'intéressent au développement des drones et de leurs applications. On constate dès lors que la localisation de ceux-ci est un enjeu majeur. Toutefois, cela reste porté sur une localisation longue distance en extérieur et d'une précision discutable. Par ailleurs, on constate que peu s'intéressent à la localisation intérieure de précision malgré les nombreuses applications potentielles.

Notre projet porte donc sur la localisation d'un drone en intérieur. Nous sommes chargés de mettre en place un système capable de visualiser et d'exploiter des résultats de position donnés grâce à un algorithme de localisation fourni.

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 et la navigation autonome de ceux-ci passent 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 et si le temps le permet nous tenterons de travailler en temps réel.

A l'aboutissement du projet, nous devrions être capable via l'interface homme machine développée d'afficher 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

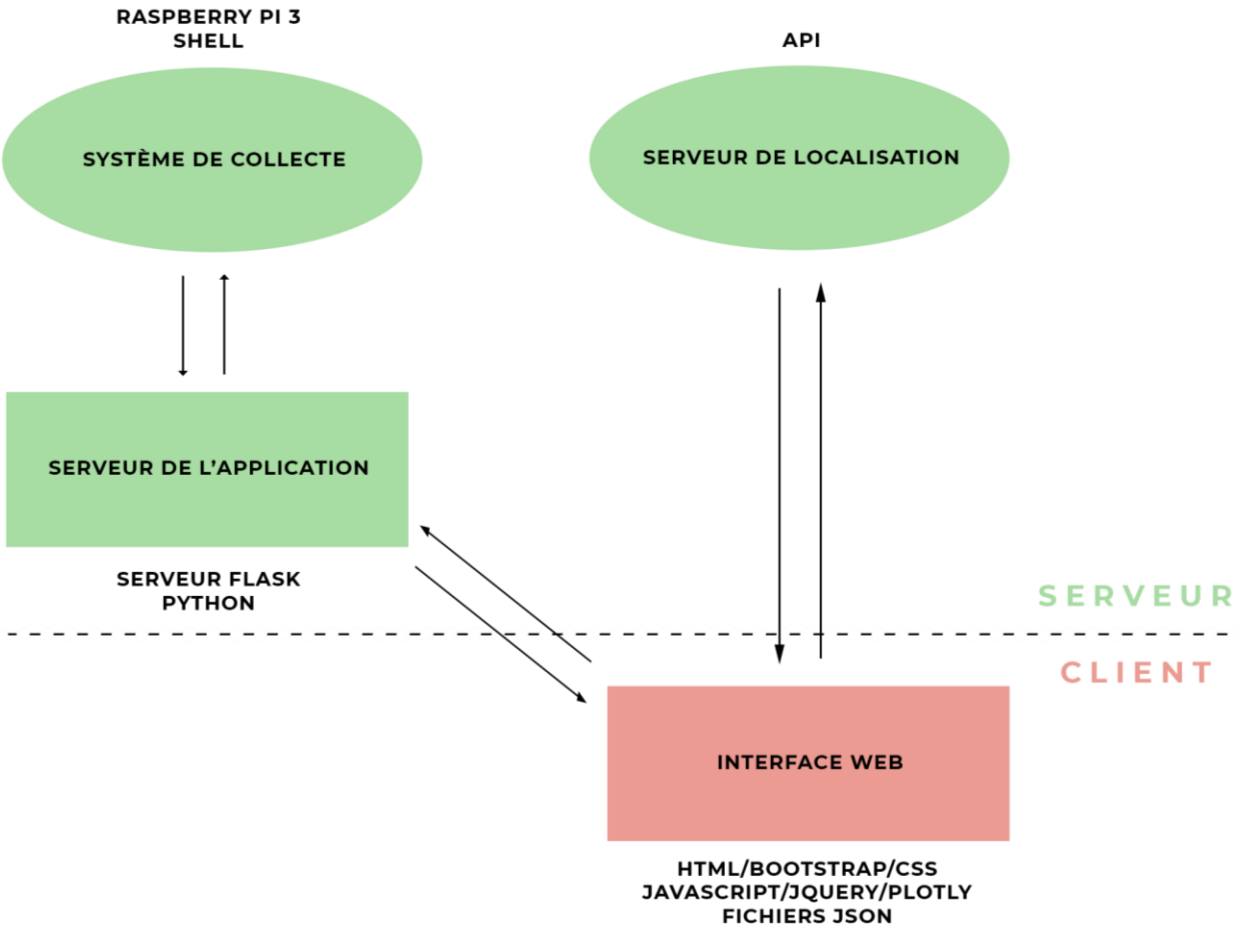
- API
- Python / Serveur Flask

• Serveur de localisation (fourni) : Calcule grâce à l'algorithme de localisation et la matrice D les positions spatiales des 4 noeuds 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:

- 1) Choisir les technologies utilisées ainsi que l'architecture (OS / APIs)
    - o Se documenter sur le projet
    - o Langages et technologies utilisées
    - o Matériel utilisé
- 2) Développer les différentes parties du projet
    - o Mettre en place du travail sur le Git
    - o Réaliser le serveur de collecte
    - o Récupérer les trames envoyées par Bluetooth par le TAG via la Raspberry Pi
    - o Intégrer le serveur de localisation pour interpréter les trames
    - o Tester et optimiser

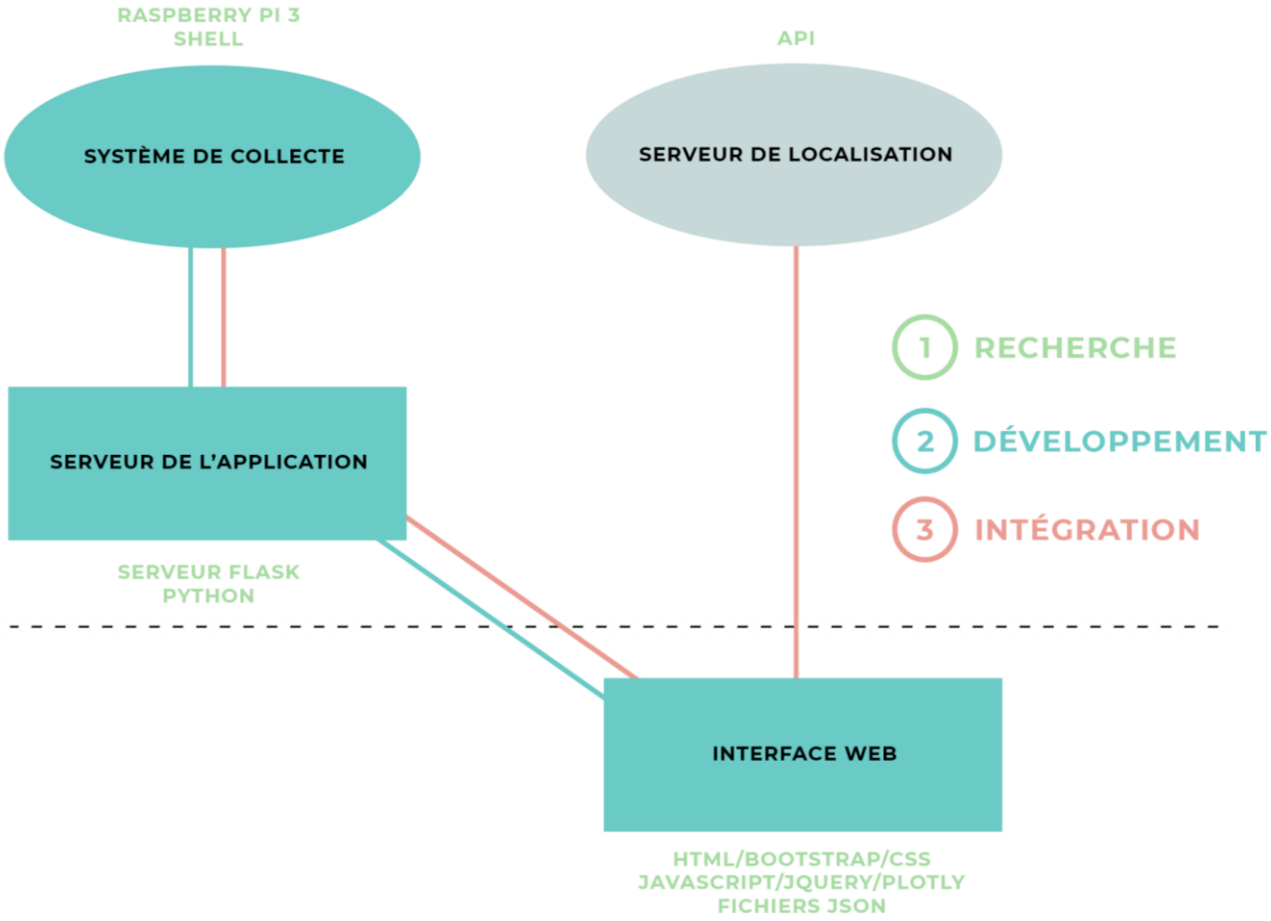
o Réaliser l'interface Web utilisateur

- 3) Intégrer et connecter les différentes parties du projet

o Implémenter le serveur de collecte et de localisation

o Afficher et traiter les données stockées dans les fichiers JSON via le serveur (temps différé + temps réel)

o Tester et optimiser



PRESENTATION DES ELEMENTS THEORIQUES

Le dispositif Loco Positioning System

Le système que nous utilisons, le Loco Positioning System (normalement accompagné du Crazyflie 2.0, le drone qui va de pair), est un système de localisation en intérieur développé par Bitcraze. L'objectif de l'équipe Defrost et de nos tuteurs de PFE sur ce projet, est de tester et vérifier un algorithme calculant la position du drone en fonction des données de distance renvoyées par ce système.

Le Crazyflie 2.0 (comme la plupart des autres quadcoptères et robots) n'a aucune notion réelle de sa position dans l'espace. Les capteurs embarqués (accéléromètres et gyroscopes) peuvent être utilisés pour donner une idée approximative de ses mouvements, mais ils manquent de précision pour un positionnement correct à long terme. La solution consiste à utiliser un système externe pouvant fournir des informations sur la position actuelle. Le système GPS peut être utilisé à l'extérieur, mais à l'intérieur, les options sont limitées et souvent complexes ou coûteuses. C'est pourquoi le système Loco Positioning a été développé.

Le système

Le système Loco Positioning est un système de positionnement local utilisé pour trouver la position 3D absolue des objets dans l'espace. Il est similaire à un système GPS miniature. La base du système est un ensemble d'ancres (ou « balises ») qui sont positionnées dans la pièce (à comparer aux satellites dans le GPS), elles servent de référence. L'autre partie du système est constituée d'un ou plusieurs Tags (à comparer au récepteur GPS) qui sont fixés aux objets à positionner. En envoyant de courts messages radio haute fréquence entre les ancres et les Tags, le système mesure la distance entre chaque ancre et les Tags et calcule la position des balises à partir de ces informations. Toutes les informations nécessaires pour calculer la position sont disponibles dans le Tag qui permet l'estimation de la position sur le drone même, contrairement à beaucoup d'autres systèmes de positionnement où la position est calculée dans un ordinateur externe et envoyée au Crazyflie. En ajoutant la connaissance de sa position à un Crazyflie 2.0, il est capable de voler de manière autonome sans contrôle manuel. Cela ouvre un éventail de cas d'utilisation et d'applications important.

Mode de positionnement

Le Loco Positioning System intègre deux modes de positionnement différents : **Two Way Ranging** (TWR) and **Time Difference of Arrival** (TDoA).

En mode **TWR**, le Tag ping les ancres séquentiellement, ce qui lui permet de mesurer la distance entre le Tag et les ancres. En utilisant cette information, un minimum théorique de 4 ancres est nécessaire pour calculer la position 3D d'une étiquette, mais un nombre plus réaliste est 6 pour ajouter de la redondance et de la précision. Un avantage de ce mode est qu'il reste précis même lorsque le Tag quitte l'espace délimité par les ancres. Dans ce mode, il est également très facile d'ajouter plus d'ancres pour étendre la portée

du système, par exemple pour couvrir plusieurs pièces. Le principal inconvénient est que la balise communique activement avec les ancres, ce qui signifie que l'ajout de balises utilisera plus de bande passante radio et nécessitera que chaque Tag partage le système de balises. TWR a été utilisé avec succès pour un Crazyflie. Faire voler plusieurs Crazyflies, ce mode est possible en utilisant le mode expérimental TDMA qui a été testé pour jusqu'à 4 Crazyflies, mais avec des performances réduites.

En mode **TDoA**, le système d'ancres envoie en continu des paquets de synchronisation. Un Tag écoutant ces paquets peut calculer la distance relative à deux ancres en mesurant la différence de temps d'arrivée des paquets. A partir des informations du TDoA, il est possible de calculer la position 3D dans l'espace. Le grand avantage de ce mode est que le Tag est seulement passivement à l'écoute, donc les nouveaux Tags n'ajoutent aucune charge au système, ce qui permet de positionner n'importe quel nombre de tags ou Crazyflies. Cela en fait un mode parfait pour l'essaimage. Un inconvénient est que TDoA est plus sensible au placement de l'ancrage, idéalement le Tag doit toujours être à l'intérieur, ou très proche, de l'espace délimité par le système d'ancrage. Cela signifie que TDoA fonctionne mieux avec 8 ancrages placés dans les coins de l'espace de vol. De plus, puisque le système d'ancres est synchronisé, il n'est pas possible (du moins pas aisément) d'utiliser plus de 8 ancres pour étendre la portée du système.

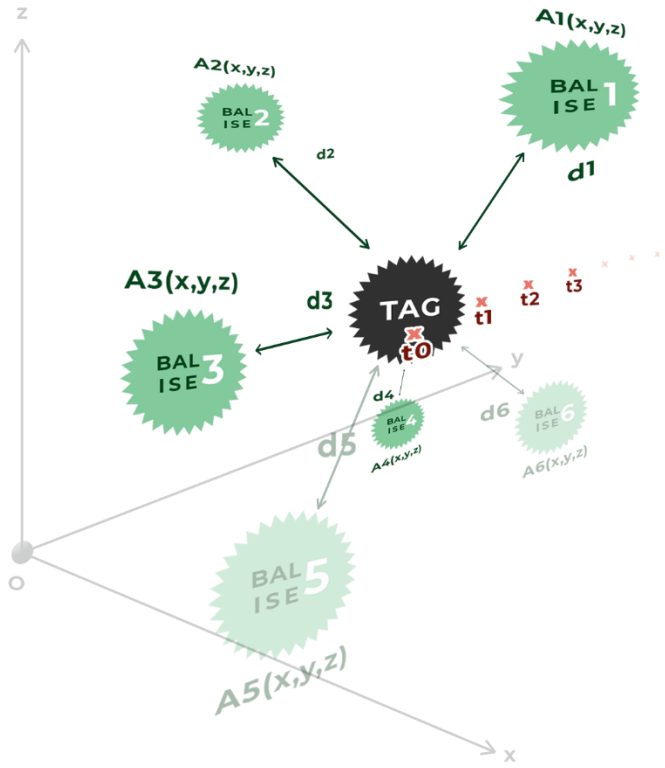
Performance

Le système Loco Positioning est basé sur la puce Decawave DWM1000 et a une précision théorique de 10 cm. Les performances de positionnement 3D dépendent de la configuration du système (positionnement des ancres, mode de positionnement) et de l'environnement. Le système Loco Positioning a été principalement conçu pour une utilisation en intérieur.

Hypothèses de fonctionnement : conditions minimales/optimales, limites

Nous avons dans un premier temps étudié et compris le mode de fonctionnement qu'il nous a été demandé d'utiliser pour notre projet. Le schéma ci-dessous représente une configuration aléatoire du système, avec le Tag encadré par les balises. Nous pouvons configurer jusque 6 balises, mais nous n'en disposons que de 4 pour notre projet, ainsi qu'un Tag. Le Tag calcule sa distance relative à chaque balise séquentiellement et donc on dispose de 4 distance à chaque tour (un tour correspond à chaque balise interrogée une

fois). Pour que l'algorithme fonctionne, il faut disposer de 10 tours de ces équations pour 1 instant t.



A partir de ce dispositif, on récupère donc pour chaque instant t 10 distances relatives entre le Tag et les balises, ce qui permet de résoudre le système pour 1 instant t, comme présenté ci-dessous.

t0	t1	t2	t3
n { $(d1\ d2\ d3\ d4)_0$ $(d1\ d2\ d3\ d4)_1$ $(d1\ d2\ d3\ d4)_2$ $(d1\ d2\ d3\ d4)_3$ ... $(d1\ d2\ d3\ d4)_{10}$	...	...	...

Nous avons aussi des hypothèses de fonctionnement qui nous ont été données par la datasheet du dispositif (selon la configuration du protocole de communication) ainsi que par nos tuteurs, qui ont au préalable réalisé une étude théorique et produit l'algorithme. D est la matrice de distance de tous les instants t relevés, avec pour chaque t, n=10 équations de distances relatives. A et X sont respectivement les deux matrices contenant les données de position de tous les instants de la collecte des balises et du Tag.

On a donc pour t=t0 par exemple :

$$D = (A , X)$$

Pour t0:

$$D_0 = (A_0 , X_0)$$

$$\begin{pmatrix} d1_0 \\ d2_0 \\ d3_0 \\ d4_0 \end{pmatrix} = \begin{pmatrix} x1_0 & x2_0 & x3_0 & x4_0 & , & xT_0 \\ y1_0 & y2_0 & y3_0 & y4_0 & , & yT_0 \\ z1_0 & z2_0 & z3_0 & z4_0 & , & zT_0 \end{pmatrix}$$

$4 < nb(balises) \leq 6$
$1 < nb(tags) \leq 4$
n = 10 mesures
pas d'orientation 3D

Les hypothèses précédentes représentent les conditions de fonctionnement minimales du dispositif, à savoir :

- Un nombre de balises minimal égal à 4, 6 permettant un meilleure précision
- Un nombre de Tags utilisables compris entre 1 et 4
- L'algorithme ne permet pas d'orienter spatialement le motif de positionnement obtenu car aucune balise n'est calibrée par rapport au repère virtuel utilisé pour l'affichage

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 ET FONCTIONNEMENT

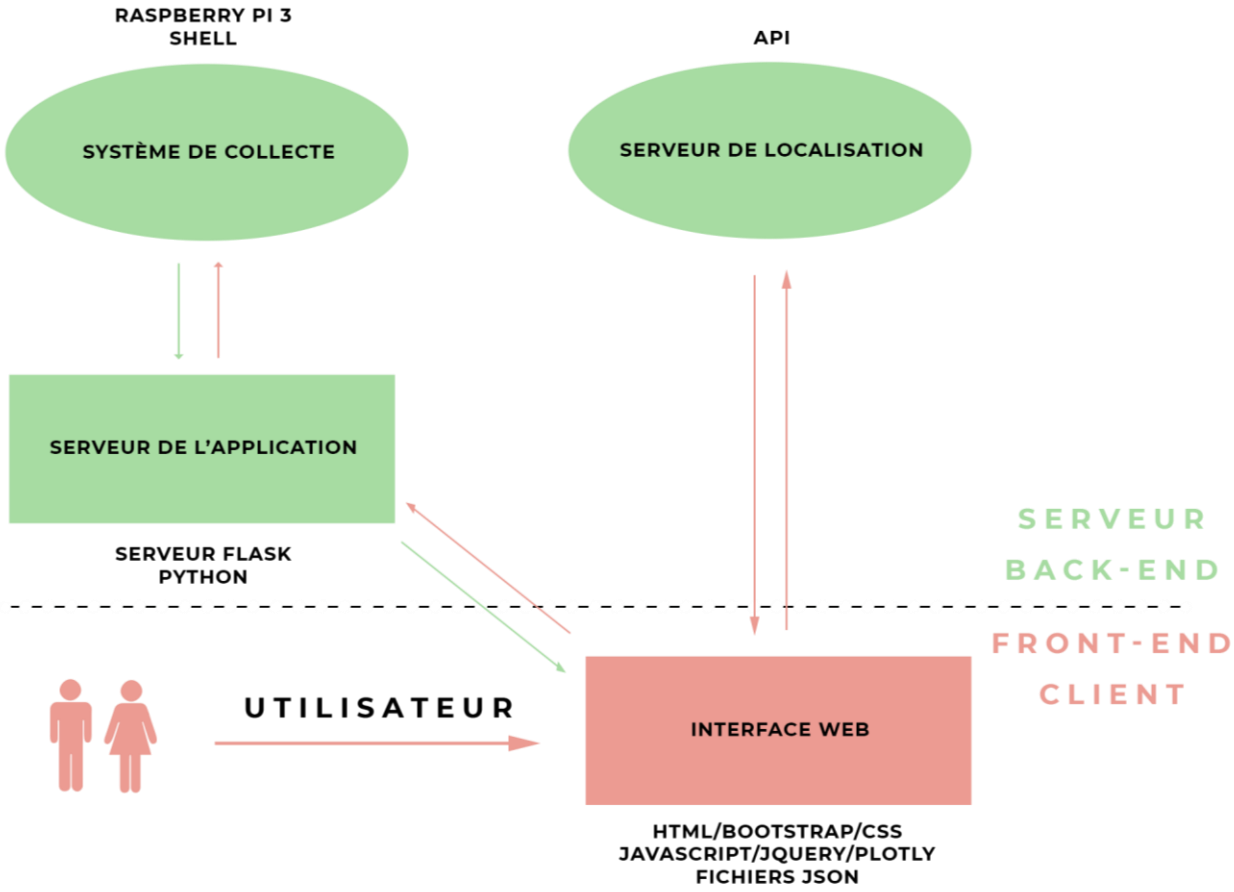
Comme expliqué précédemment, le but de ce projet est de mettre en place une application qui permet d'un côté :

- 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

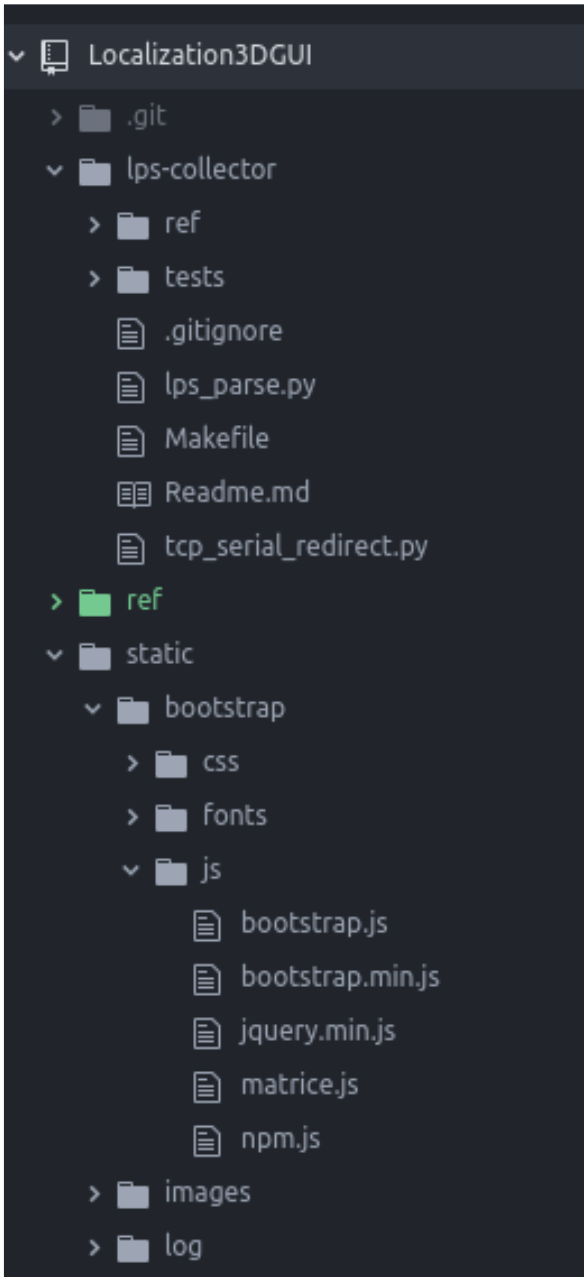
Pour une meilleure compréhension de notre travail, il faut savoir comment se décline notre dépôt de projet:

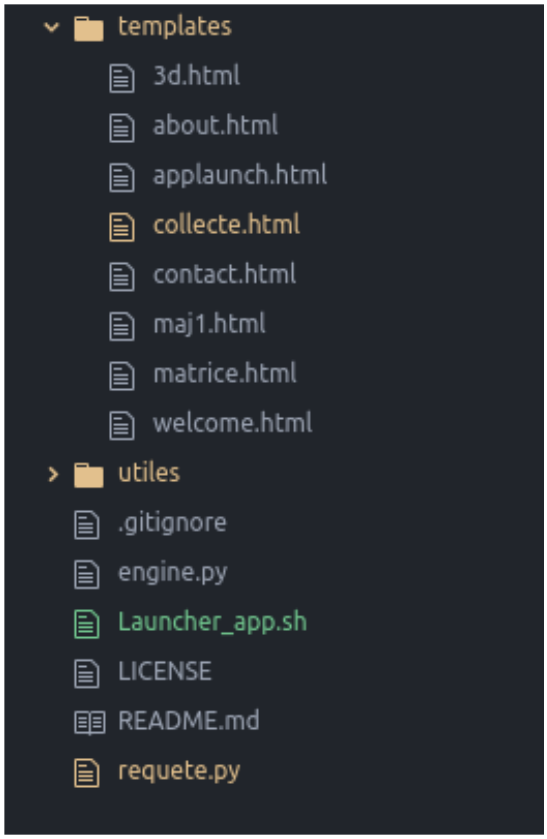
- les scripts engine.py et requete.py qui regroupent les fonctions de l'application côté serveur

- 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 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" à 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 alors de stopper la collecte d'éléments manuellement. Elle utilise la même commande vue précédemment.

```
def parse():
    print("Parsage du fichier")
    global date
    x=subprocess.Popen("python lps-collector/lps_parse.py static/log/dist-capture"+date+".log",stdin=None,stdout=subprocess.PIPE
    while True:
        output = x.stdout.readline()
        err = x.stderr.readline()
        if output == '' and x.poll() is not None:
            time.sleep(1)
            nom_fich= "static/log/dist-capture"+date+".log"
            data = json.load(open('static/log/dist-capture'+date+'.json','r'))
            data['nom'] = 'dist-capture'+date+'.json'
            json_data=json.dumps(data)
            date = -1
            return json_data
        if output:
            print output.strip()
            if output.strip() == "-1":
                return "-1"
        if err:
            print err.strip()
```

La fonction parse() récupère le "fichier.log" venant d'être créée 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. Cette fonction a changé depuis son premier prototype. En effet, nous récupérons via un "pipe" relié à la fonction python "lps_parse.py". Nous pouvons comme ça gérer les erreurs retournées par celle-ci.

La fonction "lps_parse.py" était à la base une fonction dite "boite noire" mais nous avons fait de nombreux changements pour pouvoir gérer les cas d'erreur pendant la collecte qui n'étaient pas pris en compte à la base. Même si cette fonction python n'est pas définie dans engine.py nous allons vous l'expliquer ici même par souci de logique et de clarté.

```
def parse(file_name):
    # A parser for the tag log messages where each line are in the form
    # distance <id>:  <range>mm.
    # returns the distance MxN matrix having M measurements of N distance tuples.
    fid = open(file_name)
    ids = []
    ranges = []
    pattern = r"^distance (?P<id>\d+)\:\s+(?P<range>\d+)mm$"
    pattern2 = r"^distance (?P<id>\d+)\:\s+(?P<range>-?\d+)mm$"
    for line in fid:
        tokens = re.match(pattern, line.strip())
        if tokens != None:
            id = eval(tokens.group('id'))
            range_mm = eval(tokens.group('range'))
            ids.append(id)
            ranges.append(range_mm)
        else:
            tokens = re.match(pattern2, line.strip())
            if tokens != None:
                id = eval(tokens.group('id'))
                ids.append(id)
                ranges.append(0)
    fid.close()
```

Tout d'abord, cette fonction était composée d'une partie qui ouvrait le fichier log venant d'être créé après une collecte. Grace à des expressions régulières, il récupérait les id des balises et la valeur de distance associés pour les placer dans deux tableaux. Nous nous sommes rendus compte que si une balise est trop près du tag, les distances deviennent négatives et le programme les éliminait, ce qui provoquait des trous dans les tableaux et donc des décalages. Nous avons donc créé une nouvelle expression régulière qui reconnaît ces valeurs négatives et stock la valeur "0" à la place. Les valeurs négatives signifiaient que la balise et le drone était assez proches mais pas assez précise pour définir une distance exploitable et nous avons donc décidé que l'on pouvait associer une distance petite à une distance nulle.

Il s'avère que si la connexion n'est pas correctement établie et que nous faisons une collecte, nous récupérons un fichier log vide. Nous récupérerions donc des tableaux vides après cette première étape. Nous avons donc rajouté une condition vérifiant que les tableaux contiennent bien des valeurs, sinon le programme ne va pas plus loin.

```
if not ids :
    return (None,-1)
```

Les balises et le tag communiquent en permanence. Lorsque nous lançons une collecte, nous pouvons donc récupérer une première valeur qui ne correspond pas à la

balise avec l'id le plus faible. De la même manière, la collecte pouvait se terminer alors que nous n'avions pas récupéré le dernier set de valeurs complet. Voici les deux fonctions qui permettent de supprimer le premier set de valeur et le dernier si ils ne sont pas complets.

```
minimum=min(ids)
maximum=max(ids)

#suppr of the bad first values
while ids[0] != minimum:
    del ids[0]
    del ranges[0]
#suppr of the bad end values
taille = len(ids)
while ids[taille-1] != maximum:
    del ids[taille-1]
    del ranges[taille-1]
    taille = len(ids)
```

Nous nous sommes aperçus qu'il y avait une chance qu'un échange entre une balise et le tag se passe mal et qu'une valeur n'apparaisse pas. Il fallait définir une façon de retrouver ces valeurs manquantes et de les combler par une valeur "None" ou "0".

```
prec = ids[0]-1
i=0
test=[False,False,False,False,False,False]
while(i<6):
    if(test[ids[i]-1]== False):
        test[ids[i]-1]= True
    i+=1

i=0
while( i < len(ids)):
    if ( ids[i] != prec+1 and prec != maximum and test[ids[i]-1]==True):
        #ajout ligne
        ids.insert(i,prec+1)
        ranges.insert(i,0)
        print("----------------------------")
    elif prec == maximum:
        if ids[i] != minimum:
            ids.insert(i,minimum)
            ranges.insert(i,0)
            print("////////////////////////////")

    #print(i,ids[i],ranges[i])
    prec = ids[i]
    i+=1
```

Tout d'abord, grâce à la première boucle "while", nous vérifions quelles étaient les balises branchées pendant la collecte, car il en faut 4 au minimum mais nous pouvons en brancher jusque 6 et chacune possède un ID différent. Ensuite, via la deuxième boucle "while" nous vérifions que tous les sets soient bien complets. Si dans un set une valeur manque, nous retrouvons l'id de la balise n'ayant pas réussi à communiquer. Nous la rajoutons au bon emplacement dans le tableau et lui affectons la distance "0". Il est vrai que remplacer une valeur inconnue par 0 fausse complètement la position du drone sur le set en question mais nous ne savons pas si le serveur de localisation pourra gérer la valeur "None".

Après avoir fait ces différents traitements, nous récupérions des tableaux plutôt stables et complets. Il nous restait à transformer ces deux tableaux "ids" et "ranges" en une matrice de sets. Nous calculons d'abord le nombre de balises pour connaître la taille d'un set puis nous créons ensuite des paquets de valeurs.

```
N = 0
while True:
    N=N+1
    if ids[N-1]>=ids[N]:
        break
```

```
data = [[] for x in range(len(ranges)/N)]
for i,range_mm in enumerate(ranges):
    data[i/N].append(range_mm)
return (np.matrix(data),0)
```

Cette fonction parse est appelée par la fonction "main" du programme qui ressemble à ceci:

```
def main(file_name, position=()):
    # parse range measurements from log file
    print('Parsing ' +  file_name)
    (D,status) = parse(file_name)
    print status
    if status == 0:
        info = {'comments': 'Created on ' + time_stamp(),
                'position': position,
                'ranges': D.tolist()}
        print 'Generating ' + file_name.rpartition('.')[0] + '.json'
        with open(file_name.rpartition('.')[0] + '.json', 'w') as outfile:
            json.dump(info, outfile, sort_keys=True, indent=2, ensure_ascii=False)
        outfile.close()
        print 'Generating ' + file_name.rpartition('.')[0] + '.csv'
        np.savetxt(file_name.rpartition('.')[0] + '.csv', D, delimiter=",")
    if status == -1:
        print "-1"
```

Si la collecte s'est bien passée, nous pouvons créer les deux fichiers (json et csv) grâce à la matrice "D" retournée par la fonction "parse". Si le fichier était vide, alors le

statut retourné est "-1" et cette même fonction "main" affiche "-1" dans le shell et donc dans le "pipe" qui est réceptionné par la fonction "parse" de engine.py.

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.

```
#pages webs

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

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

@app.route('/matrice')
def matrice():
    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')
def collecte():
    return render_template('collecte.html', titre="collecte")
```

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.

Dans notre projet, on retrouve la route /collecte qui mène vers la page principale de notre site web. C'est via cette page que nous ferons les collectes et le stockage de celles-ci. On retrouve la route "/3d" qui nous permettra d'afficher en grand les visualisations 3D sauvegardées et la route "/matrice" qui nous permet de visualiser dans un tableau les valeurs de distance ou de position en pleine page.

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'
```

Le web service appelé à la route "/commandecol" permet de lancer une collecte selon un certain temps transmis par un formulaire. La collecte terminée est triée via la fonction "parse" on renvoie une valeurs d'erreur ou de succès.

```
@app.route('/commandecol')
def commandecol():
    temps=request.args['temps']
    collecter(temps)
    var=parse()
    return var
```

L'API suivante permet de récupérer la liste des fichiers contenus dans "static/log" et de la renvoyer au client.

```
@app.route('/chargement')
def chargement():
    log=json.dumps(os.listdir("static/log"))
    return log
```

Cette dernière API permet de faire l'inverse d'un chargement. Via le client on envoie un fichier json qui sera stocké dans le dossier "static/log".

```
@app.route('/stock',methods=['GET', 'POST'])
def stock():
    if request.method=='POST':
        data=request.json
        nom=data['nom']
        with open('static/log/'+nom, 'w') as f:
            f.write(json.dumps(data))
    return "0"
```

Tous les services web ont le même squelette et fonctionnent de la même manière. 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.

Trois services web ont été supprimés depuis la dernière version. Ces services web permettaient l'initialisation de la connexion avec la Raspberry Pi, et la configuration de cette connexion. Nous avons décidé que ces fonctions seraient lancées en dehors de l'application Flask, en même temps que celle-ci dans un fichier bash. Ainsi nous n'avons plus besoin d'appuyer sur des boutons pour les activer. Tout est lancé au démarrage de l'application.

Le script app.sh

```
#!/bin/bash

# Redirect to serial port
python lps-collector/tcp_serial_redirect.py /dev/rfcomm0 9600 &
PROC_REDIRECT=$!
sleep 5
echo "cu -s9600 -lttyACM0" | nc localhost 7777

# Launch flask server
python requete.py

# User types ctrl+c
# stop tcp redirect
kill -9 $PROC_REDIRECT
```

Apres s'être connecté en Bluetooth, il nous reste plus qu'à lancer "app.sh" pour lancer la redirection des ports, la commande de "call up" pour récupérer les données de distances qui transitent, et le serveur flask.

Par ailleurs, nous avons créé un script /etc/init.d/autoconnect.sh pour automatiser la connexion bluetooth.

!/bin/bash
echo -e 'power on \nconnect B8 :27 :EB :80 :DC :89 \nquit' | Bluetooth

Ensuite, on fait un sudo chmod +x sur ce fichier pour donner les droits d'exécution sur ce fichier, puis au redémarrage ou à chaque démarrage, en ayant la RPi déjà allumée, le script se lance et permet de se connecter automatiquement à la RPi. Toutefois, il reste tout à fait possible de lancer la connexion via le terminal avec la commande rfcomm connect.

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 "navbar" 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.

```
<div class="row">
  <div class="col-lg-5" style="border-width:3px;border-style:solid;border-color:black;">
    <div class="row">
      <h3><span class="label label-default">Outils</span></h3>
      <h4 style="text-decoration:underline;text-size:4;font-weight:bold;">Que voulez vous faire?</h4>
      <br/>
      <div id="collecte">
        <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>
      </div>
      <button type="button" class="btn btn-danger btn-block" onclick="alert('Mesure stoppée');">Stopper la mesure</button>
      <br/>
    </div>

    <div class="row">
      <div class="col-lg-5">
        <div class="dropdown" >
          <button class="btn btn-info dropdown-toggle" type="button" id="dropdownMenu1" data-toggle="dropdown"
          aria-haspopup="true" aria-expanded="true">
          fichiers de distance enregistrés
          <span class="caret"></span>
        </button>
        <ul id="droplog" class="dropdown-menu scrollable-menu" aria-labelledby="dropdownMenu1 ">
        </ul>
      </div>
    </div>
    <div class="col-lg-offset-1 col-lg-5">
      <div class="dropdown">
        <button class="btn btn-info dropdown-toggle" type="button" id="dropdownMenu2" data-toggle="dropdown"
        aria-haspopup="true">
        fichiers de position enregistrés
        <span class="caret"></span>
      </button>
      <ul id="droplog2" class="dropdown-menu scrollable-menu" aria-labelledby="dropdownMenu1 ">
      </ul>
    </div>
  </div>
</div>
```

On distingues deux parties:

- Celle avec les boutons de collecte et d'arrêt
- Celle avec les deux boutons de class "dropdown".

Pour les premiers 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>

Les deux menus déroulants ayant la classe "dropdown" sont des menus déroulants qui ne contiennent rien comme on peut le voir à la balise . Ces menus déroulants

sont en fait chargés, des dossiers de distance et position, 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 garder à jour ces menus déroulants, les fichiers sont rechargés dedans après chaque nouvelle collecte.

Pour finir avec cette page collecte.html, on peut constater qu'elle contient un tableau ne contenant qu'une en-tête et une balise div ayant l'ID "tester". Ces deux éléments seront eux aussi remplis grâce à des fonctions 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 composé, sur le même principe que engine.py, d'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 8 fonctions qui le composent pour l'instant.

```
function chargement(){
  var reg = new RegExp("([0-9].json)$");
  var reg2 = new RegExp("(_output.json)$");
    $.ajax({
      url : "chargement", // 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){
        console.log("rechargé");
        $('#droplog').text('');
        $('#droplog2').text('');
        for(var i=0; i<code_json.length;i++){
          if(reg.test(code_json[i]) != 0){
            $('#droplog').append('<li><button type="button" onclick="charger_json(\''+code_json[i]+'\');">'+code_json[i]+'</li>');
          }
          if(reg2.test(code_json[i]) != 0){
            $('#droplog2').append('<li><button type="button" onclick="charger_json(\''+code_json[i]+'\');">'+code_json[i]+'</li>');
          }
        }
      },
      error : function(resultat, statut, erreur){
        console.log(resultat + '////'+ erreur+ '////' + statut);
      }
    });
}
```

Cette fonction est la fonction lancée grâce à l'attribut onLoad dans collecte.html ou après une collecte. Elle comporte deux définitions d'expression régulière. Ces expressions nous permettent dans la suite de récupérer les fichiers de distance ou de position se trouvant dans un dossier de stockage. On différencie un fichier de position d'un fichier de distance par la terminaison "_output.json". On ajoute ensuite ces noms sous forme de bouton à la balise contenant l'id "droplog" ou "droplog2" selon la correspondance. Ces balises ne sont autres que les menus déroulants 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 boutons, 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;
  var reg = new RegExp("([0-9].json)$");

  $.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('');
      if (reg.test(fichier) != 0){
        for(var i=0; i<code_json.ranges.length;i++){
          $('#r').append('<tr id="val'+i+'"><td>distance: '+i+'</td><td>'+code_json.ranges[i][0]+'</td><td>'+coc
        }
      }
      else {
        console.log(code_json.result.X);
        for(var i=0; i<code_json.result.X.length;i++){
          $('#r').append('<tr id="val'+i+'"><td>distance: '+i+'</td><td>'+code_json.result.X[i][0]+'</td><td>'+c
        }
      }
    },
    error : function(resultat, statut, erreur){
    alert(resultat + '////'+ erreur+ '////' + statut);
  }
 });
}

function commandecol(){
```

"Commandecol" est la fonction principale de notre application, qui par une requête ajax vers un service web de notre serveur, lance la commande de collecte de données vers un fichier JSON. En parallèle de cette requête ajax, on introduit à la place du bouton de collecte une barre de chargement nous permettant de suivre la progression de la collecte.

```
function commandecol(){

  temps=$('#temps').val();
  $('#collecte').html(`
    <div class="progress progress-striped active">
    <div class="progress-bar"></div>
  </div>`);
  timer(temps);
  $.ajax({
    url : '/commandecol', // La ressource ciblée
    type : 'GET', // Le type de la requête HTTP
    data : 'temps=' + temps,
    dataType :"json",
    success : function(data,status){
      $('#collecte').html(`
        <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>
        `);
      if(data == "-1"){
        alert("erreur collecte");
      }
      else{
        console.log(data);
        chargement();
        traitement_loc(data,data.nom);
      }
      console.log("test");
    },

    error : function(resultat, status, erreur){
      console.log(resultat);
      $('#collecte').html(`
        <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>
        `);
      chargement();
      alert("erreur lors de la collecte, nom de l'erreur: "+ erreur);
    }
  });
}

function clicStop(){
```

Lorsque la requête ajax est un succès, cela veut dire que le serveur lui renvoie bien une information. Nous vérifions si cette information est égale à "-1". Dans ce cas, le serveur a rencontré une erreur lors de la collecte et prévient le client. Sinon, on récupère un fichier de distance. Dans ce cas on peut donc lancer la fonction "rechargement()" qui mettra à jour le menu déroulant de distance avec ce nouveau fichier json. On lance ensuite la fonction "traitement_loc" qui est le point d'entrée dans le traitement des fichiers de distance grâce au "serveur de localisation" mis en place par le tuteur. Elle appelle une autre fonction "api_position" qui appelle elle-même la fonction "graph". Ces trois fonctions permettent d'envoyer le fichier de distance au serveur, d'aller chercher le fichier de position lorsqu'il est prêt et de l'afficher sous forme de représentation 3D sur le client.

Nous ne nous attarderons pas sur ces fonctions dans cette partie car elles sont assez conséquentes et qu'elles seront toutes présentées dans le chapitre "Serveur de Localisation".

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

Nous avons une fonction récursive "timer" qui met à jour la barre de progression toutes les secondes lors d'une collecte. Elle permet simplement de la faire progresser au cours du temps en changeant l'attribut css "width".

```
function timer(n,i=0) {
    $(".progress-bar").css("width", (100/n)*i + "%");
    if( i < n) {
      setTimeout(function() {timer(n,i+1);}, 1000);
    }
  }
```

La dernière des fonctions est la fonction "comparaison(A,Q)". Cette fonction, comme son nom l'indique, nous permet de comparer grâce aux positions des balises, les distances qui les séparent. On mesure ensuite à la main les distances réelles et nous pouvons ainsi apprécier la qualité de la localisation des balises.

LE SERVEUR DE LOCALISATION

Le serveur de localisation et son API dédiée

Le serveur de localisation est un serveur auquel on peut accéder à distance via des requêtes Ajax. Il a été développé par l'un de nos tuteurs, Roudy DAGHER, et permet d'obtenir les positions des balises et du Tag à partir d'un fichier de distances que nous lui envoyons. Ce serveur de localisation est codé d'une manière qui nous oblige à suivre un protocole d'envoi et de récupération de données composées de plusieurs requêtes ajax. Voici la description de ce protocole nous permettant de récupérer la matrice de position qui nous intéresse :

Du côté client, lors du lancement d'une collecte de données de distances, nous recevons un fichier à envoyer au serveur de localisation. Il est tout d'abord nécessaire d'envoyer une requête de type POST contenant notre fichier. Le serveur, à la réception de celui-ci, constitue un "job" contenant les adresses URL permettant de retrouver :

- Notre fichier envoyé
- Un fichier log de l'opération

- Le fichier traité (celui de position)

Ce job est identifié par un numéro. Ce numéro sera renvoyé par le serveur en cas de succès.

Nous faisons donc une seconde requête de type GET en direction de l'adresse du job délivrée précédemment, cette fois pour récupérer l'adresse URL où se trouve notre fichier traité par l'algorithme.

Maintenant que nous savons où est stocké notre fichier de position sur le serveur de localisation, nous pouvons exécuter une troisième et dernière requête de type GET qui nous renverra un fichier JSON contenant deux matrices de position: L'une pour le tag l'autre pour les balises.

Rentrons plus en détail sur la conception des différentes fonctions qui réalisent ce travail. Nous appelons tout d'abord "traitement_loc" qui enverra notre fichier de distance via une requête ajax POST. On constate que nous avons rajouté un token dans la partie "headers Authorization". En effet, après s'être inscrit sur le site "allgo.inria.fr", où est stocké le serveur, nous avons reçu un code pour pouvoir s'authentifier lors d'une connexion à leur API.

```
function traitement_loc(distances,nom_fich){
  var f = new File([JSON.stringify(distances, null, 2)], nom_fich, {type: "application/json"});
  //the formData embed an uploaded file and the webapp id
  formData = new FormData();
  formData.append("job[webapp_id]","166/");
  formData.append("job[param]",nom_fich);
  formData.append("files[0]",f);

//envoie des donnee vers loc
$.ajax({
  type: 'POST',
  url: "https://allgo.inria.fr/api/v1/jobs",
  data: formData,
  cache: false,
  contentType: false,
  processData: false,
  headers:{
    'Authorization': 'Token token=36a9cc9ec37d2d8095b820e384df9d53',
    'Accept':'application/json'
  },
  success: function(data) {
    console.log("post success");
    console.log(data);
    var url=data.url;
    var job=/[0-9]*$/.exec(url);
    api_position(url,job,nom_fich);
  },
  error: function(data) {
    console.log(data);
  }
});
}
```

Le serveur de localisation nous renvoie l'adresse url du "job" nous permettant de récupérer les adresses de stockage du fichiers de position, de distance et le .log associé à ce job. Nous stockons l'url dans une variable et nous isolons le numéro du job, via une expression régulière, qui nous servira par la suite. Nous appelons ensuite la fonction "api_position "en lui donnant comme paramètres ceux que nous venons de récupérer ainsi que le nom du fichier de base.

```
function api_position(url,job,nom_fich){
  setTimeout(function(){
    $.ajax({
      type: 'GET',
      url: url,
      dataType:"json",
      headers:{
        'Authorization': 'Token token=36a9cc9ec37d2d8095b820e384df9d53'
      },
      success: function(data) {
        if(data.status == "in progress"){
          console.log(data.status);
          api_position(url,job,nom_fich);
        }
        else{
          var reg = new RegExp("[a-zA-Z0-9_-]*");
          console.log(nom_fich);
          let nom_base= reg.exec(nom_fich);
          address=data[job][nom_base+"_output.json"];
          console.log("address:"+ address);
          graph(address,nom_base+"_output.json");
        }
      },
      error: function(data) {
        console.log("error"+data); //formatted JSON data
      }
    });
  }, 500);
}
```

Cette fonction a pour but de récupérer, grâce au job créé, l'adresse où se trouve le fichier de position. Le principale problème à prendre en compte et que le serveur de localisation prend un certain temps à traiter le fichier de distance afin de renvoyer le job contenant le fichier de position. Nous avons donc créé une fonction récursive qui vérifie si la valeur renvoyée est bien l'adresse du fichier de position pour passer à la fonction d'affichage ou alors le message "in progress" qui déclenche redéclenche l'appel à "api_position". Le "setTimeout" empêche de boucler trop de fois en temporisant un appel toutes les 500ms. Si nous recupérons bien le bon fichier, alors nous pouvons récupérer l'adresse finale qui aura comme clé "<nom du fichier envoyé>_output.json". On enlève donc le ".json" du nom du fichier de base, on rajoute le complément et nous cherchons dans le fichier json cette clé.

Nous pouvons enfin effectuer la dernière requête Ajax à l'adresse récupérée. Ce dernier appel se trouve dans la fonction "graph".

```
function graph(fichier,nom_output){

  Plotly.d3.json(fichier, function(error,figure){

    figure.nom = nom_output;

    $.ajax({
        url: "/stock",
        type: "POST",
        data : JSON.stringify(figure),
        contentType: "application/json",
        success: function(response) {
            console.log(response);
        },
        error: function(error) {
            console.log(error);
        }
    });
    let xl = []
    let yl = []
    let zl = []
    let pos = figure.result.X;
    for (var i=0; i< pos.length; i++){
      xl.push(pos[i][0])
      yl.push(pos[i][1])
      zl.push(pos[i][2])
    }

    var drone = {
      x: xl, y: yl, z:zl,
      mode: 'lines+markers',
      marker: {
        size: 4,
        line: {
        color: 'rgba(217, 217, 217, 0.14)',
        width: 0.5},
        opacity: 0.8},
      type: 'scatter3d'
    };

    xl = []
    yl = []
    zl = []
    pos = figure.result.A;
    for (var i=0; i< pos.length; i++){
      xl.push(pos[i][0])
      yl.push(pos[i][1])
      zl.push(pos[i][2])
    }
```

```
    var balises = {
      x: xl, y: yl, z:zl,
      mode: 'markers',
      marker: {
        size: 6,
        line: {
        color: 'rgba(217, 217, 0, 0.14)',
        width: 0.5},
        opacity: 0.8},
      type: 'scatter3d'
    };
    var layout = {
      title: "position of the drone in space",
      titlefont: {
        color: "#7f7f7f",
        size: 25
      }
    };
    Plotly.plot('tester',[drone,balises],layout);
  });
}
```

Ici, pas de requête ajax explicite pour récupérer le fichier de position car il est implicitement fait dans la fonction "plotly.d3.json". Avant même d'afficher cette matrice de position dans un graphe 3D on l'envoie via une requête Post vers un webservice appelé "stock" qui s'occupera de stocker ce fichier position dans le dossier "static/log".

On définit ensuite trois tableaux xl, xy et xz qui contiendront successivement les valeurs de la matrice X puis celles de la matrice A. On distingue ensuite trois variables :

- "Drone" dans laquelle on vient associer les valeurs de positions. On en profite pour définir les options d'affichage de ces points. En effet on décide de relier les points par des lignes pour faire apparaitre la trajectoire.
- "Balises" est basé sur le même principe que "Drone". On lui alloue les positions des balises. On définit aussi différemment son affichage. Sa couleur n'est pas la même et nous ne relions pas les points entre eux.
- "Layout" permet de définir les caractéristiques du cadre globale, le titre, etc.

Il ne nous reste enfin plus qu'à utiliser la fonction "plotly.plot" avec comme paramètre un id de balise se trouvant dans "collecte.html" où implanter le graphe ainsi que les différentes valeurs à implanter soit ici "[drone,balises]". Le dernier paramètre n'est autre que le "layout" général défini précédemment.

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)

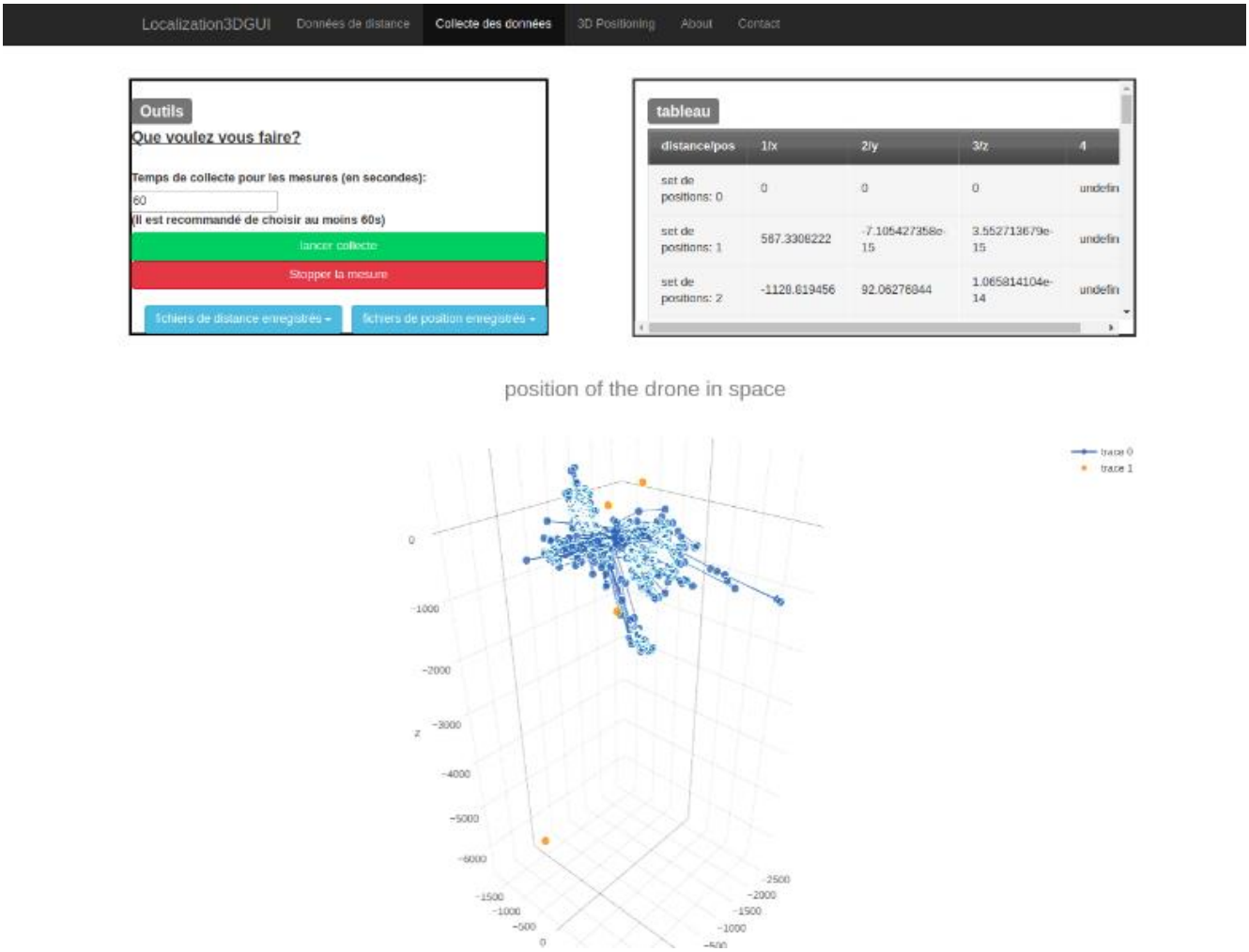
• un bouton pour lancer une commande cu

• 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.



LA VISUALISATION

La visualisation des données de distance

Matrice de distances

fichiers de distance enregistrés ▾ fichiers de position enregistrés ▾

tableau

distance/pos	1/x	2/y	3/z	4
set de distances: 0	2755	565	3383	2242
set de distances: 1	2750	560	3369	2246
set de distances: 2	2738	568	3353	2233
set de distances: 3	2754	567	3349	2234
set de distances: 4	2783	534	3382	2233
set de distances: 5	2756	560	3383	2242
set de distances: 6	2787	523	3375	2238
set de distances: 7	2799	577	3346	2243
set de distances: 8	2807	548	3335	2236
set de distances: 9	2787	555	3346	2243
set de distances: 10	2801	587	3383	2243
set de distances: 11	2436	559	3356	2232
set de distances: 12	2493	560	3370	2229
set de distances: 13	2841	567	3375	2227
set de distances: 14	2864	585	3344	2203
set de distances: 15	2810	565	3368	2235

La visualisation des données de position

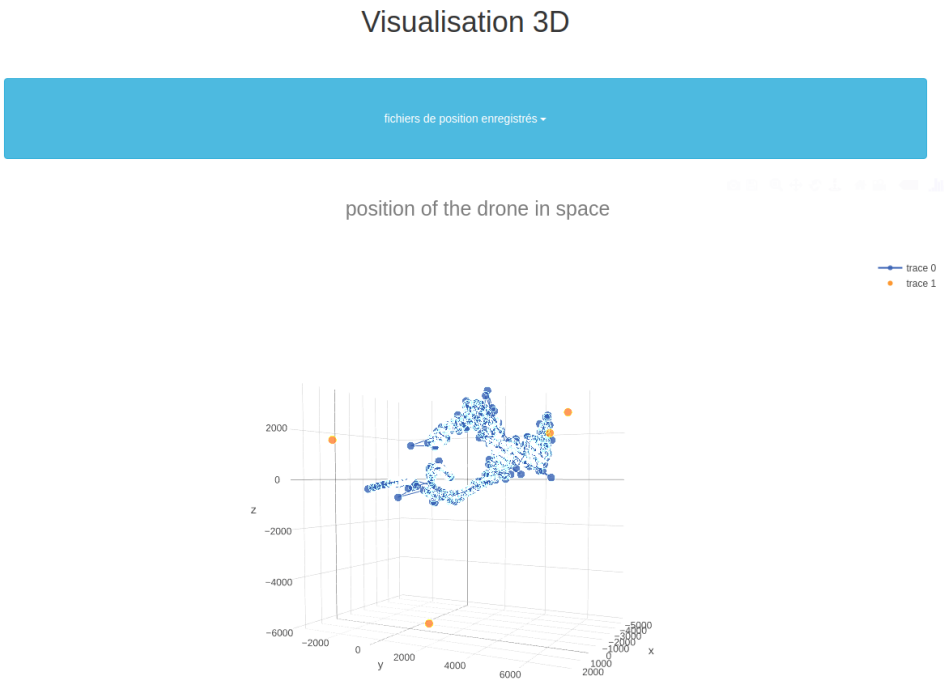
Matrice de positions

fichiers de distance enregistrés ▾ fichiers de position enregistrés ▾

tableau

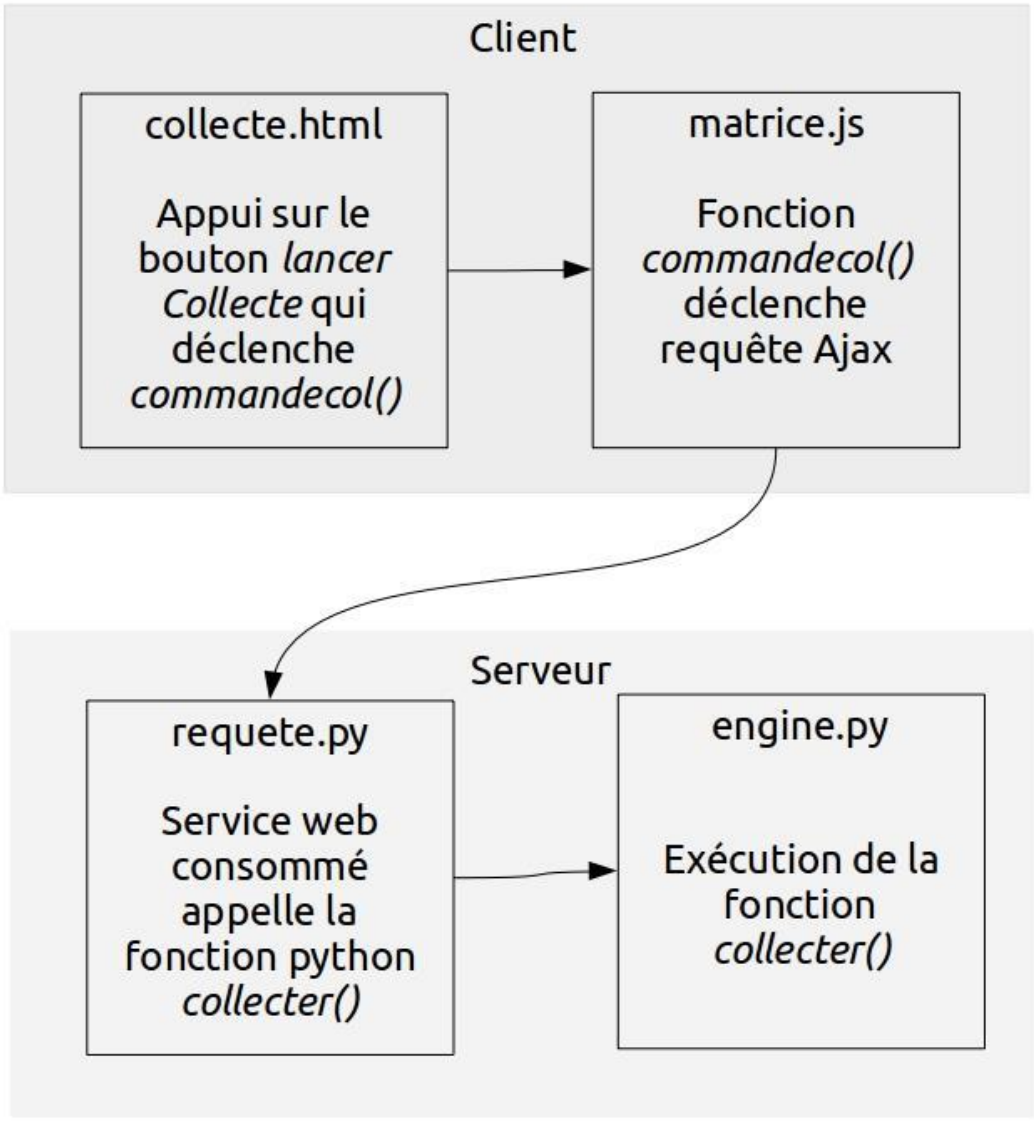
distance/pos	1/x	2/y	3/z	4
set de positions: 0	0	0	0	undefined
set de positions: 1	209.7206443	1.776356839e-15	7.105427358e-15	undefined
set de positions: 2	79.85905217	203.1743452	3.552713679e-15	undefined
set de positions: 3	180.4060518	275.9340623	28.36526211	undefined
set de positions: 4	302.1969868	521.2506834	29.7326427	undefined
set de positions: 5	316.1465012	-13.09503867	62.39231487	undefined
set de positions: 6	424.8527538	-52.43000648	80.93148481	undefined
set de positions: 7	434.0152115	-61.01313868	81.40881516	undefined
set de positions: 8	436.8739523	-356.6181059	78.70580297	undefined

La visualisation des positions dans l'espace



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 – PARTIE UNE

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.

RESULTATS OBTENUS ET ANALYSE – PARTIE DEUX

Fonctionnalités de l'application à ce jour

En plus des fonctionnalités déjà énoncées, nous avons réussi a comprendre et intégrer via une API le serveur de localisation dans notre projet. Les fichiers de distances récupérées auparavant sont envoyé au serveur de localisation et les fichiers de postions retournés sont stockés afin d'être affiché et étudié par la suite. L'application permet

l'affichage de ces matrices de positions dans un tableaux afin d'analyser en détail les valeurs résultantes ou alors d'afficher cette matrice via une map 3D afin de visualiser la position des balises et le déplacement du drone. Nous avons mis à jour la page 3D positioning qui nous permet en plus de la carte 3D de visualiser un coefficient de qualité retourné par le serveur de localisation ainsi que la distance supposée entre chacune des balises. Une page entière est réservée à la visualisation des tableaux de position ou de distance.

Analyse de l'application: avantages et défauts

Apres avoir réussi cette application en temps différé, il nous avait été demandé de chercher une méthode temps réel et de la réaliser si cela était possible. En principe nous pourrions voir le déplacement instantané du drone pendant sa collecte et non à la fin de celle-ci.

Nous avons rencontré des problèmes qui nous ont empêché de mener à bien cette partie. Le serveur de localisation fourni des positions viables seulement si nous lui fournissons une certaine quantité de données de distance. Il faut donc forcément attendre que la collecte dure au minimum plusieurs secondes. Ensuite le seconds problème est que le serveur ne renvoie pas instantanément la matrice de positions. Le travail de l'algorithme nous coute encore des secondes. A cause de ces différents temps d'attentes nous ne pouvons pas définir notre solution comme "temps réel".

Enfin il y a un troisième problème moins important mais à soulever. A chaque matrice envoyée au serveur, celui-ci créé un job. En temps réel, nous devrions renvoyer régulièrement des "petites" matrice au serveur qui remplir la mémoire de jobs contenant des petites matrices. Nous ne savons pas la conséquence que cela peut avoir sur le programme mais nous pensons que le serveur n'a pas été conçu pour fonctionner selon ce schéma.

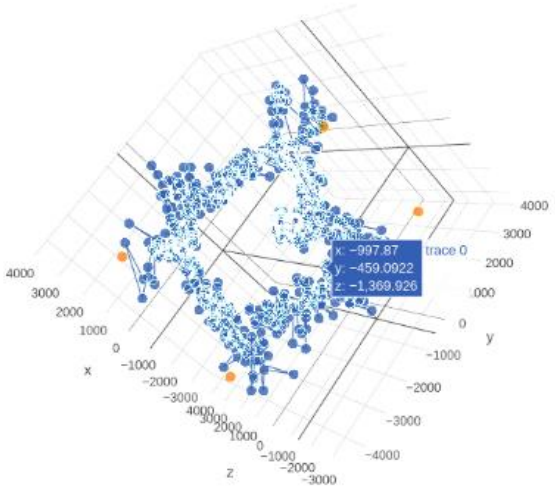
Dans notre application nous trouvons de nombreuses requête ajax. Nous avons eu au terme de la première partie, des problèmes avec le bon fonctionnement asynchrone de celles-ci. En effet une requête ajax est par default asynchrone et il nous ai donc normalement possible d'interagir avec le serveur via différentes requêtes en parallèle.

Ici les requêtes étaient exécutées à la suite et non en parallèle. Il s'avère que le problème vient du serveur Flask. En effet, Flask est un micro-framework python qui permet de pouvoir écrire plus facilement un serveur web en python. Flask est un serveur

compatible WSGI. Un serveur est dit "compatible WSGI" quand il est capable de transmettre une requête HTTP normale à votre application (ici le code python) via le protocole WSGI, et qu'il est capable de récupérer une réponse HTTP depuis votre application via le protocole WSGI pour en faire une requête HTTP normale. Le protocole WSGI étant synchrone notre serveur exécute les taches une par une et il nous est impossible à ce stade d'exécuter plusieurs requêtes asynchrones.

Tests et analyse des résultats obtenus

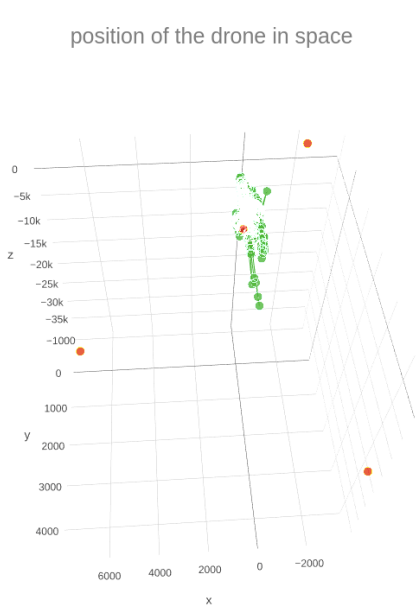
Les conseils de notre tuteur était de faire un test de référence afin d'apprécier la précision de l'algorithme pour la localisation des balises. Nous avons donc réparti l'ensemble de nos 4 balises au 4 coins de la salle puis nous avons fait une collecte de 100 secondes. Les résultats que nous avons obtenus sont les suivants.

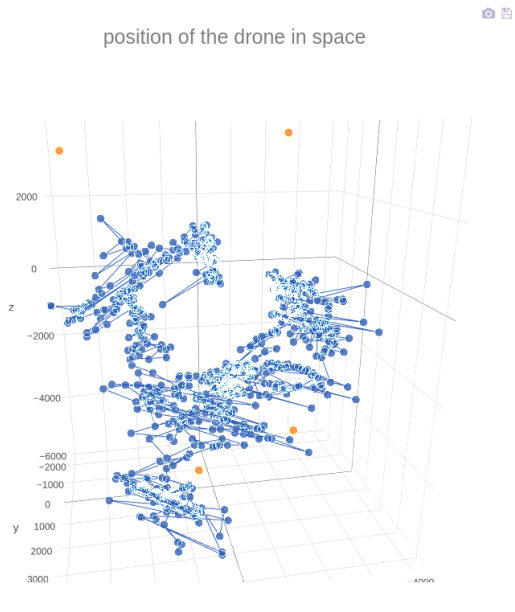


distances entre les balises

d12=5520.824209582885
d13=8130.169068993899
d14=9111.207819070149
d23=7763.163700818323
d24=8725.17839195227
d34=6557.283960710319
Quality=0.1960308244

Voici les distances séparant chacune des balises entre elles (mesurées avec le télémètre)**: d12:** 5 m, **d13:** 8,5 m, **d14:** 7,9 m, **d23:** 7,5 m, **d24:** 9,2 m, **d34:** 4.5 m. Il est évident que les valeurs ne sont pas exactement les mêmes, toutefois celles retournées par l'algorithmes se rapprochent très fortement de celle mesurées avec le télémètre. Ce résultat est donc encourageant pour la suite du projet.





distances entre les balises

d12=25401.14789725152
d13=20383.749481778603
d14=17725.431544459272
d23=43179.62249951296
d24=40044.73845027929
d34=4210.4164081227345
Quality=35.59941934

distances entre les balises

d12=8931.60678598935
d13=6055.1688423846035
d14=8548.94275901336
d23=10638.139401940918
d24=5288.597809454667
d34=9085.266554633396
Quality=0.2322382835

On observe sur le tableau ci-dessus deux collectes différentes, de 120 secondes. A gauche, on observe une collecte dans laquelle on place le Tag au milieu de la pièce sans le bouger pendant la collecte. A droite, on bouge en revanche le Tag pendant les 120 secondes de la collecte. En se basant sut le modèle théorique décrit en partie 1, on comprend donc que déplacer le Tag permet d'ajouter des équations de distance au système de 10 équations par instant t à résoudre, et donc d'améliorer la qualité de la mesure, ce que l'on peut observer ici (qualité beaucoup supérieure sur la colonne de droite, avec déplacement).

De plus, nous avons réalisé toute une batterie de tests pour essayer au mieux d'évaluer l'algorithme et dégager des conjectures en ce qui concerne les paramètres d'influence sur la mesure.

Tout d'abord, comme évoqué ci-avant, nous avons pu conjecturer que déplacer le Tag pendant la collecte permet d'améliorer la qualité de la mesure. Ensuite, nous avons mis des balises dans des pièces différentes ; elles étaient donc éloignées, et nous avons pu en déduire que les obstacles représentés par les murs empêchaient la communication entre Tag et balises. Les balises ayant un plan de masse et une antenne patch, nous avons essayé plusieurs orientations dans la pièce et en avons déduit que une directivité des ondes peut peut-être influer sur la qualité des relevés. De plus, nous avons essayé de délimiter une zone restreinte à l'aide des balises (l'une étant par exemple positionnée au centre de la pièce), et en avons déduit que sortir de cette zone avec le Tag donnait une qualité moins bonne. Enfin, nous avons essayé des collectes avec un temps plus long de 5 minutes : bien que notre tuteur avait dit que le temps influait sur la précision de la collecte, nous avons eu des résultats moins performants avec une collecte plus longue. Nous pouvons supposer alors que le temps n'est pas le seul critère qui fait varier la qualité ; cette hypothèse n'étant pas vérifiable étant donné que nous ne pouvons pas accéder à l'algorithme. Par ailleurs, nous avons pensé à effectuer des tests en bougeant les balises (ce qui est théoriquement possible sous certaines conditions), mais ne les avons pas réalisés étant donné que l'algorithme fourni ne nous renvoie qu'une position unique pour chaque balise.

Travail supplémentaire effectué : analyse des paramètres caractéristiques influant sur la précision du résultat de l'algorithme

Nous nous sommes intéressés aux résultats que nous rendait le serveur de localisation. Nous ne voulions pas bêtement afficher ces résultats mais pouvoir aussi avoir un avis critique dessus et réfléchir à comment l'améliorer. Etant donné que nous ne pouvions pas voir l'algorithme utilisé, nous avons fait de multiples tests pour connaitre et cibles les bonnes conditions d'utilisation afin d'obtenir le résultat le plus correct.

Nous en avons déduit plusieurs choses :

- Il est obligatoire que les balises ne soient pas toutes à moins d'un mètre les unes des autres sinon les valeurs retournées ne seront pas du tout cohérentes.
- Il est préférable que les balises soient donc relativement éloignées et qu'elles ne soient pas sur le même plan de façon à avoir une représentation spatiale la plus complète.
- Plus le temps de mesures est long et donc la quantité de mesure récupérée est grande plus le coefficient de qualité est petit et donc la précision accrue.

- Les antennes utilisées sont des antennes patch. Il nous semble que les cartes reliées à ces antennes contiennent un plan de masse. Ces antennes sont donc directives. Il vaut donc mieux éviter de les plaquer vers un mur dans un mauvais sens sinon les mesures risques d'êtres faussées.

Approfondissement et amélioration

Nous n'avons pas mis en place de solution pour l'utilisation de notre serveur Flask de façon asynchrone par manque de temps. Nous avons préféré cibler d'autre points plus pertinents comme les tests de fonctionnement et l'interprétation des résultats.

Toutefois nous avons tout de même cherché des solutions qui pourraient être exploitées afin de palier le problème rencontré. Nous avons découvert "Celery" qui est un gestionnaire de queue asynchrone. Cette librairie python devrait pouvoir être capable de nous aider à lancer la tache de collecte de façon asynchrone au sein de notre serveur, laissant ainsi nos autres taches courtes se dérouler normalement. Il serait nécessaire d'utiliser un broker en parallèle (comme rabbitmq) qui nous servirai de queue où se trouverai les taches en attente.

D'autre part, nous pourrions améliorer la manipulation (stockage, transmission) de fichier sur l'interface Web. Par exemple, il serait important d'archiver le fichier log de résultat en sortie du serveur allgo également ainsi que sauvegarder (exporter) l'image 3D avec plotly dans l'archive.

Comme on peut le voir précédemment sur les visualisations, la continuité des points peut s'avérer mauvaise sur certains passages. En effet, ceci peut être dû à des bruits de mesures, qui influent sur la précision des estimations de distance et produit donc des discontinuités de la position au cours du temps. Une des pistes d'amélioration pour améliorer la qualité de la mesure et donc le rendu, est d'implémenter un filtrage dans l'algorithme même, pour lisser la courbe de position en sortie.

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
Avant fin Décembre	Interface logicielle	
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. L'affichage des données via une matrice 3D n'a pas été trop fastidieuse. Nous avons utilisé la bibliothèque prévue au départ (plotly.js). Apres avoir trouvé le graphique qui nous intéressait et comment l'utiliser, nous avons pu rapidement afficher les valeurs désirées.

Pour conclure sur le calendrier prévisionnel, nous pouvons dire que la seconde partie du projet s'est déroulée sans replanification nécessaire, bien que l'intégration du serveur de localisation se soit faite tardivement car son accès ne nous a été fourni qu'à la toute fin de notre projet.

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.

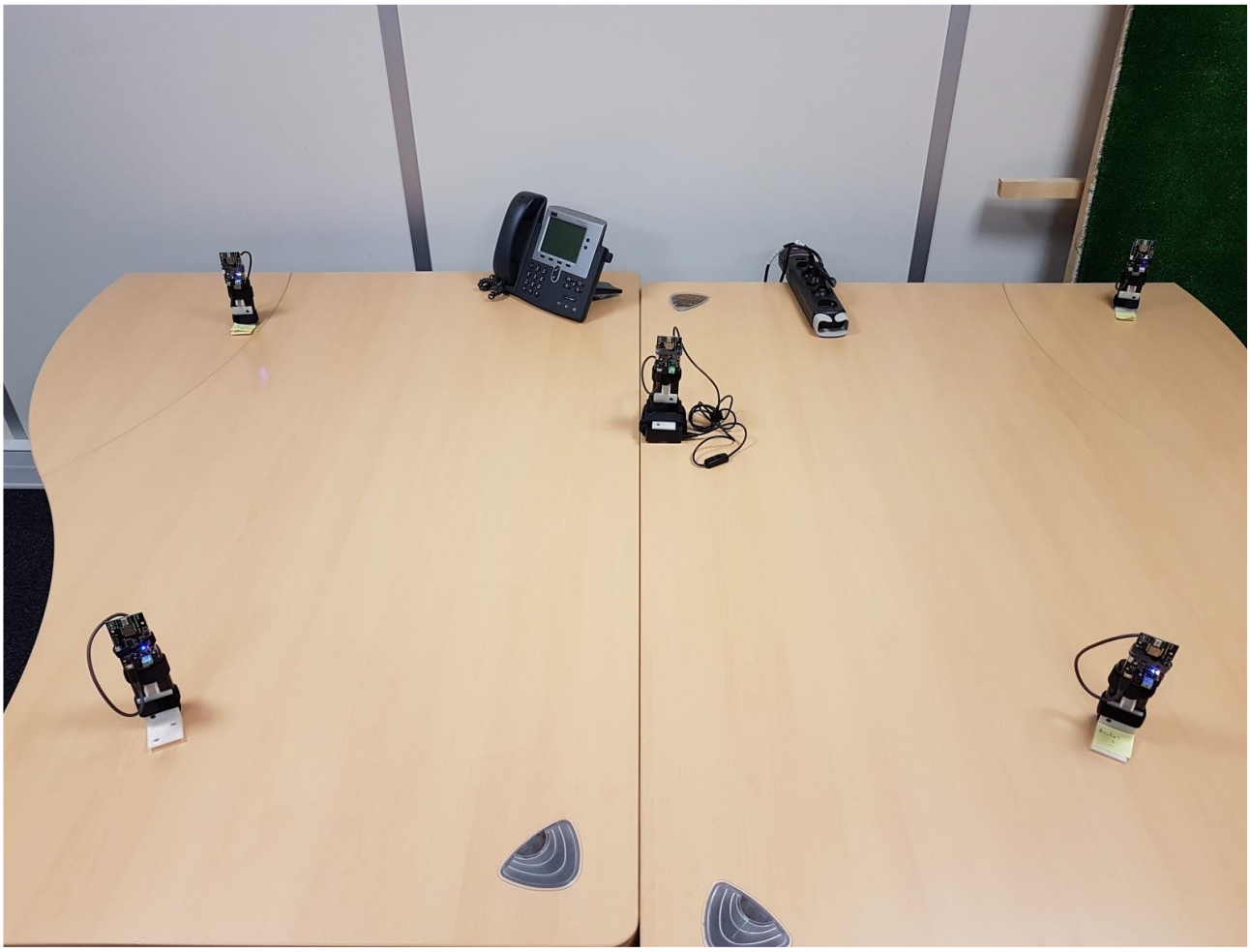
Possédant une grande autonomie, nous avons dû nous confronter à plusieurs problématiques comme la gestion du temps et des échéances. Le travail étant en binôme, nous avons donc dû apprendre à concevoir un logiciel stable en travaillant sur plusieurs features en parallèle.

Apres avoir réussi à obtenir les premiers résultats de visualisation, nous pouvons confirmer le fonctionnements de l'algorithme proposé par le laboratoire. Il est compliqué pour nous de cibler les limites de celui-ci car nous ne pouvons l'étudier. Par contre nous pouvons cibler les limites de notre mise en pratique. Dans l'ensemble, les technologies choisies étaient cohérentes même si il serait envisageable de changer le serveur flask par une technologie moins manuelle. Les capacités du matériel fourni ne sont pas infinies non plus. Nous ne pouvons pas mettre plus de 6 balises ce qui peut être une mauvaise chose suivant l'application.

A l'heure actuelle, le serveur de localisation n'est pour nous pas encore assez précis et le système de communication n'est pas encore assez stable pour permettre une future industrialisation. Il faudrait de plus réellement réfléchir aux applications concrètes qui exploiteraient ce genre de système.

Annexes

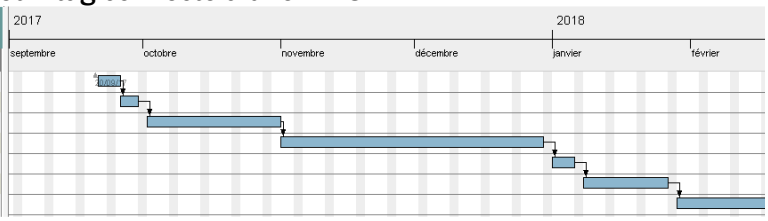
MATERIEL ET PLANNING



Système Loco Positioning

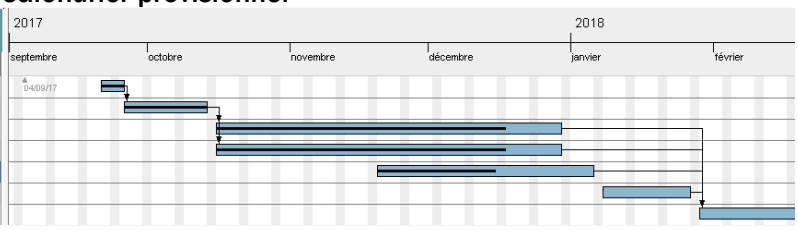
4 anchors et 1 tag connecté à une RPi3

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



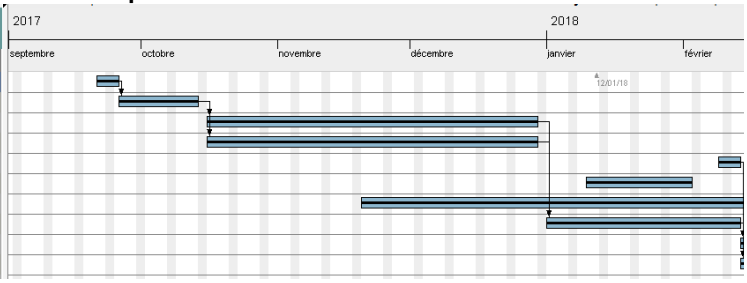
Calendrier prévisionnel

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

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
réalisations des liens avec le serveur d...	09/02/18	13/02/18
refonte du systeme de parsage	10/01/18	02/02/18
débuggage/optimisation/modification	20/11/17	16/02/18
intégration des différents modules	01/01/18	13/02/18
étude de la partie temps réel	14/02/18	16/02/18
Tests	14/02/18	16/02/18



Echéances réelles