PROJET TUTEURE

Session 2015/2016

Drones et véhicules terrestres : évaluation des plateformes, commande, navigation et cartographie

Tuteur : Mr. Adrien REVAULT D'ALLONNES

Formation : Master Informatique 1ère année

Étudiant : DEVARADJOU Stéphane – AURAY-LORIVAL Joffrey

Numéros Étudiant : 14502019 - 14507468

SOMMAIRE

INTRODUCTION SUR LE PROJET

Le projet tutoré se déroule durant l'année scolaire 2015/2016.
L'intitulé du sujet pour le projet tutoré qu'on a pu obtenir est : « *Drones et véhicules terrestres : évaluation des plateformes, commande, navigation et cartographie* ».
Le tuteur de notre projet est Mr. Adrien REVAULT D'ALLONNES, il est professeur depuis le premier septembre 2013, il est aussi maître de conférences dans l'UFR MITSIC de l'Université de Paris 8. À ce titre, il enseigne dans la licence MIME, en M1 général et en M2 ISE de l'UFR.

Pour ce projet, on dispose d'un AR.DRONE 2 de Parrot, qu'il faut exploiter pour qu'on puisse le piloter à partir du PC avec un logiciel que l'on aura programmé au préalable durant le projet. Ensuite, il faudrait que l'on puisse évoluer ce programme, pour faire du vol autonome sur un espace fermé avec le drone et enfin cartographier cet espace fermé, en détectant et en mémorisant tous les différents obstacles murs et autres.

DEVELOPPEMENT DU PROJET

NODE.JS

Pour débuter l'explication sur le développement du projet, nous allons commencer par justifier notre choix final par rapport à l'utilisation du logiciel NODE.JS. Avant de choisir celui-ci, nous avions plusieurs autres possibilités : le système d'exploitation ROS (UBUNTU), le logiciel AUTOMGEN (WINDOWS XP). Nous ne les avions pas retenues car leur mise en place était trop compliquée pour arriver à un résultat plutôt moyen par rapport à ce qui était attendu.

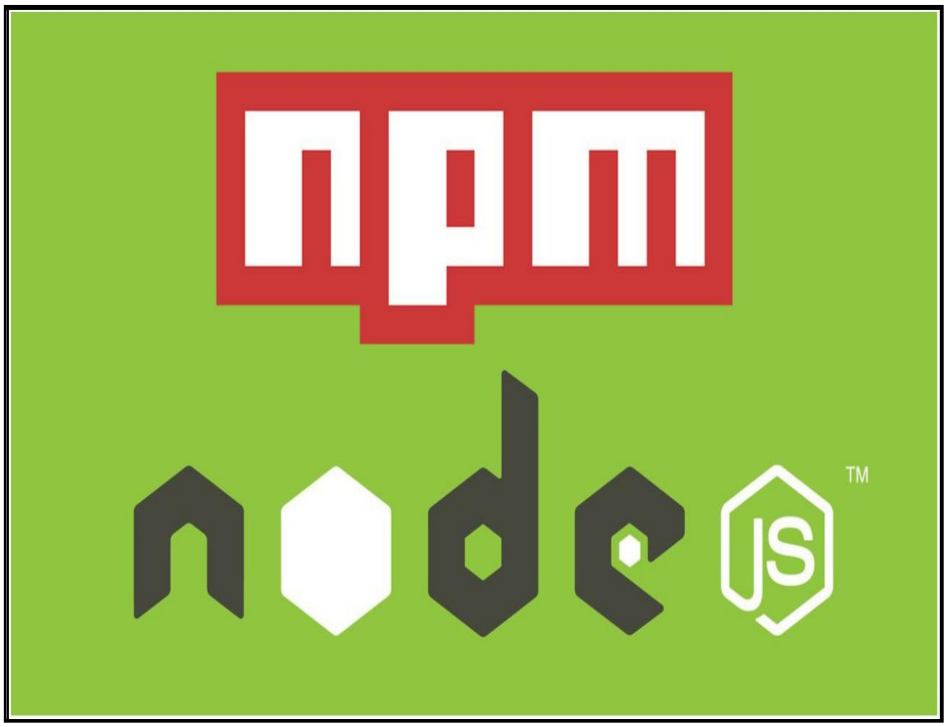
Node.js est un logiciel permettant d'exécuter du JavaScript côté serveur, contrairement à ce qu'on a l'habitude de voir avec le JavaScript côté client.
L'avantage d'utiliser Node.js est que JavaScript permet l'exécution de tâches **asynchrones**, ce qui peut être pratique dans certaines situations. C'est de plus en plus souvent le cas avec le « nouveau » web qui arrive (html5/css3, etc.).
De plus, Node.js permet de créer des applications « serveur » facilement grâce à des applications tierces qu'il prend en charge via un logiciel similaire à un gestionnaire de paquets.

Après quelques recherches, nous sommes tombés sur un tutoriel expliquant comment mettre en œuvre un AR.Drone 2 et pouvoir le piloter avec Node.js. Pour faire les premiers tests, on connecte le PC à la connexion Wifi du drone. Ensuite, on a plusieurs possibilités : avec la console de Node.js on peut rentrer les lignes de code une par une et le drone réagira en fonction de celles-ci ; sinon il est possible de créer un fichier .js et de le lancer à partir de l'invite de commande de Windows.

Nous sommes donc partis d'un exemple simple de chat en TCP/IP sur Node.js et nous avons adapté cette solution d'envoi et de réception de simples messages pour exécuter des commandes qui pilotent le drone. Le serveur est en mode écoute pour pouvoir exécuter les ordres donnés par le client sous forme de messages, qui seront ensuite traduits par le serveur en tant que commande de pilotage. En fonction du message reçu, le serveur exécute une action (commande) sur le drone. Ensuite, avec l'aide d'un projet de pilotage similaire au notre fait par Rohit Ghatol, nous avons regardé comment nous pouvions récupérer le flux vidéo de la caméra frontale du drone. Ces deux projets ont vraiment posés de bonnes bases pour notre projet.

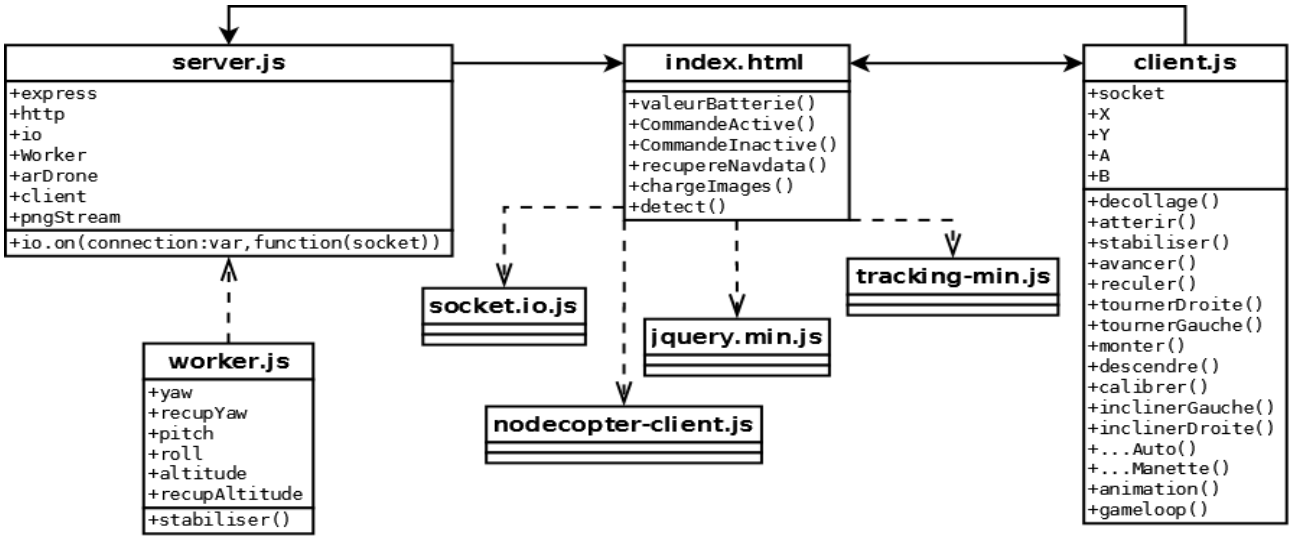
L'avantage de Node.js par rapport aux autres solutions que nous avions exploré, c'est qu'il y une vraie base de données complète donnant accès à différents modules existants appelée NPM (Node Package Manager). Sur cette plate-forme, on peut y voir une communauté très active. Il faut tout de même vérifier les dates de publications des modules car comme le JavaScript est un langage qui évolue très rapidement, il se peut que certains modules deviennent obsolètes d'une année à l'autre si l'utilisateur ne s'en occupe pas. Les modules peuvent être téléchargés puis installés sur NodeJS avec la commande "npm install".

Nous avons eu divers problèmes suite à cette précision, mais nous les avons comblés en utilisant à chaque fois des modules adaptés pour le drone (AR Drone 2) et aussi par rapport aux objectifs visés.



STRUCTURE DES FICHIERS DU PROJET

Nous allons présenter ici tous les fichiers que nous avons crées et utilisés pour mener à bien notre projet.



server.js : Ce fichier que nous avons crée est le fichier principal du projet. C'est le fichier qui contient le code serveur sur le drone, c'est celui-ci que l'on doit déployer/installer sur le drone avec l'invité de commande de Windows avec la commande Node pour ensuite piloter le drone.
A son déploiement, ce fichier se place sur le drone et attend les messages envoyés par le client, et pour que tout fonctionne normalement il appelle au préalable certains modules de node.js nécessaires comme 'ar-drone' pour le drone, 'ar-drone-png-stream' pour recevoir le flux vidéo de la camera, 'socket.io' pour pouvoir communiquer en TCP/IP et d'autres.
Il a pour fonction d'exécuter les commandes sur le drone.

worker.js : Après avoir fait quelques recherches sur les threads (tâche en parallèle) en JavaScript, nous avons trouvé la solution d'utiliser un Webworker (thread en JavaScript). Pour cela, nous avons créé un fichier "worker.js" qui s'exécute à partir du serveur, le serveur le lance ou l'arrête dès qu'il en a besoin, après un événement par exemple. Nous avons crée ce fichier car nous voulions développer un code de stabilisation pour le drone qui s'exécuterait parallèlement au serveur. Son code tourne en tâche de fond et s'exécute seulement si le serveur reçoit l'évènement attendu, sinon il est arrêter. Dans le code on récupère les coordonnées de rotation sur les axes du drone (yaw en Z, pitch en Y, roll en X) et l'altitude, ensuite on a mis des intervalles de valeurs (Xmin, Xmax, Ymin, Ymax, Zmin, Zmax) que le drone ne doit pas dépasser durant sa phase de stabilisation, s'il les dépasse on envoie la commande correspondante pour corriger sa stabilisation. Par exemple, si le drone se met trop en avant, ce qui modifie sa valeur de pitch, on lui fait faire une commande de recul pour corriger le mouvement.

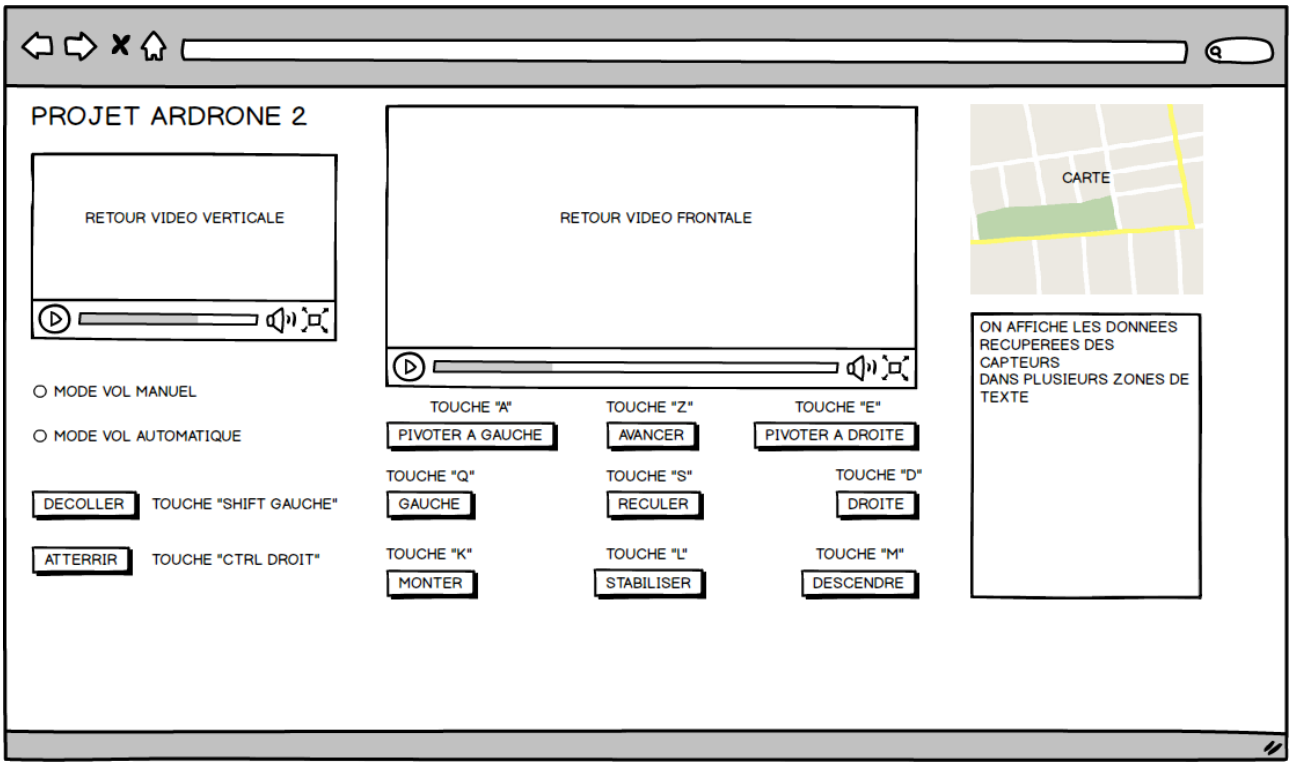
client.js : Ce fichier contient tous les messages à envoyer sur le serveur donc le drone pour pouvoir le piloter. Il est actif sur les différents événements qui peuvent se produire sur la page principale de notre interface, comme l'utilisation des boutons de l'interface au clic ou au clavier par exemple, ou bien l'activation du mode vol automatique qui va permettre d'envoyer des messages au drone automatiquement. Il contient aussi le code permettant d'utiliser la manette de Xbox 360 pour pouvoir contrôler le drone. C'est une passerelle entre le fichier "server.js" et le fichier "index.html".

index.html : Ce fichier est le fichier principal du projet pour son fonctionnement externe. C'est le fichier qui contient tout le code HTML de l'interface ainsi que son code CSS pour changer l'apparence des composants dessus. Il comporte aussi les scripts essentiels pour récupérer les informations de navigation du drone et la valeur de la batterie, pour afficher, charger et sauvegarder les images récupérées par la camera du drone. Mais aussi le script pour faire de la détection des couleurs sur les cibles jaunes. Tous ces scripts utilisent des fichiers JavaScript qui sont appelés au début du code HTML, **socket.io.js** pour la communication client/serveur, **nodecopter-client.js** pour la commande du drone, **tracking-min.js** pour utiliser la détection de couleurs sur le flux video.

PRESENTATION DE L'INTERFACE

Cette partie va présenter l'interface de commande qui a été produite pour piloter le drone à distance.

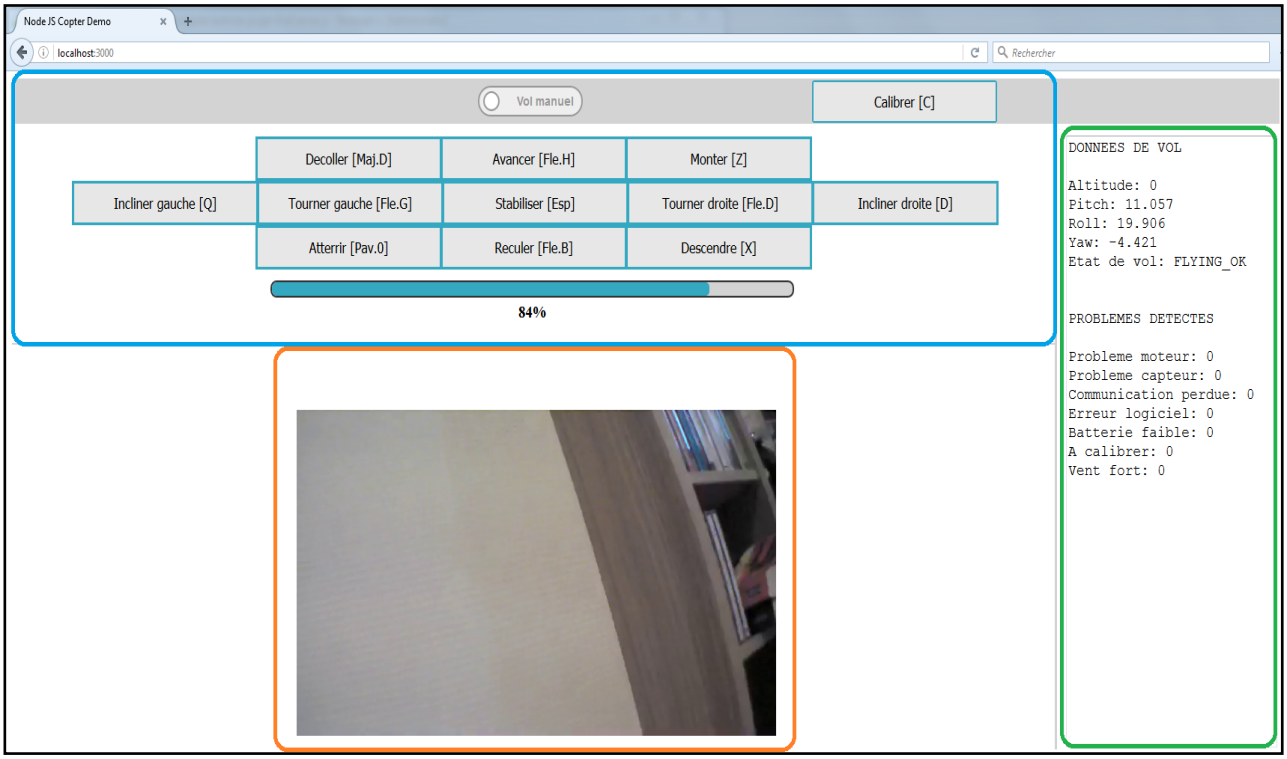
Interface théorique



Voici ci-dessus, l'interface de pilotage qui a été imaginée et dessinée sur le logiciel Balsamiq Mockups 3 au départ. Elle a été faite pour avoir une idée de ce à quoi pourrait ressemblé l'interface finale avec toutes les fonctionnalités développées et implémentées en une seule interface. Donc, cette interface comporte :

- les différents boutons pour contrôler le drone,
- les boutons radio pour changer le mode de vol,
- le retour vidéo des deux caméras (frontale, verticale),
- la zone pour afficher les valeurs des capteurs du drone,
- la carte de la zone explorée.

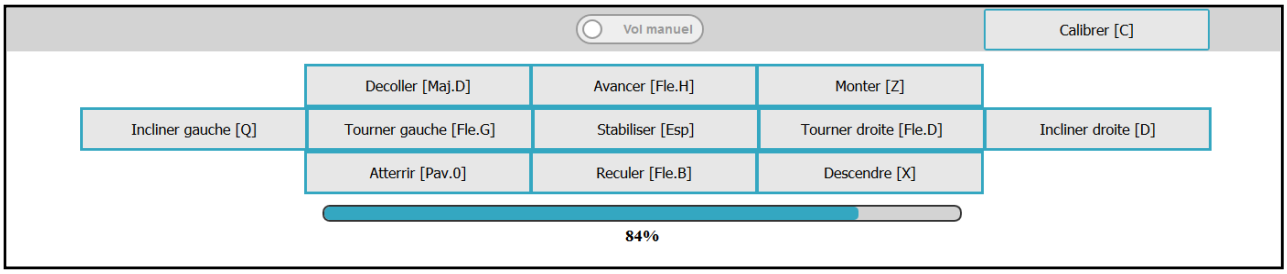
Interface finale



Voici ci-dessus, l'interface de pilotage qui a été produite finalement pour ce projet. C'est une page HTML, mise en forme avec du CSS et avec plusieurs scripts en JavaScript pour gérer par exemple les événements de clic sur les boutons ou la récupération des différentes données. Sur cette image on a encadré cette interface pour l'expliquer et elle se divise en trois grandes parties :

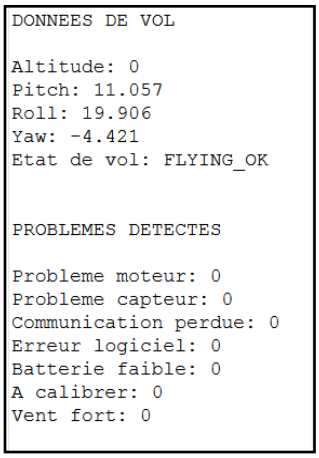
- **la zone orange** avec le retour de la vidéo frontale et l'affichage de la distance entre la cible et le drone qui est active seulement en mode de vol autonome.
- **la zone bleue** avec les boutons pour contrôler le drone, l'interrupteur pour changer de mode de vol, le pourcentage de batterie restante.
- **la zone verte** avec la zone de texte montrant les différentes données récupérées des capteurs du drone.

La zone Bleue



Cette zone est la partie centrale de notre interface car on a les boutons de pilotage, qui peuvent être activé avec le clic de la souris ou en pressant le bouton du clavier correspondant à l'action ou en utilisant la manette de jeu. On peut voir une barre représentant le niveau de la batterie du drone, et l'interrupteur pour changer le mode de vol. Quand on active le mode automatique et qu'on place le drone devant une cible, il va la détecter et afficher un cadre jaune sur la vidéo ainsi que la distance pour atteindre la cible.

La zone Verte



Dans cette zone, on peut observer en temps réel le positionnement du drone par rapport au sol avec les coordonnées de pitch, roll et yaw, son altitude et son état de vol.
On peut également voir les divers problèmes qui peuvent survenir sur le drone, comme un problème sur les moteurs ou les capteurs, un problème de calibration ou la signalisation de la batterie faible (capacité inférieure à 21%).

COMMANDES

Nous avons utilisé un module appelé 'ar-drone' et développé par Felixge, un utilisateur de Github qui l'a posté sur la bibliothèque en ligne NPM. Il permet de piloter le drone ainsi que d'intercepter certains événements et de récupérer des données. Voici les commandes principales que nous avons utilisées :

arDrone.createClient([options])
Renvoie un nouvel objet client. options incluent:
ip: L'IP du drone. Par défaut '192.168.1.1'.
FrameRate: Le taux de la PNGEncoder de trame. Par défaut, 5.
ImageSize: La taille de l'image produite par PNGEncoder. Null par défaut.

client.getPngStream()
Renvoie un objet PNGEncoder qui émet des tampons d'image png individuels comme des événements 'data'. Plusieurs appels à cette méthode retourne le même objet. Le cycle de vie de connexion (par exemple la reconnexion en cas d'erreur) est géré par le client.

client.getVideoStream()
Renvoie un objet TcpVideoStream qui émet des paquets TCP comme des événements 'data'. Plusieurs appels à cette méthode retournent le même objet. Le cycle de vie de connexion (par exemple la reconnexion en cas d'erreur) est géré par le client.

client.takeoff(callback)
Définit l'état de vol interne à true, callback est invoquée après que le drone signale qu'il est en vol stationnaire.

client.land(callback)
Définit l'état de vol interne à false, callback est invoquée après que le drone rapporte qu'il a atterri.

client.up(speed) / client.down(speed)
Fait gagner ou réduire de l'altitude au drone. La vitesse peut être une valeur comprise entre 0 et 1.

client.clockwise(speed) / client.counterClockwise(speed)
Provoque la rotation du drone. La vitesse peut être une valeur comprise entre 0 et 1.

client.front(speed) / client.back(speed)
Contrôle du pitch en Y, où un mouvement horizontal en utilisant l'appareil photo comme un point de référence. La vitesse peut être une valeur comprise entre 0 et 1.

client.left(speed) / client.right(speed)
Contrôle du roll en X, qui est un mouvement horizontal à l'aide de l'appareil photo comme point de référence. La vitesse peut être une valeur comprise entre 0 et 1.

client.stop()
Définit tous les mouvements du drone commandes à 0, ce qui rend le vol stationnaire effectivement en place.

client.calibrate(device_num)
Demande au drone de calibrer un dispositif. Actuellement, le firmware AR.Drone ne supporte qu'un seul appareil qui peut être calibré: le magnétomètre, qui est le numéro de périphérique 0.
Le magnétomètre ne peut être étalonné tandis que le drone vole, et la routine d'étalonnage provoque la rotation sur place du drone à 360 degrés.

client.config(key, value, callback)
Envoie une commande de configuration pour le drone.
Par exemple, cette commande peut être utilisée pour charger le drone d'envoyer tous les Navdata.
client.config («général: navdata_demo ',' FALSE ');

client.animate(animation, duration)
Effectue une séquence de vol préprogrammée pour une durée donnée (en ms).
animation peut être l'un des éléments suivants:

['phiM30Deg', 'phi30Deg', 'thetaM30Deg', 'theta30Deg', 'theta20degYaw200deg', 'theta20degYawM200deg', 'turnaround', 'turnaroundGodown', 'yawShake', 'yawDance', 'phiDance', 'thetaDance', 'vzDance', 'wave', 'phiThetaMixed', 'doublePhiThetaMixed', 'flipAhead', 'flipBehind', 'flipLeft', 'flipRight']

PERIPHERIQUES DE CONTROLE

Sur cette partie, nous allons présenter les trois périphériques de contrôle utilisés pour faire voler le drone.

La souris et le clavier : au début quand on a commencé à créer l'interface, nous avons placés des boutons sur la page et nous devions cliquer dessus pour déclencher l'événement de décollage par exemple. Afin de faciliter le contrôle, nous avons utilisé les touches du clavier, en ajoutant donc une partie de code comportant les valeurs ASCII de toutes les touches souhaitées. Les valeurs des commandes envoyées pour la souris et le clavier sont les mêmes.

Décoller = Shift Droit ; Stabiliser = Espace ; Atterrir = 0 pavé numérique ;
Touches fléchées = Avancer, Reculer, Tourner à droite/gauche ; Calibrer = C ;
Z = Monter; X = Descendre ; Q = Incliner à gauche ; D = Incliner à droite ;

La manette de jeu (XBOX 360) : par la suite, toujours avec le même objectif de faciliter le pilotage du drone, nous avons voulu utiliser une manette de jeu pour rendre le drone plus réactif aux actions demandées. Grâce à un module NodeJS, qui a été développé pour pouvoir utiliser une manette sur une interface web (page HTML), nous avons programmé les touches et les joysticks. Chaque mouvement du drone contrôlé avec la manette le rendait plus sensible et bien trop rapide pour ce qui était exécuté. Donc il a fallu que l'on optimise les valeurs de toutes les commandes envoyées avec la manette, pour que cela puisse correspondre à cette vitesse d'envoi.

Décoller = A ; Stabiliser = X ;
Calibrer = Y ; Atterrir = B ;
Déplacements = Joystick Gauche ;
Rotations = Joystick Droit ;
LT = Descendre ; RT = Monter ;

LA COMMUNICATION

- **Server.js (Exécution côté serveur)**

C'est dans ce fichier que nous créons et configurons (port) le serveur ainsi que le socket qui nous permet de communiquer (envoyer / recevoir) avec l'ensemble du projet.
La communication se fait par l'intermédiaire de messages.

Pour cela, nous utilisons différents modules récupérés sur NPM: "express" qui est un framework qui initialise la communication, "http" qui nous permet de configurer le serveur et "socket.io" qui intercepte différents événements comme par exemple ci-dessous dès lors qu'un utilisateur va sur le site Web (http://localhost:3000)
io.on() nous permet d'attendre l'événement / le message ('connection' dans l'exemple qui suit).

```
var express = require('express');
var app = express();
var http = require('http').Server(app);
var io = require('socket.io')(http);
// On gère les requêtes HTTP des utilisateurs en leur renvoyant les fichiers du dossier 'public'
app.use("/", express.static(__dirname + "/public"));


// On lance le serveur en écoutant les connexions arrivant sur le port 3000
http.listen(3000, function(){
  console.log('Server is listening on *:3000');
});

var callback = function(err) { if (err) console.log(err); };

io.on('connection', function(socket){
      console.log('utilisateur connecte');
}
```

Les fichiers du sous-dossier "public" sont notamment "index.html" et "client.js". La variable "app" nous permet d'associer la page index.html au serveur. Ainsi, les utilisateurs qui se connectent sont automatiquement sur cette page.

Nous avons vu comment attendre / recevoir un message, nous allons maintenant voir comment nous en émettons un. Une ligne suffit:

```
socket.emit('event', { name: 'battery',value: batteryLevel});
```

Ici nous émettons l'événement 'event' et lui passons en paramètre un nom et une valeur associé à ce nom. Le passage de paramètres est facultatif. Il est donc possible d'envoyer et d'intercepter plusieurs données dans un même événement, nous verrons cela par la suite.

Le fichier server.js reçoit les événements émis par le fichier "client.js" mais lui, envoie ses événements directement au fichier "index.html"

- **Client.js (Exécution côté client)**

Ce fichier sert de passerelle entre le fichier "server.js" (serveur) et le fichier "index.html" (client). Il est inclus dans le fichier "index.html", ainsi il est possible d'appeler directement ses fonctions depuis ce-dernier.

Toutefois, il se sert du socket pour communiquer avec le fichier "server.js". Il ne fait qu'envoyer des messages / événements (notre développement fait que nous n'avons pas besoin d'en recevoir).
Le serveur lui, les reçoit et selon le message effectue la commande associée.

L'initialisation ici, est plus simple à mettre en place:

```
var socket = io();

function decollage() {
    socket.emit('decoller');
}
```

- **Index.html (Exécution côté client)**

Ce fichier représente notre client. Il inclut les fichiers "server.js" et "client.js" ainsi que toutes les dépendances nécessaires au bon fonctionnement de la communication:

```
<script src="/socket.io/socket.io.js"></script>
<script src="client.js"></script>
<script src="server.js"></script>
```

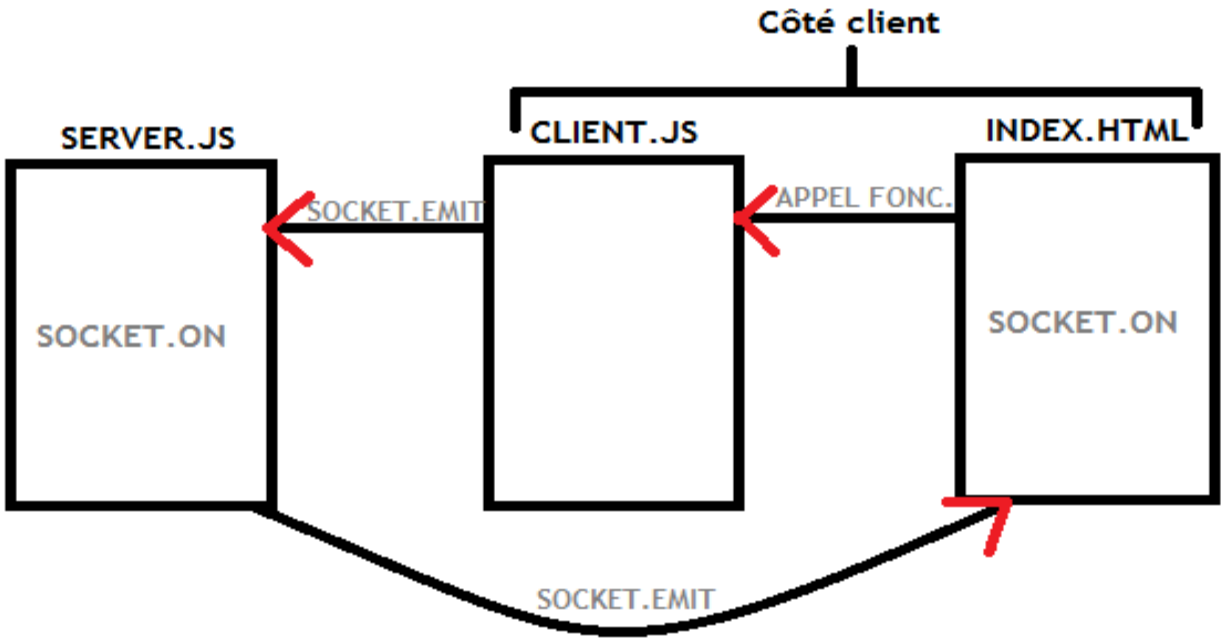
Il est donc possible d'appeler directement des fonctions du fichier "client.js".
Il ne fait que recevoir des messages / événements provenant du fichier "server.js" (c'est par le fichier "client.js" que se font tous les envois).

```
socket.on('EnvoiNavdata', function (data) {
    if(data.name=="altitude"){
        altitude=data.value;
    }
    if(data.name=="pitch"){
        pitch=data.value;
    }
});
```

Ici il est en attente de l'événement 'event' (socket.on()). Dès qu'il le reçoit, il récupère les paramètres (facultatifs) passés à l'événement.

- **Schéma de communication**

  Pour résumer les différents échanges qui se font, voici un graphe :



Récupération des données

L'AR.Drone 2 est doté de nombreux capteurs: altitude, gyromètre, accéléromètre, GPS, ... et il peut nous être utile de savoir ce que ces capteurs nous retourne comme valeurs. Nous avons donc récupéré les données (Navdata) principales de ces capteurs (yaw, pitch, roll, altitude, problèmes éventuels).

De plus, les capteurs ne sont pas les seules données qui puissent être récupérées. Comme vu dans la présentation, le drone dispose de deux caméra, une orientée vers l'avant, une vers le bas. Il existe donc un flux vidéo, nous le récupérons aussi. Pour ce projet nous utilisons seulement celle de l'avant.

- **Navdata**

Pour récupérer les données qui nous intéressent, nous utilisons le module ar-drone dont nous avons parlé plus haut.

Celui-ci en plus des commandes, gère directement la réception des données côté serveur (server.js) par cet événement:

```
client.on('navdata', function(data) {
    if (!data.demo) return;
    altitude=data.demo.altitudeMeters;
    pitch = data.demo.rotation.pitch;
    roll  = data.demo.rotation.roll;
    yaw   = data.demo.rotation.yaw;
});
```

Nous envoyons ensuite ces différentes variables au client (index.html) par l'intermédiaire du socket afin de les afficher sur l'interface. Dans notre code nous avons mis une temporisation avant l'envoi de certaines Navdata car c'est trop rapide et prend beaucoup de ressources.

- **Flux vidéo**

Le flux vidéo est en réalité une succession très rapide d'images (30 images par seconde en général).
C'est à partir de ce principe que nous avons récupéré côté serveur un flux d'images formant une vidéo.

```
var arDrone = require('ar-drone');
var client = arDrone.createClient();
require('ar-drone-png-stream')(client, { port: 3001 });
var fs = require('fs');
var pngStream = client.getPngStream();
var period = 90; // Save a frame every 90 ms.
var lastFrameTime = 0;

pngStream
  .on('error', console.log)
  .on('data', function(pngBuffer) {
    var now = (new Date()).getTime();
    if (now - lastFrameTime > period) {
      lastFrameTime = now;
      fs.writeFile('cheminimage' + '.png', pngBuffer, function(err) {
        if (err) {
          console.log('Erreur sauvegarde PNG: ' + err);
        }
      });
    }
  });
```

Pour récupérer cette vidéo, nous avons besoin des modules "ar-drone" et "ar-drone-png-stream".

Le flux d'image est configuré sur le port 3001 (port 3000 déjà utilisé par le serveur).
Toutes les 90ms, l'image du flux est sauvegardée sur le PC et affichée côté client sur l'interface.
Ce délai nous est impératif car l'ensemble du projet prend beaucoup de ressources (Navdata, gestion / attente des commandes, détection,..). Toutefois cela reste plutôt fluide et l'ensemble est fiable.

- **Détection**

Afin d'implémenter la fonctionnalité de vol autonome, il nous à fallu réfléchir à comment mettre en place ce mode de vol et voir si il existait des librairies pouvant nous aider.

C'est le cas. Après de nombreuses recherches, nous avons pris connaissance de la librairie tracking.js.
Comme son nom l'indique cette librairie permet de suivre (détecter) des couleurs, visages, dans une vidéo ou une image.
Nous étions parti sur la détection de couleur dans une vidéo mais n'avons pas pu car la librairie détecte dans une balise <video>, hors, le flux vidéo que nous avions récupéré (avant de partir sur un flux d'images) grâce au module NPM, dessinait la vidéo dans une balise <canvas>, la détection ne marchait donc pas.

Actuellement, nous récupérons donc une image du flux d'images toutes les 90ms dans une balise <img> et c'est sur cette image que nous réalisons la détection.
Nous avons choisi de détecter la couleur jaune.
Tout cela se fait intégralement en JavaScript et côté client dans le fichier "index.html".

<u>La détection se fait en plusieurs étapes:</u>

-On paramètre la détection (couleur(s) à détecter)

```
var tracker = new tracking.ColorTracker(['yellow']); //detection de la couleur jaune
```

-On supprime les anciens rectangles de détection si il y'en a

```
while(elements.length > 0){ //tant qu'il y'a des rectangles
    elements[0].parentNode.removeChild(elements[0]);
}
```

-On passe l'image à traiter

```
tracking.track('#img', tracker);
```

-Pour chaque élément détecté, un rectangle de détection est dessiné autour.

```
window.plot(rect.x, rect.y, rect.width, rect.height, rect.color);
```

Une fois la couleur souhaitée détectée, nous avons réfléchis à l'algorithme nous permettant d'approcher le drone de cet élément de manière totalement autonome.

Pour cela, nous nous basons sur le rectangle dessiné.
En connaissant la résolution du flux d'image (...x...), nous pouvons savoir si le rectangle est centré dans l'image ou pas.

En partant de ce principe, si le rectangle est à peu près dessiné au centre de l'image, cela veut dire que le drone est bien aligné en face de l'élément.

Sinon, nous regardons la position du rectangle et en fonction, des commandes s'exécutent sur le drone.

Par exemple, l'élément est détecté à droite de la caméra, le rectangle se dessine donc sur la droite. Pour aligner le drone avec l'élément, une commande fait automatiquement tourner le drone sur sa droite.

L'idée étant toujours de garder le rectangle le plus au centre possible.
Dès lors que le rectangle est à peu près centré, le drone avance vers la cible (la couleur jaune).

De plus, à partir du moment où nous connaissons la taille de notre cible (20cm de large), nous avons développé un algorithme permettant d'estimer la distance de la cible en fonction de la largeur du rectangle dessiné lors de la détection.

```
var pixtometre = (rect.width * 0.000264583);
var distanceCible = (0.0393699984 / pixtometre);
var arrondi=Math.round(distanceCible*100)/100;
```

Ces nombres ne sont pas hasardeux puisque nous les avons calculés pour qu'ils correspondent au focus de la caméra avant du drone.
La distance est fiable et nous permet de stabiliser le drone, de le faire atterrir et d'arrêter la détection automatiquement dès que la cible est à 1m60 ou moins de distance.

Tout cela se fait uniquement si le mode "vol autonome" est activé avec le Switch.
Aucune détection ou estimation de distance n'est effectué en mode "vol manuel".

PROBLEMES RENCONTRES ET SOLUTIONS

Dans cette partie, nous allons vous présenter et expliquer les différents problèmes que nous avons rencontré tout au long de ce projet. Nous allons notamment revenir sur quatre problèmes principaux ainsi que les solutions trouvées pour pallier à ceux-ci.

- **MODULES OBSOLETES :** Tout d'abord, on a eu un problème concernant des modules obsolètes. Effectivement, certains modules disponibles sur NPM que ce soit pour commander le drone ou pour réaliser la détection, n'étaient pas à jour. Cela nous a donc valu une grande perte de temps afin de comprendre l'origine des dysfonctionnements de certains de ces modules, qui étaient dût à l'ancienneté de leur code JavaScript et de leurs fonctionnalités.
  Nous avons donc cherché et utilisé des modules plus récents. Toutefois, aucun module alternatif au module OpenCV n'existait sur NPM, nous avons donc utilisé une librairie externe (tracking.js).

- **VIDEO :** La récupération du flux vidéo fut vraiment difficile car il avait été réalisé sur le projet fait par Rohit Ghatol mais les boutons pour contrôler le drone, eux, ne fonctionnaient pas à cause de l'ancienneté de son code.
  Finalement, nous sommes partis d'un projet de tchat TCP/IP en JavaScript. Une fois que l'on avait réussi à faire fonctionner le bouton « décoller » avec ce projet, nous avons effacé toutes les parties obsolètes du projet de Rohit et nous les avons remplacés par nos boutons du tchat afin de bénéficier de leur bon fonctionnement tout en conservant le flux vidéo.

- **DETECTION :** La détection des cibles de couleurs à partir du flux vidéo nous a posé problème au départ car on ne pouvait pas utiliser notre flux vidéo en temps réel. La librairie **tracking.js** que l'on utilise pour faire la détection nécessitant une balise de type <VIDEO> alors que le flux vidéo se dessinait dans une balise de type <CANVAS>. Ce n'était donc pas fonctionnel.
  Pour résoudre cela, nous avons décidé d'utiliser un flux d'images très rapide avec des balises <IMG> (compatible avec tracking.js) pour réaliser la détection.

- **DRONE :** Enfin, le drone lui même nous a posé des problèmes sur la fin du projet car à force de l'utiliser, il a fini par s'user (hélices,...) petit à petit. Les batteries ne tiennent plus aussi longtemps qu'au début, il a du mal à décoller en restant stable, il a tendance à pencher vers la droite même quand il est censé se stabiliser... etc. On a donc crée le fichier **worker.js** pour gérer la stabilisation.

CONCLUSION

Au final nous avons passé six mois sur ce projet sans voir le temps passer.

Sans atteindre tous les objectifs fixés au début, nous avons pourtant appris ou approfondis un certain nombre de notions techniques et surtout nous comprenons mieux comment des systèmes embarqués tels que les drones communiquent avec nous, utilisateurs, ou même parfois entre eux.

Dès le début de ce projet, nous savions déjà que nous ne réaliserions pas la cartographie car cela est certes réalisable mais très ambitieux au vue du temps qui nous à été accordé et du drone qui n'est pas forcément adapté à cet usage. Toutefois, nous ne pensions pas développer et atteindre autant d'objectifs différents.
Nous avons toujours eu la volonté de faire au plus simple tout en réalisant un projet le plus complet et fonctionnel possible.

-Depuis une seule et même page du site Web, il est ainsi possible d'effectuer un vol manuel. Les commandes peuvent être exécutées par les boutons de l'interface, par clavier et par manette de XBOX 360.
-Le mode de vol autonome permet quant à lui de faire voler le drone de façon entièrement autonome en direction d'une cible précise que nous avons défini.
-Une estimation de la distance de la cible est aussi réalisée.
-Nous récupérons en continu les informations (Navdata) principales du drone à savoir son orientation, des problèmes éventuels, ... ainsi que le flux vidéo de sa caméra avant.

Nos recherches et choix nous ont poussés à apprendre puis connaitre des outils de développement (NodeJS, NPM) que jusque là nous ne connaissions pas ou que de nom.

Très impliqués dans ce projet, nous sommes plutôt contents de la solution obtenue.