Conception d’un système de pilotage par l’interprétation des mouvements du corps

ANTHEAUME LUBIN
N°43545
Jeux et sport

Plan

- Cahier des charges du projet
- Choix du dispositif d’acquisition respectant le cahier des charges
- Conception et test du système
- Conclusion

Cahier des charges

Contraintes:

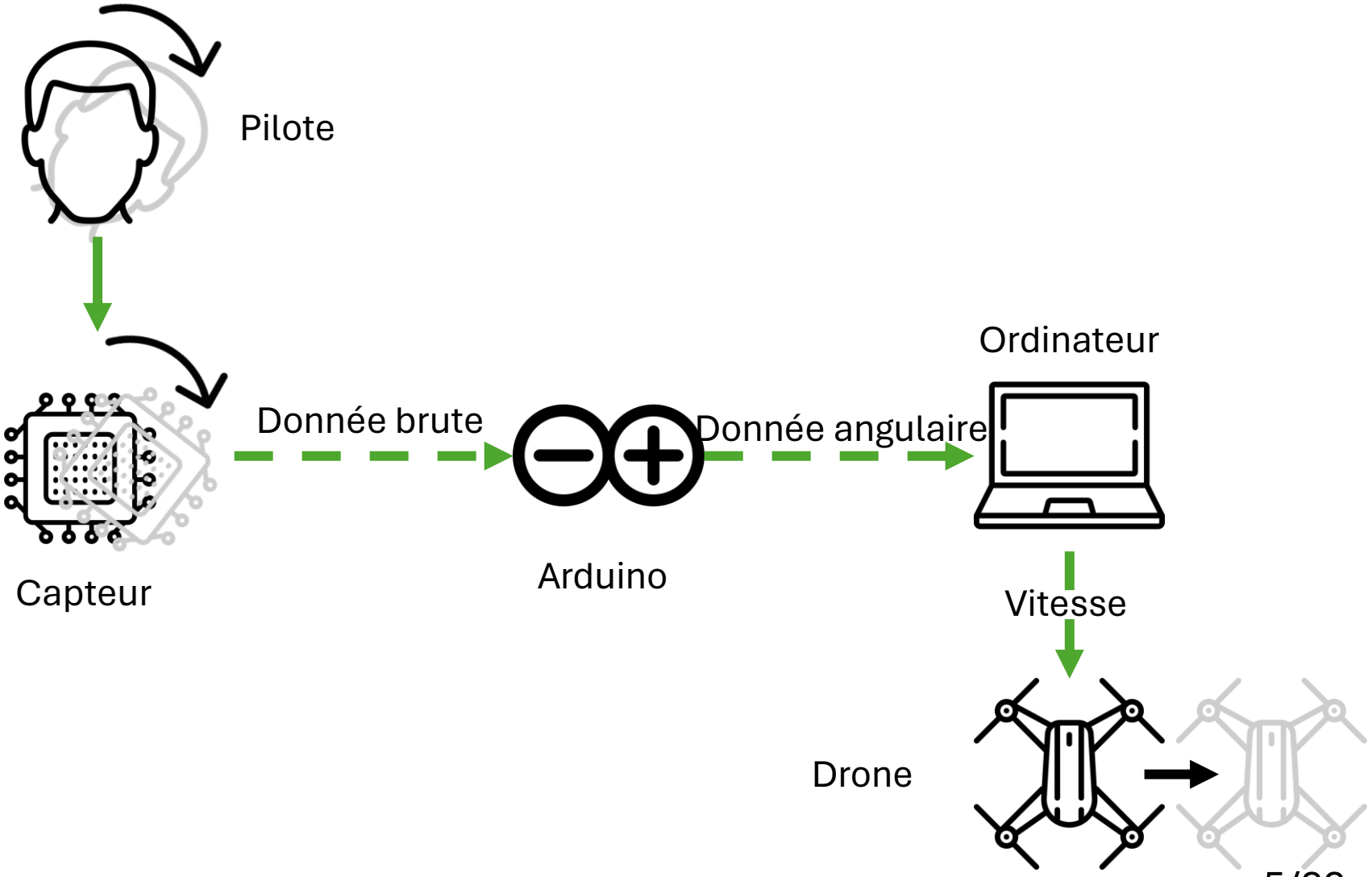
- Coût le plus faible possible et open source
- Facilité de prise en main (solution plug and play dans l'idéal)
- Autonomie
- Facilement transportable
- Immersif

Choix de l'engin volant (imposé) : drone quadricoptère DJI Ryze Tello (€115 dji.com) – programmable en python, connexion en wifi

Comparation dispositifs d'acquisition existants

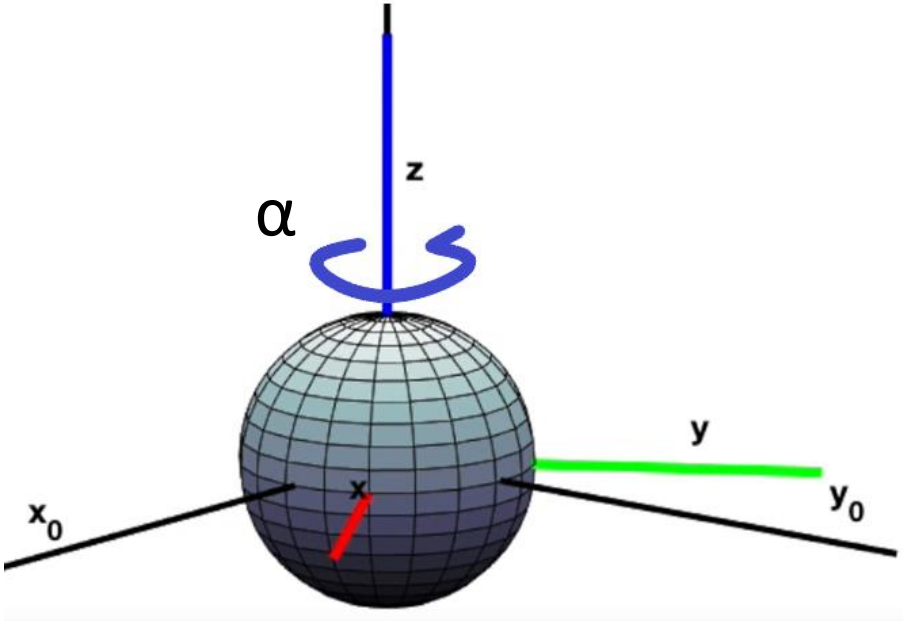
	Coût + matériel	Installation/utilisation	Complexité technique	immersion
Tracking par caméra	-Smartphone/ Caméra (100/150€ prix caméra)	Contraintes de luminosité et de maintien de la caméra	Système de reconnaissance des mouvements par IA	À revoir
Casque VR compatible avec le SDK OpenVR	Casque VR compatible (oculus rift s : 150€ d'occasion)	Aucune	Complexité abordable (pas d'IA)	Excellente (retour VR de la caméra du drone)
Arduino et association de capteurs	Arduino Mega 2560 (42€ officiel; 5~15€ clone) Capteurs (estimé 30€)	Aucune	Complexité simple (pas d'IA et programmation arduino très documentée)	Cardboard VR + smartphone

Solution envisagée

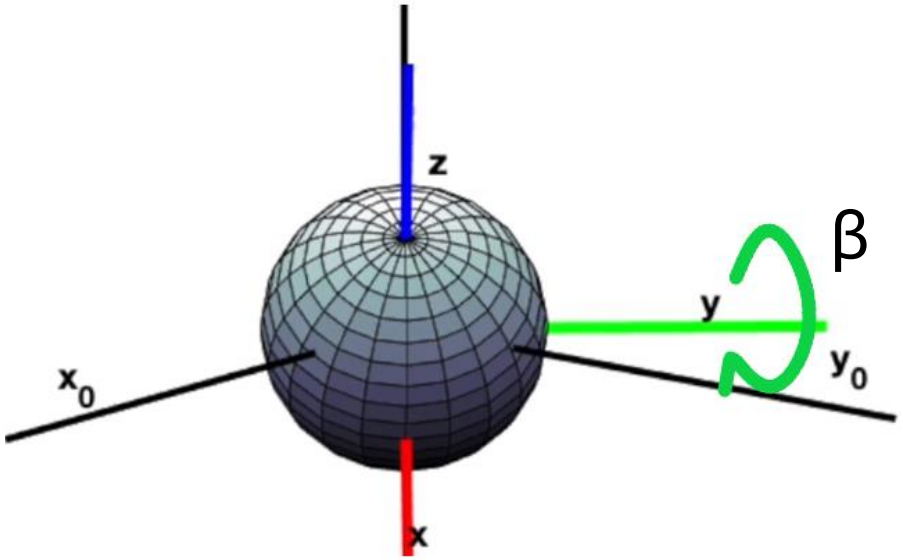


Solution en images

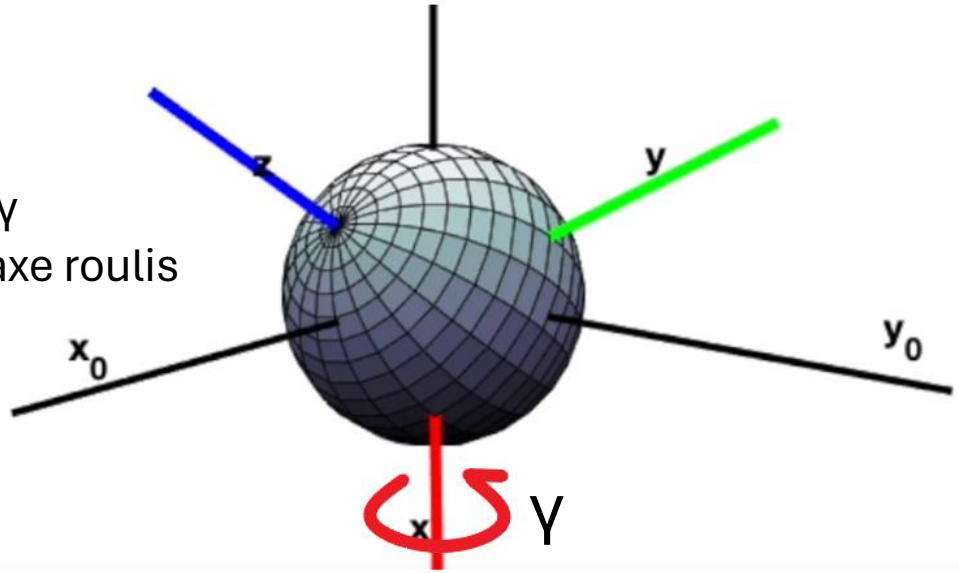
Angle de précession α
⇔ Drone rotation axe lacet



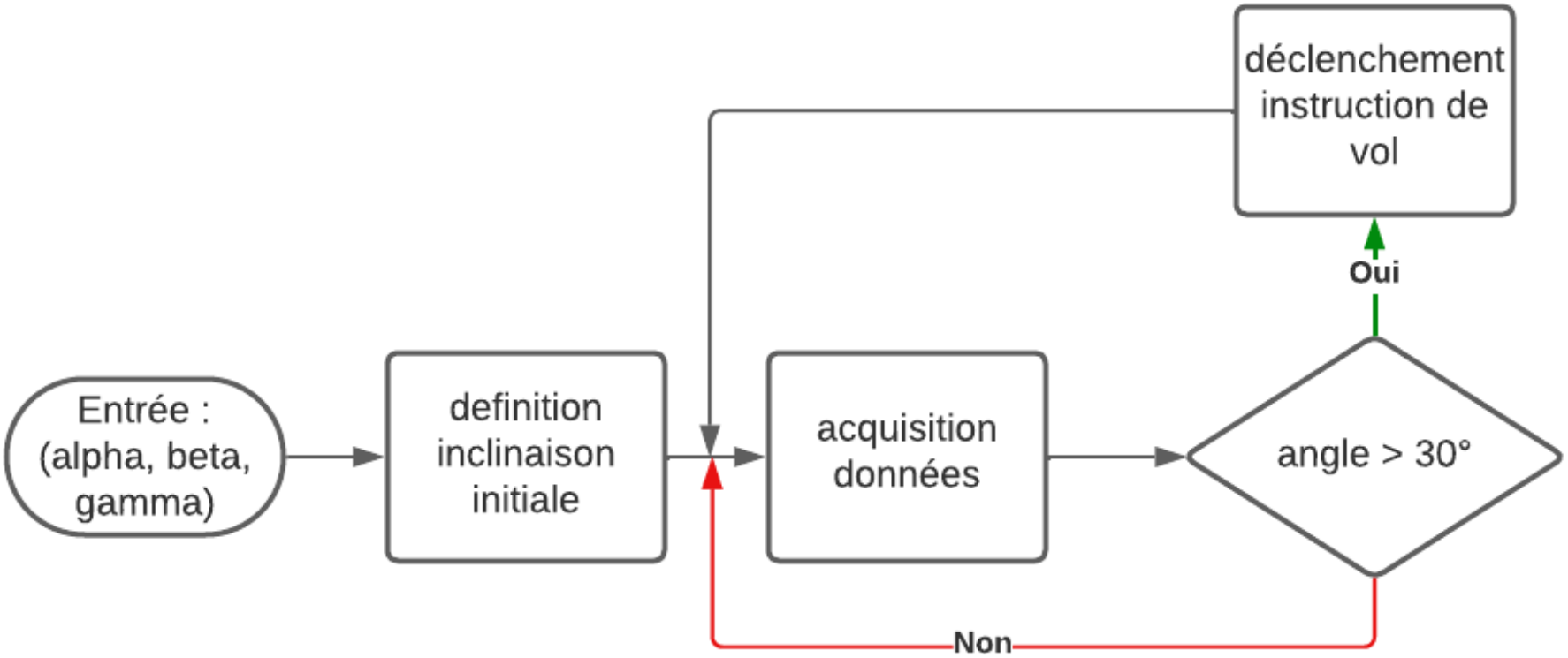
Angle de nutation β
⇔Drone montée/descente en altitude



Angle de « roulis » γ
⇔Drone rotation axe roulis



Récupération des données angulaires sur l'ordinateur et commande du drone



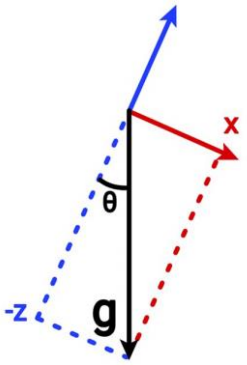
Première solution : capteur de champ magnétique BMM150 + poignée

Composantes du vecteur champ magnétique en 3D.

Champ magnétique terrestre (isolé)

Deuxième solution : capteur type 9DoF IMU : Adafruit BNO085 + poignée

Calcul des angles de tangage et roulis



Accéléromètre – Gyromètre – Magnétomètre

Calibration de l'accéléromètre

Calcul de l'angle de lacet

Traitement des données renvoyées par le capteur

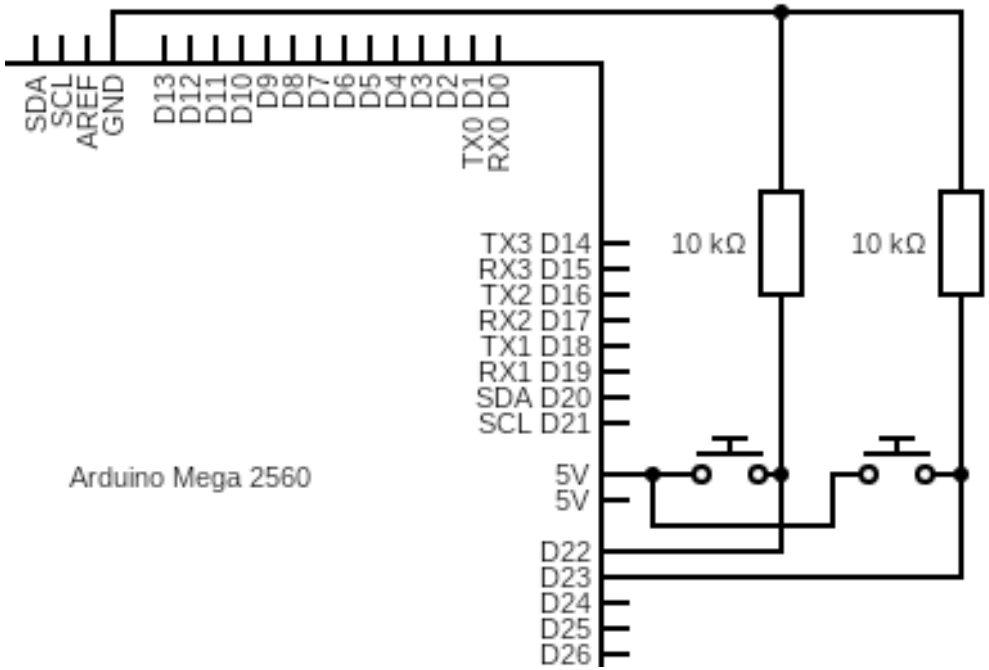
- Protocole i2c

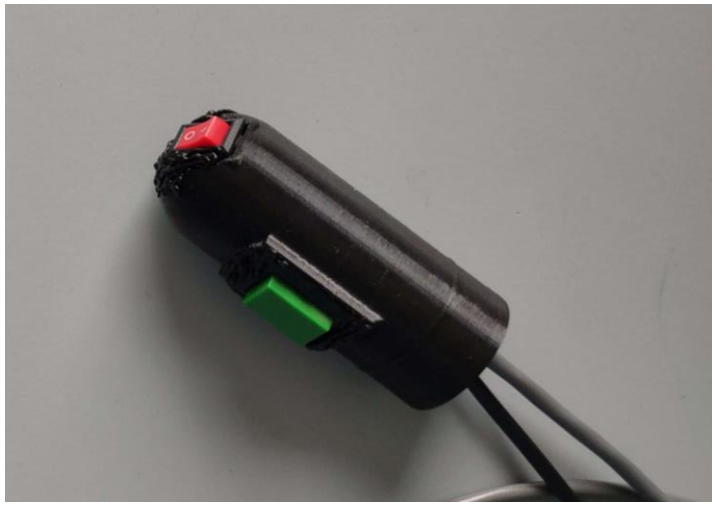
Calibration « statique » en sortie d'usine + - calibration « dynamique » en fonctionnement

- Données renvoyées sous forme de quaternion (vecteur en 4D → vecteur directeur de l'axe de révolution + valeur de l'angle)

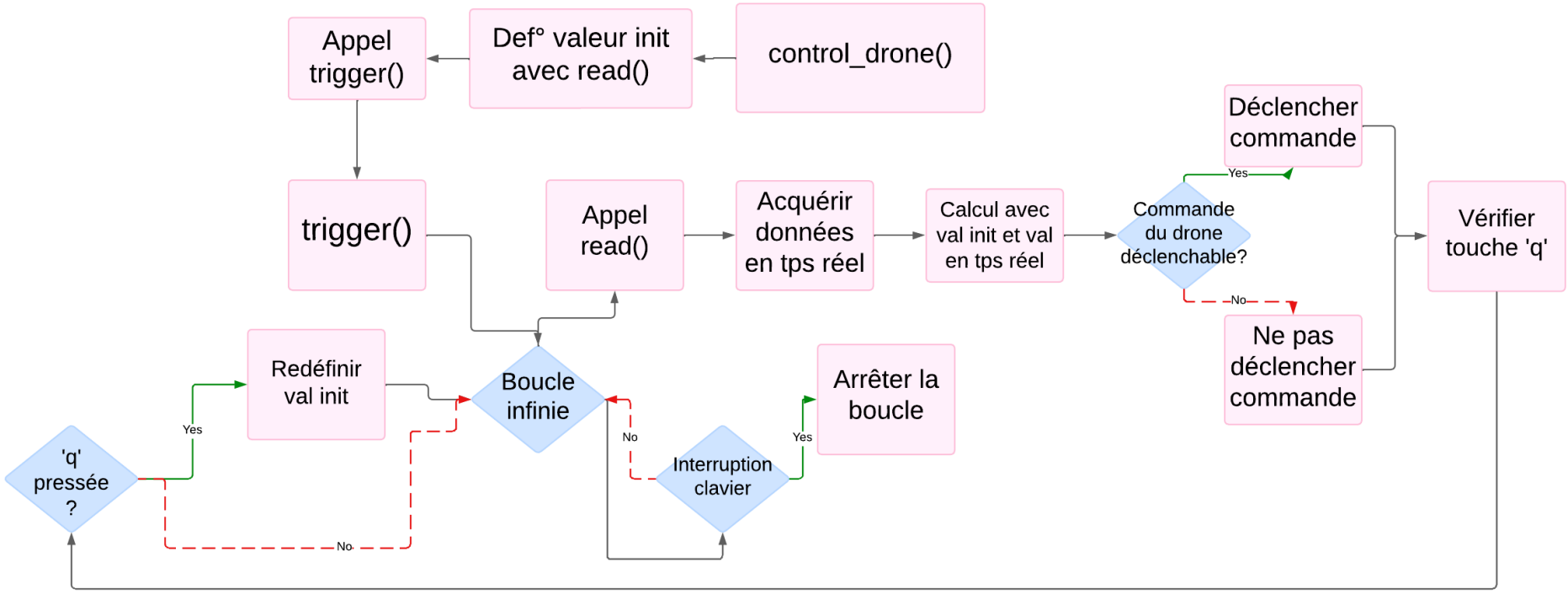
- Calcul des angles de Tait-Bryan (convention intrinsèque ZYX)

Explication fonctionnement poignée



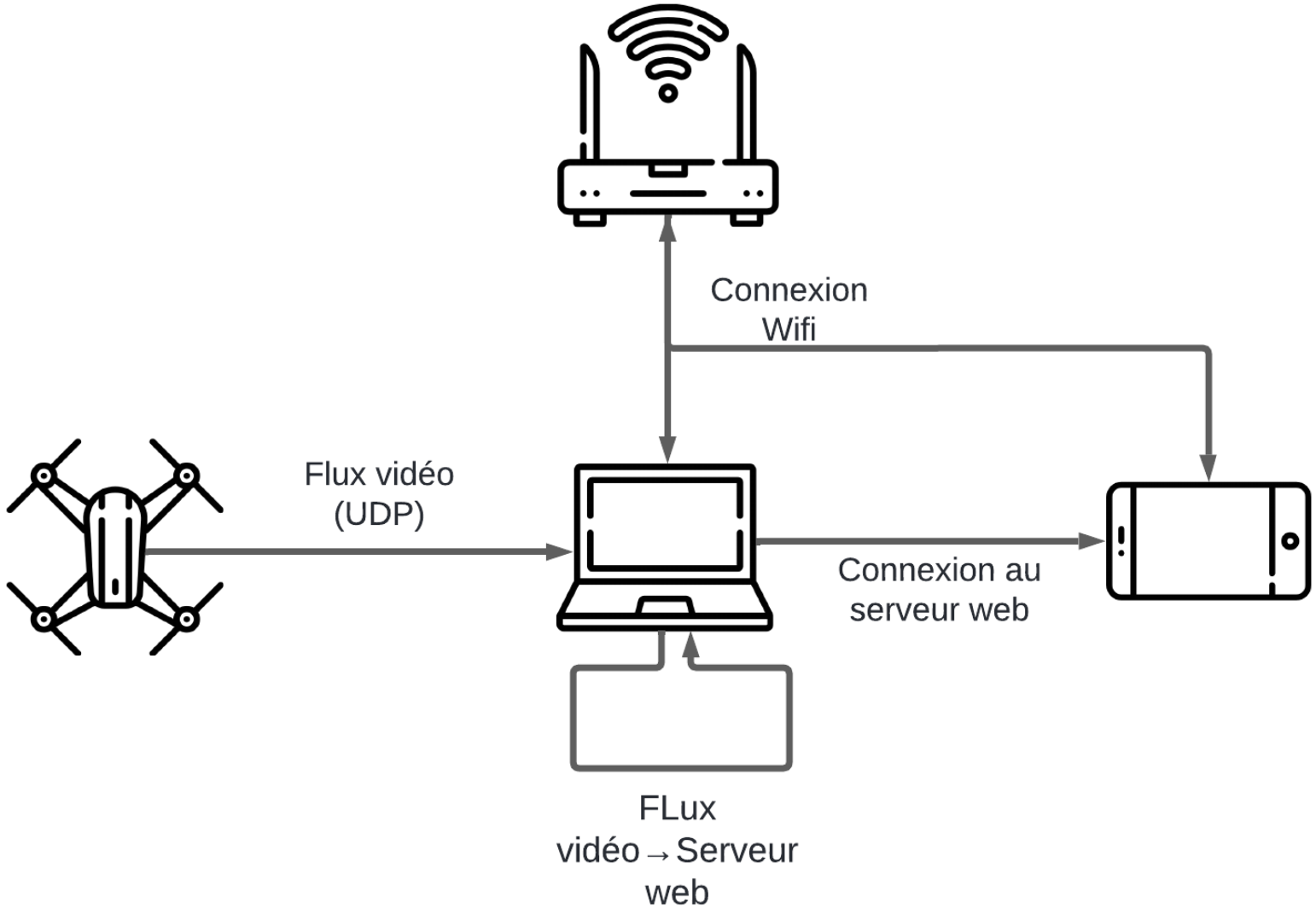


Architecture du code de traitement des données

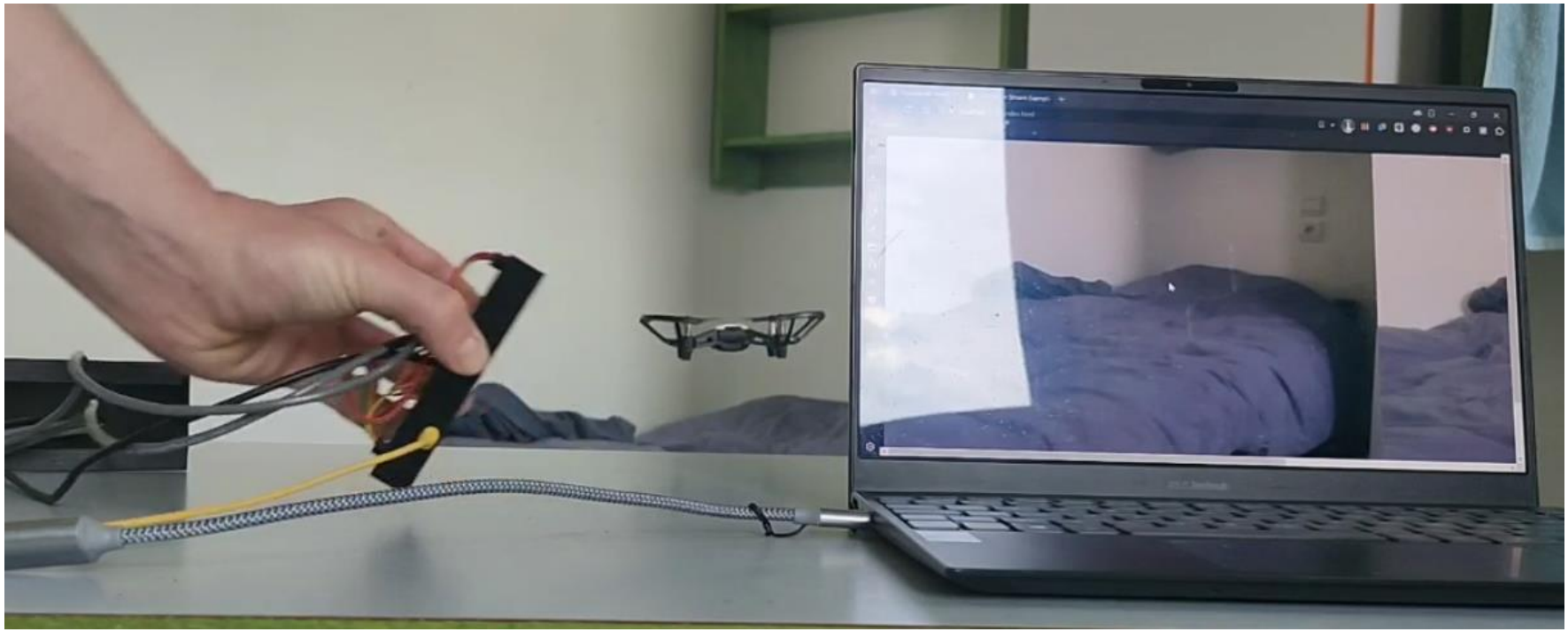


Solution pour l’immersivité

Explication du fonctionnement



Test du drone – immersivité



Temps de réponse mouvement/drone : 134 ± 0,3 ms
Temps de réponse drone/vidéo : 764, ± 0,3 ms
Temps de réponse total : 898 ± 0,42 ms

Vérification des exigences

Valider le cahier des charges :

- coût : solution la moins cher de celles proposées | V
- autonomie : utilisation de batteries | V
- Facilement transportable : transportable dans un sac à dos | V
- facilité de prise en main : un seul script à exécuter | V
- Immersion : temps de réponse trop élevé | X

Solutions alternatives et améliorations envisageables

- Serveur web en python (librairie Flask)
- Réduire la latence vidéo (optimisation ffmepg qui cause la latence)
- Améliorer le confort : commande + vr en « un bloc » imprimé en 3D
- Changer le mode d'envoi de commande (vitesse en fonction de l'inclinaison de la tête)
- Ordinateur et partie web en moins :
  - Application smartphone + cardboard vr
  - Raspberry Pi + petit écran compatible +cardboard vr

Annexe : fonction read()

```
10  arduino = serial.Serial(port='COM5',  baudrate=115200, timeout=.1)
11
12  def read(arduino):
13      try:
14          arduino.reset_input_buffer()
15          data = arduino.readline()
16          string = data.decode()
17          stripped_string = string.strip()
18          if stripped_string:
19              liste = stripped_string.split(",")
20              liste_flt = list(map(float, liste))
21              return liste_flt
22      except (ValueError, UnicodeDecodeError):
23          return None
24      return None
```

Annexe : fonction trigger()

```python
def trigger(tello, arduino):
    init = np.array(read(arduino)[2:])
    print(init)
    while True:
        value = np.array(read(arduino))
        dif = value[2:] - init
        bwd, fwd = value[:2]
        alpha, beta, gamma = ((dif + 180) % 360) - 180
        if keyboard.is_pressed('q'):
            init = value[2:]  # Commandes d'urgence pour réinitialiser la position initiale
        forward_backward, left_right, up_down, yaw = 0, 0, 0, 0
        set_angle, set_speed = 30, 100
        if fwd == 1:
            forward_backward = set_speed
        elif bwd == 1:
            forward_backward = -set_speed
        if gamma < -set_angle:
            left_right = -set_speed
        elif gamma > set_angle:
            left_right = set_speed
        if beta < -set_angle:
            up_down = set_speed
        elif beta > set_angle:
            up_down = -set_speed
        if alpha > set_angle:
            yaw = -set_speed/4
        elif alpha < -set_angle:
            yaw = set_speed/4
        tello.send_rc_control(left_right, forward_backward, up_down, yaw)
        print(forward_backward, left_right, up_down, yaw)
```

Annexe : fonction control_drone()

```python
def control_drone():
    global send_address, send_sock, recv_sock
    tello = Tello()
    tello.connect()
    battery_level = tello.get_battery()
    print(f"Battery level: {battery_level}%")
    tello.streamon()
    udp_address = ('0.0.0.0', 11111) # Adresse UDP du drone
    # Socket de récupération des frames du drone
    recv_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    recv_sock.bind(udp_address)
    # Socket d'envoi des frames vers une autre adresse UDP
    send_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    send_address = ('127.0.0.1', 22222)
    tello.takeoff()
    try:
        trigger(tello, arduino)
    finally:
        recv_sock.close()
        send_sock.close()
        tello.streamoff()
        tello.land()
        print("Landed")

control_drone()
```

Annexe : code arduino

```
#include <Arduino.h>
#include <Adafruit_BNO08x.h>

struct euler_t {
  float yaw;
  float pitch;
  float roll;
} ypr;

Adafruit_BNO08x  bno08x(BNO08X_RESET);
sh2_SensorValue_t sensorValue;

#ifdef FAST_MODE
  // Top frequency is reported to be 1000Hz (but freq is somewhat variable)
  sh2_SensorId_t reportType = SH2_GYRO_INTEGRATED_RV;
  long reportIntervalUs = 2000;
#else
  // Top frequency is about 250Hz but this report is more accurate
  sh2_SensorId_t reportType = SH2_ARVR_STABILIZED_RV;
  long reportIntervalUs = 5000;
#endif
void setReports(sh2_SensorId_t reportType, long report_interval) {
  Serial.println("Setting desired reports");
  if (! bno08x.enableReport(reportType, report_interval)) {
    Serial.println("Could not enable stabilized remote vector");
  }
}
```

```
String str = "";
void(* resetFunc) (void) = 0;
void setup(void) {
  pinMode(12, INPUT);
  pinMode(13, INPUT);
  Serial.begin(115200);
  while (!Serial) delay(10);     // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit BNO08x test!");
  // Try to initialize!
  if (!bno08x.begin_I2C()) {
  //if (!bno08x.begin_UART(&Serial1)) {  // Requires a device with > 300 byte UART buffer!
  //if (!bno08x.begin_SPI(BNO08X_CS, BNO08X_INT)) {
    Serial.println("Failed to find BNO08x chip");
    while (1) { delay(10); }
  }
  Serial.println("BNO08x Found!");


  setReports(reportType, reportIntervalUs);

  Serial.println("Reading events");
  delay(100);
}
```

```
53  void quaternionToEuler(float qr, float qi, float qj, float qk, euler_t* ypr, bool degrees = false) {
54
55      float sqr = sq(qr);
56      float sqi = sq(qi);
57      float sqj = sq(qj);
58      float sqk = sq(qk);
59
60      ypr->yaw = atan2(2.0 * (qi * qj + qk * qr), (sqi - sqj - sqk + sqr));
61      ypr->pitch = asin(-2.0 * (qi * qk - qj * qr) / (sqi + sqj + sqk + sqr));
62      ypr->roll = atan2(2.0 * (qj * qk + qi * qr), (-sqi - sqj + sqk + sqr));
63
64      if (degrees) {
65        ypr->yaw *= RAD_TO_DEG; //precession
66        ypr->pitch *= RAD_TO_DEG; //nutation
67        ypr->roll *= RAD_TO_DEG; // intrinsic rotation
68      }
69  }
70
71  void quaternionToEulerRV(sh2_RotationVectorWAcc_t* rotational_vector, euler_t* ypr, bool degrees = false) {
72      quaternionToEuler(rotational_vector->real, rotational_vector->i, rotational_vector->j, rotational_vector->k, ypr, degrees);
73  }
74
75  void quaternionToEulerGI(sh2_GyroIntegratedRV_t* rotational_vector, euler_t* ypr, bool degrees = false) {
76      quaternionToEuler(rotational_vector->real, rotational_vector->i, rotational_vector->j, rotational_vector->k, ypr, degrees);
77  }
```

```
79  void loop() {
80    if (Serial.available()) {
81      String command = Serial.readStringUntil('\n');
82      if (command == "RESET") {
83        resetFunc(); // Appelle la fonction de réinitialisation
84      }
85    }
86    if (bno08x.wasReset()) {
87      Serial.print("sensor was reset ");
88      setReports(reportType, reportIntervalUs);
89    }
90    if (digitalRead(12) == HIGH) {
91    str = str + "1,";
92    }
93    else if (digitalRead(12) == LOW) {
94      str = str + "0,";
95    }
96    if (digitalRead(13) == HIGH) {
97      str = str + "1,";
98    }
99    else if  (digitalRead(13) == LOW){
100     str = str + "0,";
101   }
```

```
  if (bno08x.getSensorEvent(&sensorValue)) {
    // in this demo only one report type will be received depending on FAST_MODE define (above)
    switch (sensorValue.sensorId) {
      case SH2_ARVR_STABILIZED_RV:
        quaternionToEulerRV(&sensorValue.un.arvrStabilizedRV, &ypr, true);
      case SH2_GYRO_INTEGRATED_RV:
        // faster (more noise?)
        quaternionToEulerGI(&sensorValue.un.gyroIntegratedRV, &ypr, true);
        break;
    }

    Serial.print(str);
    Serial.print(ypr.yaw);                Serial.print(",");
    Serial.print(ypr.pitch);              Serial.print(",");
    Serial.println(ypr.roll);
  }
  str="";

}
```

Annexe : programme Javascript

```
12  // Import necessary modules for the project
13  // A basic http server that we'll access to view the stream
14  const http = require('http');
15
16  // To keep things simple we read the index.html page and send it to the client
17  const fs = require('fs');
18
19  // WebSocket for broadcasting stream to connected clients
20  const WebSocket = require('ws');
21
22  // We'll spawn ffmpeg as a separate process
23  const spawn = require('child_process').spawn;
24
25  // For sending SDK commands to Tello
26  const dgram = require('dgram');
27
28  // HTTP and streaming ports
29  const HTTP_PORT = 3000;
30  const STREAM_PORT = 3001
```

```
/*
  1. Create the web server that the user can access at
  http://localhost:3000/index.html
*/
server = http.createServer(function(request, response) {

  // Log that an http connection has come through
  console.log(
    'HTTP Connection on ' + HTTP_PORT + ' from: ' +
    request.socket.remoteAddress + ':' +
    request.socket.remotePort
  );

  // Read file from the local directory and serve to user
  // in this case it will be index.html
  fs.readFile(__dirname + '/www/' + request.url, function (err,data) {
    if (err) {
      response.writeHead(404);
      response.end(JSON.stringify(err));
      return;
    }
    response.writeHead(200);
    response.end(data);
  });

}).listen(HTTP_PORT); // Listen on port 3000
```

```
/*
  2. Create the stream server where the video stream will be sent
*/
const streamServer = http.createServer(function(request, response) {

  // Log that a stream connection has come through
  console.log(
    'Stream Connection on ' + STREAM_PORT + ' from: ' +
    request.socket.remoteAddress + ':' +
    request.socket.remotePort
  );

  // When data comes from the stream (FFmpeg) we'll pass this to the web socket
  request.on('data', function(data) {
    // Now that we have data let's pass it to the web socket server
    webSocketServer.broadcast(data);
  });

}).listen(STREAM_PORT); // Listen for streams on port 3001
```

```
/*
  3. Begin web socket server
*/
const webSocketServer = new WebSocket.Server({
  server: streamServer
});

// Broadcast the stream via websocket to connected clients
webSocketServer.broadcast = function(data) {
  webSocketServer.clients.forEach(function each(client) {
    if (client.readyState === WebSocket.OPEN) {
      client.send(data);
    }
  });
};
```

```
// Delay for 3 seconds before we start ffmpeg
setTimeout(function() {
  var args = [
    "-i", "udp://127.0.0.1:22222",      You, 2 days ag
    "-r", "30",
    "-vf", "scale=960x720,split=2[a][b];[a][b]hstack",
    "-codec:v", "mpeg1video",
    "-b:v", "800k",
    "-f", "mpegts",
    "http://127.0.0.1:3001/stream"
  ];


  // Spawn an ffmpeg instance
  var streamer = spawn('ffmpeg', args);
  // Uncomment if you want to see ffmpeg stream info
  //streamer.stderr.pipe(process.stderr);
  streamer.on("exit", function(code){
      console.log("Failure", code);
  });
}, 3000);
```

Annexe : Calcul des angles de Tait-Bryan en convention intrinsèque ZYX

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} = w + \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \cos(\alpha/2) + \vec{u}\sin(\alpha/2)$$

$$\vec{u} = \frac{1}{\sqrt{x^2 + y^2 + z^2}} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

On pose : $a = w$; $\begin{pmatrix} b \\ c \\ d \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Formule d'Euler – Rodrigues :

$$\begin{pmatrix} a^2 + b^2 - c^2 - d^2 & 2bc - 2ad & 2ac + 2bd \\ 2ad + 2bc & a^2 - b^2 + c^2 - d^2 & 2cd - 2ab \\ 2bd - 2ac & 2ab + 2cd & a^2 - b^2 - c^2 + d^2 \end{pmatrix}$$

=

$$\sqrt{a^2 + b^2 + c^2 + d^2} \times \begin{bmatrix} c_\alpha c_\beta & c_\alpha s_\beta s_\gamma - c_\gamma s_\alpha & s_\alpha s_\gamma + c_\alpha c_\gamma s_\beta \\ c_\beta s_\alpha & c_\alpha c_\gamma + s_\alpha s_\beta s_\gamma & c_\gamma s_\alpha s_\beta - c_\alpha s_\gamma \\ -s_\beta & c_\beta s_\gamma & c_\beta c_\gamma \end{bmatrix}$$

Annexe : passage de quaternions à angles

$$\alpha = atan2(2(ad + bc), a^2 + b^2 - c^2 - d^2)$$

$$\beta = \arcsin(-2(bd - ac))$$

$$\gamma = atan2(2(ab + cd), a^2 - b^2 - c^2 + d^2)$$