

Documentation drone Pixhawk4 mini

Baptiste BEHELLE

Septembre 2021

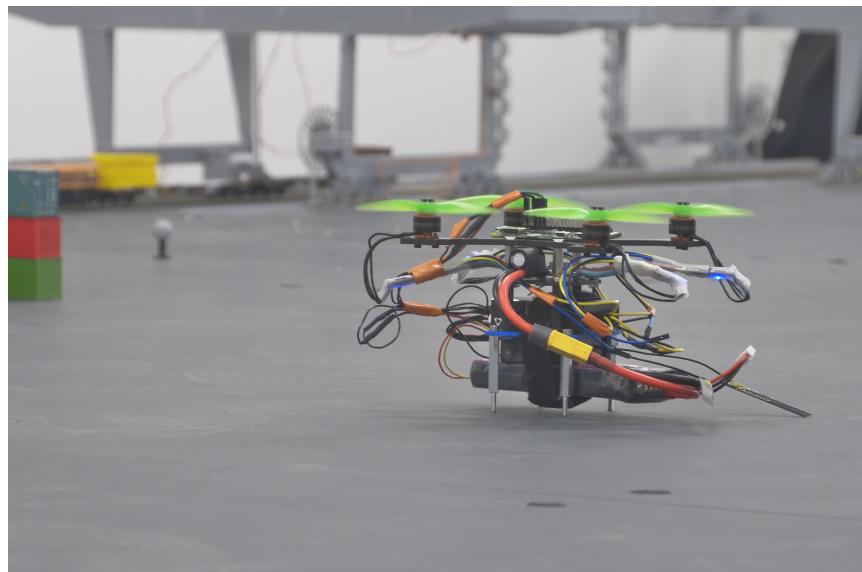


Table des matières

1	Setup du contrôleur	2
2	Drivers utilisés	2
3	Utilisation de Mavros	3
4	Nodes de contrôle à distance	5
5	Test en simulation	16
6	Connectivité entre Raspberry et le Pixhawk4 Mini	21
7	Connexion avec l'Optitrack pour le positionnement externe	24

8 Connexion avec le jumeau numérique et tous autres applications	29
9 Commande pour faire fonctionner les différents programmes	30
10 Notes	30

1 Setup du contrôleur

Pour setup le contrôleur, il faut installer l'application QGroundControl pour mettre à jour le firmware, calibrer les capteurs, tester les moteurs, modifier les paramètres de PID, etc.

Pour plus de détail sur le paramétrage ou sur la suite du document voir la documentation officiel du contrôleur :

<https://docs.px4.io/master/en/>

2 Drivers utilisés

Nécessite QgroundControl pour modifier simplement les paramètres internes du contrôleur et activer et calibrer les différents capteurs :

<http://qgroundcontrol.com/>

Pour communiquer avec le contrôleur via le protocole MAVLINK, il faut installer et utiliser le package Mavros ainsi que les extensions pour utiliser l'optitrack par exemple :

<http://wiki.ros.org/mavros>

http://wiki.ros.org/mavros_extras

Pour utiliser la partie simulation, il faut installer les différents éléments de développement de PX4-Autopilot pour pouvoir utiliser le simulateur :

https://docs.px4.io/master/en/dev_setup/dev_env_linux_ubuntu.html

Pour avoir un dossier permettant de simuler un ou plusieurs drones avec gazebo sur plusieurs simulateurs dont jMAVSIM et Gazebo

Gazebo, jMAVSim and NuttX (Pixhawk) Targets

Use the [ubuntu.sh](#) Gazebo 9 and jMAVSim simulators, and/or the NuttX/Pixhawk toolchain.

WARNING

ROS users must follow the instructions for: [ROS/Gazebo](#).

To install the toolchain:

1. [Download PX4 Source Code](#):

```
git clone https://github.com/PX4/PX4-Autopilot.git --recursive
```

sh

2. Run the [ubuntu.sh](#) with no arguments (in a bash shell) to install everything:

```
bash ./PX4-Autopilot/Tools/setup/ubuntu.sh
```

sh

- Acknowledge any prompts as the script progress.
- You can use the `--no-nuttx` and `--no-sim-tools` options to omit the NuttX and/or simulation tools.

3. Restart the computer on completion.

FIG. 1 – méthode pour pouvoir utiliser les simulateurs sous ROS

3 Utilisation de Mavros

Mavros peut être lancé selon 2 modes pour px4 ou pour ardupilot (apm), dans notre cas on utilise la version px4.

On peut le lancer avec un roslaunch de la façon suivante :

```
roslaunch mavros px4.launch
```

Il peut prendre plusieurs paramètres :

fcu_url: donne le chemin sur quoi on se connecte au drone (localhost pour le simulateur, usb pour l'antenne par exemple) /dev/ttyACM0:57600...

fcu_protocol: donne la version du protocole MAVLink utilisée (v1.0 ou v2.0) nous 2...

log_output: donne à ros l'indication d'où afficher les print (souvent screen)

target_system_id + target_component_id: FCU Mavlink System et Component ID (unique pour chaque drone)

system_id + component_id: Node Mavlink System et Component ID (unique pour chaque drone)

gcs_url: pour la connexion à QgroundControl en UDP

name: (ajouté par moi) pour lancer plusieurs instances de mavros

Et il permet également d'envoyer des commandes sur le contrôleur de drone qui fonctionne comme des services que l'on réutilisera par la suite pour le contrôle distant.

Pour cela, on a accès aux services suivant :

/mavros/set_mode: pour changer le mode de contrôle du drone (nous on veut le passer en offboard pour pouvoir le télépiloter)

/mavros/cmd/arm: pour armer le drone une fois en mode offboard

/mavros/cmd/land: pour envoyer une commande d'atterrissement

/mavros/cmd/takeoff: pour envoyer une commande de décollage

Puis, les topics suivant pour récupérer les données du drone :

/mavros/state: donne l'état du drone et plusieurs informations

/mavros/battery: donne les infos sur la batterie, le voltage, la charge, etc

/mavros/local_position/odom: donne la position du drone à partir des données du FCU (à partir de l'imu)

/mavros/global_position/global: donne la position du drone à partir des données de GPS

Et pour envoyer les instructions de déplacements du drone on peut écrire sur les topics suivant qui sont des objectifs que va tenter d'atteindre le drone (soit une position précise, soit une vitesse)

/mavros/setpoint_position/local: donne une position à atteindre par le drone en indiquant la local_position à atteindre

/mavros/setpoint_velocity/cmd_vel_unstamped: donne une vitesse à atteindre par le drone

(il existe aussi une version attitude et raw de disponible dans le package)

Pour faire fonctionner notre architecture avec plusieurs drones, il faut modifier les fichiers de launch pour donner un nom unique à chaque instance de ROS :

```

<launch>
    <!-- vim: set ft=xml noet : -->
    <!-- example launch script for PX4 based FCU's -->

    <arg name="fcu_url" default="/dev/ttyACM0:57600" />
    <arg name="gcs_url" default="" />
    <arg name="tgt_system" default="1" />
    <arg name="tgt_component" default="1" />
    <arg name="log_output" default="screen" />
    <arg name="fcu_protocol" default="v2.0" />
    <arg name="respawn_mavros" default="false" />
    <arg name="name" default="mavros" />

    <include file="$(find mavros)/launch/custom_node.launch">
        <arg name="pluginlists_yaml" value="$(find mavros)/launch/px4_pluginlists.yaml" />
        <arg name="config_yaml" value="$(find mavros)/launch/px4_config.yaml" />

        <arg name="fcu_url" value="$(arg fcu_url)" />
        <arg name="gcs_url" value="$(arg gcs_url)" />
        <arg name="tgt_system" value="$(arg tgt_system)" />
        <arg name="tgt_component" value="$(arg tgt_component)" />
        <arg name="log_output" value="$(arg log_output)" />
        <arg name="fcu_protocol" value="$(arg fcu_protocol)" />
        <arg name="respawn_mavros" default="$(arg respawn_mavros)" />
        <arg name="name" value="$(arg name)" />
    </include>
</launch>

```

FIG. 2 – *launch pour lancer mavros avec l’option name en plus*

On peut alors lancer plusieurs nodes en changeant l’attribut name entre chaque mavros.

4 Nodes de contrôle à distance

Pour cette partie, on connecte la radio télémetrique de Holybro de 433MHz sur le pc en USB avec sur le drone le récepteur associé. Les codes sont dans le package px4_package. On peut aussi faire fonctionner les algos en connectant le drone depuis un rpi en wifi.

Il y a un node offb_node_tuto.cpp et offb_node_tuto.py qui sont juste des exemples de la documentation officielle du contrôleur de vol.

Il y a également trois autres nodes :

teleop_offb.py : pour contrôler le drone à partir du clavier

autopilot_pose_offb.py : pour faire faire des circuits préprogrammés au drone
(à partir de la position)

autopilot_vel_offb.py : pour faire faire des circuits préprogrammés au drone
(en envoyant des vitesses et non des positions à atteindre)

Chaque algorithme mette en place un système de zone pour éviter les collisions entre chaque drone et une fonction pour reposer le drone à sa position initiale.

Pour décomposer un peu les codes :

On commence par initialiser la node, les topics, les services et les différentes variables.

```
"""
initialisation de la node de contrôle à distance du drone depuis le clavier
"""

def __init__(self):
    #speed
    self.speed = 0.5 # en m/s
    self.speed_ang = 0.2 # en rad/s
    self.speed_alt = 0.5 # en m/s

    # Start ROS node
    rospy.init_node('teleop_offboard_node')

    # Load subscribers, publishers, services
    self.cmd_vel_pub = rospy.Publisher("/mavros/setpoint_velocity/cmd_vel_unstamped", Twist, queue_size=10)
    self.state_sub = rospy.Subscriber("/mavros/state", State, self.state_cb)

    self.arming_client = rospy.ServiceProxy("/mavros/cmd/arm", CommandBool)
    self.takeoff_client = rospy.ServiceProxy("/mavros/cmd/takeoff", CommandTOL)
    self.land_client = rospy.ServiceProxy("/mavros/cmd/land", CommandTOL)
    self.set_mode_client = rospy.ServiceProxy("/mavros/set_mode", SetMode)

    self.current_state = State()
    self.prev_state = self.current_state

    self.rate = rospy.Rate(50.0) # MUST be more than 2Hz
    self.last_request = 0
    self.now = 0

    rospy.loginfo('Teleop offboard node initialized')
```

FIG. 3 – initialisation du noeud ros

On envoie ensuite une série d'instructions inutiles au drone pour tester et faire la connexion entre le pc et le drone

```

cmd = Twist()
cmd.linear.x = 0
cmd.linear.y = 0
cmd.linear.z = 0
cmd.angular.x = 0
cmd.angular.y = 0
cmd.angular.z = 0

# send a few setpoints before starting
for i in range(100):
    self.cmd_vel_pub.publish(cmd)
    self.rate.sleep()

# wait for FCU connection
while not self.current_state.connected:
    self.rate.sleep()

rospy.loginfo('Teleop offboard node connected')
self.last_request = rospy.get_rostime()

```

FIG. 4 – connexion avec le Pixhawk4 Mini

Puis, on peut alors commencer notre code.

Dans les trois nodes on a des fonctions pour armer et changer le mode du drone

On commence toujours par armer et passer en offboard

```

def offboard_mode(self):
    print("offboard mode")
    self.set_mode_client(base_mode=0, custom_mode="OFFBOARD")
    self.rate.sleep()

def arm(self):
    print("arming drone")
    self.arming_client(True)
    self.rate.sleep()

def land_mode(self):
    print("land mode")
    self.set_mode_client(base_mode=0, custom_mode="AUTO.LAND")
    self.rate.sleep()

def disarm(self):
    print("disarming drone")
    #fonctionne à tous les coups
    self.cmd(broadcast=False, command=400, confirmation=0,
             param1=0, param2=21196, param3=0,param4=0, param5=0, param6=0, param7=0)
    self.rate.sleep()

```

FIG. 5 – Fonction pour changer le mode du drone

Pour vérifier l'état du drone il y a une fonction de callback pour mettre à jour la variavle état du drone dans la node

```
"""
fonction de callback pour rafraîchir la variable état du drone pour le conserver en offboard et armé
"""
def state_cb(self,state):
    self.current_state = state
```

FIG. 6 – *Fonction callback de l'état du drone*

Une fois fait les trois algorithmes différent dans celui sur la téléopération on récupère l'entrée clavier et on l'associe à une instruction de vitesse que le drone doit atteindre

```
def key_action(self):
    cmd = Twist()

    #aquisition de la touche clavier
    key = ''

    tty.setraw(sys.stdin.fileno())

    rlist, _, _ = select.select([sys.stdin], [], [], 0.5)
    if rlist:
        key = sys.stdin.read(1)

    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)

    #arrêt de la node si contrôle + c
    if(key == '\x03'):
        print("control + C")
        self.set_mode_client(base_mode=0, custom_mode="AUTO.LAND")
        time.sleep(0.5)
        self.arming_client(False)
        self.rate.sleep()
        return 0
```

```
#envoie de la bonne consigne pour se déplacer dans la direction voulu
if key == 'z':
    cmd.linear.y = self.speed
    print("speed forward (y): ")
    print(cmd.linear.y)
```

```
#envoie de l'objectif de vitesse  
self.cmd_vel_pub.publish(cmd)  
self.rate.sleep()  
return 1
```

Si l'utilisateur n'appuie sur aucun bouton on envoie une consigne vide pour ne pas perdre le mode offboard du drone qui serait changé automatiquement en cas d'arrêt des communications.

On vérifie à chaque tour de boucle que l'on est encore dans la zone du drone, sinon on bloque l'entrée clavier et on remet le drone dans la zone :

```

def verif_zone(self):
    cmd = Twist()
    flag_depassemement = False

    #sinon on arrive jamais à décoller
    if longueur_zone > 0 and largeur_zone > 0 and hauteur_zone > 0 :
        if self.last_pos.pose.position.y > self.init_pos.pose.position.y + longueur_zone :
            flag_depassemement = True
        if self.last_pos.pose.position.x > self.init_pos.pose.position.x + largeur_zone :
            flag_depassemement = True
        if self.last_pos.pose.position.y < self.init_pos.pose.position.y - marge_zone :
            flag_depassemement = True
        if self.last_pos.pose.position.x < self.init_pos.pose.position.x - marge_zone :
            flag_depassemement = True
        if self.last_pos.pose.position.z > self.init_pos.pose.position.z + hauteur_zone :
            flag_depassemement = True
        #sinon on decolle jamais a cause de la derive
        if self.last_pos.pose.position.z < self.init_pos.pose.position.z + alt_minimum :
            flag_depassemement = False

    while flag_depassemement :

        print("#####")
        print("init")
        print(self.init_pos.pose.position)
        print("actual")
        print(self.last_pos.pose.position)

        if self.last_pos.pose.position.y > self.init_pos.pose.position.y + longueur_zone :
            cmd.linear.y = -self.speed_corr
        if self.last_pos.pose.position.y < self.init_pos.pose.position.y :
            cmd.linear.y = self.speed_corr
        if self.last_pos.pose.position.x > self.init_pos.pose.position.x + largeur_zone :
            cmd.linear.x = -self.speed_corr
        if self.last_pos.pose.position.x < self.init_pos.pose.position.y :
            cmd.linear.x = self.speed_corr
        if self.last_pos.pose.position.z > self.init_pos.pose.position.z + hauteur_zone :
            cmd.linear.z = -self.speed_corr

        self.cmd_vel_pub.publish(cmd)
        self.rate.sleep()
        cmd = Twist()

    #condition de fin
    flag_depassemement = False

    if self.last_pos.pose.position.y > self.init_pos.pose.position.y + longueur_zone :
        flag_depassemement = True
    if self.last_pos.pose.position.y < self.init_pos.pose.position.y :
        flag_depassemement = True
    if self.last_pos.pose.position.x > self.init_pos.pose.position.x + largeur_zone :
        flag_depassemement = True
    if self.last_pos.pose.position.x < self.init_pos.pose.position.x :
        flag_depassemement = True
    if self.last_pos.pose.position.z > self.init_pos.pose.position.z + hauteur_zone :
        flag_depassemement = True
    if self.last_pos.pose.position.z < self.init_pos.pose.position.z + alt_minimum :
        flag_depassemement = False

```

FIG. 7 – Fonction pour ne pas sortir du périmètre attribué au drone

Le node de contrôle autopiloté basé sur la position fonctionne sur le même principe pour le début, initialisation du node, passage en mode offboard et armement du drone.

Mais pour la suite il diffère, car on envoie des positions que le drone va essayer d'atteindre :

pour la fonction de déplacement :

on détermine la position finale à atteindre et on évite de dépasser la zone puis on renvoie la commande jusqu'à atteindre la destination

```

if direction == "bottom":
    print("bottom :"+str(distance))
    pos_dest.pose.position.z = pos_start.pose.position.z - distance
    #pour limiter l'altitude
    if largeur_zone > 0 and longueur_zone > 0 and hauteur_zone > 0 :
        if pos_dest.pose.position.z < self.init_pos.pose.position.z :
            pos_dest.pose.position.z = self.init_pos.pose.position.z

self.offboard_mode()
self.loc_pos_sub.publish(pos_dest) #sinon bug au démarrage ou le topic de pos et égale à la destination
self.rate.sleep()

while flag_condition or flag_depassemement:
    self.offboard_mode()
    self.loc_pos_sub.publish(pos_dest)
    self.rate.sleep()

    #verifie que la consigne est valide
    if (direction == "forward" or direction == "back") and abs(pos_dest.pose.position.y - self.last_pos.pose.position.y) <= precision_lin :
        flag_condition = False
    elif (direction == "left" or direction == "right") and abs(pos_dest.pose.position.x - self.last_pos.pose.position.x) <= precision_lin :
        flag_condition = False
    elif (direction == "top" or direction == "bottom") and abs(pos_dest.pose.position.z - self.last_pos.pose.position.z) <= precision_alt :
        flag_condition = False
    else :
        flag_condition = True

    #verifie que l'on dépasse pas la zone
    if largeur_zone > 0 and longueur_zone > 0 and hauteur_zone > 0 :
        if self.last_pos.pose.position.y > self.init_pos.pose.position.y + longueur_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.x > self.init_pos.pose.position.x + largeur_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.y < self.init_pos.pose.position.y - marge_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.x < self.init_pos.pose.position.x - marge_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.z > self.init_pos.pose.position.z + hauteur_zone :
            flag_depassemement = True
        else :
            flag_depassemement = False

```

FIG. 8 – Fonction pour un déplacement linéaire

pour la fonction de rotation :

```

if sens == "left":
    print("rota gauche :" +str(angle))
    deg_z_obj = (deg_z+angle)%360

    qua_x, qua_y, qua_z, qua_w = euler_to_quaternion(deg_x,deg_y,deg_z_obj)

    pos_dest.pose.orientation.x = qua_x
    pos_dest.pose.orientation.y = qua_y
    pos_dest.pose.orientation.z = qua_z
    pos_dest.pose.orientation.w = qua_w

self.offboard_mode()
self.loc_pos_sub.publish(pos_dest) #sinon bug au démarrage ou le topic de pos et égale à la destination
self.rate.sleep()

while (deg_z >= deg_z_obj + precision_ang or deg_z <= deg_z_obj - precision_ang) or flag_depassemement :

    self.offboard_mode()
    self.loc_pos_sub.publish(pos_dest)
    self.rate.sleep()

    deg_x,deg_y,deg_z = quaternion_to_euler(self.last_pos.pose.orientation)
    if deg_z > 0:
        deg_z = deg_z
    else:
        deg_z = deg_z + 360

    #verifie que l'on depasse pas la zone
    if largeur_zone > 0 and longueur_zone > 0 and hauteur_zone > 0 :
        if self.last_pos.pose.position.y > self.init_pos.pose.position.y + longueur_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.x > self.init_pos.pose.position.x + largeur_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.y < self.init_pos.pose.position.y - marge_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.x < self.init_pos.pose.position.x - marge_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.z > self.init_pos.pose.position.z + hauteur_zone :
            flag_depassemement = True
        else :
            flag_depassemement = False

```

FIG. 9 – Fonction pour un déplacement angulaire

Il y a également une fonction de callback pour actualiser la position du drone dans les variables du programme :

```
def pose_cb(self,pose):
    if self.flag_init_pos == True:
        self.flag_init_pos = False
        self.init_pos = pose
    self.last_pos = pose
```

FIG. 10 – Fonction callback de la position linéaire et angulaire du drone

On peut alors programmer des déplacements avec les deux fonctions de déplacement pour que le drone fasse des trajets précis de façon autonomes.

Enfin, le node autopiloté basé sur la vitesse repose sur l'envoi de vitesse comme dans la node de téléopération. Il est moins précis que le programme précédent cependant, on a un contrôle sur la vitesse de déplacement du drone vers ça cible. Voici une partie des fonctions de déplacement et de rotation adapté à ce mode : fonction de déplacement :

```

if direction == "top" or direction == "bottom" :
    if pos_dest.pose.position.z > self.last_pos.pose.position.z :
        cmd.linear.z = self.speed
    else:
        cmd.linear.z = -self.speed

    #correction des erreurs
    if pos_dest.pose.position.x > self.last_pos.pose.position.x:
        cmd.linear.x = self.speed_corr
    elif pos_dest.pose.position.x < self.last_pos.pose.position.x :
        cmd.linear.x = -self.speed_corr
    if pos_dest.pose.position.y > self.last_pos.pose.position.y :
        cmd.linear.y = self.speed_corr
    elif pos_dest.pose.position.y < self.last_pos.pose.position.y :
        cmd.linear.y = -self.speed_corr

self.offboard_mode()
self.cmd.vel_pub.publish(cmd)
self.rate.sleep()
cmd = Twist()

#verifie que la consigne est valide
if (direction == "forward" or direction == "back") and abs(pos_dest.pose.position.y - self.last_pos.pose.position.y) <= precision
    flag_condition = False

elif (direction == "left" or direction == "right") and abs(pos_dest.pose.position.x - self.last_pos.pose.position.x) <= precision
    flag_condition = False

elif (direction == "top" or direction == "bottom") and abs(pos_dest.pose.position.z - self.last_pos.pose.position.z) <= precision
    flag_condition = False

else :
    flag_condition = True

#verifie que l'on depasse pas la zone
if largeur_zone > 0 and longueur_zone > 0 and hauteur_zone > 0 :
    if self.last_pos.pose.position.y > self.init_pos.pose.position.y + longueur_zone :
        flag_depassemement = True
    elif self.last_pos.pose.position.x > self.init_pos.pose.position.x + largeur_zone :
        flag_depassemement = True
    elif self.last_pos.pose.position.y < self.init_pos.pose.position.y - marge_zone :
        flag_depassemement = True
    elif self.last_pos.pose.position.x < self.init_pos.pose.position.x - marge_zone :
        flag_depassemement = True
    elif self.last_pos.pose.position.z > self.init_pos.pose.position.z + hauteur_zone :
        flag_depassemement = True
    else :
        flag_depassemement = False

```

FIG. 11 – Fonction pour un déplacement linéaire

fonction de rotation :

```

if sens == "left":
    print("rota gauche :" +str(angle))
    deg_z_obj = (deg_z+angle)%360

while deg_z > deg_z_obj + precision_ang or deg_z < deg_z_obj - precision_ang or flag_depassemement :
    #correction des perturbations
    if pos_dest.pose.position.x > self.last_pos.pose.position.x:
        cmd.linear.x = self.speed_corr
    elif pos_dest.pose.position.x < self.last_pos.pose.position.x:
        cmd.linear.x = -self.speed_corr

    if pos_dest.pose.position.y > self.last_pos.pose.position.y:
        cmd.linear.y = self.speed_corr
    elif pos_dest.pose.position.y < self.last_pos.pose.position.y:
        cmd.linear.y = -self.speed_corr

    if pos_dest.pose.position.z > self.last_pos.pose.position.z:
        cmd.linear.z = self.speed_corr
    elif pos_dest.pose.position.z < self.last_pos.pose.position.z:
        cmd.linear.z = -self.speed_corr

    if sens == "right" :
        cmd.angular.z = -self.speed_ang
    else :
        cmd.angular.z = self.speed_ang

    #commande de l'angle
    self.offboard_mode()
    self.cmd_vel_pub.publish(cmd)
    self.rate.sleep()
    cmd = Twist()

    deg_x,deg_y,deg_z = quaternion_to_euler(self.last_pos.pose.orientation)
    if deg_z > 0:
        deg_z = deg_z
    else:
        deg_z = deg_z + 360

    #verifie que l'on depasse pas la zone
    if largeur_zone > 0 and longueur_zone > 0 and hauteur_zone > 0 :
        if self.last_pos.pose.position.y > self.init_pos.pose.position.y + longueur_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.x > self.init_pos.pose.position.x + largeur_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.y < self.init_pos.pose.position.y - marge_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.x < self.init_pos.pose.position.x - marge_zone :
            flag_depassemement = True
        elif self.last_pos.pose.position.z > self.init_pos.pose.position.z + hauteur_zone :
            flag_depassemement = True
        else :
            flag_depassemement = False

```

FIG. 12 – Fonction pour un déplacement angulaire

Les fonctions de home reposent sur les mêmes principes soit on envoie une position, soit on envoie des vitesses jusqu'à ce que l'on arrive à destination.

Pour lancer les différents nodes on a un launch associé à chacun, il y a plusieurs paramètres :

- topic_name** : la racine des topics mavros pour le multidrone
- node_name** : pour pouvoir lancer plusieurs fois le node avec des noms différents
- longueur_zone** : Détermine la taille de la zone suivant l'axe y en partant du drone avec une mage de 5 cm derrière le drone
- largeur_zone** : Détermine la taille de la zone suivant l'axe x en partant du drone avec une mage de 5 cm derrière le drone
- hauteur_zone** : Détermine la hauteur max que peut atteindre le drone en partant de sa position initiale

Si on ne met pas longueur_zone, largeur_zone ou hauteur_zone la fonctionnalité de zone est désactivée et le drone évolue librement sans restriction

```
<?xml version="2.0" ?>
<launch>
  <arg name="topic_name" default="mavros"/>
  <arg name="node_name" default="node_teleop_offb"/>
  <arg name="longueur_zone" default="0"/>
  <arg name="largeur_zone" default="0"/>
  <arg name="hauteur_zone" default="0"/>

  <param name="topic_name" value="$(arg topic_name)" />
  <param name="node_name" value="$(arg node_name)" />
  <param name="longueur_zone" value="$(arg longueur_zone)"/>
  <param name="largeur_zone" value="$(arg largeur_zone)"/>
  <param name="hauteur_zone" value="$(arg hauteur_zone)"/>

  <group>
    <node pkg="px4_package" type="teleop_offb.py" name="$(arg node_name)" output="screen"/>
  </group>
</launch>
```

FIG. 13 – Fichier launch du node de téléopération

5 Test en simulation

Pour le mono-drone :
 On peut utiliser deux simulateurs pour ce mode jmavsim et gazebo :
 Pour lancer l'un ou l'autre des simulateurs, il faut aller à la racine du dossier PX4-Autopilot et faire :

```
make px4_sitl jmavsim
make px4_sitl gazebo
```

On a alors l'environnement suivant :

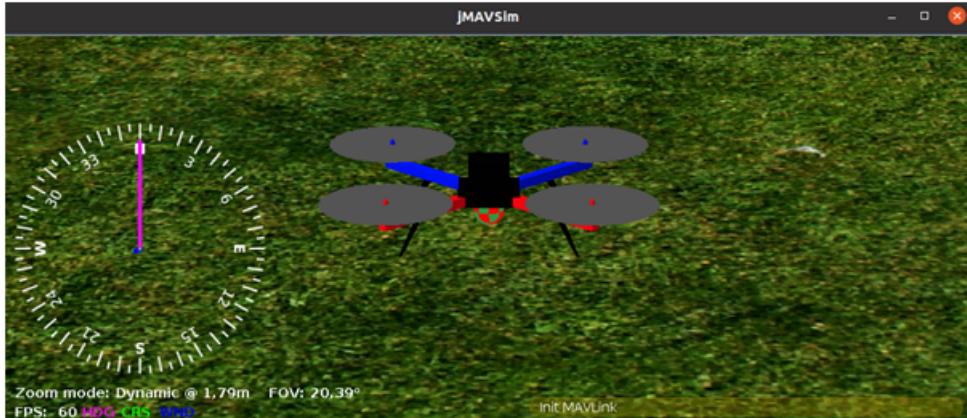


FIG. 14 – Environnement de simulation JMAVSIM

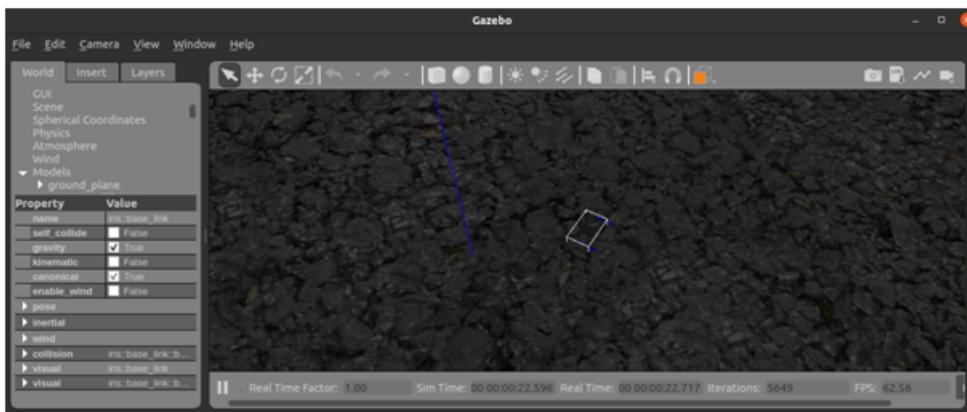


FIG. 15 – Environnement de simulation Gazebo

On peut alors lancer mavros avec le paramètre fcu_url suivant pour se connecter au simulateur via localhost et le port du protocole mavlink.

```
fcu "-url="udp://:14540@127.0.0.1:14557"
```

Enfin, on lance nos nodes pour envoyer des codes interpréter dans le simulateur.
(cf commandes en fin de section)

Pour le mode multi-drone, on ne peut le faire qu'avec gazebo :

Pour cela, il faut source le dossier PX4-Autopilot pour qu'il soit dans les chemins visibles par ROS :

```
source Tools/setup_gazebo.bash $(pwd) $(pwd)/build/px4_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd):$(pwd)/Tools/sitl_gazebo
```

Puis, on crée un fichier launch qu'on lancera et qui générera l'environnement :
On commence par setup l'environnement :

```
<arg name="est" default="ekf2"/>
<arg name="vehicle" default="iris"/>
<arg name="world" default="$(find mavlink_sitl_gazebo)/worlds/empty.world"/>
<!-- gazebo configs -->
<arg name="gui" default="true"/>
<arg name="debug" default="false"/>
<arg name="verbose" default="false"/>
<arg name="paused" default="false"/>
<!-- Gazebo sim -->
<include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="gui" value="$(arg gui)"/>
    <arg name="world_name" value="$(arg world)"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="verbose" value="$(arg verbose)"/>
    <arg name="paused" value="$(arg paused)"/>
</include>
```

Puis les objets multicopters :

```

<!-- UAV0 -->
<group ns="uav0">
    <!-- MAVROS and vehicle configs -->
    <arg name="ID" value="0"/>
    <arg name="fcu_url" default="udp://:14540@localhost:14580"/>
    <!-- PX4 SITL and vehicle spawn -->
    <include file="$(find px4)/launch/single_vehicle_spawn.launch">
        <arg name="x" value="0"/>
        <arg name="y" value="0"/>
        <arg name="z" value="0"/>
        <arg name="R" value="0"/>
        <arg name="P" value="0"/>
        <arg name="Y" value="0"/>
        <arg name="vehicle" value="$(arg vehicle)"/>
        <arg name="mavlink_udp_port" value="14560"/>
        <arg name="mavlink_tcp_port" value="4560"/>
        <arg name="ID" value="$(arg ID)"/>
        <arg name="gst_udp_port" value="$(eval 5600 + arg('ID'))"/>
        <arg name="video_uri" value="$(eval 5600 + arg('ID'))"/>
        <arg name="mavlink_cam_udp_port" value="$(eval 14530 + arg('ID'))"/>
    </include>
</group>
<!-- UAV1 -->
<group ns="uav1">
    <!-- MAVROS and vehicle configs -->
    <arg name="ID" value="1"/>
    <arg name="fcu_url" default="udp://:14541@localhost:14581"/>
    <!-- PX4 SITL and vehicle spawn -->
    <include file="$(find px4)/launch/single_vehicle_spawn.launch">
        <arg name="x" value="1"/>
        <arg name="y" value="0"/>
        <arg name="z" value="0"/>
        <arg name="R" value="0"/>
        <arg name="P" value="0"/>
        <arg name="Y" value="0"/>
        <arg name="vehicle" value="$(arg vehicle)"/>
        <arg name="mavlink_udp_port" value="14561"/>
        <arg name="mavlink_tcp_port" value="4561"/>
        <arg name="ID" value="$(arg ID)"/>
        <arg name="gst_udp_port" value="$(eval 5600 + arg('ID'))"/>
        <arg name="video_uri" value="$(eval 5600 + arg('ID'))"/>
        <arg name="mavlink_cam_udp_port" value="$(eval 14530 + arg('ID'))"/>
    </include>
</group>

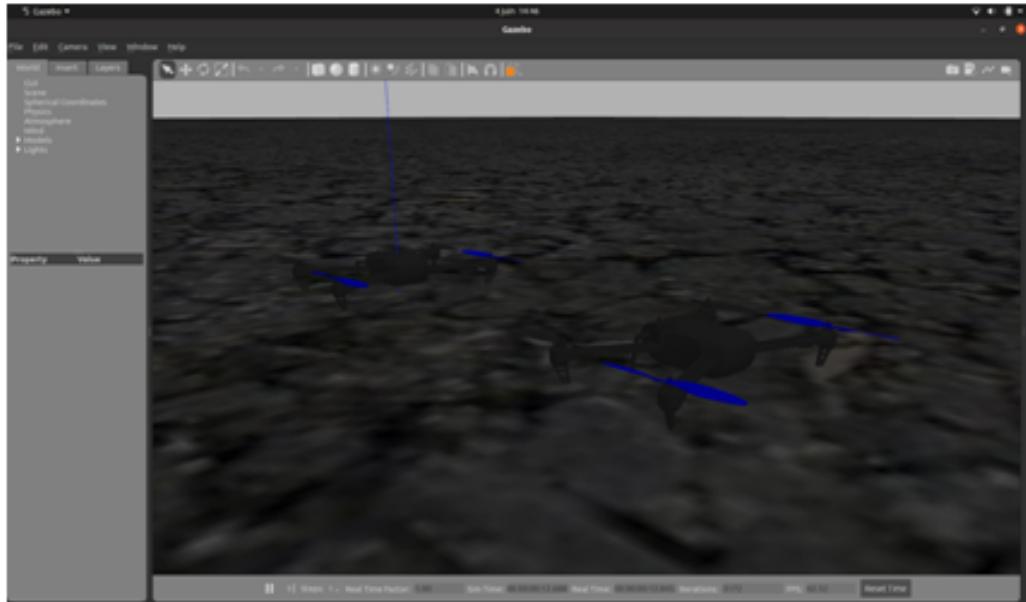
```

Pour en mettre plus, il faut les dupliquer, modifier leur position initiale x,y,z.
Il faut modifier leur adresse et le port sur l'argument fcu.url. Il faut un id unique
et modifier aussi le mavlink_udp_port et mavlink_tcp_port.
Pour faire simple on peut juste incrémenter les attributs cités précédemment
pour chaque drone ajouté.

Enfin, pour lancer le simulateur on fait alors

`roslaunch px4 <nom du fichier>.launch`

Quand on lance l'environnement on a alors :



Pour contrôler les deux drones il faut 2 instances de mavros, il y a donc une modification à apporter aux fichiers launch de mavros pour pouvoir renommer les nodes et en lancer plusieurs :

```
roslaunch mavros custom_px4.launch fcu_url:=“udp://:14540@localhost:14580” name:=“drone_1” tgt_system:=“1”
```

```
roslaunch mavros custom_px4.launch fcu_url:=“udp://:14541@localhost:14581” name:=“drone_2” tgt_system:=“2”
```

On a alors 2 mavros qui gère chacun 1 drone avec les topics associés qui auront pour racines /drone_n/...

Enfin, on peut alors activer nos nodes de contrôle de la façon suivante :

```
roslaunch px4_offboard autopilot_pose_offb.launch topic_name:=“drone_1” node_name:=“autopilot_1” longueur_zone:=2 largeur_zone:=2 hauteur_zone:=1
```

```
roslaunch px4_offboard autopilot_vel_offb.launch topic_name:=“drone_2” node_name:=“autopilot_2” longueur_zone:=2 largeur_zone:=2 hauteur_zone:=1
```

```
roslaunch px4_offboard teleop_offb.launch topic_name:=“drone_1” node_name:=“teleop_1”
```

6 Connectivité entre Raspberry et le Pixhawk4 Mini

On peut travailler avec un Raspberry 0wh pour des raisons de tailles, mais il fera parfaitement l'affaire. Pour faire ce lien on commence par installer une version lite de Rasbian puis on compile depuis les sources ROS. On crée un catkin workspace et on compile mavros et le package pour la caméra depuis les sources en incluant toutes les dépendances.

Puis on compile le workspace ce qui prend plusieurs heures la première fois à cause de la faible puissance du Rpi.

Sinon il est possible de faire la même chose avec les versions plus récentes de Raspberry

Il faudra également augmenté le swap, activer le port spi, activer le port série et activer le ssh pour opérer à distance.

Pour ce qui est du câblage, il faut se référer à la documentation de ardupilot qui propose un schéma :

<https://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html>

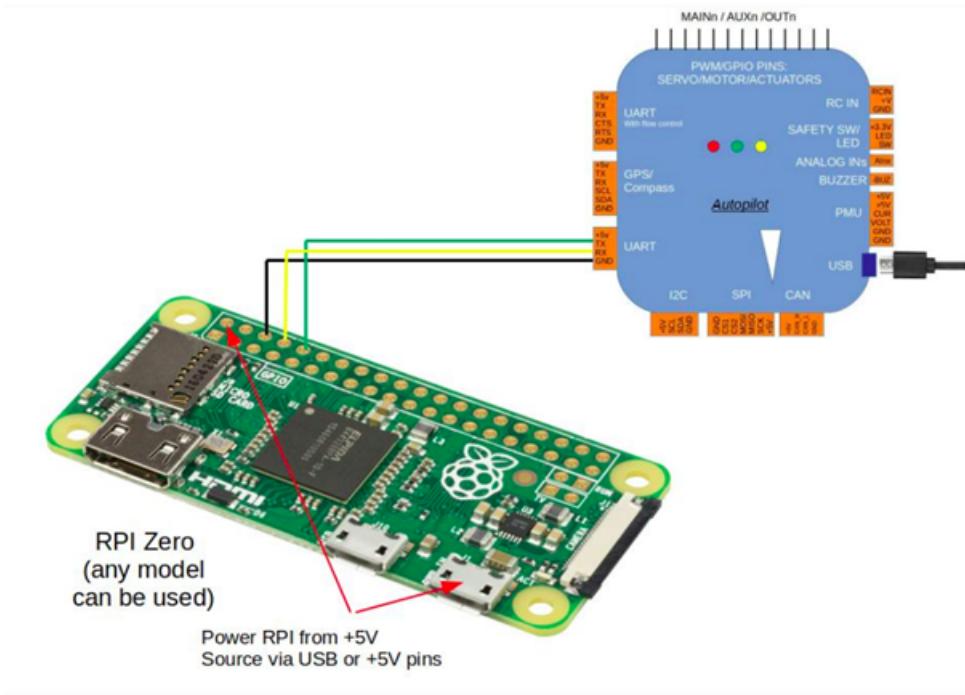


FIG. 16 – Cablage entre le Raspberry et le Pixhawk 4 Mini

Comme on le voit sur le schéma on peut alimenter le raspberry depuis le contrôleur de vol.

Vérifiez votre câblage à partir de la documentation officielle de holybro sur les pinouts du Pixhawk4 :

<http://www.holybro.com/manual/Pixhawk4-Pinouts.pdf>

TELEM1, TELEM2 ports

Pin	Signal	Volt
1(red)	VCC	+5V
2(black)	TX(out)	+3.3V
3(black)	RX(in)	+3.3V
4(black)	CTS(in)	+3.3V
5(black)	RTS(out)	+3.3V
6(black)	GND	GND

UART & I2C B port *

Pin	Signal	Volt
1(red)	VCC	+5V
2(black)	TX(out)	+3.3V
3(black)	RX(in)	+3.3V
4(black)	SCL2	+3.3V
5(black)	SDA2	+3.3V
6(black)	GND	GND

FIG. 17 – Tableau des pins du Pixhawk4

Voilà les deux tableaux qui nous intéressent, on utilise le VCC pour alimenter le Raspberry, TX et RX pour faire communiquer les données sur le port série et GND pour la masse.

Enfin, avant de tous faire fonctionner, il faut configurer le Pixhawk4 mini pour lui indiquer la présence du raspberry.

Il faut aller dans les paramètres et modifier les lignes suivantes (selon l'endroit où le raspberry est câblée) :

MAVLink	MAV_0_RATE	1200 B/s	Maximum MAVLink sending rate for instance 0
Multicopter Rate Control	MAV_1_CONFIG	TELEM/SERIAL 4	Serial Configuration for MAVLink (instance 1)
Mixer Output	MAV_1_FORWARD	1	Enable MAVLink Message forwarding for instance 1
Multicopter Attitude Control	MAV_1_MODE	Onboard	MAVLink Mode for instance 1
Mount	MAV_1_RADIO_CTL	1	Enable software throttling of mavlink on instance 1
	MAV_1_RATE	921600 B/s	Maximum MAVLink sending rate for instance 1

FIG. 18 – Paramètre pour la connexion entre le Raspberry et Pixhawk4 mini à modifier dans QGroundControl

On peut maintenant connecter ROS :

Roscore tourne sur le pc de supervision dans notre architecture.

On lance un ssh sur le raspberry voulu, puis on modifie 2 variables globales :

ROS_MASTER_URI qui indique le chemin vers Roscore ici
http://<ip machine maître>:11311

et

ROS_IP pour envoyer les messages en utilisant son ip comme identifiants et pas son hostname
(car pas de dns dans mes différents tests sur mon réseau de téléphone)
on met donc l'ip du raspberry

On peut alors lancer mavros de la façon suivante :

```
roslaunch mavros custom_px4.launch node_name:="drone_1" fcu_url:="/dev/serial0"
```

Et on va recevoir toutes les données sur le PC de supervision.

7 Connexion avec l'Optitrack pour le positionnement externe

On place 3 pastilles réfléchissantes minimum sur le drone de façon asymétrique qui seront traquées par les caméras. L'asymétrie de placement permet d'avoir une orientation plus propre sinon il pourrait être dans 2 sens différents.

On configure le rigid body dans l'Optitrack pour avoir notre objet drone :

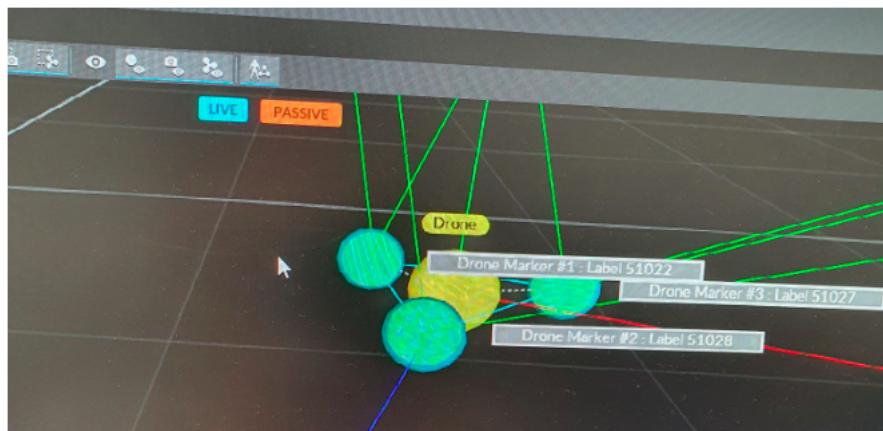


FIG. 19 – création du rigid body

On configure les paramètres réseaux comme suit :



FIG. 20 – paramètre du streaming panel

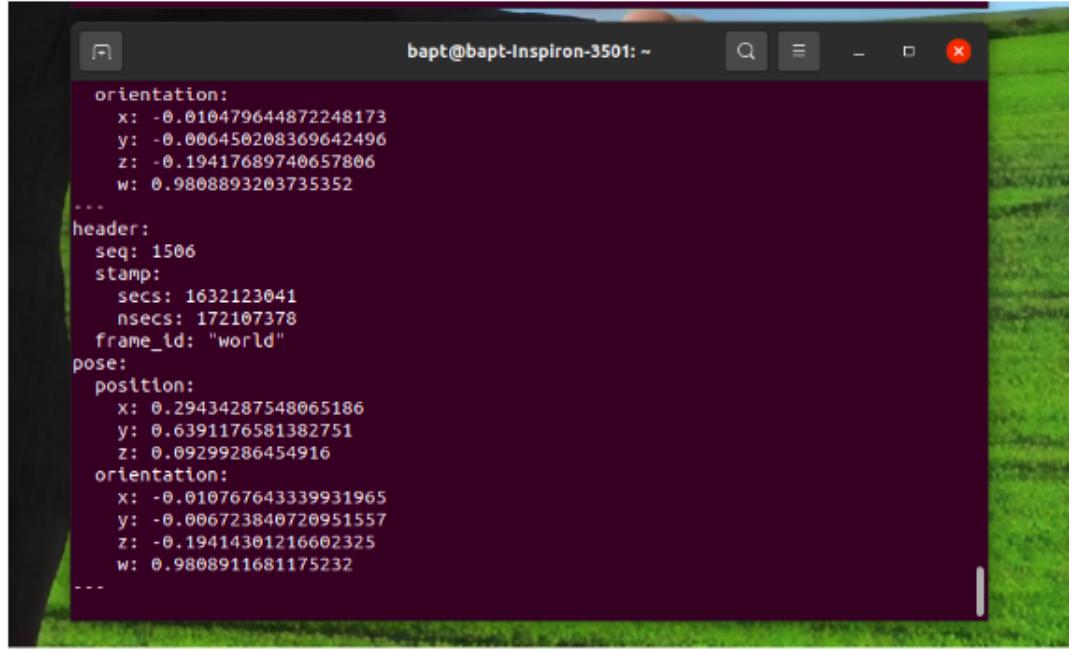
On câble le pc connecté à ROS sur le même routeur que l'Optitrack et on peut alors lancer nos noeuds pour récupérer les données.

Il y a deux méthodes possibles :

Soit, on utilise le noeud `mocap_optitrack` avec le fichier de config suivant et le paramètre `up axis` dans `motive` sur `y axis` (car il fait la transformation de repère)(topic `/mocap_node/Drone/pose`) :

```
1 #
2 # Definition of all trackable objects
3 # Identifier corresponds to Trackable ID set in Tracking Tools
4 #
5 rigid_bodies:
6
7     '1':
8         pose: Drone/pose
9         pose2d: Drone/ground_pose
10        odom: Drone/odom
11        child_frame_id: Drone/base_link
12        parent_frame_id: world
13        tf: tf
14
15
16 optitrack_config:
17     multicast_address: 224.0.0.1
18     command_port: 1510
19     data_port: 1511
20     enable_optitrack: true
```

FIG. 21 – fichier de config du mocap_optitrack

A screenshot of a terminal window titled "bapt@bapt-Inspiron-3501: ~". The window displays a series of ROS message logs. The first log shows orientation data with values: x: -0.010479644872248173, y: -0.006450208369642496, z: -0.19417689740657806, w: 0.9808893203735352. The second log shows header information with seq: 1506, stamp: secs: 1632123041, nsecs: 172107378, and frame_id: "world". The third log shows pose data with position: x: 0.29434287548065186, y: 0.6391176581382751, z: 0.09299286454916, and orientation: x: -0.010767643339931965, y: -0.006723840720951557, z: -0.19414301216602325, w: 0.9808911681175232.

```
bapt@bapt-Inspiron-3501: ~
orientation:
  x: -0.010479644872248173
  y: -0.006450208369642496
  z: -0.19417689740657806
  w: 0.9808893203735352
...
header:
  seq: 1506
  stamp:
    secs: 1632123041
    nsecs: 172107378
  frame_id: "world"
pose:
  position:
    x: 0.29434287548065186
    y: 0.6391176581382751
    z: 0.09299286454916
  orientation:
    x: -0.010767643339931965
    y: -0.006723840720951557
    z: -0.19414301216602325
    w: 0.9808911681175232
...
```

FIG. 22 – affichage du topic ou on voit les données optitrack

Soit, on utilise le noeud vrpnc_client_ros avec le paramètre up axis sur z axis (car il ne fait pas la transformation sinon)(topic : vrpnc_client_node/Drone/pose) :

```

bapt@bapt-Inspiron-3501: ~
[{"header": {"seq": 3306, "stamp": {"secs": 1632123129, "nsecs": 777206209}, "frame_id": "world"}, "pose": {"position": {"x": 0.2943422198295593, "y": 0.6391341090202332, "z": 0.09300439804792404}, "orientation": {"x": -0.01049045566469431, "y": -0.006246812175959349, "z": -0.19430698454380035, "w": 0.9808647632598877}}, "cmd_": {"header": {"seq": 3306, "stamp": {"secs": 1632123129, "nsecs": 777206209}, "frame_id": "world"}, "pose": {"position": {"x": 0.2943422198295593, "y": 0.6391341090202332, "z": 0.09300439804792404}, "orientation": {"x": -0.01049045566469431, "y": -0.006246812175959349, "z": -0.19430698454380035, "w": 0.9808647632598877}}}

```

FIG. 23 – affichage du topic où on voit les données optitrack

Dans Motive :

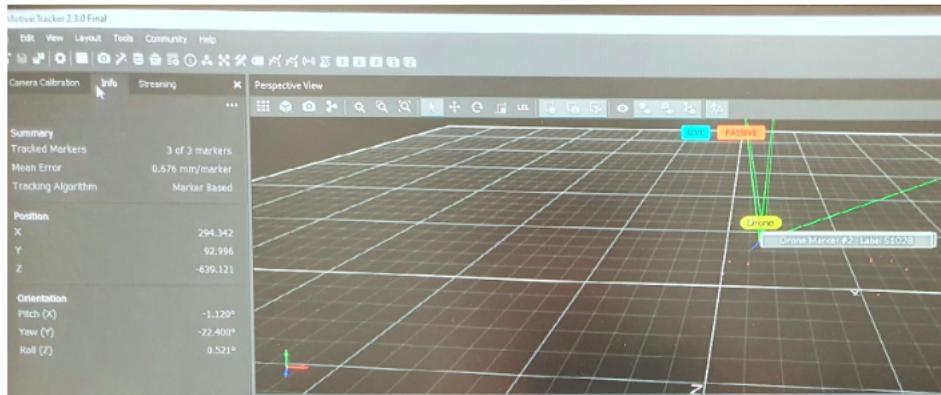


FIG. 24 – position en millimètre relevé par l'application Motive qui gère les caméras et envoie les données

Pour connecter le système de positionnement du Pixhawk4 mini, il faut changer quelques paramètres sur Qgroundcontrol pour activer le positionnement via l'Optitrack :

Multicopter Position Control	EKF2_AID_MASK	24	Integer bitmask controlling data fusion and aiding methods
DShot	EKF2_ANGERR_INIT	5.730 deg	1-sigma tilt angle uncertainty after gravity vector alignment
EKF2	EKF2_ARSP_THR	0.0 m/s	Airspeed fusion threshold

Multicopter Position Control	EKF2_HGT_MODE	Vision	Determines the primary source of height data used by the EKF
DShot	EKF2_IMU_POS_X	0.000 m	X position of IMU in body frame (forward axis with origin relative to vehicle centre of gravity)
EKF2	EKF2_IMU_POS_Y	0.000 m	Y position of IMU in body frame (right axis with origin relative to vehicle centre of gravity)
	EKF2_IMU_POS_Z	0.000 m	Z position of IMU in body frame (down axis with origin relative to vehicle centre of gravity)

Enfin, pour que la position soit bien utilisée par le contrôleur de vol, il faut lui envoyer avec la commande relay :

```
rosrun topic_tools relay /mocap_node/Drone/pose /mavros/vision_pose/pose
On a alors une position qui sera calculé par le drone sur le topic
/mavros/local_position/pose
```

8 Connexion avec le jumeau numérique et tous autres applications

Pour connecter le drone au jumeau numérique on crée deux nodes udp_client. Un premier juste pour envoyer les positions du drone et un autre pour également recevoir les ordres de positions voulus par le jumeau numérique.

Il s'agit d'un simple client udp qui se connecte sur le serveur généré par le jumeau numérique et qui envoie les positions du drone sur une fréquence de 10Hz.

On peut les lancer de la façon suivante :

Envoye juste les données de position

```
roslaunch px4_offboard udp_client.launch ip_srv:="192.168.43.242"
port_srv:=1234 drone_nb:=1 topic_name:="drone_1" node_name:="client_udp_1"
```

Envoye les données de position et reçoit la nouvelle position à atteindre

```
roslaunch px4_offboard udp_client_offboard.launch ip_srv:="192.168.43.242"
port_srv:=1234 drone_nb:=1 topic_name:="drone_1" node_name:="client_udp_1"
```

9 Commande pour faire fonctionner les différents programmes

On connecte le raspberry et le pc sur le même réseau

Sur le Pc principal :

On ouvre plusieurs terminaux

On fait un ROS_MASTER_URI sur chaque avec l'ip de la machine

On fait un ROS_IP sur chaque avec l'ip de la machine

```
roscore  
roslaunch mocap_optitrack sample.launch  
rostopic relay /mocap_node/Drone/pose /mavros/vision_pose/pose
```

On peut alors lancer sur un dernier terminal quand le drone est prêt et mavros aussi :

```
roslaunch px4_offboard autopilot_pose.launch (par exemple)
```

Sur le Raspberry

On fait un ROS_MASTER_URI sur chaque avec l'ip de la machine

On fait un ROS_IP sur chaque avec l'ip du raspberry

On lance :

```
roslaunch mavros px4.launch fcu_url:="/dev/serial0:57600"  
      ges_url:="udp://@ip_machine:14550"
```

On peut lancer aussi :

```
roslaunch raspicam_node camerav2_640x480.launch
```

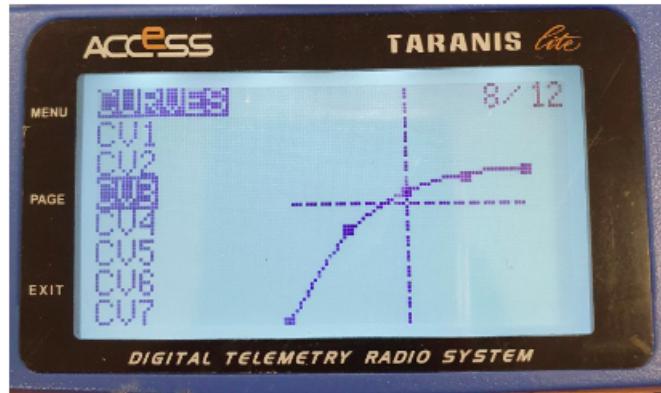
10 Notes

Je ne recommande pas de faire voler le drone de façon prolongée pour le moment sans changer les vis sous les moteurs par des plus courtes (5mm)

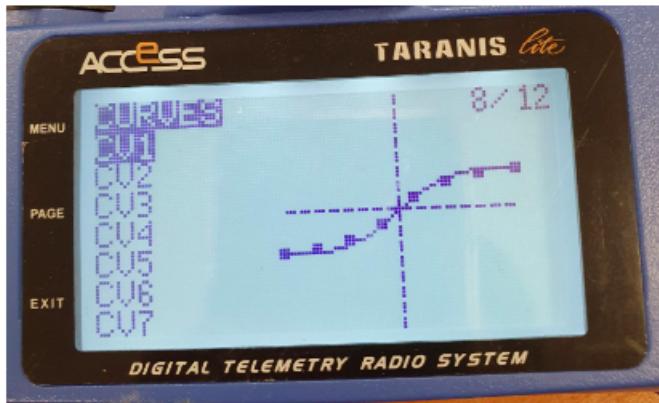
S'il y a un moteur qui ne tourne pas, vérifié le bobinage si le vernis n'a pas brûlé, sinon il peut s'agir de l'esc et là, c'est plus embêtant.

Pour le vol sur radiocommande, je conseille de modifier les entrées pour ne pas faire aller trop vite le drone et que le tout soit contrôlable. Je conseille les caractéristiques suivantes :

Pour le thrust :



Pour les autres directions :



Si pendant le contrôle distant avec mavros le drone ne s'arme pas vérifié dans le topic diagnostics s'il n'y a pas d'erreur (sur la batterie trop basse par exemple)

les erreurs sur le gps et le rc receiver ne sont pas vitales à l'armement du drone

Si le drone ne passe pas en offboard tenté sans la radiocommande, elle prend le dessus sur la connexion série par moment.

S'il perd la communication de sa position il va finir par arrêter les moteurs (c'est modifiable dans la rubrique sécurité sur Qgroundcontrol)

Enfin, pour faire fonctionner le tout je suis passé sur la version 1.11.3 du firmware de px4 car dans la version 1.12.3 le paramètre eflk2_aid_mask ne fonctionnait pas et ne me permettait alors pas de choisir uniquement l'optitrack comme source dans le calcul du positionnement.