

TP Utilisation plateforme TIRREX-UAV sur ROS

Amaury Nègre, Jonathan Dumon, Pierre Susbielle, Alexis Offermann

May 2024

1 Installation

- Si ce n'est pas déjà fait, téléchargez le fichier `tirrex_uav_tutorial.tar.gz` ici : <https://filesender.renater.fr/?s=download&token=cb258d02-4e37-4f2f-8911-527242b71ab9>
- Décompressez l'archive (`$ tar zxvf tirrex_uav_tutorial.tar.gz`) et suivez les instructions du fichier `README.md`

2 Découverte de la simulation Gazebo / ROS

2.1 Lancement de la simulation

- Lancez l'image docker à l'aide du script `run.bash` :

```
$ ./run.bash
```

- La simulation du drone “frelon” se lance grâce à un launch file ros, présent dans le package `tirrex_uav` :

```
$ ros2 launch tirrex_uav sim_frelon_launch.py
```

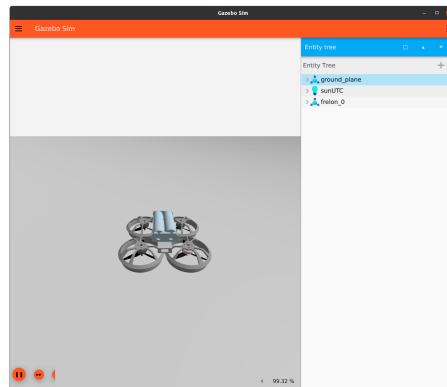


Figure 1: Fenêtre de la simulation Gazebo

2.2 Monitoring et configuration avec QGroundControl

2.2.1 Lancement de QGroundControl

Le logiciel “QGroundControl” permet de configurer et de monitorer les drones sous PX4. On peut également l'utiliser pour piloter un drone avec une manette de jeu.

Pour lancer QGroundControl, ouvrez tout d'abord un nouveau terminal et connectez vous sur la machine virtuelle avec le script `join.bash` puis lancez QGroundControl :

```
$ ./join.bash
$ ./QGroundControl.AppImage
```

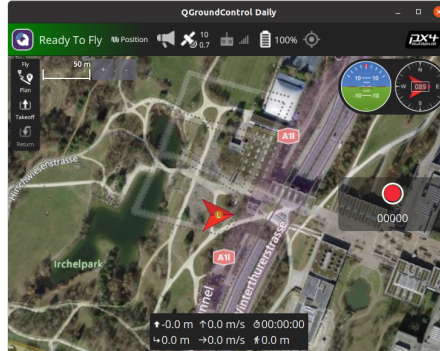


Figure 2: Fenêtre QGroundControl

2.2.2 Pilotage du drone à la manette

Pour configurer la manette, cliquez sur l'icône , puis sur le bouton *VehicleSetup* et allez dans l'onglet *Joystick*.

Remarque. *Le joystick n'est pas toujours reconnu sous docker, si c'est le cas, vous pouvez lancer QGroundControl sur le PC hôte en téléchargeant QGroundControl ici : https://docs.qgroundcontrol.com/master/en/qgc-user-guide/getting_started/download_and_install.html*

Activez le joystick (*Enable joystick input*), faites la calibration, puis assignez les fonctions suivantes sur 4 boutons :

- **Arm** : arme le drone pour pouvoir décoller
- **Disarm** : désarme le drone (une fois au sol)
- **Position** : passe en mode de vol "position"
- **Offboard** : passe en mode "offboard" (envoi de commande par un PC)


Vous devriez maintenant être capable de piloter le drone simulé :

1. Passez en mode **position** (mode par défaut).
2. Armez le drone avec le bouton **arm**. Les hélices devraient se mettre à tourner.
3. Faites décoller le drone en augmentant le throttle (joystick gauche vers le haut).
4. Enjoy !

Remarque. *Si vous mettez trop de temps à décoller, le drone se désarme automatiquement, il faudra donc le réarmer pour pouvoir décoller.*

2.2.3 Configuration du drone

Par défaut, le drone utilise le GPS comme référence de hauteur, cependant le GPS simulé est un GPS standard avec une erreur de l'ordre du mètre, ce qui est trop pour permettre un contrôle fin. On va donc utiliser la position fournie par le simulateur comme référence d'altitude. Celle-ci est envoyée à PX4 comme un capteur d'odométrie visuelle.

Pour changer ce paramètre, il faut aller dans le menu *Vehicle Setup* (icone ) , puis dans l'onglet *Parameters*, rechercher le paramètre **EKF2_HGT_REF** et changer la valeur en *Vision* (3). Il sera peut-être nécessaire de redémarrer le simulateur pour que le paramètre soit pris en compte.

2.2.4 Visualisation des messages mavlink

Il est possible de visualiser les messages mavlink en allant dans le menu *Analyse Tools / MAVLink Inspector*. Vous pouvez en particulier regarder le message **LOCAL_POSITION_NED** qui indique la position du drone dans le repère NED.

2.3 Visualisation des données ROS

Nous allons maintenant visualiser les données sous ROS. Pour cela, ouvrez une nouvelle console et listez les topics ros :

```
$ ./join.sh
$ ros2 topic list
```

Les topics commençant par “/fmu/out” sont les données sortant du contrôleur de vol PX4 et les topics commençant par “/fmu/in” sont les données que reçoit le contrôleur de vol.

Pour afficher l'odométrie complète du drone, utilisez la commande suivante :

```
$ ros2 topic echo /fmu/out/vehicle_odometry
```

3 Programmation d'un noeud de contrôle

L'objectif de cette partie est d'écrire une boucle de contrôle en position à l'aide d'un noeud ROS.

3.1 Classe python *uav_proxy*

Afin de faciliter la gestion des messages PX4 et les changements de repère (PX4: NED / ROS: ENU) nous avons développé une librairie python notée *uav_proxy*

Pour écrire un noeud de contrôle de drone en python, il faut écrire une classe qui hérite de la classe `UAVProxy.UAVProxyMicroros`. Cette classe possède les attributs et méthodes suivantes :

position : position du drone (repère NED)

orientation : orientation du drone (repère FLU / NED)

velocity : vitesse du drone (repère NED)

offboard : True si le drone est en mode “offboard”

armed : True si le drone est armé

sendAttitudeCommand(quat, thrust) : envoie une commande en attitude + thrust

sendRateCommand(rate, thrust) : envoie une commande en vitesse de rotation + thrust

sendPositionCommand(pos, yaw) : envoie une commande en position + orientation

3.2 Contrôle du drone via le mode position de PX4

Dans un premier temps, on va utiliser la boucle de commande en position interne à PX4.

- Ouvrez et analysez le fichier `tirrex_uav_tutorial/simple_controller_node.py`
Le code permet de créer un noeud ros qui hérite de la classe `UAVProxy.UAVProxyMicroros`. Ce noeud s'abonne au topic `/goal_pose` et crée un timer à 100hz qui envoie une commande en position au drone en fonction du goal.
- Pour lancer ce noeud, il faut tout d'abord compiler le package ros dans la machine docker puis lancer le launch file `launch/simple_control_launch.py` :

```
$ ./join.bash
$ cd tirrex_uav_ws
$ colcon build
$ source ./install/setup.bash
$ ros2 launch tirrex_uav_tutorial simple_control_launch.py
```

- Pour que le drone bouge, il faut qu'une nouvelle pose soit publiée sur le topic `/goal_pose` et que le drone soit en mode "offboard". Pour publier une pose sur le topic `/goal_pose`, on peut utiliser `rviz2` et le bouton "2D Goal Pose" :

```
$ ./join.bash
$ rviz2
```

Remarque. *Vou pouvez ajouter la visualisation de l'origine en cliquant sur "Add" puis "Axes". Il est également possible de visualiser la position du drone en changeant la valeur de Fixed Frame par "frelon_0/odom" et en rajoutant la visualisation du topic "/odom" ("Add" / "By topic" / "/odom [Odometry]").*

- Enfin, pour passer en mode offboard, relancez `QGroundControl` (si ce n'est pas déjà fait), faites décoller le drone avec la manette et appuyez sur le bouton `offboard`.

3.3 Contrôle du drone par des commandes en attitude/thrust

Pour contrôler le drone en position à l'aide de commande en attitude/thrust, nous allons utiliser un simple contrôleur linéaire.

- On note $X_{uav} = [x, y, z, v_x, v_y, v_z, \Psi]^T$ l'état du drone et $X_{ref} = [x_r, y_r, z_r, 0, 0, 0, \Psi_r]^T$ la position et l'orientation de référence.
- La commande linéaire en attitude + thrust est calculée de la manière suivante :

$$\Phi_d = \sin(\Psi) \cdot [k_p \cdot (x_{ref} - x) - k_d \cdot v_x] - \cos(\Psi) \cdot [k_p \cdot (y_{ref} - y) - k_d \cdot v_y] \quad (1)$$

$$\Theta_d = \cos(\Psi) \cdot [k_p \cdot (x_{ref} - x) - k_d \cdot v_x] + \sin(\Psi) \cdot [k_p \cdot (y_{ref} - y) - k_d \cdot v_y] \quad (2)$$

$$\Psi_d = \Psi_{ref} \quad (3)$$

$$T_d = T_{hover} + (k_{tp} \cdot (z - z_{ref}) + k_{td} \cdot v_z) \quad (4)$$

Où k_p , k_d , k_{tp} et k_{td} sont des gains à paramétrer (on peut commencer avec des valeurs de 0.1 pour tous les gains). T_{hover} correspond à la valeur de thrust qui compense uniquement la gravité, pour le drone simulé, sa valeur est de 0.69.

- Il faut ensuite saturer ces valeurs :

$$\Phi'_d = \text{saturate}(\Phi_d, -0.5, 0.5) \quad (5)$$

$$\Theta'_d = \text{saturate}(\Theta_d, -0.5, 0.5) \quad (6)$$

$$\Psi'_d = \Psi_d \quad (7)$$

$$T'_d = \text{saturate}(T_d, 0, 1) \quad (8)$$

- Pour tester ce contrôle, éditez le fichier `tirrex_uav_tutorial/attitude_controller_node.py` et implémentez les équations ci-dessus dans la fonction `timer_callback`.

L'envoi de la commande au drone se fait grâce à la fonction `sendAttitudeCommand(att, thrust)` qui prend un quaternion pour l'orientation (il faut donc convertir $[\Phi'_d, \Theta'_d, \Psi'_d]$ en quaternion, et le thrust normalisé (entre 0 et 1).

- Pour exécuter le code, il faut recompiler le package et lancer le launchfile `attitude_control_simu_launch.py`:

```
$ ./join.bash
$ cd tirrex_uav_ws
$ colcon build
$ source ./install/setup.bash
$ ros2 launch tirrex_uav_tutorial attitude_control_simu_launch.py
```

- Pour analyser les performances du contrôleur, vous pouvez analyser les données avec `plotjuggler`:

```
$ ros2 run plotjuggler plotjuggler
```

A Annexe : formalisme contrôle linéaire

$$m\vec{v} = -mg \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \implies \begin{cases} \ddot{x} = \frac{T}{m} (c_\phi s_\theta c_\psi + s_\phi s_\psi) \\ \ddot{y} = \frac{T}{m} (c_\phi s_\theta s_\psi - s_\phi c_\psi) \\ \ddot{z} = -g + \frac{T}{m} c_\phi c_\theta \end{cases} \quad (9)$$

En considérant le modèle non linéaire 9, on identifie l'état $\xi = [x \ y \ z \ \dot{x} \ \dot{y} \ \dot{z}]^T$ et le vecteur de contrôle $u = [\phi \ \theta \ T]^T$

$$\dot{\xi} = f(\xi, u)$$

Linéarisation autour de l'équilibre en vol stationnaire, $\xi_0 = [0 \ 0 \ 0 \ 0 \ 0 \ 0]^T$ et $u_0 = [0 \ 0 \ mg]^T$.

$$\dot{\xi} \simeq \xi_0 + \left. \frac{\partial f}{\partial \xi} \right|_{\xi_0, u_0} (\xi - \xi_0) + \left. \frac{\partial f}{\partial u} \right|_{\xi_0, u_0} (u - u_0) \quad (10)$$

Définitions $\tilde{\xi} = \xi - \xi_0$ et $\tilde{u} = u - u_0$.

On arrive alors au système linéaire suivant :

$$\left\{ \begin{aligned}
 \begin{bmatrix} \ddot{\tilde{x}} \\ \ddot{\tilde{y}} \\ \ddot{\tilde{z}} \\ \ddot{\tilde{x}} \\ \ddot{\tilde{y}} \\ \ddot{\tilde{z}} \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \dot{\tilde{x}} \\ \dot{\tilde{y}} \\ \dot{\tilde{z}} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ g * s_\psi & g * c_\psi & 0 \\ -g * c_\psi & g * s_\psi & 0 \\ 0 & 0 & \frac{1}{m} \end{bmatrix} \cdot \begin{bmatrix} \tilde{\phi} \\ \tilde{\theta} \\ \tilde{T} \end{bmatrix} \\
 &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \dot{\tilde{x}} \\ \dot{\tilde{y}} \\ \dot{\tilde{z}} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & \frac{1}{m} \end{bmatrix} \cdot \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \tilde{\phi} \\ \tilde{\theta} \\ \tilde{T} \end{bmatrix} \\
 &= A \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \dot{\tilde{x}} \\ \dot{\tilde{y}} \\ \dot{\tilde{z}} \end{bmatrix} + B f(\psi) \begin{bmatrix} \tilde{\phi} \\ \tilde{\theta} \\ \tilde{T} \end{bmatrix}
 \end{aligned} \right.$$

Le système est linéaire en $\tilde{\xi}$ et $v = f(\psi) \begin{bmatrix} \tilde{\phi} \\ \tilde{\theta} \\ \tilde{T} \end{bmatrix}$, avec $f(\psi)$ une matrice de rotation 3D, "projetant"

le contrôle dans le repère du drone. On propose alors par exemple un contrôle par retour d'état :

$$\begin{aligned}
 v &= -K(\tilde{\xi} - \tilde{\xi}_{ref}) \\
 &= -[K_p^{3,3} \quad K_d^{3,3}] * \left(\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \dot{\tilde{x}} \\ \dot{\tilde{y}} \\ \dot{\tilde{z}} \end{bmatrix} - \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \dot{\tilde{x}} \\ \dot{\tilde{y}} \\ \dot{\tilde{z}} \end{bmatrix}_{ref} \right)
 \end{aligned}$$

Enfin, on retrouve u par

$$\begin{bmatrix} \tilde{\phi} \\ \tilde{\theta} \\ \tilde{T} \end{bmatrix} = f^{-1}(\psi)v = f^T(\psi)v = - \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} * K * \left(\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \dot{\tilde{x}} \\ \dot{\tilde{y}} \\ \dot{\tilde{z}} \end{bmatrix} - \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \dot{\tilde{x}} \\ \dot{\tilde{y}} \\ \dot{\tilde{z}} \end{bmatrix}_{ref} \right)$$