Autonomous Drone Software E01: OFFBOARD Control and Gazebo Simulation

OFFBOARD mode (Guided mode in the case of Ardupilot) is a powerful function that allows you to control your drone with companion computers, and we will be testing it in Gazebo simulation environment.

We suggest developers to install GAAS mirror to use the project:

→ GAAS v0.7 Release Mirror - x64 /handy-tools/v0.7-release-mirror-x86

Join our slack

Your drone can be controlled in OFFBOARD mode by a companion computer primarily through a set of MAVROS Commands (don't worry, we will dive into that later) which is a higher level wrapper of MAVLink API that saves us a great deal of efforts when controlling your drone. With the help of MAVROS, we can easily realize many functions such as Takeoff, Land, Position Target, Yaw control etc.

In this tutorial, we will begin with a simple yet powerful function that runs on your companion computer and controls your drone in Python! To do this, let's start with setting up your environment.

> ⓘ This tutorial is tested on Ubuntu 16.04 LTS with ROS-Kinetic, you can also choose to use Ubuntu 18.04 LTS with ROS-Melodic

Environment Setup

You can choose to set up the environment by building each package from source or you can directly use a Docker to run everything. For better performance, we recommend using the first method.

1. Setup from Source

General Dependencies

To use all provided utilities, there are some packages we need to install first:

```
sudo apt install -y \
  ninja-build \
  exiftool \
  python-argparse \
  python-empy \
  python-toml \
  python-numpy \
  python-yaml \
  python-dev \
  python-pip \
  ninja-build \
  protobuf-compiler \
  libeigen3-dev \
  genromfs
```

```
pip install \
  pandas \
  jinja2 \
  pyserial \
```

```
  cerberus \
  pyulog \
  numpy \
  toml \
  pyquaternion
```

ROS-Kinetic

ROS (Robot Operating System) is a powerful framework that contains many libraries and tools that can help you with developing robots, and we will be using ROS throughout our tutorial.

To install ROS Kinetic, please follow the instructions below:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6B
sudo apt-get update
sudo apt-get install ros-kinetic-desktop-full
sudo rosdep init
rosdep update
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
sudo apt install python-rosinstall python-rosinstall-generator python-catkin-tools

# install ros-gazebo plugins
sudo apt install ros-kinetic-gazebo-* \
```

> Visit http://wiki.ros.org/kinetic/Installation/Ubuntu for more information.

If you would like to use Ubuntu 18.04 LTS with ROS-Melodic , just replace `ros-kinetic` with `ros-melodic` . For example , replace:

```
sudo apt-get install ros-kinetic-desktop-full
```

as:

```
sudo apt-get install ros-melodic-desktop-full
```

Once you have installed ROS, you can test it by opening a terminal and type:

```
roscore
```

If you have installed it properly, you will see something like:

```
... logging to /home/.ros/log/6a1b2330-2eb3-11e9-a39c-9cb6d0e498fb/roslaunch-gishr-
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://XPS-15:44361/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES
```

```
auto-starting new master
process[master]: started with pid [4463]
ROS_MASTER_URI=http://XPS-15:11311/

setting /run_id to 6a1b2330-2eb3-11e9-a39c-*********
process[rosout-1]: started with pid [4476]
started core service [/rosout
```

Generate your catkin workspace. This is the place where you will be storing ROS projects such as MAVROS etc.

```
mkdir -p ~/catkin_ws/src
```

MAVROS

MAVROS is a communication node based on MAVLink for ROS that is specially designed for communication between the drone and the companion computer. To install it, follow the following instructions:

```
# you can either choose to use apt or build from source

# method 1
sudo apt-get install ros-kinetic-mavros ros-kinetic-mavros-extras
# and instal geographic lib by :
wget https://raw.githubusercontent.com/mavlink/mavros/master/mavros/scripts/install
sudo ./install_geographiclib_datasets.sh


# method 2
cd ~/catkin_ws
catkin init && wstool init src

rosinstall_generator --rosdistro kinetic mavlink | tee /tmp/mavros.rosinstall
rosinstall_generator --upstream mavros | tee -a /tmp/mavros.rosinstall

wstool merge -t src /tmp/mavros.rosinstall
wstool update -t src -j4
rosdep install --from-paths src --ignore-src -y

sudo ./src/mavros/mavros/scripts/install_geographiclib_datasets.sh
sudo apt install ros-kinetic-catkin python-catkin-tools

catkin build
```



```
visit the following web for more information:
https://github.com/mavlink/mavros/blob/master/mavros/README.md#installatic
```

PX4 Firmware

We will be using PX4 v1.8.0 throughout our tutorial.

```
cd ~/catkin_ws/src
#it could take a while
git clone https://github.com/PX4/Firmware.git
cd Firmware
git checkout v1.8.0
make posix_sitl_default gazebo
```

Now you should see a window pop out and a drone is centered in the middle of the environment, but let's close the window for now.

Modifying your 'bashrc' so that your environment remains the same every time you open a new terminal:

```
# Use your favorite editor, we will be using gedit
# NOTE: you will need to use ROOT to edit bashrc
sudo gedit ~/.bashrc

# in the new terminal, scroll down to the bottom and add the following lines to the
# if you used apt to install MAVROS (method 1), there would be no setup.bash under
# and you can safely neglect the first step.
source ~/catkin_ws/devel/setup.bash
source ~/catkin_ws/src/Firmware/Tools/setup_gazebo.bash ~/catkin_ws/src/Firmware/ ~
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/catkin_ws/src/Firmware
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/catkin_ws/src/Firmware/Tools/sitl_gazeb
```

Open a new terminal and type the following command:

```
# This will launch Gazebo simulation
roslaunch px4 posix_sitl.launch
```

In another terminal:

```
# This will launch MAVROS
roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.1:14557"
```

Or you can simply use:

```
# This will launch MAVROS and Gazebo simulation at the same time
roslaunch px4 mavros_posix_sitl.launch
```

A window should pop out which looks like:

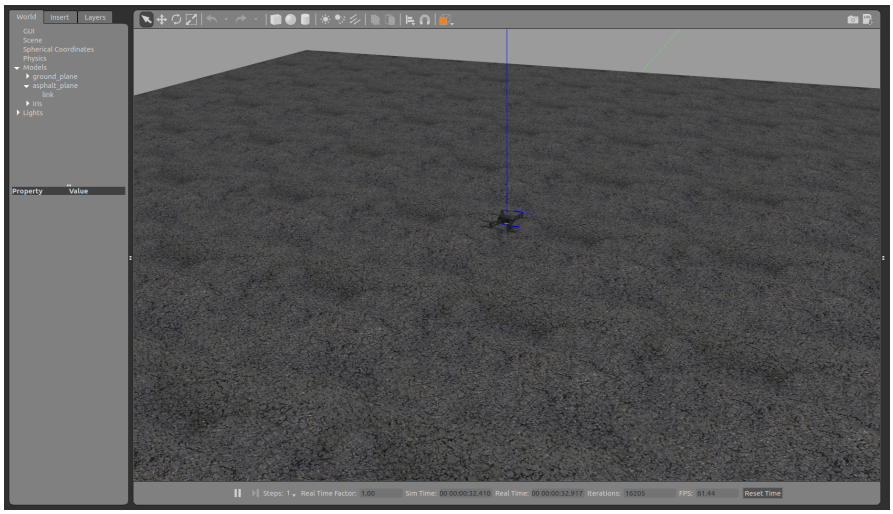


Figure 1, Gazebo and PX4 simulation environment in an empty world

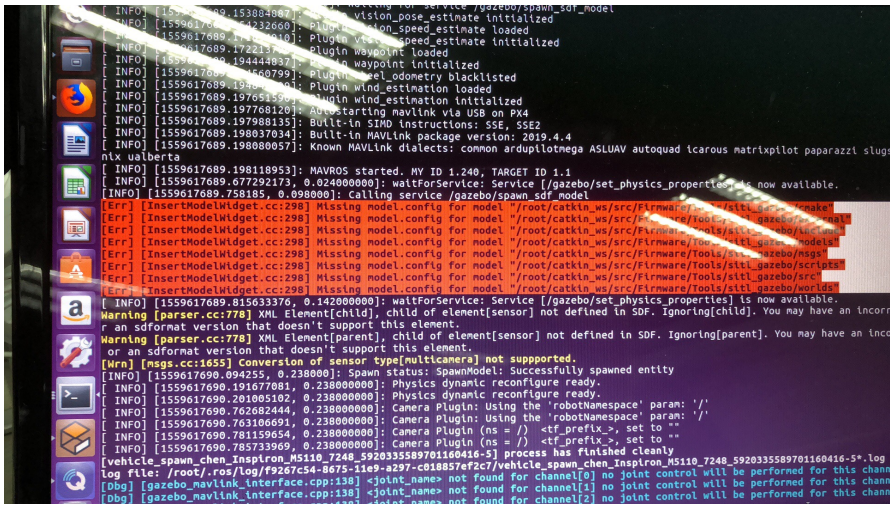
In a new terminal, if you type:

```
rostopic echo /mavros/state
```

You should see:

```
header:
  seq: 1
  stamp:
    secs: 730
    nsecs: 280000000
  frame_id: ''
connected: True
armed: False
guided: False
mode: "MANUAL"
system_status: 3
---
```

If you see the above results and 'connected: True', it means your simulation environment is properly set up and MAVROS connection is successful.



> While running Gazebo, if everything works properly with the exception of the above Err message, the Err message itself DOES NOT affect the performance of Gazebo.
>
> If you happen to know how to resolve this Err message, please submit a PR on GitHub to help us make it go away.

2. Using Docker

If you'd like to skip all the processes mentioned in the last part, you can quickly jump to the next part by using provided Docker. Docker is easy to use but resource-consuming, meaning the frame rates might be slow, but VNC clients are widely available on multiple platforms such as MacOS, Windows and Android etc.

> It is recommended to set up your environment using the first procedure.

Using Virtual Network Computing (VNC) and Pulling Docker Image

In order to use GUI applications within Docker environment, you will need to install VNC-viewer in your local environment.

For Linux systems, you can directly download the following binary file to your local environment by:

```
wget https://www.realvnc.com/download/file/viewer.files/VNC-Viewer-6.19.107-Linux-x64
```

You can checkout this website for more information:

```
https://www.realvnc.com/en/connect/download/viewer/
```

Then:

```
# NOTE the exact file name might differ
chmod +x VNC-Viewer-6.19.107-Linux-x64
./VNC-Viewer-6.19.107-Linux-x64
```

The installation is quite simple and straight forward, and we will skip this process for other systems.
After installing VNC-viewer, we need to pull the Docker Image to your local environment by:

```
# NOTE you probably need ROOT privilege
docker pull gaas/mavros-gazebo7-px4
```

Now, to start a Container, open a terminal and type:

```
# It might take a while depending on your hardware setup
docker run -p 6080:80 -p 5900:5900 gaas/mavros-gazebo7-px4:latest
```

In another terminal:

```
./VNC-Viewer-6.19.107-Linux-x64
```

You should see a window which looks like:

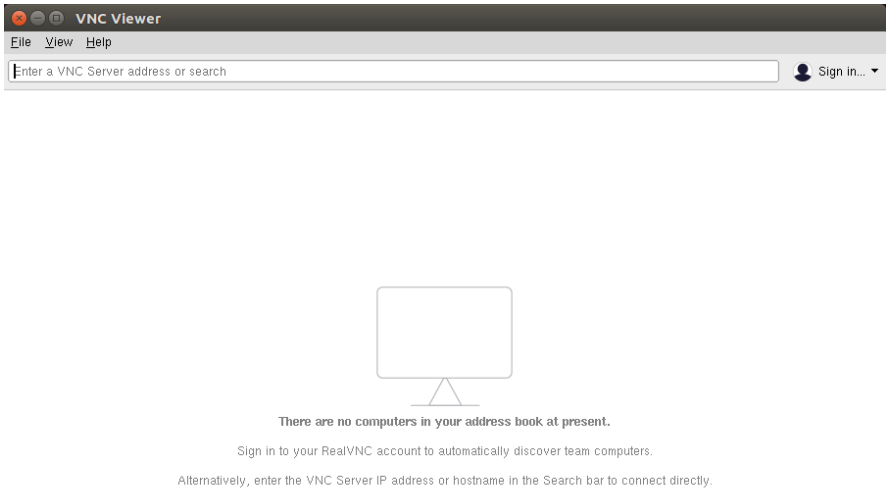


Figure 2, VNC viewer window

Enter the following address:

```
127.0.0.1:5900
```

Hit enter, an interactive window will pop out and you can use it just like any other Ubuntu system:

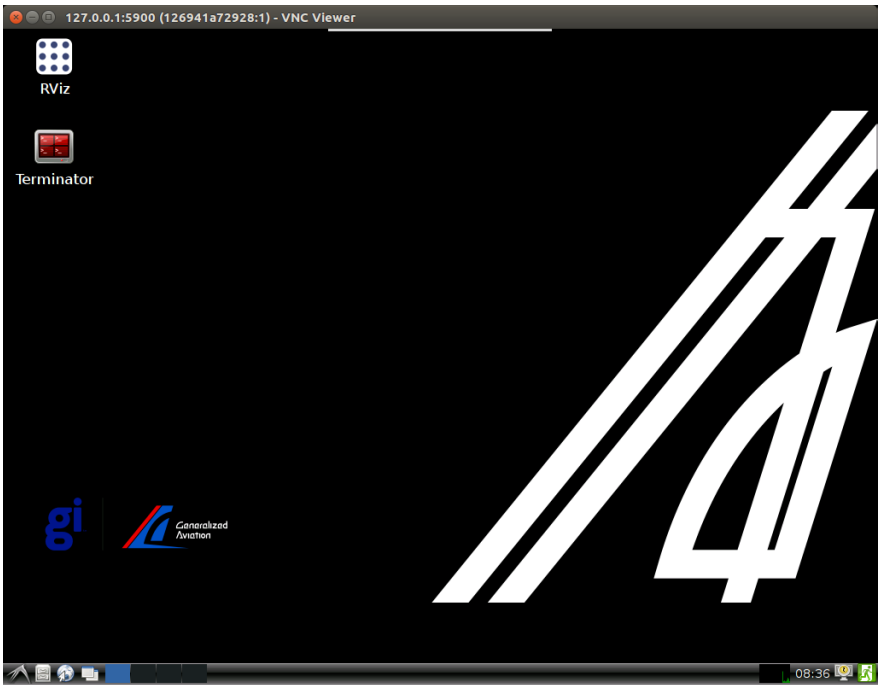


Figure 3, Docker Image

Everything you need for this tutorial can be found in the following folder:

```
/root/gi/GAAS/demo/tutorial_1
```

At last, you will need to manually conduct the following commands before running the simulation every time you use a Docker:

```
cd /root/gi/px4/Firmware && make posix_sitl_default gazebo
```

Now, you can jump to the last section and start doing practical!

OFFBOARD Control

> You can skip this part and jump to Control Your Drone if you are using Docker.

Before diving into the python scripts, let's give it a go. Firstly, clone the scripts to your local environment. It

contains custom drone models, world models as well as many other things that will help you build your drone, for now we will only be looking at python MAVROS examples, but feel free to test other things if you are interested.

```
git clone git@github.com:generalized-intelligence/GAAS.git
```

You will see four folders:

- demo: everything we need for this tutorial can be found in this folder;
- hardware: it contains relevant information on how to build the hardware;
- simulator: it contains drone simulation models, world models etc. ;
- software: it contains other packages such as ObstacleMap, SLAM, Local and Global path planner etc.

You will find more information in each folder's README.MD.

Next, add customized model path to bashrc by:

```
echo "export GAZEBO_MODEL_PATH=${GAZEBO_MODEL_PATH}:(GAAS_PATH)/simulator/models" >>
```

Copy models and config files to corresponding PX4 folders:

```
# if you cloned GAAS to a different place
cp -r (GAAS_PATH)/simulator/models/* ~/catkin_ws/src/Firmware/Tools/sitl_gazebo/mod
cp -r (GAAS_PATH)/GAAS/simulator/worlds/* ~/catkin_ws/src/Firmware/Tools/sitl_gazeb
cp -r (GAAS_PATH)/GAAS/simulator/posix-config/* ~/catkin_ws/src/Firmware/posix-conf
```

Control Your Drone

If you are using Docker, remember to conduct the following command every time you try to run the simulation:

```
cd /root/gi/px4/Firmware && make posix_sitl_default gazebo
```

A Gazebo simulation environment should pop out but let's close the window for now.

Next, before executing any of the following scripts, remember to launch the simulation first:

```
roslaunch px4 mavros_posix_sitl.launch
```

And don't forget to check MAVROS connection status by:

```
rostopic echo /mavros/state
```

If MAVROS connection is successful, in a new terminal, change directory to the place where you put GAAS and go to DEMO by:

```
# if you are using docker
cd /root/gi/GAAS/demo/tutorial_1/1_px4_mavros_offboard_controller
python px4_mavros_run.py

# if you installed everything from apt
```

```
cd (GAAS_PATH)/demo/tutorial_1/1_px4_mavros_offboard_controller
python px4_mavros_run.py
```

You will see a drone gradually takes off to 3 meters high and hovers at this height. In another terminal:

```
python commander.py
```

You will see the drone moves in the following order:

1. moves 1 meter to the right;
2. turns 90 degrees counter-clockwise;
3. land.

Or you can open a new terminal and control the drone using provided API like:

```
# in folder '1_px4_mavros_offboard_controller'
python

# import packages
from commander import Commander
import time

# create Commander instance
con = Commander()

# control the drone to move 1 meter to the right
con.move(1,0,0)
# wait for 2 seconds
time.sleep(2)

# control the drone to move 1 meter to the front
con.move(0,1,0)
# wait for 2 seconds
time.sleep(2)

# control the drone to move 1 meter to the left
con.move(-1,0,0)
# wait for 2 seconds
time.sleep(2)

# control the drone to move 1 meter to the back
con.move(0,-1,0)
# wait for 2 seconds
time.sleep(2)

# control the drone to move 1 meter above
con.move(0,0,1)
# wait for 2 seconds
time.sleep(2)

# land
con.land()
```

You will see the drone moves in a square, goes up and then land. You have probably found that the drone was moving relative to its "current" position and the movement frame is defined as BODY_OFFSET_ENU, or FLU (Forward, Left and Up, see reading materials) which means each movement command will control the drone to move relative to its body frame. To illustrate, the first parameter in function move(x, y, z) means going Forward, the second means going to the left and the third means going up. The movement frame is set to BODY_OFFSET_ENU by Default. If you wish to use LOCAL_ENU which means the movement is relative to its takeoff position, you can add a fourth parameter like:

```
con = Commander()

# for BODY_OFF_SET_ENU or FLU frame
con.move(x,y,z)

# for LOCAL_ENU frame
```

```
con.move(x,y,z,BODY_OFF_SET_ENU=False)
```

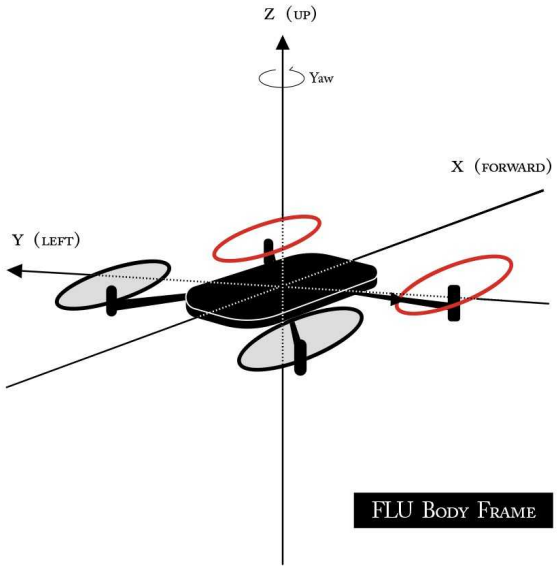


Figure 4, FLU body frame

Now you can checkout other APIs and test them in the simulation environment.

In the next tutorial, we will talk about how to build a 3D model using OFFBOARD mode and popular SfM algorithms in simulation.

If you find any bug or need more help, please let us know either by opening an issue or through our facebook group.

Give us a Star on Github if you find this tutorial useful.

Support us on Patreon

Help us to understand what you want to learn in the upcoming tutorials by fill out this three-question survey.

Reading Materials

We find the following materials very useful during developing our work, so if you are interested, you can have a look.

1. General ROS introduction: http://wiki.ros.org/
2. MAVROS: http://wiki.ros.org/mavros
3. MAVLINK: https://mavlink.io/
4. Gazebo: http://gazebosim.org/
5. rep-105: https://www.ros.org/reps/rep-0105.html
6. https://github.com/PX4/Devguide/blob/master/en/ros/external_position_estimation.md