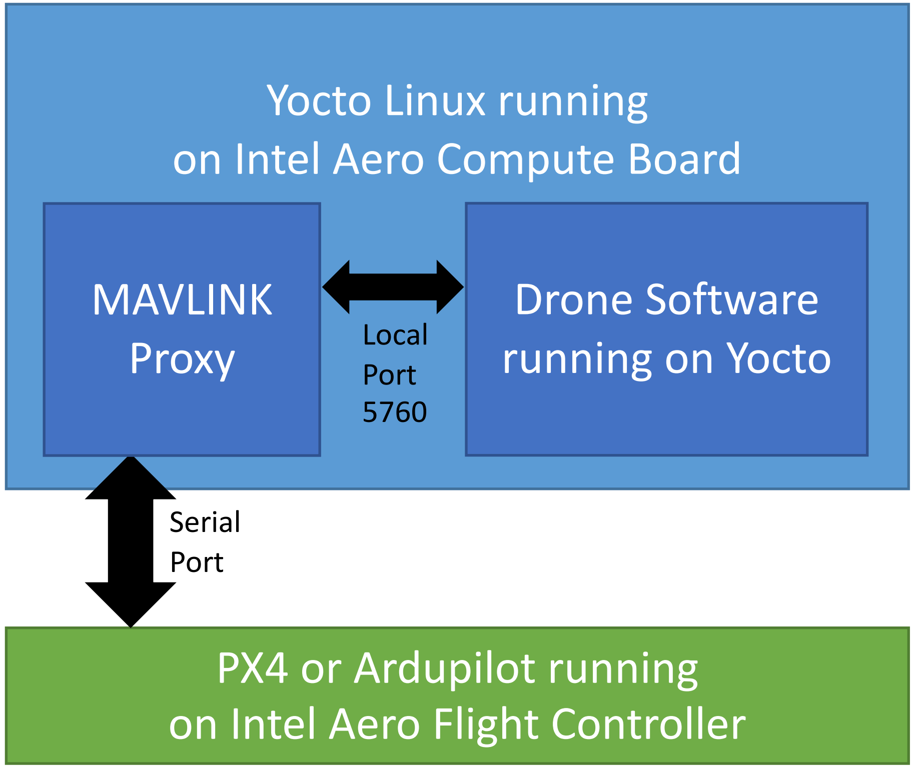
# Autonomous Drone Engineer

## D1 – Hello World – Motor Control

Paul.Guermonprez@intel.com

*Autonomous Drone Solutions Architect*

# Architecture Summary



To summarize the software architecture described in the module B4:

- Flight Controller and Compute Board are linked with a **serial port**
- The protocol is the standard **MAVLINK**
- A proxy is exposing this serial port as a network socket on **port 5760**

**Connect your code to the socket 5760 with tcp using the MAVLINK protocol**

# Hello World – Python-MAVLINK

# Hello World in Python-MAVLINK

While connected as root on Intel Aero, Aero being connected to Internet:

Get the code from Intel’s github repository:

```
git clone https://github.com/01org/mavlink-router.git
cd mavlink-router/
```

and execute a simple example:

```
python examples/heartbeat-print-tcp.py 127.0.0.1:5760
```

you’ll see messages like:

```
HEARTBEAT {type : 2, autopilot : 12, base_mode : 29, custom_mode : 84148224,
system_status : 3, mavlink_version : 3}
HEARTBEAT {type : 2, autopilot : 12, base_mode : 29, custom_mode : 84148224,
system_status : 3, mavlink_version : 3}
```

**You’re connected to the flight controller!**

# Hello World in Python-MAVLINK

We’re using the mavlink library in Python: **pymavlink** (already installed)

```
import pymavlink.mavutil as mavutil
```

We’re connected to the local IP with a **tcp socket on port 5760** (I’m replacing sys.argv[1] by it’s value):

```
mav = mavutil.mavlink_connection('tcp:127.0.0.1:5760')
```

Waiting for **heartbeat**:

```
mav.wait_heartbeat()
```

# Arming Motors in Python-MAVLINK

**IMPORTANT: REMOVE THE PROPELLERS FROM THE MOTORS FIRST**

Edit the previous file heartbeat-print-tcp.py,
add “import time” to the required imports,
then add after the wait_heartbeat the 2 following lines: ARM and DISARM

```
mav.mav.command_long_send(mav.target_system, mav.target_component,
mavutil.mavlink.MAV_CMD_COMPONENT_ARM_DISARM,0,1,0,0,0,0,0,0)
time.sleep(5)
mav.mav.command_long_send(mav.target_system, mav.target_component,
mavutil.mavlink.MAV_CMD_COMPONENT_ARM_DISARM,0,0,0,0,0,0,0,0)
```

**You’re spinning the motors for 5 seconds**

# Summary of Python-MAVLINK

Summary:

- Use **TCP** sockets on port **5760**
- After the initial connection, wait for the first **heartbeat**
- In MAVLINK, messages are endoded as **frames**
- Frames have names: MAV_CMD_COMPONENT_ARM_DISARM
- And arguments (here it’s 1 for ARM, 0 for DISARM)
- The MAVLINK library is **easy** to use
- There’s **interfaces** for Python but also other languages

# Hello World – Python-DroneKit

# DroneKit

It’s important to know the basics of MAVLINK,
as it the base of all communications with the Flight Controllers.

But coding frames with python-mavlink is not developer friendly.

DroneKit, developed by 3D Robotics (http://3drobotics.com),
is one of the friendly python abstractions available under Apache v2
Licence : http://python.dronekit.io

To install on Intel Aero:

```
pip install dronekit
```

# Hello World in Python-DroneKit

```
from dronekit import connect, VehicleMode, LocationGlobalRelative
import time

vehicle = connect('tcp:127.0.0.1:5760', wait_ready=True)

print "Arming motors: "
vehicle.mode    = VehicleMode("GUIDED")
vehicle.armed   = True

while not vehicle.armed:
        print "  Waiting for arming to be finished "
        time.sleep(1)

print "Keeping motors armed for 5s "
time.sleep(5)

print "Disarming "
vehicle.armed   = False
```

# Summary of Python-DroneKit

Summary:

- Using the regular Python-MAVLINK as base
- Same connection (tcp 5760) as all the other methods
- Developer friendly and well documented: http://python.dronekit.io/guide/index.html

# Hello World – Python-MAVProxy

# MAVProxy

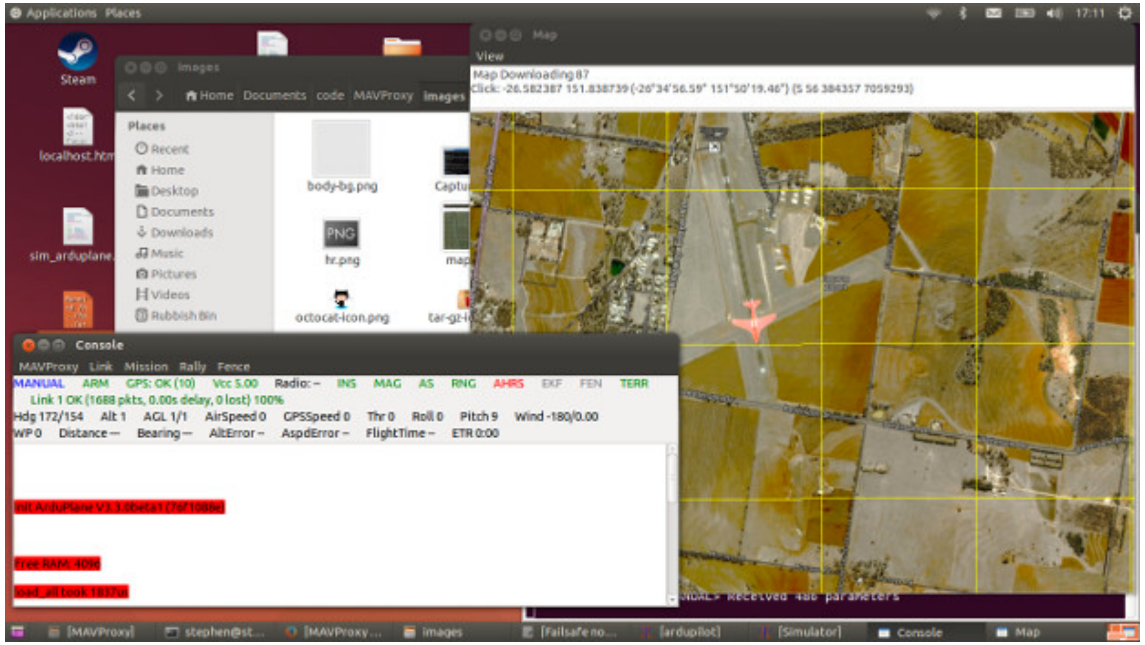
On top of being a developer friendly layer on top of MAVLINK, MAVProxy was designed to bridge the gap between programming-only libraries like DroneKit and graphical-only tools like QGroundControl.

Check: [http://ardupilot.github.io/MAVProxy](http://ardupilot.github.io/MAVProxy)

Some people use it on a remote computer to control the drone,
but you an use on the drone itself
for autonomous drone development.

To install on Intel Aero:

pip install MAVProxy



```
Map Downloading 87
Click: -26.582387 151.838739 (-26°34'56.59" 151°50'19.46") (S 56 384357 7059293)
```

```
MANUAL   ARM   GPS: OK (10)   Vcc 5.00   Radio: --   INS   MAG   AS   RNG   AHRS   EKF   FEN   TERR
  Link 1 OK (1688 pkts, 0.00s delay, 0 lost) 100%
Hdg 172/154   Alt 1   AGL 1/1   AirSpeed 0   GPSSpeed 0   Thr 0   Roll 0   Pitch 9   Wind -180/0.00
WP 0   Distance --   Bearing --   AltError --   AspdError --   FlightTime --   ETR 0:00

Init ArduPlane V3.3.0beta1 (76f1088e)

Free RAM: 4096
load_all took 1837us
```

# Hello World in Python-MAVProxy

We’re launching the console on Intel Aero:

```
mavproxy.py  --master=tcp:127.0.0.1:5760 --quadcopter
```

And typing a few commands:

```
arm throttle
disarm
bat
```

# Summary of Python-MAVProxy

Summary:

- You can use it **locally** on Intel Aero as a shell
- You can use it **remotely** on your desktop for maps, joysticks and more
- Using the regular Python-MAVLINK as base
- Same connection (tcp 5760) as all the other methods
- Developer friendly and well documented: http://ardupilot.github.io/MAVProxy/html/

# Thanks

Paul.Guermonprez@intel.com