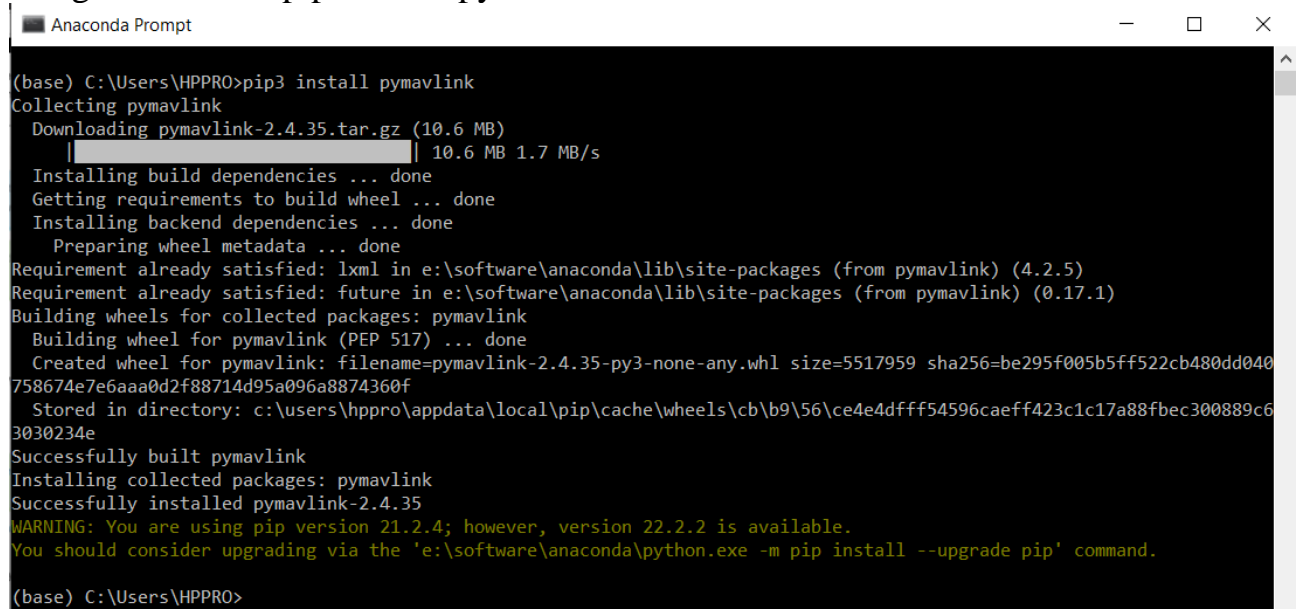


Software control of the Unmanned Aerial Vehicle

While the direct communication using QGroundControl is possible to perform necessary actions, even more advanced way of UAV control is to use the software that controls the processes concerned with the mission. This opportunity is provided by the MAVlink library directly or its variations for different programming languages. For instance, the pymavlink library is provided as a binding of MAVlink in Python. So, the objective of the current work is to establish the skills of programming the UAV using software library and to execute certain steps in the configuration of the UAV.

Workflow

First, speaking about the Python library that ensures necessary functionality, pymavlink is required to install. Hence, Python is required initially and then you can install the library using pip3 manager (Python 3 version is recommended). Install it using a command pip3 install pymavlink as shown below.



```
(base) C:\Users\HPPRO>pip3 install pymavlink
Collecting pymavlink
  Downloading pymavlink-2.4.35.tar.gz (10.6 MB)
    |#####| 10.6 MB 1.7 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
  Preparing wheel metadata ... done
Requirement already satisfied: lxml in e:\software\anaconda\lib\site-packages (from pymavlink) (4.2.5)
Requirement already satisfied: future in e:\software\anaconda\lib\site-packages (from pymavlink) (0.17.1)
Building wheels for collected packages: pymavlink
  Building wheel for pymavlink (PEP 517) ... done
  Created wheel for pymavlink: filename=pymavlink-2.4.35-py3-none-any.whl size=5517959 sha256=be295f005b5ff522cb480dd040758674e7e6aaa0d2f88714d95a096a8874360f
  Stored in directory: c:\users\hpro\appdata\local\pip\cache\wheels\cb\b9\56\ce4e4dfff54596caeff423c1c17a88fbec300889c63030234e
Successfully built pymavlink
Installing collected packages: pymavlink
Successfully installed pymavlink-2.4.35
WARNING: You are using pip version 21.2.4; however, version 22.2.2 is available.
You should consider upgrading via the 'e:\software\anaconda\python.exe -m pip install --upgrade pip' command.
(base) C:\Users\HPPRO>
```

As soon as the installation is done, you can transition to the writing of basic script that should verify connection between components.

Prepare Python script file with the following content:

```
from pymavlink import mavutil

# Start a connection listening on a UDP port
the_connection = mavutil.mavlink_connection('udpin:localhost:14540')

# Wait for the first heartbeat
# This sets the system and component ID of remote system for the link
the_connection.wait_heartbeat()
print("Heartbeat from system (system %u component %u)" %
      (the_connection.target_system, the_connection.target_component))
```

Before you can actually run the script, you are supposed to launch QGroundControl and then an instance of jmavsim simulator (previous lab work). During the start of the simulator, check the information that is shown in diagnostics messages. Find the number of port that the simulator uses for communications. This is the port that should be indicated in the script for the connection.

```
cygdrive/e/Software/px4/PX4-Autopilot

copy_res:

BUILD SUCCESSFUL
Total time: 0 seconds
Options parsed, starting Sim.
Starting GUI...
INFO [simulator] Simulator connected on TCP port 4560.
Init MAVLink
ERROR [px4_work_queue] setting sched params for wq:lp_default failed (134)
INFO [commander] Mission #2 loaded, 9 WPs, curr: 0
INFO [commander] LED: open /dev/led0 failed (22)
INFO [init] Mixer: etc/mixers/quad_w.main.mix on /dev/pwm_output0
INFO [init] setting PWM_AUX_OUT none
INFO [mavlink] mode: Normal, data rate: 4000000 B/s on udp port 18570 remote port 14550
INFO [mavlink] partner IP: 127.0.0.1
INFO [mavlink] mode: Onboard, data rate: 4000000 B/s on udp port 14580 remote port 14540
INFO [mavlink] mode: Onboard, data rate: 4000 B/s on udp port 14280 remote port 14030
INFO [mavlink] mode: Gimbal, data rate: 400000 B/s on udp port 13030 remote port 13280
```

When both QGroundControl and jmavsim tools are running, run the script in any convenient way. The example below shows launch from the command line. The code was saved to the file with the name “run1.py”.

```
MINGW64:/e/Work materials/Аерознімання та управління БПЛА/scripts

HPPRO@DESKTOP-24N52NA MINGW64 /e/Work materials/Аерознімання та управління БПЛА/scripts
$ python run1.py
Heartbeat from system (system 1 component 0)
```

What you should observe is a single diagnostic message that is received from the simulator. That message indicates that the connection is successful and you can move on to the next step.

Sending commands to the vehicle

Not only we can check the status of the device connected from the software part, but also commands submission is available. For instance, the device can be armed or disarmed from the code directly. **N.B.** Arming is the operation of submitting route data to the target.

First, the command is represented in MAVlink in the following way. The developer has a specific interface (function) that is used for commands submission [2]. The meaning of the command is defined by the ID which can be found here [3].

Message/Enum Summary

Message	Description
COMMAND_INT	Message for encoding a command (MAV_CMD). The message encodes commands into up to 7 parameters: parameters 1-4, 7 are floats, and parameters 5,6 are scaled integers. The scaled integers are used for positional information (scaling depends on the actual command value). The coordinate frame of positional parameters is explicitly specified in a frame field. Commands that require float-only properties in parameters 5, 6 cannot be sent in this message (e.g. commands where NaN has an explicit meaning).
COMMAND_LONG	Message for encoding a command (MAV_CMD). The message encodes commands into up to 7 float parameters. The coordinate frame used for positional co-ordinates is implementation dependent. Any command may be packaged in this message, but there may be some loss of precision for positional co-ordinates (latitude, longitude).
COMMAND_ACK	Command acknowledgement. Includes result (success, failure, still in progress) and may include progress information and additional detail about failure reasons.
COMMAND_CANCEL	Cancel a long running command.

The specific messages that can be sent using commands is described at [3]. The peculiar command that is used in our case is shown below.

MAV_CMD_COMPONENT_ARM_DISARM (400)

[Command] Arms / Disarms a component

Param (:Label)	Description	Values
1: Arm	0: disarm, 1: arm	<i>min</i> :0 <i>max</i> :1 <i>increment</i> :1
2: Force	0: arm-disarm unless prevented by safety checks (i.e. when landed), 21196: force arming/disarming (e.g. allow arming to override preflight checks and disarming in flight)	<i>min</i> :0 <i>max</i> :21196 <i>increment</i> :21196

From the point of view of the script, it is going to look a little bit different since the library declares its own convention about the naming of the constants. Yet, it remains recognizable. You can either create a new script based on the previous one or just add some lines below.

```

the_connection.mav.command_long_send(the_connection.target_system,
the_connection.target_component,
                                     mavutil.mavlink.MAV_CMD_COMPONENT_ARM_DISARM,
                                     0,
1, 0, 0, 0, 0, 0, 0, 0,)

msg = the_connection.recv_match(type='COMMAND_ACK', blocking=True)
print(msg)

```

As a result of the script execution you should be able to observe the change of status at the top left corner of QGroundControl window. Additionally, the acknowledge message should also appear in the console window.

The control over the simulated UAV can be performed in the following fashion further (submission of waypoint, takeoff, return commands, etc.). Unfortunately, to cover complete set of commands the build should support a guided mode. This version of jmavsim seems to be deficient of this feature. May be corrected in the future.

To finalize the work you need to use the commander interface of the px4 simulator. Use the commands to implement takeoff and land procedure. Also demonstrate the status of the simulator instance using the same commander procedure (available in command line after the launch of the simulator).

References

1. https://mavlink.io/en/mavgen_python/
2. <https://mavlink.io/en/services/command.html>
3. <https://mavlink.io/en/messages/common.html>