DIY Hexacopter

Development Environment

Andrew Zitter

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Table of Contents

Acknowledgements

The work in this document was guided and influenced by Dr. Denzel Hamilton, Dr. Reuben Johnston, and Dr. Lanier Watkins from the Johns Hopkins University Applied Physics Laboratory (JHU APL). Their mentorship and expertise were instrumental; providing invaluable insights and feedback that significantly enhanced the quality and depth of this project.

Introduction

In this document, we will prepare, install, and configure a Software in the Loop (SITL) environment and deploy a custom Python application to the UAV that was assembled in **DIY Hexacopter: Assembly and Basic Configuration**. Unlike the previous guide, there is no physical assembly here. Instead, we will focus on the software necessary to properly write, test, and deploy an offboard API to the UAV.

The diagram below shows the major components of the UAV and their communication channels. The items shaded in black are expected to have already been completed.

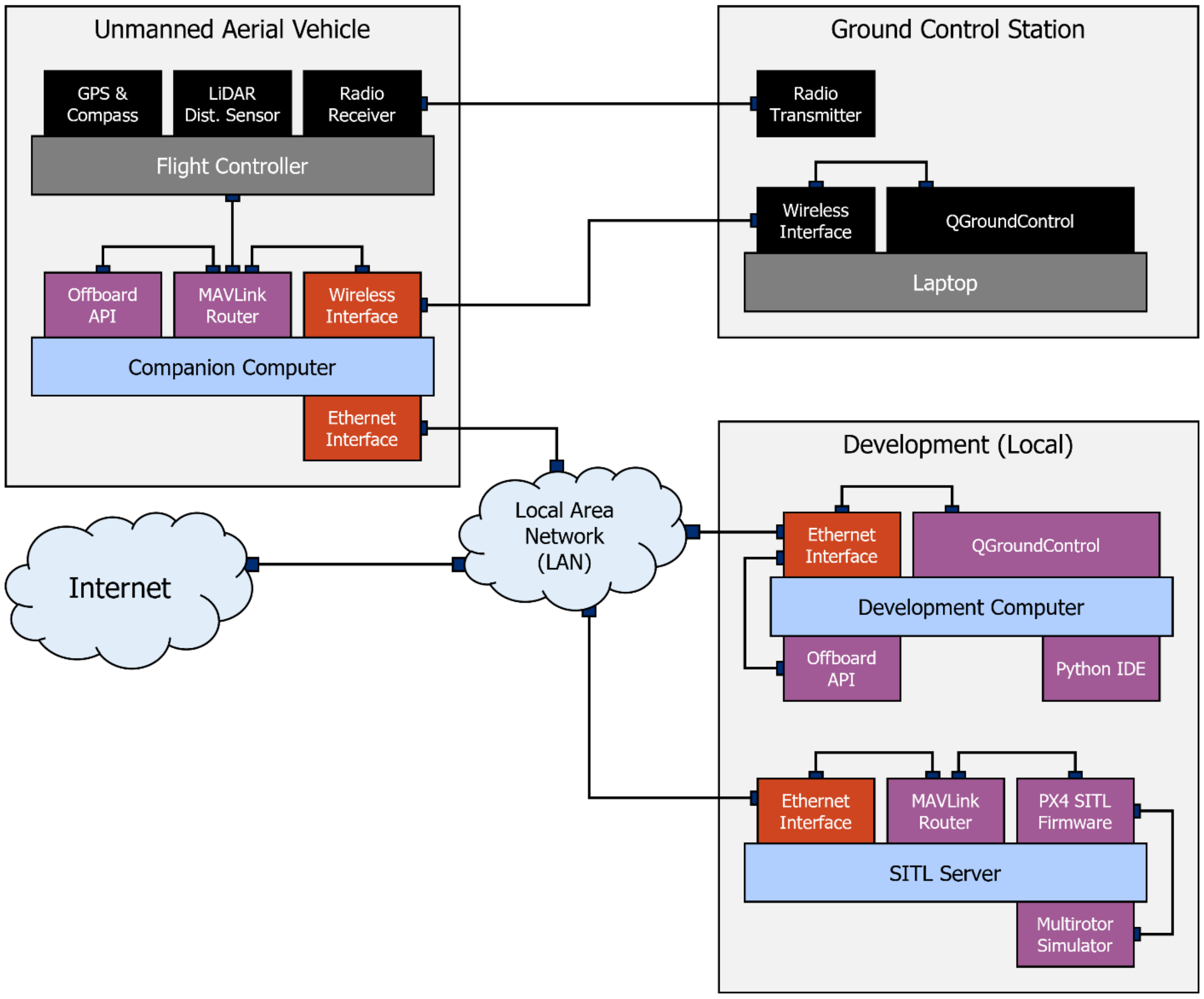


Figure 1 *A diagram showing the major components of the UAV and connected Ground Station*

Configure the SITL Testbed

Before we deploy custom code to a very expensive and powerful flying machine, we must thoroughly test it in a simulated environment. This will ensure that it behaves as expected. Fortunately, there is a large community of DIY drone developers that have created a number of tools that will give us a robust testbed on which to validate our code. This is called Software in the Loop (SITL), since there is no actual flight controller hardware involved.

Deploy the Virtual Machines

In this environment, the virtual machines are hosted in a VMWare vSphere environment. The process to create the virtual machines in VMWare Workstation Pro, Proxmox VE, and other solutions should be nearly identical. You can (and should) deviate from these instructions so that the environment fits your needs. For example, a developer using a Linux computer could run all of the tools on their own host machine, without needing a virtual environment at all.

Create the Virtual Machine

If you are replicating the environment shown at the beginning of this document, **you will perform these steps twice;** once for the **SITL VM** and once for the **development VM**. These instructions are derived from reference [1].

1. Download the latest versions of Ubuntu Server and Ubuntu Desktop and upload the .iso files to the vSphere datastore.
2. In the appropriate directory, select **Actions > New Virtual Machine…** as shown in **Figure 2**.
3. Select **Create a new virtual machine** as shown in **Figure 3**.
4. Name the VM and select a folder for it to reside in as shown in **Figure 4**.
5. Select a compute resource to host the VM (most likely, you will keep the default value) as shown in **Figure 5**.
6. Select the storage policy as shown in **Figure 6**. Be sure to change the provisioning mode to **Thin Provision** to conserve storage space.
7. Set the guest operating system type by selecting **Linux** as the OS Family and **Ubuntu Linux (64-bit)** as the OS Version as shown in **Figure 7**.
8. Configure the VM's hardware according to the following specifications:

CPU	4
Memory	8 GB
Hard Drive	50 GB
Video Card	Auto-detect Settings

9. For the CD drive, select **Datastore ISO** File as shown in **Figure 8**, then navigate to and select the Ubuntu Server ISO (for SITL VM) or Ubuntu Desktop ISO (for Development VM). Verify that the VM settings appear as shown in **Figure 9**, then click through the remaining steps in the configuration wizard.
10. Power on the virtual machine by selecting the green triangle as shown in **Figure 10**.
11. Navigate through the installation steps, keeping the default options selected. If possible, when

you get to the **network** configuration, statically assign the IP address to make it easier to remember where the MAVLink traffic will be routed to [1].

12. After the installation is finished, select **Reboot** Now.
    a. In vSphere, remove the CD-ROM by selecting **Actions** (near the green triangle you selected to power on the VM) and then **Edit Settings** as shown in **Figure 11**.
    b. Revert the CD drive source to **Client Device** as shown in **Figure 12**.
13. Update the VM by opening a terminal and typing `sudo apt update && sudo apt upgrade -y`.
14. Install Open VM Tools by entering `sudo apt install open-vm-tools`.
15. **For the SITL** (Ubuntu Server) **VM only**, run the following commands to install a minimal GUI.
    a. When prompted, after entering the second command, select **lightdm** as shown in **Figure 13**.
    b. After starting the lightdm service in the fourth command, you will have to log back in. Select **Terminator** from the **Show Applications** menu at the bottom of the screen, as shown in **Figure 14**. Optionally, right-click it and select **Add to Favorites**.

```
sudo apt install terminator nano -y
sudo apt install lightdm tasksel net-tools inetutils-traceroute gnome-
   tweaks -y
sudo tasksel install xubuntu-desktop
sudo service lightdm start
sudo systemctl disable NetworkManager
```

    c. In the terminal, run gnome-tweaks. Navigate to **Window Title Bars** and enable the **Maximize** and **Minimize** buttons as shown in **Figure 15**.
16. Save a snapshot of the VM so that it doesn't need to be completely re-built in the event of an error in the future. Using VMWare Workstation Pro, select **VM > Snapshot > Take Snapshot** as shown in **Figure 16**.

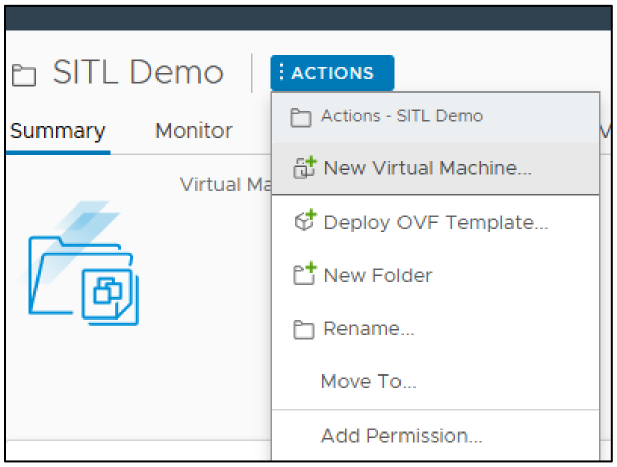


***Figure 2** Selecting the New Virtual Machine… action*

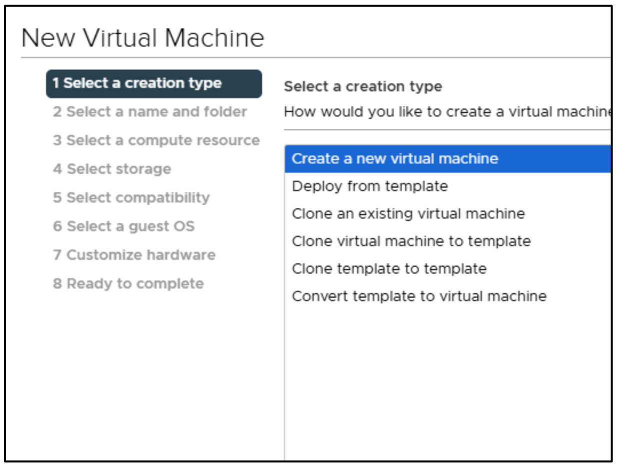
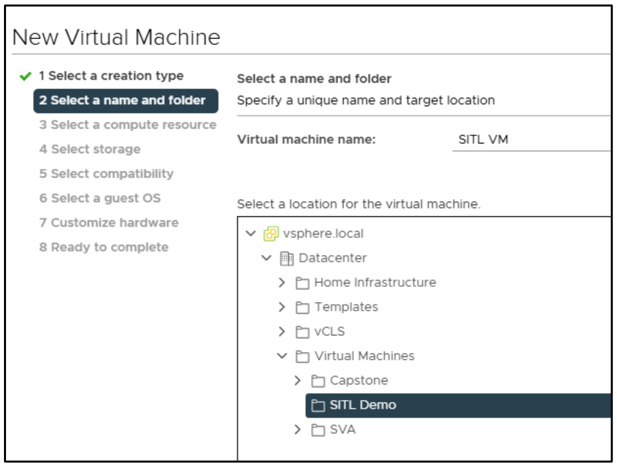

New Virtual Machine
1 Select a creation type
2 Select a name and folder
3 Select a compute resource
4 Select storage
5 Select compatibility
6 Select a guest OS
7 Customize hardware
8 Ready to complete
Select a creation type
How would you like to create a virtual machine
Create a new virtual machine
Deploy from template
Clone an existing virtual machine
Clone virtual machine to template
Clone template to template
Convert template to virtual machine

***Figure 3** Selecting Create a new virtual machine*

[1] This is more important for the development VM than it is for the SITL VM, as QGroundControl and dronekit will both wait for traffic from the flight controller, and then send replies to the IP and port messages were received from.

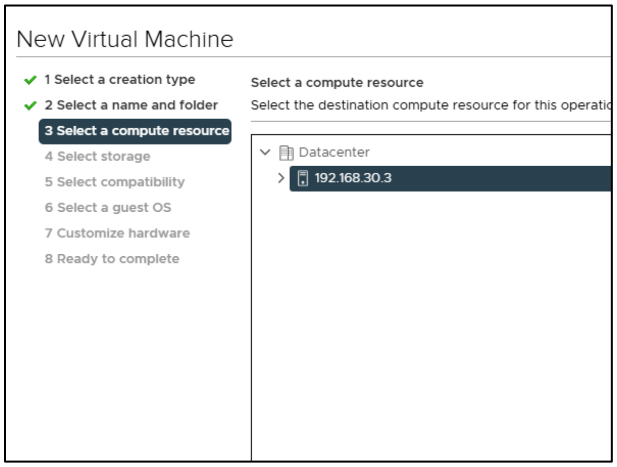


Select a name and folder
Specify a unique name and target location

Virtual machine name: SITL VM

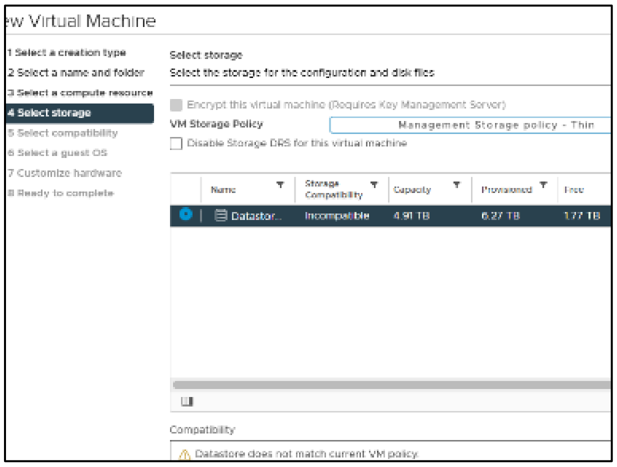
Select a location for the virtual machine.

***Figure 4** Naming the VM and selecting a location*



Select a compute resource
Select the destination compute resource for this operatio

***Figure 5** Selecting the compute resource to host the VM*



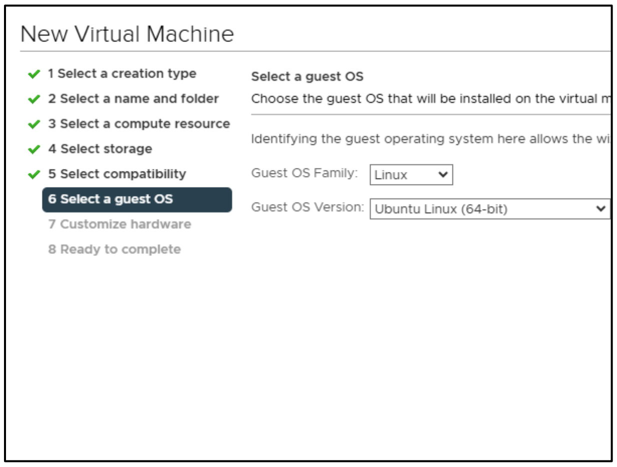
Select storage
Select the storage for the configuration and disk files

Encrypt this virtual machine (Requires Key Management Server)
VM Storage Policy: Management Storage policy - Thin
Disable Storage DRS for this virtual machine

	Name	Storage Compatibility	Capacity	Provisioned	Free
●	Datastor...	Incompatible	4.91 TB	6.27 TB	1.77 TB

Compatibility
Datastore does not match current VM policy.

***Figure 6** Selecting the storage policy*

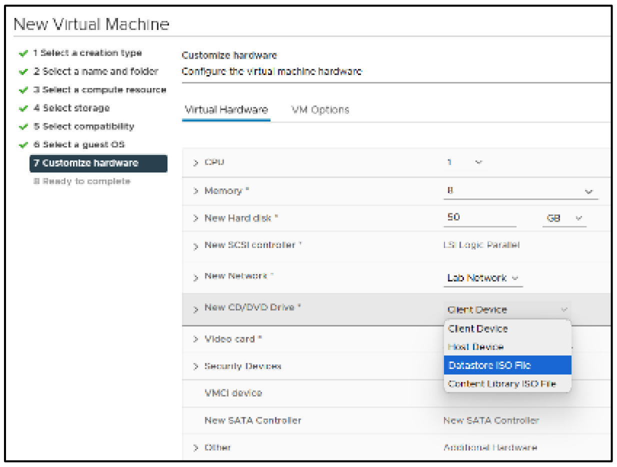


Select a guest OS
Choose the guest OS that will be installed on the virtual m

Identifying the guest operating system here allows the wi

Guest OS Family: Linux
Guest OS Version: Ubuntu Linux (64-bit)

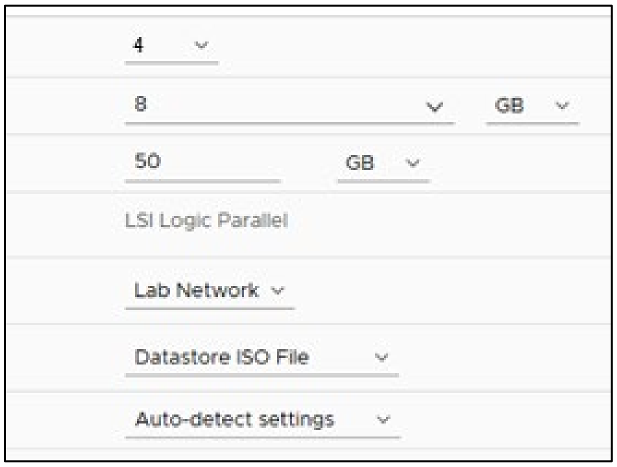
***Figure 7** Selecting the operating system*



Customize hardware
Configure the virtual machine hardware

CPU	1
Memory *	8
New Hard disk *	50 GB
New SCSI controller *	LSI Logic Parallel
New Network *	Lab Network
New CD/DVD Drive *	Client Device
Video card *	
Security Devices	
VMCI device	
New SATA Controller	New SATA Controller
Other	Additional Hardware

***Figure 8** Selecting the storage policy*



4
8 GB
50 GB
LSI Logic Parallel
Lab Network
Datastore ISO File
Auto-detect settings

***Figure 9** Configuring the settings*

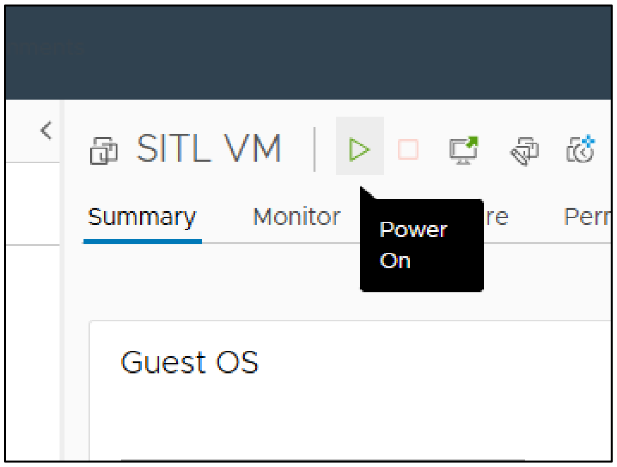


Figure 10 *Powering on the virtual machine*

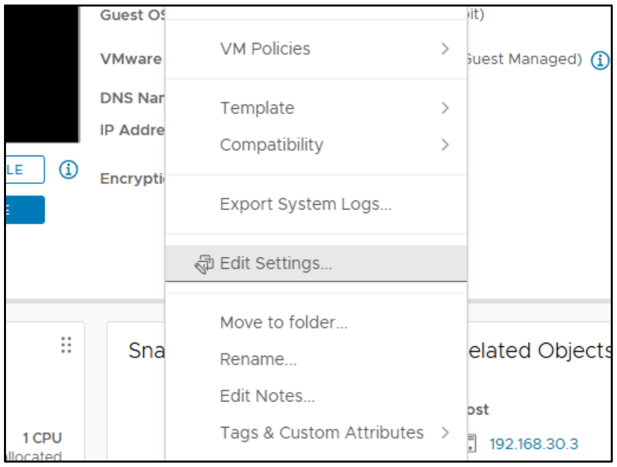


Figure 11 *Selecting the Edit Settings option*

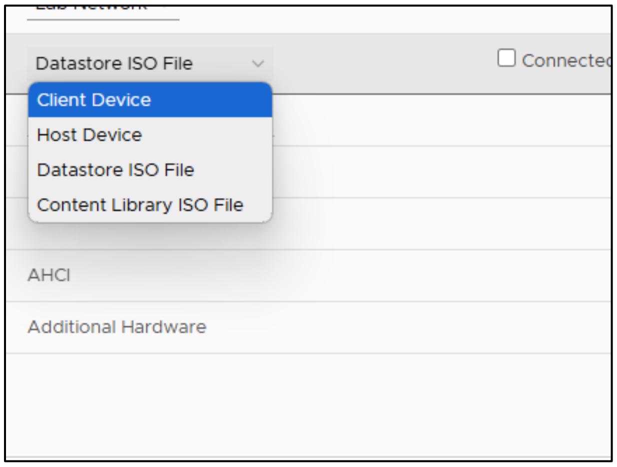
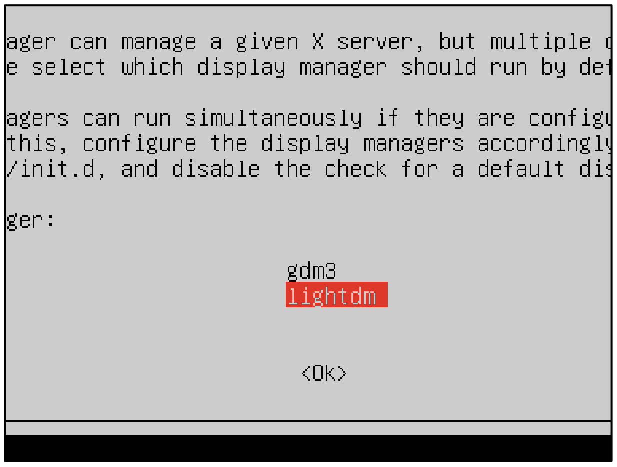


Figure 12 *Selecting the Client Device option*



```
ager can manage a given X server, but multiple d
e select which display manager should run by def

agers can run simultaneously if they are configu
this, configure the display managers accordingly
/init.d, and disable the check for a default dis

ger:

                    gdm3
                    lightdm

                     <Ok>
```

Figure 13 *Selecting the lightdm option*

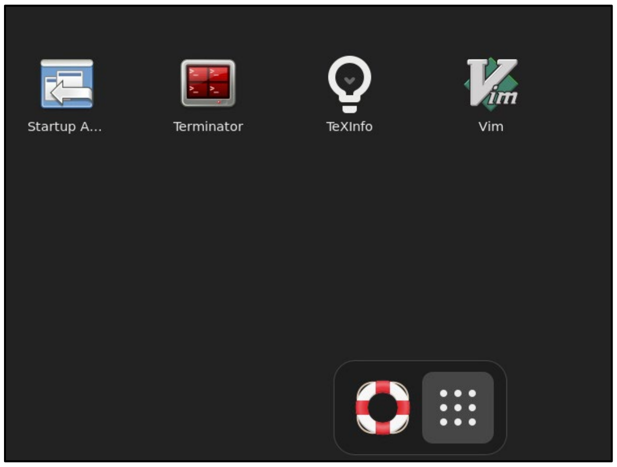
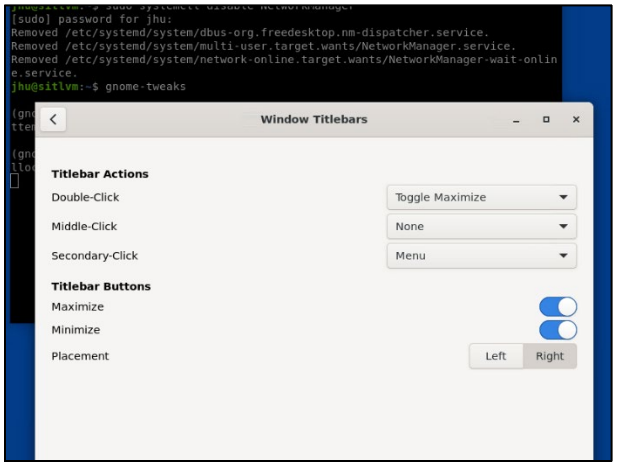


Figure 14 *Selecting the Terminator application*



```
[sudo] password for jhu:
Removed /etc/systemd/system/dbus-org.freedesktop.nm-dispatcher.service.
Removed /etc/systemd/system/multi-user.target.wants/NetworkManager.service.
Removed /etc/systemd/system/network-online.target.wants/NetworkManager-wait-onlin
e.service.
jhu@sitlvm:~$ gnome-tweaks
```

Window Titlebars

Titlebar Actions

Double-Click: Toggle Maximize
Middle-Click: None
Secondary-Click: Menu

Titlebar Buttons

Maximize: on
Minimize: on
Placement: Left [✓ Right]

Figure 15 *Enabling the title bar buttons*

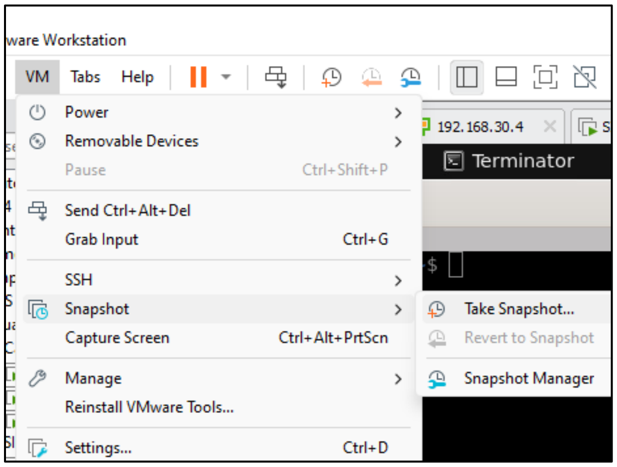


***Figure 16** Taking a snapshot of the clean install*

Configure the SITL VM

The SITL VM will host the PX4_SITL firmware and the multirotor simulator, as well as MAVLink Router to relay messages from the virtual flight controller, ground control software, and the external API that will be hosted on the development VM.

Install the PX4 SITL Technology Stack

The PX4 firmware is hosted on GitHub. In this section, you will download it, update its dependencies, and select the correct version of the firmware you wish to modify. These instructions are derived from references [2], [3], [4], and [5].

1. Ensure that `git` is installed in your environment by entering `sudo apt install git`.
2. Navigate to your working directory and download the PX4 source code and submodules with the following commands, **then reboot the development machine**:

```
git clone https://github.com/PX4/PX4-Autopilot --recursive
bash ./PX4-Autopilot/Tools/setup/ubuntu.sh
```

3. Verify that the toolchain is installed by navigating to the PX4-Autopilot directory and entering the following command: `HEADLESS=1 make px4_sitl gz_x500` [2] [3] and looking for output matching **Figure 17**. If everything is successful, you should be able to press Enter and receive a `pxh>` prompt.

[2] The HEADLESS parameter is necessary because the virtual machine it is running on does not have access to a GPU. If you are performing these steps on a traditional Linux computer and have access to a GPU, you can omit this parameter.

[3] If running Ubuntu 22.04 or older, use Gazebo Classic instead of Gazebo. There are additional configuration steps that have to be performed; refer to the PX4 Simulator documentation.

Install MAVLink Router

During the initial configuration of the Raspberry Pi in **DIY Hexacopter: Assembly and Basic Configuration**, we used a Python package called MAVProxy to handle MAVLink messages between the flight controller and companion computer. For this implementation, we will instead use MAVLink Router, which performs essentially the same function but does not need to run inside a Python virtual environment. These instructions are derived from reference [6].

1. Navigate to your working directory.
2. Ensure that the required packages are installed in your development environment with the following command:

   ```
   sudo apt install git meson ninja-build pkg-config gcc g++ system -y
   ```

3. Download the source code and submodules, then build and install the application with the following commands:

   ```
   git clone https://github.com/mavlink-router/mavlink-router
   cd mavlink-router
   git submodule update --init --recursive
   meson setup build .
   ninja -C build
   sudo ninja -C build install
   ```

Verify that MAVLink Router is installed by entering `mavlink-routerd -h` and looking for output matching **Figure 18**.

Write Bash Scripts

Writing and executing simple bash scripts will make starting and re-starting the simulation easier.

1. Navigate to your home directory by entering `cd ~/`.
2. Create a new file by entering `touch simulator_start.sh`.
   a. Open the file with `nano simulator_start.sh` (or your preferred text editor)
   b. Copy the text shown in **Appendix A. Bash Scripts for SITL Startup: PX4 SITL Startup Script** on **page 18**.
   c. Make the script executable by entering `chmod +x simulator_start.sh`.
3. Create a new file by entering `touch mavlink_router_start.sh`.
   a. Open the file with `nano mavlink_router_start.sh` (or your preferred text editor)
   b. Copy the text shown in **Appendix A. Bash Scripts for SITL Startup: MAVLink Router Startup Script (Development)** on **page 18**.
   c. Make the script executable by entering `chmod +x mavlink_router_start.sh`.
4. Test both scripts by entering `./simulator_start.sh` and `./mavlink_router_start.sh` and verifying that the terminal displays the expected output.

Configure Terminal

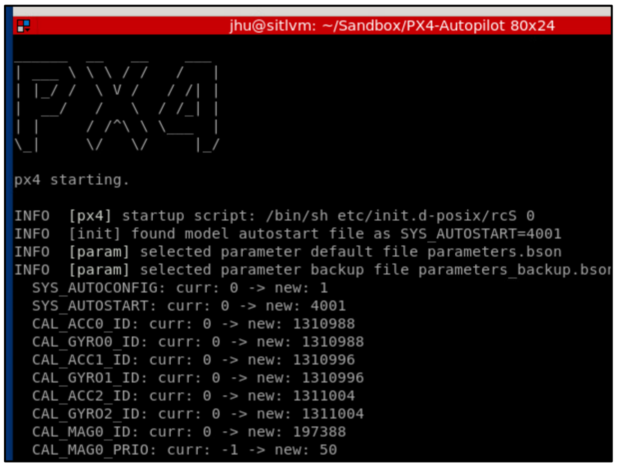
Optionally, configure the terminal to minimize the amount of effort it takes to start or restart the SITL components.

1. Open a Terminator window, and maximize it.
2. Right-click anywhere in the main terminal output and select **Split Vertically**, then in the top partition, right-click again and select **Split Horizontally**.
3. Double-click each of the red title bars and re-name them to the following:

a.	**Top Left**	PX4 SITL
b.	**Top Right**	MAVLink Router
c.	**Bottom**	Working Terminal

4. Launch the SITL and MAVLink Router bash scripts in the top left and top right windows, respectively. The bottom terminal is used for whatever additional commands you need to run.

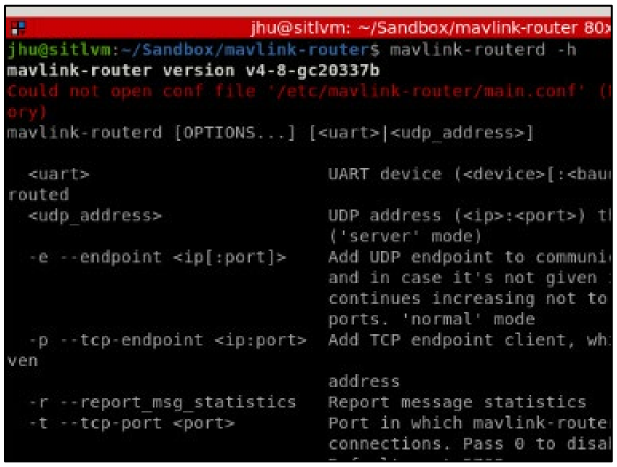
An example of this configuration with all components running is shown in **Figure 19**.



```
jhu@sitlvm: ~/Sandbox/PX4-Autopilot 80x24
 ______  __   __    ___
| ___ \ \ \ / /   /   |
| |_/ /  \ V /   / /| |
|  __/   /   \  / /_| |
| |     / /^\ \ \___  |
\_|     \/   \/     |_/

px4 starting.

INFO  [px4] startup script: /bin/sh etc/init.d-posix/rcS 0
INFO  [init] found model autostart file as SYS_AUTOSTART=4001
INFO  [param] selected parameter default file parameters.bson
INFO  [param] selected parameter backup file parameters_backup.bson
  SYS_AUTOCONFIG: curr: 0 -> new: 1
  SYS_AUTOSTART: curr: 0 -> new: 4001
  CAL_ACC0_ID: curr: 0 -> new: 1310988
  CAL_GYRO0_ID: curr: 0 -> new: 1310988
  CAL_ACC1_ID: curr: 0 -> new: 1310996
  CAL_GYRO1_ID: curr: 0 -> new: 1310996
  CAL_ACC2_ID: curr: 0 -> new: 1311004
  CAL_GYRO2_ID: curr: 0 -> new: 1311004
  CAL_MAG0_ID: curr: 0 -> new: 197388
  CAL_MAG0_PRIO: curr: -1 -> new: 50
```

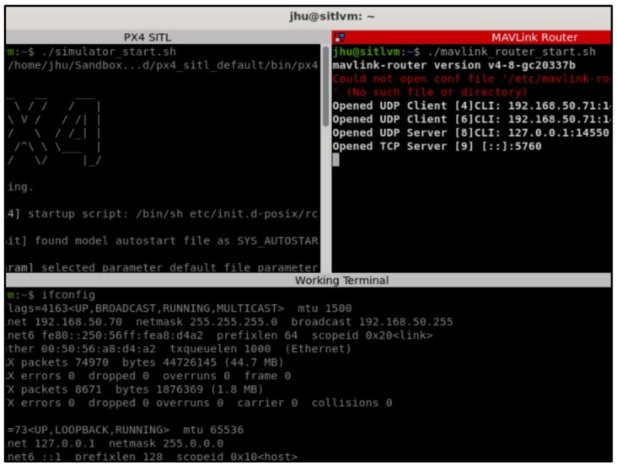


```
jhu@sitlvm: ~/Sandbox/mavlink-router 80x
jhu@sitlvm:~/Sandbox/mavlink-router$ mavlink-routerd -h
mavlink-router version v4-8-gc20337b
Could not open conf file '/etc/mavlink-router/main.conf' (
ory)
mavlink-routerd [OPTIONS...] [<uart>|<udp_address>]

  <uart>                       UART device (<device>[:<bau
routed
  <udp_address>                UDP address (<ip>:<port>) t
                               ('server' mode)
  -e --endpoint <ip[:port]>    Add UDP endpoint to communi
                               and in case it's not given
                               continues increasing not to
                               ports. 'normal' mode
  -p --tcp-endpoint <ip:port>  Add TCP endpoint client, wh
ven
                               address
  -r --report_msg_statistics   Report message statistics
  -t --tcp-port <port>         Port in which mavlink-route
                               connections. Pass 0 to disa
```

Figure 17 Output showing the PX4-Autopilot toolchain

Figure 18 Output showing the mavlink-routerd help



```
jhu@sitlvm: ~
PX4 SITL
m:~$ ./simulator_start.sh
/home/jhu/Sandbox...d/px4_sitl_default/bin/px4

 _  __    ___
 \ \ / /   /   |
 \ V /   / /| |
 /   \  / /_| |
 /^\ \ \___  |
/   \/     |_/

ing.

4] startup script: /bin/sh etc/init.d-posix/rc

it] found model autostart file as SYS_AUTOSTAR

ram] selected parameter default file parameter
```

```
MAVLink Router
jhu@sitlvm:~$ ./mavlink_router_start.sh
mavlink-router version v4-8-gc20337b
Could not open conf file '/etc/mavlink-ro
' (No such file or directory)
Opened UDP Client [4]CLI: 192.168.50.71:1
Opened UDP Client [6]CLI: 192.168.50.71:1
Opened UDP Server [8]CLI: 127.0.0.1:14550
Opened TCP Server [9] [::]:5760
```

```
Working Terminal
m:~$ ifconfig
lags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
net 192.168.50.70  netmask 255.255.255.0  broadcast 192.168.50.255
net6 fe80::250:56ff:fea8:d4a2  prefixlen 64  scopeid 0x20<link>
ther 00:50:56:a8:d4:a2  txqueuelen 1000  (Ethernet)
X packets 74970  bytes 44726145 (44.7 MB)
X errors 0  dropped 0  overruns 0  frame 0
X packets 8671  bytes 1876369 (1.8 MB)
X errors 0  dropped 0 overruns 0  carrier 0  collisions 0

=73<UP,LOOPBACK,RUNNING>  mtu 65536
net 127.0.0.1  netmask 255.0.0.0
net6 ::1  prefixlen 128  scopeid 0x10<host>
```

Figure 19 Three-terminal layout for the SITL VM

Configure the Development VM

Install Python 3.9

At the time of writing, the `dronekit` package, which is at the core of our application, will only run on Python 3.9 and earlier. This version is so old that it had been removed from all common distributions, and the more recent versions have to be manually compiled from their source.

In a **Windows** environment, the easiest solution is to download the latest version of the Python 3.9 installer, which is version 3.9.13. Simply download and install it from the official web site.

If using **Linux**, follow the instructions below which are derived from reference [7].

1. Navigate to your working directory.
2. Ensure that the required packages are installed in your development environment with the following command:

```
sudo apt-get install -y build-essential tk-dev libncurses5-dev
   libncursesw5-dev libreadline6-dev libdb5.3-dev libgdbm-dev libsqlite3-
   dev libssl-dev libbz2-dev libexpat1-dev liblzma-dev zlib1g-dev libffi-
   dev
```

3. Download and extract the source code for Python 3.9.20 (the latest version at the time of writing).

```
cd ~/Downloads
wget https://www.python.org/ftp/python/3.9.20/Python-3.9.20.tgz
sudo tar zxf Python-3.9.20.tgz
```

4. Build and install this version of Python.

```
cd Python-3.9.20
sudo ./configure --enable-optimizations
sudo make -j 4
sudo make altinstall
```
[4]

Regardless of your installation method, verify that Python 3.9 is installed on the system with `python3.9 --version` [5]. Compare this output to `python3 --version` as shown in **Figure 20**.

[4] The -j 4 argument indicates the number of CPU cores available, which decreases the time it takes for this command to run. If you are not setting up the environment as written, or you encounter issues with this command, repeat it with the correct number of cores or without the -j argument (e.g. “sudo make”).

[5] The exact prompt used to access Python 3.9 may be different on your computer, depending on what versions of Python are previously installed.

Activate the Python 3.9 Virtual Environment

Creating a virtual environment is trivial; the version that you use to instantiate the environment is the version that will be assigned to it. Developing the application in this environment will isolate it from the system's default Python library, allowing you to continue using what is an otherwise obsolete version of Python.

1. Navigate to your working directory, where your Python code will reside.
2. Create the virtual environment with the following command:

   ```
   python3.9 -m venv [name-of-virtual-environment]
   ```

3. Activate the virtual environment with one of the following commands, depending on your operating system:
   a. **Windows:** `[name-of-virtual-environment]\Scripts\activate.bat`
   b. **Linux:** `source [name-of-virtual-environment]/bin/activate`
4. Verify that the virtual environment is activated (the terminal prompt will be prefixed with your virtual environment name in parentheses) and contains an instance of Python 3.9 by entering the following command: `python --version` as shown in **Figure 21**.
5. Deactivate the virtual environment with the following command:
   a. **Windows:** `[name-of-virtual-environment]\Scripts\deactivate.bat`
   b. **Linux:** `dectivate`

Install Required Software

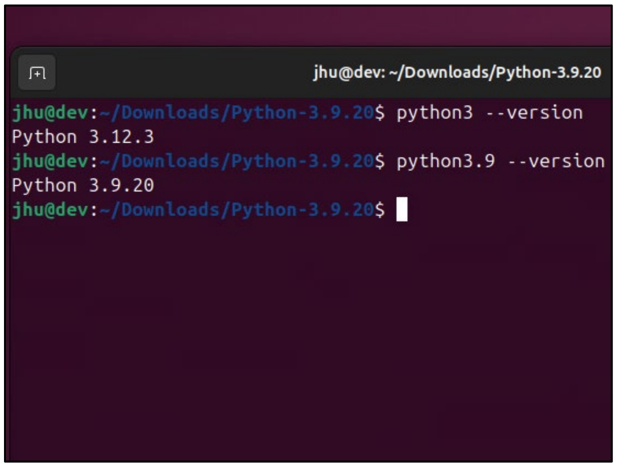
Install a copy of the ground control software, as well as a Python IDE, on the development machine. The instructions to install QGroundControl are derived from reference [8].

1. Navigate to your working directory.
2. Prepare the development environment by running the following commands:

   ```
   sudo usermod -a -G dialout $USER
   sudo apt-get remove modemmanager -y
   sudo apt install gstreamer1.0-plugins-bad gstreamer1.0-libav gstreamer1.0-
      gl -y
   sudo apt install libfuse2 -y
   sudo apt install libxcb-xinerama0 libxkbcommon-x11-0 libxcb-cursor-dev -y
   ```

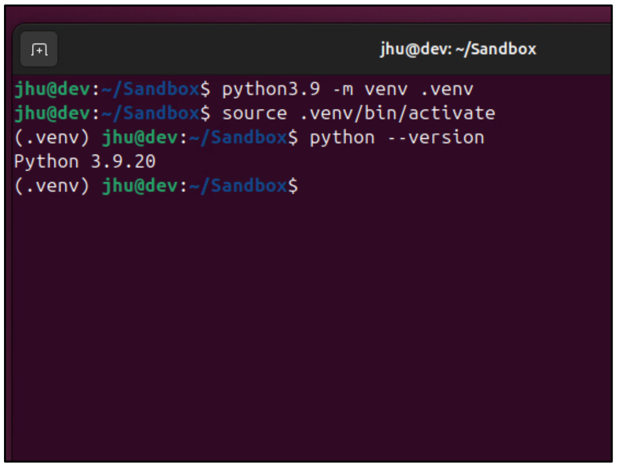
3. Log out and back in to force the permissions changes to take effect.
4. Download the AppImage file from the link at the reference at the beginning of this section.
5. Make the file executable by navigating to the directory where the file was downloaded and entering `chmod +x QGroundControl.AppImage`.
6. Double click on the file, or execute it in the terminal by entering `./QGroundControl.AppImage` as shown in **Figure 22**.

Set up your Python development environment as you see fit. Popular IDEs include Visual Studio Code and PyCharm.



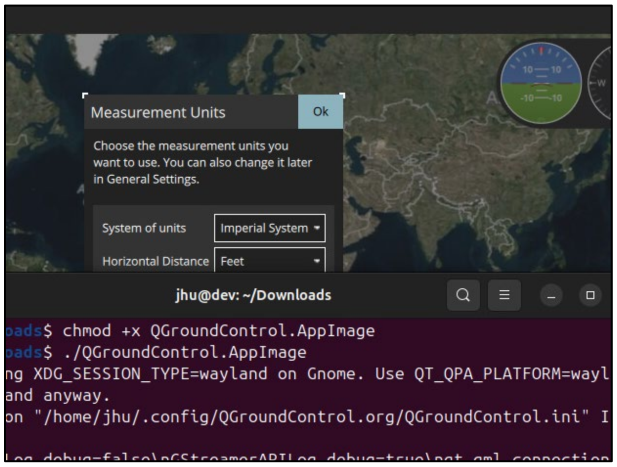
```
jhu@dev: ~/Downloads/Python-3.9.20
jhu@dev:~/Downloads/Python-3.9.20$ python3 --version
Python 3.12.3
jhu@dev:~/Downloads/Python-3.9.20$ python3.9 --version
Python 3.9.20
jhu@dev:~/Downloads/Python-3.9.20$
```

***Figure 20** Primary and alternate Python versions*



```
jhu@dev: ~/Sandbox
jhu@dev:~/Sandbox$ python3.9 -m venv .venv
jhu@dev:~/Sandbox$ source .venv/bin/activate
(.venv) jhu@dev:~/Sandbox$ python --version
Python 3.9.20
(.venv) jhu@dev:~/Sandbox$
```

***Figure 21** Activating and verifying the Python 3.9 venv*



Measurement Units

Choose the measurement units you want to use. You can also change it later in General Settings.

System of units: Imperial System

Horizontal Distance: Feet

```
jhu@dev: ~/Downloads
oads$ chmod +x QGroundControl.AppImage
oads$ ./QGroundControl.AppImage
ng XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayl
and anyway.
on "/home/jhu/.config/QGroundControl.org/QGroundControl.ini" I

Log debug=false\nGStreamerAPILog debug=true\nqt qml connection
```

***Figure 22** Opening QGroundControl on Linux*

Develop the Application

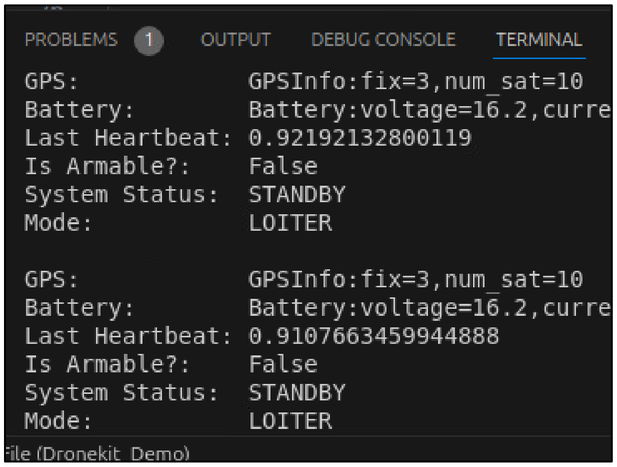
Import a basic application that can be used to report some of the flight controller's data. This is only meant as a proof of concept, and is intended to be a launchpad for your own custom applications. The code can be found in **Appendix B. Basic Offboard API** on **page 20**. When executed, the API will await a connection from the flight controller, and then retrieve some attributes from it every second until the connection is lost or the pilot terminates the program with Ctrl+C (a keyboard interrupt).

Test the Application

To run the application and see the output, perform the following steps:

1. Save the application code in your working directory as `main.py`. Replace the IP address in the connection string (on the last line of the code) with either the VM's IP address or `0.0.0.0`.
2. Activate the virtual environment according to the instructions in **Activate the Python 3.9 Virtual Environment** on **page 13**.
3. After verifying that the correct version of Python is running (3.9.x), install the `dronekit` package by entering `python -m pip install dronekit`.
4. Run the application with `python main.py`. The connection should time out after 30 seconds, and an error message should be printed to the screen.
5. In the SITL VM, activate the PX4 simulator and mavlink router.
6. Re-run the application and verify that you see output similar to **Figure 23**.

Optionally, open QGroundControl and verify that it connects to the SITL flight controller.



```
GPS:            GPSInfo:fix=3,num_sat=10
Battery:        Battery:voltage=16.2,curre
Last Heartbeat: 0.92192132800119
Is Armable?:    False
System Status:  STANDBY
Mode:           LOITER

GPS:            GPSInfo:fix=3,num_sat=10
Battery:        Battery:voltage=16.2,curre
Last Heartbeat: 0.9107663459944888
Is Armable?:    False
System Status:  STANDBY
Mode:           LOITER
```

Figure 23 *Output from the offboard API*

Deploy the Application

Once the code has been thoroughly tested on the simulator, it can be deployed to the real companion computer. In the previous document, we performed the minimum modifications necessary to test certain aspects of the environment. In this chapter, we will expand on this configuration to deploy our custom application. The companion computer will need to be powered on and connected to the Internet before completing this section.

1. Access the companion computer's terminal either directly or via SSH.
2. Install MAVLink Router according to the instructions in **Install MAVLink Router** on **page 10**.
3. Install Python 3.9 according to the instructions in **Install Python 3.9** on **page 12**. Remove the `-j` argument from the `make` command when compiling the Python source code.
4. Complete steps 1-3 according to the instructions in **Test the Application** on **page 15**. Replace the IP address in the connection string (on the last line of the code) with `127.0.0.1`.
5. Navigate to your home directory by entering `cd ~/`.
6. Create a new file by entering `touch api_start.sh`.
   a. Open the file with `nano api_start.sh` (or your preferred text editor)
   b. Copy the text shown in **Appendix A. Bash Scripts for SITL Startup: API Startup Script** on **page 19**.
   c. Make the script executable by entering `chmod +x api_start.sh`.
7. Create a new file by entering `touch mavlink_router_start.sh`.
   a. Open the file with `nano mavlink_router_start.sh` (or your preferred text editor)
   b. Copy the text shown in **Appendix A. Bash Scripts for SITL Startup: MAVLink Router Startup Script (Production)** on **page 19**.
   c. Make the script executable by entering `chmod +x mavlink_router_start.sh`.
8. Test both scripts by entering `./api_start.sh 127.0.0.1` and `./mavlink_router_start.sh` and verifying that the terminal displays the expected output [6].

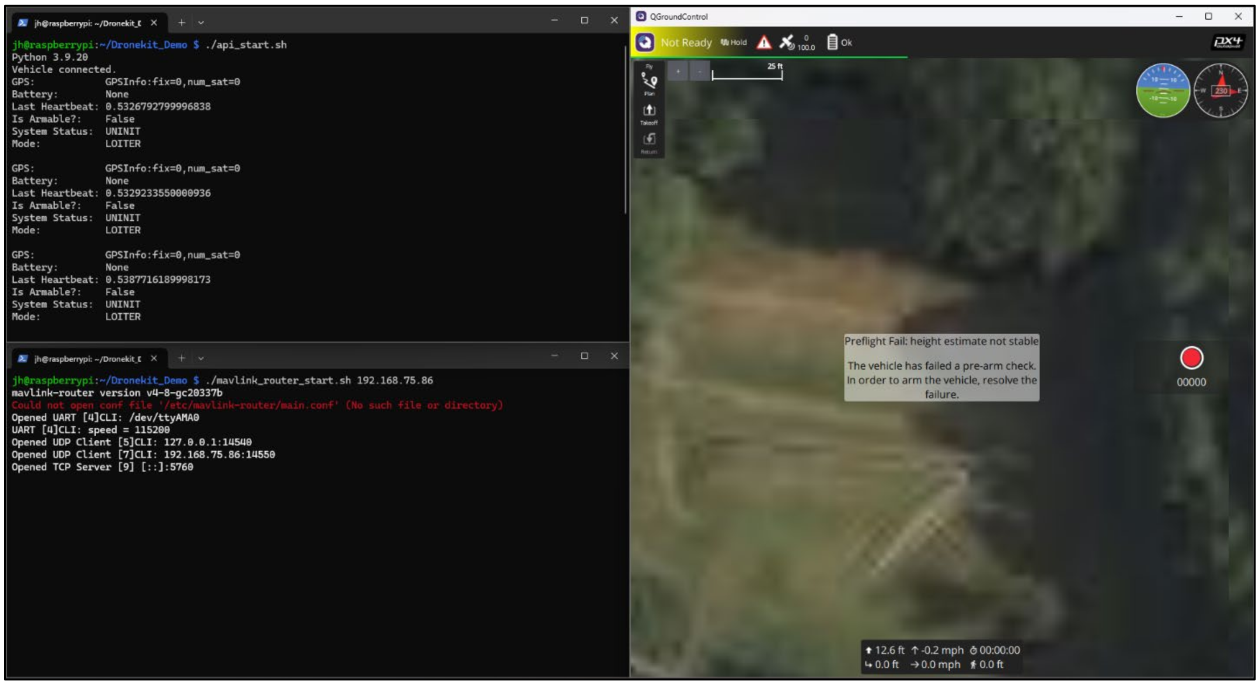
Test Flight

This section assumes that you have set up the flight controller, companion computer, and ground control station laptop according to the instructions in **DIY Hexacopter: Assembly and Basic Configuration**.

1. Connect the ground control station laptop to the companion computer via the Wi-Fi access point.
2. Open two terminal windows and connect to the companion computer via SSH in each one.
3. Open QGroundControl.
4. In one terminal window, navigate to and activate the `mavlink_router_start.sh` script, providing the IP address of the ground control station laptop as an argument.
5. Verify that QGroundControl receives data from the flight controller.
6. After the mavlink-routerd application starts, in the second terminal window, activate the `api_start.sh` script.
7. Verify that the API receives data from the flight controller.

[6] Note that api_start.sh takes the IP address of the ground control station laptop as an argument. For now, we are simply testing that the script functions, so providing it with localhost (127.0.0.1) is sufficient

An example of the ground control station laptop receiving data from the flight controller to both the API and QGroundControl is showin in **Figure 24**. In this example, the top left terminal is the API, the bottom left terminal is the MAVLink Router application, and QGroundControl is on the right.


jh@raspberrypi: ~/Dronekit_[
QGroundControl
Not Ready
Hold
0
100.0
Ok
Fly
Plan
Takeoff
Return
25 ft
00000
↑ 12.6 ft ↑ -0.2 mph ⏱ 00:00:00
↳ 0.0 ft → 0.0 mph ↑ 0.0 ft

```
jh@raspberrypi:~/Dronekit_Demo $ ./api_start.sh
Python 3.9.20
Vehicle connected.
GPS:            GPSInfo:fix=0,num_sat=0
Battery:        None
Last Heartbeat: 0.5326792799996838
Is Armable?:    False
System Status:  UNINIT
Mode:           LOITER

GPS:            GPSInfo:fix=0,num_sat=0
Battery:        None
Last Heartbeat: 0.5329233550000936
Is Armable?:    False
System Status:  UNINIT
Mode:           LOITER

GPS:            GPSInfo:fix=0,num_sat=0
Battery:        None
Last Heartbeat: 0.5387716189998173
Is Armable?:    False
System Status:  UNINIT
Mode:           LOITER
```

```
jh@raspberrypi:~/Dronekit_Demo $ ./mavlink_router_start.sh 192.168.75.86
mavlink-router version v4-8-gc20337b
Could not open conf file '/etc/mavlink-router/main.conf' (No such file or directory)
Opened UART [4]CLI: /dev/ttyAMA0
UART [4]CLI: speed = 115200
Opened UDP Client [5]CLI: 127.0.0.1:14540
Opened UDP Client [7]CLI: 192.168.75.86:14550
Opened TCP Server [9] [::]:5760
```

Preflight Fail: height estimate not stable

The vehicle has failed a pre-arm check. In order to arm the vehicle, resolve the failure.

__Figure 24__ The ground control station receiving data to both QGroundControl and the offboard API

Appendix A. Bash Scripts for SITL Startup

PX4 SITL Startup Script

Modify the **highlighted code** to fit your environment.

```
# Put the correct path to the PX4-Autopilot directory here
cd ~/working_directory/PX4-Autopilot
# These coordinates are to a flat, open space with little visual noise
export PX4_HOME_LAT=39.962538
export PX4_HOME_LON=-113.475976
export PX4_ALT=28.5
# Start the simulator in headless mode, which will not start the GUI
export HEADLESS=1
# Make the simulated vehicle
make px4_sitl gz_x500
```

MAVLink Router Startup Script (Development)

Modify the **highlighted code** to fit your environment.

```
# Put the correct IP address to the development VM in place of 192.168.1.100
# Do not alter 127.0.0.1; it is the source of the messages being proxied
#    14550 is the standard listening port for ground control software
#    14540 is the standard listening port for external APIs
mavlink-routerd -e 192.168.1.100:14550 -e 192.168.1.100:14540 127.0.0.1:14550
```

API Startup Script

Modify the **highlighted code** to fit your environment.

```
# Replace this with the path to the venv activation script
source /path/to/venv/bin/activate
# Print the Python version (should be 3.9.x)
python –version
# Replace this with the path to the API code
cd /home/user/working_directory/
# Run the application
python main.py
# After the application stops, deactivate the virtual environment
deactivate
```

MAVLink Router Startup Script (Production)

Modify the **highlighted code** to fit your environment.

```
# Check if the script received exactly one argument
if [ "$#" -ne 1 ]; then
  echo "Usage: $0 [IP Address of Ground Control Station]"
  exit 1
fi
# Extract the IP address from the argument
IP_ADDRESS=$1
# Run the mavlink-routerd command with the specified IP address
# In this version, the offboard API is on localhost, so only
#    the ground control listening port needs to be proxied
#    externally. The external API port is proxied internally.
# Note that the below code is all on one line
# 115200 is the baud rate set in the assembly instructions
mavlink-routerd --verbose -e 127.0.0.1:14540 -e "$IP_ADDRESS:14550" /dev/ttyAMA0:115200
```

Appendix B. Basic Offboard API

Modify the **highlighted code** to fit your environment. This code is based on reference [9].

```
import dronekit
import time

def main(connection_string):
    """Run the monitoring program."""
    try:
        # Await a heartbeat from the flight controller
        vehicle = dronekit.connect(connection_string, wait_ready = True)
        print("Vehicle connected.")
        # Run the event loop until it is closed by the pilot
        event_loop(vehicle)
        # Close vehicle object before exiting script
        vehicle.close()
        print("Vehicle disconnected.")
    except dronekit.APIException:
        print("Could not connect to vehicle.")

def event_loop(vehicle):
    """Run the event loop until the pilot interrupts it."""
    while True:
        try:
            # Print some attributes
            print(f"GPS:            {vehicle.gps_0}")
            print(f"Battery:        {vehicle.battery}")
            print(f"Last Heartbeat: {vehicle.last_heartbeat}")
            print(f"Is Armable?:    {vehicle.is_armable}")
            print(f"System Status:  {vehicle.system_status.state}")
            print(f"Mode:           {vehicle.mode.name}")
            print()
            time.sleep(1)
        except KeyboardInterrupt:
            break

if __name__ == "__main__":
    # The socket that the API will recieve data from
    # Set the IP address to this host
    main("udp:192.168.1.100:14540")
```

References

[1] R. A. Johnston, "Ubuntu x86 64 VM," [Online]. Available: https://github.com/jhu-information-security-institute/infrastructure/wiki/Ubuntu-x86-64-VM. [Accessed 30 November 2024].

[2] PX4 Autopilot, "Ubuntu Development Environment," [Online]. Available: https://docs.px4.io/main/en/dev_setup/dev_env_linux_ubuntu.html. [Accessed 21 November 2024].

[3] PX4 Autopilot, "GIT Examples," [Online]. Available: https://docs.px4.io/main/en/contribute/git_examples.html. [Accessed 21 November 2024].

[4] PX4 Autopilot - Open Source Flight Control., "Building PX4 And Uploading to Pixhawk," 31 July 2023. [Online]. Available: https://www.youtube.com/watch?v=unONnXr3Tr8. [Accessed 21 November 2024].

[5] PX4 Autopilot, "jMAVSim with SITL," [Online]. Available: https://docs.px4.io/v1.14/en/simulation/jmavsim.html. [Accessed 30 November 2024].

[6] L. De Marchi, "MAVLink Router," GitHub, [Online]. Available: https://github.com/mavlink-router/mavlink-router. [Accessed 30 November 2024].

[7] W. Pyburn, "Update/Install specific Python on Raspberry Pi," 08 June 2023. [Online]. Available: https://hub.tcno.co/pi/software/python-update/. [Accessed 06 November 2024].

[8] QGroundControl, "Download and Install," [Online]. Available: https://docs.qgroundcontrol.com/master/en/qgc-user-guide/getting_started/download_and_install.html. [Accessed 01 December 2024].

[9] 3D Robotics, "Quick Start," 2016. [Online]. Available: https://dronekit.netlify.app/guide/quick_start.html. [Accessed 01 December 2024].