SOFTWAREPILOT INSTALLATION GUIDE

Dec 2019

Version 1.0

By: Jayson Boubin

Special Assistance from: Sadaqat Ali, Anthony Baietto, Jack Dubs, Peida Han, Bowen Li, Nat Shineman, Yujie Zhao, Chengyuan Zhao

Document Revisions

Date	Version Number	Document Changes
12/01/2019	0.1	Initial Draft
12/26/2019	1.0	First Release Version

SoftwarePilot Installation Guide

Table of Contents

SoftwarePilot Installation Guide

1 Introduction

1.1 Scope and Purpose

SoftwarePilot is an open source middleware and API that supports aerial applications. SoftwarePilot allows users to connect consumer DJI drones to programmable Java routines that provide access to the drones flight controller, camera, and navigation system as well as computer vision and deep learning software packages like OpenCV, DLIB, and Tensorflow.

SoftwarePilot is used by researchers to create and test aerial systems that use novel autonomy policies, architectural configurations, and vision algorithms with application domains ranging from agriculture to autonomous photography. Educators also use SoftwarePilot to teach middle school to university students about drones, autonomous systems, computational thinking, and programming in general

SoftwarePilot comes with a dockerfile and installation scripts for all requisite software, as well as an Android-x86 virtual machine to communicate with DJI drones from most systems.

This guide will show you how to download, install, and run SoftwarePilot, and provide information on the hardware and software requirements of systems which can execute SoftwarePilot.

1.2 System Requirements

SoftwarePilot requires both an edge device and DJI UAV.

Edge Device Minimum Requirement:

- 4GB of RAM
- 2 CPU Cores
- 20GB of available disk space
- IEEE 802.11 WIFI Capabilities

Edge Device Recommended Specs:

- 8GB of Ram or more
- 4 or more CPU Cores
- 5.8GHz WIFI Capabilities

Compatible DJI UAVs:

- Spark
- Mavic (Any)
- Matrice (Any)

1.3 Required Software

SoftwarePilot has a series of software dependencies. Note: some operating systems may require additional software, as discussed in section 2.

Required Software:

- A SoftwarePilot Compatible Operating System:
  - Linux (Kernel version > 4.0, Ubuntu 18.04 or greater recommended)
  - Windows (Version > Windows 7)
  - MacOS (Version > Snow Leopard)
- VirtualBox version 5.x
- Docker (any version)
  - Windows users should also install Docker Toolbox
- Git

1.4 Terminology

Below is a list of important terminology that will be used throughout this document and associated definitions.

- Driver
  - A SoftwarePilot driver is one of two critical software components for executing FAAS missions. A driver is a Java file, compiled to a JAR file, that specifies a specialized microservice API that describes certain functions of autonomous flight. For instance, the FlyDroneDriver specifies functions for UAV flight. Drivers are detailed in the SoftwarePilot development guide.
- Routine
  - Routines are Java files, compiled to JAR files, that specify a series of driver calls and other logic that constitute a FAAS mission. For instance, the AutoSelfie routine flys a UAV to autonomously capture high-quality human facial images. Routines are detailed in the SoftwarePilot development guide.
- Kernel
  - The SoftwarePilot Linux Kernel is an instance of SoftwarePilot that executes outside of the SoftwarePilot virtual machine. The kernel is used for both simulating SoftwarePilot to test functionality, and executing components of SoftwarePilot in-mission that can not be run on the VM, like complex machine learning algorithms. The kernel and its functionality are detailed in the SoftwarePilot development guide

- $AUAVHOME
    - AUAVHOME is a SoftwarePilot docker container environment variable that is used to specify the root directory of the SoftwarePilot codebase, both for the purpose of reference in this guide, and to allow SoftwarePilot to specify absolute paths for all requisite operations in code.
- APK
    - The SoftwarePilot APK is the android application build to execute in the SoftwarePilot android-x86 VM. The APK includes all drivers and routines as JAR files, and can be installed using the SoftwarePilot compile script detailed in Section 3 of this guide.

2 Installation

2.1 Software Download

Once all software dependencies are installed, SoftwarePilot requires simply the SoftwarePilot Git Repository and the SoftwarePilot Virtual Machine to be installed to begin execution and development.

> Note: The SoftwarePilot Docker container contains an environment variable ($AUAVHOME) which contains the root directory of the SoftwarePilot git repository. Throughout the rest of this document, we will refer to $AUAVHOME in this context.

- The SoftwarePilot Codebase can be downloaded here:
  - https://github.com/boubinjg/SoftwarePilot
  - Clone the repository using Git for your chosen operating system
- The SoftwarePilot VirtualBox OVA can be found here:
  - https://reroutlab.org/softwarepilot/SoftwarePilot.ova
  - Download this OVA file to be installed into VirtualBox in step 2.1.2

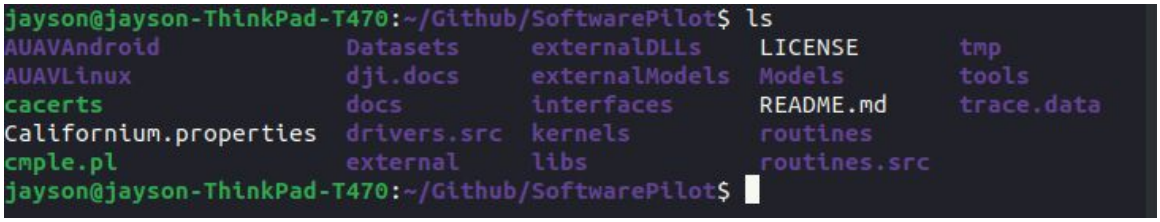
2.2 Container and Virtual Machine Setup

2.2.1 Docker Setup

Step 1: Build the Docker image

A Dockerfile is available in the Git repository at:
$AUAVHOME/tools/Docker/build/Dockerfile

$AUAVHOME on an Ubuntu 18.04 machine:



The Dockerfile in $AUAVHOME/tools/Docker/:

To build the Dockerfile on Mac or Linux, run the bulid.sh script using the following command:

```
jayson@jayson-ThinkPad-T470:~/Github/SoftwarePilot/tools/Docker$ bash build.sh
```

On Windows, enter the following command in the bash CLI provided by DockerToolbox

Note: Building the Docker image may take between 30-60 minutes depending on internet connection and processor speed.

Once the Dockerfile has been used to build the Docker image, the SoftwarePilot docker container can be accessed. Using the Mac or Linux terminal, or the DockerToolbox CLI in Windows, the script $AUAVHOME/tools/Docker/runAUAVLinux.sh can be used to enter an interactive session of the SoftwarePilot docker container.

```
jayson@jayson-ThinkPad-T470:~/Github/SoftwarePilot/tools/Docker$ bash runAUAVLinux.sh
[sudo] password for jayson:
root@acf52e5667f4:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@acf52e5667f4:/# cd home/SoftwarePilot/
root@acf52e5667f4:/home/SoftwarePilot# ls
AUAVAndroid             Models     docs            interfaces    tmp
AUAVLinux               README.md  drivers.src     kernels       tools
Californium.properties  cacerts    external        libs          trace.data
Datasets                cmple.pl   externalDLLs    routines
LICENSE                 dji.docs   externalModels  routines.src
```

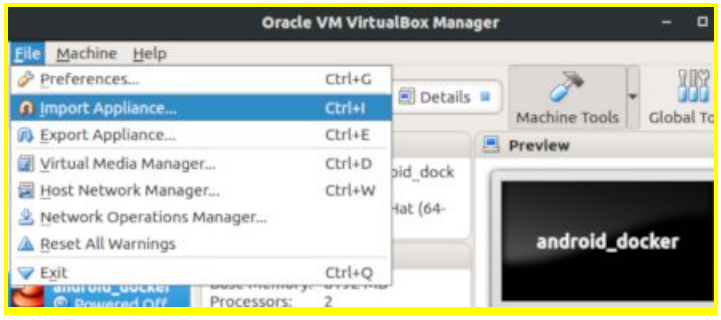
The above image shows the following sequence of steps:

- The runAUAVLinux.sh script is used to enter the SoftwarePilot docker container in an interactive session.
    - Note: sudo/administrator access is required
- Once in the container, CD to /home/SoftwarePilot, which is $AUAVHOME in the docker container.

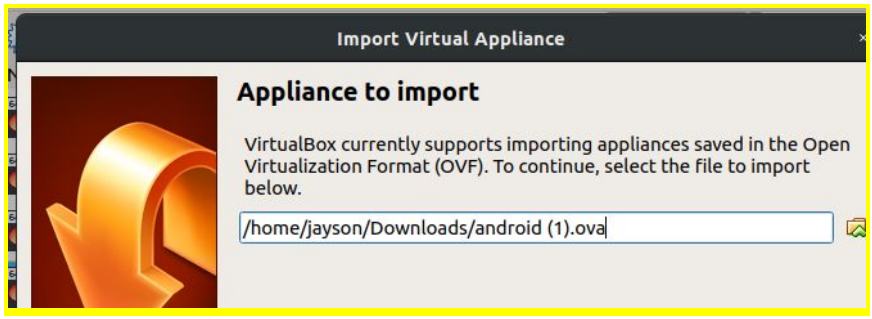
2.2.2 Virtual Machine Setup

Once you have downloaded the SoftwarePilot OVA and installed VirtualBox, you can set up the SoftwarePilot Virtual Machine

First, Open VirtualBox, and select File->Import Appliance to install the OVA:

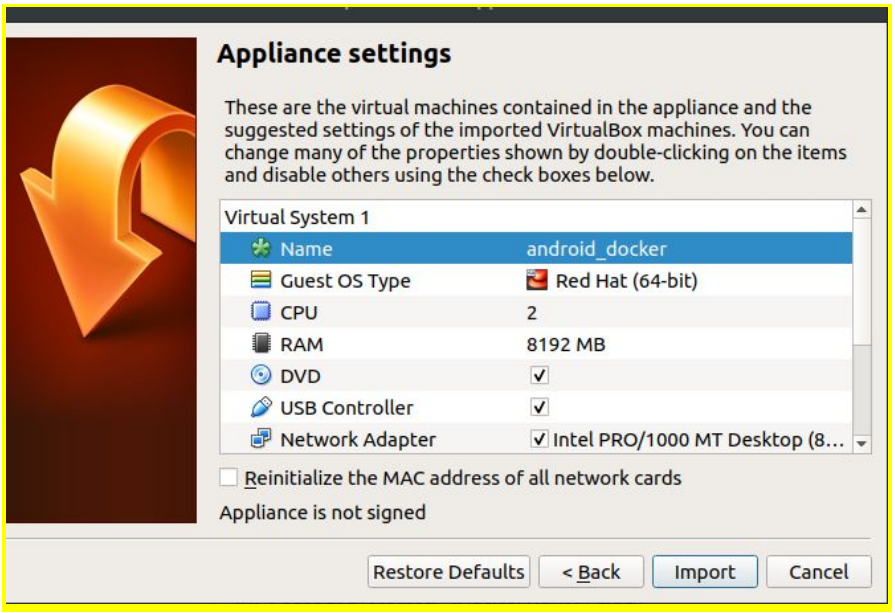


Select the OVA from your file system and import it:

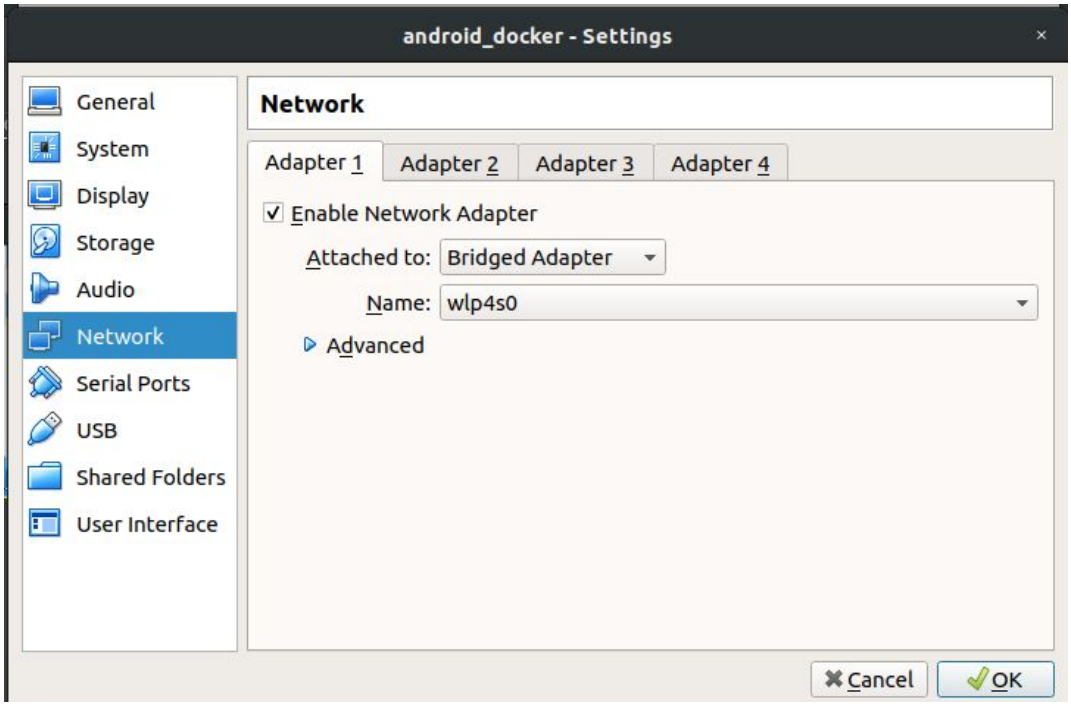


Name your OVA and allocate RAM and CPU cores:

Note: Make sure you allocate at least 2GB of ram to VirtualBox. We recommend at least 4GB, but more is (almost) always better as long as you leave some for your OS.



Once your OVA is installed, go to settings->network in VirtualBox to set up your network adapter. Make sure to enable at least one network adapter. Assure that your adapter is Bridged. Set your adapter name to the name of your wireless NIC. In the example below, the wireless network adapter of the host machine is named "wlp4s0".



3 Compiling and Executing SoftwarePilot

Due to SoftwarePilot's complexity, it can be difficult to compile and execute routines and driver calls. In this section, we describe the full process of compiling, transfering, and executing SoftwarePilot code on a UAV and edge system. In this section, we detail the SoftwarePilot Docker container, the principal development environment for SoftwarePilot code. We discuss the SoftwarePilot Virtual Machine, used for executing SoftwarePilot code on edge systems, as well as the process of compiling and transferring code between the Docker container and Virtual Machine.

3.1 The Docker Environment

The SoftwarePilot Docker container is a fully functional Ubuntu Linux environment capable of compiling and running any SoftwarePIlot code. The purpose of this Docker container is to allow any user on most modern operating systems to use and modify SoftwarePilot code. The docker container contains all SoftwarePilot code, as well as all Software to compile, transfer, and add to the SoftwarePilot codebase. The Docker container is meant to be a development environment as well as a mechanism for simply compiling and deploying existing code. Aside from the SoftwarePilot codebase, the Docker environment contains the following software features

- Editing:
  - Vim
  - Nano
- System Software
  - Git
  - Cmake
  - Curl
  - Unzip
  - Wget
  - Net-tools
- Programming
  - Python3.6
    - Python3 pip
    - opencv-python
    - Python face_recognition
    - yolo34py: YOLO for python
  - Java 8
    - ca-certificates for Java
  - Android 26

- Gradle 4.4

Under $AUAVHOME/tools/Docker/build, we have a second Dockerfile, Dockerfile.nvidia, which is identical to our original dockerfile, but adds Nvidia driver support. This script is currently only tested under Linux, but may work for other operating systems. If you wish to execute machine learning algorithms in the SoftwarePilot Docker container, build this version of the container.

Note: the SoftwarePilot codebase also opens the following ports on the host system's wireless NIC for communication between the Docker Container and Virtual Machine:

- 12013: Transfer data between SoftwarePilot code running on the VM and SoftwarePilot code running on the Docker container
- 5117: Access to the external commands driver (Detailed in the development guide)

3.1.1 Docker Scripts

The docker environment contains a series of shell scripts that make navigating the Docker environment slightly easier. None of these scripts are required to operate SoftwarePilot, but help to simplify the use of Docker. In this section, we detail each script and its purpose.

- build.sh:
  - This script builds the auavLinux docker image, the image for the SoftwarePilot docker environment. This script uses the Dockerile at $AUAVHOME/tools/docker/build/Dockerfile, but can be modified to use the Nvidia compatible Docker file.
- save.sh:
  - This script will commit the last active AUAVLinux docker container, saving all code and updates to the container to the AUAVLinux image.
- cleanDocker.bash:
  - This script cleans up docker images and containers that have been exited, reclaiming portions of the filesystem taken up by dead images.
- runAUAVLinux.sh:
  - This script will start an interactive session of the AUAVLinux container. This allows the user to enter the container and access the SoftwarePilot codebase and programming environment. This script also opens ports 5117 and

12013 on the host machine's principal network interface, and will not construct an interactive session unless those ports are available.

3.2 Compile Script

Inside the SoftwarePilot docker container, at $AUAVHOME (/home/SoftwarePilot/), there are a series of code-containing directories, along with a compile script (compile.py).

Note: Each file and directory in $AUAVHOME is detailed in the SoftwarePilot development guide. In the user-guide, we simply concentrate on compiling and building code.

To compile any component of the SoftwarePilot codebase, or deploy code to the VM, you can use the compile.py python script. The compile.py script has a number of functionalities and arguments, detailed below.

Compile.py flags and arguments:

- -h
  - The -h argument provides helpful statements detailing all arguments and flags
- -drv
  - The -drv flag compiles all SoftwarePilot drivers in the drivers.src directory. This flag takes no arguments, and is included in both the -code and -all flags. This Compilation step is critical to APK construction. SoftwarePilot can not be deployed without driver compilation using -drv, -all, or -code.
  - Example: python3 compile.py -drv
- -rtn
  - The -rtn flag compiles all SoftwarePilot routines in the routines.src directory. This flag takes no arguments, and is included in both the -code and -all flags. This Compilation step is critical to APK construction. SoftwarePilot can not be deployed without routine compilation using -rtn, -all, or -code.
  - Example: python3 compile.py -rtn
- -lin
  - The -lin flag compiles the SoftwarePilot linux kernel and interfaces needed for simulation. This flag takes no arguments and is included in both the -code and -all flags. This Compilation step is critical to APK construction. SoftwarePilot can not be deployed without interface and kernel compilation using -lin, -all, or -code.

    - Example: python3 compile.py -lin
- -mdl
    - The -mdl flag compiles select external models (secondary software not required for UAV flight that exist outside of routines, drivers, and interfaces) that require compilation. This flag takes no arguments and is included in both the -code and -all flags
    - Example: python3 compile.py -mdl
- -doc
    - The -doc flag compiles all javadoc associated with drivers, routines, interfaces, and the SoftwarePilot linux kernel. This flag is included in the -all flag, and is not APK critical.
    - Example: python3 compile.py -doc
- -cln
    - The -cln flag cleans temporary files from all gradle builds from the -drv, -rtn, -lin, and -andr flags. -cln is not included in any other flags, and will run before any other flags if called.
    - Example: python3 compile.py -cln
- -andr
    - The -andr flag compiles the SoftwarePilot android app APK. It is included in the -all flag. This compilation step should not be executed unless all drivers, routines, and interfaces at minimum have been compiled, either previously or in the execution. This Compilation step is critical to APK construction. SoftwarePilot can not be deployed without driver compilation using -andr or -all. Note: the first time you build an android APK on the SoftwarePilot docker container, you should connect your system to the broader internet, as you will need to download Android updates.
    - Example: python3 compile.py -andr
- -code
    - The -code flag compiles all sourcecode, meaning all code required to build the SoftwarePilot android APK, but not the APK itself. This flag compiles only mission-critical code, and is not inherently required to build the APK, but if combined with the -andr flag, will compile all code and build a complete APK.
    - Example: python3 compile.py -code
- -all

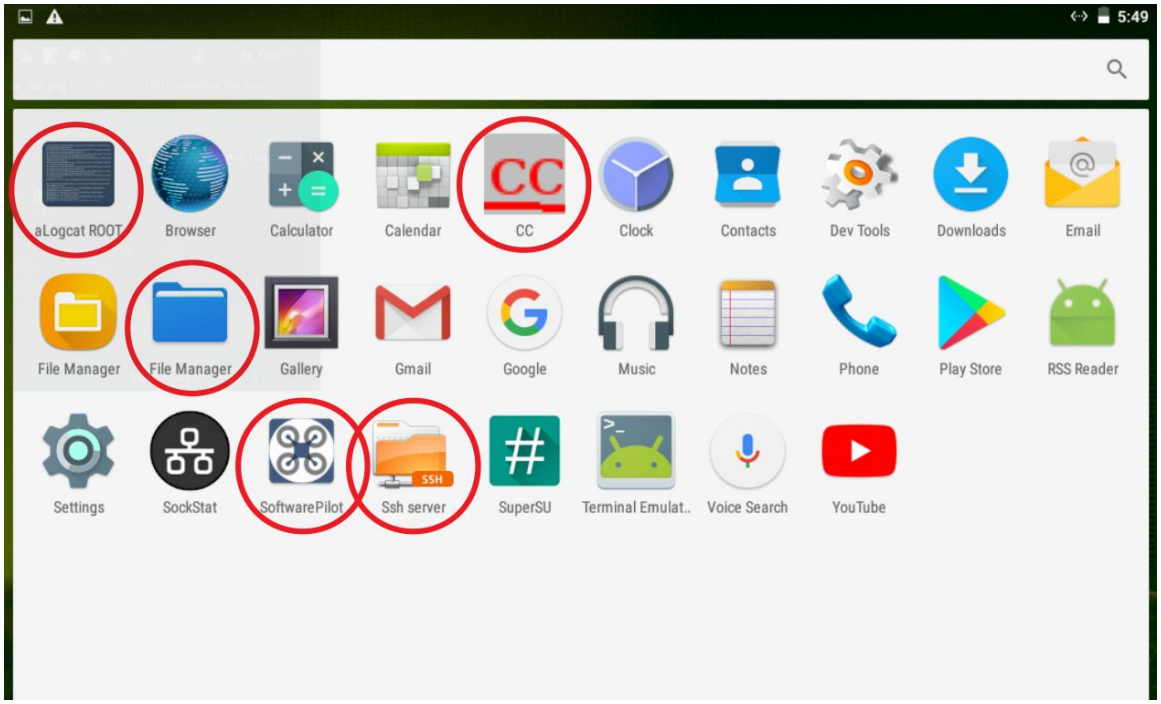
- The -all flag compiles all sourcecode and the SoftwarePilot APK, akin to running compile.py with the -code and -andr flags. Given successfull compilation of all steps, the -all flag will compile all code from scratch, and a complete Android APK. This flag is not technically required to build a successful APK, but will generate one given successful compilation of all required steps.
- Example: python3 compile.py -code

- -ins
  - The -ins argument will deploy and execute the SoftwarePilot android APK onto a SoftwarePilot virtual machine. The virtual machine is specified by its IP address. One the script is executed, this argument will transfer an APK onto the virtual machine, install it, and begin executing it. Note: This argument should only be executed when the host machine and VM are connected to the WIFI SSID of a DJI drone. This allows the APK to transfer directly across the bridged adapter instead of through the broader internet. Note: This process WILL fail on the first installation of a new VM. Check section 3.4 for details and next steps
  - Example: python3 compile.py -ins 192.168.2.21

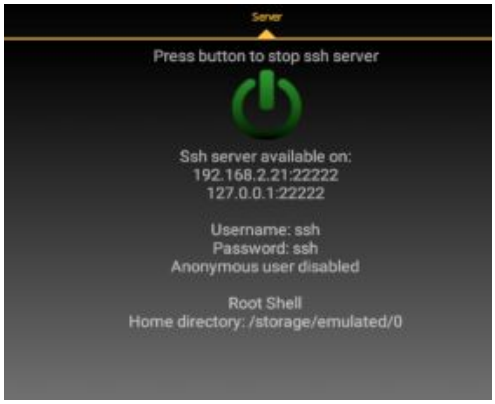
3.3 The SoftwarePilot Virtual Machine

The SoftwarePilot virtual machine is a modified x86 Android virtual machine build for two purposes: 1, to perform basic UAV actions, like capturing data, transferring data, movement, etc. with SoftwarePilot drivers, and 2) to manage data movement between the UAV and the edge system for routines that require complicated machine learning. Development of SoftwarePilot apps is detailed in the development guide, but setup and execution of the SoftwarePilot app is detailed in this section, and section 3.4.

The SoftwarePilot VM is a base Android system, with a series of important apps listed below:



- The File Manager
  - The File manager app provides access to the basic android filesystem. It may be required to access the SoftwarePilot APK if your APK must be manually installed
- SSH Server
  - The SSH Server app can be used to both open an SSH server to transfer files to and from the VM and find the IP address of the VM when transferring files using the compile script detailed in section 3.2. An example is shown below:



IMPORTANT Note

- The SoftwarePilot App
    - A base installation of the SoftwarePilot app is already available on the VM. To start this installation, or an updated version, simply click on the app when the host WIFI is connected to a compatible DJI UAV.
- CC
    - CC is a Coap Client used to execute SoftwarePilot drivers and routines when the SoftwarePilot app is running. This is the primary interface between the user and SoftwarePilot. Its use is documented in section 3.5
- Logcat
    - Logcat allows users to view live Android logs. App printouts, error messages, and more can be viewed by the user to aid with development. This app is covered further in the Development Guide.

3.4 Installing the SoftwarePilot App

SoftwarePilot installation can usually be done using the compile script detailed in section 3.2. There are important situations, however, where this process will not work. The intention of the -ins flag of the compile script is to install SoftwarePilot on the VM over a prior installation of SoftwarePilot, as an update. Usually, this process works just fine, but under two specific conditions, it will fail.

- The first installation of SoftwarePilot on a new VM is a special circumstance. the SoftwarePilot VM comes with a pre-installed version. Given that the normal development and testing process of SoftwarePilot requires updating the APK on the VM, the principal issue is that android apps build on two different systems are usually not compatible for updates. For this reason, we suggest that on your first installation (which, if following the -ins instructions should fail after APK transfer), to first uninstall the preinstalled APK (click and hold the SoftwarePilot app on the VM homescreen and choose uninstall). Then, enter the filesystem and doubleclick the AUAVAndroid.APK file to install the new APK. IMPORTANT NOTE: Every time you install the SoftwarePilot app on any android device, it MUST be connected to the broader internet, not a DJI UAV. The DJI SDK must authenticate itself with DJI's servers. Once you have executed the app when connected to the broader internet, you should be able to connect with a DJI drone when connected to the drones WIFI.
- Second, and less commonly, when major updates are added into the APK (a new library, updates to core library versions, etc), or to the underlying system (major OS version updates, total system changes), the APK may be incompatible with the prior installed version. This requires a full reinstall as described above, and authentication with DJI's servers.

3.5 Executing SoftwarePilot Drivers and Routines

SoftwarePilot drivers and routines can be executed both in the VM or on the host using CoAP command. The format and tools to execute commands are detailed below. Some SoftwarePilot routines and drivers also require code to run in the SoftwarePilot docker container. This process is also detailed below.

3.5.1 Command Format

The command format is as follows:

dn=$DRIVERNAME-dc=$DRIVERCOMMAND-dp=$DRIVERPARAM

- dn specifies a driver name (e.g. FlyDroneDriver, which handles all UAV flight actions).
- dc specifies a driver command specific to each driver (e.g. lft, a FlyDroneDriver command that lifts the UAV off the ground).
- dp specifies a driver parameter. Not all drivers require parameters, but some may require multiple parameters. Any number of driver parameters may be added to the end of the execution string

Executing Drivers:

- To execute a driver, simply run the command sequence above, specifying the name of the driver in the "dn" parameter.

Executing Routines:

- To execute a routine, set "dn" to "rtn". Set "dc" to "start", and set one parameter to dp=$ROUTINE_NAME, where $ROUTINE_NAME is the name of your routine.
- Example: to start the BasicRoutine routine, the following line can be used:
    - dn=rtn-dc=start=dp=BasicRoutine

3.5.2 Executing Commands

Commands can be executed both on the VM or host system.

- On the VM, commands can be executed using the CC app. Once the SoftwarePilot app is running and a UAV is connected, CC can be used to execute commands of the form described in section 3.5.1.
- Below is an example of the CC command to execute BasicRoutine, a simple SoftwarePilot routine.

- SoftwarePilot CoAP commands can also be executed on the host. SoftwarePilot contains a CoAP client application in $AUAVHOME/tools/coap-tool/ which can be used to execute well-formed coap commands of any type.
- We have also included a Pearl script, auav.pl, which takes a series of arguments allowing for SoftwarePilot command execution
- Arguments are as follows:
    - IP: The IP address of the SoftwarePilot VM
    - DN, the name of the driver you wish to invoke (or "rtn" for routines)
    - DC, the command for your given driver
    - DP, any series of driver parameters
- Example: ./auav.pl 192.168.2.21 rtn start BasicRoutine

```
jayson@jayson-ThinkPad-T470:~/Github/DroneProje
t/reroutlab.cstewart.code.auav/tools/coap-tool$
./auav.pl 192.168.2.21 rtn start BasicRoutine
v:1 t:CON c:PUT i:d7b4 {} [ ]
BasicRoutine: Started
```

3.5.3 Drivers and Routines with External Dependencies

Some SoftwarePilot calls require external dependencies. For instance, if a driver requires access to Tensorflow or DLIB library functions, these often can not be packaged into an APK and executed in Android. For this reason, any complex machine learning in SoftwarePilot is performed in the docker container. If you are executing any driver or routine that requires, you should execute the SoftwarePilot Linux kernel by performing the following commands:

- CD into $AUAVHOME/kernels
- type: ./AUAVLinux.sh

This will start the kernel and allow the VM to speak to the docker container. Note: any SoftwarePilot routine or driver that requires access to the Docker container will require, as a driver parameter, the IP address of your host WIFI adapter (the adapter provided to VirtualBox for use by the VM).