**MAVSDK-Python API Installationsanleitung:**

Die Installationnsanleitung soll als Ergänzung zur Anleitung auf <https://github.com/mavlink/MAVSDK-Python> verstanden werden. Zuvor muss AirSim, PX4 SITL und die Python API erfolgreich eingerichtet sein.

* Anaconda Command Prompt öffnen
* Mit dem folgenden Befehl das erstellte Conda Environment (myenv) öffnen, was bereits für die AirSim Python API verwendet wird.

conda activate myenv

* MAVSDK und asyncio Packet installieren:

pip3 install mavsdk

pip3 install aioconsole

* Falls MAVSDK auf einem Windows Host laufen soll muss der passende mavsdk\_server.exe heruntergeladen werden. Dieser kann auf dem MAVSDK github unter Releases gefunden werden <https://github.com/mavlink/MAVSDK/releases/tag/v1.4.2> . Wird MAVSDK auf Linux verwendet, wird das executable meistens mit installiert. Der mavsdk\_server muss unter folgendem Link abgelegt werden (Speicherort des MAVSDK Python Package: C:\Users\janik\anaconda3\envs\airsimpy2\Lib\site-packages\mavsdk\bin
* Damit nun der vorhin installierte mavsdk server auf dem Windows host mit dem PX4 SITL auf der Ubuntu Virtual Machine kommunizieren kann, muss Mavlink UDP Broadcast auf dem PX4 aktiviert werden. Dazu im PX4-Autopilot Ordner die Datei px4-rc.mavlink öffnen (zu finden unter: PX4/PX4-Autopilot/build/px4\_sitl\_default/etc/init.d-posix

Die Datei enthält die folgenden Befehle. Die gelb hinterlegten Befehle von -m auf -p ändern.

*#!/bin/sh*

*# shellcheck disable=SC2154*

*udp\_offboard\_port\_local=$((14580+px4\_instance))*

*udp\_offboard\_port\_remote=$((14540+px4\_instance))*

*[ $px4\_instance -gt 9 ] && udp\_offboard\_port\_remote=14549 # use the same ports for more than 10 instances to avoid port overlaps*

*udp\_onboard\_payload\_port\_local=$((14280+px4\_instance))*

*udp\_onboard\_payload\_port\_remote=$((14030+px4\_instance))*

*udp\_onboard\_gimbal\_port\_local=$((13030+px4\_instance))*

*udp\_onboard\_gimbal\_port\_remote=$((13280+px4\_instance))*

*udp\_gcs\_port\_local=$((18570+px4\_instance))*

*# GCS link*

*mavlink start -x -u $udp\_gcs\_port\_local -r 4000000 -f -p*

*mavlink stream -r 50 -s POSITION\_TARGET\_LOCAL\_NED -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s LOCAL\_POSITION\_NED -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s GLOBAL\_POSITION\_INT -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s ATTITUDE -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s ATTITUDE\_QUATERNION -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s ATTITUDE\_TARGET -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s SERVO\_OUTPUT\_RAW\_0 -u $udp\_gcs\_port\_local*

*mavlink stream -r 20 -s RC\_CHANNELS -u $udp\_gcs\_port\_local*

*mavlink stream -r 10 -s OPTICAL\_FLOW\_RAD -u $udp\_gcs\_port\_local*

*# API/Offboard link*

*mavlink start -x -u $udp\_offboard\_port\_local -r 4000000 -f -p onboard -o $udp\_offboard\_port\_remote*

*# Onboard link to camera*

*mavlink start -x -u $udp\_onboard\_payload\_port\_local -r 4000 -f -p onboard -o $udp\_onboard\_payload\_port\_remote*

*# Onboard link to gimbal*

*mavlink start -x -u $udp\_onboard\_gimbal\_port\_local -r 400000 -p gimbal -o $udp\_onboard\_gimbal\_port\_remote*

Mit den gemachten Änderungen wird mavlink vom PX4 Kontroller nun die entsprechenden UDP Ports gebroadcasted. Der Windows Host kann nun auf den entsprechenden UDP Ports die Signale abhören und für Befehle empfangen.

* Damit nun AirSim Python API und MAVSDK API beide Offboard Controls an PX4 Senden können, muss eine zweite Instanz für Offboard Links via Mavlink geöffnet werden. Dazu wird das selbe File wie folgt bearbeitet (gelb hinterlegt):

*#!/bin/sh*

*# shellcheck disable=SC2154*

*udp\_offboard\_port\_local=$((14580+px4\_instance))*

*udp\_offboard\_port\_remote=$((14540+px4\_instance))*

*[ $px4\_instance -gt 9 ] && udp\_offboard\_port\_remote=14549 # use the same ports for more than 10 instances to avoid port overlaps*

*udp\_offboard\_port\_local\_mavsdk=$((14581+px4\_instance))*

*udp\_offboard\_port\_remote\_mavsdk=$((14541+px4\_instance))*

*udp\_onboard\_payload\_port\_local=$((14280+px4\_instance))*

*udp\_onboard\_payload\_port\_remote=$((14030+px4\_instance))*

*udp\_onboard\_gimbal\_port\_local=$((13030+px4\_instance))*

*udp\_onboard\_gimbal\_port\_remote=$((13280+px4\_instance))*

*udp\_gcs\_port\_local=$((18570+px4\_instance))*

*# GCS link*

*mavlink start -x -u $udp\_gcs\_port\_local -r 4000000 -f -p*

*mavlink stream -r 50 -s POSITION\_TARGET\_LOCAL\_NED -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s LOCAL\_POSITION\_NED -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s GLOBAL\_POSITION\_INT -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s ATTITUDE -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s ATTITUDE\_QUATERNION -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s ATTITUDE\_TARGET -u $udp\_gcs\_port\_local*

*mavlink stream -r 50 -s SERVO\_OUTPUT\_RAW\_0 -u $udp\_gcs\_port\_local*

*mavlink stream -r 20 -s RC\_CHANNELS -u $udp\_gcs\_port\_local*

*mavlink stream -r 10 -s OPTICAL\_FLOW\_RAD -u $udp\_gcs\_port\_local*

*# API/Offboard link*

*mavlink start -x -u $udp\_offboard\_port\_local -r 4000000 -f -p onboard -o $udp\_offboard\_port\_remote*

*mavlink start -x -u $udp\_offboard\_port\_local\_mavsdk -r 4000000 -f -p onboard -o $udp\_offboard\_port\_remote\_mavsdk*

*# Onboard link to camera*

*mavlink start -x -u $udp\_onboard\_payload\_port\_local -r 4000 -f -p onboard -o $udp\_onboard\_payload\_port\_remote*

*# Onboard link to gimbal*

*mavlink start -x -u $udp\_onboard\_gimbal\_port\_local -r 400000 -p gimbal -o $udp\_onboard\_gimbal\_port\_remote*

Es wird eine neuer Local Port und ein neuer Remote Port für die Mavlink API geöffnet und dieser Link dann beim Start des PX4 Kontrollers ebenfalls gestartet. **Beim Windows Host muss der entsprechende Remote Port in der Firewall geöffnet werden. Im Beispiel hier ist dies Port 14541.**

* Um die MAVSDK-Python Schnittstelle nun zu aktivieren muss der mavsdk\_server gestartet werden. Unter Windows muss dies manuell geschehen, sofern das Executable (mavsdk\_server.exe) nicht im gleichen Ordern wie das verwendete Python Skript liegt. Dazu via Anaconda Prompt das Verzeichnis der executable Datei öffenen und den Server mit dem folgenden Befehl starten.

Mavsdk\_server udp://:14541

Der Befehl udp://: sagt dem Server, wo es die einkommenden Mavlink Signale empfangen soll. Es muss mit dem Remote Port aus der obigen Datei übereinstimmen.

* Liegt das Python Script und die mavsdk\_server Datei im selben Ordner kann der mavsdk\_server direkt via Python Script gestartet werden. Dazu im nachfolgenden Befehl den UDP Port wie in Schritt zuvor beschrieben angeben. Wird der Server manuell gestartet kann der Befehl drone.connect() leer gelassen werden.

await drone.connect(system\_address=’udp://:14541’)

Für weitere Infos zu den Skripts und Befehlen sie die Beispiele auf: <https://github.com/mavlink/MAVSDK-Python>