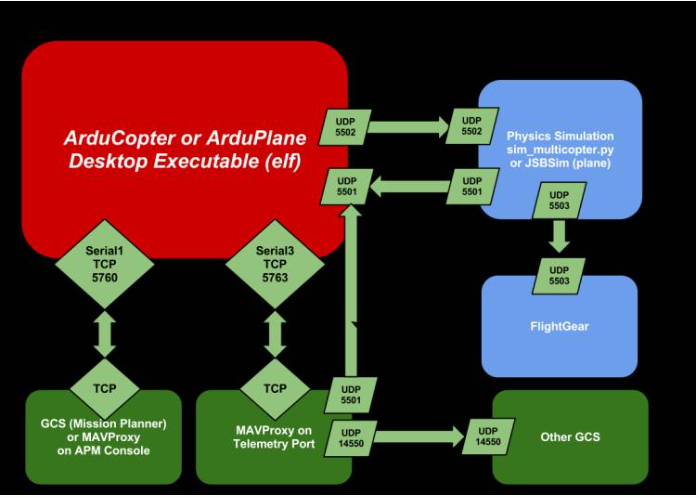
Dronekit-python 정리

픽스호크오면 아두파일럿 시틀에 실제 픽스호크 연결해서 시뮬가능할 듯?

[https://ardupilot.org/dev/docs/using-sitl-for-ardupilot-testing.html#using-sitl-for-ardupilot-testing](https://ardupilot.org/dev/docs/using-sitl-for-ardupilot-testing.html" \l "using-sitl-for-ardupilot-testing)



미션플래너or Qground control같은 GCS에서 드론과 연결하려면 TCP5760포트

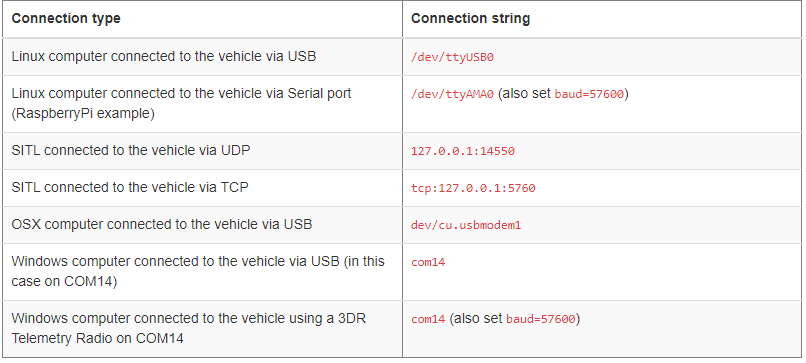
아두파일럿 sitl에서 마브프록시로 TCP연결하려면 TCP5763포트 사용

UDP는 14550포트 사용

* **HelloDrone**

1. **connect함수**

**connection\_string:** UDP, TCP통신으로 하면 IP주소값 맞추고 아닐 시 직렬통신(실물드론)으로 연결한 기기의 모델을 입력 직렬통신시 **baudrate값 맞출 것**



**wait\_ready**: GCS(미션플래너)와FC연결하면 파라메터를 읽어오는데 그 파라메터를 읽어읽어올 것인지 안읽을것인지 플래그

True -> FC에 설정된 파라메터를 읽어온다.

False -> FC의 파라메터를 읽어오지않음.

* **Vehicle\_state**

**지정가능한 속성값: (home location, gimbal, airspeed, groundspeed, vehicle mode, vehicle armed)**

**Global Location:** 해발고도 기준 보통 고도(alt)가 200~500정도로 높음

**Global Location(relative altitude):** home위치 기준->home에 착륙상태면 0

**Local Location:** north, east, west, south기준으로 위치정보

**Attitude:** (pitch,yaw,roll)기체의 자세정보 보통 라디안값 (수평인지 이런 것)

**Velocity:** x, y, z축 기준의 속도정보

**GPSInfo**: (eph, epv, ifx\_type ,satellites\_visible)

**Eph**: HDOP 수평위치 오차를 뜻함 (0에 가까울수록 좋음. 0은 불가능 2~3정도> 우수 , 4~5 >보통 , 6이상은 위험)

**Epv**: VDOP 수직위치 오차

**Fix**\_**type**: 0-1:no fix 2: 2D fix 3: 3D fix -> fix3되야 안정적

**satellites**\_**visible**: 기기에 잡혀있는 위성의 개수

**EKF**: 확장 칼만 필터로 속도 자이로스코프, 가속도계, 나침반, GPS, 대기 속도 및 기압 측정을 기반으로 차량 위치, 속도 및 각도 방향을 추정하는 데 사용

**home\_location:** 홈위치를 리턴 ( 설정할떄 vehicle\_state의 111번째줄 참고)

From dronekit inport connect,VehicleMode,**LocationGlobal # 홈위치 변경시 LocationGlobal 추가**

my\_location\_alt = LocationGlobal(37.389991869, 126.73789196, 0)

my\_location\_alt.alt=222.0 #이건 없어도 될 듯

vehicle.home\_location = my\_location\_alt

여기서 vehicle.home\_location에 좌표값넣어주면 될 듯?

**System status:** 시스템 상태를 나타냄

* UNINIT: FC를 인식하지못함
* BOOT: 부팅상태
* CALIBRATING: Calibration수행중 (눈금매기기?)
* STANBY: 시동 걸기 전
* ACTIVE:시동ON 상태
* CRITICAL:위험한상태지만 비행은 가능
* EMERGENCY: 비상상황 추락상황
* POWEROFF: POWER OFF상태

**Heading**: 360도 이내의 각도 0도가 북쪽

**Is\_armable**:시동 가능한지

**Airspeed**:속도 (지정가능) 미션비행or가이드비행에서 지정가능 m/s

**Groundspeed:(**세팅가능)

**Battery:** Level을 통해 현재 배터리 상황 파악

**Last heartbeat**:마지막 통신텀 1초가 넘어가면 문제있음

**비행모드:** STABLIZE=0 GUIDED=4 AUTO= 3 프로그래밍 기체제어가 끝나면 보통 LOITER=5 RTL=6 LAND=9



Vehicle.mode는 “스트링”으로 입력해도 되고 숫자값으로 넣어도 됨

실내에서 비행하고자 한다면 Althold모드로 설정

**상태를 바꾸면 상태값이 바꼈는지 확인하는 코드 필수**

Vehicle.mode=VehicleMode(“GUIDED”)

time.sleep(2)

vehicle.armed= True

cnt=0

while not vehicle.mode.name==’GUIDED’ and not vehicle.armed :

print(“Getting ready to take off..”)

time.sleep(1)

cnt+=1

if(cnt>10):

print(“Fail to arming”)

break

**Listeners:**내가 리스너로 걸어놓은값이 변경되면 ex)위치정보,YAW값 콜백함수로 알려줌 wildcard\_callback을 리스너로 지정시 모든 값에 리스너 지정

def attitude\_callback(self, attr\_name, value):

    # `attr\_name` - the observed attribute (used if callback is used for multiple attributes)

    # `self` - the associated vehicle object (used if a callback is different for multiple vehicles)

    # `value` is the updated attribute value.

    global last\_attitude\_cache

    # Only publish when value changes

    if value!=last\_attitude\_cache:

        print(" CALLBACK: Attitude changed to", value)

        last\_attitude\_cache=value

콜백 매소드 정의(Listener방법)

vehicle.add\_attribute\_listener('attitude', attitude\_callback)

Listener추가

vehicle.remove\_attribute\_listener('attitude', attitude\_callback)

Listener제거

@vehicle.on\_attribute('mode')

def decorated\_mode\_callback(self, attr\_name, value):

    # `attr\_name` is the observed attribute (used if callback is used for multiple attributes)

    # `attr\_name` - the observed attribute (used if callback is used for multiple attributes)

    # `value` is the updated attribute value.

    print(" CALLBACK: Mode changed to", value)

Decorator 메소드 정의(Decorator방법)

* **이륙 및 드론컨트롤**

1. **Taking off(이륙)**
2. Pre-arm check

일단 드론킷파이썬으로 픽스호크 연결후 vehicle\_armable속성으로 시동가능 여부 체크

1. Flight Mode Change

vehicle.mode =VehicleMode(“GUIDED”)설정

1. ARMING

vehicle.armed =TRUE 설정

1. Taking off

Vehicle.simple\_takeoff(고도값 미터)명령을 실행

드론이 지정된 고도로 이륙한뒤 그 자리에서 대기 take-off명령을 수행하기 위해서 비행모드 꼭”GUIDED”로 설정, ARMING상태여야함

\*드론이 지정고도에 도달해도 특정 리턴값없음 수동으로 비교하여 확인

\*이륙도중 다른명령어 수신되면 즉시 수신받은 명령어 수행

이륙 도중 Simplegot명령어 수신되면 기체가 즉시 이동한다

1. **Simple\_goto**
2. Simplegoto는 가이드모드에서 실행
3. 이동하고자 하는 위치는 위도, 경도, 고도 사용 함수의 인자값으로 LocationGlobal(해수면 고도) or LocationGlobalRelative(이륙위치 기준의 상대고도) 중 한가지 사용
4. 이동속도(m/s)는 airspeed(optional) 또는 groundspeed(optional) 중 한가지 사용

**Airspeed**: m/s로 일정한 값을 가짐

**Groundspeed**: airspeed에 바람속도 값이 반영된 값 바람을 등지면 속도가 빨라지고 맞바람이면 느려짐 (역풍이 불면 속도가 느려지고 ,바람이 밀어주면 더 빠르게 감)

1. **Airspeed 또는 groundspeed를 simplegoto()메소드 호출시 주석으로 명시하는 것이 바람직함** speed가 설정되지 않았을 경우와 둘다 설정되어 있다면 자동비행시 기준이 명료하지 않게된다.
2. 이동을 완료하면 해당지점에서 대기 takeoff와 마찬가지로 리턴값이 없으므로 수동으로 이동완료를 확인해야함

**\*드론킷으로 코딩시 명령어실행후 꼭 명령어가 실행이 잘 됐는지 확인하는 코드 넣어서 확인할 것!**

**\*드론의 경위도 좌표의 거리의 차이를 구하고 1m이내이면 완료된 것이라 판단하면 됨**

**\*simple goto()명령어의 해당좌표로 이동이 완료되었다는 기준은 목표지점 반경 몇cm이내 도달하면 목적지에 왔다라는 것을 판별 WPNAV\_RADIUS파라메터 값이 있다. Ex)50으로 설정하면 50cm? 이내에 도달하면 웨이포인트에 도착했다고 판별하고 호버링**

**\*simple goto()명령어 수행시 compass calibration 과 GPS 3D fix 를 반드시 점검할 것 드론세팅이 완벽하지 않은 상태에서는 목표지점에 도달하고 전후좌우로 짧게 몇번 더 이동하는 경우 발생**

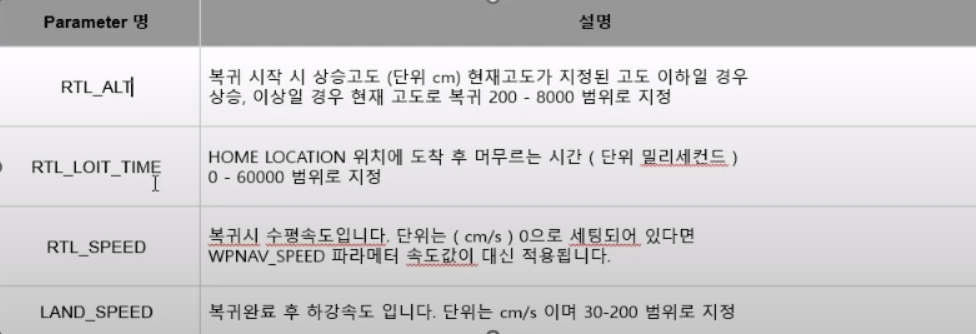
**\*안전한 비행을 위해서 드론이 자동으로 목표지점에 도달했는지 수동으로 거리차이를 구한 뒤 1m이내라면 Auto,Loiter,Poshold모드 등 으로 비행모드 전환 후 그 자리에서 대기하는 코드를 구현하는것이 바람직함!**

* **주의할점:코드상에서는 30초 걸린후 바로 두번쨰WayPoint로 가는 명령어를 실행한다. 하지만 기체는 여러가지 환경에 의해 첫번째 웨이포인트에 도착하지 않았을수도 있다. 이부분은 감안하고 프로그래프 짤 것 시간 좀 넉넉하게 해야할 듯?**

1. RTL

기지(home location)으로 복귀하는 기능 vehicle.mode = VehicleMode(“RTL”)로 구현 하지만 RTL은 언제든지 실행할 준비가 되어있어야 할 중요사항

RTL 관련 파라미터 항목 꼭 기억하고 비행전에 미션플래너에서 설정 확인해야하며 ,**긴급한경우 드론킷 파이썬으로 설정하여 비행가능**



**WPNAV\_SPEED 기준비행속도**

* **위치 및 속도기반 기체 컨트롤**

하늘에서 기체를 원하는 곳으로 이동시키기 위해서는 기준이 되는 좌표계가 필요

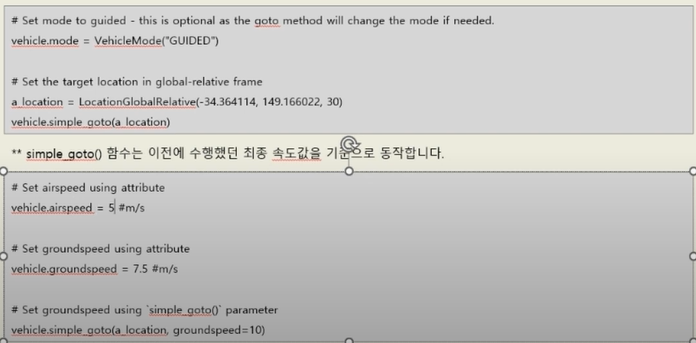
위치기반 이동에선 **WGS84(World Geodetic System 1984)** 경위도 좌표를 사용하여 특정지점 및 고도로 기체를 이동시킴

또 다른 한가지는 **NED(North, East, Down)좌표계**이다. 이때 남, 서, 위로는 어떻게 움직이냐면 북쪽에서 -값을 주면 남쪽으로, 동쪽에서 -값을 주면 서쪽으로 이동한다. (0, 0, 0)이 드론이 이륙 전 현재위치이다. **기체의 헤딩방향이 아닌 지도상의 북쪽을 의미한다.**



1. **위치기반 이동**

고도값은 LocationGlobal(해발고도) 또는 LocationGlobalRelative(상대고도) 속성중 한가지를 사용



1. **속도기반 이동**

속도기반 이동은 위치기반 이동보다 특수상황(사물 추적 등)에서 움직임이 자연스러워서 많이 사용

send\_ned\_velocity()아래 함수는 프레임 SET\_POSITION\_TARGET\_LOCAL\_NED에서 차량의 속도 구성요소를 직접 지정하는 데 사용되는 마브링크 메시지 생성 **MAV\_FRAME\_LOCALNED(**홈 위치에 상대적임)메시지는 지정된 시간 동안 매초마다 다시 전송

사용할때 SOUTH=-4이런식으로 알아보기 쉽게 변수를 만들어 사용하면 편함

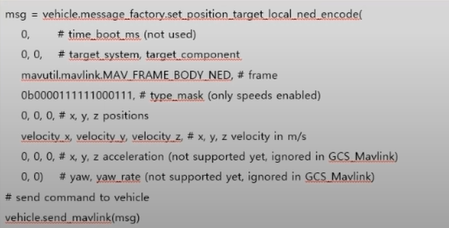


1. **가속도 조절**

**드론킷에서는 가속도 조절 불가**

1. **GUIDED모드 MAVLINK커맨드**
2. **메시지 보내기**

MAVLink 메시지는 message\_factory()를 이용해 메시지를 MMAVLink메시지로 인코딩 한 뒤 send\_mavlink() 메소드를 이용하여 보냄. 예를들어 SET\_POSITION\_TARGET\_LOCAL\_NED 메세지는 message\_factory 메소드 뒤에 \_encode()를 붙여서 보내게 된다.



**\***target-system id 항목은 ‘0’을 넣어주면 드론킷이 알아서 connect메소드에서 찾은 시스템ID로 넣어줌. 한대의 드론만 컨트롤 할때는’0’ 넣어 사용 -> **2개 이상 컨트롤 할때는 찾아보자**

<https://dronekit-python.readthedocs.io/en/latest/automodule.html>

드론킷 API Reference 참고

<https://mavlink.io/en/messages/common.html>

마브링크 메시지 참고

**드론 개발에서 꼭 알아가야 좋다.**

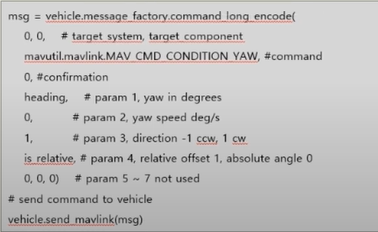
1. **MAVLink 커맨드 보내기**

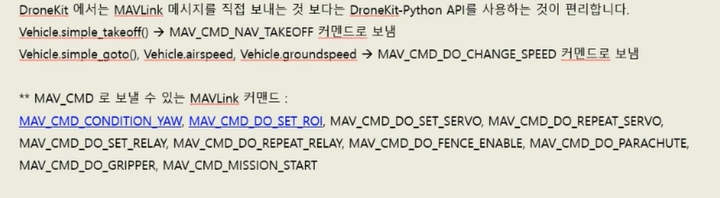
**\*마브링크에는 메시지와 커맨드가 따로 존재 메시지로 상태정보 또는 명령어를 보낼 수 있고 지정된 커맨드로도 명령을 보낼 수 있다.**

COMMAND\_LONG message 도 마찬가지로 message\_factory 메소드로 사용 가능

Ex)

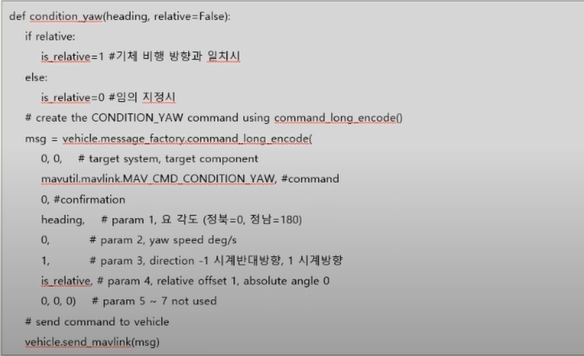
yaw방향 변경하기 위한 **MAV\_CMD\_CONDITION\_YAW** MAVLink 커맨드를 보내는 방법





1. **YAW(기수방향) 세팅**

아두콥터에서는 기수의 방향이 비행방향과 일치할 필요 X 따라서 기수의 방향을 원하는 지점으로 변경 할 수 있음. 기체의 YAW방향은 MAV\_CMD\_CONDITION\_YAW커맨드 사용하여 변경

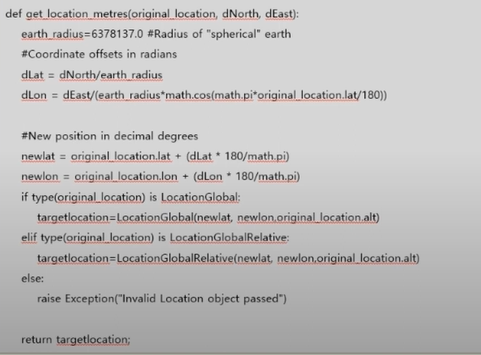


**heading:**지도상 북쪽 방향기준 0 남쪽방향은 180

**relative:** False로 설정시 각도의 기준을 NED의 북쪽기준으로 설정 1로 설정시 기체의 방향기준으로 설정

1. **프레임 변환 함수들**
2. **get\_location\_meters(original\_location, dNorth, dEast)**

현재 위치를 기준으로 미터단위 거리를 이동할 때 자동으로 LocalGlobal 프레임으로 변환해 줍니다.

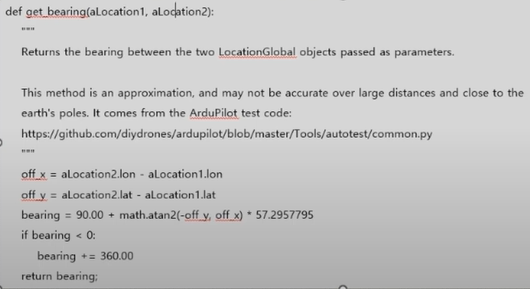


Ex)어느 위치에서 북쪽으로 (50)미터 동쪽으로 (100)미터 이동하려할 때 그 위치의 위도경도 좌표값 리턴해줌.

1. **get\_bearing(aLocation1, aLocation2)**

매개변수로 전달된 두 LocationGlobal 객체 사이의 베어링을 반환

앞서 내가 가고싶은 위치의 좌표를 얻어왔을 때 현재위치와 목표위치의 기수방향의 각도를 구할 때 사용함 (기수의 YAW방향 설정할 때 사용하면 됨

)

지구에서 방위각 구하는 공식을 사용