ELA NUR ZAMUR / 1800002622 DRONE MODEL : AR.Drone 2.0*0*

# **Drone Programmıng**

**The first step of drone programming is knowing how to use the right programming languages, which could be low-level or high-level programming. Next, you can begin programming by using a programmable drone to access its programming and build your own app using a programming language.**

**Low-level programming, the focus is the firmware that the drone runs on. This firmware is what communicates or coordinates with the drone’s hardware such as the propellers and the motor so that it would be able to make use of its basic functions. In short, the firmware is what allows the drone to use all of its different components.**

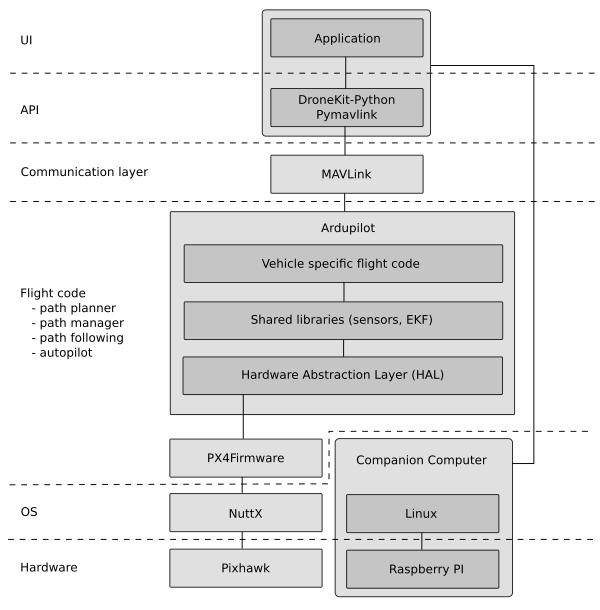
**So, for example, the drone’s firmware is actually what is responsible for determining how much power it should deliver to the motors and its propellers depending on how it is controlled or how it analyzes the drone’s inertial measurement unit. In that way the drone is able to stabilize itself as the firmware communicates with the drone’s motors and propellers while in the middle of a flight. When it comes to low-level programming C and C++ are programming languages that you need to learn how to use.**

**High-level programming deals with the applications of the drone. In short, you are already looking at the drone as entirely complete and you only need to come in with an app that will allow you to control the drone on command. So, the different functions that will allow you to program the drone to do a lot of different things such as flying to a certain altitude or knowing how to use information so that it can automatically fly to that location via GPS are functions that can be programmed using high-level programming.**

**What happens here is that high-level programming works hand in hand with the firmware so that the drone can interpret the response that it should perform. So, for example, if you use the drone to fly at a controlled altitude on command, the app will communicate with the firmware, which will then communicate with the drone’s hardware.** **This is where a programming language such as Python comes into play because this language tends to be the easiest programming language to learn.**

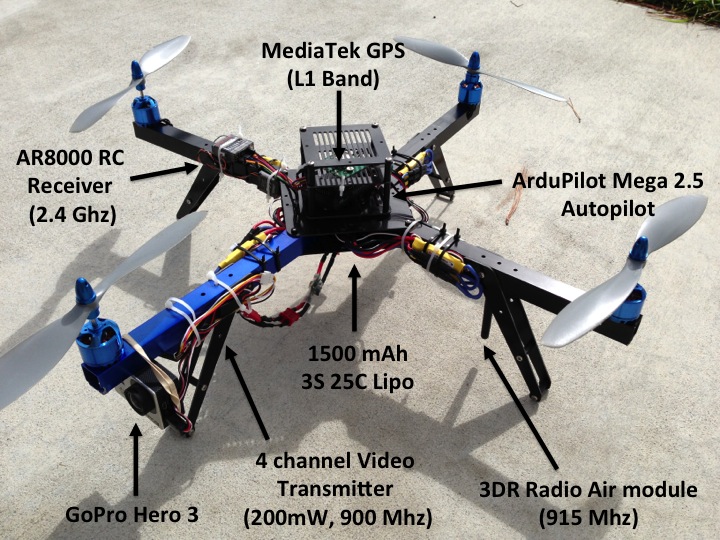
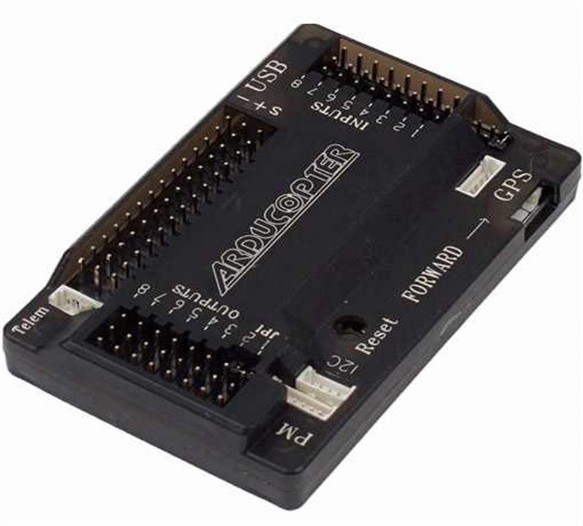
DRONE MODEL : AR.Drone 2.0*0*

# **PROGRAMMING LANGUAGES FOR DRONE**

****

* **Firmware: Ardupilot**

**Just like how Windows firmware is responsible for communicating with the computer hardware, Ardupilot is responsible for commanding a drone’s hardware. Actually, without ArduPilot, or some flight control firmware, it would be impossible to fly multirotor uavs. That is because ArduPilot sends around 400 commands a second to the drone’s motors, which translates into smooth and steady flight.**

**** 

* **Software: Dronekit python**

**Dronekit python is an open source python library that provides high level functions to command the drone’s movement, check vehicle status and many other things. Essentially, this unlocks the application layer to drone programmers. It’s possible to perform an autonomous drone delivery mission in 200 lines of code. Also, dronekit and openCV combine to implement computer vision into your drone programming.**

**Examples of dronekit library:**

1. **Connecting to a Vehicle**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**The first parameter specifies the target address (in this case the loopback address for UDP port 14550). The second parameter (wait\_ready) is used to determine whether connect() returns immediately on connection or if it waits until *some* vehicle parameters and attributes are populated. In most cases you should use wait\_ready=True to wait on the default set of parameters.**

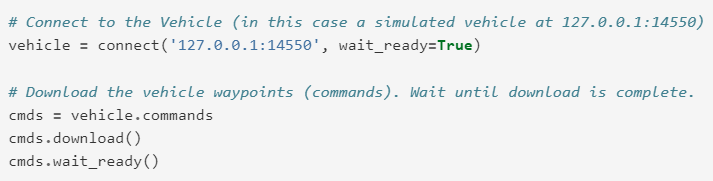
1. **Taking Off**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**The function first performs some pre-arm checks.** **These checks are encapsulated by the Vehicle.is\_armable attribute, which is true when the vehicle has booted, EKF is ready, and the vehicle has GPS lock.** **Once the vehicle is ready, we set the mode to GUIDED and arm it. We then wait until arming is confirmed before sending the**[**takeoff**](https://dronekit-python.readthedocs.io/en/latest/automodule.html#dronekit.Vehicle.simple_takeoff)**command.** **When the function returns the app can continue in GUIDED mode or switch to AUTO mode to start a mission.**

1. **Missions**
   * + **Download current mission**

****

**The mission commands for a vehicle are accessed using the**[**Vehicle.commands**](https://dronekit-python.readthedocs.io/en/latest/automodule.html#dronekit.Vehicle.commands)**attribute. The attribute is of type [CommandSequence](https://dronekit-python.readthedocs.io/en/latest/automodule.html" \l "dronekit.CommandSequence" \o "dronekit.CommandSequence), a class that provides ‘array style’ indexed access to the waypoints which make up the mission. Waypoints are not downloaded from vehicle until**[**download()**](https://dronekit-python.readthedocs.io/en/latest/automodule.html#dronekit.CommandSequence.download)**is called.**

* + - **Modifying missions**

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

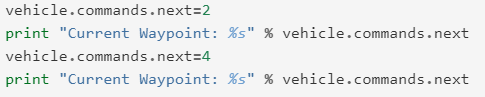
**While you can**[**add new commands**](https://dronekit-python.readthedocs.io/en/latest/guide/auto_mode.html#auto-mode-adding-command)**after**[**downloading a mission**](https://dronekit-python.readthedocs.io/en/latest/guide/auto_mode.html#auto-mode-download-mission)**it is not possible to directly modify and upload existing commands in Vehicle.commands (you can modify the commands but changes do not propagate to the vehicle).Instead you copy all the commands into another container (e.g. a list), modify them as needed, then clear Vehicle.commands and upload the list as a new mission.**

* + - **Running and monitoring missions**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**You can stop/pause the current mission by switching out of AUTO mode (e.g. into GUIDED mode). If you switch back to AUTO mode the mission will either restart at the beginning or resume at the current waypoint - the behavior depends on the value of the**[**MIS\_RESTART**](http://copter.ardupilot.com/wiki/arducopter-parameters/#mission_restart_when_entering_auto_mode_mis_restart)**parameter (available on all vehicle types).**

****

**You can monitor the progress of the mission by polling the [Vehicle.commands.next](https://dronekit-python.readthedocs.io/en/latest/automodule.html" \l "dronekit.CommandSequence.next" \o "dronekit.CommandSequence.next) attribute to get the current command number. You can also change the current command by setting the attribute to the desired command number.**

* **MAVLink Protocol**



**When you try and call someone’s phone number, there is structure to this. You must dial the country code, then the local area code, and then the individuals number. This is a standardized system. If I want to call my Uncle Ron, I must use this standard system AND know the numbers which represent him.**

**In essence, this is what MAVLink is for drone communication. The**[**MAVLink protocol**](http://mavlink.io/en/)**consists of two things:**

1. **Packet Structure: Just like how all phone numbers have the same (country)-(local)-(personal) structure, MAVLink offers a standard packet structure so it is easy to receive and transmit data. MAVLink 1 data consists of just 8 bytes, and MAVLink 2 data consists of 14 bytes.**
2. **Standard Messages: If I want to call my Uncle Ron, I simply dial his number 1-999-9999 and he picks up (hopefully). Similarly, the MAVLink protocol offers specific messages with specific functions behind them. For example, MAVLink message 78 and command 22 is a takeoff command that would launch the drone into the air, with the target height contained within the message. In addition to commands, MAVLink messages can be strictly informational as well. For example, MAVLink message 25 holds the current GPS status of the vehicle.**

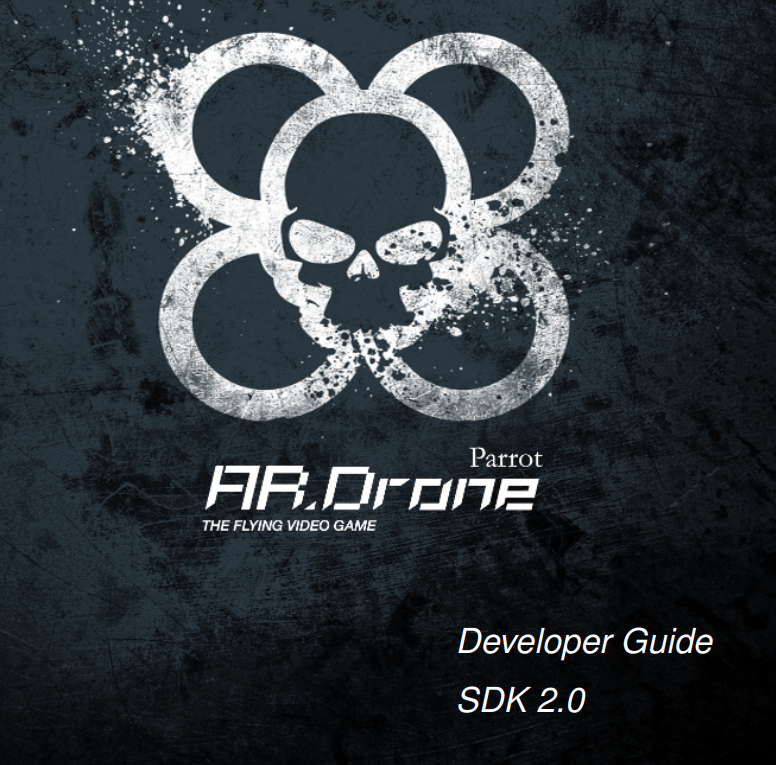
**How is the MAVLink Protocol Implemented?**

**Think of MAVLink as the glue that allows Dronekit Python and ArduPilot to communicate. It offers standard messages which can be accepted and understood by the ArduPilot firmware.**

**The neat thing about MAVLink is that it works with any drone firmware which is MAVLink enabled, not just Ardupilot. There are many different types of MAVLink enabled drone firmware, another one being PX4.**

# SDK (SOFTWARE DEVELOPMENT KIT)

Welcome to the AR.Drone 2.0 Software Development Kit !

The AR.Drone 2.0 product and the provided host interface example have innovative and exciting features such as:

• intuitive touch and tilt flight controls

• live video streaming

• video recording and photo shooting

• updated Euler angles of the AR Drone

• embedded tag detection for augmented reality games

The AR.Drone 2.0 SDK allows third party developers to develop and distribute new games

based on AR.Drone 2.0 product for Wifi, motion sensing mobile devices like the Apple iPhone,

iPad, iPod touch, personal computers or Android devices.

<https://jpchanson.github.io/ARdrone/ParrotDevGuide.pdf>

This SDK includes :

• this document explaining how to use the SDK, and describes the drone communications

protocols;

• the AR.Drone 2.0 Library (ARDroneLIB ), which provides the APIs needed to easily communicate and configure an AR.Drone 2.0 product;

• the AR.Drone 2.0 Tool (ARDroneTool ) library, which provides a fully functionnal drone

client where developers only have to insert their custom application specific code;

• the AR.Drone 2.0 Control Engine library which provides an intuitive control interface developed by Parrot for remotely controlling the AR.Drone 2.0 product from an iOS Device;

• several code examples that show how to control the drone from a Linux personal computer.

• source code for iOS and Android1 AR.FreeFlight 2.0 applications.