**Introduction**

The C++ Linux example is meant to showcase recommended application-layer usage of the DJI Onboard API.

This example eases a new developer into the world of programming for drones - the sample makes heavy use of the well-abstracted [osdk-wrapper](https://github.com/dji-sdk/Onboard-SDK/tree/3.2/osdk-wrapper) library. Packaged with the Linux sample is a pthread-based threading implementation as well as an efficient serial device driver that implements many checks (on x86 systems) to ensure reliable communication between your Onboard Embedded System (OES) and your drone.

Since OSDK 3.1.9, the Linux example app comes in two flavors - synchronous (Blocking) and asynchronous (Nonblocking).

Version 3.2.0 adds support for [LiDAR Collision Avoidance](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/modules/collision-avoidance/collision-avoidance.html), [LiDAR Mapping](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/modules/lidarmapping/lidar-mapping.html) and the updated and improved [Precision Missions](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/modules/missionplan/README.html) suite. Unlike the previous version, the sample is [configurable](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/github-platform-docs/Linux/README.html) and can run with or without any of these modules.

**Synchronous Linux Example App**

APIs for the following user-facing functionality are implemented in the Linux sample:

| **Setup/Teardown** | **Camera Control** | **Flight Control & GPS Missions** | **New! Advanced Features** |
| --- | --- | --- | --- |
| Activation | Still Image Capture | Takeoff/Landing | [Precision Missions](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/modules/missionplan/README.html) |
| Obtain/Release Control | Video Capture | Return to Home | [LiDAR Collision Avoidance](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/modules/collision-avoidance/collision-avoidance.html) |
|  | Gimbal Control | Position/Velocity/Attitude Control | [LiDAR Mapping](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/modules/lidarmapping/lidar-mapping.html) |
|  |  | Waypoint Missions | [LiDAR Logging](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/sensor-integration-guides/velodyne/readme.html#Software-Guide) |

**Asynchronous Linux Example App**

Apart from the Blocking Linux sample, we also support a reduced feature set Non-Blocking (callback-based) sample for this release. This allows the callbacks to run on a different thread, allowing the send commands to run independent from the callbacks.

The following user-facing functionality is available in the new Non-Blocking Linux sample:

| **Setup/Teardown** | **Flight Control** |
| --- | --- |
| Activation | Takeoff/Landing |
| Obtain/Release Control |  |

The examples are extensible - if you want to build additional functionality, it is easy to do so within the framework of these examples.

**Setup**

**1. Hardware**

The [Hardware Setup](https://developer.dji.com/onboard-sdk/documentation/development-workflow/hardware-setup.html) guide talks about setting up your OES of choice. Make sure your setup matches that in the document before proceeding further.

**2. Software**

The following instructions are valid for both Blocking and Non-blocking Linux samples. Advanced features are only available on the Blocking sample.

**2.1 Toolchain**

To build either Linux example, you need:

* A supported C++ compiler (Tested with gcc 4.8.1/5.3.1)
* A bash shell
* CMake
* A modern Linux distribution

**2.2 Compilation**

| **OSDK Version** | **Build system** | **How to build** | **Compatibility** |
| --- | --- | --- | --- |
| 3.1.8 | makefile | In sample/Linux, type make | Builds on all Linux systems |
| 3.1.9 | CMake | See [here](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/github-platform-docs/Linux/README.html#2-3-cmake-build-steps) | Only builds on Ubuntu 16.04/14.04 (x86\_64) or Ubuntu 14.04 (ARM 32-bit) |
| 3.2.0 | CMake | See [here](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/github-platform-docs/Linux/README.html#2-3-cmake-build-steps) | Builds on all platforms, advanced features limited to Ubuntu 16.04 x86\_64 |

To access the serial port, add your username to the dialout group by typing sudo usermod -a -G dialout $USER (you do not need to replace $USER with your username). **Then logout and login again.**

**2.3 CMake Build Steps**

For **3.1.9**:

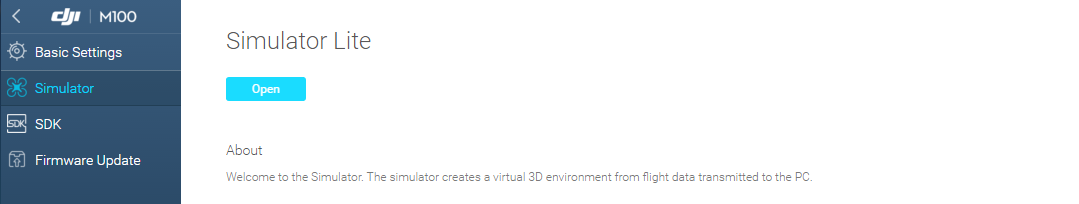
* In sample/Linux/Blocking, create a build directory and cd into it.
* Type cmake .. -DLIDAR\_LOGGING=[ON|OFF]. If you want LiDAR logging, choose -DLIDAR\_LOGGING=ON. Logging is OFF by default. Do not use the [] braces while specifying your option.

For **3.2.0**:

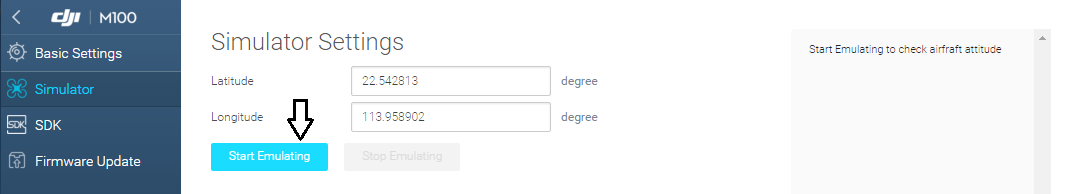
* In top level Onboard-SDK, create a build directory and cd into it.
* Type cmake .. -DUSE\_PRECISION\_MISSIONS=[ON|OFF] -DUSE\_COLLISION\_AVOIDANCE=[ON|OFF] -DUSE\_POINTCLOUD2LAS=[ON|OFF]. All the options are off by default. Do not use the [] braces while specifying your option.
* The resulting executable is Onboard-SDK/build/bin/djiosdk-linux-sample

**2.4 Using the Simulator**

* Connect your M100/M600/A3/N3 to a PC through USB.
* Open up DJI Assistant 2. Click on the DJI M100/M600/A3 button. If this button doesn't show up, try disconnecting and reconnecting the USB.
* Click on the Simulator tab, and then click on the 'Open' button. A separate window should pop up in a few seconds.

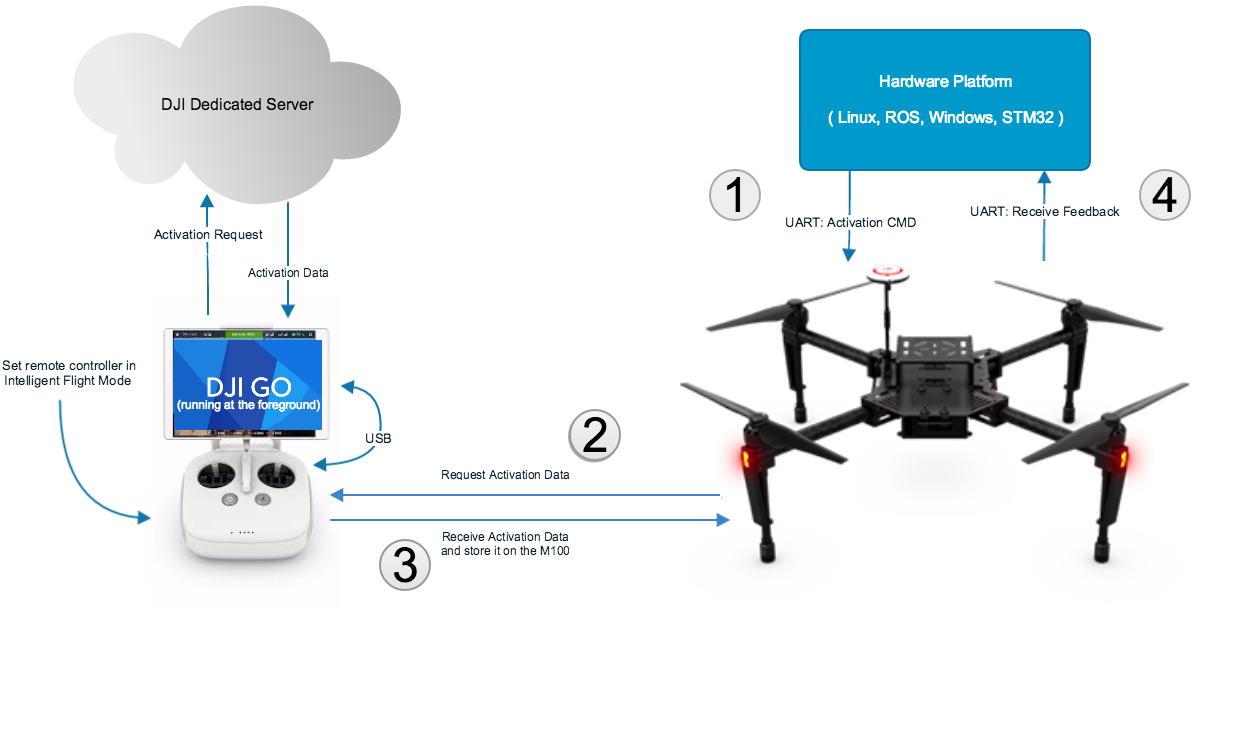


* In the main window, click on 'Start Emulating'.



**Activation**

The first time a drone/OES combination is used, it needs to be activated. Activation requires an App ID and a key got from the DJI website, and requires an internet connection and an RC connected to a mobile device running DJI GO. The image below shows the flow of information during activation.

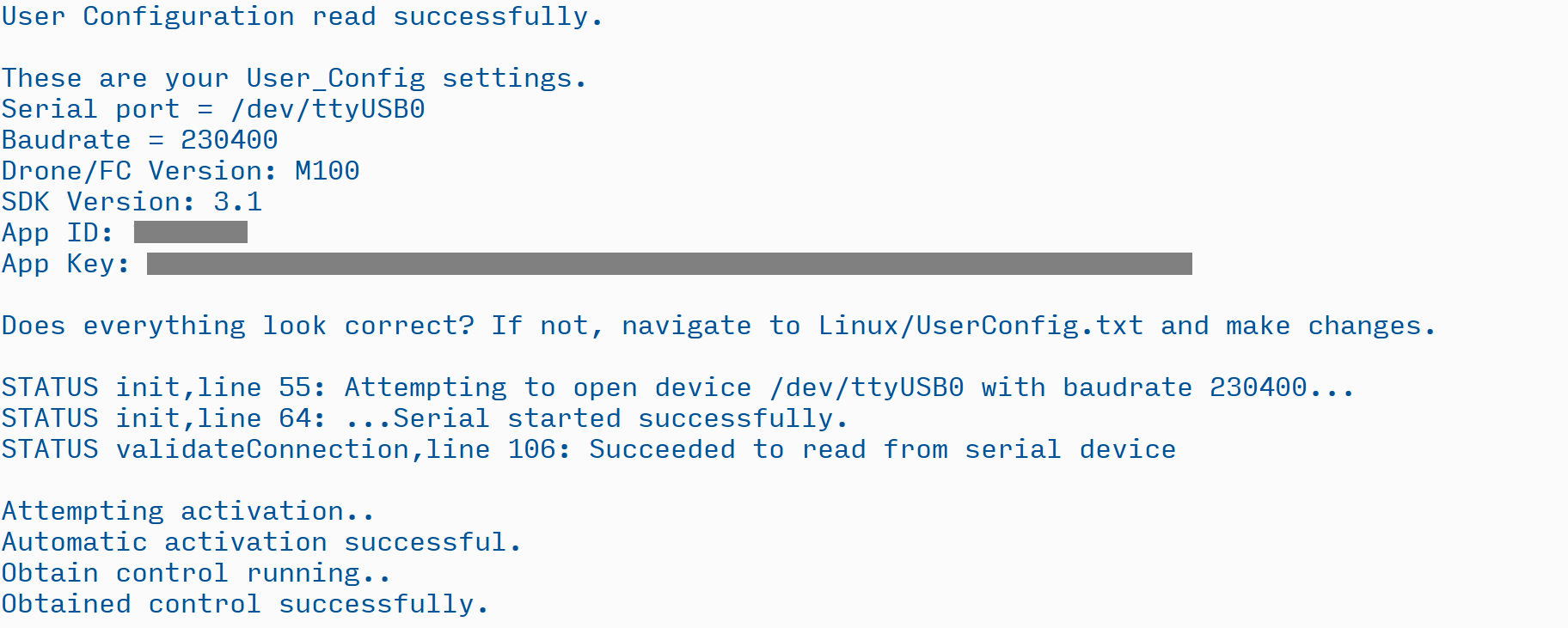


1. [Enable API control](https://developer.dji.com/onboard-sdk/documentation/development-workflow/environment-setup.html#Enable-Flight-Controller-API-control) and [get an app ID and key](https://developer.dji.com/register/).
2. Navigate to sample/Linux/UserConfig.txt and enter your App ID and Key in place of the defaults in that file.
3. Starting from OSDK 3.2.1, you do not need to enter your drone version. The SDK queries the drone and activates accordingly.
4. Proceed to the instructions under [Operation](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/github-platform-docs/Linux/README.html#Operation)

**Operation**

To run the Linux Blocking sample for Onboard SDK 3.2.0, follow these steps:

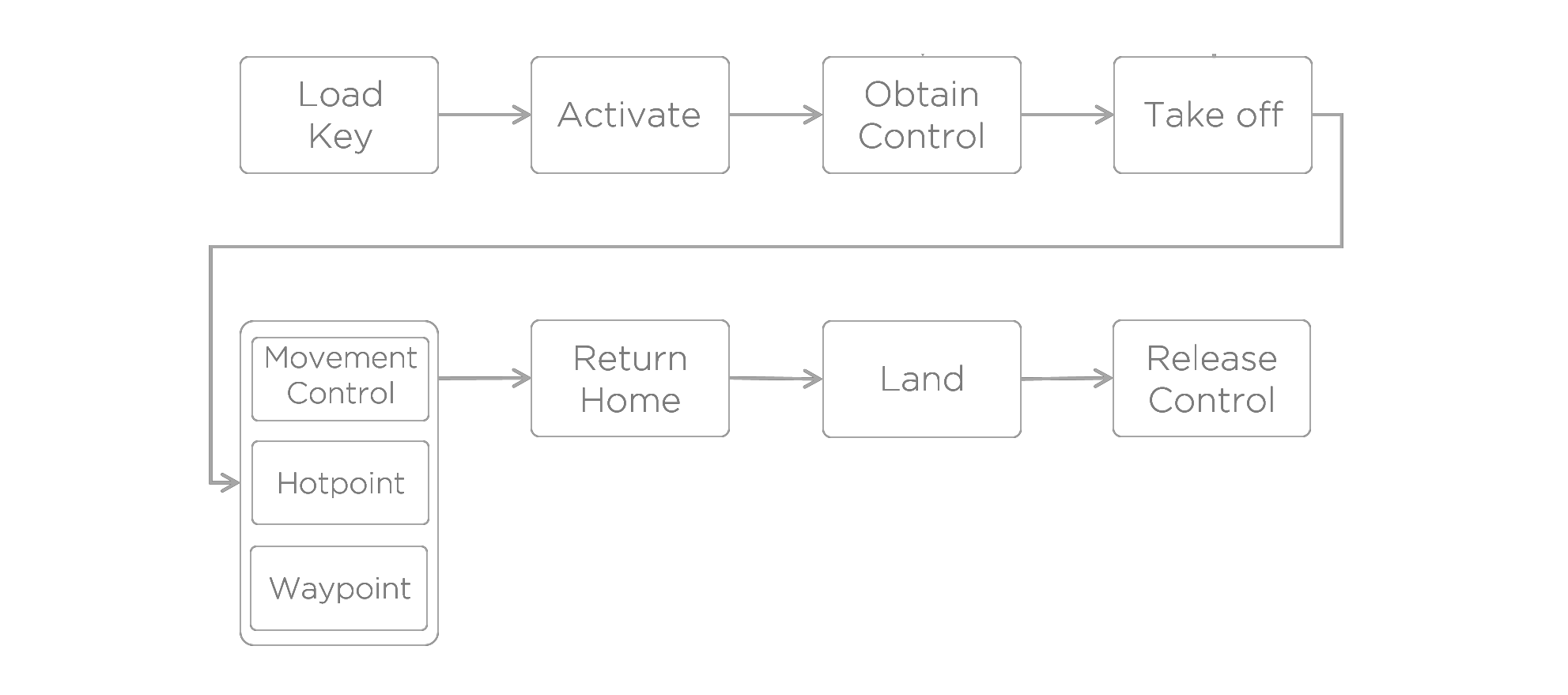
* Navigate to build/bin/ and copy the UserConfig.txt file from onboardsdk/onboardsdk/sample/Linux/Blocking. In this file, enter your serial port in the DeviceName and baud rate in the BaudRate field.
  + The default baudrate is 230400. If you change this, remember to also change it in DJI Assistant 2.
  + The default port is /dev/ttyUSB0. This should be correct if you are using a USB-Serial adapter. On Manifold, you will be using /dev/ttyTHSx (x = 0,1,2) - refer to the [Hardware Setup Guide](https://developer.dji.com/onboard-sdk/documentation/development-workflow/hardware-setup.html) for more information.
* In the build/bin folder, run djiosdk-linux-sample [mode\_of\_operation] [(optional)path\_to\_spiral] [(optional)path\_to\_tuning] where
  + mode\_of\_operation can be (more information about the modes [here](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/github-platform-docs/Linux/README.html#modes-of-operation)):
    - -interactive : Recommended mode for new developers. Shows a basic terminal UI and users can execute single commands with key presses.
    - -mobile : Use with the brand new Mobile OSDK App. Useful for mobile-based triggering of OSDK commands with keys on iOS device.
    - -programmatic : Use for automated execution. By default, the sample will first takeoff, then execute a waypoint mission, then automatically land and exit.
  + (optional)path\_to\_spiral is the file path of a pre-planned spiral (json file) for precision trajectory planning. For more information, look at [Precision Missions](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/modules/missionplan/README.html#software-setup)
  + (optional)path\_to\_tuning is the file path of a controller tuning (json file) for precision trajectory planning. For more information, look at [Precision Missions](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/modules/missionplan/README.html#software-setup)
* Proceed to run the sample in one of these modes. Before the sample enters one of the three modes, it will attempt to activate the drone and obtain control. If everything goes well, you should see the following information on the terminal:



* You may follow the same instructions to run the Non-Blocking Linux sample. The Non-Blocking Linux sample supports a reduced feature set for '-interactive' and '-mobile' modes and does not support '-programmatic' mode. It also does not support any of the advanced features.

Note that the activation step is necessary each time. After the first time, the activation command merely performs a local activation check and you are not required to be connected to the internet.  
The sample will attempt automatic activation each time it is started.

**Workflow**



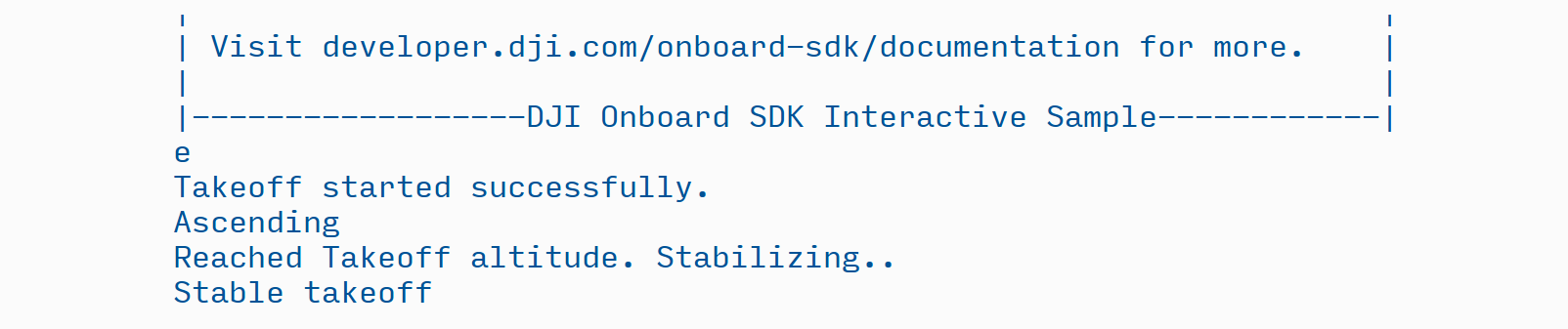
**Modes of Operation**

**Interactive Mode**

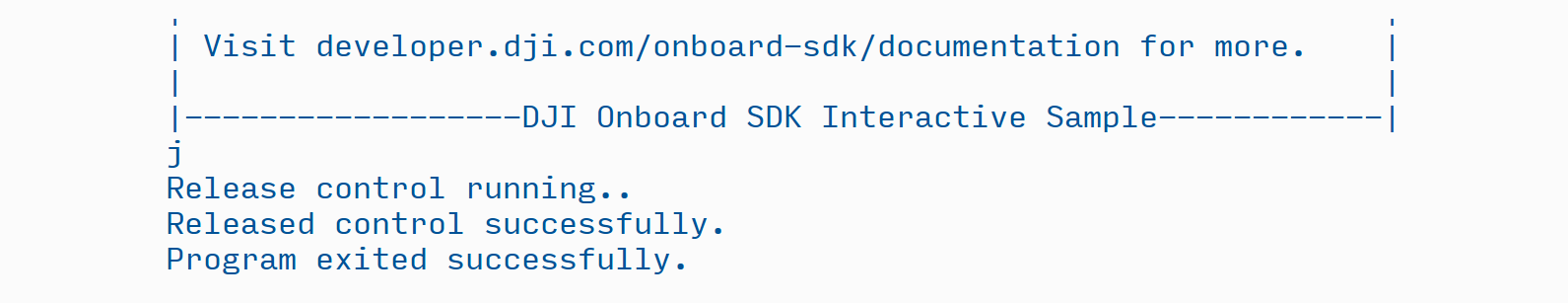
The interactive mode is meant to give you a taste of the kinds of things you can do with the DJI Onboard SDK. The UI looks like this:



As the UI tells us, the sample has already activated and taken control on our behalf. Let's try the takeoff command - to do so, press e and then hit the enter key.



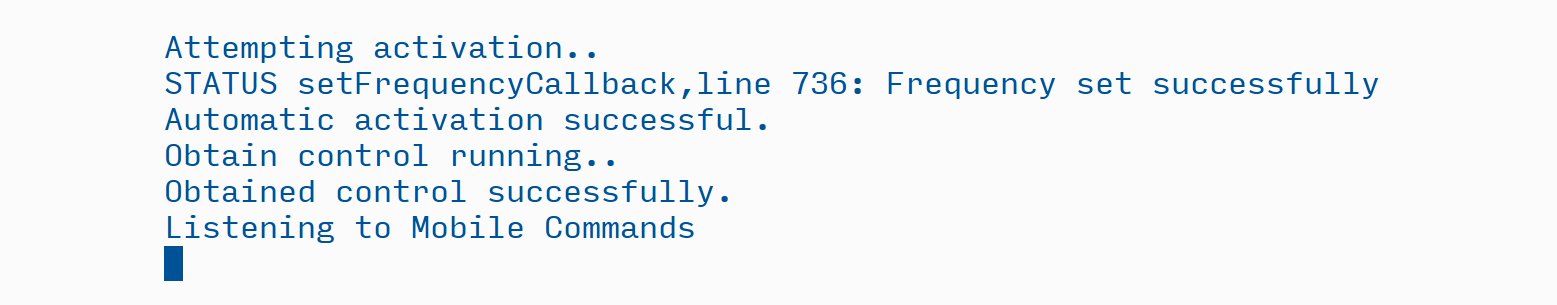
Once you have taken off, you may try some of the flight control functionality. After you have landed, type the exit letter - n - to let the sample clean up before exiting.



**Mobile Mode**

The mobile mode is especially useful when you want to test out functionality on the Onboard SDK in a staged or interactive manner - simply make your code poll for commands from the companion mobile app and execute your mission/task/commands based on the mobile command sent. The mobile link is set up with a companion iOS [app](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/github-platform-docs/MobileOnboardSDK/Mobile-OSDK.html).

When you run the program in mobile mode, you can see a simple message after the standard initialization:



The mobile mode listens to mobile commands for ~150 minutes before exiting.

**Programmatic Mode**

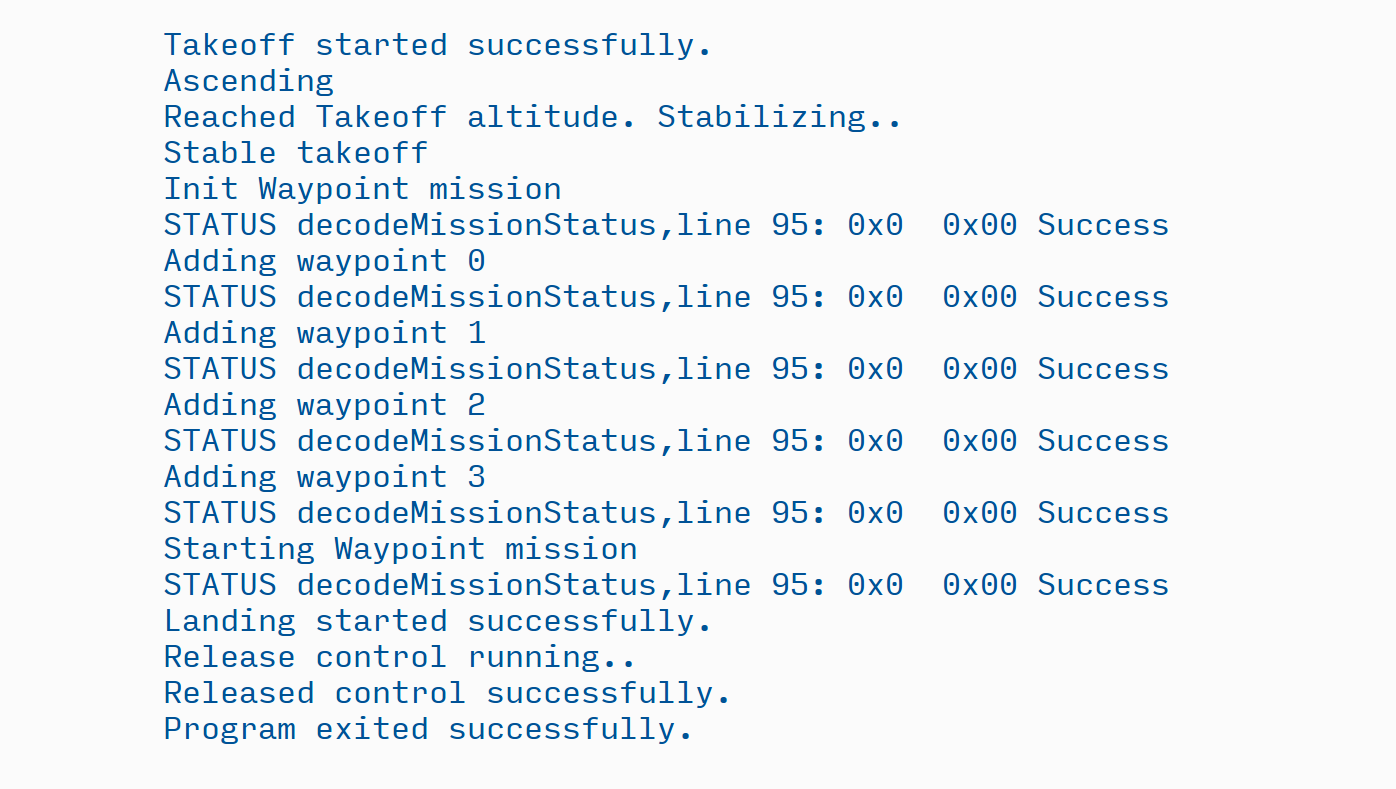
This is the most powerful mode of the new Linux sample - you can implement a complex, custom mission involving local navigation, trajectory following and waypoints using the programmatic mode. Together with [core API calls](https://developer.dji.com/onboard-sdk/documentation/M100-Docs/application-development-guides/programming-guide.html), you can add hotpoint missions, automated picture and video to your automated program in this mode.

By default, the programmatic mode starts with automated, monitored takeoff (checks to see if takeoff actually executed, as well as how stable the takeoff is). Users can enter their own core API calls or Linux-sample calls after the takeoff returns and before landing is called. Here is the relevant code in main.cpp:



Line 108 is an example of an application layer function called within the programmatic mode.

The result of this code is something like this:



**Examples**

The Linux Blocking Application comes with two examples you can call through Interactive, Mobile or Programmatic mode:

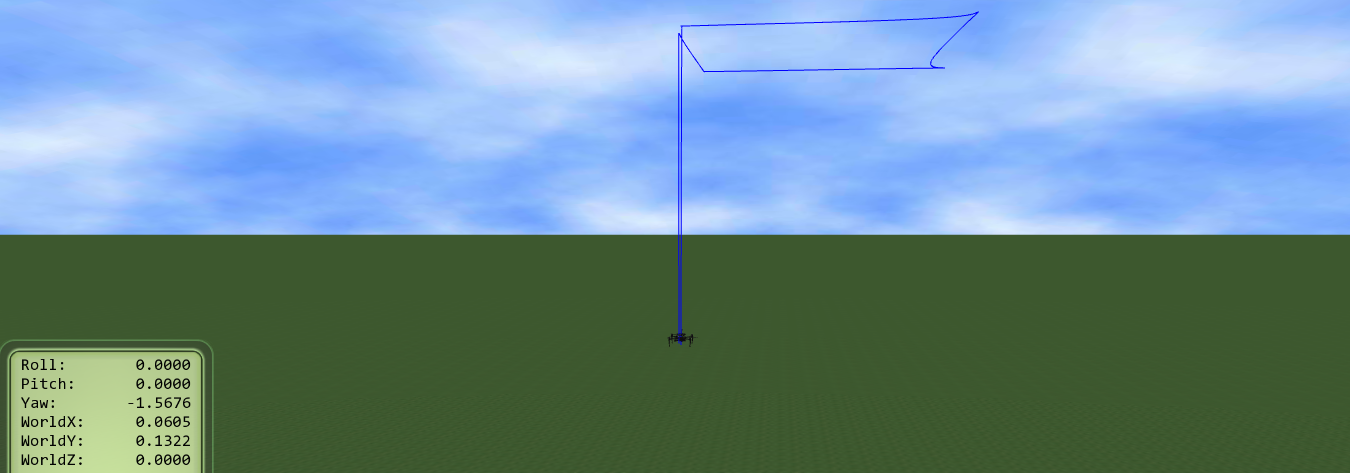
**1. Waypoint Mission Example**

This example takes your current position (lat, lon, alt) and traces out a square of side 20 m. It will go north, then east, then south, and then west; if you are unaware of where your aircraft is pointing then **make sure to leave a clearing of 20m in all directions** when you try this example.

To try out the sample:

* In interactive mode, first takeoff with the e key and then try the waypoint sample with the f key.
* In mobile mode, first takeoff from the Core Functions tab. Then go to the Custom Missions tab and choose Waypoint Mission Test.
* In programmatic mode, call wayPointMissionExample after calling monitoredTakeoff.

The output looks like this (takeoff - waypoint test - landing):



**2. Draw a Square : Position Control Example.**

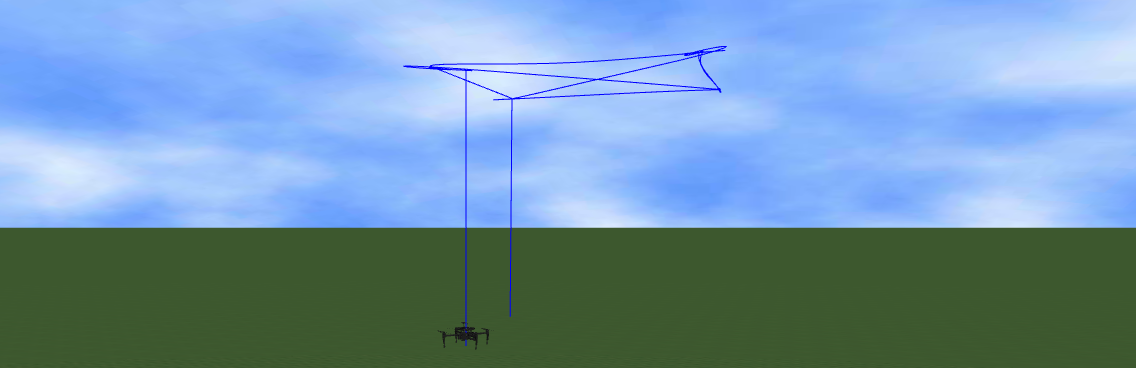
The Position control example shows how to execute custom trajectories in local coordinates, useful for planning in a local space, executing complex trajectories and for use in GPS denied environments (with [DJI Guidance](http://www.dji.com/product/guidance) or [Velodyne Puck](http://velodynelidar.com/vlp-16-lite.html), for example) This example builds on the movement control functionality offered by the core API and adds a very simple receding setpoint algorithm to maintain constant speed. This sample will draw a square of side 10m and then proceed to draw two diagonals. First, the drone ascends 10 m from its current location. Then, it first moves right, then forward, then left, and backwards to its starting point. Then it will draw a diagonal at +45 degrees, go left along the side of the square, and draw the other diagonal at -45 degrees. **Leave a clearing of 20m around the drone before trying this example.**

The implementation of this example in LinuxFlight.cpp can serve as a great reference for any custom patterns you might want to draw.

To try out the sample:

* In interactive mode, first takeoff with the e key and then try the Draw a Square sample with the g key.
* In mobile mode, first takeoff from the Core Functions tab. Then go to the Custom Missions tab and choose Draw a Square.
* In programmatic mode, call drawSqrPosCtrlSample after calling monitoredTakeoff.

The output looks like this (takeoff - draw a square - landing):



**3. Camera And Gimbal Controls**

X3 and X5 camera fixed to an aircraft will record images that pitch and roll with the aircraft as it moves. Multi rotor aircraft need to pitch and roll simply to move horizontally, and so getting a stable horizontal shot is not possible. A gimbal is used to keep a camera or sensor horizontal when its mount (e.g. aircraft) is moving. The gimbal has three motors controlling rotation in orthogonal axes. The gimbal feeds gyroscope information back to the motor controllers to compensate for rotational movement of the mount.

In addition to stabilization, the three motors can be used to control the direction the camera is pointing, and can be used to smoothly track a target, or pan a shot. The three axes of rotation are referred to as Pitch, Roll and Yaw, and the gimbal orientation is referred to as its attitude. Explanations of these axes can be found in the [Flight Control Concepts](https://developer.dji.com/mobile-sdk/documentation/introduction/flightController_concepts.html) section of the Mobile SDK.

Gimbals have mechanical limits (or stops) to their rotation around each axis. When a sensor is mounted on a gimbal, many data and control lines are required to go from mount to sensor. These control lines are usually bundled in a cable assembly or flex circuit, both of which will limit the available rotation of the gimbal. Additionally, gimbals will also limit rotation so cameras cannot see landing gear or the product itself.

**Moving the Gimbal**

* Move to an angle over a duration
* Move at a speed in a direction

When using angle mode to rotate the gimbal's pitch, roll and yaw, the rotation angle of the gimbal can be defined as either Absolute(relative to the aircraft heading), or Relative (relative to its current angle). When using speed to rotate the gimbal's pitch, roll and yaw, the direction can either be set to clockwise or counter-clockwise. The gimbal can be reset by setting its pitch, roll and yaw to 0 degrees. The reset position is pointing horizontally and being in the same direction as product heading. Gimbals will be automatically calibrated on power up. During calibration, the product should be stationary (not flying, or being held) and horizontal. For gimbal with adjustable payloads, the payload should be present and balanced before doing a calibration. At the moment, there is no option to calibrate the Gimbal through Onboard SDK APIs.

Implementation of this example is in LinuxCamera.cpp can be great reference for any Camera and Gimbal controls supported by DJI Onboard SDK.

To try out the sample in interactive mode:

* Press j to set Gimbal rotation angle (input range is 360 degrees)
* Press k to set Gimbal rotation speed
* Press l to perform still image capture
* Press m to perform video capture (5 seconds of video recording)

Sample output for setting Gimbal rotation angle:

|  |
| --- |
| Gimbal Angles Description [roll, pitch, yaw]:  Roll angle: unit 0.1º, input range [-350,350]  Pitch angle: unit 0.1º, input range [-900,300]  Yaw angle: unit 0.1º, input range [-3200,3200]  (NOTE: Yaw output rotation angle represented in [-π, π] range (see simulator output))  Waiting for Gimbal to sync rotation angle...  Initial Gimbal rotation angle: [0, 0, -0.3]  Setting new Gimbal rotation angle to [0,20,200] using incremental control:  New Gimbal rotation angle is [0 19 -160 ], with precision error: [0 1 0 ]  Setting new Gimbal rotation angle to [0,-50,-200] using absolute control:  New Gimbal rotation angle is [0 -50 159 ], with precision error: [0 0 1 ]  Setting new Gimbal rotation angle to [25,0,150] using absolute control:  New Gimbal rotation angle is [24 0 149 ], with precision error: [1 0 1 ]  Setting new Gimbal rotation angle to [5,0,100] using incremental control:  New Gimbal rotation angle is [29 0 -110 ], with precision error: [0 0 1 ] |

NOTE:

1. Please make sure you have SD card inserted into the Gimbal

Known Issues:

1. If set Gimbal rotation angle without simulator, initial Yaw Axis will report 120 however, it will not affect accuracy. For questions please refer to our forum.