Group 3 Connecting to the Drone:

This project was interesting, because as a group many of us have never explored this area in using a program to control a drone with a prewritten script. As you examine the code you will find how it was very apparent that steps were taken into place to help ensure the safety of students and the drone were accounted for. The reason is, we would like this project to continue well past this class so that other students in the future can have their starting steps in controlling machines via this project.

This document will be a timeline from understanding, writing code, and coming together as a team to write this document.

Before the code can work, you need the right Python environment and the correct library installed. The DJI Tello drone does not natively speak Python; it communicates over Wi-Fi using UDP(User Datagram Protocol) commands. The [2]`djitellopy` library is a community-maintained wrapper that translates simple Python commands like `me.takeoff()` into the low-level SDK(Software Development Kit) instructions that the drone understands.

The [1]DJI Tello drone is a small, consumer-friendly quadcopter that offers students, hobbyists, and educators an accessible platform for learning the fundamentals of robotics and programming.

The script begins with two import statements. The first imports the `tello` class from the `djitellopy` library, which wraps the Tello SDK in the Python interface. This allowed group 3 to call methods such as `takeoff()`, `land()`, and `move_up()`, rather than manually sending text strings over a socket.

The second import, `time`, brings in Python's built-in time module. This library is crucial because drone commands need pauses between them. The function `time.sleep(seconds)` is needed because without it commands can collide. For example, If you fire the next command immediately, the drone is still mid-move. Tello will often reply "error" or silently drop the new command because it's busy.

Before the drone even takes flight we mentioned how safety is paramount when flying drones. This code prompts the operator to ensure that a safe flight space (about two cubic meters) is available. It runs inside a while True loop that repeats until the user enters a valid response.

The `main()` function contains the flight logic. First, an instance of the Tello class is created with `me = tello.Tello()`. At this point, the object exists in Python but is not yet linked to the drone. The call to `me.connect()` sends the initial "command" message over UDP to the drone's default IP address (192.168.10.1). If the connection succeeds, the drone begins responding with telemetry packets.

Before takeoff, the program checks the battery. The method `me.get_battery()` asks the drone for its current percentage. If the value is less than 15%, the script aborts for safety reasons. A drone with too little charge could crash mid-flight. If the battery query itself fails, the script prints a warning but continues cautiously.

The drone's movement speed is set to 15 centimeters per second, which is a slow and controlled pace suitable for indoor testing. The `takeoff()` command launches the drone and causes it to rise automatically to about 80 centimeters. A pause of two seconds allows it to stabilize before being given another command to operate off of.

The bulk of the script is a pre-planned flight path. Each movement is in centimeters: `move_up(50)` raises the drone by half a meter, while `move_left(100)` shifts it one meter left. After each major movement, the `getInfo` function is called to print telemetry. Short sleep commands give the drone time to complete its actions before new commands are sent.

Next within the code we have a helper function called `getInfo()` that will print telemetry data at various stages of the flight. It will tell you how many times it has been displayed, the height of the drone, as well how long it has been in flight.

This works via queries to the drone for telemetry. The command `me.get_height()` requests the altitude in centimeters, as measured by the downward-facing sensor. The command `me.get_flight_time()` asks how long the drone has been airborne. Each request is wrapped in a try/except block. This is good defensive programming: if telemetry cannot be retrieved (for example, after landing or during a poor Wi-Fi connection), the script does not crash. Instead, it prints that the data is "unavailable." Finally, the function prints the stop number, current height, and flight time.

The landing sequence is straightforward. A message is printed to the console, the program waits two seconds, and then `me.land()` is called. This causes the drone to descend and cut its motors once it touches the ground. Another `getinfo()` call is made to capture data right after landing.

While looking though the script you can see how all the flight logic is wrapped in a `try/except` block. The reason is that if an unexpected error occurs (for example, loss of Wi-Fi connection or a malformed command), the script tries to land the drone safely. If that fails, it issues an emergency motor cutoff (me.emergency()), which immediately stops the propellers.

Our Closing Thoughts:

The program we developed is more than just a demonstration of how to control a drone with Python. It represents the intersection of software and hardware which we have been reading about in chapters 1 -3 so far.

Writing simple lines of code can directly influence the motion of a real-world machine. Every movement, from takeoff to landing, illustrates the importance of planning, safety, and communication between systems.

The Tello drone is a training ground for bigger ideas: autonomous delivery drones, coordinated swarms for search and rescue, and intelligent aerial platforms that can map, monitor, and interact with our environment.

By starting with structured sequences of movements, we gain confidence in controlling the drone safely. From there, the possibilities expand toward integrating computer vision, machine learning, and collaborative robotics. This project is a great example of seeing theory and concepts in a physical way.

Resources:
[1] https://www.ryzerobotics.com/tello-edu
[2] https://github.com/hanyazou/TelloPy

https://docs.ros.org/en/kinetic/api/tello_driver/html/classtellopy_1_1__internal_1_1tello_1_1Tello.html
https://alfredo-reyes-montero.gitbook.io/tello-dji/frameworks/python/tellopy