Lab 5b: DJI Tello control

1 Objectives

The goal of this week's lab is to apply our newly gained control knowledge to hardware. Specifically, you will design a yaw controller for the DJI Tello quadrotor as depicted in Figure 1. To do so, you will need to first identify a model to represent the system. Unfortunately, DJI does not disclose its proprietary information about the model nor the controller it uses. Hence, this lab approaches the Tello as a black box. In this lab, you will probe the yaw command input at different frequencies to determine the frequency response of the yaw dynamics of the system. That information will then be used to design a controller to track a reference yaw angle. This controller will be tested and compared with DJI's controller to reject disturbances and track reference signals.

Figure 1: DJI Tello quadrotor. Source: Amazon.com

2 Equipment and Safety

2.1 Lab Rules

Please read and follow the rules of this lab:

- All lab and project experiments are to be done indoors.
- Wear safety glasses when flying drone.
- Do not try to grab the Tello drone in flight with bare hands.
- Keep children, pets, and others away from the experiment.
- Assume controllers can become unstable, leading to unexpected or dangerous behavior.
- The Tello Drone is not a toy - personal injury can result from contact with spinning blades. Keep prop guards on. Do not touch propellers when powered on.
- Choose an indoor area without ceiling obstacles, and away from fragile objects (china etc.).
- Follow safety guidelines as listed on the Ryze Robotics Website: in particular sections Propulsion

System and Flight Batteries: `https://dl-cdn.ryzerobotics.com/downloads/Tello/20180211/Tello+Disclaimer+and+Safety+Guidelines+(EN)+v1.0.pdf`

- Make sure operation area is well-lit (flight control relies on visual feedback).
- Power down drone when not in use (it can overheat in less than 1 minute).
- Use only approved LiPo battery charger. Do not charge unattended.
- If you live in University property, please be aware of UCB campus drone policy: `https://campuspol.berkeley.edu/policies/drone(campus).pdf`

2.2 Lab Equipment

This lab will use two main pieces of hardware: your laptop and your Tello. Your laptop will communicate with your Tello over wifi, so when running experiments you will not be able to access the internet unless you have a second wifi/network card. The python script will be running two threads. The first thread will be to listening to the Tello on UDP port 8890 for state data from the Tello's sensors. The other thread will be running your controller and sending commands to your Tello on UDP port 8889.

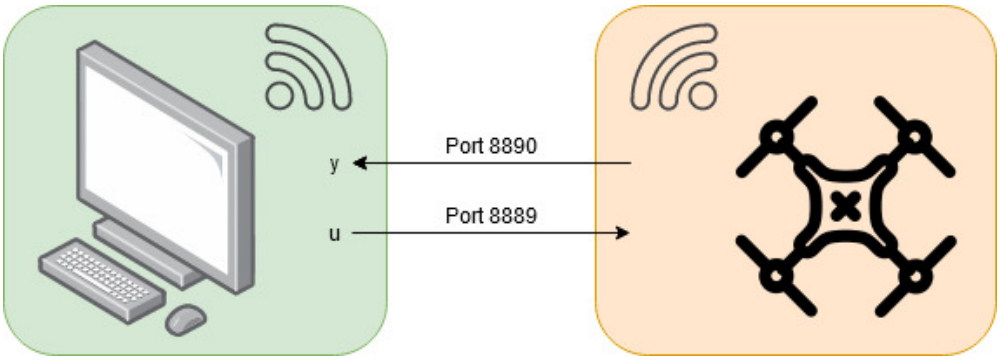

Figure 2: A system diagram how your laptop and your Tello communicate.

2.2.1 Common Errors

This is the first trial of this lab, errors are expected to arise. For this reason, there is a piazza post labeled 'Tello Errors'; please post all errors, problems, or challenges you are facing related to the hardware. The teaching staff will try to resolve the problems in a timely manner. Also, if you are a guru with Python or any other topics related to this lab, the teaching staff will greatly appreciate tips and potential remedies to problems posted on this piazza post. The teaching staff will award you imaginary extra credit points that wont count towards your grade (only redeemable for boasting purposes).

Here are some common errors that the teaching staff have encountered setting up the lab:

- When Tello battery is low, the script will run. However, upon receiving the `takeoff` command, the system will return an error and just sit there.
- When tests are run with too long of a waiting period between tests, the Tello may shut itself off. You will just have to reset your Tello along with your connection to it.
- When your Tello's sensors are not getting enough light, the Tello will tend to drift and might send out error messages that there is 'no IMU detected'. To fix this, try to provide more light for the Tello in your test environment.

3 Theory

3.1 Inner Control

In lab 5a, you were tasked with coming up with a control for a Parrot Mambo quadrotor. You had direct access to the model, the sensors, and the motors. However, DJI only gives you limited access to some sensors. This means you cannot develop a low level controller for the Tello like you did for the Mambo. The Tello will already have some sort of controller feeding the motors commands based on what control engineers at DJI decided on. For this reason, the Tello will be considered a black box, where you can only read sensor outputs and set certain references to the controllers designed by DJI's engineers.

3.2 Outer Control

For this lab, you will wrap a controller around the system provided by DJI. Your controller will compute the necessary inputs to the Tello so the yaw angle tracks a triangle wave reference.

First, you will focus on modelling your system. Your model will consist of a frequency response plot, aka a bode plot. To model the black box system, you will input different frequencies into the Tello yaw control command and note the response of the yaw signal received from the Tello. From the outputs you will be able to compute the gain and phase for that specific frequency. To compute the gain, you can look at FFT of the output signal or you can take the ratio of the output signal's amplitude over the input signal's amplitude. To compute the phase, you can measure the time it takes the output signal to reach a peak from the peak of the input signal. This time has to be multiplied by the frequency to get the phase shift (watch out for units!).

This frequency response will give you some understanding of the system that DJI did not disclose to you. As you learned in class, you will be able to compute the gain and phase margins to infer more information about the Tello. Using this information you can now design various controllers to make your Tello better at tracking a specific yaw angle.

4 Pre-Lab

4.1 Installing Python

This lab will be using Python 3 to interact with the Tello. If you already have Python 2, please download Python 3. Additionally, when running Python scripts in the lab, make sure you specify that you want to run Python 3 by using the command `python3` instead of just `python`. More on running Python code in section 4.3.

4.1.1 Installing on Windows

Go to `https://www.python.org/downloads/`. Click the button that says 'Download Python 3.x.x'. Once downloaded, you should see a new file in your downloads: 'python-3.....exe'. The '.exe' means the file is an executable. Run that file by double clicking on it. On the first page of the installation wizard, be sure to check add Python to your PATH. This allows you to run Python from any folder on your computer. Use the recommended/default options for the remainder of the installation.

4.1.2 Installing on MAC

First, you should check that if you have Python 3 already installed. MAC will have Python 2.x installed. You can check if Python 3 is installed by opening your terminal and typing `python --version`. If you see a version of the form 3.x, then you are good to go. If you see a version of the form 2.x, then you

may want to check `python3 --version`. If you don't have Python 3, the following alternatives can be used:

- Similar to the Windows installation you can go to `https://www.python.org/downloads/` and download Python 3.x.
- An easy way to install Python will be to install Anaconda: `https://www.anaconda.com/products/individual` You can use the Graphical Installer provided there. After installation you can check again your versions by using the mentioned commands in your terminal. Anaconda provides numpy and additional packages.
- A more advanced method will be using Homebrew. If you already have Homebrew installed, you can use `brew install python` and you will be good to go. You can follow more detailed instructions here: `https://docs.python-guide.org/starting/install3/osx/`

4.1.3 Installing on Ubuntu

You should be able to install Python on recent versions of Ubuntu by typing in your terminal:

```
sudo apt-get update
sudo apt-get install python3.8
```

The following guide should cover you for other Linux distributions: `https://docs.python-guide.org/starting/install3/linux/`

4.1.4 Install numpy

The Python scripts used in this lab will also use a library called `numpy`. Install this package by opening your command window (Windows) or terminal (MAC or Linux) and entering the command `pip install numpy`. If successful, it should return that you either already have `numpy` or `numpy` has been successfully installed. You may get some warnings, but still have a successful installation. However, if there are any errors, this means `numpy` was not installed. If this is the case for you please go to the 'Tello Errors' piazza post for more directions.

4.2 Updating your Tello

DJI sells the Tello all around the world. In order to make purchasing a Tello easy, stores around the world keep a stock that may be replenished at different times. This means that some Tellos have been made earlier and therefore had an earlier version of software uploaded. To minimize discrepancies, please update your Tello to the most recent software version. This can be done by connecting to your Tello using your smartphone with DJI's Tello App. Please download the app using the app store or the playstore (don't worry it is free). Make sure your airspace is free and clear then follow the app's instructions to do a short flight. If your Tello's software is not the latest version, the app will prompt you to update the Tello's software. Follow the instructions in the app to update your Tello.

4.3 Using Python to control your Tello

For the lab, there will be a set of steps to run each test. Go through these steps so that you will be set for the lab:

1. Place your Tello on the ground and turn it on. It is important for the Tello to be stationary during startup because it uses the startup time to calibrate its gyros. Moving it during startup can cause

the Tello to drift while flying. Also, note which direction you turn the Tello on, this direction will be the 0° direction for the yaw angle.

2. Connect to the Tello wifi hotspot. This will disconnect your device from the internet (unless you have a second wifi card on your device), so make sure you have everything needed already downloaded.
3. Open your computer's command prompt (for Windows) or terminal (for MAC and Linux), and navigate to the folder/directory that you downloaded the lab files into using the `cd` command. For example, if the files are in your `Lab5b` folder, that is in your `ME134` folder, that is in your `Documents` folder you would enter: `cd Documents/ME134/Lab5b`.
4. Run the experiment. Enter the command `python TelloPrelabFlight.py`. You should get the following output:

```
GSI_Bob/Documents/ME134/Lab5b>python TelloPrelabFlight.py
Started rcsvstate thread
Type in a Tello SDK command and press the enter key. Enter "quit" to exit ...
    this program.
Received message: ok
```

   Now, make sure your airspace is free and clear, then enter the command `takeoff`. You should see your Tello take to the skies to perform the tasks encoded in `TelloPrelabFlight.py`. Once the Tello lands, enter the command `quit` to stop and cleanly close the program. If you received an error at any point, restart this process from step 1. There is a piazza post, "Tello Errors", that will let you post errors you have encountered, so others who might have already resolved the issues can help you through.
5. Retrieve the data. You should have noticed that a new file, statedata.txt, has appeared in your directory. This file contains the data in a csv format from your test. Rename the to something that will make sense when referring to it (i.e. prelab5bTest.txt) or else you risk overwriting your data.
6. Import the data into Matlab. Use the `testDataDecoder.m` file provided to you to load your flight data into your Matlab workspace. The function takes in the filename as a string: `testDataDecoder("prelab5bTest.txt");`

Plot the results of the flight. Plot the altitude, velocities, and rotation angles (yaw, pitch, and roll) all with respect to time. Make sure to plot only plot signals with the same units on the same plots.

5 Lab

5.1 Frequency Response

The first part of the lab is to identify the model of the system. Unfortunately, DJI did not provide a model of the Tello with its controller. For this reason, the Tello will be approached as a black box, so you have access to the inputs and outputs but nothing in between. As you learned in class, a way to get to know the system is by inputting signals at various frequencies and noting the gain and the phase shift of the outputs. The frequencies that you test can provide you points to visualize the bode plot of the system. This bode plot will represent your system identification.

The focus of this lab will be on identifying and controlling the yaw dynamics in this lab. Choose 5

equally spaced (on log scale) input frequencies from 0.1 rad/s to 10 rad/s to simulate. Using the file `TelloOpenLoop.py` and *only* change line 186 to reflect one of the frequencies that you chose. Keep in mind that Python will assume numbers without a decimal point are integer types, so if you chose a frequency of 2 rad/s make sure you input `2.0`, not `2`.

Plot the input signal and the yaw signal versus time for each frequency on separate plots. Compute the gain of the amplitudes from the input to the output as well as the phase shifts for each input frequency. Hint: Try using FFT's to compute the gains.[1] Plot these values on a `loglog` plot for magnitude and a `semilogx` plot for phase shift in one figure (use subplots).

Looking at the bode plot, what is the number of poles? Number of zeros? Is phase response consistent with a system with just poles and zeros? Hint: check if there is a delay in the system.

5.2 Proportional Control

Now you will implement a proportional controller to track a triangle wave. First, chose a proportional gain such that you have a sufficient gain and phase margin. Report the proportional gain constant, the gain margin, and the phase margin.

The reference yaw signal that the Tello will be trying to track is a triangle wave that has period of length 20 seconds and an amplitude of 180°. In `TelloClosedLoop.py`, code the reference signal in line 193, 194, and 205. The variable `ptime` is the current time in seconds of the experiment. Use the variable name `reference` for the yaw reference signal. If you are new to Python, look in the appendix (section 6) for useful commands. Now code your proportional controller in lines 186, 209, and 210. The yaw signal is labeled with the variable `yaw`. Your code should resemble the following:

```
1 kp = 1000000.0 # please don't use this value, it won't work well
2 while (experimentIsRunning):
3
4     # Some stuff
5     reference = # your code computing the triangle wave that uses ptime
6
7     control_YA = kp*error
8     # Other stuff
```

Plot the yaw control input and the reference signal superimposed with the yaw angle (1 figure, 2 subplots). Comment on how well the proportional controller tracks the triangle wave.

5.3 Better Triangular Wave Tracking

The Python script operates in discrete time. This means that the input commands are executed a steady (*ideally*) time step. Implementing a proportional controller for a discrete time system is the same as a continuous time system as we saw in section 5.2. To implement an integrator, you must keep track of the integrated error. This can be done by using a Reimann sum where the width of the rectangles is the time step:

```
1 integratedError = 0.0
```

[1] Don't forget to clip your data first. Using a FFT on the signal *only* when the sinusoidal input is being applied.

```
while (experimentIsRunning):

    # Some stuff
    error = reference - yaw
    integratedError = integratedError + timeStep*error

    control_YA = kp*error + ki*integratedError
    # Other stuff
```

To implement a differentiator, you only have to keep track of the current error as well as the previous error, then you can just take the slope to estimate the derivative of the signal:

```
errorStore = 0.0
while (experimentIsRunning):

    # Some stuff
    error = reference - yaw
    errorDerivative = (error - errorStore) / timeStep
    errorStore = error

    control_YA = kp*error + kd*errorDerivative
    # Other stuff
```

5.3.1 Design a Better Controller

Using the information provided on implementing a integral and derivative control, replace the proportional controller from section 5.2 in `TelloClosedLoop.py` with a controller that will decrease the errors when tracking the triangle wave. Hint: What *Type* does the system have to be to track a triangle wave with no error. Remember that a triangle wave is just a bunch of ramps.

Test your new controller. Were you able to get 0 steady state error? Plot the yaw control input and the reference signal superimposed with the yaw angle (1 figure, 2 subplots).

5.3.2 Check DJI's Controller

Now copy and paste your reference signal trajectory code into `TelloDJIController.py`, lines 186, 187, and 198 and run the file. Plot the same signals that you did in section 5.3.1. How do the two compare? Are you starting on your DJI control engineer application yet?

5.4 Lab 5a Comparisons

We will now check what happens when a significant disturbance is applied to the system. Update the `TelloClosedLoopDist.py` with your controller and your reference signal. Plot the yaw control input, the up/down control input, and the reference signal superimposed with the yaw angle (1 figure, 3 stacked subplots). Was your controller able to perform as well as it did in section 5.3.1? If not, what kept it from performing as well?

5.5 Bonus: Preparation for Mini Project

The mini project will be a student-designed project that can potentially use the camera as a sensor to get state information. This optional part of the lab is for you to be able to get a head start for your mini project. These projects may range from tracking an object, to using the camera as a room

positioning system. For some guidance, you can follow the Python 2 example found at `https://github.com/dji-sdk/Tello-Python` to retrieve a picture using Python. If you get the Tello's camera to show an image over Python, show it off and include the picture in your report!

6 Appendix: Coding in Python

Here are some coding in Python tips that are different to Matlab:

- **Spacing/tabs:** Python, unlike Matlab, does not have keywords to end if statements, while loops, for loops, etc. Python instead keeps track of these statements by having them use proper indentation. Make sure your code is all lined up and uses the same characters for the indentation (stick to either tabs or spaces but don't interchange them).
- **if statements:** To code an if statement, use the following setup:

```
1    if (condition):
2        # Do something
3    else:
4        # Do a different thing
5
6    # other code
```

  Note the indentation for the if statement. Additionally the 'not' operator in Matlab is `~`, but in Python it is `!`.
- **Modulo operator:** The modulo operator is a helpful tool to find the remainder of a quantity when divided by another quantity. For example, `14.5 % 4.0` (read 14.5 mod 4) is equal to `2.5`. This can be helpful to code the triangle wave by only having to code a single period of the wave.
- **Indexing array/vectors:** Python starts indexing at `0`, unlike Matlab's first index, `1`. To reference an element in an array (vector), `myArray`, at index `i`, you can call `myArray[i]`.
- **Trigonometric functions:** To utilize trigonometric functions, you have to use the `numpy` library by calling `np.sin(input)` for sine and `np.cos(input)` for cosine. The inputs in both cases are in radians.