ME 586: Biology-Inspired Robotics

University of Washington, Prof. Sawyer B. Fuller

Problem Set 5*(revised 2023.03.02)*

Goals: Create a biology-inspired altitude estimation system on a robotic drone using closed-loop instability, estimate altitude using data analysis

Required reading:

- Guido C. H. E. de Croon, "Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy," *Bioinspiration & Biomimetics* (2016). (link)

You will implement a honeybee-inspired altitude estimation system on the Crazyflie inspired by the article above. Your system will operate in a feedback-controlled loop that responds to optic flow. Such a feedback loop has an instability that is the result of a delay in the feedback system due to its discrete nature (being executed on a computer) and a gain that depends on its altitude because of the distance-dependence of optic flow. You will find a function that estimates your altitude based on the gain at which the system goes unstable. In principle, this could be used for a drone (or animal) that does not carry a laser rangefinder.

Instead of moving vertically as in the article above, you will command your Crazyflie to make lateral oscillations. This is because the optic flow sensor on the Crazyflie cannot measure "divergence", a measure of the rate of expansion that is an indicator of vertical velocity. The x-component of the optic flow observed by a camera pointed in the negative body z-direction is given by

$$\Omega'_x = \omega'_y - \frac{v'_x}{r} \text{ [rad/s]},$$

where v'_x is the robot's lateral velocity given body coordinates and r is the distance to the ground.

To move laterally, the Crazyflie must tilt itself, but we are only interested in the component of optic flow induced by translation. We will use the gyroscope onboard the Crazyflie, which provides a measurement of its angular velocity ω'_{ym}, to subtract out rotation-induced optic flow, leaving the translation-induced optic flow:

$$\Omega'_{xTm} = \Omega'_{xm} - \omega'_{ym}.$$

Now suppose our feedback model simply specifies a desired velocity that is a gain times the measured translational optic flow:

$$v_{xd} = K\Omega'_{xTm}, \tag{1}$$

where K a proportional feedback gain. Our computer operates in discrete-time, updating the desired horizontal velocity intermittently. We can describe our system's dynamic behavior by combining the three equations above into the following discrete-time equation:

$$v_{k+1} = -\frac{K}{r}v_k. \tag{2}$$

Note that we have omitted subscripts here for simplicity.

1. Analyze the stability of Equation (2). Solving a discrete-time equation like this entails assuming a solution of a certain form and then solving for it, much like for continuous-time differential equations. For a discrete-time linear system (assuming r is constant), that assumption is that $v_k = v_0 a^k$ for $k = 0, 1, 2, ...$, where v_0 is the initial velocity of the system at $k = 0$.
   (a) Solve for a.
   (b) For what range of values of a is the system stable? (Hint: it is not that a is in the left-half plane).

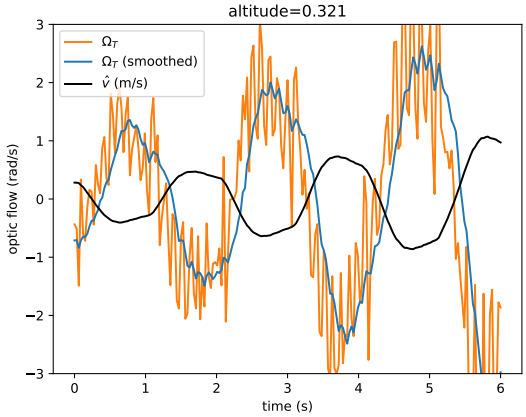

Figure 1: Example oscillations plot

2. Next, implement this control system on the Crazyflie. Skeleton code has been provided on the public course website (link), `crazyflie_oscillate_class.py`, that automatically prints the gyroscope readings ω'_{ym}, pure optic flow measurements Ω'_{xm}, and Ω'_{xTm} as `pm`, `ofxm`, and `tofxm`, respectively. Noise in the optic flow measurement makes the `tofxm` measurements very noisy, so the code also implements `tofxm_smoothed` that is a "moving average" of the last few measurements. The code also starts by having the Crazyflie perform a forward maneuver to excite its translational dynamics, and then concludes by reversing to its approximate original location.

   Implement in your code the translational feedback system given in Equation (1). Repeatedly perform motion commands using `mc.start_forward()`, where the argument you provide is given by that equation. Between velocity command, wait for a time interval of 1 s by calling `time.sleep(1)`. Continue this for about 6 seconds using a while loop (`while time.time() - t0 < 6`), and then end the program.

3. Now assess the performance of your physical system by finding approximate mapping between gain and stability. To so, you will experimentally find the gain K at which your system goes unstable at each of a small number of different altitudes, and then find a linear fit. The idea is that you will be providing the information that an estimator would need–in the form of a relation between gain and altitude–to estimate altitude.

   (a) To facilitate analysis, it is suggested that you modify your Crazyflie control code to log an additional variable that indicates when your actual experiment is running. You will later use this to "mask" your data, leaving out data that you are not interested in such as during takeoff and landing (see postscript below, "array masks"). Add an additional global variable with a descriptive name, such as "`go`", that will be an integer that is 0 initially and 1 while your experiment is running. Modify the `log_callback` function to print out its value along with the other variables it is already printing out. Then, in the feedback control part of your code, make sure to set that variable to 1 when you start executing the controller, and back to 0 when it has concluded operating. In your analysis code where you load in your data, `go` will appear as an integer when you load it from your text file. Convert it to a boolean array in your analysis code for use as a mask using `go = go.astype(bool)`.

   (b) Now run the experiment on the Crazyflie and collect data using a caret ("`>`") to send the data to a file with a descriptive name such as `data_gainX_altitudeY.csv` (replacing `X` and `Y` with the values you used in that experiment), as we did in Problem Set 3. To find the gain at which there is instability onset at each altitude, you will have to do a search: perform multiple flights in which you gradually increase the gain until the system goes unstable.

It is suggested that you choose the following three altitudes: 0.3, 0.5, and 1.0 m. For each altitude, you will repeatedly fly the helicopter to collect data and then run a separate analysis script that plots the results, as you did in preparation for this exercise in Problem Set 3. An example of how your data might look is shown in Figure 1. One way to determine the gain at which your system goes unstable is to compare the amplitudes during the first oscillation to the amplitude at the last oscillation. This can be done by computing the peak of `tofxm_smoothed` during the first two seconds, using `max(abs(masked_data))`, and comparing it to the peak during the last oscillation. If the change is less than about 25%, then you are very close to onset of stability. **See the postscript below about masking data.**

(c) Create a plot of gain K along the x axis vs. altitude at which instability occurs on the y axis. You should see in your plot that as the altitude increases, the gain of instability onset increases. Use a least-squares fit to find the trend line and provide the coefficients.

Submit: An example plot showing oscillations, your least squares fit plot showing coefficients, your control and analysis scripts, and an answer to the question: does your trend line match what was predicted in Problem 1?

Extra Credit 1 (+2pts): In principle, your robot could use this information to move to a desired altitude by setting the gain corresponding to the desired altitude and slowly descending until instability is detected as a growing oscillation in `tofxm_smoothed`. Implement this behavior in your Crazyflie, that is, devise a controller that descends until it reaches an altitude you specify.

Extra Credit 2 (+2pts): We took a shortcut above that is not entirely realistic. Our outer loop system sends velocity commands, must rely on a velocity estimate that is not observable without a laser rangefinder (you can show this using the observability criterion). A more realistic system would command motions that *can* be observed without a rangefinder. Inclination, which can be observed by using the onboard accelerometer as an inclinometer, is one possibility. Derive the discrete-time dynamics of such an outer loop system, which would use a feedback law of the form $\theta_d = K\Omega_T$. Show that it goes unstable above a certain K and Δt. Assume the inner loop attains the desired attitude θ_d instantaneously and that thrust equals gravity.

Postscript: array masks Array masks are a convenient way to pick out values from an array of data that occur during a specific time interval. They tend to be more convenient than slices when your data has time stamps. An array mask is a numpy array of boolean values (`True` or `False`) that have the same number of elements as the array it is masking. The subset of the values that correspond to when the mask is `True` is returned. Example:

```
values = np.array((1, 2, 3, 4, 5, 6))
mask = np.array((True, True, True, False, False, False))
values[mask] # np.array((1, 2, 3))
```

Masks can quickly be created by performing boolean tests element-wise on the array:

```
mask = values > 4 # np.array((False, False, False, False, True, True))
values[mask] # np.array((5, 6))
```

Masks can be combined using boolean operators & (AND) and | (OR):

```
values[(values > 1) & (values < 5)] # np.array((2, 3, 4))
```

We can use this to access values from our data that occur at a specific desired time, for example: `tofxm_smoothed[(t > 1) & (t < 3)]`.