Quadrotor Parameter Identification Code

Marc Rigter, marcrigter@gmail.com

1 Summary

This Python/numpy code can be used to estimate parameters for a quadrotor with flight data including measurements of: propeller RPM, position, velocity, angular position, and angular velocity. A non-linear least squares batch estimation is used, based heavily on this paper [2]. This document presents brief details of the model estimated, and usage of the code. If more detailed information is required, feel free to contact me (email above) - I will be writing a lengthy report on this work in January 2018 (I have not had much time to prepare this document, so it is brief, and I'm sure it has omissions).

2 Parameters

The following model parameters, however the code could easily be adapted for a different model. Everything is defined in North East Down (NED) coordinates. All units SI.

- m - mass in kilograms.
- I_{xx}, I_{yy}, I_{zz} - diagonal inertia tensor. Sensible for a quadrotor due to near symmetry.
- ΔDx - offset from geometric centre to CG in x direction.
- ΔDy - offset from geometric centre to CG in y direction.
- Dx, Dy - distance from geometric centre to propeller hubs
- C_q - torque coefficient.
- k_1, k_2 - thrust model. See this paper for details [3]. If k2 is set to 0 and only k1 is estimated, this becomes the usual static thrust coefficient model.
- k_4 - this it the equivalent propeller disc area. I'm using 80% of the measured propeller disc area to account for tip losses.
- C_{dh} - coefficient for hub drag on propellers [1].
- C_{dxy} - in-body plane airframe drag coefficient.
- C_{dz} - perpendicular to body plane airframe drag coefficient.
- R_0, R_1, R_2, R_3 - quaternion describing rotation between body frame and pose estimates.
- $tangoDx, tangoDy, tangoDz$ - vector offset fro geometric centre to pose estimates.

3 Methodology

Maximum likelihood estimation. Inputs are the pose estimates and RPMs. The parameter vector θ, containing the parameters as above are estimated, as well as the state at each time step. Because it is a batch estimation for all of the states at once it runs slowly. Because the matrices are large and I have used numpy matrices, the memory requirements can be very large for large data files which is a limitation.

At each iteration Jacobians are calculated to update parameters and state estimates to increase likelihood. Gauss Newton optimization is used to make the updates. Should converge within 10 iterations.

4 Main Functions

- main - are the files to run.
- FlightData - reads CSV data and applies low pass filters (forwards and backwards to eliminate phase offset) to raw data. To generate the CSVs from .ulg files, I am using ulog2csv from pyulog.
- Estimator - contains all of the code for estimating gradients and making updates. This is where most of the code is.
- Dynamics - is the dynamics model.
- Plot - includes plotting functions. plotResiduals to assess the residuals to see how well the model fits. plotResults to see the optimization over the iterations. plotData for the input data.

5 User Input

All of this is in the files starting with “main”. I have everything set up in such a way that all of the files starting with “main” can just be run as they are, but this section details user inputs which could be changed.

- directories - list of directories for files to use. If you include more than one loops through all of them.
- startRows - list of the rows of the CSV files to start using the data from. You want to remove data where the drone is on the ground, in ground effect, or isn’t actually doing maneuvres which activate the appropriate dynamics.
- nRows - the number of rows of data after startRows to use
- attCutoffHz, rpmCutoffHz - frequencies in hz for low pass filtering raw data from your CSVs.
- tol, breakEarly - if breakEarly is true, and the cost decreases by less than tol, the optimization stops.
- maxIter - if maxIter is reached the optimization stops.
- rpmStep - use every nth RPM from the data. The only reason to increase this above 1 is to improve run time.

- n - use every nth position observation. Our position data is sampled at 200hz and I usually downsample by 4 to 50hz.
- step - multiplying factor on Gauss-newton update. If optimization unstable, decreasing this may help.
- method - options are "WLS" for weighted least squares, and "fastLTS" for fast least trimmed squares. fastLTS samples many random intialisations in an attempt to increase robustness (google fast least trimmed squares). However it is extremely slow (hours run time), so I would recommend sticking to WLS (1-4 min run time).
- Q - Gaussian white noise force and moments assumed (forcex, forcey, forcez, momentx, momenty, momentz). See the Burri paper.
- covSens - variance of pose estimates. (posx, posy, posz, velx, vely, velz, q0, q1, q2, q3, rollrate, pitchrate, yawrate)
- Theta - this is the main table containing the params. if Estimate is flagged to be true, this param will be attempted to be estimated from this file. It will be initiliased with the value in the value field. If Estimate is set to false, the value field remains constant. The stddev is an output at the end. Any user input in this field is not used.

6 Usage

A key point to note is that you can only estimate a parameter if the associated dynamics are observed (eg. you cannot estimate inertia if you never have angular acceleration). For my work, a large focus was on high velocity flight where aerodynamics are important. It was difficult to create a trajectory in our framework to activate all dynamics including the high speed aerodynamic effects. For this reason I have estimated parameters separately across three flights, however this process could certainly be modified. I detail below this process I have used to estimate separate parameters in three separate flights.

The parameters which must be set by the user to start with are the mass, and the distances from the centre of the quad to the propeller hubs, Dx, Dy, and the propeller disc area, k4.

Flight 1.

First of all, the propeller thrust model is estimated from vertical flights, using the file main_estimate_thrust. I have used vertical flights up to about 3ms for this. The data set is in the /data folder. Inertia can just be set to a sensible value, it doesn't make much difference as you are not rotating much in this flight. Same goes for coefficient of torque. Thrust model values need to be initialised on the right order of magnitude. The values I have put in this file should suffice. After running this, you should get values for k1 and k2, the thrust model. These are saved to a .yaml file in the results folder. There will also be a plot showing this. Copy these values across to main_estimate_drag Theta table (I have done this already).

Flight 2.

Next run main_estimate_drag to estimate drag parameters. For this, the flight data needs to contain flight at speeds up to at least 7m/s. If you are not flying this fast I would skip this step as you will probably not be observing much drag. If you do use this, copy across the drag parameters into the Theta table in main_estimate_inertia (I have already done this).

Flight 3.

Run main_estimate_inertia to estimate everything else. For this you need files containing large angular accelerations about all axes. I have used flights with consistently up to 6rad/s rotations about all axes for our 250mm drone. I suspect smaller rotation rates might be sufficient for larger platforms, but not sure. The final values written to the final .yaml in the results file are the resulting characterisation.

Note: CG is estimated in each of the 3 flights, as well as the sensor rotation as these are always observable.

In short: to run through the process I am using, you need to run: main_estimate_thrust, main_estimate_drag, main_estimate_inertia, copying across the thrust parameters from the first run, and the drag parameters from the second run, and the final parameters are in the third run (main_estimate_inertia).

References

[1] M. Bangura, R. Mahony, et al. Nonlinear dynamic modeling for high performance control of a quadrotor. In *Australasian conference on robotics and automation*, pages 1–10, 2012.

[2] M. Burri, J. Nikolic, H. Oleynikova, M. W. Achtelik, and R. Siegwart. Maximum likelihood parameter identification for mavs. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4297–4303. IEEE, 2016.

[3] C. Powers, D. Mellinger, A. Kushleyev, B. Kothmann, and V. Kumar. Influence of aerodynamics and proximity effects in quadrotor flight. In *Experimental robotics*, pages 289–302. Springer, 2013.