



第5章

航空機と同様の方法によるドローン飛行制御&実験

状態フィードバック制御の プログラム&飛行実験

藤原 大悟



写真1 航空機と同様の線形モデルによる飛行制御実験に挑戦!

本章ではSTEVAL-DRONE01の x , y 軸の角速度制御について、元のFCUソースコードにあるPID制御の代わりとして、状態フィードバック制御を題材に、ドローン用の角速度制御器を設計して実装し、飛行実験を行います(写真1)。

数式も多くなり、ややアカデミックな内容となりますが、MATLAB/Simulinkのツール・ボックスを有効活用しながら解説していきます。

モデルの線形化

● 航空機は線形モデルで制御系を設計する

前章で紹介したSTEVAL-DRONE01の数学モデル

は単純化しているものの、変数の積/べき乗などのいわゆる非線形な計算式がところどころにある非線形モデルとなっています。

一般に航空機の制御系設計では、トリム飛行状態を定め、非線形モデルをトリム飛行状態の近傍で線形化した線形モデルを作成し、線形モデルに基づいて制御系設計を行います。

線形モデルに基づく制御理論は、長年の制御技術の研究により十分体系化されていて、さまざまな対象に適用され実績が豊富で、数式の見通しが良く取り扱いが容易であることから、計算機技術が発達した現代においても線形制御が多く適用されています。

● 線形化のステップ1…状態変数の抽出

モデルの線形化に当たっては、まず系の状態を保持するために必要最小限の数の状態変数を抽出します。入力信号と出力信号を持つ、ある系について、ある時刻における出力値が、同じ時刻の入力値のみで定まる場合、状態変数はありません。このような系は静的な系と呼びます。

一方、出力値を定めるために同時刻の入力値だけでなく別の変数が必要となる場合、その変数を状態変数と呼びます。例えば、時間積分という系は一定値の入力に対して出力値が増加し続け、入力値のみでは出力値が定まらないため、状態変数を持ちます。このような系を動的な系と呼びます。

前章で説明したドローン・モデルの場合、状態変数は以下の16個があります。

- 推力/アクチュエータ・モデル1つにつき状態変数1個、計4個。(1次遅れ系の状態変数は1個)
- クォータニオン時間微分値の積分…状態変数3個 (クォータニオンは4成分あるが、ノルムが1という拘束があるため状態変数は1つ減って3つになる)
- 3軸並進速度の積分…1軸につき状態変数1個、計3個
- 3軸並進速度時間微分値の積分…1軸につき状態変数1個、計3個
- 3軸角加速度の積分…1軸につき状態変数1個、計3個

● 線形化のステップ2…状態空間モデルを作る

モデルへの入力ベクトルを u_{ss} 、モデルからの出力ベクトルを y_{ss} 、状態変数ベクトルを x_{ss} としたとき、次式で表される微分方程式であり線形モデルの数式表現の1つです。

$$\dot{x}_{ss} = A_{ss}x_{ss} + B_{ss}u_{ss} \dots\dots\dots (1)$$

$$y_{ss} = C_{ss}x_{ss} + D_{ss}u_{ss} \dots\dots\dots (2)$$

ただし、 u_{ss} 、 y_{ss} 、 x_{ss} はトリム値からの変化分とします。 $\dot{x}_{ss}$ は x_{ss} の各成分を時間微分したもの、 A_{ss} 、 B_{ss} 、 C_{ss} 、 D_{ss} は係数行列と呼ばれる行列です。式(1)、式(2)をそれぞれ状態方程式、出力方程式と呼びます。

式(1)、式(2)は、 x_{ss} や u_{ss} の各成分がわずかに変化したとき、 $\dot{x}_{ss}$ や y_{ss} の各成分がどの程度変化するかという対応関係を表しています。つまり、 A_{ss} は x_{ss} に関する $\dot{x}_{ss}$ の微係数、 B_{ss} は u_{ss} に関する $\dot{x}_{ss}$ の微係数、 C_{ss} は x_{ss} に関する y_{ss} の微係数です。

これらの微係数は、前章で紹介した非線形モデルをトリム値周りで解析的に微分して求めるか、あるいは、非線形モデルが複雑であれば数値微分して求めます。極力、解析的に求める方が望ましいです。なぜなら、数値微分で求めた場合に有意でない微小値(ノイ

ズ)が係数行列内に生じて、モデルの物理的意味が不明確になる場合があるためです。ただし、数値微分による方法は自動化が可能で手軽であるという利点もあります。

● Simulinkを使って線形モデルを求めた

MathWorks社 のMATLAB/Simulinkのツール・ボックス(アドオン製品)の1つであるSimulink Control Designを用いて数値微分により線形モデルを求めました。この結果を次式に示します。

$$x_{ss} = [x_{actt1} \ x_{actt2} \ x_{actt3} \ x_{actt4} \ x_{att1} \ x_{att2} \ x_{att3} \ p_{xN} \ p_{yN} \ p_{zN} \ v_{xB} \ v_{yB} \ v_{zB} \ \omega_x \ \omega_y \ \omega_z]^T \dots\dots\dots (3)$$

$$u_{ss} = [\Delta \delta_1 \ \Delta \delta_2 \ \Delta \delta_3 \ \Delta \delta_4]^T \dots\dots\dots (4)$$

$$y_{ss} = [\phi \ \theta \ \psi \ \omega_x \ \omega_y \ \omega_z \ Z_{total} \ L_a \ M_a \ N_a]^T \dots\dots\dots (5)$$

$$A_{ss} = \begin{bmatrix} -10 & 0 & 0 & 0 & 0 & 0 & 0 & & & & & & & & & & & & \\ 0 & -10 & 0 & 0 & \vdots & \vdots & \vdots & & & & & & & & & & & & \\ 0 & 0 & -10 & 0 & & & & & & & & & & & & & & \\ 0 & 0 & 0 & -10 & & & & & & & & & & & & & & \\ 0 & \cdots & \cdots & 0 & & & & & & & & & & & & & & \\ \vdots & & & \vdots & & & & & & & & & & & & & & \\ & & & & & & \vdots & \vdots & & & & & & & & & & \\ & & & & & & & 0 & 0 & & & & & & & & & \\ \vdots & & & & & & \vdots & 0 & -19.6 & & & & & & & & & \\ 0 & \cdots & \cdots & 0 & -19.6 & 0 & & & & & & & & & & & & \\ 0.0262 & 0.0262 & 0.0262 & 0.0262 & 0 & 0 & & & & & & & & & & & & \\ -0.97 & 0.97 & 0.97 & -0.97 & \vdots & \vdots & & & & & & & & & & & & \\ -0.97 & -0.97 & 0.97 & 0.97 & \vdots & \vdots & \vdots & & & & & & & & & & & \\ 0 & \cdots & \cdots & 0 & 0 & 0 & 0 & 0 & & & & & & & & & & \\ \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & & & & & & & & & \\ & \vdots & \vdots & & \vdots & 0 & 0 & 0 & & & & & & & & & & \\ & & & & & 0 & 0 & 0 & & & & & & & & & & \\ & & & & & & 0.5 & 0 & 0 & & & & & & & & & \\ & & & & & \vdots & \vdots & 0 & 0.5 & 0 & & & & & & & & \\ & & 0 & \cdots & 0 & 0 & 0 & 0 & 0.5 & & & & & & & & & \\ & & 1 & 0 & 0 & 0 & 0 & \cdots & 0 & & & & & & & & & \\ & & 0 & 1 & 0 & \vdots & & & \vdots & & & & & & & & & \\ & & 0 & 0 & 1 & & & & & & & & & & & & & \\ & & 0 & \cdots & 0 & & & & & & & & & & & & & \\ & & \vdots & & \vdots & & & & & & & & & & & & & \\ & & & & & \vdots & \vdots & & & & & & & & & & & & \\ \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & & & & & & & & & & & \end{bmatrix} \dots\dots\dots (6)$$

特別付録 ドローンの飛行制御プログラム解析

$$B_{ss} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \dots & \dots & 0 \\ \vdots & & & \vdots \\ 0 & \dots & \dots & 0 \\ 0.0027 & -0.0027 & 0.0027 & -0.0027 \end{bmatrix} \dots\dots\dots(7)$$

$$C_{ss} = \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & 2 & 0 \\ & & & & 0 & 0 & 2 \\ & & & & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.002 & 0.002 & 0.002 & 0.002 & 0 & 0 & 0 \\ -9.7E-5 & 9.7E-5 & 9.7E-5 & -9.7E-5 & 0 & 0 & 0 \\ -9.7E-5 & -9.7E-5 & 9.7E-5 & 9.7E-5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \dots & \dots & 0 & 0 & 0 & 0 \\ \vdots & & & \vdots & 0 & 0 & 0 \\ & & & & 0 & 0 & 0 \\ & & & & 1 & 0 & 0 \\ & & & & 0 & 1 & 0 \\ & & & & 0 & 0 & 1 \\ & & & & 0 & 0 & 0 \\ & & & & 0 & 0 & 0 \\ \vdots & & & \vdots & 0 & 0 & 0 \\ 0 & \dots & \dots & 0 & 0 & 0 & 0 \end{bmatrix} \dots\dots\dots(8)$$

$$D_{ss} = \begin{bmatrix} 0 & \dots & \dots & 0 \\ \vdots & & & \vdots \\ 0 & \dots & \dots & 0 \\ 5.5E-7 & -5.5E-7 & 5.5E-7 & -5.5E-7 \end{bmatrix} \dots\dots\dots(9)$$

出力方程式については、オイラー角と角速度、機体胴体に働く推力と3軸トルクを出力信号にとっています。ここで、 x_{actti} ($i=1, 2, 3, 4$) は推力/アクチュエータ・モデルに使用した1次遅れ系の状態変数です。また、 x_{atti} ($i=1, 2, 3$) は、クォータニオンの時間微分値の積分の状態変数、 Z_{total} は F_{total} の z 軸成分 (第3成分) です。 E^{xx} の表記は $\times 10^{xx}$ と読み替えてください。

係数行列は、サイズが大きい割に0が多いため、省略した書き方になっていることに気をつけてください。今回はロールとピッチの姿勢角速度の制御系設計を行うので、このモデルを使いやすいように変形します。

係数行列を低次元化する

低次元化とは、不要な状態変数を省いて係数行列の大きさを小さくすることです。 x_{actti} は、プロペラの推力に関係する変数ですが、プロペラが推力によって作る機体胴体の力とトルクは、推力とロール/ピッチのトルクの3つしかないので状態変数として4つ存在するのは冗長です。そこで、 x_{actti} を Z_{total} 、 L_a 、 M_a の3つに変換して状態変数を1つ減らすことを考えます。このような変換は、 C_{ss} の7～9行目かつ1～4列目の12個の成分を利用することで可能です。

次に、入力を各アクチュエータのPWMではなく、スロットル $\Delta\delta_{thr}$ 、エレベータ $\Delta\delta_{ele}$ 、エルロン $\Delta\delta_{ail}$ 、ラダー $\Delta\delta_{rud}$ といった直感的な値に変換します。このような変換は、コマンド分配則 C_{dist} を利用することで可能です。以上の2つの変換をまとめて式で書くと、次のようになります。

$$x_{ss1} = [Z_{total}, L_a, M_a, L_a, x_{att1}, x_{att2}, x_{att3}, p_{xN}, p_{yN}, p_{zN}, v_{xB}, v_{yB}, v_{zB}, \omega_x, \omega_y, \omega_z]^T \dots\dots\dots(10)$$

$$u_{ss1} = [\Delta\delta_{thr}, \Delta\delta_{ele}, \Delta\delta_{ail}, \Delta\delta_{rud}]^T \dots\dots\dots(11)$$

$$y_{ss1} = y_{ss} \dots\dots\dots(12)$$

$$A_{ss1} = T_L A_{ss} T_R \dots\dots\dots(13)$$

$$B_{ss1} = T_L B_{ss} C_{dist} \dots\dots\dots(14)$$

$$C_{ss1} = C_{ss} T_R \dots\dots\dots(15)$$

$$D_{ss1} = D_{ss} C_{dist} \dots\dots\dots(16)$$

$$T_L = \begin{bmatrix} 0.002 & 0.002 & 0.002 & 0.002 \\ -9.7E-5 & 9.7E-5 & 9.7E-5 & -9.7E-5 \\ -9.7E-5 & -9.7E-5 & 9.7E-5 & 9.7E-5 \\ O_{12 \times 4} & I_{12} \end{bmatrix} \dots\dots\dots(17)$$

$$T_R = T_L^+ \dots\dots\dots(18)$$

ここで、変換後の新たな状態変数ベクトルと入力変数ベクトルをそれぞれ x_{ss1} 、 u_{ss1} 、係数行列を A_{ss1} 、 B_{ss1} 、 C_{ss1} 、 D_{ss1} とします。 $O_{n \times m}$ は $(n \times m)$ 次のゼロ行列、 I_n は n 次の単位行列 (対角成分が全て1の行列)、 T_L^+ は T_L の疑似逆行列を表します。さらに、角速度制御系の設計に不要な状態、入力、出力を全て削除すれば、次の低次元化モデルを得ます。

$$\dot{x}_{ssr} = A_{ssr} x_{ssr} + B_{ssr} u_{ssr} \dots\dots\dots(19)$$

$$y_{ssr} = C_{ssr} x_{ssr} + D_{ssr} u_{ssr} \dots\dots\dots(20)$$

$$x_{ssr} = [L_a, M_a, \omega_x, \omega_y]^T \dots\dots\dots(21)$$

$$u_{ssr} = [\Delta\delta_{ele}, \Delta\delta_{ail}]^T \dots\dots\dots(22)$$

$$y_{ssr} = [\omega_x, \omega_y]^T \dots\dots\dots(23)$$

$$A_{ssr} = \begin{bmatrix} -10 & 0 & 0 & 0 \\ 0 & -10 & 0 & 0 \\ 10000 & 0 & 0 & 0 \\ 0 & 10000 & 0 & 0 \end{bmatrix} \dots\dots\dots(24)$$

$$B_{ssr} = \begin{bmatrix} 0.000388 & 0 \\ 0 & 0.000388 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \dots\dots\dots(25)$$

$$C_{ssr} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots\dots(26)$$

$$D_{ssr} = O_{2 \times 2} \dots\dots\dots(27)$$

角速度制御系の設計

● 制御方法は「状態フィードバック制御」

FCUのリファレンス・デザインのソースコードでは、PID制御により角速度制御がなされていますが、ここでは状態フィードバック制御という方法で設計します。

状態フィードバックとは、その名の通り、制御対象の状態変数をフィードバックする制御手法です。これに対し、制御対象の出力信号をフィードバックするPID制御は出力フィードバックと呼ばれます。状態フィードバック制御の歴史はPID制御と同じように長く、多くの研究者によって設計手法が提案され体系化されています。

● 積分型最適サーボ系でゲインを設計する

設計には、参考文献で紹介されている積分型最適サーボ系と呼ばれる手法を取り入れます。これは、PID制御の積分制御と同じように積分器を使用し、かつ、ある評価関数を最小化する（この意味で「最適」な）ゲインを計算により求める設計手法です。この手法の特徴としては、数ある制御理論の中では比較的数式が単純で理解しやすく、設計手法が確立されていて適用しやすいことに加え、多少の制御対象のモデル化誤差に対しても制御系の安定性を保つことができる（安定余有が大きい、ロバスト安定などと呼ぶ）といった点が挙げられます。

● ゲインの設計方法

以下では参考文献の一部を引用しますが、理論の詳細には踏み込まないので興味のある方や詳しく知りたい方は、書籍やドキュメントを参照してください。

積分型最適サーボ系のブロック線図は、図1のように書けます。制御量は、角速度 ω_x 、 ω_y であり、それに対応する目標値を ω_x^{ref} 、 ω_y^{ref} と書くことにします。設計すべきゲインは、行列 F 、 G 、 H です。また、状態変数4つのうち、 L_a と M_a はセンサで直接測ることができないため、これらの値を推定するための状態推定器も設けます。

積分器は、制御器の一部ですがゲインを設計する際

は制御対象の一部に含めます。つまり、制御対象の状態方程式を拡大します。このように拡大した対象を拡大系と呼びます。さらに、目標値に追従しているときに状態変数が0になるように式を書き換えます。このような系を誤差システムと呼びます。

$$\dot{\mathbf{x}}_a = \mathbf{A}_a \mathbf{x}_a + \mathbf{B}_a \mathbf{u}_a \dots\dots\dots(28)$$

$$\mathbf{e} = \mathbf{C}_a \mathbf{x}_a \dots\dots\dots(29)$$

$$\mathbf{x}_a = [\tilde{\mathbf{x}}_{ssr}^T \tilde{\mathbf{w}}^T]^T \dots\dots\dots(30)$$

$$\mathbf{u}_a = \tilde{\mathbf{u}}_{ssr} \dots\dots\dots(31)$$

$$\mathbf{A}_a = \begin{bmatrix} \mathbf{A}_{ssr} & 0 \\ -\mathbf{C}_{ssr} & 0 \end{bmatrix} \dots\dots\dots(32)$$

$$\mathbf{B}_a = \begin{bmatrix} \mathbf{B}_{ssr} \\ 0 \end{bmatrix} \dots\dots\dots(33)$$

$$\mathbf{C}_a = [-\mathbf{C}_{ssr} \ 0] \dots\dots\dots(34)$$

$$\tilde{\mathbf{x}} = \mathbf{x}_{ssr} - \mathbf{x}_{ssr\infty} \dots\dots\dots(35)$$

$$\tilde{\mathbf{w}} = \mathbf{w} - \mathbf{w}_{\infty} \dots\dots\dots(36)$$

$$\tilde{\mathbf{u}} = \mathbf{u}_{ssr} - \mathbf{u}_{ssr\infty} \dots\dots\dots(37)$$

ここで、 $\mathbf{e}$ は2次の追従偏差ベクトル、 $\mathbf{w}$ は積分器の2次の状態変数ベクトル、 $\mathbf{x}_{ssr\infty}$ 、 $\mathbf{w}_{\infty}$ 、 $\mathbf{u}_{ssr\infty}$ は目標値に追従しているときに $\mathbf{x}_{ssr}$ 、 $\mathbf{w}$ 、 $\mathbf{u}_{ssr}$ がとる値です。続いて、次の評価関数 J_a を考えます。

$$J_a = \int_0^{\infty} (\mathbf{e}^T \mathbf{Q}_1 \mathbf{e} + \tilde{\mathbf{w}}^T \mathbf{Q}_2 \tilde{\mathbf{w}} + \tilde{\mathbf{u}}_{ssr}^T \mathbf{R} \tilde{\mathbf{u}}_{ssr}) dt \dots\dots\dots(38)$$

$\mathbf{Q}_1$ 、 $\mathbf{Q}_2$ 、 $\mathbf{R}$ は正定行列（実対称かつ固有値が全て正である行列）で、設計者が決める設計パラメータとなります。この評価関数を最小化するということは、制御量を目標値へ追従させつつ、制御入力をできるだけ小さくすることを意味します。設計パラメータが与えられたとき、ゲイン行列は次式で与えられます。

$$\mathbf{F} = -\mathbf{R}^{-1} \mathbf{B}_{ssr}^T \mathbf{P}_{11} \dots\dots\dots(39)$$

$$\mathbf{G} = -\mathbf{R}^{-1} \mathbf{B}_{ssr}^T \mathbf{P}_{12} \dots\dots\dots(40)$$

$$\mathbf{H} = [-\mathbf{F} + \mathbf{G} \mathbf{P}_{22}^{-1} \mathbf{P}_{12}^T \ \mathbf{I}] \begin{bmatrix} \mathbf{A}_{ssr} & \mathbf{B}_{ssr} \\ \mathbf{C}_{ssr} & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \mathbf{I} \end{bmatrix} \dots\dots\dots(41)$$

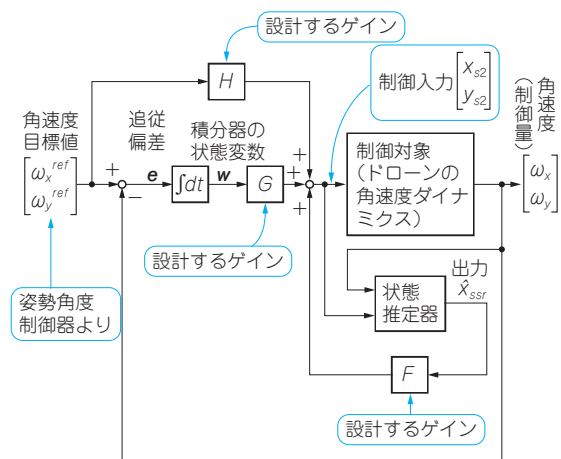


図1 積分型最適サーボ系で角速度を制御する

特別付録 ドローンの飛行制御プログラム解析

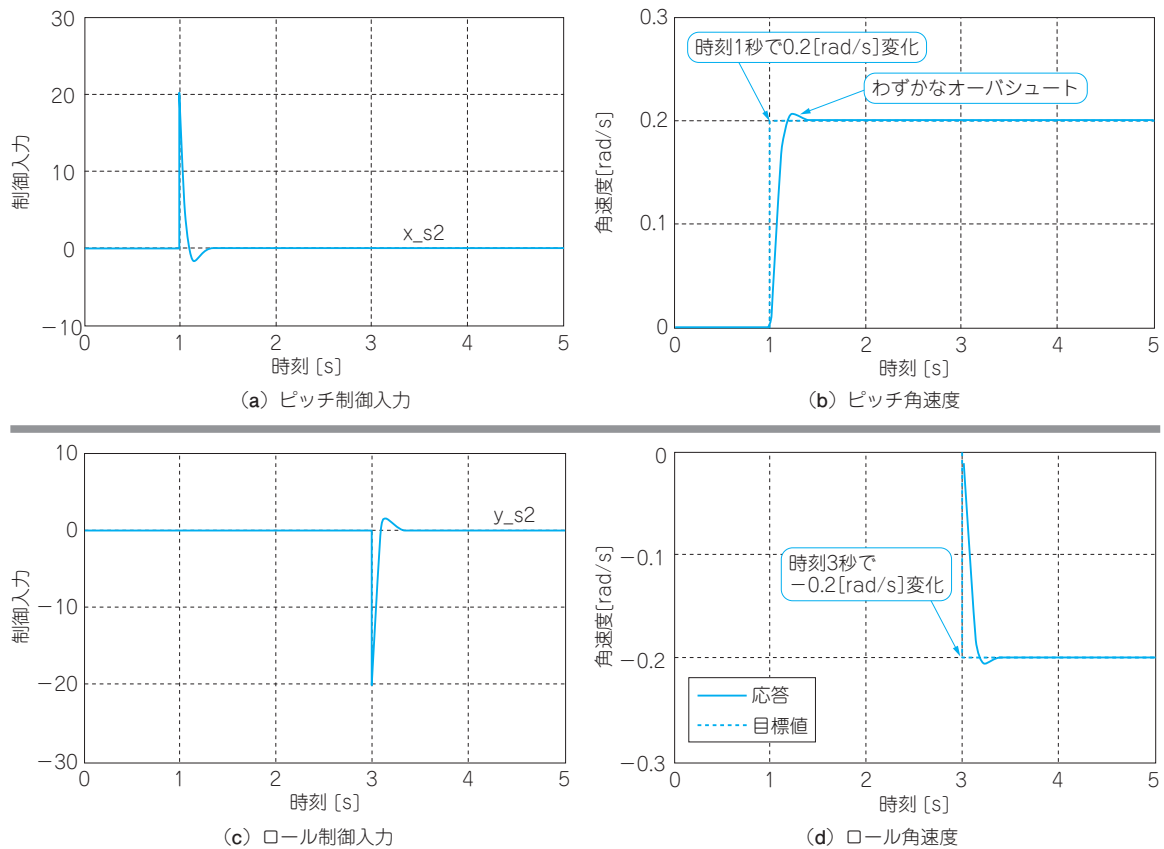


図2 求めたゲインでシミュレーションその1…角速度のステップ目標値応答

ここで、行列 P_{11} , P_{12} , P_{22} は次式の Riccati 方程式を行列 P について解いたもののうち、正定行列で与えます。

$$A_a^T P + P A_a - P B_a R^{-1} B_a^T P + \begin{bmatrix} C_{ssr}^T Q_1 C_{ssr} & 0 \\ 0 & Q_2 \end{bmatrix} = 0 \quad (42)$$

$$P = \begin{bmatrix} P_{11} & P_{12} \\ P_{12}^T & P_{22} \end{bmatrix} \quad (43)$$

以上を用いて、実際にゲインを設計します。また、設計パラメータを次のように与えます。

$$Q_1 = Q_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (44)$$

$$R = 0.0001 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (45)$$

このとき、ゲイン行列は次のように求まります。筆者は、Riccati 方程式の解を求めるため、MATLAB/Simulink のツール・ボックスの1つである Control System Toolbox を利用しました。

$$F = \begin{bmatrix} -5.3E+4 & 0 & -108 & 0 \\ 0 & -5.3E+4 & 0 & -108 \end{bmatrix} \quad (46)$$

$$G = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \quad (47)$$

$$H = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \quad (48)$$

シミュレーション

● 状態推定器の設計にオンライン・シミュレーションを活用

状態推定器は、制御対象のモデルである式 (19) ～ 式 (27) に制御入力を入れてオンラインでシミュレーションを行い、そのときの状態変数の値を推定値として持ってくる、という簡単な方法としました。

なお、角速度は直接計測できるので、オンライン・シミュレーションの値ではなく計測値とします。状態推定器の出力 $\hat{x}_{ssr}$ は次式の通りです。

$$\hat{x}_{ssr} = [\hat{L}_a \ \hat{M}_a \ \omega_x \ \omega_y]^T \quad (49)$$

$\hat{L}_a$, $\hat{M}_a$ はそれぞれ L_a , M_a の推定値です。入力 $\Delta\delta_{ele}$, $\Delta\delta_{ail}$ から直接計測できない状態変数 L_a , M_a までのダイナミクスが安定なので、今回はこうした簡単な方法をとりましたが、本来の状態推定器の設計は、対象のダイナミクスが安定とは限らないので、オンライン・シミュレーションの出力信号(角速度)が実際の制御

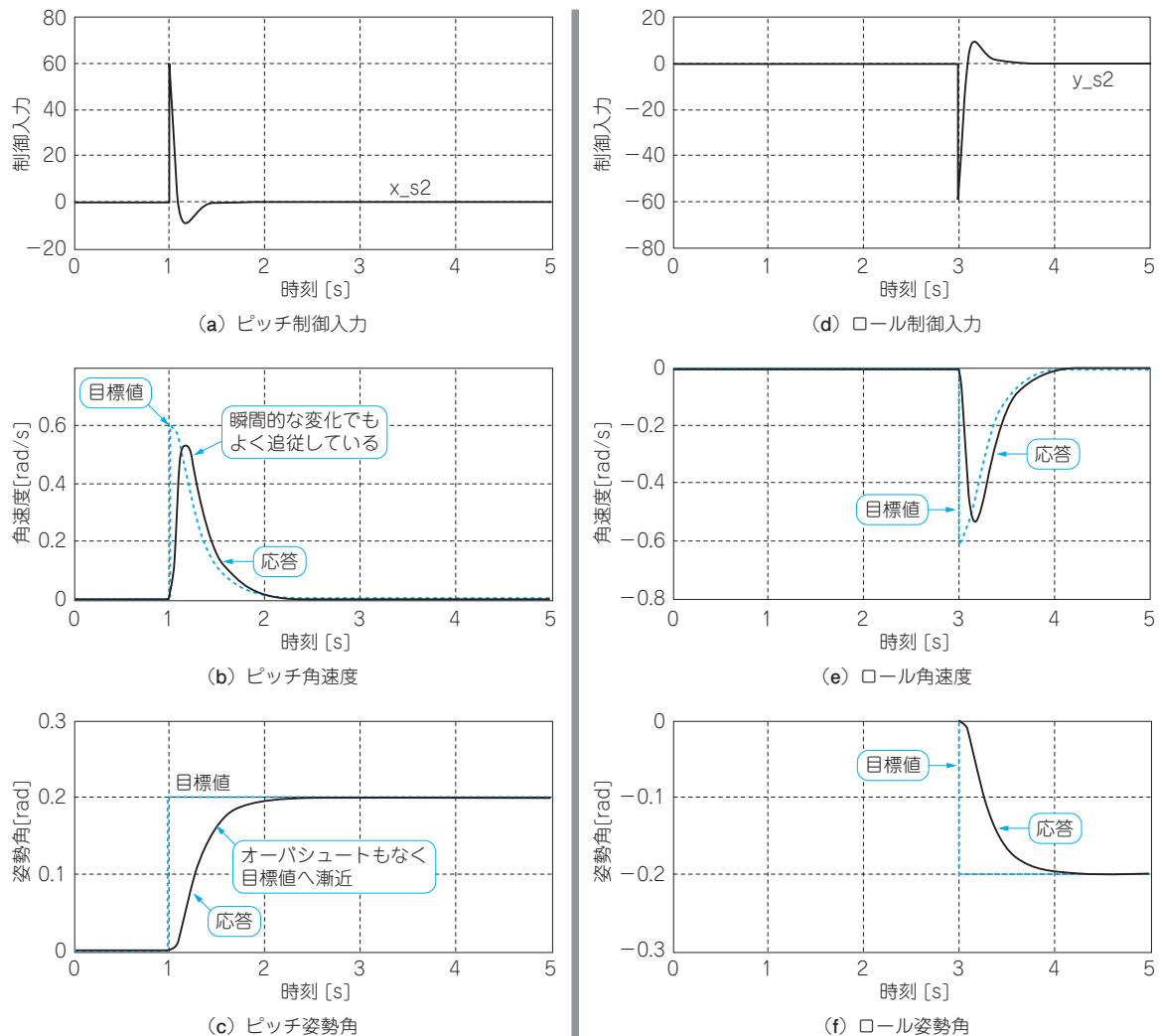


図3 求めたゲインでシミュレーションその2…姿勢角度のステップ目標値応答

対象の出力信号と合うようにフィードバックをかけるのが一般的です。

● 角速度/姿勢角制御の応答

上で求めたゲイン行列を用いて制御シミュレーションを行います。まずは、設計した角速度制御系のステップ目標値応答を見てみます。

ステップ目標値とは、一定値の目標値です。ここでは、時刻1秒で ω_x^{ref} を0から0.2 rad/sへ、時刻3秒で ω_y^{ref} を0から-0.2rad/sへそれぞれ変化させた場合の応答を確認します。シミュレーション結果を図2に示します。わずかにオーバーシュート(目標値を超える行き過ぎ)があるものの、きちんと目標値へ追従する様子が確認できました。

次に、アウト・ループの姿勢角度制御も加え、ピッチとロールの姿勢角度をステップ目標値に追従させる

シミュレーションを行いました。シミュレーション結果は図3に示します。姿勢角は、行き過ぎることなく目標値へ到達し、角速度は瞬間的に変化する目標値に対しても応答が良く追従している様子が確認できました。目標値は決して大きくないですが、制御入力 $\Delta \delta_{\text{ele}}(x_{s2})$, $\Delta \delta_{\text{ail}}(y_{s2})$ が過大になることなく、FCUへの実装も問題なさそうです。

● 繰り返しのシミュレーションでパラメータを調整する

今回は誌面の都合上省略しますが、シミュレーションは質量などのモデルのパラメータを変えたり、外乱を入れたり、さまざまな条件で行って制御性能を評価します。シミュレーションをしっかりとやっていると、次の段階である飛行実験においてトラブルを少なくできます。シミュレーションの過程で、望ましくな

特別付録 ドローンの飛行制御プログラム解析

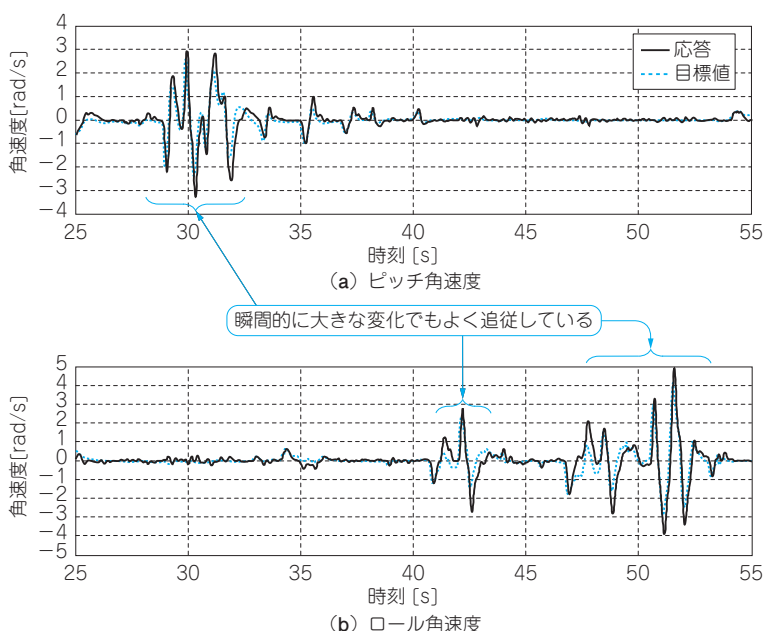


図4 飛行実験で得られたピッチとロール角速度の応答

い結果が出た場合は、設計パラメータ Q_1 , Q_2 , R の値を変えて再設計しシミュレーションを行う、という手順を繰り返してチューニングしていきます。

状態フィードバック角速度制御器の実装

状態フィードバック制御の角速度制御器をFCUに実装します。FCUのソースコードの改修箇所は `flight_control.h` と `flight_control.c` で、改修後のソースコードをそれぞれリスト1, リスト2に示します。

状態フィードバック制御は x , y 軸 (ピッチ, ロール) の制御に適用します。 z 軸 (ヨー) は元のPID制御をそのまま使います。なお、プログラムは以下のURLまたはQRコードから開けます。

<https://interface.cqpub.co.jp/drone2204/>

● `flight_control.h` (リスト1) にはパラメータ値を追加した

▶ 115 ~ 142行目

元のソースコードの末尾に、新たに必要となるパラメータを書き入れました。

▶ 115 ~ 126行目

定数 `RP_RCTRL_FA`, `RP_RCTRL_GA`, `RP_RCTRL_HA` は、それぞれ設計した積分型最適サーボ系のゲイン行列 F , G , H の値です。

▶ 127 ~ 138行目

定数 `RP_RCTRL_AOD`, `RP_RCTRL_BOD`, `RP_RCTRL_COD` は、状態推定器の状態空間モデルの係数行列 (A , B , C 行列) の値です。

この状態空間モデルは、状態変数と出力がどちらも $[\hat{L}_a \ \hat{M}_a]^T$, 入力が $\hat{u}_{ssr} = [\Delta\delta_{ele} \ \Delta\delta_{ail}]^T = [x_s2 \ y_s2]^T$ となっていて、事前にインナ・ループ制御のサンプリング時間で離散化した離散時間系です。

● `flight_control.c` (リスト2) には状態フィードバック角速度制御の計算処理を実装した

▶ 22 ~ 31行目

状態フィードバック制御に用いる各行列や、制御器および状態推定器の状態変数の宣言と初期化を追加しました。

▶ 211行目

使用する角速度制御則を選択するための定数 `TYP_CTRL` を追加しました。これが0の場合は元のPID制御器が、1の場合は新たに実装する状態フィードバック制御器が動作します。

▶ 214 ~ 374行目

`FlightControlPID_innerLoop()` 関数は、インナ・ループ角速度制御器の実装です。元はPID制御ですが、この関数を書き換えて状態フィードバック制御を実装します。

▶ 218 ~ 229行目

状態フィードバック制御で使用する変数を宣言します。



▶ 242 ~ 300行目

ここが、新たに追加した状態フィードバック制御です。TYP_CTRLが1の場合に実行されます。

▶ 244 ~ 250行目

追従偏差の積分値と状態推定器の状態変数は、スロットル操作量がしきい値未満になると0に初期化されます。

▶ 253 ~ 254行目

状態推定器の出力方程式を計算し、状態変数の推定値を得ます。mx, myはそれぞれ $\hat{L}_a, \hat{M}_a$ です。dmxe, dmyeは現在時刻における状態推定器の状態変数で、値はmx, myと同じです。

▶ 256 ~ 257行目

u1_F, u2_Fは、積分型最適サーボ系のうち、状態フィードバックにあたる、制御ゲイン行列Fを用いた制御入力計算です。

▶ 258 ~ 259行目

x, y軸の角速度の追従偏差を求めます。

▶ 260行目

x軸について、追従偏差の積分を求めます。

▶ 261 ~ 264行目

260行目で求めた積分値が $\pm \text{EGX_I_LIMIT}$ の範囲を超えないように制限をかけるリミッタです。

▶ 265 ~ 269行目

x軸と同様に、y軸についても、追従偏差の積分を求め、リミッタにより飽和させます。

▶ 270 ~ 271行目

u1_G, u2_Gは、積分型最適サーボ系のうち、積分制御にあたる、制御ゲイン行列Gを用いた制御入力計算です。

▶ 272 ~ 273行目

u1_H, u2_Hは、積分型最適サーボ系のうち、フィードフォワード制御にあたる、制御ゲイン行列Hを用いた制御入力計算です。

▶ 274 ~ 275行目

上で求めたu1_F, u1_G, u1_Hおよびu2_F, u2_G, u2_Hをそれぞれ足し合わせて、制御出力であるpid->x_s2, y_s2を求めます。元のPID制御のときの変数を流用しているため、変数名にpidと書かれています。

▶ 277 ~ 281行目

制御出力が過大にならないようにするリミッタです。

▶ 283 ~ 298行目

状態推定器の計算を行う部分です。

▶ 283 ~ 292行目

制御出力pid->x_s2, y_s2は状態推定器への入力となります。過大な値が入力されないよう、pid->x_s2, y_s2を別の変数x_s2_1, y_s2_1にコピーしたうえでリミッタをかけます。

▶ 293 ~ 296行目

状態推定器の状態方程式を計算します。計算結果であるdmxe_next, dmye_nextは、次の時刻におけるdmxe, dmyeの値となります。

▶ 297 ~ 298行目

dmxe, dmyeを次の時刻の値に更新します。

▶ 304 ~ 336行目

元のPID制御によるx, y軸の角速度制御です。304行目、336行目の追加により、定数TYP_CTRLが0の場合に実行されるよう変更しました。

実装が終わったら飛行実験

プログラムをコンパイルしてマイコンに書き込んだら、いよいよ飛行実験です。

● 飛行実験でスティック操作に対するレスポンスを計測

今回は飛行中にスティックをさまざまな方向に振ってみて、そのときの機体の運動応答を計測しました。角速度のデータを図4に示します。飛行実験では、機体がひっくり返って墜落するため角速度を一定値にできないので、ステップ目標値応答は見られませんが、瞬間的に、かつ大きく変化する角速度目標値に応答が良く追従している様子が図4から確認できます。

ちなみに、あえて積分制御だけ消して(図1の行列Gの信号を切って)飛行させると、応答と目標値の間に定常的な追従偏差が発生し、スティックが中央からずれた位置でないとホバリングできなくなり、操縦がしづらくなります。その主な原因は、機体の重心が胴体中央から少しずれていて、これがx, y軸に一定値のトルク外乱として入るためです。積分制御を行えば、このような重心のずれは自動的に補償されます。ただし、積分制御による補償は、積分値がたまるまでの時間飛行させ続けてから効果が出ます。従って、あまりにも重心のずれが大きいと、飛行させる以前に離陸できなくなる場合があるので、制御に頼りきるのではなく、飛行できるようにハードウェアを健全に作り込むことが肝要です。

● 飛行実験の注意点

飛行実験を行う際に、プログラムにはバグがないとも限らないので最初は慎重に行ってください。機体の挙動を注意深く観察し、もし予期しない挙動を示したら、すぐにスロットルを下げるようにしてください。

参考文献

(1) 池田 雅夫, 藤崎 泰正: 多変数システム制御, コロナ社, 2010年.

ふじわら・だいご