第2部

第8章 状態フィードバック制御のプログラム＆飛行実験

航空機と同様の方法による ドローン飛行制御＆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の数学モデルは簡単化しているものの，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$\boldsymbol{u}_{ss}$，モデルからの出力ベクトルを$\boldsymbol{y}_{ss}$，状態変数ベクトルを$\boldsymbol{x}_{ss}$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．

$$\dot{\boldsymbol{x}}_{\mathrm{ss}}=A_{\mathrm{ss}}\boldsymbol{x}_{\mathrm{ss}}+B_{\mathrm{ss}}\boldsymbol{u}_{\mathrm{ss}} \quad\cdots\cdots(1)$$

$$\boldsymbol{y}_{\mathrm{ss}}=C_{\mathrm{ss}}\boldsymbol{x}_{\mathrm{ss}}+D_{\mathrm{ss}}\boldsymbol{u}_{\mathrm{ss}} \quad\cdots\cdots(2)$$

ただし，$\boldsymbol{u}_{ss}$, $\boldsymbol{y}_{ss}$, $\boldsymbol{x}_{ss}$はトリム値からの変化分とします．$\dot{\boldsymbol{x}}_{ss}$は$\boldsymbol{x}_{ss}$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，$\boldsymbol{x}_{ss}$や$\boldsymbol{u}_{ss}$の各成分がわずかに変化したとき，$\dot{\boldsymbol{x}}_{ss}$や$\boldsymbol{y}_{ss}$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$\boldsymbol{x}_{ss}$に関する$\dot{\boldsymbol{x}}_{ss}$の微係数，$B_{ss}$は$\boldsymbol{u}_{ss}$に関する$\dot{\boldsymbol{x}}_{ss}$の微係数，$C_{ss}$は$\boldsymbol{x}_{ss}$に関する$\boldsymbol{y}_{ss}$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．

$$\boldsymbol{x}_{ss}=[x_{\mathrm{actt1}}\ \ x_{\mathrm{actt2}}\ \ x_{\mathrm{actt3}}\ \ x_{\mathrm{actt4}}\ \ x_{\mathrm{att1}}\ \ x_{\mathrm{att2}}\ \ x_{\mathrm{att3}}\ \ p_{x\mathrm{N}}\ \ p_{y\mathrm{N}}\ \ p_{z\mathrm{N}}\ \ v_{x\mathrm{B}}\ \ v_{y\mathrm{B}}\ \ v_{z\mathrm{B}}\ \ \omega_x\ \ \omega_y\ \ \omega_z]^T \quad\cdots\cdots(3)$$

$$\boldsymbol{u}_{ss}=[\Delta\delta_1\ \ \Delta\delta_2\ \ \Delta\delta_3\ \ \Delta\delta_4]^T \quad\cdots\cdots(4)$$

$$\boldsymbol{y}_{ss}=[\phi\ \ \theta\ \ \psi\ \ \omega_x\ \ \omega_y\ \ \omega_z\ \ Z_{\mathrm{total}}\ \ L_{\mathrm{a}}\ \ M_{\mathrm{a}}\ \ N_{\mathrm{a}}]^T \quad\cdots\cdots(5)$$

$$A_{\mathrm{ss}}=\left[\begin{array}{ccccccc}
-10 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & -10 & 0 & 0 & \vdots & \vdots & \vdots\\
0 & 0 & -10 & 0 & & & \\
0 & 0 & 0 & -10 & & & \\
0 & \cdots & \cdots & 0 & & & \\
\vdots & & & \vdots & & & \\
 & & & & & & \\
 & & & & \vdots & \vdots & \\
 & & & & 0 & 0 & \\
\vdots & & & \vdots & 0 & -19.6 & \\
0 & \cdots & \cdots & 0 & -19.6 & 0 & \\
0.0262 & 0.0262 & 0.0262 & 0.0262 & 0 & 0 & \\
-0.97 & 0.97 & 0.97 & -0.97 & \vdots & \vdots & \\
-0.97 & -0.97 & 0.97 & 0.97 & \vdots & \vdots & \vdots\\
0 & \cdots & \cdots & 0 & 0 & 0 & 0
\end{array}\right.$$

$$\left.\begin{array}{cccccccc}
\cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 0\\
 & \vdots & \vdots & & \vdots & 0 & 0 & 0\\
 & & & & & 0 & 0 & 0\\
 & & & & & 0 & 0 & 0\\
 & & & & & 0.5 & 0 & 0\\
 & & \vdots & & \vdots & 0 & 0.5 & 0\\
 & & 0 & \cdots & 0 & 0 & 0 & 0.5\\
 & & 1 & 0 & 0 & 0 & \cdots & 0\\
 & & 0 & 1 & 0 & \vdots & & \vdots\\
 & & 0 & 0 & 1 & & & \\
 & & 0 & \cdots & 0 & & & \\
 & & \vdots & & \vdots & & & \\
 & & & & & & & \\
 & \vdots & \vdots & & \vdots & \vdots & & \vdots\\
\cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0
\end{array}\right] \quad\cdots\cdots(6)$$

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

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

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

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 $\boldsymbol{F}_{total}$ 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．

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

$$\boldsymbol{u}_{ss1}=[\Delta\delta_{thr}\ \ \Delta\delta_{ele}\ \ \Delta\delta_{ail}\ \ \Delta\delta_{rud}]^T \cdots\cdots(11)$$

$$\boldsymbol{y}_{ss1}=\boldsymbol{y}_{ss} \cdots\cdots(12)$$

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

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

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

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

$$T_L=\begin{bmatrix} 0.002 & 0.002 & 0.002 & 0.002 & \\ -9.7E^{-5} & 9.7E^{-5} & 9.7E^{-5} & -9.7E^{-5} & O_{3\times 12} \\ -9.7E^{-5} & -9.7E^{-5} & 9.7E^{-5} & 9.7E^{-5} & \\ & O_{12\times 4} & & & I_{12} \end{bmatrix}$$

$$\cdots\cdots(17)$$

$$T_R=T_L^{+} \cdots\cdots(18)$$

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 $\boldsymbol{x}_{ss1}$，$\boldsymbol{u}_{ss1}$，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．

$$\dot{\boldsymbol{x}}_{ssr}=A_{ssr}\boldsymbol{x}_{ssr}+B_{ssr}\boldsymbol{u}_{ssr} \cdots\cdots(19)$$

$$\boldsymbol{y}_{ssr}=C_{ssr}\boldsymbol{x}_{ssr}+D_{ssr}\boldsymbol{u}_{ssr} \cdots\cdots(20)$$

$$\boldsymbol{x}_{ssr}=[L_a\ \ M_a\ \ \omega_x\ \ \omega_y]^T \cdots\cdots(21)$$

$$\boldsymbol{u}_{ssr}=[\Delta\delta_{ele}\ \ \Delta\delta_{ail}]^T \cdots\cdots(22)$$

$$\boldsymbol{y}_{ssr}=[\omega_x\ \ \omega_y]^T \cdots\cdots(23)$$

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

入門

空ドローン制御

自走ロボ制御

水中ドローン制御

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

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

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

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．

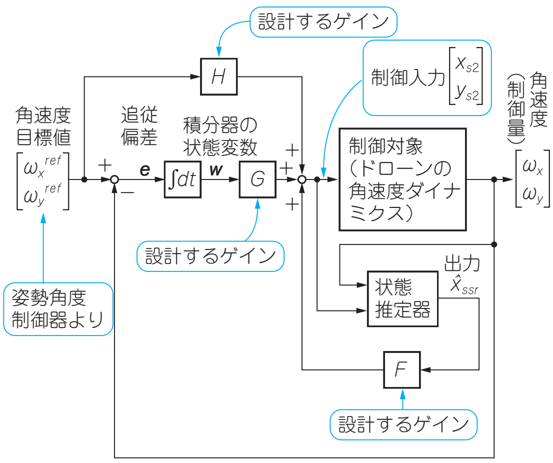

図1　積分型最適サーボ系で角速度を制御する

● 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．

$$\dot{\boldsymbol{x}}_a=A_a\boldsymbol{x}_a+B_a\boldsymbol{u}_a \quad \cdots\cdots(28)$$

$$\boldsymbol{e}=C_a\boldsymbol{x}_a \quad \cdots\cdots(29)$$

$$\boldsymbol{x}_a=[\tilde{\boldsymbol{x}}_{ssr}^T \ \ \tilde{\boldsymbol{w}}^T]^T \quad \cdots\cdots(30)$$

$$\boldsymbol{u}_a=\tilde{\boldsymbol{u}}_{ssr} \quad \cdots\cdots(31)$$

$$A_a=\begin{bmatrix} A_{ssr} & 0 \\ -C_{ssr} & 0 \end{bmatrix} \quad \cdots\cdots(32)$$

$$B_a=\begin{bmatrix} B_{ssr} \\ 0 \end{bmatrix} \quad \cdots\cdots(33)$$

$$C_a=[-C_{ssr} \ \ 0] \quad \cdots\cdots(34)$$

$$\tilde{\boldsymbol{x}}=\boldsymbol{x}_{ssr}-\boldsymbol{x}_{ssr\infty} \quad \cdots\cdots(35)$$

$$\tilde{\boldsymbol{w}}=\boldsymbol{w}-\boldsymbol{w}_{\infty} \quad \cdots\cdots(36)$$

$$\tilde{\boldsymbol{u}}=\boldsymbol{u}_{ssr}-\boldsymbol{u}_{ssr\infty} \quad \cdots\cdots(37)$$

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

$$J_a=\int_0^{\infty}(\boldsymbol{e}^T Q_1\boldsymbol{e}+\tilde{\boldsymbol{w}}^T Q_2\tilde{\boldsymbol{w}}+\tilde{\boldsymbol{u}}_{ssr}^T R\tilde{\boldsymbol{u}}_{ssr})\,dt \quad \cdots\cdots(38)$$

Q_1，Q_2，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．

$$F=-R^{-1}B_{ssr}^T P_{11} \quad \cdots\cdots(39)$$

$$G=-R^{-1}B_{ssr}^T P_{12} \quad \cdots\cdots(40)$$

$$H=\left[-F+GP_{22}^{-1}P_{12}^T \ \ I\right]\begin{bmatrix} A_{ssr} & B_{ssr} \\ C_{ssr} & 0 \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ I \end{bmatrix} \quad \cdots\cdots(41)$$

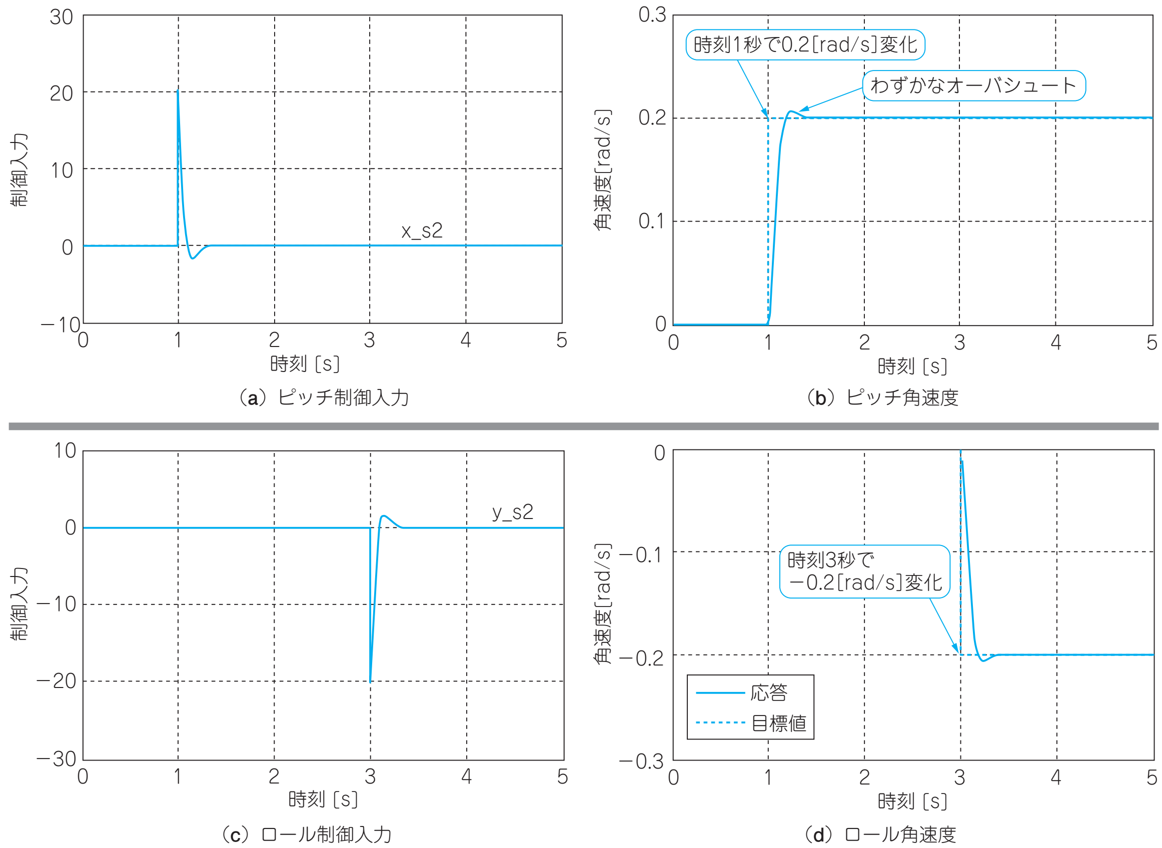

図2　求めたゲインでシミュレーションその1…角速度のステップ目標値応答

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．

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

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

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るとします．

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

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

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．

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

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

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

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．

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

L_a，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

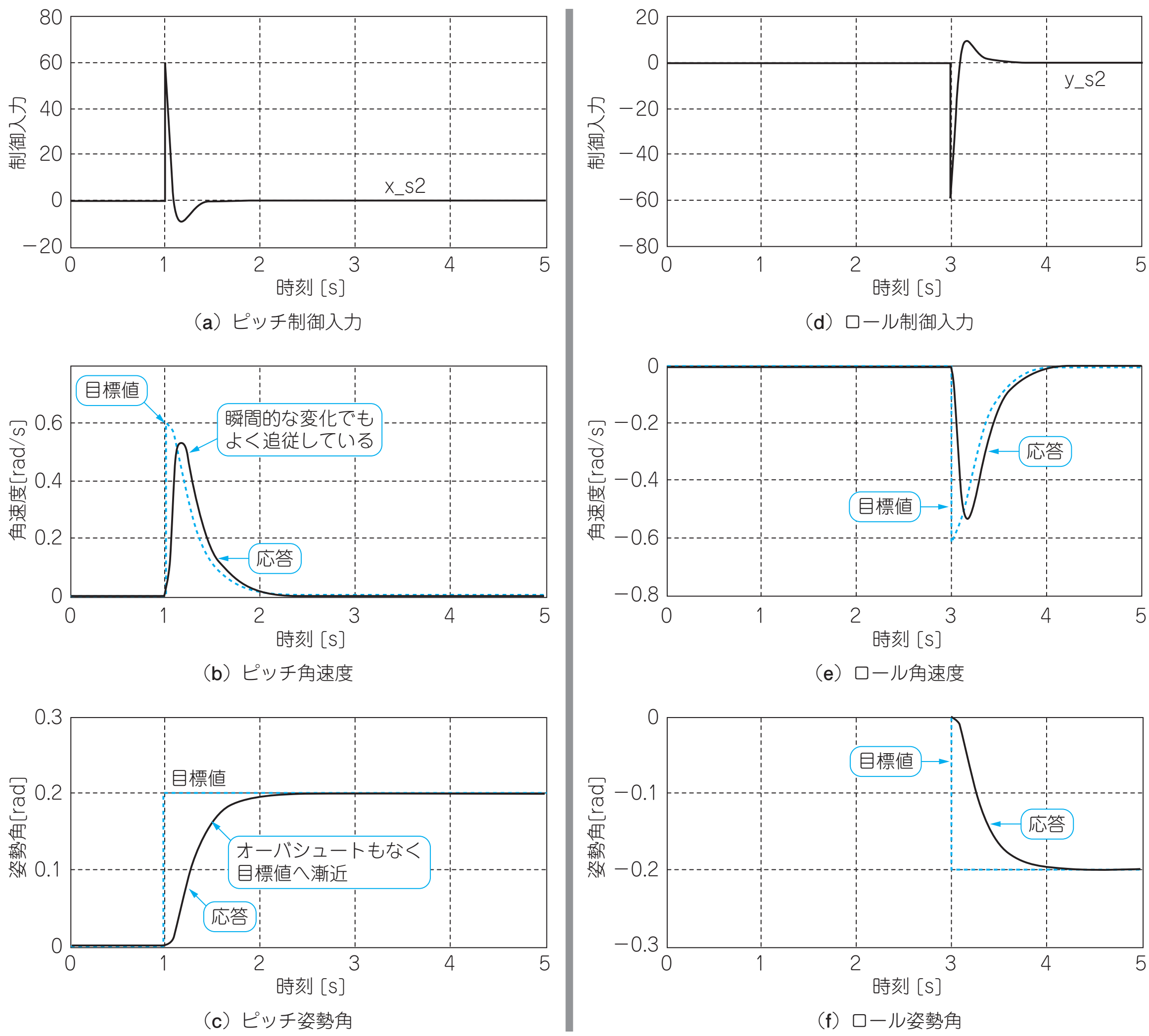

図3　求めたゲインでシミュレーションその2…姿勢角度のステップ目標値応答

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

上で求まっ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，アウタ・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$\Delta\delta_{ele}$(x_s2)，$\Delta\delta_{ail}$(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，外乱を入れたりと，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やっておく

リスト１　飛行実験用に`flight_control.h`を変更した…元の113行目に挿入：制御に使用するパラメータの値

```
#define RP_RCTRL_FA { ¥
 -53032.F,      0.F, -107.592F,        0.F, ¥
      0.F, -53032.F,        0.F, -107.592F ¥
 }
#define RP_RCTRL_GA { ¥
 100.F,    0.F, ¥
   0.F, 100.F ¥
 }
#define RP_RCTRL_HA { ¥
 100.268F,      0.F, ¥
      0.F, 100.268F ¥
 }
#define RP_RCTRL_AOD { ¥
 0.987578F,      0.F, ¥
      0.F, 0.987578F ¥
 }
#define RP_RCTRL_BOD { ¥
 4.81981e-007F,          0.F, ¥
            0.F, 4.81981e-007F ¥
 }
#define RP_RCTRL_COD { ¥
 1.F, 0.F, ¥
 0.F, 1.F ¥
 }
#define EGX_I_LIMIT  (20.F)
#define EGY_I_LIMIT  (20.F)
#define X_S2_LIMIT_O  (10.F)
#define Y_S2_LIMIT_O  (10.F)
```

リスト２　飛行実験用に`flight_control.c`を変更した

元の21行目に挿入：変数初期化

```
float rp_rctrl_Fa[] = RP_RCTRL_FA;
float rp_rctrl_Ga[] = RP_RCTRL_GA;
float rp_rctrl_Ha[] = RP_RCTRL_HA;
float rp_rctrl_Aod[] = RP_RCTRL_AOD;
float rp_rctrl_Bod[] = RP_RCTRL_BOD;
float rp_rctrl_Cod[] = RP_RCTRL_COD;
float egx_integ = 0;   // gxの追従偏差の積分値
float egy_integ = 0;   // gyの追従偏差の積分値
float dmxe = 0;    // MXの推定値
float dmye = 0;    // MYの推定値
:
```

元の197行目に挿入：変数の宣言

```
float u1_F, u2_F;
float u1_G, u2_G;
float u1_H, u2_H;
float mx;   // モーメントMX推定値
float my;   // モーメントMY推定値
float egx; // gxの追従偏差
float egy; // gyの追従偏差
float x_s2_l, y_s2_l;
   // x_s2, y_s2の値（状態推定器用）
float dmxe_next; // dmxeの1サンプル更新後の値
float dmye_next; // dmyeの1サンプル更新後の値
:
```

元の205～237行目をコメントアウト

```
    /*
      //X Axis
      error = pid->x_s1 - gyro_rad->gx;
:
      if(pid->y_s2 > MAX_ADJ_AMOUNT)  pid->y_s2 =
                                  MAX_ADJ_AMOUNT;
      if(pid->y_s2 < -MAX_ADJ_AMOUNT)  pid->y_s2 =
                                 -MAX_ADJ_AMOUNT;
    */
:
```

以下は元の205行目に挿入：*X*，*Y*軸の角速度制御

スロットルを下げているとき（着陸中など）は追従偏差積分値と状態推定器の変数を初期化する

```
  if(gTHR<MIN_THR)
  {
    egx_integ = 0;
    egy_integ = 0;
    dmxe = 0;
    dmye = 0;
  }
```

状態推定器の出力方程式

```
  //XY Axis
  mx = rp_rctrl_Cod[0] * dmxe + rp_rctrl_Cod[1] *
                                             dmye;
  my = rp_rctrl_Cod[2] * dmxe + rp_rctrl_Cod[3] *
                                             dmye;
```

状態フィードバック部

```
  u1_F = rp_rctrl_Fa[0] * mx + rp_rctrl_Fa[1] * my +
 rp_rctrl_Fa[2] * gyro_rad->gx + rp_rctrl_Fa[3] *
                                    gyro_rad->gy;
  u2_F = rp_rctrl_Fa[4] * mx + rp_rctrl_Fa[5] * my +
rp_rctrl_Fa[6] * gyro_rad->gx + rp_rctrl_Fa[7] *
                                    gyro_rad->gy;
```

積分制御部

```
  egx = pid->x_s1 - gyro_rad->gx;
  egy = pid->y_s1 - gyro_rad->gy;
  egx_integ += egx * pid->ts;
  if(egx_integ > EGX_I_LIMIT)
    egx_integ = EGX_I_LIMIT;
  else if(egx_integ < -EGX_I_LIMIT)
    egx_integ = -EGX_I_LIMIT;
  egy_integ += egy * pid->ts;
  if(egy_integ > EGY_I_LIMIT)
    egy_integ = EGY_I_LIMIT;
  else if(egy_integ < -EGY_I_LIMIT)
    egy_integ = -EGY_I_LIMIT;
  u1_G = rp_rctrl_Ga[0] * egx_integ +
                      rp_rctrl_Ga[1] * egy_integ;
  u2_G = rp_rctrl_Ga[2] * egx_integ +
                      rp_rctrl_Ga[3] * egy_integ;
```

フィード・フォワード制御部

```
  u1_H = rp_rctrl_Ha[0] * pid->x_s1 +
                      rp_rctrl_Ha[1] * pid->y_s1;
  u2_H = rp_rctrl_Ha[2] * pid->x_s1 +
                      rp_rctrl_Ha[3] * pid->y_s1;
```

制御出力

```
  pid->x_s2 = u1_F + u1_G + u1_H;
  pid->y_s2 = u2_F + u2_G + u2_H;
```

制御出力に対するリミッタ

```
  if(pid->x_s2 > MAX_ADJ_AMOUNT)  pid->x_s2 =
                                  MAX_ADJ_AMOUNT;
  if(pid->x_s2 < -MAX_ADJ_AMOUNT)  pid->x_s2 =
                                 -MAX_ADJ_AMOUNT;

  if(pid->y_s2 > MAX_ADJ_AMOUNT)  pid->y_s2 =
                                  MAX_ADJ_AMOUNT;
  if(pid->y_s2 < -MAX_ADJ_AMOUNT)  pid->y_s2 =
                                 -MAX_ADJ_AMOUNT;
```

状態推定器の状態方程式

```
  x_s2_l = pid->x_s2;
  if (x_s2_l > X_S2_LIMIT_O)
    x_s2_l = X_S2_LIMIT_O;
  else if (x_s2_l < -X_S2_LIMIT_O)
    x_s2_l = -X_S2_LIMIT_O;
  y_s2_l = pid->y_s2;
  if (y_s2_l > Y_S2_LIMIT_O)
    y_s2_l = Y_S2_LIMIT_O;
  else if (y_s2_l < -Y_S2_LIMIT_O)
    y_s2_l = -Y_S2_LIMIT_O;
  dmxe_next = rp_rctrl_Aod[0] * dmxe +
                         rp_rctrl_Aod[1] * dmye +
    rp_rctrl_Bod[0] * x_s2_l +
                        rp_rctrl_Bod[1] * y_s2_l;
  dmye_next = rp_rctrl_Aod[2] * dmxe +
                         rp_rctrl_Aod[3] * dmye +
    rp_rctrl_Bod[2] * x_s2_l + rp_rctrl_Bod[3] *
                                          y_s2_l;
  dmxe = dmxe_next;
  dmye = dmye_next;
```

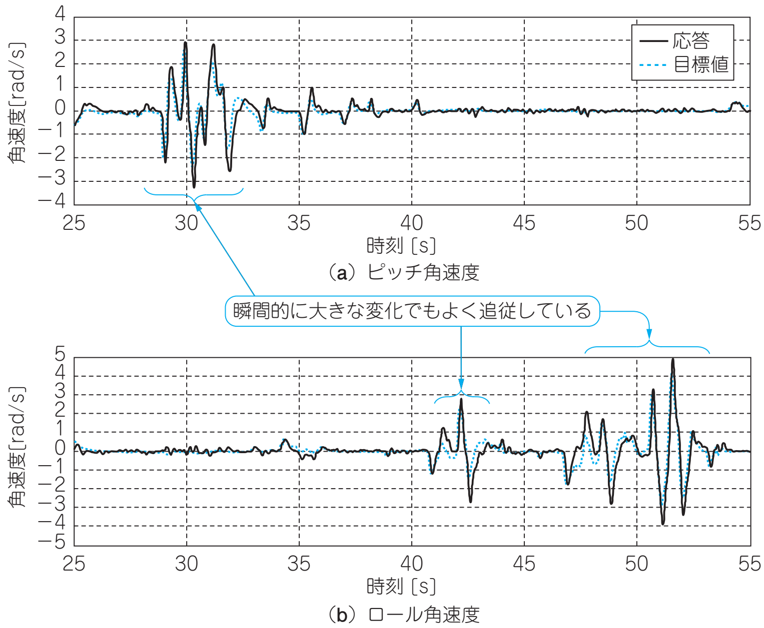

図4　飛行実験で得られたピッチとロール角速度の応答

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．

角速度制御器を実装して飛行実験

● 実験に合わせてFCUソースコードを変更

まず，角速度制御器をFCUに実装します．FCUのソースコードの変更箇所は，**リスト1**，**リスト2**の通りです．**リスト1**（`flight_control.h`）には設計した制御ゲインや状態推定器に使うモデル，リミッタの値といったパラメータ値を新たに書き入れました．

リスト2（`flight_control.c`）については，新たなグローバル変数の定義と，`FlightControlPID_innerLoop()`関数の変更を行いました．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