制御系のシミュレーション環境とリアルタイム制御環境の融合

正 員　古 賀 雅 伸　(東 工 大)

An Integrated Environment for Simulation and Real-time Control of Control Systems

Masanobu Koga, Member, (Tokyo Institute of Technology)

In this paper, development of a CAD of control systems which realizes an integrated environment for simulation and real-time implementation is introduced. It enables us to do not only analysis of control systems and design of controllers but also simulation and real-time implementation with a digital controller in the same software environment. By utilizing this software, the control engineer is able to repeat the procedure of design of control systems efficiently to achieve the best performance in much shorter time. It is also shown that a real-time control program is generated from a simulation program with a slight modification by making two programs have the similar structure.

キーワード: リアルタイム制御，シミュレーション，制御系 CAD

1. はじめに

これまで，制御系 CAD(Computer Aided Design)，すなわちコンピュータ支援による制御系設計のためのソフトウェアが大学や企業で数多く開発されてきた[1,2,3,4,5]。近年，CAD により設計されたコントローラをディジタルコントローラで実現する作業を自動化する試みが行われている。これは，制御系の設計から (ディジタルコントローラによる) 実現までに要する時間の短縮と CAD により設計されたコントローラと実現されたコントローラの一貫性を保つことを目指している。多くの制御系 CAD は便利で使いやすいシミュレーション環境を提供しており，シミュレーション環境で性能を評価したコントローラを簡単にリアルタイム制御環境に移植できることが望まれている。

すでにいくつかの方法が発表されている中で，たとえば，Matrixx[2]の AutoCode や Simulink[6]の C-code 生成ツールボックスのようにシミュレーション用に作成したブロック線図に対応する C 言語のコードを出力するモジュールをソフトウェアパッケージに付加する方法，Simnon[7]のコードを Modula2 のコードに変換する方法[8]などが提案されている。

本論文では，制御系の解析・設計用のプログラミング言語を拡張し，シミュレーション用プログラム及びリアルタイム制御用プログラムを直接記述する方法を提案する。これにより制御系の解析・設計環境，シミュレーション環境，リアルタイム制御環境の融合が可能となり，コントローラの移植作業に要する時間が短縮される。ブロック線図に対応する C 言語のコードを生成するモジュールを付加する方法と比較して，本方法は実験プログラムに必要に応じて制御系解析・設計ルーチンを組み込めるので，プログラムをコンパイルし直さずにコントローラの修正と実験を繰り返し実行できるという点で優れている。[9]

本研究では，提案するソフトウェア環境を実現するため制御系 CAD 言語 MaTX[5]にシミュレーション機能とリアルタイムプログラムを生成する機能を付加し，リアルタイム制御系 CAD 言語 RtMaTX を開発した。具体的な使用例により本ソフトウェアの有効性を本論文で示す。また，シミュレーションプログラムとリアルタイム制御プログラムにできるだけ似た構造をもたせる手法を検討し，シミュレーションプログラムからリアルタイム制御プログラムを最小限の変更で作成する方法を提案する。

論文の構成は以下の通りである。まず，2 章で本研究で扱うシステムのクラスについて述べる。そして，3 章でシミュレーション環境について，4 章でリアルタイム制御環境について述べ，5 章で論文のまとめを行う。

2. システムの表現

本章では，本論文で扱うシステムのクラスについて述べる。システム理論およびその応用に関する多くの文献で取り扱われるシステムは，時 (不) 変常微分方程式また

は差分方程式で表される状態方程式と出力方程式で記述される。偏微分方程式で表される分布定数系 (熱や物質移動を伴うシステムなど) も集中系で近似し，常微分方程式で表すことが多い。本論文では状態方程式が常微分 (または差分) 方程式 (1) 式，出力方程式が代数方程式 (2) 式，入力方程式が代数方程式 (3) 式で表されるシステムについて考える。ただし，$X(t) \in R^n$ は状態ベクトル，$Y(t) \in R^p$ は出力ベクトル，$U(t) \in R^m$ は入力ベクトルであり，R^n は n 次元の実ベクトルを表す。また，連続時間系の場合 $\mathcal{P}X(t)$ は $\frac{d}{dt}X(t)$ を表し，離散時間系の場合 $\mathcal{P}X(k)$ は $X(k+1)$ を表す。 $f(\cdot)$ と $g(\cdot)$ は状態と入力と時刻，$h(\cdot)$ は状態と出力と時刻の (連続性を仮定しない) 任意の関数である。

$$\mathcal{P}X(t) = f(t, X(t), U(t)) \quad (1)$$
$$Y(t) = g(t, X(t), U(t)) \quad (2)$$
$$U(t) = h(t, X(t), Y(t)) \quad (3)$$

システムが N 個のサブシステムからなるとき，状態ベクトル，出力ベクトル，入力ベクトルを

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \quad U = \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix}$$

のようにサブシステムの状態，出力，入力からなるベクトルで表す。ここで，入力ベクトルと出力ベクトルをまとめた入出力ベクトル

$$UY(t) = \begin{bmatrix} U(t) \\ Y(t) \end{bmatrix} \quad (4)$$

を導入すると，(1) 式–(3) 式は

$$\mathcal{P}X(t) = F(t, X(t), UY(t)) \quad (5)$$
$$UY(t) = G(t, X(t)) \quad (6)$$

となる。ただし，$F(\cdot)$ は $f(\cdot)$ と等価であり，$G(\cdot)$ は $g(\cdot)$ と $h(\cdot)$ をまとめたものである。

本論文では，状態方程式と入出力方程式が (5) 式と (6) 式で与えられるシステムのシミュレーション環境及びリアルタイム制御環境について考える。ただし，シミュレーションあるいはリアルタイム制御を実行する時間内で，方程式の解の存在を仮定する。

3. シミュレーション環境

〈3·1〉 状態方程式と入出力方程式の分離

Matlab[1]の `ode23()` や `ode45()` のような常微分方程式を解く関数を用いてシミュレーションを実行する多くのソフトウェアにおいて，状態方程式と入出力方程式は 1 個の関数内にまとめて記述される (図 1 参照)。これに対し，本研究では状態方程式と入出力方程式を 2 個の関数として別々に記述する手法を用いる (図 2 参照)。 本手法には

```
// 線形システムの状態方程式と入出力方程式
Func FG(...)
{
  Y = C*X;
  U = K*X;
  dX = A*X + B*U
}
```

図 1 状態方程式と入出力方程式の関数
Fig. 1. Function of state and input-output equations

```
// 線形システムの状態方程式
Func F(...)
{
  dX = A*X + B*U;
}

// 線形システムの入出力方程式
Func G(...)
{
  Y = C*X;
  U = K*X;
}
```

図 2 状態方程式の関数と入出力方程式の関数
Fig. 2. Function of state eq. and input-output eq.

- 連続時間系のシミュレーションプログラムとサンプル値系のシミュレーションプログラムが同じ構造をもつ。
- サンプル値系のシミュレーションプログラムとリアルタイム制御プログラムが似た構造をもつ。

という特徴がある。ただし，連続時間系のシミュレーションとは連続時間の制御対象に連続時間コントローラを用いるシミュレーション，サンプル値系のシミュレーションとは連続時間の制御対象にディジタルコントローラを用いるシミュレーションを意味する。

〈3·2〉 連続時間系のシミュレーション

本節では，サブシステムの数が $N = 2$ の場合について連続時間系のシミュレーションについて考える。ただし，サブシステムの数が 2 以上の場合も同様に扱える。システム 1 は制御対象であり，状態方程式と入出力方程式が

$$\frac{d}{dt}x_1(t) = f_1(t, x_1(t), u_1(t)) \quad (7)$$
$$y_1(t) = g_1(t, x_1(t), u_1(t)) \quad (8)$$
$$u_1(t) = h_1(t, x_1(t), y_1(t)) \quad (9)$$

で，システム 2 はコントローラであり，状態方程式と入出力方程式が

$$\frac{d}{dt}x_2(t) = f_2(t, x_2(t), u_2(t)) \quad (10)$$
$$y_2(t) = g_2(t, x_2(t), u_2(t)) \quad (11)$$
$$u_2(t) = h_2(t, x_2(t), y_2(t)) \quad (12)$$

で与えられるとする。図3と図4に状態方程式の関数と入出力方程式の関数の概略を示す。 関数 diff_eqs() は (7) 式と (10) 式にしたがって状態の微分 dX を計算する。一方，関数 link_eqs() は (8),(9),(11),(12) 式にしたがってシステムの入出力 UY を計算する。ただし，関数 link_eqs() 内で h1() と g1()，h2() と g2() の計算の順番を適当に決める必要がある。一般に，直達項が無いサブシステムの計算を先に行うよう順番を決めるのが良い。代数ループがある場合，代数方程式をニュートン法で解くなどの工夫が必要となる。 実際のシミュレーション計算は図5に示すメイン関数により実行される。 X0 はシステムの初期状態，t0 と tf はシミュレーションの初期時刻と終端時刻である。関数 Ode() は diff_eqs() と link_eqs() を繰り返し呼び出しながら計算を実行し，状態の時系列 Xs，入出力の時系列 UYs，計算を行った時刻の系列 Ts を求める。また，別の関数 OdeAuto() を用いれば，適応刻み幅制御[10]により指定された精度を保証し，かつ計算時間を短縮する時間刻みを自動で決めることができる。

```
Func diff_eqs(t,X,UY)
{
  dx1 = f1(t, x1, u1);  // 状態方程式 1
  dx2 = f2(t, x2, u2);  // 状態方程式 2
  dX  = [dx1; dx2];     // 状態の微分
}
```

図 3　連続時間系の状態方程式

Fig. 3.　State equation of continuous-time system

```
Func link_eqs(t, X)
{
  u1 = h1(t, x1, y1);    // 入力方程式 1
  y1 = g1(t, x1, u1);    // 出力方程式 2
  u2 = h2(t, x2, y2);    // 入力方程式 1
  y2 = g2(t, x2, u2);    // 出力方程式 2
  UY = [u1; u2; y1; y2]; // 入出力
}
```

図 4　連続時間系の入出力方程式

Fig. 4.　Input-output eq. of continuous-time system

```
Func main()
{
  X0 = [x10; x20];
  {Ts, Xs, UYs} =
       Ode(t0, tf, X0, diff_eqs,link_eqs);
}
```

図 5　連続時間系のシミュレーションのメイン関数

Fig. 5.　Simulation of continuous-time system

〈3·3〉　サンプル値系のシミュレーション

一般にサンプル値系では，ディジタルコントローラが (連続時間系の) 制御対象の出力をサンプラーを通して取り込み，制御出力をホールドを通して制御対象に出力する。一般化サンプラー[11]や一般化ホールドに関する研究もあるが，最も多く用いられるのは一定時間 (サンプル点) 毎に制御対象の出力を取り込む標準型サンプラーとサンプリング点間で一定の値を保つ標準型 0 次ホールドである。ここで考えるサンプル値系のシミュレーションは，この標準型サンプル値系のシミュレーションである。

前節で述べた連続時間系のシミュレーションと同様に，システムの状態方程式と入出力方程式を 2 個の関数に別々に記述する。ただし，diff_eqs() には常微分方程式で与えられる制御対象の状態方程式のみを記述し，差分方程式で与えられるディジタルコントローラの状態方程式は link_eqs() に記述する。特別な場合として，状態フィードバック制御では，両シミュレーションの diff_eqs() と link_eqs() は一致する。

2 番目のシステム，すなわち連続時間コントローラをディジタルコントローラ

$$x_2[k+1] = f_{2d}(k, x_2[k], u_2[k]) \quad (13)$$
$$y_2[k] = g_2(k, x_2[k], u_2[k]) \quad (14)$$
$$u_2[k] = h_2(k, x_2[k], y_2[k]) \quad (15)$$

で置き換えたサンプル値系のシミュレーションについて考える。図6と図7に状態方程式の関数と入出力方程式の関数の概略を示す。 実際のシミュレーション計算は，図8に示すメイン関数により実行される。 dt はサンプリング周期，x20 はディジタルコントローラの初期状態である。関数 OdeHybrid() は link_eqs() をサンプル点毎に呼び出し，サンプル点間で繰り返し diff_eqs() を呼び出

```
Func diff_eqs(t, X, UY)
{
 dx1 = f1(t,x1,u1); // 制御対象の状態方程式
 dX  = dx1;         // 状態の微分
}
```

図 6　サンプル値系の制御対象の状態方程式

Fig. 6.　State equation of sampled-date system

```
Func link_eqs(t, X)
{
  u1 = h1(t, x1, y1);  // 制御対象への入力
  y1 = g1(t, x1, u1);  // 制御対象の出力
  u2 = h2(t, x2, y2);  // 制御器への入力
  y2 = g2(t, x2, u2);  // 制御器の出力
  x2 = f2d(t, x2,u2);  // 制御器の状態方程式
  UY = [u1; u2; y1; y2]; // 入出力
}
```

図 7　サンプル値系の入出力方程式

Fig. 7.　Input-output eq. of sampled-date system

```
Func main()
{
  x2 = x20;         // 制御器の初期状態
  X0 = x10;         // 制御対象の初期状態
  {Ts, Xs, UYs} =
  OdeHybrid(t0,tf,dt,X0,diff_eqs,link_eqs);
}
```

図 8　サンプル値系シミュレーションのメイン関数
Fig. 8.　Simulation of sampled-date system

しながら計算を実行し，状態の時系列 Xs，入出力の時系列 UYs，計算を行った時刻の系列 Ts を求める。

図 3と図 6，図 4と図 7を比較し，コントローラの状態方程式のみが diff_eqs() から link_eqs() に移動することに注意されたい。このように，状態方程式と入出力方程式を 2 個の関数として別々に記述することで，連続時間系のシミュレーションプログラムとサンプル値系のシミュレーションプログラムに似た構造をもたせることができ，プログラムの変更が容易になる。そして，コントローラに関係するのは link_eqs() だけなので，制御則を変更しても diff_eqs() を変更する必要がないので，コントローラ変更に伴うバグの混入を防ぐ効果もある。

〈3·4〉　MATX のシミュレーション環境

本研究では，制御系 CAD 言語 MATX(5)にルンゲ・クッタなどの積分アルゴリズムを組み込み，提案するシミュレーション環境を実現した。本節では，図 9に示す倒立振子の安定化制御のシミュレーションを例題として，開発したシミュレーション環境について説明する。 状態変数

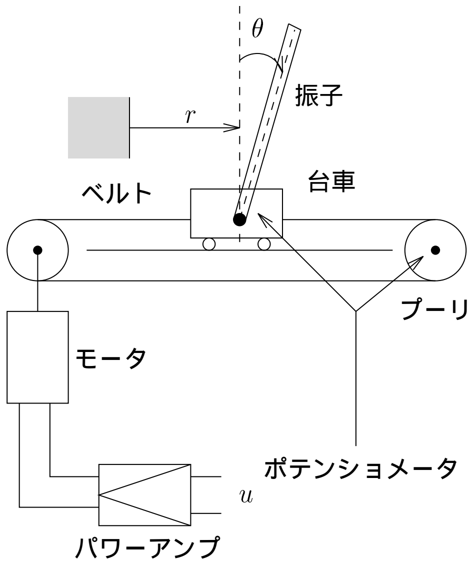

図 9　倒立振子制御系
Fig. 9.　Inverted pendulum

として，台車の変位 r，振子の角度 θ，台車の速度 $\dot{r}$，振子の角速度 $\dot{\theta}$ をとり，状態ベクトルを

$$x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T = \begin{bmatrix} r & \theta & \dot{r} & \dot{\theta} \end{bmatrix}^T$$

とすると，倒立振子の状態方程式は次のように書ける。

$$\dot{x} = f(x,u) \triangleq \begin{bmatrix} x_3 \\ x_4 \\ (1+\alpha\sin^2 x_2)^{-1} \times \{a_{32}\sin x_2 \cos x_2 + a_{33}x_3 + a_{34}\cos x_2 x_4 + a_{35}\sin x_2 x_4^2 + b_3 u\} \\ (1+\alpha\sin^2 x_2)^{-1} \times \{a_{42}\sin x_2 + a_{43}\cos x_2 x_3 + a_{44}x_4 + a_{45}\sin x_2 \cos x_2 x_4^2 + b_4 \cos x_2 u\} \end{bmatrix}$$

また，台車の変位 r と振子の角度 θ を出力変数とすると，出力方程式は

$$y = \begin{bmatrix} c_1 & 0 & 0 & 0 \\ 0 & c_2 & 0 & 0 \end{bmatrix} x = Cx$$

となる。ただし，$\alpha, a_{32}, \ldots, a_{45}, b_3, b_4, c_1, c_2$ は質量や摩擦係数などから求まるパラメータである。平衡点近傍で線形近似して得られた倒立振子の線形モデルに対して状態フィードバック

$$u = -Fx$$

とオブザーバを設計し，制御を行う。ディジタル制御を行うためオブザーバを適当な方法で離散化して

$$\begin{aligned} z[k+1] &= A_{hd}z[k] + B_{hd}y[k] + J_{hd}u[k] \\ x_h[k] &= C_{hd}z[k] + D_{hd}y[k] \end{aligned}$$

を求める。

倒立振子の状態方程式を記述した関数を図 10に，倒立振子の入出力関係およびディジタルコントローラを記述した関数を図 11に，シミュレーションのメイン関数を図 12に示す。

MATX 言語を用いれば，行列を直接記述できるので数学的表現をほぼそのままプログラムとして記述できる。ステップ状目標値変化や白色雑音などの関数を追加すれば，構成後の制御系の基本特性をシミュレーションで調べられる。また，過去の任意時刻の状態と入出力を取り出す関数 OdeXY() を用いれば，異なるサンプリング周期が存在するシステムやむだ時間が存在するシステムのシミュレーションができる。なお，関数 OdeXY() は過去のデータから補間法により任意時刻のデータを生成する。

4.　リアルタイム制御環境

一般にディジタルコントローラによるリアルタイム制御では，コントローラはサンプル点毎に制御対象の出力を取り込み，制御入力の計算 (コントローラの出力の計算，コントローラの状態の更新) を実行し，制御対象に出力する。一方，前章で述べたサンプル値系のシミュレーショ

```
Func void diff_eqs(dX, t, x, UY)
  Matrix dX, x, UY;
  Real t;
{
  Real x1, x2, x3, x4, u, c2, s2, dm;

  x1 = x(1);        // 台車の位置
  x2 = x(2);        // 振子の角度
  x3 = x(3);        // 台車の速度
  x4 = x(4);        // 振子の角速度
  u = UY(1);        // 制御入力

  c2 = cos(x2); s2 = sin(x2);
  dm = (1 + alpha*s2^2);

  dX = Z(4,1);      // 状態方程式
  dX(1) = x(3);
  dX(2) = x(4);
  dX(3) = (a32*s2*c2 + a33*x3 + a34*c2*x4
           + a35*s2*x4^2 + b3*u)/dm;
  dX(4) = (a42*s2 + a43*c2*x3 + a44*x4
           + a45*s2*c2*x4^2 + b4*c2*u)/dm;
}
```

図 10　倒立振子の状態方程式

Fig. 10.　State equation of inverted pendulum

```
Func void link_eqs(UY, t, x)
  Matrix UY, x;
  Real t;
{
  Matrix u, y, xh;
  extern Matrix z;

  y  = C*x;              // 出力方程式
  xh = Chd*z + Dhd*y;    // 状態の推定
  u  = - F*xh;           // 状態フィードバック
  z  = Ahd*z+Bhd*y+Jhd*u; // 観測器の更新
  UY = [[u][y]];
}
```

図 11　倒立振子のディジタルコントローラ

Fig. 11.　Digital controller of inverted pendulum

ンでは，制御対象の入出力方程式，コントローラの入出力方程式および状態方程式を記述した link_eqs() の計算がサンプル点毎に実行される。このように，サンプル値系のシミュレーションプログラムでサンプル点毎に実行される処理とリアルタイム制御プログラムでサンプル点毎に実行される処理はほとんど一致する。これはシステムの状態方程式と入出力方程式を 2 個の関数として別々に記述する手法を用いる効果である。

著者らは，この性質を利用してサンプル値系のシミュレーションプログラムからリアルタイム制御プログラムを最小限の変更で作成する方法を検討した。そして，制御系 CAD 言語 MATX を拡張し，リアルタイム制御プログラムを直接記述できるリアルタイム制御系 CAD 言語 RTMATX を開発した。

〈4·1〉　RTMATX のリアルタイム制御環境

```
Matrix z;             // 観測器の状態

Func void main()
{
  Real t0, tf, dt;
  Matrix Ts, Xs, Us, x0;

  x0 = [0 1 0 0]';    // 初期状態
  z  = Z(2,1);        // 観測器の初期状態

  {Ts, Xs, Us} =
  OdeHybrid(t0,tf,dt,x0,diff_eqs,link_eqs);
}
```

図 12　倒立振子のディジタル制御

Fig. 12.　Digital control of inverted pendulum

本節では，前章で述べた倒立振子のサンプル値系のシミュレーションプログラムからリアルタイム制御プログラムを作成する作業を通して RTMATX のリアルタイム制御環境について説明する。

まず，コントローラに関係する link_eqs() を変更し，図 13に示す on_task() を作成する。 加えるべき変更は，

```
Func void on_task()
{
  Matrix u, y, xh;

  y  = sensor();         // センサから入力
  xh = Chd*z + Dhd*y;    // 観測器による推定
  u  = - F*xh;           // 制御則計算
  actuator(u);           // 制御出力
  z = Ahd*z+Bhd*y+Jhd*u; // 観測器の更新
}
```

図 13　リアルタイム制御 (リアルタイム処理関数)

Fig. 13.　On-line function of real-time control

制御対象の出力をセンサーで取り込む関数 sensor()，制御入力をアクチュエータへ出力する関数 actuator() を追加することである。制御則を変更しても，これらの関数を変更する必要はないので，入出力ボードなどのハードウェア仕様をこれらの関数に隠蔽できる。なお，コンピュータの入出力ポートを直接アクセスする関数を使って sensor() や actuator() も RTMATX の関数として記述できる。

図 14に示す実験用のメイン関数関数では，パラメータの設定，サンプル点毎に呼び出す関数 (on_task()) の設定をする。 rtStart() を実行すると，on_task() がサンプル点毎に実行される。 off_task_loop() は変数の表示やユーザからのコマンド入力を担当する。

〈4·2〉　制御系設計統合環境

シミュレーション環境とリアルタイム制御環境の融合により実現される制御系設計統合環境における設計手順を図 15に，メイン関数の例を図 16に示す。この例では 10 行目から 19 行目が無限ループになっており，14 行目の

```
Func void experiment()
{
  para_init();         // パラメータ設定
  rtSetTask(on_task); // リアルタイム処理
  rtStart();           // リアルタイム開始
  off_task_loop();     // 画面表示など
  rtStop();            // リアルタイム停止
}
```

図 14　リアルタイム制御 (実験メイン関数)
Fig. 14.　Main function of real-time control

design() と 16 行目の experiment() を繰り返すことによりコントローラの修正と実験が繰り返し実行できる。

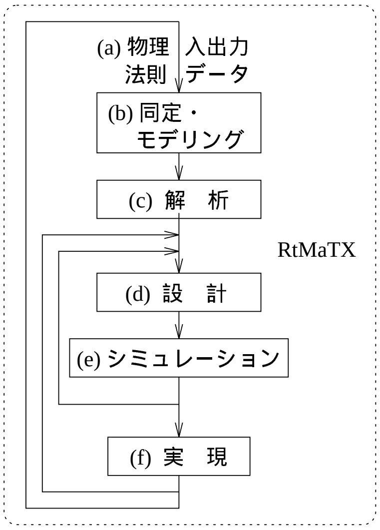

図 15　制御系設計統合環境における設計手順
Fig. 15.　Design procedure of control systems

統合環境で作成したリアルタイム制御プログラムとシミュレーションプログラムは同じライブラリを用いるので，次の長所がある。

- シミュレーション環境と実験環境でデータを共有でき，デバッグ作業が容易になる。
- シミュレーションプログラムと実験プログラムのユーザインターフェースを統一でき，作業の繰り返しが簡単になる。
- 図 17に示すように，制御系解析・設計ルーチンを組み込めるので，プログラムをコンパイルし直さずにコントローラの修正と実験を繰り返し実行できる。

特に，3 番目の長所は 1 章で紹介した他のアプローチでは得られない。

5.　おわりに

本論文では，シミュレーション用プログラム及びリアルタイム制御用プログラムを直接記述できるプログラミング言語を用い，制御系の解析・設計環境，シミュレーション環境，リアルタイム制御環境を融合する方法を提案した。本手法を用いれば，CAD で設計したコントローラをディジタルコントローラで実現する作業に要する時間が短縮されるだけでなく，実験プログラムに必要に応じて制御系解析・設計ルーチンを組み込めるので，プログラムをコンパイルし直さずにコントローラの修正と実験を繰り返し実行でき，制御系の性能を短時間で改善できる。提案する環境は特に高速プロトタイピングに有効である。プロトタイプを用いて指定された制御性能を満たす制御系を設計後，必要な部分だけを取り出し製品に組み込むことにより，ディジタルコントローラの実現に要する時間が短縮される。今後は，さまざまな実プラントに RTMATX を応用していく予定である。

```
Func void main()
{
  set_parameter();  // パラメータの設定

  main_menu = {"《倒立振子の安定化制御》",
      "モデリングと同定", "制 御 系 解 析",
      "制 御 系 設 計", "シミュレーション",
      "制　御　実　験", "終　了"};

  while (1) {
    switch (menu(main_menu)) {
      case 1: modelling();  break;
      case 2: analysis();   break;
      case 3: design();     break;
      case 4: simulation(); break;
      case 5: experiment(); break;
      case 6: return;
    }
  }
}
```

図 16　制御系設計統合環境のメイン関数
Fig. 16.　Main function of integrated environment

```
Func void design_pend(A,B,C,Q,R,Po,T)
  Matrix A,B,C,Q,R,Po;
  Real T;
{
  Matrix P,F;
  Matrix Ah,Bh,Ch,Dh,Jh,Ahd,Bhd;

  P = Riccati(A, B, Q, R);
  F = R~ * B' * P;
  {Ah,Bh,Ch,Dh,Jh} = Gopinath(A,B,C,Po);
  Bh = [Bh Jh];
  {Ahd, Bhd} = Discretize(Ah,Bh,T);
  return {F,Ahd,Bhd,Ch,Dh};
}
```

図 17　コントローラの設計
Fig. 17.　Function for design of controller

文　　献

(1) The Math Works Inc. *The Student Edition of MATLAB*. Prentice-Hall, Inc., Englewood Cliffs, New Jer-

sey 07632, 1992.

(2) Inc. Integrated Systems. *MATRIXx Core*. 3260 Jay Street Santa Clara, California 95054, USA, 8 edition, 1991.

(3) 赤掘一郎, 原辰次. オブジェクト指向に基づく制御系の解析・設計支援システム. 計測自動制御学会論文集, Vol. 24, No. 5, pp. 506–513, 1989.

(4) 高橋直樹, 藤田充, 斉藤制海, 阿部健一. 数式・数値処理による制御系 CAD. 電気学会論文誌 C, Vol. 111, No. 7, pp. 292–298, 1991.

(5) 古賀雅伸, 古田勝久. 数値処理と数式処理を融合した制御系 CAD 言語 MaTX. 計測自動制御学会論文集, Vol. 29, No. 10, pp. 1192–1198, 1993.

(6) The Math Works Inc. *SIMULINK – User's Guide*. 24 Prime Park Way, Natick, Mass. 01760-1500, USA, first edition, 1992.

(7) K. J. Astrom. Computer aided modeling, analysis and design of control systems – a perspective. *Control Systems Magazine*, pp. 4–16, 1983.

(8) O. Dalh. An interactive environment for real time implementation of control systems. *CACSD'91*, pp. 518–523, 1991.

(9) 池田雅夫. モデリングと制御器設計の不可分性. システム / 制御 / 情報, Vol. 37, No. 1, pp. 7–14, 1993.

(10) William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Veterling. *Numerical Recipes in C (The Art of Scientific Computing)*. Cambridge University Press, 1988.

(11) 荒木光彦. ディジタル制御理論の現状と動向. 計測と制御, Vol. 32, No. 2, pp. 97–100, 1993.

古賀　雅伸　(正員) 1965 年 9 月 21 日生。1993 年，東京工業大学理工学研究科博士課程修了 (工学博士)。東京工業大学工学部助手を経て現在，東京工業大学情報理工学研究科助手。制御系 CAD，線形制御理論とその応用などの研究に従事。日本ロボット学会，計測自動制御学会，IEEE などの会員。