'95 KACC (1995. 10. 23 ~ 25)

AN INTERACTIVE ENVIRONMENT FOR SIMULATION AND REAL-TIME IMPLEMENTATION OF CONTROL SYSTEMS

°Masanobu Koga*

Department of mechanical and Environmental Informatics
Graduate School of Information Science and Engineering
Tokyo Institute of Technology, 2-12-1, Oh-okayama, Meguro-ku, Tokyo 152, JAPAN
Tel: +81-3-5734-2328; Fax: +81-3-5734-2552; E-mail: koga@mei.titech.ac.jp

Abstracts An approach to efficient implementation of real-time control systems is presented in this paper. A compiler for translation of control algorithms is used in combination with a general program for real-time control. The compiler translates control algorithms, written for the simulation in a design language, to an implementation language. The translated algorithms are then automatically incorporated in the real-time control program.

Keywords Real-time control, Simulation, CADCS

1. INTRODUCTION

Current Computer Aided Design of Control System (CADCS) packages cover most spectrum of classical and modern control theory, and are being continuously updated with the latest advances in numerical control algorithms. In this way modern control engineers have access to the most recent powerful and robust control techniques to meet the still increasing demands from industry. However in real-time applications the execution speed of the numerical control algorithms has also to be considered. As the newly developed control algorithms tend to be more and more complex, the on-line implementation requires much more time for programming and consideration of real-time problems.

An approach to efficient implementation of real-time control systems is presented in this paper. A compiler for translation of control algorithms is used in combination with a general program for real-time control. The compiler translates control algorithms, written in a design language, to an implementation language, and generates code for real-time program. The resulting executable program have a number of interactive facilities such as matrix editor, plotting and textual display of all variables, and data logging. The design language is chosen as MATX [1, 2, 3, 4], and the implementation language is chosen as C. The system has been used in research and education, and has reduced the implementation time considerably, e.g. when developing new laboratory exercises, or when a control algorithm is tested in a laboratory experiment.

2. MATX

MATX [1, 2, 3, 4] is a high-performance programming language for general scientific and engineering numerical and symbolic computation. It is a type-oriented language and is equipped to recognize several data types such as integer, real number, complex number, string, polynomial, rational polynomial, matrix, array, index, and list. It has also inherited many features of C language such as variable declaration, control-flow, and style of function.

MATX provides not only command-line interpreter (**matx**) whose interfaces are similar to the use of MATLAB [5] but also compiler (**matc**) which accepts the same syntax as that of matx and outputs portable C language code. The user can extend the functionality of a program by realizing algorithms as functions in "mm-files". The interpreter enables us to test out the mm-files interactively file-by-file or line-by-line. If the mm-file implementation of an algorithm is not efficient enough, the examined mm-files are compiled into portable C language files with the compiler. Then those C files are compiled and linked up with the class library (**MATX-Lib**) [3] with a C compiler to generate the desired executable program. It is possible to call user C functions from MATX functions by linking C language routines to the executable program.

3. SIMULATION

How to write a simulation program in MATX is stated in this section. We consider the inverted pendulum shown in Fig. 1. The detailed explanation is not given due to limited pages of the paper. We design a LQ optimal controller and an observer for the linearized model, and descritize the observer with the sampling interval.

3.1 *Simulation of inverted pendulum*

List 1 shows the main function for a hybrid-time simulation of the pendulum. The term 'hybrid-time simulation' means simulation of continuous-time plants with digital controller. The function `Ode45HybridAuto()` takes six arguments: the initial time `t0`, the terminal time `tf`, sampling interval `dt` between which the constant input `u` is applied to the plant, initial state vector

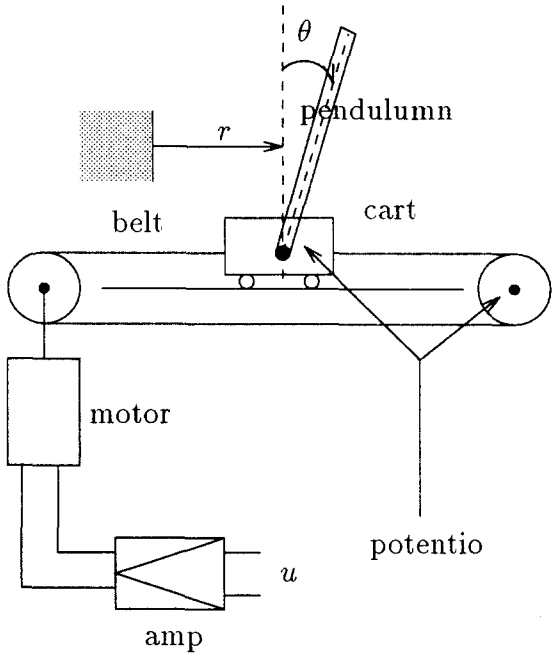

Figure 1: Inverted pendulum

x0, name (diff_eqs) of the function which calculates the derivative dx of state vector, and name (link_eqs) of function which calculates input u. The differential equation is integrated according to RKF45 algorithm automatically changing the step size to guarantee the prespecified error.

```
Real a32, a33, a34, a35, a42,a43,a44,a45;
Real b3, b4, alpha;
Matrix C, F, Ah, Bh, Ch, Dh, Jh, Xo;

Func void main()
{
  Real t0, tf, dt;
  Matrix TC, XC, UC, x0;
  void para_init(),diff_eqs(),link_eqs();

  para_init(); // parameter initialization
  t0 = 0.0; tf = 3.0; dt = 0.005;

  x0 = [0 30.0/180.0*PI 0 0]';
  Xo = Z(2,1); // state of observer

  {TC, XC, UC} =
     Ode45HybridAuto(t0, tf, dt, x0,
                     diff_eqs, link_eqs);
}
```

List 1: Main function of simulation

List 2 and List 3 show the function diff_eqs() and link_eqs() (in this example) which must have certain format. The format of the former specifies that the arguments list contain four variables and the latter specifies that the arguments list contain three variables.

```
Func void diff_eqs(DX, t, X, U)
  Matrix DX, X, U;
  Real t;
{
  Real x1, x2, x3, x4, u, c2, s2, dm;

  x1 = X(1); // position of cart
  x2 = X(2); // angle of pendulum
  x3 = X(3); // velocity of cart
  x4 = X(4); // angular velocity of pend.
  u = U(1);  // input

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

  DX = Z(4,1);
  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;
}
```

List 2: Function for dynamic equation

In the function diff_eqs(), the first variable dx is derivative of state (return variable) and has the same dimension as x, the second variable t is a real number which represents time, the third variable x is a vector of the state values, and the fourth variable u is the input to the system. In the function link_eqs(), the first variable u is the input (return variable), the second variable t is a real number which represents time, the third variable x is a vector of the state values.

```
Func void link_eqs(U, t, X)
 Matrix U, X;
 Real t;
{
 Matrix Y, Xh;

 Y  = C * X;           // output equation
 Xh = Ch*Xo + Dh*Y;    // state estimation
 U  = - F*Xh;          // state feedback
 Xo = Ah*Xo + Bh*Y + Jh*U; // observer
}
```

List 3: Function for relation between signals

4. REAL-TIME CONTROL

4.1 *Real-time program*

A real-time program in MATX consists of three functions: main function main(), on-line function on_task(), and off-line function off_task_loop().

List 4 shows an example of the main function. Once the function rtStart() is called, on_task() is called every sampling period.

```
Func void main()
{
  para_init();        // initialize param.
  var_init();         // initialize var.
  machine_ready();    // prepare machine

  rtSetClock(stime);  // sampling interval
  rtSetTask(on_task); // set on-line func
  rtSetBreak(break_task); // set break func

  rtStart();          // start on-line func
  off_task_loop();    // off-line task
  rtStop();           // stop on-line func

  machine_stop();     // stop machine
}
```

List 4: Main function of real-time control

The on-line function for the control of double-cart system is shown in List 5. The function `sensor()` returns measured outputs and `actuator()` gives the torque of the motor.

```
Func void on_task()
{
  Real r;
  Matrix yy, y, xf, u;

  // reference signal
  r=m1*sin(w1*stime*t)+m2*sin(w2*stime*t);
  t++;

  yy = sensor();          // sensor output
  y = [[  r  - yy(1)]     // measured outp1
       [y(1) - yy(2)]]; // measured outp2
  xf = A * x + B * y;     // controller state
  u =  C * x + D * y;     // control input
  x = xf;                 // save controller state
  actuator(u);            // output to actuator
}
```

List 5: On-line function of real-time control

List 6 shows an example of off-line function. The function `rtIsTimeOut()` checks if the on-line function finished in the sampling interval.

```
Func void off_task_loop()
{
  Integer c;

  while (1) {
    if (rtIsTimeOut()) {
      warning("Time Out\n");
      break;
    }

    printf("y1 = %f, y2 = %f", y(1), y(2));
    if (kbhit() && getch() == 0x1b) break;
  }
}
```

List 6: Off-line function of real-time control

4.2 *Memory management*

In real-time applications the execution speed of the numerical control algorithms are to be considered. As the newly developed control algorithms tend to be more complex, the on-line implementation requires much more time for programming. An approach to efficient implementation of real-time control systems is to use a compiler which translates control algorithms, written in a design language, to an implementation language. And the design language should support useful data types, such as matrix, polynomial, and rational polynomial. Since these data types require dynamic memory allocation, we have to consider the efficient memory management. The on-line memory management in MATX is stated in this subsection.

The all program in MATX are build upon MaTX-Lib which is a collection of useful class libraries. Since the all data types use the same memory management, we treat matrix class here.

The memory allocation of matrix class is managed according to the state transition of the object shown in Fig. 2.

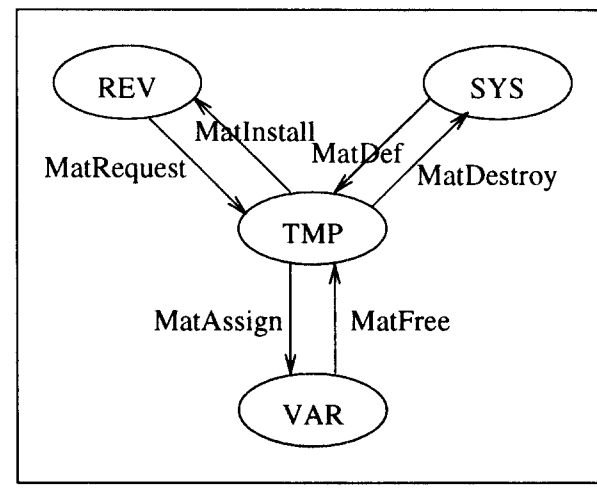

Figure 2: Memory management in MATX

The matrix object takes four states:

1. `SYS` (Maintained in the operating system)
2. `TMP` (Allocated by `malloc()`)
3. `VAR` (Assigned to the variable)
4. `REV` (Maintained in the table of free object)

The state changes by the following function calls.

- `MaDef()` gets the memory from the operating system by calling `malloc()`.
- `MatDestroy()` returns the memory to the operating system by calling `free()`.
- `MatAssign()` assigns a matrix to a variable. The state of the matrix changes from `TMP` to `VAR`. The matrix in `VAR` is not effected by `MatTmpUndef()`.
- `MatFree()` changes the state of the matrix from `VAR` to `TMP`. This means that the variable is released.

- `MatInstall()` installs a matrix object to the table of free object. The state of the matrix changes from `TMP` to `REV`.
- `MatRequest()` gets the memory from the table of free object. The state of the matrix changes from `REV` to `TMP`. Not that `MatRequest()` takes less time than `MatDef()`.

When the memory for a matrix object is required, a suitable size matrix in the table of free object is used. If there is no suitable size matrix object in the table, the allocation function `MatDef()` calls `malloc()` to get more memory from the operating system. This memory allocation strategy reduces the function calls of `malloc()` and `free()` and makes the execution speed fast. It is especially efficient for the program, in which the same function is repeatedly called, such as simulation and real-time control.

5. EXAMPLE

5.1 *Inverted pendulum*

The implementation of real-time control of the inverted pendulum, whose simulation program is shown in section 3, is very easy. We only change the function `link_eqs()` in the simulation program to `on_task()` as follows.

```
Func void on_task()
{
  Matrix Y, Xh;

  Y  = sensor();          // measured output
  Xh = Ch*Xo + Dh*Y;      // state estimation
  U  = - F*Xh;            // state feedback
  actuator(U);            // control input
  Xo = Ah*Xo + Bh*Y + Jh*U; // observer
}
```

List 7: Real-time control of inverted pendulum

5.2 *Double-cart system*

We compare the execution speed of the real-time control programs written in MATX and C. The MATX program and C program are shown in List 5 and List 8, respectively. The IBM-PC/AT computer with pentium 60MHz is used for experiment. The shortest sampling intervals by both programs are shown in Table 1.

6. CONCLUDING REMARKS

An approach to efficient implementation of real-time control systems was presented. The compiler translates control algorithms, written in a design language, in which we can use matrix, polynomial, and so on, to an implementation language. We showed an illustrative example, written in the design language, whose execution speed is significantly faster than that of the program written in C.

Table 1: The shortest sampling interval of RTC

C program	MATX program
1.6 [ms]	1.1 [ms]

```
void on_task()
{
  int i, j;
  double r, yy[2], y[2], xf[8], u[2];

  r=m1*sin(w1*stime*t)+m2*sin(w2*stime*t);
  t++;
  sensor(yy);
  y[0] =   r  - yy[0];
  y[1] = y[0] - yy[1];

  /* xf = A*x + B*y */
  for (i = 0; i < 8; i++) {
    xf[i] = 0.0;
    for (j = 0; j < 8; j++)
      xf[i] += A[i][j]*x[j];
    xf[i] += B[i][0]*y[0] + B[i][1]*y[1];
  }

  /* u =  C*x + D*y */
  u[0] = u[1] = 0.0;
  for (i = 0; i < 8; i++) {
    u[0] += C[0][i]*x[i];
    u[1] += C[1][i]*x[i];
  }
  u[0] += D[0][0]*y[0] + D[0][1]*y[1];
  u[1] += D[1][0]*y[0] + D[1][1]*y[1];

  for (i = 0; i < 8; i++) x[i] = xf[i];
  actuator(u);
}
```

List 8: Real-time control of double-cart system

REFERENCES

[1] Masanobu Koga and Katsuhisa Furuta. MaTX : A high-performance interactive software package for scientific and engineering computation. *Proc. of CADCS'91, Swansea, U.K.*, pages 39-44, 1991.

[2] Masanobu Koga and Katsuhisa Furuta. MaTX : A high-performance programming language (interpreter and compiler) for scientific and engineering computation. *Proc. of CACSD'92, Napa, U.S.A.*, pages 15-22, 1992.

[3] Masanobu Koga and Katsuhisa Furuta. Programming language MaTX for scientific and engineering computation. In Derek A. Linkens, editor, *CAD for Control Systems*, chapter 12, pages 287-317. Marcel Dekker, Inc., July 1993.

[4] Masanobu Koga, Mitsuji Sampei, and Katsuhisa Furuta. A compiler of matlab to MaTX: Compiling and linking of m-files to an executable program. *Proc. of CACSD'94, Tucson, Arizona, U.S.A.*, pages 137-142, 1994.

[5] C. Moler, J. Littel, and S. Bangert. *PC-MATLAB - User's Guide*. N. Main St., Sherborn, MA 01770, USA, 3.1 edition, 1987.