MATX/RTMATX: A Freeware for Integrated CACSD

Masanobu Koga

Department of Mechanical and Environmental Informatics
Tokyo Institute of Technology
2-12-1, Oh-okayama, Meguro-ku, Tokyo 152-8552, JAPAN
Tel: +81-3-5734-2328; Fax: +81-3-5734-2328
E-mail: koga@mei.titech.ac.jp

1 Introduction

The purpose of this paper is to give an overview of a cost-efficient integrated CACSD environment MATX/RTMATX. The software supports not only the analysis of control systems, and the design of controllers, but also the real-time implementation of controllers.

MATX/RTMATX is distributed as a free software and is used in many universities and several companies mainly in Japan. This session focuses on the applications in some fields, such as robot motion control, visual simulation, mathematical modeling with symbolic manipulation, economics, and education in control engineering.

2 MATX and RTMATX

MATX [1] is a high-performance programming language for scientific and engineering 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. MATX provides not only command-line interpreter (matx) whose interfaces are similar to the use of matlab [2] but also compiler (matc).

During the last several years, there has been an increased interest in automating the process of implementing digital controllers. It aims to have a better consistency between the designed and implemented controllers. RTMATX enables us to do analysis of plant, synthesis of controller, modeling of plant, and real-time implementation for the experiment, all in one software environment. Comparing the conventional method which generates C code from block diagram [3, 4, 5], it is possible to repeat tuning of controllers and experiment without re-compile, since the proposed method enables us to include routines for design of control systems to the implemented programs.

2.1 Program developing process

Figure 1 and Fig. 2 show the program developing process in Matlab and MATX, respectively. In the program developing process in Matlab, the user usually realizes the algorithms in an m-file (`test.m`) according to the requirement (`specs`). The m-file is tested several times until it satisfies the requirement. It is possible to call the user's C or Fortran subroutines (`ctest.mtex`), from Matlab which are generated from the source codes (`ctest.c`) by the C- or Fortran- compiler.

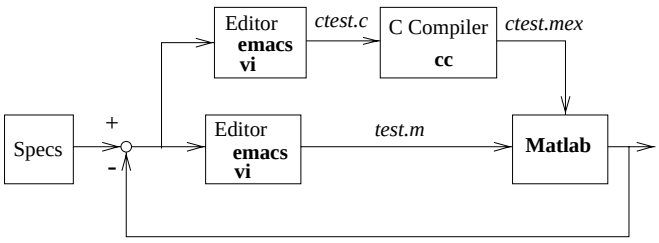

Figure 1: Program Developing Process in Matlab

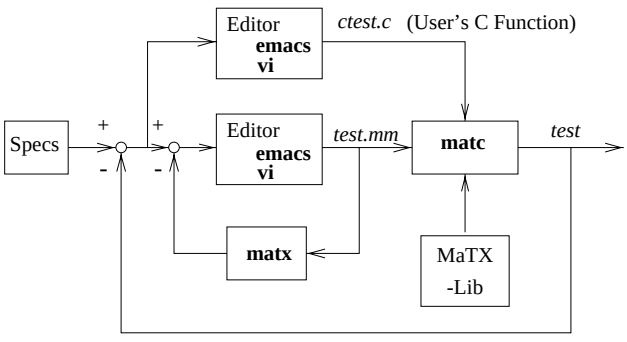

Figure 2: Program Developing Process in MATX

In the program developing process in MATX, the user makes several procedures (mm-files) using a favorite editor (vi, emacs, or so) according to the requirement (specs). The interpreter (matx) 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 in the interpreter environment, the mm-files are compiled to the executable program (`test`) by the compiler (`matc`). It is possible to call the C functions from the MATX program by linking C code files (`ctest.c`). Finally, the executable file is executed to check whether all requirements are satisfied.

The compiler generates the portable C code files if it is invoked with the option `-mm`, and it generates the object file if it is invoked with the option `-c`. These options are useful when `matc` is invoked by `make` in `makefile`. Figure 3 shows an example of the generation of the executable program. On the 1st step, the C files `sub1.c` and `sub2.c` are generated from `sub1.mm` and `sub2.mm`, respectively. On the 2nd and 3rd step, the object files `sub3.o` and `sub1.o` are generated from `sub3.mm` and `sub1.c`, respectively. The 4th step generates the executable file `main` from `main.mm`, `sub1.o`, `sub2.c`, and `sub3.o`.

```
% matc -mm sub1.mm sub2.mm
% matc -c sub3.mm
% matc -c sub1.c
% matc main.mm sub1.o sub2.c sub3.o
```

Figure 3: Steps to generate executable program

2.2 Running environment

At the moment, MATX is checked to run in the environment shown in Table 1, and RTMATX is checked to run in the environment shown in Table 2.

Table 1: MATX Running environment

Computer	OS & Compiler
CRAY C916	Cray UNICOS
NEWS5000X	NEWS-OS 4.2.1R
JCC PowerPC	JCC_BSD+ 1.0
HP 9000/755	HP-UX
DEC (Alpha)	Digital-Unix V3.2G
SGI (R10000)	IRIX 6.4 System V R.4
Mips RC6280	UMIPS4.52C
Sun(Sparc)	SunOS4.1.x, Solaris 2.(5|6).x
Sun(X86)	Solaris 2.(5|6).x
IBM-PC/AT	Linux (1.2.x, 2.0.x)
IBM-PC/AT	FreeBSD 2.1.x, FreeBSD 2.2.x
IBM-PC/AT	BSD/OS 2.0
IBM-PC/AT	Windows 95/NT, (Visual C++)
IBM-PC/AT	Windows 95/NT, (DJGPP)
IBM-PC/AT	DOS, (DJGPP)

RTMATX for Windows 95 with Visual C++ uses thread-API and multimedia-timer-API to implement the real-time control facility. RTMATX for Windows 95 and DOS with DJGPP uses DPMI (Dos Protected Mode Interface) service to implement the protected-mode interruption. RTMATX for DOS with Borland C++ generates the real-mode DOS program.

Table 2: RTMATX Running environment

Computer	OS & Compiler
IBM-PC/AT	Windows 95, (Visual C++)
IBM-PC/AT	Windows 95, (DJGPP)
IBM-PC/AT	DOS, (DJGPP)
IBM-PC/AT	DOS, (Borland C++)

3 Interpreter and Compiler

MATX provides not only the interpreter but also the compiler. The users can extend the functionality of a program by implementing algorithms as functions in mm-files. It is very easy to call the user C functions from MATX programs by linking C code files to the mm-files. This allows us to utilize huge pre-exist C routines and speed up the rate of computation and improve the efficiency of memory usage.

3.1 Script file and library module

One mm-file may be used both as a script file in the interpreter environment and a library module for the compiler. The interpreter calls the C preprocessor before parsing the file with defining as '`__MATX__=1`', while the compiler calls the C preprocessor with defining as '`__MATC__=1`'. It enables the user to make an mm-file which works in the different way for the interpreter and the compiler without modification.

If the user wants to run an mm-file as a script file, e.g. for debugging, the following example in Fig. 4 shows a method. If the mm-file is executed as a script file in the interpreter, the variables A, B and C are defined as a global variables which are accessible from the command-line after the execution. If the mm-file is compiled to a library module, those variables are defined as local variables which exist only in the function.

3.2 Executable script file

A shell script in UNIX operating system is a sequence of shell commands stuffed into a text file. The file is made executable by turning on the execute bit (via `chmod +x filename`) and then the name of the file is typed at a shell prompt. Similarly, a MATX program is a bunch of MATX statements and definitions thrown into a file. The user then turn on the execute bit and type the name of the file at a shell prompt. However, the file has to indicate that this is a MATX program

```
#if __MATC__
Func void afo()
{
   Matrix A,B,C;
#endif

   A = [[1 2][3 4]];
   B = [[5 6][7 8]];
   C = A + B;
   print A,B,C;

#if __MATC__
}
#endif
```

Figure 4: Script and library module

and not a shell program, so we need an additional step. In the standard UNIX-like operating system, this step involves placing the line

```
#! /usr/local/bin/matx
```

as the first line of the file. Figure 5 shows how to make an executable mm-file script which calculate the addition of two matrices.

```
#! /usr/local/bin/matx

Func void main()
{
  Matrix A,B,C;

  read A,B;
  C = A + B;
  print A,B,C;
}

main();
```

```
% emacs addmat.mm
% chmod a+x addmat.mm
% addmat
```

Figure 5: Executable Script file

3.3 Independent application

The compiler matc generates the executable program which works independently of MATX environment. The arguments to the command can be accessed as strings from the mm-file. Figure 6 shows a program which shows the name of the command and the result of the addition of the arguments.

```
Func void main(argc, argv)
  Integer argc;
  List argv;
{
  String name, arg1, arg2;
  Integer a, b;

  {name, arg1, arg2} = argv;
  print "Command name is", name, "\n";
  a = Integer(arg1);
  b = Integer(arg2);
  print a + b;
}
```

Figure 6: Command line arguments

4 Integration of Design and Real-time Implementation

MATX/RTMATX supports not only off-line design, but also on-line design. In other words, it is possible to undertake an analysis of a plant, synthesis of a controller, modeling of the plant, and real-time implementation for experimental use, all in one software environment. By utilizing this software, it is possible to repeat the design of control-system cycle efficiently, to achieve good performance of the closed-loop system.

The typical procedure for the design of control systems supported by the conventional CACSD software is shown in Fig 7. First, the plant, whose mathematical model has been determined by input-output data or a physical formula, is analyzed. Then, the synthesis of the controller, simulations, and experiments are repeated until the design specification is satisfied. After the performance of the closed-loop system has been verified by simulation, the program for the experiment is implemented in another software environment. It is apparent that the design of the control system and the real-time implementation are separated in this procedure.

MATX/RTMATX provides a software environment to deal with a procedure for the design of control systems, in which the design specification is satisfied by iterating the analysis of the plant, the synthesis of the controller, simulations, experiments, and the modeling of the plant. MATX/RTMATX comprehensively supports the whole design cycle. Figure 8 shows the proposed design cycle for control systems.

4.1 Simulation and real-time implementation

This subsection deals with an examples to illustrate the facility of MATX/RTMATX for simulation and real-time implementation. The first plant is the well-known inverted pendulum, shown in Fig. 9. When the state

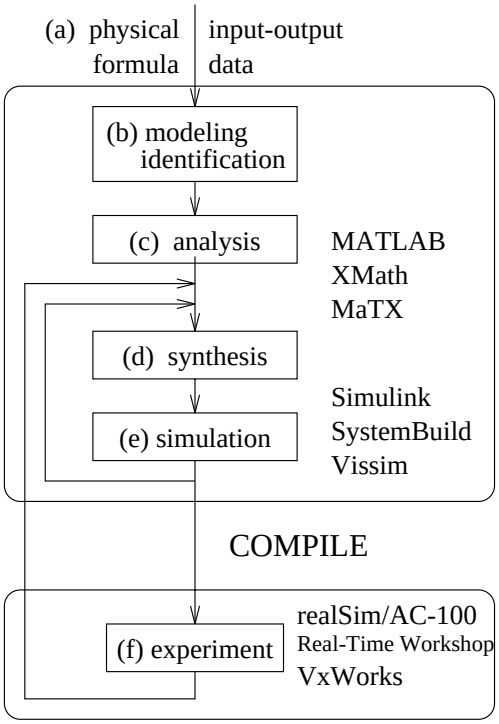

Figure 7: Conventional design cycle of control systems

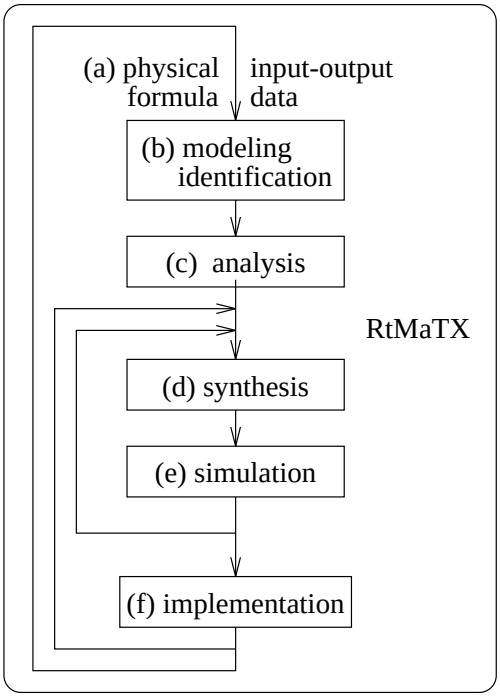

Figure 8: New design cycle of control systems

of the system is chosen as

$$x^T = \left[\begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array}\right] = \left[\begin{array}{cccc} r & \theta & \dot{r} & \dot{\theta} \end{array}\right],$$

then the state equation is described as

$$\frac{dx}{dt} = \left[\begin{array}{l} x_3 \\ x_4 \\ (1+\alpha\sin^2 x_2)^{-1}\{a_{32}\sin x_2\cos x_2 + a_{33}x_3 \\ (1+\alpha\sin^2 x_2)^{-1}\{a_{42}\sin x_2 + a_{43}\cos x_2 x_3 \end{array}\right.$$

$$\left.\begin{array}{l} + a_{34}\cos x_2 x_4 + a_{35}\sin x_2 x_4^2 + b_3 u\} \\ + a_{44}x_4 + a_{45}\sin x_2\cos x_2 x_4^2 + b_4\cos x_2 u\} \end{array}\right],$$

where a_{ij}, b_i, and α are parameters of the plant. Next, design an LQ state feedback control law and an observer for the linearized model of the plant. The observer is discretized for digital control such that

$$\begin{aligned} z[k+1] &= \hat{A}_d z[k] + \hat{B}_d y[k] + \hat{J}_d u[k] \\ u[k] &= -F\hat{x}[k] \\ \hat{x}[k] &= \hat{C} z[k] + \hat{D} y[k]. \end{aligned}$$

4.1.1 Simulation: The simulation program consists of three functions, `main()`, `diff_eqs()`, and `link_eqs()`. Function `diff_eqs()` is a function that calculates the derivative of state vector $\dot{x}$, and `link_eqs()` is a function that calculates the control inputs u. In `main()` the differential equation is integrated according to the RKF45 algorithm, automatically changing the step size to guarantee the specified computational error. The listing of `diff_eqs()` and `link_eqs()` are shown in Figures 10 and 11. Note that the program can be written in almost the same way as the mathematical notation. See [6] for the detailed description.

4.1.2 Real-time implementation: The real-time implementation program consists of three functions, the main function, `main()`, the on-line function, `on_task()`, and the off-line function, `off_task()`. The function `on_task()` is a function for the calculation of control inputs, and `off_task()` is the user-interface function for showing and changing parameters. Once the real-time control starts, `on_task()` is called every sampling period. The listing of `on_task()` is shown in Fig. 12. The function `sensor()` returns the output of the the plant, as measured by the sensors, and `actuator()` operates the actuator. Since the variables, `Ah`, `Bh`, `Ch`, `Dh`, `Jh`, `F` are declared as real-time variables, they can be changed while the real-time control is running. Note that the difference between the functions `on_task()` and `link_eqs()` in the simulation program consists of two lines. Therefore, it is possible to obtain the on-line function for the real-time experiment from the simulation program with a two-line change. This

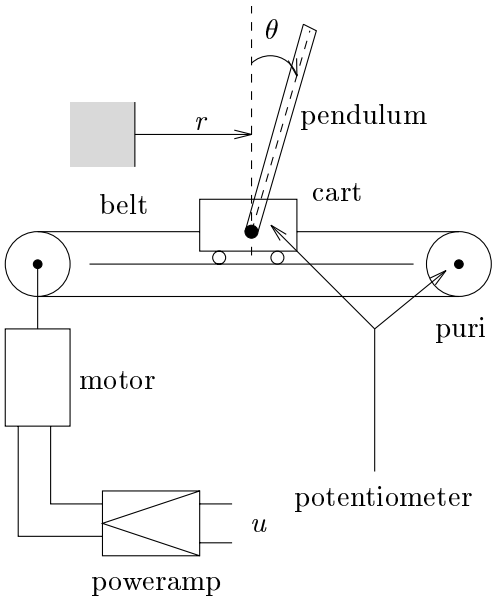

Figure 9: Inverted pendulum

is one of the most useful features of RTMATX.

4.2 Toolbox for analysis and design

At the moment, MATX provides the following toolbox for the analysis and design.

- Matrix operation toolbox
- Signal processing toolbox
- Graph drawing toolbox
- Control system toolbox
- ...

Each toolbox contains a set of mm-files for the interpreter and a library for the compiler. The algorithm is implemented as the functions in each mm-file. The interpreter (matx) loads the mm-files if they are required from the user program or the command-line, while the compiler (matc) links the library with the user program.

4.3 Memory management for real-time implementation

If people write a program with a matrix notation like Matlab, the program is easy to read. However, the size of memory required for the calculations is not determined until the execution time. This is why functions like `malloc()` are necessary. These functions are slow, since they are general-purpose memory-management functions. Therefore methods to improve the speed of computation should be considered.

MATX has its own memory-management mechanism to reduce the number of calls to the memory-allocation function [7]. In particular, no memory-allocation function is called when real-time control is executed [8].

```
Func void diff_eqs(dx, t, x, u)
  Matrix dx, x, u;
  Real t;
{
  Real c2, s2, dm;

  c2 = cos(x(2));
  s2 = sin(x(2));
  dm = 1 + alpha*s2^2;
  dx =
  [[                x(3)              ]
   [                x(4)              ]
   [dm~*(a32*s2*c2 + a33*x(3)
    +a34*c2*x(4)+a35*s2*x(4)^2+b3*u(1)]
   [dm~*(a42*s2 + a43*c2*x(3)+a44*x(4)
    +a45*s2*c2*x(4)^2+b4*c2*u(1)    ]];
}
```

Figure 10: Function for state equation

```
Matrix Ah, Bh, Ch, Dh, Jh;

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
 z  = Ah*z + Bh*y + Jh*u; // observer
}
```

Figure 11: Function for control input

5 Interface to Other Softwares

5.1 Process communication

One realization of coupling the other software with MATX is to use a pipe, which can be used in standard UNIX operating system. The function `popen()` takes the name of the other software as the argument and returns a process descriptor with which MATX program can communicate with the software by using the file handling functions, such as `fprintf()` and `fscanf()`.

Figure 13 shows the program which communicates with the drawing tool gnuplot to draw a $\sin(x)$ curve. The graph toolbox communicates with `gnuplot` in the similar fashion.

```
Realtime Matrix Ah,Bh,Ch,Dh,Jh,F;

Func void on_task()
{
  Matrix y, xh;

  y  = sensor();
  xh = Ch*z+Dh*Y;  // state estimation
  u  = - F*xh;     // state feedback
  z = Ah*z+Bh*y+Jh*u; // observer
  actuator(u);
}
```

Figure 12: Function for real-time control

```
if ((pid = popen("gnuplot")) < 0) {
    error("Can't open %s", "gnuplot");
}

fprintf(pid, "plot sin(x)\n");
pause;
fprintf(pid, "quit\n");
pclose(pid);
```

Figure 13: Process communication with pipe

5.2 Link C code to mm-file

It is possible to call the C functions from MATX programs by linking those files to the mm-files. This allows us to utilize huge pre-exist C routines and the results generated by the other softwares.

The symbolic manipulation softwares, such as Mathematica [9] and Maple [10], are very useful tools for the physical and mathematical modeling. The user can save the calculated model as the C code file, which is linked with the mm-file. The companion paper [11] deals with this method to generate the model for the simulation.

Even if the source code is not available, the libraries for the C program can be linked with the MATX program. The functions contained in the library can be called from the MATX program either directory or through the simple interface function. The companion paper [12] deals with this method to make the three dimensional graph (3DG) and the graphical user interface (GUI) by linking OpenGL library[13].

5.3 Data conversion

MATX provides the lower level file access functions such as fread() and fwrite() which works similar as those of C functions. These functions can be used for the data-conversion between the other softwares.

For example, `matlab_read()` and `matlab_write()` reads and writes the matlab mat-format (V4) data. In Fig. 14, the 1st statement writes four matrices as `double` precision to the file `qq.mat`, and the 3rd statement reads four matrices from the file `rr.mat` which was generated by matlab.

```
matlab_write("qq.mat",{A,B,C,D},"double");
pause "Run matlab to generate rr.mat";
{A2,B2,C2,D2} = matlab_read("rr.mat");
```

Figure 14: Data conversion with matlab

References

[1] 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.

[2] Inc. The Math Works. *MATLAB User's Guide*. The Math Works, Inc., 24 Prime Park Way, Natick, Mass. 01760-1500, USA, 1992.

[3] Inc. Integrated Systems. *MATRIXx Core*. 3260 Jay Street Santa Clara, California 95054, USA, 8 edition, 1991.

[4] Inc. The Math Works. *SIMULINK User's Guide*. 24 Prime Park Way, Natick, Mass. 01760-1500, USA, first edition, 1992.

[5] Ola Dahl. An interactive environment for real time implementation of control system. *CADCD '91,*, pages 518–523, 1991.

[6] Masanobu Koga. An interactive environment for simulation and real-time implementation of control systems. *Proc. of KACC'95, Seoul, Korea*, pages 336–339, 1995.

[7] Masanobu Koga, Hiroaki Toriumi, and Mitsuji Sampei. Real-time cad of control systems achieving cooperation of modeling and design of controllers. *Proc. of CACSD'96, Dearborn, Michigan, U.S.A.*, pages 457–462, 1996.

[8] Masanobu Koga, Hiroaki Toriumi, and Mitsuji Sampei. An integrated software environment for design and real-time implementation of control systems. *11th IFAC Symposium on System Identification*, pages 1603–1609, 1997.

[9] Stephen Wolfram. *Mathematica*. Addison-Wesley Publishing Company, Inc., 1991.

[10] K.M.Heal, M.L.Hansen, and K.M.Rickard. *Maple V – Learning Guide*. Springer, 1996.

[11] Tasuku Hoshino and Katsuhisa Furuta. Mixed numeric/symbolic manipulation of equations using matx and mathemaitca. *CACSD'99*, pages ??–??, 1999.

[12] Kenichiro Nonaka. VRSC: Visual robot simulation and control with rtmatx. *CACSD'99*, pages ??–??, 1999.

[13] Ron Fosner. *OpenGL Programming for Windows 95 and Windows NT*. Addison-Wesley Developers Press, 1997.