A DIGITAL CONTROL ALGORITHM FOR MAGNETIC SUSPENSION SYSTEMS*

Thomas C. Britton
Lockheed Martin Engineering & Sciences
Hampton, VA

ABSTRACT

An ongoing program exists to investigate and develop magnetic suspension technologies and modelling techniques at NASA Langley Research Center. Presently, there is a laboratory-scale large air-gap suspension system capable of five degree-of-freedom (DOF) control that is operational and a six DOF system that is under development. Those systems levitate a cylindrical element containing a permanent magnet core above a planar array of electromagnets, which are used for levitation and control purposes. In order to evaluate various control approaches with those systems, the Generic Real-Time State-Space Controller (GRTSSC) software package was developed. That control software package allows the user to implement multiple control methods and allows for varied input/output commands. The development of the control algorithm is presented. The desired functionality of the software is discussed, including the ability to inject noise on sensor inputs and/or actuator outputs. Various limitations, common issues, and trade-offs are discussed including data format precision; the drawbacks of using either Direct Memory Access (DMA), interrupts, or program control techniques for data acquisition; and platform dependent concerns related to the portability of the software, such as memory addressing formats. Efforts to minimize overall controller loop-rate and a comparison of achievable controller sample rates are discussed. The implementation of a modular code structure is presented. The format for the controller input data file and the noise information file is presented. Controller input vector information is available for post-processing by mathematical analysis software such as MATLAB[1].

* Work done on contract to NASA Langley Research Center at Lockheed Martin Engineering & Sciences, NAS1-19000.

[1] Use of names of products in this report does not constitute an official endorsement of such products, either expressed or implied, by the National Aeronautics and Space Administration or Lockheed Martin Engineering & Sciences.

INTRODUCTION

In order to investigate magnetic suspension technologies and modelling techniques, a laboratory-scale large air-gap magnetic suspension system was developed at NASA-Langley Research Center. The laboratory system, the Large Angle Magnetic Suspension Test Fixture (LAMSTF), consists of an array of five electromagnets arranged in a circular planar configuration and a suspended element that is magnetized along the long axis of the cylinder [1,2], see Figure 1. The planar array is used to provide both levitation and control of the suspended element in five degrees-of-freedom. A six degree-of-freedom (DOF) suspension system is under development. Various control approaches [3,4,5,6] have been investigated with the LAMSTF suspension system, each requiring individual control algorithms to be developed. In order to simplify the overall control system design process, the Generic Real-Time State-Space Controller (GRTSSC) software package was developed. The software package allows for the implementation of varied control algorithms without the need for code development for each algorithm, and was achieved by forming the desired control method into a standardized state-space format. By using that format, multiple types of control methods may be loaded and investigated in sequence. The developed state-space control algorithm is able to be paced at varied rates with the use of an interrupt service routine (ISR). Due to the nature of the state-space based control algorithm, the GRTSSC has other applications besides the control of magnetic suspension systems.

SOFTWARE DESIGN CRITERIA

An interrupt-driven control system is discussed in [7], but is limited in use due to the control algorithm that was implemented. Other approaches have used high-powered DSP processor chips to execute the control law [8], with custom hardware in some situations to handle the data acquisition process [9]. The use of those powerful DSP processors allow high controller sample rates to be achieved, but at a relatively high cost. Certain requirements had to be met in the development process of the GRTSSC. Those requirements were primarily driven by the need for versatility of the software and ease of use. Based on the system under control, the software package needed to be configurable by the user. Certain input and output conditions were required of the control algorithm, and other requirements of the software package were due to the desired use of the GRTSSC as an investigative tool. Thus, the emphasis of this system was not driven by the type of the system hardware, but by the design and the method of implementation of the control system software.

User Configurability

The most important requirement of the software package was to allow the user to configure the GRTSSC to handle different types and sizes of systems, by specifying the number of sensor inputs, actuator outputs, and the number of degrees-of-freedom of the system under control. This was required due to the differences between the LAMSTF and the six DOF suspension system, which is being developed. The LAMSTF consists of a state-space plant model with a ten-element state vector and requires a control system capable of handling five sensor inputs and five actuator outputs. Also, the six DOF suspension system consists of a state-space plant model with a twelve-element state vector and requires a controller that can handle a larger number of sensor inputs and actuator outputs.

Control Algorithm Functionality

The control algorithm was required to be able to handle the injection of noise on sensor inputs and actuator outputs, in addition to being able to apply sensor and actuator reference commands. The noise injection is handled off-line by initializing a data structure from a user supplied data file containing the noise information. The command reference inputs for the sensors and actuators can be calculated either on-line or supplied in a data file similiar in format to the noise data file. The ability to individually select the desired command reference channel was required in order to be able to preload setpoint data for all reference channels and, then, to be able to investigate each channel individually. Another requirement of the control algorithm was the ability to save pertinent data for post-processing with mathematical analysis software.

Software Package Functionality

In order to easily evaluate controllers with different design characteristics, the ability to select from multiple control methods that are loaded during the initialization sequence was required. The GRTSSC software package must have the ability to dynamically switch between control systems during suspension. The control algorithm should be paced by a timed interrupt, which would imply that each controller that was loaded could execute at different sample rates. It was also required that the software package provide a concise summary of commands and the status of the control system to the user. Another requirement of the GRTSSC was the indepen-

dent pacing of the data acquisition system. This was to help minimize the dependence of the software package upon a particular data acquisition system. A final requirement of the GRTSSC was for the source code to be portable between platforms.

LIMITATIONS AND TRADE-OFFS

To achieve the best code efficiency possible, a variety of areas were studied. Those included the precision of the data variables, the differences between compilers, the machine architectures, the methods of acquiring the sensor input data, and the general code optimizations. Some advantages and disadvantages exist for each of these areas.

Data Precision

The precision of the data variables that are used to process the control algorithm can affect both the speed and the amount of memory usage of the algorithm. Typically, machines perform memory access at a faster rate by using single precision data, thus increasing the speed of memory intensive routines. The state-space representation of the control system was represented by floating-point memory pointers. That allows dynamic memory allocation for each controller, thereby maximizing the efficiency of the memory usage. Single precision floating-point numbers are used, resulting in approximately a fifty percent reduction in the memory usage that was required for data and variable storage.

Machine Architectures and Compilers

The original version of the GRTSSC software package was developed on an Amiga A3000T system by Commodore Business Machines, which was based on the MC68030 microprocessor with MC68881 coprocessor support. The SAS/C compiler [10], version 6.55, that was used produced faster code with the large memory model and used double precision data. However, the current version of the software was developed on a standard 486 DX2/66 class PC by using the Microsoft C compiler [11], version 6.0. Faster code was produced by using the large memory model and single precision data. It is widely known that different compilers can produce significantly different code, both in speed and size of the executable file. For example, in a comparison

of version 6.0 of the Microsoft C compiler and the freeware GNU/C compiler, experimental results with other software suggest that a 20 percent increase in speed is achievable with the use of the GNU/C compiler.

A significant advantage that the A3000T platform possessed was the flat memory model inherent in the architecture and the true pre-emptive multitasking operating system. Also, the data acquisition subsystem was executed by a plug-in 386-PC CPU board, independent of the main processor of the A3000T. Acquired data was transmitted through dual-port random access memory (RAM), which existed on the 386-CPU board. That relieved the main processor of the A3000T of the task of data acquisition, and thus it could be dedicated to the control algorithm. The flat memory model allowed data storage space to be allocated up to the total amount of free RAM that remained. This allowed the code to save controller input, output, and state vectors for step responses on each degree of freedom of the suspension system. Data runs long enough to be able to perform system identification techniques on the linear plant model were achievable. However, sample rates on the order of 1000 Hz will be required for initial control designs for the six DOF suspension system. The A3000T using an accelerator board with a 40 MHz MC68040 was able to achieve a sample rate of approximately 500 Hz.

The PC class machine proved quite capable of generating the required controller sample rates for the six DOF suspension system. A comparison of achievable sample rates is shown in Figure 2. The data presented for the five DOF LAMSTF system was obtained with the required number of system inputs and outputs. The data presented for the six DOF suspension system was obtained with assumed number of sensor inputs of seven and assumed number of actuator outputs of eight. The plots shown in Figure 2 demonstrate the effect that the number of controller states has upon the maximum achievable controller sample rate. The affect of the number of controller inputs and outputs can also be seen by the difference between the two suspension systems that are compared. One disadvantage of a standard PC class machine is the segmented memory architecture due to the 80x86 chipset. As a result of the segmented memory architecture, most common compilers, such as the Microsoft C compiler, are limited by the operating system to the lower 640 kilobytes (KB) of RAM. In order to access the memory beyond one megabyte (MB), 32-bit compilers with a DOS-extender must be used, such as the GNU/C compiler. Unless a 32-bit compiler is used, limitations are placed on the amount of data that can be stored real-time, due to the limited amount of memory that would be available. In order to save the required number of data samples to perform system identification, only the measured sensor signals are stored for further post-processing. Other disadvantages of this class of machine are the need of the CPU to handle the data acquisition cycle and the necessity of a separate high resolution timer to provide the pacing for the timed interrupt of the control algorithm.

Data Acquisition Interface

Data acquistion is performed by a plug-in analog-to-digital convertor (ADC) data acquisition PC card. Those types of cards typically offer several programming modes, including DMA data transfer, interrupt driven data acquisition, and program control techniques. The program control technique for data acquisition is the most straight-forward of all approaches. It also tends to be slower and also must take place within the control algorithm's iterative loop. Another approach deals with the use of interrupts to acquire data. That approach has the benefit of occuring outside of the control loop and typically has a higher data transfer rate than the program control method. Significantly higher data transfer rates are achievable with the use of DMA. DMA data transfer also has the benefit of occuring as a background process relative to the control loop. The large throughput rates that are achievable by using DMA can adversely affect the timing of other interrupts due to the fact that during a single DMA transfer, the CPU address, data, and control buses are under the control of the DMA controller. It should be noted that this discussion of DMA is specific to the standard PC class computer. The DMA process can vary between platforms. As the need for faster controller sample rates increases, there exists the possibility of the time skews between acquired data samples becoming large relative to the controller sample rate. This can be detrimental to the designed control method under investigation. A simultaneous sample-and-hold can be utilized to eliminate that effect. There is a certain amount of overhead involved in using the general purpose driver software provided by the vendors. If the use of the general purpose vendor software is not desired, it is necessary to develop routines to handle the interface to the ADC board through its command registers.

Code Optimizations

Several approaches can be used to minimize the loop time of the control algorithm cycle. One method that can be used is to apply the use of boolean logic to control events within the algorithm, instead of the use of conditional statements. That has the effect that the same amount of code would be executed every iteration of the control loop, thus preventing variances in the loop time of the control algorithm. Other steps that can be taken is to remove variable declarations or equations that are constant from within any iterative loop. Another step that can be performed is to enable optimizations within the compiler, primarily any time-based optimizations. This can result in significant increases in the speed of routines that are computationally intensive. Any duplicate code that can be written as a separate routine should be implemented

as separate function calls, allowing the compiler optimizer to function efficiently.

ALGORITHM IMPLEMENTATION

The GRTSSC software package is composed of three distinct processes. The foreground process provides for user interface with the GRTSSC software package. The process consists of routines that load controller initialization information, save measured data information, allow for parameter modification by the user between control runs, and provide a status update of the software to the screen. The data input process provides for the data acquisition of the sensor inputs from the magnetic suspension system. The control algorithm process of the GRTSSC performs the control law calculations of the desired control algorithm.

A modularized approach was used in developing the GRTSSC software package. The approach minimizes the amount of modifications required for implementing the package on various platforms. Any machine dependencies due to hardware or architecture were implemented in separate routines, where possible. It would be left up to the user to write the necessary routines for a new platform. Those routines deal primarily with the data acquisition subsystem, the pacing of the control algorithm via interrupt, and some user interface via the keyboard. The software was implemented in C and adheres to the ANSI compliance standard, with the exception of platform dependent code.

Foreground Process

The foreground process of the GRTSSC software package provides a menu from which the user can load controller state-space information, modify reference input/output parameters, save acquired data to a file, and execute the control algorithm. Two configuration files exist, which cannot be altered while the software is running. The structure of those configuration files is presented in Table I. The first of the files determines the number of system inputs, system outputs, degrees-of-freedom of the system under control, and certain setpoint reference data for the sensors and actuators. The second configuration file provides the data necessary for noise injection on either the sensors or actuators. During run-time, the user may load pre-calculated controller state-space information from an input data file. The controller input data file is summarized in Table II. That file contains specific information concerning the size of the input, output, and state vector of the controller; initial conditions for various vectors; and the control system state-

space matrices. By using the format specified, beginning with the number of controller states, n_x, multiple controllers can be appended to the end of the controller input data file. The format for the data output file is presented in Table III. A time base and measured sensor inputs are written to the file for post-processing by MATLAB.

Data Input Process

An existing DAS-40 ADC card by Keithley-Metrabyte [12] is utilized for data input functions. The card provides 12-bit 16-channel single-ended/8-channel differential input capabilities and can achieve a 250 KHz throughput rate by using DMA to transfer acquired data from the ADC to buffer memory residing on the PC. By using the on-board pacer clock for the DMA data transfer process, the card is programmed with vendor provided routines to sample a sequence of channels continuously. The acquistion of each channel and the DMA transfer is paced at 70 KHz per channel by the on-board pacer clock. The data input process starts prior to the beginning of the control algorithm process and terminates at the end of the control algorithm process.

Control Algorithm Process

The control algorithm process is summarized in Figure 3. The control algorithm executes in an ISR paced by an interrupt generated by a CIO-CTR05 Counter/Timer board by ComputerBoards [13]. The timer board possesses a one MHz oscillator and a single Am9513 counter timer chip with five 16-bit counters. By using register level programming, a single counter is enabled to decrement from a loaded value and generates a CPU interrupt upon reaching zero. The number of counts required is a function of the oscillator frequency and the desired sample rate, as shown in equation (1).

$$Counts = \frac{fosc/2}{\tau} \tag{1}$$

The ADC input routine, shown in Figure 3, consists of copying the measured sensor data from the DMA memory buffer to memory pointers used by the control algorithm. A CIO-DAC16/12 digital-to-analog convertor (DAC) card by ComputerBoards [14] is used for the data output of the control algorithm. That card provides 12-bit 16-channel single-ended/8-channel differential output capabilities for the control system. The DAC output routine, shown in Figure 3, outputs the calculated control actuator commands, using register level programming.

The control algorithm implements the discrete-time state-space equations shown in equations (2) and (3).

$$\{X_{c_{k+1}}\} = [A_c]\{X_{c_k}\} + [B_c]\{U_{c_k} + U_{noise} - U_{ref}\} \tag{2}$$

$$\{Y_{c_k}\} = [C_c]\{X_{c_k}\} + [D_c]\{U_{c_k} + U_{noise} - U_{ref}\} + \{Y_{noise}\} - \{Y_{ref}\} \tag{3}$$

Note that the discrete-time state-space equations are written from the standpoint of the control computer. The input vector, $\{U_{c_k}\}$, is the measured sensor signals, and the output vector, $\{Y_{c_k}\}$, is the calculated control actuator commands to the plant model. Noise injection occurs via the sensor and actuator noise inputs, U_{noise} and Y_{noise}. Reference inputs are provided for the sensors and actuators, U_{ref} and Y_{ref}, see Figure 4. A state-space model in modal canonical form is utilized in the control algorithm. That significantly reduces the amount of matrix-vector multiply and addition operations performed by the control algorithm. The controller system matrix, $[A_c]$, is ordered as shown in equation (4).

$$[A_c] = \begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & \cdots & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & & & \vdots \\ 0 & a_{32} & a_{33} & a_{34} & 0 & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & 0 & a_{n-2n-3} & a_{n-2n-2} & a_{n-2n-1} & 0 \\ \vdots & & & 0 & a_{n-1n-2} & a_{n-1n-1} & a_{n-1n} \\ 0 & \cdots & \cdots & \cdots & 0 & a_{nn-1} & a_{nn} \end{bmatrix} \tag{4}$$

Without the modal canonical form of the state-space controller, the $[A_c]$ matrix would require n^2 multiplications and $n(n-1)$ additions, where n is the number of states of the controller. By using the modal canonical form, the number of multiplications can be reduced to $3n-2$, and the number of additions can be reduced to $2(n-1)$.

The input and output operations of the control algorithm take place as close to the same time point as possible. That has the effect of minimizing the amount of computational processing time, δ, that occurs between input and output. The majority of control calculations then occur after the output of the actuator control command, but before the next sensor input during the next control iteration, refer to Figure 5. This allows the user to implement control algorithms with or without frame delays, simply by including the delays in the state-space model The feedthrough of the measured sensor inputs via the controller state-space matrix, $[D_c]$, is the only calculation required between the input and output functions. The larger the ratio of the computational processing time, δ, to that of the controller sample rate, τ, the more adverse the

effects upon the system, if not properly modelled during the controller design phase. The computational processing time, δ, can be modelled as a transport delay in simulation efforts for the closed-loop system.

CONCLUSIONS

A digital control algorithm for magnetic suspension systems has been developed, and design criteria for the software package has been presented and discussed, including the limitations and trade-offs inherent in any design. Due to the nature of the design of the GRTSSC, applications towards other types of systems exist. System hardware requirements were presented and discussed. Development of the control software package was discussed, including the design of the control algorithm. The efforts involved in producing the fastest possible control sample rate are discussed. The resulting package has a versatile configuration and can be used to investigate a variety of control approaches and applications without the need of developing separate source code for each control approach.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the assistance of Nelson J. Groom and David E. Cox of the Guidance & Control Branch, Flight Dynamics & Controls Division at NASA Langley Research Center

REFERENCES

1. Groom, N.J.: *Analytical Model of a Five Degree of Freedom Magnetic Suspension and Positioning System.* NASA TM-100671, March 1989.

2. Britcher, C.P.; Ghofrani, M.; Britton, T.C.; and Groom, N.J.: *The Large Angle Magnetic Suspension Test Fixture.* International Symposium on Magnetic Suspension Techonology, NASA Langley Research Center, NASA CP-3152, August 1991.

3. Britcher, C.P.; Ghofrani, M.; Haj, A.; and Britton, T.C.: *Analysis, Modelling and Simulation of the Large-Angle Magnetic Suspension Test Fixture.* Proceedings of the Third International Symposium on Magnetic Bearings, July 1992.

4. Groom, N.J.; and Britcher, C.P.: *A Description of a Laboratory Model Magnetic Suspension Test Fixture with Large Angular Capability.* The First IEEE Conference on Control Applications, Dayton, OH, September 1992.

5. Cox, D.; Groom, N.J.: *A Decoupled Control Approach for a Large-Gap Magnetic Suspension*

System. 2nd International Symposium on Magnetic Suspension Technology, Seattle, WA, August 1993.

6. Lim, K.B.; and Cox, D.E.: *Experimental Robust Control Studies on an Unstable Magnetic Suspension System.* Proceedings of the 1994 American Control Conference, June 1994.

7. Lawson, M.A.; and Gillies, G.T.: *Interrupt-Driven Digital Controller for a Magnetic Suspension System.* Rev. Sci. Instrum., vol. 60, no.3, March 1989, pp.456-465.

8. Sinha, A.; Wang, K.W.; and Mease, K.L.: *Robust and Real-Time Rotor Control with Magnetic Bearings.* AIAA-91-3626, 1991.

9. Wassermann, J.: *A Very Fast Digital Processing Controller System for Magnetic Bearing Research.* Proceedings of MAG '95: Magnetic Bearings, Magnetic Drives and Dry Gas Seals, 1995, pp.101-108.

10. *SAS/C Development System User's Guide, Version 6.5,* SAS Institute Inc., Cary, NC, 1993.

11. *Microsoft C Reference, Version 6.0,* Microsoft Corporation, 1990.

12. *User Guide for the DAS-40G1 & DAS-40G2 A/D & D/A Data Acquisition Boards, Revision B,* Keithley Metrabyte Corp., Taunton, MA, 1991.

13. *CIO-CTR User's Manual, Revision 2.0,* ComputerBoards Inc., Mansfield, MA, 1994.

14. *CIO-DAC16/12 User's Manual, Revision 2.0,* ComputerBoards Inc., Mansfield, MA, 1994.

Table I. Format Summary of Configuration Files

System.cfg		Noise.cfg	
Line #	Data	Line #	Data
1	# sensors (Pts_{sen})	1	input pts (Pts_{in})
2	# actuators (Pts_{act})	2	output pts (Pts_{out})
3	# DOF's	3	recyle input (0/1)
4	# saved data (Pts_{sav})	4	recyle output (0/1)
5	sensor zero bias	5	input noise data
6	auto DOF cycle (0/1)	⋮	⋮
7	initial sensor ref	5+Pts_{in}-1	input noise data
8	max sensor ref	5+Pts_{in}	output noise data
9	min sensor ref	⋮	⋮
10	# sens ref pts	5+Pts_{in}+Pts_{out}-1	output noise data
11	# sens delay counts		
12	sens ref recycle (0/1)		
13	initial actuator ref		
14	max actuator ref		
15	min actuator ref		
16	# act ref pts		
17	# act delay counts		
18	act ref recycle (0/1)		
19	P2S Matrix		
⋮	⋮		
19+Pts_{sen}-1	P2S Matrix		

Table II. Format Summary of Controller Input Data File

Filename = Ctrl_dat.m	
Line #	Data
1	Suspension Currents
2	Amplifier Gain Adjust
3	# of Controller State (n_x)
4	# of Controller Inputs (n_u)
5	# of Controller Outputs (n_y)
6	Controller Sample Rate (τ)
7	Initial Controller State Vector
8	$[A_c]$ Matrix
⋮	⋮
8+n_x-1	$[A_c]$ Matrix
8+n_x	$[B_c]$ Matrix
⋮	⋮
8+2*n_x-1	$[B_c]$ Matrix
8+2*n_x	$[C_c]$ Matrix
⋮	⋮
8+2*n_x+n_y-1	$[C_c]$ Matrix
8+2*n_x+n_y	$[D_c]$ Matrix
⋮	⋮
8+2*n_x+2*n_y-1	$[D_c]$ Matrix

Table III. Format Summary of Saved Data Output File

Filename = Out_dat.m	
Line #	Data
1	Time Vector Header
2	Time Vector Data
⋮	⋮
2+Pts_{sav}-1	Time Vector Data
2+Pts_{sav}	Time Vector Trailer
2+Pts_{sav}+1	Measured Data Header
2+Pts_{sav}+2	Measured Data, U_{c_k}
⋮	⋮
2+2*Pts_{sav}+1	Measured Data, U_{c_k}
2+2*Pts_{sav}+2	Measured Data Trailer

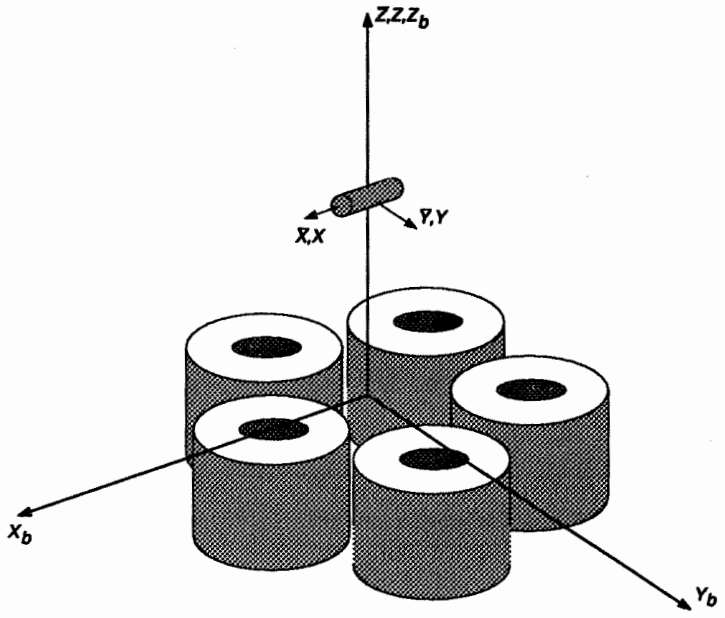

Figure 1. Coil layout of the LAMSTF suspension system.

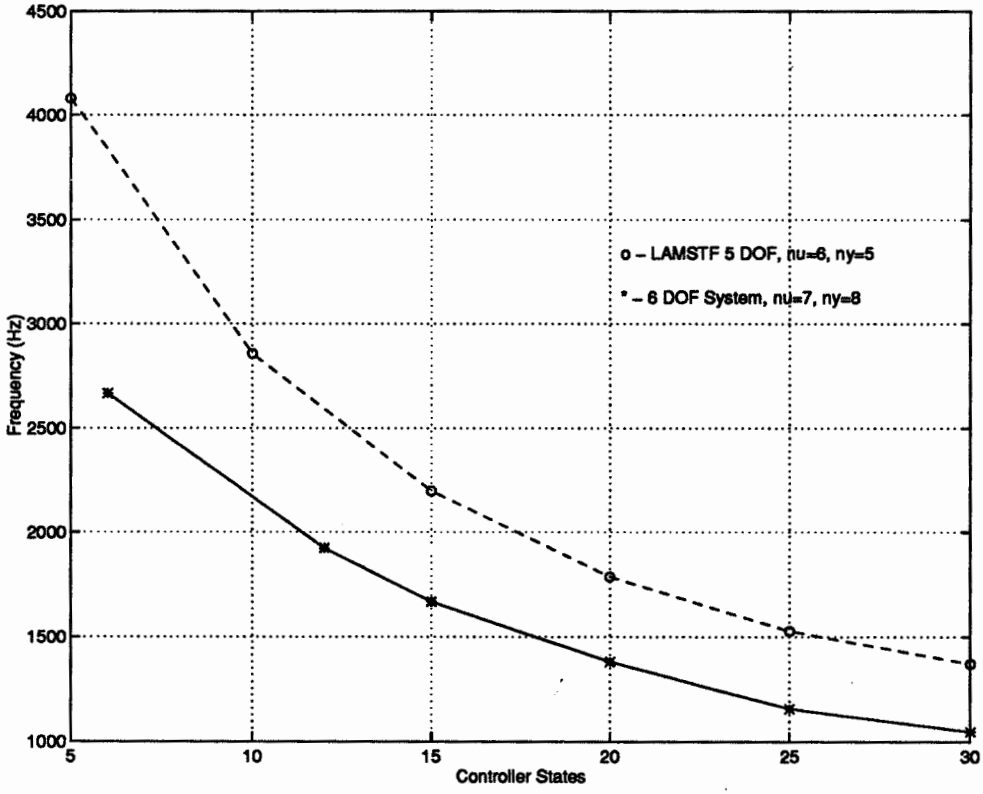

Figure 2. Maximum achievable controller sample rates, using a 486 DX2/66 PC.

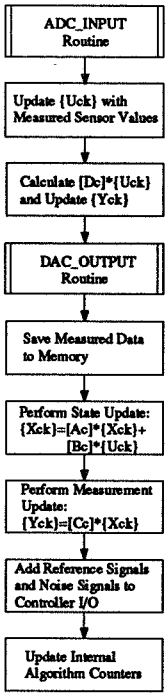

Figure 3. Control algorithm flow diagram.

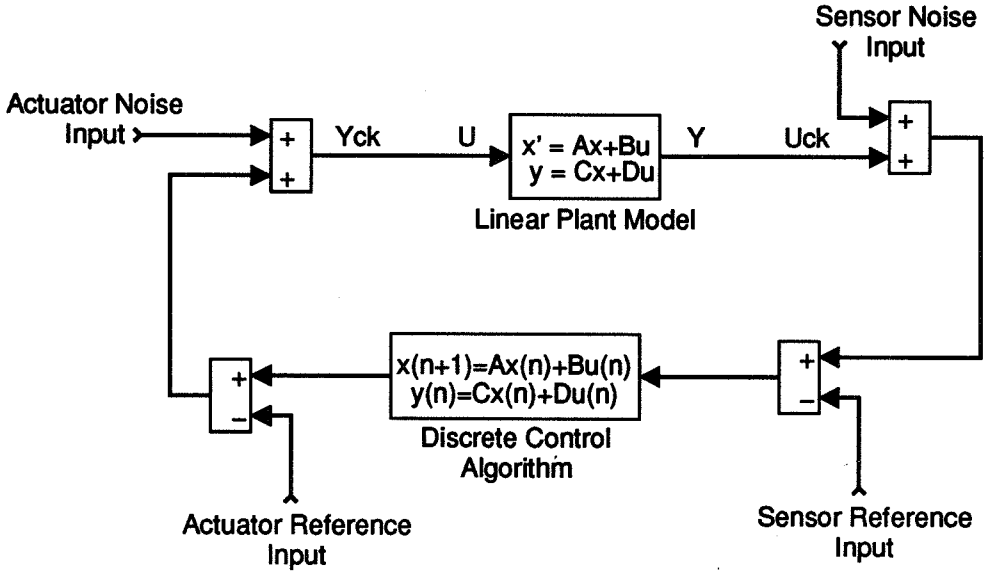

Figure 4. Closed-loop block diagram.

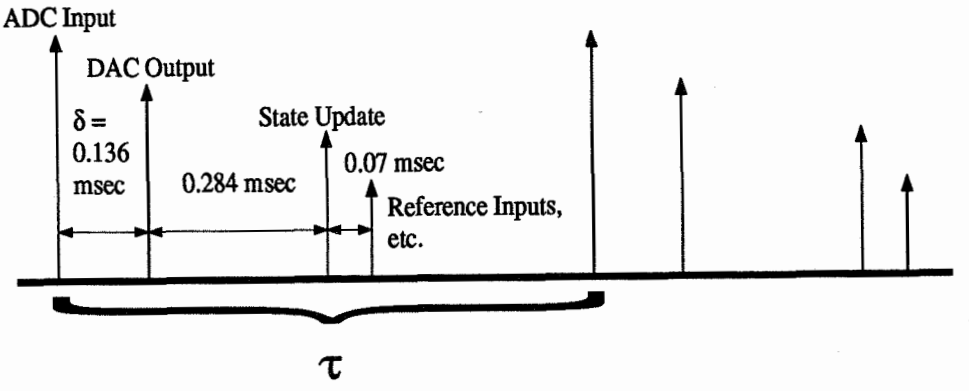

Figure 5. Timeline of control algorithm.