AN EXPERT CONTROLLER

by

Lawrence G. Lebow

AN EXPERT CONTROLLER

Lawrence G. Lebow

Preliminary Report: September 12, 1986

This report breifly describes the "expert controller" as it is envisioned at this early stage in its development. The discussion concentrates on the predicted requirements of the total system along with a preliminary evaluation of the TI Scheme programming environment. An expert controller is a complex machine and many possible difficulties will be unforeseen. Hence, the design of the system is likely to evolve as development takes place. One crucial factor and likely limitation lies in the intention to embed the expert controller in a "personal computer" or microcomputer technology. While this places restraints on the power of the system the hope is to achieve a relatively inexpensive self-contained unit that is an "intelligent" supervisor for industrial process control.

What is an expert controller? Many algorithms exist for the purposes of system modeling, identification, analysis and control law implementation. Many present implementations of controllers employ a subset of these algorithms usually specifically adapted to the process involved. Heuristics are used primarily for safety net situations and again are very specific to the task. An expert controller employes a greater number of algorithms encompassing a much larger variety of situations. "Artificial intelligence" methods, such as production rules, are used to determine which algorithm(s) should be used and when. An expert system program within the expert controller uses identification and analysis algorithms to determine facts about the process to be controlled. Based on this information the "best" control law algorithm is implemented and the results are monitored. If problems arise the expert system should take notice and attempt to correct the situation by implementing more analysis procedures and determining a new control law. When things are not running smoothly in a conventional controller, normally a human operator will have to intervene to diagnose and solve the problem. Also human operators are often required to be directly involved during startup procedures in order to "tune" the system to its normal on line state. The expert controller ultimately would handle these responsibilities and more, allowing human intervention for the purposes of monitoring the process control or perhaps changing the control law being implemented. The advantages of this kind of system are many. Included is the ability to control very complex multi-loop processes (perhaps an expert controller in each loop with another overseeing the total picture) where each controller can

PI and PID, reduced observers, pole placement, adaptive algorithms as well as robust start-up algorithms. In particular the adaptive control laws which include model reference, self oscillating, self-tuning and possibly gain scheduling are of special interest. By definition, these algorithms adapt to a process with unknown parameters and hence allow an expert controller to handle a more general body of processes.

The third portion of the expert controller is the expert system itself. This is perhaps the most interesting as it must monitor and control the other two processes as well as doing the "thinking" which transforms this machine from a simple controller with heuristics to an expert controller. The concept for the implementation of this expert system is a single large program that runs in a continuous loop. On each pass through this loop the program would perform some or all of the following functions. First and foremost, alarms that indicate undesirable or disastrous process behavior would be checked. If such an alarm condition were found all other tasks would be suspended while production rules (or some other methodology) were employed to determine the "best" course of action. Other crucial indicators such as stability and ringing in the process response curve would then be checked and acted upon accordingly. If these steps are successfully accomplished the program would then next determine whether the present mode was process identification or process control. Such indications would be stored in a "situation" table or list. Once the mode has been determined, production rules can be applied to determine if a change in algorithms is desired and what that change might be. Then the decision is turned over to a special subroutine or procedure that knows how to communicate this information to the parallel process of ongoing control law (or identification scheme) implementation. This communicator procedure might also be responsible for monitoring the most recent parameter information from the process and supplying this information back to the expert system. Finally in the midst of all this, provision must be made for communicating with and implementing the user interface described above. This is envisioned as a concurrently programmed process. Certainly, another parallel processor seems a bit cumbersome for the purpose. A queue can be set up to store user commands (or requests). On each pass through the loop this queue can be checked. If it is not empty the first command in the queue is executed. In this way, only one user interface process can absorb processor time on any pass through the loop. Even more efficient, each possible user interface process could be divided into equally timed subprocesses which will then be queued such that only one occurs on each pass. Thus, the time spent on user interfacing is constant for each pass and the ongoing process control proceeds without any significant interruption. The rough diagram that follows this report gives a graphical representation of this overall design. The diagram includes the process itself in the loop along with other already mentioned portions of the expert controller. The process may be actual or simulated but in order to actually test the expert controller a "real time process" must eventually be made available and properly coupled to

the expert controller.

The programming language and environment, TI Scheme by Texas Instruments, Inc. is the intended vehicle for the expert system. This language is a dialect of LISP with several special features and improvements. With a LISP type language, the ability to write programs that exhibit "artificial intelligence" is facilitated. While this system may not require any new state of the art "artificial intelligence" methods the ability to process symbols in a dynamically scoped environment will provide the means for the application of production rules to tables or lists that describe each given situation. This language should readily accomodate the core of the expert system program. The user interface also can be handled quite nicely by Scheme. TI Scheme provides graphics applications and hence visual aids such as graphs can be implemented. Also, a facility for separating the screen output into windows is incorporated in the language. Even more interesting, a structure called an "engine" is provided which will execute a specified duration of time and can then be continued later from that same point. This would seem to handle the difficulty of concurrent user interface operations (discussed above) rather well. There are some foreseeable difficulties, however. Although Scheme seems to perform some operations quite quickly, (especially for a LISP type language) it moves too slow for some of the algorithms that must be applied in "real time". Speed also seems to depend quite heavily on available memory space which does get usee up quickly with a dynamically scoped language. Also as space becomes scarce garbage collection routines are automatically invoked more often, producing an ever more unpredictable timing situation. Some of this can certainly be handled with good programming but it seems the "real time" algorithms should be in a compiled language to insure dependable execution times. This may not really present any direct difficulty as these algorithms should be run on a parallel processor if possible and hence Scheme need not be involved at all. Fortran or C would make good choices for alternate programming languages for this purpose. Scheme does provide the means to execute code compiled from other languages from within the Scheme environment but this would only suffice as questionable option to a parallel microprocessor. Within the present design Scheme has one more difficulty. There seems to be no means of directly accessing or partitioning memory. (Certainly not a common feature with most programming languages anyway.) This facility, or something like it, is necessary for communication between the expert system program and the control law implementation processor as discussed before. Nonetheless, TI Scheme seems to provide many of the necessary features to implement the expert controller and will be used as the central programming environment.

Many difficulties will arise in such a complex undertaking and conversely some things that seemed hard will turn out to be easy (or easier than they looked). As indicated in the opening paragraph, the objective is to put this system in a compact affordable package that does the most possible for its size and relative complexity.

AN EXPERT CONTROLLER

Preliminary Report: September 10, 1986

Lawrence G. Lebow

This report breifly describes the "expert controller" as it is envisioned at this early stage in its development. The discussion concentrates on the predicted requirements of the total system along with a preliminary evaluation of the TI Scheme programming environment. An expert controller is a complex machine and many possible difficulties will be unforeseen. Hence, the design of the system is likely to evolve as development takes place. One crucial factor and likely limitation lies in the intention to embed the expert controller in a "personal computer" or microcomputer technology. While this places restraints on the power of the system the hope is to achieve a relatively inexpensive self-contained unit that is an "intelligent" supervisor for industrial process control.

What is an expert controller? Many algorithms exist for the purposes of system modeling, identification, analysis and control law implementation. Many present implementations of controllers employ a subset of these algorithms usually specifically adapted to the process involved. Heuristics are used primarily for safety net situations and again are very specific to the task. An expert controller employs a greater number of algorithms encompassing a much larger variety of situations. "Artificial intelligence" methods, such as production rules, are used to determine which algorithm(s) should be used and when. An expert system program within the expert controller uses identification and analysis algorithms to determine facts about the process to be controlled. Based on this information the "best" control law algorithm is implemented and the results are monitored. If problems arise the expert system should take notice and attempt to correct the situation by implementing more analysis procedures and determining a new control law. When things are not running smoothly in a conventional controller, normally a human operator will have to intervene to diagnose and solve the problem. Also human operators are often required to be directly involved during startup procedures in order to "tune" the system to its normal on line state. The expert controller ultimately would handle these responsibilities and more, allowing human intervention for the purposes of monitoring the process control or perhaps changing the control law being implemented. The advantages of this kind of system are many. Included is the ability to control very complex multi-loop processes (perhaps an expert controller in each loop with another overseeing the total picture) where each controller can react to unusual situations and solve problems as an ongoing task. Or perhaps as an onboard controller in an airplane or satellite where a control engineer is not likely to be present. For a more complete picture of an expert controller the papers of K.J. Astrom, J.J. Anton and K.E. Arzen

should be consulted. The basic design and concept is there and much of the content of this document is based on their papers on expert control.

The complete system for expert control is quite complex as are each of the elements that join to make up the system. This paper only attempts to give an overview of the expert controller. The expert controller has three main elements: the user interface, a unit for implementing control laws and perhaps most important the expert system that orchestrates the actions of the entire system. The purpose of the user interface is to allow the control engineer to monitor the process and its current control as well as to change the control law being implemented. The information should include such statistics as current and past input and output values (u[k] and y[k]), with their standard deviations, control law parameters and indications of stability. Also available should be a schedule of control laws implemented to date and the control law presently in effect. Graphical representations of some of this information and particularly of the system responses should also be available via the user interface. Many other details such as sampling period etc. could also be included in the available information base. The user interface could be menu driven or command driven and quite likely some combination of both will be used. This user interface must be available as a concurrent or parallel process as the activities of control law implementation and expert system execution should be ongoing while the relatively time consuming task of user interaction takes place.

The process of specific control law implementation must be a truly parallel process. These algorithms must be implemented in "real time". The decision making process of the internal expert system is a much slower process and should not interfere with control law execution. Of course, when the expert system determines that a new algorithm is needed, the control law implementation must be interrupted and changed. Otherwise it should be allowed to run independently of the rest of the expert controller. This unit, which may take the form of a dedicated microprocessor, would also implement identification and analysis type algorithms as well as those for control laws. A crucial factor in this arrangement is how communication between the expert system and the parallel process of control law implementation is to be accomplished. A serial storage unit such as a disk drive would quite likely be to slow for data flow. However, it should be noted that disk storage would probably be needed as a means of retaining all the potential executable control and identification algorithms. Nevertheless, data flow (parameter values, u[k] and y[k]) may have to be accomplished by a common memory directly accessable by both processes. A great number of algorithms may be employed but likely to be included are the following. For identification and analysis: the method of least squares for self-tuning regulators and Zeigler-Nichols auto-tuning regulator. Control laws: PI and PID, reduced observers, pole placement, adaptive algorithms as well as robust start-up algorithms. In particular the adaptive control laws which include model reference, self-oscillating, self-tuning and possibly gain scheduling are of special interest. By definition,

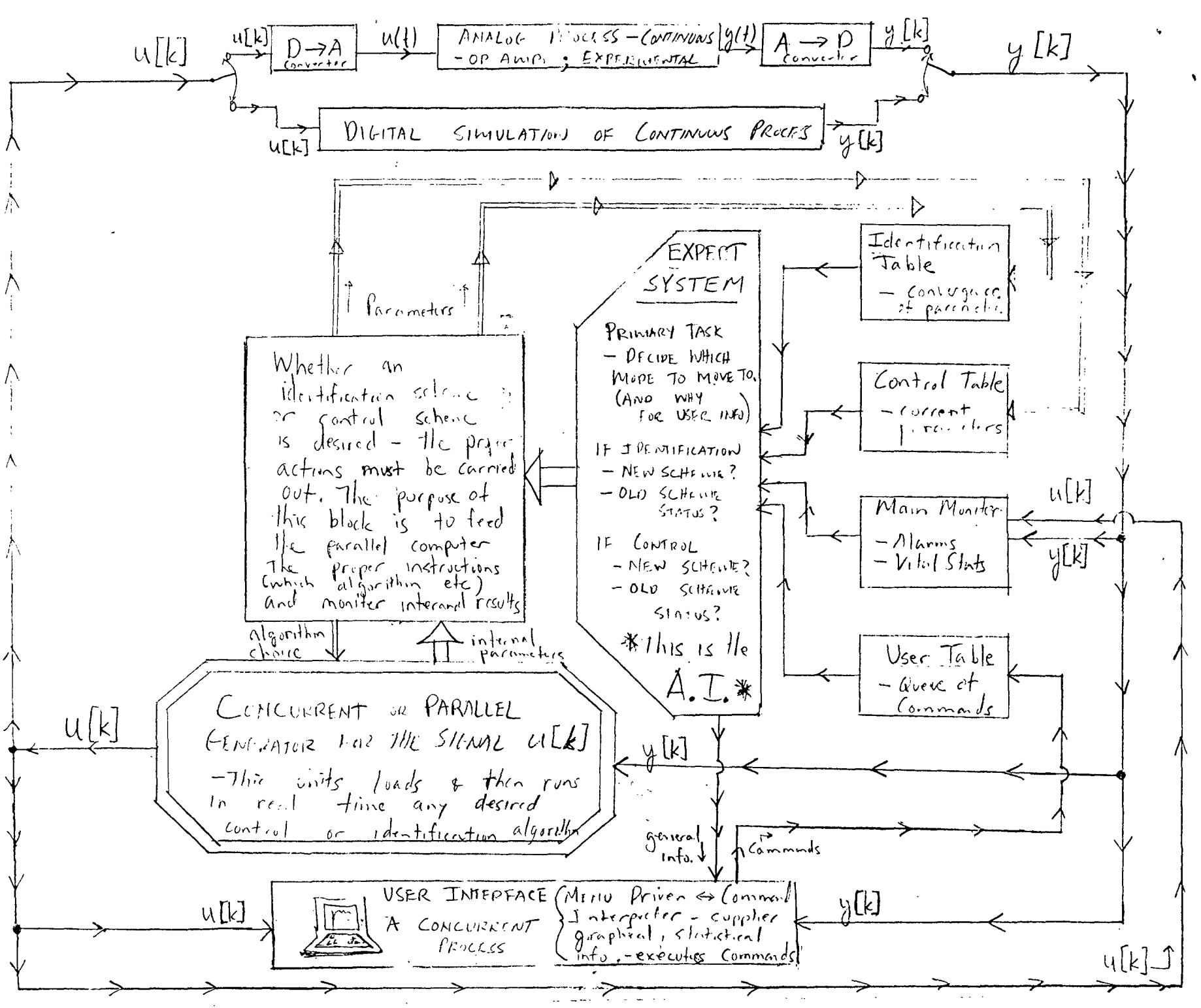
these algorithms adapt to a process with unknown parameters and hence allow an expert controller to handle a more general body of processes.

The third portion of the expert controller is the expert system itself. This is perhaps the most interesting as it must monitor and control the other two processes as well as doing the "thinking" which transforms this machine from a simple controller with heuristics to an expert controller. The concept for the implementation of this expert system is a single large program that runs in a continuous loop. On each pass though this loop the program would perform some or all of the following functions. First and foremost, alarms that indicate undesirable or disastrous process behavior would be checked. If such an alarm condition were found all other tasks would be suspended while production rules (or some other methodology) were employed to determine the "best" course of action. Other crucial indicators such as stability and ringing in the process response curve would then be checked and acted upon accordingly. If these steps are successfully accomplished the program would then next determine whether the present mode was process identification or process control. Such indications would be stored in a "situation" table or list. Once the mode has been determined, production rules can be applied to determine if a change in algorithms is desired and what that change might be. Then the decision is turned over to a special subroutine or procedure that knows how to communicate this information to the parallel process of ongoing control law (or identification scheme) implementation. This communicator procedure might also be responsible for monitoring the most recent parameter information from the process and supplying this information back to the expert system. Finally in the midst of all this, provision must be made for communicating with and implementing the user interface described above. This is envisioned as a concurrently programmed process. Certainly, another parallel processor seems a bit cumbersome for the purpose. A queue can be set up to store user commands (or requests). On each pass through the loop this queue can be checked. If it is not empty the first command in the queue is executed. In this way, only one user interface process can absorb processor time on any given pass through the loop. Even more efficient, each possible user interface process could be divided into equally timed subprocesses which will then be queued such that only one occurs on each pass. Thus, the time spent on user interfacing is constant for each pass and the ongoing process control proceeds without any significant interruption. The rough diagram that follows this report gives a graphical representation of this overall design. The diagram includes the process itself in the loop along with other already mentioned portions of the expert controller. The process may be actual or simulated but in order to actually test the expert controller a "real time process" must eventually be made available and properly coupled to the expert controller.

The programming language and environment, TI Scheme by Texas Instruments Inc. is the intended vehicle for the expert system. This language is a dialect of LISP with several special features and improvements. With a LISP type language, the ability to write

programs that exhibit "artificial intelligence" is facilitated. While this system may not require any new state of the art "artificial intelligence" methods the ability to process symbols in a dynamically scoped environment will provide the means for the application of production rules to tables or lists that describe each given situation. This language should readily accommodate the core of the expert system program. The user interface also can be handled quite nicely by Scheme. TI Scheme provides graphics applications and hence visual aids such as graphs can be implemented. Also, a facility for separating the screen output into windows is incorporated in the language. Even more interesting, a structure called an "engine" is provided which will execute a specified duration of time and can then be continued later from that same point. This would seem to handle the difficulty of concurrent user interface operations (discussed above) rather well. There are some foreseeable difficulties, however. Although Scheme seems to perform some operations quite quickly, (especially for a LISP type language) it moves too slow for some of the algorithms that must be applied in "real time". Speed also seems to depend quite heavily on available memory space which does get used up quickly with a dynamically scoped language. Also as space becomes scarce garbage collection routines are automatically invoked more often, producing an ever more unpredictable timing situation. Some of this can certainly be handled with good programming but it seems the "real time" algorithms should be in a compiled language to insure dependable execution times. This may not really present any direct difficulty as these algorithms should be run on a parallel processor if possible and hence Scheme need not be involved at all. Fortran or C would make good choices for alternate programming languages for this purpose. Scheme does provide the means to execute code compiled from other languages from within the Scheme environment but this would only suffice as questionable option to a parallel microprocessor. Within the present design Scheme has one more difficulty. There seems to be no means of directly accessing or partitioning memory. (Certainly not a common feature with most programming languages anyway.) This facility, or something like it, is necessary for communication between the expert system program and the control law implementation processor as discussed before. Nonetheless, TI Scheme seems to provide many of the necessary features to implement the expert controller and will be used as the central programming environment.

Many difficulties will arise in such a complex undertaking and conversely some things that seemed hard will turn out to be easy. (or easier than they looked). As indicated in the opening paragraph, the objective is to put this system in a compact affordable package that does the most possible for its size and relative complexity.


u[k]
u[k]
D→A Converter
u(t)
ANALOG PROCESS - CONTINUOUS - OP AMPS ; EXPERIMENTAL
y(t)
A→D Converter
y[k]
y[k]
u[k]
DIGITAL SIMULATION OF CONTINUOUS PROCESS
y[k]
Parameters
EXPERT SYSTEM
PRIMARY TASK
- DECIDE WHICH MODE TO MOVE TO. (AND WHY FOR USER INFO)
IF IDENTIFICATION
- NEW SCHEME?
- OLD SCHEME STATUS?
IF CONTROL
- NEW SCHEME?
- OLD SCHEME STATUS?
* THIS IS THE A.I. *
Identification Table
- Convergence of parameters
Control Table
- current parameters
Main Monitor
- Alarms
- Vital Stats
User Table
- Queue of Commands
u[k]
y[k]
Whether an identification scheme or control scheme is desired - the proper actions must be carried out. The purpose of this block is to feed the parallel computer the proper instructions (which algorithm etc) and monitor internal results
algorithm choice
internal parameters
CONCURRENT OR PARALLEL GENERATOR FOR THE SIGNAL u[k]
- This units loads & then runs in real time any desired control or identification algorithm
u[k]
y[k]
general info.
Commands
u[k]
USER INTERFACE A CONCURRENT PROCESS
(MENU Driven ⇔ Command Interpreter - supplier graphical, statistical info. - executes Commands)
y[k]
u[k]