A USER ORIENTED MICROCOMPUTER FACILITY FOR DESIGNING LINEAR QUADRATIC GAUSSIAN FEEDBACK COMPENSATORS*

PAUL K. HOUPT*,**
JAWED WAHID***
TIMOTHY L. JOHNSON*,**
STEVEN A. WARD*,***

*M.I.T. Department of Electrical Engineering and Computer Science
**M.I.T. Laboratory for Information and Decision Systems
*** M.I.T. Laboratory for Computer Science

Cambridge, Massachusetts 02139

ABSTRACT

A laboratory design facility for digital microprocessor implementation of Linear-Quadratic-Gaussian feedback compensators is described. Outputs from user interactive programs for solving infinite time horizon LQ regulator and Kalman filter problems are conditioned for implementation on a laboratory microcomputer system. The software consists of two parts: (1) an off-line high-level program for solving the LQ Ricatti equations and generating associated feedback and filter gains and (2) a cross compiler/macro assembler which generates object code for the target microprocessor system. A digital Equipment Corporation PDP 11/70 with a UNIX Operating System is used for all high level program and data management, and the target microprocessor system is an Intel MDS (8080-based processor). Application to the control of a two dimensional inverted pendulum is presented and issues in expanding the design/prototyping system to other target machine architectures are discussed.

I. INTRODUCTION

The potential impact of microprocessors on control technology has been widely acclaimed. While control design methods for sampled-data systems have been known for a good many years (Kuo, (1); Bryson and Ho, (2)), they were until recently only economical for relatively large systems, where the cost of A/D, D/A interfaces and minicomputer could be justified (typically $20-30K). However, the complexity of such large systems often made testing and evaluation difficult, further impeding the implementation of computer controls.

While the microprocessor has indeed substantially reduced a major cost barrier, and more signfiicantly, opened up a wider range of simpler control applications, we consider it premature to view the microprocessor as a

---

This research was supported in part in the Laboratory for Information and Decision Systems by NASA under Grant NGL-22-009-124 and in the Laboratory for Computer Science by the Advanced Research Projects Agency of D.O.D. under Contract N0014-75-C-0061 monitored by the Office of Naval Research

panecea for the experienced control engineer, let alone the college or graduate student. Ironically, the microprocessor has presented new barriers to the control designer: assembly-language programming, coding for real-time execution, and many other hardware synchronization problems. These problems are by no means insurmountable but in our opinion they are much alleviated by a development system involving a larger machine with higher level language and multiprocessing capability.

Our purpose herein will be to describe a prototype microprocessor design facility and to communicate the key insights which have been gained thus far in its development. The facility consists of high and low-level software, computer hardware, and a continuous target application involving a two degree of freedom inverted pendulum.

II. SOFTWARE DESCRIPTION

2.1 Design Goals

The utility of a laboratory microprocessor control design facility depends critically on its software and hardware organization. At the outset of this project, several goals were established to address both educational and research needs of digital control system designers:

(1) High level languages should be employed at all design levels, in flexible, multi user-interactive (time-sharing) environment;

(2) A control oriented symbolic language for required data processing activities (control laws and filters) should be available to simplify use of the system;

(3) Software developed in the high level language should be applicable to many target machines;

(4) Object code generated should be optimized for the target machine to permit exploitation of available processing speed

With these as long term goals for the structure of desirable software, a prototype system has been developed in the context of a specific (PDP-11) operating system and target microprocessors (Intel 8080).

The MIT System incorporates several novel features which allow the user to accomplish algorithm design in a higher-level language (FORTRAN), develop real-time code in using powerful assembly-language macro commands, which are translated into relocatable object code and finally cross-assembled automatically for the microprocessor itself, and automatically down-loaded into microprocessor memory.

2.2 LQG Compensator Design

The organization of the higher-level Fortran routines for LQG compensator design is shown in Figure 1. Note that other alternative design packages, such as classical frequency-domain or multivariable frequency domain methods could be used equally well here. In the prototype system, the designer does off-line calculations of controller gains by writing very elementary FORTRAN programs using these higher-level routines; alternatively he can employ a specially developed user-interactive program which covers most infinite time horizon continuous and discrete LQG filter and control problems. The solution method for this off-line design phase in the prototype system employs fairly standard eigenvalue decomposition techniques which were developed at the MIT Electronic Systems Laboratory (3). These FORTRAN programs in turn depend on the "EISPAK" eigenvalue/

eigenvector package developed at Argonne Laboratory; this is widely available. User developed or available interactive programs for post-processing of control and filter designs (including simulation) are useful before proceeding to generate code for the target microcomputer systems.

2.3 Code Generator for Target Micro Computer System

After completing the off-line design phase described in 2.2, code to implement the control law and/or filter algorithm on the target micro computer system is generated following the procedure outlined in Figure 2. This activity consists of three main parts: (1) Generation of an assembly program using a collection of previously defined control-specific macros (MACGEN Figure 3); (2) Cross assembly of the user generated program (a source collection of macros) into executable absolute object code (DOMIC Figure 4); (3) down loading of the absolute code into the target system. The DOMIC program performs the steps illustrated in Figure 4 which includes binding of the relocatable user object files from the (MICAL) cross assembly with the previously defined control macros, and conversion of the resulting files into absolute (executable) code for the target microcomputer system. The tedium of performing these procedures manually is thus eliminated for the user. Note that the user-defined macro source program written with MACGEN is canonical, in that by use of a different cross assembler in MICAL, other target processors can be employed. For example, in our prototype system, both Intel 8080 and Motorola 6800 processors can be employed as the target system. Furthermore, the use of special architectual units such as hardware multiply, array processors, etc. can be made essentially invisible to the control designer since the special needs of these units (or even I/O devices) are encoded in the library of MACGEN routines. All program generation in the above system is handled under the supervision of the host operating system, which in the MIT prototype was "UNIX" as implemented on a DEC PDP 11/70. The powerful text editing and file handling capabilities of such systems (or similar) are, in our view, essential in an educational or research environment so that the user (student or researcher) can concentrate on the control/estimation aspects of the problem.

2.4 A Simple Example

It would be impossible in this space to enumerate all the capabilities of the current system implemented at MIT. We will be content to illustrate the heart of the software, the MACGEN program, with a simple Example. A program for a state regulator with (second-order) dynamics (one-input and two-outputs) is illustrated in Figure 5. The general flow of this program is illustrated Figure 6, which also illustrates some additional special features in our prototype system. The processor is assumed to be interrupt driven by an external clock. At each sample time the interrupt routine (INT5:) is serviced. A/D conversions are performed setting

$$x_1 = \text{A/D PORT } \phi$$
$$x_2 = \text{A/D PORT 1}$$

Then

$$u = g_1 x_1 + g_2 x_2$$

is output through port ϕ of the D/A converter; the elements (g_1, g_2) of "gain" are set in the routine called "ready". The real time operation of

this program as it resides in the target machine is explained with the aid of Figure 5a. When the hardware is initialized, the program sets up specified conditions on I/O devices and fetches the required gain data and puts it in a table. A real time monitor program (READY) is then entered which can accept simple commands from a terminal to run, quit, change the gains, etc., all on-line. The control up-dates are accomplished by an external (prioritized) clock interrupt, which causes the branch to "INT5". The program then returns to the READY monitor program.

This example illustrates the relative simplicity of using the mnemonic macros to generate code control updating for real-time (interrupt-driven) operation. More complex control laws including filters, observers, or classical lead-lag compensators are easily constructed with the currently available collection of macros.

III. HARDWARE

3.1 Prototype System

The prototype software described above was implemented as shown in Figure 7 on a DEC PDP 11/70 with 64K of memory, in conjunction with an Intel 8080 based MDS-800 (Development) as the target micro computer system. The MDS system employed hardware multiply, 16K of memory and was outfitted with the Datel D/A and A/D cards shown. (32 A/D inputs, 16 A/D outputs). Two precision D.C. servo amplifiers were provided to drive high power loads (up to 10A. @ 28V.) for simple control designs. A number of custom software modules were written or adapted from available software to handle communications between the MDS monitor and UNIX at the 9.6 K-band rate (including down-load and plotting of hardware results). We believe that our system is well above the minimal "threshold" to support the design software, although no attempt was made to define what lower bound on hardware would suffice. One the other-hand, more sophisticated target systems could have been employed.

3.2 An Application

The prototype system was applied to the design of a two degree of freedom inverted pendulum. The pendulum was a 60" long aluminum pole mounted on a gimbal as shown in Figure 8. The gimbal is mounted to a large (48" x 48") drafting-type X-Y plotter. It consists of a cart which can be translated along an aluminum rail to obtain "X" displacement, and the rail in turn can be translated to obtain "Y" displacements. Motion is achieved with a system of cables and pulleys connected to two 28 volt D.C. servo-motors which can be driven with the amplifier outputs from the system described in 3.1. Linearized analysis of the one-dimensional inverted pendulum is well known and will not be repeated here. Extension to the two dimensional case is straightforward and the reader is referred to Wahid (4) for details. The state variables for the pendulum are:

$$\underline{x} = (x, \dot{x}, \theta_x, \dot{\theta}_x, y, \dot{y}, \theta_y, \dot{\theta}_y)$$

where the angles and relative x,y coordinates are defined in Figure 8. The unforced linearized dynamics decouple between the (x,θ_x) and (y,θ_y) coordinates around $\underline{x} = \underline{0}$. Four observations are available via potentiometers:

$$y_1 = \theta_x$$

$$y_2 = x + .36y$$

$y_3 = y$

$y_4 = \theta_y$

(The coupling between y_2 and y_3 results from the geometry of the cable/potentiometer connections). An LQ^3 (full-state feedback) regulator was designed as described in Wahid (4) (Figure 9). Rate variables were obtained by digital "lead-lag" software (direct differencing with high-frequency rolloff to prevent noise aliasing). This approach was adopted so that sufficiently fast update rates could be achieved with a single 8080 C.P.U. A full-order Kalman filter or observer, could not be up-dated fast enough with the 8080. Reduced order compensators, use of hardware multiply and two (parallel) processor implementations are currently being investigated with the single C.P.U. design. The pendulum was easily stabilized with the LQ gains obtained using an eighth-order dynamic model.

IV. CONCLUSIONS

The prototype control system design tool described has served to meet at least some of the goals outlined in Section 2. Where larger time sharing systems are present, interactive high-level languages for control algorithm design, simulation, and laboratory evaluation provide a favorable environment for both educational and research needs. The essence of our approach is the use of a library of macros for performing a broad spectrum of common control activities. The user can tailor these program modules as his design needs dictate, with code that is relatively easy to use and debug. A second generation of the MACGEN language is currently planned which will provide an even more transparent collection of mnemonics for the required signal processing activities.

V. REFERENCES

[1] Kuo, B.C., Discrete Data Control Systems, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1970.

[2] Bryson, A.E., and Ho, Y.-C., Applied Optimal Control, Blaisdell Publishing, Waltham, Mass., 1969.

[3] Sandell, N.R. and M. Athans, Fortran Computer Routines for the Solution of Linear Quadratic Gaussian Optimal Control Problems, MIT Center for Advanced Engineering Study, Cambridge, Mass., 1974.

[4] Wahid, J., "Implementation of Linear Quadratic Gaussian Compensator on Microprocessors", S.M.E.E. Thesis, MIT Department of Electrical Engineering and Computer Science, May 1978.

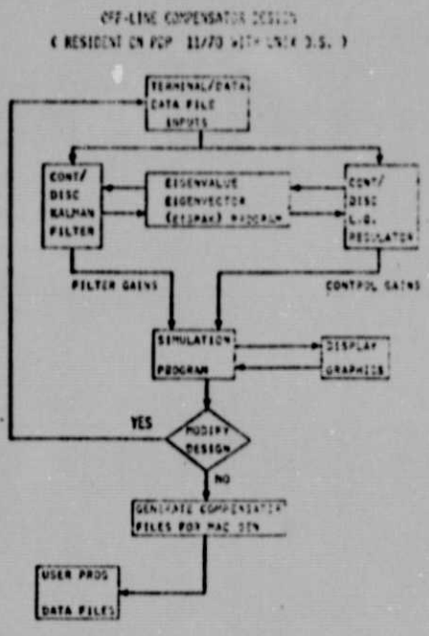

Fig. 1 Interactive(FORTRAN) Compensator Design Program

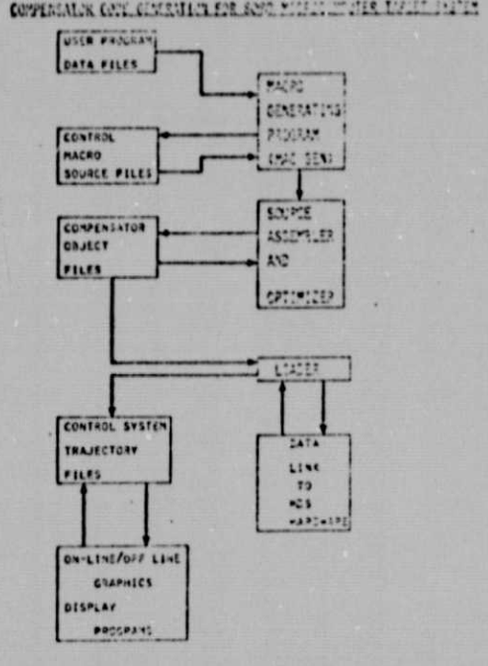

Fig. 2 Compensator Code Generation for Target System

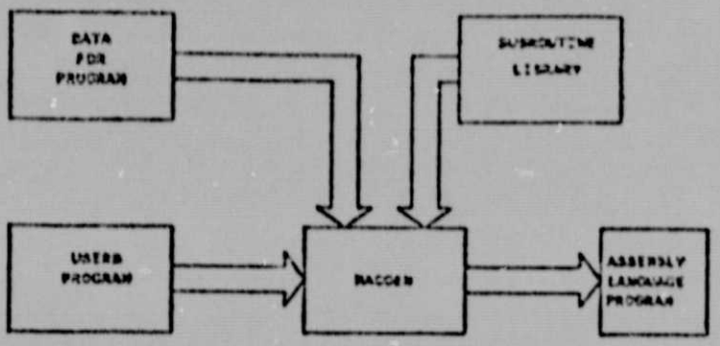

Fig. 3 Basic Operating Scheme of MACGEN

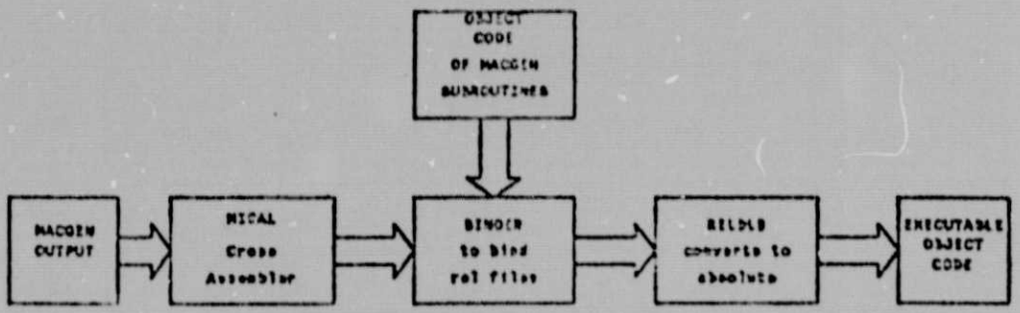

Fig. 4 Converting MACGEN Output to Executable Form

```
.globl int5                    |set interrpt level to 5.
.var x1                        |declare first state variable.
.var x2                        |declare second state variable.
.varp gain 4                   |reserve four bytes for the gains.
.var u                         |declare the control variable.
.var one 1                     |declare a one byte variable called one.
.var two 1                     |declare a one byte variable called two.
.var port0 1                   |declare a one byte variable called port0.
.var port1 1                   |declare a one byte variable called port1.

.asgn one 1                    |assign one = 1.
.asgn two 2                    |assign two = 2.
.asgn port0 0                  |assign port0 = 0.
.asgn port1 1                  |assign port1 = 1.

init                           |call the system initializtion routine.
date two gain                  |get two gain values from data.
ready                          |call the real time monitor.
                               |this is the end of the main program loop.
                               |now comes the interrupt driven portion of
                               |the program.
                               |that does that computation of the control.

int5:          stods port0 x1  |input first state from port 0 of A/D.
stods port1 x2                 |input second state from port 1 of A/D.
mvmul one two one gain x1 u    |compute the control.
dtoas port0 u                  |output new control to port 0 of D/A.
return                         |return from the interrupt loop.
```

Fig. 5 Sample MACGEN Program Implementing a Second Order State Regulator

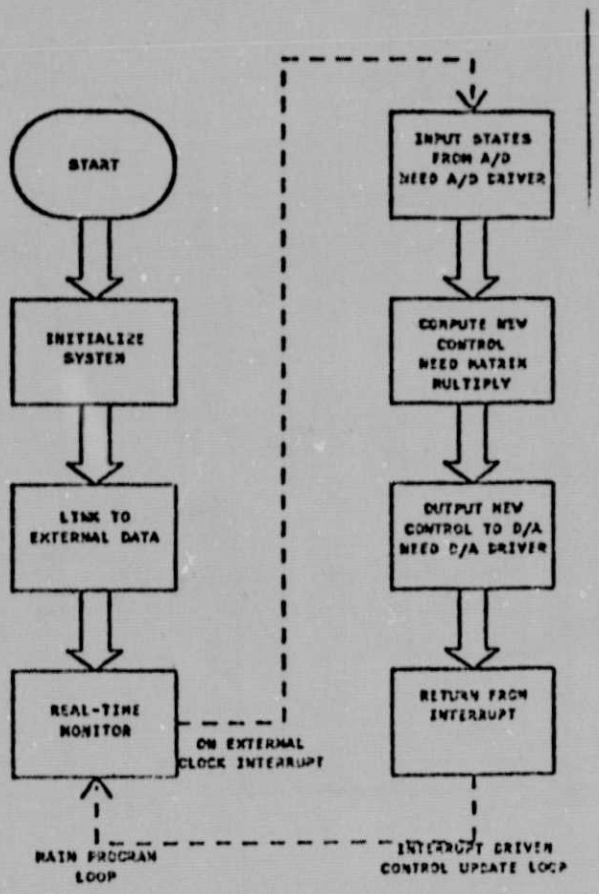

Fig. 6 On-line Computation Required by State Feedback Regulator

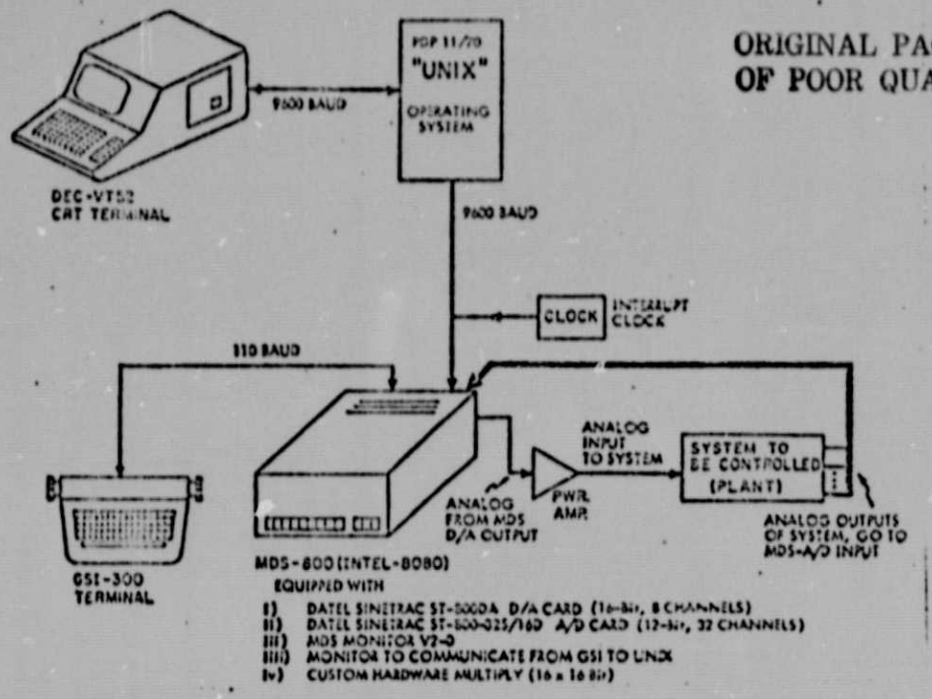

Fig. 7 Prototype Hardware Realization for 8080 Target System

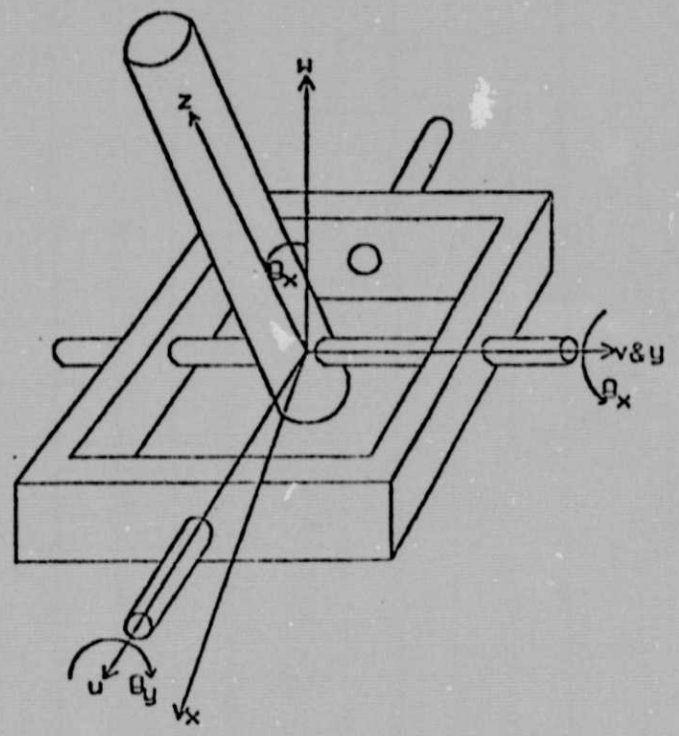

Fig. 8 Inverted Pendulum and its Gimbal Mounting

A USER ORIENTED MICROCOMPUTER FACILITY FOR DESIGNING LINEAR QUADRATIC GAUSSIAN FEEDBACK COMPENSATORS*

PAUL K. HOUPT*,**
JAWED WAHID***
TIMOTHY L. JOHNSON*,**
STEVEN A. WARD*,***

*M.I.T. Department of Electrical Engineering and Computer Science
**M.I.T. Laboratory for Information and Decision Systems
*** M.I.T. Laboratory for Computer Science

Cambridge, Massachusetts 02139

ABSTRACT

A laboratory design facility for digital microprocessor implementation of Linear-Quadratic-Gaussian feedback compensators is described. Outputs from user interactive programs for solving infinite time horizon LQ regulator and Kalman filter problems are conditioned for implementation on a laboratory microcomputer system. The software consists of two parts: (1) an off-line high-level program for solving the LQ Ricatti equations and generating associated feedback and filter gains and (2) a cross compiler/macro assembler which generates object code for the target microprocessor system. A digital Equipment Corporation PDP 11/70 with a UNIX Operating System is used for all high level program and data management, and the target microprocessor system is an Intel MDS (8080-based processor). Application to the control of a two dimensional inverted pendulum is presented and issues in expanding the design/prototyping system to other target machine architectures are discussed.

I. INTRODUCTION

The potential impact of microprocessors on control technology has been widely acclaimed. While control design methods for sampled-data systems have been known for a good many years (Kuo, (1); Bryson and Ho, (2)), they were until recently only economical for relatively large systems, where the cost of A/D, D/A interfaces and minicomputer could be justified (typically $20-30K). However, the complexity of such large systems often made testing and evaluation difficult, further impeding the implementation of computer controls.

While the microprocessor has indeed substantially reduced a major cost barrier, and more signfiicantly, opened up a wider range of simpler control applications, we consider it premature to view the microprocessor as a

---

This research was supported in part in the Laboratory for Information and Decision Systems by NASA under Grant NGL-22-009-124 and in the Laboratory for Computer Science by the Advanced Research Projects Agency of D.O.D. under Contract N0014-75-C-0061 monitored by the Office of Naval Research

panecea for the experienced control engineer, let alone the college or graduate student. Ironically, the microprocessor has presented new barriers to the control designer: assembly-language programming, coding for real-time execution, and many other hardware synchronization problems. These problems are by no means insurmountable but in our opinion they are much alleviated by a development system involving a larger machine with higher level language and multiprocessing capability.

Our purpose herein will be to describe a prototype microprocessor design facility and to communicate the key insights which have been gained thus far in its development. The facility consists of high and low-level software, computer hardware, and a continuous target application involving a two degree of freedom inverted pendulum.

II. SOFTWARE DESCRIPTION

2.1 Design Goals

The utility of a laboratory microprocessor control design facility depends critically on its software and hardware organization. At the outset of this project, several goals were established to address both educational and research needs of digital control system designers:

(1) High level languages should be employed at all design levels, in flexible, multi user-interactive (time-sharing) environment;

(2) A control oriented symbolic language for required data processing activities (control laws and filters) should be available to simplify use of the system;

(3) Software developed in the high level language should be applicable to many target machines;

(4) Object code generated should be optimized for the target machine to permit exploitation of available processing speed

With these as long term goals for the structure of desirable software, a prototype system has been developed in the context of a specific (PDP-11) operating system and target microprocessors (Intel 8080).

The MIT System incorporates several novel features which allow the user to accomplish algorithm design in a higher-level language (FORTRAN), develop real-time code in using powerful assembly-language macro commands, which are translated into relocatable object code and finally cross-assembled automatically for the microprocessor itself, and automatically down-loaded into microprocessor memory.

2.2 LQG Compensator Design

The organization of the higher-level Fortran routines for LQG compensator design is shown in Figure 1. Note that other alternative design packages, such as classical frequency-domain or multivariable frequency domain methods could be used equally well here. In the prototype system, the designer does off-line calculations of controller gains by writing very elementary FORTRAN programs using these higher-level routines; alternatively he can employ a specially developed user-interactive program which covers most infinite time horizon continuous and discrete LQG filter and control problems. The solution method for this off-line design phase in the prototype system employs fairly standard eigenvalue decomposition techniques which were developed at the MIT Electronic Systems Laboratory (3). These FORTRAN programs in turn depend on the "EISPAK" eigenvalue/

eigenvector package developed at Argonne Laboratory; this is widely available. User developed or available interactive programs for post-processing of control and filter designs (including simulation) are useful before proceeding to generate code for the target microcomputer systems.

2.3 Code Generator for Target Micro Computer System

After completing the off-line design phase described in 2.2, code to implement the control law and/or filter algorithm on the target micro computer system is generated following the procedure outlined in Figure 2. This activity consists of three main parts: (1) Generation of an assembly program using a collection of previously defined control-specific macros (MACGEN Figure 3); (2) Cross assembly of the user generated program (a source collection of macros) into executable absolute object code (DOMIC Figure 4); (3) down loading of the absolute code into the target system. The DOMIC program performs the steps illustrated in Figure 4 which includes binding of the relocatable user object files from the (MICAL) cross assembly with the previously defined control macros, and conversion of the resulting files into absolute (executable) code for the target microcomputer system. The tedium of performing these procedures manually is thus eliminated for the user. Note that the user-defined macro source program written with MACGEN is canonical, in that by use of a different cross assembler in MICAL, other target processors can be employed. For example, in our prototype system, both Intel 8080 and Motorola 6800 processors can be employed as the target system. Furthermore, the use of special architectual units such as hardware multiply, array processors, etc. can be made essentially invisible to the control designer since the special needs of these units (or even I/O devices) are encoded in the library of MACGEN routines. All program generation in the above system is handled under the supervision of the host operating system, which in the MIT prototype was "UNIX" as implemented on a DEC PDP 11/70. The powerful text editing and file handling capabilities of such systems (or similar) are, in our view, essential in an educational or research environment so that the user (student or researcher) can concentrate on the control/estimation aspects of the problem.

2.4 A Simple Example

It would be impossible in this space to enumerate all the capabilities of the current system implemented at MIT. We will be content to illustrate the heart of the software, the MACGEN program, with a simple Example. A program for a state regulator with (second-order) dynamics (one-input and two-outputs) is illustrated in Figure 5. The general flow of this program is illustrated Figure 6, which also illustrates some additional special features in our prototype system. The processor is assumed to be interrupt driven by an external clock. At each sample time the interrupt routine (INT5:) is serviced. A/D conversions are performed setting

$$x_1 = \text{A/D PORT } \phi$$

$$x_2 = \text{A/D PORT } 1$$

Then

$$u = g_1 x_1 + g_2 x_2$$

is output through port ϕ of the D/A converter; the elements (g_1, g_2) of "gain" are set in the routine called "ready". The real time operation of

this program as it resides in the target machine is explained with the aid of Figure 5a. When the hardware is initialized, the program sets up specified conditions on I/O devices and fetches the required gain data and puts it in a table. A real time monitor program (READY) is then entered which can accept simple commands from a terminal to run, quit, change the gains, etc., all on-line. The control up-dates are accomplished by an external (prioritized) clock interrupt, which causes the branch to "INT5". The program then returns to the READY monitor program.

This example illustrates the relative simplicity of using the mnemonic macros to generate code control updating for real-time (interrupt-driven) operation. More complex control laws including filters, observers, or classical lead-lag compensators are easily constructed with the currently available collection of macros.

III. HARDWARE

3.1 Prototype System

The prototype software described above was implemented as shown in Figure 7 on a DEC PDP 11/70 with 64K of memory, in conjunction with an Intel 8080 based MDS-800 (Development) as the target micro computer system. The MDS system employed hardware multiply, 16K of memory and was outfitted with the Datel D/A and A/D cards shown. (32 A/D inputs, 16 A/D outputs). Two precision D.C. servo amplifiers were provided to drive high power loads (up to 10A. @ 28V.) for simple control designs. A number of custom software modules were written or adapted from available software to handle communications between the MDS monitor and UNIX at the 9.6 K-band rate (including down-load and plotting of hardware results). We believe that our system is well above the minimal "threshold" to support the design software, although no attempt was made to define what lower bound on hardware would suffice. One the other-hand, more sophisticated target systems could have been employed.

3.2 An Application

The prototype system was applied to the design of a two degree of freedom inverted pendulum. The pendulum was a 60" long aluminum pole mounted on a gimbal as shown in Figure 8. The gimbal is mounted to a large (48" x 48") drafting-type X-Y plotter. It consists of a cart which can be translated along an aluminum rail to obtain "X" displacement, and the rail in turn can be translated to obtain "Y" displacements. Motion is achieved with a system of cables and pulleys connected to two 28 volt D.C. servo-motors which can be driven with the amplifier outputs from the system described in 3.1. Linearized analysis of the one-dimensional inverted pendulum is well known and will not be repeated here. Extension to the two dimensional case is straightforward and the reader is referred to Wahid (4) for details. The state variables for the pendulum are:

$$\underline{x} = (x, \dot{x}, \theta_x, \dot{\theta}_x, y, \dot{y}, \theta_y, \dot{\theta}_y)$$

where the angles and relative x,y coordinates are defined in Figure 8. The unforced linearized dynamics decouple between the (x,θ_x) and (y,θ_y) coordinates around $\underline{x} = \underline{0}$. Four observations are available via potentiometers:

$$y_1 = \theta_x$$

$$y_2 = x + .36y$$

$y_3 = y$

$y_4 = \theta_y$

(The coupling between y_2 and y_3 results from the geometry of the cable/potentiometer connections). An ^3LQ (full-state feedback) regulator was designed as described in Wahid (4) (Figure 9). Rate variables were obtained by digital "lead-lag" software (direct differencing with high-frequency rolloff to prevent noise aliasing). This approach was adopted so that sufficiently fast update rates could be achieved with a single 8080 C.P.U. A full-order Kalman filter or observer, could not be up-dated fast enough with the 8080. Reduced order compensators, use of hardware multiply and two (parallel) processor implementations are currently being investigated with the single C.P.U. design. The pendulum was easily stabilized with the LQ gains obtained using an eighth-order dynamic model.

IV. CONCLUSIONS

The prototype control system design tool described has served to meet at least some of the goals outlined in Section 2. Where larger time sharing systems are present, interactive high-level languages for control algorithm design, simulation, and laboratory evaluation provide a favorable environment for both educational and research needs. The essence of our approach is the use of a library of macros for performing a broad spectrum of common control activities. The user can tailor these program modules as his design needs dictate, with code that is relatively easy to use and debug. A second generation of the MACGEN language is currently planned which will provide an even more transparent collection of mnemonics for the required signal processing activities.

V. REFERENCES

[1] Kuo, B.C., Discrete Data Control Systems, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1970.

[2] Bryson, A.E., and Ho, Y.-C., Applied Optimal Control, Blaisdell Publishing, Waltham, Mass., 1969.

[3] Sandell, N.R. and M. Athans, Fortran Computer Routines for the Solution of Linear Quadratic Gaussian Optimal Control Problems, MIT Center for Advanced Engineering Study, Cambridge, Mass., 1974.

[4] Wahid, J., "Implementation of Linear Quadratic Gaussian Compensator on Microprocessors", S.M.E.E. Thesis, MIT Department of Electrical Engineering and Computer Science, May 1978.

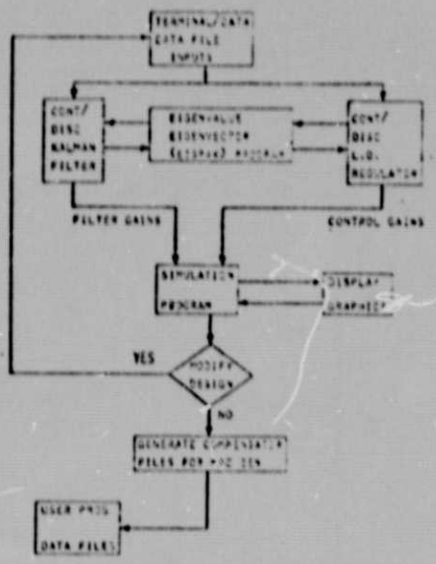

Fig. 1 Interactive(FORTRAN) Compensator Design Program

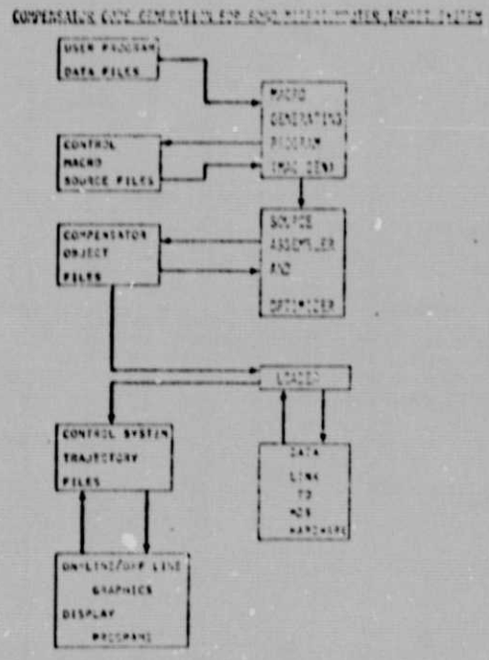

Fig. 2 Compensator Code Generation for Target System

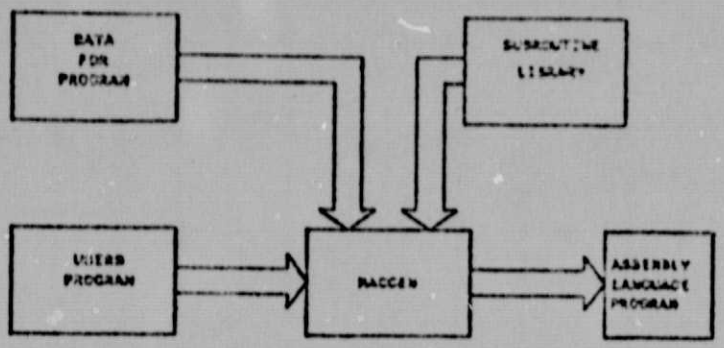

Fig. 3 Basic Operating Scheme of MACGEN

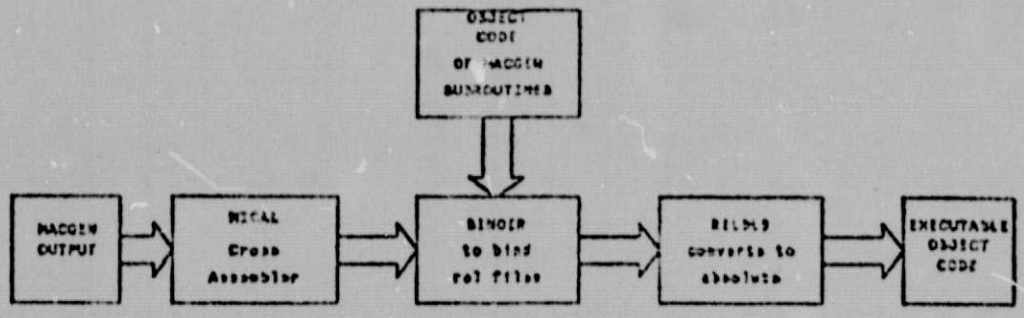

Fig. 4 Converting MACGEN Output to Executable Form

```
.globl int5                    |set interrpt level to 5.
.var x1                        |declare first state variable.
.var x2                        |declare second state variable.
.varp gain  4                  |reserve four bytes for the gains.
.var u                         |declare the control variable.
.var one 1                     |declare a one byte variable called one.
.var two 1                     |declare a one byte variable called two.
.var port0 1                   |declare a one byte variable called port0.
.var port1 1                   |declare a one byte variable called port1.

.asgn one 1                    |assign one = 1.
.asgn two 2                    |assign two = 2.
.asgn port0 0                  |assign port0 = 0.
.asgn port1 1                  |assign port1 = 1.

init                           |call the system initialization routine.
data two gain                  |get two gain values from data.
ready                          |call the real time monitor.
                               |this is the end of the main program loop.
                               |now comes the interrupt driven portion of
                               |the program.
                               |that does that computation of the control.

int5:          atods port0 x1  |input first state from port 0 of A/D.
atods port1 x2                 |input second state from port 1 of A/D.
mvmul one two one gain x1 u    |compute the control.
dtoas port0 u                  |output new control to port 0 of D/A.
return                         |return from the interrupt loop.
```

Fig. 5 Sample MACGEN Program Implementing a Second Order State Regulator

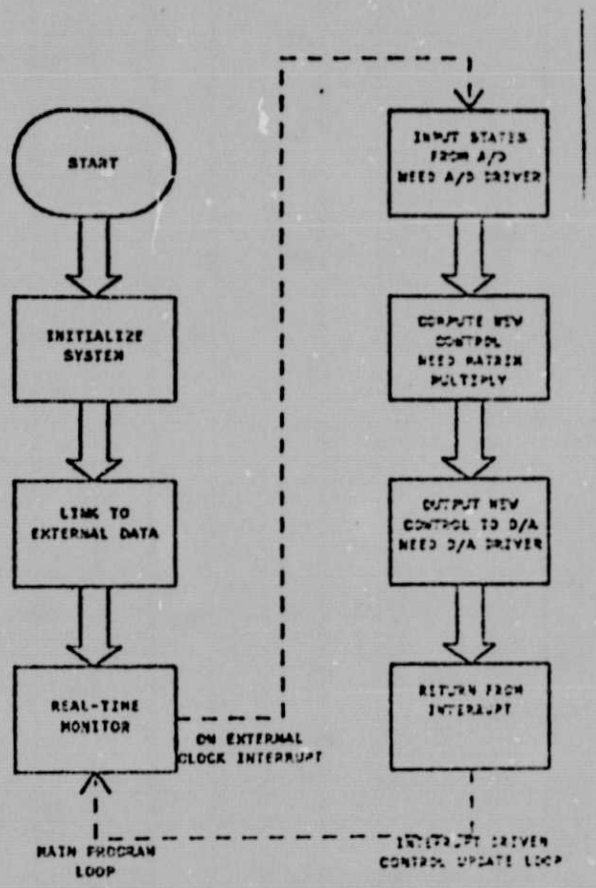

Fig. 6 On-line Computation Required by State Feedback Regulator

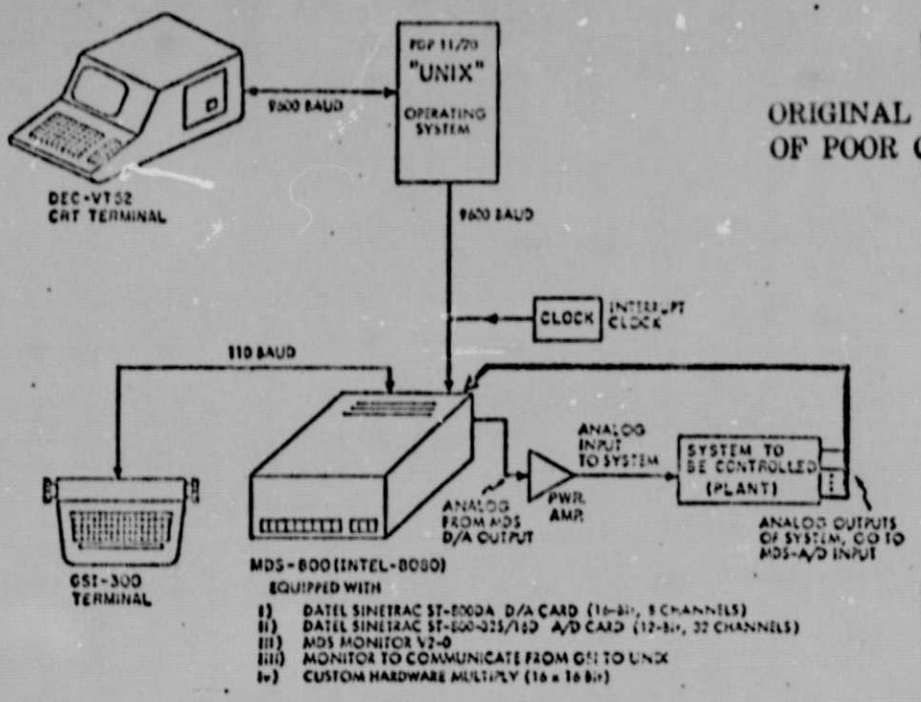

Fig. 7 Prototype Hardware Realization for 8080 Target System

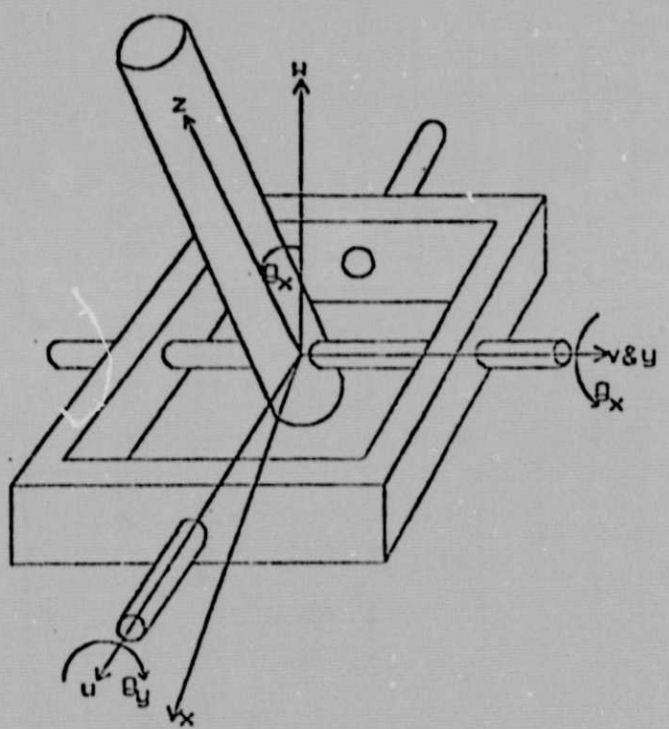

Fig. 8 Inverted Pendulum and its Gimbal Mounting

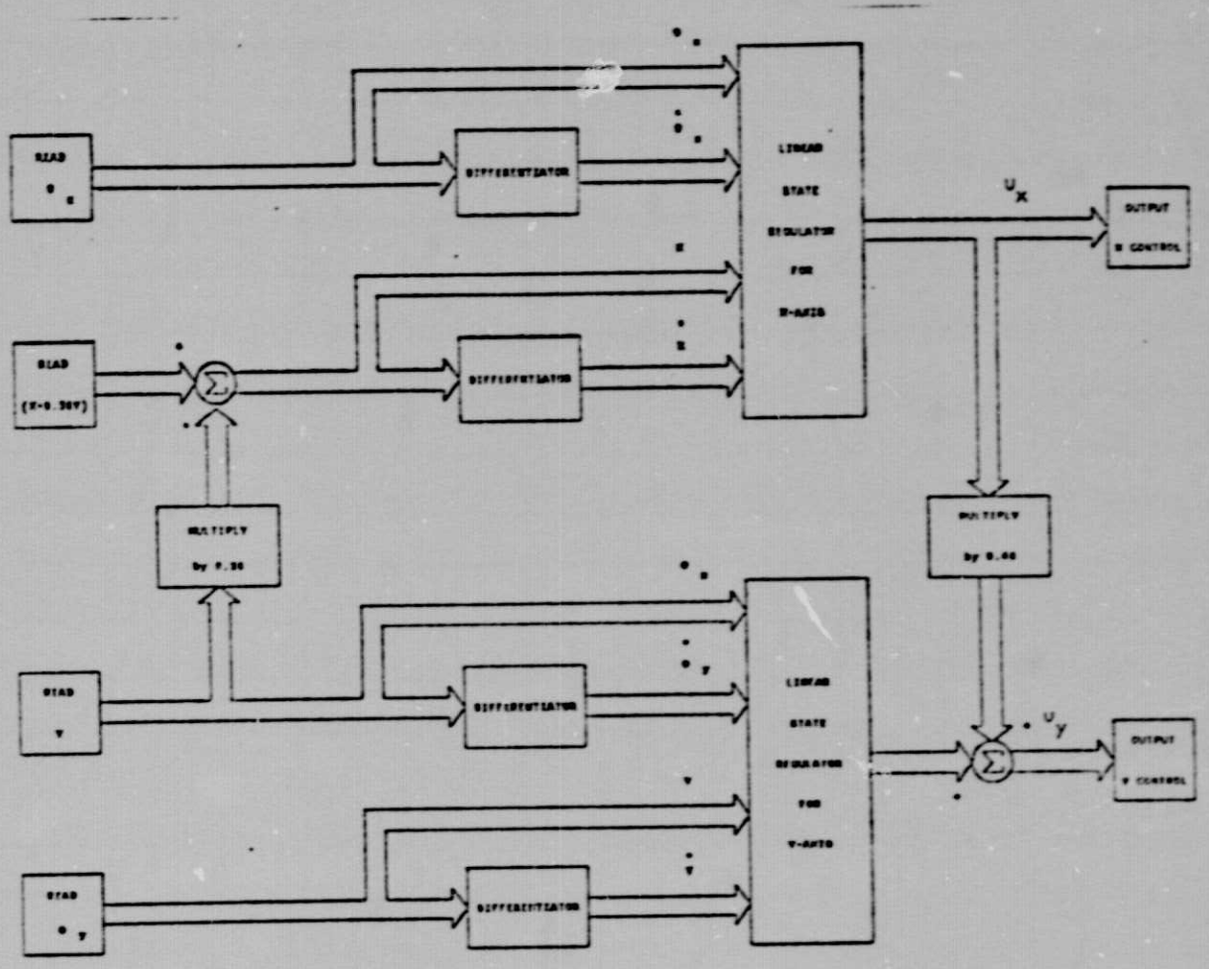

Fig. 9 Flowchart for the Feedback Compensator using Software Differentiators

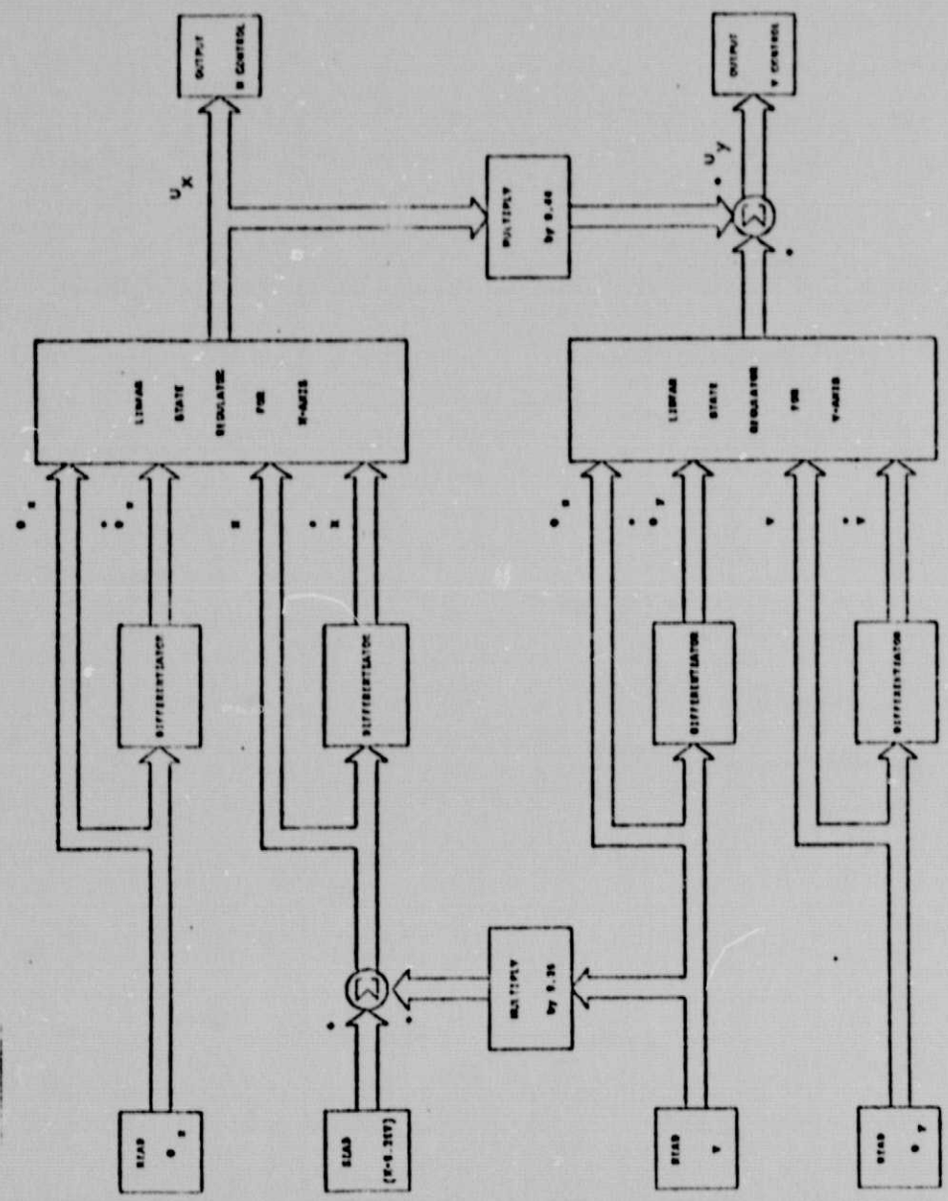

Fig. 9 Flowchart for the Feedback Compensator using Software Differentiators