

LABORATORY FOR
COMPUTER SCIENCE

(formerly Project MAC)



MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-74

THE DESIGN OF A MODULAR LABORATORY
FOR CONTROL ROBOTICS

NIKHIL MALVANIA

SEPTEMBER 1976

MIT/LCS/TM-74

THE DESIGN OF A MODULAR LABORATORY
FOR CONTROL ROBOTICS

Nikhil Malvania

September 1976

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Laboratory for Computer Science
(formerly Project MAC)

CAMBRIDGE

MASSACHUSETTS 02139

This research was supported by the Advanced Research Projects Agency of the Department of Defense, and was monitored by the Office of Naval Research under contract no. N00014-75-C-0661.

THE DESIGN OF A MODULAR LABORATORY FOR CONTROL ROBOTICS

by

Nikhil Malvania

Submitted to the Department of Electrical Engineering and Computer Science on April 12, 1976 in partial fulfillment of the requirements for the Degree of Master of Science.

ABSTRACT

Computers have been used for the control of physical processes since the early sixties. In this thesis, we look at Control Robotics, the procedural control of physical processes. Based upon this new approach, a design for a modular laboratory is proposed. The laboratory consists of a set of experiments which can be synthesized using certain conversion and processing modules. The laboratory also entails the generation of algorithms and programs for each experiment. Experiments are proposed and analysed, and a common and in a sense, minimal set of hardware modules is selected using a minimax approach. Power, torque, strength, resolution and other similar requirements for the modules are discussed. A theoretical model is developed for predicting and analyzing the capability of a processor to perform real-time control. The model is based upon the so-called Earliest Deadline algorithm for scheduling a number of tasks on a single processor. The model relates the bandwidths of different tasks a processor can perform to the total number of tasks; the average instruction execution time for the processor; and the complexity of its instruction set. This model is used to exhibit and compare the controlling capacities of two processors - Digital Equipment Corporation's PDP 11/45 and Intel 8080. It is also used to predict the processor requirements for the experiments of the proposed modular laboratory. Thesis results include measures of relative power of the tested processors in the context of real-time control, and their capabilities in carrying out the experiments of the proposed laboratory.

THESIS SUPERVISOR: Michael L. Dertouzos
TITLE: Professor of Electrical Engineering and
Computer Science

ACKNOWLEDGEMENTS

The author is greatly indebted to his thesis supervisor, Professor Michael L. Dertouzos without whose aid and constant encouragement, this work could never have been completed. Prof. Dertouzos suggested the project, and many of his conceptual ideas have been incorporated in the thesis. He also provided the necessary financial aid during the course of this research.

Thanks are also due to Professor Steve Ward who made useful suggestions from time to time and also participated in the early phases of the project. The author would like to acknowledge the efforts of the Delphi computer system staff in recovering various fragments of this work after some usual and unusual system crashes.

Finally, the author is, of course, indebted to his parents who encouraged and inspired him to pursue higher studies abroad and who have financed his studies from 'way back when'.

TABLE OF CONTENTS

List of Tables	7
List of Figures	8
Chapter 1 Introduction to Control Robotics	10
1.1 Classical Control Systems	11
1.2 Digital Control Systems	13
1.3 Computers in Control Systems	15
1.4 Control Robotics	17
1.5 Outline of the Thesis	18
Chapter 2 Control Robotics - A New Approach to Computer Control	20
2.1 Problems in Computer Control	20
2.2 Daemonized Approach to Computer Control	23
2.3 Advantages of Daemonized Control	27
Chapter 3 Analysis of Controlling Power of a Processor	32
3.1 The Problem and Some Preliminaries	32
3.1.1 The Scheduling Mechanism	33
3.2 A Model for Processor Utilization	36
3.2.1 Useful Tasks	39
3.2.2 Overheads	40
3.3 Simplifications and Analysis of Processor Power	47
Chapter 4 Modular Laboratory for Control Robotics	60
4.1 Overall Setup of the Laboratory	61
4.2 The Modular Laboratory	63

4.2.1	The Experiments	64
4.2.2	Hardware Modules	66
4.2.3	Minimization of Modules	71
4.2.4	Processor Requirements of the Experiments	77
4.3	Laboratory Functions and Its Effectiveness	83
Chapter 5	Experiments for the Modular Laboratory	87
5.1	List of Experiments	87
5.2	Description of Experiments	88
5.2.1	Turtle War	88
5.2.2	Metal Ball Balancing	94
5.2.3	Car to Follow a Laid-out Track	98
5.2.4	Slide Flute	103
5.2.5	Recorder	108
5.2.6	Inverted Pendulum	112
5.2.7	Guitar	117
5.2.8	Violin	121
5.2.9	Yo-yo	125
5.2.10	Stilt Walker	129
5.2.11	Paddle Pool	133
5.2.12	Table Soccer	138
5.2.13	Tilt Maze	142
5.2.14	Software Simulation Experiments	146

Chapter 6	Summary and Conclusions	153
6.1	Summary	153
6.2	A Criticism of the Thesis	155
6.2.1	The Modular Laboratory	156
6.2.2	Model for Processing 157g Power	157
6.3	Suggestions for Further Work	158
References		161

LIST OF TABLES

Table 3.1	Comparison of PDP 11/45 and Intel 8080 Codes	50
Table 3.2	Coding Factors for PDP 11/45 and Intel 8080	57
Table 4.1	A Partial List of Experiments Considered for the Control Robotics Laboratory	65
Table 4.2	List of Experiments for Tables 4.3 - 4.5, 4.8	73
Table 4.3	Preliminary Matrix for the Minimization of Modules	74
Table 4.4	Final Matrix of Experiments and Conversion Modules	76
Table 4.5	Power - Resolution Matrix	78
Table 4.6	Final Set of Conversion Modules and Their Requirements	79
Table 4.7	Module Requirements for the Laboratory	80
Table 4.8	Processor Requirements for the Experiments	84
Table 5.1	Key to Symbols Used in the Figures of Chapter 5	90

LIST OF FIGURES

Figure 1.1	Classical Control Systems	12
Figure 1.2	Digital Control Systems	14
Figure 2.1a	A Typical Servo-Loop	29
Figure 2.1b	Daemon Behavior	29
Figure 3.1	Queue System of the Scheduler	35
Figure 3.2	Assumed Disturbance Waveform for Determining the Factor, k_c	45
Figure 3.3	Plot of Bandwidth, B versus the Number of Tasks, m_t	54
Figure 3.4	Plot of Bandwidth, B versus the Number of Tasks, m_t on a Log-Log Scale	55
Figure 3.5	Plot of B versus m_t with the Approximate Model (for m_t less than 10)	58
Figure 4.1	Overall Setup of the Proposed Laboratory	62
Figure 4.2	An Example Experiment (Car to Follow a Laid-Out Track)	69
Figure 4.3	Graphical Method for Determining the Processor Requirements	82
Figure 4.4	Determining the Processor Requirements for the Laboratory Experiments	85
Figure 5.1a	Turtle War	91
Figure 5.1b	Turtle War - Modular Setup	92
Figure 5.2a	Metal Ball Balancing	96
Figure 5.2b	Metal Ball Balancing - Modular Setup	97
Figure 5.3a	Car to Follow a Laid-Out Track	101
Figure 5.3b	Car to Follow a Laid-Out Track - Modular Setup	102

Figure 5.4a	Slide Flute	105
Figure 5.4b	Slide Flute - Modular Setup	106
Figure 5.5a	Recorder	110
Figure 5.5b	Recorder - Modular Setup	111
Figure 5.6a	Inverted Pendulum	114
Figure 5.6b	Inverted Pendulum - Modular Setup	115
Figure 5.7a	Guitar	119
Figure 5.7b	Guitar - Modular Setup	120
Figure 5.8a	Violin	123
Figure 5.8b	Violin - Modular Setup	124
Figure 5.9a	Yo-Yo	127
Figure 5.9b	Yo-Yo - Modular Setup	128
Figure 5.10a	Stilt Walker	131
Figure 5.10b	Stilt Walker - Modular Setup	132
Figure 5.11a	Paddle Pool	135
Figure 5.11b	Paddle Pool - Modular Setup	136
Figure 5.12a	Table Soccer	140
Figure 5.12b	Table Soccer - Modular Setup	141
Figure 5.13a	Tilt Maze	144
Figure 5.13b	Tilt Maze - Modular Setup	145
Figure 5.14	Space War	148
Figure 5.15	Simulated Ping-Pong	149
Figure 5.16	Simulated Inverted Pendulum	150
Figure 5.17	Simulated Billiards	151

CHAPTER 1

INTRODUCTION TO CONTROL ROBOTICS

Control of physical processes has been one of the prime concerns of almost any kind of activity ever since the beginning of the industrial revolution, if not earlier. In order that a mechanism perform as it is designed, it has been observed that unless another mechanism to control its performance is used in tandem with it, the desired performance is not always guaranteed. This is because of many disturbance factors that upset the stability of the system. The study of control systems which are used to arrest these disturbances and hence guarantee the desired system performance, has undergone a lot of changes as have the control mechanisms.

Briefly, first there are the so-called classical control systems which are characterized by their linear nature and are representative of work done upto the 1950's. The related nonlinear control systems can also be termed classical. A significant difference is seen in the digital control systems which do not monitor a control signal continuously. While the classical control systems were mainly implemented using the vacuum tube technology, the digital control systems employed transistors in the

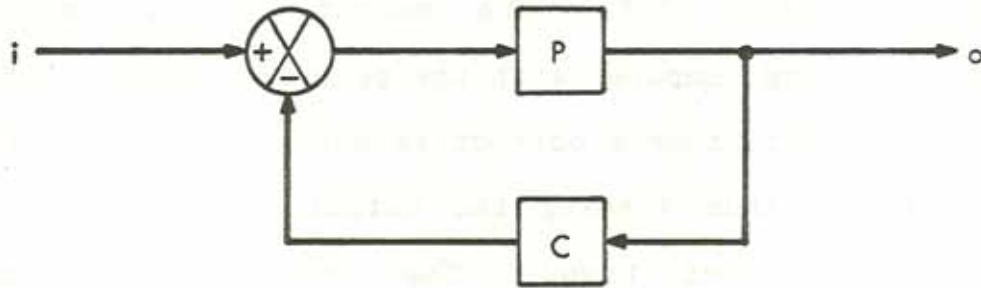
construction of the controllers. With the advances being made in the semiconductor industry and the computer technology, many control systems are now implemented using special and general-purpose computers, and computer programming languages.

In this thesis, following a brief review of the aforementioned approaches, a different method of attacking the problem of computer control of physical processes will be described, and finally, a laboratory based upon this new approach will be discussed.

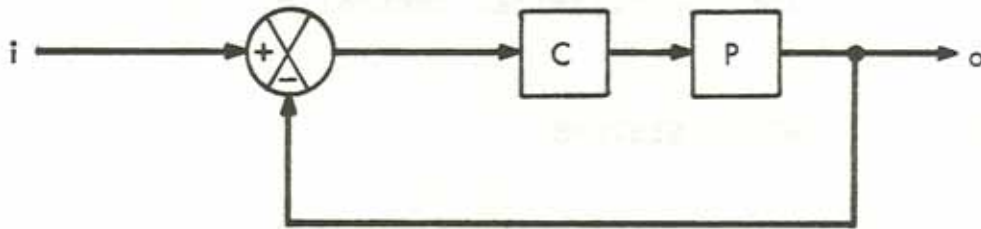
1.1 CLASSICAL CONTROL SYSTEMS

Classical controllers are typically designed by considering the continuous feedback control system design principles. This involves an analysis of the physical process (plant) characteristics together with the desired characteristics for the plant. Next, a decision as to what kind of controller (lag, lead, lag-lead) is required, is made. Once this has been decided, the design is pursued using one of several methods such as Root Locus, Transient Response, Frequency Response[1], which exploit the continuous and linear nature of the overall system.

Figure 1.1a shows one configuration of a classical control system. The output "o" is to be controlled so as to



(a)



(b)

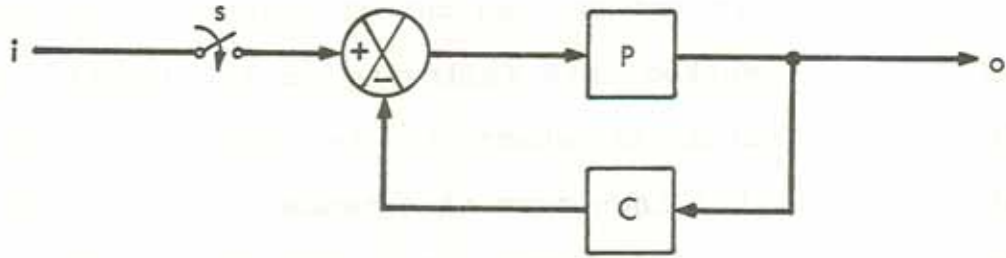
- i - INPUT SIGNAL
- o - OUTPUT SIGNAL
- P - PLANT
- C - COMPENSATOR (CONTROLLER)

Figure 1.1 Classical Control Systems.

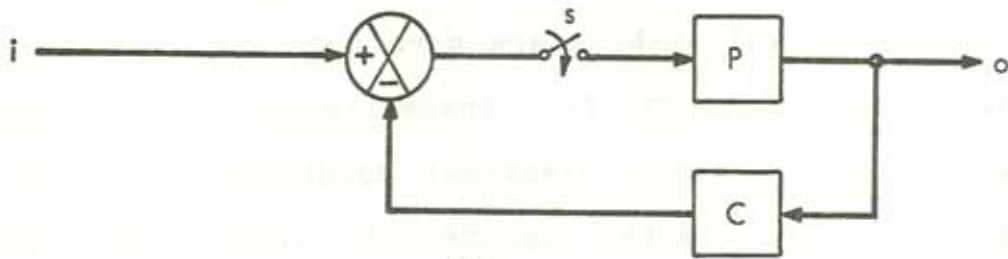
follow the input "i". The output "o" is monitored continuously and compared with the desired standard "i". If these two differ, then a corrective action is applied by the controller C, thus forcing the output of the plant P to follow the desired trend. The processes of sampling (checking the output against a standard), and taking corrective action, are both continuous, and take place simultaneously. Figure 1.1b shows another possible configuration for the classical control systems.

1.2 DIGITAL CONTROL SYSTEMS

Digital (sampled) control systems are not much different from the above classical analog systems. Figure 1.2a shows a digital control system. Here, the comparison between the output signal "o" and the input signal "i", is not done continuously. Either the input or the output is sampled at discrete intervals, or correction is made to the plant periodically. This is made possible by the use of the switch "s" which may be visualised as opening and closing a fixed number of times in one second. The design methods make use of the z-Transform techniques as opposed to the Laplace Transforms used in the design of continuous linear control systems[1]. Figure 1.2b shows another configuration for the digital control systems.



(a)



(b)

- i - INPUT SIGNAL
- o - OUTPUT SIGNAL
- P - PLANT
- C - CONTROLLER
- s - SWITCH

Figure 1.2 Digital Control Systems.

Digital systems evolved out of practical considerations which are described next following an illustrative example. Consider a situation where it is desired to obtain an accuracy of 0.001 inch over an extended range of 100 inches. This calls for a resolution of 1 part in 100,000 - clearly an impossible task to achieve if we use such analog devices as a potentiometer. However, a digitally encoded sensor or an actuator will easily provide the required resolution.

The digital systems are more accurate, and have better noise insensitivity for transmission over considerable distances than their classical counterparts. Also, time-sharing of the digital computer (if used) necessitates data sampling at discrete intervals. Then again certain control components like radar deliver their output in discretized form. All of these point to the superiority of the digital control systems.

1.3 COMPUTERS IN CONTROL SYSTEMS

The use of digital computers for controlling physical processes can be traced back to at least the early 1960's[2,3,4]. These approaches have primarily been a direct extension of the older control systems, i.e. control implemented by digital computers is predominantly linear, and involves function-oriented control strategies. That is,

computers have been largely used as an extension of the classical servomechanism approach. Single processors have been programmed to control physical activities, but this has been done mostly in sequentially oriented languages[5]. Also, a great amount of effort is involved in coping with the hard real-time constraints through interrupt handlers and their counterparts.

Computer usage for control of physical processes has been restricted to particular fields of process control, such as glass manufacture, cement manufacture, and chemical process control[6,7]. The same is true of a number of programmable logic controllers which are available in the market today[5]. The read-only memories supplied with these controllers are tailored for specific sets of control functions.

Computer control schemes which use mini- and micro-computers, are programmed mostly in assembly-level languages with a few extensions for handling input-output and interrupts[5]. There are also available some general-purpose higher-level languages which are more or less extensions of popular programming languages like FORTRAN and PL/1[8]. Other software approaches to computer control, involve the use of special-purpose languages whose applicability is severely restricted not only in the domain of physical processes that can be handled, but also in the

variety of controlling algorithms that can be used. PROSPRO and AUTRAN[8,9,10] are representative of such languages.

1.4 CONTROL ROBOTICS

The process of effecting control involves not a set of sequential happenings, but more an aggregate of parallel (in time) operations. In order to maintain control of the overall system, each of the parallel jobs must be performed by the digital computer at certain instances.

We need to approach the problem by considering goal-oriented activities rather than the function-oriented strategies involving simple servo type activities. Also, the task of effecting control of physical processes in itself does not involve interrupts which are processor-centred activities. Thus, a user who wishes to write a computer program to control a physical process, ideally should not bother himself with the problems of interrupt handling and job scheduling.

We need new tools and approaches to better the interaction between the computers and the physical world. This collection of approaches that couple goal-oriented activities with the physical world of sensors and actuators has been called Control Robotics[11]. It is more advanced than the classical servos in that the Control Robotics approach utilises goal-oriented powers of the digital

computers, and in that the interprocessor communication which it makes possible, allows hierarchies of control. At the heart of this approach is the concept of 'daemon', which is a user-specified process. We will examine this in greater detail in the next chapter. In this thesis, salient features of the Control Robotics approach will be discussed, and then a modular laboratory utilising this approach, will be described.

1.5 OUTLINE OF THE THESIS

In Chapter 2 a brief presentation of the Control Robotics approach is made. In particular, we will look at 'daemons' which form the backbone of this approach, how a user can employ them in a computer program to control physical processes, and the advantages of Control Robotics over the classical and digital approaches. We will also discuss some of the problems arising out of the Control Robotics approach, and the solutions that have been suggested for them.

Chapter 3 discusses in detail the problem of processor utilization. A model is proposed to facilitate the estimation of the power of a processor in the context of real-time control of physical processes. The results of the comparative studies of specific processors using this model

are displayed both graphically and in a tabular form.

A Control Robotics laboratory is envisioned in its modular form in Chapter 4. The process of developing the modular laboratory is described by starting from a list of control experiments and the components required for them, creating a matrix of experiments vs. components, and then attempting to maximize the number of experiments while minimizing the number of components required for them. Certain issues regarding the structure of the laboratory are discussed.

In Chapter 5, the model developed in Chapter 3 will be used to predict the processor requirements for each of the experiments. A list of the experiments with sketch descriptions, schematic diagrams of modular setups, and the requirements of modules to implement the experiments is also to be found in this chapter.

Suggestions for further work are given in Chapter 6 and also, the entire project is viewed in retrospect.

CHAPTER 2

CONTROL ROBOTICS - A NEW APPROACH TO COMPUTER CONTROL

Whenever a new approach for almost anything is being presented, the question that is foremost on the minds of the skeptic is -why a new approach? Various reasons for this are presented in this chapter, following an examination of some of the problems one is faced with in computer control of physical processes.

2.1 PROBLEMS IN COMPUTER CONTROL

A computer does the tasks assigned to it in a sequential manner, whereas in the real-world many processes are running concurrently. Thus, a need for a strategy to make the computer handle concurrent processes arises. Of course, this could very well be taken care of by using as many processors as there are processes or control loops to be handled, and then letting each processor take care of exactly one unique control loop. However, the control phenomenon is not this simple: the processes are not usually independent, i.e. more often than not, there is a lot of interaction between the processes, which arises out of their acting upon a set of variables. Therefore, if one were to resort to a one-processor-for-each-control-loop scheme, one

would then be faced with the problem of inter-processor communications. Even if only one processor were used, there would still be a problem in sharing data and resources. This problem arises because there will in general be more than one process operating on some parameters, and all these processes must be executed by the single processor, thereby necessitating the sharing of data (the variables information) and resources (the single processor).

The problem of process concurrency can be linked to another issue - continuity. The computer is only a digital machine and hence can take action in only discrete intervals of time, while the physical processes to be controlled are continuous in the real-world. However, it is known that even if a process is continuous, it need not be monitored or controlled continuously. Depending on its bandwidth, it may be sampled at fixed time intervals. Thus, "continuity" for a process in the context of computer control is specified to within a certain amount of time which separates two consecutive samples of it by the processor. Thus, for each process there are certain instances in time when that process must be monitored. We refer to these times as the monitor "deadlines". If in between two successive monitor deadlines, after having serviced the process, the processor has capability (that is, it is idle) to attend to other processes, it is free to do so. Multiprogramming and

multiprocessing concepts have been used in the design of computer control systems, reflecting the observations made above.

Another problem one must consider in computer control is the digitization of data. Computers can accept and manipulate only discretized data. This necessitates the use of analog-to-digital and digital-to-analog converters for the computer to communicate with the physical-world, and also the development of special interfaces between the computer and the physical-world. One of the assumptions made regarding the discretization of data is that the finite number of values a variable is represented by, is large enough to faithfully reflect the analog signal.

Because there are many processors handling a set of processes, certain conditions might arise which necessitate the halting of some processes and the starting of some other processes. An instance of this is provided by the following example: Consider the problem of a robot carrying a glass of water across a room. Suppose that the activity now being performed by the processor is that which results in the robot moving across the room. If the glass tilts during this time, the water will spill out. Hence, one needs to monitor the level of the glass periodically, or some mechanism must be provided by means of which it can be known when there is danger of water spilling out of the glass. In

either case, when the level of the glass reaches a certain position, the processor must perform that activity which will steady the glass, thereby preventing water from spilling out. This necessitates the halting of previous activity, namely that of moving the robot across the room, and the starting of a new activity, namely that of steadying the glass of water.

Also, the processes must all be executed by the processors in some order dictated by their bandwidths, deadlines and other factors. These issues introduce the problem of scheduling and interrupt handling.

Other issues one needs to consider include the cycle time and the complexity of the instruction set of the processor. These considerations determine whether the processor will be able to implement in real-time a control strategy for a given system--this subject is treated in considerable detail in Chapter 3.

2.2 DAEMONIZED APPROACH TO COMPUTER CONTROL

The central concept of the new approach is the "daemon", a user-specified process exhibiting some "continuity"[11,12]. A daemon basically consists of two parts: the first part is called the condition of the daemon, which is to be monitored. This condition corresponds to

values received from a sensor or a set of sensors which may be hardware or software. If this condition is "true", then the second part of the daemon takes some appropriate "corrective" action. This action usually involves an actuator or a set of actuators and is called the expression of the daemon. Both the condition and the expression are specified with their own user-declared measure of tolerable continuity. We can see that a daemon with its condition and expression, behaves like a control loop, where the condition of the daemon corresponds to the error detector and the expression corresponds to the compensator (error corrector).

A "control programmer" may specify as many daemons as are necessary. The order of execution of daemons is not relevant for the programmer while writing a daemonized program. The implementation details are kept from the user so that all daemons appear to run concurrently.

The programmer declares a daemon by specifying its five arguments. The first argument is the name of the daemon. The created procedure is referred to in the control program by this name. The second argument is the daemon condition to be monitored. The condition is a piece of code which returns true or false depending on whether a certain event took place or not, respectively. The third daemon argument is the expression of the daemon which is used to apply some corrective action. The expression is a piece of code which

upon completion of its execution returns control to its caller. The fourth argument is called the recognize-within time. This is the measure of continuity for the monitoring of the daemon condition. It is defined as the time within which if a condition arises in the physical process, it will be recognized. The last daemon argument is the service-within time within which the daemon expression must be fully serviced (executed) after the corresponding condition has returned true. Thus we may think of the service-within time and the recognize-within time as specifying certain "deadlines" for some jobs. Both the recognize-within time and the service-within time should be larger than the actual code execution times of condition and expression respectively. The user can establish priorities among various daemons by simply manipulating the recognize-within and the service-within times; however, a user should not worry about priorities which after all are implementation details. The user should only worry about getting the job done on time.

Once the daemons are created, they must be activated for their conditions to be scheduled for execution. The daemons are created in the deactivated state, and ACTIVATE and DEACTIVATE commands are used to either activate a possibly inactive daemon, and to deactivate a possibly active daemon. This is done in order to avoid the situation

where two daemons with conflicting objectives are active simultaneously. The resident scheduler (transparent to the control programmer) will have these conditions executed according to the dictates of the deadlines. The algorithm followed for the single processor scheduling is the Earliest Deadline (ED) algorithm which causes that job to be scheduled first that has the earliest deadline[11]. If any of the conditions returned true, the corresponding expression is scheduled along with other active expressions by the ED algorithm and that condition is deactivated until the corresponding expression has been run fully. This deactivation of condition (done by the scheduler automatically) is necessary if thrashing (continually turning "on" an expression which is already "on") is to be avoided. This increases the efficiency of the system. If a condition returns "false", it is rescheduled for later execution. In between condition checks expressions are executed from the active list of expressions.

For further details of the daemonized approach to computer control of physical processes, the reader is referred to Steven Geiger's Masters Thesis "A New Language Approach To Computerized Process Control"[12].

2.3 ADVANTAGES OF THE DAEMONIZED CONTROL

This section examines the advantages of this new approach to computer control.

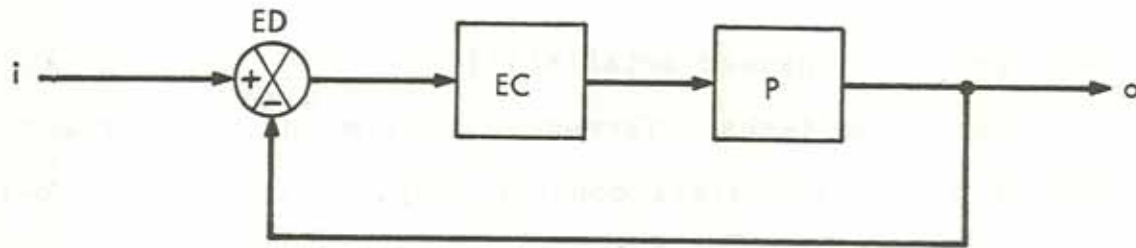
There have been other approaches to computer control using extensions of sophisticated programming languages like PL/1. While it is reasonable to adopt such an approach, the extensions used must provide grounds for making such a venture. In the last section it was seen that the daemonized approach handles multitasking quite easily. Some might argue that PL/1 type languages do provide the multitasking capability. However, the structure of these tasks leaves much to be desired. The programmer is constrained to keep in mind the relationships between the various tasks. Also, the tasks cannot be attached or removed at will. Daemons do not have such problems.

If desired, daemons can be made completely independent of each other and the programmer need not worry about the effect of one daemon on others. Thus, daemons can be specified in any order and can be activated or deactivated from the physical processes which are to be controlled. These features permit a control programmer to add extra loops or modify existing loops or even delete some loops in order to achieve better control in a very natural manner without his having to worry about the timing considerations

and specific language details[13].

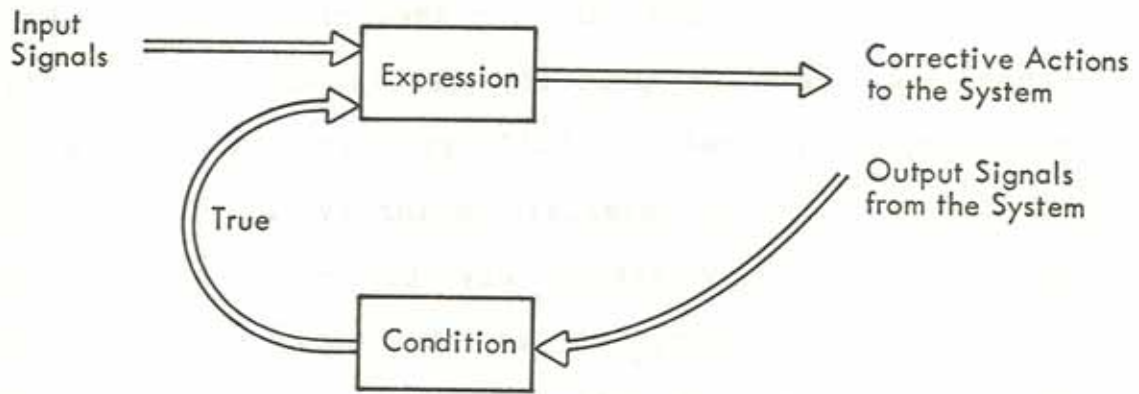
PL/1 type tasks referred to earlier do not reflect the nature of the classical control loop. A typical servo-loop illustrated in Figure 2.1a, has an error monitoring device ED and an actuator scheme EC which corrects any errors that are present. A daemon corresponds to this kind of loop very naturally as indicated in Fig. 2.1b. The error monitor is modelled by the condition, while the counterpart of the actuator is the expression of the daemon. Thus, the control loop is neatly discretized into two separate functions which makes easy the task of translation of a control loop in software.

Another advantage that the daemonized approach has over others is that the priority levels are never explicitly mentioned as against the PL/1 type tasks which are related to one another by explicit priority levels. While the notion of priority levels may be quite natural to the classical programmer, it is the contention of the new approach that such a notion is quite foreign to a control programmer. He is only interested in specifying a number of control loops that appear to run simultaneously, and this is exactly what the daemonized approach offers him while hiding from him the unnecessary details of the mechanism used to time-share processor power among loops. If, however, the control engineer does want to incorporate some kind of



- i INPUT SIGNAL
- o OUTPUT SIGNAL
- P PLANT
- ED ERROR DETECTOR
- EC ERROR CORRECTOR

Figure 2.1a A Typical Servo-Loop.



EXPRESSION IS ACTIVE ONLY IF TRUE IS RECEIVED FROM CONDITION

Figure 2.1b Daemon Behavior.

priority levels among the daemons of his system, he may do so implicitly by using the recognize-within and the service-within times in the context of the ED or some other algorithm used for scheduling.

Daemons allow the control programmer the notions of recognize-within and the service-within times. These reflect directly the time constants of the physical processes that his control loops regulate. Since he will have a pretty good estimate of the time constants of the processes involved, he will be able to specify the recognize-within and the service-within times easily.

Decision-making is another feature incorporated into the daemon structure. This enables one to model any arbitrary non-linear system that conventional approaches cannot. Also, when the daemon condition is "false", the expression is not executed. In classical loops (even in loops with digital filters), the correction mechanism is active even if no correction is necessary. This is sometimes detrimental to efficiency. This is so because when correction is not required, and yet the correction mechanism is on, this will lead to wastage of power and processor time (if a processor is used for control purposes). The ability to specify two separate times, namely the recognize-within and the service-within times, permits the running of the error detectors and the error

correctors at separate speeds as against a common speed forced in the traditional approaches. Thus, in the daemonized version, the user can obtain finer control and can make optimizations on the processor utilization to be able to control more complex tasks.

With this background of the daemonized approach to computer control, we investigate next the controlling power of a processor.

CHAPTER 3

ANALYSIS OF CONTROLLING POWER OF A PROCESSOR

3.1 THE PROBLEM AND SOME PRELIMINARIES

In this chapter we shall attempt to develop a theoretical model which can be used to estimate the controlling power of a processor or in general, the processing power of a computing facility in the context of control of physical processes in real-time by computers via the daemonized approach described in Chapter 2.

The model and the resulting analyses will be used to answer such queries as: (i) can a processor perform a given task? (ii) is it possible to execute a certain number of control tasks of given complexities on a single given processor? (iii) what is the maximum bandwidth allowable of a loop to be serviced by a certain processor? and so on.

The model will be constructed by considering how the processor time is utilized among various tasks that must be performed, and the overheads incurred in scheduling them. The following parameters are involved either explicitly or implicitly in the model: instruction cycle time of the processor, complexity of the instruction set for the processor, the number of tasks to be run on the processor,

bandwidths of these tasks. Ultimately, we would like to present a simple correspondence between the real-world parameters and the processor parameters. Once the model has been constructed, we would like to simplify it considering only the dominant factors and then plot graphs which show the relationships between the processor power and the real-time control parameters.

Before we present the actual model, we shall explain briefly the scheduling mechanism for a single processor and m tasks. We shall make use of this mechanism to formulate the model.

3.1.1 SCHEDULING MECHANISMS

The Earliest Deadline (ED) algorithm is used for scheduling m jobs on a single processor. Briefly, the ED algorithm causes the processor to execute that job whose deadline for completion is the soonest. This necessitates keeping sorted lists of deadlines for completions of jobs. Elsewhere[11], it has been shown that for a single processor case, the ED algorithm is the optimal one.

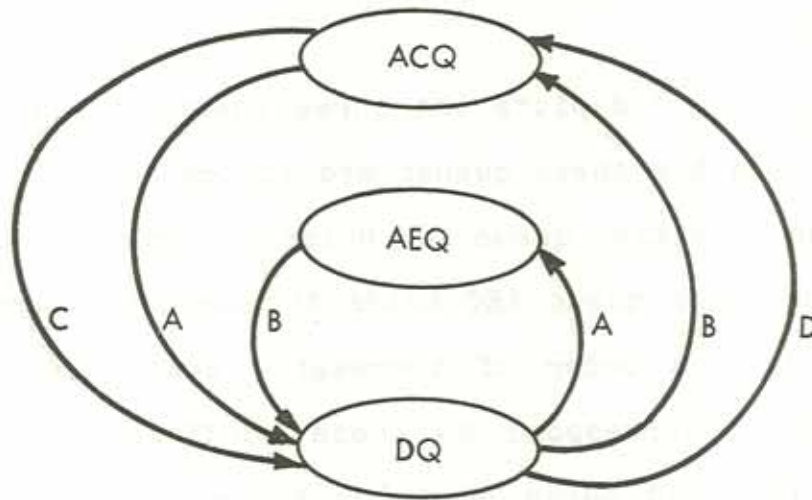
As we have seen earlier, we can look upon each job as a daemon with a condition and an expression. Conditions have the highest priority[12]. Thus, a condition check cannot be interrupted, whereas the running of an expression may be

interrupted by conditions and other expressions. Expressions are also run by the processor using the ED algorithm.

Figure 3.1 depicts the three queues which exist in the system, and how these queues are linked to one another.

The condition queue ACQ holds the active conditions and the expression queue AEQ holds the active expressions, both sorted in the order of increasing deadlines. The active expressions correspond to those conditions which returned true. The dead queue DQ holds those jobs which have been deactivated, whose expressions are on the expression queue (here jobs is used synonymously with conditions), and whose condition checks have returned false (here the jobs refer to dead expressions). Of course, no condition or expression can be on more than one queue. In fact, if a condition is on an active queue, the corresponding expression is on the dead queue, and vice versa.

At scheduled times, the conditions on the active queue are run, interrupting the expressions if necessary. If the condition check returns true then this condition is taken out of the active queue, and is placed in the dead queue, while its corresponding expression is activated and placed in the expression queue. The expressions are ordered with the earliest deadlines first and then, the first expression on this list is scheduled to execute by the processor. If



ACQ ~ Active Condition Queue

AEQ ~ Active Expression Queue

DQ ~ Dead Queue

A: Condition returned true, so deactivate condition and activate corresponding expression

B: Expression executed fully, so deactivate it and activate corresponding condition

C: DEACTIVATE command causing condition deactivation

D: ACTIVATE command causing condition activation

Figure 3.1 Queue System of the Scheduler.

the condition returns false, a new deadline is evaluated for it and then the condition is reinserted into the active queue using the ED algorithm. Whenever the processor completes the execution of an expression, that expression is taken out of the active queue, placed in the dead queue, and the corresponding condition is reactivated and placed in its proper place in the condition queue.

We notice that in running condition checks at scheduled times, and in executing expressions when corrective actions are needed, a certain amount of overhead is introduced. This consists of overheads incurred while switching from one condition to another condition, from an expression to a condition and back to the same or to another expression. Since the ED algorithm has been shown to be the optimal one for a single processor case, the results we shall obtain will reflect the best that the processor can do.

With this background we present in the next section, a model which will aid in answering the queries raised earlier.

3.2 A MODEL FOR PROCESSOR UTILIZATION

This section presents a model which can be used to answer the queries raised earlier about running m jobs on a single processor. The model which will be constructed is

only the first step in the process of arriving at a linearized result which will help our understanding of the power of a processor to perform real-time control. In particular, we wish our model to show the relationship between the number of tasks a processor can do, the bandwidths of these tasks, the cycle time of the processor and the complexity of its instruction set.

For any given control system, the processor spends part of its time in executing useful tasks such as running a condition check or executing an expression. At other times, it is either idle or it is executing such overhead tasks as the scheduling of conditions and expressions. Thus,

$$t_p = t_u + t_o + t_i \quad (1)$$

where, t_p is processor time, t_u is time spent in doing useful work, t_o is the overhead time, and t_i is the idle time.

Since we wish to obtain the limiting values for the number of tasks a processor can handle, and the bandwidths for these tasks, we shall assume that the processor is busy all the time. Thus, for total processor-time utilization, the idle time,

$$t_i = 0 \quad (2)$$

Hence,

$$t_p = t_u + t_o \quad (3)$$

For developing the model, we further assume that there

are m jobs with bandwidths B_1, B_2, \dots, B_m . Associated with each job are a condition and an expression. Thus, there are m conditions and m expressions. We also assume that t_p in equation (3) represents one cycle time which is defined by the following:

$$B_m = 1/k*t_p \quad (4)$$

where without loss of generality, B_m is taken to be the smallest bandwidth among the m tasks, and k is the sampling constant.

The activities performed by the processor in time t_p include the running of a number of condition checks, executing some expressions if required, and the overheads incurred in scheduling these conditions and expressions.

While the number of conditions and expressions to be executed in one cycle depends on the system under consideration, we can make a few generalizations. Normally, a task should be executed once during each cycle (m th task is referred here); however, this is true only if the corresponding daemon is in the active state all the time. It has been shown elsewhere[12] that often, it is necessary to deactivate a daemon through the DEACTIVATE statement. Let us assume that a daemon is active only during $100k_{c1}\%$ of the time, k_{c1} being a fraction. Then, on an average, in one cycle time the m th condition will be checked only k_{c1} times. A condition will not always return true. Thus, the number

of expression executions is usually less than the number of condition checks. Again, averaging over a large period of time, let us represent by k_{e1} the fraction of expression execution in one cycle time. Therefore, during time t_p , the m th condition is checked k_{c1} times and the m th expression is executed k_{e1} times.

Over a large period of time t , t being a multiple of the least common multiple of time periods t_1, t_2, \dots, t_m , where

$$t_i = 1/B_i \quad \text{for } i=1,2,\dots,m \quad (5)$$

tasks 1 through $(m-1)$ will be executed B_i/B_m times the number of executions of the m th task. Let

$$k_i = B_i/B_m \quad \text{for } i=1,2,\dots,m \quad (6)$$

Then statistically, if in a certain time t the m th task is executed once, the i th task will be executed k_i times in the same period t .

From this discussion we conclude that in one cycle time t_p , the i th condition is checked $k_i * k_{c1}$ times and that the i th expression is executed $k_i * k_{e1}$ times. We next deal with the useful task time, t_u and the overheads, t_o separately.

3.2.1 USEFUL TASKS

Let t_{cu} , t_{eu} denote the parts of t_u required to execute condition checks and expressions respectively, i.e.

$$t_u = t_{cu} + t_{eu} \quad (7)$$

Assuming that all the condition checks and the expression executions take the same times t_{cui} , t_{eui} respectively,

$$t_{cu} = S * k_{c1} * t_{cui} \quad (8)$$

$$t_{eu} = S * k_{e1} * t_{eui} \quad (9)$$

where $S = (\sum_{i=1}^m k_i)$.

Therefore,

$$t_u = S(k_{c1} * t_{cui} + k_{e1} * t_{eui}) \quad (10)$$

From equation (10) we see that in one cycle t_p , $(\sum_{i=1}^m k_i)k_{c1}$ condition checks and $(\sum_{i=1}^m k_i)k_{e1}$ expression executions are performed.

3.2.2 OVERHEADS

First, we shall look into the nature of the overheads. There are basically two kinds of overheads - those associated with condition checks, and those incurred while scheduling the expression executions. For the model derivation, the computer system overheads are ignored. Thus,

$$t_o = t_{co} + t_{eo} \quad (11)$$

where t_{co} , t_{eo} are respectively the overheads associated with the scheduling of conditions and expressions.

The overheads for a condition check or an expression execution are seen to be of two types: fixed and variable.

The fixed overheads are independent of the number of jobs in the active queue, while the variable overheads are directly proportional to the number of active conditions or the expressions. For some conditions and expressions, only fixed type overheads are incurred, while for other conditions and expressions, both the fixed and the variable kinds of overheads occur. The reason for this is that there exist situations which by their very nature force different types of overheads. For example, whenever a condition returns true, it is removed from the active condition queue until the corresponding expression has been completely executed by the processor. This is done in order to avoid "thrashing" as explained in Chapter 2. Thus, whenever such a situation occurs, we do not need to reorder the conditions in the active condition queue for the scheduling of the next condition; in this case, the next condition on the active queue, is executed next at its scheduled time. Another reason for different overheads in different situations is that when a condition returns false, no new expression is added to the active expression queue. For this case, the sorting of the expression execution deadlines on the active queue is not required.

Thus we see that there are situations when the sorting of deadlines is not required. When this is so, the only kind of overheads incurred is the fixed type which results

from the switchings between different conditions and expressions. Noting that at times there are only fixed overheads, and at others there are both fixed and variable overheads, we can write down the equations for the condition and the expression overheads:

$$t_{co} = k_c * S * k_{c1} * t_{cof} + (1 - k_c) * S * k_{c1} * t_{coi} \quad (12)$$

$$t_{eo} = k_e * S * k_{e1} * t_{eof} + (1 - k_e) * S * k_{e1} * t_{eoi} \quad (13)$$

where k_c (k_e) are the fractions denoting the number of time a condition returns true (false), t_{cof} (t_{eof}) is the fixed part of the condition (expression) overhead, and t_{coi} (t_{eoi}) represents the condition (expression) overhead when the variable overhead also occurs. t_{coi} , t_{eoi} are expressed in terms of fixed and variable parts of overhead as:

$$t_{coi} = t_{cof} + m * k_{c2} * t_{cov} \quad (14)$$

$$t_{eoi} = t_{eof} + m * k_{e2} * t_{eov} \quad (15)$$

where k_{c2} (k_{e2}) is a fraction denoting the number of active conditions (expressions) on the queue, and t_{cov} (t_{eov}) is the variable part of the condition (expression) overhead. Obviously,

$$k_c + k_e = 1 \quad (16)$$

Hence, combining equations (11) - (16), the total overhead may be expressed as the following:

$$t_o = k_c * S * k_{c1} * t_{cof} + (1 - k_c) * S * k_{c1} * (t_{cof} + m * k_{c2} * t_{cov}) \\ + (1 - k_c) * S * k_{e1} * t_{eof} + k_e * S * k_{e1} * (t_{eof} + m * k_{e2} * t_{eov}) \quad (17)$$

Simplifying the right-hand-side, we obtain

$$t_o = k_{c1} * S * t_{cof} + (1 - k_c) * S * m * k_{c1} * k_{c2} * t_{cov} + k_{e1} * S * t_{eof} + k_c * S * k_{e1} * k_{e2} * m * t_{eov} \quad (18)$$

Then, combining with equations (3) and (10),

$$t_p = S * [k_{c1} (t_{cui} + t_{cof}) + k_{e1} (t_{eui} + t_{eof})] + S * m * [(1 - k_c) k_{c1} * k_{c2} * t_{cov} + k_c * k_{e1} * k_{e2} * t_{eov}] \quad (19)$$

Therefore, from the defining equation (4) of t_p ,

$$B_m = 1/k * t_p$$

we obtain,

$$B_m = 1/k * S * [k_{c1} (t_{cui} + t_{cof}) + k_{e1} (t_{eui} + t_{eof}) + m \{ (1 - k_c) k_{c1} * k_{c2} * t_{cov} + k_c * k_{e1} * k_{e2} * t_{eov} \}] \quad (20)$$

The various times in equation (20) can be expressed in terms of the number of instructions required to implement these activities, and the cycle time of the processor.

Thus, we have the following relations:

$$\begin{aligned} t_{cui} &= k_{cui} * n_{cui} * T; & t_{eui} &= k_{eui} * n_{eui} * T; \\ t_{cof} &= k_{cof} * n_{cof} * T; & t_{eof} &= k_{eof} * n_{eof} * T; \\ t_{cov} &= k_{cov} * n_{cov} * T; & t_{eov} &= k_{eov} * n_{eov} * T. \end{aligned} \quad (21)$$

where the various k 's are some constants, n 's are the number of instructions, and T is the cycle time of the processor.

If T is taken to be the average instruction time, then the k 's can be equated to unity. Thus, assuming that T represents the average time of an instruction execution,

$$B_m = 1/k * S * T * [k_{c1} (n_{cui} + n_{cof}) + k_{e1} (n_{eui} + n_{eof}) + m \{ (1 - k_c) k_{c1} * k_{c2} * n_{cov} + k_c * k_{e1} * k_{e2} * n_{eov} \}] \quad (22)$$

Equation (22) represents the results of our model in a

very general form, and relates the bandwidths B_1, B_2, \dots, B_m of the m tasks, the number of tasks m , the average instruction cycle time T , and the complexity of the processor instruction set (through the various n 's).

In order to use this model, we have to determine the constants: $k, k_c, k_{c1}, k_{c2}, k_{e1}$, and k_{e2} .

Note that k is the constant value which determines the sampling rate, e.g. if B is the bandwidth, then the sampling rate = kB . From Shannon's theorem, k must be greater than or equal to two if the information is not to be lost. Usually, k is selected between 4 and 10 in order to ensure that the task is controllable. We will use $k=5$ in further refinements of the model.

k_c is the fraction that represents the times when conditions return true. To determine k_c , we assume that the disturbance to the equilibrium of the system is a sinusoid as illustrated in Fig. 3.2. It is also assumed that some action must be taken when the disturbance exceeds an acceptable limit. This limit will vary from system to system and will also depend on the type of disturbance. For the sake of simplicity in the model, it is assumed that the tolerance limit is half the amplitude of the disturbance sinusoid.

From Figure 3.2 it is clear that during one cycle, the

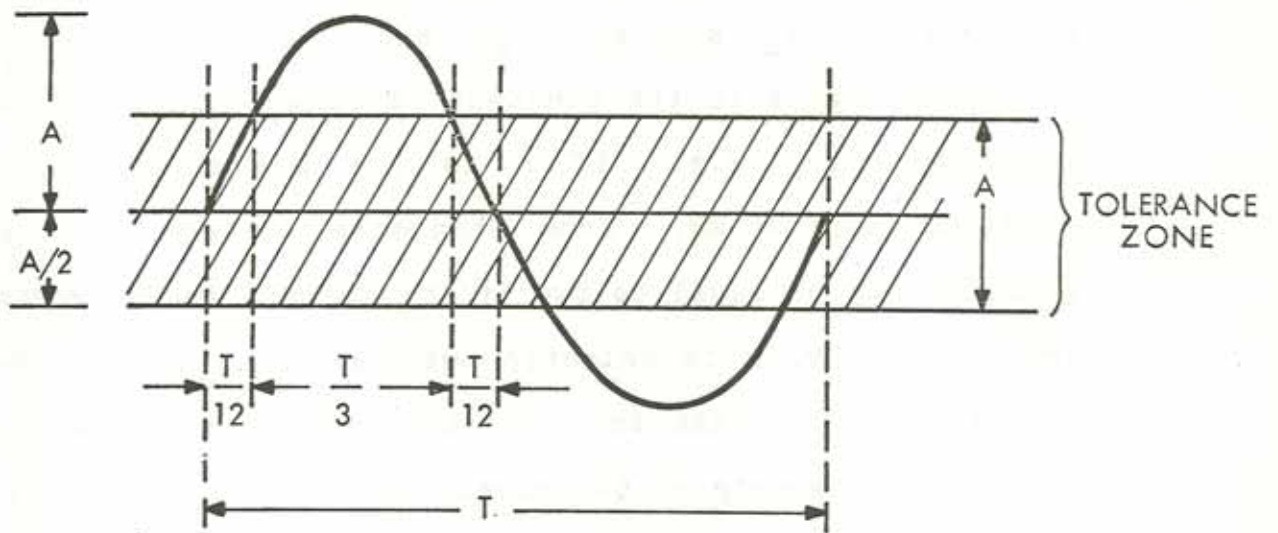


Figure 3.2 Assumed Disturbance Waveform for Determining the Factor, k_c .

disturbance exceeds the tolerance limits for 2/3rds of a cycle. Usually, when corrective action is being taken, sampling of the corresponding condition is halted to avoid thrashing, the sampling being resumed when the corrective action has been completed. Thus, the condition will return false for 1/3rd (or greater) of the time. We assume that this factor is one half, i.e. the conditions will return true only half of the time, the other half of the time they will come out false. Then,

$$k_c = 1/2.$$

k_{c1} is the fraction representing the number of conditions that will be run in unit time (one cycle). In what follows we are concerned only with the m th task. The unit time refers to t_p defined earlier in equation (4). Also, we will assume that the only tasks that exist are the ones with bandwidth B_m . Normally, if there are ten daemons in the system, we may expect, on the average, two of them to be deactivated through the DEACTIVATE command. For these daemons, neither the expressions nor the conditions will be active. Hence,

$$k_{c1} = 0.8.$$

k_{c2} represents the fraction of the total number of jobs, on the active queue. Since all the jobs are assumed to be identical and we have seen that the conditions return true only half the times, only half of the jobs

corresponding to these conditions will be on the active queue. Hence

$$k_{c2} = k_c * k_{c1} = 0.4.$$

Since the conditions return true only half the times, we assume that only those expressions corresponding to these conditions are run in unit time. Therefore,

$$k_{e1} = 0.5 * k_{c1} = 0.4.$$

Since k_{e2} is the fraction representing the number of active expressions on the queue,

$$k_{e2} = (1 - k_c) k_{c1} = 0.4.$$

We now have the following evaluated constants:

$k = 5$	number of samples per period
$k_c = 0.5$	percentage of conditions firing <u>true</u>
$k_{c1} = 0.8$	active task time/total time for conditions
$k_{c2} = 0.4$	active condition/total conditions
$k_{e1} = 0.4$	active task time/total time for expressions
$k_{e2} = 0.4$	active expressions/total expressions (23)

3.3 SIMPLIFICATIONS AND ANALYSIS OF PROCESSOR POWER

In this section, further simplifications of the model will be presented following which various approximate relationships involving the real-world and the processor parameters will be given, and also the simplified models will be analysed. The starting point is the equation (22).

If all the m tasks have identical bandwidths, the model simplifies into a much more compact equation. Thus, equating all the bandwidths to B , and substituting equations (23) into (22), we obtain the following:

$$B = 2.5/T[m^2(2n_{cov}+n_{eov})+m(10n_{cui}+10n_{cof}+5n_{eui}+5n_{eof})] \quad (24)$$

or converting back to the different times involved:

$$B = 2.5/[m^2(2t_{cov}+t_{eov})+m(10t_{cui}+10t_{cof}+5t_{eui}+5t_{eof})] \quad (25)$$

In equations (24),(25) n 's refer to the number of programming instructions required for respective tasks and the t 's refer to the various times required for different activities engaged in by the processor. Equation (24) may be rewritten as:

$$B = 2.5/T[a*m^2+b*m] \quad (26)$$

where,

$$a = 2n_{cov} + n_{eov}, \text{ and}$$

$$b = 10n_{cui}+10n_{cof}+5n_{eui}+5n_{eof}.$$

Equation (26) is alternatively expressed as:

$$B = 2.5/[aa*m^2 + bb*m] \quad (27)$$

where $aa=a*T$, $bb=b*T$.

Table 3.1 presents representative figures of a , b , aa , and bb for PDP11/45 and Intel 8080 processors. There are two entries for Intel in the table. One of these refers to the brute force translation of code written for the PDP processor. This is termed Intel Emulated. The other is the Efficiently Coded Intel, which assumes that an assembler is

available. How these figures are obtained is explained next.

The figures listed in Table 3.1 correspond to or are directly computed from the various overhead and condition-expression times. For example, the parameters t_{cof} , t_{cov} , t_{eof} , t_{eov} , are the overheads. These were evaluated for the two processors mentioned above, by following the scheduling algorithm written in the macro control language on the DELPHI system[16]. It was found that both the condition and the expression overheads included a queueing operation which contributes the variable part of the overhead. It was a simple matter of following the course of the scheduling algorithm, considering various possible paths (whether a condition returned true, or whether a job was already on an active queue, etc.), in order to calculate the overhead parameters. For evaluating t_{cui} , t_{eui} , an estimate was made on the basis of some attempts to program a particular expression or a condition (representative), using the PDP 11/45 code and then making appropriate modifications to obtain these figures for the Intel 8080 processor.

To arrive at all of these figures, PDP code was used. (In this thesis, whenever reference is made to PDP, it should be taken to mean PDP 11/45, unless otherwise stated.) For converting the PDP figures to the corresponding Intel

Table 3.1 Comparison of PDP 11/45 and Intel 8080 codes

	PDP 11/45 (DELPHI)	Intel 8080 emulated	Efficiently coded Intel
a (no. of instructions)	70	375	115
b (# of instructions)	6000	27,500	10,000
aa (msec)	0.070	0.750	0.225
bb (msec)	6.0	55.0	20.0
al	28	150	46
bl	2400	11,000	4000

8080 values, the following procedure was used. First, all the different types of instructions in the PDP code of the scheduler were listed. Next, assuming that no assembler was available for the Intel processor, code was written for it to simulate the PDP instructions. Then, the same was done for a few of the instructions assuming that an assembler was available. The first kind of coding resulted in the Emulated Intel figures, while the second type of coding along with the consideration of a few other factors like Intel having only 8 eight bit registers contrasted with PDP's sixteen bit registers, resulted in the Efficiently-Coded Intel figures.

An example of this instruction translation follows:
Consider the PDP instruction

MOV NAME+K1,X+K2(Rn)

Emulated Intel coding for this is quite inefficient-

```
LXI H,X
DAD D
XCHG
SHLD TEMPR4
LXI K2
DAD D
LBH
LCL
LXI D,0
LXI H,NAME
DAD D
LXI D,K1
DAD D
LEM
INX H
LDM
LHB
LLC
```

```
LME
INX H
LMD
LHLD TEMPR4
XCHG
```

This inefficient coding requires 37 bytes, and takes 106 μ s to execute. As against this, compare the following code, written expressly for Intel 8080, assuming an assembler was available-

```
LHLD NAME1
XCHG
LHLD NAME2
LME
INX H
LMD
```

The last code takes up only 10 bytes, and executes in just 32 μ s.

Using this approach, we were able to evaluate the figures listed in Table 3.1. These figures were subsequently used in order to construct graphs showing the relationship between the bandwidths of the jobs, and the number of jobs.

Figure 3.3 is a plot of the bandwidth of each job versus the number of tasks for PDP11/45 and Intel 8080. It can be seen that for all cases, as the number of tasks increases, the bandwidth falls with some kind of an inverse relationship. Figure 3.4 is the same plot but on a log-log scale. It can be ascertained from this plot that for m less than ten, the relationship is almost linear. Thus, we may

drop the higher order m^2 term from the model for number of jobs less than ten, and we have a simple inverse relationship between the number of jobs and their bandwidth.

Next, we introduce the notion of the bandwidth of a processor, B_p . This is related to the processor cycle time T through

$$B_p = 1/T$$

Then by rewriting (26),

$$B_t/B_p = 1/[a_1*m^2+b_1*m] \quad (28)$$

where $B_t=B$, the bandwidth of each task and

$$a_1=a/2.5, b_1=b/2.5$$

A plot of B_t/B_p versus m would be identical to the plots in Figures 3.3 and 3.4. It is seen both from these plots and from mathematical calculations that the m^2 term has little effect in the model of equation (28) if the number of tasks is less than 10. Hence, as an approximation, one can write -

$$B_t/B_p = 1/[b_1*m] \quad (29)$$

This last equation can be alternately expressed as any of the following:

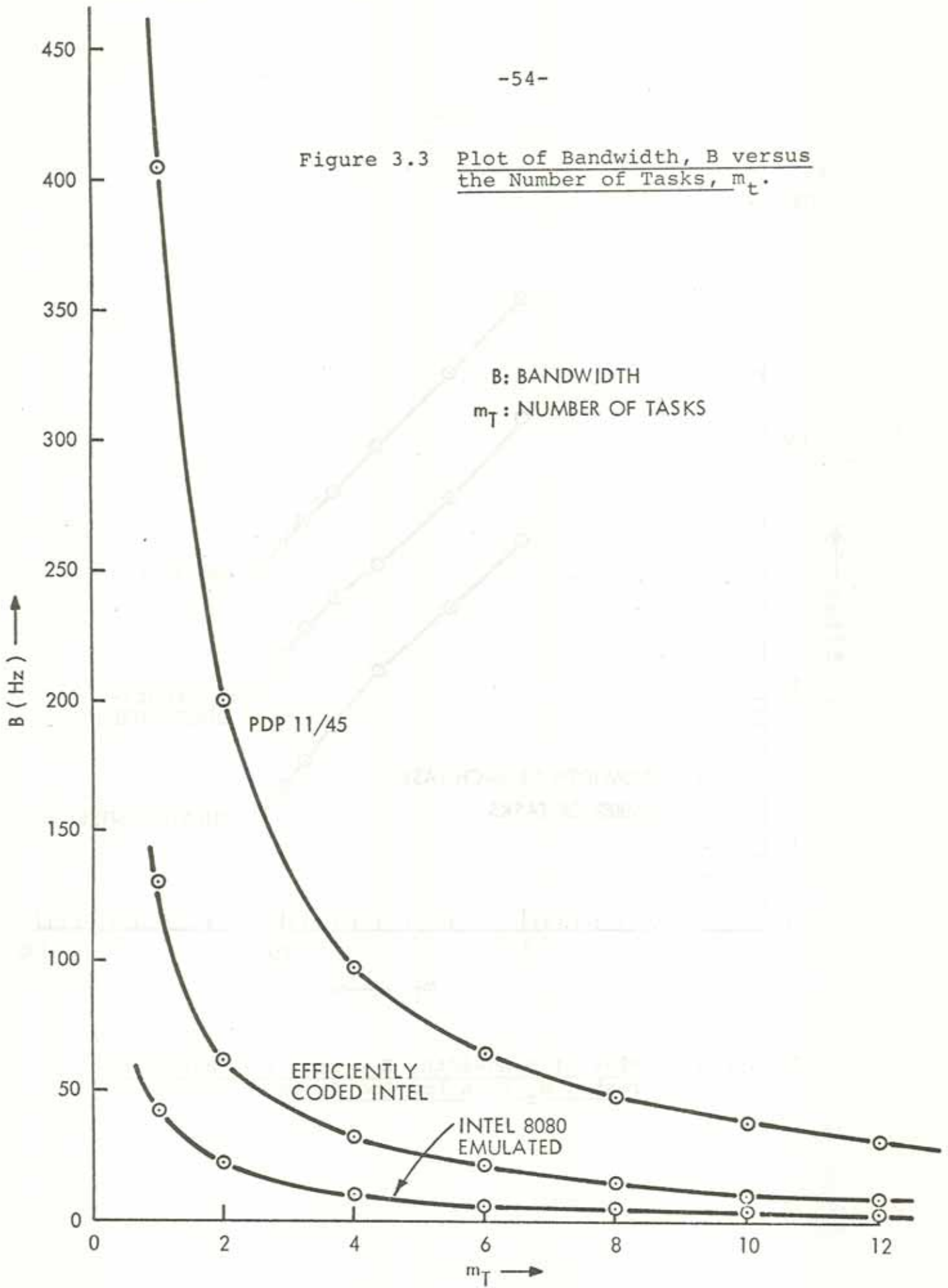
$$B_t*m_t = K \quad (30)$$

$$B_t*m_t = B_p*K_1 \quad (31)$$

$$B_t*m_t = B_p(E*K_2) \quad (32)$$

where $m_t = m$, the number of tasks, (the subscript t will be used for the real world quantities),

Figure 3.3 Plot of Bandwidth, B versus the Number of Tasks, m_t .



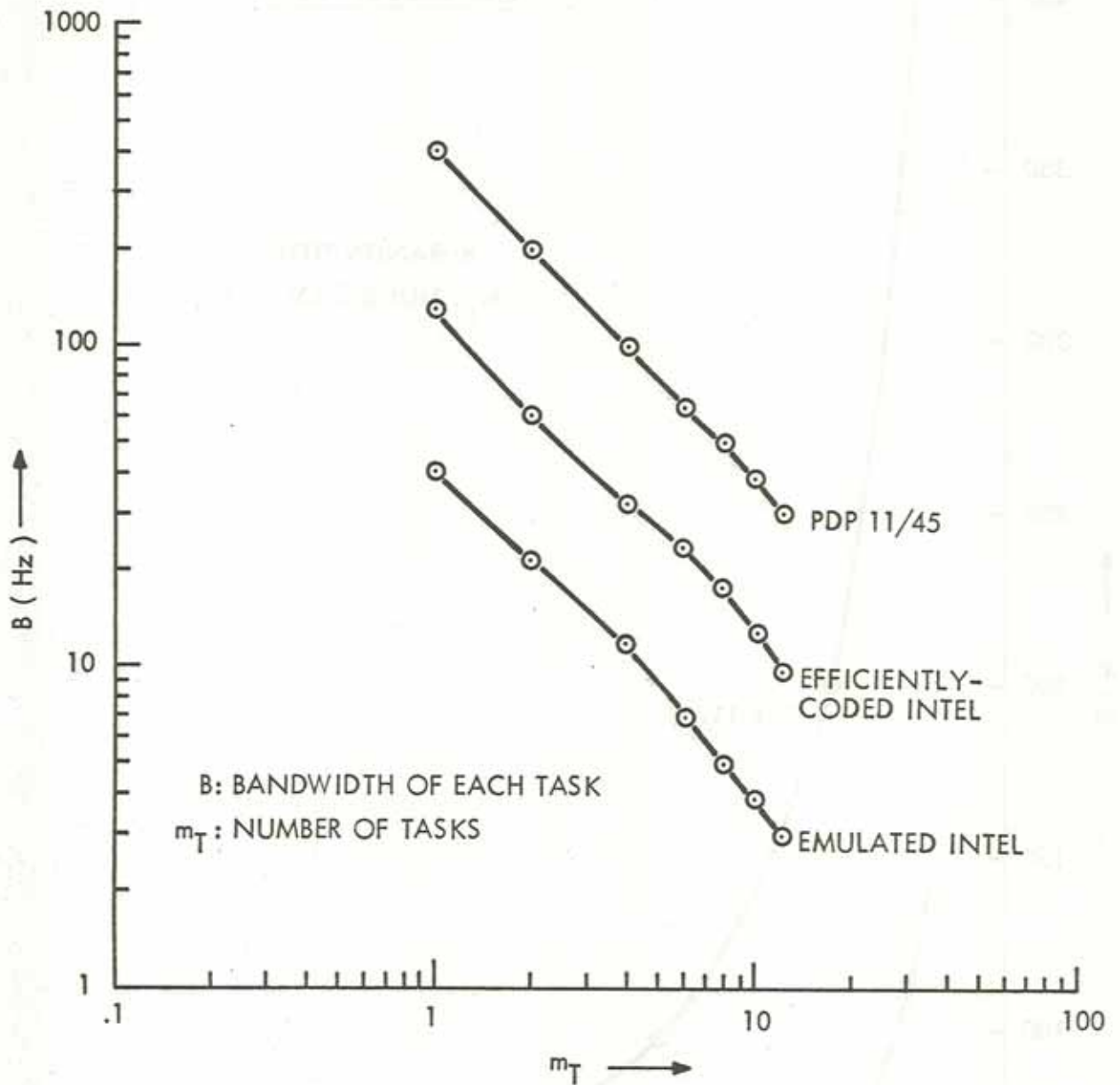


Figure 3.4 Plot of Bandwidth, B versus the Number of Tasks, m_t on a Log-Log Scale.

$K = B_p/b_1$, $K_1 = 1/b$,

and K_2 is a constant to be used in the model when the efficiency of coding is 100%, while E denotes the efficiency of coding and takes values between 0.20 and 0.95, B_t is the task bandwidth and B_p is the processor bandwidth.

Equation (32) represents the relationship between the real world quantities on the left-hand-side and the processor quantities on the right-hand-side. By introducing the coding efficiency factor E , we have eliminated from the model the parameter corresponding to the complexity of the instruction set, which is now reflected in the coding efficiency factor. Table 3.2 shows these various factors for the PDP11/45 and the Intel processors.

Using equation (32), a plot of B_t/B_p versus m_t is drawn on a log-log scale in Figure 3.5. This is essentially a linear plot with the lines rising up as shown when the coding efficiency improves. This is, of course, an expected result since improved code will result in more computing power.

Usually, the bandwidths of different tasks are not the same, hence in order to predict the processor requirements for a general set of tasks, we must follow the general model of equation (22). As we did for the case of all tasks having identical bandwidths, we may drop the higher order

Table 3.2 Coding factors for PDP 11/45 and Intel 8080

	PDP 11/45 (DELPHI)	Emulated Intel 8080	Efficiently coded Intel
k (Hz)	415	45	125
T (sec)	1	2	2
K_1 (Hz/MHz)	415	90	250
B_p (MHz)	1	0.5	0.5
E (%)	0.92	0.20	0.56

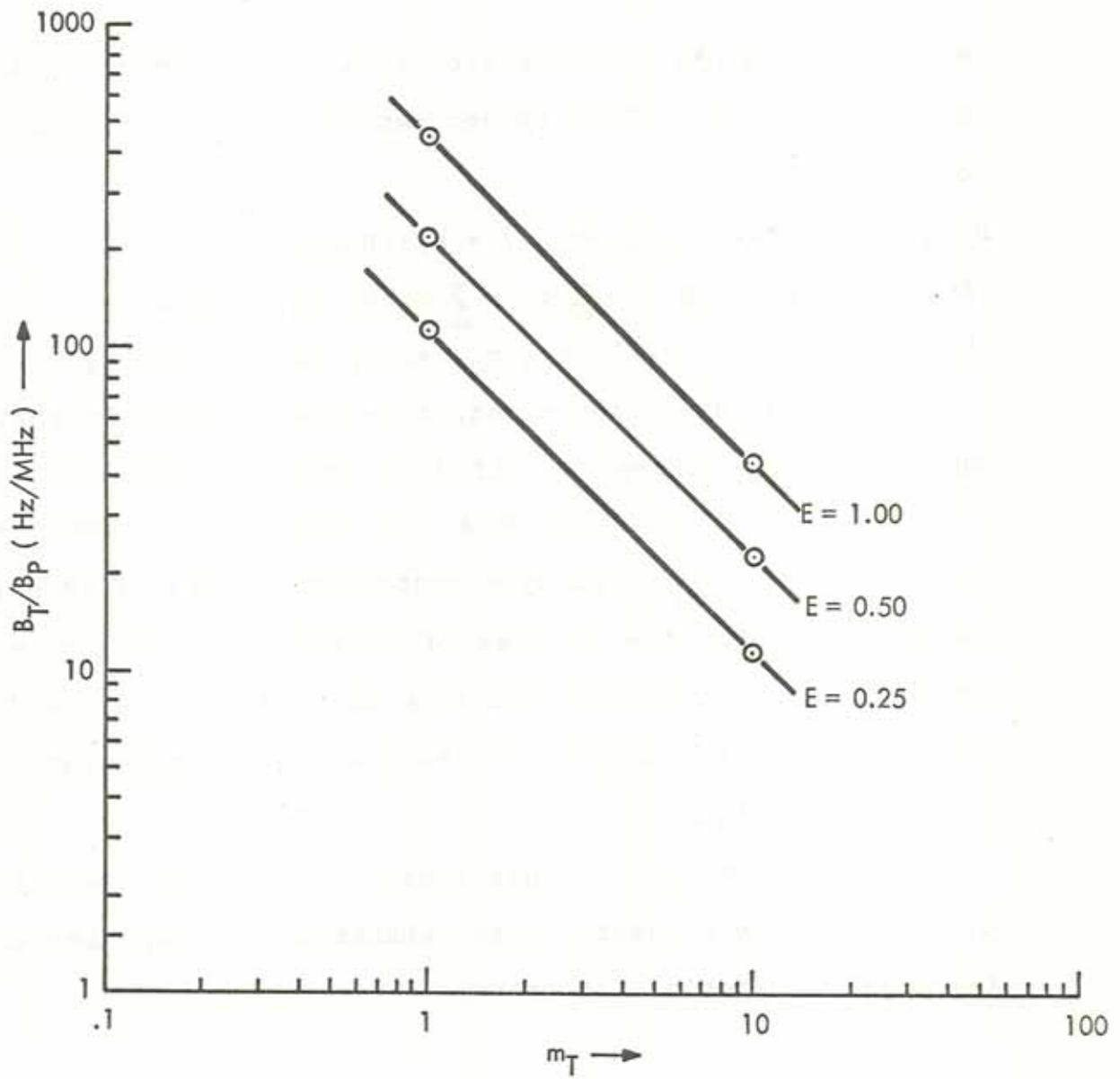


Figure 3.5 Plot of Normalized Bandwidth, B_t/B_p versus m_t with the Approximate Model - m_t less than ten.

term involving $m \cdot S$ from equation (22) if the number of tasks is less than ten. This approximation yields the following equation:

$$B_m = 1/k \cdot S \cdot T \cdot [k_{c1}(n_{cui} + n_{cof}) + k_{e1}(n_{eui} + n_{eof})] \quad (33)$$

Then, substituting $S = \sum_{i=1}^m k_i = \sum_{i=1}^m B_i / B_m$ into (33),

$$(B_1 + B_2 + \dots + B_m) = 1/k \cdot T \cdot [k_{c1}(n_{cui} + n_{cof}) + k_{e1}(n_{eui} + n_{eof})] \quad (34)$$

The right-hand-side of equation (34) reduces to that of equation (31). Therefore, if the number of tasks is less than ten, we can replace a given system of m tasks with bandwidths B_1, B_2, \dots, B_m by a system of m tasks of identical bandwidths B for the purpose of determining the processor power in the context of real-time control. The bandwidth B is computed as the average of the m different bandwidths:

$$B = (B_1 + B_2 + \dots + B_m) / m \quad (35)$$

Use will be made of this result in Chapters 4 and 5 for determining the processor requirements of the experiments of the Control Robotics Laboratory.

CHAPTER 4

MODULAR LABORATORY FOR CONTROL ROBOTICS

Automation in an academic laboratory is a concept which is not new. People have been experimenting with control for a long time. However, as we have said earlier, all of these approaches have been predominantly classical. As new approaches to computer control are being developed, they must be reflected in the design of academic laboratories as well.

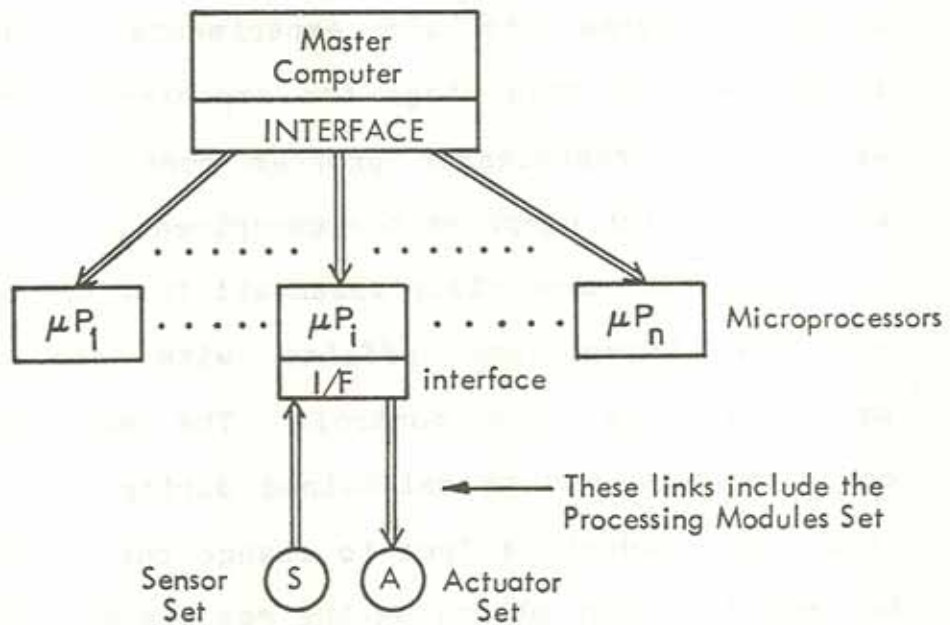
The modular laboratory to be described in this chapter, attempts to combine the control robotics approach discussed in chapter 2 using microprocessors for control and at the same time, present the students with a collection of fun-experiments the functioning of which requires complicated as well as simple control of physical processes.

In this chapter, we will present some criteria for the selection of experiments for the modular laboratory. Next, we will modularize the experiments considering only the sensors and actuators required for them, after which we minimize the number of modules (sensors and actuators) by using a matrix approach. Finally, we will direct our attention to the power, resolution requirements of the experimental modules.

4.1 OVERALL SETUP OF THE LABORATORY

In this section, we shall describe the overall organization of the laboratory setup. Figure 4.1 broadly illustrates the components involved.

For a particular experiment that a student wishes to conduct, he will select the components required. These are selected from a set of processing and conversion modules to be described in the next sections. Guidelines for the selection are presented in Chapter 5 which lists each experiment with recommended set of modules necessary to perform them. Having selected the modules, the student will proceed to connect them up using the required interfaces to obtain the physical setup for the experiment. Either the student will already have written and stored a control program written in a macro control language or some other appropriate control language, on a master computer system or he can proceed to do so now. Once the program is on a master computer system like DELPHI, a time-sharing PDP11/45 system in the department of Electrical Engineering and Computer Science at M.I.T.[16], a microprocessor such as Intel 8080 is "charged up" with the translated version of the control program using an appropriate communications channel between the master computer and the microprocessor



EACH MICROPROCESSOR μP HAS AN INTERFACE WITH LINKS TO ACTUATORS AND LINKS FROM SENSORS

Figure 4.1 Overall Setup of the Proposed Laboratory.

and suitable interfaces. Now the student can connect the microprocessor whose programmable memory contains the control program, to the experimental setup through an interface. At this stage the experiment can be performed with the microprocessor program controlling the physical processes which comprise the experiment.

It is not absolutely essential that the student perform all the experiments offline with the microprocessor exercising the sole control. The microprocessor-master computer link can be maintained during the experiment and then, the student is free to change the control program as he sees fit after observing the results of the experiments.

4.2 MODULAR LABORATORY

It was decided to select the experiments and the components needed to perform them, in a manner that would permit the modularization of the laboratory. This will also allow the students to choose from a large variety of experiments without too great an expenditure in equipment. Thus, the standardized modular laboratory will do away with a lot of specialized equipment, but it will allow the students to perform a variety of experiments with the modular components.

In this section, we shall study the selection criterion

for the experiments, the hardware modules, and the minimization of the modules with the maximization of the number of experiments.

4.2.1 THE EXPERIMENTS

It was decided that the laboratory should consist of such experiments that will stimulate interest, and provide situations where one can apply intuitive control strategies. We started with a long list of possible experiments. To this list we continually added some experiments, and sometimes a few experiments were dropped from the list. Part of this list is presented in Table 4.1. It is our contention that the experiments which involve physical movement are the most interesting ones since everyone identifies with motion. Indeed, motion suggests excitement and excitement is interesting. It was because of this belief that we ruled out experiments that involve no motion such as detection of heat, or putting out of fires by automated sprinkler systems. Furthermore, we decided to consider only those experiments which are associated with games since everybody likes some form of game or other.

Exciting experiments, however, can sometimes be extremely difficult to reproduce in the laboratory. For an example, consider playing out a game of basketball using

Table 4.1 A partial list of experiments considered for the Control Robotics Laboratory.

- * 1. Balancing an inverted pendulum
2. A riderless bicycle
3. Controlling the motion of a sailboat
- * 4. Playing a recorder
- * 5. Turtle war
- * 6. Playing a violin
- * 7. Playing a guitar
- * 8. Playing a slide-flute
9. Controlling the flight of a model aeroplane
10. Resetting a clock physically to correct time
- * 11. Balancing a metal ball with a solenoid
- * 12. Table soccer
13. Billiards
- * 14. Space war
- * 15. Simulated billiards
- * 16. Simulated ping-pong
17. Peanut-butter-and-jelly sandwich making machine
18. A mechanical arm carrying a glass of water

Detailed descriptions of experiments marked with an asterisk will be found in Chapter 5.

mechanical men controlled by a computer. Such a simulation involves tasks of such great complexities that one would not even need the discussion of Chapter 3 to realize that the computational power required of a processor would be tremendous, and that the hardware required for the experiment would be highly specialized and sophisticated. It was for reasons such as this that we decided that experiments such as controlling the path of a sailboat, a riderless bicycle, etc. exciting as they are, are very complex yet too difficult to perform in a laboratory. In conclusion, it was decided to include in the laboratory those experiments which are not only interesting but can also be reasonably implemented through a set of standardized modules. Also in the light of the Control Robotics approach described in Chapter 2, it was decided that the experiments, as far as possible, should allow a student to use intuitive strategies of control.

4.2.2 HARDWARE MODULES

Once the list of experiments to be modularized was finalized, the experiments were further examined to see what hardware was needed to perform them. Since the experiments were going to be modular in nature, the hardware was of course assumed as modules.

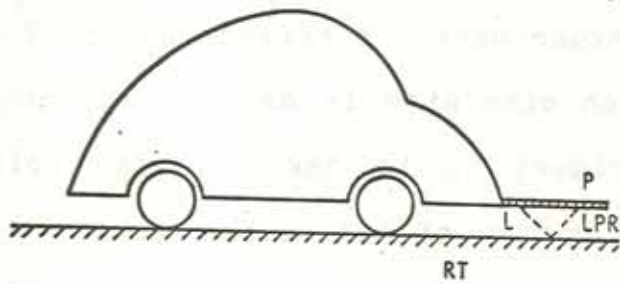
Two distinct kinds of hardware modules were identified. The first kind consisted of conversion modules which include all the sensors and the actuators. The sensors would be used to detect the presence or the absence of certain conditions in the physical world, and the actuators would be used to alter the physical environment in order to effect control on it. All that the sensors and the actuators can do is to transform the physical world quantities into electrical signals and vice versa respectively.

In order for the physical world to communicate with the computing system, a second kind of hardware modules are needed. These are the processing modules, which include analog to digital converters, digital to analog converters, analog multiplexers, motor controllers, etc. These modules would be used to transform electrical signals from the sensors to signals that can be used by the computing system, and to transform the digital signals of the computing system to those that can operate the actuators. A common set of processing modules would include various interfaces and the associated communications channels.

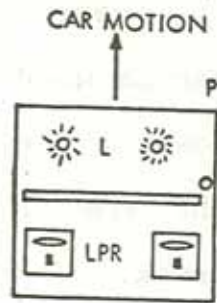
Next, we shall describe the modularization process for a particular experiment. The selected experiment is: Making a Car Follow a Laid out Track.

Description of the Experiment: A remote controlled toy car is used for this experiment. A reflecting track is laid down on the floor which otherwise is assumed nonreflecting. The aim of the experiment is to make the car follow the track. If during the motion of the car travel it goes off course, this must be sensed and proper turning directions must be sent to it so as to correct its course. Figure 4.2 illustrates this setup.

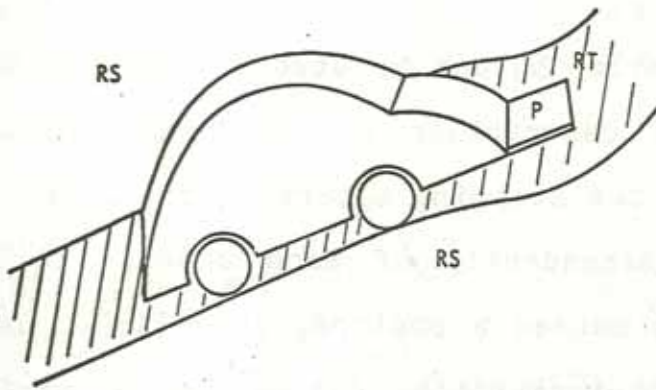
A mechanical turtle[15] can be used as the car in this experiment. A brief description of the turtle appears in Chapter 5. This has two stepping motors which can be run in either direction independently of each other. Under the base of the "car" is housed a photoresistor light and lens assembly. The two photoresistors act as the sensors. The light and lens assembly is such that when the car is on the track fully, both the resistors are equally illuminated. This corresponds to minimum resistance of the photoresistors. If the car deviates from the track (we assume that the deviations are such that at least part of one photoresistor is illuminated), then it can be determined which way the car is going off the track by monitoring the electrical resistances of the two photoresistors. Thus, if the car is moving off to the left, the left motor speed can be increased and the right motor speed decreased so as to make the car turn to the right and thus bring it back on



1. CAR AND REFLECTING TRACK
(Above and Below)

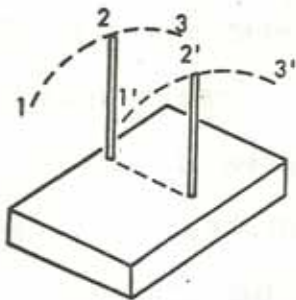


2. THE PLATFORM

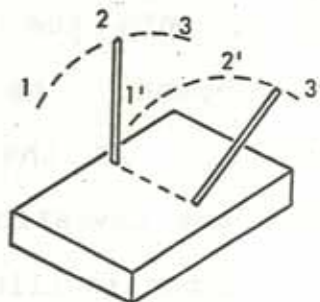


KEY:

- L: LIGHT SOURCE
- LPR: LENS/PHOTORESISTOR ASSEMBLY
- O: OBSTRUCTION TO PREVENT DIRECT ILLUMINATION OF PHOTORESISTORS
- P: PLATFORM HOUSING L's AND LPR's
- RT: REFLECTING TRACK
- RS: ROUGH SURFACE



2-2' STOP POSITION



2-3' FORWARD RIGHT TURN

3. TWO STICKSHIFT CONTROL POSITIONS

Figure 4.2 An Example Experiment (Car to Follow a Laid-Out Track)

track.

The exact control strategy will not be as naive as this, but it suffices for the sake of illustration.

Module Requirements For The Experiment :

Conversion Modules:

Sensors:

1. Two photoresistors, with lens and concentrated light source assemblies.

Actuators:

1. Two stepping motors.

Processing Modules:

1. Two voltage level converters (voltage across the photoresistors to voltages accepted by the multiplexer).
2. Analog multiplexer.
3. One analog-to-digital converter.
4. Two digital-to-analog converters.
5. Two voltage level converters (output voltage of the D/A converters to voltages accepted by the motor controllers).
6. Two motor controllers.
7. One transmitter and one receiver for remote sensing and actuating.

4.2.3 MINIMIZATION OF MODULES

Once the chosen experiments are modularized using the process described in the previous section, we must tackle the task of minimizing the number of different modules so as to be able to perform a maximum of experiments. The scheme that we shall employ for this minimax problem involves the construction of a few matrices. The rows in these matrices will correspond to the hardware modules of the set selected, while each column will correspond to an experiment. Then, depending on certain factors to be discussed in this section, these matrices will be altered for the minimization problem.

As a first approach no considerations are applied as to the resolution or the value ranges of the modules. On the basis of the modularization in Section 4.2.2, a matrix is created. The rows and columns of this matrix are defined as above. For each column, the intersection corresponding to a conversion module is marked with an X if that experiment utilizes this module. We consider only the conversion modules, because it may be readily seen that the type and the number of processing modules depend entirely on the conversion modules. Next, we look over the matrix and see if there are any modules which have a sparseness in X's, i.e. which are used by very few experiments. If this is

the case, the module is dropped from the set of chosen hardware modules, and the experiments which make use of such modules are either dropped out, or substitute modules which do not have sparse X's in the matrix, are selected. If one is willing to make a compromise between the sparseness of a module's usage and the interesting nature of the experiments involved (how is one to measure this "interest factor"?), these modules and the experiments can stay. This approach, though highly intangible, is effective.

After playing around with some matrices, it was decided to stay with a fixed number of experiments. No justification was provided for this decision. The set of experiments that were selected appears in Table 4.2. In tables to follow, each of the experiments in Table 4.2 will be referred to by the numeral preceding it.

Table 4.3 illustrates one of the matrices that were constructed during this project. We can see from this table that the stepping motor is a module that is used very frequently. This is desirable because all the experiments which use stepping motors can share them from a common pool which does not require separate motors for separate experiments. Also seen in the table is the sparseness of the strain gauge module. By our criteria in the preceding discussion, the strain gauge as a sensor should be dropped from consideration, as should the corresponding experiments

Table 4.2 List of experiments for Tables 4.2 - 4.4, 4.7

-
1. Turtle war
 2. Metal ball balancing
 3. Car to follow a laid-out track
 4. Slide-flute
 5. Recorder
 6. Inverted pendulum
 7. Guitar
 8. Violin
 9. Yo-yo
 10. Stilt walker
 11. Paddle pool
 12. Table soccer
 13. Tilt maze
 14. Space war
 15. Simulated ping-pong
 16. Simulated inverted pendulum
 17. Simulated billiards

Descriptions of all these experiments are given in Chapter 5.

Table 4.3 Preliminary matrix for minimization of modules

	Experiments																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
<u>Sensors:</u>																		
Photoresistor	x	x	x							x								
Contact swtch	x					x					x	x	x	x				
Tone detector				x	x		x	x										
Strain gauge								x	x		SPARSE							
Potentiometer				x	x	x		x		x	x	x	x			x	x	x
TV camera	x	x	x	EXPENSIVE						x		x	xx	x	& DIFFICULT TO IMPLEMENT			
<u>Actuators:</u>																		
DC motor (or stepping motor)	x		x	x	x	x		x		x	x	x	x		x	x	x	
Solenoid	x	x			x		x	x	x	x					x			

which require strain gauges. However, these experiments are quite "interesting", hence we decided to keep the strain gauge module, and looked to see if some other modules could not be replaced by strain gauges which are also very convenient to use as sensors. We discovered that we needed to locate the position of a ball on a surface for a number of experiments. We had decided earlier to use TV cameras to detect this position. After some thought it became evident that this would not only be expensive, but that it would also generate a lot of difficulties in implementation. Hence, in the light of the matrix of Table 4.3 we decided to use the strain gauges themselves to detect the position of the ball on a surface, thus keeping alive the strain gauge and the corresponding experiments in the matrix. In fact, finally, we decided to use special resistance sheets to detect the position of the ball. This is described in some detail in Chapter 5. Table 4.4 presents the final version of the matrix.

Once the matrix was beyond this stage of "minimization", we decided to consider the voltage, current, power and resolution requirements of the conversion modules for further refinement. These results were drawn up in another matrix, illustrated as Table 4.5. From this matrix it was easy to select a common set of modules which can perform all the experiments on the matrix. For example, if

Table 4.4 Final matrix of experiments and conversion modules

Experiments

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
<u>Sensors:</u>																	
Photoresistor	x	x	x						x								
Contact switch	x					x					x	x	x	x			
Tone detector				x	x		x	x									
Potentiometer				x	x	x		x		x	x	x	x				
Resistance sheet											x	x	x				
Strain gauge								x	x		o	o	o				
<u>Actuators:</u>																	
Motors	x		x	x	x	x		x		x	x	x	x		x	x	x
Solenoids	x	x			x		x	x	x								x

OPTIONAL USE OF A MODULE IS INDICATED BY 'O'

an experiment needs two motors of 0.2 HP, and there is an experiment which needs higher powered motors of, say 0.4 HP, then the selected set of modules should include motors of 0.4 HP, while the number of motors selected depends on how many experiments will be run at the same time. This permits the correct functioning of all the experiments. Table 4.6 lists the finalized set of conversion modules with their requirements. Table 4.7 lists the maximum number of modules required to do one experiment, and to do 3 to 4 experiments at a time.

4.2.4 PROCESSOR REQUIREMENTS OF THE EXPERIMENTS

In Chapter 3, we had developed a model for analysing the processor power in the context of real-time control tasks. Here, we show how to use the model to predict the processor requirements for the experiments in the Control Robotics Laboratory. It will be shown in Chapter 5 that all the experiments break up into jobs which number less than ten. Therefore, we may use the simplified model given by equation 3.31 which is rewritten below:

$$B_t * m_t = B_p * K_1 \quad 3.31$$

This will be combined with equations 3.34 and 3.35 which are reproduced below for convenience:

$$B_1 + B_2 + \dots + B_m = 1/k * T * [k_{c1}(n_{cui} + n_{cof}) + k_{e1}(n_{eui} + n_{eof})] \quad 3.34$$

Table 4.5 Power-Resolution Matrices

	Experiments																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
<u>ACTUATORS</u>																	
Motor (#)	4	2	2	2	1	2	6				2	2					
(power, HP)	1/5	1/5	1/7	1/7	1/7	1/4	1/5				1/4	1/10					
(resolution, bits)	8	8	8	6	6	8	6				6	8					
Motor (#)										2	2	2	2	1	2	2	3
(torque, Nt-m)										5	1	5	4	.1	.1	.1	.1
(resolution, bits)										8	8	4	6	.8	.8	.8	.8
Solenoid (#)	2	1			8	78	4	1	2					3			
(strength, Nt)	5				2.5	10	10	10	10					1			
<u>SENSORS</u>																	
Photoresistors (#)	2	2	2							2							
(resolution, bits)	1	8	8							8							
Contact switch (#)	4					4					2	2	1	3			
Tone detectors (#)				2	2		2	2									
(resolution, bits)				10	10		10	10									
Potentiometers (#)				1	1	4	4	4		2	2	2	2	1	2	2	3
(resolution, bits)				8	6	8	6	6		8	8	6	6	.8	.8	.8	.8
Resistance sheet (#)										1	1	1	1				
(resolution, bits)										8	8	8	8				
Strain gauge (#)								1	1		4	4	4	4	4	4	4
(resolution, bits)								6	8		8	8	8	8	8	8	8

refers to the number of each module required.
 The underscored entries for strain gauge are the optional uses for strain gauge.
 Resolutions of solenoids and contact switches are one bit each

Table 4.6 Final set of Conversion Modules and their requirements.

Sensors:

	Maximum resolution required (bits)
Potentiometer	10
Contact switch	1
Tone detector	10
Photoresistor	8
Resistance sheet	8
Strain gauge	8

Actuators:

	Maximum requirements
DC motor	1/4 HP, 10 bits
Stepping motor	1/4 HP, 1K steps/sec
Solenoid (push-type)	120 oz lift at 1/8 inch
Solenoid (pull-type)	80 oz lift at 1 inch
Solenoid switches	3 oz lift at 1/2 inch
Stepping motors for software experiments	50 oz-inch holding torque

Table 4.7 Module requirements for the laboratory.

<u>Conversion modules</u>	for 1 experiment	for 3-4 simultaneous experiments
<u>Sensors:</u>		
Potentiometers	4	12
Contact switches	4	12
Tone detectors	2	6
Photoresistors	2	6
Resistance sheets	1	3
Strain gauge	1	2
<u>Actuators:</u>		
DC motors	6	10
Stepping motors	4	10
Solenoids (push-type)	78	90
Solenoids (pull-type)	2	3
Solenoid switches	3	3
Stepping motors for software simulation expts	3	6
<u>Processing modules</u>		
Resistance to voltage converters	5	12
Pressure to voltage converters	1	2
Voltage level changer (actuator side)		
-for push-type solenoids and solenoid switches	78	90
-for pull-type solenoids	2	3
-for DC motors	6	10
-for stepping motors	4	10
Voltage level changer (sensor side)	10	25
A/D converters	1	3
D/A converters	6	14
Analog multiplexers	1	3
Number of bits	105	260

$$B = (B_1+B_2+\dots+B_m)/m \quad 3.35$$

In the above equations B and B_t are identical and represent the bandwidth of each of m equal tasks that can be used to replace the given m tasks of bandwidths B_1, B_2, \dots, B_m for the purpose of determining the processor requirements. (The right-hand-sides of equations 3.31 and 3.34 are identical as shown in Chapter 3.) The methodology for computing the processor requirements for the experiments is described next.

Each experiment is examined in order to determine the number of tasks required for it. As we have seen earlier, each task corresponds to a daemon. Since these tasks are interrelated, their bandwidths are almost identical. These bandwidths B_i are next computed. On a plot of overall system bandwidth versus number of control tasks (Figure 3.4), a point is entered corresponding to the location

$$\left(\sum_{i=1}^m B_i/m, m\right)$$

where m is the number of daemons required. If this point is below the plot for a given processor (point X in Figure 4.3), the experiment in question can be performed using that processor. In case the point lies above the plot for the processor (point Y), additional plots are drawn with the bandwidths being doubled, then tripled, and so on until the point

$$\left(\sum_{i=1}^m B_i/m, m\right)$$

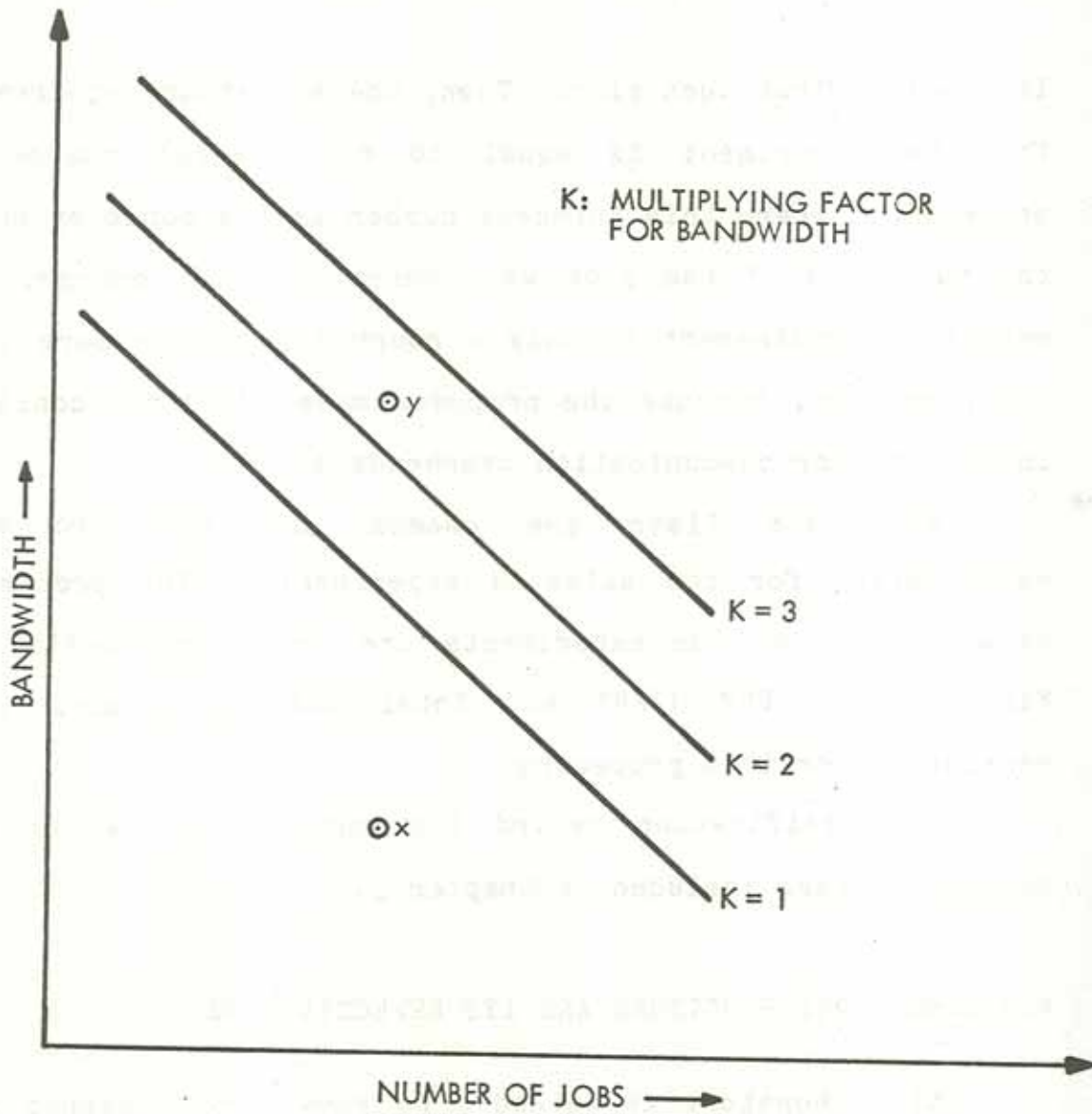


Figure 4.3 Graphical Method for Determining the Processor Requirements.

lies below first such plot. Then, the processor requirement for the experiment is equal to an integral number of processors, where this integral number is the scale by which the bandwidth of the plot was increased. (Of course, the resulting requirement is only a rough figure for more than one processor, because the proposed model does not consider interprocessor communication overheads.)

Table 4.8 lists the daemon and the processor requirements for the selected experiments. The processor requirements for the experiments are shown graphically in Figure 4.4. PDP 11/45 and Intel 8080 processors were considered for this procedure.

The justification behind the entries in Tables 4.3 through 4.8 are included in Chapter 5.

4.3 LABORATORY FUNCTIONS AND ITS EFFECTIVENESS

This laboratory is intended to serve the students who have had one course in programming and who intend studying automatic control of physical processes. The laboratory introduces control in a very intuitive manner, which leaves to the imagination and intuition of the student, the discovery of an appropriate control strategy.

The daemon reflects the classical system of error detection and error correction very well. Since the daemon

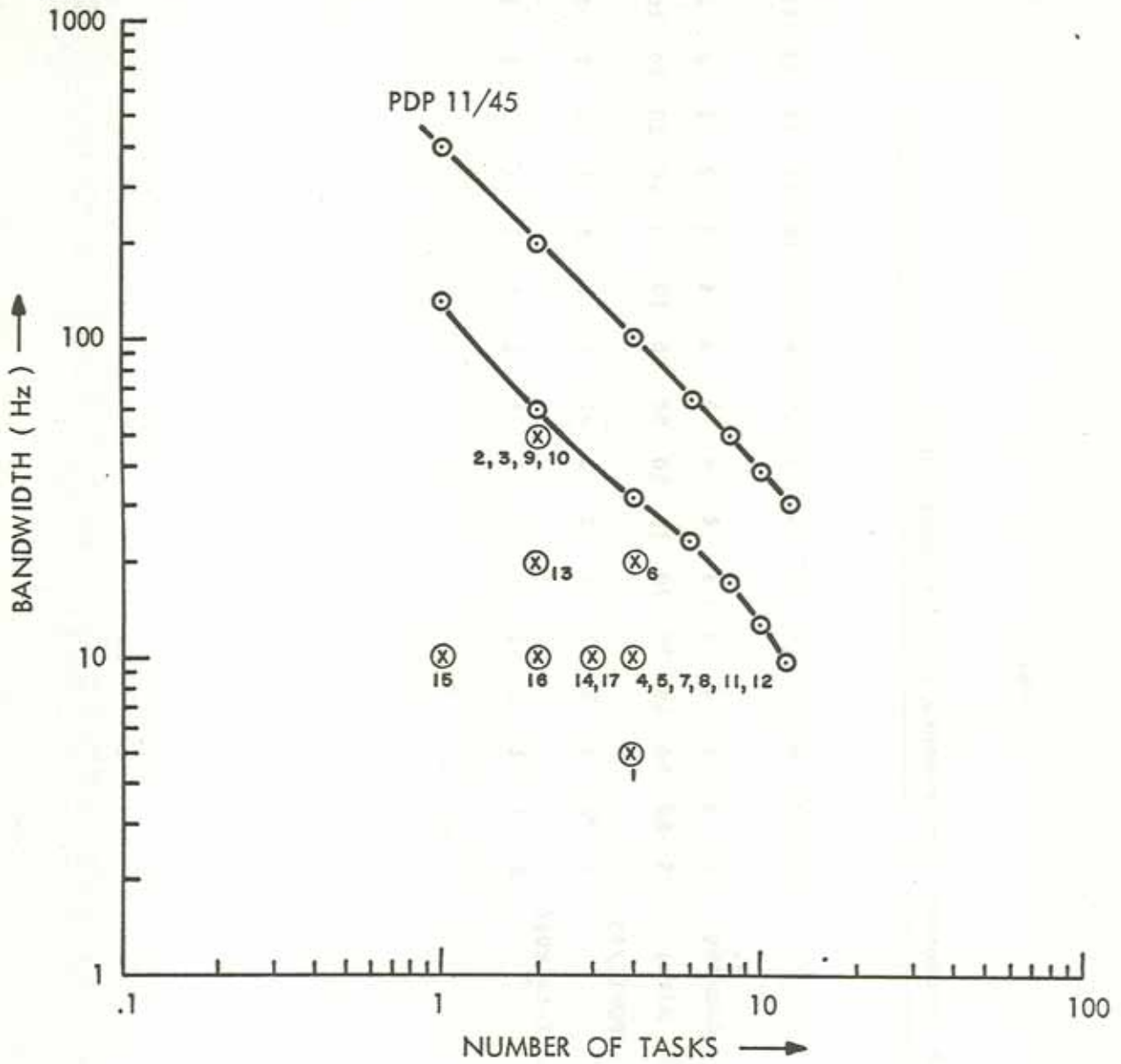


Figure 4.4 Determining the Processor Requirements for the Laboratory Experiments.

itself can be thought of as a software module, it is envisioned that the same kind of modularity in software as in hardware can be exploited. Also, the recommended intuitive control strategy approach helps the student see the physical phenomena directly in terms of his daemons, thus helping his understanding of the control process. Surely, this is a hit-and-miss approach for the first few runs, but we believe that this is not at all time-consuming, and at the same time it gives better understanding of the control problem. Once a daemonized experiment is found to be reasonably good, classical control principles may certainly be used incorporating daemon strategies to achieve finer results; by exploring various daemon combinations through different strategies, you might achieve even better control.

The laboratory by the very nature of its modularity, affords the students another advantage. The student need not visualise the control process through hardware alone. He can approach a control problem by first writing a daemonized control program, and then using the correspondence between the daemons and the conversion modules to set up the experiment physically.

CHAPTER 5

EXPERIMENTS FOR THE MODULAR LABORATORY

In this chapter, we will describe the selected experiments of the Control Robotics laboratory. For each experiment, we will first give a sketch description, and then present the requirements for its implementation.

5.1 LIST OF EXPERIMENTS

The experiments the descriptions of which follow in this chapter, are listed below:

TURTLE WAR

METAL BALL BALANCING

CAR TO FOLLOW A LAID-OUT TRACK

SLIDE FLUTE

RECORDER

INVERTED PENDULUM

GUITAR

VIOLIN

YO-YO

STILT WALKER

PADDLE POOL

TABLE SOCCER

TILT MAZE

SPACE WAR

SIMULATED PING-PONG

SIMULATED INVERTED PENDULUM

SIMULATED BILLIARDS

5.2 DESCRIPTION OF THE EXPERIMENTS

5.2.1 TURTLE WAR

There are two hemispherical objects (turtles[15]) at war. Each is equipped with sensors to locate the other. When the correct direction has been ascertained, an ejection mechanism is activated in the turtle which causes the shooting of golf balls at the adversary. The idea of the game is, of course, to score as many hits on the opponent as possible.

The basic module turtle can be put to a variety of uses, e.g. drawing a line on the floor, making a vacuum-cleaner that finds its way around a room, etc.

With reference to Figure 5.1a, each turtle, a hemispherical object, carries a concentrated light source L on its top such that this light can be spotted from all around the turtle. The turtle also houses a photocell assembly PR which can detect the light of the adversary turtle. When the enemy is detected, the turtle stops rotating about its vertical axis. At the same time, an

ejection mechanism G is activated to shoot the enemy with the golf balls. Each turtle is equipped with two bumper switches- S1 to detect any obstruction in the path of the turtle, and S2 to detect a hit scored by the enemy with golf balls. If an obstruction causes a turtle to stop, it can be made to reverse, rotate on its axis, and then moved on.

One needs a master daemon to score the hits by either turtle and relay this information to both. Also needed are daemons which upon detecting the location of the enemy turtle, activate the shooting mechanism. Another daemon may be used to handle the obstructions.

The modular setup for the experiment has been illustrated in Figure 5.1b. The key to the symbols used in the diagrams of the modular setups of this experiment and the following ones, appears in Table 5.1.

Module Requirements:

We begin by making what we feel are justifiable assumptions regarding the physical sizes and other parameters of the turtles. For this experiment, it is reasonable to assume that the turtle body weighs about ten kilograms, the maximum velocity it can attain is 2 meters per second and that the maximum acceleration anticipated is 8 meters per sec². These are justifiable assumptions to make for a turtle which is expected to operate in the confines of a normal laboratory space. Also assumed is that


















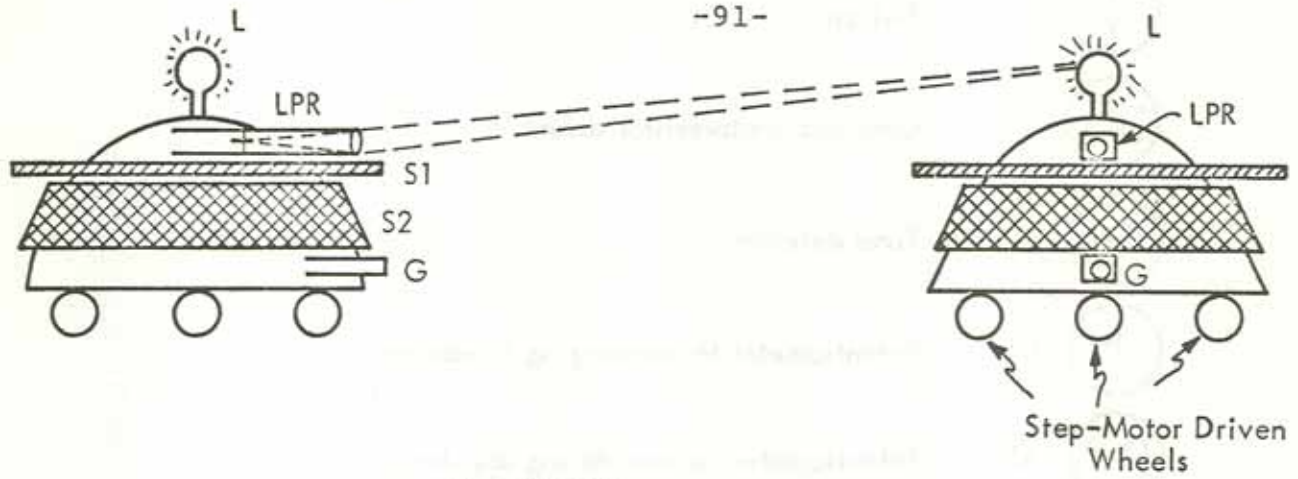
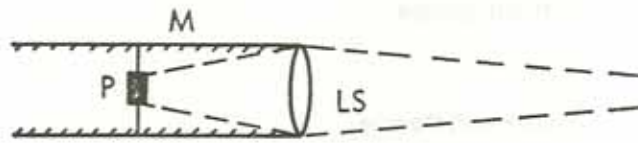
	S	: Switch
	LPR	: Lens and photoresistor assembly
	TD	: Tone detector
	LP	: Potentiometer for monitoring linear positions
	AP	: Potentiometer for monitoring angular positions
	SG	: Strain gauge
	TR	: Sheet resistance
	M	: DC motor
	M	: Stepping motor
	So	: Solenoid
	LC	: Level changer (voltage or current)
	R-V	: Resistance-to-voltage converter
	MP	: Analog multiplexer
	A/D	: Analog-to-digital converter
	D/A	: Digital-to-analog converter
	FF	: Flip-flop
	CG	: Current generator

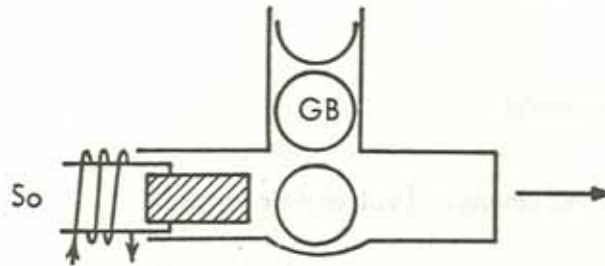
Table 5.1 Key to Symbols Used in the Figures of Chapter 5.



1. TURTLES



2. PHOTORESISTOR ASSEMBLY

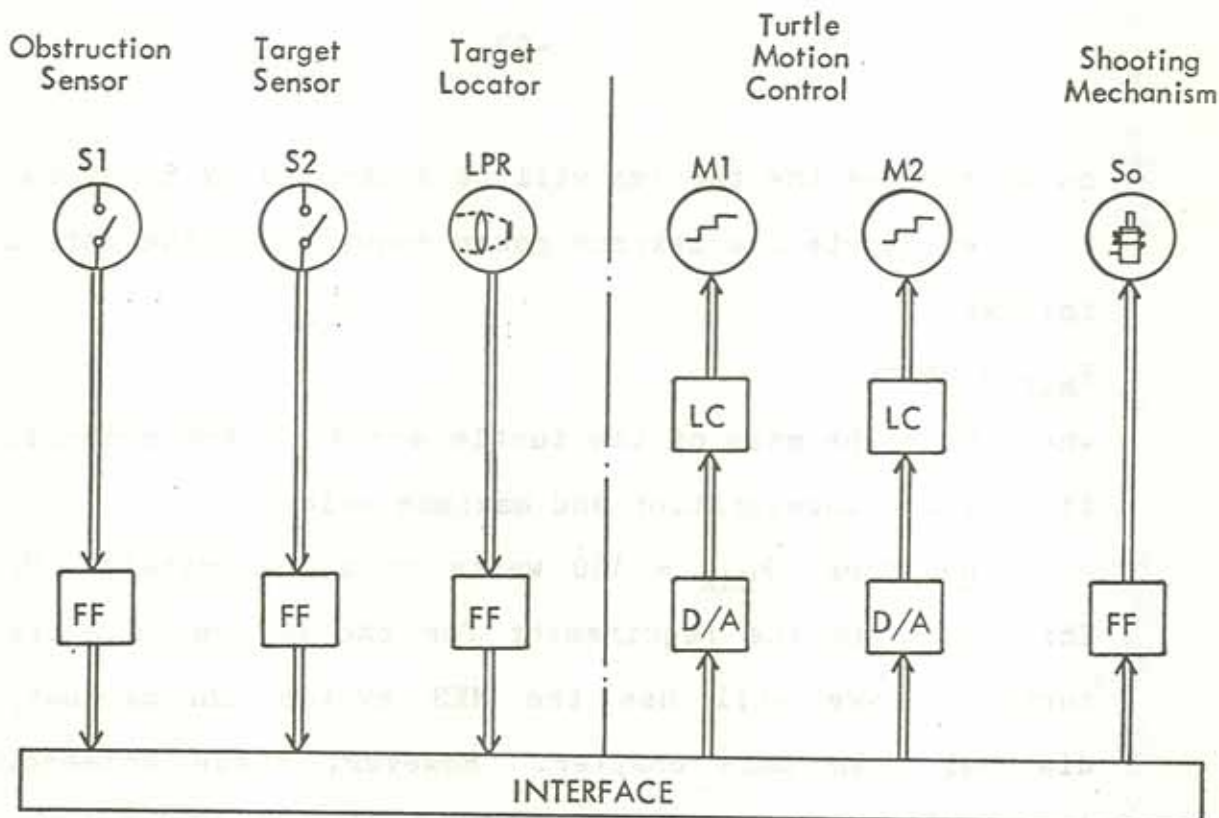


3. SHOOTING MECHANISM

KEY:

- L: LIGHT SOURCE
- LPR: PHOTORESISTOR ASSEMBLY
- S1: OBSTRUCTION SWITCH
- S2: TARGET SWITCH
- G: SHOOTING MECHANISM
- M: METAL TUBING
- P: PHOTORESISTOR
- LS: LENS
- GB: GOLF-BALL
- So: SOLENOID

Figure 5.1a Turtle War.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

MODULES FOR ONLY ONE TURTLE ARE SHOWN
THE OTHER TURTLE IS SIMILAR

Figure 5.1b Turtle War - Modular Setup.

on an average the turtles will be separated by 5 meters.

We compute the maximum power required of the motors, as follows:

$$P_{MAX} = M * A * V$$

where M is the mass of the turtle and A, V are respectively its maximum acceleration and maximum velocity.

Therefore, $P_{MAX} = 160$ watts or approximately 1/5 HP. This then, is the requirement for the motors to drive the turtles. (We will use the MKS system throughout the discussion in this chapter. However, since commercially available motors are mostly rated in the British system, we will give the requirements for the motors in Horse-Power: HP.)

To determine the requirements for the ejector gun solenoid, let us further assume that the maximum velocity of the golf balls is 20 m /sec, and that the time of travel for the ball is 1/4 second, while the time of impact is one-fifth of a second. The mass of the golf balls is assumed 50 grams.

Then, force = change of momentum in unit time.

$$FORCE = m * v / \text{time of impact}$$

Hence, solenoid force required is 5 newtons.

All sensors are digital, so one bit resolution is required for each sensor. For the solenoid also, only one bit resolution is needed. The stepping motors are provided

a resolution of 8 bits in order to control their speeds to within one cm/sec.

Summarizing the requirements,

Each motor should be 1/5 HP

Solenoid should have a 5 Nt rating

Photoresistors, contact switches, and solenoid all have one bit resolution, while each step motor has an 8 bit resolution.

Processor Requirements:

This experiment requires four daemons, two for each turtle. One daemon is needed to locate the enemy, and fire at it, and the other daemon is needed to register hits scored by the other turtle, and to take evasive action. Five hits per second is a reasonable attack, hence the bandwidths of each of the daemons is taken as 5 Hz. Then plotting the point (5,4) in Figure 4.4, indicates that both PDP 11/45, and Intel 8080 can singly perform this experiment.

5.2.2 METAL BALL BALANCING

This simple experiment involves the use of a magnetic field of a solenoid to counteract the force of gravity in order to balance a metal ball in mid-air. Initially, the current in the solenoid coils is set such that the metal ball is just about balanced. This state of equilibrium can

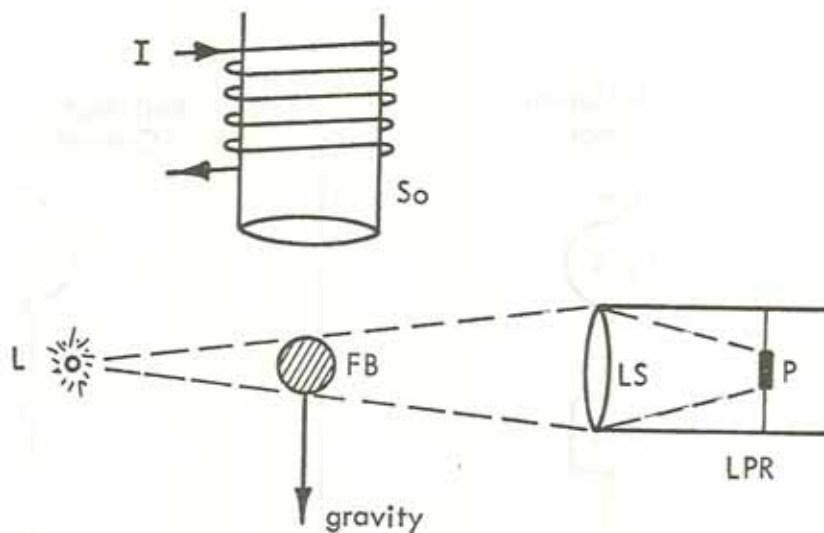
be disturbed either purposely or otherwise. In Figure 5.2a, the ball FB can only fall or rise vertically. Photoresistor-lens (LPR) assembly is used to detect whether the ball is rising or falling. Also, the speed with which it does this can be monitored. Depending on whether the ball is falling or rising, an expression can be activated either to increase the magnetic field or to reduce it respectively. In place of the photoresistor assembly one can use a tv-camera to detect the motion of the ball.

Figure 5.2b illustrates the modular setup for this experiment.

Module Requirements:

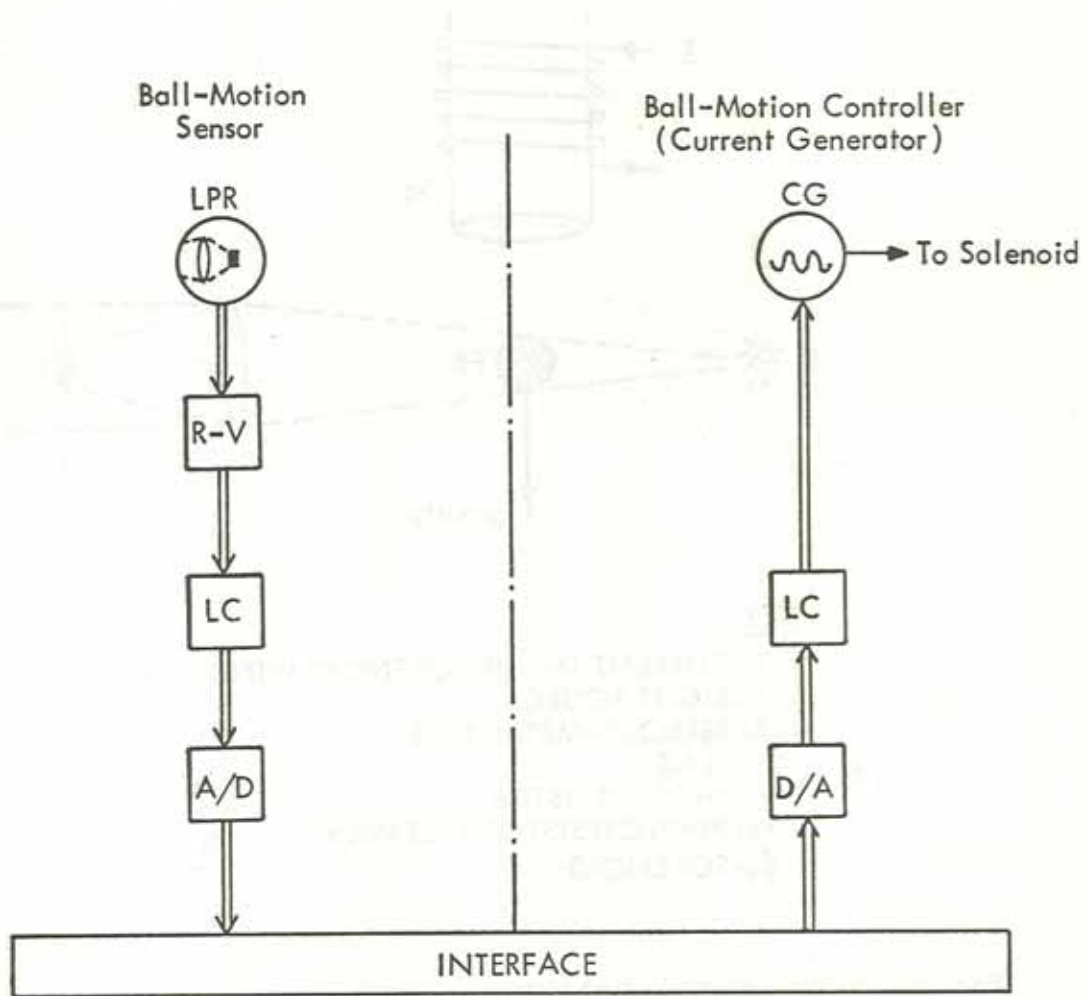
Here in order to compute the resolution requirement of the sensor, a trial and error method is employed. Assuming a certain resolution, the time required for the ball to fall through the middle part of the photoresistor is computed. If this time is less than a certain time called condition time, this resolution will be acceptable. We assume that .5 ms is an acceptable condition time. Since the time to fall through one unit of photoresistor will be less than this condition time, the corresponding resolution will be acceptable.

Assuming an effective length of 5 cm for the photoresistor lens assembly, and an 8 bit resolution, we determine the time to fall through the middle $5/2^8$ i.e.



- KEY:**
- I:** CURRENT IN THE SOLENOID WINDING
 - L:** LIGHT SOURCE
 - FB:** FERROUS-METAL BALL
 - LS:** LENS
 - P:** PHOTORESISTOR
 - LPR:** PHOTORESISTOR ASSEMBLY
 - S_o :** SOLENOID

Figure 5.2a Metal Ball Balancing.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.2b Metal Ball Balancing - Modular Setup.

5/256 cm.

Velocity at the mid-portion of the photoresistor,

$$V = (2gS)^{1/2}$$

Since $S = 2.5$ cm, $V = 70$ cm/sec. Then, substituting this value in

$$s = V*t + .5*g*t^2$$

we obtain $t = .3$ ms, which is less than the condition time. Thus, an 8 bit resolution for the photoresistors is acceptable. (Actually this value was arrived at by trial and error.) The actuator which is a current generator should also have this same resolution.

Processor Requirements:

This experiment requires two daemons in order to balance the metal ball. One daemon is for monitoring upward motion of the ball, while the other is for the downward motion. In a free fall, it would take 0.1 second for the ball to drop 5 cm. Hence, assuming that the motion of the ball is to be arrested within 1 cm, the bandwidths of the daemons may be expressed as 50 Hz. Then, marking (50,2) on the plot of Figure 4.4, we notice that either of PDP 11/45 or Intel 8080 will suffice.

5.2.3 CAR TO FOLLOW A LAID-OUT TRACK

A remote controlled model car is made to track a reflecting strip on the floor in this experiment. A

transmitter and a receiver reside in the car. The transmitter will relay the information about the motion of the car to the microprocessor setup, which will send out information to the receiver in the car regarding the control of its motion. This control may be effected through two stickshift movements various positions of which are indicated in Fig. 5.3a with the explanation of the controls. An alternative is to use two step motors as in the turtle module and then control the relative speeds of the motors. Fig. 5.3b shows the modular experimental setup for the latter approach.

The sensor is a pair of lens and photoresistor assemblies(LPR). When the car is entirely on the track, a concentrated light source L illuminates these photoresistors fully; thus their electrical resistance is minimum when the car is entirely on the track. If the car deviates from the track, it can be easily determined which side it is slipping off to, by monitoring the photoresistances. It must be noted that only the track is assumed to be highly reflective, the floor being considered rough. The velocity with which the car is leaving the track can also be known, and then corrective action can be applied through an expression which causes a change in the relative speeds of the motors. For example, if the car is to be turned to the right, the left motor speed is made greater than the right

motor speed, causing a pivot on the right side thereby achieving the desired result.

A modification of the experiment consists in using a model car that turns, employing a steering wheel, and then controlling the motion of the car by using a stepping motor to turn the steering wheel.

Module Requirements:

The basic module here is a turtle, hence the requirements for the stepping motors is the same as those in the Turtle War experiment. The sensor is a photoresistor-lens set. Again considering an effective length of five cm for the photoresistor, a resolution of 8 bits is quite adequate since this allows for deviations of $5/256$ cm to be monitored. If the toy car is used in this experiment, then a resolution of only 2 bits per stepping motor is required per motor, since the the number of positions to control are only three as evidenced in Figure 5.3a. The requirements for the modules of this experiment are listed below:

Power rating of motors = 1/5 HP

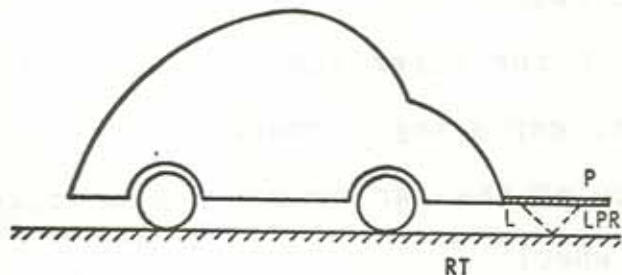
Photoresistor resolution = 8 bits

Stepping motor resolution = 8 bits or 2 bits

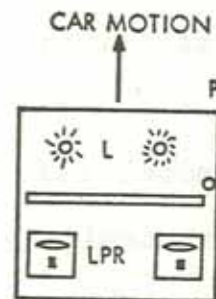
Also a transmitter and receiver set is required just as in the Turtle War experiment.

Processor Requirements:

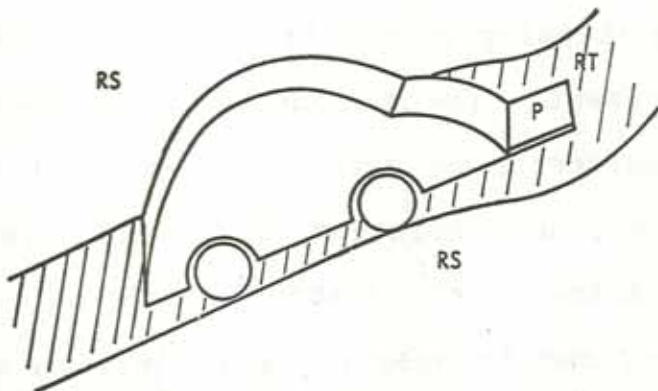
Two daemons are needed for this experiment: one for the



1. CAR AND REFLECTING TRACK
(Above and Below)

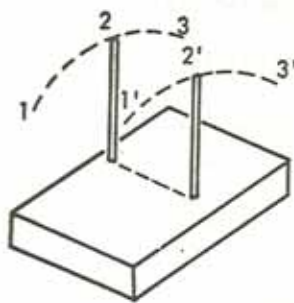


2. THE PLATFORM

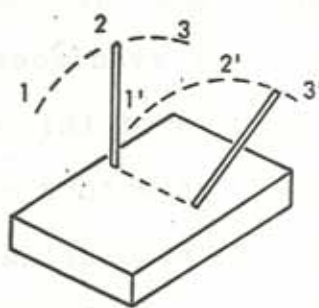


KEY:

- L: LIGHT SOURCE
- LPR: LENS/PHOTORESISTOR ASSEMBLY
- O: OBSTRUCTION TO PREVENT DIRECT ILLUMINATION OF PHOTORESISTORS
- P: PLATFORM HOUSING L's AND LPR's
- RT: REFELCTING TRACK
- RS: ROUGH SURFACE



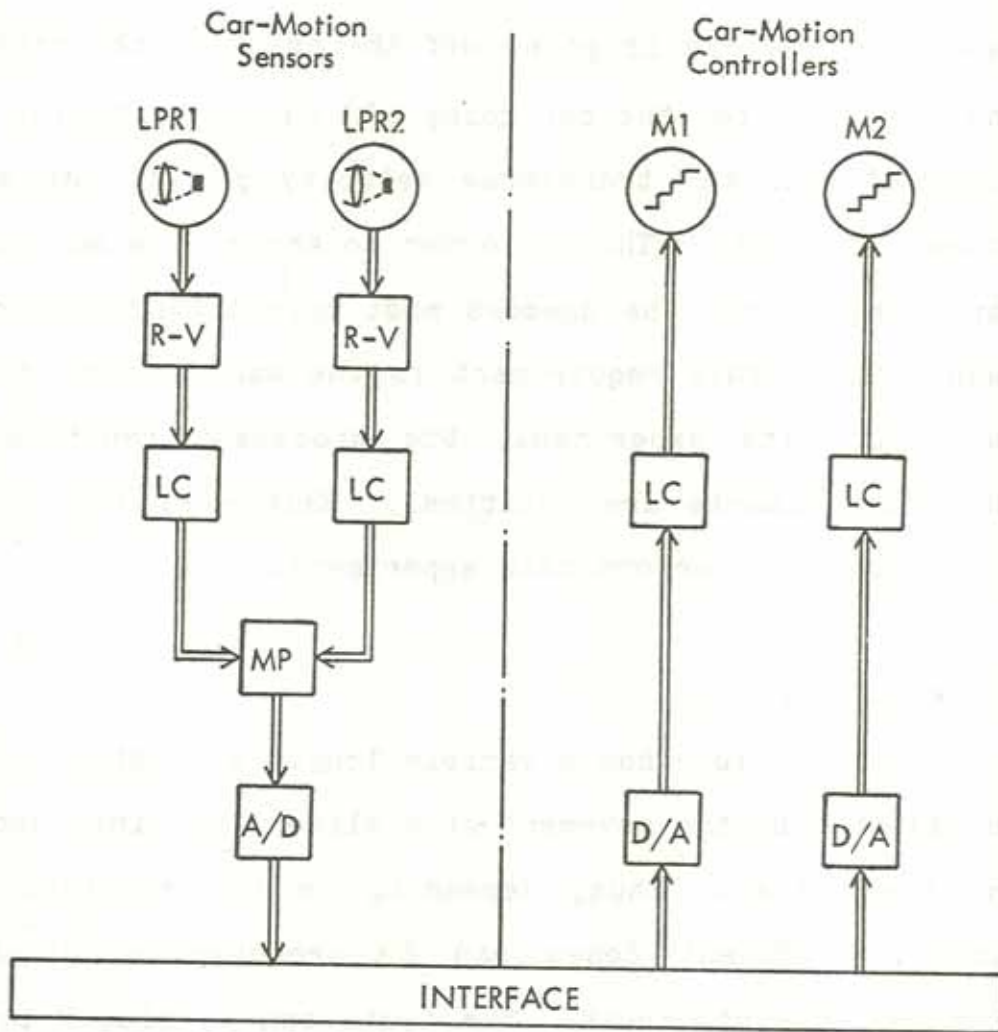
2-2' STOP POSITION



2-3' FORWARD RIGHT TURN

3. TWO STICKSHIFT CONTROL POSITIONS

Figure 5.3a Car to Follow a Laid-Out Track.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.3b Car to Follow a Laid-Out Track - Modular Setup.

case when the car is going off the track to the right side, and the other for the car going off to the left side. It is expected that the transverse velocity of the car will not exceed 50 cm/sec. Then in order to arrest the motion of the car within 1 cm, the daemons must have a bandwidth of 50 Hz each. Since this requirement is the same as for the Metal Ball Balancing experiment, the processor requirement for these experiments are identical. Either of PDP 11/45 and Intel 8080 can perform this experiment.

5.2.4 SLIDE FLUTE

A slide flute has a variable length air column which can be adjusted by the movement of a slider tube into and out of the flute body. Thus, depending on the position of the slider, different tones can be produced by blowing air through the flute mouth. The flute can be played by itself in an open-loop configuration or it could be played as an accompaniment to some other instruments.

The musical tones are detected both for self-playing and accompaniment, by a preprocessor- a tone detector circuit which merely counts the number of zero crossings for the tonal waveform and obtains the frequency (fundamental) by appropriate computations. When the frequency has been obtained, a linear actuator (a stepping motor whose motion is converted to rectilinear form) puts the slider SA(Fig.

5.4a) in motion to obtain the correct length of air column and then air is blown into the mouth of the flute.

The air delivery system is simply a piston crankshaft type, with input and output valves. Thus, the expression of the daemon must actuate both the air delivery system and the slider.

The modular configuration of the experiment is exhibited in Figure 5.4b.

Module Requirements:

For this experiment, the two main computations required are the power requirements for the air-blowing motor and the slide actuation motor. For the air-blowing motor, we assume that the pressure of air to be blown out is 1.5 pounds/sq. in. This is sufficient for the sharpest notes of interest. Considering a pipe of diameter 6 mm for blowing the air into, area through which this air is blown is

$$A = \pi d^2/4 = 0.3 \text{ sq.cm.}$$

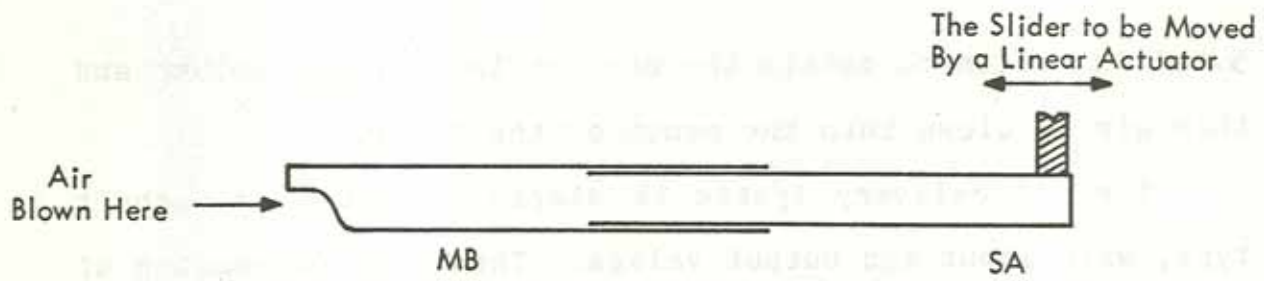
From Bernoulli's principle we can deduce,

$$P/D = V^2/2g$$

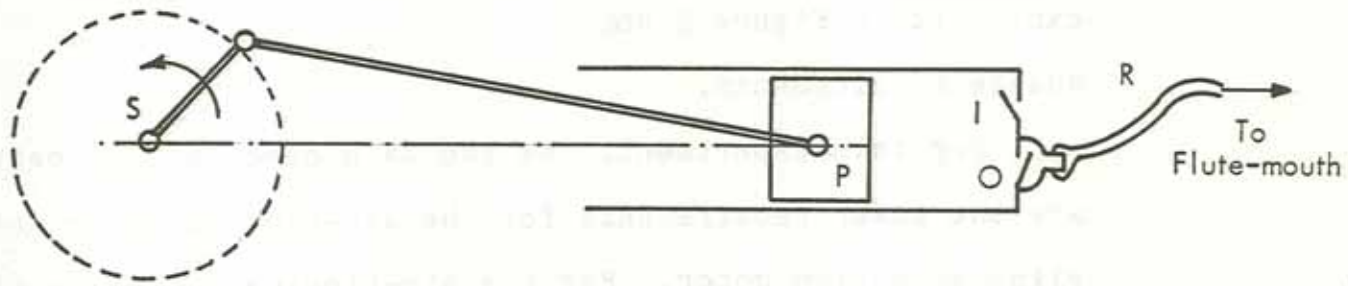
where P denotes the pressure, D the density of air in appropriate units, V is the velocity of air blown, and g is the acceleration due to gravity. Therefore, we calculate V as

$$V = (2gP/D)^{1/2} = 350 \text{ m/sec}$$

Hence, the volume of air blown is,



1. THE FLUTE

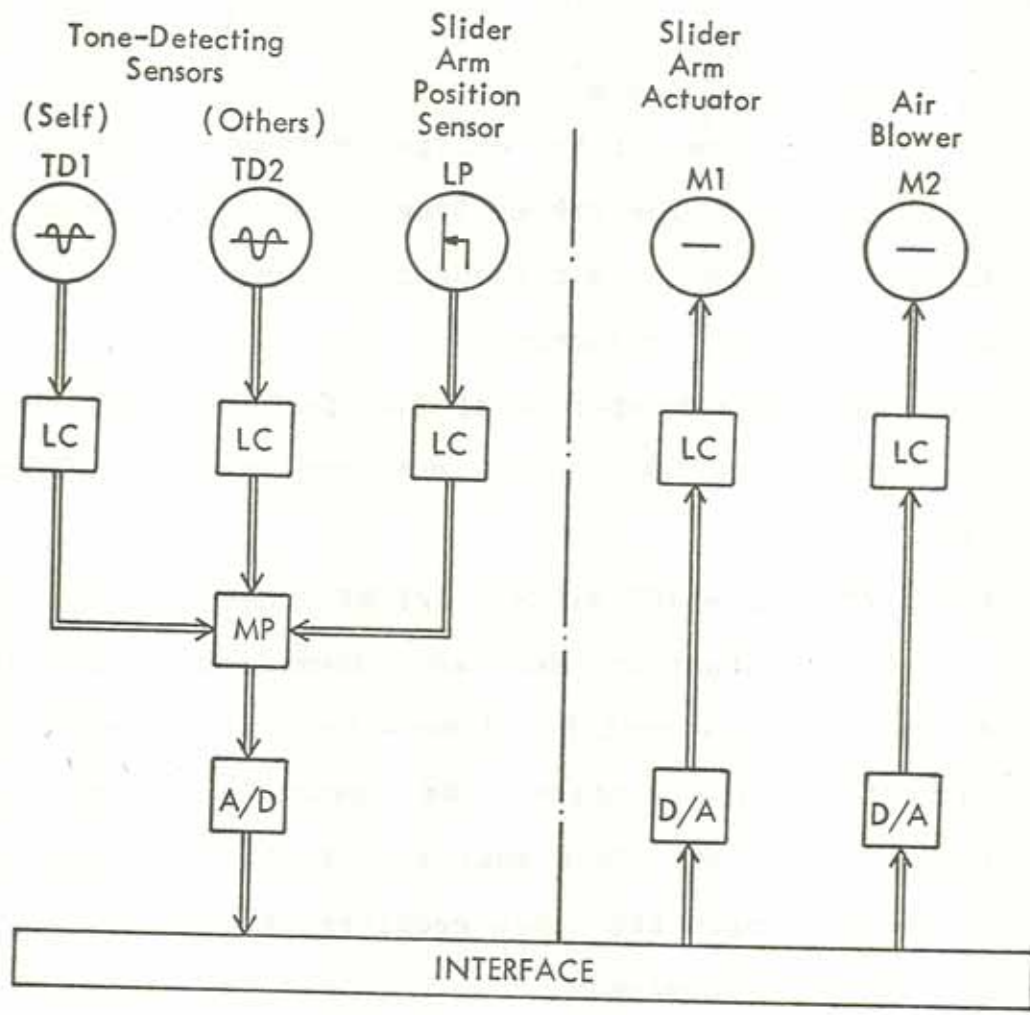


2. AIR DELIVERY SYSTEM

KEY:

- MB: MAIN BODY OF THE FLUTE
- SA: SLIDER ARM
- S: SHAFT (DRIVEN BY A MOTOR)
- P: PISTON
- I: INLET VALVE
- O: OUTLET VALVE
- R: RUBBER TUBING

Figure 5.4a Slide Flute.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.4b Slide Flute - Modular Setup.

$$VA = V \cdot A = 105 \cdot 10^{-4} \text{ m}^3/\text{sec.}$$

Let the area of the driving piston P (Fig. 5.4a) be $a = 100 \text{ sq. cm.}$ The stroke length is $l = 10 \text{ cm,}$ and the stroke time is assumed to be t seconds. Then,

$$a \cdot l / t = 105 \cdot 10^{-4} \text{ m}^3/\text{second.}$$

since the left-hand-side of the last equation is equal to the volume of air blown out per second. Hence, power required is

$$POW = P \cdot a \cdot l / t = 105 \text{ watts} = 1/7 \text{ HP}$$

Slide Actuation: Here we assume that in one second ten different notes must be played- this is the maximum that is encountered in practice. We assume that the slider has length of 30 cm. This must be actuated with acceleration A in order to meet the above requirements.

$$A = 2 \cdot S / t^2 = 60 \text{ m/sec}^2$$

Considering a slider mass of one kilogram, and the fact that the centre of mass will move through a distance of $0.3/2 = 0.15$ meters, the power required of the slider motor is given as,

$$POW = M \cdot A \cdot s / t = 90 \text{ watts} = 1/7 \text{ HP approximately.}$$

Resolutions: The tone detector requires a resolution of 10 bits in order to allow the frequencies upto 16 KHz. The potentiometers resolution need be only 8 bits, which will allow the generation of tones within 9 Hz.

The air-blower resolution of 6 bits is sufficient,

while the slide actuator resolution can be either 8 or 10 bits. The requirements are summarized below:

Air-blowing motor: 1/7 HP, 6 bits

Slider-motor: 1/7 HP, 8 or 10 bits

Tone detector resolution: 10 bits

Potentiometric resolution: 8 bits

Processor Requirements:

Two tone detector daemons, one slider daemon, and one airblowing daemon are required for the experiment. We assume that at most 10 notes are to be played per second. Hence, the daemons must have bandwidths of 10 Hz each. In Figure 4.4, marking (10,4) indicates that one PDP 11/45 or one Intel 8080 is sufficient to do this experiment.

5.2.5 RECORDER

The basic principle behind playing the recorder with mechanical elements is the same as for the Slide Flute experiment. This can also be played by itself, and made to accompany another instrument using some fixed strategies. The music played by the main instrument can be monitored using the preprocessor described earlier. When a certain kind of pattern is recognized, proper fingering sequences for the recorder can be done by the activation of appropriate expression. Also to be controlled is the amount of air that is to be blown across the recorder mouth.

Fingering is done through the use of push-type solenoids S_0 (Fig. 5.5a) which when activated cover the designated holes. Solenoid is also used for thumbing to change scales. The air delivery system is similar to that in the Slide Flute experiment. The difference here is that the air pressure must be regulated. For high-pitched notes the air must be blown hard into the recorder. The pressure of air can be detected using a vane V in the path of the air stream. The vane itself is connected to a potentiometer which can be calibrated properly.

Figure 5.5b shows the modular picture for the experiment.

Module Requirements:

For air-blowing motor, the requirements are the same as in the Slide Flute experiment. Solenoids must provide sufficient force to lift up the plungers against gravity. Assuming a plunger weight of 0.25 kg, force required is 2.5 Newtons. (Force = Mass*Acceleration). The requirements for this experiment are:

Air-blowing motor: 1/7 HP, 6 bits

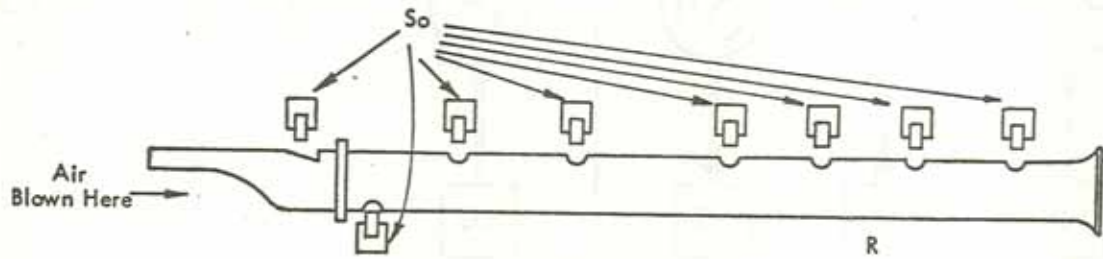
Solenoids: 2.5 Nt, 1 bit

Tone detector resolution: 10 bits

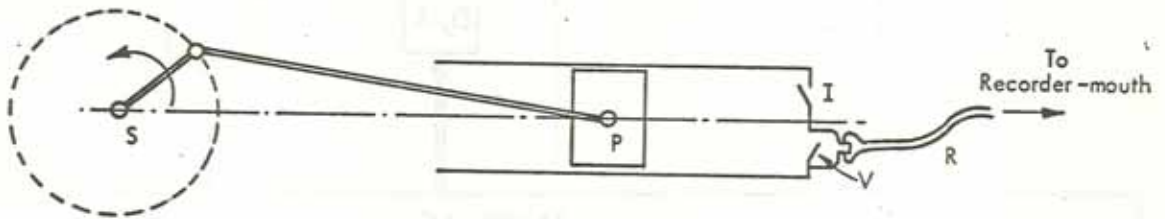
Potentiometric resolution: 6 bits

Processor Requirements:

Four daemons are needed: two for tone detection, one



1. RECORDER

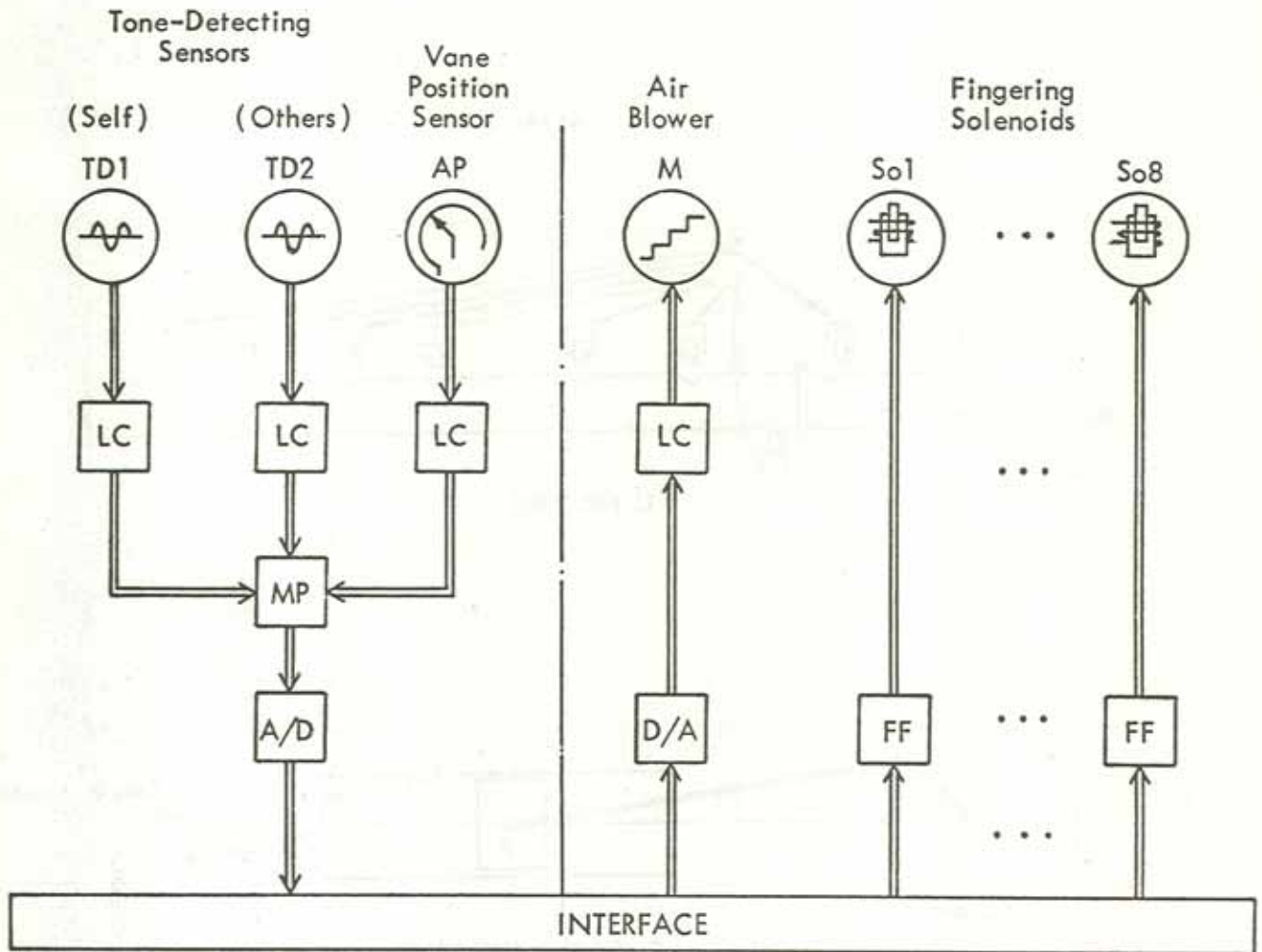


2. AIR BLOWING SHAFT

KEY:

- So: SOLENOID
- R: RECORDER
- S: MOTOR DRIVEN SHAFT
- P: PISTON
- V: VANE
- R: RUBBER TUBING
- I: INLET VALVE

Figure 5.5a Recorder.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.5b Recorder - Modular Setup.

for air-blowing, and one for fingering. Again, we assume that at most ten notes are to be played per second. Then, the requirements are the same as for the Slide Flute experiment - one PDP 11/45 or one Intel 8080.

5.2.6 INVERTED PENDULUM

This experiment is concerned with balancing a "broom" - an inverted pendulum - in the vertical direction, and at the same time transporting it to a specified location.

In Figure 5.6a, to locate the broom IP in the horizontal plane, two sensors are used. These are the x- and y-potentiometers. Two further potentiometers are needed to locate the pendulum in space. Its angles (θ_{xz}, θ_{yz}) in space with the x-z and y-z planes are monitored by these sensors. The objective of the experiment is to transport the broom to a specified location such that when it does get there, the angles it makes with the xz and yz planes are within some acceptable limits. If exact vertical balance is desired, angles and are monitored. If they exceed a tenth of a radian, it is very likely that the balance will be lost and that the pendulum will fall. If the pendulum is falling off to the left, the base B of the pendulum must be moved fast to the left in order to arrest the fall by getting the center of gravity of the pendulum within the base area. The prime concern in this experiment is to balance the broom

rather than centering it, so if the broom begins to fall it may have to be shifted to a side to capture the fall and then work the balanced broom to the desired location.

Four limit switches LS are to be provided so that it can be known if the broom is going to hit any of the boundaries. This is only a precautionary measure, since without this arrangement, the motors will keep struggling to move into the boundaries in order to balance the broom.

This experiment requires at least 4 daemons, two for the angle balance and two for the positioning. Figure 5.6b illustrates the modular setup for the Inverted Pendulum experiment.

Module Requirements:

We assume that the weight of the pendulum assembly is 5 kilograms, maximum acceleration required is 12 m/sec^2 . (In order to balance the broom, we must provide an acceleration equivalent to that due to gravity.) We also assume that the broom may be moved about in an enclosure measuring 2 meters square.

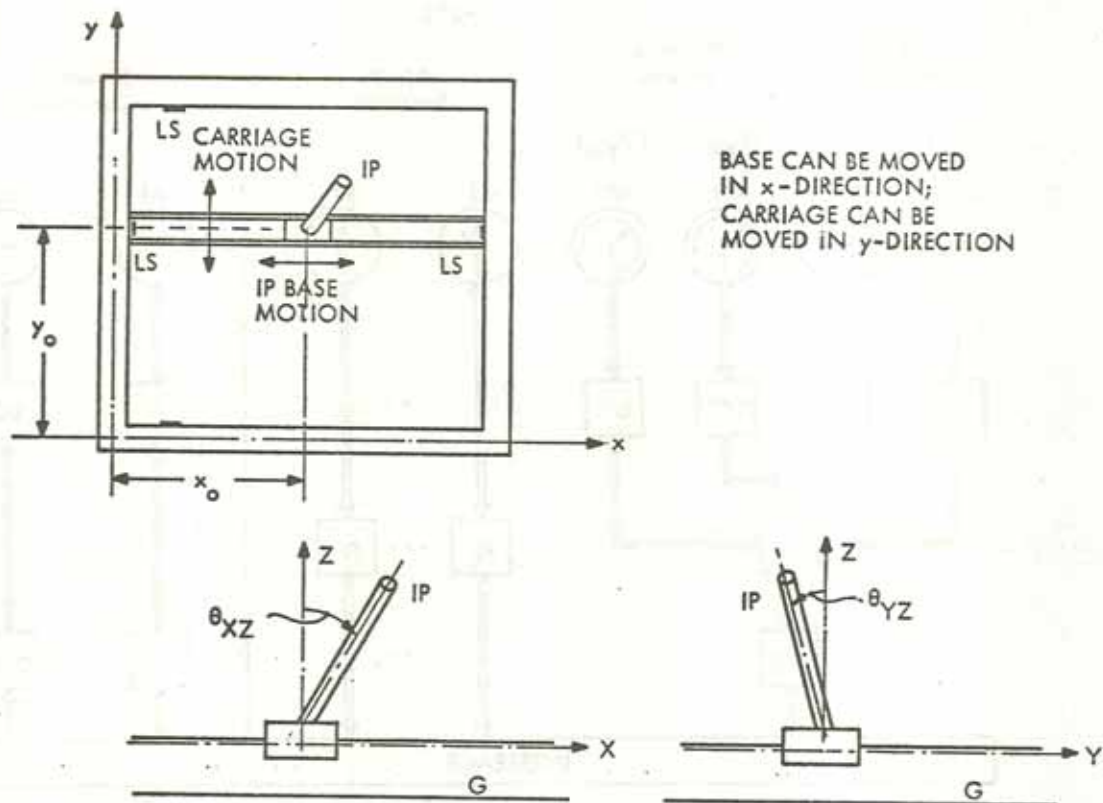
Then, considering the worst case where the pendulum must be moved the entire 2 meters, the time required is:

$$T = (2*S/A)^{1/2} = 0.6 \text{ seconds}$$

Hence, the power requirement for the balancing motors is:

$$\text{POWER} = M*A*S/T = 200 \text{ watts} = 1/4 \text{ HP approximately.}$$

In order to detect the position of the inverted

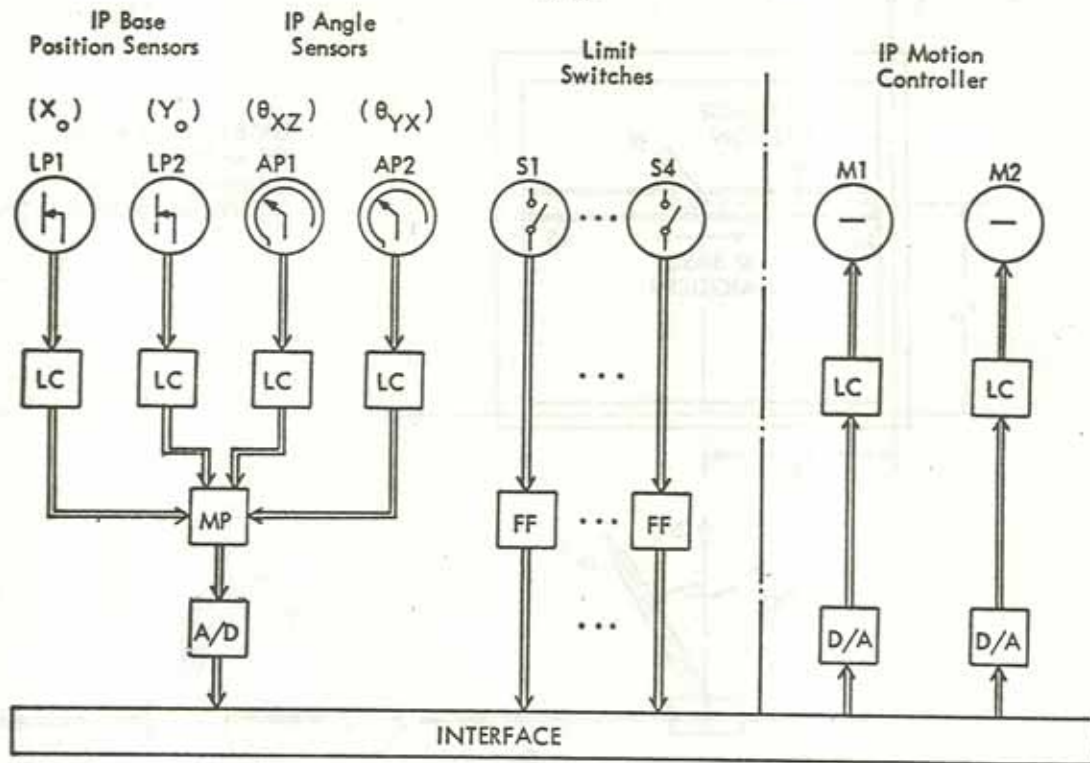


ANGLES ARE POSITIVE WHEN MEASURED FROM Z-AXIS TOWARDS POSITIVE X- AND Y-AXES

KEY:

- IP : INVERTED PENDULUM
- LS : LIMIT SWITCHES
- B : PENDULUM BASE
- G : GUIDE

Figure 5.6a Inverted Pendulum.



IP : INVERTED PENDULUM

FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.6b Inverted Pendulum - Modular Setup.

pendulum within 0.5 degree, the angular potentiometers need a resolution of 8 bits on the justifiable assumption that the maximum angular deviation will not exceed 90 degrees.

If centring of the pendulum within one centimeter is permissible, an 8 bit resolution suffices for the potentiometers monitoring the linear position of the pendulum base. For balancing the inverted pendulum, we need fine control of the balancing motors, hence an 8 bit resolution is imposed on them.

In conclusion, the requirements may be listed as:

Motors: 1/4 HP each, 8 bits.

Limit switches: 1 bit each.

Potentiometers: 8 bits each.

Processor Requirements:

Two daemons for angular position, and two daemons for the linear position are necessary in this experiment. The natural frequency of the inverted pendulum is computed to be 5 Hz. Thus, in order to control the pendulum, angle-daemon bandwidths must be 5×4 , i.e. 20 Hz. This bandwidth is deemed sufficient for the linear-position daemons, since this allows the motion of the pendulum base to be arrested within 10 cm. Plotting the point (20,4) in Figure 4.4, we see that one Intel 8080 or one PDP 11/45 can perform this experiment.

5.2.7 GUITAR

As with other musical instruments, a guitar can also be played automatically by itself or it may accompany another instrument. As before, a preprocessor must be used as the tone-detector. For guitar, however, a harmonic analyser would be more appropriate, considering that the music produced by guitar does not always consist of pure tones. Once the frequencies have been detected, the actuators can be activated to play the guitar. If only the fundamental frequency is to be monitored, a low pass filter may be used with the standard tone detector.

For fingering, two separate schemes may be used. The first of these employs a vast number of solenoids, but is extremely simple to implement in practice. This consists in placing one solenoid at each string-fret junction J . Proper fingering may then be accomplished by activating the appropriate solenoids S_o (Fig. 5.7a). In fact, with this scheme, one could play guitar as it can never be played by human beings who can use at most four fingers and a thumb for fingering purposes. The strumming can be achieved by using a solenoid each for the six strings along with the harpsichord like plucking arrangement. This is shown in its modular format in Fig. 5.7b.

Figure 5.7a shows the fingering arrangement as well as the plucking system. In order to cause a string-fret

contact, the appropriate solenoid S_0 is activated. This results in the U-shaped lever pivoting and forcing the string against the desired fret.

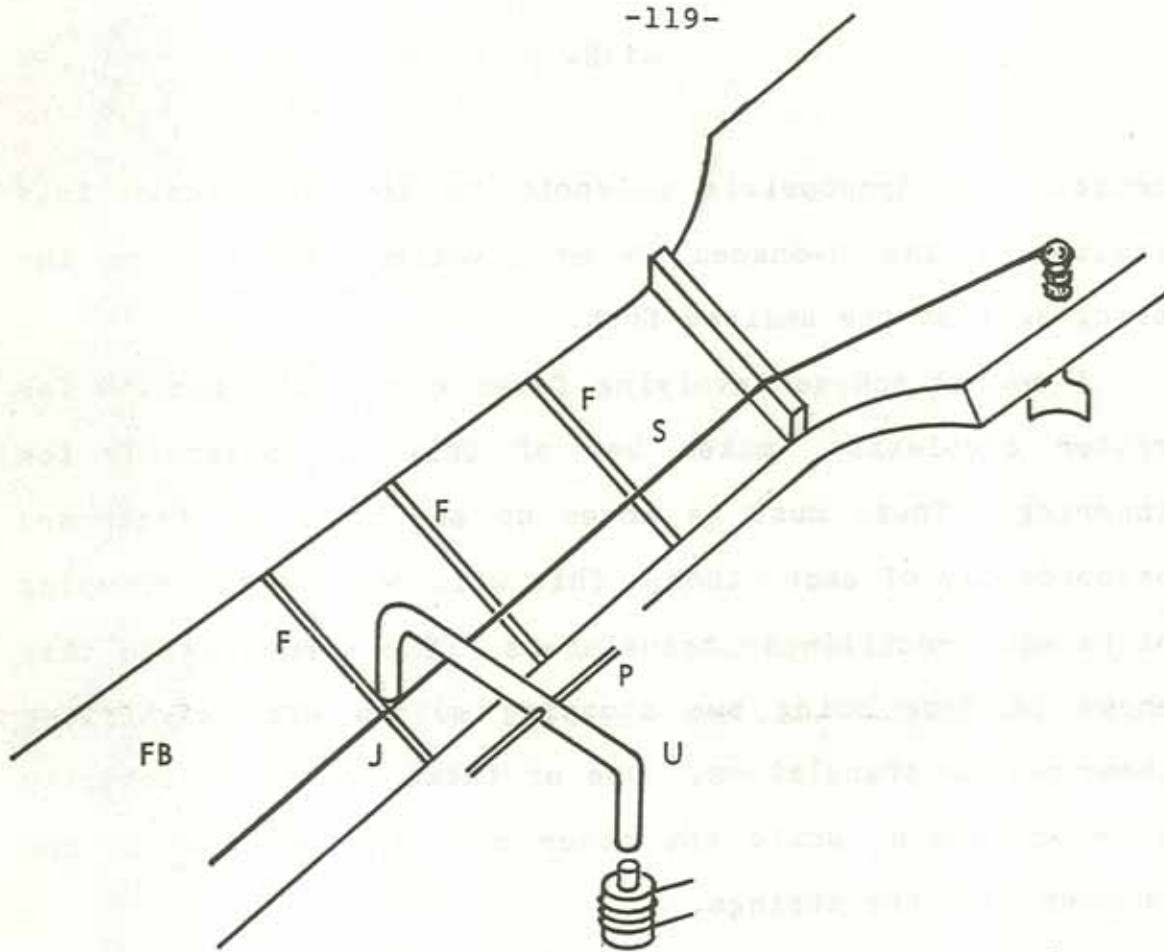
A second scheme involving fewer components but of far greater complexity, makes use of only six solenoids for fingering. These must be moved up and down the fretboard independently of each other. This will require six stepping motors with rectilinear translators. The strumming in this scheme is done using two stepping motors with associated linear motion translators. One of these serves to actually do the strumming while the other sets the position of the strummer over the strings.

Module Requirements:

We will consider only the simple version of the experiment which makes use of push-type solenoids, one at each string-fret junction. These solenoids must be powerful enough to cause the strings to depress upto the frets. A weight of one kilogram is sufficient for this purpose. Hence, from the computations of the Recorder experiment, the solenoids must have a 10 Newton rating. Tone detector requirements are again the same as for other musical instrument experiments. A summary of module requirements follows:

Tone detectors: 10 bits.

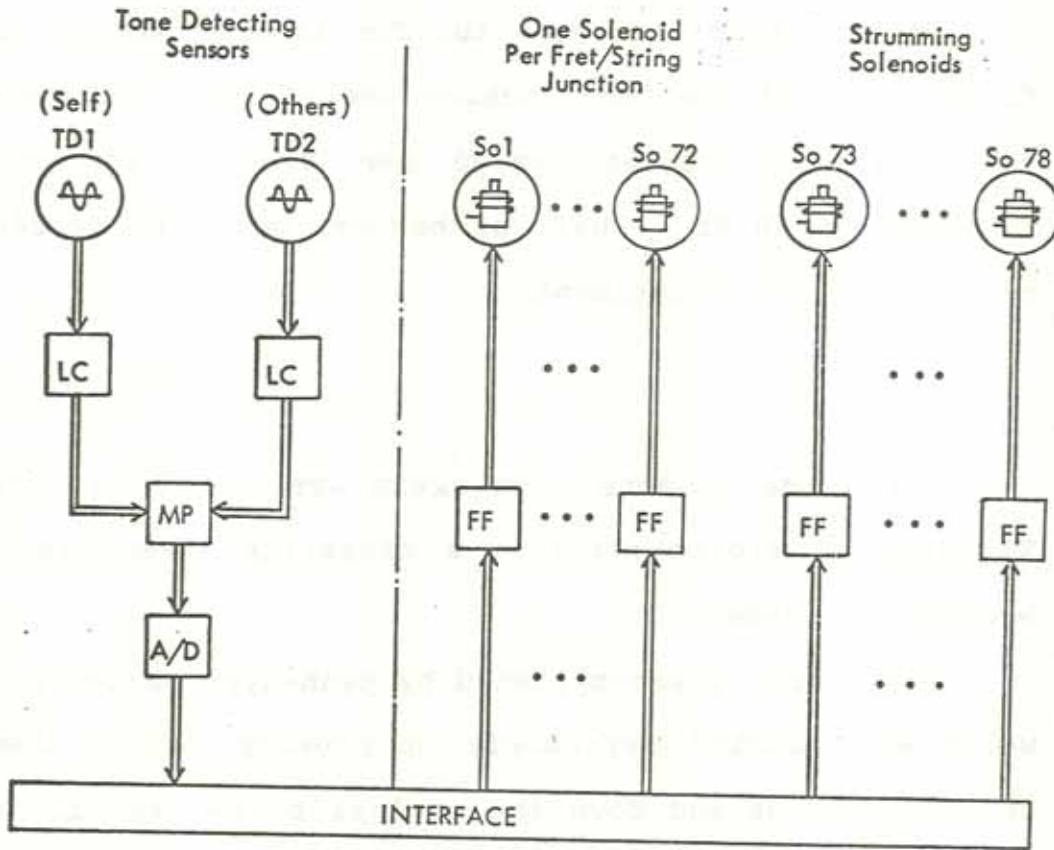
Solenoids: 10 Newtons, 1 bit.



KEY:

- So : SOLENOID
- S : GUITAR STRING
- F : FRET
- J : STRING-FRET JUNCTION
- FB : FRET-BOARD
- U : U-SHAPED BAR FOR DEPRESSING
STRING ON A FRET
- P : PIVOT (ABOUT WHICH U CAN ROTATE)

Figure 5.7a Guitar.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.7b Guitar - Modular Setup.

Processor Requirements:

We need four daemons - two for tone detection, one for fingering, and one for plucking the strings. Since at most ten notes are to be played per second, the bandwidth involved is 10 Hz. Thus, as before, only one processor is required by the experiment.

5.2.8 VIOLIN

Here, the sensors once again are the tone detectors. To play the violin requires a fingering arrangement and a bowing arrangement.

Fingering is accomplished by push-type padded solenoids which are mounted separately on housings which themselves can be moved up and down the strings by mechanical movement driven by DC motors. This is somewhat similar to the scheme described for the guitar experiment.

Bowing is provided by another DC motor whose motion is converted to a linear one and applied to the bow. A strain gauge is mounted on the bow to determine the pressure of bowing against the violin strings. This bow pressure can be adjusted by using another linear actuator (linear motion derived from a DC motor) which changes the angle of attack for the bow.

The positions of the fingers are monitored by potentiometers, one for each string.

In all the musical instruments experiments, two tone detectors are provided. One is for listening to itself, and the other is to detect music played by another instrument which is being accompanied. Figure 5.8a indicates the fingering and the bowing arrangements, while fig. 5.8b shows how this experiment is modularized.

Module Requirements:

Let the weight of the solenoid with the housing for it be one kilogram. On a violin, normally the fingers must move a maximum distance of 30 centimeters. If ten notes are to be played per second, the solenoid must be moved 30 cm in 0.1 second. Therefore, required acceleration, A can be computed.

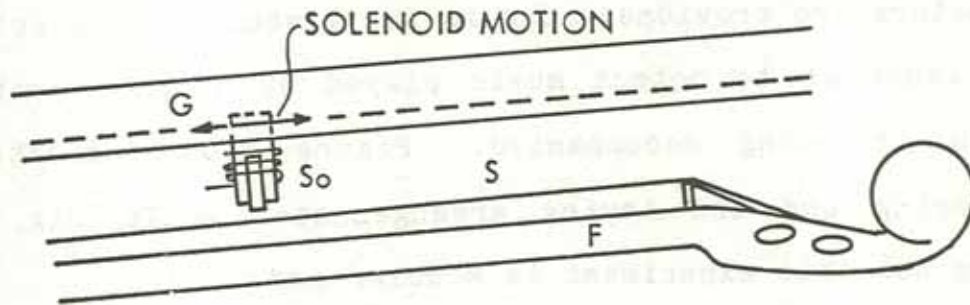
$$A = 2 * \text{distance} / \text{time}^2 = 60 \text{ m/sec}^2$$

The power of the driving motors is then given by:

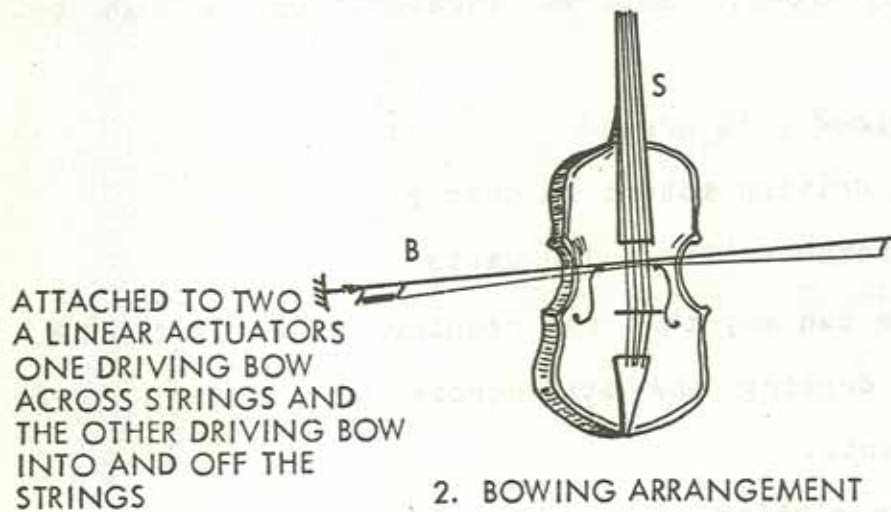
$$\text{Power} = \text{mass} * A * \text{distance} / \text{time} = 180 \text{ watts}$$

Approximately, we can say that the required power is 1/4 HP. The motors for driving the bow across the strings have similar requirements.

If 6 bits resolution is provided for the motors which move the solenoids, the solenoids can be located within 0.5 centimeters from the desired positions. The same resolution suffices for the bowing motors. The potentiometers and the strain gauge have similar resolutions. A summary of module requirements follows:



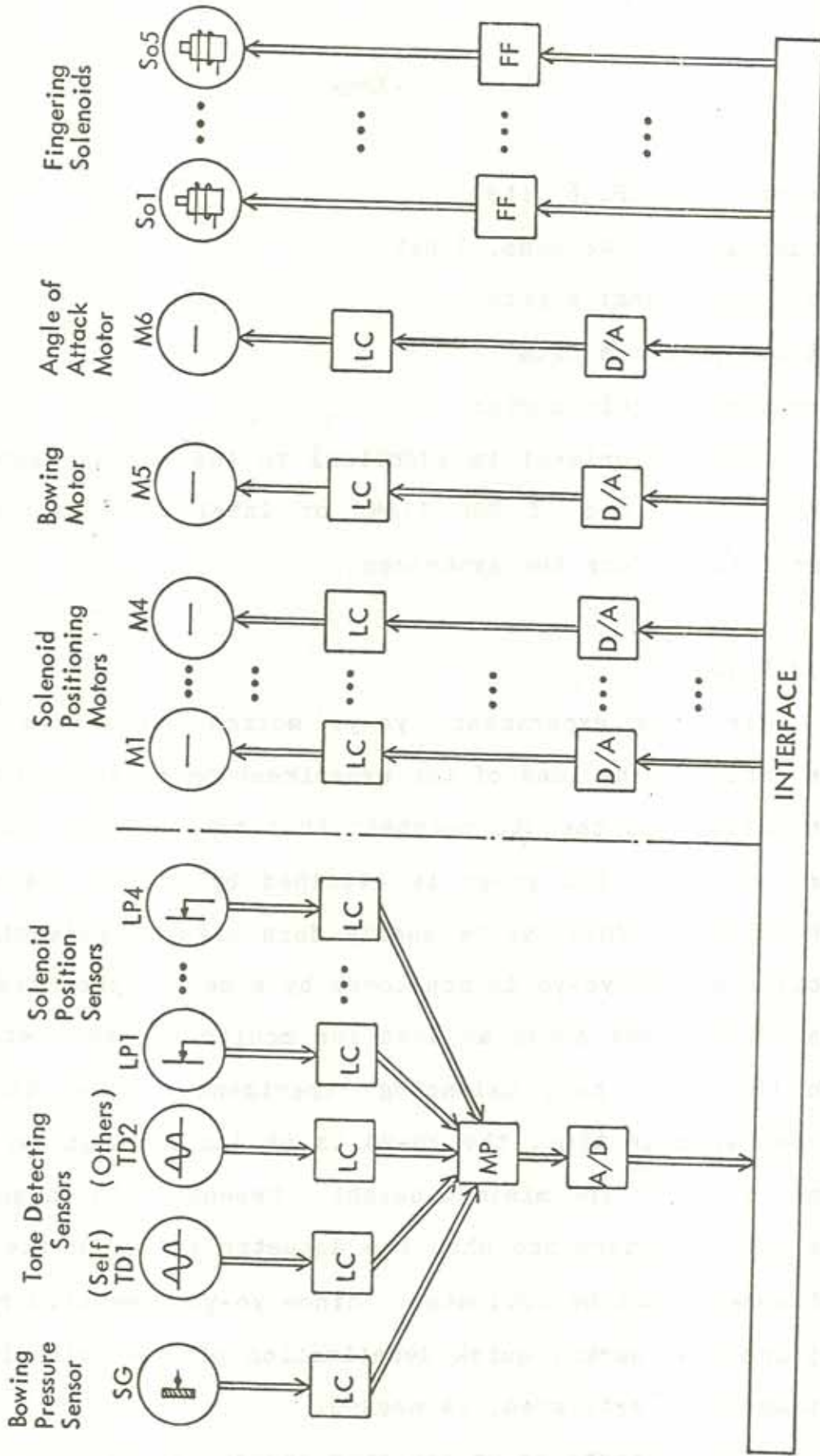
1. FINGERING ARRANGEMENT



KEY:

- F : FRET BOARD
- S : VIOLIN STRING
- S_o : SOLENOID (FOR FINGERING)
- G : GUIDE (TO MOVE THE SOLENOID ALONG LENGTH OF S)
- B : BOW

Figure 5.8a Violin.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.8b Violin - Modular Setup.

Motors: 1/4 HP, 6 bits

Solenoids: 10 Newtons, 1 bit

Potentiometers: 6 bits

Strain gauge: 6 bits

Processor Requirements:

This experiment is identical to the Guitar experiment. Hence, only one of PDP 11/45 or Intel 8080 is needed in order to perform the experiment.

5.2.9 YO-YO

In this experiment, yo-yo motion sustenance is the objective. The idea of the experiment is to start the yo-yo in motion and then to maintain this motion. To start, the arm to which the yo-yo is attached by string, is given a sharp jerk. This can be easily done using a solenoid. The motion of the yo-yo is monitored by a set of photoresistors, using the same setup as used for monitoring the metal ball in the metal ball balancing experiment. Thus, it can be known when in time, the yo-yo is at its maximum height and when it is at the minimum height. Depending on these times, it can be determined when the actuator to which the arm is attached, must be activated. Since yo-yo operation requires up and down jerks, quick deactivation of the solenoid after it has been activated, is needed.

An alternate yo-yo position sensor can be a tv- camera.

If fancy tricks like catching the yo-yo at the top of its height, i.e. at the end of the string, and then whipping it back, are to be performed, a catcher must be provided. This will catch the yo-yo when it winds right up to the end of the string. This can be achieved using two solenoid of push type. These will sandwich the yo-yo in between them when activated. This catcher assembly can be connected to another solenoid which can give the jerk required to start and sustain the motion.

Figure 5.9a shows the physical setup, while Figure 5.9b gives the modular structure for the experiment.

Module Requirements:

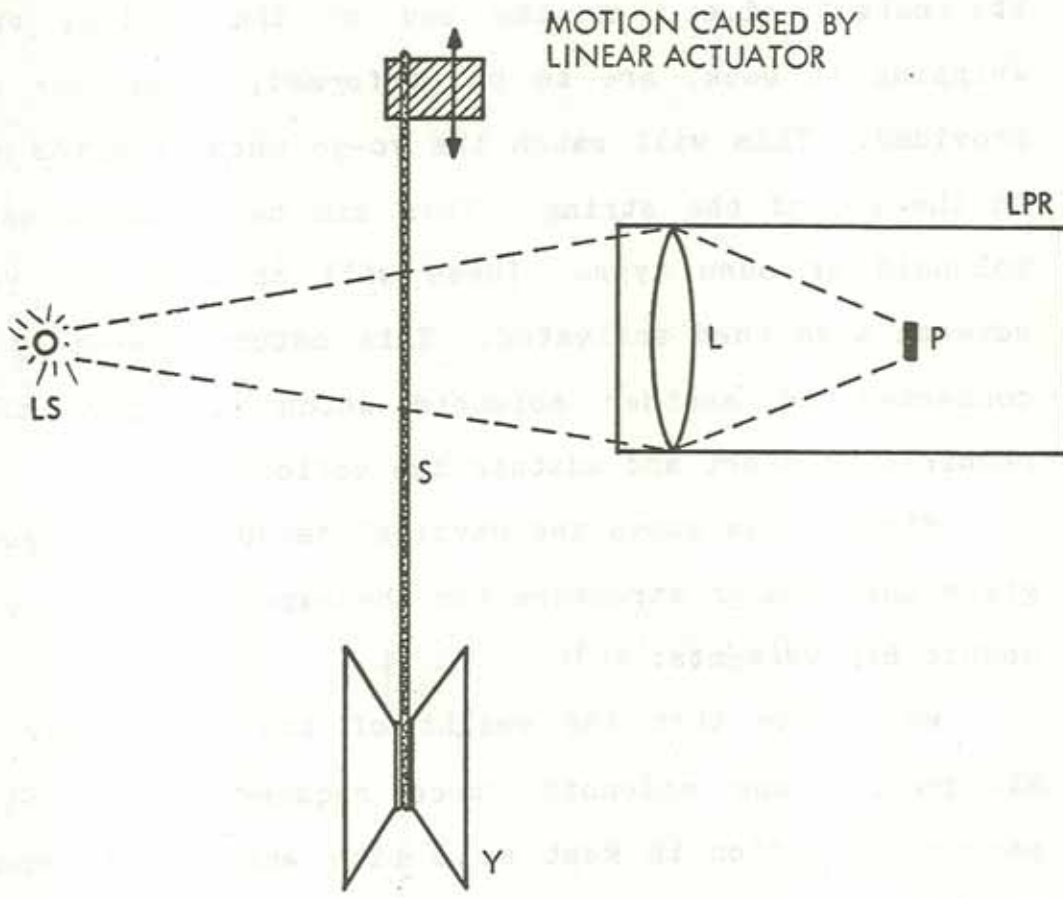
We assume that the weight of the yo-yo body is one kilogram, hence solenoid force required is 10 Nt. The sensor resolution is kept at 8 bits which will permit the monitoring of the yo-yo within $5/256$ cm as in the Metal Ball Balancing experiment. The requirements are:

Solenoid : 10 Newtons, 1 bit

Photoresistor: 8 bits

Processor Requirements:

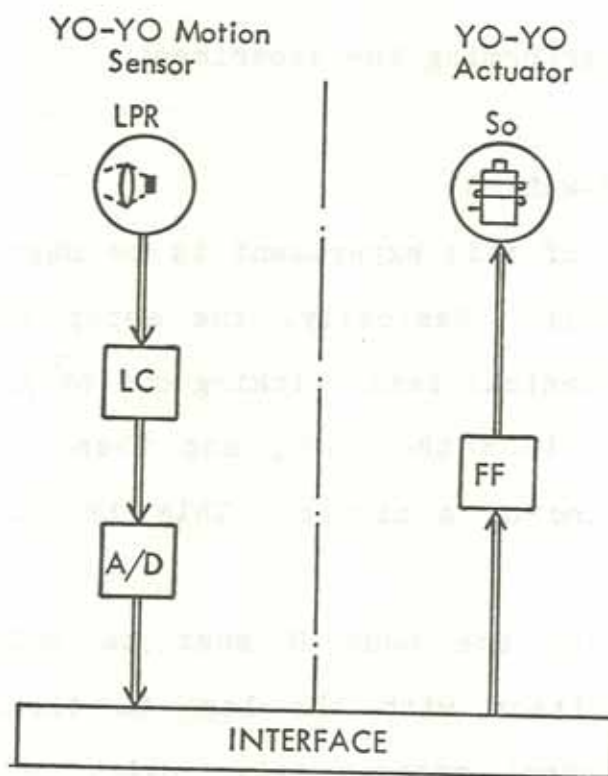
Two daemons are needed, one for the upward motion of the yo-yo, and one for its downward motion. In order to arrest the motion of the yo-yo within one centimeter, a bandwidth of 50 Hz per daemon is required. Hence, plotting (50,2) in Figure 4.4, we note that only one processor is



KEY:

- Y : YO-YO
- LS : LIGHT SOURCE
- S : STRING
- L : LENS
- P : PHOTORESISTOR
- LPR : LENS/PHOTORESISTOR ASSEMBLY

Figure 5.9a Yo-Yo.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.9b Yo-Yo - Modular Setup.

capable of performing the experiment.

5.2.10 STILT-WALKER

The aim of this experiment is to make a machine bi-ped walk by itself. Basically, the setup consists of a body with two mechanical legs sticking out of it. These legs can be retracted into the body, and these can also be swung through an arc of a circle. This is indicated in Figure 5.10a.

To start, the body B must be held manually in a vertical position with the legs SL firmly on the floor. When the control program takes over, one of the legs is retracted. The weight of the body makes it fall in one direction since now the body is just supported by one leg. As the body starts to fall, the retracted leg is caused to swing in an arc in the direction of fall. When this leg is just past the center of gravity of the body, its motion is stopped and the leg is extended so as to support the body. At the same time, the other leg is retracted and this is now off the floor. The momentum of the body makes it lean further into the direction it was falling in. The retracted leg is now swung in an arc as before and the cycle continues with alternate legs, advancing the body along.

The sensors for this experiment are potentiometers which determine how far the body is leaning, while the

actuator set consists of pull-type solenoids and stepping motors.

Figure 5.10b illustrates the modular setup for the experiment.

Module Requirements:

The weight of the retractable legs is assumed to be one kilogram each. Therefore, the solenoid strength required is 10 Newtons. The leg must be swung through about 100 degrees (2 radians) in 0.5 seconds which is the (assumed) time it would take for the body to topple forward to the point of irrecoverable balance. Then the desired angular acceleration is,

$$AA = \text{ANGLE} \cdot 2 / \text{TIME}^2 = 16 \cdot 6 \text{ radians/sec}^2$$

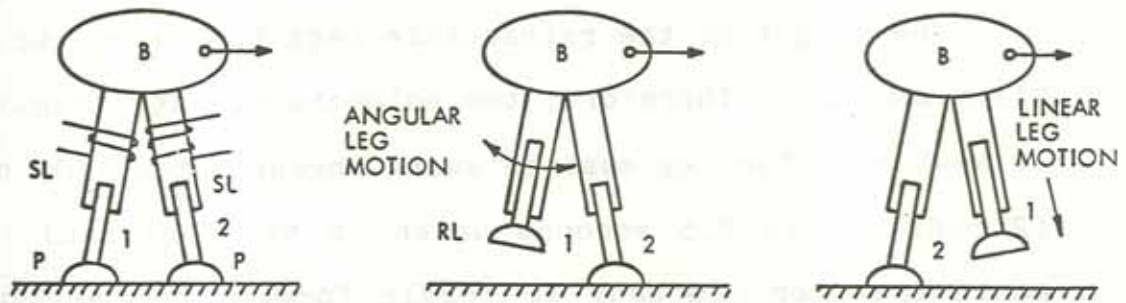
Assuming a total weight of five kilogram and a radius of gyration of 0.25 meters, the torque required of each motor is:

$$\begin{aligned} \text{TORQUE} &= \text{ANGULAR MOMENT OF INERTIA} \cdot \text{ANGULAR ACCELERATION} \\ &= 5 \text{ Newton-meters.} \end{aligned}$$

Since the angle of the leg can vary from 0 to 100 degrees, to track within 0.5 degrees of the angle, a resolution of 8 bits is provided for these potentiometers. Similarly the motor resolution is also set at 8 bits. The requirements are summarized below:

Solenoid: 10 Nt, 1 bit

Potentiometer: 8 bits

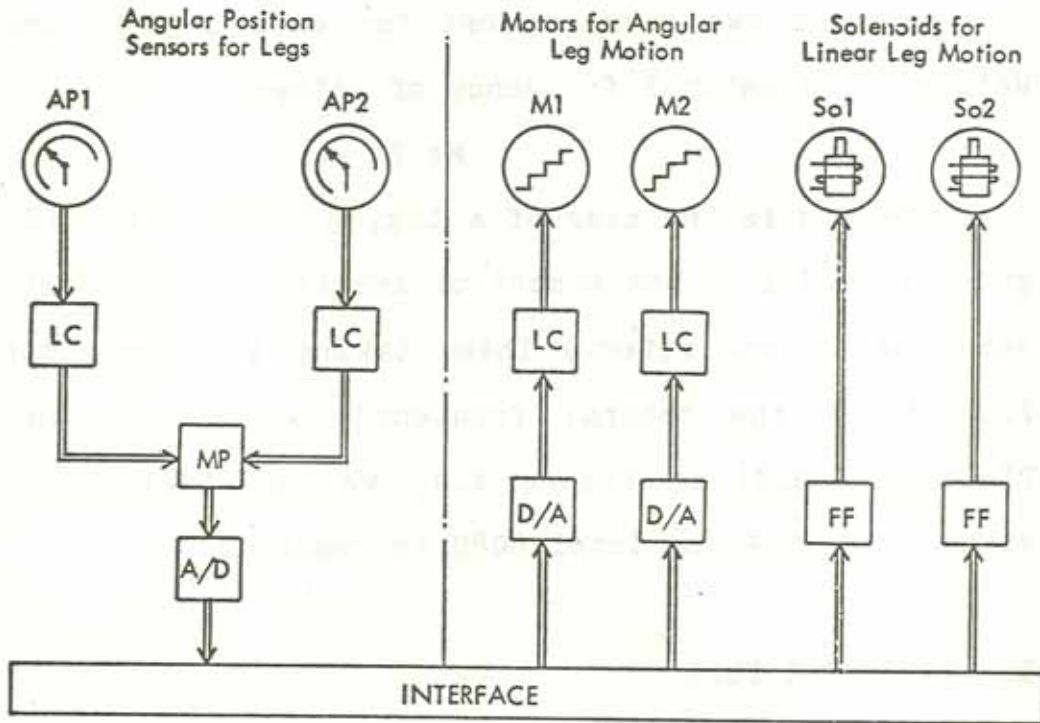


KEY:

- SL : STILT LEGS (EXTENDED)
- RL : RETRACTED LEG
- B : BODY
- P : PAD



Figure 5.10a Stilt Walker.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.10b - Stilt Walker - Modular Setup.

Motor: 5 Nt-m, 8 bits

Processor Requirements:

We need two daemons, one for each leg of the Stilt-Walker. The natural frequency of either leg is

$$\text{Mg/I}$$

where M is the mass of a leg, g is acceleration due to gravity, and I is the moment of inertia. (The length of the each leg is one meter.) Then, taking the bandwidth to be four times the natural frequency, we get B_i as 50 Hz. Plotting (50,2) in Figure 4.4, we see that only one of either PDP 11/45 or Intel 8080 is required.

5.2.11 PADDLE POOL

This game is to be played by two players for the sake of simplicity. With reference to Figure 5.11a, there is a board PL with two cups G, one at either end. These cups are the equivalent of goals as in soccer or hockey. Each player must protect his cup, and at the same time try to score in the other player's cup. A small ball B is placed in the center of the board to start the game. Each player is provided with a blower BL. This can be a simple bellows construction, or an electrical blower can be used. For this experiment, the same setup is used for air blowing as was used in the recorder and Slide Flute experiments. The idea then is to direct the bellows onto the ball in such a way as

to prevent it from coming into one's own pocket, and to make it fall into the opponent's cup. The blowers are mounted such that they may be swivelled in a semicircular arc. The blowers are so positioned that they always force air out on the board only.

The position of the ball is sensed by a resistance sheet arrangement. There are three static electrodes, while the dynamic electrode contact is at the bottom of the ball. The three resistances from the dynamic contact to the three static contacts should determine the position of the ball. This physical setup is illustrated in fig. 5.11a. Figure 5.11b gives the modular representation for the experiment.

Each cup contains a normally open switch, which closes when the ball settles in the cup, and registers the score. The blowers can be moved laterally by using stepping motors, and their lateral positions can be monitored with potentiometers.

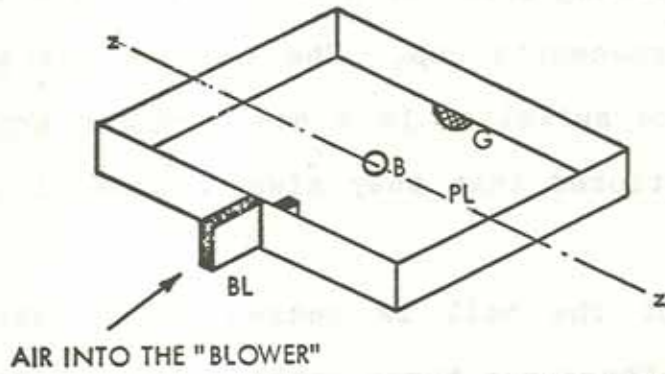
Module Requirements:

Assumptions: The weight of the ball is one kg, its diameter is 3 cm. Also, assume that the maximum acceleration is 2 m/sec^2 (for moving the ball one meter in one second). Then the force needed is

$$F = M \cdot A = 2 \text{ Nt}$$

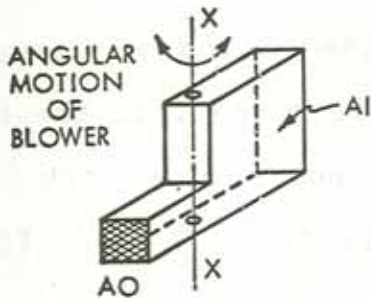
and the pressure required is

$$P = F/\text{AREA} = 4 \cdot F/\pi \cdot D^2 = 2500 \text{ Nt/m}^2$$

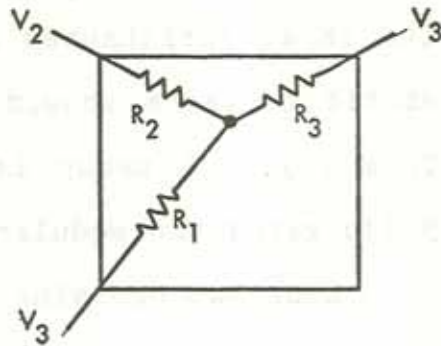


ONLY ONE BLOWER AND ONE GOAL ARE SHOWN THE OTHER BLOWER AND GOAL ARE SYMMETRICALLY ACROSS THE zz -AXIS

1. THE GAME BOARD



2. AIR BLOWER

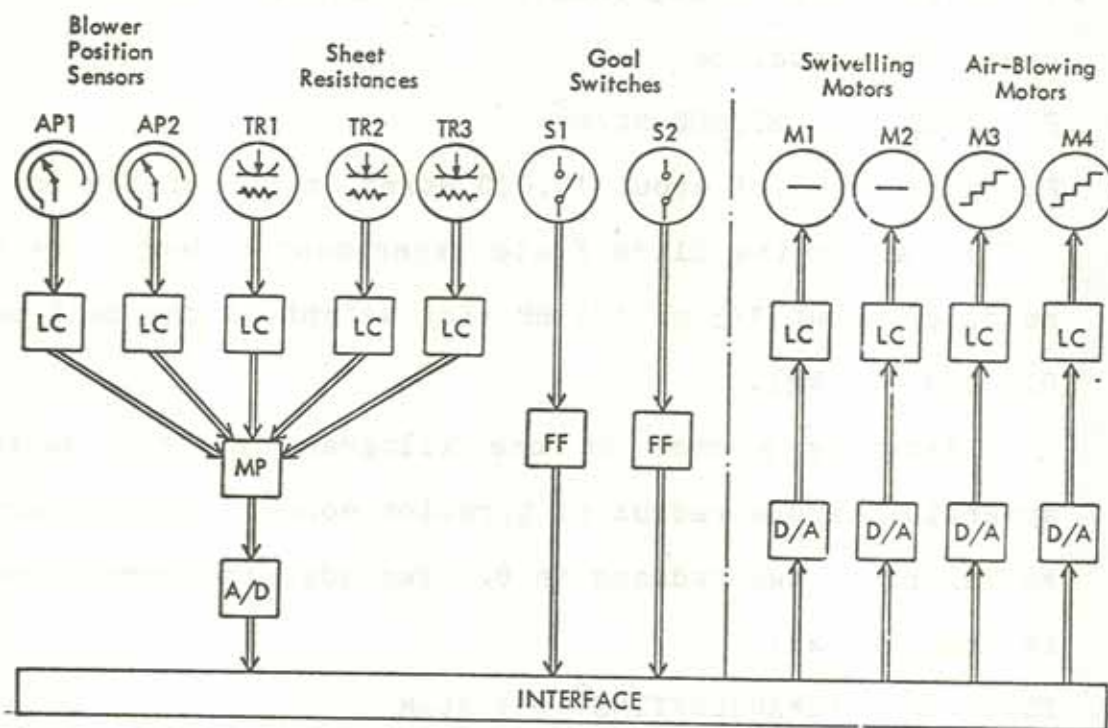


3. RESISTANCE SHEET METHOD FOR LOCATING THE BALL

KEY:

- B : BALL
- BL : AIR BLOWER
- G : GOAL
- PL : PLAYING AREA
- XX : AXIS ABOUT WHICH BL CAN SWIVEL
- AI : AIR-IN
- AO : AIR-OUT
- V_1, V_2, V_3 : VOLTAGES AT THREE CORNERS
- R_1, R_2, R_3 : RESISTANCES FROM THE BALL TO THE THREE CORNERS

Figure 5.11a Paddle Pool.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.11b Paddle Pool - Modular Setup.

Since the area of the pipe through which air is forced out is much smaller than the frontal area of the ball (approximately one-eighth), the actual pressure to be created is equal to

$$P = 8 * \text{OLD } P = 20,000 \text{ Nt/m}^2$$

For a pressure of about $10,000 \text{ Nt/m}^2$, power of the motor was $1/7 \text{ HP}$ as in the Slide Flute experiment. Hence, we need a motor of about $1/3$ or $1/4 \text{ HP}$ (for weight of the ball between 0.5 and 1.0 kg).

Assuming a mass of one kilogram for the swivelling apparatus, and a radius of gyration equal to 0.2 meters, the swivel being two radians in 0.5 seconds, the torque required is computed as:

$$\text{TORQ} = M * R^2 * 2 * \text{ANGLE} / \text{TIME}^2 = 1 \text{ Nt-M}$$

Once again the contact switch resolution is one bit. To locate the ball within 0.16 sq.cm. , 8 bit resolution is required of the sheet resistance measurements. The requirements:

Air blowing motor: $1/4 \text{ HP}$, 6 bits

Swivelling motor: 1 Nt-m , 8 bits

Sheet resistance: 8 bits

Contact switch: 1 bit

Processor Requirements:

Two air-blowing daemons, and two airblower-swivelling daemons are needed. If the motion of the ball is to be

checked within 5 cm (the size of the goal), the time for this will be 0.1 second. Therefore, the bandwidths of the daemons is taken as 10 Hz each. Then marking (10,4) point in the plot of Figure 4.4, we note that one processor is sufficient for the experiment.

5.2.12 TABLE SOCCER

This game as it is normally played in the pinball-machine places has three rows to four rows of players per side. The first row is the goaltender, the next the defensemen, then the midfieldmen, and then the forward line. In order to simplify the experiment, we shall use only the goaltender and one line of players per side. For each row, there are two possible actuations. One is the lateral movement of the players, achieved by using a stepping motor with a linear motion translator (F), while the other is just a spinning movement (τ) implemented simply by dc motor. This latter movement is to be used to kick the ball, while the lateral movement brings a player in line with the ball, as shown in Figure 5.12a. The position of the ball is once again monitored by employing the resistance sheet system described for the Paddle Pool experiment. The goals are cups which hold contact switches, one per goal. When the ball B crosses the goal G, it settles in the cup and closes the switch, thereby registering the score.

Figure 5.12a shows the experimental setup, and figure 5.12b gives its modular form.

Module Requirements:

The weight of a row of men is four kg, the speed of the ball is assumed 5 m/sec. The rows of men is separated by 0.5m, so that the row of men must be moved ten cm in this time to prevent the ball from passing this line (the men on the row are separated by ten centimeters). Thus power required of the transverse linear motion of the men is

$$\text{POWER} = 2 * M * S * S / \text{TIME}^3 = 80 \text{ watts} = 1/10 \text{ HP}$$

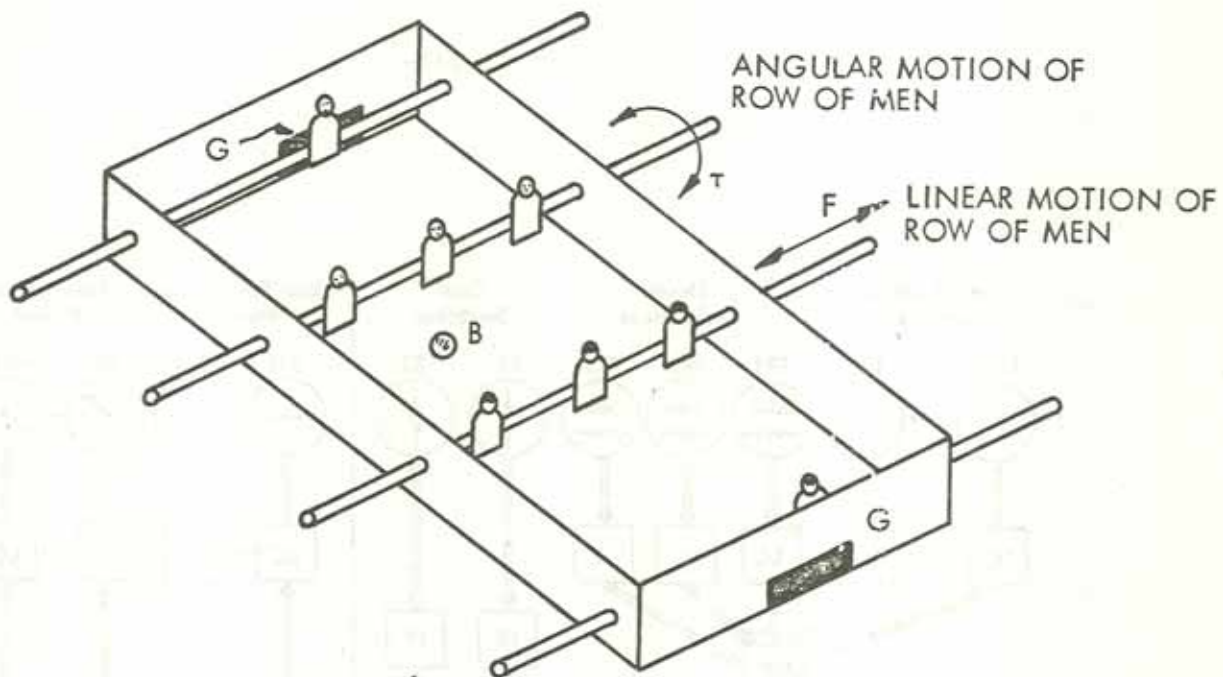
Consider that the angular inertia of the row of men is 0.01 kg-m²(radius of gyration being 5 cm). The men must be swung through three radians in 0.1 second. This gives a torque requirement of 5 Nt-m. The resolution requirement of sheet resistance is as in the last experiment. To get the men within one centimeter, linear motion motor resolution should be eight bits. To locate the men within 1 cm the potentiometer resolution is 6 bits. To provide sixteen positions for striking a four bit resolution is provided for the striking motor. Once again the requirements are listed out:

Transverse motion motor: 1/10 HP, 8 bits

Ball striking motor: 5 Nt-m, 4 bits

Contact switch: 1 bit

Sheet resistance: 8 bits



KEY:

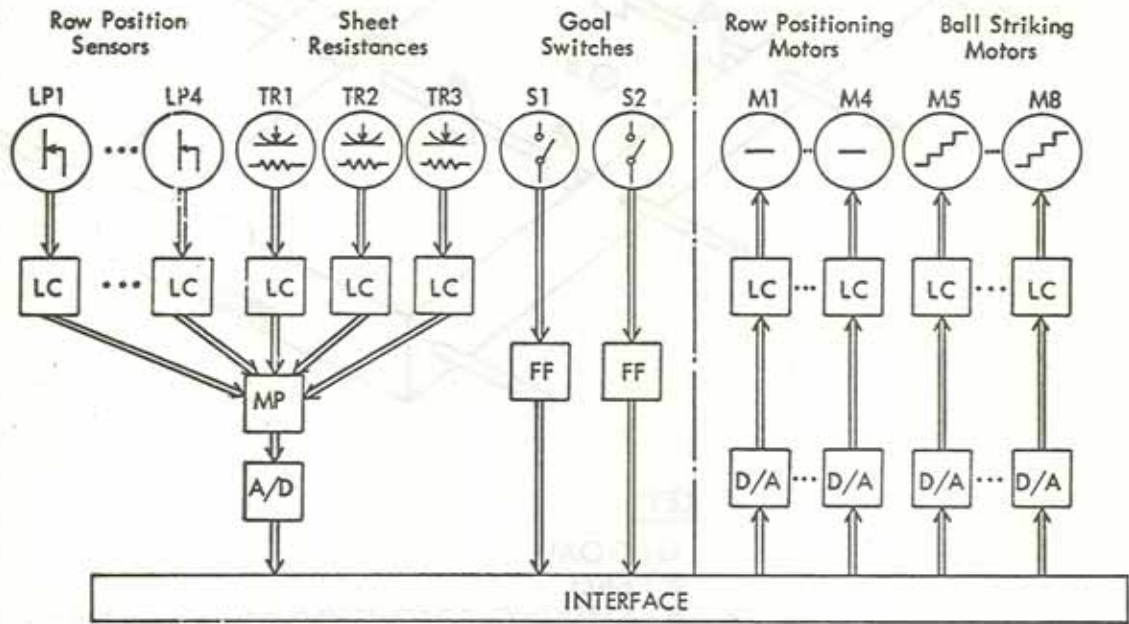
G: GOAL

B: BALL

τ : STRIKING TORQUE (TO ROTATE THE ROW OF MEN)

F: FORCE TO MOVE THE ROWS SIDWAYS

Figure 5.12a Table Soccer.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.12b Table Soccer - Modular Setup.

Potentiometer: 6 bits

Processor Requirements:

Two daemons are needed for sliding the rows of men, and two more daemons are required to rotate the rows of men. We have seen that it is sufficient to move the rows of men in 0.1 second in order to prevent a goal or to strike the ball. Thus, the bandwidths required are 10 Hz each. Then, from the Paddle Pool experiment, we see that one of PDP 11/45 and Intel 8080 will do the experiment.

5.2.13 TILT MAZE

A maze of obstructions is created on a board, the top surface of which is a part of the resistance sheet system. Detection of the position of the ball placed on the board is as before.

The objective of this exercise is to get the ball from the start position to the finish position as quickly as possible by manipulating the maze up and down. The location of the maze openings are preprogrammed. Then, depending on the position of the ball detected by the sensor and the known positions of the maze openings, the maze is tilted as required using two motors, one for tilting about the x direction, and the other for tilting about the y direction. When the ball reaches the "finish", it drops into a cup and closes a switch, thus registering that the job is done. The

angular positions of the board are monitored by a pair of potentiometers.

Figure 5.13a describes the physical arrangement for the experiment, while figure 5.13b presents the modular setup for the same.

Module Requirements:

We assume that the weight of the maze-board is one kilogram, and that the radius of gyration for the board is 0.25 meters. Thus, the angular moment of inertia is $1/16 \text{ kg-m}^2$. Also assumed is that the board will never be tilted by more than one radian from the horizontal position. If the response must be within 0.25 seconds, the required angular acceleration is $64 \text{ radians per second}^2$. (The mechanics of these calculations are shown in Section 5.2.10.)

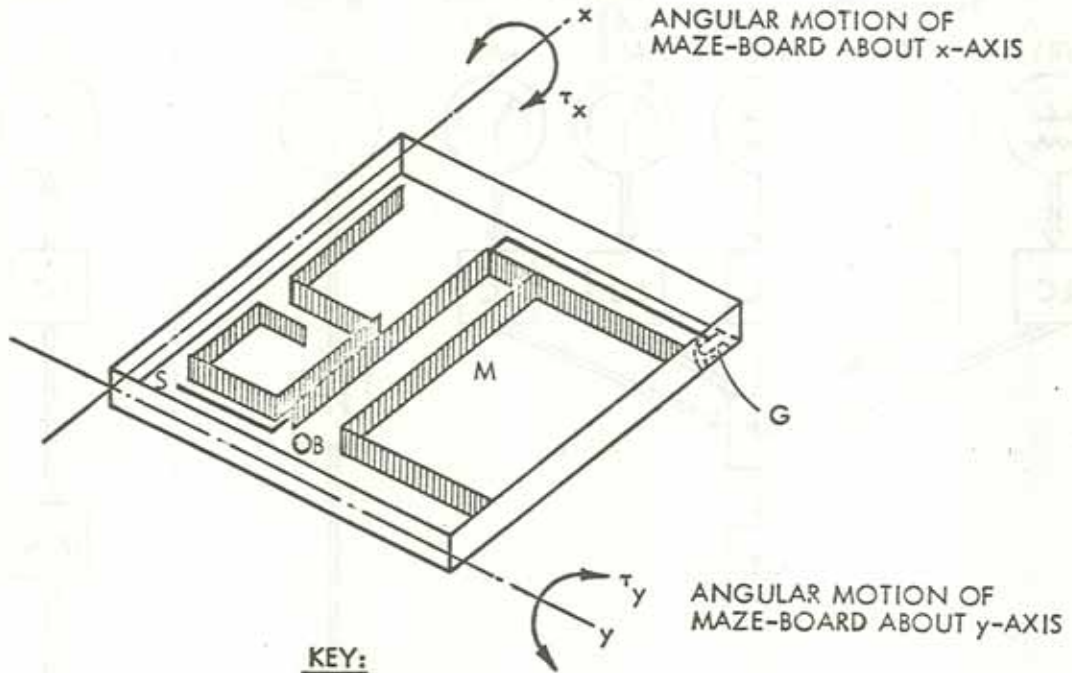
Therefore the torque required of the motors used for tilting the maze is given by:

$$\begin{aligned} \text{Torque} &= \text{angular moment of inertia} * \text{angular acceleration} \\ &= 4 \text{ Newton-meters.} \end{aligned}$$

Resolution of 6 bits will tilt the board within 2 degrees of the desired position. The resolution requirements for the potentiometers are also 6 bits. The module requirements for Tilt Maze are:

Motors: 4 Nt-m, 6 bits

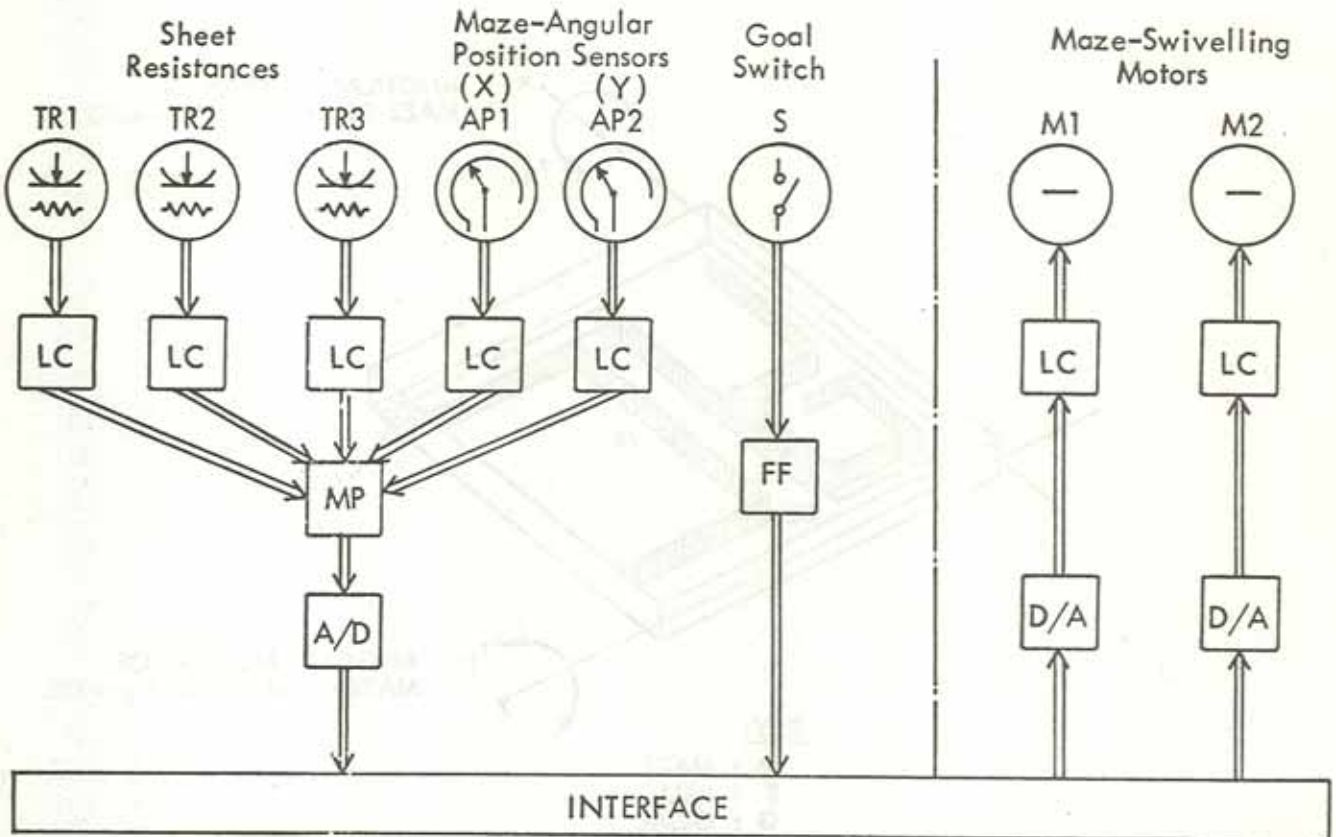
Sheet resistance: 8 bits



KEY:

- M : MAZE
- B : BALL
- G : GOAL
- τ_x, τ_y : MOTOR TORQUE ABOUT x,y AXES
- S : START POSITION

Figure 5.13a Tilt Maze.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.13b Tilt Maze - Modular Setup.

Potentiometers: 6 bits

Processor Requirements:

This experiment requires two daemons - one for the x-axis motor, and one for the y-axis motor. We assume that the maze openings are 5 cm wide. This requires that the motion of the ball be arrested in 0.05 second. Therefore, the bandwidth of the daemons can be put down as 20 Hz each. Plotting (20,2) in Figure 4.4, we note that only one processor is necessary for this experiment.

5.2.14 SOFTWARE SIMULATION EXPERIMENTS

There are four experiments in this set. Basically, these can either be programmed or they might be operated using a set of hardware modules along with the software programs. To illustrate, consider the simulated inverted pendulum experiment. In the hardware version of the experiment, we had employed sensors and actuators to realize the system. After setting up the system, we wrote a program to control the system. In the software version of the experiment, the first change is that there are no physical sensors. If any position or angle or whatever is to be monitored, this is done through programming. In the case of simulated inverted pendulum, the position of the broom and its angles to the horizontal are monitored by a program and then depending on this information relayed to the main

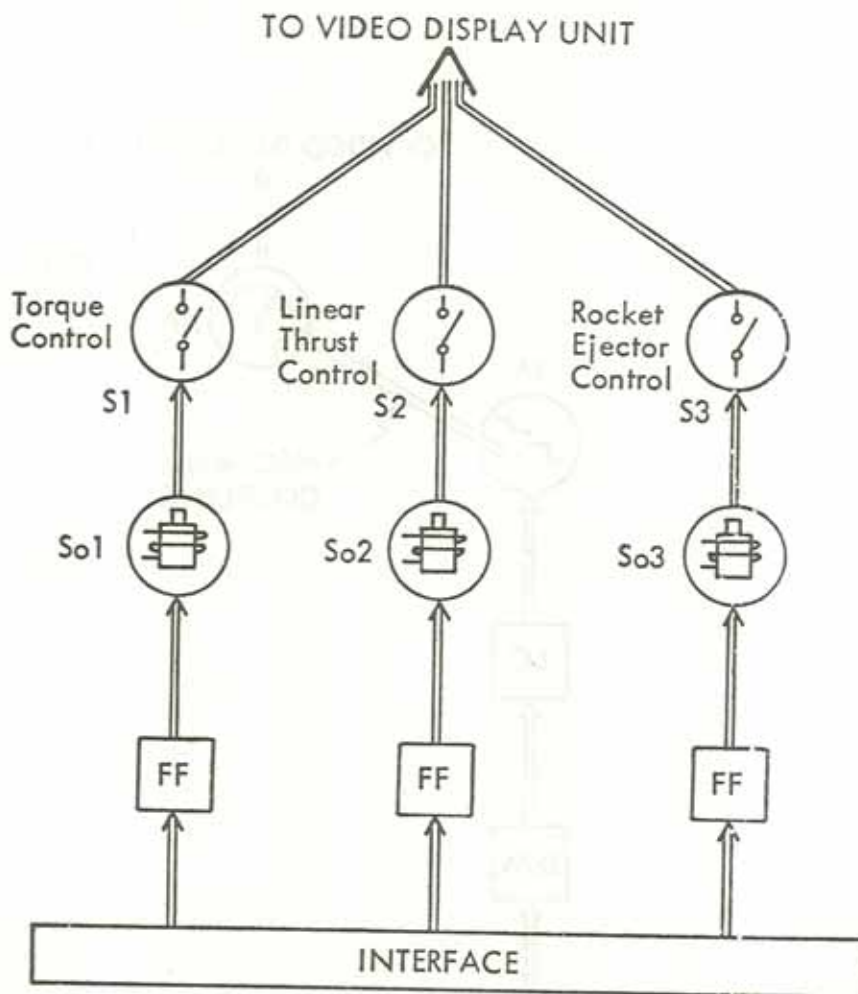
program, the latter activates certain actuators to move the patterns on a cathode ray tube. For the software experiments it is found that the actuators required are the same as the sensors for the hardware experiments. These new actuators require the standard actuators as the processing modules.

In the Space War experiment, we need solenoid switches to change the directions of either the rockets or the missiles. For simulated Ping-Pong, a step motor is required to turn a potentiometer knob for manipulating the image ball on the cathode ray tube. Simulated Inverted Pendulum requires two step motors to turn two potentiometers in order to control the position and the balance of the bar of light on the CRT. Finally, simulated Billiards needs three motors: two for positioning the cue and one for striking the ball.

Diagrams of the modular setups required for these experiments are given in Figures 5.14 - 5.17.

Module Requirements:

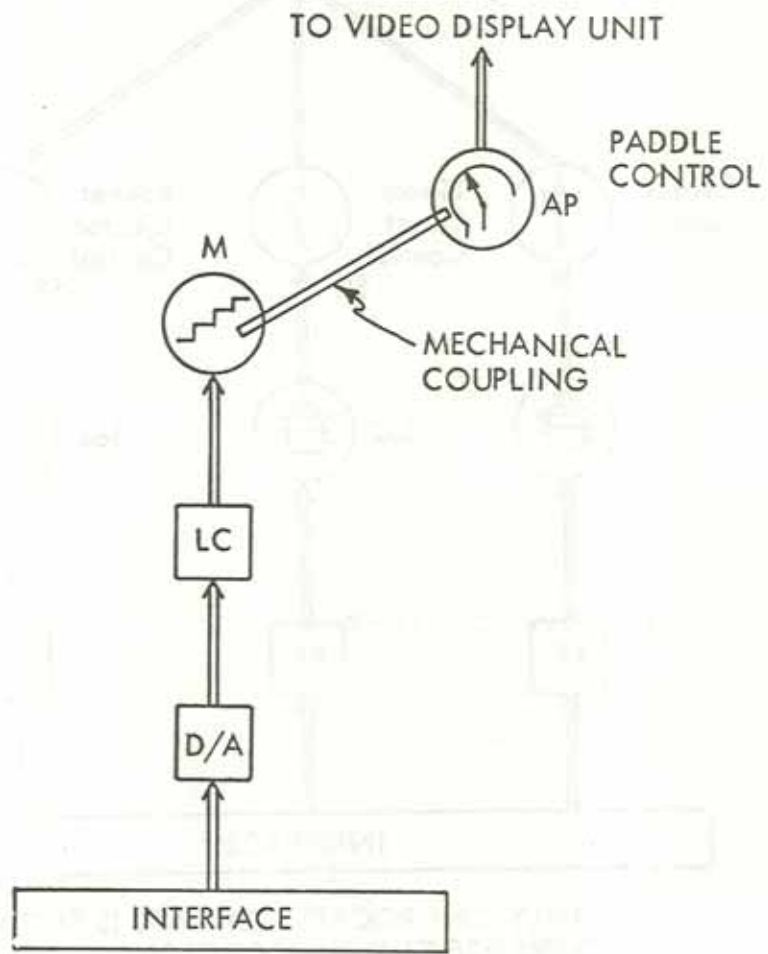
The components required for these experiments are solenoid switches, stepping motors. Since much force is not needed to actuate a switch, solenoids with a 1 Nt rating will do. Assume a weight of 0.5 kg as maximum for a potentiometer which has a radius of gyration of one centimeter. At most, we will need to turn the potentiometer



ONLY ONE ROCKET CONTROL IS SHOWN
THERE ARE TWO SUCH ROCKETS

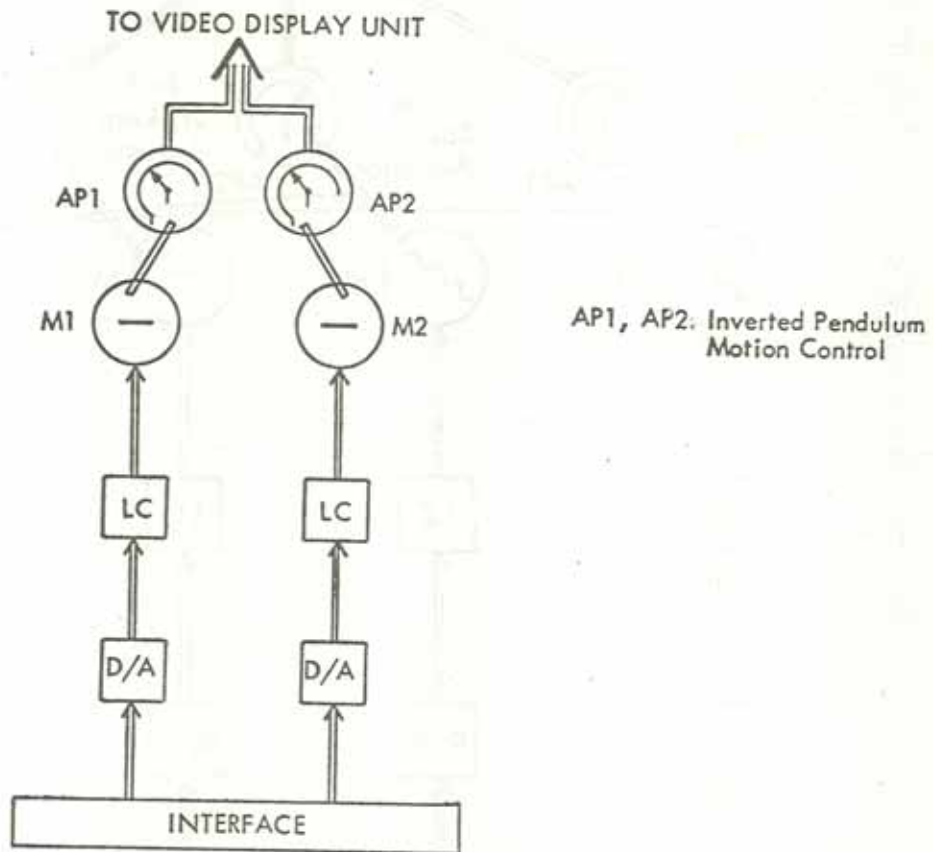
FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.14 Space War.



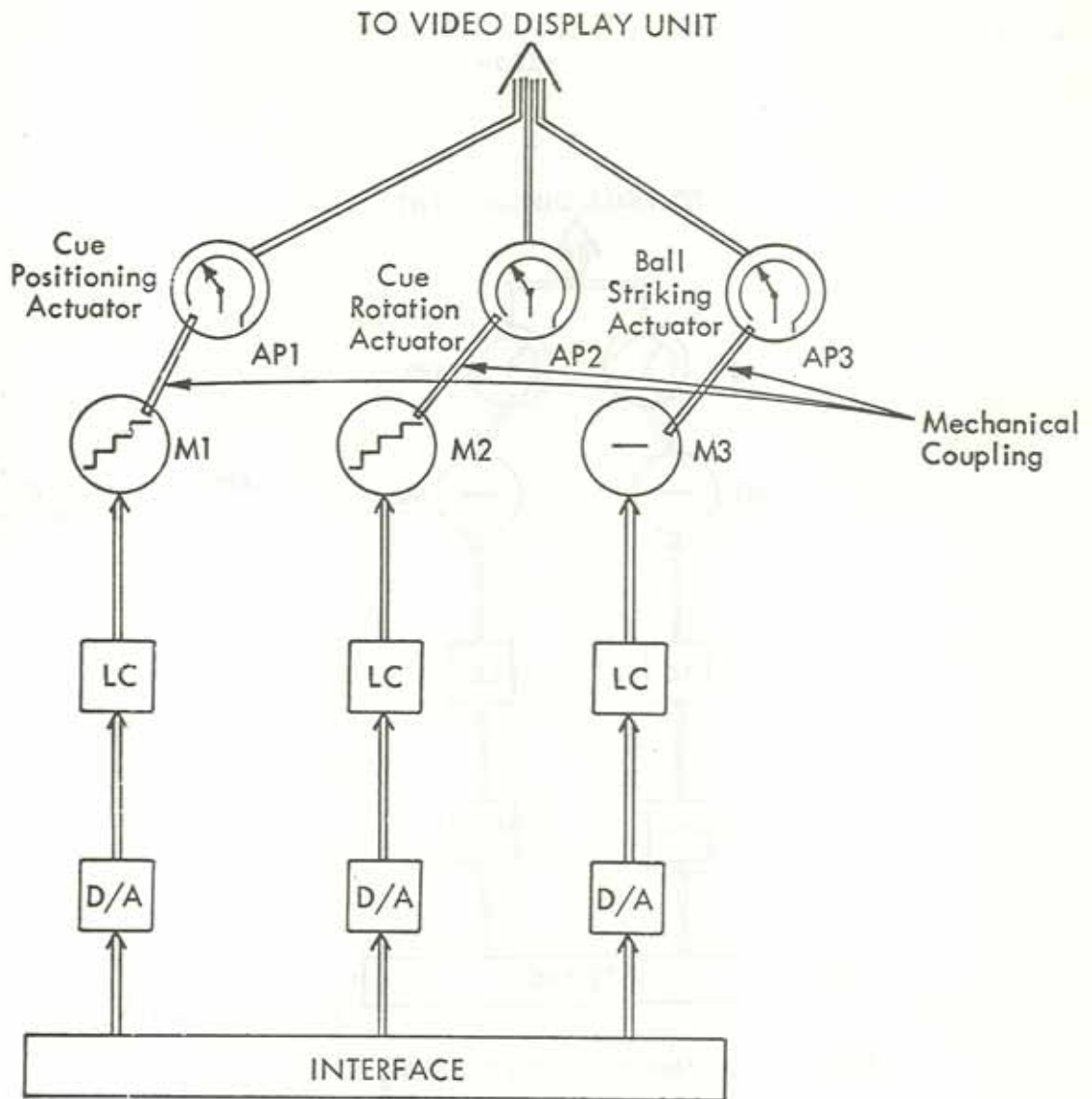
FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.15 Simulated Ping-Pong.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.16 Simulated Inverted Pendulum.



FOR KEY TO SYMBOLS ABOVE SEE TABLE 5.1

Figure 5.17 Simulated Billiards.

through six radians in 0.1 second. This imposes that the motor that is used gives a torque of 0.1 Nt-m. In order to set the potentiometer within two degree of the desired position, a resolution of eight bits is required. The requirements of the modules for the software experiments are listed below:

Stepping motors: 0.1 Nt-m, 8 bits

Solenoid: 1 Nt, 1 bit

Processor Requirements:

The number of daemons required for the software simulation experiments range from one for Simulated Ping-Pong to three for Space War and Simulated Billiards. Simulated Inverted Pendulum needs two daemons. It has been noticed in the arcades that the fastest that a person can change the controls for any game, is 10 times per second. Hence, following the methodology of the earlier experiments, we note that only one of PDP 11/45 and Intel 8080 is sufficient to perform the software simulation experiments.

CHAPTER 6

SUMMARY AND CONCLUSIONS

Here, we briefly summarize the contents of this thesis, present a criticism of the work done, and finally make some recommendations on further research and implementations.

6.1 SUMMARY

This project was inspired by the research being carried out in the Control Robotics Group at Project MAC, M.I.T. In particular, it came about in order to implement a new software approach to control of physical processes by computers, namely the daemonized approach to computer control, suggested by Dertouzos[11] and Geiger[12].

As a first step to implementing the new approach for industrial applications, it was decided to evolve a laboratory course which will make use of these ideas for computer control. In order to keep down the cost of building such a laboratory, it would have to be modular in structure. This way, the same module could be used for different experiments. Thus, the objective of this thesis was fixed as the presentation of criteria for selecting a minimum number of modules in order to make possible the construction of as large a number of experiments as

possible. It was also decided to investigate the processing capabilities required to implement some given control tasks in order to ascertain how much "control" a given microprocessor could be expected to provide.

We started by briefly describing the control phenomenon from the early days to the computer age. Next, we presented the daemonized approach for computer control of physical processes as the foundation on which the laboratory was to be built. Then, to establish the power of a given processor we conducted a more detailed study of the power of processor to perform control under the daemonized approach. A model relating the maximum permissible bandwidths of the tasks to be performed, the average instruction cycle time of the processor, the complexity of the processor instruction set, and the number of tasks the processor must handle, was developed. This model was simplified by making further assumptions, and then a comparison was made between two processors - the Digital Equipment Corporation's PDP 11/45 and Intel 8080. The results of this comparison were presented in both tabular and graphical forms.

The Control Robotics Laboratory was described next. Here, the criteria for the selection of experiments for the laboratory were discussed. Then, these experiments were envisioned in the modular form. Initially, only the conversion modules were considered. This was done for the

simplicity accorded in the minimax that was to be performed. Once the required conversion modules for the experiments were determined, a matrix was drawn up with the rows corresponding to the modules, and the columns to the experiments, and an X was marked for each intersection that required a row module for that column experiment. Then, depending on the sparseness of the use of the modules, these were dropped, as were the experiments, or some modifications in the experimental modular setups were made. When the satisfactory minimax was achieved, the processing modules were determined for each experiment. The power, voltage, current, resolution, and other requirements for these hardware modules were also determined. Finally, the processor requirements for the experiments were computed using the model developed in Chapter 3.

6.2 A CRITICISM OF THE THESIS

In this section, we will present a criticism of the work done on this project. We will restrict the discussion to two topics, namely the modular laboratory, and the model for the power of a processor to perform real-time control.

6.2.1 THE MODULAR LABORATORY

Since no prior work has been reported as regards the modular laboratory for Control Robotics, it is difficult to judge the approach, and the results in arriving at the form of the laboratory. That such a laboratory will be of help in developing the concepts of creative control, cannot be denied.

The criteria presented for the selection of the experiments appear to be quite vague - in particular, the criterion that the selected experiments should be interest-capturing. It is very difficult to ascertain what is interesting and what is not, since this is a subjective issue. To one, watching a batter take a powerful cut at a fastball might be ultimate ecstasy, but then there are others who find intense delight in watching the movement of pieces on a chess-board.

The minimization of modules and the maximization of the experiments were also done in a rather loose fashion. However, the idea was not to present a rigid formalism but rather to achieve practical results. We simply wished to reduce the number of modules, and at the same time be able to conduct many experiments. The matrix method that was presented, we feel, is useful in achieving a reduction in the cost of the final laboratory. Another advantage of the

matrix approach is that it avoids the extraneous details of linear programming or some such formal methods for tackling the minimax problem.

It is possible that there may be interesting experiments that have been overlooked, and also, there might be conversion modules which are superior to the ones selected. These experiments may be included in the laboratory set by considering an additional column in the matrix of experiments and conversion modules. Likewise, new modules may be included by introducing additional rows in the matrix, and deleting rows corresponding to the modules which are being replaced by the new modules.

6.2.2 MODEL FOR PROCESSING POWER

While there are several models available in the literature to measure the performance of a processor, we felt that either they are too detailed, requiring a vast amount of information, or they are not applicable to the case of daemonized control. It is true that the model presented in this thesis, is itself quite simple, and hinges on a number of assumptions; however, it does give measures of performance that could be used for comparative purposes, or by themselves with a fair degree of accuracy.

A few deficiencies in the model have their roots in the

underlying assumptions. One of the assumptions, for example, was that one may expect only eight active daemons out of a system of ten daemons. To have any faith in the veracity of such assumptions, one must have statistical evidence pertaining to that. Even if such evidence were available, there would be exceptions for which predictions made by the model would not be valid. In the example given above, this could mean a revision by 25% of the computed values. Another deficiency that is quite apparent is the uncertainty in determining the coding efficiency.

In spite of these shortcomings, the model provides a simple means for estimating in an approximate way, the computing needs for a given set of tasks, for establishing the superiority of one processor over others in the context of daemonized control, and for handling related issues.

6.3 SUGGESTIONS FOR FURTHER WORK

It would be nice if the results of this project could be physically implemented through a laboratory. The work involved includes the design and construction of various hardware electronic modules. The thesis has only made reference to the problem of interfacing between the microprocessors and the master computer, and between the microprocessor and the real world.

One aspect of modularity that was not mentioned in the thesis has to do with the mechanical parts and hardware needed. Since most of the experiments involve physical motion, a lot of mechanical hardware is required. Some of this is the standard nut-and-bolt variety, but there are also mechanisms required for motion translations like rotary to linear motion, etc. It might be economical to look at this mechanical part modularity.

Once the set of modules has been chosen, appropriate additions may be made by considering some other experiments. This task might necessitate the formulation of just a smaller matrix than before.

In this thesis, a model has been provided for finding out the interaction between a single processor, and the real-world tasks. At present, research is being carried out in the Domain Specific Systems Research Group on multiprocessor scheduling. The problems being investigated include finding answers to questions such as: does there exist an optimal algorithm to schedule a number of tasks among some processors? if no optimal algorithms exist, what is the best that we can do? and so on. In light of this research, another area of investigation is the modification of the processor utilization model described in this thesis, for the multiprocessor environment.

In the formulation of the model, certain informal

assumptions had been made. These may be made more formal either by supplying proofs of their validity, or by considering an altogether different set of statistical assumptions.

REFERENCES

- [1] Elgerd, O.I., Control Systems Theory, McGraw-Hill, 1967.
- [2] Griem, P.D. Jr, "Direct digital control of a glass furnace", presented at the 20th Annual ISA Conference, Los Angeles, Calif., 1965.
- [3] Miller, A., "Automation in the steel industry", Automation, November 1966.
- [4] Bedworth, D.D. and Faillace, J.R., "Instrumenting cement plant for digital computer control", ISA Journal, November 1963.
- [5] Holzer, J.M. et al, "The black box: programmable logic for repetitive control", Minicomputers: Hardware, Software, and Applications, ed. James D. Schoeffler and R.H. Temple, IEEE Press, 1972.
- [6] Mouly, R.J., "Systems engineering in the glass industry", Ibid.
- [7] Gautier, E.H. Jr, and Hurlbut, M.R., "Recent developments in automation of cement plants", Ibid.
- [8] "New process language uses English terms", Control Engineering, October 1968.
- [9] Pike, H.E. Jr, "Process control software", Proceedings of the IEEE, vol 58, No 1, January 1970.
- [10] Pike, H.E. Jr, "Future trends in software developments for real-time industrial automation", Spring Joint Computer Conference, 1972.
- [11] Dertouzos, M.L., "Control Robotics: the procedural control of physical processes", IFIPS Summer 1974, IFIPS Congress, Sweden, 1974.
- [12] Geiger, S.P., "A new language approach to computerized process-control", Master's Thesis, Project MAC, February 1974.
- [13] Dertouzos, M.L., Engineering Robotics section of Project MAC Progress Report, 1972-1973.

- [14] Geiger, S.P., "Macro control language documentation", Control Robotics Group TM 12, Project MAC, MIT, December 1973.
- [15] Papert, S., and Solomon, C., "Twenty things to do with a computer", A.I. Memo 248, M.I.T., June 1971.
- [16] Dertouzos, M.L., DELPHI description from 6.031 class notes, M.I.T., Spring 1976.
- [17] Handbook of PDP 11/45 processor, user manual.
- [18] Handbook of Intel 8080 processor, user manual.

