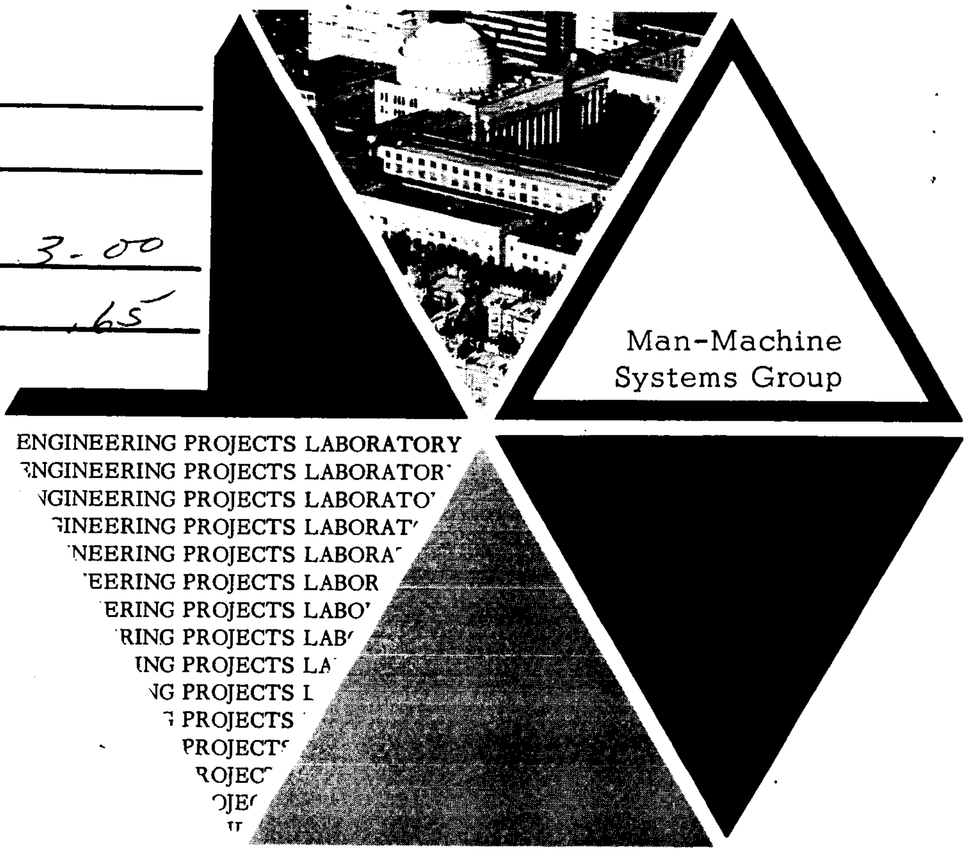
SIMULATION AND ANALYSIS OF OPTIMAL SELF-PACED SECOND ORDER CONTROL SYSTEMS

Philip A. Hardin

DSR 70283-2

July 1967

Engineering Projects Laboratory
Department of Mechanical Engineering
Massachusetts Institute of Technology



DSR 70283-2

SIMULATION AND ANALYSIS OF OPTIMAL SELF-PACED SECOND ORDER CONTROL SYSTEMS

Philip A. Hardin

July 1967

NASA Grant NsG-107-61

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This report covers two topics, and is divided into two sections. The first is a local variation of Bellman's Dynamic Programing. The second topic is the simulation of a second-order self-paced control system. Self-pacing implies ability to control velocity in forward as well as lateral direction; where the inputs are given as values of two space coordinates and have no time dependence.

Section I

The local variation of the Dynamic Programing algorithm is described. Its disadvantages and advantages are discussed in respect to the computing equipment available in 1966. The major advantage is that the grid upon which a continuous system is simulated can be made several orders of magnitude smaller than when using Bellman's Dynamic Programing to simulate the same system. The major disadvantage is that this method is a local method, and is subject to getting trapped in local minima. A procedure for overcoming this difficulty is described.

Section II

A second-order, optimal, self-paced control system was simulated using the I. B. M 7094 computer at the M. I. T. Computation Center. The methods of simulation used were:

a) Bellman's Dynamic Programing

b) a modification of a) that uses the Dynamic Programing algorithm to consider points around a nominal, non-optimal trajectory instead of all points in the state space (a local variation of the regular Dynamic Programing) and

c) a classical gradient analysis.

Methods b) and c) were used to achieve greater resolution than was possible with method a).

The method using Bellman's Dynamic Programing produced satisfactory results, although the grid on which the self-paced system was simulated was very coarse. Method b) increased the grid resolution, and produced better (lower cost) trajectories. Method b) had one weakness in that it could become trapped in local minima. The gradient method of analysis was found to be effective in analyzing only certain types of self-paced system.

I. A LOCAL VARIATION TECHNIQUE OF BELLMAN'S DYNAMIC PROGRAMING

A. Introduction

While working to simulate the self-paced systems, it became apparent that Bellman's Dynamic Programing would not be sufficient to simulate a continuous second order dynamic system of two degrees of freedom. The grid size that had to be used was so coarse that the system simulated was unlike any familiar continuous system. When considering methods to overcome the program of the grid size, as local variation of Bellman's Dynamic Programing was thought to be the best of the methods that were formulated. Further investigation convinced the author that the method might have some application in other areas.

B. Description of the Basic Procedure

The regular Dynamic Programing considers all points in a state space and guarantees an optimal solution in the space represented. The new method, on the other hand, considers only a set of trajectories around a nominal, non-optimal trajectory. It chooses the best trajectory in this set using the regular Dynamic Programing algorithm in this smaller volume of the cost space. It then uses this best trajectory as the new, non-optimal trajectory. This process is repeated until the program makes no change in the last "non-optimal" trajectory. The program then terminates, calling this last trajectory optimal for the system.

The following illustration should help clarify the procedure. The space in Figure 1 is considered as a cost space with the cost axis coming out of the paper and the many other state variables' axes collapsed into the plane of the paper. Constant cost (contour) lines have been excluded

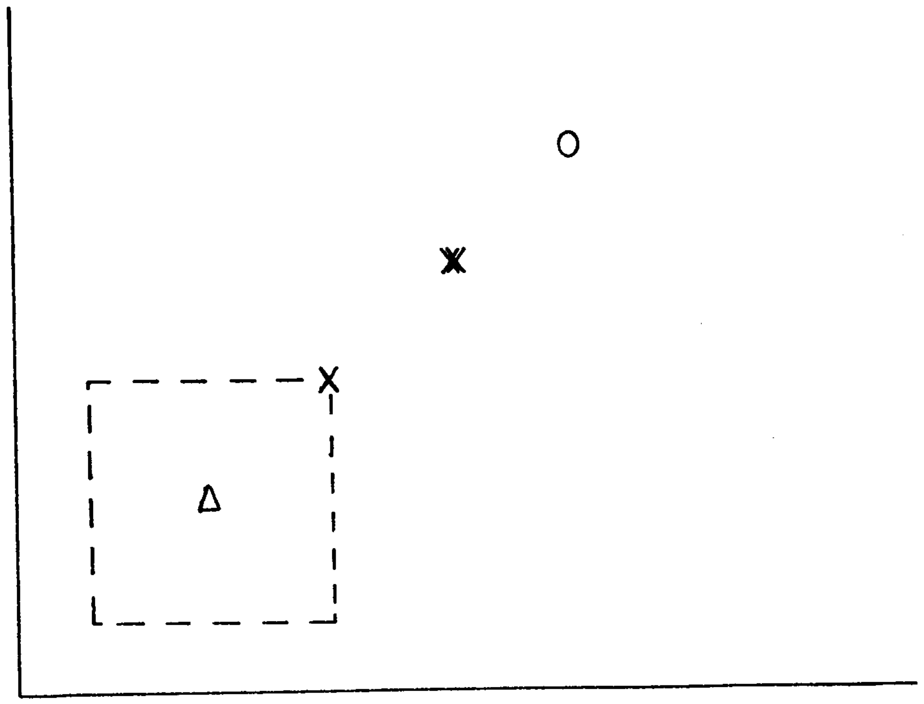


Figure 1

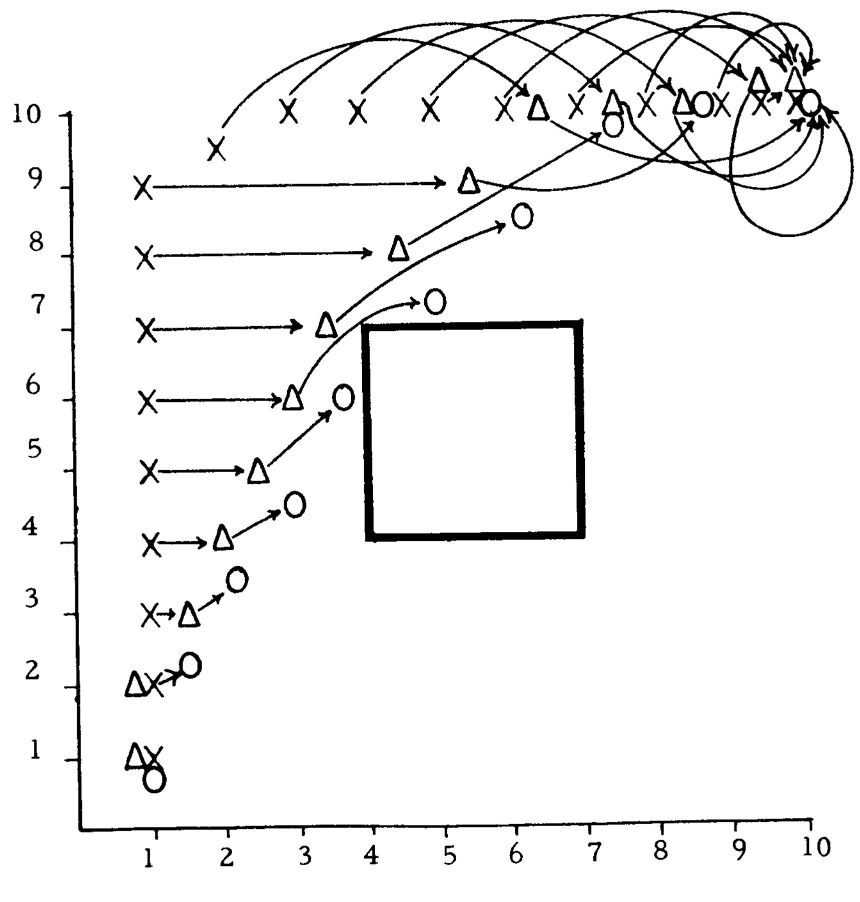


Figure 2

from the figure for the sake of clarity. Assume the nominal, non-optimal trajectory is represented by a triangle at the (1, 1) point, if the gradient at the (1, 1) point were to lie parallel to the X = Y line in the plane, the program would operate in the following manner. The program will consider a set of trajectories enclosed by the square. Given the previously defined gradient direction, the program will choose the upper right corner as the lowest cost point (designated by the X). Then it will consider the set of points in a square around the X. (The second square is not shown but it would be centered about the X and be of the same size and orientation as the previous square.) Now the lowest cost point which the system could consider would be the double X point. The program would continue and the next point to be chosen would be the 0 point. If this point were a local minimum, the program would pick the 0 point on the subsequent trial and then terminate.

Figure 2 is an example of what the process would look like when altering the trajectory of a second order system trying to minimize a function of both time and fuel and moving from start to finish in two dimensions while at the same time trying to avoid an obstacle. The system starts at rest and is stopped at the finishing point. The input to the program is the trajectory marked by the X's. The program moves this trajectory (in several iterations) over to the Δ trajectory and finally to the 0 trajectory, which the program settles on as optimal. The grid that is formed around each point of a nominal non-optimal trajectory (the X's) is shown in Figure 3. This is only a representative configuration, but shows which points the program may consider. A sample output trajectory of one interaction is also shown (the 0's).

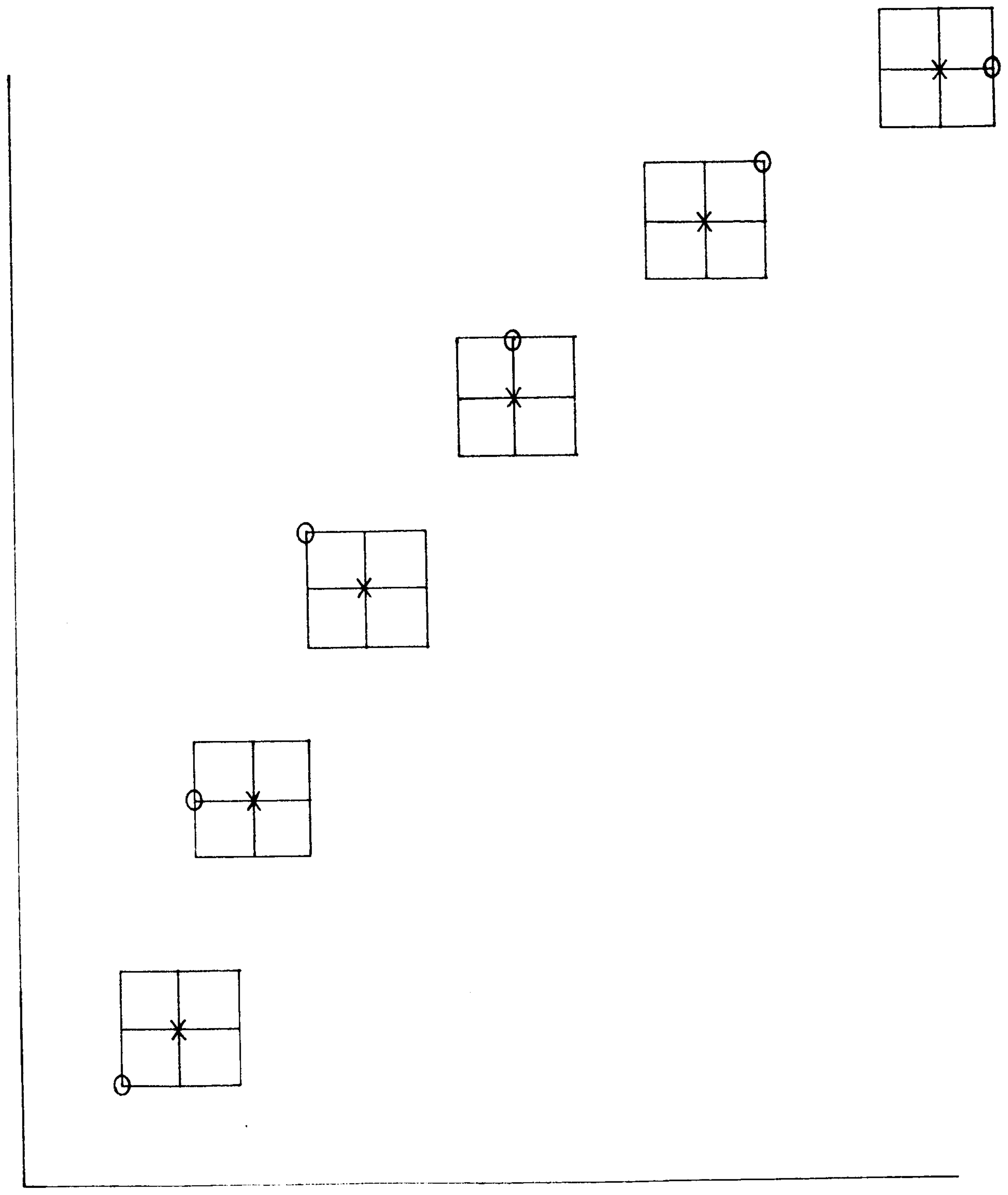


Figure 3

C. Advantages and Disadvantages, and Procedures to Help Overcome the Disadvantages of the Method

The first problem with local Dynamic Programing is that the program can get caught on very shallow plateaus where it cannot determine a cost difference between any of the points it is considering. Of course, the precision to which the cost function is known depends mainly on the accuracy afforded by the available computing machines. Second, this is a local procedure, and it will find only local minima. This is a major restriction to the usefulness of this method. However, an operating procedure was developed which overcomes this objection in part.

To overcome the problem of local minima, a Dynamic Programing program was formulated to find a solution which is the best in the space. (This program should include all values of each state variable that the investigator thinks may be a coordinate of a minimum in the cost space.) The answer from the Dynamic Programing can then be used as the input to the first set of iterations of the local Dynamic Programing routine. It is suggested that the grid size of this first set of iterations be about 1/2 (or more) the size of the grid size of the regular Dynamic Programing. The answer from this first set of iterations can then be used as the input to the next set of iterations of the local Dynamic Programing, with a grid size one half as large as before. This procedure can continue until the answer is refined to any desired degree of precision. The reduction in grid size of only one-half in each set of iterations is suggested because of the non-linearities introduced by discretization of the continuous system. If the grid size is decreased very quickly, there is a greater chance that the system may become trapped on a small plateau or in a small local ravine (possibly created by the discretization of the problem) when happens to be close to the true minimum.

There are several properties which make this method better than other minimum finding techniques. The first and most important is that the complete procedure, utilizing the regular Dynamic Programing will find the true minimum of a cost function to any degree of accuracy desired _if_ the cost function is sufficiently smooth. (In dealing with functions that are very irregular, this procedure will probably yield good results.) The second property is that this method has to calculate no derivatives. A third is that this method is absolutely convergent. If the cost function has one minimum (or more) the procedure will find it (or one of them.) The recommended program procedure also allows for the inclusion of any number of tests for local minimum that the investigator may wish to include.

D. _Conclusions and Recommendations_

The procedure outlined in this section gives the investigator the opportunity to get solutions to continuous control problems to almost any degree of accuracy that is desired. This is, of course, under the condition that the cost function is sufficiently smooth to allow the regular Dynamic Programing, on a successively finer grid, to find a solution sufficiently close to the true minimum.

Ideally, one would like a method that could find the absolute minimum of a cost function that has many local minima and is not smooth or continuous.

II. THE SIMULATION AND ANALYSIS OF SECOND-ORDER OPTIMAL SELF-PACED CONTROL SYSTEM

A. Introduction

For the past several years, researchers have been investigating human behavior using the techniques of conventional control theory. This method of describing the human operator was adapted for two main reasons:

1) this procedure produces a fairly accurate description of experimental results, and

2) the mathematical theory of these systems is reasonably well understood.

This procedure assumes that people perform tasks in a forced-paced mode of behavior. For some tasks, this characterization is adequate; but for most tasks that people perform, they do so in a self-paced manner.

Forced paced systems and self-paced systems can be differentiated as follows:

1) a forced-paced system has as independent variables the sequence of ideal states r, r as a function of time r(t), and the instantaneous time derivatives of r, dr/dt, d^2r/dt^2, etc.

2) a self-paced system has as an independent variable a sequence of ideal states r, but r(t) and the instantaneous time derivatives of r, dr/dt, d^2r/dt^2, etc., are dependent variables. Often r is given as a sequence of space coordinates, i.e., $\left[r\right]$ defines a path in space.

The following example should help illustrate the definition. A car-driver system can be thought of as a self-paced system. On a road where there are both straight sections and winding ones, the driver will go faster on the straight stretches of road than on the winding parts. The driver attempts to keep the probability of having an accident small, but he also values his time and does not want to travel too slowly. Consequently, there is a trade-off between speed (or time consumed in travel) and the probability of having an accident while traveling. Were the speed control taken from the driver and set to be constant throughout the trip, the system would be forced-paced. If the speed were set to be appropriate for the winding sections of the road, the driver would be bored, and probably angry at wasting so much time on the straight sections. If the speed were set to be appropriate for the straight parts of the road, the driver probably could not negotiate the curves, and most likely would meet with disaster.

From experience, one realizes that the speed at which one drives, or at which one performs other activities, is dependent upon many factors. Because these factors are difficult to identify and the interactions among these factors are not well understood, it was considered best to leave modeling a self-paced control system in terms of these "real-world" human factors until after certain fundamental problems were better understood. For the purpose of this report, the system was simplified until it could be reasonably simulated on an I.B.M. 7094 computer. (The reader who is interested in the study of the self-paced car-driver system should consult reference 1.)

The research for this report was undertaken to provide a background or basis for modeling the human operator as a self-paced system. As such, this report is designed to give investigators some feel for the behavior of self-paced systems as well as algorithms and programs for computing the trajectories of optimal self-paced systems.

B. The Self-Paced System Investigated

1. Cost Functions

Throughout the research one basic self-paced system model was used. It was used exclusively so that data from one set of conditions could be easily compared to that of another. The cost function used was

$$J = \left[\sum_{n=0,1}^{N} \left| \ddot{X}(n) \right|^{k} + \left| \ddot{Y}(n) \right|^{k} + PC + TC \right] * \Delta t \qquad (1)$$

where k is a power, and other terms are defined in the Notation List. For most of the simulation, k was left equal to one. Raising its value tended to keep X and Y equal to one or zero. For the simulation using the gradient method of analysis, the cost function was altered somewhat. The alterations and reasons for them will be discussed in the section describing the gradient method of analysis.

There were several other models proposed for the simulation. They were generally of the form

$$J = \sum_{n=0,1}^{N} \left[\left| \ddot{X}(n)^2 + \ddot{Y}(n)^2 \right|^{k/2} + b \left| X(n)^2 + Y(n)^2 \right|^{L/2} + PC + TC \right] * \Delta t \qquad (2)$$

where k, b, and L are constants. This cost function is more realistic than (1), for it computes the absolute magnitudes of acceleration and velocity, penalizes for friction in an elementary sort of way (constant times the absolute velocity), and penalizes for being "within" or "on" an "obstacle."

A cost function based on computation of the absolute magnitude of velocity and acceleration was not used. The absolute values of these quantities were used because of the difference in computation time (about two vs. twenty or more computation cycles). The computation of the friction factor (e.g. $-k(X^2+Y^2)^{1/2}$ was not used because of the required additional computation time. Keeping the program's running time to fifteen minutes or less was necessary because of the requirements of the M. I. T. computation center.

2. The Complete System

The space in which the system was to be simulated was a flat, two dimensional region with an obstacle in the center. The system was to start with $\dot{X}(0) = \dot{Y}(0) = 0$ from a starting point (the 1, 1 position in this investigation), miss the obstacle, and stop at the finishing point (the 10, 10 point). Figure 4 is a pictorial representation of this space.

This model, although not very realistic when friction, high dimensionality, etc. must be considered, was thought to be adequate for this simulation. It represents a second-order self-paced system that is concerned with getting to a finishing point, not hitting an obstacle (in this case, it is probably better to say "stepping on" the obstacle rather than "hitting it"), and operating in an optimal manner. It is penalized for the total acceleration (acceleration and deceleration) it uses, for hitting the obstacle, and for not being at the selected stopping point.

When dealing with a specific problem and with the new generation of computers appearing, it should be possible to simulate more complex systems so that more realistic situations can be analyzed.

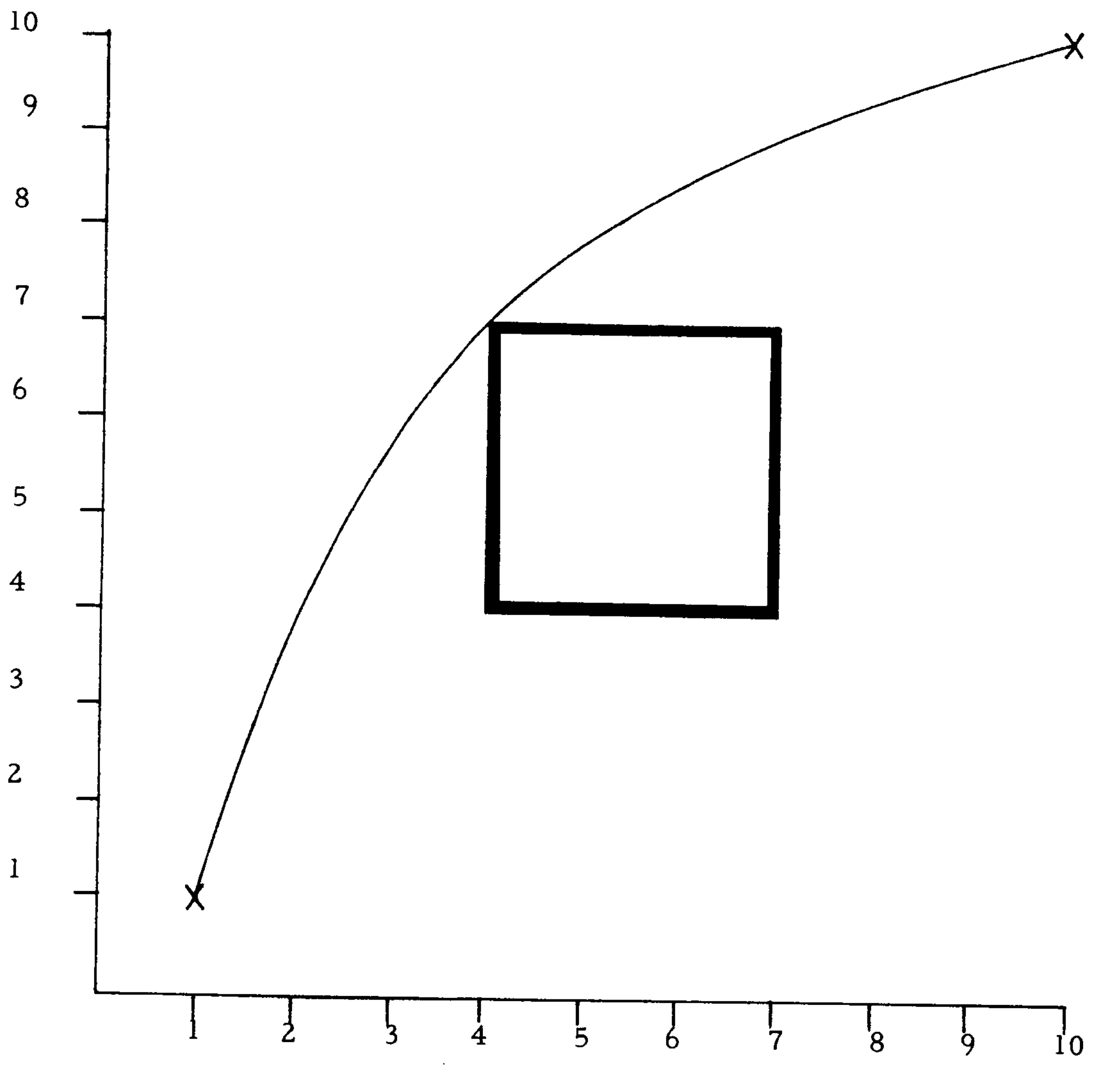


Figure 4

C. The Simulation

1. Bellman's Dynamic Programing

a. Special Aspects of the Program

When this research was undertaken initially, it was thought that the only simulation method that would be needed was Bellman's dynamic programing[+]. While constructing the algorithm for the program, it was discovered that the grid on which the model could be simulated would have to be very coarse. The grid size that was used was 10X units by 10Y units by 10 time units, with the minimal grid divisions being 1 unit in each direction. These divisions imply that the minimal velocity and acceleration units also must be 1. At first thought, it would seem that the number of grid points could be increased. The following argument, however, will prove that there cannot be much improvement in the dynamic programing method at this time without the use of high capacity (and moderate speed) storage devices.

The I. B. M. 7094 computer at M. I. T. has available to the user about 24,000 words (each of 36 bits) of high speed storage (8,000 words of high speed storage are reserved to maintain the system for the high speed processing of programs). The total available information storage is about 850,000 bits. In order to compute the trajectories for a second order system, the dynamic programing algorithm requires that each point in the space be stored as a function of each previous position, velocity, and time. Since it takes seven bits to store ones position in one of 100 different possibilities, and since there are 10X velocities, 10Y velocities, 10X positions,

[+]For the basic theory and further example of the use of dynamic programing, the reader should see reference 2.

10Y positions and 10 time units, there must be a total of 7×10^5 bits of available for information storage. This leaves 1.5×10^5 bits of 4,300 words available to store the program and all other variables. In spite of the problem of the coarse grid, it was believed that some value could be obtained from the model using the dynamic programing method.

There is one fundamental difference between Bellman's dynamic programing algorithm and the dynamic programing algorithm used in this research.

A part of Bellman's method computes a cost, and using this cost, computes the position to which the trajectory will go. If this computed position is close to an allowable position, the trajectory is required to go to that position, and the computed cost is stored as the cost to get to that position. Bellman's method introduces some error into the cost calculations. The method used in this research was to pick a position, go to it, and calculate the cost for getting there by one route. This cost would then be compared to the cost of getting there by all other routes. Then the lowest cost trajectory to that position would be chosen and stored. In the present context, this method appears to be as satisfactory as Bellman's estimated cost method as it calculates the exact cost of getting to a position. As of now, no difficulties have been encountered using this latter method.

The systems simulated by dynamic programing methods should not be thought of as producing truly continuous trajectories. The simulations are more like walking, where the fundamental step lengths are 1 unit in the X dimension, 1 unit in the Y dimension, and they are taken every 1 unit of time. The system path does go from point to point.

However, the system cannot exercise any control except at the discretly spaced grid points. Also it is penalized only for stepping on (or within) the obstacle .

b. Simulation Results

Some results of this simulation are in Figure 5. These show several trajectories are shown for the time cost (TC) equal to 1. The upper left corner of the obstacle is at the 7, 4 position.

In all cases, the trajectories miss the obstacle. But one might question why they are not smoother. There are three reasons for this phenomenon, and two are interrelated.

i. the cost is a linear sum of accelerations wherein two acceleration units at one time cost the same as one acceleration unit at each of the two different times.

ii. in conjunction with i), there is no constraint on power, i. e. acceleration per unit time, for this system model. In real systems, there are always power constraints. If some power constraints were introduced, the trajectories would be smoother.

iii. the grid is very coarse. No matter how limited the acceleration, the trajectory will always have sharp corners.

Another problem was that of several optimal trajectories for a system operating with one penalty criterion. Depending on the conditions of the system, the number of optimal trajectories found were 5, 6, 7, or more. After some investigation, it was concluded that the cause of these many optimal trajectories was the lack of a power constraint in the model. By including an appropriate power constraint, the number of optimal trajectories for a system using one penalty criterion would reduce to a single optimal solution.

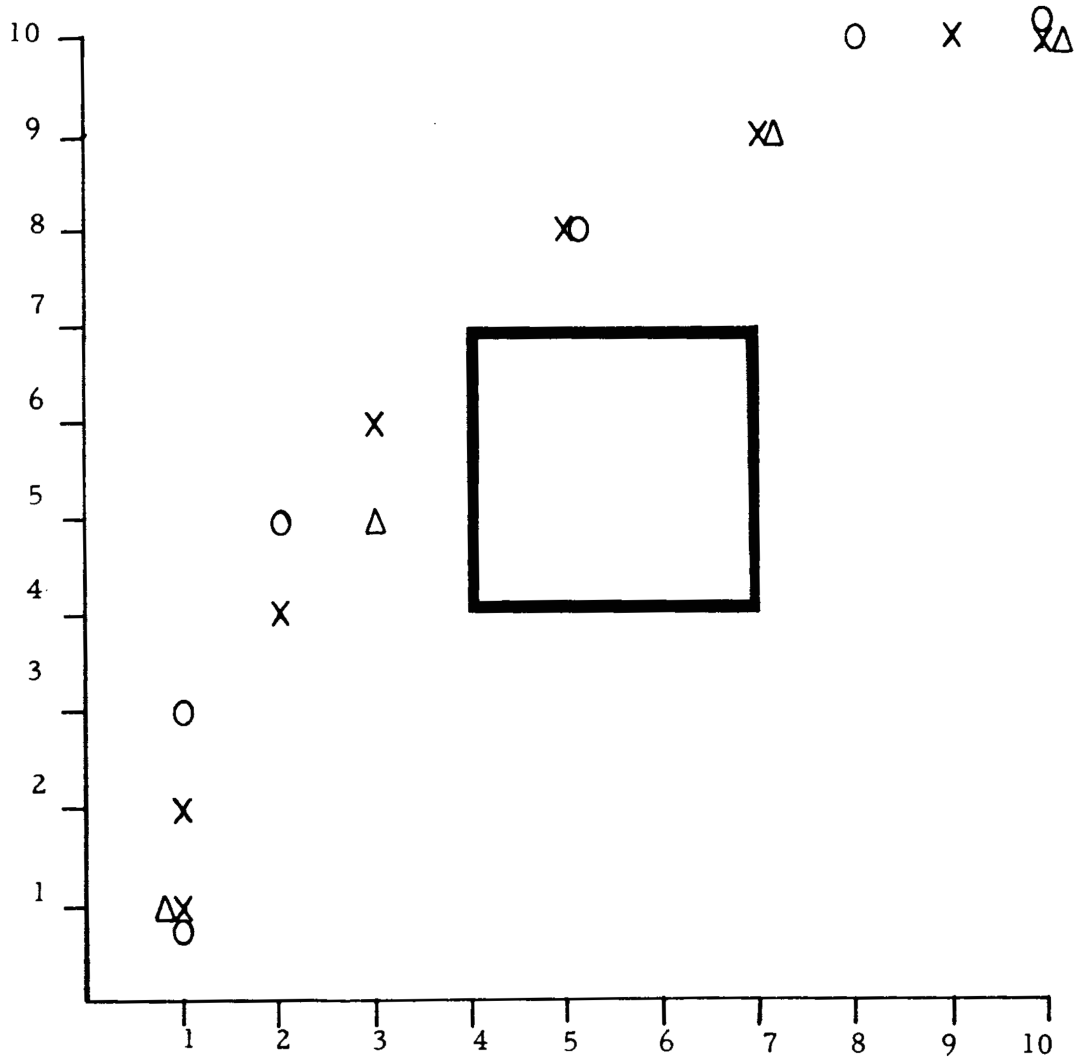


Figure 5

The cost function is that shown in part II, with k = 1, ΔT = 1. The x's represent the trajectory with TC = 1, cost = 15. The 0's represent the trajectory with TC = 2, cost = 22. The Δ's represent the trajectory with TC = 3, Cost = 25.

In the process of the investigation, it was noticed that if in the penalty criterion the absolute value of the acceleration were raised to a power greater than one (as was done in the previous examples), the same result as the power (acceleration per time) constraint would ensu e. This would limit the number of optimal trajectories.

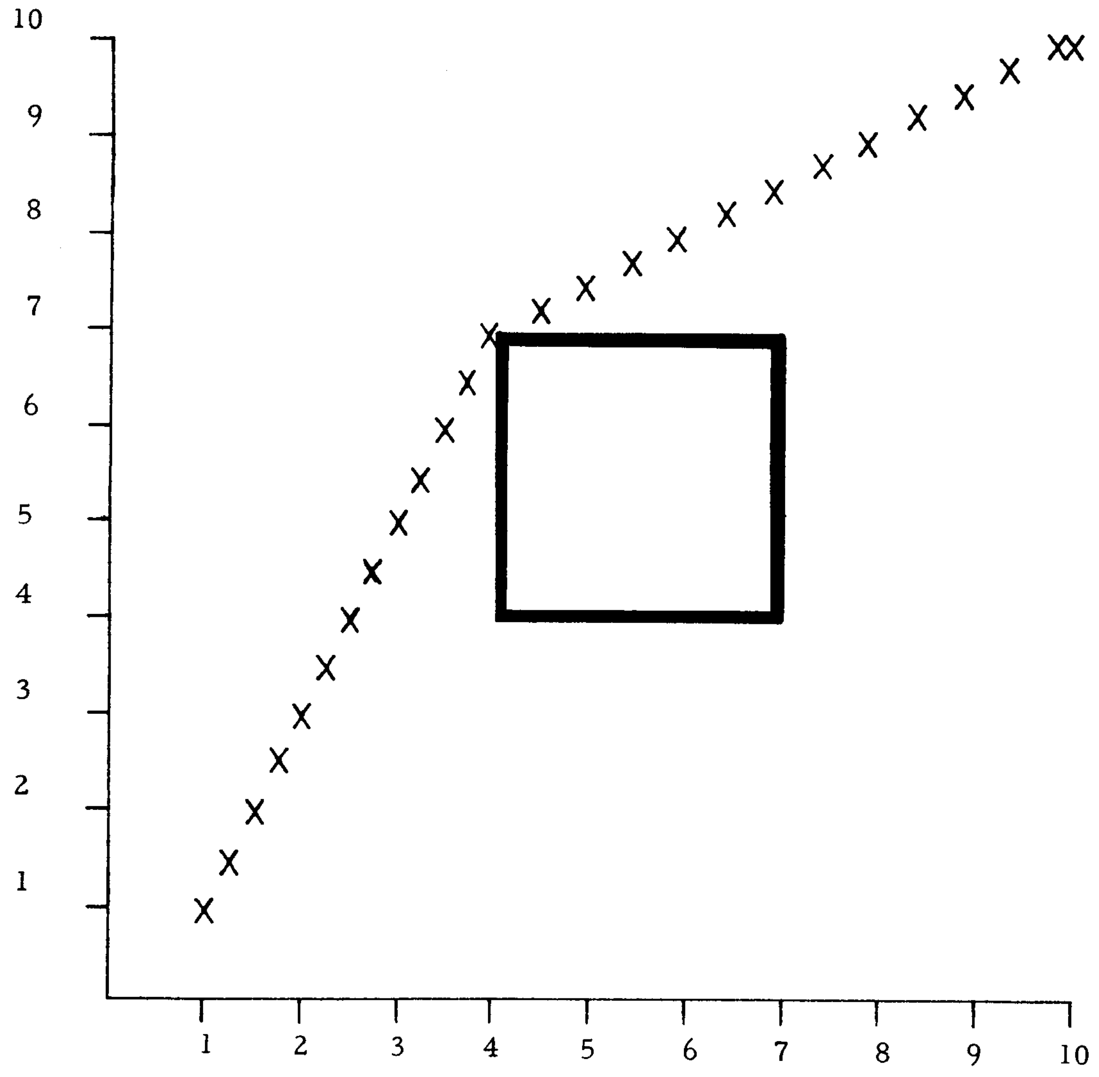
2. The Local Dynamic Program

Because of the extremely coarse grid with which the system had to be simulated using Bellman's dynamic programing, it was thought that some method should be devised which would simulate the self-paced system using a much finer grid. The variation emphazed allows the system to be simulated using almost as fine a grid as is desired. The present limit is about 500 divisions in each of X, Y, and time spaces.

When the local dynamic programing method was first formulated, a grid of 49 points was placed around each of the points on the nominal trajectory. It was found that the running time of a 7094 program to compute an optimal path for TC = 2 and $\Delta t = 1/2$ was over twelve minutes. This prompted efforts to reduce the running time. A resultant program had only nine points on a grid, even though it was recognized that this alteration would make the program more apt to get caught in local minima on the cost surface.

Because of the long running time of the program using 49 points around each nominal point, only a few trajectories were computed using this method. It was found that with only slightly different starting trajectories the program with only nine grid points around each nominal point converged on a cheaper path than the program without 49 grid point. All the results mentioned subsequently will be from the program with the nine grid points around each nominal point. An illustration of one of the results of this method is included in Figure 6.

The results of this simulation substantiated that if appropriate precautions were taken, the grid size could be



TC = 1, ΔT = 1/4, Cost = 14.25

Figure 6

decreased. The major problem to be overcome for this system model was that when a short trajectory (i.e. few steps) was used as the initial nominal trajectory in the program, the program was trapped in a local minimum on the cost surface and produced a non-optimal path. Optimal trajectories (or at least much lower cost trajectories) were successfully computed when long nominal trajectories were input to the program.

At first it was hoped that the problem of getting trapped in the local minima a programing error and not by the presence of minima in the cost space. The program, however, if operating in a flat cost space, tended to lengthen the trajectory, not shorten it. Thus it was concluded that if one attempted to approach the minimum cost trajectory from the direction of a short nominal trajectory, the system would tend to get trapped in a local minimum. If the minimum cost trajectory was approached from the direction of a long nominal trajectory, the system would tend to find the true minimum cost (or at least a lower cost) trajectory.

D. The Gradient Method of Simulation

The gradient method of analysis and simulation was attempted in order to try to decrease the computing time required for the first two methods and to give increased trajectory resolution. The theoretical basis for this method and a general description of the mechanics of the method can be found in reference 7.

When the gradient method was first considered, it was thought that the system model could be simulated as it was defined in Part II, equation 1. Little thought was needed to realize that a gradient method would not work in a cost space that has vertical regions (i.e. infinite gradients). The system was changed to make the vertical regions have finite slopes. This eliminated any infinite or undefined gradients in the region, and the program, it was thought, should have been able to find a minimum cost path.

The method did not work, and the author convinced himself that no gradient method could ever produce an optimal trajectory (excepting some kind of accident) in the cost space defined in Part II. It was concluded that the gradient method would not be of much help in finding optimal solutions of cost functions of the type of concern in this report. This is because of the terminal conditions used with this particular cost function. The conclusions were drawn that if self-paced systems were to be investigated, then a cost function which is suitable to the gradient analysis technique should be used. Of particular importance in the cost function is the term penalizing the system for requiring an extra step or an extra unit of time to satisfy the terminal conditions.

In the interest of discovering if the gradient techniques could be used for simulating some self-paced systems, a system model was developed which has a two dimensional (X and Y) cone for the obstacle (the cone has its vertex pointing upward), and a four dimensional (X, Y, VX, and VY) cone (with its vertex pointing downward), for the stopping condition. The obstacle cone has a height of 50 cost units, and does not extend down beyond COST=0). (The equations for this cone are: $\mathrm{COST} = 50 - k\sqrt{X^2+Y^2}$, and if COST is negative, then COST is set equal to 0.) The cone for the stopping condition covers the entire space.

A program was written to simulate the model. It has 100 time divisions and extremely fine divisions in the X and Y space (significant to eight digits). The actual results of this computation are not included in the results section, but sketches of an input trajectory and the trajectory that the program had computed at the end of five minutes of computing time are included in Figure 7. (Execution of the program was terminated because the maximum running time had been exceeded; prior to stopping, the program had not indicated that the output trajectory was optimal.)

Sketch of the input trajectory (dashed line) and the output trajectory (solid line) of the Gradient Program.

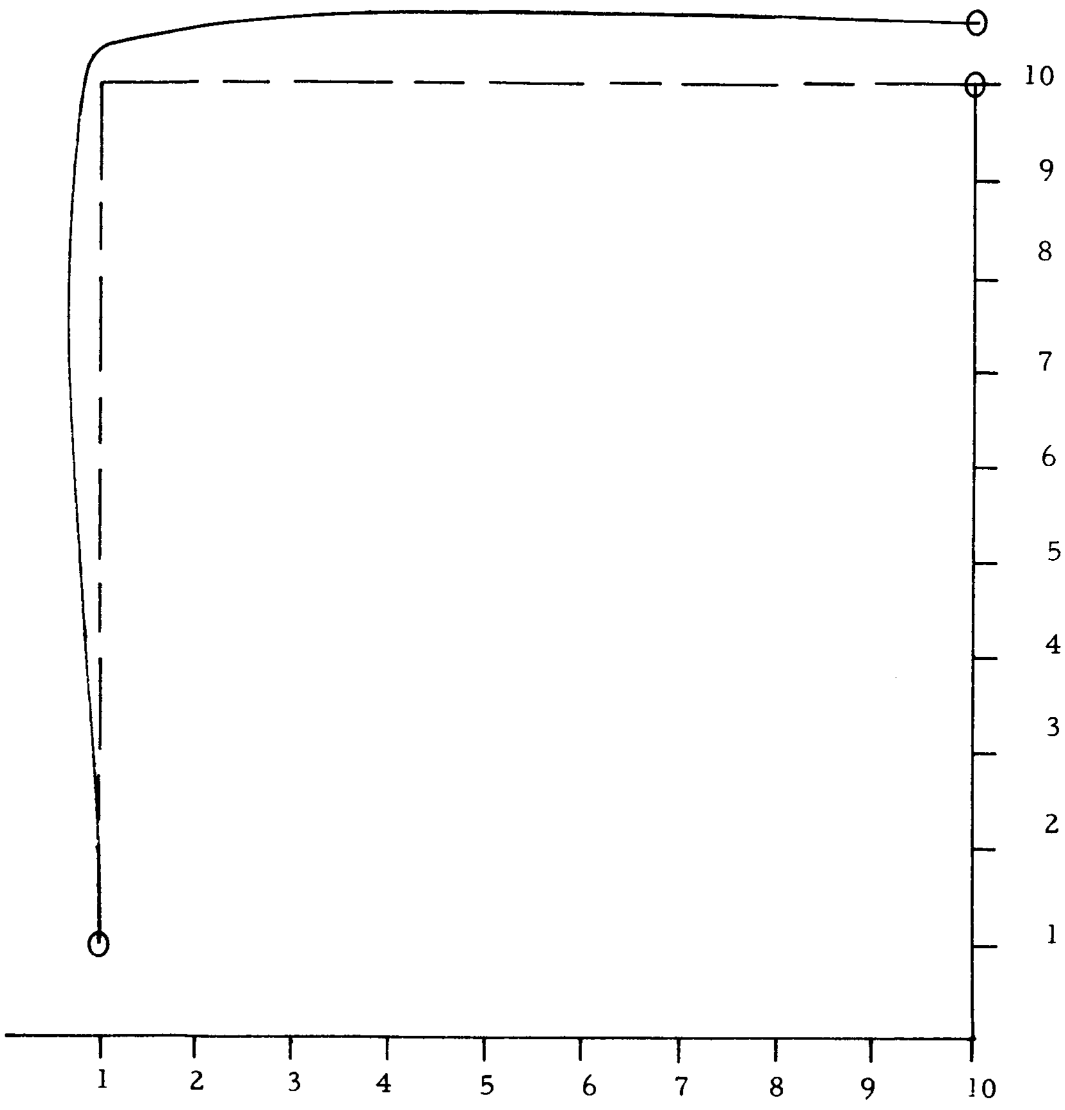


Figure 7

III. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE STUDY

The results of the simulations using dynamic programing are satisfactory. They indicate that optimal self-paced systems can be simulated, and that the resulting trajectories will have relevance to the real trajectories if the system to be simulated is defined with certain restrictions. When larger and faster computers are available, or when much higher capacity, moderate speed storage and an abundance of computing time (on the order of several hours) per run is practical, better estimates of the behavior of general optimally self-paced second (and higher) order systems can be made.

At this time, for investigators attempting to optimize high order continuous systems, the procedure using the local gradient technique will probably produce satisfactory results.

The gradient method of analysis will not aid in the computation of optimal trajectories for the self-paced systems used in this research (i. e with the strictly defined terminal conditions. For the type of systems described in the section dealing with the gradient method of analysis and many others, the gradient method will compute optimal trajectories.

Some of the results of this report should be immediately useful to investigators attempting to compare human performance to optimal self-paced systems. Certain human tasks can be considered to be very much like the program task.

NOTATION LIST FOR CHAPTER II

J	=	the total cost
n	=	an index of summation
Δt	=	the minimal difference between two points on the time grid
t	=	the running time. Generally, $t = n \times \Delta t$
T	=	the time at which the end conditions are satisfied
N	=	the total number of summations, generally $N = T/\Delta t$
X	=	one of the dimensions of a two space region
Y	=	one of the dimensions of a two space region
X	=	velocity in the X direction
Y	=	velocity in the Y direction
VX	=	X
VY	=	Y
X	=	acceleration in the X direction
Y	=	acceleration in the Y direction
PC	=	a cost incurred whenever the system is within a "cost zone" or obstacle.
TC	=	a cost incurred whenever the end conditions are not satisfied, (i. e. whenever the system is not stopped at the preselected stopping point.)

BIBLIOGRAPHY

1. Hardin, Philip A., Simulation and Analysis of Self-Paced Second Order Control Systems. M.S. Thesis, M.I.T.

2. Bolt, Beranek and Newman Inc., Report no. 1335, Job. no. 11164, "Vehicle Control Behaviour and Driver Information Processing," January 7, 1966.

3. Bellman, R.E., and S.E. Dreyfus, Applied Dynamic Programing, Princeton: Princeton University Press, 1962.

4. Hildebrand, F.B., Advanced Calculus for Applications, Englewood Cliffs: Prentice-Hall, Inc., 1962.

5. Tou, J.T., Modern Control Theory, New York: McGraw-Hill Book Company, 1964.

6. Gibson, J.E., Nonlinear Automatic Control, New York: McGraw-Hill Book Company, 1963.

7. Leitmann, George, Optimization Techniques, New York: Academic Press, 1962.