INTRODUCTION

Optimal control systems are of considerable practical and theoretical interest. Although solutions of certain optimal control problems have been known for many years, it is only recently that fairly general, rigorous solution techniques have been developed. Unfortunately, the computational aspects of these solution techniques still present formidable problems. The time-optimal problem which is treated in what follows has a very well developed theory. Our purpose here is to show the utility of hybrid computer techniques. Some of the programming procedures described may also be useful in the solution of other problems.

THE TIME-OPTIMAL CONTROL PROBLEM

It is assumed that the physical system to be controlled (hereafter called the plant) satisfies the following system of first order linear differential equations.

\[ x_i = \sum_{j=1}^{n} a_{ij}(t)x_j(t) + b_i(t)u(t), \quad x_i(0) = x_{i0}, \quad i = 1, 2, \ldots n \]  

To simplify notation vector-matrix notation is employed.

\[ x = A(t)x + b(t)u, \quad x(0) = x^0 \]  

Thus \( x \) and \( b(t) \) are \( n \) vectors and \( A(t) \) is an \( nxn \) matrix. For mathematical convenience it is assumed that \( A(t) \) and \( b(t) \) are continuous.

Given an initial state \( x(0) = x^0 \), the scalar control signal \( u(t) \) must be selected to achieve a specified motion \( x(t) \) for \( t > 0 \). To make the problem of practical interest it is required that \( u(t) \) be piecewise continuous and

\[ |u(t)| \leq 1. \]  

A control which satisfies these two conditions is said to be admissible.

The time-optimal control problem is stated as follows: Given the plant \( (2) \) with initial state \( x(0) = x^0 \) and a desired terminal state \( x^d \), find an admissible control which makes \( x(t) = x^d \) for the smallest possible \( t \). If such an optimal control exists it will be called \( u^*(t) \). The minimum time and optimal motion associated with \( u^*(t) \) will be denoted by \( t^* \) and \( x^*(t) \), i.e., \( x^*(t^*) = x^d \). Our concern is the computation of \( u^*(t) \).

The theory of the problem has been treated by many authors (see, for example, the article by LaSalle\(^1\) or the book by Pontryagin et al\(^2\)). The basic result of the theory is the following. If a \( u^*(t) \) exists it is given by

\[ u(t) = \text{sgn} \: v(t) \quad \dagger \]  

where

\[ v(t) = -(b, \xi) \quad \dagger \]  

and \( \xi(t) \) satisfies the adjoint differential equation

\[ \xi' = -A'(t)\xi, \quad \xi(0) = \eta \quad \dagger \]  

\( \dagger \) The notation is conventional. \( \text{sgn} \: v = 1, \: v > 0; \text{sgn} \: v = -1, \: v < 0; \text{sgn} \: v \) undefined, \( v = 0. \quad (b, \xi) = \sum_{i=1}^{n} b_i(t)\xi_i(t). \)

\( A'(t) \) is the transpose of \( A(t) \).
for some initial condition vector \( \eta = \eta^* \). Equation (4) says that \( u^*(t) = \pm 1 \), i.e., it is a “bang-bang” control.

Since \( \text{sgn} \ v \) is not defined for \( v = 0 \), \( u^*(t) \) is undefined unless \( v(t) = 0 \) only at isolated instants of time (the switching times). If for all \( \eta \neq 0, v(t) = 0 \) only at isolated instants of time, the system (2) is called “normal” or equivalently, “completely controllable.” LaSalle considers the normality condition at length. If \( A \) and \( b \) are constant he shows that (2) is normal if the vectors \( A^k b, k = 0, 1, \ldots, n-1 \), are linearly independent. In what follows it is assumed that (2) is normal.

The fact that \( \eta^* \) is not explicitly defined is a disadvantage in practical computation of \( u^*(t) \). Each value of \( \eta \) produces a control \( u = \text{sgn} \ v \) which, when applied to the plant equations (2), produces a time-optimal motion. In fact, when the set of all values of \( \eta \) is used, the set of all time-optimal motions is generated. The computational problem is to select the particular optimal motion \( x^*(t) \) which passes through the desired terminal state \( x^d \). This can be done by searching through all value of \( \eta \) until an optimal motion through \( x^d \) is obtained. Since for each \( \eta \), (2) and (6) must be solved together, it is impossible to consider all \( \eta \). Instead, a finite set of \( \eta \) values must be used with the hope that one of the optimal solutions will pass close to \( x^d \). A better approach is to formulate an algorithm which generates a sequence of \( \eta \)'s \( \{\eta^*, \eta^1, \ldots\} \) which converges to \( \eta^* \). In later sections both this approach and the search approach are programmed on a hybrid computer.

Before considering the hybrid computer a few additional facts should be noted. First, the magnitude of the \( \eta \) vector is immaterial. This is so because \( v \) is proportional to the magnitude of \( \eta \) but \( u^*(t) \) depends only on the sign of \( v(t) \). Thus the magnitude scaling of \( \eta \) (and \( \xi \)) is immaterial. Finally, normality implies that if there is an optimal motion through \( x^d \) it is unique and so is the associated control. Thus even though there are many \( \eta^* \) which produce the same optimal control, there can be only one optimal control.

THE HYBRID COMPUTER

The hybrid computer employs both analog and digital computing devices, which may conveniently be organized in three categories: analog, analog-digital, and digital, according to the kind of signals with which they deal. For the class of problems considered here it is most natural to let the analog devices integrate the differential equations and to reserve the digital devices for problem control. Because of the rather modest requirements on the digital portion of the system it can consist of rather simple logic elements.

Figure 1 gives symbols for the basic computer elements. Conventional analog elements are not shown. To make signal designations clear, light lines and lower-case letters are used for analog signals and heavy lines and upper-case letters are used for logic signals. \( A \) denotes the complement of \( A \). Elements (a) - (f) are analog-digital and elements (g) - (k) are digital.

**Integrator** (a) and (b). Each integrator is separately controlled through its operate (O) and hold (H) inputs.

\[ A = 0: e_b = -\epsilon, \text{ initial condition mode, } B \text{ has no effect} \]
\[ A = 1, B = 0: e_b = \alpha \int_0^t e_* dt - \epsilon, \text{ operate mode} \]
\[ A = 1, B = 1: e_b = \text{ constant, hold mode} \]

**Switch** (c) single throw, (d) double throw.

\[ A = 0: i = 10e_b, e_b = -\epsilon \text{ (for (e), } e_2 = 0) \]
\[ A = 1: i = 10e_b, e_b = -\epsilon \]

**Comparator** (e). The comparator has hysteresis with dead zone \( \pm \epsilon \). \( B = 1 \) locks the comparator output \( A \) at the level present when \( B = 0 \rightarrow 1 \).

\[ e_1 + e_2 > \epsilon: A = 1 \]
\[ e_1 + e_2 < -\epsilon: A = 0 \]
\[ |e_1 + e_2| < \epsilon: A = 0 \text{ or } 1, \text{ depending on past history of } e_1 + e_2 \]

**Analog Memory or Track-Transfer** (f)

\[ B = 0: e_b = e, \text{ initial condition mode, } A \text{ has no effect} \]
\[ B = 1, A = 0 \rightarrow 1: \text{ take } e_1 + e_2 \text{ and transfer to } e_b \text{ hold } e_b \text{ constant until next } A = 0 \rightarrow 1. \]

**AND Gate** (g)

\[ D = A \cdot B \cdot C, D = 1 \text{ only if } A = B = C = 1 \]

**OR Gate** (h)

\[ D = A + B + C, D = 0 \text{ only if } A = B = C = 0 \]

**Flip-flop** (i). The inputs \( A \) and \( C \) cannot simultaneously equal 1

\[ A = 1, C = 0: D = 1, \text{ set flip-flop} \]
\[ A = 0, C = 1: D = 0, \text{ clear flip-flop} \]
\[ A = C = 0: D = 1, 0, \text{ store 1 or 0} \]
\[ A = C = 0, B = 0 \rightarrow 1: \text{ reverse } D \]

* The notation \( B = 0 \rightarrow 1 \) is used to indicate the transition of \( B \) from logic 0 to logic 1.
HYBRID COMPUTER SOLUTION OF TIME-OPTIMAL CONTROL PROBLEMS

Figure 1. Computer Symbols.

Single-shot (j)
\[ A = 0 \rightarrow 1: \quad D = 1 \text{ for } T \text{ seconds, otherwise } \]
\[ D = 0 \]

Time-delay (k)
\[ B = A \text{ delayed by } \Delta \text{ seconds} \]

Some additional comment on the general function of these elements should be made. The comparator dead zone \( \pm \epsilon \) is small but necessary since it is the only way to assure uniformly fast and positive transition of the comparator output. The track-transfer memory element combines in a single unit the function of two track-hold memory units connected in cascade, along with initial condition provisions. For ease of programming, each element with a logic output also has the complemented output. Finally, it should be noted that the elements are not controlled by a master system clock. This simplifies programming and prevents possible errors due to the quantizing of time.

Additional computer equipment would include input control buttons, display lights, shift registers, clocks, counters, problem check provisions, etc. However, it is not necessary to go into these details here.

THE COMPUTATION OF OPTIMAL SOLUTIONS AND THE REACHABLE SET

Figure 2 shows the computer programming for generation of time-optimal solutions. For brevity the details of programming in blocks I and III are not shown. The integrators must all be started together; thus the operate (O) inputs on blocks I and III are joined. Each time \( A = 0 \rightarrow 1 \) an optimal solution \( x(t) \) is generated. By driving \( A \) from an oscillator and running the integrators on a fast time scale, conventional repetitive operation is obtained. By systematically changing \( \eta \) a search for \( \eta^* \), which produces the desired optimal control \( u^* \) and motion \( x^* \), can be made.

Actually, if all \( \eta \) values are to be scanned, much more general information concerning the control of (2) may be obtained. In particular, the reachable set of (2) may be determined. The reachable set \( R(t, x^o) \) is the set of all states which may be obtained at time \( t \) (starting from \( x^o \) at time \( t = 0 \)) by means of admissible controls. Thus \( R(t, x^o) \) determines the extent of possible motion with admissible controls. It can be shown that the set of all time optimal solutions delineates the boundary of \( R(t, x^o) \).\(^3\)

Thus to obtain a boundary point of \( R(T, x^o) \) (\( T \) is a particular value of \( t \)) an \( \eta \) is chosen and a solution from Figure 2 generated. At \( t = T, x(t) \) is the boundary point. By taking a suitable set of \( \eta \)'s enough boundary points are obtained to define \( R(T, x^o) \). Let us determine a computer program for the second order case \( (n = 2) \).

First, a procedure for scanning through a set of values for \( \eta \) is required. Since the magnitude of \( \eta \) is immaterial, only its direction is essential. Thus take \( \eta_1 = \sin \theta \) and \( \eta_2 = \cos \theta \). By scanning \( \theta \) in the interval \((0, 2\pi)\) a suitable set of time optimal solutions is generated.
Figure 3 shows one procedure for generating $\sin \theta$ and $\cos \theta$. The equation
\[ \ddot{z} + z = 0, \quad z(0) = 1, \quad \dot{z}(0) = 0 \quad (7) \]
is solved, producing the functions $z = \cos \tau$ and $-\dot{z} = \sin \tau$ which, when held at $\tau = \theta$, produce the desired outputs. Operation of the various computer elements should be clear from the timing diagram (a heavy line in the diagram indicates the presence of logic 1). The outputs $\cos \theta$ and $\sin \theta$ are available $\tau = \theta$ seconds after the input command $B$ is initiated and must be read out before $B$ returns to logic zero.

Figure 4 shows the program for generating the boundary points of the reachable set $R(T, x^r)$. Before the start of the computation ($S = 0$) the oscillator is turned off ($A = 0$) and the integrators in Figure 2 are in the reset mode. Since the single-shot is in its rest state, $B = 0$ and the function generator is in its reset state. Thus $\eta_0 = 0$ and $\eta_1 = 1$. ($\theta = 0$). At $S = 0 \rightarrow 1$ the first run begins since then $A = 1$. At $t = T$ the comparator output $D = 0 \rightarrow 1$, triggering the memory units and transferring the first boundary point to their outputs. Since the outputs are held until $t = T$ in the next run, there is ample time for read out (an $x, y$ recorder or oscilloscope depending on the time scale). In the interval $(0, 1)$ the single-shot causes $B = 0$ and the function generator is reset. At time $1 + \theta$ the $\theta$ for the next run is set in. By generating $\theta$ in a slow time integrator which is controlled by $S$, $\theta$ is allowed to increase slightly from run to run. When $\theta = 2\pi$ the computation is complete. In the interval $(9, 10)$ of the first run the integrators are reset for the next run, the new values of $\eta_1$ and $\eta_2$ being used. In the second and subsequent runs the above pattern is the same.

Extension of the above program to higher order systems is fairly straightforward. The main complication is the scanning of the higher dimensional $\eta$ vectors. Also the number of runs must be increased greatly because of the higher dimensionality of the $R(T, x^r)$ boundary. In fact, finding a suitable way of storing the boundary points may be more of a problem than their computation.

THE REGULATOR PROBLEM, ITS ITERATIVE SOLUTION

In this section an iterative procedure is described for solving the time-optimal regulator problem where the terminal state $x^d = 0$. A sequence of vectors $\{\eta', \eta''', \ldots\}$ is produced which converges to $\eta^*$. Additional detail and proofs are given in the paper by Neustadt.\(^3\)

First a condition which $\eta^*$ must satisfy is given. In what follows $\eta$ is always taken so that
\[ (\eta, x^r) > 0 \quad (8) \]
For each $\eta$ there is always a corresponding $v(t)$ defined by (5) and (6). Thus the function
is always defined. Because of (8), \( f(0, \eta) > 0 \).
Also \( f(t, \eta) \) is clearly nonincreasing. In fact, if there is a time-optimal solution \( f(t, \eta) \) eventually becomes zero. Let \( t_s(\eta) \) be the time when \( f(t, \eta) = 0 \). Neustadt shows that the following facts are true: \( t_s(\eta) \leq t^* \), \( t_s(\eta) = t^* \) only if \( \eta = \eta^* \), and \( \text{grad} \ t_s(\eta) = 0 \) only if \( \eta = \eta^* \). Thus by choosing \( \eta \) to maximize \( t_s(\eta) \), \( t_s = t^* \) and \( \eta = \eta^* \). Since grad \( t_s(\eta) = 0 \) only if \( \eta = \eta^* \) the method of steepest ascent is appropriate to maximizing \( t_s(\eta) \). Thus
\[
\eta^{i+1} = \eta^i + k \text{grad} \ t_s(\eta^i)
\]
(10)
For \( k > 0 \) sufficiently small and \( (\eta^*, x^*) > 0 \) the sequence of vectors \( (\eta^i, \eta^i, \ldots) \) converges to \( \eta^* \).
Neustadt also gives a formula for determining grad \( t_s(\eta) \). It can be shown that evaluating the formula is equivalent to the following:
\[ t = sgn \ v(t) \text{ and solve } \]
\[ \dot{w} = A(t)w + b(t)u, \ w(0) = x^0 \]  
(11)
from \( t = 0 \) to \( t = t_s \), at \( t = t_s \) run time backwards and solve (11) with \( u = 0 \), when \( t = 0 \) determine \( w \) and call it \( \gamma(\eta) \), then
\[ \text{grad} \ t_s = K(\eta) \gamma(\eta), \ K(\eta) > 0 \]  
(12)
Since \( K(\eta) \) is a positive quantity it is not necessary to determine its value in order to employ the method of steepest ascent. Instead of (10)
\[ \eta^{i+1} = \eta^i + \rho \gamma(\eta^i) \]  
(13)

is used \( k = \frac{\rho}{K} \).

To summarize, the iterative procedure consists of the following steps:
(a) Choose an initial \( \eta^0, t^0 = 0 \) so that \( (\eta^0, x^0) > 0 \)
(b) Use (5), (6) and (9) to define \( f(t, \eta^i) \)
(c) Determine \( t_s(\eta^i) \) from \( f(t, \eta^i) = 0 \)
(d) Let \( u = sgn \ v(t) \) and solve (11) from \( t = 0 \) to \( t = t_s(\eta^i) \)
(e) At \( t = t_s(\eta^i) \) set \( u = 0 \), reverse time and solve (11) from \( t = t_s(\eta^i) \) to \( t = 0 \)
(f) At \( t = 0 \) set \( w = \gamma(\eta^i) \)
(g) Determine the next value of \( \eta^i \) from (13)

THE PROGRAMMING OF THE ITERATIVE SOLUTION

The steps in the iterative procedure are sufficiently involved that it is wise to break the programming down into a series of subproblems: the solution of (11), the control of (11) to produce \( \gamma(\eta) \), the generation of \( t_s \), and the implementation of (13).

Figure 5 shows the programming of (11). Block I, as before, generates \( v(t) \) for a given \( \eta \). The operate \( O \) input \( F = 0 \rightarrow 1 \) at \( t = 0 \) and \( F = 1 \) until \( t = t_s(\eta) \). During the reverse time integration of (11) \( v(t) \) is not required so during this time \( F = 0 \). Block II generates \( u = sgn v \) for \( t = 0 \) to \( t = t_s \) and \( u = 0 \) as \( t \) goes backward from \( t = t_s \) to \( t = 0 \). The logic signal \( E \), which is 1 for forward time and 0 for reverse time, exercises the desired control. When \( E = 0 \) both AND gates have 0 output causing both switches in II to be open. On the other hand, when \( E = 1 \), either one switch or the other is closed depending on the sign of \( v(t) \). Block III solves (11), a typical integrator being shown in Figure 5. When \( G = 0 \rightarrow 1 \) the integrators begin to operate. From \( t = 0 \) to \( t = t_s, E = 1 \) producing forward time. At \( t = t_s, E = 1 \rightarrow 0 \) and the integrators begin to move backwards. When \( t = 0 \) again, the hold signal \( H = 0 \rightarrow 1 \). Thus \( \gamma(\eta) \) is held at the integrator outputs as long as \( G = H = 1 \).

The control program for generating \( F, E, G, \) and \( H \) is shown in Figure 6. From what has been said it is clear that \( E = F \). The desired sequencing of \( E, G, \) and \( H \) is shown in the timing diagram. The signals \( E \) and \( I \) (never 1 at the same time) are the inputs to the control program. When \( I = 1 \) the integrators in Figure 5 are reset in preparation for a run. The 0 stored in the flip-flop maintains this condition even after \( I = 1 \rightarrow 0 \). To produce a run, \( E = 0 \rightarrow 1 \) and remains at 1 until \( t = t_s(\eta) \), when \( E = 1 \rightarrow 0 \) and stays at 0 until the next run. When \( E = 0 \rightarrow 1 \) the flip-flop is set and \( G = 1 \) until the flip-flop is cleared by \( I \) for the next run. The integrator, comparator, and AND gate produce \( H \). The integrator output is positive during forward time \( (t) \) and reverse time \( (t') \) until \( t' = 0 \) is reached, when the comparator causes \( H = 0 \rightarrow 1 \). The inputs \( E \) and \( G \) to the AND gate prevent the possibility of \( H = 1 \) during reset or the beginning of forward \( t \)
when the comparator input is within the comparator dead space. The input to \( L \) on the comparator locks \( H = 1 \) until \( I = 0 \rightarrow 1 \) causes \( G = 1 \rightarrow 0 \). This assures \( H = 1 \) even when the integrator is held so long that it drifts out of the comparator dead space. \( H \) is an important logic output because it indicates when \( \gamma(\eta) \) is available.

The determination of \( t = t_s \) is made by a comparator with input \( f(t, \eta) \) and output \( J \). The programming is straightforward and is shown in block I of Figure 7. A switch and comparator connected to \( v(t) \) generate \( |v(t)| \). Since \( v(t) \) is obtained from the adjoint equations the integrator for \( f(t, \eta) \) is controlled by \( F \). \( J \) is 0 until \( t = t_s(\eta) \).

The rest of Figure 7 shows the programming of \( (13) \) and master control of the iterative cycle. On the completion of a run \( \eta' \) is updated. This computation is shown in block II. Before the computer begins the first run \( S = 0 \). This establishes the initial condition on the memory units, \( \eta' \). At the end of the first run \( S = 1 \) and \( H = 0 \rightarrow 1 \) which causes \( \eta' \) to replace \( \eta' \). The inputs \( E \) and \( I \) for Figure 6 are derived from the logic elements in Figure 7. When \( S = 0, I = 0 \) and the \( \gamma \) program is reset. When \( S = 0 \rightarrow 1 \) the first run begins. Since \( f(0, \eta'') > 0, J = 0 \) and \( E = 0 \rightarrow 1 \). Thus the \( \gamma \) program begins. When \( t = t_s(\eta'') = t_o, E = 1 \rightarrow 0 \). The \( L \) input on the comparator holds \( E \) at 0 even though the \( f \) integrator is reset by \( F = E \). At \( 2t_o, H = 0 \rightarrow 1 \), indicating that \( \gamma(\eta) \) is available, triggers the memory units. At \( 2t_o + \Delta \) the single shot causes \( I = 0 \rightarrow 1 \). This resets the \( \gamma \) program causing \( H \) and \( J \) to return to 0. The delay \( \Delta \) guarantees that \( H = 1 \) long enough to trigger the single shot. At \( 2t_o + \Delta + 1 \) the single shot returns \( I \) to 0 beginning the second run.

The choice of \( \rho \) is important. If it is too large \( t_i^{+1} \) may be smaller than \( t_i \), indicating that an overstep in the correction of \( \eta' \) has been made. Conversely, if \( \rho \) is too small the convergence of the \( \eta' \) to \( \eta^* \) may be very slow. It would be helpful in the selection of \( \rho \) to have an overstep indicator in the computer program.

Figure 8 shows such an indicator. The memory unit is triggered by \( J \) and therefore stores the value of \( t_o \) on the previous run. If during the present run \( t_i \) is larger, the comparator produces a 1 output at the end of the forward \( t \) portion of the run. The first \( \text{AND} \) gate with input \( E \) assures that this comparison is examined only during forward \( t \). The first \( \text{flip-flop} \) is cleared by \( I \) at the beginning of each run. If an overstep has not been made it will be set during the run by the comparator. Clearly, if the first \( \text{flip-flop} \) is set, the second \( \text{flip-flop} \) will remain cleared since \( H \) cannot pass through the \( \text{AND} \) gate. If however there is an overstep, the second \( \text{flip-flop} \) will be set by \( H \) and \( K = 0 \rightarrow 1 \) thus indicating an overstep.

An equipment count in Figure 5 through Figure 8 yields the following requirements:

- Integrators \( 2n + 2 \)
- Switches \( n + 4 \) (2 single throw)
- Comparitors \( 5 \)
- Track-transfers \( n + 1 \)
- \( \text{AND} \) Gates \( 9 \)
- \( \text{Flip-flops} \) \( 3 \)
- \( \text{Single-Shots} \) \( 1 \)

**CONCLUSIONS**

A hybrid computer of the type described appears to be well suited for the solution of certain optimal control problems. The required flexible control of analog elements is obtained easily with a reasonable number of logic elements. There seems to be no real obstacle in extending the techniques outlined above to more complex control problems and iterative cycles, and eventually, to on-line control computation for actual plants.
ACKNOWLEDGMENT

The author thanks Edward O. Gilbert and Edward J. Fadden for helpful discussions on hybrid computing.

REFERENCES


Figure 6. Control Program for Figure 5.

Figure 7. Iterative Solution of the Optimum Regulator Problem.
Figure 8. Program for Overstep Determination,