NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

KALMAN FILTER REAL TIME TRACKING USING
MICROPROCESSORS AND A NEW LANGUAGE
FOR MILITARY SOFTWARE

by

Jose Alberto Fernandes Ferreira

March 1986

Thesis Advisor: H. A. Titus

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
DECLASSIFICATION / DOWNGRADING SCHEDULE	
ERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION aval Postgraduage School	6b. OFFICE SYMBOL (If applicable) 62
7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
ADDRESS (City, State, and ZIP Code) onterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
NAME OF FUNDING / SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

TITLE (Include Security Classification)
LMAN FILTER REAL TIME TRACKING USING MICROPROCESSORS AND A NEW LANGUAGE FOR MILITARY FTWARE

PERSONAL AUTHOR(S)
Jose Alberto Fernandes Ferreira

TYPE OF REPORT	13b. TIME COVERED FROM ____ TO ____	14 DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
as ter's Thesis		1986 March 27	244

SUPPLEMENTARY NOTATION

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Kalman Filtering, Torpedo Tracking

ABSTRACT (Continue on reverse if necessary and identify by block number)

A sequential Extended Kalman filter routine, implemented in the "C" nguage, is developed for an embedded computer application. The design kes into account the real time constraints imposed by the sampling time the data. Filter measurements consist of four (4) transit times gen- ated by torpedo instrumentation in a TEST RANGE facility, and they are nlinear functions of the coordinates of the torpedo. Memory, speed ficiency, and portability are emphasized in the software design.

The program was tested, using simulated trajectories, in an IBM-PC d in a CIE 680/30 microcomputers. Limitations in handling small floating

DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
NAME OF RESPONSIBLE INDIVIDUAL H. A. Titus	22b. TELEPHONE (Include Area Code) 408 - 646-2560
	22c. OFFICE SYMBOL 62Ts

FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. ABSTRACT (cont'd)
point numbers degraded the performance of the algorithm in the CIE machine which could not track 1/4-G maneuvers.

Approved for public release; distribution is unlimited.

Kalman Filter Real Time Tracking Using Microprocessors and a New Language for Military Software.

by

Jose Alberto Fernandes Ferreira
Captain, Brazilian Air Force
B.S., Instituto Tecnologico de Aeronautica, 1977

Submitted in partial fulfillment of the requirements for the degree(s) of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
and
ELECTRICAL ENGINEER

ABSTRACT

A sequential Extended Kalman filter routine, implemented in the "C" language, is developed for an embedded computer application. The design takes into account the real time constraints imposed by the sampling time of the data. Filter measurements consist of four (4) transit times generated by torpedo instrumentation in a TEST RANGE facility, and they are nonlinear functions of the coordinates of the torpedo. Memory, speed efficiency, and portability are emphasized in the software design.

The program was tested, using simulated trajectories, in an IBM-PC and in a CIE 680/30 microcomputers. Limitations in handling small floating point numbers degraded the performance of the algorithm in the CIE machine which could not track 1/4-G maneuvers.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I am deeply indebted to my wife, Anne, for her love, patience and understanding during the preparation of this work. I would also like to thank Professor Hal Titus for his professional guidance. I would also like to express my gratitude to Professor Alex Gerba, Mr. Robert Limes and Mr. Albert Wong for their help and sugestions.

I. INTRODUCTION

The problems of torpedo tracking have been studied, at the Naval Postgraduate School, by several researchers, who have implemented simulations that would allow analysis of tracking algorithm performance using the Extended Kalman Filter [1]. However, those studies have been limited to simulations done on large mainframe computers, without trying to adapt them to the real time processing system at the torpedo range.

The objective here is to develop a version of the Extended Kalman Filter, adapted to the constraints of real-time processing, by generating a program in the C language. Also it is designed to run in the microcomputer C.I.E 680/30, currently utilized at the Naval Torpedo Station, Washington.

The presentation herein tries to follow as much as possible the chronological order, describing the problems in the order of appearance and the way they were solved.

II. PROBLEM DEFINITION

A. RANGE TRACKING GEOMETRY

Sets of hydrophone arrays are distributed in line along the extension of the test range. Each individual array consists of four independent elements, placed at a distance "d" apart, on the vertices of a cube, thus defining an orthogonal coordinate system in which the measurement of the transit times are made. Figure 1, shows the geometry.

Four transit times, T_c, T_x, T_y, T_z, corresponding to the four hydrophones in the array are measured and the positional information is computed from the measured transit times. The torpedos are equipped with sonar transmitters which are synchronized with a master clock, allowing new data every 1.31 seconds, with a range accuracy between 3 to 30 feet.

When multiple arrays are used, the signal from the closest hydrophone is defined as the basis for the time measurements and for the range calculations. If the computer time were not critical, an 18 hydrophone signal should be processed.

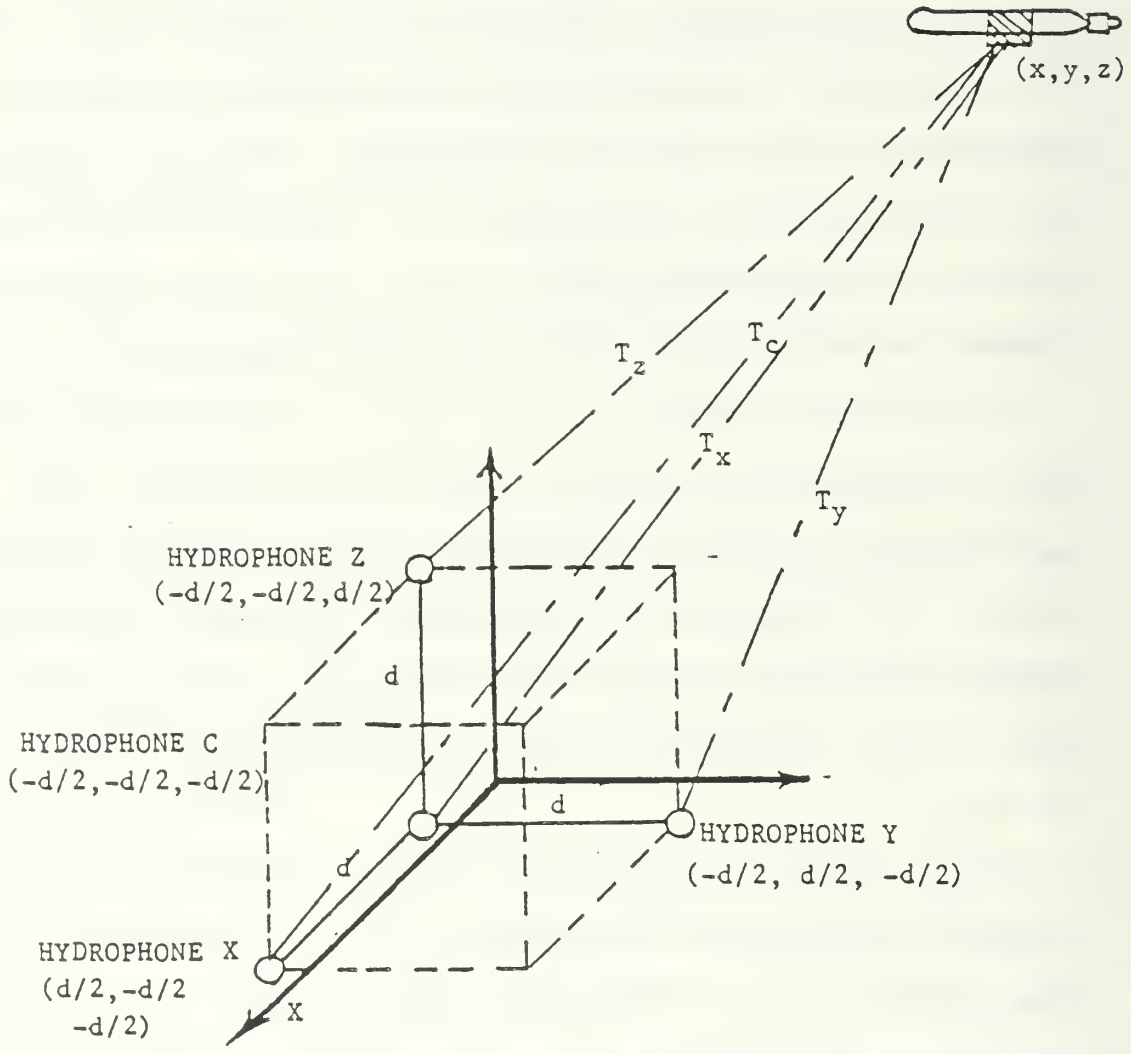


Figure 1: Geometry of a tracking array.

B. GENERAL EXTENDED KALMAN FILTER EQUATIONS

A summary of the filter and associated equations is given in this section, following the order and the conventions in [1].

1. Observation Equations

For the three-dimensional problem, in which the z-position is considered invariant, a state space consisting of the three positions, X, Y and Z, and the two velocities, V_x and V_y define the torpedo motion. Thus, the discrete linear estimation and the nonlinear observation equations are given by

$$\underline{X}(k+1) = \Phi \cdot \underline{X}(k) + \Gamma \cdot \underline{W}(k) \qquad (2.1)$$

$$\underline{Z}(k) = \underline{h}(\underline{X}(k),k) + \underline{V}(k) \qquad (2.2)$$

where: Φ and Γ are constant matrices;

$\underline{h}$ is a nonlinear function of the state $\underline{X}$;

$\underline{W}(k)$ is the plant excitation noise;

$\underline{V}(k)$ is the measurement noise.

The plant noise and the measurement noise are assumed Gaussian distributed with zero mean, and uncorrelated.

The last assumption implies that

$$E[\underline{W}(k).\underline{W}^T(j)] = \underline{Q}'(k)\ \delta_{kj}$$

$$E[\underline{V}(k).\underline{V}^T(j)] = \underline{R}(k)\ \delta_{kj}$$

where: $\delta = 1$, for $k = j$;

$\delta = 0$, for $k \neq j$;

2. Linearized Equations

Linearization of the observation equation is obtained by using the Taylor Series to expand about the prediction of the trajectory for the interval.

$$\underline{Z}(k) = \underline{H}(k) \cdot \underline{X}(k) + \underline{V}(k) \qquad (2.3a)$$

where: $$\underline{H}(k) = \frac{\partial \underline{h}}{\partial \underline{X}} \Big|_{\underline{X}(k) = \hat{\underline{X}}(k/k-1)} \qquad (2.3b)$$

$\hat{\underline{X}}(k/k-1)$ is the estimated value (predicted) at time k, given the measurements until time k-1.

3. State Error Vector

$$\tilde{\underline{X}}(k) = \hat{\underline{X}}(k) - \underline{X}(k)$$

4. Predicted State Error Vector

$$\tilde{\underline{X}}(k/k-1) = \hat{\underline{X}}(k/k-1) - \underline{X}(k)$$

5. Covariance of State Error

$$\underline{P}(k) = E[\tilde{\underline{X}}(k/k) . \tilde{\underline{X}}^T(k/k)]$$

6. Predicted Covariance

$$\underline{P}(k/k-1) = E[\tilde{\underline{X}}(k/k-1) . \tilde{\underline{X}}^T(k/k-1)]$$

7. State Excitation Matrix

$$\underline{Q}(k) = \Gamma(k) . E[\underline{W}(k) . \underline{W}^T(k)] . \Gamma^T(k)$$

8. Measurement Noise Covariance Matrix

$$\underline{R}(k) = E[\underline{V}(k) . \underline{V}^T(k)]$$

9. Kalman Filter Equations

$$\underline{P}(k+1/k)=\underline{\Phi}.\underline{P}(k/k).\underline{\Phi}^{T}+\underline{Q}(k) \quad (2.4a)$$

$$\underline{G}(k)=\underline{P}(k/k-1)\underline{H}^{T}(k)[\underline{H}(k)\underline{P}(k/k-1)\underline{H}^{T}(k)+\underline{R}(k)]^{-1} \quad (2.4b)$$

$$P(k)=[\underline{I}-\underline{G}(k).\underline{H}(k)].\underline{P}(k/k-1) \quad (2.4c)$$

$$\hat{\underline{X}}(k+1/k)=\underline{\Phi}.\hat{\underline{X}}(k/k) \quad (2.4d)$$

$$\hat{\underline{Z}}(k/k-1)=\underline{h}(\hat{\underline{X}}(k/k-1),k) \quad (2.4e)$$

$$\hat{\underline{X}}(k/k)=\hat{\underline{X}}(k/k-1)+\underline{G}(k)[\underline{Z}(k)-\underline{Z}(k/k-1)] \quad (2.4f)$$

C. APPLIED EQUATIONS [1]

In order to apply the Extended Kalman filter to the torpedo tracking problem it is necessary to define the state vector and the matrices $\underline{H}$, $\underline{\Phi}$, Γ, $\underline{W}$, $\underline{Q}$, and $\underline{R}$.

1. Transit Times

$$T_c(k)=\frac{1}{VEL}[(x+d/2)^2+(y+d/2)^2+(z+d/2)^2]^{1/2}+v(k) \quad (2.5)$$

$$T_x(k)=\frac{1}{VEL}[(x-d/2)^2+(y+d/2)^2+(z+d/2)^2]^{1/2}+v(k) \quad (2.6)$$

$$T_y(k)=\frac{1}{VEL}[(x+d/2)^2+(y-d/2)^2+(z+d/2)^2]^{1/2}+v(k) \quad (2.7)$$

$$T_z(k)=\frac{1}{VEL}[(x+d/2)^2+(y+d/2)^2+(z-d/2)^2]^{1/2}+v(k) \quad (2.8)$$

where: d=30 feet, is the distance between two hydrophones in a given array;

VEL=4860 ft/sec, is the average sound speed at Dabob Bay.

These equations must be linearized to apply Kalman

Filter Theory. The linearization is done, in this case, about the predicted trajectory, which is being generated by the filter algorithm.

The values of transit times are corrupted by noise, $v(k)$, which is assumed Gaussian with zero mean. The positions x, y, and z are all functions of the time interval.

2. State Equations

The plant state equations are defined by

$$\begin{bmatrix} x(k+1) \\ V_x(k+1) \\ y(k+1) \\ V_y(k+1) \\ z(k+1) \end{bmatrix} = \begin{bmatrix} x(k)+TV_x(k)+g_1 \\ V_x(k)+g_2 \\ y(k)+TV_y(k)+g_3 \\ V_y(k)+g_4 \\ z(k)+g_5 \end{bmatrix} \qquad (2.9)$$

where: $x(k)$, $y(k)$ and $z(k)$ are the position coordinates at time $t(k)$;

$V_x(k)$ and $V_y(k)$ are the x and y components of the velocity;

the depth $z(k)$ is maintained constant and any velocity in z is assumed random, Gaussian, uncorrelated and with zero mean;

T is the sampling interval T=1.31sec;

g_1, g_2, g_3 and g_4 are excitation terms, included to take into account random variations in speed (γ_{V_t}),

heading (γ_{θ_t}), and depth (γ_z), which are assumed to be independent, zero mean, rates of change.

Typical torpedo maneuvering parameters are

$\sigma_{\dot{\theta}_t} = 22°/\text{sec}$; $\sigma_{v_t}^2 = E[\gamma_{v_t}^2]$

$\sigma_{v_t} = 36\ \text{ft/sec}^2$; $\sigma_{\dot{\theta}_t}^2 = E[\gamma_{\theta_t}^2]$

$\sigma_{\dot{z}} = 1\ \text{ft/sec}$; $\sigma_{\dot{z}}^2 = E[\gamma_z^2]$

In state form :

$$\underline{X}(k+1) = \underline{\Phi}.\underline{X}(k) + \Gamma.\underline{W}(k) \qquad (2.10)$$

$$\underline{\Phi} = \begin{bmatrix} 1 & T & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & T & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} T^2/2 & 0 & 0 \\ T & 0 & 0 \\ 0 & T^2/2 & 0 \\ 0 & T & 0 \\ 0 & 0 & T \end{bmatrix} \qquad (2.11)$$

$$\sigma_{\dot{x}}^2 = (V_x/V_t)\sigma_{v_t}^2 + V_y^2\sigma_{\dot{\theta}}^2 \qquad (2.12a)$$

$$\sigma_{\dot{y}}^2 = (V_y/V_t)\,\sigma_{V_t}^2 + V_y^2\,\sigma_{\dot{\theta}_t}^2 \qquad (2.12b)$$

$$\sigma_{\dot{x}\cdot\dot{y}\cdot} = V_x V_y\left[\sigma_{V_t}/V_t^2 - \sigma_{\dot{\theta}}\right] \qquad (2.12c)$$

$\underline{Q}$ is a symmetric matrix given by

$$\underline{Q} = \begin{bmatrix} q0 & q1 & q2 & q3 & q4 \\ q5 & q6 & q7 & q8 & q9 \\ q10 & q11 & q12 & q13 & q14 \\ q15 & q16 & q17 & q18 & q19 \\ q20 & q21 & q22 & q23 & q24 \end{bmatrix} \qquad (2.13)$$

where: $q0=(T^2/2)\,\sigma_{\dot{x}}^2$; $q1=(T^3/2)\,\sigma_{\dot{x}}^2$; $q2=(T^2/2)^2\,\sigma_{\dot{x}\cdot\dot{y}\cdot}$;

$q3=(T^3/2)\,\sigma_{\dot{x}\cdot}\sigma_{\dot{y}\cdot}$; $q4=0$; $q6=T^2\,\sigma_{\dot{x}}^2$;

$q7=(T^3/2)\,\sigma_{\dot{x}\cdot\dot{y}\cdot}$; $q8=T^2\,\sigma_{\dot{x}\cdot\dot{y}\cdot}$; $q9=0$;

$q12=(T^2/2)^2\,\sigma_{\dot{y}}^2$; $q13=(T^3/2)\,\sigma_{\dot{y}}^2$; $q14=0$;

$q18=T^2\,\sigma_{\dot{y}}^2$; $q19=0$; $q24=T^2\,\sigma_{\dot{z}}^2$;

$$\underline{Z}(k) = [T_c(k) \quad T_x(k) \quad T_y(k) \quad T_z(k)]^T$$

$$\underline{R}(k) = \begin{bmatrix} \sigma_{T_c} & 0 & 0 & 0 \\ 0 & \sigma_{T_x}^2 & 0 & 0 \\ 0 & 0 & \sigma_{T_y}^2 & 0 \\ 0 & 0 & 0 & \sigma_{T_z}^2 \end{bmatrix}$$

Typically $\sigma_{T_c}^2 = \sigma_{T_x}^2 = \sigma_{T_y}^2 = \sigma_{T_z}^2 = 10^{-10}$ sec.

$$H(k) = \begin{bmatrix} h0 & h1 & h2 & h3 & h4 \\ h5 & h6 & h7 & h8 & h9 \\ h10 & h11 & h12 & h13 & h14 \\ h15 & h16 & h17 & h18 & h19 \\ h20 & h21 & h22 & h23 & h24 \end{bmatrix}$$

where: $h0=(x'(k)+d/2)/DEN1$; $h1=0$;

$h2=(y'(k)+d/2)/DEN1$; $h3=0$;

$h4=(z'(k)+d/2)/DEN1$; $h5=(x'(k)-d/2)/DEN2$;

$h6=0$; $h7=(y'(k)+d/2)/DEN2$;

$h8=0$; $h9=(z'(k)+d/2)/DEN2$;

$h10=(x'(k)+d/2)/DEN3$; $h11=0$;

$h12=(y'(k)-d/2)/DEN3$; $h13=0$;

$h14=(z'(k)+d/2)/DEN3$; $h15=(x'(k)+d/2)/DEN4$;

$h16=0$; $h17=(y'(k)+d/2)/DEN4$;

$h18=0$; $h19=(z'(k)-d/2)/DEN4$;

and $DEN1=[(x'(k)+d/2)^2+(y'(k)+d/2)^2+(z'(k)+d/2)^2]^{1/2}$

$DEN2=[(x'(k)-d/2)^2+(y'(k)+d/2)^2+(z'(k)+d/2)^2]^{1/2}$

$DEN3=[(x'(k)+d/2)^2+(y'(k)-d/2)^2+(z'(k)+d/2)^2]^{1/2}$

$DEN4=[(x'(k)+d/2)^2+(y'(k)+d/2)^2+(z'(k)-d/2)^2]^{1/2}$

D. THE SEQUENTIAL EXTENDED KALMAN FILTER APPROACH [1]

This approach assumes that all four transit times measurements are statistically independent of each other. In this way one can process the transit times sequentially and independently. The result of each computation step is used immediately, as soon as it is obtained. Thus more accurate estimates of the filter states are obtained.

This procedure also circumvents the matrix inversion when determining the filter gain, and allows one to disregard any particular time measurement that was found invalid, and not the total four times for a time slot (T=1.31sec).

Erroneous transit times are detected by implementing a three sigma gate using the covariance of the measurement noise (R) and the covariance of the estimation error (P(k/k). For i=1 to 4 ,

$$gate(i) = 3*(\sqrt{(P_{jj}\,maximum)/4860 + R_{ii}}).$$

It can also be used to minimize position errors during high dynamic maneuvers, by repeating the calculations for the same time interval until the transit time residual error falls below a preset value.

III. THE IMPLEMENTATION PROBLEMS

A. THE HARDWARE AND THE SOFTWARE PROBLEMS

Since the objective is to use the developed software for torpedo tracking at the NAVTORPSTA, in Keyport, Washington, there was no other choice but to implement the program to run on the C.I.E. 680/30 microcomputer, and to implement it in the C language.

Two problems immediately arose.

First, there was only badly written and incomplete documentation. The manual was found difficult to read and to understand. In addition, there was no information about the computer hardware.

Secondly, poor software was available. The CIE 680/30 is a Unix based machine, designed for business applications, working with the Regulus operating system. As far as known, there is no software available (specifically written for the CIE) on the market.

As a result of these problems, there was no full-screen editor available for the C.I.E machine, no communications program, and no built in graphics capability (although the first screen that appears after booting the system shows the message "CIE GRAPHICS SYSTEM #7").

B. THE FAST APPROACH

In order to be able to write a reasonably long and complex program, in a reasonable time, debug it and make it work, it would be necessary to have a good compiler, a good full screen editor and good documentation on software and hardware.

It was then decided to make use of the known software and hardware information available for the IBM-PC microcomputer. It was not difficult to find Public Domain Programs able to communicate with Unix based machines, to emulate the VT-100 terminals, and to transfer files. Also it was not difficult to find a powerful easy to use full-screen editor for the IBM-PC.

The program was then written, trying to keep the C code the most standard possible, using the Lattice C compiler. It was executed first in the IBM-Pc, then transferred (source code) to the C.I.E using a public domain program (SIMTERM). It was then adapted, making minor changes, and recompiled on the C.I.E to generate the executable code.

C. ODDS AND EVENS OF C

The C language is not very high level but it is a very powerful language, allowing bit operations, low level read and write, system calls and easy interfaces with machine language. The general mathematical expressions can be kept

the same way as in Fortran, but some new operations, functions and expressions are also used.

Incrementing a variable:

"i=i+1;" can be "i++" or "++i", depending if you first use the variable (++i) and increment it or vice versa (i++).

Incrementing i by j: "i+=j;"

Do loop, testing the counting variable after using it:

"for (i=0;i<imax;i++);"

Blocks are delimited by "{ }";

Every program, subprogram, functions or procedures are seen as FUNCTIONS in C, including the main program. This is indicated by the calling statements as in:

main() .... for the main program that is not passing any arguments;

plot(x,y) .... for a routine that passes X and Y vectors for plotting, and so on;

C uses passing by value approach. So the value of the variable "x" is copied in and used inside a function. Arrays are equivalent to pointers and thus they can return new values from functions.

There are automatic, local, global and external variables. There are character, integer, long integer, string, floating point (double and single precision), string, and enumeration data types. Short integers have standard size. Double precision floating point allows one to

represent numbers in the range $\pm 10^{\pm 308}$ in the IBM-PC (Lattice C compiler) and in the range $10^{\pm 38}$ in the CIE. There are IF , IF THEN ELSE, WHILE, DO .. WHILE, and CASE statements.

There are no input and output statements as in Fortran or other languages. Input-output is done by standard functions, which must be carefully used.

Header files and " #include " statements are used to hide from the user definitions of parameters that are machine dependent or contain environment-specific information. "#define" statements are used to define constants and macros and to hide machine dependent code segments.

Reference [4] is the definition manual of the language. The other references for the C language were used primarily to look over the example programs, and functions.

IV. THE REAL-TIME KALMAN FILTER PROGRAM IN C

A. THE INITIAL APPROACH

The first idea was the simple translation of the program described in reference [1] to the C language, without worrying about specific problems that would arise when trying, later on, in a more realistic approach, to simulate as close as possible the actual running conditions.

The initial program had a structure similar to the Fortran program in [1], that is, a main program, the actual Kalman Filter, and several functions to calculate the trajectory, to add, transpose and multiply matrices and to multiply two vectors. All the floating point operations were done in double precision and the matrices were defined as two dimensional arrays.

The initial approach failed in many aspects.

It did not have enough speed, producing, on the average, one estimate every 22 sec, in an IBM-PC without math-coprocessor (8087), running at 5 MHz. The same program generated a new estimate every 5 sec in an IBM-PC with an 8087 chip. It did not have the main frame graphics capability, readily available and, finally, due to the way the C language passes parameters to the subroutines, the matrix routines worked correctly only if matrices were defined as square matrices.

B. OPTIMIZATION OF EXECUTION TIME

Although the C.I.E microcomputer runs a Motorola 68000 cpu at 8 MHz, and the IBM-PC runs at 5 MHz, executing the same program in both machines showed a difference of 10% to 20% in execution time. In this way, optimizing the execution in the IBM-PC (with 8087) would be a valid approach since it would allow a safety margin, and would require almost the same effort.

Optimization of execution time led to :

1. Transforming all the matrices from bidimensional to unidimensional arrays (or vectors). That saved memory space, and forced the access to the elements to conform with the way these are stored (C stores array elements by rows), thus speeding up retrieving when needed.

2. Eliminating functions, actually by having everything in the main program.

3. Eliminating all the unnecessary operations. That involved restructuring the program to avoid loops when possible and reasonable, avoiding multiplying by zero, avoiding adding zeros, and avoiding repeating calculations when the result was known to be constant.

4. Avoiding double precision where possible, and when improvement of the execution time could be achieved without compromising the results. The Lattice C compiler, in the IBM-PC, is defined for precision (all the floating point

calculations are done in double precision). Then, if single precision is desired, you lose in both, precision and speed. The CIE machine calculations uses the Motorola "Fast Floating Point", a format which is not compatible with the IEEE format. The IEEE floating point values lose some precision due to the fact that the calculations use the fast floating point routines and a translation to and from the FFP format [10]. So, the FFP format was used, in the CIE.

5. Writing single precision routines to avoid calling standard mathematical routines, which are all double precision, in C.

6. Reproducing more closely the way the final program would have to work, when in the real time application.

7. Testing the program for several values and parameters, to determine bounds that would allow results of good precision without increasing the execution time. It was found, for example, that the results are highly sensitive to the way the columns of the matrix $\underline{H}$ are calculated, thus, requiring calculation of $\underline{H}$ using double precision.

8. Measuring execution time of tasks inside the program.

9. Developing software tools to allow for interactive execution and checking of correctness of the results.

V. TESTING

A. CHECKING THE TRAJECTORY-FOLLOWING PERFORMANCE

The trajectory following performance was verified for the following trajectories :

1. Straight line parallel to the array axis, and y-coordinate = 0.0 , single array tracking;

2. Straight line parallel to the array axis and y-coordinate = 2121.0 , single array tracking;

3. Straight line, 45^{o} inclination relatively to the array axis, single array;

4. Straight line parallel to the array axis, long run, multiple array tracking;

5. Straight line and circular maneuver, short run, slow turn (10.0dg/sec , 1/4-G), single array tracking;

6. Straight line and circular maneuver, long run, slow turn (10.0dg/sec , 1/4-G), multiple array tracking;

7. Straight line and circular maneuver, short run, fast turn (20dg/sec , 1-G), single array tracking, adaptive maneuver implemented with 4 iterations allowed;

8. Straight line and circular maneuver, long run, fast turn (20 dg/sec , 1-G), multiple array tracking, adaptive maneuver implemented with 4 iterations allowed;

TABLE 1 , and TABLE 2 summarize the results in terms of maximum absolute error for the IBM and the CIE microcomputers, respectively. For the single array cases, only the region inside a 3000 feet radius around the center of the array being used is considered in determining the errors.

The corresponding plots can be found in the following pages. The plots were obtained with the value 10^{-10} for the elements in the diagonal of the matrix $\underline{R}$ (the Error Covariance Matrix) for the IBM-PC microcomputer and with the value 10^{-8} for the CIE. The plots are numbered sequentially and contain headings to identify and relate them to the Table 1 and Table 2. The actual time is obtained by multiplying the time scale by the length of the time slot, 1.31 seconds.

B. CHECKING THE EXECUTION TIME REQUIREMENT

The execution time in the IBM-PC, was verified by checking the printouts containing the values of the internal computer time for a cycle. Later, an "waiting loop" was introduced to generate an interval of 1.31 sec between each data input (that is, reading files TRJ.DAT and ZIC.DAT).

Since the "reading internal time" was not incorporated in the CIE program version (CIEKR1), and since that proved to be a faster machine, the time requirement was just checked against a chronometer for 150 iterations.

TABLE 1

IBM-PC MICROCOMPUTER

RUN #	MAX. ABS. POS. ERROR			TRAJEC TYPE	ARRAY TYPE	INITIAL CONDIT.
	X_ERR	Y_ERR	Z_ERR			
1	0.025	4.5	2	linear	singl	x =3000 y = 0 Vx=-50 Vy= 0
2	4.5	5.4	2.5	linear	singl	x =3000 y =2121 Vx=-33.8 Vy= 0
3	6.0	6.0	2.7	linear	singl	x =3000 y =-3000 Vx=-33.8 Vy= 33.8
4	0.2	5.3	1.8	linear	multp	x =34000 y =6050 Vx=-84.4 Vy= 0
5	1.0	4.3	5.6	circl 10dg/s t_0=40 tf=70	singl	x =3000 y = 0 Vx=-50 Vy= 5
6	17	3.3	2.8	circl 10dg/s t_0=40 tf=70	multp	x =10000 y = 0 Vx=-50 Vy= 5
7	2.4	7	3.7	circl 20dg/s t_0=35 tf=45	singl	x =5000 y = 0 Vx=-92.5 Vy= 0
8	22	5.2	12	circl 20dg/s t_0=120 tf=130	multp	x =20000 y = 0 Vx=-92.5 Vy= 5

TABLE 2

CIE MICROCOMPUTER

RUN #	MAX. ABS. POS. ERROR			TRAJEC TYPE	ARRAY TYPE	INITIAL CONDIT.
	X_ERR	Y_ERR	Z_ERR			
1	0.031	4	1	linear	singl	x =3000 y = 0 Vx=-50 Vy= 0
2	7.1	10	1.2	linear	singl	x =3000 y =2121 Vx=-33.8 Vy= 0
3	3	3	0.9	linear	singl	x =3000 y =-3000 Vx=-33.8 Vy= 33.8
4	0.2	3.5	0.7	linear	multp	x =34000 y =6050 Vx=-84.4 Vy= 0
5	---	---	---	circl 10dg/s t_0=40 tf=70	singl	x =3000 y = 0 Vx=-50 Vy= 5
6	---	---	---	circl 10dg/s t_0=40 tf=70	multp	x =10000 y = 0 Vx=-50 Vy= 5
7	---	---	---	circl 20dg/s t_0=35 tf=45	singl	x =5000 y = 0 Vx=-92.5 Vy= 0
8	---	---	---	circl 20dg/s t_0=120 tf=130	multp	x =20000 y = 0 Vx=-92.5 Vy= 5

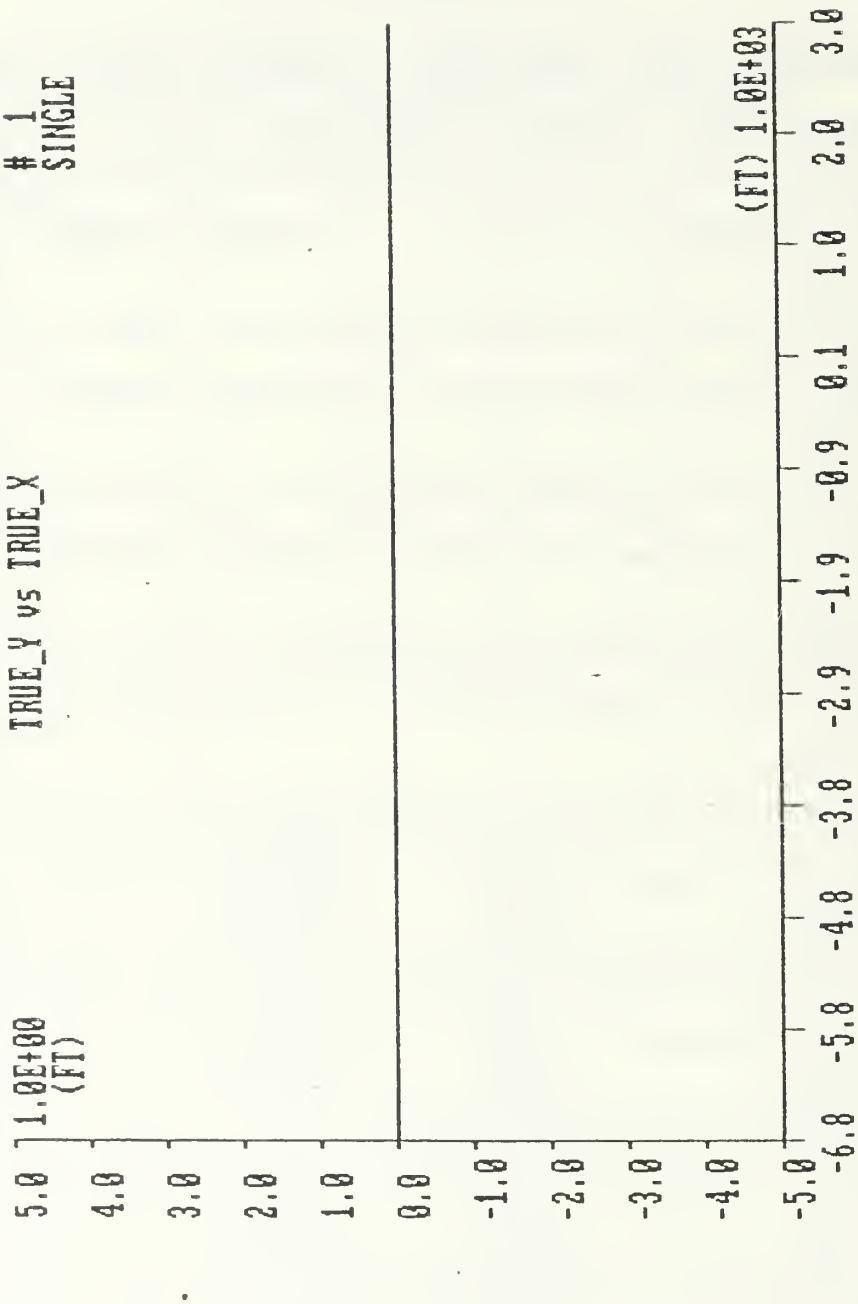


Figure 2. IBM-PC - True trajectory of the torpedo in the area of a single array. Initial conditions : x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

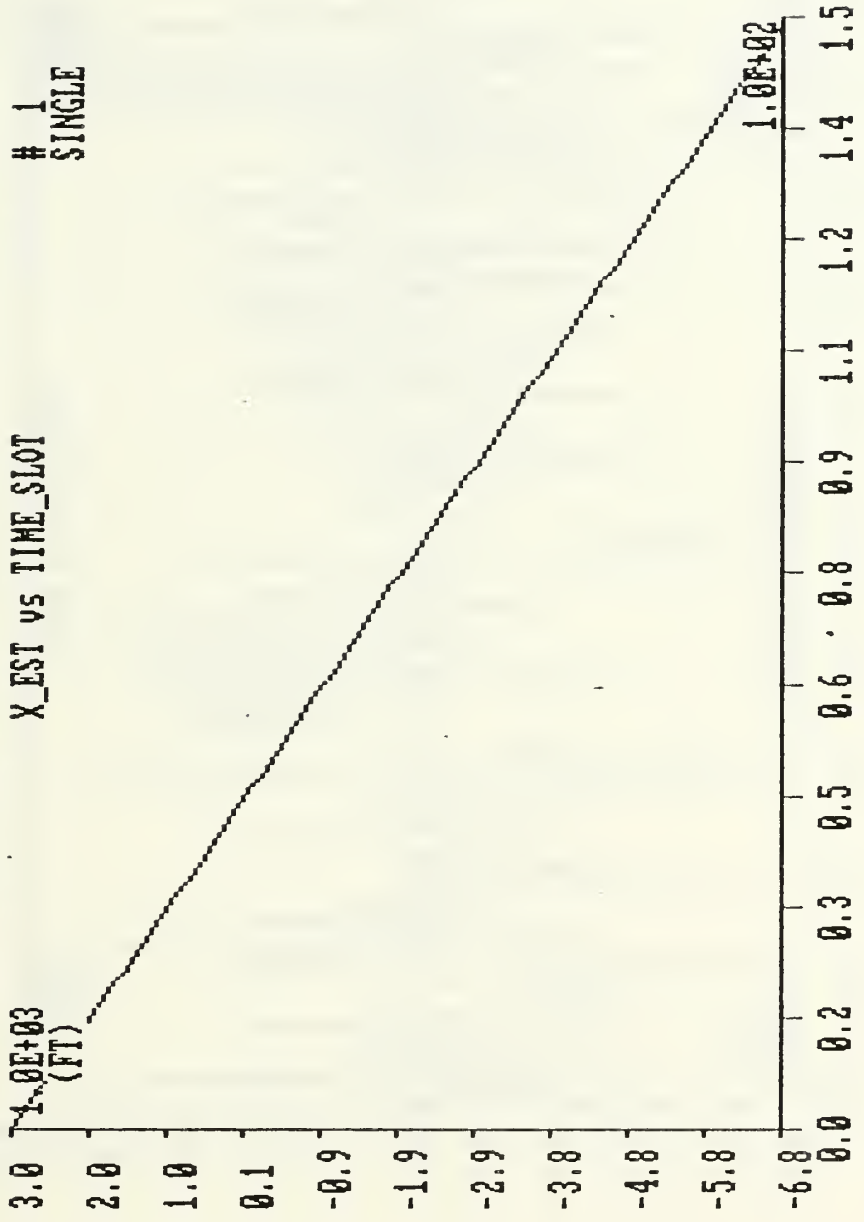


Figure 3. IBM-PC - Estimate of torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

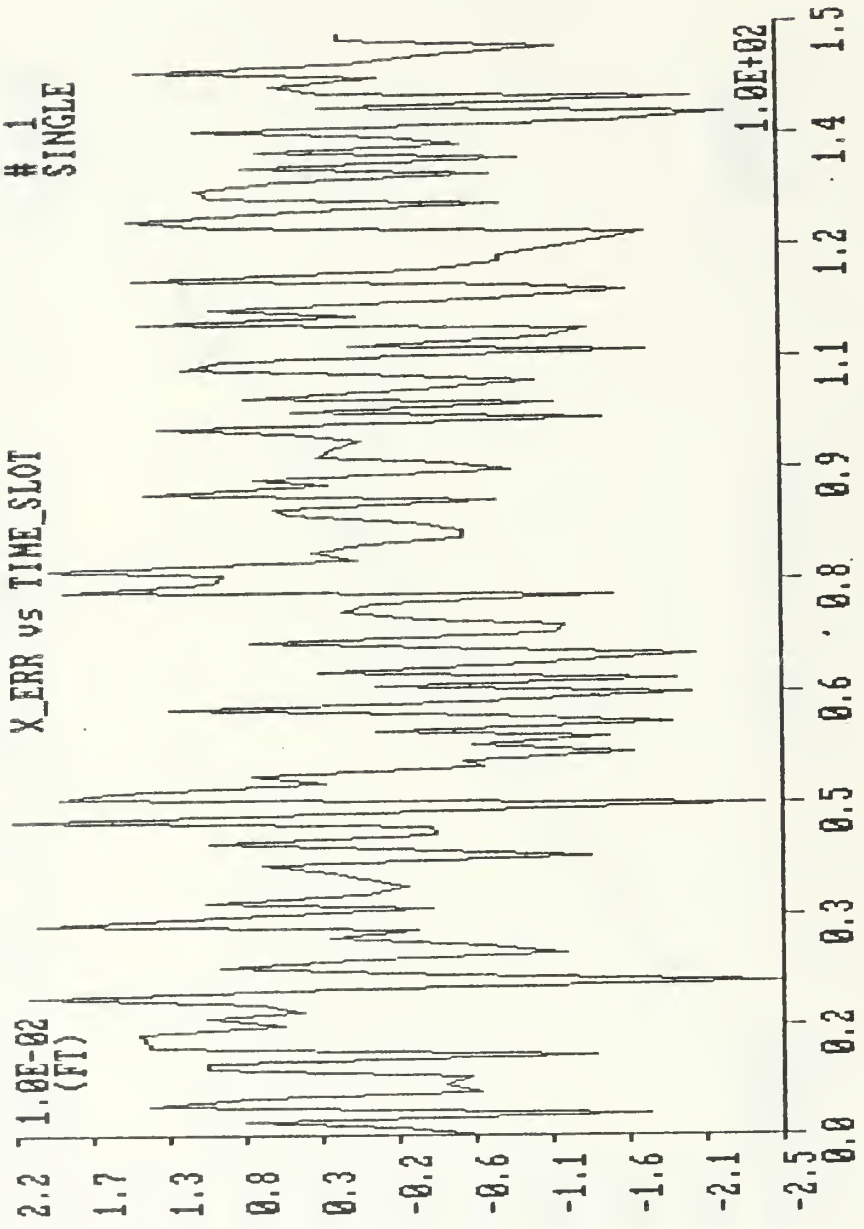


Figure 4. IBM-PC - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

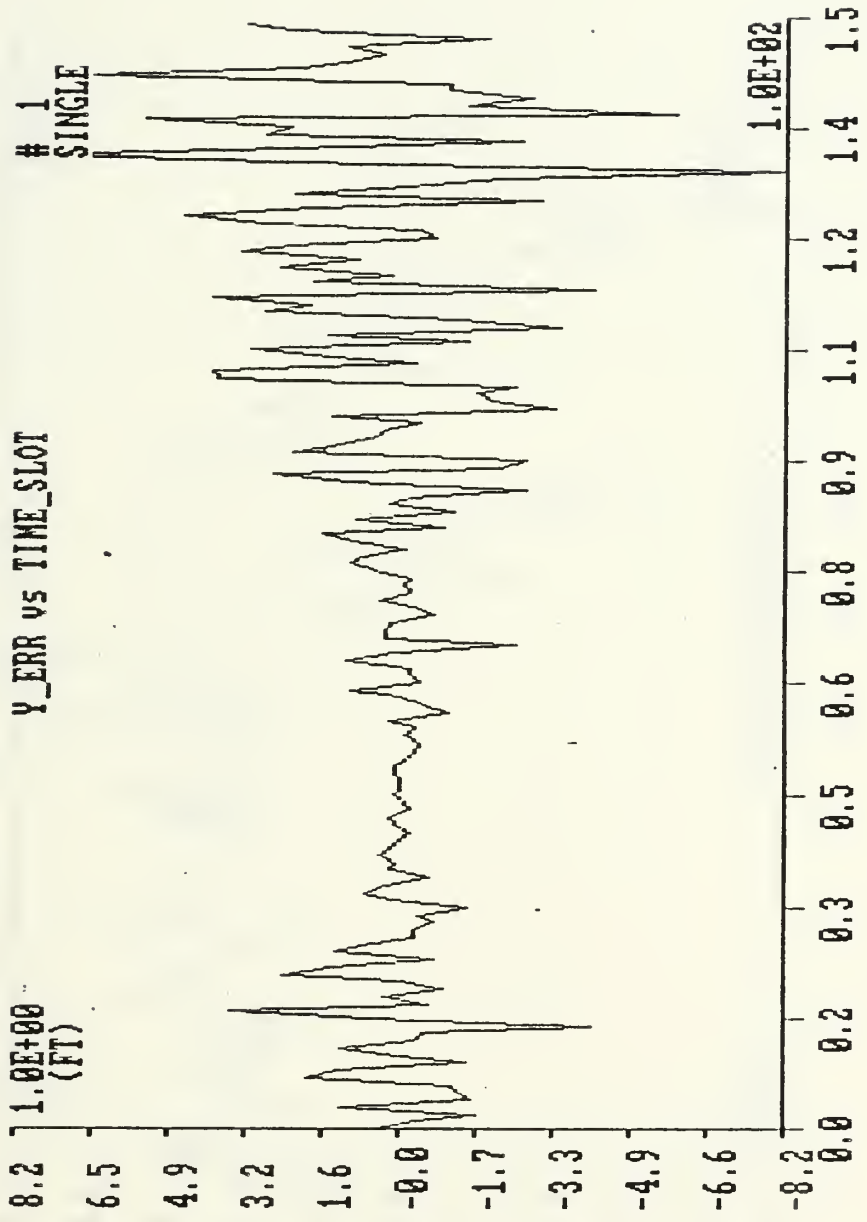


Figure 5. IBM-PC - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

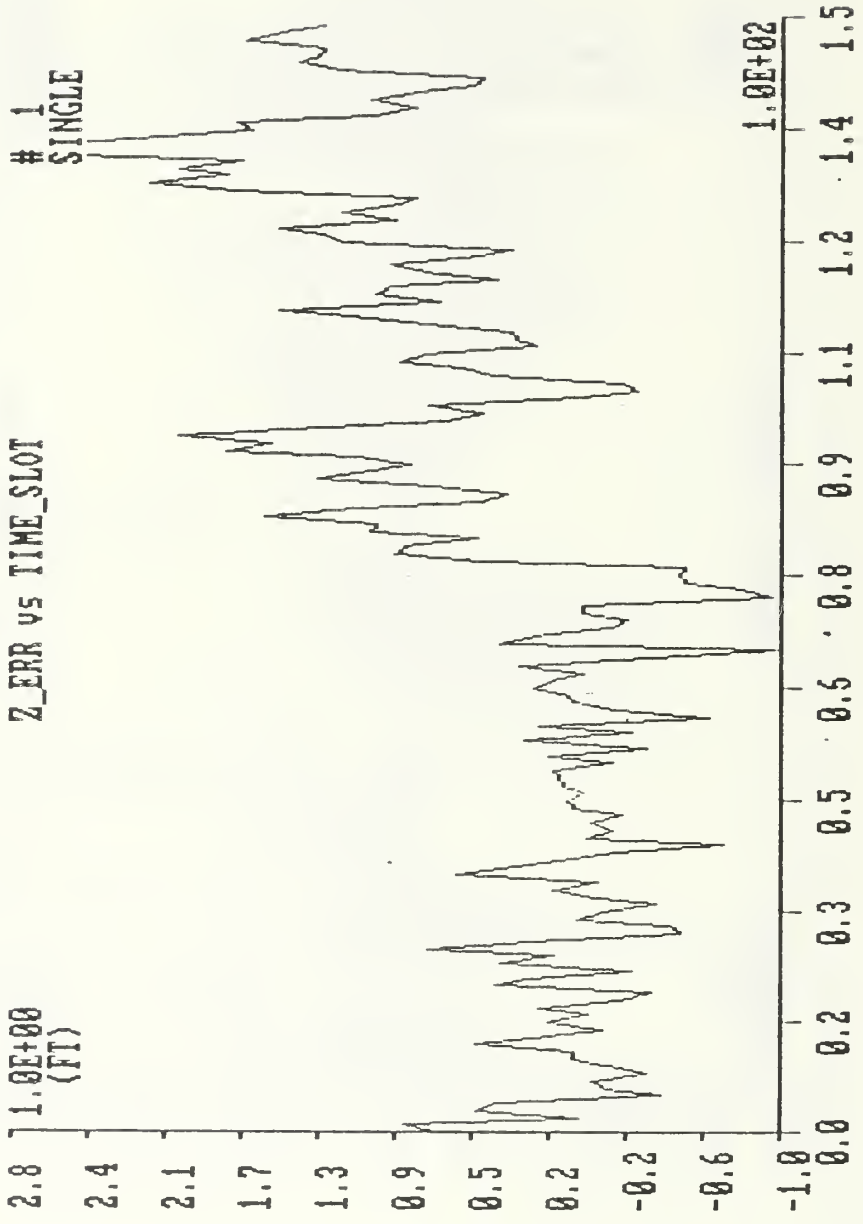


Figure 6. IBM-PC - Error in torpedo depth during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

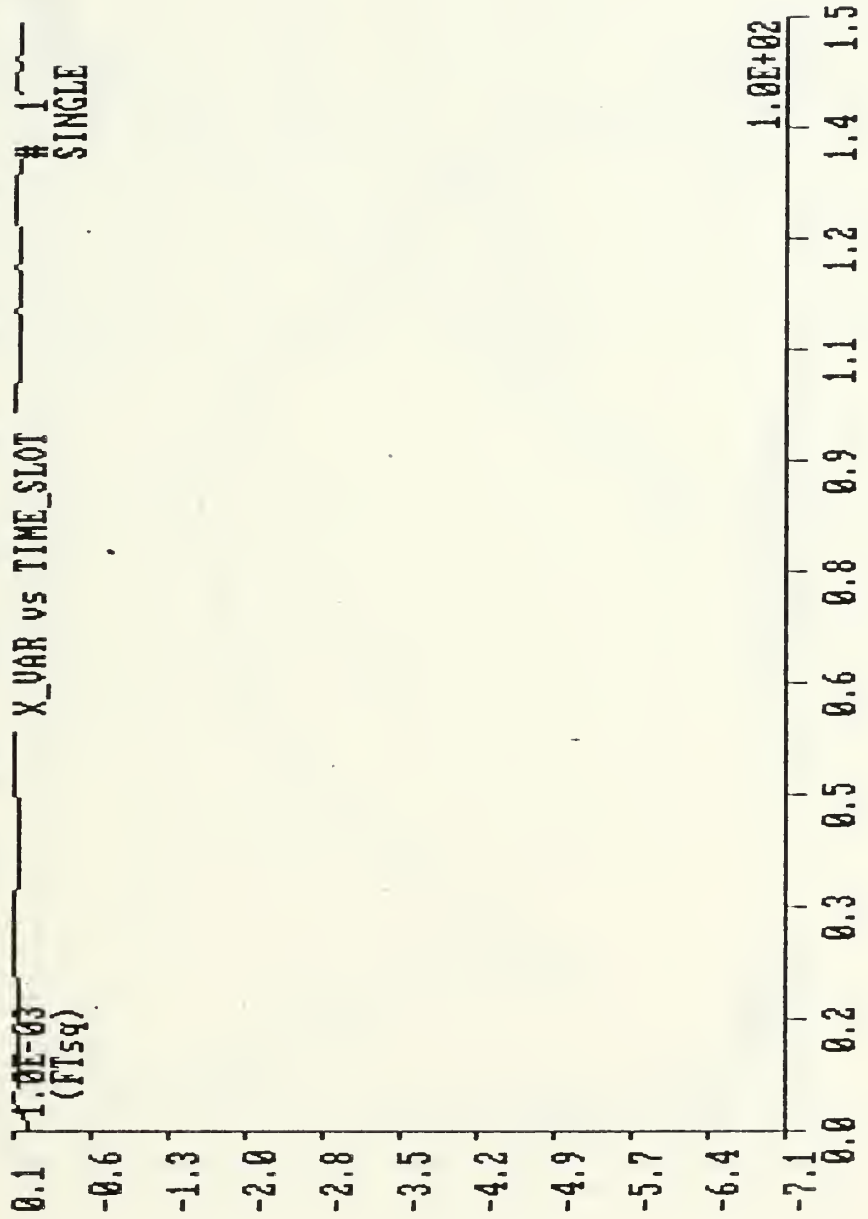


Figure 7. IBM-PC - Variance of position error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

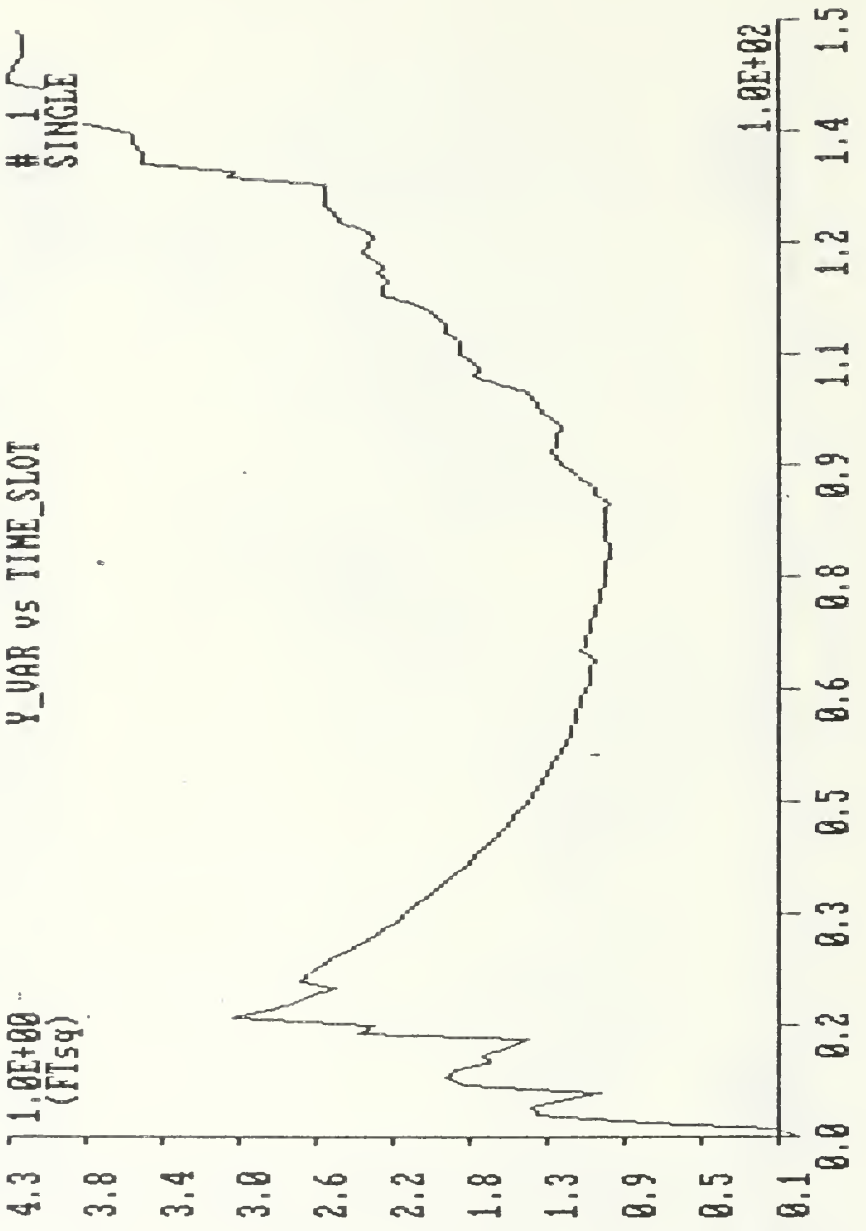


Figure 8. IBM-PC - Variance of position error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

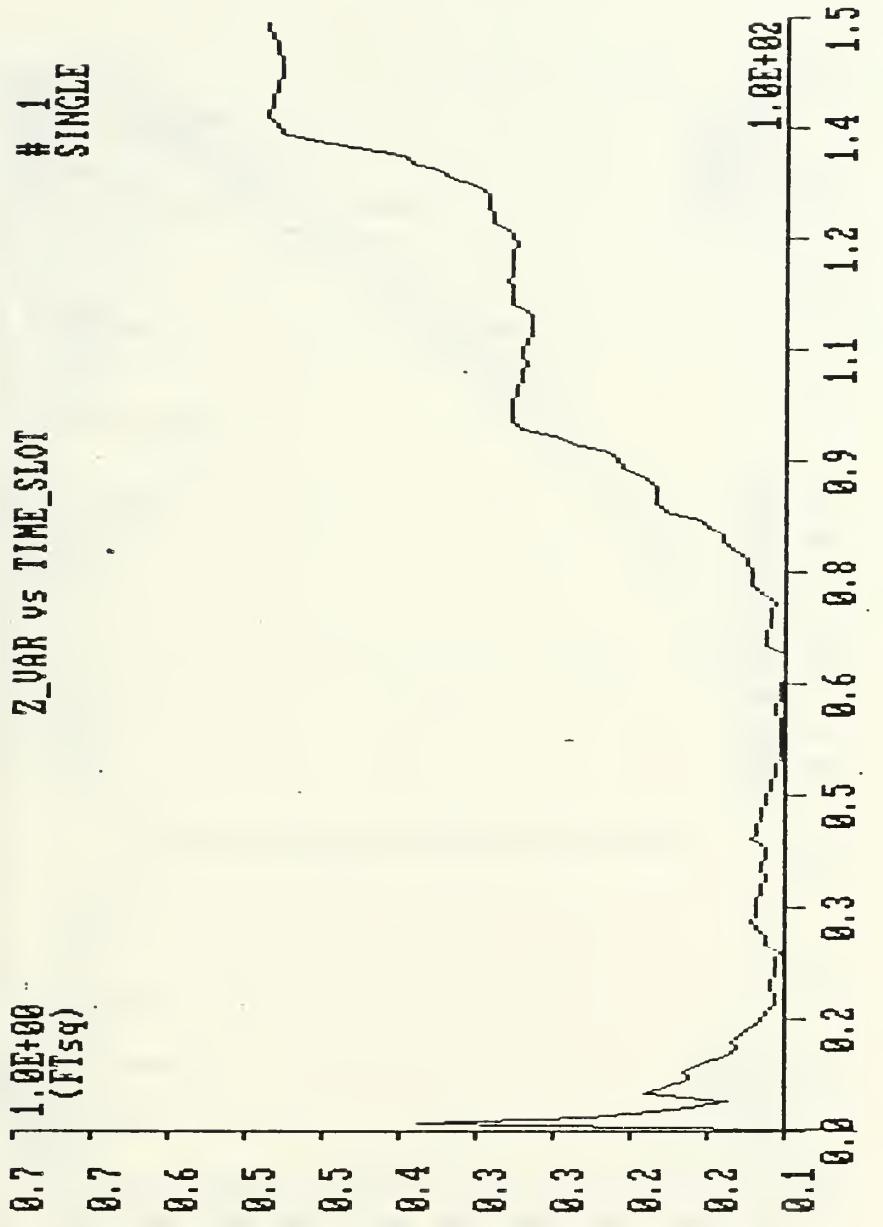


Figure 9. IBM-PC - Variance of depth error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

Figure 10. IBM-PC - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

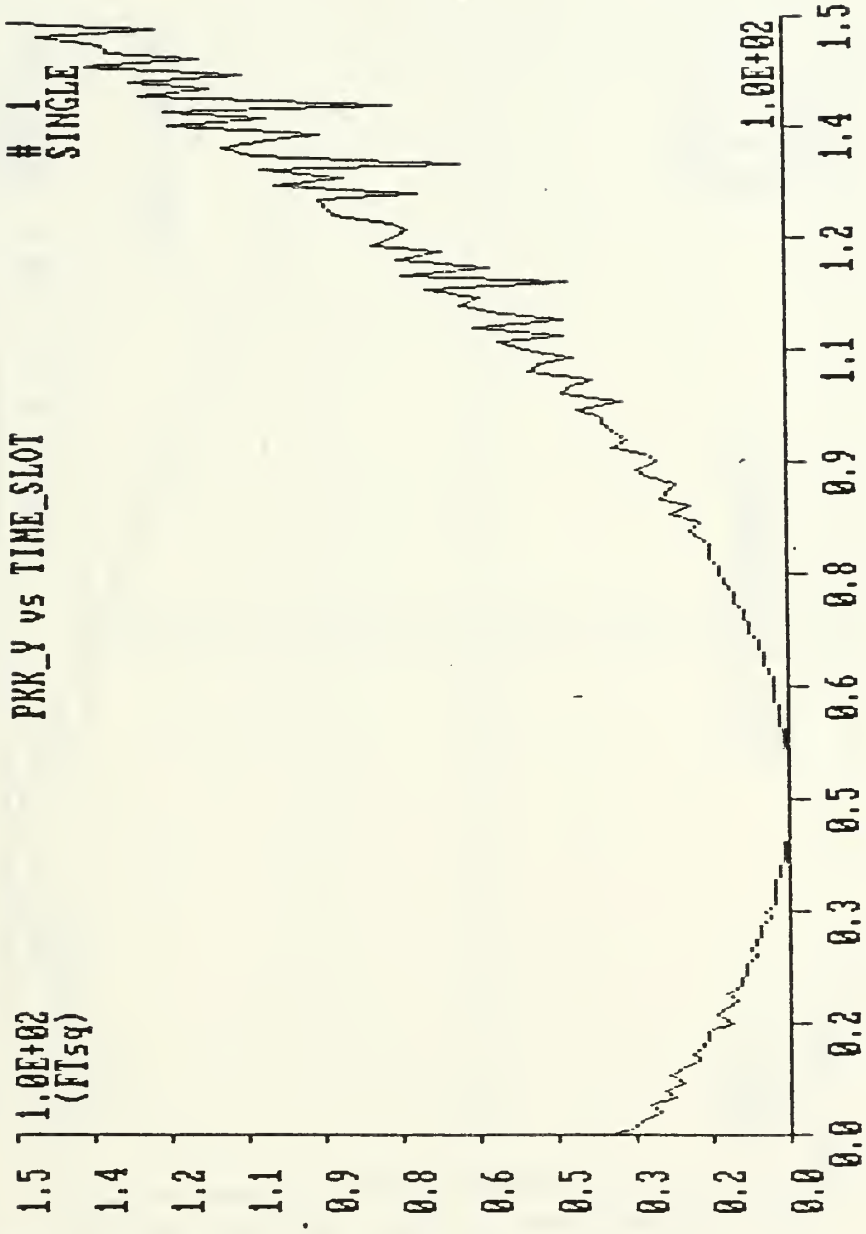


Figure 11. IBM-PC - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

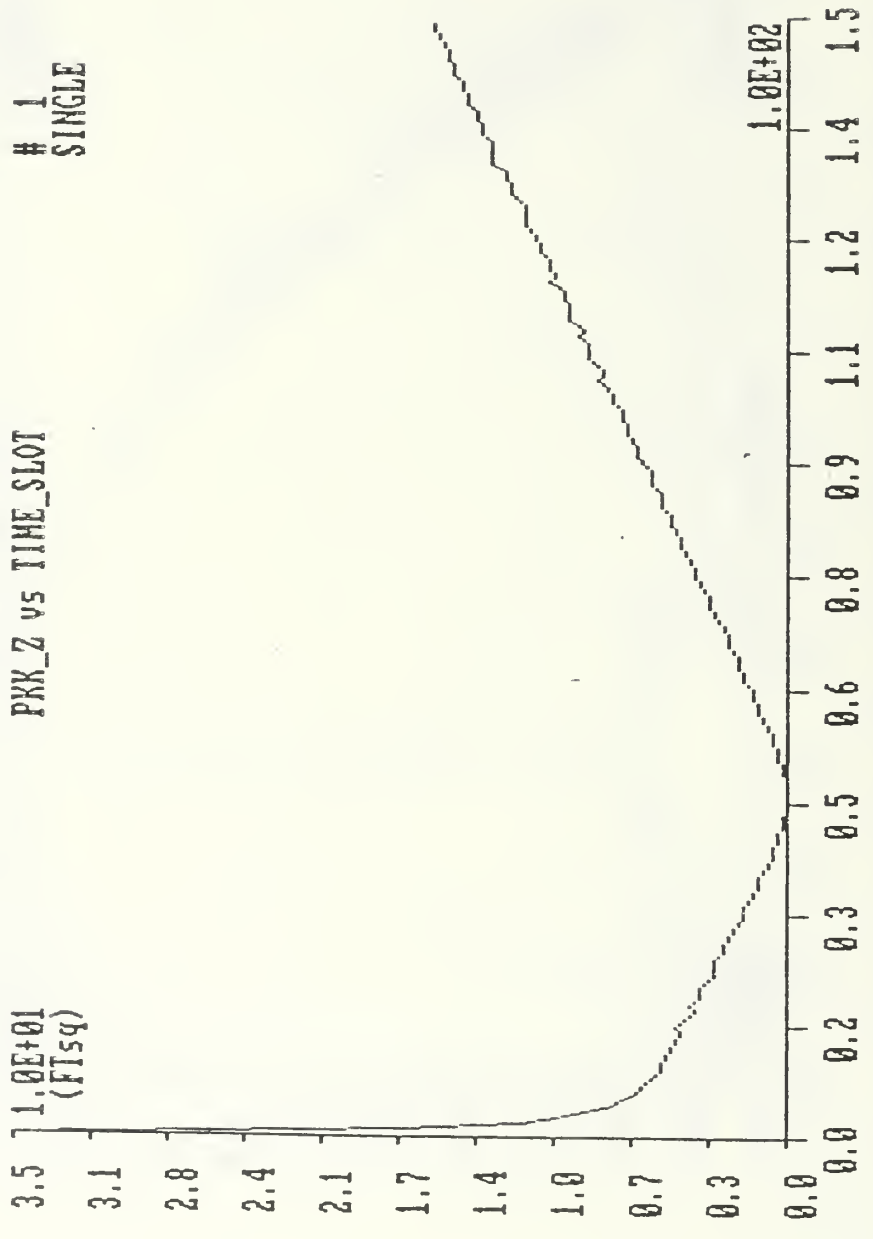


Figure 12. IBM-PC - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

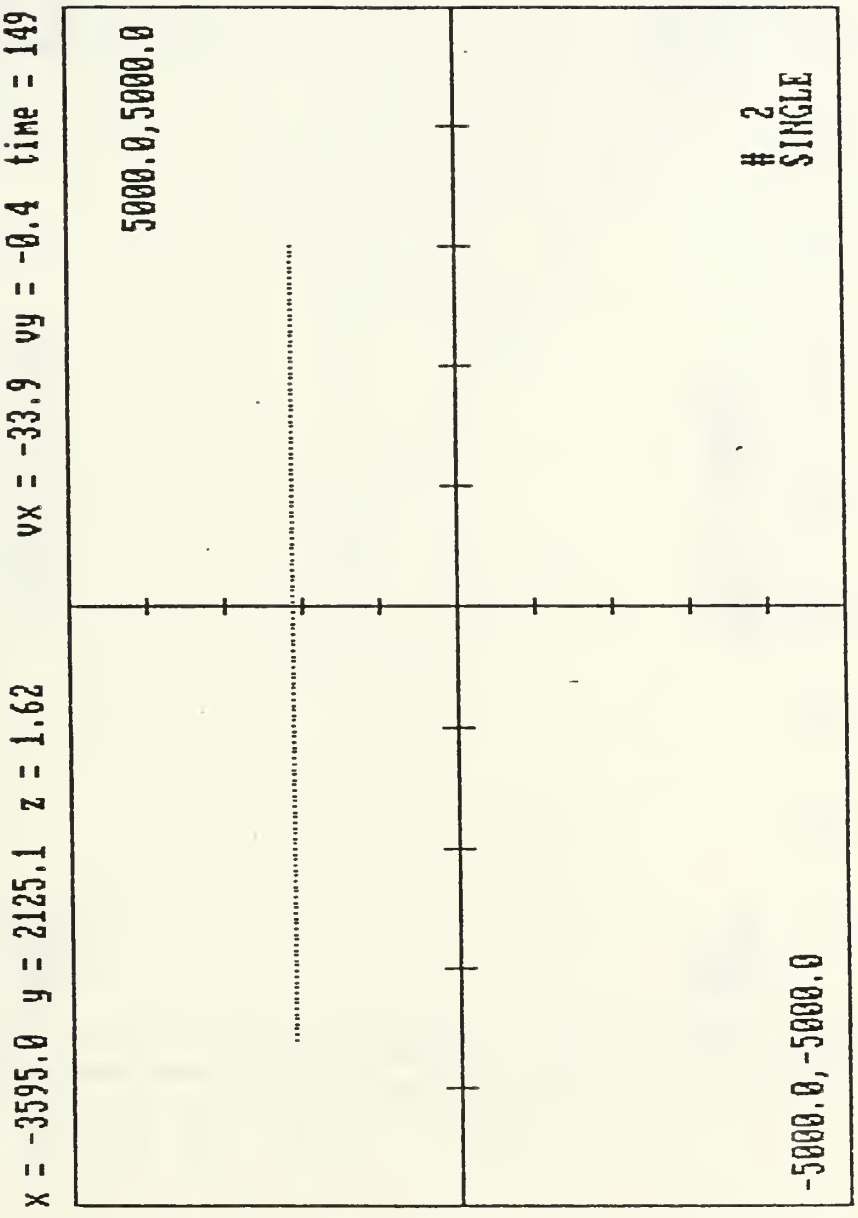


Figure 13. IBM-PC - Real time plot of estimated trajectory of the torpedo in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

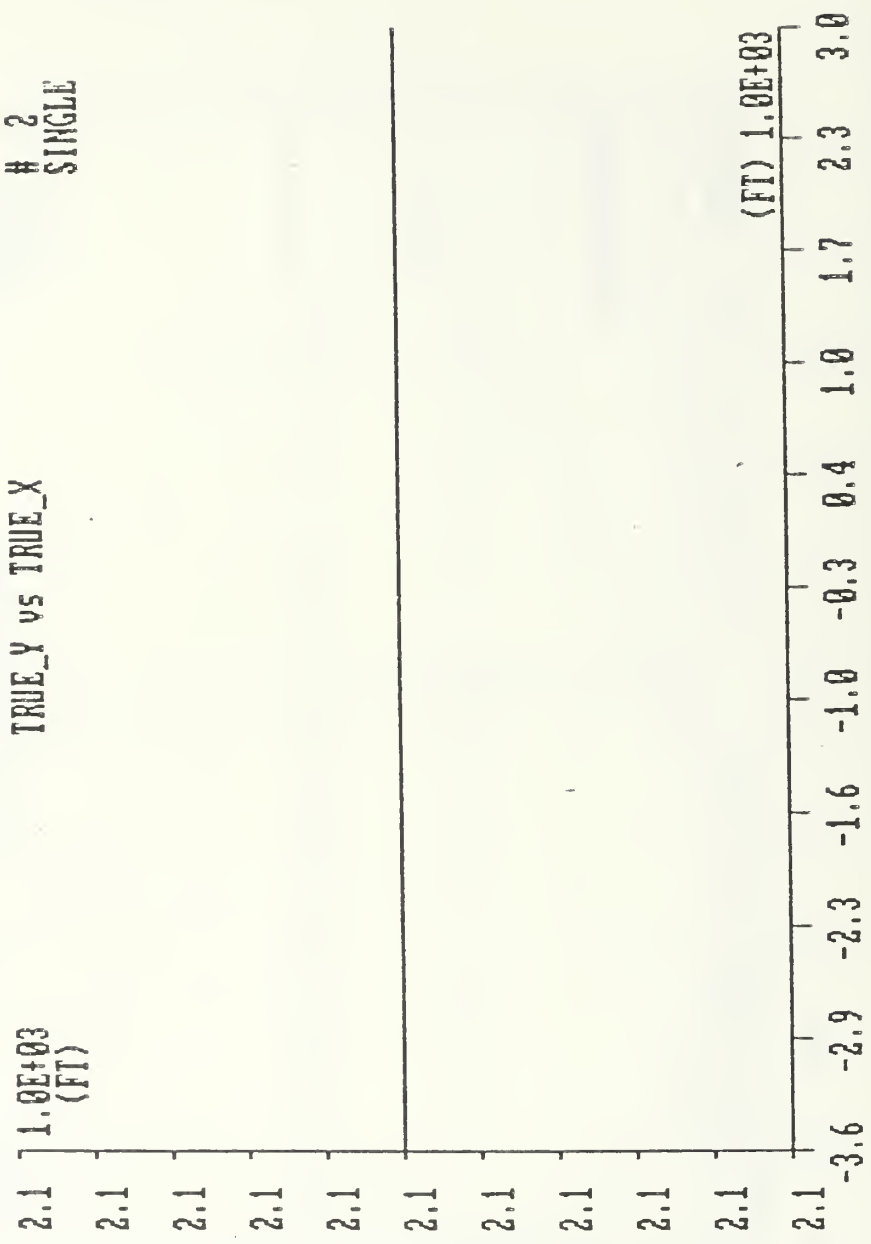


Figure 14. IBM-PC - True trajectory of the torpedo in the area of a single array. Initial conditions : x=3000 ft, y=2121 ft, Vx=-33.8 ft/sec, Vy=0 ft/sec.

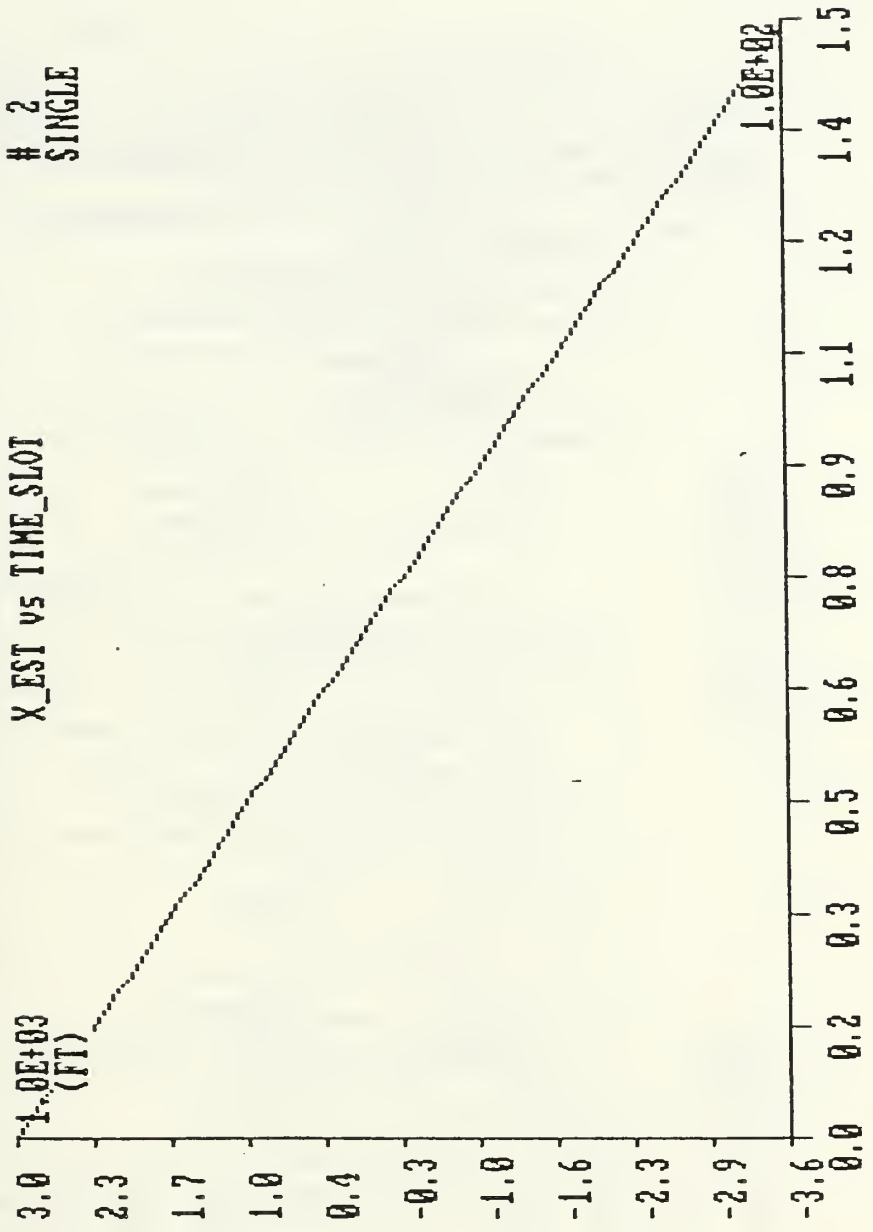


Figure 15. IBM-PC - Estimate of torpedo position during a straight run in the area of single array. Initial conditions are: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

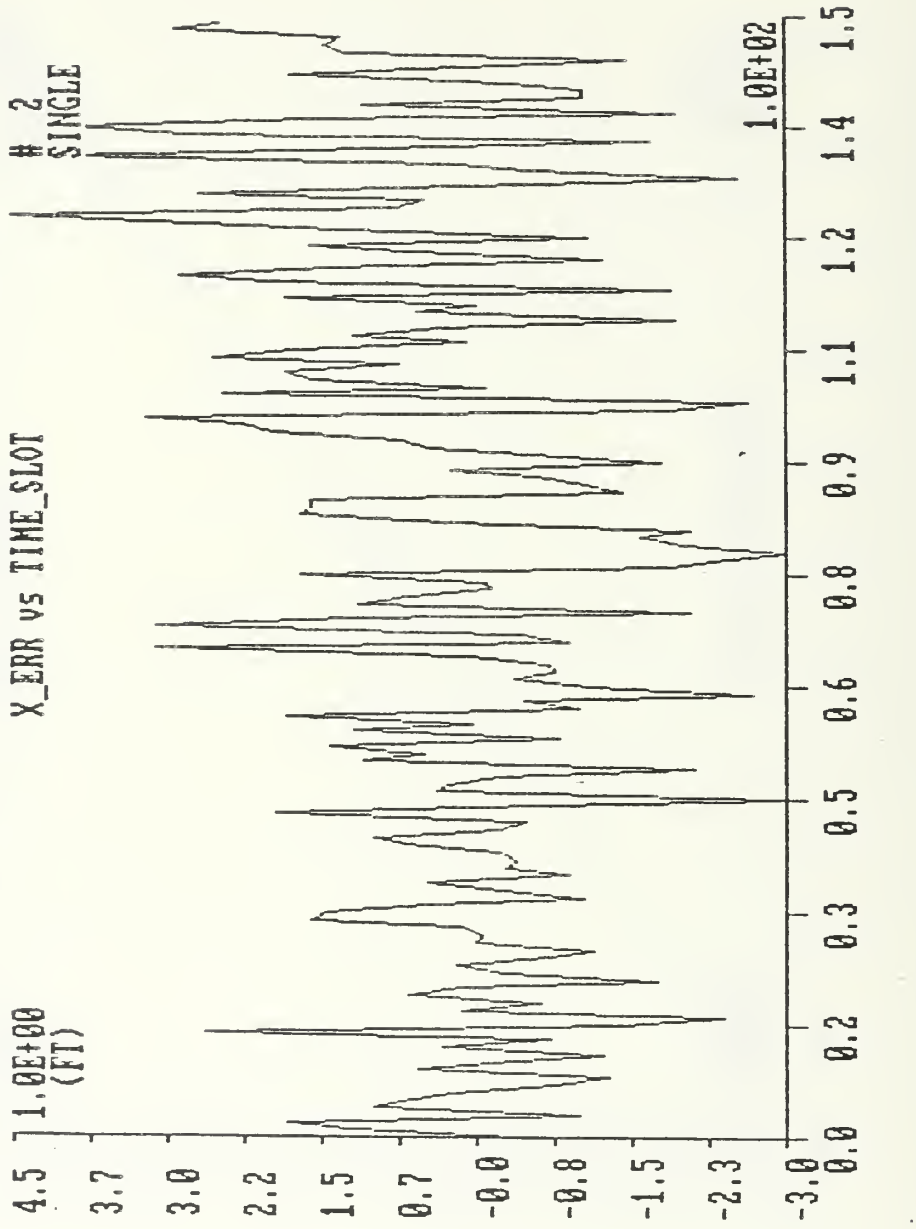


Figure 16. IBM-PC - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

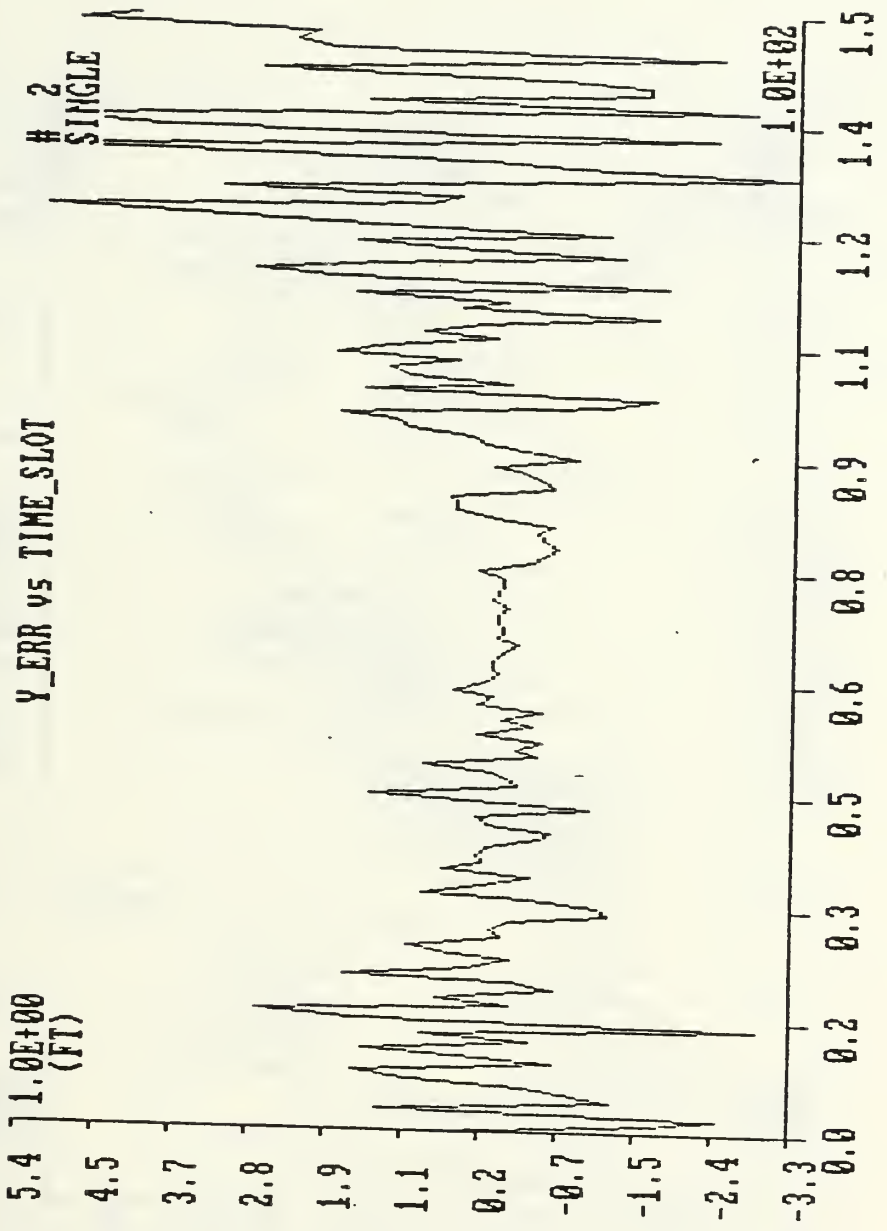


Figure 17. IBM-PC - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

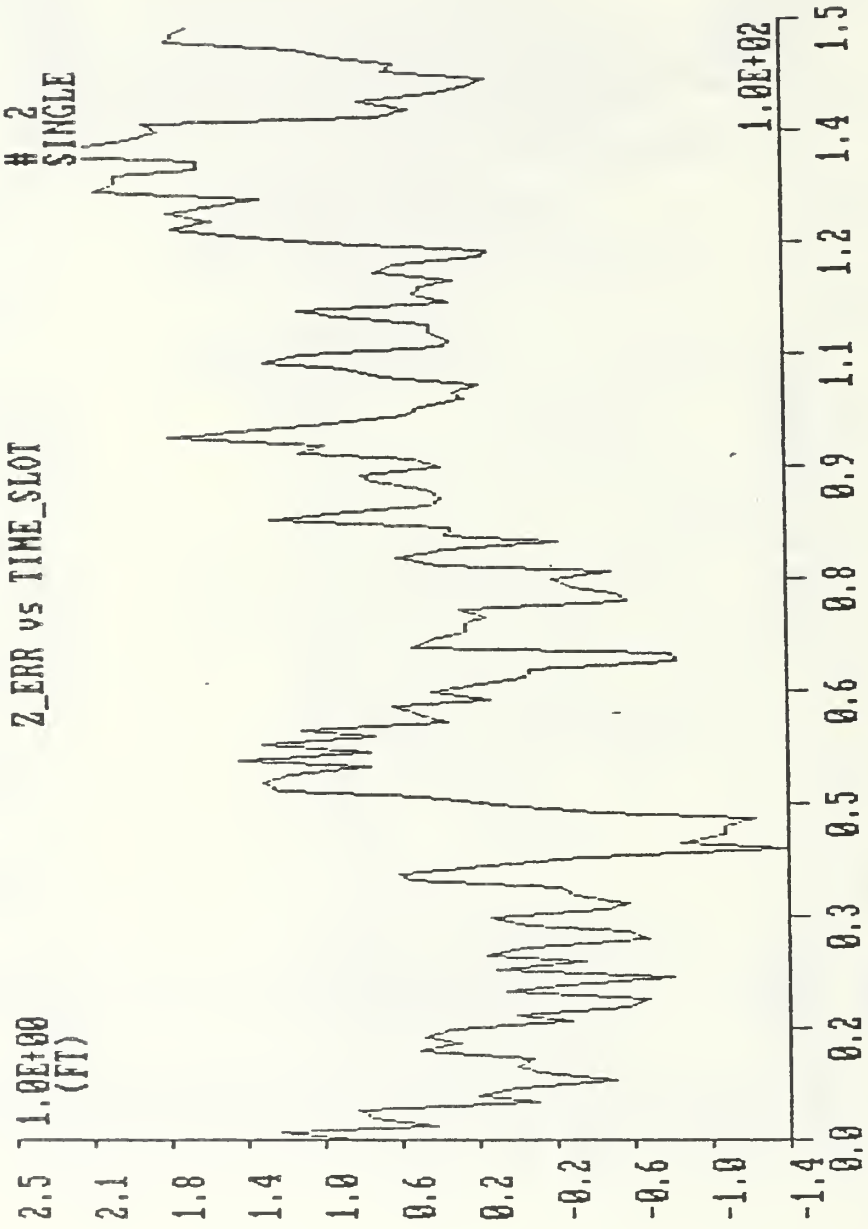


Figure 18. IBM-PC - Error in torpedo depth during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

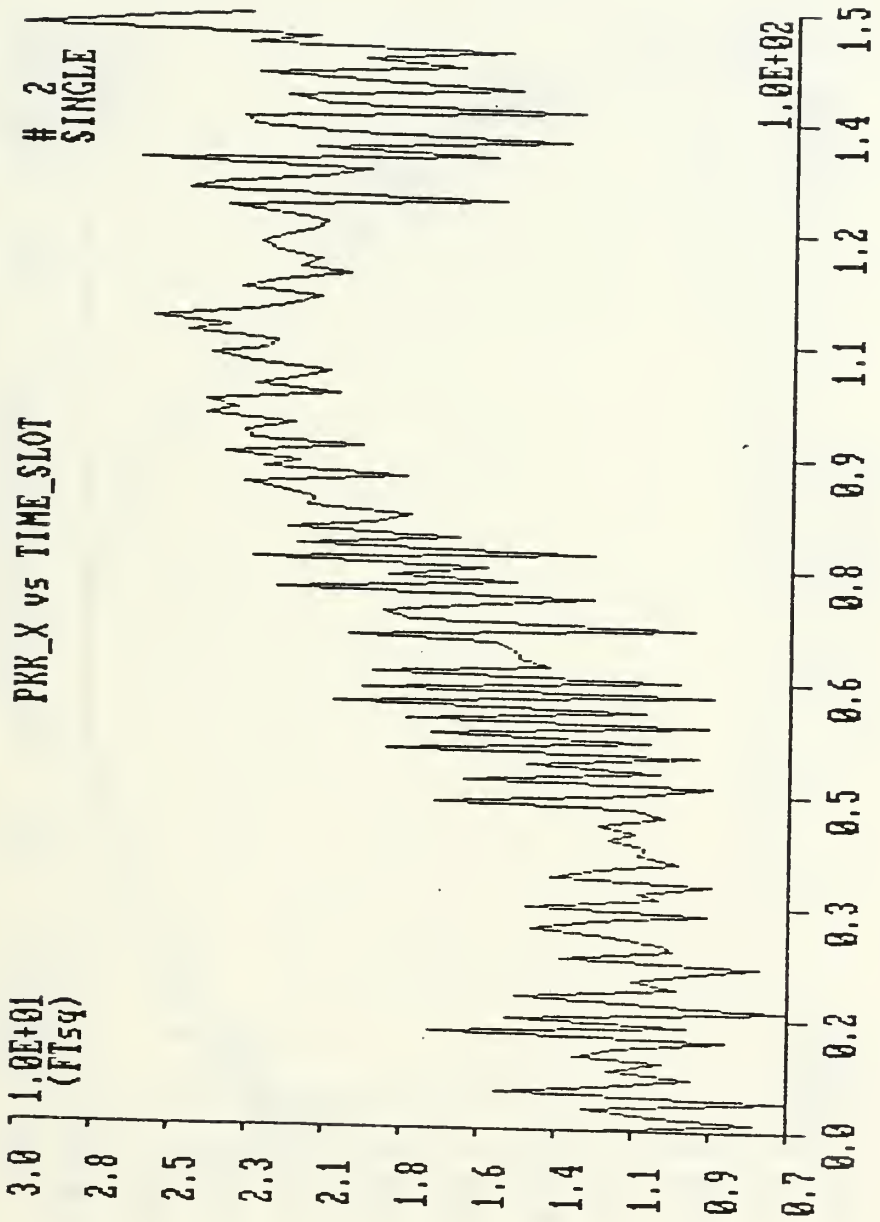


Figure 19. IBM-PC - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

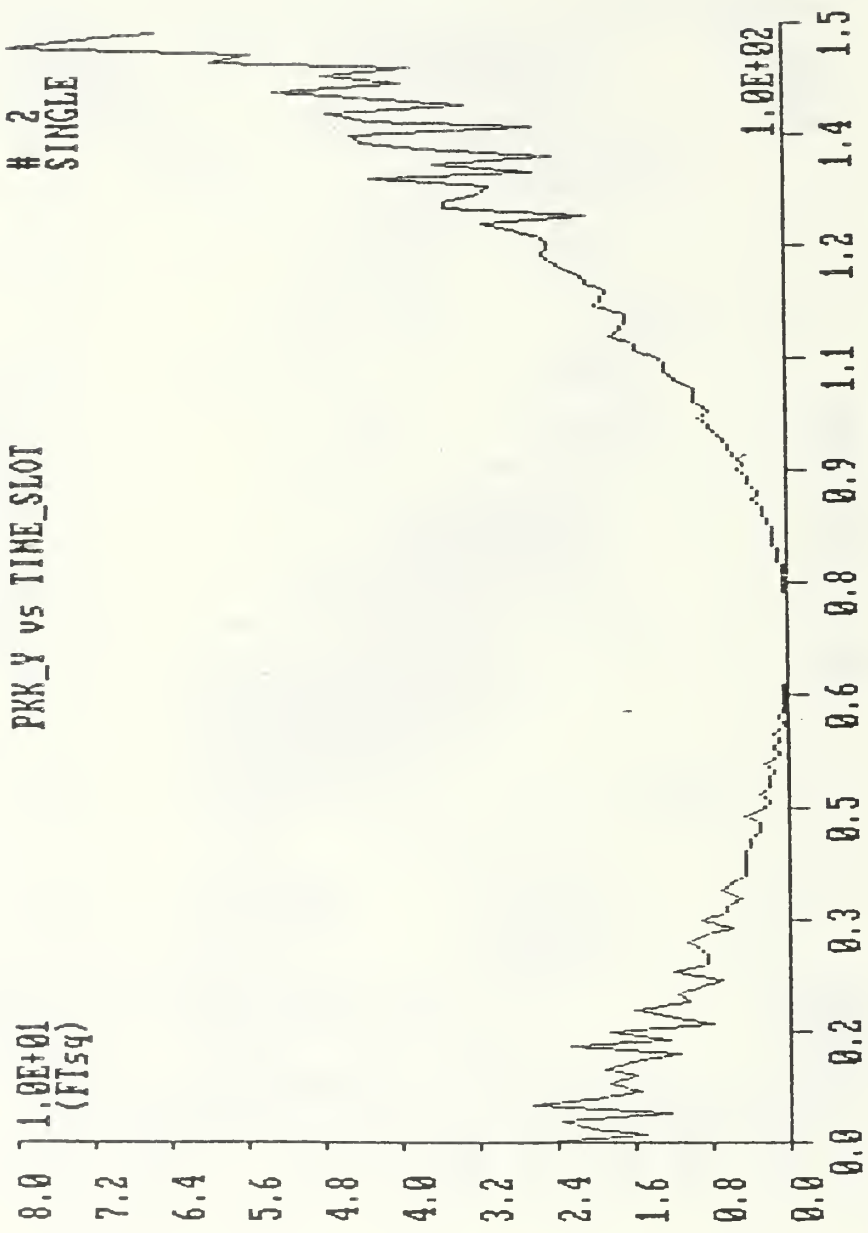


Figure 20. IBM-PC - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

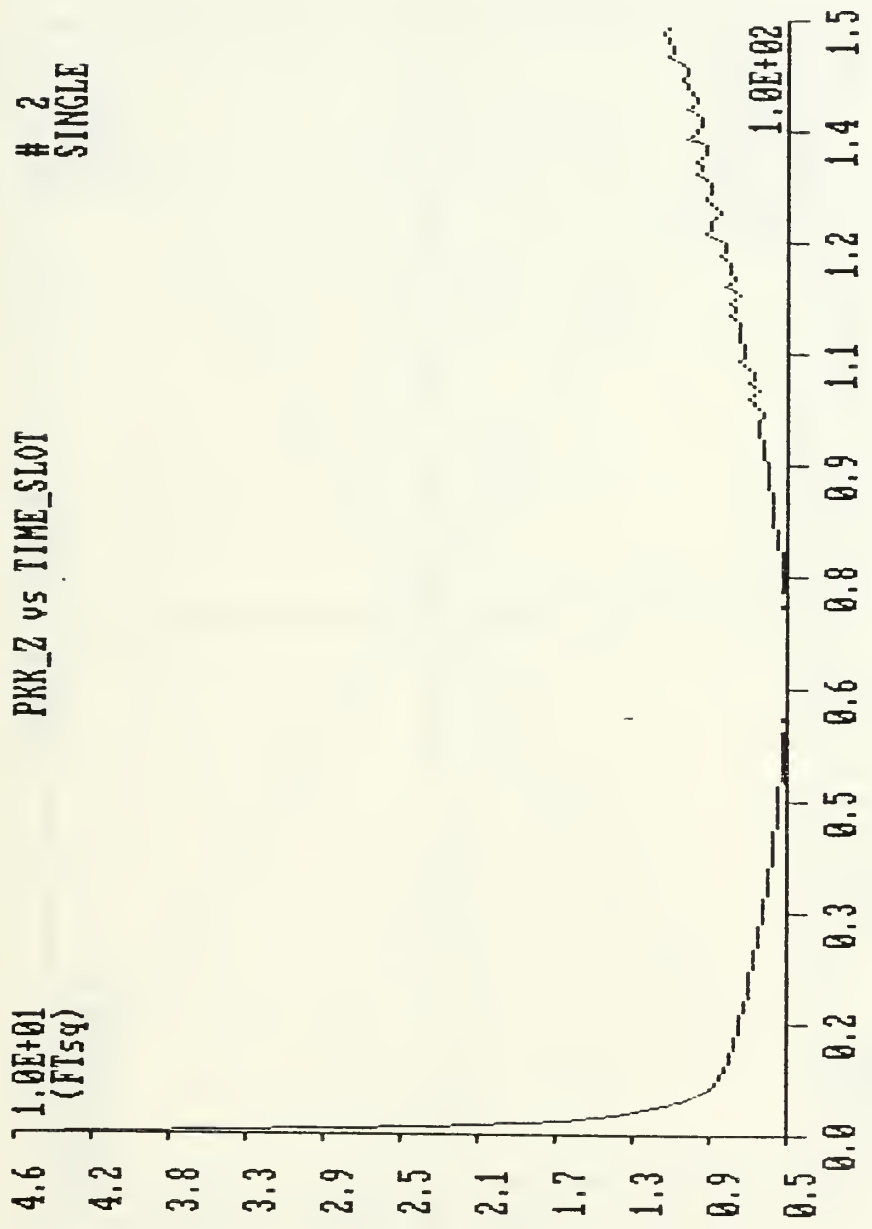


Figure 21. IBM-PC - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

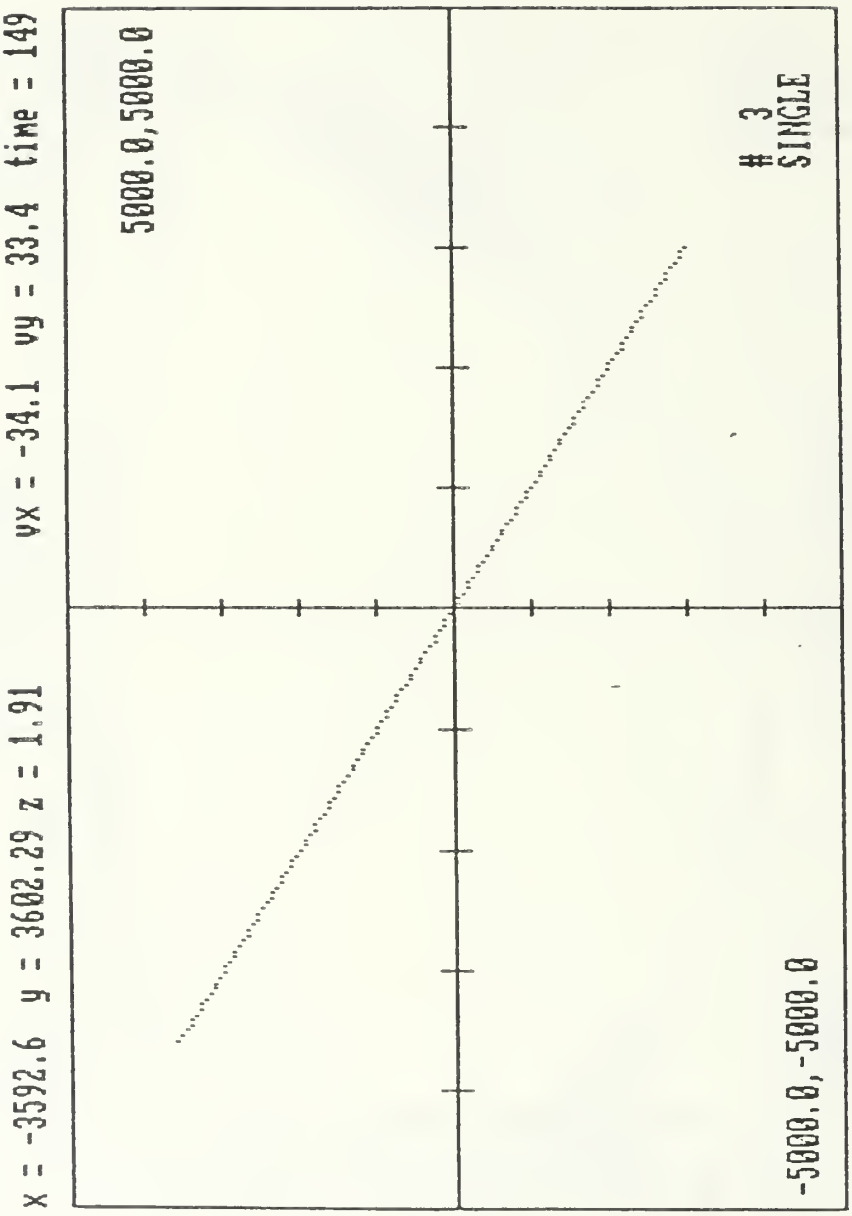


Figure 22. IBM-PC - Real time plot of estimated trajectory of the torpedo in the area of a single array. Initial conditions: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=-33.8 ft/s.

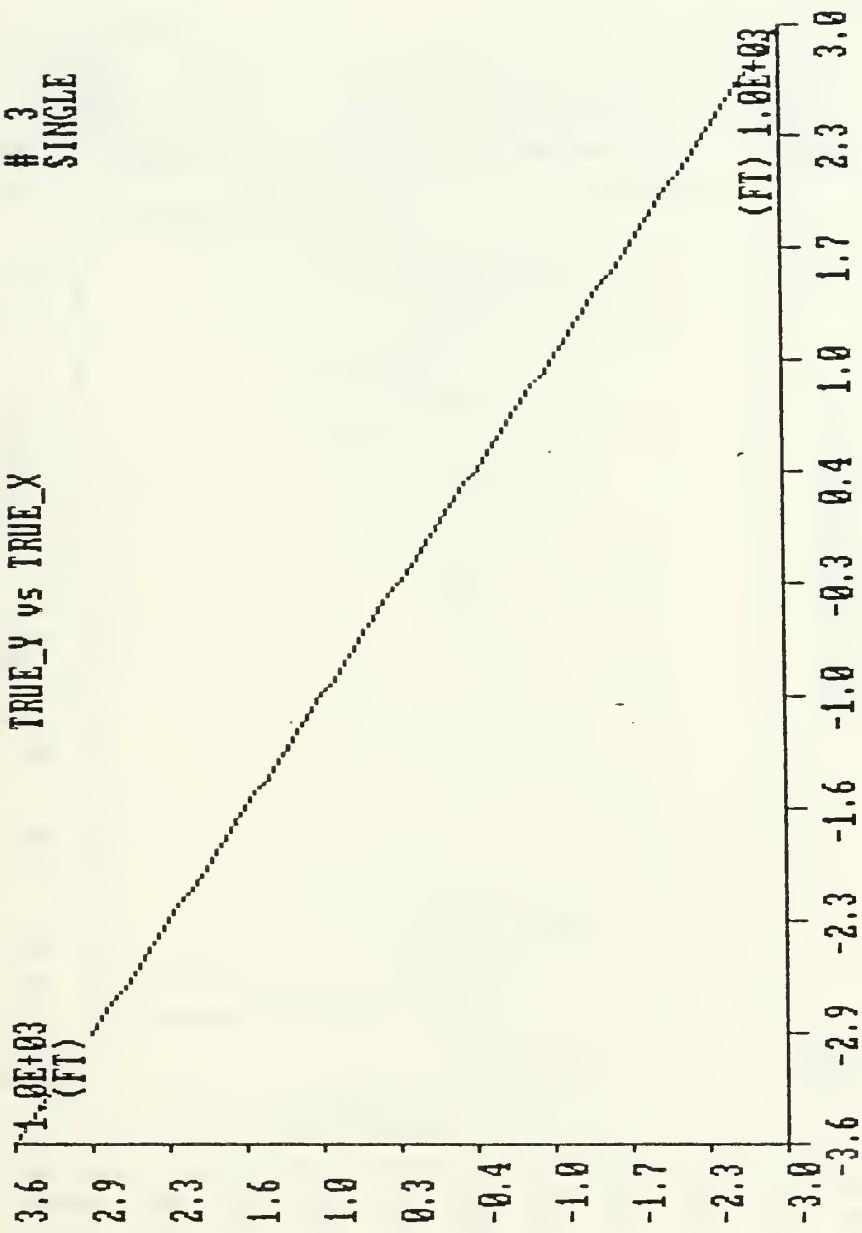


Figure 23. IBM-PC - True trajectory of the torpedo in the area of a single array. Initial conditions : x=3000 ft, y=-3000 ft, Vx=-33.8 ft/sec, Vy=33.8 ft/sec.

Figure 24. IBM-PC - Estimate of torpedo position during a straight run in the area of single array. Initial conditions are: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

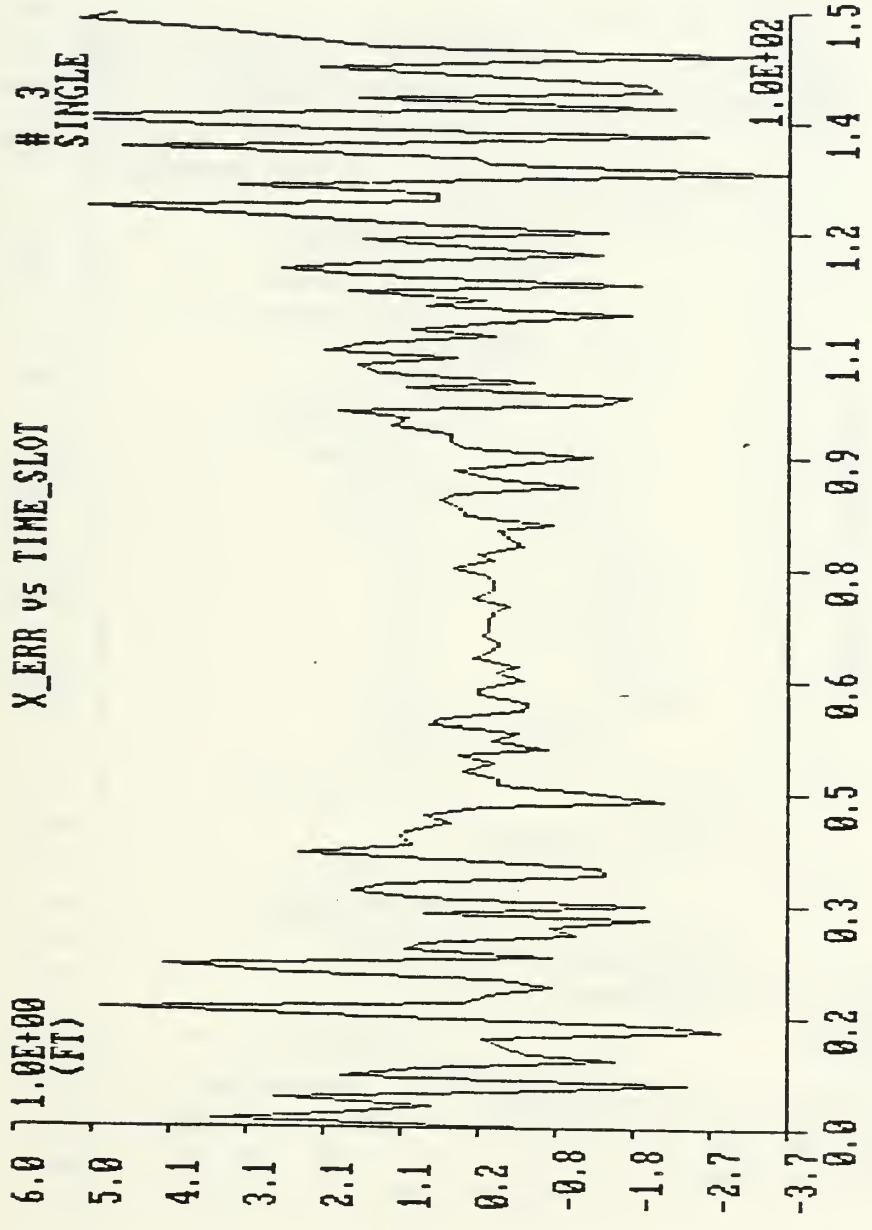


Figure 25. IBM-PC - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

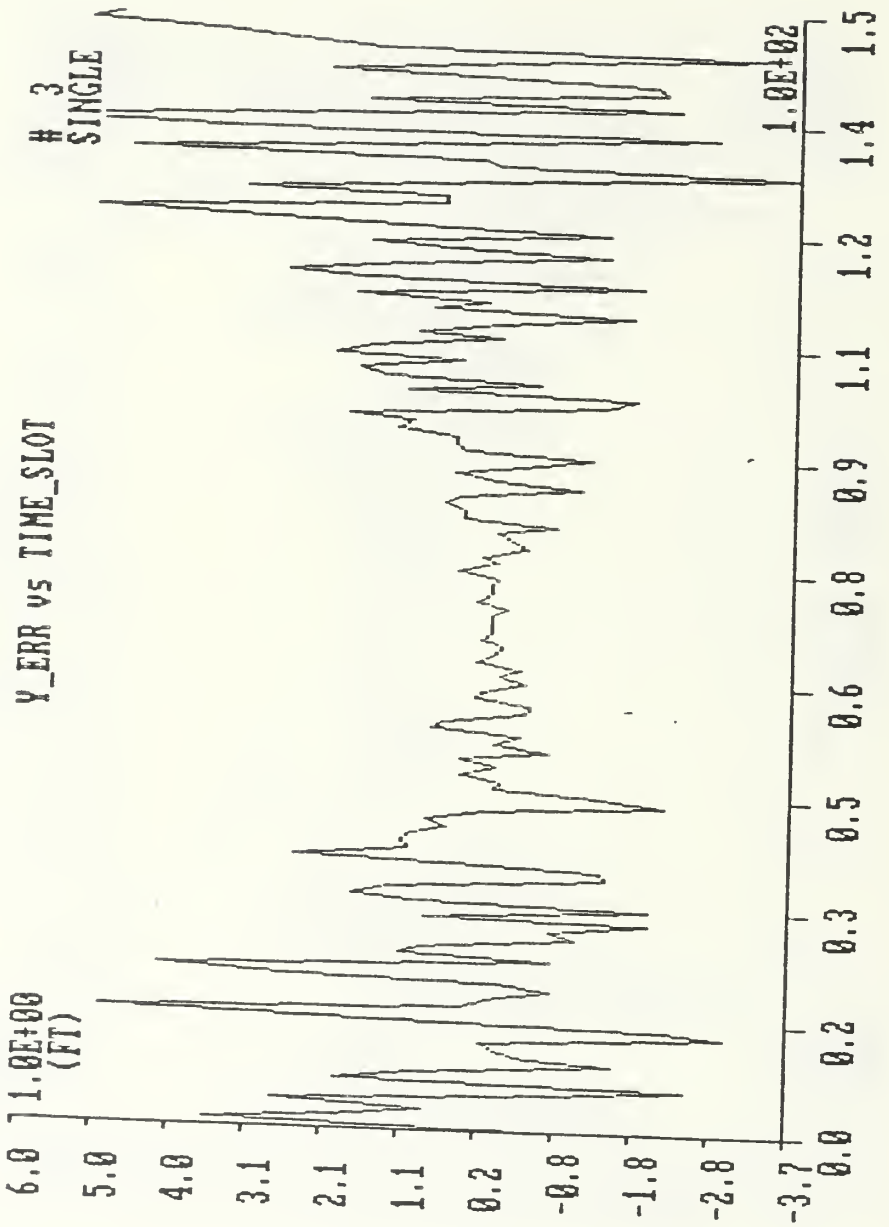


Figure 26. IBM-PC - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

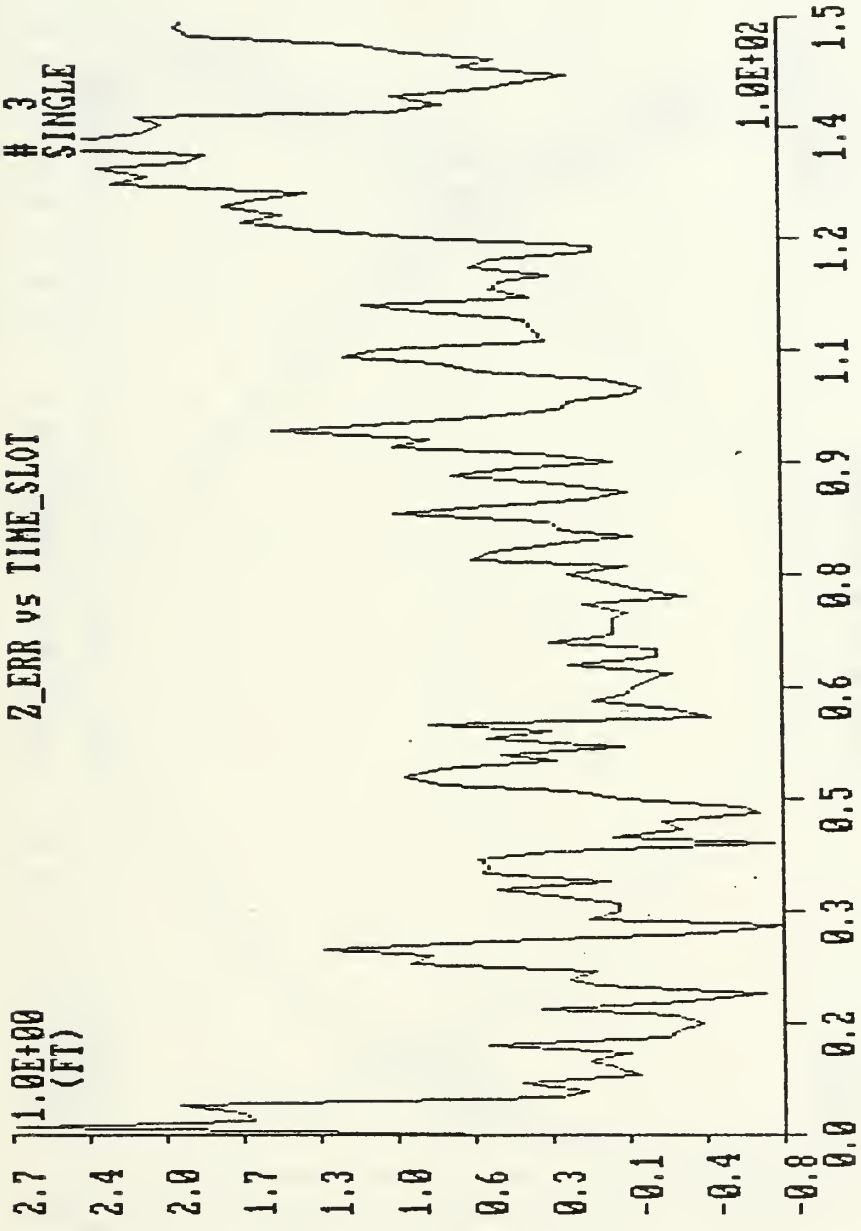


Figure 27. IBM-PC - Error in torpedo depth during a straight run in the area of single array. Initial conditions : x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

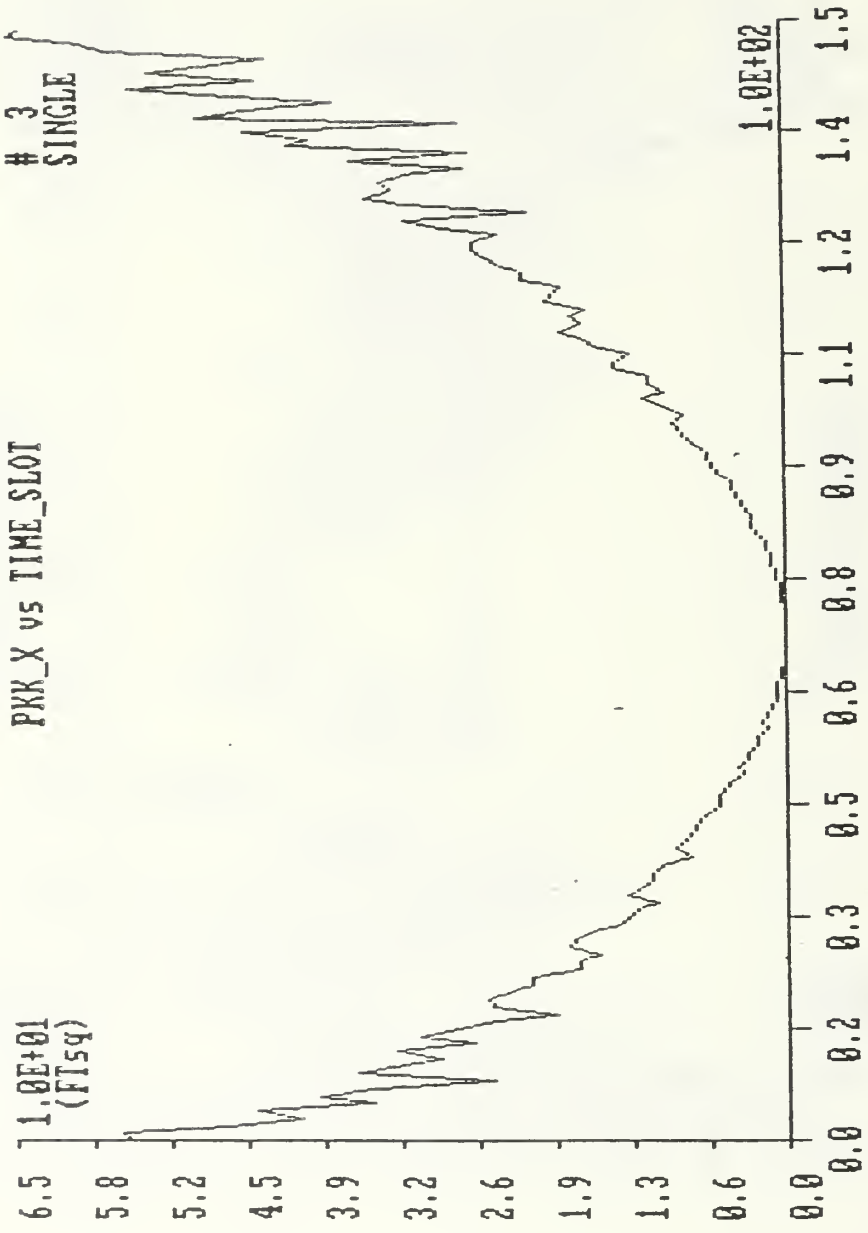


Figure 28. IBM-PC - Filter estimate of mean-square error for a straight run in the area of single array. Initial conditions: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

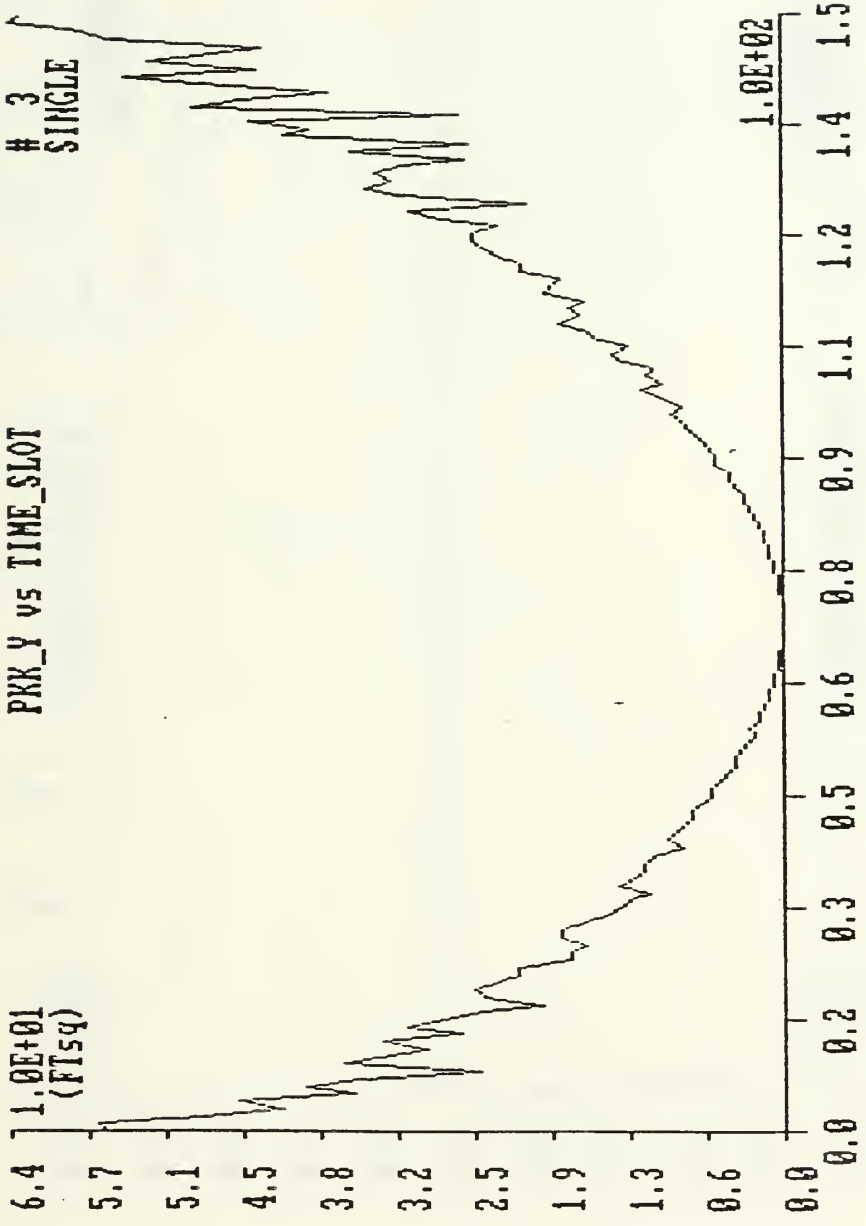


Figure 29. IBM-PC - Filter estimate of mean-square error for a straight run in the area of single array. Initial conditions: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

Figure 30. IBM-PC - Filter estimate of mean-square error for a straight run in the area of single array. Initial conditions: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

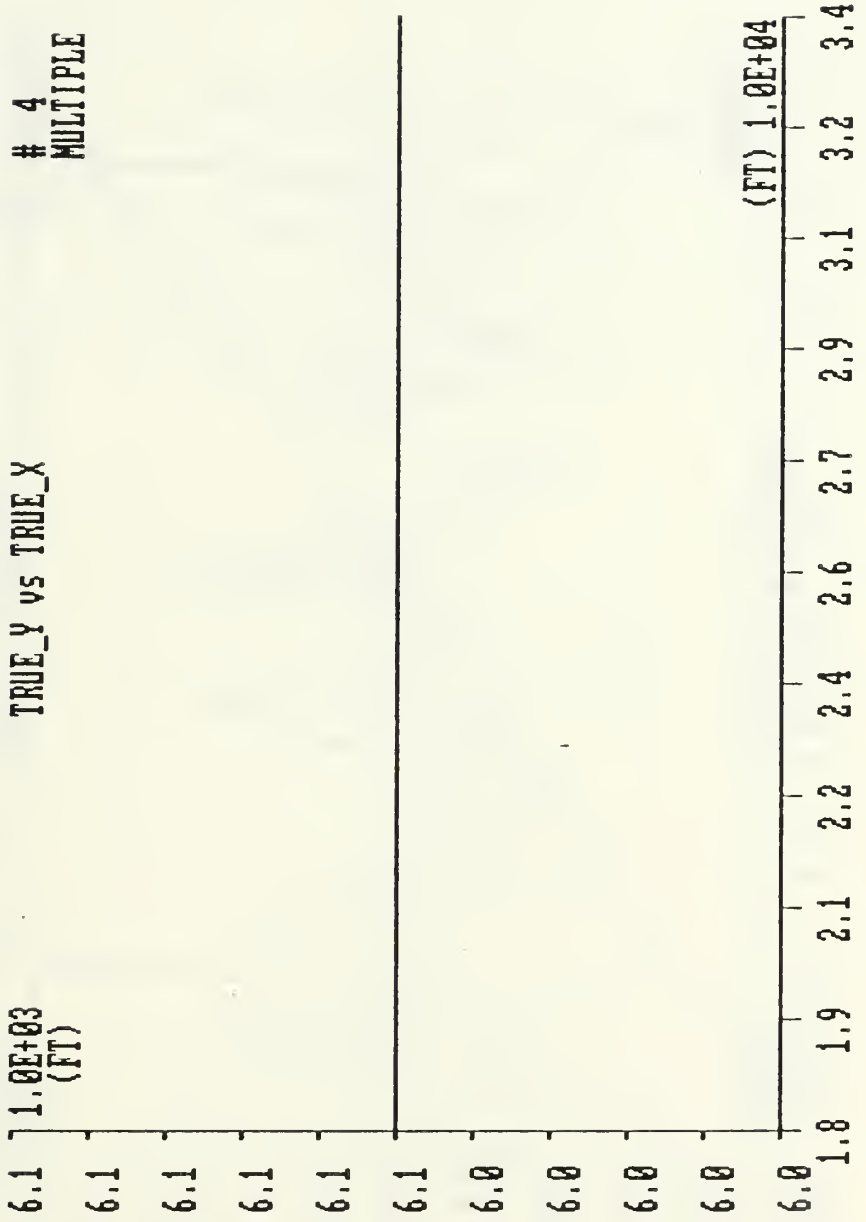


Figure 31. IBM-PC - True trajectory of the torpedo in the during a straight runthrough multiple arrays. Initial conditions: x=34000 ft, y=6050 ft, Vx=-84.4 ft/sec, Vy=0 ft/sec.

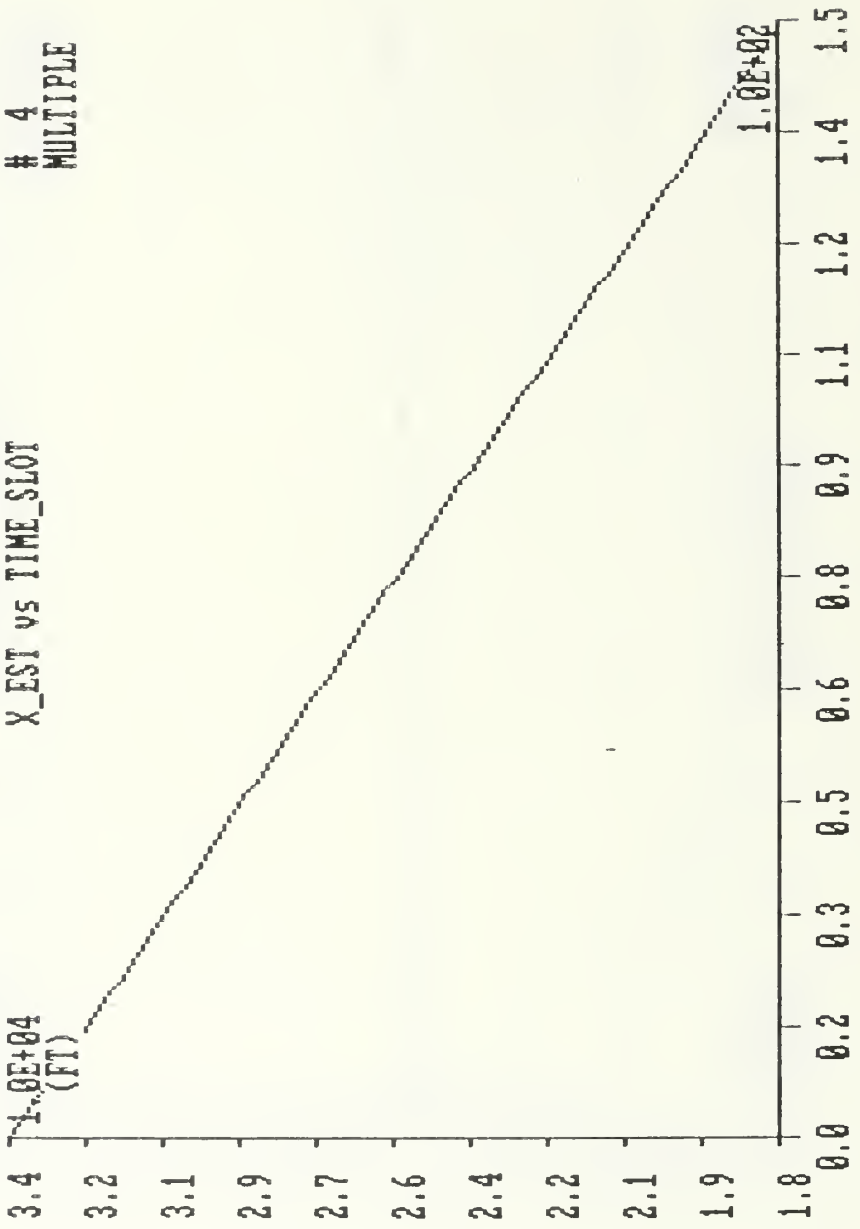


Figure 32. IBM-PC - Estimate of torpedo position during a straight run through multiple arrays. Initial conditions are: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

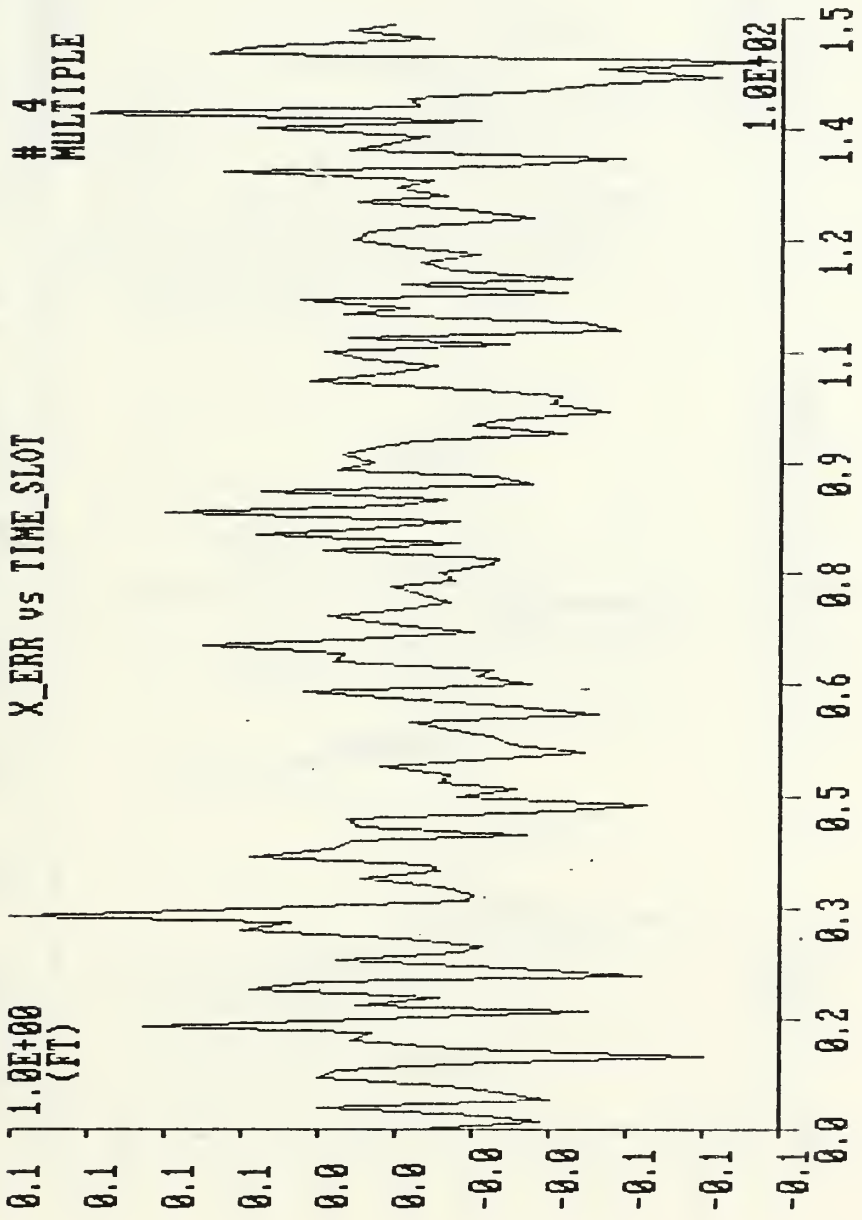


Figure 33. IBM-PC - Error in torpedo position during a straight run through multiple arrays. Initial conditions are: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

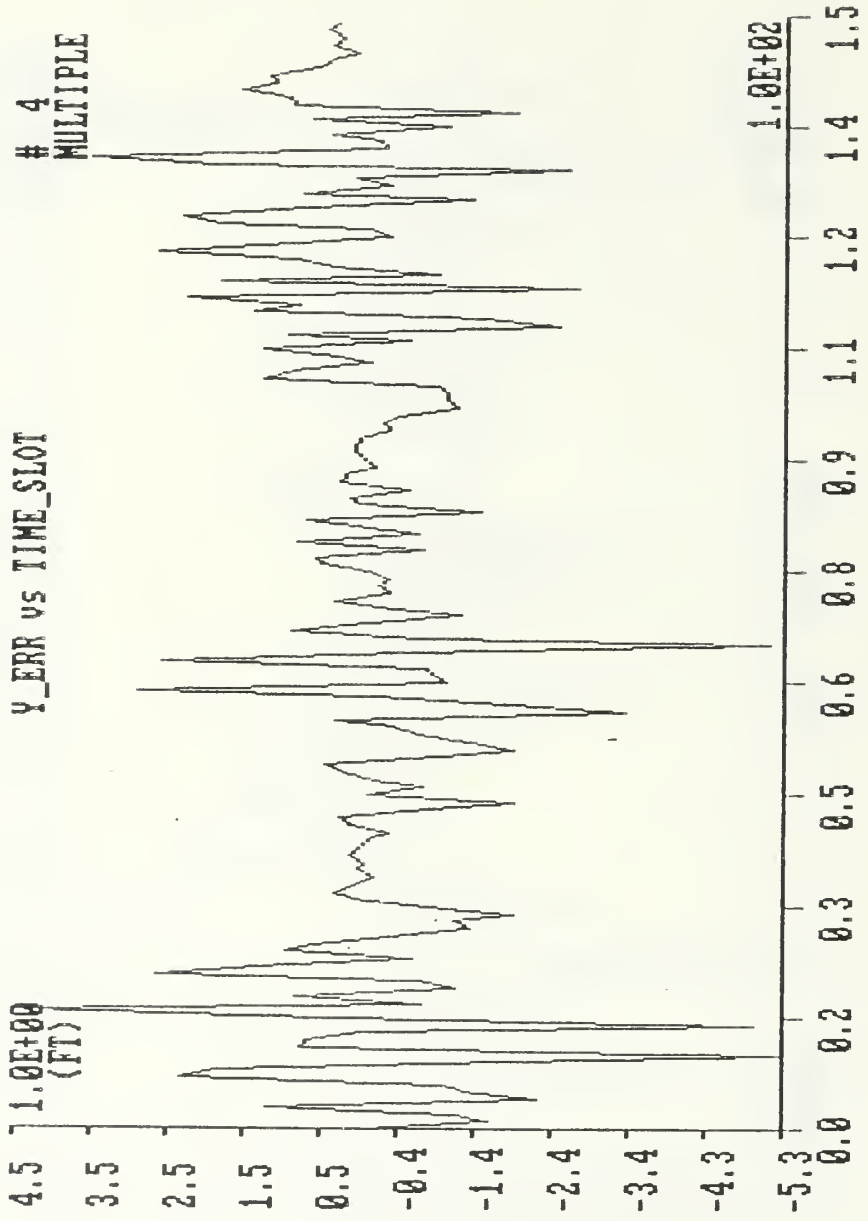


Figure 34. IBM-PC - Error in torpedo position during a straight run through multiple arrays. Initial conditions are: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

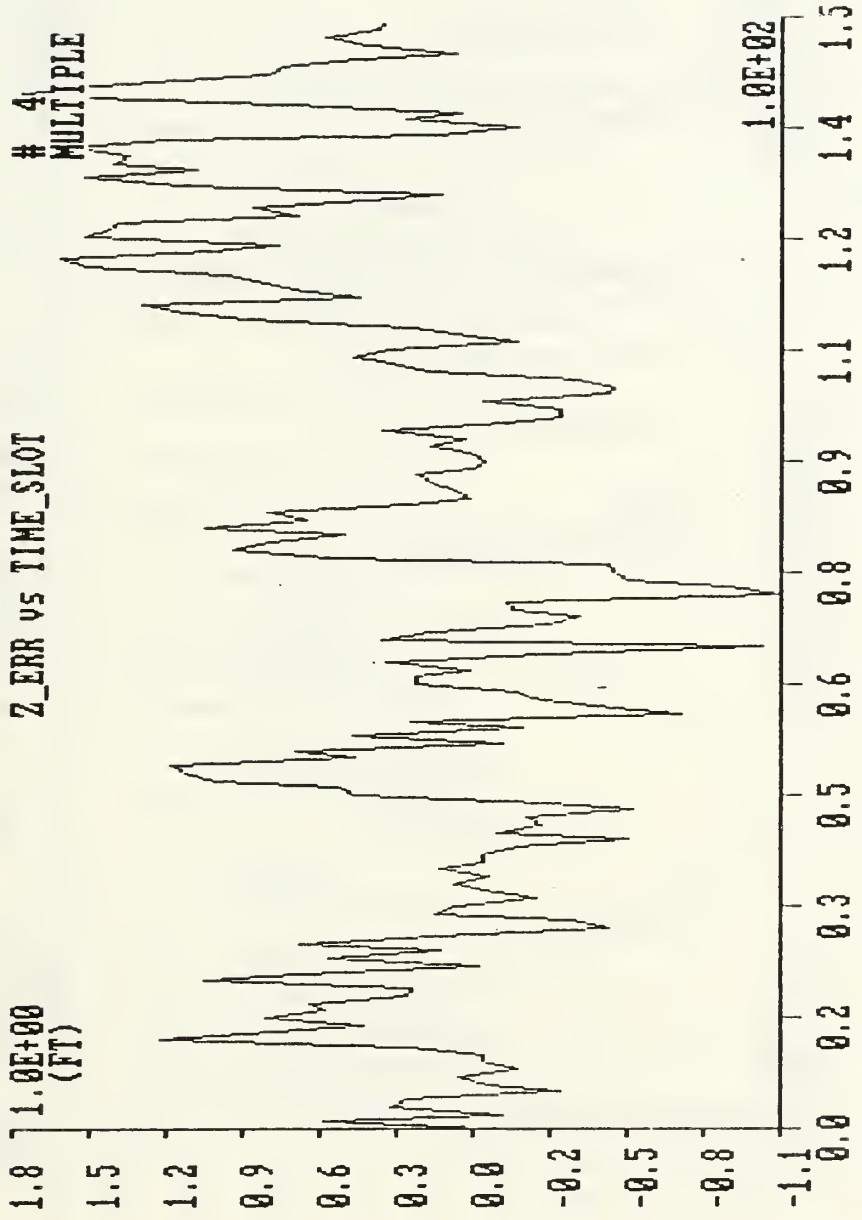


Figure 35. IBM-PC - Error in torpedo depth during a straight run through multiple arrays. Initial conditions are: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

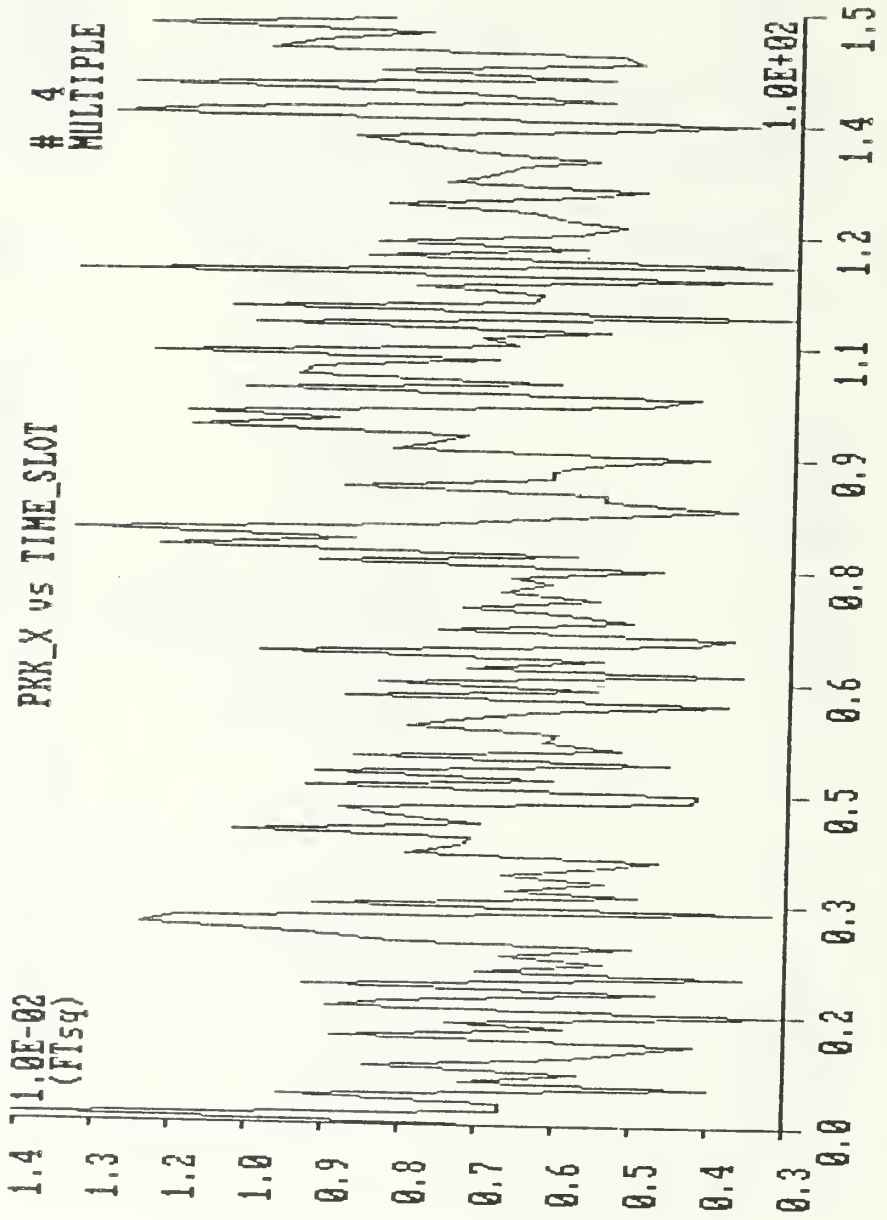


Figure 36. IBM-PC - Filter estimate of mean-square error for a straight run through multiple arrays. Initial conditions: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

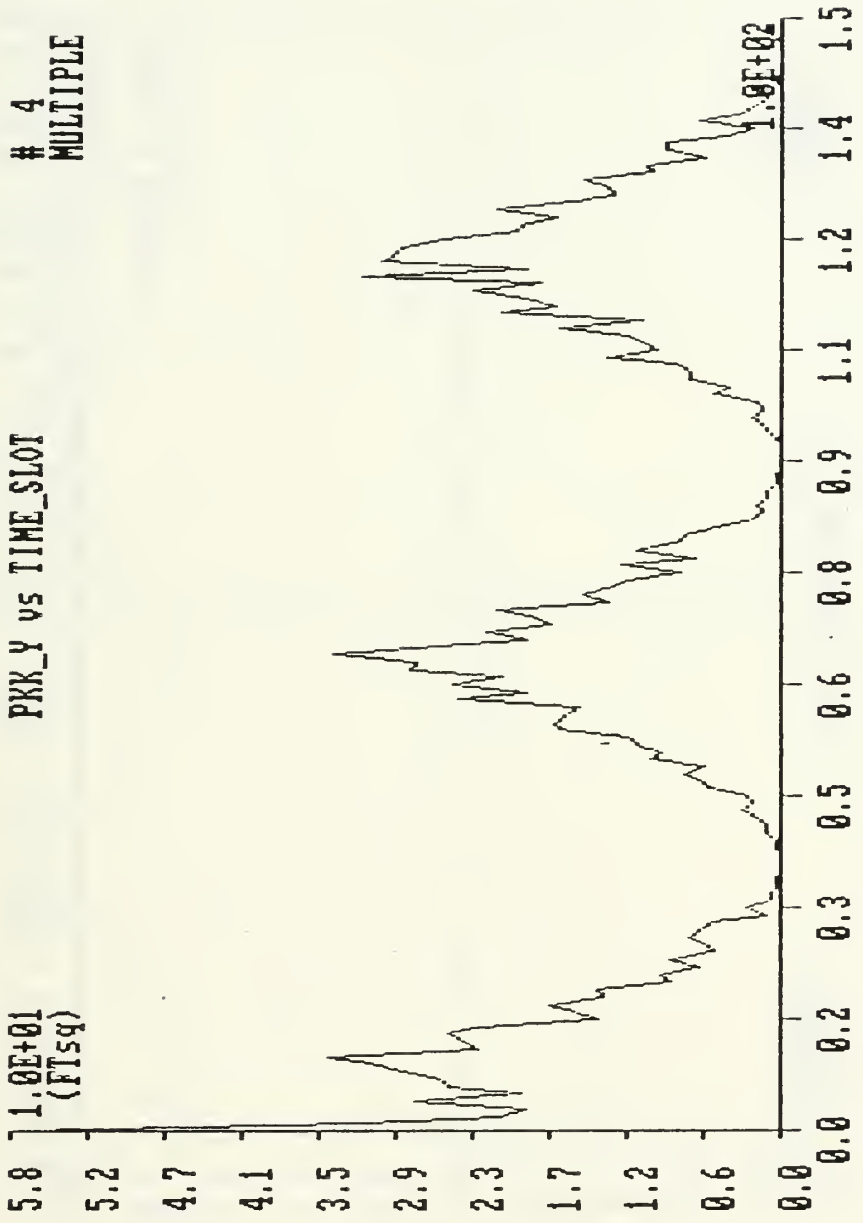


Figure 37. IBM-PC - Filter estimate of mean-square error for a straight run through multiple arrays. Initial conditions: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

Figure 38. IBM-PC - Filter estimate of mean-square error for a straight run in the area of single array. Initial conditions: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

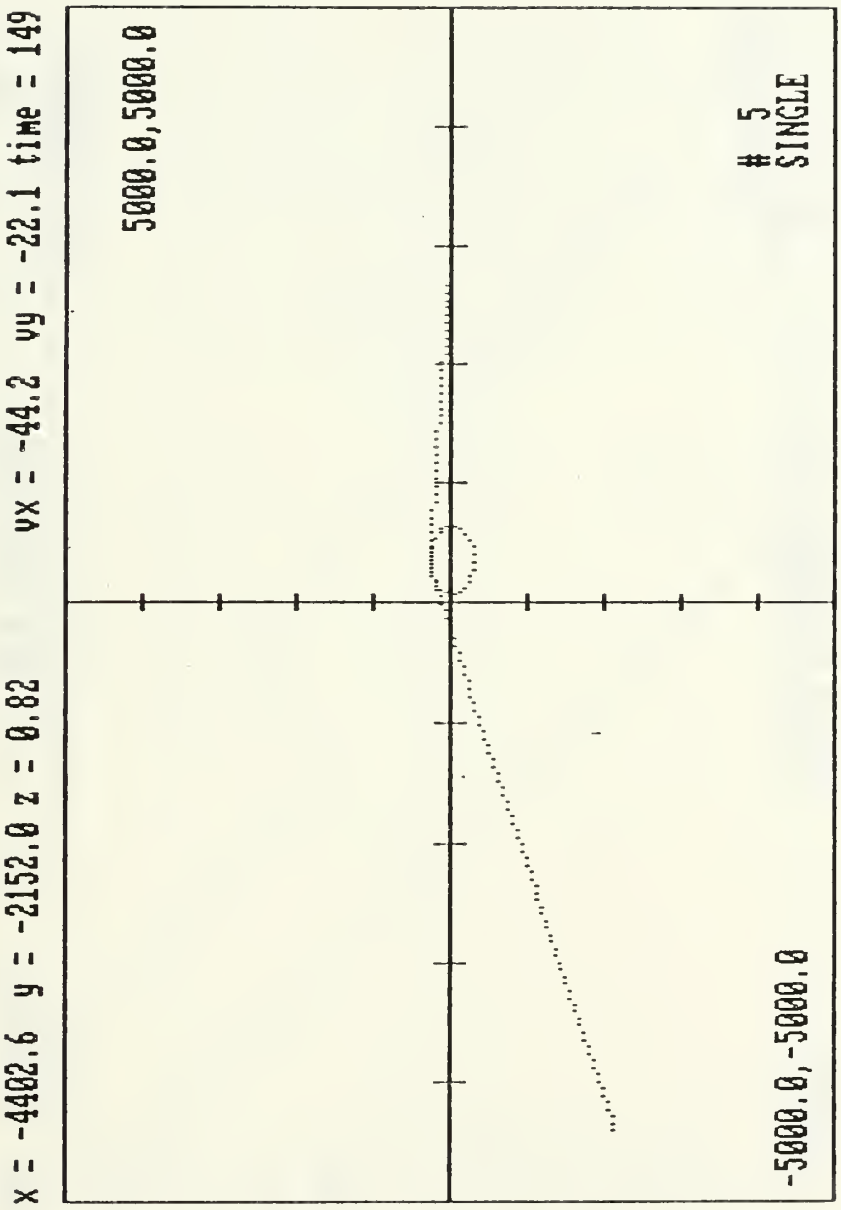


Figure 39. IBM-PC - Real time plot of estimated trajectory of the torpedo in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=50 ft/s, Vy=5 ft/s.

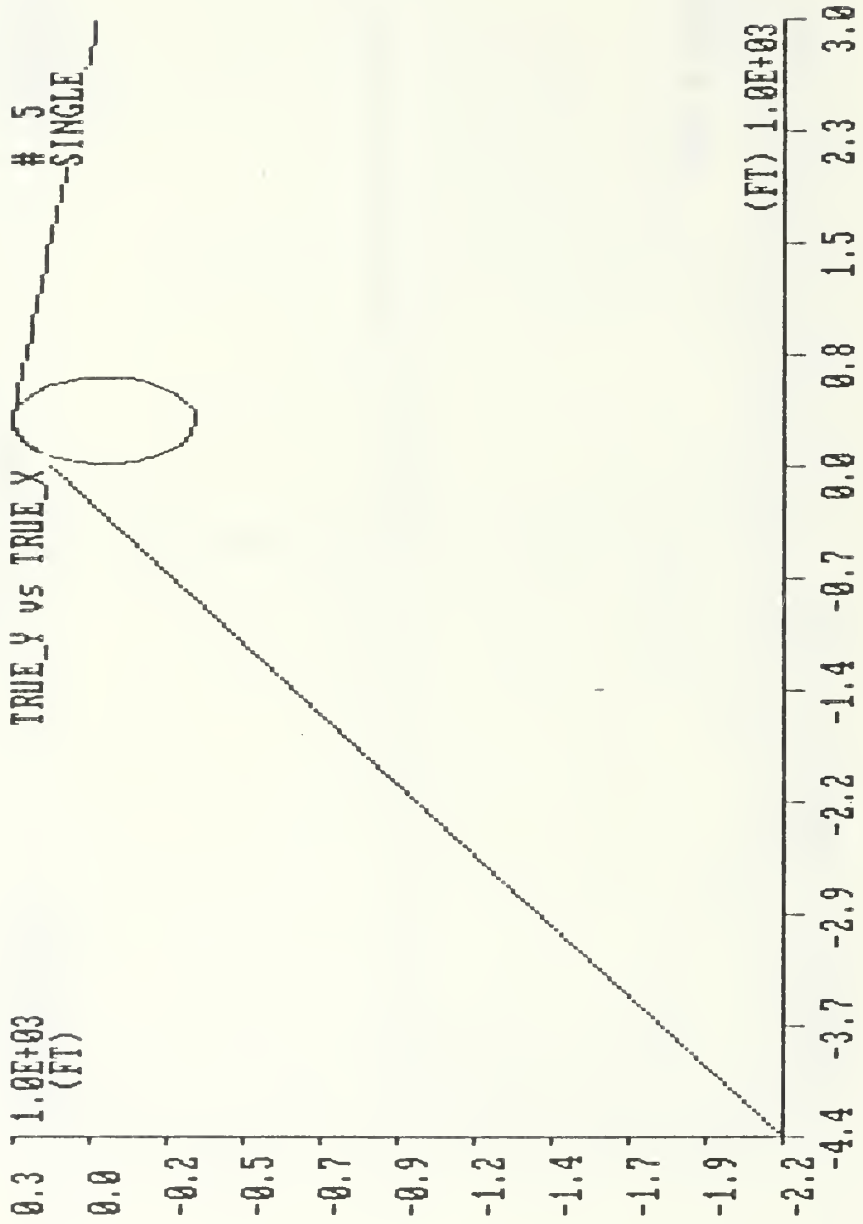


Figure 40. IBM-PC - True trajectory of the torpedo during a 10 dg/s maneuver in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

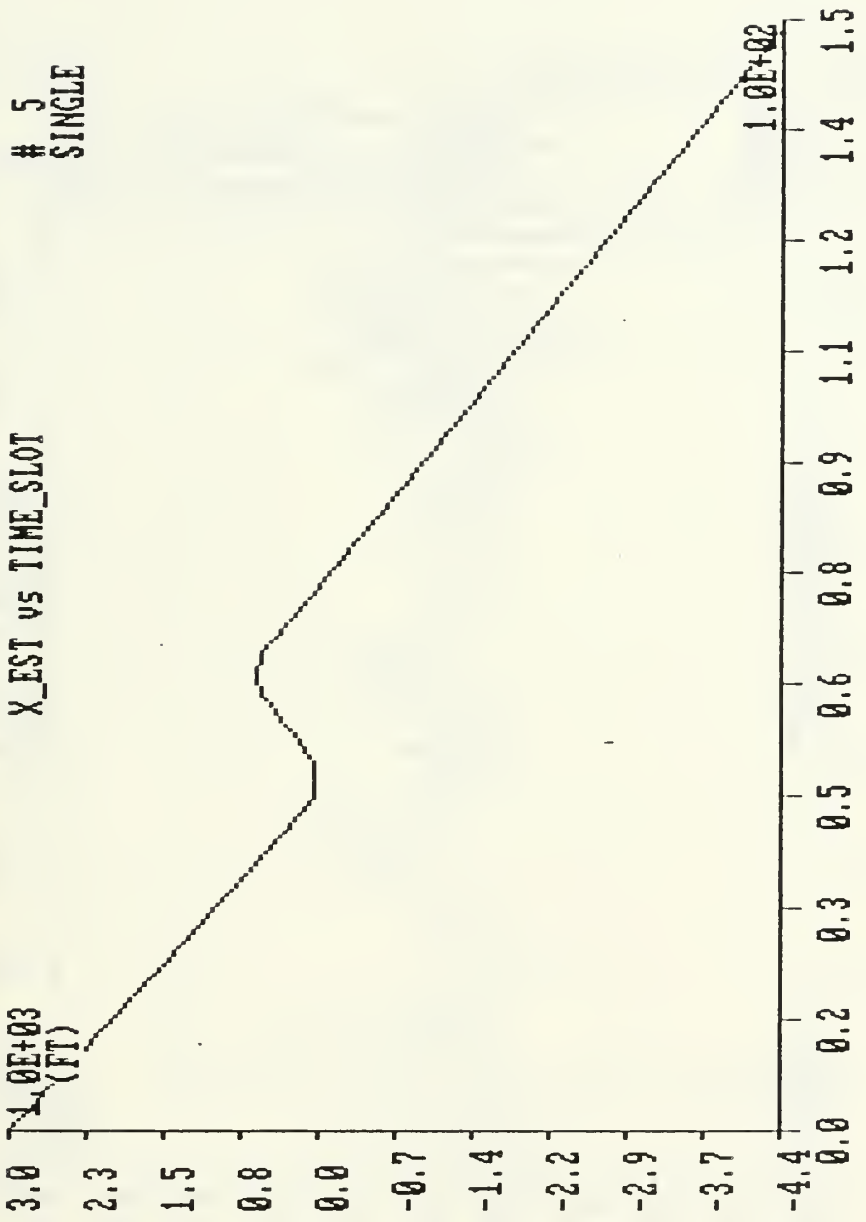


Figure 41. IBM-PC - Estimate of torpedo position during a 10 dg/s maneuver in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

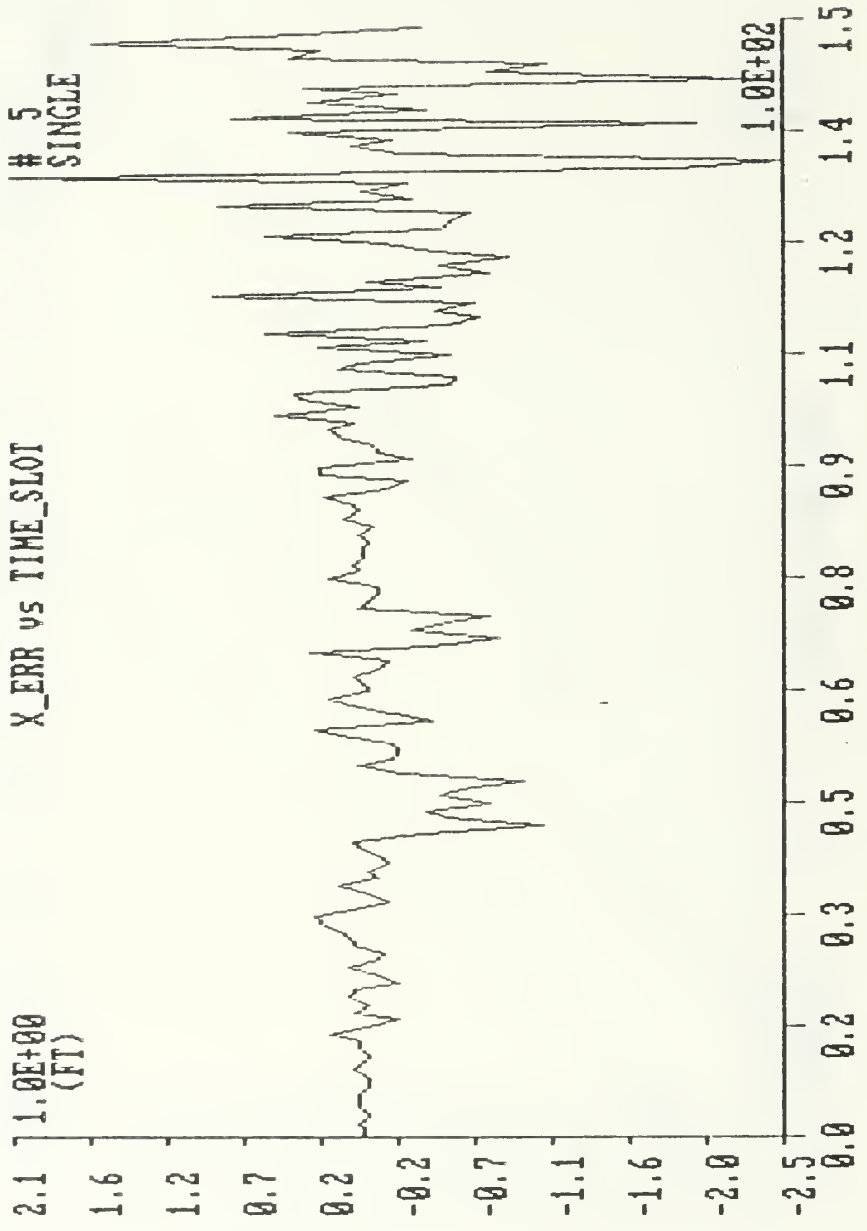


Figure 42. IBM-PC - Error in torpedo position during a 10 dg/s maneuver in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

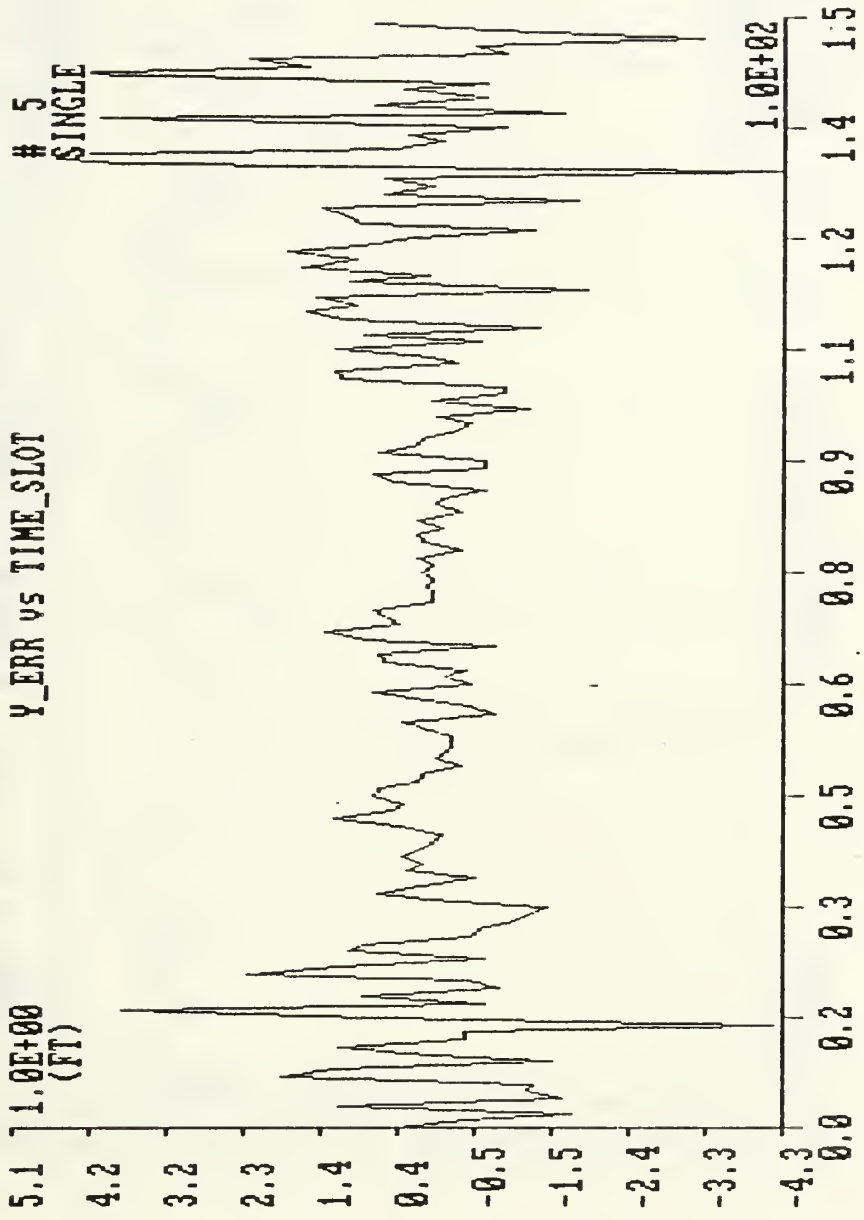


Figure 43. IBM-PC - Error in torpedo position during a 10 dg/s maneuver in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

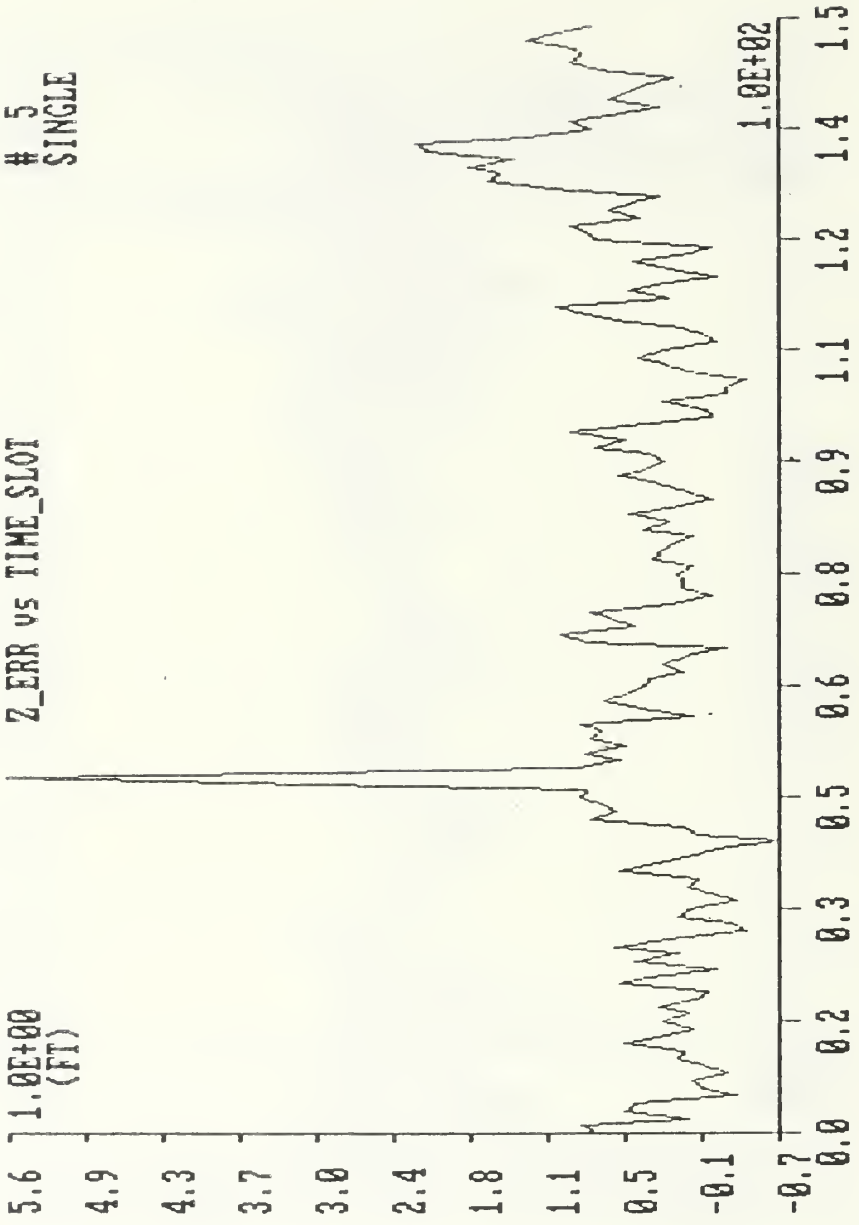


Figure 44. IBM-PC - Error in torpedo depth during a 10 dg/s maneuver in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

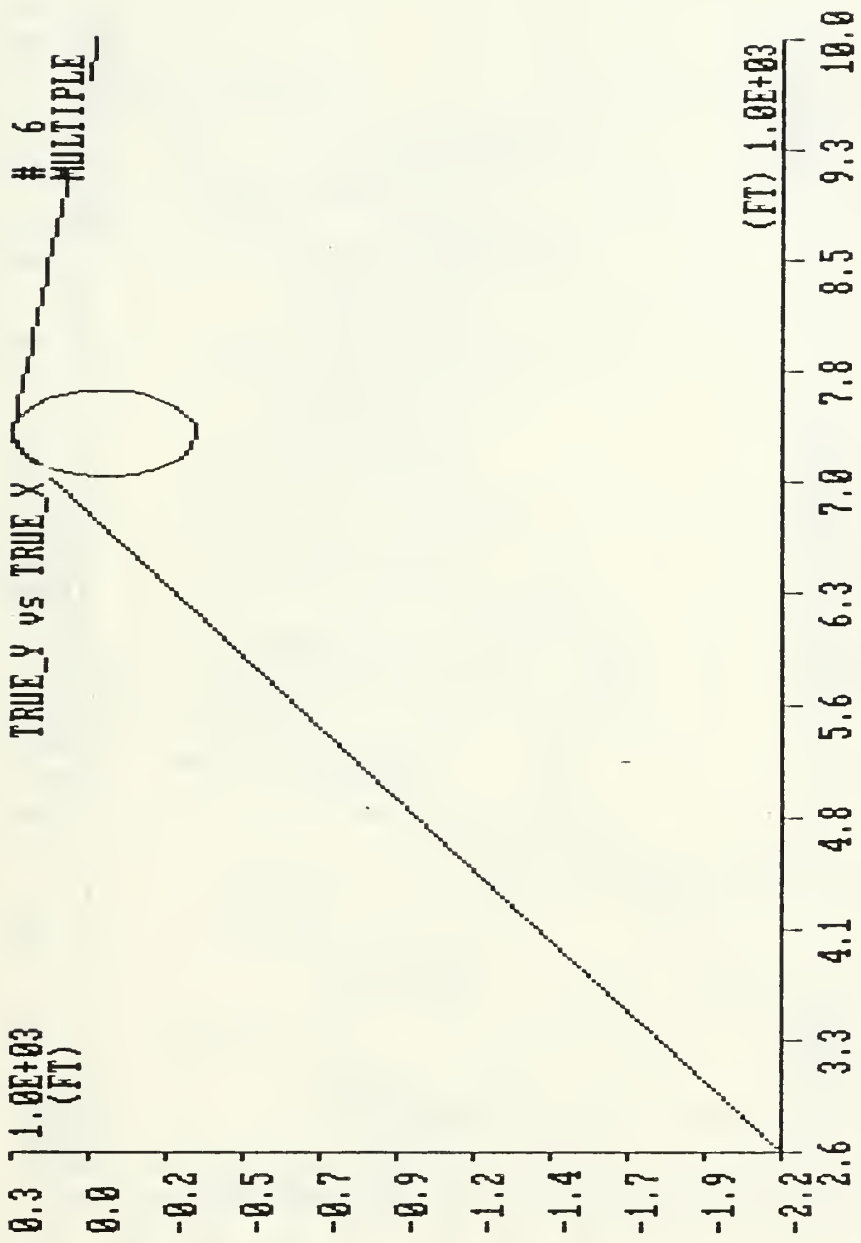


Figure 45. IBM-PC - True trajectory of the torpedo during a 10 dg/s maneuver through multiple arrays. Initial conditions: x=1000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

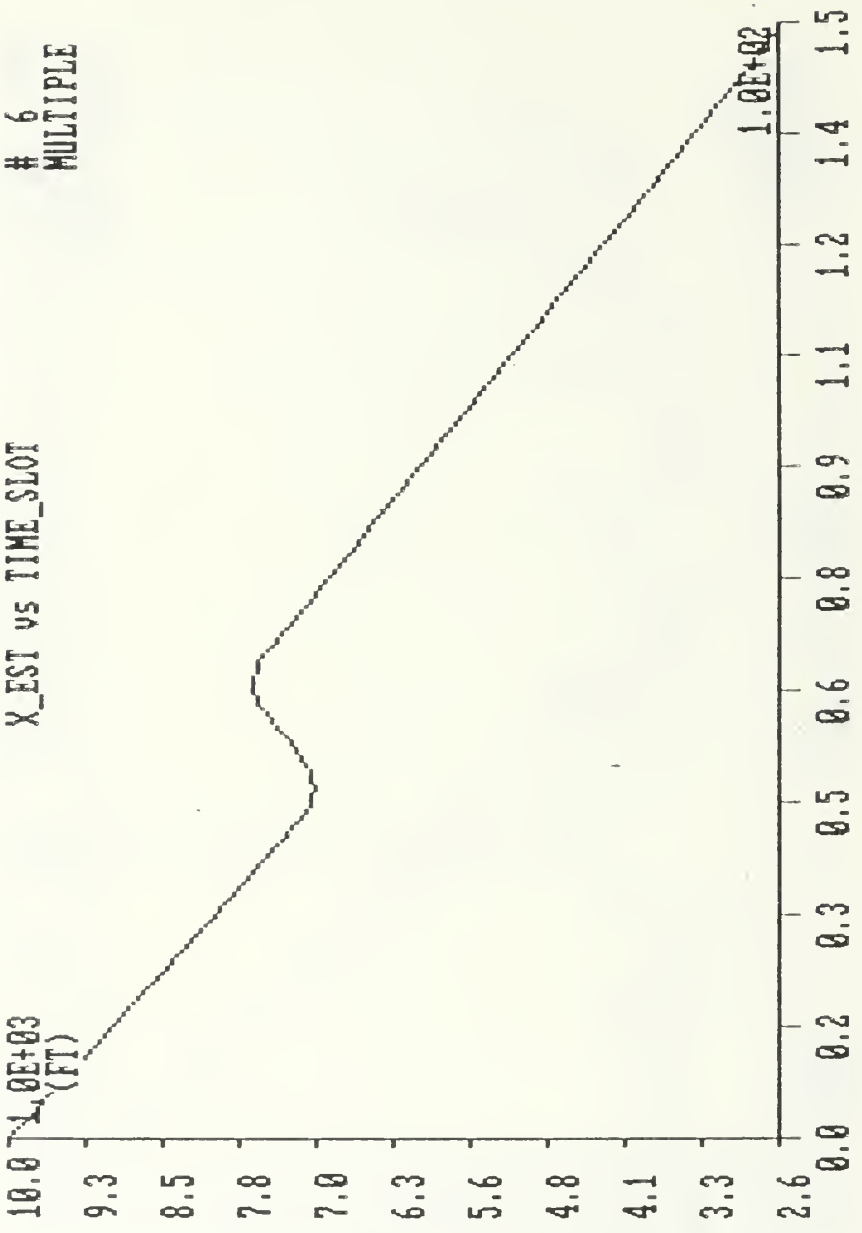


Figure 46. IBM-PC - Estimate of torpedo position during a 10 dg/s maneuver thruough multiple arrays. Initial conditions: x=1000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

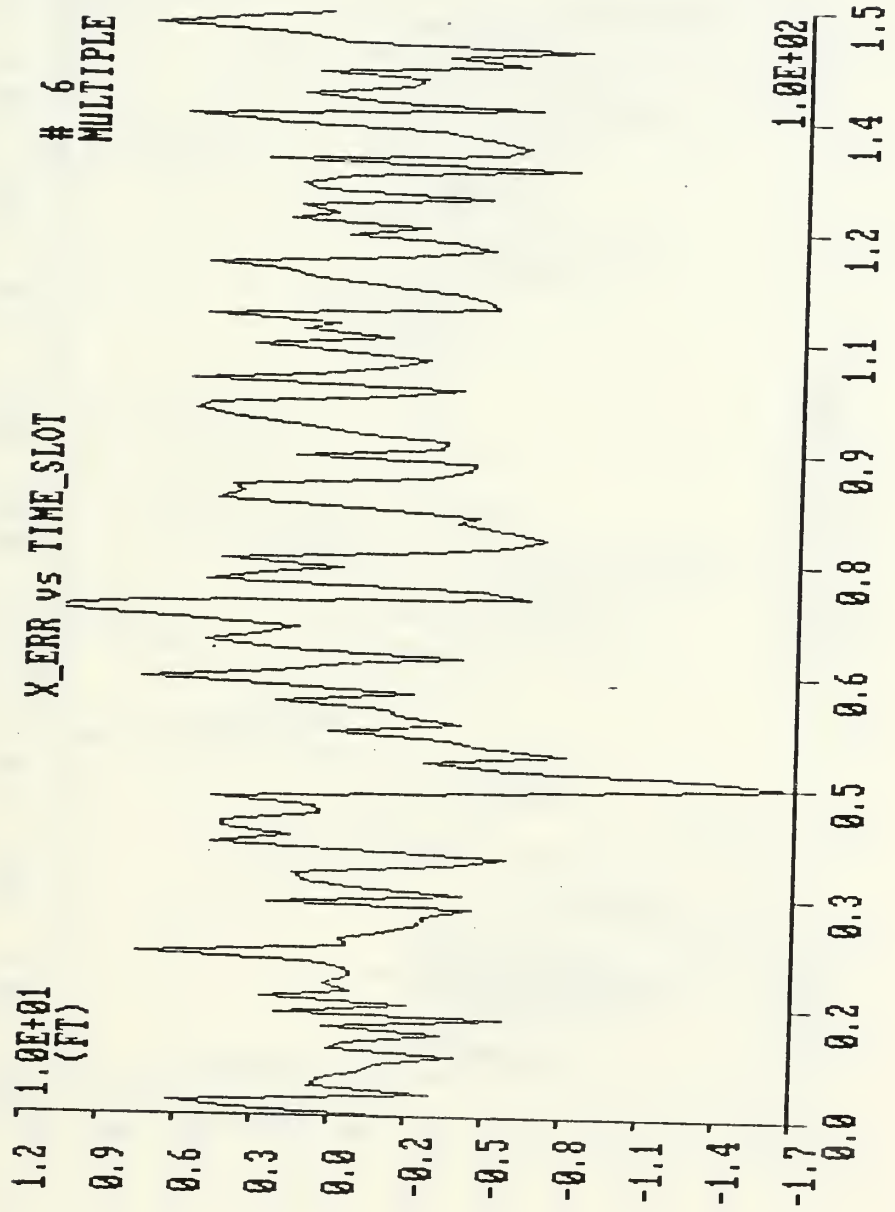


Figure 47. IBM-PC - Error in torpedo position during a 10 dg/s maneuver through multiple arrays. Initial conditions: x=1000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

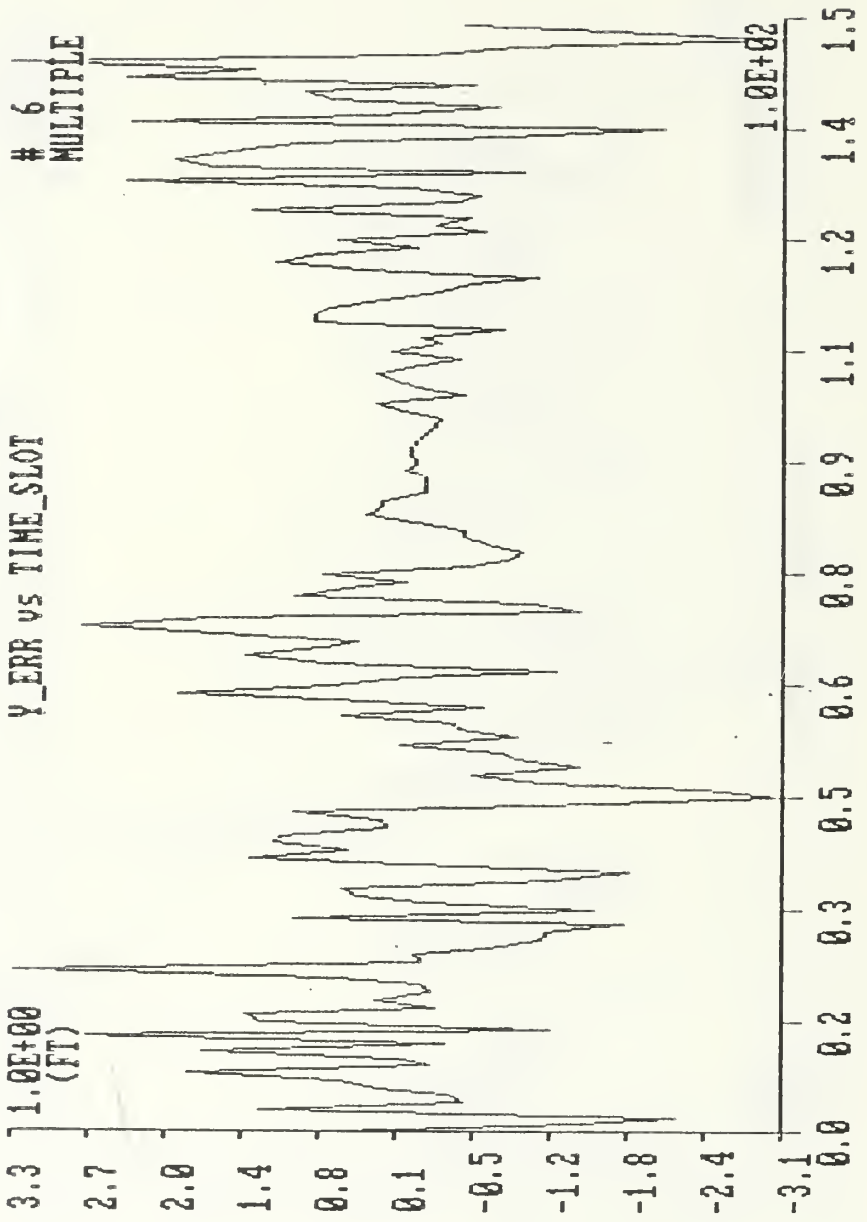


Figure 48. IBM-PC - Error in torpedo position during a 10 dg/s maneuver through multiple arrays. Initial conditions: x=1000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

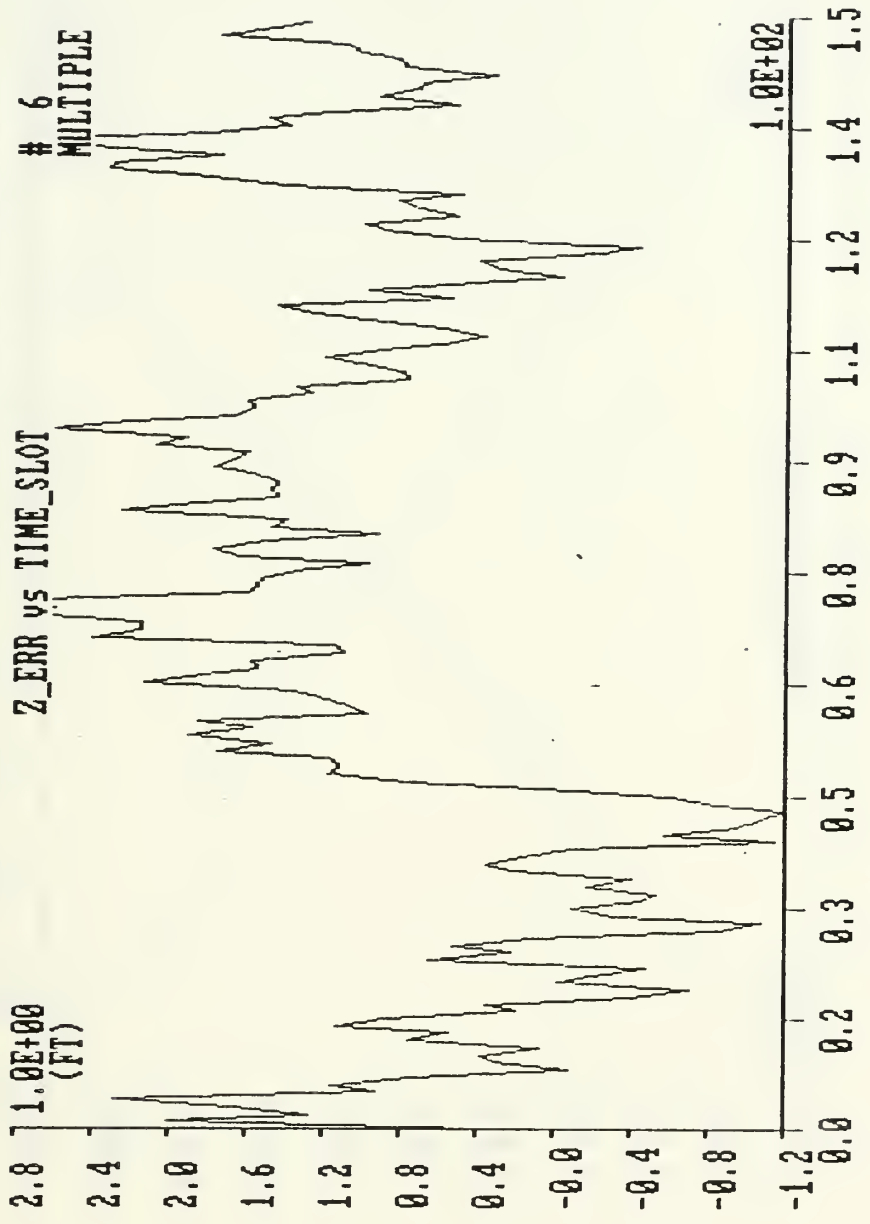


Figure 49. IBM-PC - Error in torpedo position during a 10 dg/s maneuver through multiple arrays. Initial conditions: x=1000 ft, y=0 ft, Vx=-50 ft/sec, Vy=5 ft/sec.

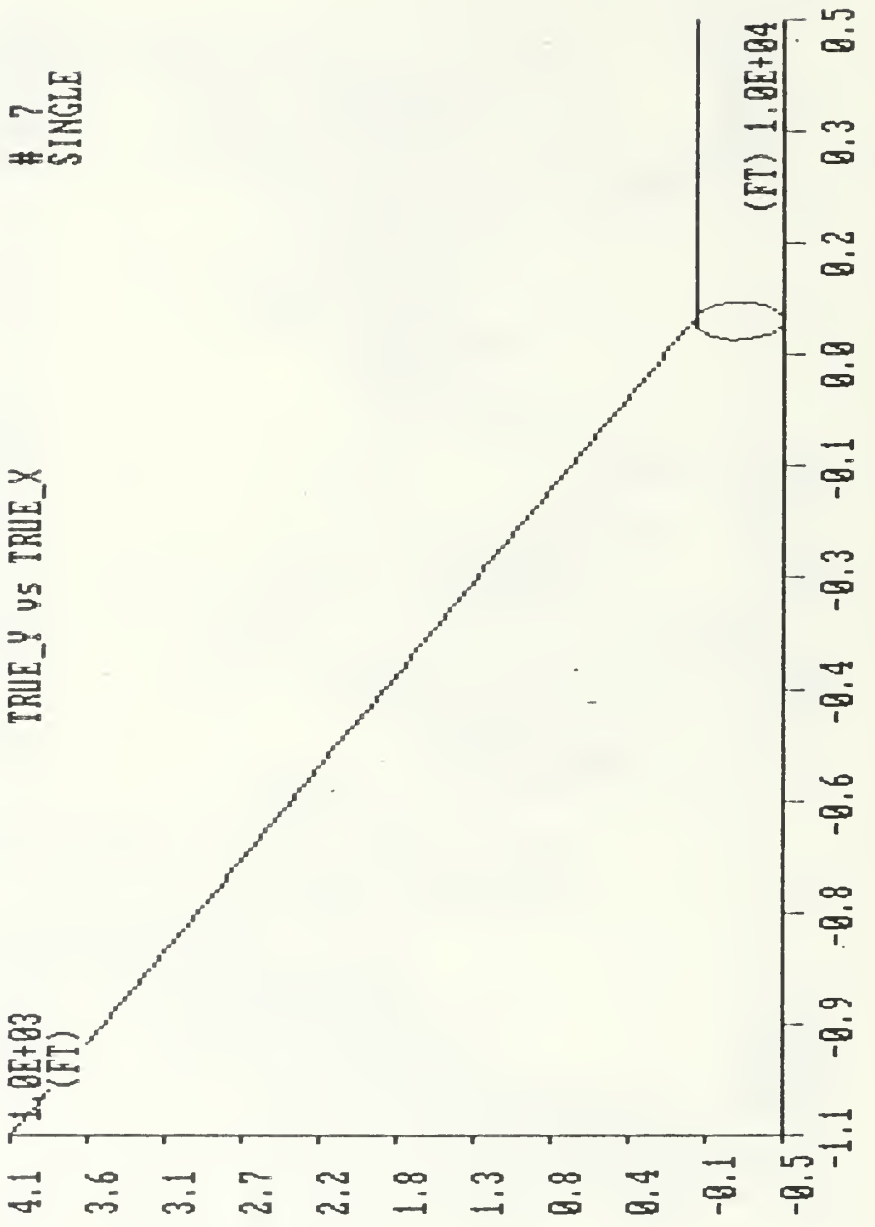


Figure 50. IBM-PC - True trajectory of the torpedo during a 20 dg/s maneuver in the area of a single array. Initial conditions: x=5000 ft, y=0 ft, Vx=-92.5 ft/sec, Vy=0 ft/sec.

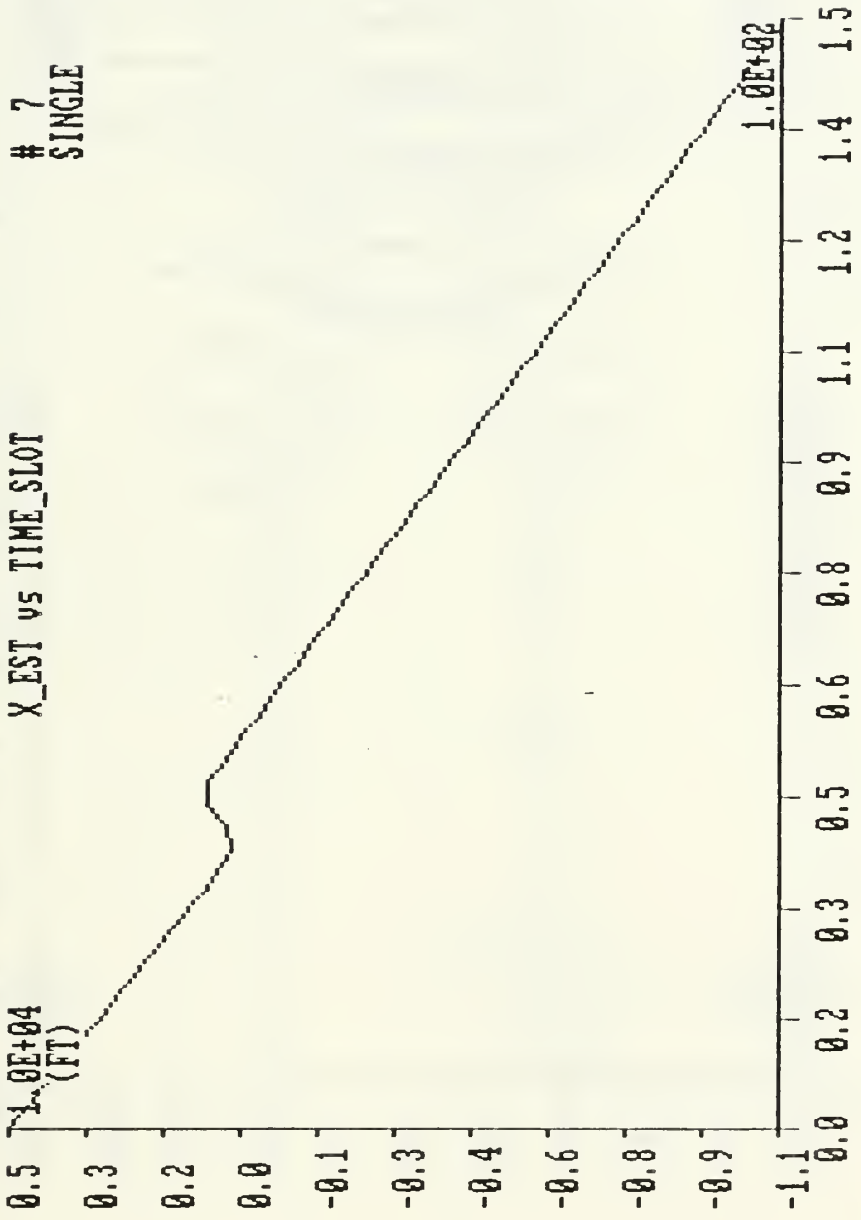


Figure 51. IBM-PC - Estimate of torpedo position during a 20 dg/s maneuver in the area of a single array. Initial conditions: x=5000 ft, y=0 ft, Vx=-92.5 ft/sec, Vy=0 ft/sec.

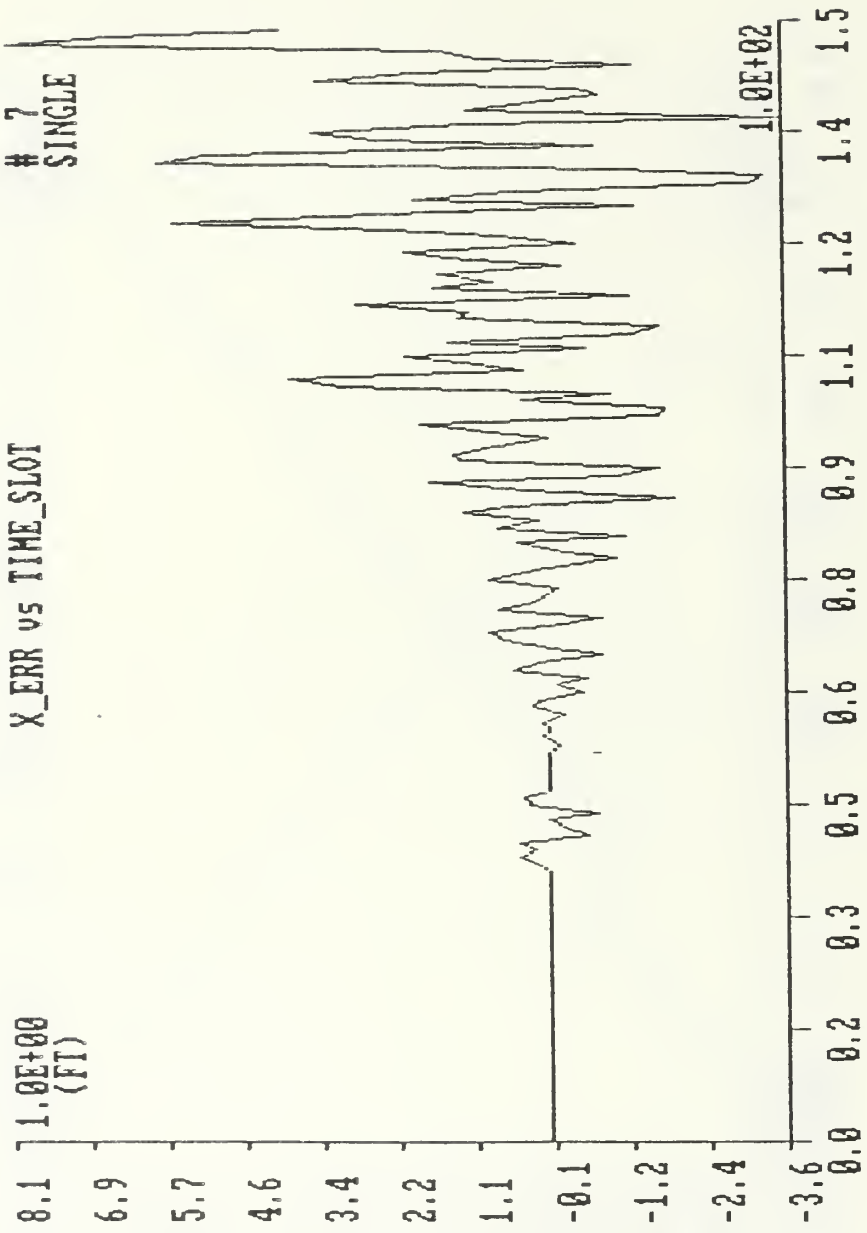


Figure 52. IBM-PC - Error in torpedo position during a 20 dg/s maneuver in the area of a single array. Initial conditions: x=5000 ft, y=0 ft, Vx=-92.5 ft/sec, Vy=0 ft/sec.

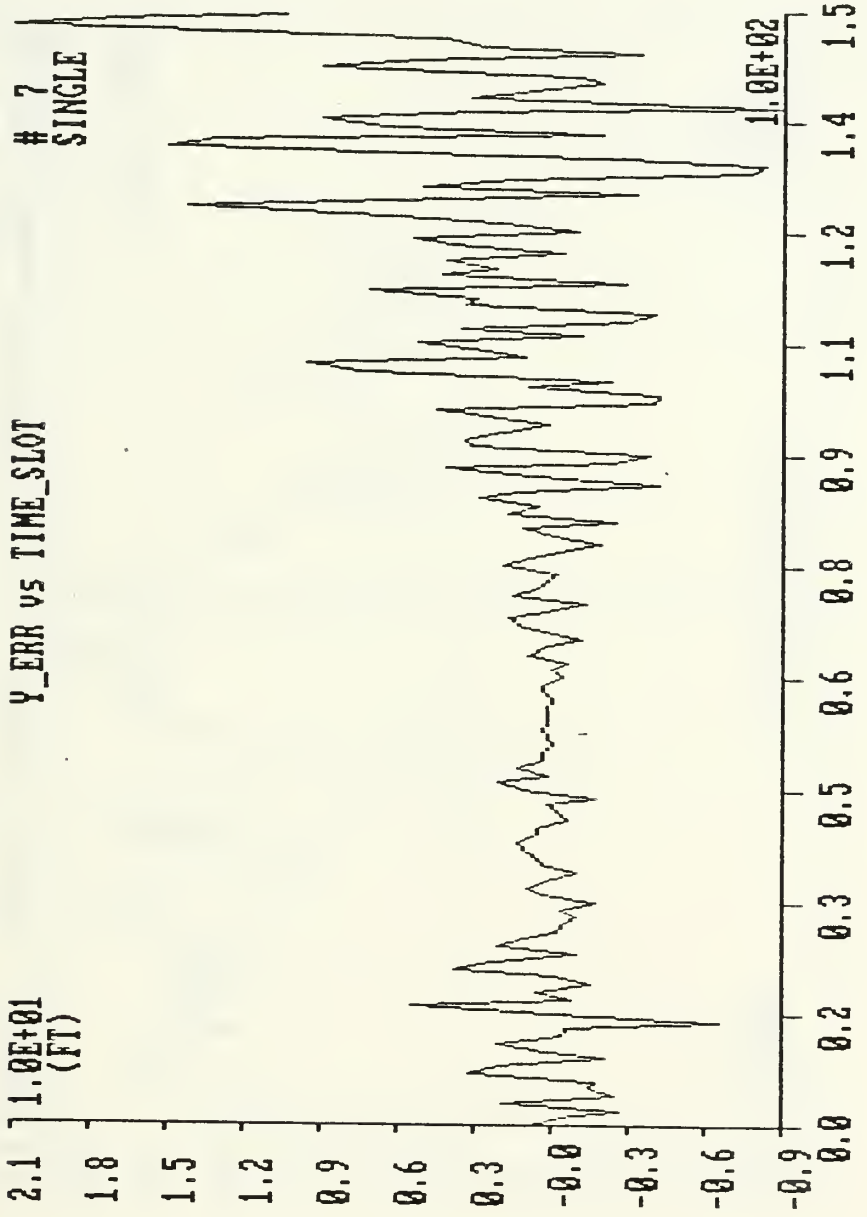


Figure 53. IBM-PC - Error in torpedo position during a 20 dg/s maneuver in the area of a single array. Initial conditions: x=5000 ft, y=0 ft, Vx=-92.5 ft/sec, Vy=0 ft/sec.

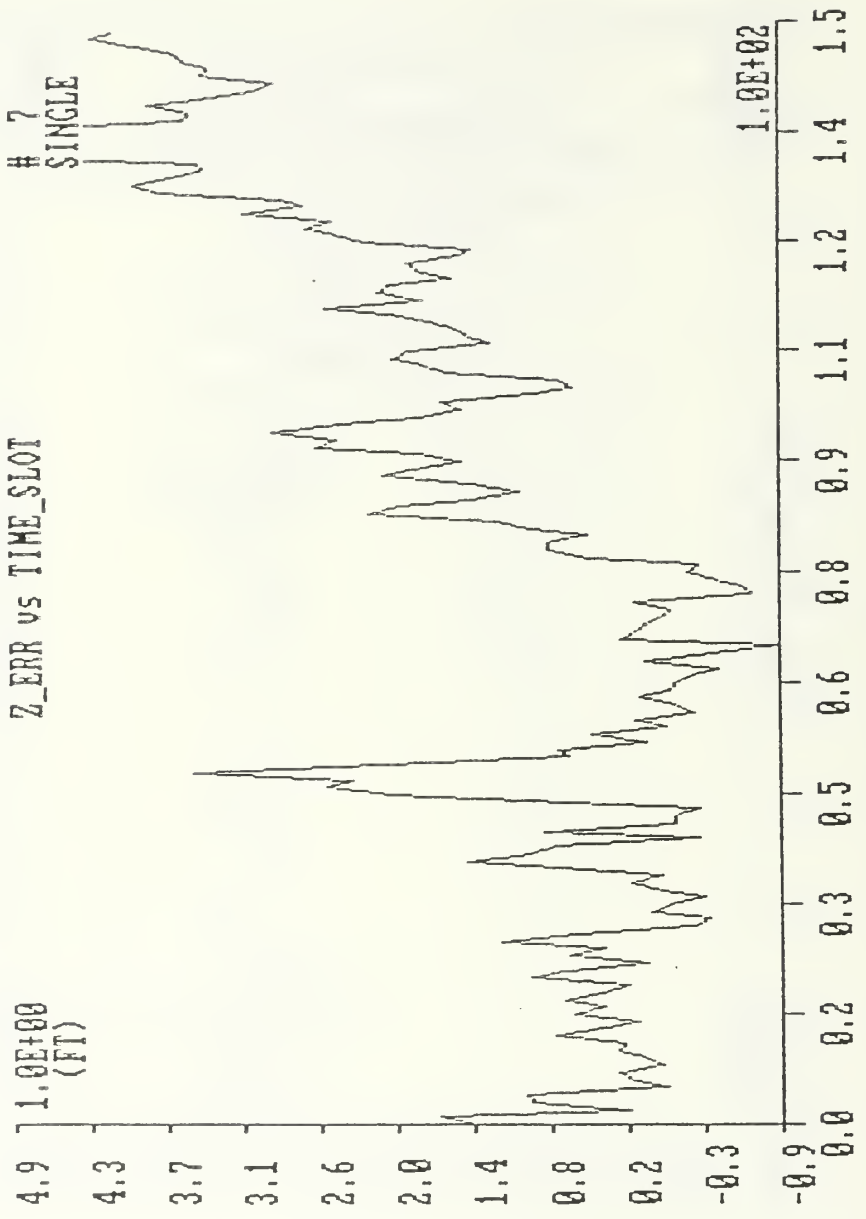


Figure 54. IBM-PC - Error in torpedo depth during a 20 dg/s maneuver in the area of a single array. Initial conditions: x=5000 ft, y=0 ft, Vx=-92.5 ft/sec, Vy=0 ft/sec.

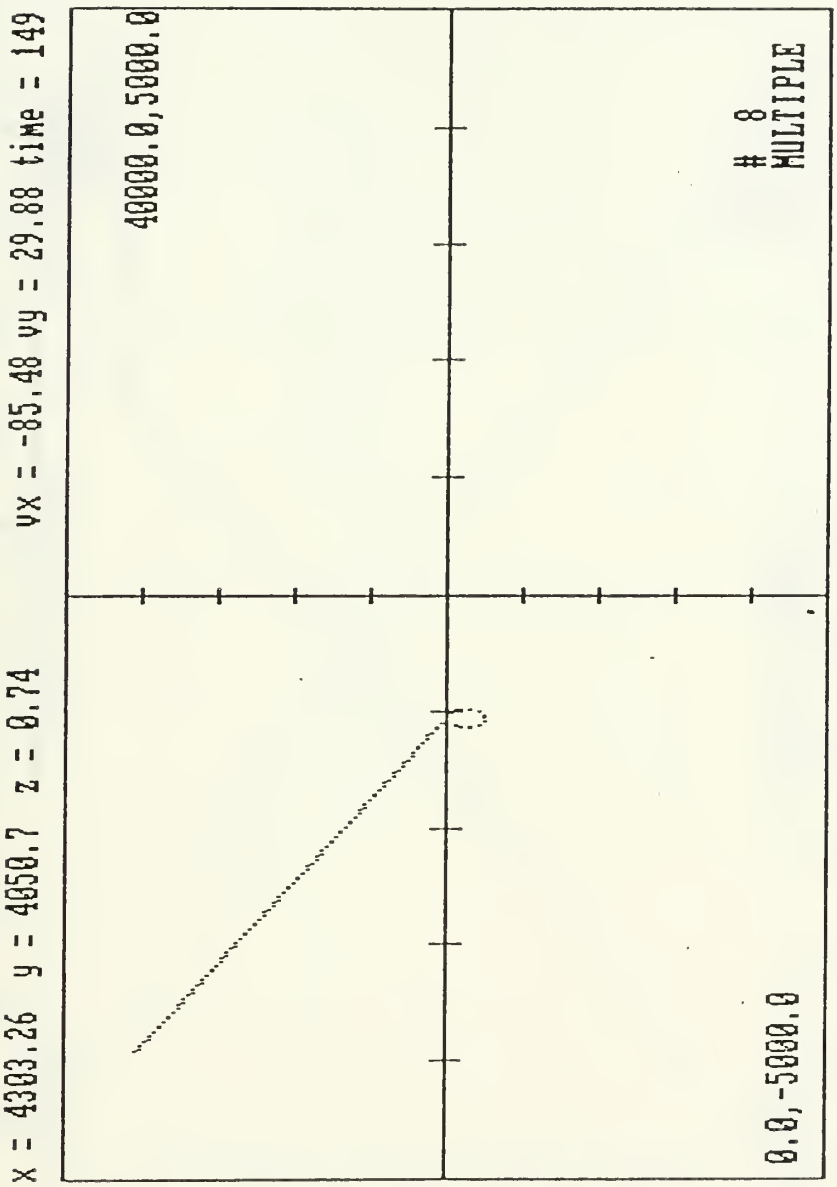


Figure 55. IBM-PC - Real time plot of estimated trajectory of the torpedo through multiple arrays. Initial conditions: x=20000 ft, y=0 ft, Vx=-92.5 ft/sec, Vy=0 ft/sec.

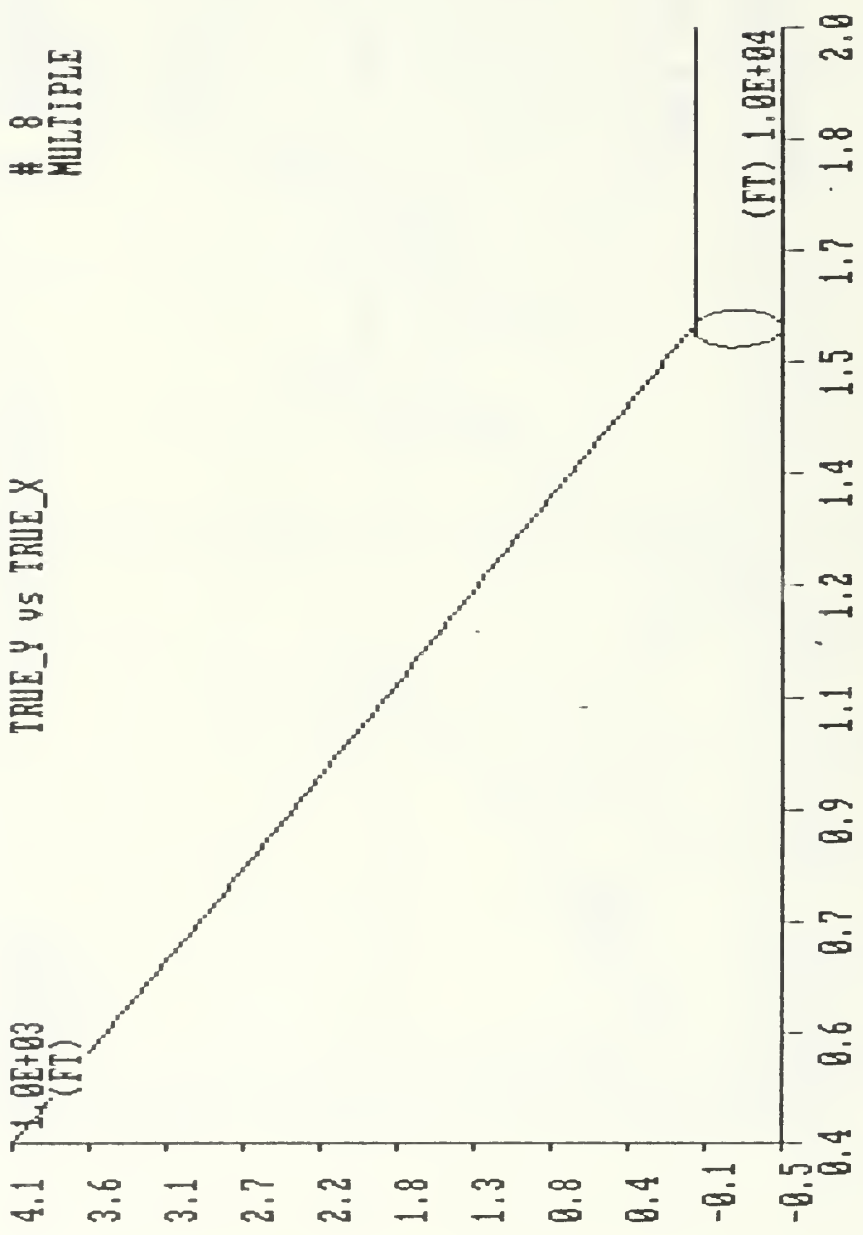


Figure 56. IBM-PC - True trajectory of the torpedo in the during a 20 dg/s maneuver through multiple arrays. Initial conditions: x=20000 ft, y=0 ft, Vx=-92.5 ft/s, Vy=0 ft/s.

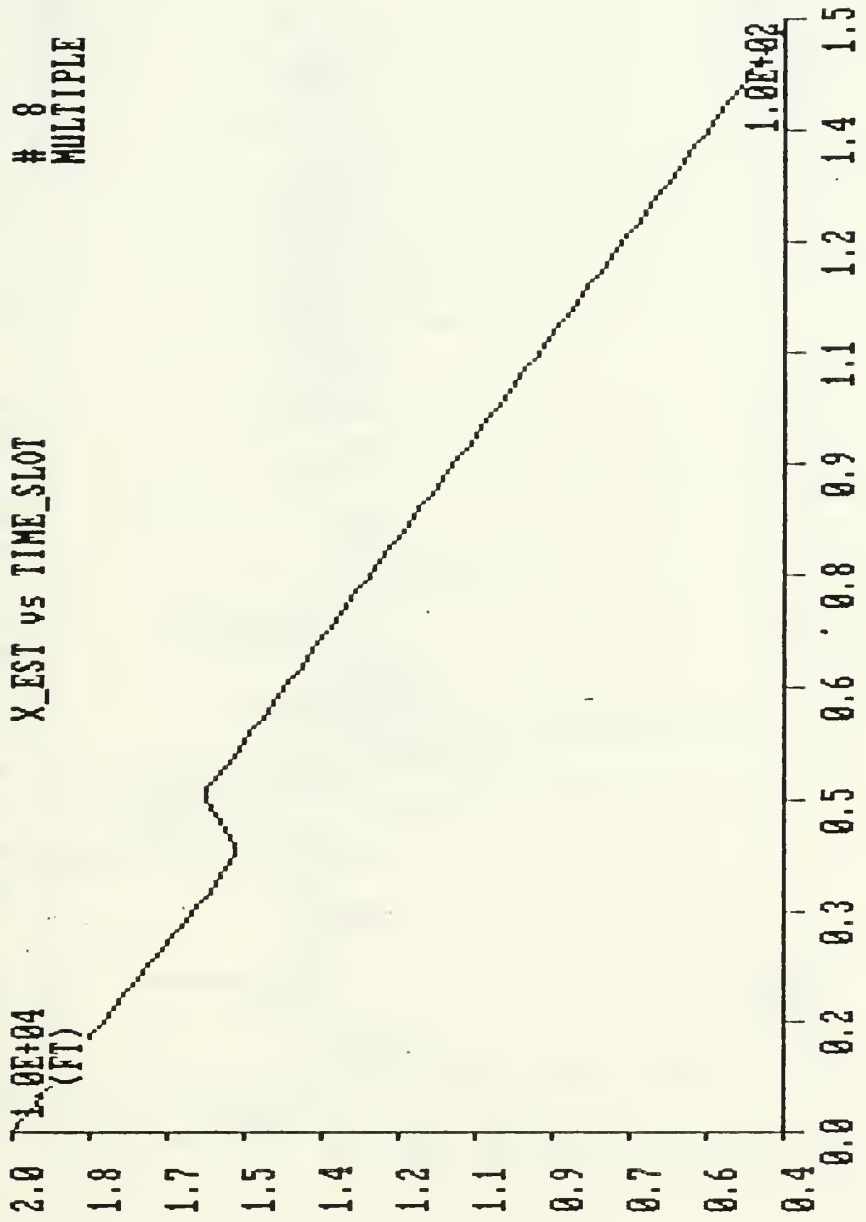


Figure 57. IBM-PC - Estimate of torpedo position during a 20 dg/s maneuver through multiple arrays. Initial conditions: x=20000 ft, y=0 ft, Vx=-92.5 ft/s, Vy=0 ft/s.

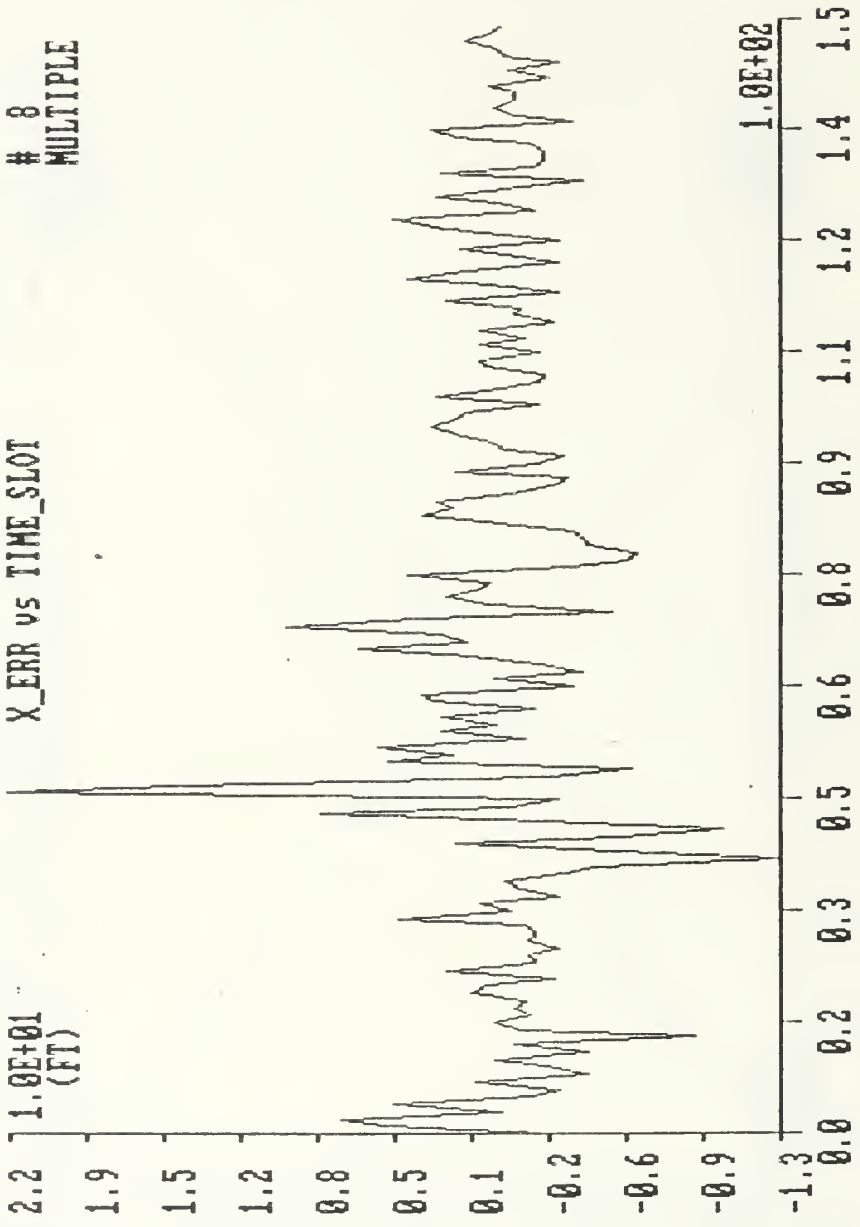


Figure 58. IBM-PC - Error in torpedo position during a 20 dg/s maneuver through multiple arrays. Initial conditions: x=20000 ft, y=0 ft, Vx=-92.5 ft/s, Vy=0 ft/s.

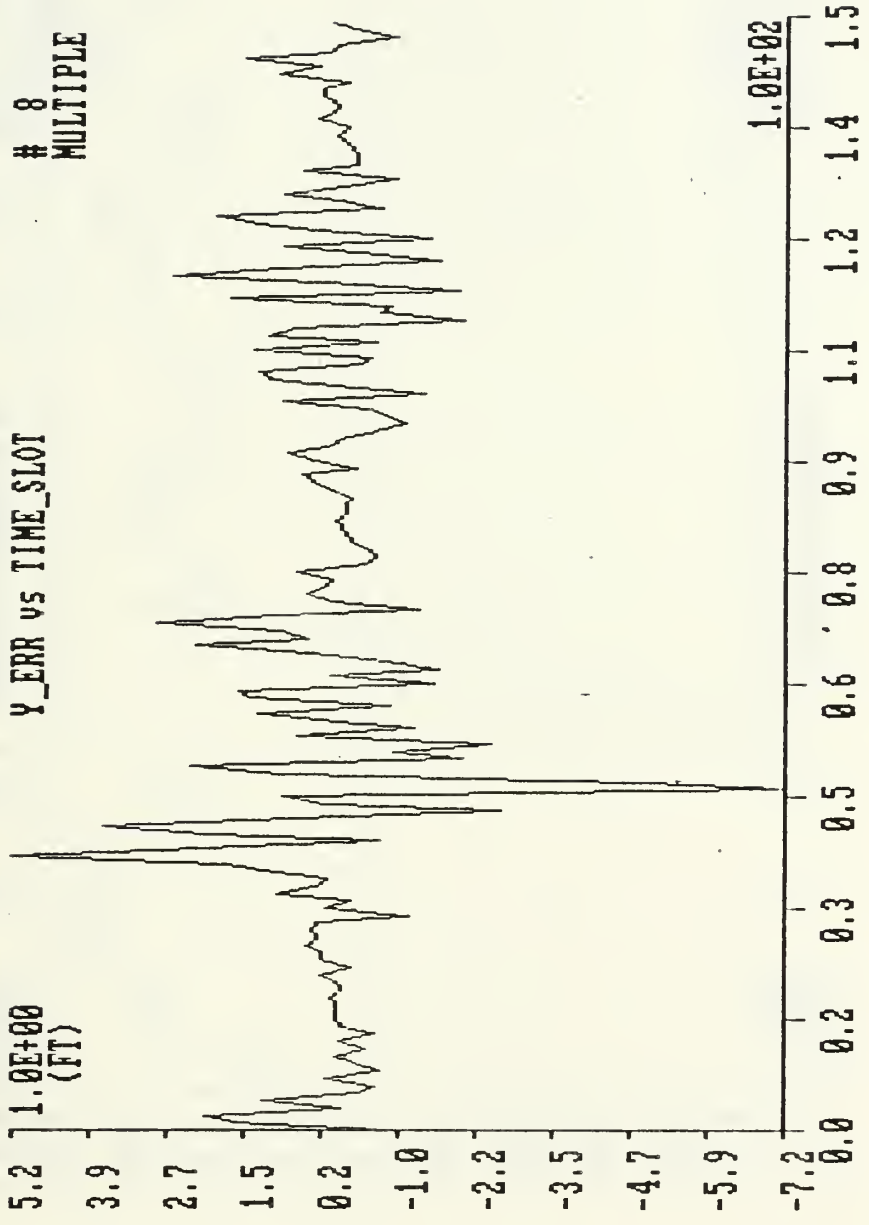


Figure 59. IBM-PC - Error in torpedo position during a 20 dg/s maneuver through multiple arrays. Initial conditions: x=20000 ft, y=0 ft, Vx=-92.5 ft/s, Vy=0 ft/s.

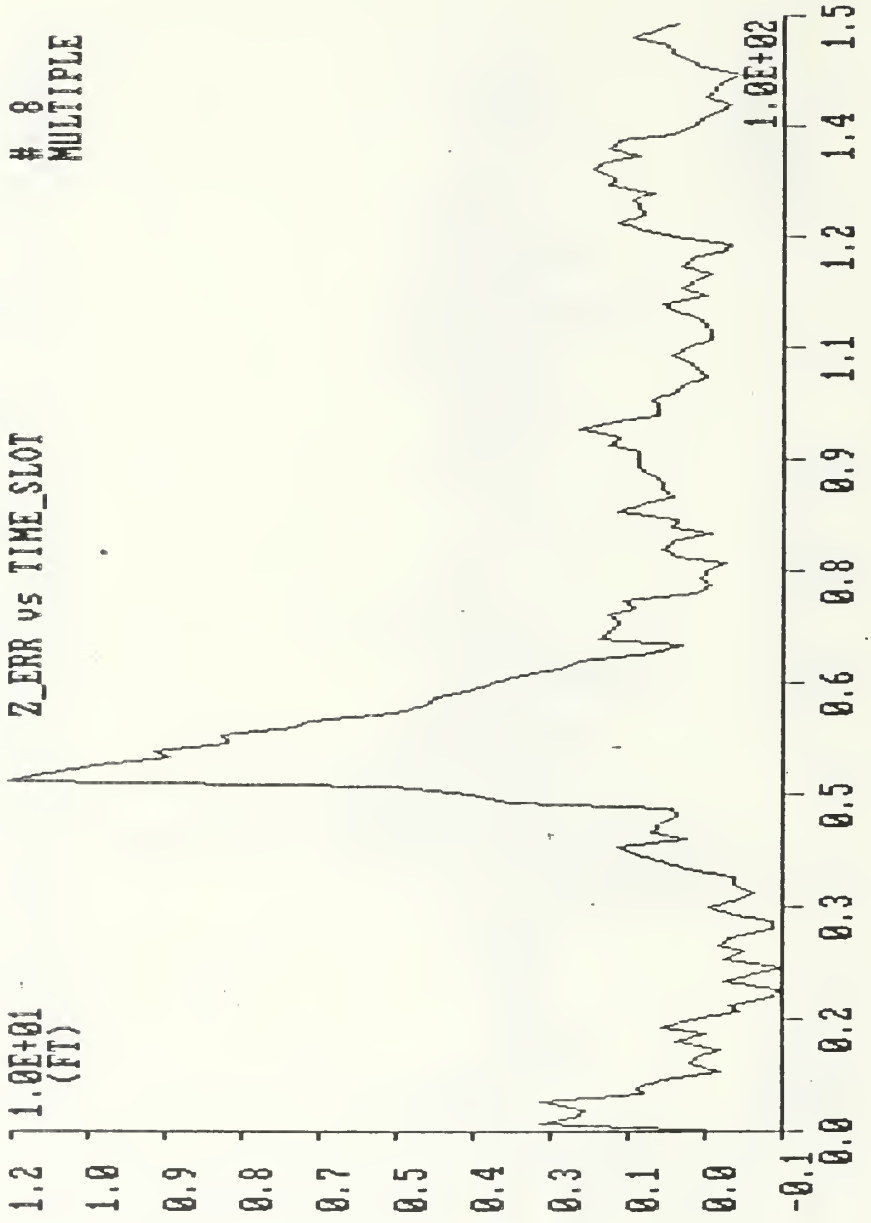


Figure 60. IBM-PC - Error in torpedo position during a 20 dg/s maneuver through multiple arrays. Initial conditions: x=20000 ft, y=0 ft, Vx=-92.5 ft/s, Vy=0 ft/s.

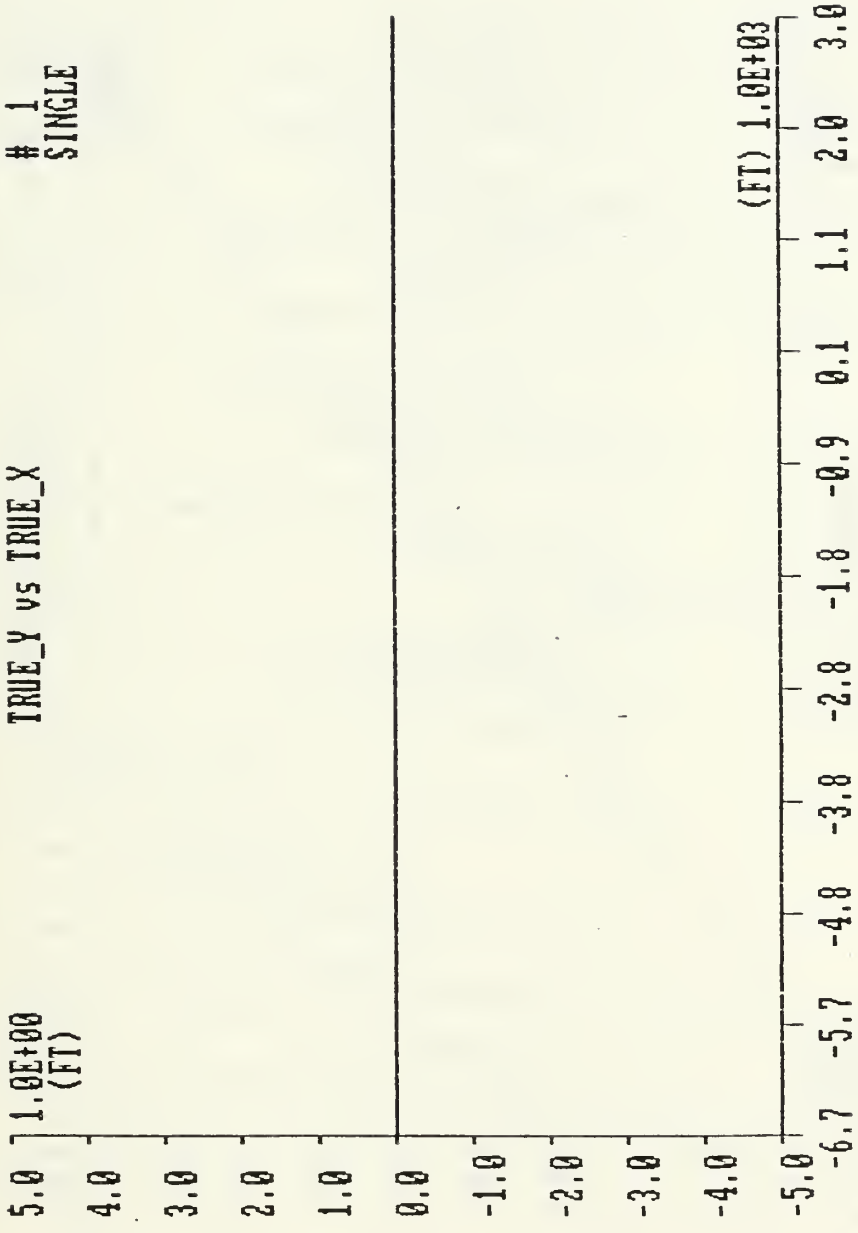


Figure 61. CIE - True trajectory of the torpedo in the area of a single array. Initial conditions : x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

Figure 62. CIE - Estimate of torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

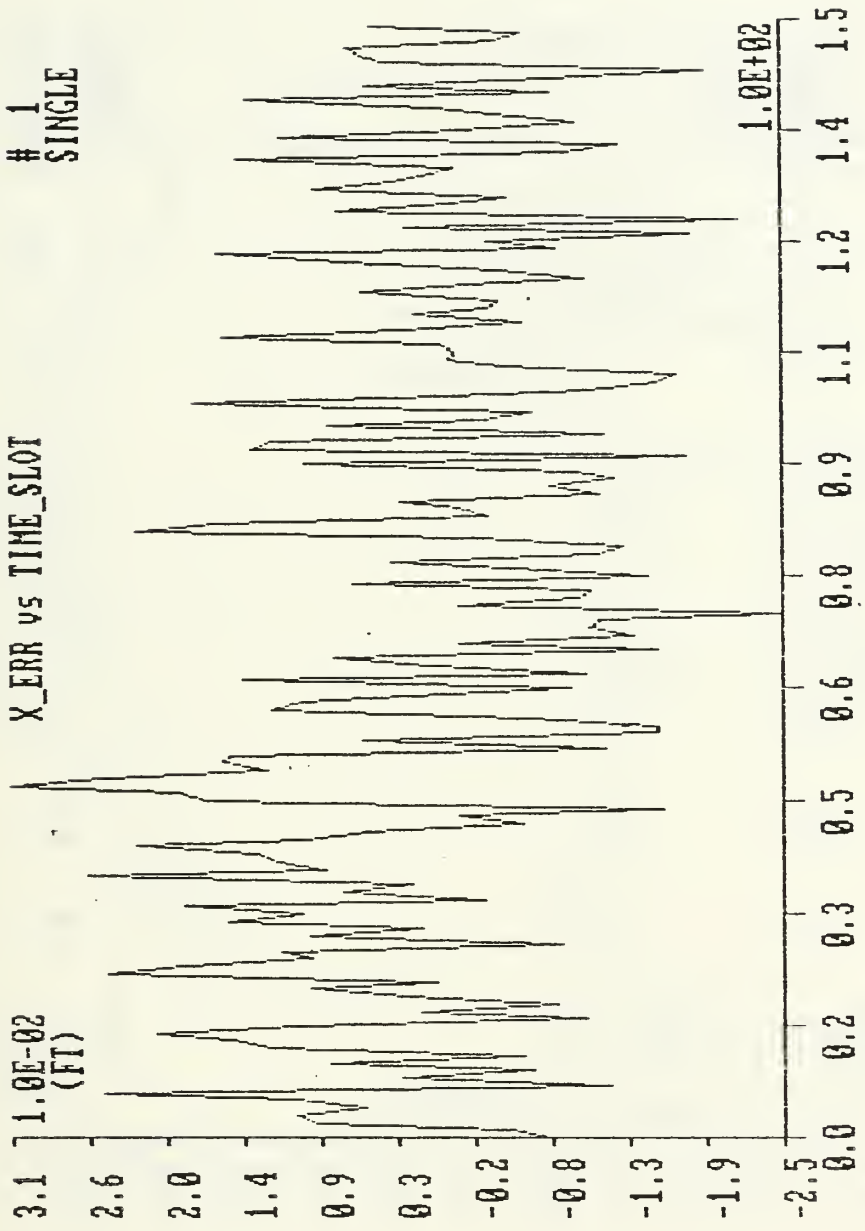


Figure 63. CIE - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

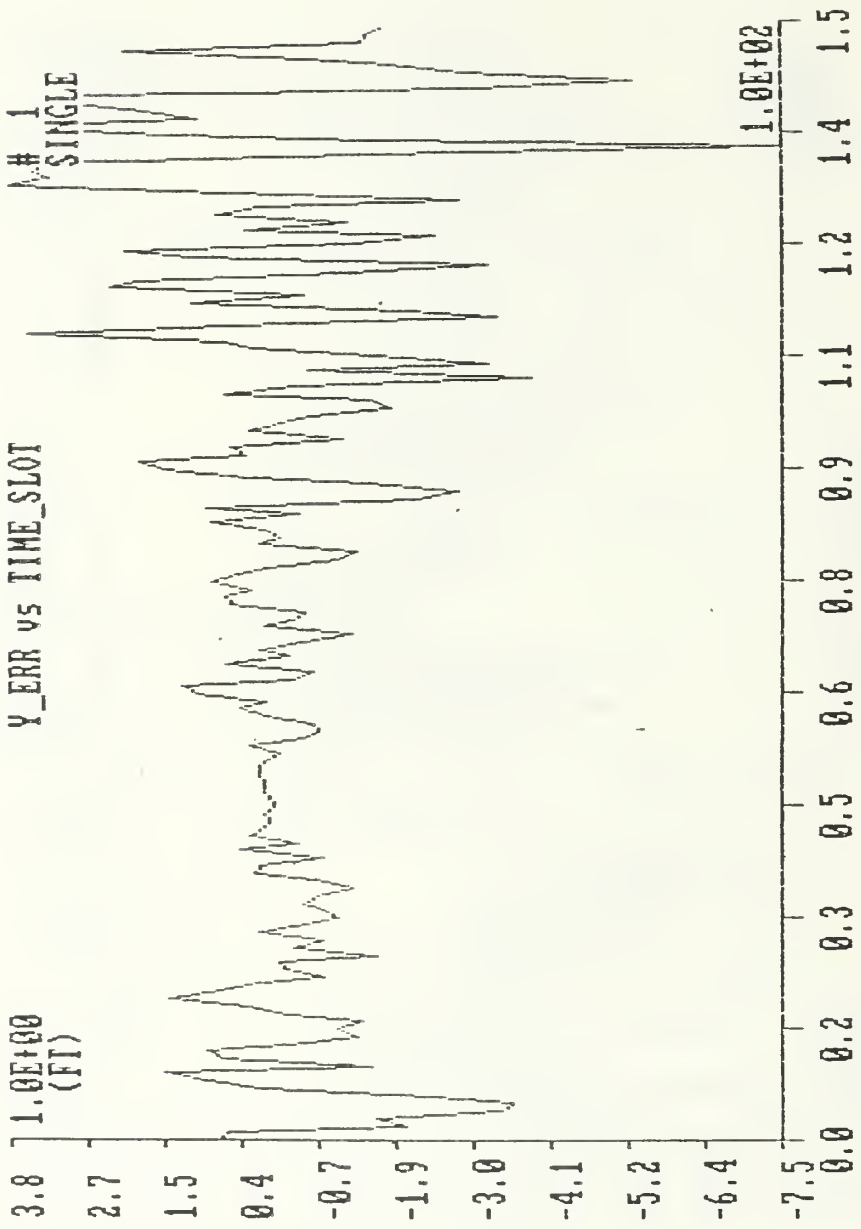


Figure 64. CIE - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

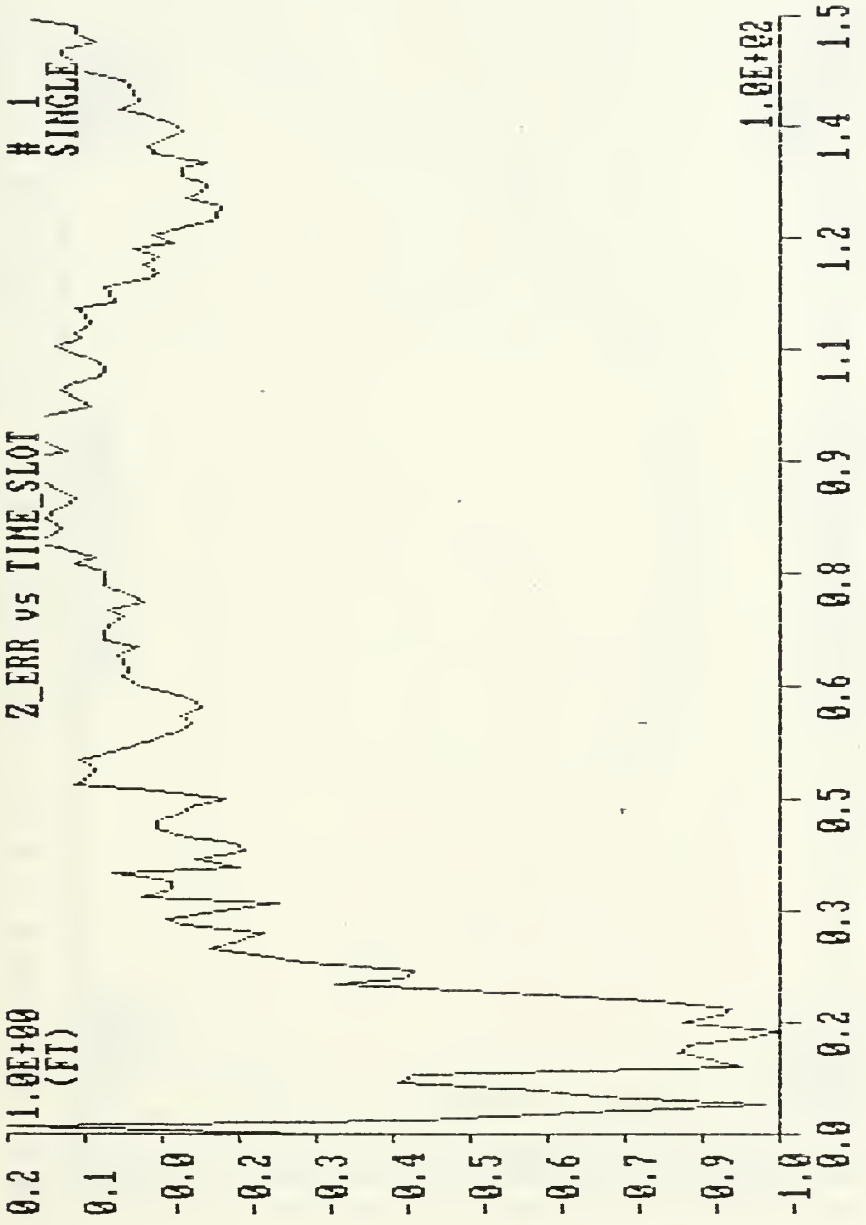


Figure 65. CIE - Error in torpedo depth during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

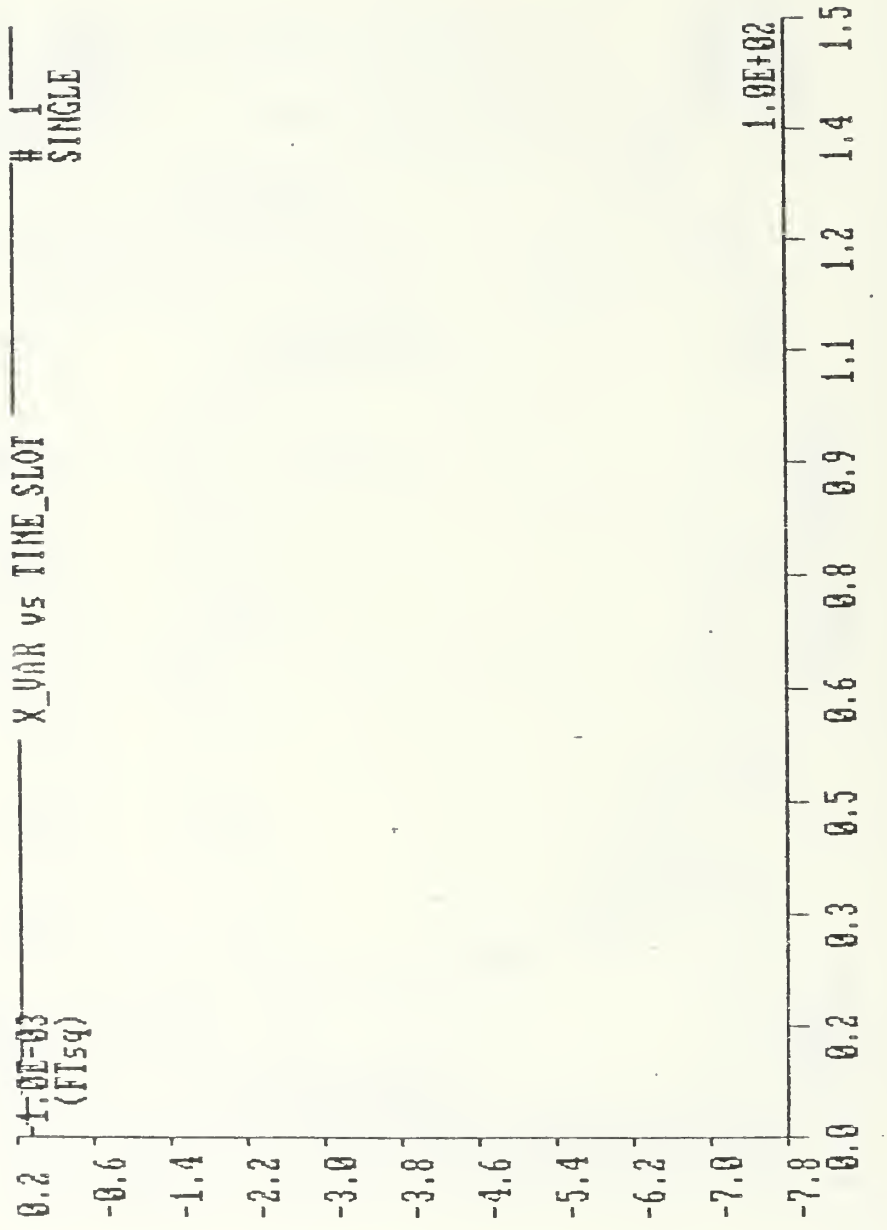


Figure 66. CIE - Variance of position error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

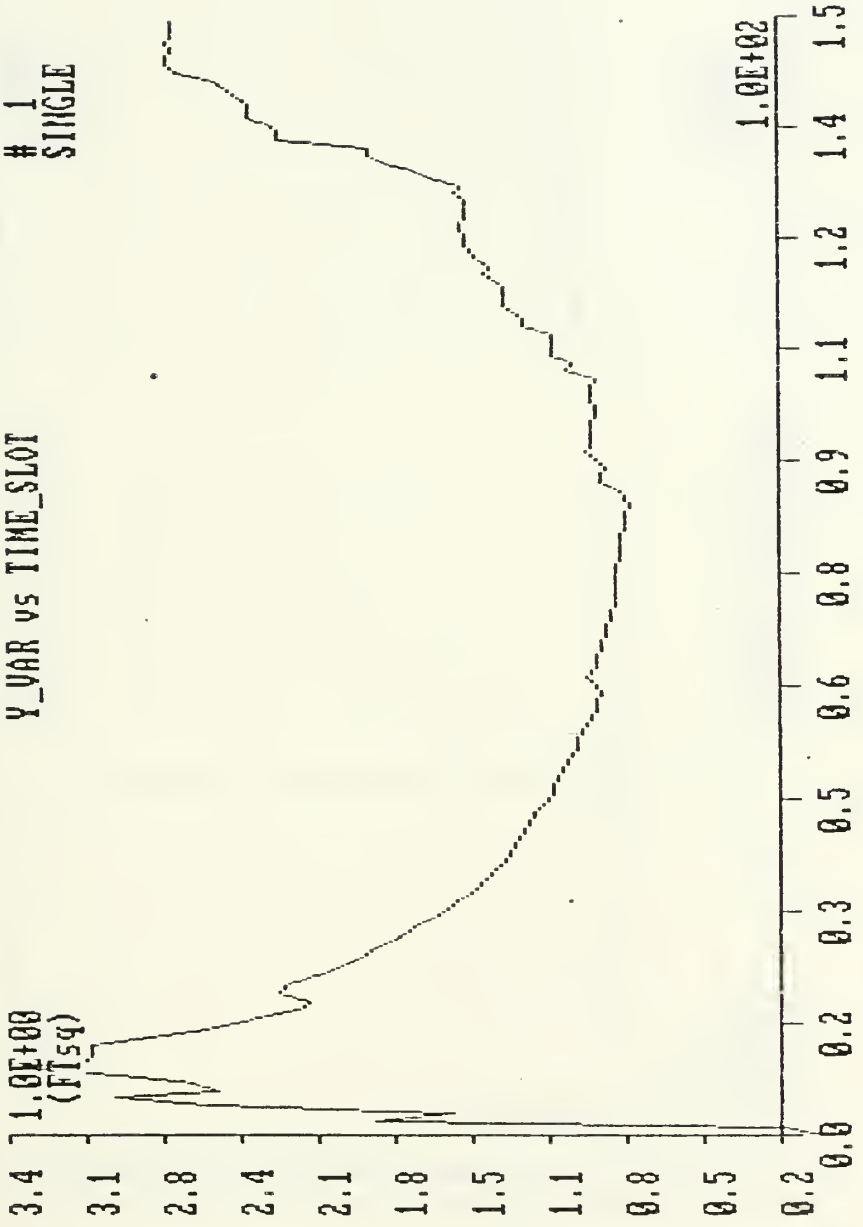


Figure 67. CIE - Variance of position error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

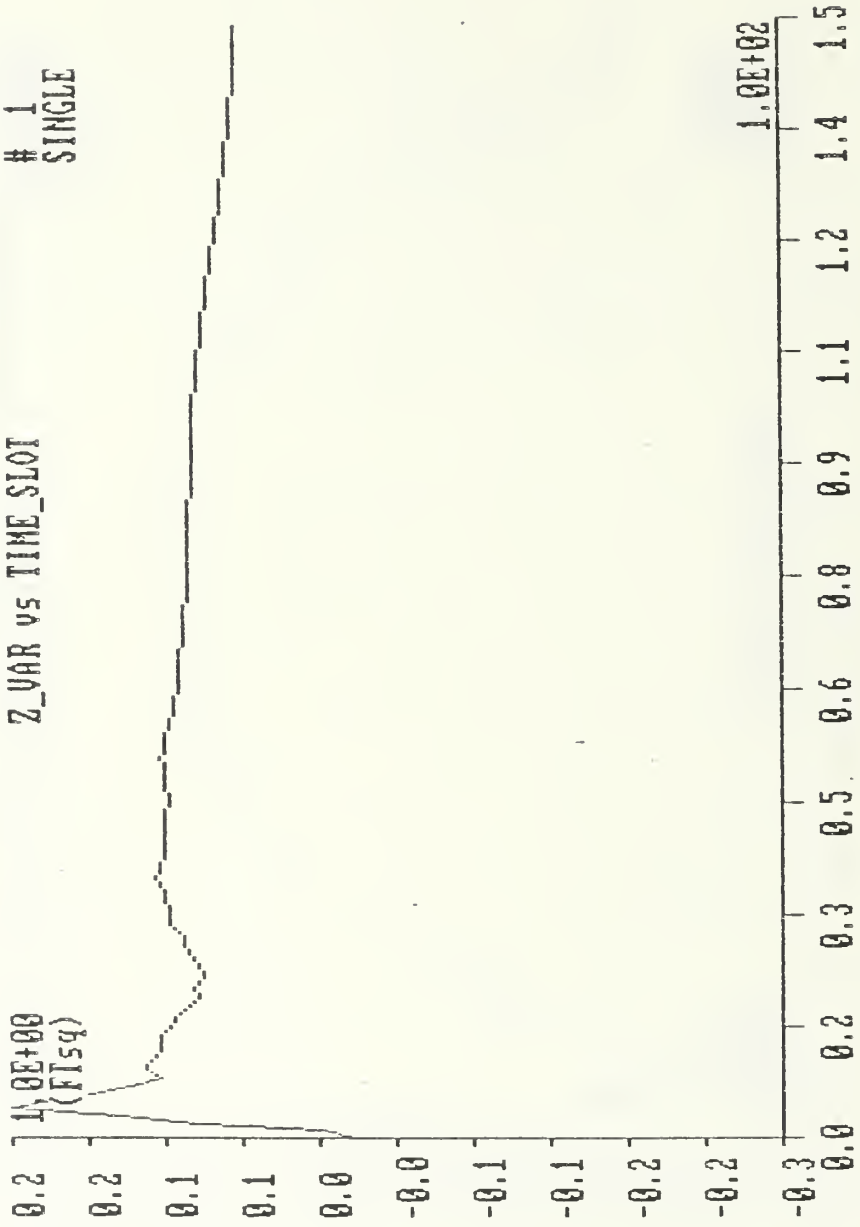


Figure 68. CIE - Variance of depth error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

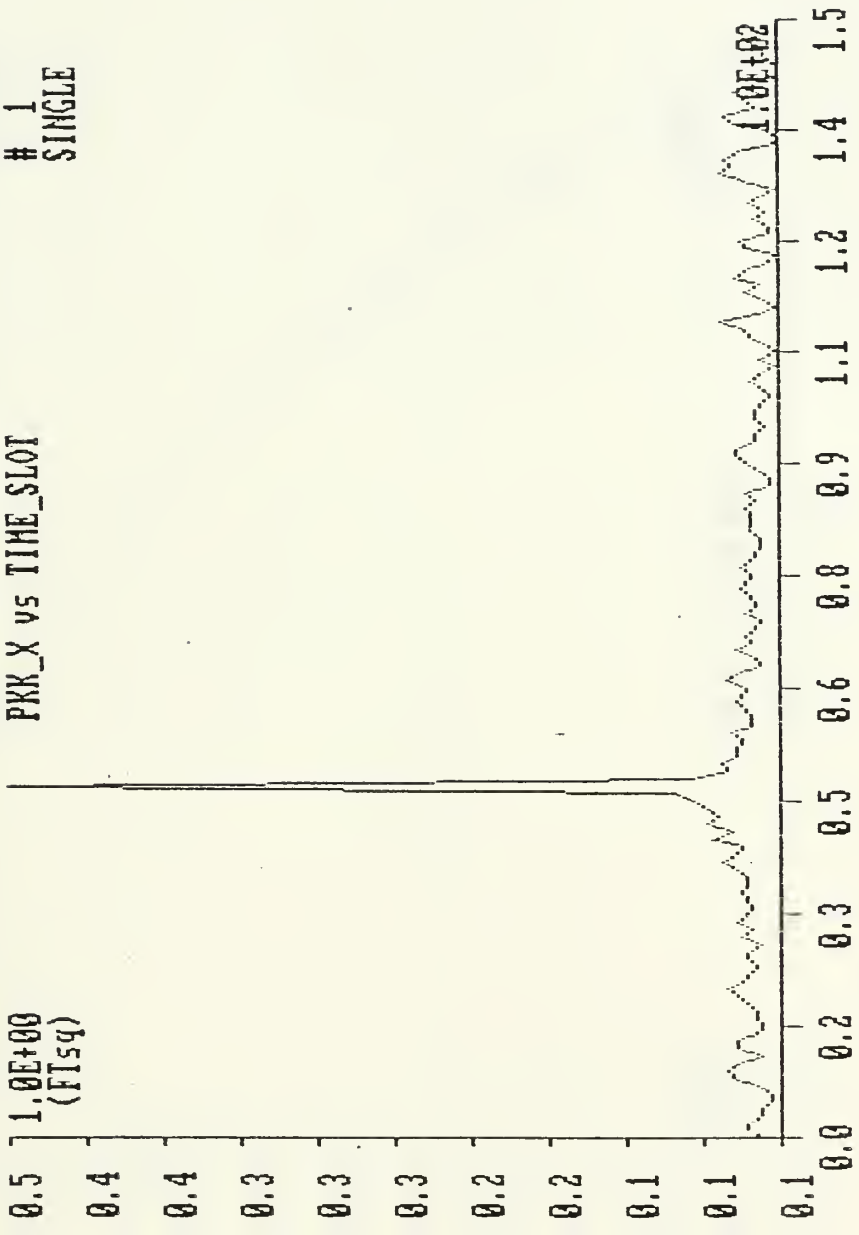


Figure 69. CIE - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

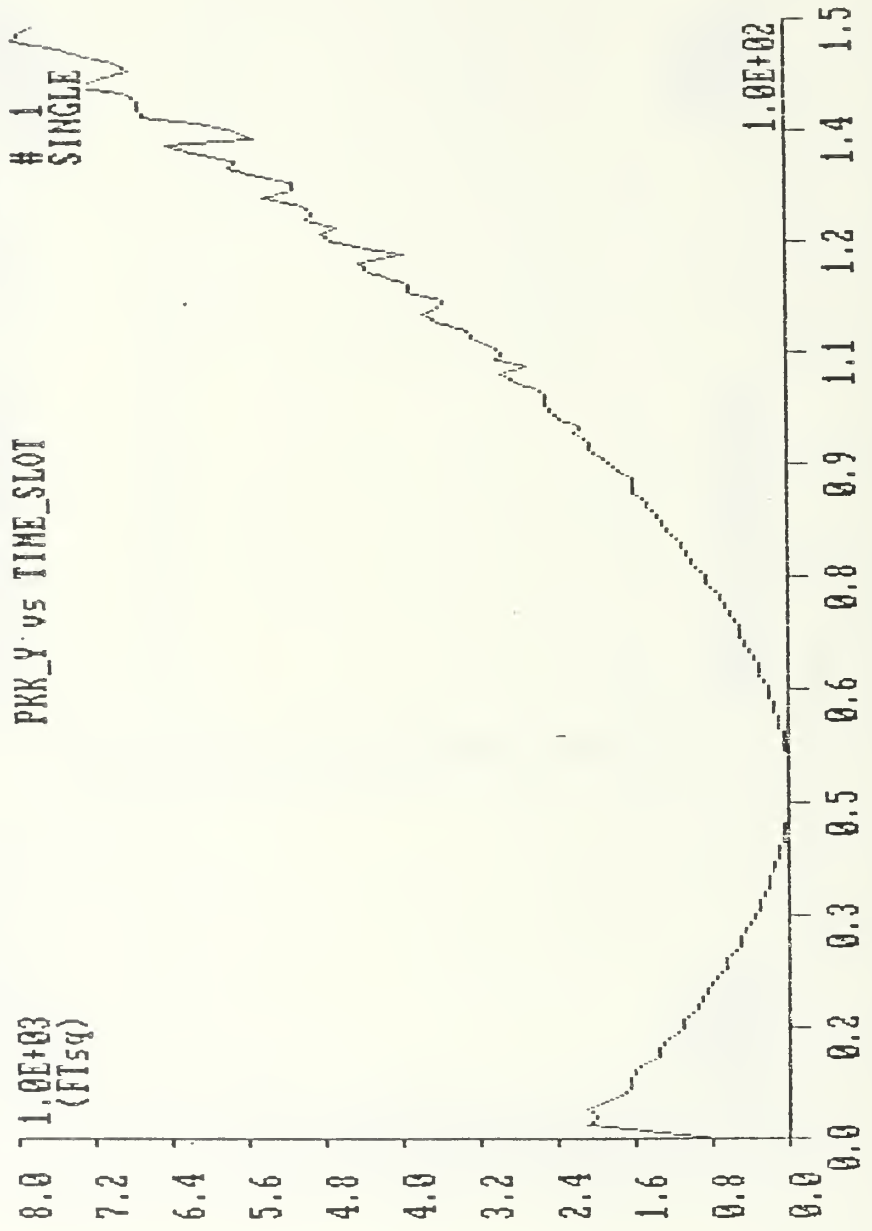


Figure 70. CIE - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

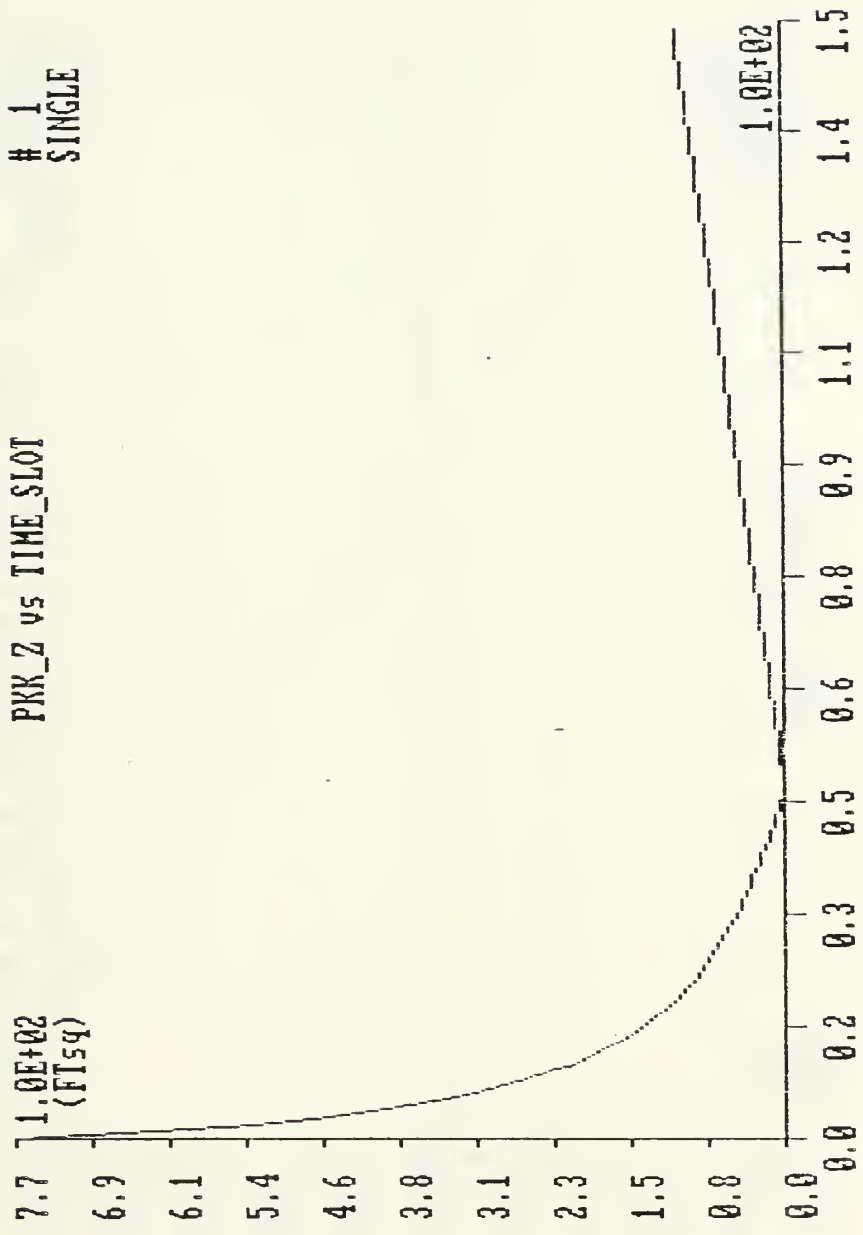


Figure 71. CIE - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=0 ft, Vx=-50 ft/sec, Vy=0 ft/sec.

Figure 72. CIE - True trajectory of the torpedo in the area of a single array. Initial conditions : x=3000 ft, y=2121 ft, Vx=-33.8 ft/sec, Vy=0 ft/sec.

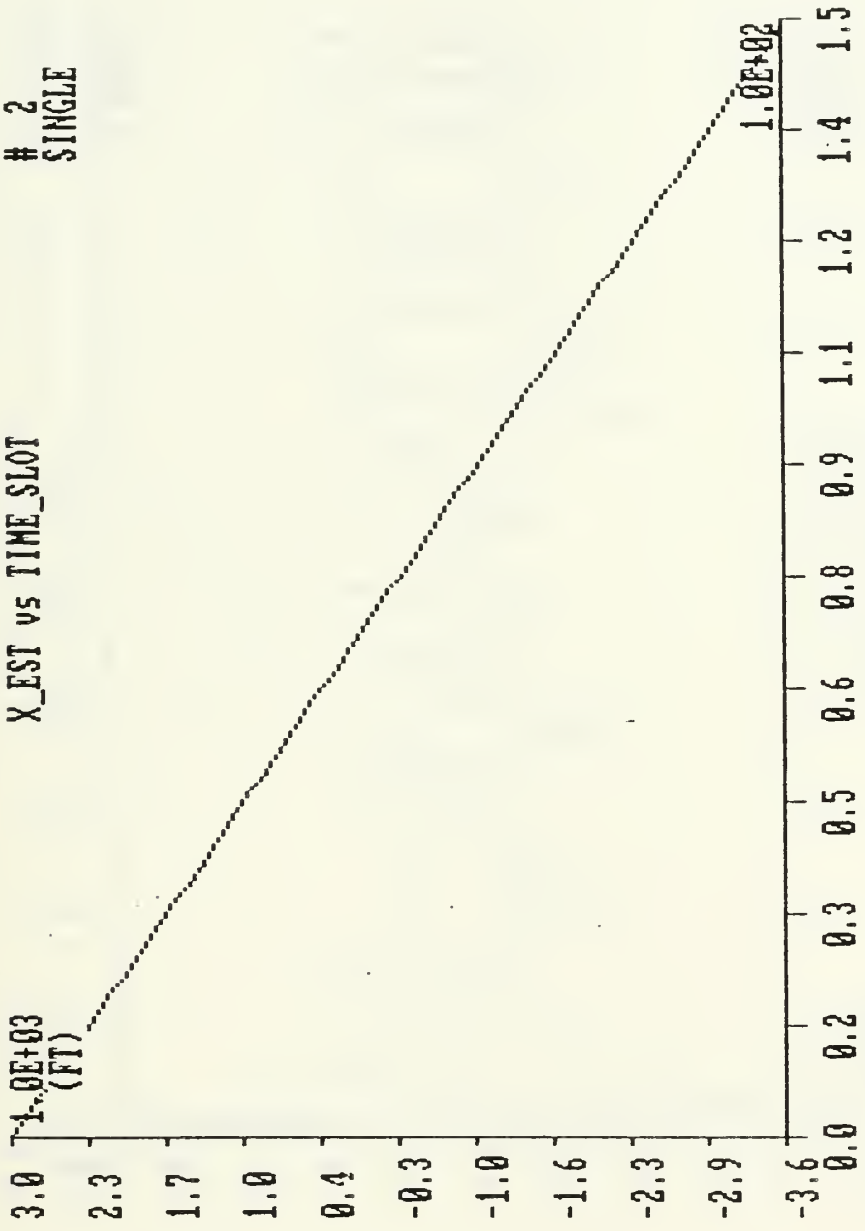


Figure 73. CIE - Estimate of torpedo position during a straight run in the area of single array. Initial conditions are: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

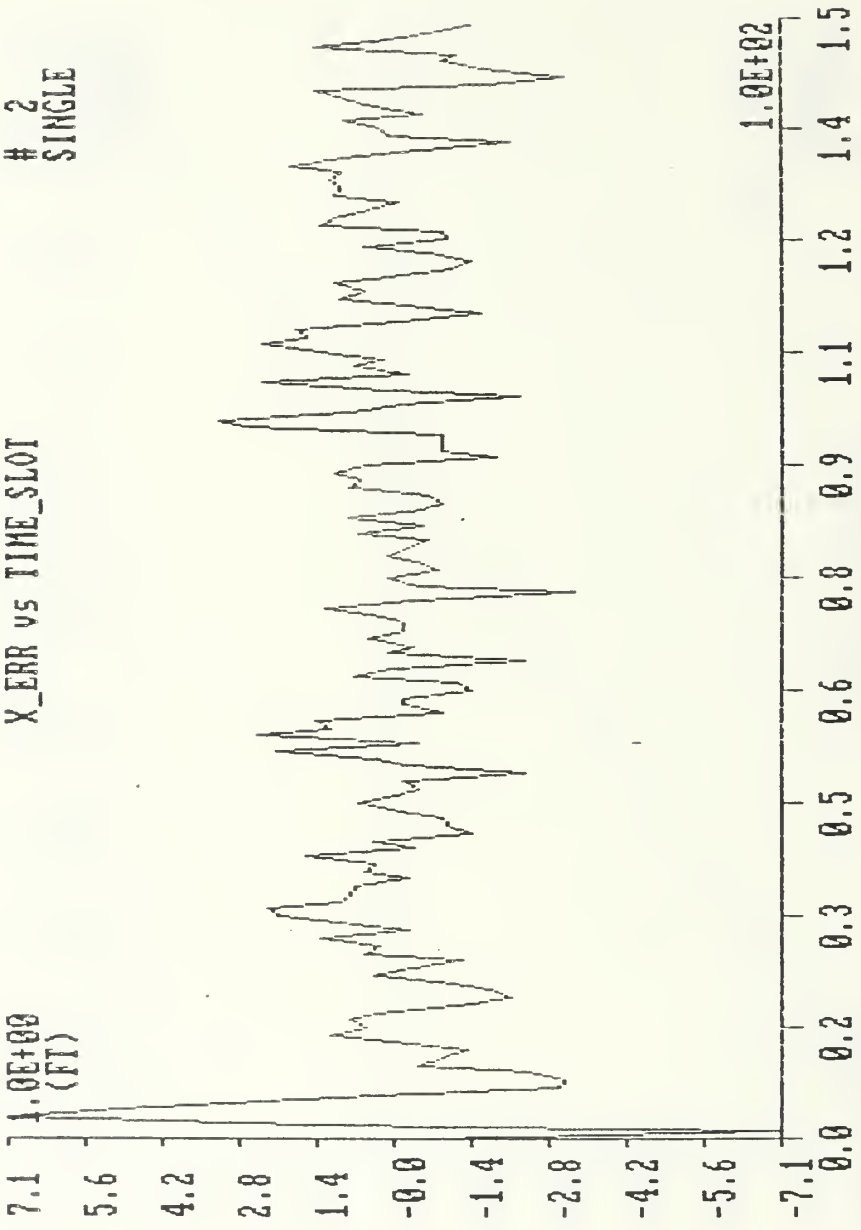


Figure 74. CIE - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

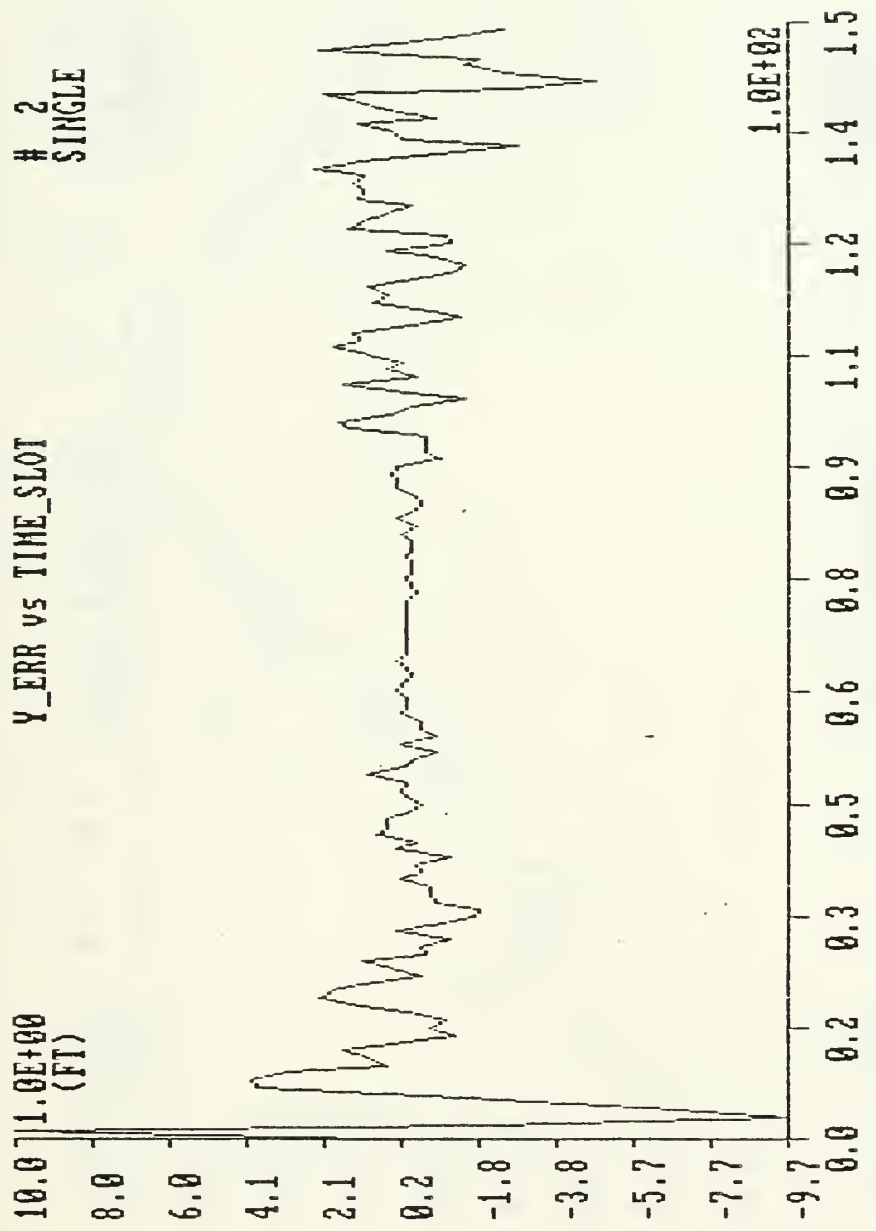


Figure 75. CIE - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

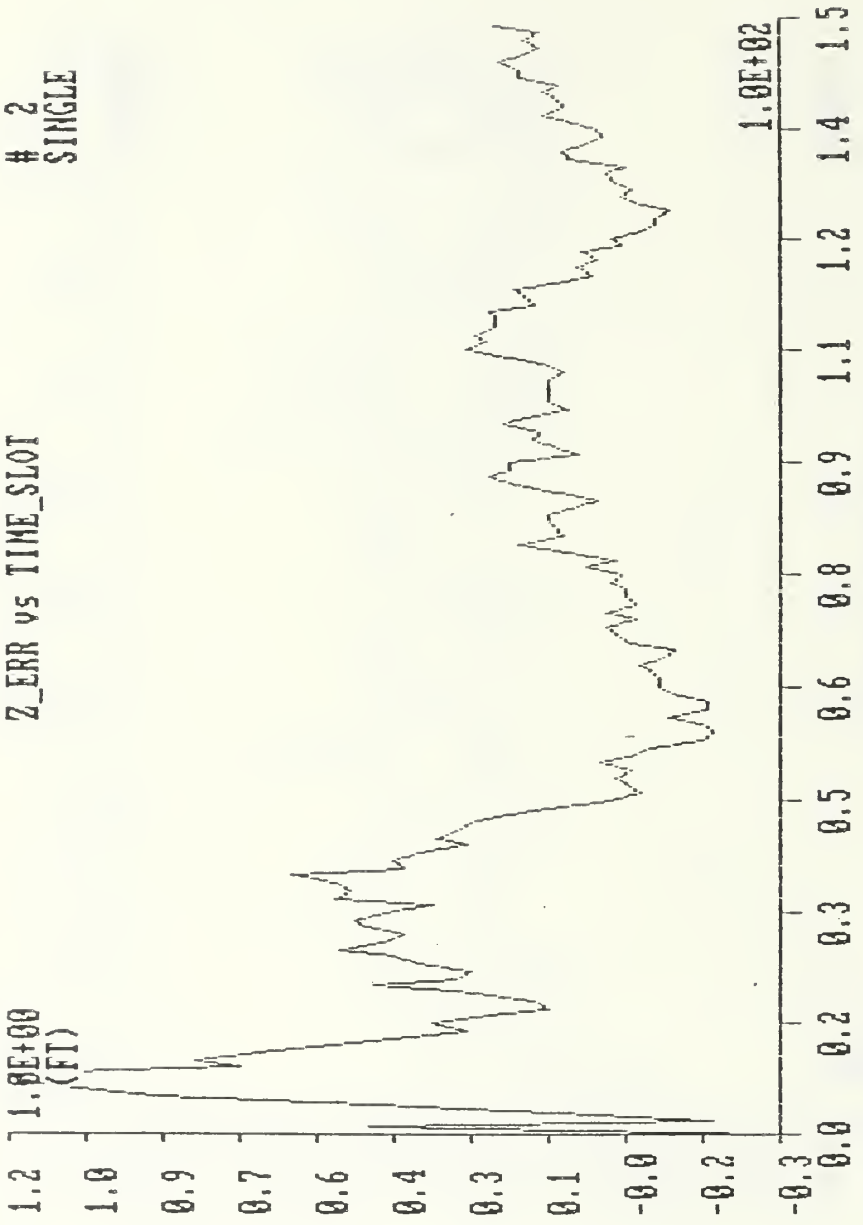


Figure 76. CIE - Error in torpedo depth during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

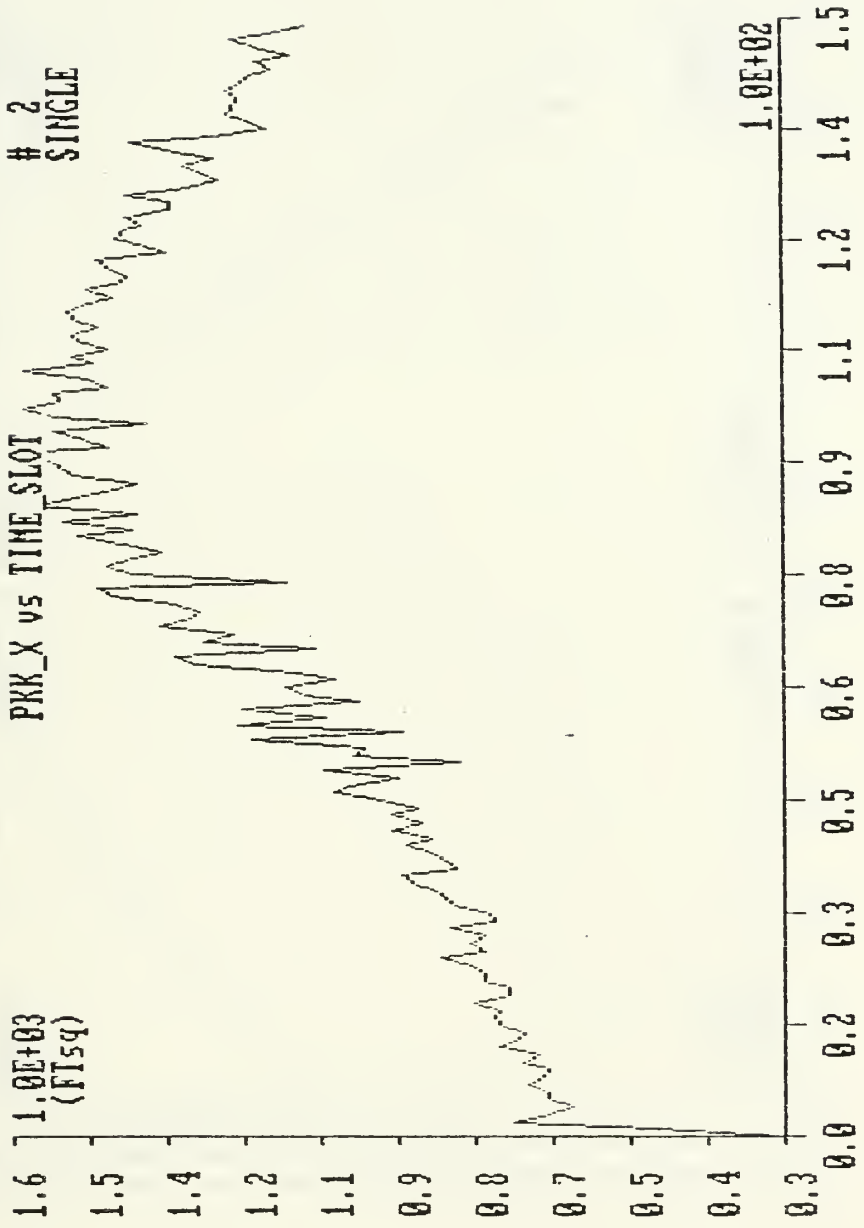


Figure 77. CIE - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

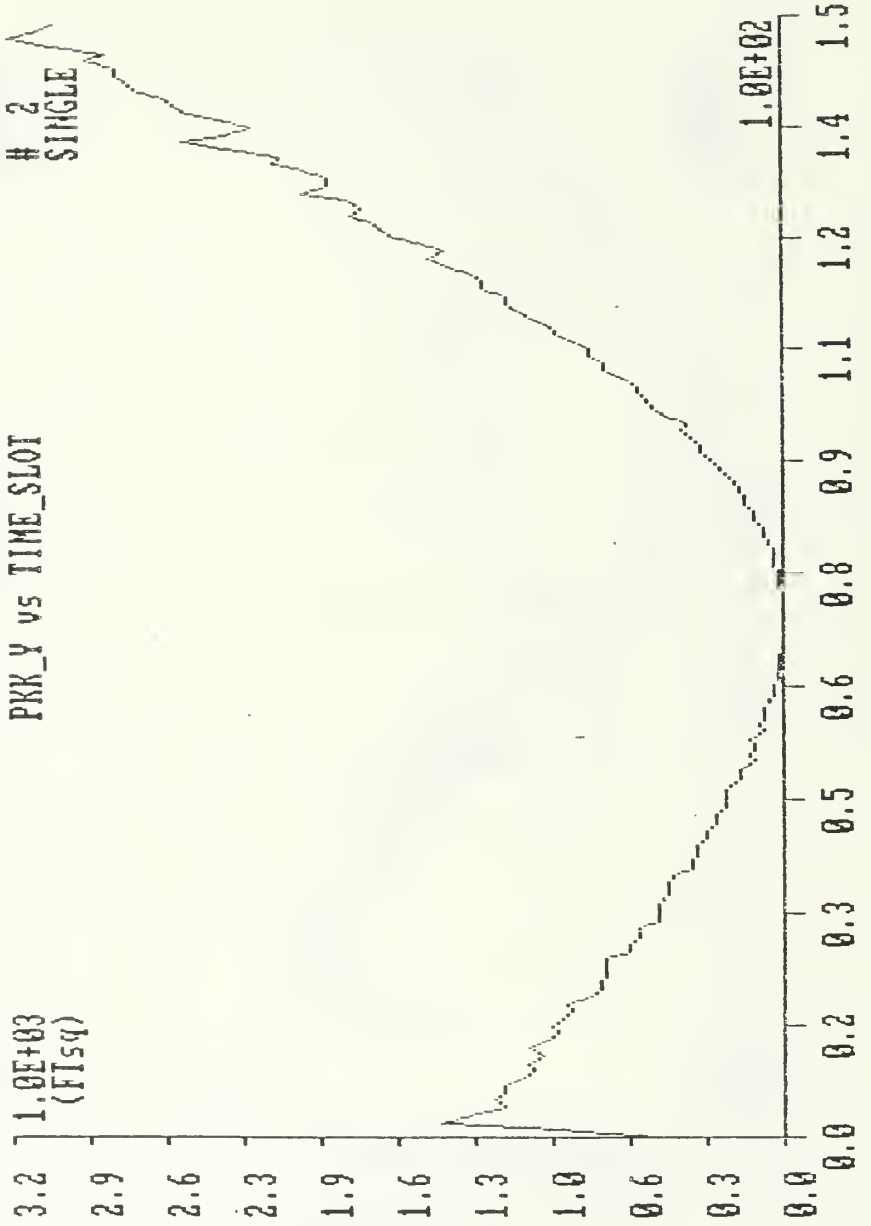


Figure 78. CIE - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

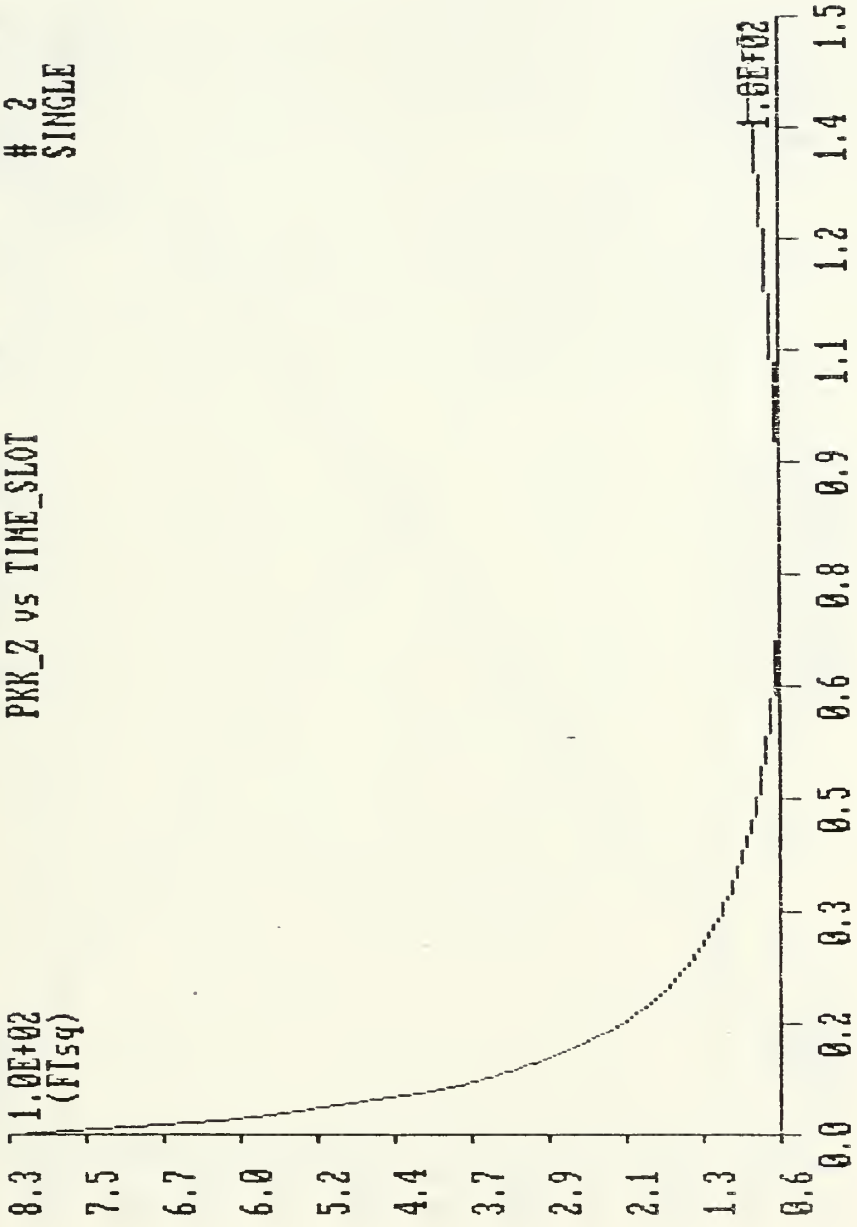


Figure 79. CIE - Filter estimate of mean-square error during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=2121 ft, Vx=-33.8 ft/s, Vy=0 ft/s.

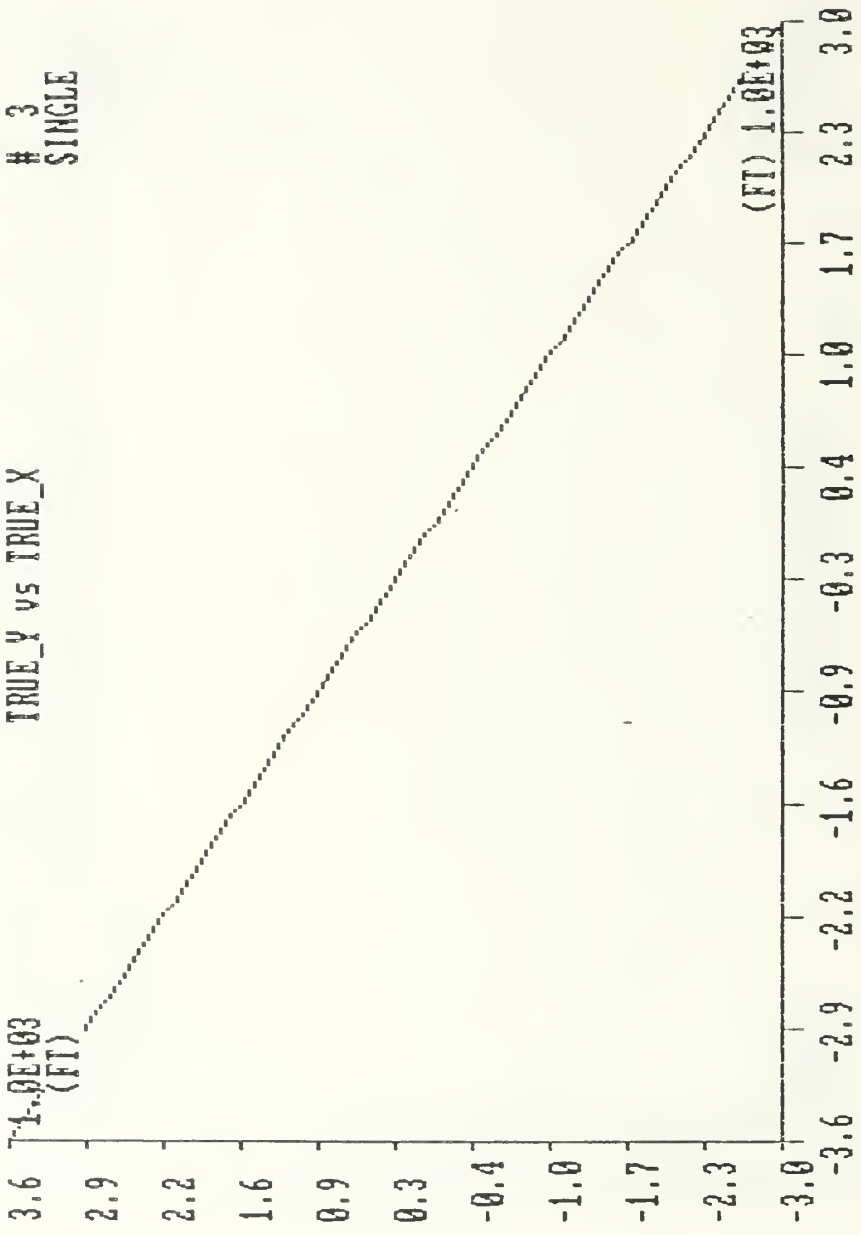


Figure 80. CIE - True trajectory of the torpedo in the area of a single array. Initial conditions : x=3000 ft, y=-3000 ft, Vx=-33.8 ft/sec, Vy=33.8 ft/sec.

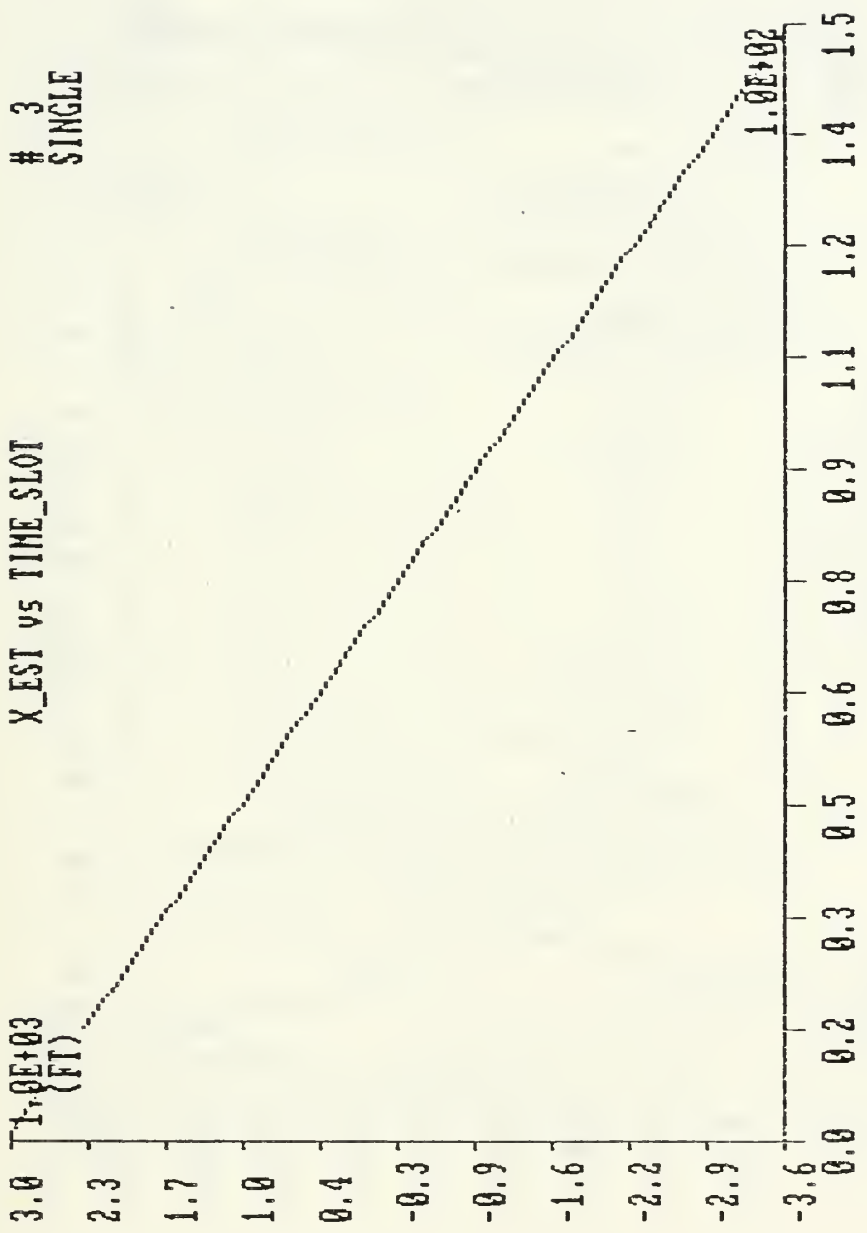


Figure 81. CIE - Estimate of torpedo position during a straight run in the area of single array. Initial conditions are: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

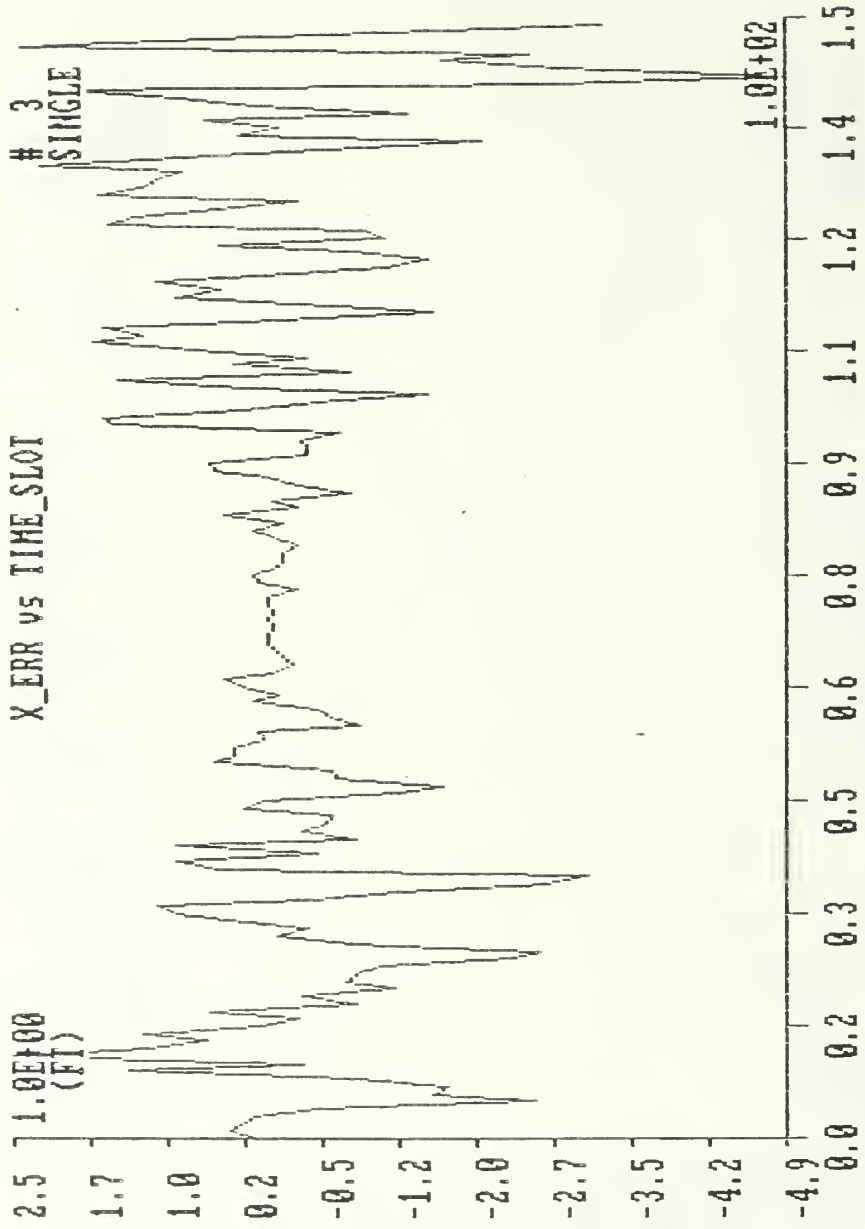


Figure 82. CIE - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

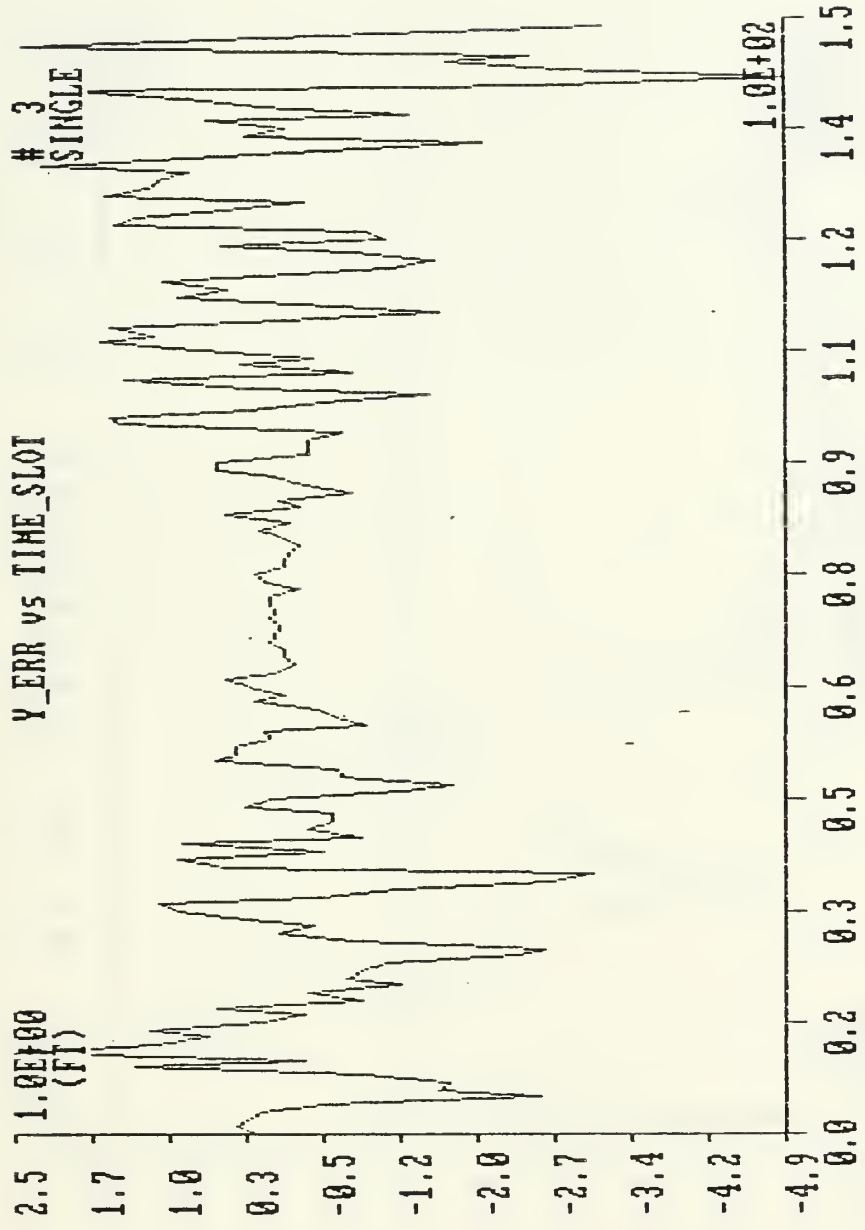


Figure 83. CIE - Error in torpedo position during a straight run in the area of a single array. Initial conditions: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

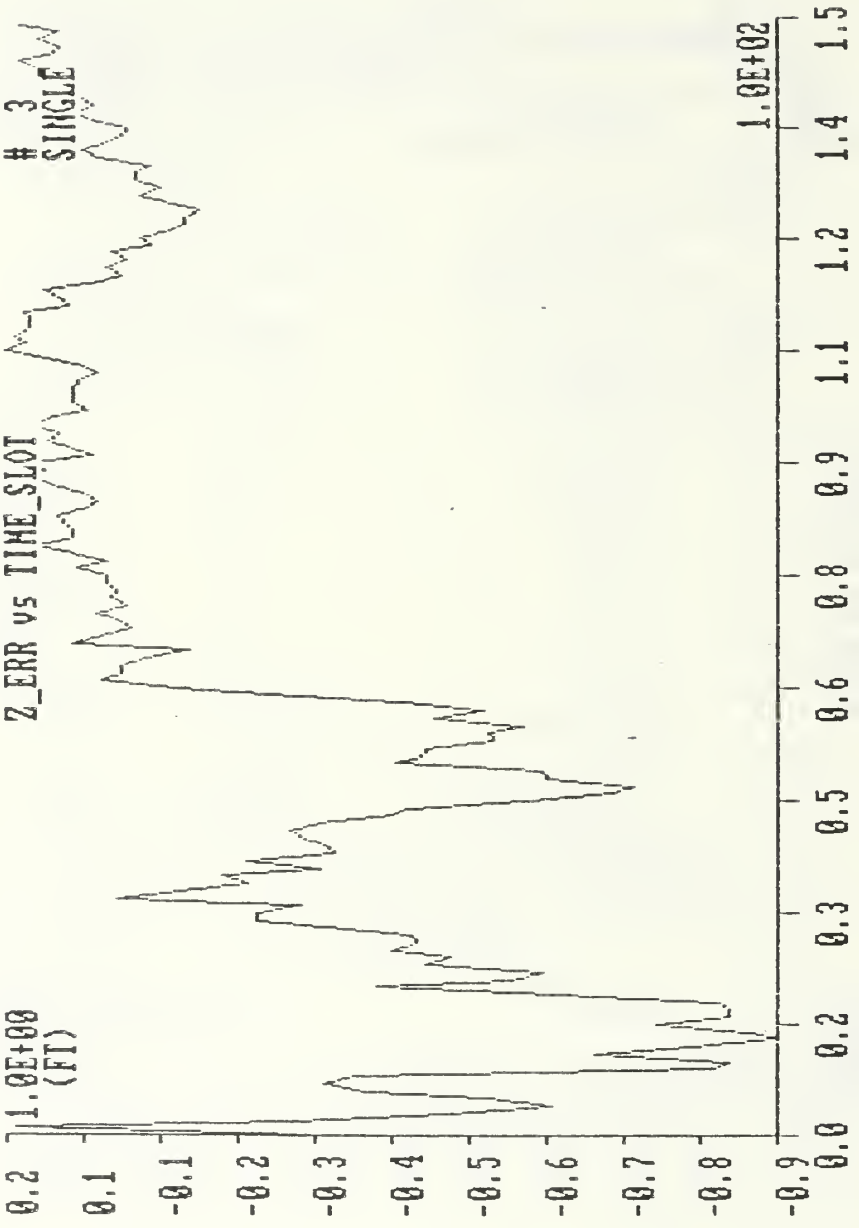


Figure 84. CIE - Error in torpedo depth during a straight run in the area of single array. Initial conditions: x=3000 ft, y=-3000 ft, Vx=-33.8 ft/s, Vy=33.8 ft/s.

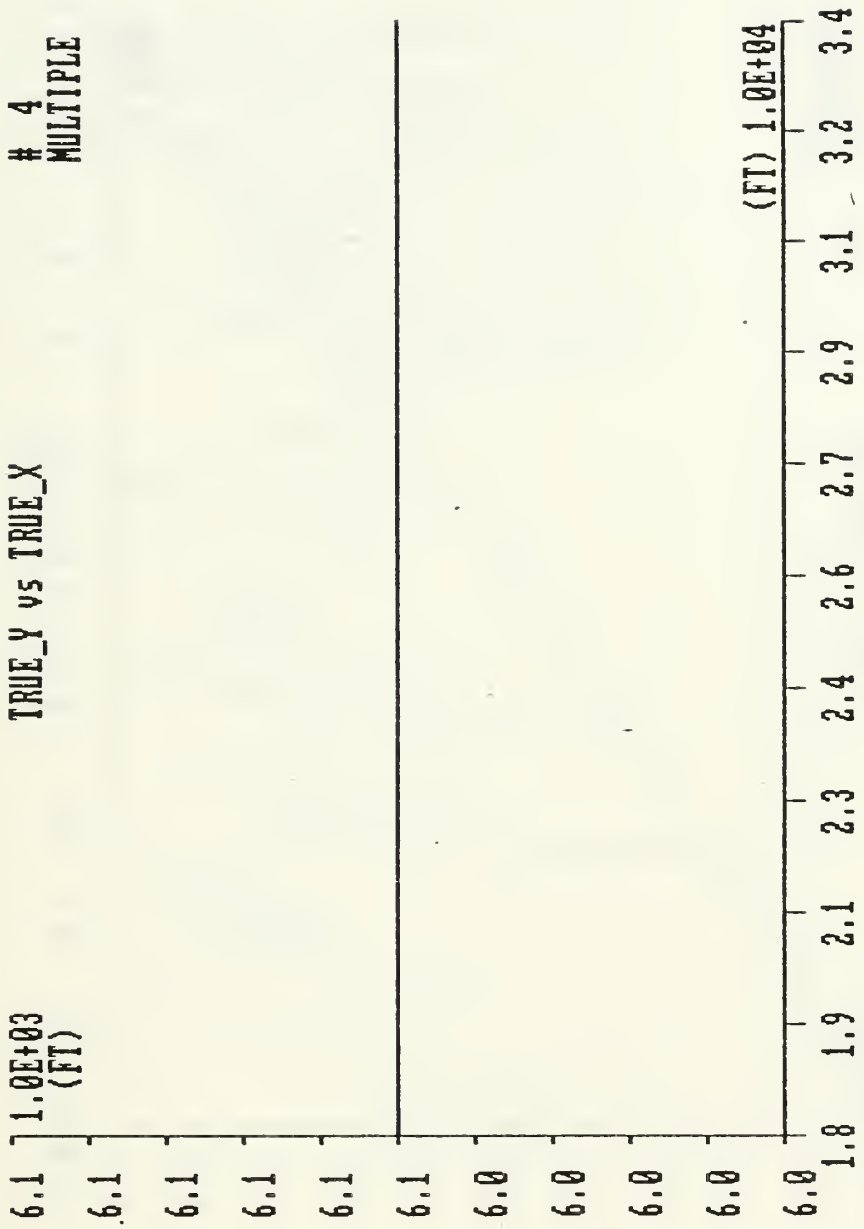


Figure 85. CIE - True trajectory of the torpedo in the during a straight runthrough multiple arrays. Initial conditions: x=34000 ft, y=6050 ft, Vx=-84.4 ft/sec, Vy=0 ft/sec.

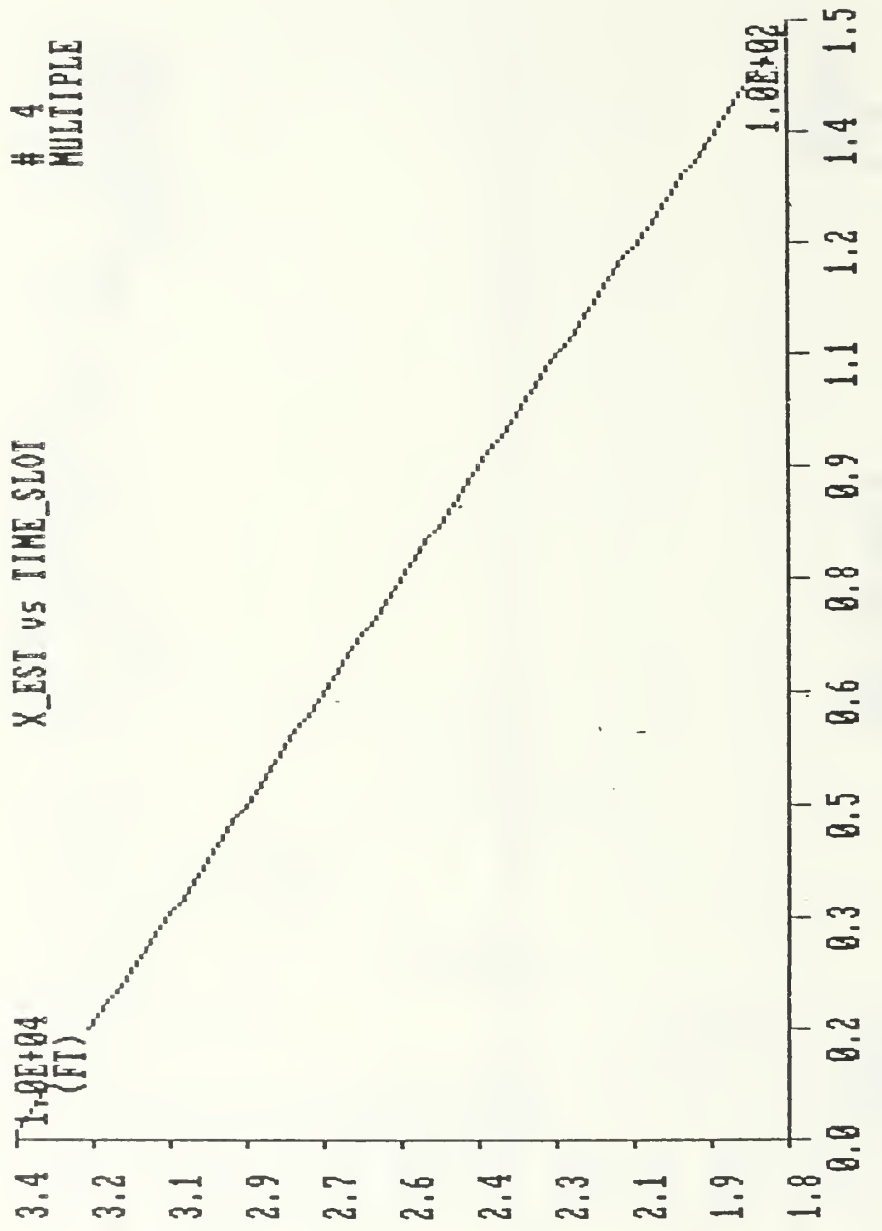


Figure 86. CIE - Estimate of torpedo position during a straight run through multiple arrays. Initial conditions are: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

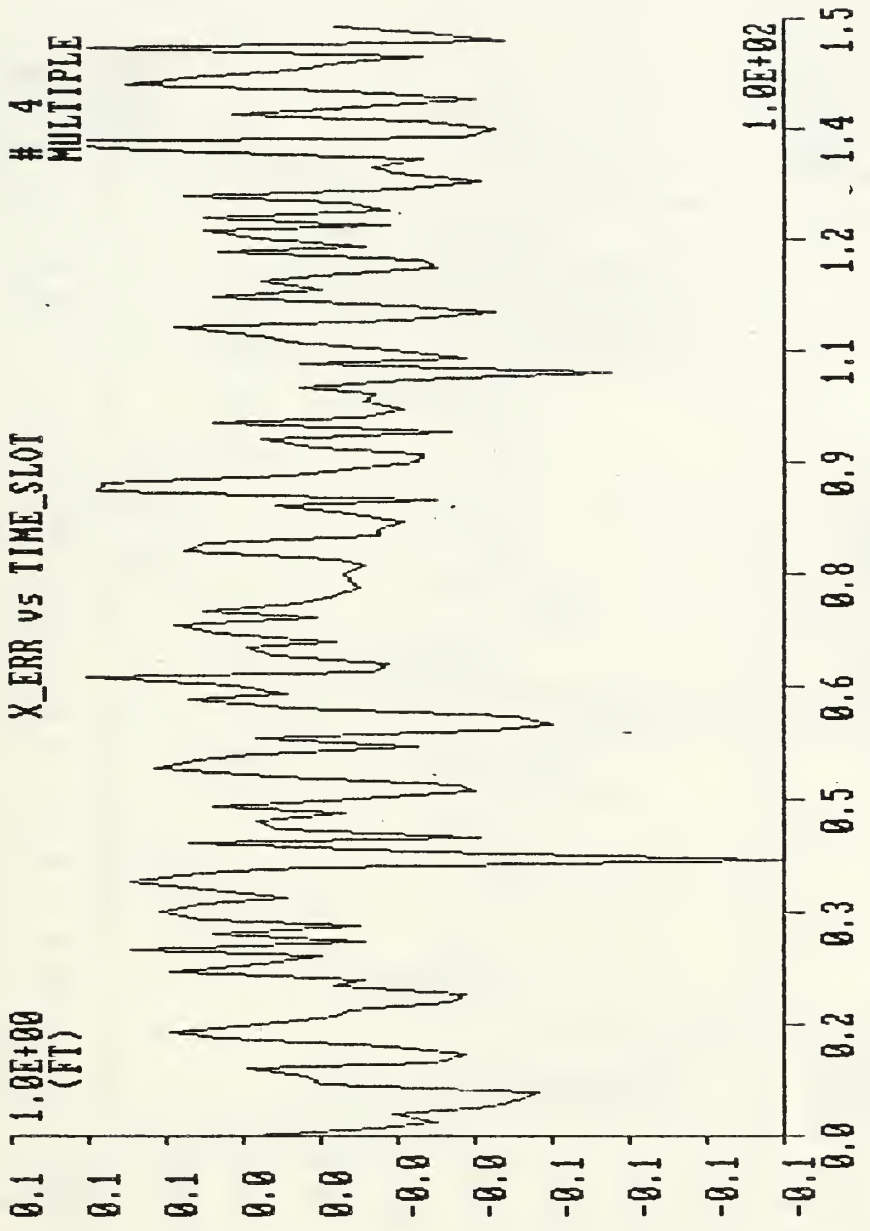


Figure 87. CIE - Error in torpedo position during a straight run through multiple arrays. Initial conditions are: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

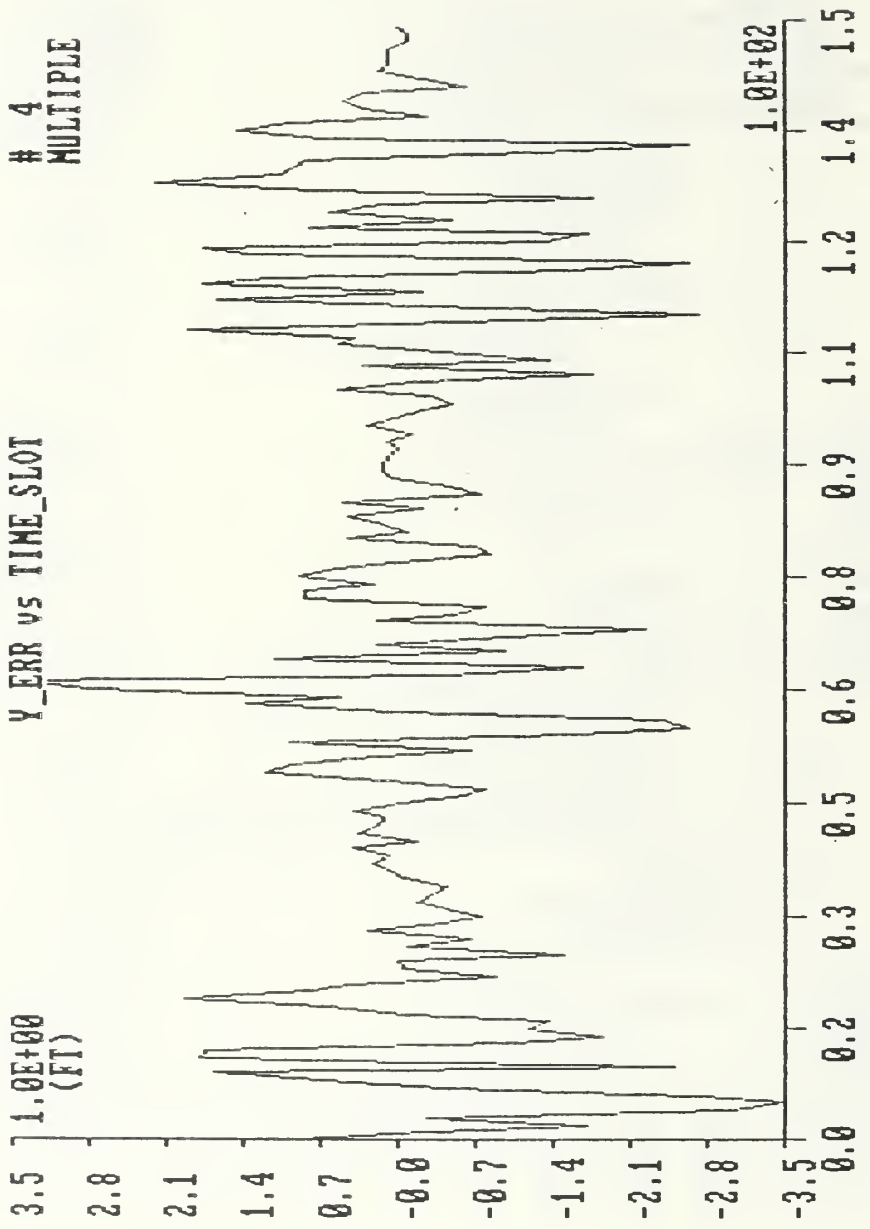


Figure 88. CIE - Error in torpedo position during a straight run through multiple arrays. Initial conditions are: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

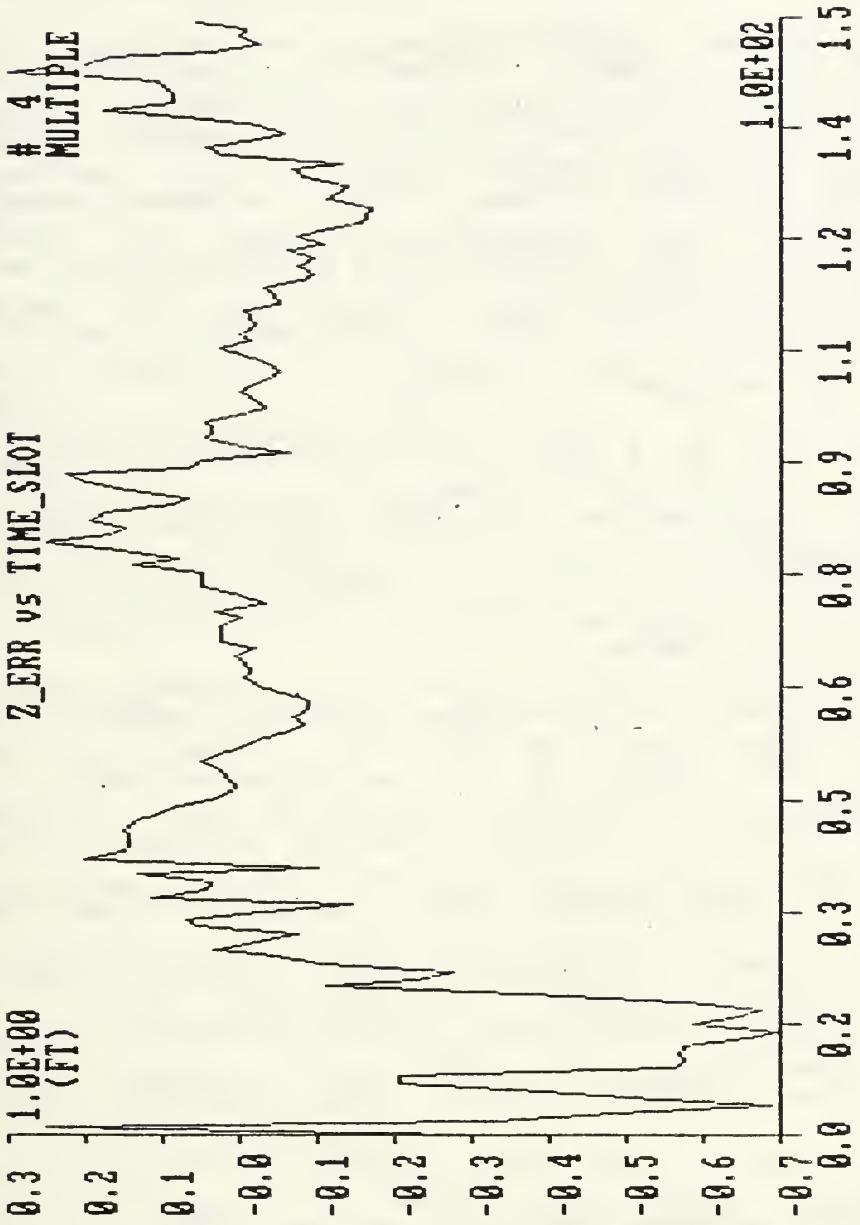


Figure 89. CIE - Error in torpedo depth during a straight run through multiple arrays. Initial conditions are: x=34000 ft, y=6050 ft, Vx=-84.4 ft/s, Vy=0 ft/s.

VI. COMMENTS AND CONCLUSIONS

There are slightly different versions of the program, for the IBM-PC and for the CIE microcomputer. However, the Sequencial Extended Kalman Filter program presented have many common characteristics. The execution time per cycle, in both machines, is less than the sample interval for the data. The program is written in the "C" programming language, and it is meant to be reasonably portable. Double precision calculations are widely used, and allowed in many cases to bound errors without repeating the calculations for the same time interval.

Calculations are performed using an adaptive $\underline{Q}$ matrix. That is, the matrix $\underline{Q}$ is recalculated each iteration, instead of using a constant value. The trajectory simulation routine was implemented using geometrical approach. The tangential speed is assumed constant during the whole run.

The Kalman Filter can track through single and multiple arrays. The algorithm always uses the coordinates of the closest array in the calculations, for the multiple array case. There is essentially, in either CIEKR1 and KR, only a main program without subroutine calls. The bidimensional matrices were transformed and are used as vectors. Most loops were eliminated to keep the execution time within the required limits.

The CIE version can also run in the IBM-PC but the IBM-PC version runs only in the IBM since it calls assembly language routines to generate graphic output. The IBM version is slower than the CIE version and the IBM-PC microcomputer must run with its 8087 math coprocessor chip to satisfy the speed requirements. Both machines were configured with hard disks, and the data files were read from and written to the hard disks. Both, the IBM-PC and the CIE versions, will run on the IBM-AT.

There is almost no documentation or software available for the CIE microcomputer. The lack of documentation, the lack of a good compiler, a full-screen editor, and the lack of graphics capability were major problems in using the CIE.

The improvement in the execution time was more noticeable in the CIE than in the IBM. This is probably due to the presence of the math coprocessor chip, 8087, and (or) due to a better compiler,in the IBM-PC. The use of the "-f" ("fast floating point format") option, when compiling code on the CIE, seemed to help both, the speed of execution and the precision of the results.

The programs were tested for torpedo speeds from 20 knots (33.8 ft/s) to 55 knots (92.8 ft/s) and turn rates of up to 20 dg/sec, or maneuvers of up to 1-G. The magnitude of the position errors for linear trajectories were maintained less than 10 ft in either computer. The CIE could not track any of the circular maneuvers (run numbers 6, 7 and 8 in

Table 2). In the IBM-PC, keeping the maneuver within the tracking radius of a single array (± 3000 ft around the array center), the magnitude of position errors were maintained less or equal to 7 feet for an 1-G maneuver and less or equal to 22 feet for the multiple array case.

The high error obtained for the circular maneuvers in either the single or multiple array case reflects the algorithm inability to follow fast turn circular trajectories. In [1] such high errors were reduced to values comparable to the errors found in the straight runs, by adding the adaptive Q matrix to PKK to produce PKKM1 and repeating the calculations for the same time interval. This was called "adaptive maneuver". The "adaptive maneuver" as implemented (see segment 2, in the algorithm, in Appendix A) was found effective for the IBM-PC when the value 10^{-10} was used for the diagonal elements of the matrix R. However, to keep the execution time per cycle within 1.31 sec, the number of adaptive maneuvers performed had to be limited. Almost no improvement in reducing tracking errors was achieved when the value used was 10^{-8}. For the later case a great deal of degradation in the execution time was observed and the time interval was exceeded even when just one more iteration (optimum maneuver) was executed. The procedure did not work on the CIE, which could not handle numbers less than 10^{-7} reliabily. The "adaptive maneuver" feature was

utilized only for the runs 7 and 8 (only 4 iterations were allowed) and have produced, for the 1-G maneuvers, results which are comparable to the results for the 1/4-G maneuvers. The single array cases showed smaller errors, meaning probably that the technique used to switch arrays can be improved.

An interesting point is the behavior of the curves for the diagonal terms of the matrix PKK. The curves PKK_X vs TIME_SLOT, PKK_Y vs TIME_SLOT and PKK_Z vs TIME_SLOT, which are the mean square errors should grow proportionally to the square of the position errors. These are depicted in the curves X_ERR vs TIME_SLOT, Y_ERR vs TIME_SLOT, and Z_ERR vs TIME_SLOT. When such does not happen it means that the filter model used is not a correct description of the process. When the value of the diagonal elements of the matrix $\underline{R}$ were 10^{-8}, only the qualitative behavior of the mean square errors agreed with the expected. When 10^{-10} was used, the quantitative behavior of the mean square errors were about the expected.

Additional tests should be performed using trajectories generated from actual torpedo runs on the Dabob test range. As a future step, there is the need to adapt the program to provide it with the correct interface formats to access the real data, before running it in the real-time conditions on the range.

APPENDIX A

PROGRAM DESCRIPTION

A. OVERVIEW

There are at this moment, three versions of the Extended Kalman Filter program. In either case the simulated trajectory must be first computed and the hydrophones array times, corresponding to the simulated positions, are stored in a file in order to be read in. All three programs write the files XKK.DAT, PKK.DAT, and XDIFF.DAT to the disk.

To avoid delays due to data writing and retrieving from disk drives, the data files can be written to and read from an electronic drive built in the RAM memory. There is also a utility program - PLOT.C - which allows fast graphic analysis after the run.

1. The CIE1.C Program

This was the first program to run in the CIE microcomputer. It reads "clean" (without noise) trajectory data from the file TRAJ.DAT and writes 3 columns, corresponding to the coordinates x, y, and z, to the disk. The code to generate Gausian noise is contained in the main program.

CIE1.C generates 3 data files, XKK.DAT, PKK.DAT, and XDIFF.DAT, containing filtered data to be post-processed. XKK.DAT, contains 4 columns, corresponding to the time slot and the filter estimates, X(k/k), for the coordinates x, y,

and z in that order. PKK.DAT contains 4 columns with the corresponding values of the time slot and the error covariance matrix for the coordinates x, y, and z, in that order. XDIFF.DAT contains 4 columns with the corresponding values of the time slot and the errors for the coordinates, x, y, and z, in that order.

2. The CIEKR1.C Program

This program runs in the CIE and provides input and output similar to the KR.C . It is able to simulate only straight line trajectories, due to the lack of an "arc tan" function in the CIE's C Compiler, and at present does not access the computer internal clock.

3. The KR.C Program

This program includes a set of assembly language routines for interfacing with the keyboard, display, and output ports of the IBM-PC/AT. It also includes a plotting routine, written in C, which calls the assembly routines to generate the frame, to print updated values, and to draw the estimated trajectory. Another assembly language routine was added to allow access to the internal clock in the IBM-PC. See references [7] and [8] and Appendix C for details.

The Kalman Filter section of KR.C needs the files TRJ.DAT and ZIC.DAT, generated by the trajectory simulation routine, for inputs.

The trajectory simulation generates transit times data already corrupted with noise. That involves inclusion

of the noise generation routine with the "clean" trajectory. The clean trajectory can be either a straight line or a circular curve.

The trajectory simulation routine creates the files TRJ.DAT, ZIC.DAT, TRJ.INP, and PARAM.DAT.

The TRJ.DAT contains the data for the clean trajectory, corresponding to the time slot, and the x, y and z coordinates. ZIC.DAT contains the noise corrupted transit times. TRJ.INP and PARAM.DAT contain information to be used by PLOT.C program in order to identify and relate the trajectory data and the parameters used in the Kalman Filter to the graphic outputs.

B. DETAILED DISCUSSION

Since the programs evolved one from the other, only the last one, KR.C, will be discussed in more detail. The routine that generates the trajectory and the tools developed to facilitate the objective analysis of results will also be briefly described.

1. The Kalman Filter Program - KR.C

The algorithm for the program KR.C is provided as Appendix B and the actual program implementation is shown in the Appendix E.

The capability of tracking trough multiple arrays is implemented by simply defining switching points along a coordinate axis, and comparing the estimated x-position against switching values. The coordinates of the tracking

array are then changed in order to always keep the closest array tracking the torpedo. Figure 90 shows the coordinate system for the multiple array tracking, the location of the switching points and the hydrophone location matrix.

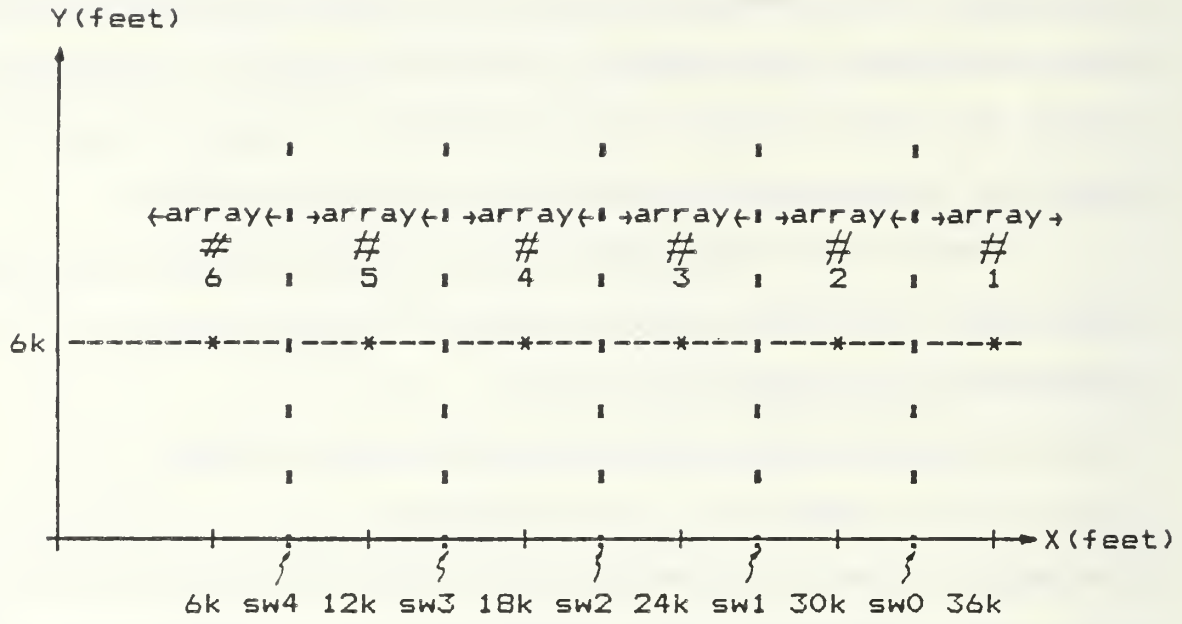
Tests performed using a straight line trajectory and keeping the Q matrix constant showed no improvement over the case where the adaptive Q is used.

2. The Trajectory Simulation Routine - TRAJEC

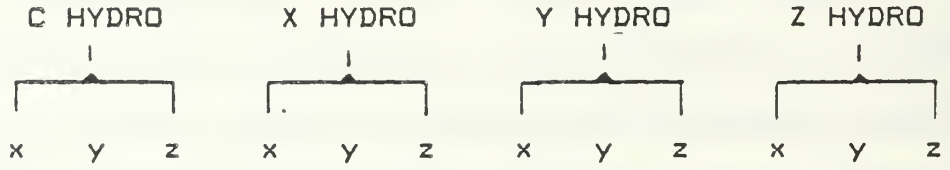
This routine was developed as an independent simulation program, to generate trajectories, where circular and linear segments can be combined to produce a desired path.

Given the initial x,y,z, positions and the initial speeds and assuming that the vertical speed is zero, this routine generates noisy data to represent the "raw" data as it would come from the hydrophones. TRAJEC finds the position of a point in a circle after a time interval T, given the tangential speed of the point, and given the desired turn rate or given the centrifugal acceleration. It is assumed that the tangential speed is constant during the maneuver. Either the turn rate or the centrifugal acceleration can be used to determine the radius.

Figure 91 represents the geometry of the problem. The straight line shown is tangent to circle. A point is initially moving on the line and at the time t_0 it is



Coordinate System for Multiple Array Tracking Case



36000	6000	0	36030	6000	0	36000	6030	0	36000	6000	30
30000	6000	0	30030	6000	0	30000	6030	0	30000	6000	30
24000	6000	0	24030	6000	0	24000	6030	0	24000	6000	30
18000	6000	0	18030	6000	0	18000	6030	0	18000	6000	30
12000	6000	0	12030	6000	0	12000	6030	0	12000	6000	30
6000	6000	0	6030	6000	0	6000	6030	0	6000	6000	30

Hydrophone Array Location Matrix

Figure 90

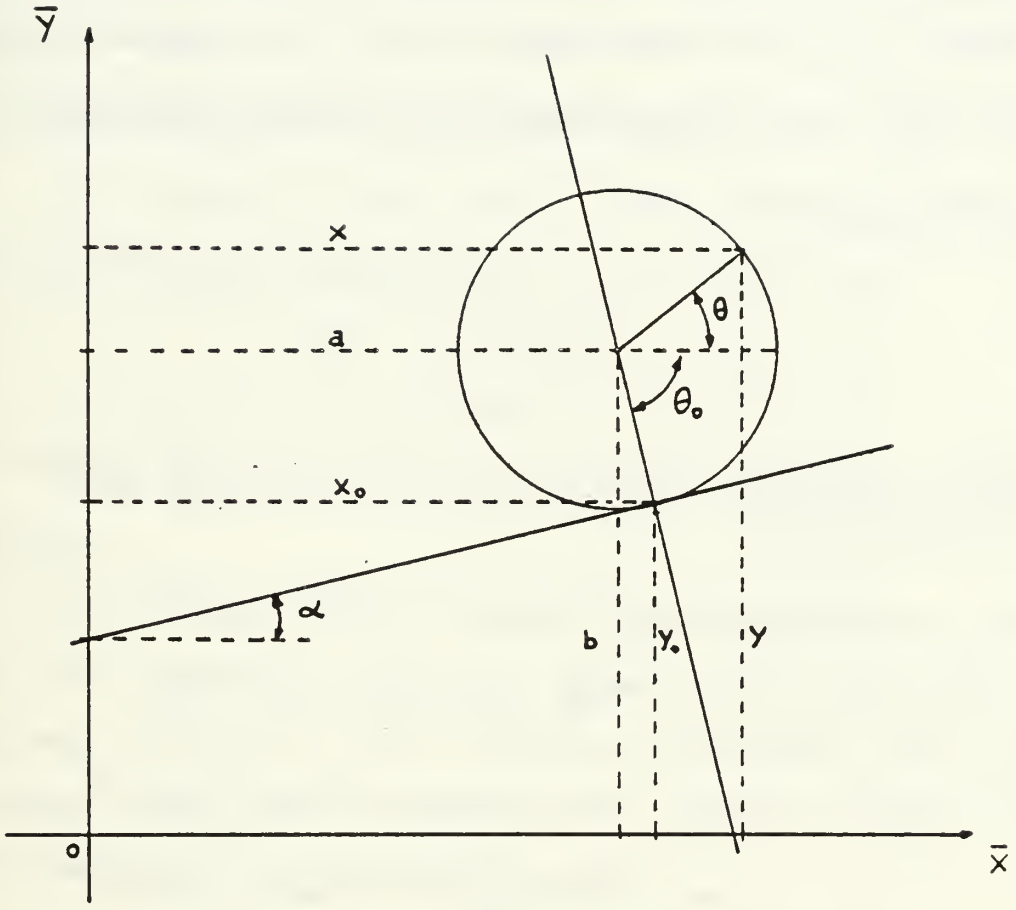


Figure 91 : The circular trajectory problem.

subjected to centrifugal force which makes it turn. The point then describes a circular movement generating a circle of radius R, and at the time t_0+T will be at the position X,Y. Assume V_y and V_x known speeds, $m = V_y/V_x$ is then the slope of the line tangent to the circle that passes at the point x_0, y_0 and has center at a,b. The angular coefficient of the line perpendicular to the circle and passing at the point x_0, y_0 is

$$(x_0 - a) / (y_0 - b) = - 1 / m \text{ , so}$$

$$x_0 - a = m * (y_0 - b)$$

and $R^2 = (x - a)^2 + (y - b)^2$ is valid for any point on the circle.

$$\text{at } x_0 , y_0 \rightarrow R^2 = (x_0 - a)^2 + (y_0 - b)^2$$

$$R^2 = (m^2 - 1) * (y_0 - b)^2$$

$$y_0 - b = \pm R / \text{sqrt}(1 + m^2)$$

$$\text{or} \quad b = y_0 \mp R / \text{sqrt}(1 + m^2)$$

$$\text{and} \quad a = x_0 + m * (y_0 - b)$$

For any point on the circle

$$x - a = R \cos \theta$$

$$y - b = R \sin \theta$$

From the Figure 91

$$\alpha = \tan^{-1} m$$

$$\theta_0 = \pi / 2 - \alpha$$

$$\theta = \theta_0 + \omega \Delta t$$

Given V_x , V_y $\rightarrow$ $V_t = \text{sqrt}(V_x^2 + V_y^2)$

Given ω $\rightarrow$ $R = V_t / \omega$

Given the central acceleration, g $\rightarrow$ $\omega = g / V_t$

At the end of the interval, the position will be x, y and the speeds will be given by

$$V_x = - V_t \sin \theta$$

$$V_y = V_t \cos \theta$$

These equations, when implemented, generate the true trajectory.

The routine also computes the measurement noise times. The Gaussian noise is generated by computing the linear approximation of the inverse of the normal curve [9]. The input comes from a sequence of uniformly distributed numbers generated by the equation below

$$r(I + 1) = r(i) * 317.\text{mod}(1) .$$

The resultant sequence is zero mean and is scaled to obtain the desired variance.

3. The Plotting Routine - PLOT

The plotting routine is contained in the file PLOT.C (IBM-PC), and was written as an independent module to be

linked with BIOSIO.OBJ, and reads data from the files XKK.DAT, PKK.DAT, XDIFF.DAT, TRJ.INP, and TRJ.DAT.

That is a menu driven program, which provides fourteen (14) different options and was designed to simplify the analysis of data generated in a torpedo run. The plots have their titles already chosen. The scales reflect the maximum and minimum values. A reference number and a label for SINGLE or MULTIPLE array cases is printed on the plot for reference. Linear interpolation is used to interconnect the sample data points. An optional plot, where the data points are simply plotted, is also available. This last option would allow the time to be associated to each point on the curve.

In an attempt to provide the CIE machine with similar capabilities, an ANSI driver was also defined and tested successfully in the IBM. However, that provides a very limited graphics capability, allowing only 80 columns by 25 rows of resolution against the 640 by 200 in the IBM, in graphics mode. The file KR1.C makes use of the defined driver to produce the graphics presentation of the estimated trajectory.

For UNIX systems, like the CIE, one might want to consider the "termcap" database of hundreds of encoded descriptions for terminal control. However, except in unusual circumstances, C's standard-library I/O functions

provide excellent portability and a wide range of display features [6] .

4. BIOSIO.ASM and BIOSIO.OBJ

These files contain the assembly language code (public domain) and the object code (IBM-PC) to interface the C programs with the keyboard, display, output ports and to access an absolute memory location.

5. DOSTIME.ASM and DOSTIME.OBJ

These files contain the assembly language code and the object code, respectively, that allow access to the internal clock (IBM-PC).

6. The SIMTERM program

This is a public domain program written by Jim & Eric Holtman that allows one to emulate an HP-like terminal to a UNIX system. It is a self contained module that allows one to connect the IBM-PC to the CIE microcomputer, directly by using the RS-232 output on the IBM and one of four (4) output ports on the CIE.

7. The KALMAN.H header file

It contains the definitions of the program constants, initial values of parameters, and the definition of the ANSI terminal driver, which can be used in the future to generate rough plots on most of the nongraphic terminals.

APPENDIX B

ALGORITHM FOR THE KR.C PROGRAM

```
#define   CONSTANTS        /* used to define constants */
#define  MACROS           /* define macros */
#include FILE.H           /* include header file */
double  xkk[200];          /* declare global variables */
main()                     /* main prog. declaration */
    {                      /* block/function delimiter */
    /* DEFINING FILES */
    FILE *fpxyz, *fopen(); /* declares fpxyz as a */
     .                          /* pointer to a file   */
     .                          /* yet to be open using*/
                                /* fopen() function    */
    /* DECLARATION OF VARIABLES */
    int i,j;             /* long integers */
    short ii,jj;         /* short integers */
    float x,y;           /* single precision */
    double xx,yy;        /* double prec. reals */
    /* OPENING FILES */
    fpp=fopen("pkk.dat","w"); /* open file pkk.dat */
                              /* for writing */
    fpe=fopen("xdiff.dat","w"); /* open xdiff.dat  */
    fpxyz=fopen("xkk.dat","w"); /* open xkk.dat    */
    /* INITIALIZATION OF VARIABLES */
        Initialize HYDRO[]
        Initialize gamma[]
        Initialize covw[]
        Initialize xkkm1[]
        Initialize xb[],yb[],zb[]
    /* DATA INPUT - INTERACTIVE */
menu:   Put menu
        Get inputs
            Performs actions
                Case statement
                    1 Run trajectory
                      Write run# and xkkm1[] to trj.inp
                      Call  trajec  to  read trj.inp and
                       generate noisy trajectory
                    2 Run Kalman filter
                    3 Modify Parameters
                       a. Optimization criteria
                       b. Single/multiple array
                       c. Initial x, y, z
                       d. Initial Vx, Vy
                       e. Noise covariance
```

```
                        f. Error covariance
                        g. Redefine maneuver
                           - Turn rate based
                           - G number
                           - Start/end time
                        h. Main menu
    /* Run Kalman filter */
run:    Get run # for reference
        Save inputted data into files
        Open files trj.dat and zic.dat for reading
        Initialize QFIND block
        Initialize hrow[]
        Initialize pdum[]
        Initialize rr[]
        Prepare screen for real time presentation
           Calculate scaling
           Draw frame
        Get time from internal clock
    /*  START TIME LOOP  */
        Start time loop : for (kk=0;kk<itime;kk++)
           {
           Get hydrophone array coordinates
    again: Get time
              If time interval < 1.31 goto again
           Update time
           Get true times e true positions (noisy data)
               Read data[] from file trj.dat
               Read zic[]  from file zic.dat
               Save true positions in true[]
    adapt:      /* GET HROW, CALCULATE GAIN, ESTIMATE */
               /*  COVARIANCE  OF  ERROR SEQUENCIALLY   *
    js:        for (i=0;i<4;i++)
                  {
                  Calculate denom using double prec. sqrt
                  Calculate hrow[0], hrow[2], hrow[4]
                  Calculate gnum[], gdenom
                  Calculate gi[] = gain column
                  Calculate covariance of error pi[]
                      for (ii=0;ii<25;ii++)
                          {
                          calculate pdum[]
                          }
                      calculate pi[]
                  Calculate measurement prediction
                      calculate zhat=denom/4860.
                      calculate zdiff[i]=zic[i]-zhat
                  Compute 3sigma gate
                      p=max(pi[0],pi[12],pi[24])
                      /* use single precision sqrt */
                      gate=3.*(sqrt(p)/4860 + rr[i])
                  Edit invalid time measurements
```

```
                    Set flag
                    Put gain, gi[]=0
                 Calculate estimate based on one meas.
                                              prediction
                    for (ii=0;ii<5;ii++)
                       xi[ii]=xkkm1[ii]+gi[ii]*zdiff[i]
                 /* Update for next column */
                    for (ii=0;ii<5;ii++)
                       {
                       xkkm1[ii]=xi[ii]
                       pkkm1[ii]=pi[ii]
                       }
                    for (ii=5;ii<25;ii++)
                       pkkm1[ii]=pi[ii]
                 if (i==3) goto 156
                 }  /* end of js loop */
              /* Accept values as updates for time */
              /* interval  */
156:          Do updates for the time interval
                 xkkm1[]=xi[]
                 pkkm1[]=pi[]
                 xkk[]=xi[]
                 pkk[]=pi[]
        |     Recalculate time measurements and form
(1)     |                absolute value of residuals
        |     Edit invalid times for adaptive maneuver
        |                                       routine
              Compute adaptive Q matrix
                 If (icurve==1) compute adaptive Q
                 else use constant Q
        |     Check time
        |        By pass maneuver if no time left
        |        Check for valid measurements
        |        By pass adapt. maneuver if all invalid
        |        By pass maneuver if not steady state
(2)     |           Check average zdiff[i] vs. zdifav
        |           By pass maneuver if avg.<zdifav
        |           /* Adaptive maneuver */
        |              pkkm1[]=pkk[]+q[]
        |           /* Reiterate same time slot */
        |              goto adapt
              Compute measurement errors
                  xdiff[0]=xkk[0]-trued[0]
                  xdiff[1]=xkk[2]-trued[1]
                  xdiff[2]=xkk[4]-trued[4]
        |     Compute switch to change between Q
(3)     |         constant : straight line trajectory
        |         or Q adaptive : circular trajectory
              Calculate predictions for the next time
                  pkkm1[] and xkkm1[]
```

```
            Output results to the screen
                x,y,z,vx,vy,time values
                plot xkk[0], xkk[2]
            Output results to files
                xkk[],pkk[],xdiff[]
        }  /* end time loop */
Close files
}   /* end main() */
```

APPENDIX C

SETTING UP FOR EXECUTION

A. USER'S NEEDS

This section describes the actual hardware and software utilized to produce the results in Appendix H.

HARDWARE

The following describes the hardware utilized and the way to interconnect the IBM-PC and the CIE microcomputer .

IBM-Pc : 2 floppy disk drives, 10 megabytes hard disk, graphic board, graphic color display, serial port (RS-232), 8087 math coprocessor chip and Epson parallel printer;

CIE 680/30 : 20 megabytes hard disk, 4 serial output ports, Amdek monitor and keyboard.

The computers were interconnected by cable, from the IBM RS-232 output port to one of the 4 ports on the CIE.

SOFTWARE

The files and/or programs utilized are:

1. SIMTERM - communications program (in the IBM). Allows transferring files between the IBM and the CIE.

2. CIE.BAT - batch file (IBM). Executes the necessary sequence to interconnect the IBM and the CIE. Needs SIMTERM.

3. KR.EXE - executable file that contains the program to simulate the torpedo trajectory and the Kalman filter program (IBM). The trajectory routine is executed directly from the KR.C program as a menu option and it generates the

data files "trj.dat", and "zic.dat". Then the trajectory and time information from the files, "trj.dat" and "zic.dat", is used by the Kalman filter algorithm to produce estimates. It generates the files "xkk.dat", "pkk.dat", "xdiff.dat", "trj.inp" and "param.dat". At the end of the run, a total of 7 files would be generated.

4. PLOT.EXE - executable file that contains the plotting routine (IBM). Allows posterior graphic analysis of the results. Uses the information contained in the files "xkk.dat", "pkk.dat","xdiff.dat","trj.dat" and "trj.inp".

5. KAL.BAT - batch file (IBM). Executes the necessary sequence to run KR.EXE and PLOT.EXE.

6. KALMAN.H - header file (IBM and CIE). Contains definitions used by CIEKR1.C , KR.EXE, PLOT.EXE, TR.EXE .

7. a.out - executable file generated from compilation of CIEKR1.C on the CIE machine.

B. PROGRAMMER'S NEEDS

It is highly recommended to have a good full-screen text editor, and the source code files included in the directory of the IBM-PC. Source code files have extension ".C". A good C compiler (IBM) is also recommended.

The software required is:

1. LATTICE C or MICROSOFT C compiler ;

2. FULL SCREEN EDITOR ;

3. SIMTERM - terminal emulation / communication program;

4. SOURCE files : TR.C, KR.C, PLOT.C ;

5. HEADER files : KALMAN.H, MATH.H, STDIO.H ;

6. ASSEMBLY LANGUAGE ROUTINES: BIOSIO.ASM, DOSTIME.ASM ;

7. IBM MACRO ASSEMBLER ;

8. MICROSOFT LINKER .

Using the Lattice C compiler requires the files LC1.EXE, LC2.EXE, LCS.LIB, LCMS.LIB, CS.OBJ, MATH.H, STDIO.H to be present. The assembly language routines must first be transformed into object files by using the Macro Assembler. The file TR.C must also be compiled using LC1 and then LC2 to generate the TR.OBJ file. These are all linked to the file KR.OBJ to produce the executable file KR.EXE .

APPENDIX D

COMPILING, EXECUTING, TRANSFERRING FILES, AND PLOTTING

A. RUNNING THE PROGRAM ON THE IBM-PC

This will assume one is running the Kalman Filter program on the IBM PC/AT with a hard disk drive defined as drive "c".. The system prompt is "c>", and the required files are in the subdirectory named "KALMAN".

If you are not in drive "c", change to the correct drive typing "c:" and hit "RETURN". The system will then answer with the prompt "c>". Type "cd kalman" to access the directory KALMAN .

1. Compiling KR.C

This will assume you are running the Lattice C compiler. Type "ce kr" to compile the file KR.C, using the batch file "CE.BAT". The files KALMAN.H, DOSTIME.OBJ, TR.OBJ, BIOSSIO.OBJ, LINK.EXE, CE.BAT, and KR.C need to be in the same directory.

The Appendix E contains a listing of the file CE.BAT.

2. Executing KR.EXE and PLOT.EXE

Type "KAL",in drive "c", to execute the batch file KAL.BAT. The required sequence of commands is initiated and can be interrupted by pressing "CTRL-BREAK" ("CTRL" and "BREAK" keys at the same time). The Appendix E contains a listing of the file KAL.BAT .

The first file to be executed is GRAPHICS. It allows dumping IBM-PC graphic screen to the printer.

Next the file KR.EXE, containing the trajectory routine and the Kalman Filter program, is executed. Menus give the indication for data input, to run the trajectory routine, and to run the Kalman Filter. One normally defines the parameters first (option # 3), then run the trajectory routine (option # 1), and then run the Kalman Filter (option # 2).

After the run, the plotting routine ("Plot") is called to produce the graphic output. Up to fourteen different plots can be generated by making the desired selection from the menu. The output to the printer is obtained by pressing "SHIFT-PRTSC" ("Shift" and "PrtSc" at the same time).

If a plot ends abnormally, for instance by the use of CTRL-BREAK, the screen is left in the graphics 640 by 200 mode, then the command "mode bw80" will set the display mode to Black and White with 80 characters per line.

B. RUNNING THE PROGRAM ON THE CIE

Turn on the system with the floppy disk drive open. This will force the system to boot and accept the internal hard disk drive as the current drive. Turn on the terminal display and when the cursor appears, press "ENTER". Follow the prompts for date and time input.

After booting, the screen will show the message "CIE GRAPHICS SYSTEM # 7" at the top, and "login : " at the bottom. At this moment one should have the SIMTERM diskette in drive A (the one at the left) on the IBM-PC. If the logged drive is not A, one should type "a:" and press "ENTER" to change to drive A . In drive A, one should type "cie" and "RETURN" (ENTER). The batch file CIE.BAT will send the correct command line to connect the IBM and the CIE. Make sure the cable connection is already done.

The message "connection established" will then appear on the screen. Press "ENTER" to get "login:". Log in using the account "jose", and press "ENTER" for the password. The system prompt is "%" . Change directory using "cd work" . One is then ready to transfer files or to execute a program in the CIE machine.

1. Transferring Files from the IBM to the CIE

To transfer a file use the redirection capability of the Unix system and the command "cat".

Type "cat > filename.ext" and press "ENTER", at the system prompt ("%"). The cursor will stop at the begining of the next line. "ALT-D" will dump a file to the communication line. One will be prompted for the filename.

If, for instance one wishes to transfer the file "kalman.h", which is in drive c: (IBM), to the CIE, given it the name "kalman.h", the following sequence should be performed, starting at the prompt "%" : Type "cat >

the name "kalman.h", the following sequence should be performed, starting at the prompt "%" : Type "cat > kalman.h"; press "ALT-D"; type "c:kalman.h" , when asked for the IBM filename, and press "ENTER" to start the transmission.

At the end of the transmission the computer bell will ring, signaling buffer overflow. One should wait until no more new lines are being written. Press "CTRL-D" to close the file in the CIE. The prompt "%" will reappear on the screen.

The new file (kalman.h) will have to be edited on the CIE to remove the inserted empty lines:

Type "ed kalman.h" and the editor will show the number of lines in the file. Then, type "g/^$/d" and press "ENTER". The number of lines in the file will be reduced to the half. This will not affect the blank lines originally in the program, just the ones inserted by the process of transmission.

Type "w" to write the modified program version to the disk, and "q" (quit) to leave the editor.

2. Compiling on the CIE

Type the command line "cc -f -v -L -7 ciekr1.c" to generate the executable file "a.out".

where : "cc" invokes the C compiler;

"-f" option , informs the compiler to use "fast floating point format";

"-v" option ...

"-L" option instructs the compiler to generate pointers to allow long jumps, for a code+data greater than 32 kbytes;

"-7" option instructs the compiler to use the Unix version 7 standard library.

3. Executing CIEKR1

Type "a.out" to start execution and then follow the menus for inputs. Since the CIEKR1.C program does not generate circular trajectories, the files containing the trajectory data (TRJ.DAT) and transit time data (ZIC.DAT) will have to be transferred from the IBM to the CIE, in order to run the Kalman Filter for the circular trajectory. Be sure to input the same data that were used to generate the circular trajectory on the IBM, and then select option #1 (Run Kalman). For straight line trajectories, just follow the menus.

4. Plotting the Results

Since there is no graphics capability built in the CIE, the only way to obtain plots is to transfer the files XKK.DAT, PKK.DAT and XDIFF.DAT, obtained on the CIE, back to the IBM and then use the routine PLOT to generate the desired graphic output. The procedure is described below:

"ALT-C", during the SIMTERM session, will open a capture file. One will be asked the name of the capture, or log file. The drive name should be specified.

For instance, assume one wants to transfer the file XKK.DAT from the CIE to the IBM in order to use it later to generate plots. Since, the routine PLOT reads specific files, be sure to erase the old file XKK.DAT, in the IBM, before copying the file XKK.DAT from the CIE, because the same name will be used.

Type "list -n xkk.dat", at the prompt "%", without pressing "ENTER". Pressing "ALT-C", will start the capture mode. Then one will be asked the name of the log file. Type "c:xkk.dat", and press "ENTER", to open the file on drive C. The command line "list -n ... " will reappear. Press "ENTER" to start the transfer and the capture. Press "ALT-C"at the end of the file listing, to close the log file.

After doing similar procedure for the files XDIFF.DAT and PKK.DAT, leave the account on the CIE by typing "logout".

Leave the SIMTERM program and drop the line, pressing "ALT-F2". One will be back to drive A on the IBM.

Type "c:", to change logged drive to C .

One may use his preferred text editor to eliminate the undesired lines in the files XKK.DAT, XDIFF.DAT, and PKK.DAT. The files should contain only four columns of numbers, except for the first row, which is blank. Then one can use the routine PLOT to obtain the desired plots, by typing "PLOT =8000", and following the menu options.

APPENDIX E

KR.C PROGRAM LISTING - IBM-PC VERSION

This Appendix contains a listing of the Extended Kalman Filter program implemented in the IBM-PC microcomputer, as well as the listings of the Trajectory Simulation routine and the header file "kalman.h", which are used by the main program KR.C . The other header files declared at the begining of the main program are standard files contained in the C compilers.

```
/*************************************************************/
/*                                                           */
/*                  HEADER FILE - KALMAN.H                   */
/*                                                           */
/*************************************************************

#define nmax 5              /* # of col. in pkk, q ... */
#define maxsq 25            /* size of equiv. vector PKK,Q...*/
#define js 4                /* # hydroph. in any hydrophone */
                            /* array */
#define hcmax 12            /* # of col. in matrix HYDRO of */
                            /* hydroph. coord. */
#define hmax 72             /* equiv. vector size of HYDRO */
#define dmax 3
#define itime 150           /* max.# of time slots */
#define npt   150
#define icmax 5             /* max.# of optimal iterations */
                            /* allowed */
#define criter 0.1          /* criteria to compare ZDIFAV */
#define na 0                /* may assume ONLY values 0 or 6 */
                            /* 0 =>single;6=> multiple arrays */
#define xpos 10000.0        /* initial torpedo x-coordinate */
#define xvel -50.0          /* initial torpedo x-velocity */
#define ypos  0.0           /* initial torpedo y-position */
#define yvel   5.0          /* initial torpedo y-velocity */
#define zpos  0.0           /* initial torpedo z-position */
#define sw0  33000.0        /* swi define points where    */
#define sw1  27000.0        /* switching from one to      */
#define sw2  21000.0        /* another                    */
#define sw3  15000.0        /*         array              */
#define sw4  9000.0         /*                 is done.   */
#define sigdiv 1.0
#define scc 0.387463094 /*  - in radians...           */
#define sacc 36.2
#define vel 4860.0          /*  speed of sound           */
#define twopi 6.28315308
#define rrvar 1.e-5
#define rvar  1.e-10
#define pkkdiag 1000.0
#define  limvar  1.5          /* defines threshold to use Q*/
                            /* constant  */
#define gm0 0.86
#define gm1 1.31
#define cw0 1.0             /* initial was 1036.84 */
#define cw1 1.0             /* initial was 1036.84 */
#define cw2 1.0
#define xminl -5000.0
#define xminh  0.0
#define xmaxl  5000.0
#define xmaxh  40000.0
```

```
#define yminl -5000.0
#define yminh -5000.0
#define ymaxl  5000.0
#define ymaxh  5000.0

/*************************************************************/
/*                                                           */
/*                  DEFINE ANSI DRIVER                       */
/*                                                           */
/*************************************************************

/*  #define STDIN 0  */
/*  #define STDOUT 1 */
/*  #define STDERR 2 */
#define CLR_SCRN fputs("\33[2J", stderr)
#define  CUR_MV(row,col)  fprintf(stderr, "\33[%d;%dH", row,
col)
#define CUR_UP(num) fprintf(stderr,"\33[%dA", num)
#define CUR_DN(num) fprintf(stderr,"\33[%dB", num)
#define CUR_RT(num) fprintf(stderr,"\33[%dC", num)
#define CUR_LT(num) fprintf(stderr,"\33[%dD", num)
#define CUR_SKIP fputs("\n", stderr)
#define CUR_SAVE fputs("\33[s", stderr)
#define CUR_REST fputs("\33[u", stderr)
#define CLR_LINE fputs("\33[K", stderr)
#define BELL fputc('\7', stderr)
#define CLR_EOS(r, c) {byte i_; CUR_MV(r,c); \
   for    (i_=r;    i_<=25;++i_)   CLR_LINE,   CUR_DN   (1);
CUR_MV(r,c); }
```

```
/*************************************************************/
/*                                                           */
/*     REAL TIME KALMAN FILTER PROGRAM - KR.C                */
/*                                                           */
/*************************************************************

#include "stdio.h"
#include "math.h"
#include "kalman.h"
double
xkkm1[nmax],pkkm1[maxsq],pkk[maxsq],xkk[nmax],data[dmax];


      /* STARTING THE MAIN PROGRAM... */
main()
      {

/*************************************************************/
/*                                                           */
/*                   VARIABLE DECLARATIONS                   */
/*                                                           */
/*************************************************************

      FILE *fpxyz, *fopen();
      FILE *fpe, *fopen();
      FILE *fpp, *fopen();
      FILE *fptrj, *fopen();
      FILE *fpzic, *fopen();
      FILE *fparam, *fopen();
      FILE *fpin, *fopen();

      int ihours,imins,isecs,iths,col,row,maxopt;
      int i8,i,j,kk,iin,icont,icurve,option;
      int param,trjopt,nrange;
      int ii,jj,narray,nzdiff,in,p1i,runn,c;
      int jstart,jend,worg;
      float rabs,zgate;
      double height,width,ymax,ymin,xmax,xmin;
      double xscale,yscale,bx,by;
      double hours,mins,secs,ths,time,oldtime,deltat;
      double sqrt(),oldx,value,x0,y0,z0,vx0,vy0;
      double denom,a14,gdenom,gate,zdifav;
      double den,velm,temp;
      double a2,g1,g2,g3,p1,xdiff[3];
      double zhat,zcs,dk,sigcc,sigacc;
      double v,trued[3],wd,g;
      double x,y,z,auxh,xd,yd,zd;
      double a1,e1,e12,e2,bb,dd,bd;
      double pk1,pk3,pk5,pgate,rgate,p,p11;
      double phipkk[maxsq],pktemp[maxsq],q[maxsq];
```

```
      double pi[maxsq],pdum[maxsq],qtemp[maxsq];
      double gnum[nmax],gi[nmax],xi[nmax],hrow[nmax];
      double zic[js],zi[js],zdiff[js],r[js],phis[js],zt[js];
      double xb[js],yb[js],zb[js],rr[js];
      double hydro[hmax],covw[3],gamma[2],opt[5];

/*************************************************************/
/*                                                           */
/*                    INITIALIZATIONS                        */
/*                                                           */
/*************************************************************/

      fpxyz=fopen("xkk.dat","w");
      fpe=fopen("xdiff.dat","w");
      fpp=fopen("pkk.dat","w");

      /*  INITIALIZE CONSTANTS, HYDROPHONE MATRIX, PHI, R,
      DATA   FOR   TRAJECTORY,  AND  LOAD  X(0/-1),  P(0/-1)
*/
      nzdiff=4;  icont=0;  icurve=1; value=criter; wd=5.730;
      runn=1; narray=na; param=0; worg=1; g=0.156; nrange=0;
      jstart=40; jend=80; i8=0;     maxopt=icmax;

      hydro[0]=36000.0;  hydro[1]= 6000.0;  hydro[2]=   0.0;
      hydro[3]=36030.0;  hydro[4]= 6000.0;  hydro[5]=   0.0;
      hydro[6]=36000.0;  hydro[7]= 6030.0;  hydro[8]=   0.0;
      hydro[9]=36000.0;  hydro[10]=6000.0;  hydro[11]= 30.0;

      hydro[12]=30000.0; hydro[13]=6000.0;  hydro[14]=  0.0;
      hydro[15]=30030.0; hydro[16]=6000.0;  hydro[17]=  0.0;
      hydro[18]=30000.0; hydro[19]=6030.0;  hydro[20]=  0.0;
      hydro[21]=30000.0; hydro[22]=6000.0;  hydro[23]= 30.0;

      hydro[24]=24000.0; hydro[25]=6000.0;  hydro[26]=  0.0;
      hydro[27]=24030.0; hydro[28]=6000.0;  hydro[29]=  0.0;
      hydro[30]=24000.0; hydro[31]=6030.0;  hydro[32]=  0.0;
      hydro[33]=24000.0; hydro[34]=6000.0;  hydro[35]= 30.0;

      hydro[36]=18000.0; hydro[37]=6000.0;  hydro[38]=  0.0;
      hydro[39]=18030.0; hydro[40]=6000.0;  hydro[41]=  0.0;
      hydro[42]=18000.0; hydro[43]=6030.0;  hydro[44]=  0.0;
      hydro[45]=18000.0; hydro[46]=6000.0;  hydro[47]= 30.0;

      hydro[48]=12000.0; hydro[49]=6000.0; hydro[50]=   0.0;
      hydro[51]=12030.0; hydro[52]=6000.0; hydro[53]=   0.0;
      hydro[54]=12000.0; hydro[55]=6030.0; hydro[56]=   0.0;
      hydro[57]=12000.0; hydro[58]=6000.0; hydro[59]=  30.0;

      hydro[60]=6000.0;  hydro[61]=6000.0; hydro[62]=   0.0;
      hydro[63]=6030.0;  hydro[64]=6000.0; hydro[65]=   0.0;
      hydro[66]=6000.0;  hydro[67]=6030.0; hydro[68]=   0.0;
```

```
hydro[69]=6000.0;  hydro[70]=6000.0; hydro[71]=  30.0;

r[0]=rvar; r[1]=rvar; r[2]=rvar; r[3]=rvar;

gamma[0]=gm0; gamma[1]=gm1;
covw[0]=cw0;  covw[1]=cw1;  covw[2]=cw2;

x0=xpos; vx0=xvel; y0=ypos; vy0=yvel;
z0=zpos;

for (i=0;i<nmax;i++)
   for (j=0;j<nmax;j++) pkkm1[i*nmax+j]=0.0;
for (i=0;i<nmax;i++) pkkm1[i*nmax+i]=pkkdiag;

phis[0]=1.31; phis[1]=0.0; phis[2]=1.31; phis[3]=0.0;

xb[0]=(-15.0);  yb[0]=(-15.0);  zb[0]=(-15.0);
xb[1]= 15.0;    yb[1]=(-15.0);  zb[1]=(-15.0);
xb[2]=(-15.0);  yb[2]= 15.0;    zb[2]=(-15.0);
xb[3]=(-15.0);  yb[3]=(-15.0);  zb[3]= 15.0;


/* INITIALIZATION FOR QFIND */
velm=1.0/vel;
a14=phis[0];
a2=a14*a14;
g1=a2*0.5;
g2=g1*g1;
g3=a14*g1;
sigacc=sacc*sacc;
sigcc=scc*scc;
for (i=0;i<nmax;i++)
    for (j=0;j<nmax;j++) q[i*nmax+j]=0.0;
q[maxsq-1]=sigdiv*a2*sigdiv;

/* INITIALIZATION FOR HROW */
hrow[1]=0.0; hrow[3]=0.0;


/* INITIALIZATION FOR PDUM */
for (ii=0;ii<nmax;ii++)
   {
   jj=ii*nmax;
   pdum[jj+1]=0.0;
   pdum[jj+3]=0.0;
   }
pdum[6]=1.0; pdum[18]=1.0;

/* INITIALIZING RR */
for (ii=0;ii<4;ii++)  rr[ii]=sqrt(r[ii]);
```

```
/*************************************************************/
/*                                                           */
/*                      DATA  INPUT                          */
/*                                                           */
/*************************************************************/
restart:
      xkkm1[0]=x0; xkkm1[1]=vx0; xkkm1[2]=y0; xkkm1[3]=vy0;
      xkkm1[4]=z0;


menu: /* OPTIONS MENU */
      printf("\n");
      printf("\n           KALMAN FILTER ");
      printf("\n");

      printf("\n           OPTIONS :");
      printf("\n");
      printf("\n      1)   RUN TRAJECTORY");
      printf("\n");
      printf("\n      2)   RUN KALMAN");
      printf("\n");
      printf("\n      3)   MODIFY PARAMETERS");
      printf("\n");
      printf("\n      4)   QUIT");
      printf("\n");
      if (param!=0)
         {
         printf("\n");
         printf("\nA parameter was changed ... You may ");
         printf("have to run TRAJECTORY before continue.");
         }
      printf("\nEnter value (1-4) : ");
      scanf("%d",&option);
      if (option==1)
         {
         fpin=fopen("trj.inp","w");
         fprintf(fpin,"%d %d",narray,runn);
         fprintf(fpin,"\n%e                %e                %e",
                xkkm1[0],xkkm1[1],xkkm1[2]);
         fprintf(fpin,"\n%e %e",xkkm1[3],xkkm1[4]);
         fclose(fpin);

trajec(narray,hydro,xb,yb,zb,xkkm1,jstart,jend,wd,g,worg);
         param=0;
         goto menu;
         }
      else if (option==2) goto run;
      else if (option==3) goto modify;
      else if (option==4) goto quit;
      else goto menu;
```

```
modify:
      printf("\n           MODIFY PARAMETERS OPTIONS :");
      printf("\n");
      printf("\n      1)  ZDIFAV stop optimization");
      printf(" criteria");
      printf("\n      2)  SINGLE / MULTIPLE ARRAY");
      printf("\n      3)  INITIAL X, Y, and Z positions");
      printf("\n      4)  INITIAL Vx and Vy velocities");
      printf("\n      5)  NOISE COVARIANCE - diag_elem of");
      printf(" R");
      printf("\n      6)  MAX. # ITERATIONS OF OPTIMAL);
      printf(" MANEUVER");
      printf("\n      7)  REDEFINE MANEUVER - w / g / time);
      printf(" interval");
      printf("\n      8)  USE LOW / HIGH RANGE ");
      printf("\n");
      printf("\n      9)  MAIN MENU - OPTIONS");
      printf("\n");
      printf("\nEnter value (1-9) : ");
      scanf("%d",&param);

      switch(param)
         {
         case 1:
            printf("\n   %s   %e","Current   CRITERIA  is:",
           value);
            printf("\nchange ? [Y/N] :");
            scanf("%s",&c);
            if (c=='n' || c=='N') break;
            printf("\nEnter new value (real) : ");
            scanf("%lf",&value);
            break;

         case 2:
            printf("\nSINGLE array is the default");
            if (narray==0)
               printf("\n%s","Current case is : SINGLE");
            else printf("\n%s","Current case is: MULTIPLE");
            printf("\nchange ? [Y/N] :");
            scanf("%s",&c);
            if (c=='n' || c=='N') break;
            printf("\nEnter S for SINGLE or");
            printf(" M for MULTIPLE array : ");
            scanf("%s",&c);
            if (c=='s' || c=='S')  narray=0;
            else if (c=='m' || c=='M') narray=1;
            else narray=0;
            break;
```

```
case 3:
   printf("\nCurrent values are : ");
   printf("\n%s %e %s %e %s %e","xpos = ",xkkm1[0],
        "  ypos = ",xkkm1[2],"  zpos = ",xkkm1[4]);
   printf("\nchange ? [Y/N] :");
   scanf("%s",&c);
   if (c=='n' || c=='N') break;
   printf("\nEnter new values : ");
   printf("\n%s","xpos = ");
   scanf("%lf",&x0);

   printf("\n%s","ypos = ");
   scanf("%lf",&y0);

   printf("\n%s","zpos = ");
   scanf("%lf",&z0);

   xkkm1[0]=x0;  xkkm1[2]=y0; xkkm1[4]=z0;

   break;

case 4:
   printf("\nCurrent values are : ");
   printf("\n%s %e %s %e","xvel = ",xkkm1[1],
        "  yvel = ",xkkm1[3]);
   printf("\nchange ? [Y/N] :");
   scanf("%s",&c);
   if (c=='n' || c=='N') break;
   printf("\nEnter new values : ");
   printf("\n%s","xvel = ");
   scanf("%lf",&vx0);

   printf("\n%s","yvel = ");
   scanf("%lf",&vy0);

   xkkm1[1]=vx0; xkkm1[3]=vy0;

   break;

case 5:
   printf("\nCurrent values are : ");
   printf("\n%s %e %s %e %s %e %s %e","r[0] = ",
          r[0]," r[1] = ",r[1]," r[2] = ",r[2],
          " r[3] = ", r[3]);
   printf("\nchange ? [Y/N] :");
   scanf("%s",&c);
   if (c=='n' || c=='N') break;
   printf("\nEnter new values : ");
   printf("\n%s","r[0] = ");
   scanf("%lf",&r[0]);
```

```
            printf("\n%s","r[1] = ");
            scanf("%lf",&r[1]);

            printf("\n%s","r[2] = ");
            scanf("%lf",&r[2]);

            printf("\n%s","r[3] = ");
            scanf("%lf",&r[3]);

            for (ii=0;ii<4;ii++)  rr[ii]=sqrt(r[ii]);
            break;

         case 6:
            printf("\nCurrent value is : ");
            printf("\n%s %d","# of iterations = ",maxopt);
            printf("\nchange ? [Y/N] :");
            scanf("%s",&c);
            if (c=='n' || c=='N') break;
            printf("\nEnter new value : ");
            printf("\n%s","maxopt # = ");
            scanf("%d",&maxopt);
            break;

         case 7:
trmenu:  printf("\n          DEFINE MANEUVER \n");
            printf("\n          1) Define maneuver based");
            printf("on  turn rate");
            printf("\n          2) Define maneuver based");
            printf(" on g-number");
            printf("\n          3) Redefine start and end");
            printf(" time slots");
            printf("\n          4) Modify menu");
            printf("\n\nOption : ");
            scanf("%d",&trjopt);
            if (trjopt==4) break;
            if (trjopt==1)
               {
               printf("\nCurrent value is : ");
               printf("%s %e","turn rate : w = ",wd);
               worg=1;
               printf("\n     Enter new turn rate (dg/s):");
               scanf("%lf",&wd);
               }
            else if (trjopt==2)
               {
               printf("\nCurrent value is : ");
               printf("%s %e","g-number  : g = ",g);
               worg=0;
               printf("\n     Enter new g-number : ");
               scanf("%lf",&g);
               }
```

```
            else if (trjopt==3)
              {
              printf("\nCurrent values are : ");
              printf("\n%s %d %s %d","tstart =  ",
              printf("jstart,"tend = ",jend);
              printf("\n     Enter start time slot : ");
              scanf("%d",&jstart);
              printf("\n     Enter end time slot : ");
              scanf("%d",&jend);
              }
           goto trmenu;

        case 8:
           printf("\nLOW range is the default");
           printf(" (use single array)");
           if (nrange==0)
              printf("\n%s","Current case is : LOW");
           else printf("\n%s","Current case is : HIGH");
           printf("\nchange ? [Y/N] :");
           scanf("%s",&c);
           if (c=='n' || c=='N') break;
           printf("\nEnter L for LOW or");
           printf(" H for HIGH range : ");
           scanf("%s",&c);
           if (c=='l' || c=='L')  nrange=0;
           else if (c=='h' || c=='H') nrange=1;
           else nrange=0;
           break;

        case 9:
           goto menu;
           break;

        default :
           break;
        }
        goto modify;

run:  /* START EXECUTION */
      printf("\n");
      printf("\nEnter run # : ");
      scanf("%d",&runn);

      fparam=fopen("param.dat","w");

 /*   fprintf(fpxyz,"%2d",runn);
      fprintf(fpe,"%2d",runn);
      fprintf(fpp,"%2d",runn);
      fprintf(fparam,"%2d",runn);
```

```
      if (narray==0)
         {
         fprintf(fpxyz,"%s","  Single");
         fprintf(fpe,"%s","  Single");
         fprintf(fpp,"%s","  Single");
         fprintf(fparam,"%s","  Single");
         }
      else
         {
         fprintf(fpxyz,"%s","  Mult");
         fprintf(fpe,"%s","  Mult");
         fprintf(fpp,"%s","  Mult");
         fprintf(fparam,"%s","  Mult");
         }                                    */

      fprintf(fparam,"\n %s %e","CRITERIA = :",value);
      fprintf(fparam,"\n%s %e %s %e %s %e","xkkm1[0] =  ",
              xkkm1[0],"  xkkm1[1] = ",xkkm1[1],
              "  xkkm1[2] = ",xkkm1[2]);
      fprintf(fparam,"\n%s %e %s %e","xkkm1[3] = ",xkkm1[3],
              "  xkkm1[4] = ",xkkm1[4]);
      fprintf(fparam,"\n%s %e %s %e %s %e %s %e",
              "r[0] = ",r[0]," r[1] = ",r[1]," r[2] =  ",
               r[2]," r[3] = ",r[3]);
      fprintf(fparam,"\n%s %e %s %e %s %e","pkkm1[0] =  ",
             pkkm1[0],"  pkkm1[1] = ",pkkm1[1],
              "  pkkm1[2] = ",pkkm1[2]);
      fclose(fparam);

      fpin=fopen("trj.inp","w");

      fprintf(fpin,"%d %d",narray,runn);
      fprintf(fpin,"\n%e %e %e",xkkm1[0],xkkm1[2],
              xkkm1[4]);/*positions*/
      fprintf(fpin,"\n%e  %e",xkkm1[1],xkkm1[3]);  /*vel*/
      fclose(fpin);
      if (option==4) goto quit;

      fptrj=fopen("trj.dat","r");
      fpzic=fopen("zic.dat","r");

wait: printf("\nPress  g and ENTER to start ... ");
      scanf("%s",&c);
      if (c!='g' && c!='G') goto wait;
/*
      if (nrange==0)
         {
         xmin=xminl;  xmax=xmaxl;
         ymin=yminl;  ymax=ymaxl;
         }
```

```
      else
         {
         xmax=xmaxh;   xmin=xminh;
         ymax=ymaxh;   ymin=yminh;
         }

      height=ymax-ymin;
      width=xmax-xmin;
      xscale=600.0/width;
      yscale=-160.0/height;
      by=196.0-ymin*yscale+0.5;
      bx=600.0-xmax*xscale+0.5;

      biosini(6);
      frame(36,0,196,600);
      biosset(23,1);
      printf("%.1f%s%.1f",xmin,",",ymin);
      biosset(6,61);
      printf("%.1f%s%.1f",xmax,",",ymax);
      biosset(22,65);
      printf("%s %d","# ",runn);
      biosset(23,65);
      if (narray==0) printf("SINGLE");
      else printf("MULTIPLE");                    */

      printf("\nwait ");

/*************************************************************/
/*                                                           */
/*                    KALMAN FILTER                          */
/*                                                           */
/*************************************************************/

      /* START THE TIME SLOT LOOP AND SET ARRAY HANDOFF */
      /* POINT */
      for (kk=0;kk<itime;kk++)
         {
         dk=kk+1;

         do
            {
            dostime(&ihours,&imins,&isecs,&iths);
            hours=ihours; mins=imins; secs=isecs; ths=iths;
            time=3600.*hours+60.*mins+secs+0.01*ths;
            deltat=time-oldtime;
            }
         while (deltat < 1.25 && kk>0);

         oldtime=time;
         if (deltat>1.31) printf("*");
         else printf(".");
```

```
/*        printf("\n%s %e","t = ",oldtime);*/
/*        printf("%s %d","  kk = ",kk);     */
/*        printf("\n**********************************");*/
/*        printf("%20s %d \n","************ TIME :",kk); */

          /* GET HYDROPHONE ARRAY COORDINATES */
          if (narray != 0)
             {
             if (xkkm1[0] > sw0) i8=0;
             else if (xkkm1[0] > sw1) i8=12;
             else if (xkkm1[0] > sw2) i8=24;
             else if (xkkm1[0] > sw3) i8=36;
             else if (xkkm1[0] > sw4) i8=48;
             else i8=60;
             for (i=0;i<4;i++)
                {
                j=3*i+i8;
                xb[i]=hydro[j];
                yb[i]=hydro[j+1];
                zb[i]=hydro[j+2];
                }
             }

          /* GET THE TRUE TIMES AND THE TRUE POSITIONS */
          fscanf(fptrj,"%lf %lf %lf",
                 &data[0],&data[1],&data[2]);
          fscanf(fpzic,"%lf %lf %lf %lf",
                 &zic[0],&zic[1],&zic[2],&zic[3]);
          trued[0]=data[0]; trued[1]=data[1];
          trued[2]=data[2];

          /* FIRST GET HROW-CALCULATE GAIN, ESTIMATE */
          /* COVARIANCE OF ERROR BASED ON ONE TIME   */
          /* MEASUREMENT-TC,TX,TY,TZ                 */
    adapt: for (i=0;i<js;i++)
             {
             x=xkkm1[0]-xb[i];
             y=xkkm1[2]-yb[i];
             z=xkkm1[4]-zb[i];

             denom=sqrt(x*x+y*y+z*z);
             auxh=velm/denom;
             hrow[0]=x*auxh;
             hrow[2]=y*auxh;
             hrow[4]=z*auxh;

             for (ii=0;ii<nmax;ii++)
                {
                iin=ii*nmax;
```

```
            gnum[ii]=pkkm1[iin]*hrow[0]+
              pkkm1[iin+2]*hrow[2]+pkkm1[iin+4]*hrow[4];
            }
            gdenom=hrow[0]*gnum[0]+
                   hrow[2]*gnum[2]+
                   hrow[4]*gnum[4]+r[i];

          /* THIS IS THE FIRST GAIN COLUMN  */
          for (ii=0;ii<nmax;ii++) gi[ii]=gnum[ii]/gdenom;

/*        printf("\nVECTOR HROW : \n");
          printvec(hrow,5);
          printf("\nVECTOR GNUM : \n");
          printvec(gnum,5);
          printf("\nDENOMINATOR  is : ");
          printf("%e \n",gdenom);
          printf("\nVECTOR GI : \n");
          printvec(gi,5);   */

          /* CALCULATE THE COVARIANCE OF ERROR PI */
          for (ii=0;ii<nmax;ii+=2)
            {
            pdum[ii]=(-gi[0]*hrow[ii]);
            pdum[ii+5]=(-gi[1]*hrow[ii]);
            pdum[ii+10]=(-gi[2]*hrow[ii]);
            pdum[ii+15]=(-gi[3]*hrow[ii]);
            pdum[ii+20]=(-gi[4]*hrow[ii]);
            }
          pdum[0]+=1.0;
          pdum[12]+=1.0;
          pdum[24]+=1.0;
          pi[0]   =pdum[0]*pkkm1[0]+pdum[2]*pkkm1[10]+
                   pdum[4]*pkkm1[20];
          pi[1]   =pdum[0]*pkkm1[1]+pdum[2]*pkkm1[11]+
                   pdum[4]*pkkm1[21];
          pi[2]   =pdum[0]*pkkm1[2]+pdum[2]*pkkm1[12]+
                   pdum[4]*pkkm1[22];
          pi[3]   =pdum[0]*pkkm1[3]+pdum[2]*pkkm1[13]+
                   pdum[4]*pkkm1[23];
          pi[4]   =pdum[0]*pkkm1[4]+pdum[2]*pkkm1[14]+
                   pdum[4]*pkkm1[24];
          pi[5]=pi[1];
          pi[6] =pdum[5]*pkkm1[1]+pkkm1[6]+
                   pdum[7]*pkkm1[11]+pdum[9]*pkkm1[21];
          pi[7] =pdum[5]*pkkm1[2]+pkkm1[7]+
                   pdum[7]*pkkm1[12]+pdum[9]*pkkm1[22];
          pi[8] =pdum[5]*pkkm1[3]+pkkm1[8]+
                   pdum[7]*pkkm1[13]+pdum[9]*pkkm1[23];
          pi[9] =pdum[5]*pkkm1[4]+pkkm1[9]+
                   pdum[7]*pkkm1[14]+pdum[9]*pkkm1[24];
          pi[10]=pi[2];
```

```
            pi[11]=pi[7];
            pi[12]=pdum[10]*pkkm1[2]+pdum[12]*pkkm1[12]+
                      pdum[14]*pkkm1[22];
            pi[13]=pdum[10]*pkkm1[3]+pdum[12]*pkkm1[13]+
                      pdum[14]*pkkm1[23];
            pi[14]=pdum[10]*pkkm1[4]+pdum[12]*pkkm1[14]+
                      pdum[14]*pkkm1[24];
            pi[15]=pi[3];
            pi[16]=pi[8];
            pi[17]=pi[13];
            pi[18]=pdum[15]*pkkm1[3]+pdum[17]*pkkm1[13]+
                      pkkm1[18]+pdum[19]*pkkm1[23];
            pi[19]=pdum[15]*pkkm1[4]+pdum[17]*pkkm1[14]+
                      pkkm1[19]+pdum[19]*pkkm1[24];
            pi[20]=pi[4];
            pi[21]=pi[9];
            pi[22]=pi[14];
            pi[23]=pi[19];
            pi[24]=pdum[20]*pkkm1[4]+pdum[22]*pkkm1[14]+
                      pdum[24]*pkkm1[24];

            /* CALCULATE FIRST MEASUREMENT PREDICTION */
            zhat=velm*denom;
            zdiff[i]=zic[i]-zhat;

            /* COMPUTE THE GATE FOR ERRONEOUS TIME */
            /* MEASUREMENTS */
            pk1=(pi[0] < 0) ? -pi[0] : pi[0];
            pk3=(pi[12] < 0) ? -pi[12] : pi[12];
            pk5=(pi[24] < 0) ? -pi[24] : pi[24];
            p11=(pk1 > pk3) ? pk1 : pk3;
            p = (p11 > pk5) ? p11 : pk5;

/*          pgate=sqrt(p)*velm;*/
            x=p+r[i];
            if (x != 0.0)
               {
               if (x < 1.0) x=1.0;
               do
                  {
                  oldx=x;
                  x=( x + p/x )*0.5;
                  temp=oldx-x;
                  rabs=(temp < 0) ? -temp : temp;
                  }
               while (rabs > 0.0);
               }

            pgate=x*velm;
            gate=3.0*pgate;
```

```
/*          printf("\nMATRIX PDUM : \n");
            printmat(pdum,5,5);
            printf("\nMATRIX PI : \n");
            printmat(pi,5,5);
            printf("\nZDIFF  is : ");
            printf("\%e \n",zdiff[i]);
            printf("\nGATE  is : ");
            printf("\%e \n",gate);   */

            /* EDIT INVALID TIME MEASUREMENTS */
            zgate=(zdiff[i] < 0) ? -zdiff[i] : zdiff[i];
            if (zgate >= gate)
               {
               for (ii=0;ii<nmax;ii++) gi[ii]=0.0;
               zdiff[i]=999.0;
                     /* TAG FOR INVALID TIME MEASUREMENT */
               }

            /* CALCULATE ESTIMATE BASED ON ONE MEASUREMENT*/
            /* PREDICTION */
            for (ii=0;ii<nmax;ii++)
                 xi[ii]=xkkm1[ii]+gi[ii]*zdiff[i];

/*          printf("\nVECTOR XI : \n");
            printvec(xi,5);    */

            for (ii=0;ii<nmax;ii++)
                {
                xkkm1[ii]=xi[ii];
                pkkm1[ii]=pi[ii];
                }
            for (jj=nmax;jj<maxsq;jj++) pkkm1[jj]=pi[jj];
            if (i == 3) goto 156;

            }  /* end JS loop */

         /* NOTE : CALLED ORIGINAL X(0/-1),XKKM1. UPDATED */
         /* AFTER 1 MEASUREMENT CALLED IT XI, THEN MADE   */
         /* XKKM1  AND WENT THRU ITERATION AGAIN. AFTER   */
         /* YOU HAVE UPDATED XKKM1 FOR EACH MEASUREMENT   */
         /* XKK=XI AND PKK=PI.                           */
    156: for (ii=0;ii<nmax;ii++)
            {
            xkk[ii]=xi[ii];
            pkk[ii]=pi[ii];
            }
         for (jj=nmax;jj<maxsq;jj++)
            {
            pkk[jj]=pi[jj];
            }
```

```
/*        printf("\nUPDATING VECTORS INTERVAL ... \n");
          printf("\nVECTOR XKKM1 : \n");
          printvec(xkkm1,5);
          printf("\nMATRIX PKKM1 : \n");
          printmat(pkkm1,5,5);

          printf("\nVECTOR XKK : \n");
          printvec(xkk,5);
          printf("\nMATRIX PKK : \n");
          printmat(pkk,5,5);      */

          /* RECALCULATE TIME MEASUREMENTS, and FORM */
          /* ABSOLUTE VALUE OF RESIDUALS.            */
          for (i=0;i<4;i++)

          /*  EDIT  INVALID  TIMES  FOR ADAPTIVE MANEUVER  */
     /* ROUTINE */
               {
               if (zdiff[i] < 999.0)
                  {
                  x=xkkm1[0]-xb[i];
                  y=xkkm1[2]-yb[i];
                  z=xkkm1[4]-zb[i];

                  /* denom=sqrt(x*x+y*y+z*z); */
                  denom=x*x+y*y+z*z;
                  x=denom;
                  if (x != 0.0)
                     {
                     if (x < 1.0) x=1.0;
                     do
                        {
                        oldx=x;
                        x=( x + denom/x )*0.5;
                        temp=oldx-x;
                        rabs=(temp < 0) ? -temp : temp;
                        }
                     while (rabs >= 1.e-12);
                     }

                  denom=x;
                  zhat=velm*denom;
                  temp=zic[i]-zhat;
                  zdiff[i]=(temp < 0) ? -temp : temp;
                  }
               else
                  {
                  zdiff[i]=0.0;
                  nzdiff-=1;
                  }
               }
```

```
        /* THIS FUNCTION COMPUTES THE ADAPTIVE Q MATRIX */
        if (icurve == 1)
            {
            bb=xkkm1[3]*xkkm1[3];
            dd=xkkm1[1]*xkkm1[1];
            a1=bb+dd;
            bd=xkkm1[1]*xkkm1[3];
            e1=dd*sigacc/a1+bb*sigcc;
            e12=bd*sigacc/a1-bd*sigcc;
            e2=bb*sigacc/a1+dd*sigcc;
            q[0]=g2*e1;
            q[1]=g3*e1;
            q[2]=g2*e12;
            q[3]=g3*e12;
            q[6]=a2*e1;
            q[7]=q[3];
            q[8]=a2*e12;
            q[12]=g2*e2;
            q[13]=g3*e2;
            q[18]=a2*e2;
            }
        else
            {
            bb=gamma[0]*gamma[0];
            dd=gamma[1]*gamma[1];
            q[0]=bb*covw[0];
            q[1]=gamma[0]*covw[0]*gamma[1];
            q[2]=0.0; q[3]=0.0;
            q[6]=dd*covw[0];
            q[7]=0.0; q[8]=0.0;
            q[12]=bb*covw[1];
            q[13]=gamma[0]*covw[1]*gamma[1];
            q[18]=dd*covw[1];
            }
            q[5]=q[1];  q[10]=q[2];  q[11]=q[7];
            q[15]=q[3]; q[16]=q[8];  q[17]=q[13];

 /*       printf("\nMATRIX Q : \n");
          printmat(q,5,5);    */

        /* IF ALL TIME MEASUREMENTS EXCEED GATE, BYPASS */
        /*  ADAPTIVE MANEUVER ROUTINE.                  */
         if (nzdiff != 0.0)
            {

zdifav=(zdiff[0]+zdiff[1]+zdiff[2]+zdiff[3])/nzdiff;
/*          printf("\nZDIFAV is : ");
            printf("\%e \n",zdifav);  */
```

```
            /* DO NOT PERFORM ADAPTIVE MANEUVER IF FILTER */
            /* HAS NOT ACHIEVED THE STEADY STATE.         */
            if (kk > 3)
               {
/*             dostime(&ihours,&imins,&isecs,&iths);
               hours=ihours; mins=imins; secs=isecs;
               ths=iths;
               time=3600.*hours+60.*mins+secs+0.01*ths;
               printf("%s %e","  time1 = ",time);        */

               /* IF ZDIFAV MEETS CRITERIA TRANSFER OUT  */
               /* OF ADAPTIVE MANEUVER ROUTINE.          */
               if (zdifav > value && icont < maxopt)
                  {
                  /* INCREASE GAIN */
                  for (j=0;j<maxsq;j++)
                      pkkm1[j]=pkk[j]+q[j];

                /*   printf("\nINCREASE GAIN AND PERFORM");
                     printf(" ADAPTIVE ... \n");
                     printf("\nMATRIX PKKM1 : \n");
                     printmat(pkkm1,5,5);            */

                     /* PERFORM ADAPTIVE MANEUVER BY */
                     /* REITERATING SAME TIME SLOT. */
                     icont+=1;
                     goto adapt;
                     }
               }
            }

         icont=0;
         nzdiff=4;
         xdiff[0]=xkk[0]-trued[0];
         xdiff[1]=xkk[2]-trued[1];
         xdiff[2]=xkk[4]-trued[2];

/*       temp=xdiff[1]*xdiff[1]/pkkm1[12];
         if (temp < limvar) icurve=0;
         else icurve=1;                    */

         fprintf(fpe,"\n %d %e %e %e",kk,
                 xdiff[0],xdiff[1],xdiff[2]);

         /* WRITE */
/*       printf("\nVECTOR ERROR : XDIFF  YDIFF  ZDIFF\n");
         printvec(xdiff,3);  */

         /* PREDICTIONS FOR PKKM1 */

         for (i=0;i<nmax;i++)
```

```
            {
            phipkk[i]=pkk[i]+pkk[i+5]*phis[0];
            phipkk[i+5]=pkk[i+5];
            phipkk[i+10]=pkk[i+10]+pkk[i+15]*phis[2];
            phipkk[i+15]=pkk[i+15];
            phipkk[i+20]=pkk[i+20];
            }

         for (i=0;i<nmax;i++)
            {
            in=i*nmax;
            pktemp[in]=phipkk[in]+phipkk[in+1]*phis[0];
            pktemp[in+1]=phipkk[in+1];
            pktemp[in+2]=phipkk[in+2]+phipkk[in+3]*phis[2];
            pktemp[in+3]=phipkk[in+3];
            pktemp[in+4]=phipkk[in+4];
            }

         for (j=0;j<maxsq;j++) pkkm1[j]=pktemp[j]+q[j];

/*       printf("\nPREDICTIONS FOR PKKM1 ... \n");
         printf("\nMATRIX PKKM1 : \n");
         printmat(pkkm1,5,5);     */

         /* CALCULATE NEW XKKM1 */
         for (ii=0;ii<nmax;ii++)
            {
            xkkm1[ii]=xkk[ii];
            /* p1=sqrt(fabs(pkk[ii])); */
            }
         xkkm1[0]=xkkm1[0]+phis[0]*xkk[1];
         xkkm1[2]=xkkm1[2]+phis[2]*xkk[3];

/*       printf("\nNEW XKKM1 ... (for next time");
         printf(" interval)\n");
         printf("\nVECTOR XKKM1 : \n");
         printvec(xkkm1,5);  */

         fprintf(fpp,"\n %d %e %e %e",kk,
                 pkk[0],pkk[12],pkk[24]);

         fprintf(fpxyz,"\n %d %e %e %e",kk,
                 xkk[0],xkk[2],xkk[4]);
/*
         /* PLOTTING */
         col=xkk[0]*xscale+bx;
         row=xkk[2]*yscale+by;
         bioswd(row,col);

         biosset(3,0);
         printf("%s %.1f","x =",xkk[0]);
```

```
            biosset(3,13);
            printf("%s %.1f","y =",xkk[2]);
            biosset(3,25);
            printf("%s %.1f","z =",xkk[4]);
            biosset(3,42);
            printf("%s %.1f","vx =",xkk[1]);
            biosset(3,54);
            printf("%s %.1f","vy =",xkk[3]);
            biosset(3,65);
            printf("%s %d","time =",kk);          */

            }  /* end TIME loop */

        fclose(fpxyz);
        fclose(fpe);
        fclose(fpp);
        fclose(fpzic);
        fclose(fptrj);
        c=keyrd();
        biosini(3);
quit:
        printf("\nquit ? [Y/N] :");
        scanf("%s",&c);
        if (c=='n' || c=='N') goto restart;

        }  /* end MAIN PROGRAM */
```

```
/*************************************************************/
/*                                                           */
/*     PRINTVEC will print the vector elements given         */
/*     the name and the dimension.                           */
/*                                                           */
/*     PRINTMAT will print the matrix elements in the        */
/*     usual matrix form, given the dimensions.              */
/*                                                           */
/*                                                           */
/*************************************************************/
/*
printvec(vector,dim)
      int dim;
      double vector[];

      {
      int i, j;
      for (j=0;j<dim;j++) printf("%e %1s",vector[j]," ");
      printf("\n");
      }

printmat(matrix,row,col)
      int row,col;
      double matrix[];

      {
      int i,j;
      for (i=0;i<row;i++)
         {
         for (j=0;j<col;j++) printf("%e
              %1s",matrix[i*col+j]," ");
         printf("\n");
         }
      }  */
```

```
/*************************************************************/
/*                                                           */
/*                          FRAME                            */
/* Draw a frame on the screen.                               */
/* *** NOTE ***                                              */
/* The upper left corner of the screen is 0,0 and the        */
/* bottom right corner is 199,639.                           */
/*                                                           */
/*                                                           */
/*************************************************************/

int frame(ulrow,ulcol,lrrow,lrcol)
int ulrow,ulcol,lrrow,lrcol;
{
int count,irow,i,iaux,icol,tcount;

/* do top line first */
for    (icol    =    ulcol;    icol    <=    lrcol;    icol++)
bioswd(ulrow,icol);

/* do both sides at once */
for (irow = ulrow+1; irow < lrrow; irow++)
 {
 bioswd(irow,ulcol);
 bioswd(irow,lrcol);
 }

/* do bottom line */
for    (icol    =    ulcol;    icol    <=    lrcol;    icol++)
bioswd(lrrow,icol);

/* ------------------------------------------------------- */

/* do vertical center line with scale */
i=0.5*(ulcol+lrcol)+0.5;
for (irow = ulrow+1; irow < lrrow; irow++)  bioswd(irow,i);
for (irow = ulrow+16; irow < lrrow; irow+=16)
    for    (tcount=i-4;    tcount    <=    i+4;    tcount++)
bioswd(irow,tcount);

/* do horizontal center line with scale */
i=0.5*(ulrow+lrrow)+0.5;
for (icol = ulcol+1; icol < lrcol; icol++)  bioswd(i,icol);
for (icol = ulcol+60; icol < lrcol; icol+=60)
    for    (tcount=i-3;    tcount    <=    i+3;    tcount++)
bioswd(tcount,icol);

/* ------------------------------------------------------- */

return(0);  /* done */
}
```

```
/*************************************************************/
/*                                                           */
/*      TRAJECTORY SIMULATION ROUTINE - file: TR.C           */
/*                                                           */
/*************************************************************/

#include "stdio.h"
#include "math.h"
#include "kalman.h"
trajec(narray,hydro,xb,yb,zb,xkkm1,jstart,jend,wd,g,worg)
      double hydro[],xb[],yb[],zb[],xkkm1[],wd,g;
      int narray,worg,jstart,jend;
      {
      int p1i,i,start,end;
      double pp1,r1,sqrt(),zcs,zic[4],zi[4];
      double xd,yd,zd,v,den,velm,gg;
      static double xn[6]={-3.01, -2.0, -1.5, 1.0, -0.6,
                           0.0};
      static double yn[6]={0.0, 0.0228, 0.0668, 0.1357,
                           0.2743, 0.5};
      static double sn[5]={43.8596, 11.3636, 7.25689,
                            2.891352, 2.65887};

      double data[5],wr;
      int runn,i8;
      FILE *fptrj, *fopen();
      FILE *fpzic, *fopen();
      int j,jj;

   /* THIS ROUTINE COMPUTES TRUE TRAJECTORY OF TORPEDO
      data[0]=true x position, data[1]= true y position
      data[2]=true z position, w=turn rate
      data[3]=x_velocity, data[4]=y_velocity
      jstart=time slot where maneuver starts
      jend=time slot where maneuver ends                  */

      data[0]=xkkm1[0]; data[1]=xkkm1[2]; data[2]=xkkm1[4];
      data[3]=xkkm1[1]; data[4]=xkkm1[3]; velm=1.0/vel;
      r1=0.49;
      wr=twopi*wd/360.0; gg=32.2*g; start=jstart; end=jend;

      printf("\n %s \n","start execution of trj.c");
      fptrj=fopen("trj.dat","w");
      fpzic=fopen("zic.dat","w");

      for (jj=0;jj<npt;jj++)
         {
         /* GET HYDROPHONE ARRAY COORDINATES */
         if (narray != 0)
```

```
            {
            if (data[0] > sw0)  i8=0;
            else if (data[0] > sw1)  i8=12;
            else if (data[0] > sw2)  i8=24;
            else if (data[0] > sw3)  i8=36;
            else if (data[0] > sw4)  i8=48;
            else i8=60;
            for (i=0;i<4;i++)
               {
               j=3*i+i8;
               xb[i]=hydro[j];
               yb[i]=hydro[j+1];
               zb[i]=hydro[j+2];
               }
            }
         /* COMPUTE THE TRUE TIMES AND THE TRUE POSITIONS
         WRITE TRUE TIME and TRUE POSITIONS */
            printf("%s",".");
            fprintf(fptrj,"\n %e %e %e",
                    data[0],data[1],data[2]);

         for (i=0;i<4;i++)
            {
            xd=data[0]-xb[i];
            yd=data[1]-yb[i];
            zd=data[2]-zb[i];
            den=sqrt(xd*xd+yd*yd+zd*zd);
            zi[i]=velm*den;

   /*       printf("\nTRUE TIMES : \n");
            printvec(zi,4);
            printf("\nTRUE POSITIONS : \n");
            printvec(data,3);  */

            /* GET NOISE,SCALE AND ADD TO TRUE */
            /* MEASUREMENT TIME */
            pp1=317.0*r1;
            p1i=pp1;
            r1=pp1-p1i;
            v=r1;
            j=0;
            if (v > 0.5) v=1.0-r1;
         r2: if (v < yn[j+1]) goto r8;
            j+=1;
            goto r2;
         r8: v=(v-yn[j])*sn[j]+xn[j];
            if (r1 >= 0.5) v=(-v);
            zcs=(v/3.00)*0.00001;

/*          printf("\nRANDOM NUMBER :");
            printf("%e \n",zcs);*/
```

```
            zic[i]=zcs+zi[i];
            }
         fprintf(fpzic,"\n %e %e %e %e",
                  zic[0],zic[1],zic[2],zic[3]);

         /* NEW DATA */
         if (start >= npt) line(jj,0,npt,data);
         else
            {
            if (end > npt) end=npt;
            line(jj,0,start,data);
            circle(jj,start,end,data,wr,gg,worg);
            line(jj,end,npt,data);
            }
         }
      fclose(fptrj);
      fclose(fpzic);
      }


circle(j,start,end,data,ww,gg,worg)
   double data[],ww,gg;
   int j,start,end,worg;
   {
   double sin(),cos(),atan();
   double r,vt,aux,a,b,teta,teta0,m,alfa,wa,ga,g,w;
   g=gg;  w=ww;
   if(j >= start && j < end)
      {
      if (data[3]==0.0 && data[4]==0.0) goto out;
      vt=sqrt(data[3]*data[3]+data[4]*data[4]);
      if (worg==1) g=w*vt;
      else w=g/vt;
      r=vt/w;
      if (data[3]==0.0)
         {
         b=data[1];
         if (data[4]>0.0)
            {
            a=data[0]-r;
            teta0=0.0;
            }
         else
            {
            a=data[0]+r;
            teta0=0.5*twopi;
            }
         }
      else if (data[4]==0.0)
```

```
            {
            a=data[0];
            if (data[3]>0.0)
                {
                b=data[1]+r;
                teta0=-0.25*twopi;
                }
            else
                {
                b=data[1]-r;
                teta0=0.25*twopi;
                }
            }
        else
            {
            m=data[4]/data[3];
            aux=r/sqrt(1.0+m*m);
            if (data[3] > 0.0) aux=-aux;
            b=data[1]-aux;
            a=data[0]+m*(data[1]-b);
            teta0=atan2(-data[3],data[4]);
            }
        teta=1.31*w+teta0;
        data[0]=r*cos(teta)+a;
        data[1]=r*sin(teta)+b;
        data[3]=-vt*sin(teta);
        data[4]=vt*cos(teta);
        if (j==end-1)
            {
            wa=w*360.0/twopi;
            ga=g/32.2;
            printf("\n  ... from TRAJEC : ");
            printf("\n");
            printf("\n%s %e %s","         turn rate : ",wa,
                    " dg/seg");
            printf("\n%s %.2f\n","          g-number  : ",ga);
            }
        out:
        }
    }

line(j,start,end,data)
    double data[];
    int j,start,end;
    {
    if((j >= start) && (j < end))
        {
        data[0]+=1.31*data[3];
        data[1]+=1.31*data[4];
        }
    }
```

APPENDIX F

LISTING OF CIEKR1.C - CIE VERSION

```
/*************************************************************/
/*                                                           */
/*                HEADER FILE - KALMAN.H                     */
/*              CIE  MICROCOMPUTER VERSION                   */
/*                                                           */
/*************************************************************

#define nmax 5              /* # of col. in pkk, q ... */
#define maxsq 25            /* size of equiv. vector PKK,Q...*/
#define js 4                /* # hydroph. in any hydrophone */
                            /* array */
#define hcmax 12            /* # of col. in matrix HYDRO of */
                            /* hydroph. coord. */
#define hmax 72             /* equiv. vector size of HYDRO */
#define dmax 3
#define itime 150           /* max.# of time slots */
#define npt   150
#define icmax 5             /* max.# of optimal iterations */
                            /* allowed */
#define criter 0.1          /* criteria to compare ZDIFAV */
#define na 0                /* may assume ONLY values 0 or 6 */
                            /* 0 =>single;6=> multiple arrays */
#define xpos 10000.0        /* initial torpedo x-coordinate */
#define xvel -50.0          /* initial torpedo x-velocity */
#define ypos  0.0           /* initial torpedo y-position */
#define yvel   5.0          /* initial torpedo y-velocity */
#define zpos  0.0           /* initial torpedo z-position */
#define sw0  33000.0        /* swi define points where    */
#define sw1  27000.0        /* switching from one to      */
#define sw2  21000.0        /* another                    */
#define sw3  15000.0        /*         array              */
#define sw4  9000.0         /*                 is done.   */
#define sigdiv 1.0
#define scc 0.387463094 /*  - in radians...           */
#define sacc 36.2
#define vel 4860.0          /*  speed of sound           */
#define twopi 6.28315308
#define rrvar 1.e-5
#define rvar  1.e-10
#define pkkdiag 1000.0
#define  limvar  1.5          /* defines threshold to use Q*/
                            /* constant  */
#define gm0 0.86
#define gm1 1.31
#define cw0 1.0             /* initial was 1036.84 */
#define cw1 1.0             /* initial was 1036.84 */
#define cw2 1.0
```

```
/*************************************************************/
/*                                                           */
/*        KALMAN FILTER PROGRAM FOR THE CIE MICRO            */
/*                  (FILE  CIEKR1.C)                         */
/*                                                           */
/*************************************************************/

#include "stdio.h"
#include "math.h"
#include "kalman.h"
double
xkkm1[nmax],pkkm1[maxsq],pkk[maxsq],xkk[nmax],data[dmax];

      /* STARTING THE MAIN PROGRAM... */
main()
      {
      FILE *fpxyz, *fopen();
      FILE *fpe, *fopen();
      FILE *fpp, *fopen();
      FILE *fptrj, *fopen();
      FILE *fpzic, *fopen();
      FILE *fparam, *fopen();
      FILE *fpin, *fopen();
      int ihours,imins,isecs,iths,col,row;
      int i8,i,j,kk,iin,icont,icurve,option,param;
      int trjopt,nrange;
      int ii,jj,narray,nzdiff,in,p1i,runn,c;
      int jstart,jend,worg;
      float rabs,zgate;
      double height,width,ymax,ymin,xmax,xmin;
      double xscale,yscale,bx,by;
      double hours,mins,secs,ths,time,oldtime,deltat;
      double sqrt(),oldx,value;
      double denom,a14,gdenom,gate,zdifav;
      double den,velm,temp;
      double a2,g1,g2,g3,p1,xdiff[3];
      double zhat,zcs,dk;
      double v,trued[3],wd,g;
      double x,y,z,auxh,xd,yd,zd,sigcc,sigacc;
      double a1,e1,e12,e2,bb,dd,bd;
      double pk1,pk3,pk5,pgate,rgate,p,p11;
      double phipkk[maxsq],pktemp[maxsq],q[maxsq];
      double pi[maxsq],pdum[maxsq],qtemp[maxsq];
      double gnum[nmax],gi[nmax],xi[nmax],hrow[nmax];
      double zic[js],zi[js],zdiff[js],r[js],phis[js],zt[js];
      double xb[js],yb[js],zb[js],rr[js];
      double hydro[hmax],covw[3],gamma[2],opt[5];

      fpxyz=fopen("xkk.dat","w");
      fpe=fopen("xdiff.dat","w");
      fpp=fopen("pkk.dat","w");
```

```
        /*  INITIALIZE CONSTANTS, HYDROPHONE MATRIX, PHI, R,
        DATA   FOR   TRAJECTORY,  AND  LOAD  X(0/-1),  P(0/-1)
*/
        nzdiff=4;  icont=0;  icurve=1; value=criter; wd=5.730;
        runn=1; narray=na;  param=0;  worg=1;  g=0.156;
        nrange=0;
        jstart=40; jend=80;

        hydro[0]=36000.0;  hydro[1]= 6000.0;  hydro[2]=   0.0;
        hydro[3]=36030.0;  hydro[4]= 6000.0;  hydro[5]=   0.0;
        hydro[6]=36000.0;  hydro[7]= 6030.0;  hydro[8]=   0.0;
        hydro[9]=36000.0;  hydro[10]=6000.0;  hydro[11]= 30.0;

        hydro[12]=30000.0; hydro[13]=6000.0;  hydro[14]=  0.0;
        hydro[15]=30030.0; hydro[16]=6000.0;  hydro[17]=  0.0;
        hydro[18]=30000.0; hydro[19]=6030.0;  hydro[20]=  0.0;
        hydro[21]=30000.0; hydro[22]=6000.0;  hydro[23]= 30.0;

        hydro[24]=24000.0; hydro[25]=6000.0;  hydro[26]=  0.0;
        hydro[27]=24030.0; hydro[28]=6000.0;  hydro[29]=  0.0;
        hydro[30]=24000.0; hydro[31]=6030.0;  hydro[32]=  0.0;
        hydro[33]=24000.0; hydro[34]=6000.0;  hydro[35]= 30.0;

        hydro[36]=18000.0; hydro[37]=6000.0;  hydro[38]=  0.0;
        hydro[39]=18030.0; hydro[40]=6000.0;  hydro[41]=  0.0;
        hydro[42]=18000.0; hydro[43]=6030.0;  hydro[44]=  0.0;
        hydro[45]=18000.0; hydro[46]=6000.0;  hydro[47]= 30.0;

        hydro[48]=12000.0; hydro[49]=6000.0;  hydro[50]=  0.0;
        hydro[51]=12030.0; hydro[52]=6000.0;  hydro[53]=  0.0;
        hydro[54]=12000.0; hydro[55]=6030.0;  hydro[56]=  0.0;
        hydro[57]=12000.0; hydro[58]=6000.0;  hydro[59]= 30.0;

        hydro[60]=6000.0;  hydro[61]=6000.0;  hydro[62]=  0.0;
        hydro[63]=6030.0;  hydro[64]=6000.0;  hydro[65]=  0.0;
        hydro[66]=6000.0;  hydro[67]=6030.0;  hydro[68]=  0.0;
        hydro[69]=6000.0;  hydro[70]=6000.0;  hydro[71]= 30.0;

        r[0]=rvar; r[1]=rvar; r[2]=rvar; r[3]=rvar;

        gamma[0]=gm0; gamma[1]=gm1;
        covw[0]=cw0;  covw[1]=cw1;  covw[2]=cw2;

        xkkm1[0]=1000.0; xkkm1[1]=(-50.0); xkkm1[2]=0.0;
        xkkm1[3]=5.0;  xkkm1[4]=0.0;

        for (i=0;i<nmax;i++)
           for (j=0;j<nmax;j++) pkkm1[i*nmax+j]=0.0;
        for (i=0;i<nmax;i++) pkkm1[i*nmax+i]=pkkdiag;

        phis[0]=1.31; phis[1]=0.0; phis[2]=1.31; phis[3]=0.0;
```

```
      xb[0]=(-15.0);  yb[0]=(-15.0);  zb[0]=(-15.0);
      xb[1]= 15.0;    yb[1]=(-15.0);  zb[1]=(-15.0);
      xb[2]=(-15.0);  yb[2]= 15.0;    zb[2]=(-15.0);
      xb[3]=(-15.0);  yb[3]=(-15.0);  zb[3]= 15.0;

      /* GET DATA INPUT */

/*************************************************************/
/*                                                           */
/*                    DATA  INPUT                            */
/*                                                           */
/*************************************************************/

menu: /* OPTIONS MENU */
      printf("\n");
      printf("\n           KALMAN FILTER ");
      printf("\n");

      printf("\n           OPTIONS :");
      printf("\n");
      printf("\n       1)  RUN TRAJECTORY");
      printf("\n");
      printf("\n       2)  RUN KALMAN");
      printf("\n");
      printf("\n       3)  MODIFY PARAMETERS");
      printf("\n");
      printf("\n       4)  QUIT");
      printf("\n");
      if (param!=0)
         {
         printf("\n");
         printf("\nA parameter was changed ... ");
         printf(" You may have to run");
         printf(" TRAJECTORY before continue.");
         }
      printf("\nEnter value (1-4) : ");
      scanf("%d",&option);
      if (option==1)
         {
         fpin=fopen("trj.inp","w");
         fprintf(fpin,"%d %d",narray,runn);
         fprintf(fpin,"\n%e %e %e",
                 xkkm1[0],xkkm1[1],xkkm1[2]);
         fprintf(fpin,"\n%e %e",xkkm1[3],xkkm1[4]);
         fclose(fpin);

trajec(narray,hydro,xb,yb,zb,xkkm1,jstart,jend,wd,g,worg);
         param=0;
         goto menu;
         }
```

```
      else if (option==2) goto run;
      else if (option==3) goto modify;
      else if (option==4) goto quit;
      else goto menu;

modify:
      printf("\n           MODIFY PARAMETERS OPTIONS :");
      printf("\n");
      printf("\n      1)  ZDIFAV stop optimization");
      printf("criteria");
      printf("\n      2)  SINGLE / MULTIPLE ARRAY");
      printf("\n      3)  INITIAL X, Y, and Z positions");
      printf("\n      4)  INITIAL Vx and Vy velocities");
      printf("\n      5)  NOISE COVARIANCE - diag_elem of");
      printf("R");
      printf("\n      6)  ERROR COVARIANCE - diag_elem of");
      printf(" PKK");
      printf("\n");
      printf("\n      7)  MAIN MENU - OPTIONS");
      printf("\n");
      printf("\nEnter value (1-7) : ");
      scanf("%d",&param);

      switch(param)
         {
         case 1:
            printf("\n %s %e",
                  "Current CRITERIA is :",value);
            printf("\nchange ? [Y/N] :");
            scanf("%s",&c);
            if (c=='n' || c=='N') break;
            printf("\nEnter new value (real) : ");
            scanf("%lf",&value);
            break;

         case 2:
            printf("\nSINGLE array is the default");
            if (narray==0)
               printf("\n%s","Current case is : SINGLE");
            else
               printf("\n%s","Current case is : MULTIPLE");
            printf("\nchange ? [Y/N] :");
            scanf("%s",&c);
            if (c=='n' || c=='N') break;
            printf("\nEnter S for SINGLE or");
            printf(" M for MULTIPLE array : ");
            scanf("%s",&c);
            if (c=='s' || c=='S')  narray=0;
            else if (c=='m' || c=='M') narray=1;
            else narray=0;
            break;
```

```
case 3:
   printf("\nCurrent values are : ");
   printf("\n%s %e %s %e %s %e","xpos = ",xkkm1[0],
        "  ypos = ",xkkm1[2],"  zpos = ",xkkm1[4]);
   printf("\nchange ? [Y/N] :");
   scanf("%s",&c);
   if (c=='n' || c=='N') break;
   printf("\nEnter new values : ");
   printf("\n%s","xpos = ");
   scanf("%lf",&xkkm1[0]);

   printf("\n%s","ypos = ");
   scanf("%lf",&xkkm1[2]);

   printf("\n%s","zpos = ");
   scanf("%lf",&xkkm1[4]);

   break;

case 4:
   printf("\nCurrent values are : ");
   printf("\n%s %e %s %e","xvel = ",xkkm1[1],
        "  yvel = ",xkkm1[3]);
   printf("\nchange ? [Y/N] :");
   scanf("%s",&c);
   if (c=='n' || c=='N') break;
   printf("\nEnter new values : ");
   printf("\n%s","xvel = ");
   scanf("%lf",&xkkm1[1]);

   printf("\n%s","yvel = ");
   scanf("%lf",&xkkm1[3]);

   break;

case 5:
   printf("\nCurrent values are : ");
   printf("\n%s %e %s %e %s %e %s %e","r[0] = ",
         r[0]," r[1] = ",r[1]," r[2] = ",r[2],
         "  r[3] = ",r[3]);
   printf("\nchange ? [Y/N] :");
   scanf("%s",&c);
   if (c=='n' || c=='N') break;
   printf("\nEnter new values : ");
   printf("\n%s","r[0] = ");
   scanf("%lf",&r[0]);

   printf("\n%s","r[1] = ");
   scanf("%lf",&r[1]);

   printf("\n%s","r[2] = ");
```

```
            scanf("%lf",&r[2]);

            printf("\n%s","r[3] = ");
            scanf("%lf",&r[3]);

            break;

         case 6:
            printf("\nCurrent values are : ");
            printf("\n%s %e %s %e %s %e","pkkm1[0] = ",
                   pkkm1[0],"  pkkm1[6] = ",pkkm1[6],
                   "  pkkm1[12] = ",pkkm1[12]);
            printf("\n%s %e %s %e","pkkm1[18] = ",pkkm1[18],
                 "  pkkm1[24] = ",pkkm1[24]);
            printf("\nchange ? [Y/N] :");
            scanf("%s",&c);
            if (c=='n' || c=='N') break;
            printf("\nEnter new values : ");
            printf("\n%s","pkkm1[0] = ");
            scanf("%lf",&pkkm1[0]);

            printf("\n%s","pkkm1[6] = ");
            scanf("%lf",&pkkm1[6]);

            printf("\n%s","pkkm1[12] = ");
            scanf("%lf",&pkkm1[12]);

            printf("\n%s","pkkm1[18] = ");
            scanf("%lf",&pkkm1[18]);

            printf("\n%s","pkkm1[24] = ");
            scanf("%lf",&pkkm1[24]);

            break;

         case 7:
            goto menu;
            break;

         default :
            break;
         }
         goto modify;

run:  /* START EXECUTION */
      printf("\n");
      printf("\nEnter run # : ");
      scanf("%d",&runn);

      fparam=fopen("param.dat","w");
```

```
/*    fprintf(fpxyz,"%2d",runn);
      fprintf(fpe,"%2d",runn);
      fprintf(fpp,"%2d",runn);
      fprintf(fparam,"%2d",runn);

      if (narray==0)
         {
         fprintf(fpxyz,"%s","  Single");
         fprintf(fpe,"%s","  Single");
         fprintf(fpp,"%s","  Single");
         fprintf(fparam,"%s","  Single");
         }
      else
         {
         fprintf(fpxyz,"%s","  Mult");
         fprintf(fpe,"%s","  Mult");
         fprintf(fpp,"%s","  Mult");
         fprintf(fparam,"%s","  Mult");
         }                                    */

      fprintf(fparam,"\n %s %e","CRITERIA = :",value);
      fprintf(fparam,"\n%s %e %s %e %s %e","xkkm1[0] =  ",
             xkkm1[0],"  xkkm1[1] = ",xkkm1[1],
             "  xkkm1[2] = ",xkkm1[2]);
      fprintf(fparam,"\n%s %e %s %e","xkkm1[3] = ",xkkm1[3],
                  "  xkkm1[4] = ",xkkm1[4]);
      fprintf(fparam,"\n%s %e %s %e %s %e %s %e",
              "r[0] = ",r[0],"  r[1] = ",r[1],
              " r[2] = ",r[2],"  r[3] = ",r[3]);
      fprintf(fparam,"\n%s %e %s %e %s %e","pkkm1[0] = ",
              pkkm1[0],"  pkkm1[1] = ",pkkm1[1],
              "  pkkm1[2] = ",pkkm1[2]);
      fclose(fparam);

      fpin=fopen("trj.inp","w");

      fprintf(fpin,"%d %d",narray,runn);
      fprintf(fpin,"\n%e %e %e",
              xkkm1[0],xkkm1[2],xkkm1[4]);/*positions*/
      fprintf(fpin,"\n%e %e",
              xkkm1[1],xkkm1[3]);/*velocities*/
      fclose(fpin);
      if (option==4) goto quit;

      fptrj=fopen("trj.dat","r");
      fpzic=fopen("zic.dat","r");

      /* INITIALIZATION FOR QFIND */
      velm=1.0/vel;
      a14=phis[0];
      a2=a14*a14;
```

```
        g1=a2*0.5;
        g2=g1*g1;
        g3=a14*g1;
        sigacc=sacc*sacc;
        sigcc=scc*scc;
        for (i=0;i<nmax;i++)
            for (j=0;j<nmax;j++) q[i*nmax+j]=0.0;
        q[maxsq-1]=sigdiv*a2*sigdiv;

        /* INITIALIZATION FOR HROW */
        hrow[1]=0.0; hrow[3]=0.0;

        /* INITIALIZATION FOR PDUM */
        for (ii=0;ii<nmax;ii++)
            {
            jj=ii*nmax;
            pdum[jj+1]=0.0;
            pdum[jj+3]=0.0;
            }
        pdum[6]=1.0; pdum[18]=1.0;

        /* INITIALIZING RR */
        for (ii=0;ii<4;ii++)  rr[ii]=sqrt(r[ii]);


        printf("\nPress  any key and ENTER to start ... ");
        scanf("%s",&c);


        /* INSERT TIME CHECKING  HERE */

        /* START THE TIME SLOT LOOP AND SET ARRAY HANDOFF */
        /* POINT */
        for (kk=0;kk<itime;kk++)
            {
/*          printf("\n%s %e","t = ",oldtime);
            printf("%s %d","  kk = ",kk);     */
            dk=kk+1;
/*          printf("\n**********************************");*/
/*          printf("%20s %d \n","************ TIME :",kk); */
            printf("\nwait ");
            printf(".");

            /* GET HYDROPHONE ARRAY COORDINATES */
            if (narray != 0)
                {
                if (xkkm1[0] > sw0)  i8=0;
                else if (xkkm1[0] > sw1)  i8=12;
                else if (xkkm1[0] > sw2)  i8=24;
                else if (xkkm1[0] > sw3)  i8=36;
                else if (xkkm1[0] > sw4)  i8=48;
```

```
            else i8=60;
            for (i=0;i<4;i++)
                {
                j=3*i+i8;
                xb[i]=hydro[j];
                yb[i]=hydro[j+1];
                zb[i]=hydro[j+2];
                }
            }

        /* INSERT TIME CHCKING + WAITING LOOP HERE */

  gettp:
        /* GET THE TRUE TIMES AND THE TRUE POSITIONS
        WRITE TRUE TIME and TRUE POSITIONS */
        fscanf(fptrj,"%lf %lf %lf",
               &data[0],&data[1],&data[2]);
        fscanf(fpzic,"%lf %lf %lf %lf",
               &zic[0],&zic[1],&zic[2],&zic[3]);
        trued[0]=data[0]; trued[1]=data[1];
        trued[2]=data[2];

        /* FIRST GET HROW-CALCULATE GAIN,ESTIMATE */
        /* COVARIANCE OF ERROR BASED ON ONE TIME */
        /* MEASUREMENT-TC,TX,TY,TZ       */
  adapt: for (i=0;i<js;i++)
            {
            x=xkkm1[0]-xb[i];
            y=xkkm1[2]-yb[i];
            z=xkkm1[4]-zb[i];

            denom=sqrt(x*x+y*y+z*z);
            auxh=velm/denom;
            hrow[0]=x*auxh;
            hrow[2]=y*auxh;
            hrow[4]=z*auxh;

            for (ii=0;ii<nmax;ii++)
                {
                iin=ii*nmax;
                gnum[ii]=pkkm1[iin]*hrow[0]+

pkkm1[iin+2]*hrow[2]+pkkm1[iin+4]*hrow[4];
                }

gdenom=hrow[0]*gnum[0]+hrow[2]*gnum[2]+hrow[4]*gnum[4]+r[i];

            /* THIS IS THE FIRST GAIN COLUMN  */
            for (ii=0;ii<nmax;ii++) gi[ii]=gnum[ii]/gdenom;

/*          printf("\nVECTOR HROW : \n");
```

```
printvec(hrow,5);
printf("\nVECTOR GNUM : \n");
printvec(gnum,5);
printf("\nDENOMINATOR  is : ");
printf("%e \n",gdenom);
printf("\nVECTOR GI : \n");
printvec(gi,5);    */

/* CALCULATE THE COVARIANCE OF ERROR PI */
for (ii=0;ii<nmax;ii+=2)
   {
   pdum[ii]=(-gi[0]*hrow[ii]);
   pdum[ii+5]=(-gi[1]*hrow[ii]);
   pdum[ii+10]=(-gi[2]*hrow[ii]);
   pdum[ii+15]=(-gi[3]*hrow[ii]);
   pdum[ii+20]=(-gi[4]*hrow[ii]);
   }
pdum[0]+=1.0;
pdum[12]+=1.0;
pdum[24]+=1.0;

pi[0]   =pdum[0]*pkkm1[0]+pdum[2]*pkkm1[10]+
         pdum[4]*pkkm1[20];
pi[1]   =pdum[0]*pkkm1[1]+pdum[2]*pkkm1[11]+
         pdum[4]*pkkm1[21];
pi[2]   =pdum[0]*pkkm1[2]+pdum[2]*pkkm1[12]+
         pdum[4]*pkkm1[22];
pi[3]   =pdum[0]*pkkm1[3]+pdum[2]*pkkm1[13]+
         pdum[4]*pkkm1[23];
pi[4]   =pdum[0]*pkkm1[4]+pdum[2]*pkkm1[14]+
         pdum[4]*pkkm1[24];
pi[5]=pi[1];
pi[6] =pdum[5]*pkkm1[1]+pkkm1[6]+
         pdum[7]*pkkm1[11]+pdum[9]*pkkm1[21];
pi[7] =pdum[5]*pkkm1[2]+pkkm1[7]+
         pdum[7]*pkkm1[12]+pdum[9]*pkkm1[22];
pi[8] =pdum[5]*pkkm1[3]+pkkm1[8]+
         pdum[7]*pkkm1[13]+pdum[9]*pkkm1[23];
pi[9] =pdum[5]*pkkm1[4]+pkkm1[9]+
         pdum[7]*pkkm1[14]+pdum[9]*pkkm1[24];
pi[10]=pi[2];
pi[11]=pi[7];
pi[12]=pdum[10]*pkkm1[2]+pdum[12]*pkkm1[12]+
         pdum[14]*pkkm1[22];
pi[13]=pdum[10]*pkkm1[3]+pdum[12]*pkkm1[13]+
         pdum[14]*pkkm1[23];
pi[14]=pdum[10]*pkkm1[4]+pdum[12]*pkkm1[14]+
         pdum[14]*pkkm1[24];
pi[15]=pi[3];
pi[16]=pi[8];
pi[17]=pi[13];
```

```
            pi[18]=pdum[15]*pkkm1[3]+pdum[17]*pkkm1[13]+
                      pkkm1[18]+pdum[19]*pkkm1[23];
            pi[19]=pdum[15]*pkkm1[4]+pdum[17]*pkkm1[14]+
                      pkkm1[19]+pdum[19]*pkkm1[24];
            pi[20]=pi[4];
            pi[21]=pi[9];
            pi[22]=pi[14];
            pi[23]=pi[19];
            pi[24]=pdum[20]*pkkm1[4]+pdum[22]*pkkm1[14]+
                      pdum[24]*pkkm1[24];

            /* CALCULATE FIRST MEASUREMENT PREDICTION */
            zhat=velm*denom;
            zdiff[i]=zic[i]-zhat;

            /* COMPUTE THE GATE FOR ERRONEOUS TIME */
            /* MEASUREMENTS */
            pk1=(pi[0] < 0) ? -pi[0] : pi[0];
            pk3=(pi[12] < 0) ? -pi[12] : pi[12];
            pk5=(pi[24] < 0) ? -pi[24] : pi[24];
            p11=(pk1 > pk3) ? pk1 : pk3;
            p = (p11 > pk5) ? p11 : pk5;

/*          pgate=sqrt(p)*velm;*/

            x=p;
            if (x != 0.0)
               {
               if (x < 1.0) x=1.0;
               do
                  {
                  oldx=x;
                  x=( x + p/x )*0.5;
                  temp=oldx-x;
                  rabs=(temp < 0) ? -temp : temp;
                  }
               while (rabs > 0.0);
               }

            pgate=x*velm;
            gate=3.0*(pgate+rr[i]);

/*          printf("\nMATRIX PDUM : \n");
            printmat(pdum,5,5);
            printf("\nMATRIX PI : \n");
            printmat(pi,5,5);
            printf("\nZDIFF  is : ");
            printf("\%e \n",zdiff[i]);
            printf("\nGATE  is : ");
            printf("\%e \n",gate);   */
```

```
            /* EDIT INVALID TIME MEASUREMENTS */
            zgate=(zdiff[i] < 0) ? -zdiff[i] : zdiff[i];

            if (zgate >= gate)
               {
               for (ii=0;ii<nmax;ii++) gi[ii]=0.0;
               zdiff[i]=999.0;
                    /* TAG FOR INVALID TIME MEASUREMENT */
               }

            /* CALCULATE ESTIMATE BASED ON ONE */
            /* MEASUREMENT PREDICTION */
            for                              (ii=0;ii<nmax;ii++)
                xi[ii]=xkkm1[ii]+gi[ii]*zdiff[i];

/*          printf("\nVECTOR XI : \n");
            printvec(xi,5);     */

            for (ii=0;ii<nmax;ii++)
                {
                xkkm1[ii]=xi[ii];
                pkkm1[ii]=pi[ii];
                }
            for (jj=nmax;jj<maxsq;jj++) pkkm1[jj]=pi[jj];
            if (i == 3) goto 156;

            }  /* end JS loop */

         /* NOTE : CALLED ORIGINAL X(0/-1),XKKM1.     */
         /* UPDATED AFTER 1 MEASUREMENT CALLED IT XI,*/
         /* THEN MADE XKKM1 AND WENT THRU ITERATION  */
         /* AGAIN. AFTER YOU HAVE UPDATED XKKM1 FOR  */
         /* EACH MEASUREMENT XKK=XI AND PKK=PI.      */
    156: for (ii=0;ii<nmax;ii++)
            {
            xkk[ii]=xi[ii];
            pkk[ii]=pi[ii];
            }
         for (jj=nmax;jj<maxsq;jj++)
            {
            pkk[jj]=pi[jj];
            }
/*       printf("\nUPDATING VECTORS INTERVAL ... \n");
         printf("\nVECTOR XKKM1 : \n");
         printvec(xkkm1,5);
         printf("\nMATRIX PKKM1 : \n");
         printmat(pkkm1,5,5);

         printf("\nVECTOR XKK : \n");
         printvec(xkk,5);
         printf("\nMATRIX PKK : \n");
```

```
printmat(pkk,5,5);      */

/* RECALCULATE TIME MEASUREMENTS, and FORM */
/* ABSOLUTE VALUE OF RESIDUALS.            */
for (i=0;i<4;i++)

/* EDIT INVALID TIMES FOR ADAPTIVE MANEUVER */
/* ROUTINE */
   {
   if (zdiff[i] < 999.0)
      {
      x=xkkm1[0]-xb[i];
      y=xkkm1[2]-yb[i];
      z=xkkm1[4]-zb[i];

      /* denom=sqrt(x*x+y*y+z*z); */

      denom=x*x+y*y+z*z;
      x=denom;
      if (x != 0.0)
         {
         if (x < 1.0) x=1.0;
         do
            {
            oldx=x;
            x=( x + denom/x )*0.5;
            temp=oldx-x;
            rabs=(temp < 0) ? -temp : temp;
            }
         while (rabs >= 1.e-12);
         }

      denom=x;
      zhat=velm*denom;
      temp=zic[i]-zhat;
      zdiff[i]=(temp < 0) ? -temp : temp;
      }
   else
      {
      zdiff[i]=0.0;
      nzdiff-=1;
      }
   }

/* THIS FUNCTION COMPUTES THE ADAPTIVE Q MATRIX */
if (icurve == 1)
   {
   bb=xkkm1[3]*xkkm1[3];
   dd=xkkm1[1]*xkkm1[1];
   a1=bb+dd;
```

```
            bd=xkkm1[1]*xkkm1[3];
            e1=dd*sigacc/a1+bb*sigcc;
            e12=bd*sigacc/a1-bd*sigcc;
            e2=bb*sigacc/a1+dd*sigcc;
            q[0]=g2*e1;
            q[1]=g3*e1;
            q[2]=g2*e12;
            q[3]=g3*e12;
            q[6]=a2*e1;
            q[7]=q[3];
            q[8]=a2*e12;
            q[12]=g2*e2;
            q[13]=g3*e2;
            q[18]=a2*e2;
            }
         else
            {
            bb=gamma[0]*gamma[0];
            dd=gamma[1]*gamma[1];
            q[0]=bb*covw[0];
            q[1]=gamma[0]*covw[0]*gamma[1];
            q[2]=0.0; q[3]=0.0;
            q[6]=dd*covw[0];
            q[7]=0.0; q[8]=0.0;
            q[12]=bb*covw[1];
            q[13]=gamma[0]*covw[1]*gamma[1];
            q[18]=dd*covw[1];
            }
            q[5]=q[1];  q[10]=q[2];  q[11]=q[7];
            q[15]=q[3]; q[16]=q[8];  q[17]=q[13];

 /*       printf("\nMATRIX Q : \n");
          printmat(q,5,5);    */

         /* IF ALL TIME MEASUREMENTS EXCEED GATE, BYPASS */
         /*  ADAPTIVE MANEUVER ROUTINE.                  */
          if (nzdiff != 0.0)
             {

zdifav=(zdiff[0]+zdiff[1]+zdiff[2]+zdiff[3])/nzdiff;
/*           printf("\nZDIFAV is : ");
             printf("\%e \n",zdifav);  */

             /* DO NOT PERFORM ADAPTIVE MANEUVER IF */
             /* FILTER HAS NOT ACHIEVED THE STEADY STATE. */
             if (kk > 3)
                {

                /* IF ZDIFAV MEETS CRITERIA TRANSFER OUT /*
                /* OF ADAPTIVE MANEUVER ROUTINE.         */
                if (zdifav > value && icont < 5)
```

```
                    {
                    /* INCREASE GAIN */
                    for (j=0;j<maxsq;j++)
                          pkkm1[j]=pkk[j]+q[j];
/*                  printf("\nINCREASE GAIN AND PERFORM");
                    printf(" ADAPTIVE ... \n");
                    printf("\nMATRIX PKKM1 : \n");
                    printmat(pkkm1,5,5);  */

                    /* PERFORM ADAPTIVE MANEUVER BY */
                    /* REITERATING SAME TIME
                    SLOT.
      */
                    icont+=1;
                    goto adapt;
                    }
                }
            }
   skip:
        icont=0;
        nzdiff=4;
        xdiff[0]=xkk[0]-trued[0];
        xdiff[1]=xkk[2]-trued[1];
        xdiff[2]=xkk[4]-trued[2];

/*      temp=xdiff[1]*xdiff[1]/pkkm1[12];
        if (temp < limvar) icurve=0;
        else icurve=1;                    */

        fprintf(fpe,"\n %d %e %e %e",
                kk,xdiff[0],xdiff[1],xdiff[2]);

        /* WRITE */
/*      printf("\nVECTOR ERROR : XDIFF  YDIFF  ZDIFF\n");
        printvec(xdiff,3);  */

        /* PREDICTIONS FOR PKKM1 */

        for (i=0;i<nmax;i++)
            {
            phipkk[i]=pkk[i]+pkk[i+5]*phis[0];
            phipkk[i+5]=pkk[i+5];
            phipkk[i+10]=pkk[i+10]+pkk[i+15]*phis[2];
            phipkk[i+15]=pkk[i+15];
            phipkk[i+20]=pkk[i+20];
            }

        for (i=0;i<nmax;i++)
            {
            in=i*nmax;
            pktemp[in]=phipkk[in]+phipkk[in+1]*phis[0];
```

```
            pktemp[in+1]=phipkk[in+1];
            pktemp[in+2]=phipkk[in+2]+phipkk[in+3]*phis[2];
            pktemp[in+3]=phipkk[in+3];
            pktemp[in+4]=phipkk[in+4];
            }

         for (j=0;j<maxsq;j++) pkkm1[j]=pktemp[j]+q[j];

/*       printf("\nPREDICTIONS FOR PKKM1 ... \n");
         printf("\nMATRIX PKKM1 : \n");
         printmat(pkkm1,5,5);     */

         /* CALCULATE NEW XKKM1 */
         for (ii=0;ii<nmax;ii++)
            {
            xkkm1[ii]=xkk[ii];
            /* p1=sqrt(fabs(pkk[ii])); */
            }
         xkkm1[0]=xkkm1[0]+phis[0]*xkk[1];
         xkkm1[2]=xkkm1[2]+phis[2]*xkk[3];

/*       printf("\nNEW XKKM1 ... (for use in next time");
         printf(" interval)\n");
         printf("\nVECTOR XKKM1 : \n");
         printvec(xkkm1,5);  */

         fprintf(fpp,"\n %d %e %e %e",
                kk,pkk[0],pkk[12],pkk[24]);

         fprintf(fpxyz,"\n %d %e %e %e",
                 kk,xkk[0],xkk[2],xkk[4]);


         }  /* end TIME loop */

      fclose(fpxyz);
      fclose(fpe);
      fclose(fpp);
      fclose(fpzic);
      fclose(fptrj);
      printf("press a key and ENTER to continue ...");
      scanf("%s",&c);
quit:
      printf("\nquit ? [Y/N] :");
      scanf("%s",&c);
      if (c=='n' || c=='N') goto menu;

      }  /* end MAIN PROGRAM */
/*
```

```
printvec(vector,dim)
      int dim;
      double vector[];

      {
      int i, j;
      for (j=0;j<dim;j++) printf("%e %1s",vector[j]," ");
      printf("\n");
      }

printmat(matrix,row,col)
      int row,col;
      double matrix[];

      {
      int i,j;
      for (i=0;i<row;i++)
         {
         for              (j=0;j<col;j++)              printf("%e
%1s",matrix[i*col+j]," ");
         printf("\n");
         }
      }  */
```

```
trajec(narray,hydro,xb,yb,zb,xkkm1,jstart,jend,wd,g,worg)
      double hydro[],xb[],yb[],zb[],xkkm1[],wd,g;
      int narray,worg,jstart,jend;
      {
      int p1i,i,start,end;
      double pp1,r1,sqrt(),zcs,zic[4],zi[4];
      double xd,yd,zd,v,den,velm,gg;
      static double xn[6]={-3.01, -2.0, -1.5, 1.0, -0.6,
                           0.0};
      static double yn[6]={0.0, 0.0228, 0.0668, 0.1357,
                           0.2743, 0.5};
      static double sn[5]={43.8596, 11.3636, 7.25689,
                           2.891352, 2.65887};

      double data[5],wr;
      int runn,i8;
      FILE *fptrj, *fopen();
      FILE *fpzic, *fopen();
      int j,jj;

   /* THIS ROUTINE COMPUTES TRUE TRAJECTORY OF TORPEDO
      data[0]=true x position, data[1]= true y position
      data[2]=true z position, w=turn rate
      data[3]=x_velocity, data[4]=y_velocity
      jstart=time slot where maneuver starts
      jend=time slot where maneuver ends                    */

      data[0]=xkkm1[0]; data[1]=xkkm1[2]; data[2]=xkkm1[4];
      data[3]=xkkm1[1]; data[4]=xkkm1[3]; velm=1.0/vel;
      r1=0.49;
      wr=twopi*wd/360.0; gg=32.2*g; start=jstart; end=jend;

      printf("\n %s \n","start execution of trj.c");
      fptrj=fopen("trj.dat","w");
      fpzic=fopen("zic.dat","w");

      for (jj=0;jj<150;jj++)
         {
         /* GET HYDROPHONE ARRAY COORDINATES */
         if (narray != 0)
            {
            if (data[0] > sw0)  i8=0;
            else if (data[0] > sw1)  i8=12;
            else if (data[0] > sw2)  i8=24;
            else if (data[0] > sw3)  i8=36;
            else if (data[0] > sw4)  i8=48;
            else i8=60;
            for (i=0;i<4;i++)
               {
               j=3*i+i8;
               xb[i]=hydro[j];
```

```
                yb[i]=hydro[j+1];
                zb[i]=hydro[j+2];
                }
            }
        /* COMPUTE THE TRUE TIMES AND THE TRUE POSITIONS
        WRITE TRUE TIME and TRUE POSITIONS */
            printf("%s",".");
            fprintf(fptrj,"\n %e %e %e",
                    data[0],data[1],data[2]);

        for (i=0;i<4;i++)
            {
            xd=data[0]-xb[i];
            yd=data[1]-yb[i];
            zd=data[2]-zb[i];
            den=sqrt(xd*xd+yd*yd+zd*zd);
            zi[i]=velm*den;

/*          printf("\nTRUE TIMES : \n");
            printvec(zi,4);
            printf("\nTRUE POSITIONS : \n");
            printvec(data,3);  */

            /* GET NOISE,SCALE AND ADD TO TRUE */
            /* MEASUREMENT TIME */
            pp1=317.0*r1;
            p1i=pp1;
            r1=pp1-p1i;
            v=r1;
            j=0;
            if (v > 0.5) v=1.0-r1;
        r2: if (v < yn[j+1]) goto r8;
            j+=1;
            goto r2;
        r8: v=(v-yn[j])*sn[j]+xn[j];
            if (r1 >= 0.5) v=(-v);

            zcs=(v/3.00)*0.00001;

/*          printf("\nRANDOM NUMBER :");
            printf("%e \n",zcs);*/
            zic[i]=zcs+zi[i];
            }
        fprintf(fpzic,"\n %e %e %e %e",
                zic[0],zic[1],zic[2],zic[3]);

        /* NEW DATA */
            line(jj,0,150,data);

        }
```

```
        fclose(fptrj);
        fclose(fpzic);
        }

line(j,start,end,data)
    double data[];
    int j,start,end;
    {
    if((j >= start) && (j < end))
        {
        data[0]+=1.31*data[3];
        data[1]+=1.31*data[4];
        }
    }
```

APPENDIX G

LISTING OF PLOT.C - PLOTTING ROUTINE

```
/*************************************************************/
/*                                                           */
/*        PLOTTING ROUTINE - FILE : PLOT.C (IBM-PC)          */
/*                                                           */
/*************************************************************/

#include "stdio.h"
#include "math.h"
#include "limits.h"
#include "kalman.h"
main()
{
      float ix[npt],var[npt],sum,sumsq;
      int i,col,row,inc,c,tt,choice,narray,runn,op,cc;
      float x[npt], y[npt], z[npt],t[npt],xx,yy,zz;
      float xmax,xmin,ymax,ymin,power,yaux;
      float height,width,xscale,yscale;
      float ymaxt,ymint,xmint,xmaxt;
      double xkkm1[5];

      FILE *fpt, *fopen();
      FILE *fpxyz, *fopen();
      FILE *fpe, *fopen();
      FILE *fpp, *fopen();
      FILE *fpin, *fopen();

      fpin=fopen("trj.inp","r");
      fscanf(fpin,"%d %d",&narray,&runn);
      fscanf(fpin,"%lf %lf %lf %lf %lf",
            &xkkm1[0],&xkkm1[2],&xkkm1[4],&xkkm1[1],
            &xkkm1[3]);
      fclose(fpin);

      CLR_SCRN;
      printf("\n           INITIAL CONDITIONS FOR RUN # ");
      printf("%d",runn);
      puts("\n \n");
      if (narray==0)
         printf("\n           CASE is SINGLE    ARRAY.");
      else printf("\n           CASE is MULTIPLE ARRAY.");
      printf("\n  %s  %e %s %e %s %e \n","          xpos = ",
            xkkm1[0],"  ypos = ",xkkm1[2],"  zpos = ",
            xkkm1[4]);
      printf("\n %s %e %s %e \n","          xvel = ",
            xkkm1[1],"  yvel = ",xkkm1[3]);

      choice=1;
```

```
      printf("\n\n\n\n\n\n\n\n\n\n\n\nPress any key to");
      printf(" continue ... ");
      c=keyrd();

   for( ; ; )
      {
      CLR_SCRN;
      printf("           PLOT OPTIONS :  ");
      printf("\n");
      printf("\n      1)    X_EST vs TIME_SLOT  ");
      printf("\n      2)    Y_EST vs TIME_SLOT  ");
      printf("\n      3)    Z_EST vs TIME_SLOT  ");
      printf("\n      4)    Y_EST vs X_EST  ");
      printf("\n      5)    X_ERR vs TIME_SLOT  ");
      printf("\n      6)    Y_ERR vs TIME_SLOT  ");
      printf("\n      7)    Z_ERR vs TIME_SLOT  ");
      printf("\n      8)    X_VAR vs TIME_SLOT  ");
      printf("\n      9)    Y_VAR vs TIME_SLOT  ");
      printf("\n     10)    Z_VAR vs TIME_SLOT  ");
      printf("\n     11)    PKK_X vs TIME_SLOT  ");
      printf("\n     12)    PKK_Y vs TIME_SLOT  ");
      printf("\n     13)    PKK_Z vs TIME_SLOT  ");
      printf("\n");
      printf("\n     14)    TRUE TRAJECTORY  ");
      printf("\n     15)    CHANGE PLOT [dot <-> linear");
      printf(" (default=linear)]");
      printf("\n     16)    EXIT  ");
      printf("\n");
      printf("\n            enter value (1-16) : ");
mop:  CUR_SAVE;
      scanf("%2d",&op);
      if (op<1 || op>16)
         {
         BELL;
         CUR_REST;
         goto mop;
         }
      printf("\n%s%d%s","           option is ",op,
             "  [Y/N] : ");
      scanf("%s",&cc);
      if (cc!='y' && cc!='Y')
         {
         BELL;
         CUR_REST;
         goto mop;
         }
      printf("\n           wait  ");

      switch(op)
         {
         case 1:
```

```
    {
    fpxyz=fopen("xkk.dat","r");
    fscanf(fpxyz,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    x[0]=xx;  t[0]=tt;
    ymax=xx;  ymin=xx;  xmax=npt; xmin=0.0;
    for (i=1;i<npt;i++)
        {
        fscanf(fpxyz,"\n %d %f %f %f",
               &tt,&xx,&yy,&zz);
        x[i]=xx;  t[i]=tt;

        if (xx>ymax) ymax=xx;
        else if(xx<ymin) ymin=xx;

        printf("%s",".");
        }
    plot(ymax,xmax,xmin,ymin,x,t,choice);
    biosset(3,30);
    printf(" %s ","X_EST vs TIME_SLOT");
    biosset(4,8);
    printf("%s","(FT)");
    fclose(fpxyz);
    break;
    }
case 2:
    {
    fpxyz=fopen("xkk.dat","r");
    fscanf(fpxyz,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    x[0]=yy;  t[0]=tt;
    ymax=yy;  ymin=yy;  xmax=npt; xmin=0.0;
    for (i=1;i<npt;i++)
        {
        fscanf(fpxyz,"\n %d %f %f %f",
              &tt,&xx,&yy,&zz);
        x[i]=yy;   t[i]=tt;

        if (yy>ymax) ymax=yy;
        else if(yy<ymin) ymin=yy;

        printf("%s",".");
        }
    plot(ymax,xmax,xmin,ymin,x,t,choice);
    biosset(3,30);
    printf(" %s ","Y_EST vs TIME_SLOT");
    biosset(4,8);
    printf("%s","(FT)");
    fclose(fpxyz);
    break;
    }
case 3:
    {
```

```
    fpxyz=fopen("xkk.dat","r");
    fscanf(fpxyz,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    x[0]=zz;  t[0]=tt;
    ymax=zz;  ymin=zz;  xmax=npt; xmin=0.0;
    for (i=1;i<npt;i++)
        {
        fscanf(fpxyz,"\n %d %f %f %f",
              &tt,&xx,&yy,&zz);
        x[i]=zz;  t[i]=tt;

        if (zz>ymax) ymax=zz;
        else if(zz<ymin) ymin=zz;

        printf("%s",".");
        }
    plot(ymax,xmax,xmin,ymin,x,t,choice);
    biosset(3,30);
    printf(" %s ","Z_EST vs TIME_SLOT");
    biosset(4,8);
    printf("%s","(FT)");
    fclose(fpxyz);
    break;
    }
case 4:
    {
    fpxyz=fopen("xkk.dat","r");
    fscanf(fpxyz,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    t[0]=xx;  x[0]=yy;
    ymax=yy;  ymin=yy;  xmax=xx;  xmin=xx;
    for (i=1;i<npt;i++)
        {
        fscanf(fpxyz,"\n %d %f %f %f",
              &tt,&xx,&yy,&zz);
        t[i]=xx;  x[i]=yy;

        if (yy>ymax) ymax=yy;
        else if(yy<ymin) ymin=yy;

        if (xx>xmax) xmax=xx;
        else if (xx<xmin) xmin=xx;

        printf("%s",".");
        }
    plot(ymax,xmax,xmin,ymin,x,t,choice);
    biosset(3,30);
    printf(" %s ","Y_EST vs X_EST");
    biosset(4,8);
    printf("%s","(FT)");
    biosset(22,63);
    printf("%s","(FT)");
    fclose(fpxyz);
```

```
        break;
        }
    case 5:
        {
        fpe=fopen("xdiff.dat","r");
        fscanf(fpe,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
        x[0]=xx;  t[0]=tt;
        ymax=xx;  ymin=xx;  xmax=npt; xmin=0.0;
        for (i=1;i<npt;i++)
            {
            fscanf(fpe,"\n %d %f %f %f",
                  &tt,&xx,&yy,&zz);
            x[i]=xx;  t[i]=tt;

            if (xx>ymax) ymax=xx;
            else if(xx<ymin) ymin=xx;

            printf("%s",".");
            }
        plot(ymax,xmax,xmin,ymin,x,t,choice);
        biosset(3,30);
        printf(" %s ","X_ERR vs TIME_SLOT");
        biosset(4,8);
        printf("%s","(FT)");
        fclose(fpe);
        break;
        }
    case 6:
        {
        fpe=fopen("xdiff.dat","r");
        fscanf(fpe,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
        x[0]=yy;  t[0]=tt;
        ymax=yy;  ymin=yy;  xmax=npt; xmin=0.0;
        for (i=1;i<npt;i++)
            {
            fscanf(fpe,"\n %d %f %f %f",
                  &tt,&xx,&yy,&zz);
            x[i]=yy;   t[i]=tt;

            if (yy>ymax) ymax=yy;
            else if(yy<ymin) ymin=yy;

            printf("%s",".");
            }
        plot(ymax,xmax,xmin,ymin,x,t,choice);
        biosset(3,30);
        printf(" %s ","Y_ERR vs TIME_SLOT");
        biosset(4,8);
        printf("%s","(FT)");
        fclose(fpe);
```

```
    break;
    }
 case 7:
    {
    fpe=fopen("xdiff.dat","r");
    fscanf(fpe,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    x[0]=zz;  t[0]=tt;
    ymax=zz;  ymin=zz;  xmax=npt; xmin=0.0;
    for (i=1;i<npt;i++)
        {
        fscanf(fpe,"\n %d %f %f %f",
              &tt,&xx,&yy,&zz);
        x[i]=zz;  t[i]=tt;

        if (zz>ymax) ymax=zz;
        else if(zz<ymin) ymin=zz;

        printf("%s",".");
        }
    plot(ymax,xmax,xmin,ymin,x,t,choice);
    biosset(3,30);
    printf(" %s ","Z_ERR vs TIME_SLOT");
    biosset(4,8);
    printf("%s","(FT)");
    fclose(fpe);
    break;
    }
 case 8:
    {
    fpe=fopen("xdiff.dat","r");
    fscanf(fpe,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    x[0]=xx;  t[0]=tt;
    var[0]=0.0;  sum=0.0;  sumsq=0.0;
    ymax=xx;  ymin=xx;  xmax=npt; xmin=0.0;
    for (i=1;i<npt;i++)
        {
        fscanf(fpe,"\n %d %f %f %f",
              &tt,&xx,&yy,&zz);
        x[i]=xx;  t[i]=tt;

        sum+=x[i];   sumsq+=x[i]*x[i];
var[i]=((t[i]+1.0)*sumsq-sum*sum)/((t[i]+1.0)*t[i]);
        xx=var[i];

        if (xx>ymax) ymax=xx;
        else if(xx<ymin) ymin=xx;

        printf("%s",".");
        }
    plot(ymax,xmax,xmin,ymin,var,t,choice);
    biosset(3,30);
```

```
    printf(" %s ","X_VAR vs TIME_SLOT");
    biosset(4,7);
    printf("%s","(FTsq)");
    fclose(fpe);
    break;
    }
 case 9:
    {
    fpe=fopen("xdiff.dat","r");
    fscanf(fpe,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    x[0]=yy;  t[0]=tt;
    var[0]=0.0;    sum=0.0;    sumsq=0.0;
    ymax=yy;  ymin=yy;  xmax=npt; xmin=0.0;
    for (i=1;i<npt;i++)
        {
        fscanf(fpe,"\n %d %f %f %f",
               &tt,&xx,&yy,&zz);
        x[i]=yy;    t[i]=tt;

        sum+=x[i];    sumsq+=x[i]*x[i];
var[i]=((t[i]+1.0)*sumsq-sum*sum)/((t[i]+1.0)*t[i]);
        yy=var[i];

        if (yy>ymax) ymax=yy;
        else if(yy<ymin) ymin=yy;

        printf("%s",".");
        }
    plot(ymax,xmax,xmin,ymin,var,t,choice);
    biosset(3,30);
    printf(" %s ","Y_VAR vs TIME_SLOT");
    biosset(4,7);
    printf("%s","(FTsq)");
    fclose(fpe);
    break;
    }
 case 10:
    {
    fpe=fopen("xdiff.dat","r");
    fscanf(fpe,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    x[0]=zz;  t[0]=tt;
    var[0]=0.0;  sum=0.0;  sumsq=0.0;
    ymax=zz;  ymin=zz;  xmax=npt; xmin=0.0;
    for (i=1;i<npt;i++)
        {
        fscanf(fpe,"\n %d %f %f %f",
               &tt,&xx,&yy,&zz);
        x[i]=zz;  t[i]=tt;

        sum+=x[i];    sumsq+=x[i]*x[i];
var[i]=((t[i]+1.0)*sumsq-sum*sum)/((t[i]+1.0)*t[i]);
```

```
        zz=var[i];

        if (zz>ymax) ymax=zz;
        else if(zz<ymin) ymin=zz;

        printf("%s",".");
        }
    plot(ymax,xmax,xmin,ymin,var,t,choice);
    biosset(3,30);
    printf(" %s ","Z_VAR vs TIME_SLOT");
    biosset(4,7);
    printf("%s","(FTsq)");
    fclose(fpe);
    break;
    }

case 11:
    {
    fpp=fopen("pkk.dat","r");
    fscanf(fpp,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    x[0]=xx;  t[0]=tt;
    ymax=xx;  ymin=xx;  xmax=npt; xmin=0.0;
    for (i=1;i<npt;i++)
        {
        fscanf(fpp,"\n %d %f %f %f",
              &tt,&xx,&yy,&zz);
        x[i]=xx;   t[i]=tt;

        if (xx>ymax) ymax=xx;
        else if(xx<ymin) ymin=xx;

        printf("%s",".");
        }
    plot(ymax,xmax,xmin,ymin,x,t,choice);
    biosset(3,30);
    printf(" %s ","PKK_X vs TIME_SLOT");
    biosset(4,7);
    printf("%s","(FTsq)");
    fclose(fpp);
    break;
    }
case 12:
    {
    fpp=fopen("pkk.dat","r");
    fscanf(fpp,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
    x[0]=yy;  t[0]=tt;
    ymax=yy;  ymin=yy;  xmax=npt; xmin=0.0;
    for (i=1;i<npt;i++)
        {
        fscanf(fpp,"\n %d %f %f %f",
               &tt,&xx,&yy,&zz);
```

```
            x[i]=yy;  t[i]=tt;

            if (yy>ymax) ymax=yy;
            else if(yy<ymin) ymin=yy;

            printf("%s",".");
            }
        plot(ymax,xmax,xmin,ymin,x,t,choice);
        biosset(3,30);
        printf(" %s ","PKK_Y vs TIME_SLOT");
        biosset(4,7);
        printf("%s","(FTsq)");
        fclose(fpp);
        break;
        }
    case 13:
        {
        fpp=fopen("pkk.dat","r");
        fscanf(fpp,"\n %d %f %f %f",&tt,&xx,&yy,&zz);
        x[0]=zz;  t[0]=tt;
        ymax=zz;  ymin=zz;  xmax=npt; xmin=0.0;
        for (i=1;i<npt;i++)
            {
            fscanf(fpp,"\n %d %f %f %f",
                   &tt,&xx,&yy,&zz);
            x[i]=zz;  t[i]=tt;

            if (zz>ymax) ymax=zz;
            else if(zz<ymin) ymin=zz;

            printf("%s",".");
            }
        plot(ymax,xmax,xmin,ymin,x,t,choice);
        biosset(3,30);
        printf(" %s ","PKK_Z vs TIME_SLOT");
        biosset(4,7);
        printf("%s","(FTsq)");
        fclose(fpp);
        break;
        }
    case 14:
        {
        fpt=fopen("trj.dat","r");
        fscanf(fpt,"\n %f %f %f",&xx,&yy,&zz);
        x[0]=xx;  y[0]=yy;
        ymax=yy;  ymin=yy;  xmax=xx;  xmin=xx;
        for (i=1;i<npt;i++)
            {
            fscanf(fpt,"\n %f %f %f",&xx,&yy,&zz);
            x[i]=xx;  y[i]=yy;
```

```
                if (yy>ymax) ymax=yy;
                else if(yy<ymin) ymin=yy;

                if (xx>xmax) xmax=xx;
                else if (xx<xmin) xmin=xx;

                printf("%s",".");
                }
            plot(ymax,xmax,xmin,ymin,y,x,choice);
            biosset(3,30);
            printf(" %s ","TRUE_Y vs TRUE_X");
            biosset(4,8);
            printf("%s","(FT)");
            biosset(22,63);
            printf("%s","(FT)");
            fclose(fpt);
            break;
            }
        case 15:
            {
            printf("\n        CHANGE PLOT ( enter option )");
            printf("\n");
            printf("\n   1)  LINEAR INTERPOLATION BETWEEN");
            printf(" POINTS");
            printf("\n   2)  SIMPLE DOT");
            printf("\n");
            printf("\n        ENTER  choice : ");
            scanf("%d",&choice);
            goto skip;
            break;
            }
        case 16: goto ending;
        default:
            {
            goto skip;
            break;
            }
        } /* end case */

    CUR_MV(4,67);
    printf("%s %d","# ",runn);
    CUR_MV(5,67);
    if(narray==0) printf("SINGLE");
    else printf("MULTIPLE");

    c=keyrd();
    biosini(3);
skip:

} /* end for-forever-loop */
```

```
ending: printf("\n");
        printf("\nEnd plotting ... Press any key");
        c=keyrd();
        CLR_SCRN;
}
```

```
/*************************************************************/
/*                                                           */
/* Function to draw two axis.                                */
/*                                                           */
/* *** NOTE ***                                              */
/* The upper left corner of the screen is 0,0 and the        */
/* bottom right corner is 199,639.                           */
/*                                                           */
/*************************************************************/

int axis(ulrow,ulcol,lrrow,lrcol,mantx,manty)
int ulrow,ulcol,lrrow,lrcol;
double mantx[],manty[];
{
int count,irow,i,iaux,icol,tcount,cinc,rinc,hinc;
double
deltax,deltay,ymax,xmin,x,y,rowmax,rowmin,colmax,colmin;
double deltar,deltac,aux;

ymax=manty[0];
xmin=mantx[1];
deltay=0.1*(manty[0]-manty[1]);
deltax=0.1*(mantx[0]-mantx[1]);
rowmax=lrrow;  rowmin=ulrow;  deltar=0.1*(rowmax-rowmin);
colmax=lrcol;  colmin=ulcol;  deltac=0.1*(colmax-colmin);
rinc=deltar+0.5;    cinc=deltac+0.5;

/* ------------------------------------------------------ */

/* do vertical ------ line with scale */
i=ulcol;
for (irow = ulrow+1; irow <= lrrow; irow++)  bioswd(irow,i);
for (irow = ulrow; irow <= lrrow; irow+=rinc)
    for     (tcount=i-4;     tcount    <=    i;    tcount++)
bioswd(irow,tcount);

/* do horizontal ------ line with scale */
i=lrrow;
for (icol = ulcol+1; icol <= lrcol; icol++)  bioswd(i,icol);
for (icol = ulcol; icol <= lrcol; icol+=cinc)
    for     (tcount=i;     tcount    <=    i+4;    tcount++)
bioswd(tcount,icol);

/* ------------------------------------------------------ */

/* write vertical  scale */
y=ymax;
for (irow = ulrow; irow <= lrrow; irow+=rinc)
    {
    i=irow/8.0+0.5;
```

```
    biosset(i,0);
    printf("%.1f",y);
    y-=deltay;
    }

/* write horizontal scale */
x=xmin;
for (icol = ulcol; icol <= lrcol; icol+=cinc)
    {
    i=icol*0.125-1.5;
    biosset(24,i);
    printf("%.1f",x);
    x+=deltax;
    }

/* ------------------------------------------------------ */

/* done */
return(0);
}

plot(yymax,xxmax,xxmin,yymin,x,t,choice)
   float yymax,xxmax,x[],t[],xxmin,yymin;
   int choice;
{
   float xscale,yscale,height,width,bx,by;
   float ymax,xmax,ymin,xmin;
   double xaux,yaux,mantx[2],manty[2],log10();
   double aux,pow(),aux1;
   int i,inc,row,col,ixexp,iyexp,c,oldrow,newrow,ii,iimax;
   int col1,col0,row1,row0,irow,oldcol,icol,newcol;
   float deltac,deltar,delta,ang;

   /* DETERMINE THE SCALING */
   ymax=yymax; xmax=xxmax; ymin=yymin; xmin=xxmin;
   if (ymax==ymin)
      {
      ymax=ymax+5.0;
      ymin=ymin-5.0;
      }
   if (xmax==xmin)
      {
      xmax=xmax+5.0;
      xmin=xmin-5.0;
      }
   height=ymax-ymin;
   width=xmax-xmin;
   xscale=560.0/width;
   yscale=-160.0/height;
   by=184.0-ymin*yscale+0.5;
   bx=600.0-xmax*xscale+0.5;
```

```
/* DETERMINE EXPONENT and MANTISSA */
xaux=1.0;
aux=abs(xmax); aux1=abs(xmin);
if (aux < aux1) aux=aux1;
if (aux!=0.0)
   {
   xaux=log10(aux);
   ixexp=xaux;
   if (ixexp<0) ixexp-=1;
   aux=ixexp;
   xaux=pow(10.0,aux);
   }
mantx[0]=xmax/xaux;
mantx[1]=xmin/xaux;

yaux=1.0;
aux=abs(ymax); aux1=abs(ymin);
if (aux < aux1) aux=aux1;
if (aux!=0.0)
   {
   yaux=log10(aux);
   iyexp=yaux;
   if (iyexp<0) iyexp-=1;
   aux=iyexp;
   yaux=pow(10.0,aux);
   }
manty[0]=ymax/yaux;
manty[1]=ymin/yaux;

printf("\n%s","Press any key to see plot.");
c=keyrd();
biosini(6);
axis(24,40,184,600,mantx,manty);

yaux=yaux;
biosset(3,6);
printf("%.1e",yaux);

xaux=xaux;
biosset(22,68);
printf("%.1e",xaux);
```

```
/*************************************************************/
/*                                                           */
/*  PLOT   USING  LINEAR  INTERPOLATION  BETWEEN  POINTS     */
/*                                                           */
/*************************************************************/

   if (choice==1)
      {
      col0=t[0]*xscale+bx;
      row0=x[0]*yscale+by;
      bioswd(row0,col0);

      for (i=1;i<npt;i++)
          {
          col1=t[i]*xscale+bx;
          row1=x[i]*yscale+by;
          deltac=col1-col0;
          deltar=row1-row0;

          if (abs(deltac)>=abs(deltar))
              {
              ang=deltar/deltac;
              iimax=abs(col1-col0);
              icol=1;
              col=col0;
              if (col1<col0)
                  {
                  icol=-1;
                  }
              for (ii=0;ii<iimax;ii++)
                  {
                  col+=icol;
                  delta=col-col0;
                  row=ang*delta+row0+0.5;

         /*       if (choice==1)
                     {
                     if (abs(deltar) > 5 && abs(deltac) >5)
                        goto discont;
                     }  */

                  bioswd(row,col);
                  }
              }  /* end_if abs(deltac)>=abs(deltar) */

           else if (abs(deltac)<abs(deltar))
              {
              ang=deltac/deltar;
              iimax=abs(row1-row0);
              irow=1;
```

```
                row=row0;
                if (row1<row0)
                   {
                   irow=-1;
                   }
                for (ii=0;ii<iimax;ii++)
                    {
                    row+=irow;
                    delta=row-row0;
                    col=ang*delta+col0+0.5;
                /*  if (choice==1)
                       {
                       if (abs(deltar) > 5 && abs(deltac) >5)
                          goto discont;
                       }  */
                    bioswd(row,col);
                    }
                }   /* end_else_if */
            col0=col1;  row0=row1;
            } /* end_if for i=0,200 */
        }  /* end_if choice=1 */

/*************************************************************/
/*                                                           */
/*    This portion is to be used if you want just the        */
/*    points without linear interpolation.                   */
/*                                                           */
/*************************************************************/

   if (choice==2)
       {
       for (i=0;i<npt;i++)
           {
           col=t[i]*xscale+bx;
           row=x[i]*yscale+by;
           bioswd(row,col);
           }
       }
       goto termino;
discont:
       {
       biosini(3);
       printf("\n DISCONTINUOUS DATA FOUND.");
       printf(" CHANGE OPTION to SIMPLE");
       printf("-DOT PLOT and TRY AGAIN.");
       printf(" Press any key to continue .");
       c=keyrd();
       }
termino:
}
```

APPENDIX H

LISTING OF THE BATCH FILES - IBM-PC

```
/*************************************************************/
/*                                                           */
/*    Batch file "CE.BAT"  (IBM-PC). Contains the            */
/*    sequence of calls to compile thefile KR.C .            */
/*                                                           */
/*    USAGE: "ce kr"                                         */
/*                                                           */
/*************************************************************/

lc1 %1
lc2 %1
link cs+%1+biosio+tr+dostime,%1,nul,lcms+lcs
erase %1.obj
pause
```

```
/*************************************************************/
/*                                                           */
/*    Batch file "KAL.BAT" (IBM-PC). Contains the            */
/*    sequence of calls to execute the files KR.C            */
/*    (Kalman Filter program) and PLOT.C  (the               */
/*    plotting routine).                                     */
/*    USAGE: "kal"                                           */
/*                                                           */
/*************************************************************/

graphics
:loop
kr =4000
rem DO YOU WANT TO PLOT RESULTS ?
pause
plot =8000
rem DO YOU WANT TO TRY ANOTHER CASE ?
rem  CTL-BREAK  will  stop execution. Press any other key to
continue.
pause
CLS
goto loop
```

APPENDIX I

SIMTERM - PROGRAM DOCUMENTATION

This Appendix contains the listing of the file "readme" found in the PC-SIG LIBRARY (Public Domain Library), disk number 362.

1. Introduction

Feel free to make copies of this program. There is no restriction on the distribution as long as the copyright notices are left on the source and that there is no charge for the distribution. This program is made available to users of IBM-PCs (and compatibles since I really like the COMPAQ/PLUS). No guarantees are made or implied. If you have any questions or suggestions, please give me a call and I will try to answer them. A recent acquisition of an interrupt driven communication driver from the University of British Columbia Computer Center has enabled me to remove the requirement that you have the IBM Async Communications Support package which included a driver I used to use. This version of SIMTERM is therefore self-contained and requires no other software to run. Have fun using it and **HAPPY COMPUTING**.

This floppy contains all the software for the terminal simulator for the IBM PC. The main function of this program is to provide a reasonable terminal simulation on the IBM PC of an HP-like terminal to a UNIX(TM) system. The file 'simterm.exe' is the load module and when invoked, assumes a default of 1200 baud, 1 stop bit, no parity, and flow control (XON/XOFF). The only programs that have to be on the floppy are SIMTERM.EXE, HELP.TXT, and MENUS. If you have a VENTEL or a Hayes modem, the file BOARDS can be copied over as a starter for a list of BBSs. The BOARDS file is set up for a VENTEL autodialer. The dialing strings will have to be changed to conform for the Hayes standard.

If you have DOS 2.0, put SIMTERM.EXE, HELP.TXT, BOARDS and MENUS in a subdirectory called \simterm. That way, if you change directories during a SIMTERM session, by specifying a path for any type of file transfer, SIMTERM will be able to find its way back to these two important files. If you don't have DOS 2.0, then just IGNORE the occasional "directory not found" messages, they're harmless.

2. Command Line Parameters

Optional parameters on the command line will override the defaults as follows:

`-a' put the simulator into the ADM3a emulation mode for those systems which don't support TERMCAP.

`-b' Causes SIMTERM to assume you are using a monochrome display, even though you may not really have one.

`-B' will cause the transmit and receive buffers to be cleared when a BREAK (End key) is sent. This is mainly for high baud rate lines where there is still junk in the buffers that you don't want to see.

`-cn' specifies which COMx device you want to use. The default is 1 (use COM1).

`-d' Enter direct connect mode. SIMTERM won't wait for carrier detect. Use this option if you're connected directly to the remote, with no modem. You can also use this option when SIMTERM seems to hang at startup, with the "Establish communications link" message. This may be case with Hayes modems if you don't have them optioned to raise the carrier detect.

`-D' When this flag is used, SIMTERM will ignore DEL (127 Ascii) characters, and the character immediately following the DEL. We use it because we have a noisy line.

`-e' specifies that the EPSON printer does not have the GRAFTRAX option. Therefore when using `ibmpr' to route stuff to the printer, the code will simulate underlining, subscripts, and superscripts. Default is an EPSON with GRAFTRAX.

`-g' Causes SIMTERM to assume you are using a graphics display, even though you may not really have one.

`-h' SIMTERM will operate in the `half-duplex' mode; i.e., it will echo the characters the user types since the host system does not.

`-in' specifies the IRQ (interrupt request level) to be

used. Default is 4, but some boards (like the Quadram) use IRQ3 for COM2, so in this case SIMTERM would have to be invoked with the parameters `-c2 -i3' to set it up correctly.

`-m' Tells SIMTERM that you have a Hayes modem instead of the default VenTel modem. This will enable SIMTERM to use the directory dialing features (Alt-V)

`-M' Redefines the cursor control keys compatible with a set of UNIX-like utilities on a PR1ME computer at UofVa.

`-nxxxx' if you have a Ventel or Hayes modem, this option will automatically dial the number when the simulator is started.

`-o' turns off the error messages caused by communications line; e.g., framing error and invalid parity.

`-px' specifies the parity where `n'-none, `o'-odd, `e'-even, `m'-mark, and `s'-space.

`-PFfil' Use file "file" as a script source file (see SCRIPTS below)

`-PDscr' Execute script "scr" upon start-up (see SCRIPTS below)

`-r' this causes SIMTERM to wait for the horizontal retrace when writing to the screen. If you notice `sparkles' on your graphics monitor, invoke SIMTERM with this option and it should clear up.

`-snnnn' specifies `nnnn' as the baud rate (e.g., -s300).

`-Sn' specifies the number of stop bits (1 or 2)

`-vxxxx' specifies the file (xxxx) to be used when the Alt-V options is invoked to dial BBS. The full path name of the file should be specified.

`-wn' set the word length to `n'. Default is 8 (7 data + 1 parity).

`-xn' this option will turn off the XON/XOFF flow control. The program will send an XOFF when the receive buffer is almost full and then an XON when it has emptied out. It will also recognize an XOFF from the host to stop transmitting and then an XON to start again.

`-kfilename' will read in the file `filename' to initialize the function keys. The file is of the form

```
n=string of chars

where 0<=n<=9

 example
 0=who^V
 1=who|wc^V
```

If the string is terminated with a `^V' (chr(174) -- obtained by holding down the `Alt' key and typing 174 on the key pad), a `carriage return' will be appended.

For example:

If you want to run at 9600 baud, even parity, without waiting for carrier detect

```
simterm -s9600 -pe -d
```

If you want to use COM2 (and it requires IRQ3) at 300 baud and run script "xyzzy" from file "scr2"

```
simterm -s300 -c2 -i3 -PFscr2 -PDxyzzy
```

If you are using a Hayes modem and want a number dialed when SIMTERM is started, use the following command line:

```
simterm -m -d -n12145551212
```

3. Program Functions (Unix specifics)

Once loaded, the program will use all available memory above the module for save space for paging through previous output. I have 256K on my system and this provides about 20 pages (~500 lines) of saved text. The TERMCAP entry for the terminal is in the file 'termcap.ibm'. This should either be added to your termcap library or the 'shell' variable TERMCAP should be set to it before trying to use any package that requires cursor addressing. The TERMCAP entry makes use of the fact that 'vi' (and 'curses') will use a special sequence to enter and leave the control mode. The simulator uses these sequences to redefine the cursor control keys so that they can be used in that mode; e.g., in 'vi' the "up arrow" will send in a "k", while the "PgDn" will send in a "^D" to scroll the display down, etc. For those of you who are running on UNIX systems that use "terminfo", there is a file "terminfo.ibm" which has the appropriate definitions.

Output to the printer can be initiated under program control with a filter similar to 'hp' which is used for HP terminals. The source file for 'ibmpr' (which is the filter that use on UNIX) is on the floppy. You should compile this UNIX and then use for controlling output to the printer. A typical use of this filter is:

```
nroff -T37 file | ibmpr -p
```

The '-p' option says to route the output to the EPSON printer. The '-i' option will cause underlined text to come out in italics. The '-c' option is used to invoke the compressed mode (132 characters/line).

If those escape sequences are output, they will be displayed appropriately on the CRT and output to the printer correctly.

4. User Defined Function Keys

There are 10 user definable functions keys that will send a character sequence of up to 30 characters. To

display the current contents of the keys, hold the 'Alt' key down and type '='. To change the contents, hold the 'Alt' key and type '-' and follow directions. To invoke the functions keys, hold down the 'Alt' key and type the corresponding digit (e.g., 'Alt 3'). A user can define a file with the contents of the keys and cause it to be read in with the '-k' option on the command line.

5. Upload/Download (Unix Specific)

This is used to transfer 'text' files (ones that only contain printable ASCII and tabs) between the PC and the host. To remember which way the data flow, just remember that the IBM PC looks 'up' to UNIX, while UNIX looks 'down' on the PC. To move files between the PC and UNIX, hit 'Cntl-PgUp' to transfer PC->UNIX. Answer the prompts. Hit 'Ctrl-PgDn' to move from UNIX- >PC. As the transfer is taking place, a message will indicate the # of bytes transferred. When going UNIX->PC, there will be more bytes on the PC disk than on UNIX because of added 'end-of-line' and 'end-of-text' bytes. This can only be used to transfer TEXT files.

6. Upload/Download (Straight ASCII & XMODEM)

The program can be used to connect to BBSs and any other systems that support async devices. To download a file to the PC, go through the normal dialog on the BBS and then when it is waiting for you to hit return to start the transfer, press Alt-C to open the 'copy' file that will receive the data. You will be put back to the original screen where you can hit return. At the end of the data transfer, hit Alt-C to close the copy file.

To upload a file from the PC to a BBS, again go through the dialog to setup the transfer and when the BBS is ready, press Alt-D to specify the data file. This will then 'dump' the file to the comm line. At the end of the transfer, you may have to type in a control sequence to indicate that you are done. This option will request a 'delay' between characters so that you do not overrun the receiving system. Use '0' for no delay; a value of '1' is probably sufficient for most systems that can not handle the normal baud rate. Experiment.

SIMTERM supports the XMODEM protocol for data transfer from a BBS. To invoke this option, hit Alt-R to receive with XMODEM at the point that the sender is ready. You will be prompted for the file to receive into and then the PC will initiate the transfer of data. Transmitting with XMODEM is invoked with Alt-T and prompts will follow.

7. Script files

Script files are used to allow SIMTERM to do mindless interactive login and other such sequences. A Scriptfile can be created by your favorite word processor and SIMTERM will compile it for you, and save the compiled version so you don't have to re-compile at each run. Script files read much like English, so they are both easy to understand and create.

Script Commands Available

Each of the following commands can appear in a script file. Note that each separate command must be separated by a space, tab or newline. The compiler ignores extra white-space, so you can have indenting, multiple commands per line, or whatever you want. Syntax notes: "str" in the following denotes a string constant delimited by quotes. A string can have multiple words. To get a quote character into a constant, write it as \". To get a backspace, write it as \\. Although one-word strings need not have quotes, they should, so that they are not confused with keywords (commands). Non-quoted strings generate warnings from the compiler.

name "str" --- defines an entry point for the script that you can use to identify this part, i.e, if a "name whux1b" appears in the file, then you can start running the script by using the name "whux1b". (See "running scripts" below) If you choose "whux1b" from the pop-up menu later, the script will start executing at the statement immediately following the "name" statement.

send "str" --- sends the string to the remote computer, with a carriage return tacked onto the end. Some escapes can be placed into the string, as follows: an "\1" causes a one second pause when it is encountered. "\m" sends

a carriage return. "\j" sends a line-feed. "\c" suppresses the carriage return that is normally tacked onto the end of a string. Remember, though, you need to type TWO backspaces to get one, so these sequences should appear as \\1,\\m, etc in your file.

dial "str" --- dials the string on a VenTel dialer. Obviously, if you don't have a ventel, don't use this command (Fake it with "send"). For example, to dial "3395" on a Hayes, put a line like "send ATDT3395" in your file.

say "str" --- outputs "str" onto the screen, so that you can put tracer statements in the script.

goto "str" --- branches to str. labels to be branched to must be followed by a colon. I.e. early in the script you have a line "label:" all by itself. Later, you can call "goto label" (no colon) to branch back up to the earlier statement

gosub "str" --- like goto, but when a "return" statement is encountered later, the program branches BACK to the statement that follows the gosub. Gosubs may be nested.

return --- returns from a "gosub" statement

input "str" --- Prompts the user with "str", reads one line, and sends it to the remote computer.

settime "str" --- sets the timeout value to "str" seconds. Default is fifteen.

quit --- execution is returned to SIMTERM. There is an understood "quit" at the end of file

The Logical Operators (IF/CASE)

Besides the COMMANDS, the script compiler recognizes two logical operators, the if/else statement and the case.

IF SYNTAX

```
if "str" stmnt1 else stmnt   OR     if "str" {
                                           stmnt
                                           stmnt
          OR                                 .
                                             .
if "str" stmnt1 (no ELSE)                 } else stmnt

                                    OR    } else {
                                             stmnt
                                               .
                                            }
```

When encountered, the program will scan the output from the remote computer until either it finds "str" or it times out. If it finds "str", stmnt1 is executed, if it times out, stmnt2 is executed. Execution then drops down to the next statement, unless the stmnt1 or stmnt2 contained a goto or gosub or quit.

stmnt1 or stmnt2 may be either single commands from above, or multiple commands delimited by { and } (note white space around the brackets. If the else following the if is missing, then execution falls to the next statement. Note that ifs may be nested. For example

```
if "str" dial "4244" else if "str2" dial "4324"
```

Be aware, however, that input is only scanned once, so if "str2" appears before "str" in the preceeding example, both tests will fail!!

CASE SYNTAX

The case statement is like multiple simultaneous IFs. It allows you to set up a few strings to be searched for SIMULTANEOUSLY, with an action specified to be executed if and when one of the strings is found. Syntax is

```
case
str1: action
str2: action
    .
otherwise action
caseend
```

Action can be either a single statement (say, send, dial, or even an if, or another case.) or multiple statements surrounded by { and } (like the if clause.). For example, if you have

```
case
eric: dial "eric's #"
holtman: dial "another #"
caseend
```

and the input is "djsfhj gfjg eric holtman" then "eric's #" will be dialed and the case will quit. Note that "holtman" is not matched since a case statement executes as soon as any of the strings is found.

Script Examples

Call number 3395, wait for "login" prompt. If I get it, gosub to a login procedure. If I don't then say something, then quit. The login subroutine will send my id, then prompt for my password, send it, then return.

```
name example
dial 3395
if "login" gosub "subr" else {
     say "3395 didn't answer"
     quit
}
quit
subr:
     send "id3434"
     input "Your password, sire? "
     return
```

Note that after the "return" the next statement to be executed will be "quit".

Compiling Scripts

To use the example above, copy it out to a file called, for example, "prog". Then invoke SIMTERM with a -PFprog flag, like so: "simterm -PFprog". This tells SIMTERM that you wish to use scripts from the file "prog". If no -PF flag appears, SIMTERM will read the file "scripts" which is where you may want to keep compiled scripts. After SIMTERM

gets running, type an Alt-L. This informs SIMTERM that you wish to use a script. It will examine the file "prog" and determine that it needs to be compiled. Tell SIMTERM what file you want to put the compiled stuff in, then sit back and watch for error messages. After it compiles, hit return and proceed to the next step. Remember the file name you used for the compiled scripts, and use it next time you run SIMTERM, so that you don't have to recompile everything again.

Using a script

After you type Alt-L, or after the compile is finished, a menu of all "name" strings will pop up. If you're still using the example, only one name ("example") will be there. Using the up and down arrows, make your choice, then hit the space bar. SIMTERM will then execute your script starting at that named point.

To execute a script from startup, use the -PD flag. For example, to run script "example" from the file "prog.scr", type

```
simterm -PFprog.scr -PDexample
```

8. Functions Available from the Keyboard

All these functions can be accessed from the keyboard by typing the appropriate character. If you forget the commands, Alt-H will bring up a menu for you. Just follow the menu selection process until you find the desired function. Note that the menu will tell you which key to use to activate that function in the future. I.e., the menu has Alt-L next to the script choice to remind you that you can use Alt-L instead of the menus to activate the script functions.

F1 Will 'toggle' the display so that usually non-displayed control characters are displayed in reverse video using the upper case letter that they are equivalent to.

F2 Will 'print' the line that the cursor is positioned at.

F3 Send a BREAK

F8 Freezes the display above the cursor. Handy to save some data on the upper portion so you can see it as you do other commands.

F9 Turns on the row/column display at the bottom of the screen so that you know where the cursor is currently positioned.

F10 Toggles XON/XOFF. A convenient way of stopping/restarting the output to the CRT when connected to a system that honors the XON/XOFF protocol for flow control.

Alt-F1 Will terminate SIMTERM, but leave the DTR line high so OR that you can restart SIMTERM and pickup where you left Alt-255 off.

Alt-F2 Terminate SIMTERM, and drop DTR so that the modem will hang up.

Alt-F3 Toggles the DTR (data terminal ready) lead on the communications line. This is used to drop the carrier at the modem so that you can redial.

Alt-F4 Toggles the graphics-mode. This will display characters with their parity bits so that the 'graphics' characters of the PC (>127) are should. This is used on some of the BBSs.

Alt-F5 Clears line 25 if it becomes cluttered.

Alt-F6 Display various mode settings on line 25.

Alt-F7 Resets the communications software. In some instances an interrupt might be missed and leave the system in a state where it is expecting data that will not be sent. If you suspect this, try an Alt-F7 before rebooting the software to see if it breaks you out of the situation.

Alt-C Puts SIMTERM in a 'capture' mode so that all input received over the communication line is logged to a disk file. Handy for picking up ASCII text from a

system that does not support XMODEM. It will prompt you for a file name.

Alt-D Will 'dump' a ASCII file on the PC to the comm line. This is a way of 'uploading' to a system that does not support XMODEM. You can also specify a delay to prevent overruns.

Alt-E Toggles the 'vi' mode. Handy when you exit SIMTERM and then come back into it with UNIX still in 'vi'. The mode should be TRUE to have the cursor keys work for 'vi'.

Alt-F Enter the 'local' line editing mode. This allows you to make corrections to a line and then reenter it. Use the cursor control keys to move to the line, 'Ins' and 'Del' to fixup the line and then hit return to reenter the line and leave the line edit mode.

Alt-H Display a 'help' menu listing the defined function keys.

Alt-L Invoke the 'script' processor

Alt-R Receive a file using the XMODEM protocol.

Alt-T Send a file using the XMODEM protocol.

Alt-V Will display a menu that is used with the 'auto-dial' capability of the Ventel. Once in this menu, select the desired feature.

Alt-= Display the user defined function keys

Alt-- Change a user defined function key

Home Goto top of file in "vi" (unix specific)

Ins Send the current line to remote (similar to the ENTER key on an HP-2621) OR if in "vi" toggle insert mode.

Del Clear display from cursor to bottom OR if in "vi" delete a character

Arrows Move through screen memory in the specified direction

PgUp Back up one page through memory

PgDn Go down one page in memory

Cntr+PrtSc Toggle output to printer

Ctrl+PgDn Download (Unix ---> IBM)

Ctrl+PgUp Upload (IBM ---> Unix)

9. Remote Control of SIMTERM

Certain sequences of characters, when received over the communications line, will cause SIMTERM to execute certain functions, like clearing the screen, moving characters, turning the printer on and off. In the following descriptions ESC is Escape (ASCII 27), and Ctrl-X denotes the character generated by holding down Ctrl and typing X. I.e. Ctrl-A is ASCII 1. Note: in the descriptions "ESC H F" for example means ESCAPE, capital H, capital F, with NO SPACES.

ESC A -- move the cursor up one line
ESC B -- move the cursor down one line
ESC C -- move the cursor right one space
ESC D -- move the cursor left one space
ESC E -- take SIMTERM out of INSERT mode
ESC F -- remotely program a function key. After the F, send a number from 1 to 9 to indicate which key to program, or a 0 to program key 10. Everything following the numeral, up to, but not including a terminating Ctrl-Z is then stored in that function key. (See User Function Keys, above)
ESC H -- moves cursor to top left of screen (HOME)
ESC K -- clears the rest of the line, from cursor position over

ESC J -- clears display from cursor to end of line, and all lines below the cursor

ESC L -- inserts a line above the cursor by rolling bottom lines down. Bottom line of text is lost.

ESC M -- current line is deleted, lower lines scrolled up to close the gap

ESC P 1 -- turn on printer (Like Ctrl-PrtSc). Everything that now appears onscreen will be copied to the printer

ESC P 2 -- like P 1, but printer is put in COMPRESSED PRINT MODE.

ESC P E -- like P 1, but printer is put in EMPHASIZED PRINT MODE.

ESC P i -- doesn't turn printer on, just sets ITALICS mode. So, to print in ITALICS, first turn on printer (ESC P 1), then have the remote computer send this

ESC P P -- turn on printer, and set into PROPORTIONAL print mode.

ESC P O -- turn off printer. Screen is no longer copied to printer

ESC Q -- Enter INSERT mode. All subsequent characters will be inserted before the cursor on this line

ESC R -- Deletes the character at current cursor position.

ESC T A -- Cause SIMTERM to begin ADM3A simulation

ESC T a -- Cause SIMTERM to end ADM3A simulation

ESC T R -- Cause SIMTERM to set up to play ROGUE, so that arrow keys work.

ESC T r -- Cause SIMTERM to end rogue playing set up

ESC V S -- Cause SIMTERM to set up for vi, so that arrows, Ins, Del, PgUp and PgDn do reasonable things in this Unix text editor

ESC V s -- End Vi mode

ESC [xc -- causes character 'c' to be printed x number of times, i.e. to print 27 B's, send ESC [ESC B, since ESC has an ASCII value of 27.

ESC &dB -- set screen attribute to a value that equals BOLD on the printer (Actual screen color varies from monochrome to graphics monitor.)
ESC &dD -- set attribute to correspond to UNDERLINE
ESC &dH -- set attribute to be SUPERSCRIPT
ESC &dL -- set attribute to be SUBSCRIPT
ESC &d@ -- set attribute back to normal
ESC=y x -- move cursor to position y,x: where y and x are the ASCII characters which equal the desired position plus 32. I.e. to move to position 3,3 send ESC = # #, since the ASCII value of # (35) minus 32 is 3
ESC ^ -- request ID. SIMTERM will send back the string "IBM PC" terminated by a newline.

10. XMODEM Under UNIX

When using XMODEM with a UNIX system, the programs 'xrecv.c' and 'xtrans.c' should be uploaded and compiled on the host. These programs are setup for the System V release of UNIX and may need rework if used on other systems (e.g., Berkeley). To send a file from the PC->UNIX, type 'xrecv filename' on UNIX and when the prompt comes up to start transmission, hit Alt-T and follow the instructions.

To send a file from UNIX->PC, type 'xtrans filename' on UNIX and when the prompt comes up to set up the receive, hit Alt-R and follow the instructions.

If you are sending ASCII Text files, invoke the 'xrecv'/'xtrans' with the '-t' argument so that conversion is done between the DOS and UNIX conventions. The XMODEM protocol using the '-t' option is the safest way to transfer text files between systems since checks are made for successful transmission and any errors introduced by noisy lines will be filtered out.

Note: if the '-d' option is given to either program, the following will occur. xrecv leaves a running commentary on the connection status in "xr.err", and xtrans leaves a corresponding file called "xt.err". They

are used for debugging purposes.

11. Using SIMTERM with Other Modems

SIMTERM has been tailored to work with the VENTEL or Hayes modem. If you are using it with another modem then make sure that the modem is optioned to present the 'modem ready' signal. If you want to be able to 'dial' the phone number from the keyboard, you will also have to option it to present the 'carrier detect' signal. SIMTERM will not get past its initialization code unless it see both the carrier detect and the modem ready signals.

If you are using an acoustic coupler or a 212 'dial up', then when the message 'establish communications' appears, dial up the computer and enable the 'data' (e.g., hit the DATA button on the 212 or put the receiver in the coupler).

If you are using any modem (or are directly connected), and SIMTERM seems to hang right after the "Establish communications link" message, it probably means that your modem (or local connection) is not outputting the correct RS-232 signals for SIMTERM. If the equipment's not yours, or you don't understand how to fix the problem, or you're just too lazy to bother, try invoking SIMTERM with the -d flag. I.e. say "simterm -d" instead of just "simterm". This will keep SIMTERM from looking at the state of your modem, and your problem will disappear. If it still doesn't work, you can always give us a call.

12. Compiling the Simulator

If you look at the link control file 'simterm.lnk' you will see the programs that have to be compiled or assembled. You must use the MACRO assembler (masm). The file 'struct.mac' has some macros for structured assembly code which makes it easier to write. The file 'mactest.asm' has some examples of the use of the structured macros. The 'relations' between variable or constants are the same as the conditional jumps with the 'j' removed. 'long' can be appended to the macro to cause a long jump to be generated.

'com.asm' is the interface to the Async Comm line. The file 'comm.inc' has the entry points defined.

The file 'comm.inc' has the entry points defined. 'xxgraph.asm' has functions for cursor control and such from PASCAL programs. The file 'graph.inc' has the entry points defined.

'arglist.obj' and 'arglist.inc' provide 'argv' and 'argc' for the parameters to the PASCAL program. See 'mainsim.pas' for examples of their use. Most of the program will be self-explanatory (hopefully). Any questions, give me a call.

If you want to run SIMTERM on a PC with 128K, you will have to have access to the compiler and macro assembler. Instead of re-compiling and linking the programs in simterm.lnk, use "smterm.lnk". This will build a SIMTERM without VenTel features and without Login Scripts, but it also knocks about 50K off the '.exe' file.

13. Funny Error Conditions

Sometimes SIMTERM will appear to lockup and not accept any input from the keyboard (you will hear the bell indicating that the input buffer is full). This is usually caused by an incomplete ESCAPE sequence that has been sent to SIMTERM and it is waiting for the rest of the characters, which will probably never come from the computer. To 'break out' of this condition, depress at the same time the 'Ctrl', 'Shift', and 'Alt' keys on the left side of the keyboard. This will cause SIMTERM to terminate its 'wait state' and continue. This sometimes happens after dialing into a BBS or other system since you might get some noise of the line.

Also the transmitter may have missed an interrupt, and what you are typing is not sent. If you suspect this, hit Alt-F7.

If SIMTERM gets into a funny condition that you can not get it out of, then terminate it with 'Alt-F1' and then reinvoke it. This procedure will leave the PC attached to the system and you can pick up where you left off.

There have been problems on some systems that use PROKEY, NEWKEY, or other software that interprets the keyboard. If SIMTERM is locking up on you and you have a

program like this loaded, reboot the system without it and try it. Whatever those programs are doing, they are not very friendly to SIMTERM.

Jim Holtman
35 Dogwood Trail
Randolph, N. J. 07869
(201)-361-3395

Eric Holtman
Watson Dorm, Room 142
Station #2
Charlottesville,VA 22904-0029

Net Address: ...!harpo!whux1b!jph ...!harpo!whux1e!eric

If you add any interesting new features to the program, please keep me informed.

I have several other public domain programs that I will be willing to trade for ones that I don't have. Drop me a note (or a floppy disk) with what you have and we can setup a trade. Changes and improvements are continually being added to the program and if you are interested in what the changes are, just drop me a note (or a floppy) for the latest information.

APPENDIX J

ASSEMBLY LANGUAGE ROUTINES - IBM-PC

This Appendix contains the listings of the file "biosio.asm" and "dostime.asm", used in the IBM-PC Kalman Filter program to generated graphic output and to evaluate execution time respectively. The "biosio.asm" is also used by the plotting routine, PLOT.C . The files "biosio.asm" and "dostime.asm" are part of the Public Domain Library and can be found in the PC-SIG library disk number 216.

```
;                       *** biosio.asm ***
;
; IBM-PC microsoft "C" under PC-DOS v2.00
;
; MICROSOFT "C" callable 8088 assembly routines that
;interface directly with the basic I/O system (BIOS).
;
; NOTE -- The IBM Technical Reference Manual contains a
;listing of the BIOS and more complete descriptions of each
;interrupt.
;
; Written by L. Cuthbertson, April 1984
;
;************************************************************
;
PGROUP  GROUP   PROG
PROG    SEGMENT BYTE PUBLIC 'PROG'

    PUBLIC  BIOSINI,BIOSSET,BIOSPOS
    PUBLIC  BIOSUP,BIOSDWN,BIOSRCA,BIOSWCA,BIOSWC
    PUBLIC  BIOSWD,BIOSTTY,BIOSCUR

    PUBLIC  COMINI,COMOUT,COMIN,COMSTAT

    PUBLIC  INP,OUTP

    ASSUME  CS:PGROUP
;
;************************************************************
;
;           *** VIDEO I/O ***
;
; NOTE - the video I/O interrupt description starts on page
;A-43 of the Tech Ref Manual.
;
;************************************************************
;
; Initialize screen I/O using the BIOS set mode call
;
; synopsis  biosini(stype);
;
;       int stype;  screen type
;
;           0 = 40x25 BW (power on default)
;           1 = 40x25 Color
;           2 = 80x25 BW
;           3 = 80x25 Color
;           graphics mode
;           4 = 320x200 Color
;           5 = 320x200 BW
;           6 = 640x200 BW
```

```
;           internal use only
;           7 = 80x25 BW card
;
BIOSINI PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV AL,[BP+4]   ; SCREEN TYPE IN AL
    MOV AH,0        ; SET MODE FUNCTION
    INT 10H     ; BIOS VIDEO I/O INTERRUPT
    POP BP
    RET
BIOSINI ENDP
;
; Set the current cursor position.
;
; synopsis  biosset(irow,icol);
;
;       *** no value returned ***
;       int irow;   0 to 24
;       int icol;   0 to 79
;
BIOSSET PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV DH,[BP+4]   ; ROW
    MOV DL,[BP+6]   ; COLUMN
    MOV BH,0        ; CURRENT PAGE NUMBER
    MOV AH,2        ; CURSOR POSITION SET FUNTION NUMBER
    INT 10H     ; VIDEO I/O INTERRUPT
    POP BP
    RET
BIOSSET ENDP
;
; Return the current cursor postion.
;
; synopsis  iret = biospos();
;
;       int iret;   high order bits contain row
;                   low order bits contain column
;
BIOSPOS PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV BH,0        ; CURRENT PAGE NUMBER
    MOV AH,3        ; CURSOR POSITION FUNCTION NUMBER
    INT 10H     ; VIDEO I/O INTERRUPT
    MOV AH,DH       ; MOVE INT RETURN INTO FUNCTION RETURN
    MOV AL,DL       ; DITTO
    POP BP
    RET
BIOSPOS ENDP
```

```
;
; Read the contents of a given screen cell.
;
; synopsis  iret=biosrca();
;
;       int iret;   high order bits contain attributes
;                   low order bits contain character
;
; NOTE - Attributes are defined on page 13-9 of the DOS v2.0
;manual
;
BIOSRCA PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV BH,0        ; ACTIVE DISPLAY PAGE
    MOV AH,8        ; READ CHARACTER + ATTRIBUTES FUNCTION
    INT 10H     ; VIDEO I/O INTERRUPT
    POP BP
    RET
BIOSRCA ENDP
;
; Write a character to the screen - with attributes.
;
; synopsis  bioswca(char,count,att);
;
;       *** no value returned ***
;       int char;   character to output
;       int count;  number of times to output character
;       int att;    character attribute
;
; NOTE - Attributes are defined on page 13-9 of the DOS v2.0
;manual
;
BIOSWCA PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV AL,[BP+4]   ; CHARACTER
    MOV CX,[BP+6]   ; NUMBER OF CHARACTERS TO WRITE
    MOV BL,[BP+8]   ; CHARACTER ATTRIBUTE
    MOV BH,0        ; ACTIVE DISPLAY PAGE
    MOV AH,9        ; WRITE CHARACTER/w ATTRIBUTES FUNCTION
    INT 10H     ; VIDEO I/O INTERRUPT
    POP BP
    RET
BIOSWCA ENDP
;
; Write a character to the screen - no attributes.
;
; synopsis  bioswc(char,count);
;
```

```
;       *** no value returned ***
;       int char;   character to output
;       int count;  number of times to output character
;
BIOSWC  PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV AL,[BP+4]   ; CHARACTER
    MOV CX,[BP+6]   ; NUMBER OF CHARACTERS TO WRITE
    MOV BL,0;       ; CHARACTER ATTRIBUTE - NULL
    MOV BH,0        ; ACTIVE DISPLAY PAGE
    MOV AH,10       ; WRITE CHARACTER ONLY FUNCTION NUMBER
    INT 10H     ; VIDEO I/O INTERRUPT
    POP BP
    RET
BIOSWC  ENDP
;
; Write a dot in graphics mode.
;
; synopsis  bioswd(irow,icol);
;
;       *** no value returned ***
;       int irow;
;       int icol;
;
BIOSWD  PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV DX,[BP+4]   ; ROW
    MOV CX,[BP+6]   ; COLUMN
    MOV AL,1        ; GREEN COLOR
    MOV AH,12       ; WRITE A DOT FUNCTION NUMBER
    INT 10H     ; VIDEO I/O INTERRUPT
    POP BP
    RET
BIOSWD  ENDP
;
;
; Return the current video state of the screen.
;
; synopsis  iret = bioscur();
;
;       int iret;   low bits are the mode currently set
;                   (see biosini for description)
;                   high  bits  are  the  number  of  columns on
screen
;
BIOSCUR PROC    NEAR
    PUSH    BP
    MOV BP,SP
```

```
    MOV AH,15       ; CURRENT VIDEO STATE FUNCTION
    INT 10H     ; BIOS VIDEO I/O INTERRUPT
    POP BP
    RET
BIOSCUR ENDP
;
;*************************************************************
;
;           *** communications port ***
;
; NOTE - the communications port I/O is described starting
;on page A-20 of the Tech Ref Manual.
;
;*************************************************************
;
; Initialize the communications port.
;
; synopsis  iret = comini(port,params);
;
;       int iret;   return status (see comstat)
;       int port;   communications port to initialize (0,1)
;       int params; bit pattern for initialization -
;
;   7   6   5   4   3   2 1 0
;     -BAUD RATE-       -PARITY- -STOP BIT- -WORD LENGTH-
;
;   000 - 110       XO - NONE 0 - 1 10 - 7 BITS
;   001 - 150       01 - ODD 1 - 2 11 - 8 BITS
;   010 - 300       11 - EVEN
;   011 - 600
;   100 - 1200
;   101 - 2400
;   110 - 4800
;   111 - 9600
;
COMINI  PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV DX,[BP+4]   ; COMM PORT TO INITIALIZE
    MOV AL,[BP+6]   ; PARAMETERS
    MOV AH,0        ; INITIALIZATION FUNCTION
    INT 14H     ; COMM PORT I/O INTERRUPT
    POP BP
    RET
COMINI  ENDP
;
; Write a character to the communications port.
;
; synopsis  iret = comout(port,c);
;
```

```
;       int iret;   error return ( >127 if error occured)
;                 - see AH under modem control
;       int port;   communications port to write (0 or 1)
;       int c;      character to write.
;
COMOUT  PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV DX,[BP+4]   ; COMM PORT TO WRITE
    MOV AL,[BP+6]   ; CHARACTER TO WRITE
    MOV AH,1        ; WRITE CHARACTER FUNCTION
    INT 14H     ; COMM PORT I/O INTERRUPT
    POP BP
    RET
COMOUT  ENDP
;
; Read a character from the communications port.
; Waits for character if one is not ready.
; See \comm\ibmtty.c for example of polling comm port
; for character without wait.
;
; synopsis  c = comin(port);
;
;       int c;      character read from comm port,
;                 > 127 if error or no character ready.
;                 - see AH under modem control
;       int port;   communications port to read (0 or 1)
;
COMIN   PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV DX,[BP+4]   ; COMM PORT TO READ
    MOV AH,2        ; READ CHARACTER FUNCTION
    INT 14H     ; COMM PORT I/O INTERRUPT
    POP BP
    RET
COMIN   ENDP
;
; Check the line and modem status
;
; synopsis  iret = comstat(port);
;
;       int iret;   ; line and modem status
;       AH - high order bits contain line control status
;       bit 7 = time out
;       bit 6 = trans shift register empty
;       bit 5 = trans holding register empty
;       bit 4 = break detect
;       bit 3 = framing error
;       bit 2 = parity error
```

```
;       bit 1 = overrun error
;       bit 0 = data ready
;
;       AL - low order bits contain modem status
;       bit 7 = received line signal detect
;       bit 6 = ring indicator
;       bit 5 = data set ready
;       bit 4 = clear to send
;       bit 3 = delta receive line signal detect
;       bit 2 = trailing edge ring detector
;       bit 1 = delta data set ready
;       bit 0 = delta clear to send
;
; Note - from page A-21 of Tech Ref Manual
;
;       int port;    ; communications port to check
;
COMSTAT PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV DX,[BP+4]    ; COMM PORT TO CHECK
    MOV AH,3         ; STATUS FUNCTION
    INT 14H      ; COMM PORT I/O INTERRUPT
    POP BP
    RET
COMSTAT ENDP
;
;*************************************************************
;
;           *** io.asm ***
;
; IBM-PC 8088 assembly for interface with microsoft "C"
;under PC-DOS
;
; "C" callable subroutines that provide absolute pointer
;addressing.
; Pointers within a IBM-PC microsoft "C" program are
;relative to the start of the programs data segment.  These
;subroutines provide a mechanism to address absolute memory
;locations.
;
; Supplied by microsoft - commented by L. Cuthbertson, April
;1984
;
;*************************************************************
;
; Read an absolute memory location.
;
; synopsis  c = readabs(loc);
;
```

```
;       int c;      contents of memory location
;       int loc;    absolute memory location in hex
;
INP PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV DX,[BP+4]   ;GET LOCATION ADDRESS
    IN  AL,DX       ;READ LOCATION
    XOR AH,AH       ;CLEAR HIGH BYTE
    POP BP
    RET
INP ENDP
;
; Write an absolute memory location.
;
; synopsis  writeabs(loc,c);
;
;       int loc;    absolute memory location in hex
;       int c;      integer to write to memory location
;
OUTP    PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV DX,[BP+4]   ;GET LOCATION ADDRESS
    MOV AX,[BP+6]   ;GET ADDRESS OF INTEGER TO WRITE
    OUT DX,AL       ;WRITE TO MEMORY LOCATION
    POP BP
    RET
OUTP    ENDP
;
;*************************************************************
;
PROG    ENDS
    END
;
;                   *** dostime.asm ***
;
; IBM-PC microsoft "C" under PC-DOS
;
; Microsoft "C" callable 8088 assembly routine that uses a
;DOS function call to obtain and return the hours, minutes,
;seconds, and hundreths of a second as integers.
;
; synopsis  dostime(&hour,&min,&sec,&ths);
;
;       *** no function value returned ***
;       int hour;   0-23  military time
;       int min;    0-59
;       int sec;    0-59
;       int ths;    0-99
```

```
;
; Written by L. Cuthbertson, May 1984
;
;*************************************************************
;
PGROUP  GROUP   PROG
PROG    SEGMENT BYTE PUBLIC 'PROG'
    PUBLIC  DOSTIME
    ASSUME  CS:PGROUP
;
;*************************************************************
;
DOSTIME PROC    NEAR
    PUSH    BP
    MOV BP,SP
    MOV AH,2CH      ; DOS GET TIME FUNCTION NUMBER
    INT 21H     ; DOS FUNCTION CALL INTERRUPT
    MOV AL,CH       ; MOVE HOURS INTO AX
    XOR AH,AH       ; CLEAR HIGH BYTE
    MOV DI,[BP+4]   ; ADDRESS OF HOURS VARIABLE
    MOV [DI],AX     ; MOVE HOURS INTO HOURS VARIABLE
    MOV BL,CL       ; MOVE MINUTES INTO BX
    XOR BH,BH       ; CLEAR HIGH BYTE
    MOV DI,[BP+6]   ; ADDRESS OF MINUTES VARIABLE
    MOV [DI],BX     ; MOVE MINUTES INTO MINUTES VARIABLE
    MOV CL,DH       ; MOVE SECONDS INTO CX
    XOR CH,CH       ; CLEAR HIGH BYTE
    MOV DI,[BP+8]   ; ADDRESS OF SECONDS VARIABLE
    MOV [DI],CX     ; MOVE SECONDS INTO SECONDS VARIABLE
    XOR DH,DH       ; CLEAR HIGH BYTE
    MOV DI,[BP+10]  ; ADDRESS OF THS VARIABLE
    MOV [DI],DX     ; MOVE 1/100THS INTO THS VARIABLE
    POP BP
    RET
DOSTIME ENDP
;
PROG    ENDS
    END
;
```

LIST OF REFERENCES

1. O'Brien, Paul A., An Application of Kalman Filtering to Underwater Tracking, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1980.

2. Kirk, Donald E., Optimal Estimation: An Introduction to the Theory and Applications, EE3310 Class Notes, Unpublished.

3. Bozic, S. M., Digital and Kalman Filtering, Edward Arnold (Publishers) Ltd. 1981.

4. Hunter, Bruce H., Understanding C, Sybex Computer Books, 1984.

5. Jamsa, Kris, The C Library, Osborne McGraw-Hill, 1985.

6. Kernighan, Brian W. and Ritchie, Dennis M., The C Programming Language, Prentice-Hall, Inc., 1978.

7. Schustack, Steve, Variations in C, Microsoft Press, 1985.

8. IBM, Technical Reference for the IBM Personal Computer, Personal Computer Hardware Reference Library, 1983.

9. Norton, Peter, The Peter Norton Programmer's Guide to the IBM PC, Microsoft Press, 1985.

10. Peerless Engineering Service, Fortran Scientific Subroutine Library, John Wiley & Sons,Inc, 1984.

11. CIE Systems, Regulus Reference Manual, Bussiness Computer Operating Systems Reference Series, Bussiness Computer Software Library, 6 DRG-0012-001, 68 DOC-3041, Second Revision.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	2
4.	Professor H. A. Titus, Code 62Ts Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	5
5.	Associate Professor A. Gerba, Code 62Gz Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6.	Commanding Officer Naval Underseas Weapons Engineering Station Keyport, Washington 98345	2
7.	Capt. Jose A. Fernandes CTA - IAE - ESB 12200 - SAO JOSE DOS CAMPOS - S.P. BRASIL	2

217420

Thesis
F26464 Ferreira
c.1 Kalman filter real time tracking using microprocessors and a new language for military software.