2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A MICROCOMPUTER-BASED CONTROLLER FOR AN AUTONOMOUS UNDERWATER VEHICLE (AUV)

by

William D. Riling

March 1990

Thesis Co-Advisor:	Roberto Cristi
Thesis Co-Advisor:	John Yurchak

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b DECLASSIFICATION DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): EC	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a NAME OF FUNDING SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) A MICROCOMPUTER-BASED CONTROLLER FOR AN AUTONOMOUS UNDERWATER VEHICLE (AUV)

12 PERSONAL AUTHOR(S) RILING, William D.

13a TYPE OF REPORT: Master's Thesis	13b TIME COVERED FROM ____ TO ____	14 DATE OF REPORT (Year, Month, Day): 1990 March	15 PAGE COUNT: 159

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those fo the author and do not reflect the official policy or position of the Department of Defense or the US Government.

17 COSATI CODES: FIELD | GROUP | SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Autonomous Underwater Vehicle; variable structure control; Doyle-Stein observer; synchro-to-resolver conversion; sensor error detection

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

Considerations of real-time control problems for an Autonomous Underwater Vehicle (AUV) are addressed in this research. Among these problems is the ability to control the submersible given its highly nonlinear operating environment. In order to account for these variations, robust control techniques must be used. In particular, Variable Structure Control (VSC) with Doyle-Stein Observer has proven to produce optimal results while maintaining a high degree of robustness. This led to the development of a real-time error detector using the robust observer to provide system redundancy through software. The culmination of this work is a real-time autopilot written in the "C" language which is ready for implementation and testing in the Naval Postgraduate School AUV prototype.

We also address the aspect of real-time signal processing and conditioning in terms of Synchro-to-Resolver Conversion and anti-aliasing filters.

20 DISTRIBUTION AVAILABILITY OF ABSTRACT: [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: R CRISTI/J YURCHAK	22b TELEPHONE (Include Area Code): 408-646-2223/3390; 22c OFFICE SYMBOL: EC/Cx CS/Yu

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603 UNCLASSIFIED

19. cont.
The synchro problem involves converting a nonpotentiometric directional gyro output to a natural binary format which calls for an intricate design of power transformers, analog-to-digital converter, and passive element components. Lastly, the use of Generalized-Immittance Converter circuitry in the design of very low frequency anti-aliasing filter applications is developed and tested.

A Microcomputer-Based Controller For An Autonomous Underwater Vehicle (AUV)

by

William D. Riling
Lieutenant, United States Navy
B.S., Middle Tennessee State University, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
MARCH 1990

Author: [illegible]
William D. Riling

Approved by: [illegible]
Roberto Cristi, Thesis Co-Advisor

[illegible]
John M. Yurchak, Thesis Co-Advisor

[illegible]
John P. Powers, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

Considerations of real-time control problems for an Autonomous Underwater Vehicle (AUV) are addressed in this research. Among these problems is the ability to control the submersible given its highly nonlinear operating environment. In order to account for these variations, robust control techniques must be used. In particular, Variable Structure Control (VSC) with Doyle-Stein Observer has proven to produce optimal results while maintaining a high degree of robustness. This led to the development of a real-time error detector using the robust observer to provide system redundancy through software. The culmination of this work is a real-time autopilot written in the "C" language which is ready for implementation and testing in the Naval Postgraduate School AUV prototype.

We also address the aspect of real-time signal processing and conditioning in terms of Synchro-to-Resolver Conversion and anti-aliasing filters. The synchro problem involves converting a nonpotentiometric directional gyro output to a natural binary format which calls for an intricate design of power transformers, analog-to-digital converter, and passive element components. Lastly, the use of Generalized-Immittance Converter circuitry in the design of very low frequency anti-aliasing filter applications is developed and tested.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

No single entity in a major endeavor such as this thesis can stand alone without support, encouragement, and love. In my case, Professor Roberto Cristi provided the support through his brilliance in the field of discrete control systems. He has left an indelible mark in my problem solving techniques for which I am truly grateful. The encouragement and drive to master an exiting programing language, such as "C", was given to me by LCDR John Yurchak. His gift for conveying concepts and his mastery of the language has instilled a passion for "C" programing with me. Finally, I would like to recognize my wife, Ramona, who provided all of the above requirements. I shall never forget her patience and love during this time.

I. INTRODUCTION

The Autonomous Underwater Vehicle (AUV) project was conceived at the Naval Postgraduate School in the early 1980's. The purpose of the project was the development of an underwater vehicle as a testbed for research. Most of the early work on this project was on model development and applications of the hydrodynamic equations essential in modeling vehicle motions accurately. More recently autonomous navigation, obstacle avoidance, path planning, and advanced control techniques have become the focus of current research, in particular adaptive and nonlinear control design.

To get an understanding of the goals of the project and the need for research in the aforementioned areas, a basic knowledge of the concept and scope of operation of the AUV is needed. The NPS vehicle is designed to be an untethered submersible capable of autonomous operation for an extended period of time. Conceptually, the vehicle is placed in an unknown environment where it will map the immediate surrounding area while evading static and dynamic obstacles. Upon completion of its mission, the vehicle will return to its base for retrieval and subsequent down-loading of its mission data.

The submersible must be capable of operating in various dynamic environments in order to complete this type of mission. This can be achieved by adding a level of intelligence, at the expense of increased complexity of the real-time control problem.

The purpose of this work is to design and implement a real-time control system for rapid maneuvers of the AUV. The technique used is based on Variable Structure Control, with subsequent computer code developed in a high level language. Controller requirements are in terms of performance (i.e., fast response) and robustness in the presence of modeling errors and sensor failures.

A robust observer has been designed for the purpose of redundancy so that system stability can still be maintained when one or more sensors (pitch and pitch rate, for example) fail. Upon successful modeling and testing of this complete autopilot, the code was converted into the "C" language for real-time implementation. The converted "C" code then was tested using an interface module that emulates real-time sampling conditions and acts as a link between model, controller, and local path planner. A further aspect of this work is the design and testing of various hardware components of the electronic signal processing and conditioning network proposed for the AUV.

This thesis discusses issues ranging from automatic control systems to hardware implementation of electronic signal processing equipment. In Chapter II the autopilot control principle and algorithm are developed. Chapter III addresses considerations of real-time code development and testing. In Chapter IV hardware design issues concerning signal conditioning and information encoding are covered. The final chapter contains conclusions and general project assessment. All pertinent computer code and hardware designs are located in the appendices.

II. REAL-TIME CONTROL METHODS FOR DEPTH AND STEERING

A. BACKGROUND INFORMATION

An accurate model of the system must exist in order to develop and test control algorithms. This model should allow for all possible dynamic situations. In the case of the AUV, a model based on the United States Navy's Swimmer Delivery Vehicle is used. This model is discussed at length in Chapter III. The estimated dynamics of the prototype AUV helped to determine the control methodology used in the real-time algorithms. The uncertainties in the dynamic response of the vehicle due to the operating enviroment means a robust control scheme has to be used. The control techniques used in steering and depth are distinctly different due to these vehicle dynamics and purposed operating conditions. This chapter discusses the development, testing, simulation, and future real-time implementation of an autopilot for the AUV. The derivation in this chapter of the variable structure algorithm is based on work by Cristi [Ref. 1].

B. VARIABLE STRUCTURE CONTROL ALGORITHM DERIVATION

This technique is used in the design of regulators for nonlinear time varying uncertain systems [Ref. 2]. Variable structure control, commonly referred to as sliding mode control, can be based on a nominal model, and it can account for uncertainties in the dynamic response of the plant. The main feature of this method of control is the fact that it is very robust in nature. Since the uncertainties can be handled in the control law by assuming a known bound on the uncertainty. In particular let us consider a nonlinear state space model of the form

$$\dot{x} = f\,(x,u,t) \tag{2-1}$$

where f is not exactly known. The function can be divided into the sum of a known portion and an uncertain component

$$f\,(x,u,t) = f_0\,(x,u,t) + \Delta f\,(x,u,t) \tag{2-2}$$

where f_0 is the known nominal function. A bound on the uncertainty Δf is assumed to be known as

$$F\,(x,u,t) \geq |\,\Delta f\,(x,u,t)\,| \tag{2-3}$$

where F is a known for every x, u, and t. Although F could be arbitrarily large, an extreme value of the bound will result in undesirable chattering of the control signal. T variable structure control technique is based on the definition of a sliding surface, $\sigma(x) = 0$, and switching law for which the feedback control is of the form

$$u(x) = \begin{cases} u_+(x) & \textit{if} \quad \sigma(x) \geq 0 \\ u_-(x) & \textit{if} \quad \sigma(x) < 0 \end{cases} \tag{2-4}$$

The switching law is determined so that the state is driven from any initial condition $x(t_0)$ to the sliding surface $\sigma(x) = 0$ where it stays (ideally, at least) for all subsequent times. In selecting the sliding surface, the only condition which must be satisfied is that of stable dynamic response to any set of initial conditions which may be imposed. This can be expressed mathematically as the following:

$$\sigma(x(t)) = 0 \quad \textit{for all} \quad t > t_0 \quad \rightarrow \quad \lim_{t \to \infty} x(t) = 0 \tag{2-5}$$

Based on this, a possible choice of $\sigma(x)$ is a linear sliding surface

$$\sigma(x) = s^T x \tag{2-6}$$

where s is a left eigenvector of A

$$s^T A = -\lambda s^T \tag{2-7}$$

This definition combined with the state space model

$$\dot{x} = Ax + bu + \Delta f \tag{2-8}$$

yields the dynamics of σ. Thus, by combining (2-7) and (2-8)

$$s^T \dot{x} = -\lambda s^T bu + s^T \Delta f \tag{2-9}$$

this expression is obtained. Next let $\sigma(t) = s^T x(t)$, a scalar signal, and substitute this expression into Equation (2-9) to obtain

$$\dot{\sigma}(t) = -\lambda\sigma(t) + s^T bu + s^T \Delta f \tag{2-10}$$

Now the control input u can be defined as

$$u(t) = -K(t)\, sign(\sigma(t)) \tag{2-11}$$

where $K(t)$ is a time-varying gain to be chosen so that we define the Lyapunov function

$$V(\sigma) = \frac{1}{2}\,(\sigma(t))^2 \tag{2-12}$$

which guarantees that the *sliding surface will reach 0, equilibrium, in a finite amount of* time. This is depicted in Figure 1 which illustrates that, regardless of intial condition, the state trajectory is driven to the sliding surface. This can be found by multiplying Equation (2-10) by $\sigma(t)$

$$\sigma(\dot{\sigma}(t)) = -\lambda\sigma(t)^2 + s^T bu\sigma + s^T \Delta f\sigma = \dot{V}(\sigma) \tag{2-13}$$

which corresponds to the first derivative of the Lyapunov function. If we choose u as in (2-11) and

$$K(t) \geq \frac{|s^T F|}{|s^T b|} \tag{2-14}$$

then

$$\dot{V}(\sigma) < 0 \qquad and \qquad V(\sigma) \to 0 \tag{2-15}$$

This in turn implies that $\sigma(x(t)) \to 0$ (i.e., the state approaches the sliding surface $\sigma(x) = 0$). In fact, by Equations (2-10) and (2-12) we obtain

$$\dot{V}(\sigma) = -\lambda\sigma(t)^2 - (s^T b)K(t)|\sigma| + s^T \Delta f \sigma \tag{2-16}$$

In order for the sliding surface to behave as in Figure 1 over a finite interval of *time, the following condition must be satisfied:*

$$\dot{\sigma}(t) = -K(t)^2 sign(\sigma(t)) \tag{2-17}$$

The condition of Equation (2-17) is satisfied by choosing the control input as

$$\hat{u} + \bar{u} = u = -(s^T b)^{-1} s^T A x - K^2 (s^T b)^{-1} sign(\sigma(t)) \tag{2-18}$$

Several important notes concerning the two previous equations should be addressed at this point. The selection of K(t) should be made only slightly larger than F, the known portion

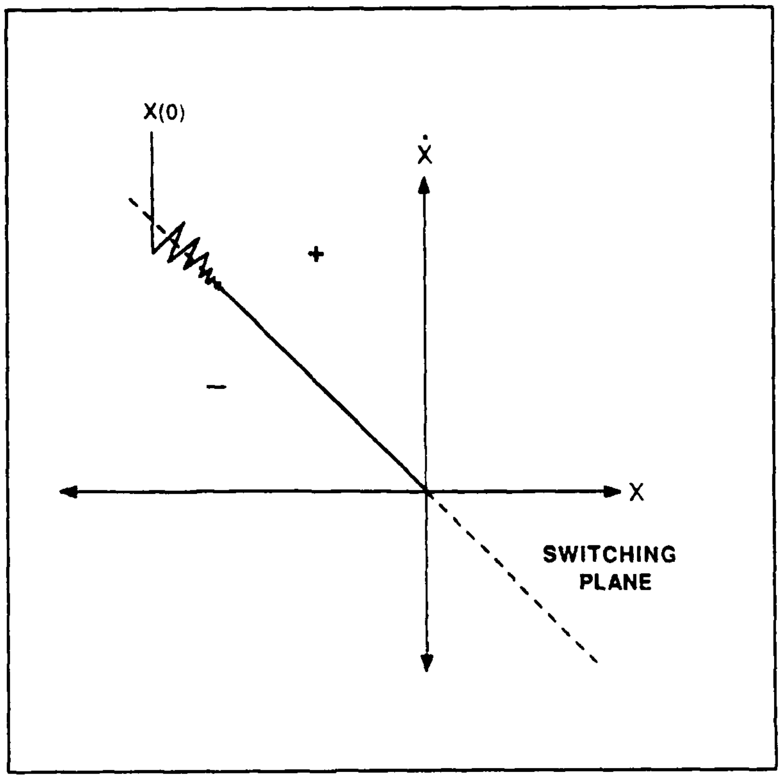


Figure 1. Sliding Surface Representation

of the uncertainty. This is due to the fact that, as the gain K increases, more chatteringwill be introduced on the sliding plane. From Equation (2-18), we note that the control u is composed of two parts,

$$\hat{u} = -(s^T b)^{-1} s^T A x \tag{2-19}$$

which is a linear feedback law, and

$$\bar{u} = -K^2 (s^T b)^{-1} sign(\sigma) \tag{2-20}$$

This part is a nonlinear feedback with its sign toggling between plus and minus according to which side of the sliding plane the system is located in. The caveat placed on the magnitude of K must also include that it must be large enough to allow the system to possess the required degree of robustness to handle the uncertainties brought about by the operating enviroment [Ref. 1].

C. COMPUTER CODING AND TESTING OF THE ALGORITHM

The first programs generated were developed on the IBM-PC using Matlab for code generation. The real-time controller that was eventually realized was much more complex than this initial routine which only tested the nonadaptive sliding mode control derived in the preceding section. The Matlab code, dauv.m, is included in Appendix A. The primary purpose of this code was to test the basic depth controller and obtain a set of steady-state feedback gains. These gains were associated with the following states: pitch, pitch rate, and depth. The steady-state feedback gains were found using the linear-quadratic-regulator function in Matlab which is based on a solution to the Riccati equation. These gains were based on the nominal state space model of the SDV, and, as such, will require adjustment in the future to enable the AUV to perform as designed.

The state space model used in the depth control algorithm is based on previous work by Schwartz [Ref. 3]. This model is discussed in detail in Chapter IV. The Matlab code is based on the variable structure development of the preceding se^tion. The steady state feedback gains are contained in the variable L which is a vector with three components. The program calls a model of the SDV which contains the dynamic equations of motion developed by Boncal [Ref. 4]. In order to pass the updated information determined by this model, an array containing all twelve states has been set up. The states of this array which are of interest to the diving algorithm are: state(5) = pitch rate, state(9) = actual vehicle depth, and state(11) = pitch. In an effort to simplify the code and to reduce the error inherent with round off, the depth error, "z", is normalized with the absolute speed. The sliding surface is equated by multiplying the left eigenvector of the state space model with the error vector composed of pitch, pitch rate, and the normalized depth error. As shown in the derivation of the sliding mode controller, the input of the dive command is composed of a linear feedback component $\hat{u}$ and a nonlinear feedback portion $\bar{u}$. The maximum deflection of the dive planes is 0.4 radians which is a software factor and has yet to be determined in hardware experimentation.

1. Simulation Results

The code is simulated using iteration techniques to varying orders of depth. In Figure 2 the graphical representation of the actual vehicle depth is shown. This graph shows that, for a commanded depth of 50 feet based on an initial or reference depth, the vehicle has no overshoot and settles to the desired depth change in a minimum of time. This simulation is based on a vehicle speed of approximately 3 knots which is the designed cruising speed of the AUV. Additional simulations have been run for varying depth commands with similar results (i.e., no overshoot with a minimum settling time). For the dive depicted in Figure 2, the corresponding dive plane action is shown in Figure 3. We note that the action of the dive plane is smooth and does not exceed 0.4 radians. In direct correlation with the graph of the vehicle depth, we note that the dive plane adjusts to not allow the vehicle to exceed a declination with respect to reference normal of approximately 0.8 radians. This is accomplished by feeding back the pitch rate and pitch states. Figure 4 is a depiction of the pitch rate which is a direct reflection of the divefin action. A measurement of the accuracy and smoothness of the vehicle trajectory can be assessed by graphical representation of the pitch rate. The pitch rate should possess a nearly symmetric shape with respect to either side of the mean value which is zero for constant trajectories. This is, in fact, the case of the graph in Figure 4. The last graph of this trial simulation represents vehicle pitch over the run. When the submersible approaches the commanded depth, a gradual decrease in pitch should occur as depicted in Figure 5. This decrease in pitch should go to zero for constant trajectories as is the case. This indicates that the vehicle is coincident to the reference axis of the system.

The last graph of this set, Figure 6, depicts the sliding surface behavior. This graph indicates that a stable sliding plane exist, and that, for the given initial condition, the state trajectory error will be driven to zero along this plane. Many simulations have been run for varying conditions and depths. All of the data collected indicated a stable and highly accurate controller which was the goal of this portion of the project.

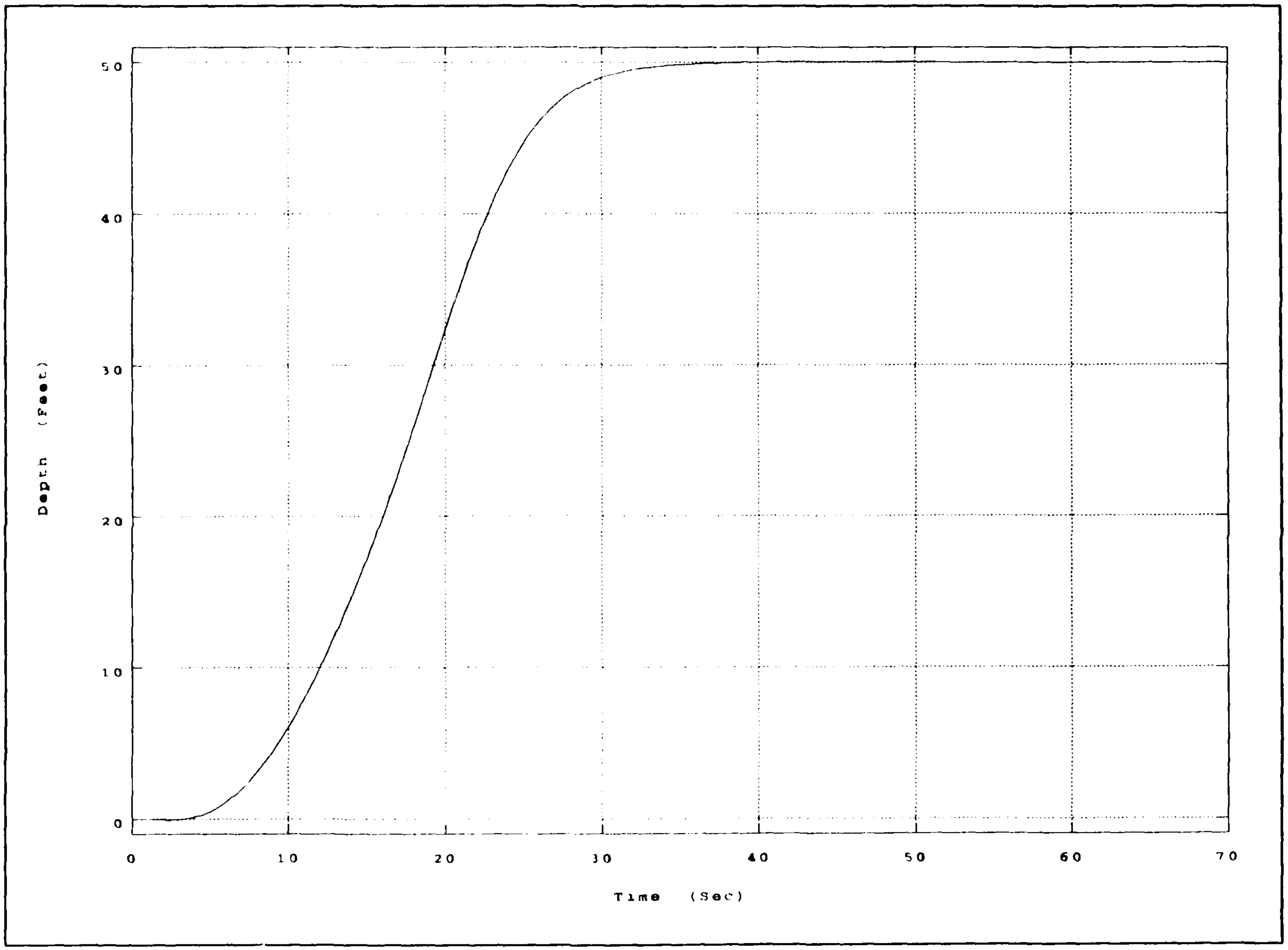


Figure 2. Actual Depth for Variable Structure Controller Test

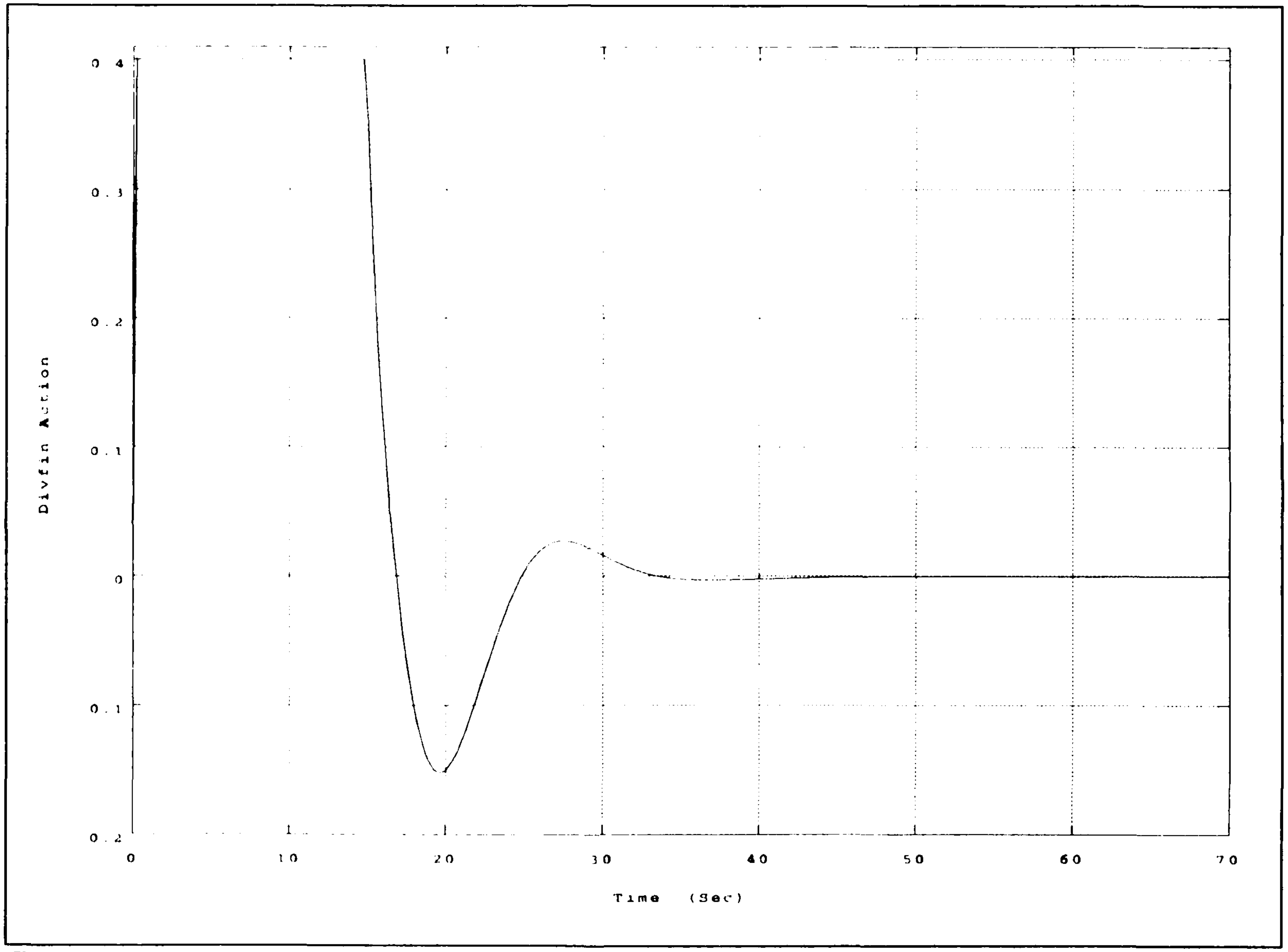


Figure 3. Divefin Response to a Change in Depth of 50 Feet

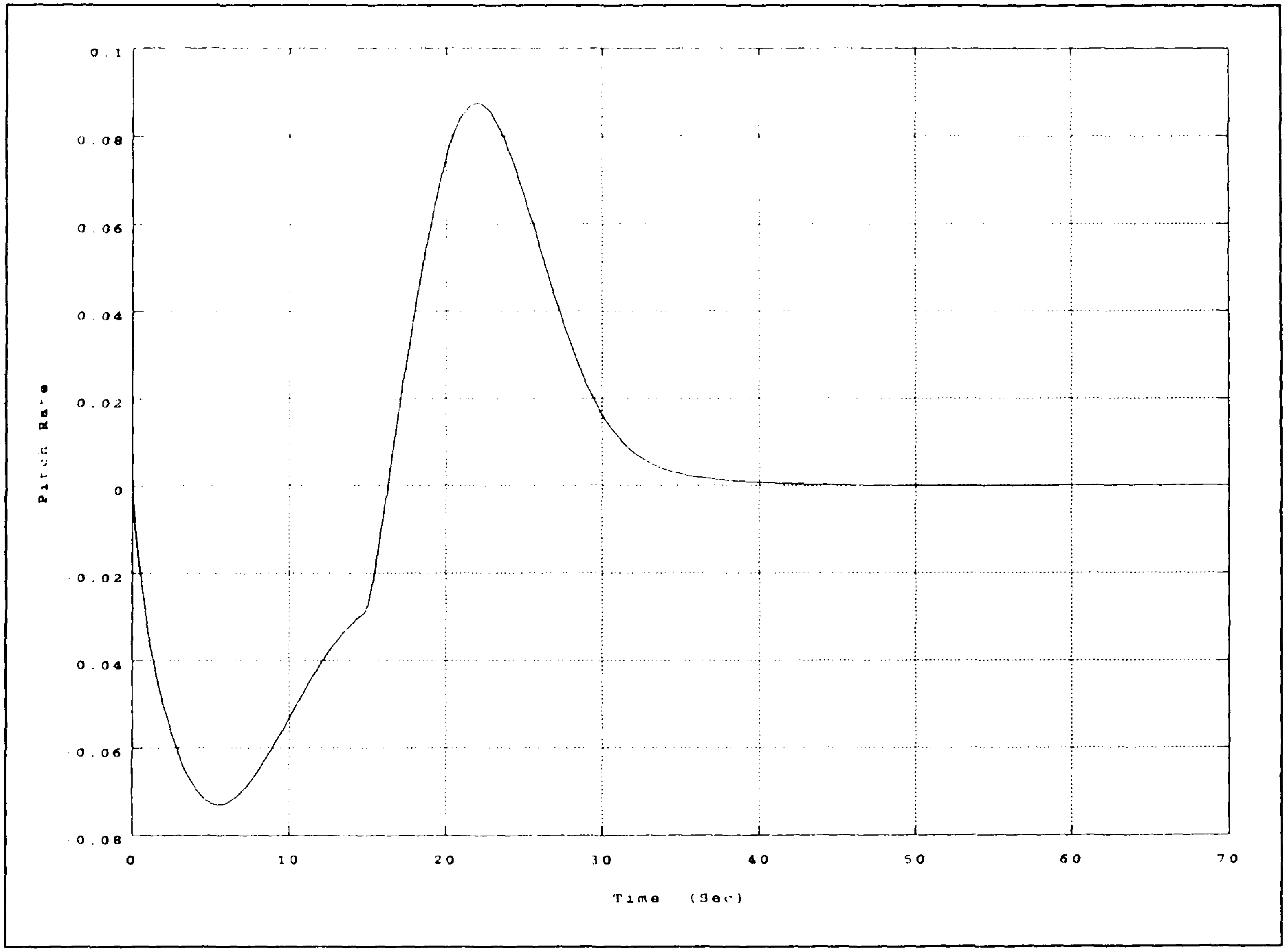


Figure 4. Pitch Rate Response to a Change in Depth of 50 Feet

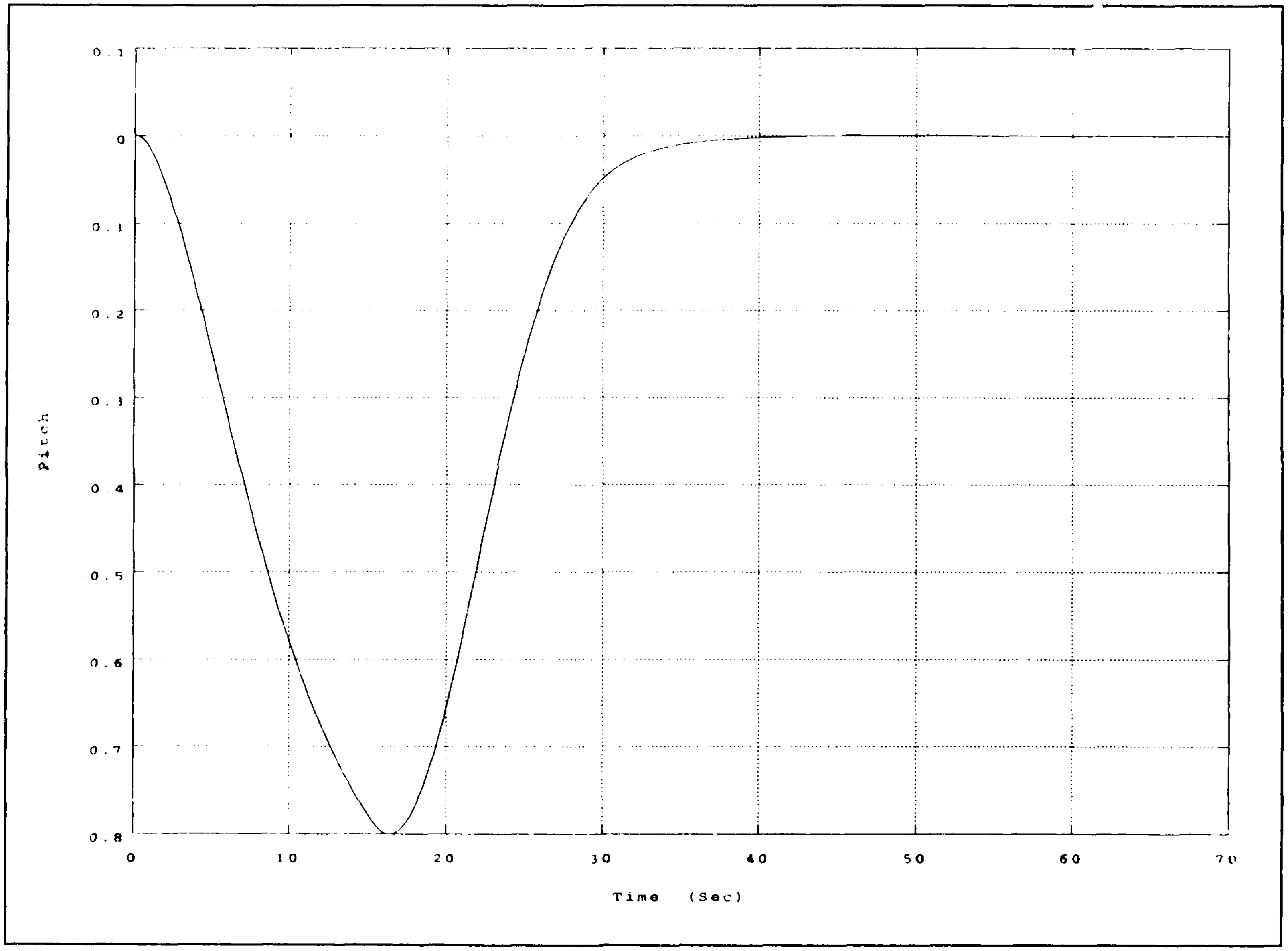


Figure 5. Pitch Response to a Change in Depth of 50 Feet

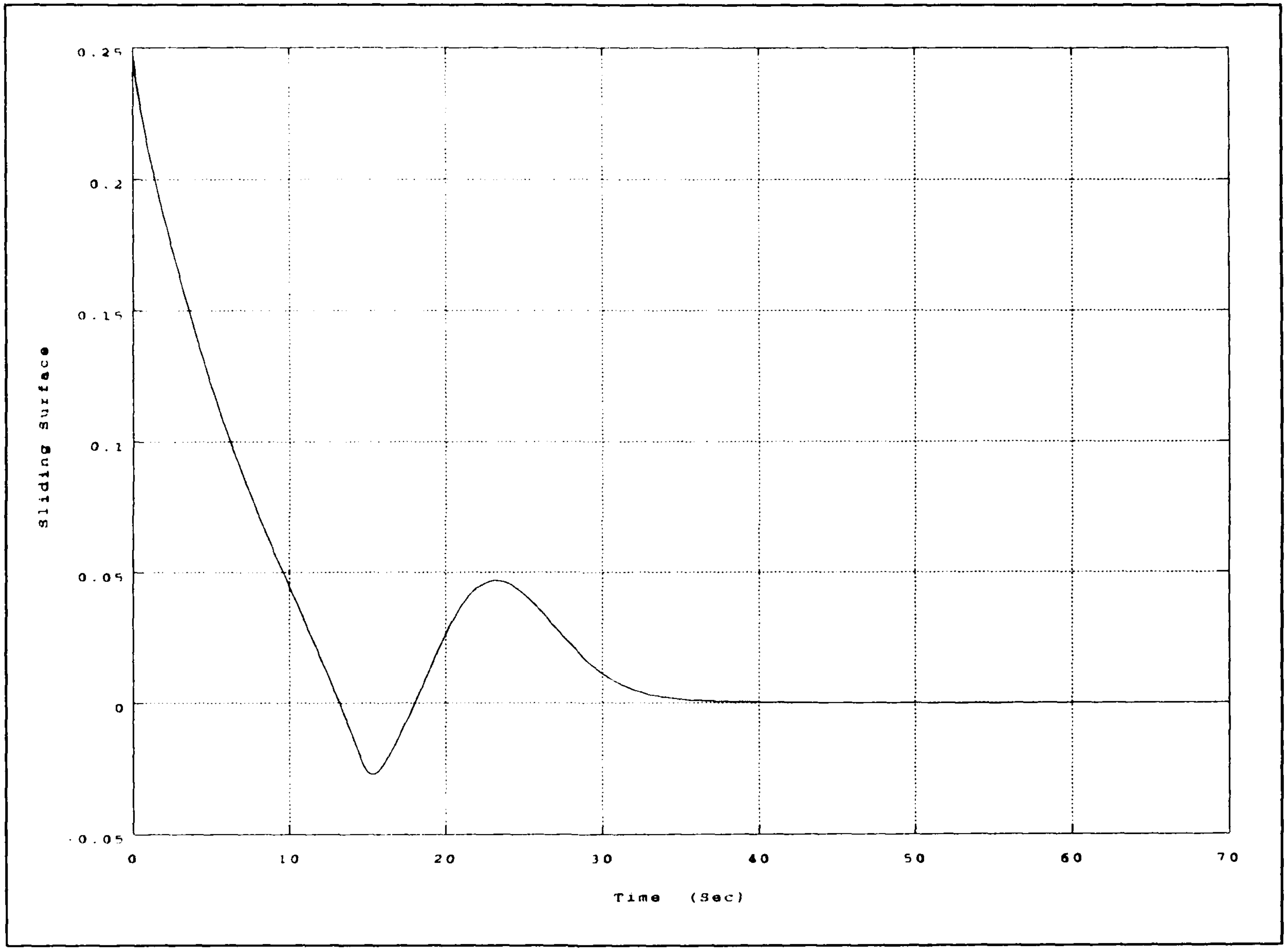


Figure 6. Sliding Plane for Change in Depth of 50 Feet

D. ROBUST OBSERVER DESIGN

The variable structure algorithm developed in the previous section is very robust in behavior. The sliding mode control in general can handle any nonlinear perturbation due to this robustness. This concept is shown in Figure 7 where f, the nonlinear perturbation, which is possibly state dependent is shown.

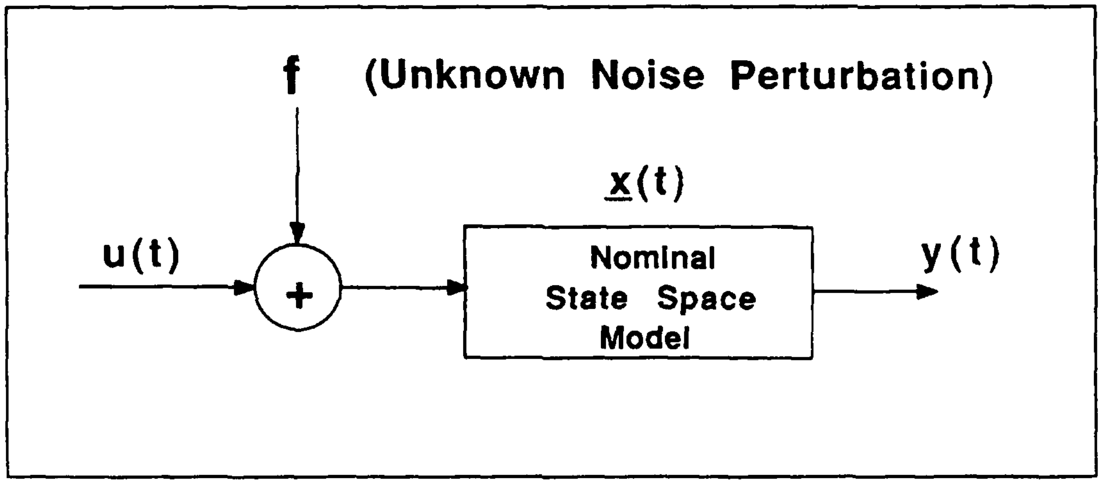


Figure 7. Typical Sliding Mode Scenario

When the state vector is replaced by its estimate, in general a degradation in the overall performance of the system occurs. This is due primarily to the fact that the estimator introduces errors into the loop. As a consequence, a controller which is satisfactory in a state feedback configuration might not perform in a desirable way once an observer is introduced. In this context we introduce the concept of robust observers in order to determine a class of observers for which the degradation of the closed loop system performance is minimal.

1. **Robust Observer Derivation based on the Doyle-Stein Condition**

It is a standard result in system theory that in ideal situations the observer dynamics do not affect the dynamics of the closed-loop system. This can be seen easily by considering the equations of the closed-loop system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \\ u &= -L\hat{x} + r \\ \frac{d\hat{x}}{dt} &= A\hat{x} + Bu + K(y - C\hat{x}) \end{aligned} \tag{2-21}$$

and deriving the transfer function (in the SISO case) Y(s)/R(s). However, this does not mean that the observer does not have any impact in the dynamics of the overall system. In particular, we can see that in the case of a disturbance entering the input, for example, as

$$\begin{aligned} \dot{x} &= Ax + b(u + f) \\ y &= Cx \\ u &= -L\hat{x} + r \\ \frac{d\hat{x}}{dt} &= A\hat{x} + Bu + K(y-C\hat{x}) \end{aligned} \tag{2-22}$$

with f the disturbance term. The transfer function relating f to y is affected by the observer, and it could cause either instability of the system or poor characteristics in terms of phase and gain margins.

However, when we know the precise point of entry of the disturbance f as in Equation (2-22), we can determine the observer gain K so to minimize its effect on the system performance. It has been shown in Ref. 10 that in the limiting case of K such that

$$K[I + C(sI - A)^{-1}K]^{-1} = B[C(sI - A)^{-1}B]^{-1} \qquad (2\text{-}23)$$

the observer dynamics do not appear in the transfer function between f and y. As a consequence, the system (2-22) and its state feedback equivalent

$$\begin{aligned} \dot{x} &= Ax + B(u + f) \\ y &= Cx \\ u &= -Lx + r \end{aligned} \qquad (2\text{-}24)$$

behave in the same manner. Condition (2-23) is called the Doyle-Stein condition and the resulting observer is referred to as a robust observer.

It turns out that the Doyle-Stein condition (2-23) can be satisfied only as a limiting case. In other words it has been shown in Ref. 10 that a sequence of observer gains K(Q), with

$$K(Q) = P(Q)c^{T}R_{w}^{-1} \qquad (2\text{-}25)$$

where R_w is an arbitary definite matrix and P(Q) is the solution of the Riccati equation

$$AP(Q) + P(Q)A^{T} + q^{2}BB^{T} - P(Q)c^{T}R_{w}^{-1}cP(Q) = 0 \qquad (2\text{-}26)$$

with q a scalar, positive parameter, is such that

$$\lim_{q \to +\infty} K(Q) = K \qquad (2\text{-}27)$$

with K satisfying the Doyle-Stein condition (2-23).

The intuitive idea behind this agrument is to design an observer "as if" the disturbance f (not necessarily white or even random) were white noise with covariance q^2. This leads to a steady-state Kalman filter design, Equations (2-25) and (2-26), where the degree of robustness of the observer increases as q increases [Ref. 5]. Therefore, we should be able to determine the optimal value of q using the Nyquist criterion and the continuous-time Nyquist plot. This plot should show a point of convergence for the maximum phase and gain margins capable of being achieved by our dive system. This point of convergence should give a value for q such that the dynamics of the observer do not effect the closed-loop characteristics of the system. Figure 8 is the continuous-time Nyquist plot for the dive system. We note that, as q increases in magnitude, the Nyquist plot converges to a optimal value. This value theoretically should give a optimum time response for depth and dive plane action. Figure 9 is a depiction of these time response plots for the value of q = 200 which was the point of convergence on the Nyquist plot. It is evident from Figure 9 that the optimal value for the parameter "q" is not 200 as suggested by the continuous-time Nyquist plot and supported by the current documentation.

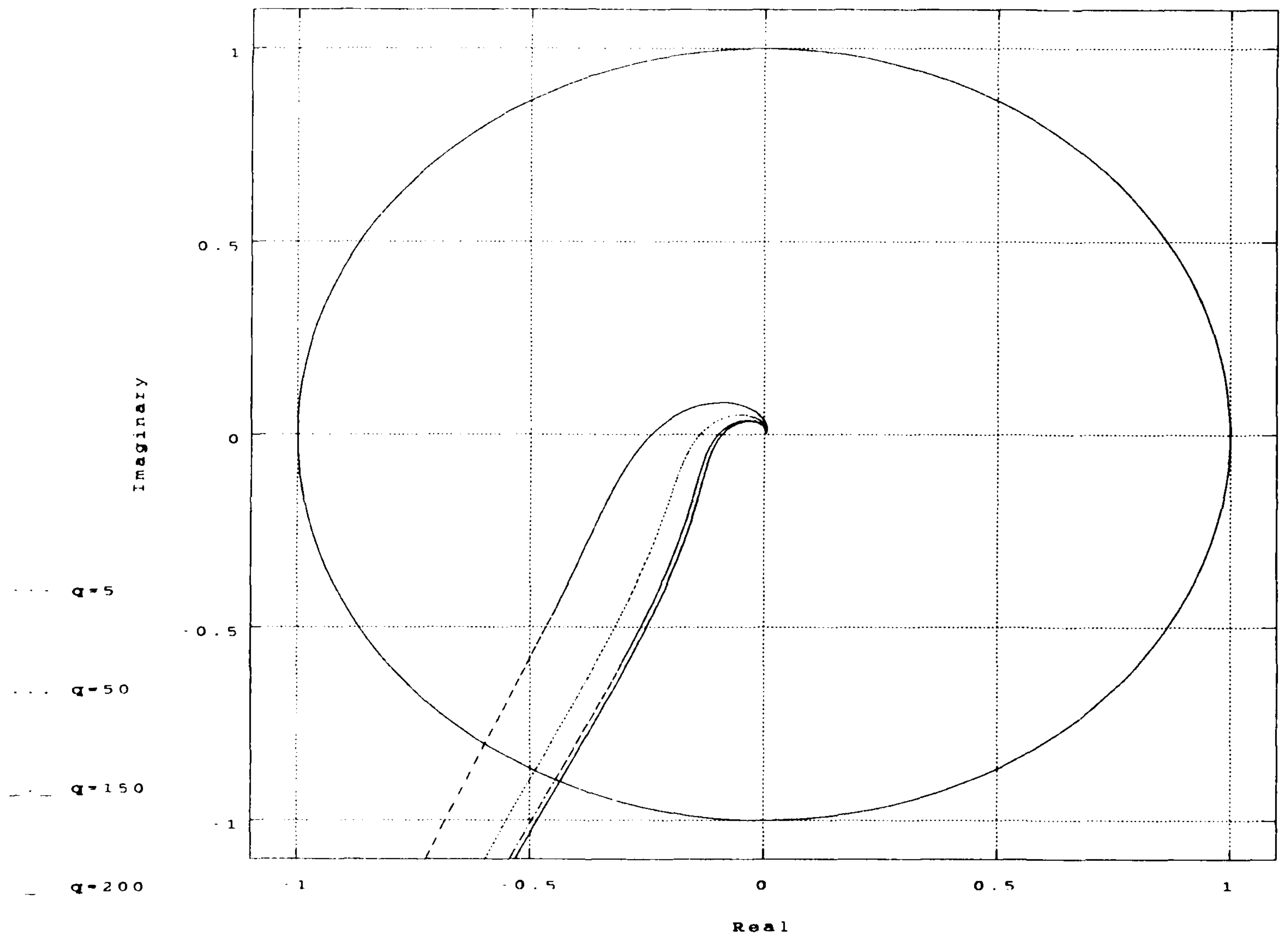


Figure 8. Continuous Time Nyquist Plot of the Dive System

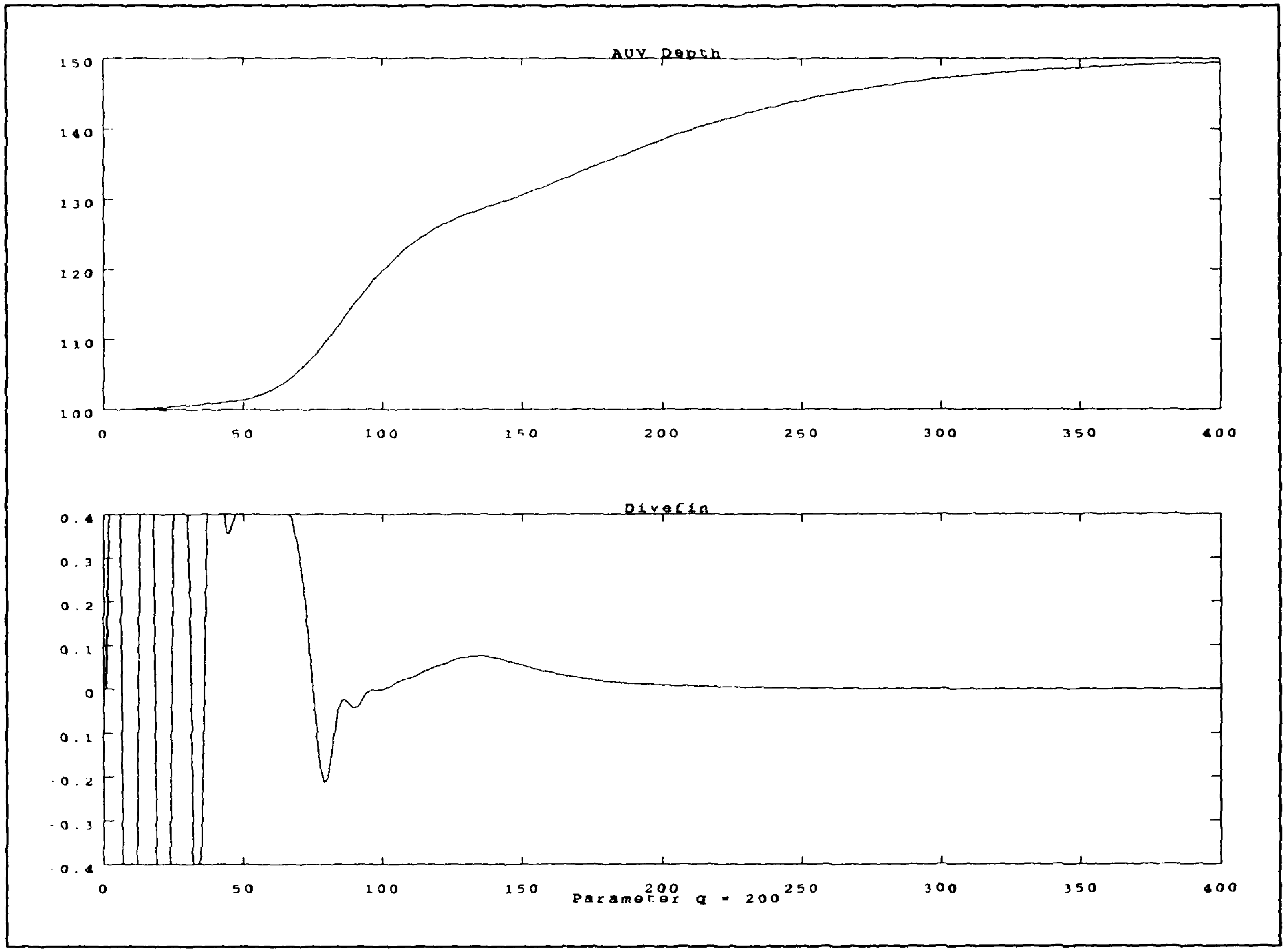


Figure 9. Time Response of Dive System with q = 200 for a 50 ft Dive

The dive plane response plot of Figure 9 suggested that q = 200 was too large in magnitude due to the chattering of the dive command. This led to a hueristic approach to determine the appropriate value for q. The optimal value for the dive system was found to be 5. The problem was how to justify this value mathematically and to prove the existence of a stability window that occurs when discrete controllers are used.

a. Graphical Evidence of a Stability Window in Discrete-Time

To study the behavior of the closed-loop system for different values of the parameter q, the discrete-time Nyquist plot was used. In discrete-time the Nyquist stability criterion is equivalent to the one in continuous-time, and the closed-loop behavior is inferred from phase and gain margins. Although we have shown in the previous section that the use of a robust observer improves the loop characteristics, in the actual implementation we have to exercise some care in the choice of the parameter q. This is due to the fact that we have assumed that the disturbance *f* in Equation (2-22) enters from the input only. If other sources of disturbances are present (i.e., entering at the output), they can introduce instabilities if the value of the parameter q in Equation (2-26) is too large. In order to verify this, let us break the loop at the input u and consider the transfer function $u_2 \rightarrow u_1$ which is shown in the block diagram of Figure 10. This transfer function is derived in the following manner:

$$\dot{x} = Ax + Bu_2 \tag{2-28}$$

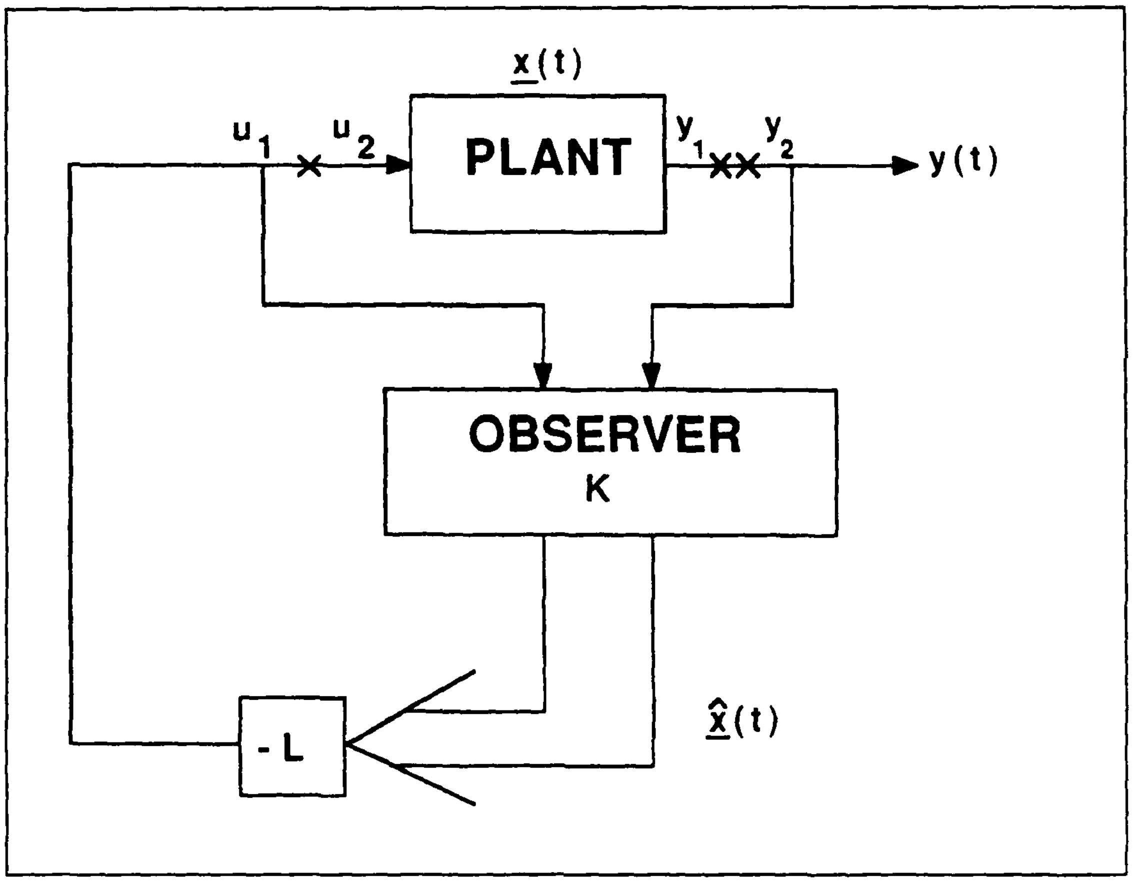


Figure 10. Block Diagram Showing System Broken at Input "X" and Output "XX"

$$y = Cx \tag{2-29}$$

$$\frac{d\hat{x}}{dt} = A\hat{x} + Bu_1 + K(y - C\hat{x}) \tag{2-30}$$

$$u_1 = -L\hat{x} \tag{2-31}$$

By substituting Equations (2-29) and (2-31) into Equation (2-30), we obtain

$$\frac{d\hat{x}}{dt} = A\hat{x} - BL\hat{x} + KCx - KC\hat{x} \tag{2-32}$$

Then collecting terms

$$\frac{d\hat{x}}{dt} = (A - KC - LB)\hat{x} + KCx \tag{2-33}$$

Now if we let

$$x_e = \begin{bmatrix} x \\ \hat{x} \end{bmatrix} = A_e x_e + B_e u_2 \tag{2-34}$$

which resolves to

$$\dot{x}_e = \begin{bmatrix} A & \underline{0} \\ KC & (A-KC-LB) \end{bmatrix} x_e + \begin{bmatrix} B \\ \underline{0} \end{bmatrix} u_2 \qquad (2\text{-}35)$$

$$u_1 = [\underline{0} \;\; L] x_e \;, \qquad (2\text{-}36)$$

the transfer function of $u_2 \rightarrow u_1$ in state-space form. We can compute the frequency response of the loop in Figure 10 open at the point marked with "X". Using this transfer function, a discrete Nyquist plot was obtained for various values of q: 5, 50, 150, and 200. This plot is shown in Figure 11. From a comparison of this plot and that of Figure 8, we see that the discrete time transformation effects caused no noticeable deviations in the frequency response for all values of the parameter q.

As mentioned above, the only perturbation on the system taken into account so far is *f* which enters at the input as depicted in Figure 7, but in fact the system is also corrupted by noise at the output. If we want to see the effect of a disturbance at the output as q changes, we have to open the loop at the point marked "XX" in Figure 10 and study this closed-loop transfer function $y_2 \rightarrow y_1$. It can be shown that the state-space representation of the dynamics $y_2 \rightarrow y_1$ is given by

$$\begin{aligned} \dot{x}_e &= \begin{bmatrix} A & -LB \\ \underline{0} & A-KC-LB \end{bmatrix} x_e + \begin{bmatrix} K \\ \underline{0} \end{bmatrix} y_2 \\ y_1 &= [C \;\; O] x_e \end{aligned} \qquad (2\text{-}37)$$

The Nyquist plot of the corresponding frequency response depicted in Figure 12 shows some of the problems encountered for large values of q. We note that, while the phase margin and gain margin increase in Figure 11, a corresponding decrease is occurring in the phase margin of Figure 12 for an associated increase in the parameter q. Therefore, the noise present at the output causes a reverse effect on the system for increasing values of q. The effect of this is chattering in the dive plane response which occurs for large values of q. As the gain and phase margins increase at the input, the phase margin is diminishing at the output of the system.

We can see from the Nyquist plots in Figure 12 that as q increases the gain margin decreases. In particular, from the phase margin, we can see that the closed-loop response of the system seen from a disturbance entering at the output becomes more and more under-damped for larger values of q. Experimental results show that a value of $q = 5$ *yields a reasonable closed-loop response. This can be seen by comparing the step-*response plots of the robust observer depicted in Figure 13 for the value of $q = 5$ with those of $q = 200$ in Figure 9.

The continuous-time behavior of the Doyle-Stein condition appears to not account for the noise at the output in a similar fashion to that of the discrete-time. However, further investigation of this topic is warranted. Graphical evidence does support the adverse effect of a nonlinear perturbation present at the output in discrete-time control for submersible vehicles such as the Naval Postgraduate School Autonomous Underwater Vehicle.

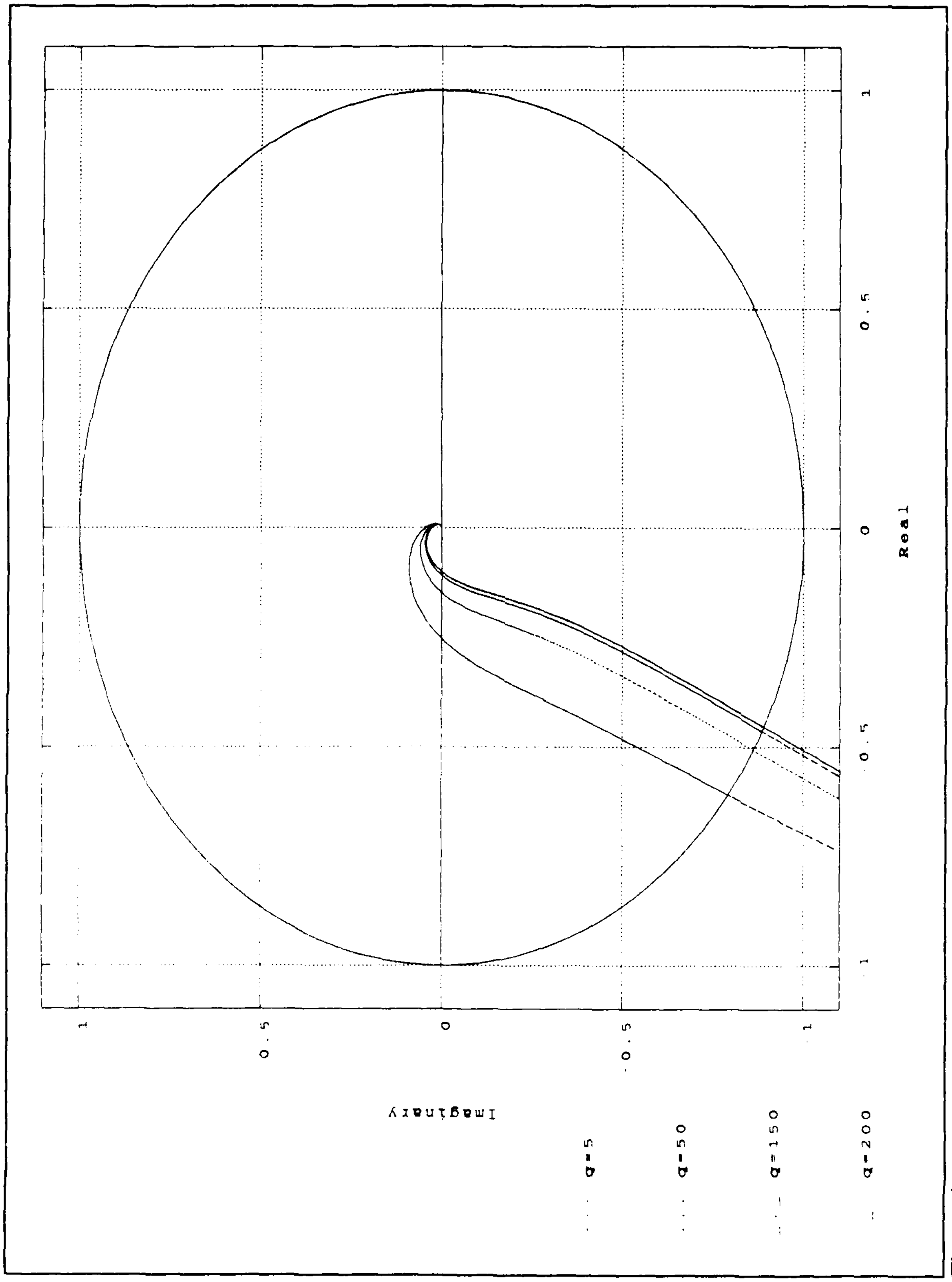


Figure 11. Discrete Nyquist Plot for Dive System Closed Loop Broken at Input

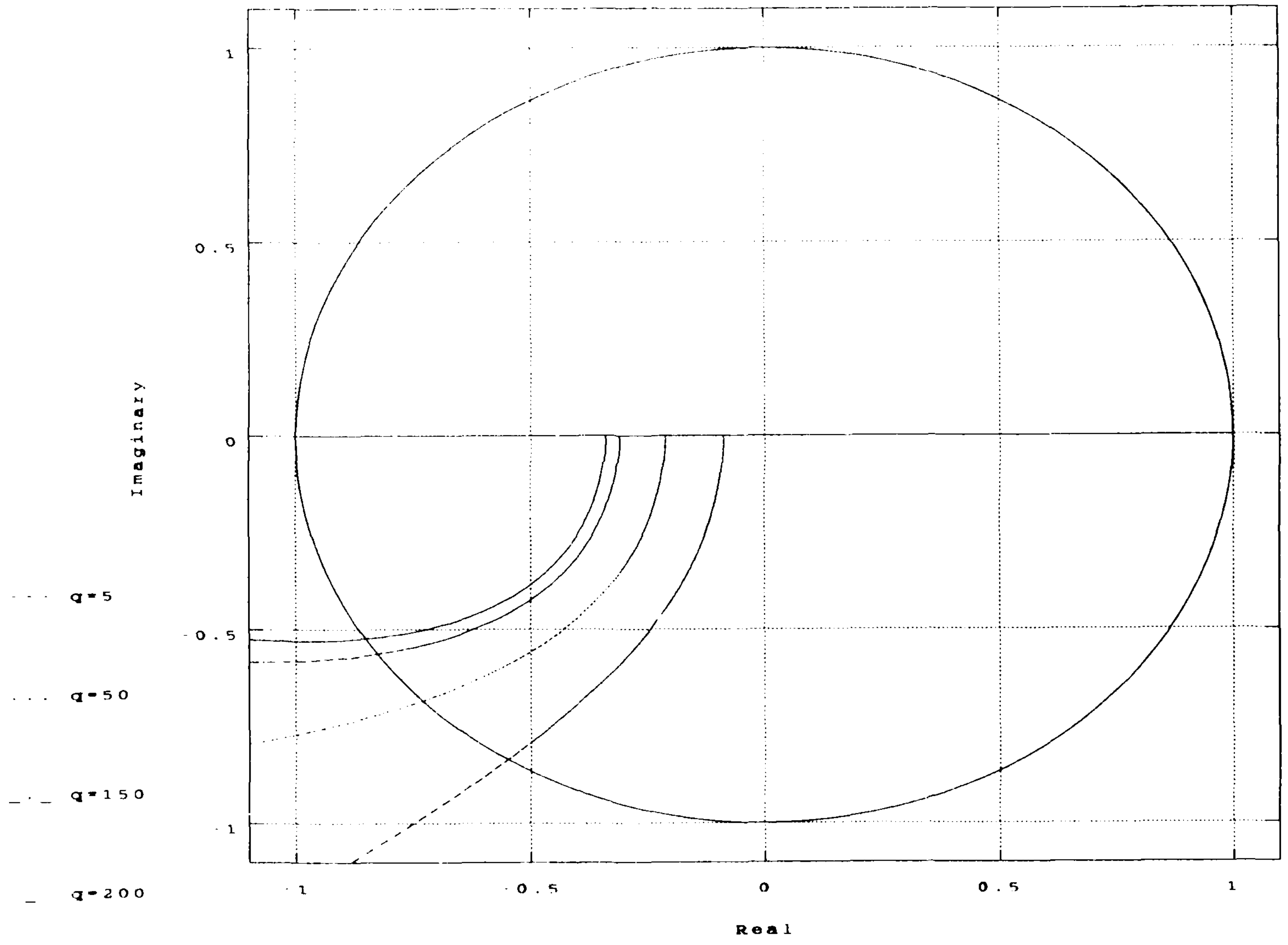


Figure 12. Discrete Nyquist Plot Closed Loop Broken at the Output

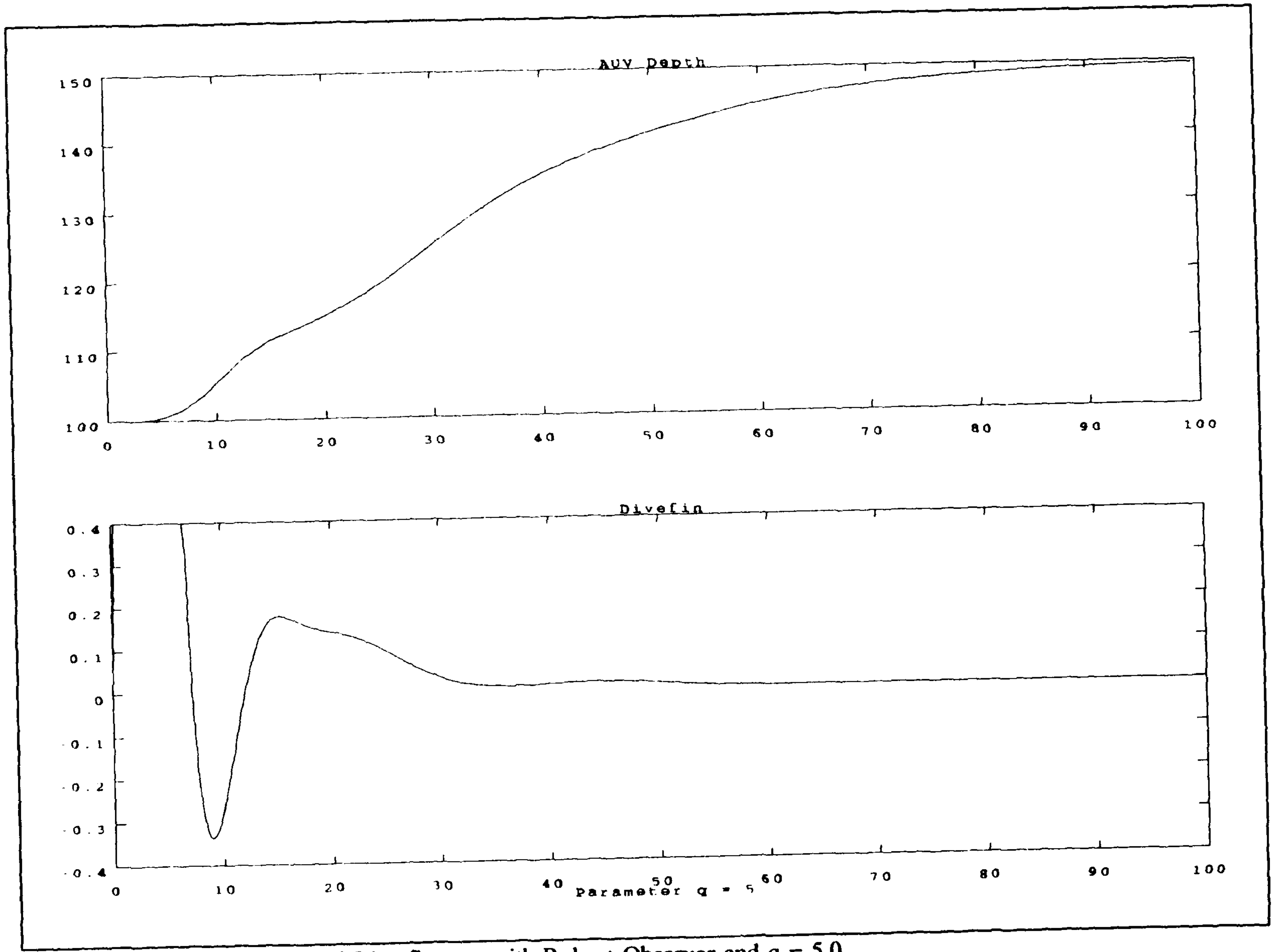


Figure 13. Time-Response of Dive System with Robust Observer and q = 5.0

E. SENSOR ERROR DETECTION AND HANDLING IN SOFTWARE

The purpose of the robust observer for the AUV, which is depicted in block diagram form in Figure 14, is to provide redundancy in systems through software vice hardware. The observer developed in the previous section provides a certain degree of redundancy in the control of the AUV. In fact, for every signal in the loop (pitch, pitch rate, and depth in our case), both the measured and estimated signals are available.

Based on this we can determine an error detection algorithm which alerts the system in case of sensor failure and replaces the faulty values by the estimates. This offers an advantage in terms of reliability, and it allows for a stable control even in the presence of one or more sensor failures. The principle of this error detection scheme is simple. Through the use of a fast computer algorithm, it checks error thresholds between measured state and observer estimated state. Then the algorithm selects the most accurate state representation and uses it for control. By this we mean that, if an error was detected in the sensor output, the estimated state would be used instead of the faulty measured state.

1. Development of the Error Detection Algorithm

The error between actual state, x, and estimated state, $\hat{x}$, is defined as

$$\tilde{x} = x - \hat{x} \qquad (2\text{-}38)$$

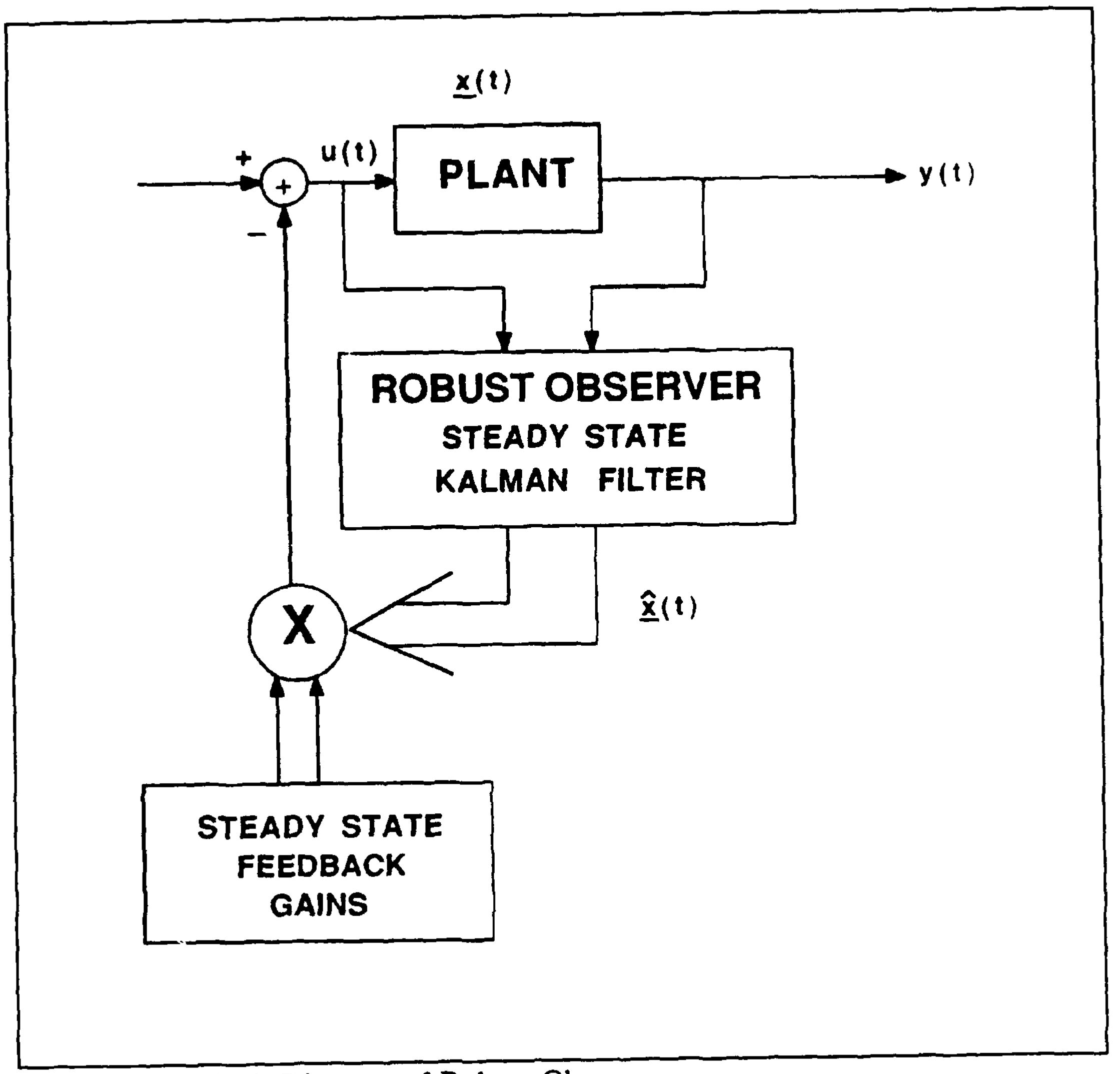


Figure 14. Block Diagram of Robust Observer

The measured state is obtained in the AUV prototype directly from the sensors (i.e., gyros, sonars, and servo-controllers). Each sensor sends the information to a signal conditioning network and then to a designated area in memory where the autopilot samples the data. In order for the error detcction concept to be realized, the typical error signal, $\tilde{x}$, should be as small as possible during normal operations, at least asymptotically; however, transient responses due to the observer dynamics introduce errors even in ideal conditions. A graphical representaion of the pitch rate and pitch error signals in Figure 15 shows that $\tilde{x}$ in the transient phase is far from zero. These transients are present due to the initial conditions placed on the observer. In this section we determine an error criterion which is not affected by the transient response of the observer. Our arguments are an application of the Cayley-Hamilton theorem.

In the ideal case the error between actual and estimated state satisfies the state space equation

$$\tilde{x}(k+1) = \Phi_0 \tilde{x}(k) \tag{2-39}$$

with Φ_0 the discrete transition matrix of the observer, of the form

$$\Phi_0 = \Phi - KC \tag{2-40}$$

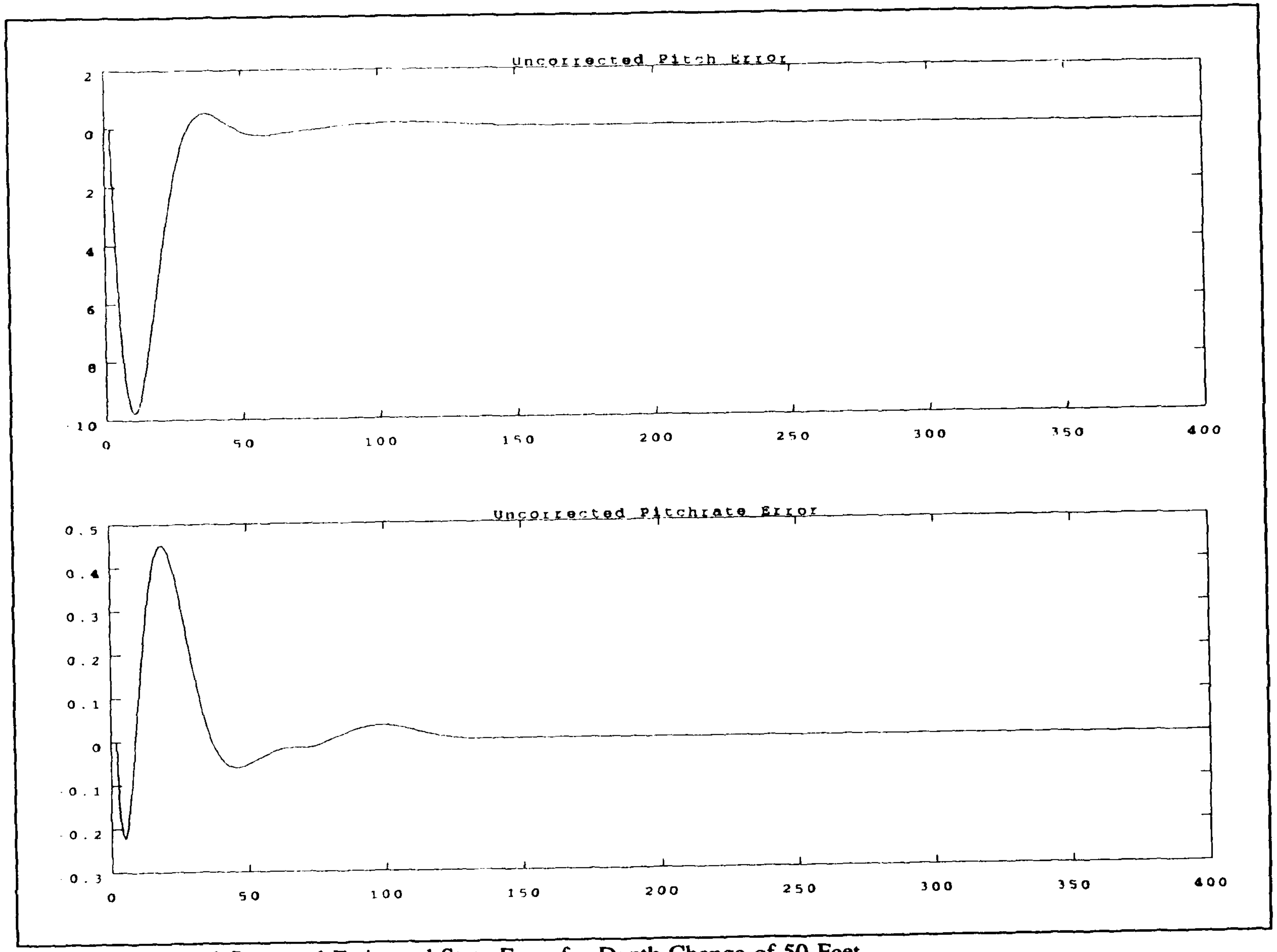


Figure 15. Actual State and Estimated State Error for Depth Change of 50 Feet

In our particular case, the state error $\tilde{x}$ is given by

$$\tilde{x} = \begin{bmatrix} q \\ \theta \\ \frac{z}{v} \end{bmatrix} - \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \end{bmatrix} \tag{2-41}$$

where v is vehicle forward speed. Since the observer gain K is chosen to guarantee exponential stability of Equation (2-39), we can see that $\tilde{x}(k)$ has an effect only during the transient response. In order to determine an error criterion which is not affected by the transient response and is zero (or very small) during normal operations, consider the Cayley-Hamilton theorem applied to the matrix Φ_0,

$$\Phi_0^{\,n} + a_1 \Phi_0^{\,n-1} + \dots + a_n I = 0 \tag{2-42}$$

where

$$A(\lambda) = \lambda^n + a_1 \lambda^{n-1} + a_2 \lambda^{n-2} + \dots + a_n = \det(\lambda I - \Phi_0) \tag{2-43}$$

is the characteristic polynomial of Φ_0. Application of Equation (2-42) to Equation (2-39) yields, at least under ideal conditions,

$$e(k) = \tilde{x}(k) + a_1 \tilde{x}(k-1) + \dots + a_n \tilde{x}(k-n) = 0 \tag{2-44}$$

for all k, since

$$\tilde{x}(k) + a_1\tilde{x}(k-1) + \cdots + a_n\tilde{x}(k-n) = \\ (\Phi_0{}^n \; a_1\Phi_0{}^{n-1} + \cdots + a_n I)\tilde{x}(k-n) \tag{2-45}$$

which is zero by the Cayley-Hamilton theorem.

2. Testing of the Error Detection Algorithm

The error detection algorithm was tested by simulating a sensor failure. This was done in the code auvobs.m found in Appendix A. To simulate a sensor failure, the sensor in question is driven to zero to indicate a broken connection and to 100,000 to indicate the program accessing a null pointer region. Both of these error conditions reflect possible error sources which can actually occur. To notify the program developer and user that the system had detected an error, "C" language standard output *printf* statements were used. The actual real-time code documents error conditions differently by setting flags in a file that contains the vehicle run data. Also a faulty sensor or broken connection will probably not result in a polled value of 0.0, but for simulation purposes this value was used to show the sensitivity of the error detection algorithm.

The algorithm takes the corrected error vector and compares the values with the threshold levels previously mentioned. If the corrected errors are within the tolerance setup by the threshold values, then the actual state is used, else the observed state is used and an error is assumed to have occurred. Several runs depicting various error conditions

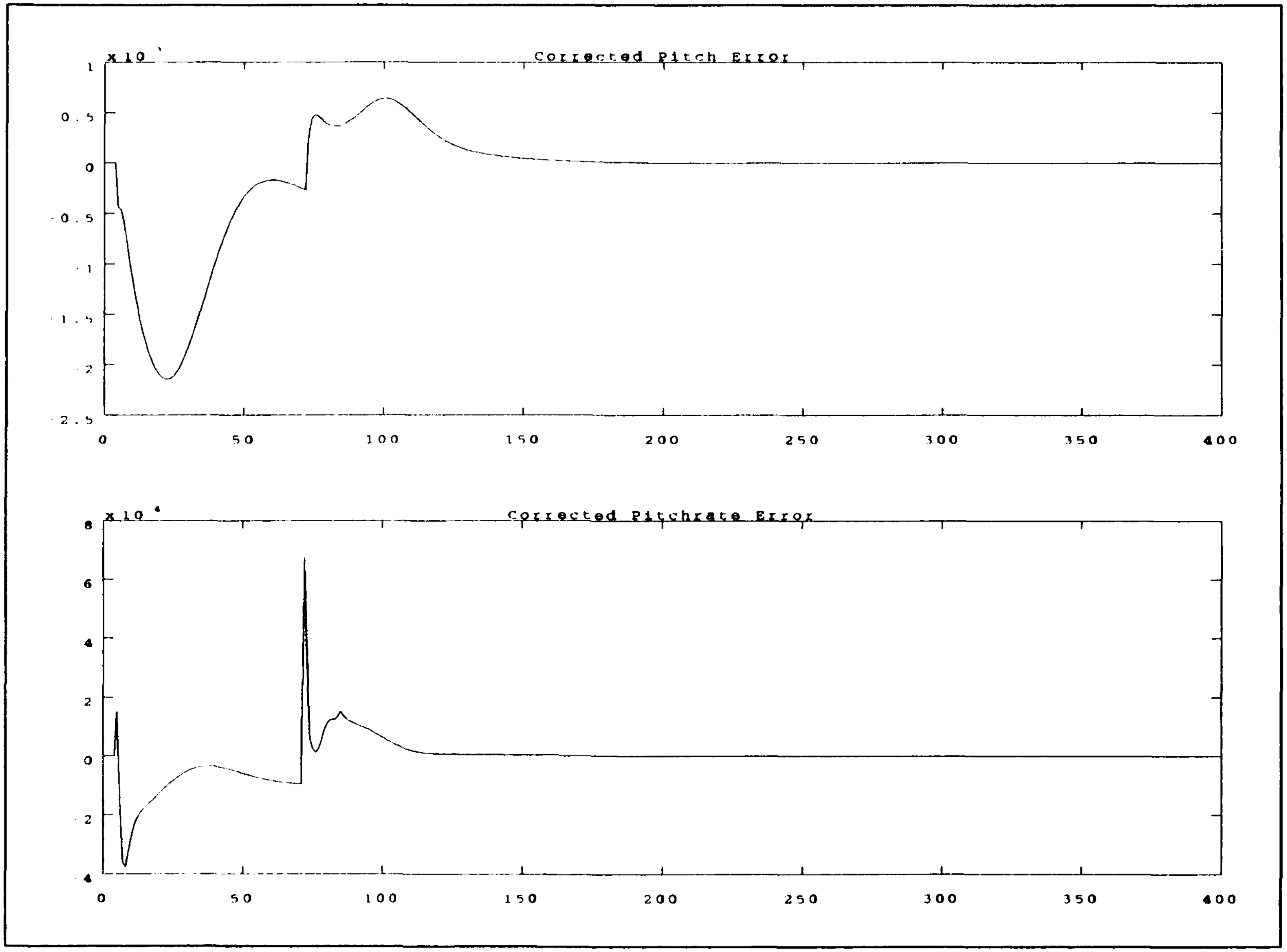


Figure 16. Actual and Observed State Errors for a Depth Change of 50 Feet

are shown in Figures 17, 18, and 19. These figures only show the magnitudes present under these error conditions. In Figure 17 theerror simulated is a broken connection to the pitch rate gyro. When the plots of Figure 17 are compared to those of Figure 16, we note the large magnitude of error present at the sensor failure point. This error diminishes rapidly, and, even though a sensor error is still present, the errors are below the threshold. Therefore, the autopilot does not need to discriminate between the observer state and the actual state.

This error diminishes rapidily because the controller drives the error to zero in steady state and thus the effects of this type of error (i.e., a returned value of 0.0 from the failed sensor) are negligible. However, in Figure 18, a null pointer assignment error is simulated. This type of error occurs when a program, such as the sensor polling program for the AUV, accesses a region of memory which is marked for null assignments. When this occurs, the value returned can take on a spurious assignment such as the one modeled in the simulation. From these plots we note that, regardless of the magnitude of the error, the other sensors are not effected. This is due to the fact that the sensors are independent of each other in hardware and software. The final set of plots shown in Figure 19 demonstrate a pitch sensor failure. This plot along with the others shows that, when a sensor failure occurs, the resultant error exceeds the threshold level and causes the observer state to be used. Once again the failure value of 0.0 was used to demonstrate the sensitivity of the error detection method. While the pitch sensor failed, we note again that the pitchrate sensor is not affected, and the error does not exceed the threshold level.

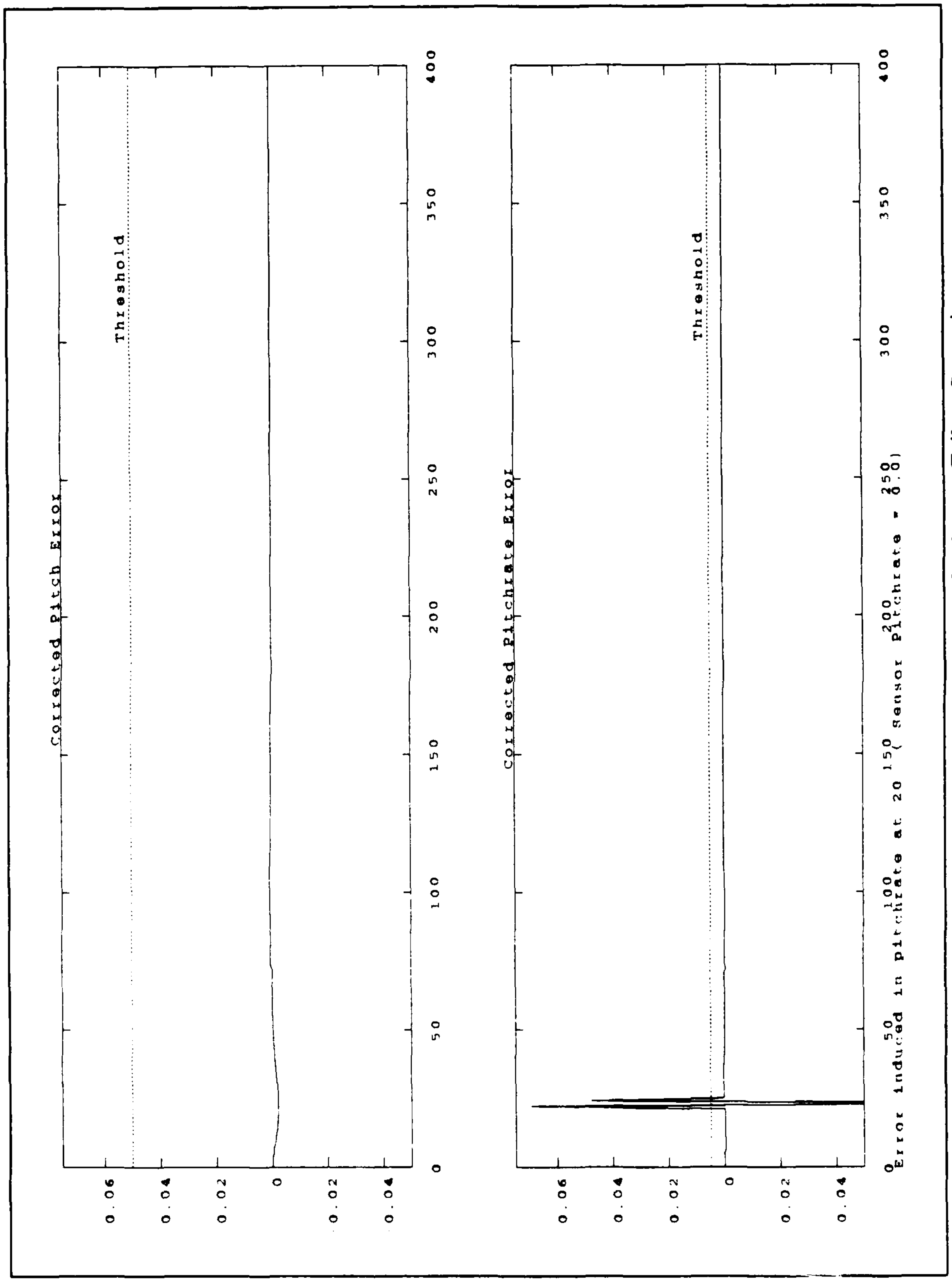


Figure 17. Plots of Pitch and Pitch Rate with Threshold Levels for Sensor Failure Detection

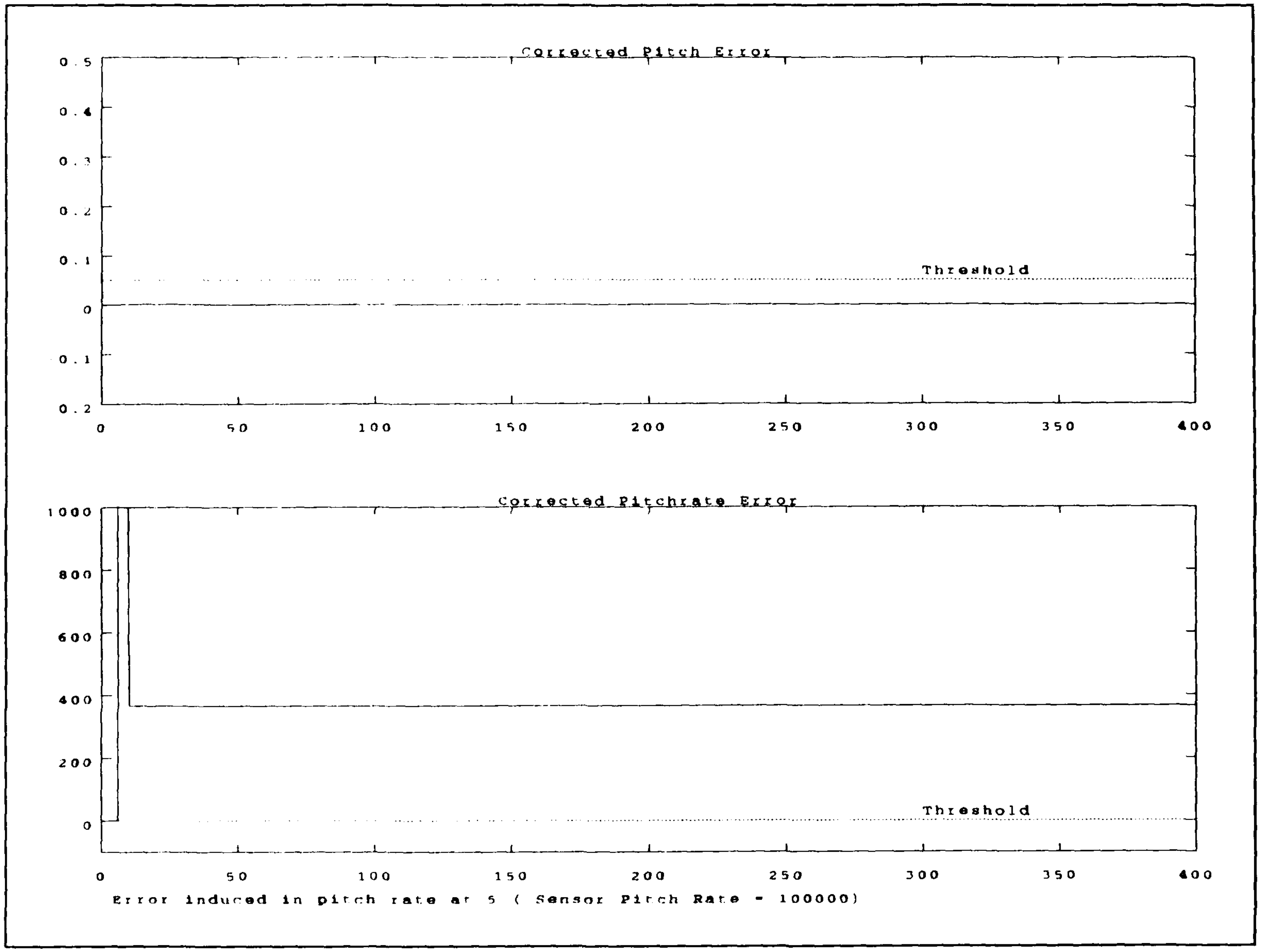


Figure 18. Plots of Pitch and Pitch Rate with Threshold Levels for Sensor Failure Detection

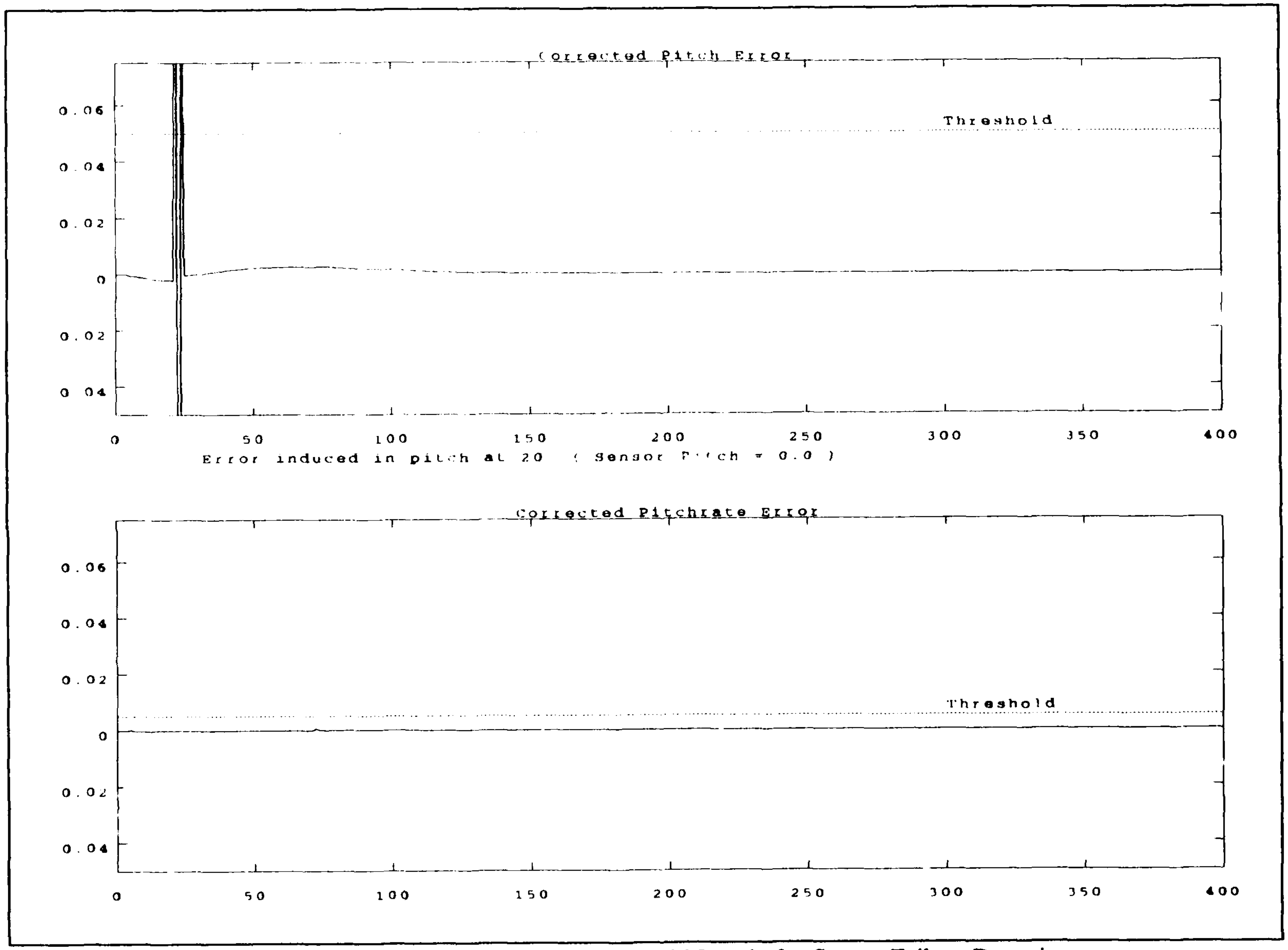


Figure 19. Plots of Pitch and Pitch Rate Errors with Threshold Levels for Sensor Failure Detection

3. Fatal Sensor Errors

Most sensors can be modeled with sufficient accuracy to be used in place of the actual state, but some errors, if detected, are considered major and cause mission termination upon detection. Three such errors are depth sensor error, directional gyro error, and erroneous input to the autopilot from the mission planner. All of these conditions are closely monitored, and if any one of these conditions is detected the AUV programmed run is terminated. The AUV is forced to surface and slows to the *minimum* required speed necessary for the vehicle to remain on the surface. The rudder is also set to zero deflection when a fatal error occurs.

Depth sensor gyro failures are detected in the same manner as those of the pitch and pitchrate gyros. A threshold level is compared to the corrected error returned by the difference of the estimated and actual states. This concept is depicted in block diagram form in Figure 20. This diagram demonstrates how all of the components are linked in the algorithm.

The error detection method used for sensing mission planner errors and directional gyro failure is different from the previous scheme. This algorithm does a check of the incoming values and compares these values to 2π. If the incoming value is $> 2\pi$, then a sensor error has occurred. The vehicle is forced to surface and the rudder is set to 0 deflection. This terminates the AUV mission and protects the prototype vehicle from being damaged due to sensor failure or incorrect desired-headings generated by the mission planner.

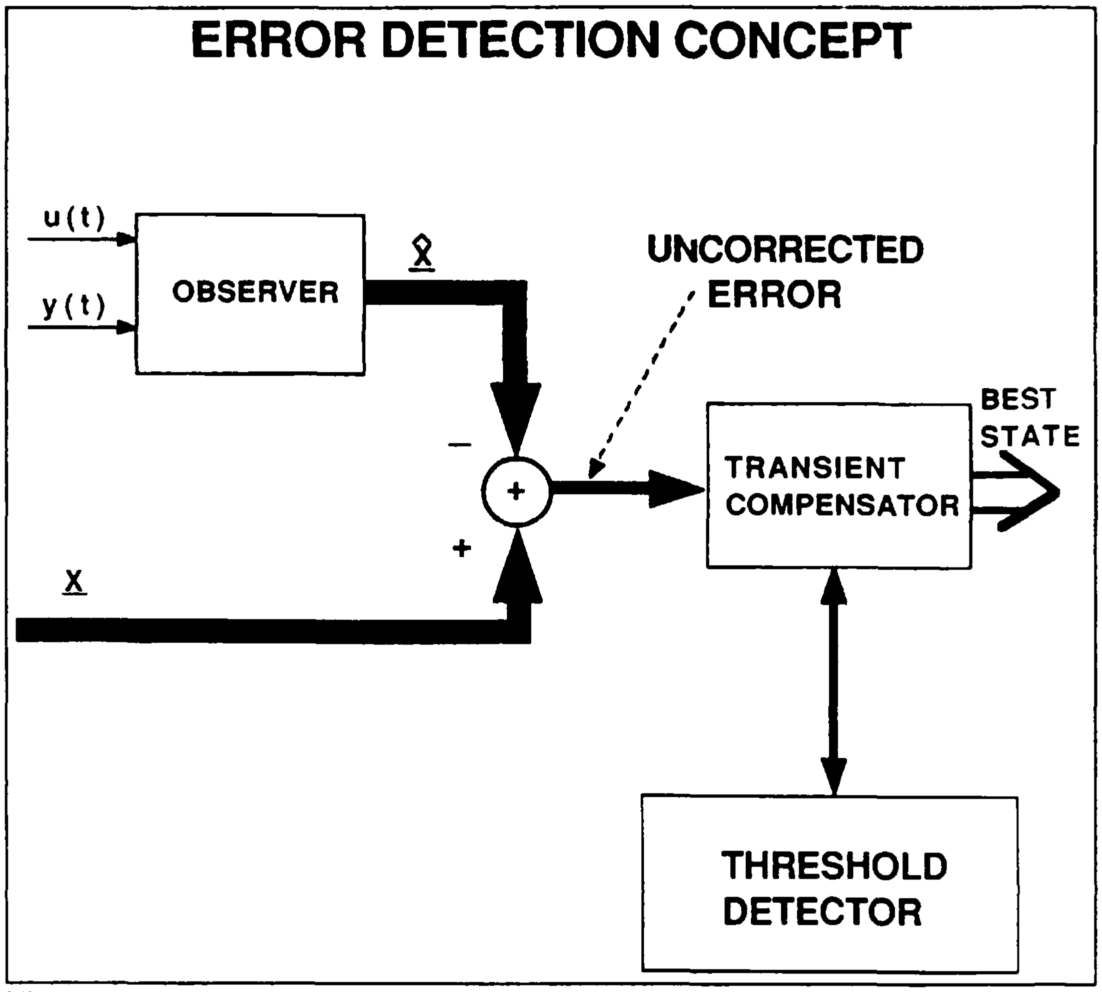


Figure 20. Block Diagram of the Error Detection Scheme

F. SIMULATION OF REAL-TIME AUTOPILOT

With the development of the robust observer and the error detection algorithm, the autopilot routine is complete and ready for simulation and implementation. The final check of the proposed autopilot was conducted simulating the "C" source code using the programs and interface module discussed in Chapter 3. The Matlab simulation code, auvobs.m, is located in Appendix A while the "C" source code, autopilot.c, is located in Appendix B.

During this final evaluation period, a multiple dive simulation was run. This simulation detected a fault in the error detection algorithm. This fault occurred because the error vectors must go through two iterations of the control routine after a new ordered depth is received. This is due to the fact that the error vectors operate on previous states as shown in Equation (2-24). To correct this fault, a flag is used which sets the variable "transient_flag" to a value such that the error vectors are allowed to update prior to being operated on by the coefficents of the characteristic equation of the A matrix.

In the C code version of the autopilot, the current ordered depth is compared to the previous ordered depth. If the values are identical then error checking continues, but, when these values are different, a flag "transient_hold" is set using a discrete counter which increments each time the autopilot routine is called. This allows the error detection algorithm to operate in a discrete mode as intended. The transient flag can only be re-initialized when a change in ordered depth has been sent to the controller.

A multiple depth change simulation is depicted in Figures 21-24. This simulation was based on the following initial conditions and simulated errors:

- Initial Depth: 100 Feet.
- Initial Velocity: 4 Knots.
- Initial Propeller Revolution Rate: 500 RPM.
- Parameter q set to 5.
- First Ordered Depth: 140 Feet.
- Second Ordered Depth: 120 Feet (Begins at Frame 226).
- Duration of Simulation Run: 400 Frames.
- Simulated Error in Pitch Rate Gyro (Pitch Rate Sensor = 100000).
- Simulated Failure of Pitch Gyro (Pitch Sensor = -35).
- Pitch Rate Threshold Error set to 0.005.
- Pitch Threshold Error set to 0.05.
- Normalized Depth Error set to 0.2.

The simulation has two induced sensor errors. The first error occurs at time-frame 5 when the value of 100000 is returned during polling of the pitch rate gyro. The second error occurs at time frame 50 when the value of -35 is returned during sampling of the pitch gyro. The pitch rate gyro error is an example of a null pointer assignment while the pitch gyro error is a general sensor failure. As noted before, these are both plausible values for their associated errors. In Figure 21 plots of the corrected pitch and pitch rate errors with their associated thresholds are shown. From these plots we note the beginning of the sensor failure by the jump in the error. Both errors cause the direct sampled states to not be used and instead the estimated states returned by the observer are used. The noticeable drop in magnitude of these two errors that occurs at frame 226 is due to the change in ordered depth which calls for resetting the error vectors. The hiatus required to reset the error vectors causes the magnitude of both errors to momentarily fall, but, when the error detection algorithm is reset, the errors rise and again cause the observed state to be used vice the actual measured state in the controller. The time in seconds of this hiatus is approximately 0.1 seconds, which even for a slow sampling system such as the AUV, is negligible. The next set of plots shown in Figure 22 are of the normalized-depth error and actual pitch rate of the vehicle. The plot of normalized-depth error demonstrates that the errors induced in the pitch and pitch rate gyros have no effect on it. From previous simulations this is what we expect. The depth error does not exceed the threshold, and, therefore, a fatal sensor error is not detected. The plot of the pitch rate of the vehicle is a good indicator of the system stability. The vehicle pitch rate curve should be symmetric about the mean which is 0 in steady state. This indicates a stable platform and is, in fact, the type of curve shown in Figure 22. Figure 23 depicts plots of the actual depth with the ordered depth overlain and the stern plane or dive fin response. This set of plots shows that the controller, even with two sensor failures, is capable of driving the vehicle to the desired depth with no overshoot and a very smooth motion. The dive plane response is nearly optimal and also does not show any influence of the sensor errors on its control. This demonstrates the robust nature of the controller-observer combination. The last plot contained in Figure 24 is the final verification of system stability. This plot is of the sliding plane. As mentioned earlier in this chapter and depicted in Figure 1, the controller should drive the state trajectories to the equilibrium point along this switching plane. This is the case shown Figure 24. From the sliding plane, we can deduce that two errors occurred during the run. The change in ordered depth also caused the states to be offset at time-frame 226. The final result of the sliding plane was the states were in fact driven to equilibrium. This is the last necessary component needed to prove stability in the variable structure controller design.

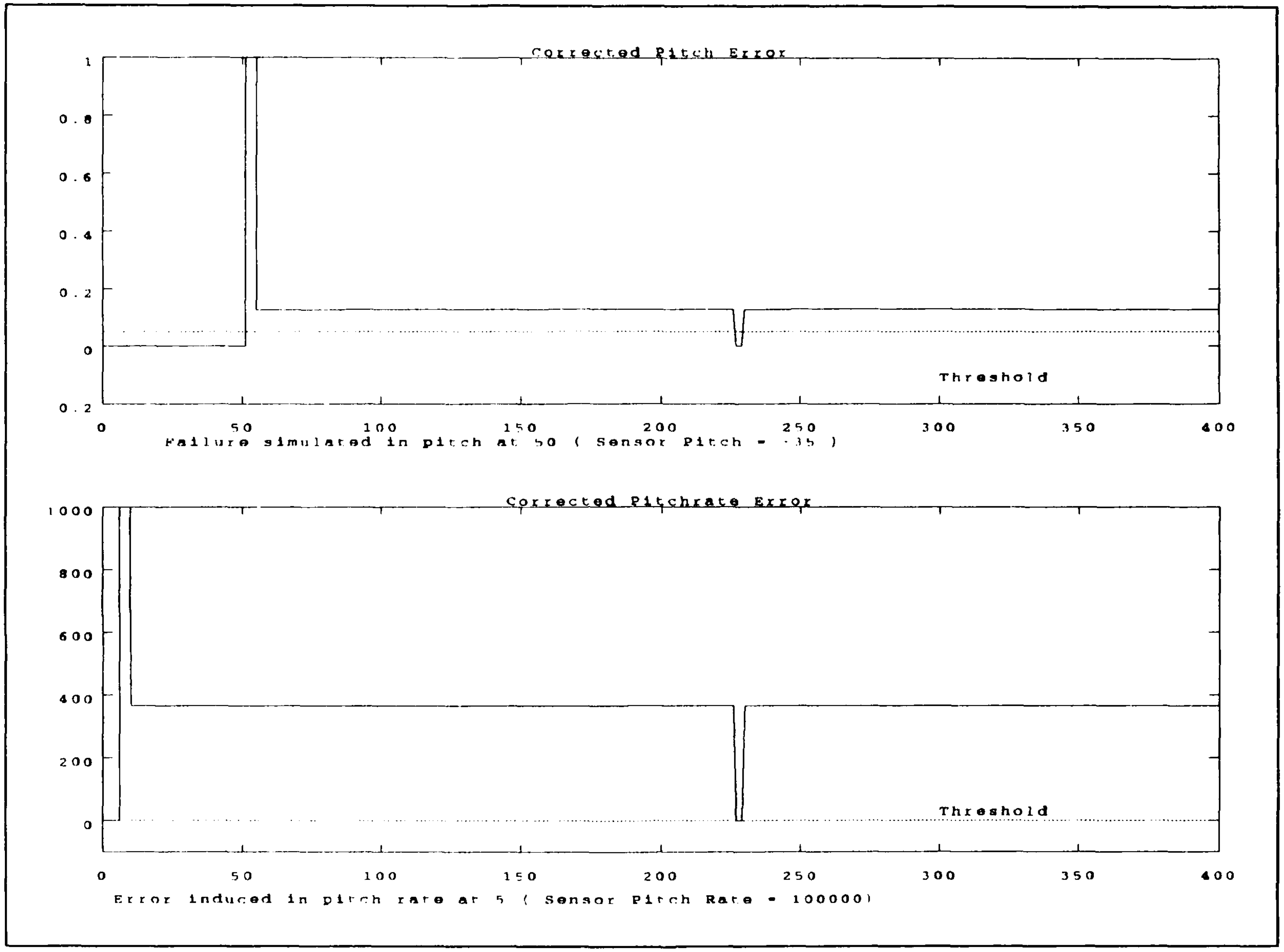


Figure 21. Plots of Corrected Pitch and Pitch Rate Errors

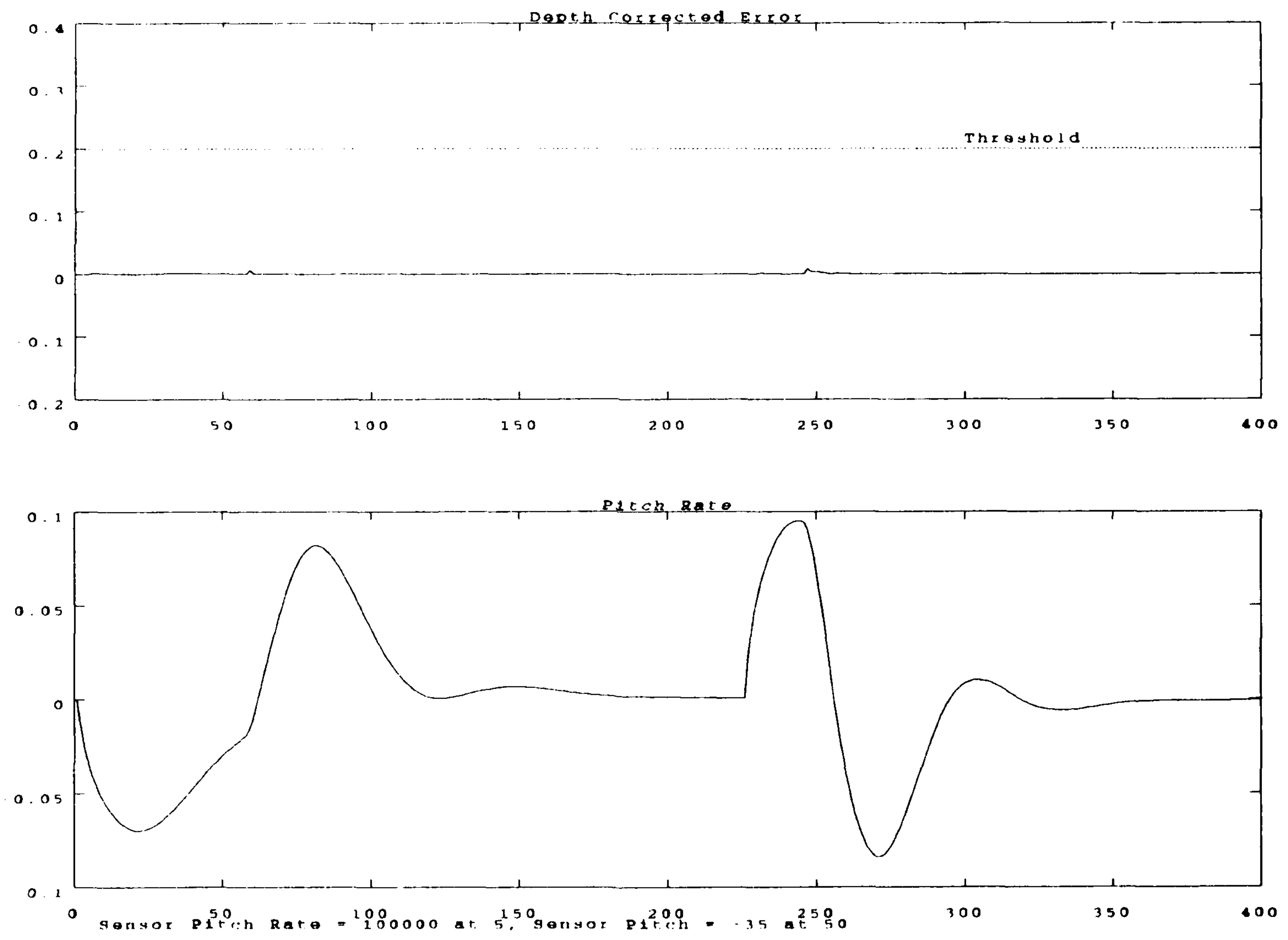


Figure 22. Plots Of Normalized-Depth Error and Actual Vehicle Pitch Rate

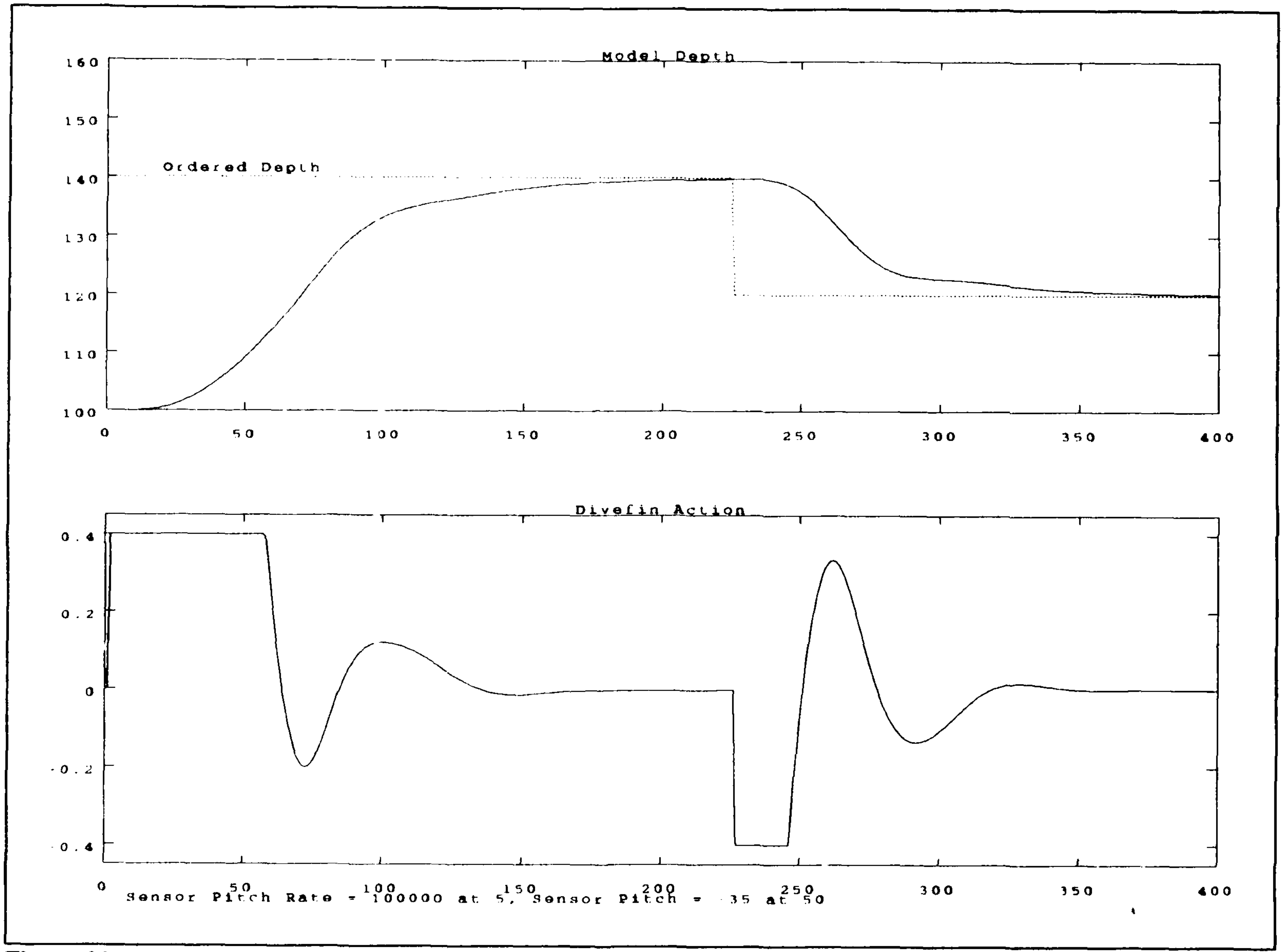


Figure 23. Plots of Actual Vehicle Depth and Dive Fin Response

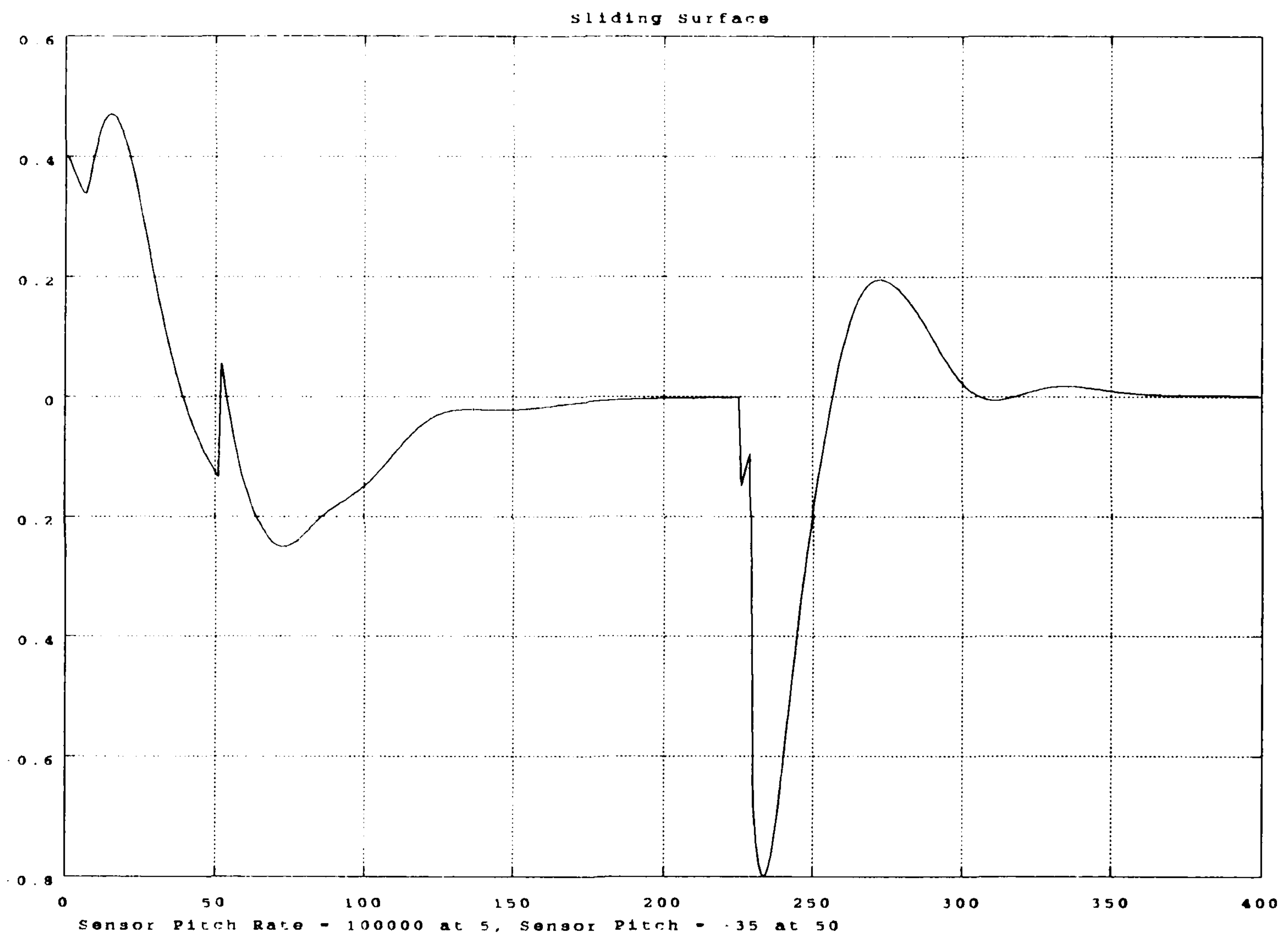


Figure 24. Plot of the Sliding Plane

III. CONSIDERATIONS FOR REAL-TIME-IMPLEMENTATION OF THE AUV CONTROLLER

A. DEVELOPMENT OF A PC-BASED AUV MODEL

To facilitate development and testing of the various programs to be implemented on the prototype-vehicle, a computer model of the vehicle dynamics has been developed. This model presents an interface that is similar in nature to the actual hardware implementation. By this we mean that the sampling of the actual measured states on the prototype by the control programs should be similar to the methods used in simulation. A model for the purposes of simulation was previously developed by Schwartz [Ref. 3] which described the dynamics of a 17.4 feet long vehicle, weighing 12000 pounds, and having neutral buoyancy. This previous work was taken as the basis of the real-time model developed here.

The model has six degrees of freedom: three referring to position, the Euler angles, and three to rotation. Figure 25 shows the right-hand coordinate system of the vehicle which the control code is based on, as well as an artist's rendition of the AUV. To define the coordinate system setup by these angles, the vector of orientation and rate of change had to be taken in to account. In order to describe the right-hand coordinate system for computer computations, the model consists of the 12 states listed in Table 1, where the mathematical symbol, state definition, and computer program variable are annotated.

In Appendix C the program, model.c, and header file, modelprm.h, which contains the parameters of the AUV have been listed. This version of the model features the states in array format which allows the calling routines to access the model by passing a pointer to the first address of the array. This is a significant improvement over the previous version of the model which used all gobal variables, a method that has been generally shunned by programmers. Due to the capabilities of the "C" language, the visibility of functions and variables can be easily modified [Ref. 6]. (A variable with global visibility can be accessed by any function and modified, where as a variable which is locally visible to its parent function only cannot be modified by any other function.)

The actual AUV implementation will use these visiblity rules in a similar manner to that of the integrated test package which uses the model to update the states for simulation purposes. This updated model also corrected several serious coding hazards such as the initialization of an array beginning with an index of 1 vice 0, which led to an extraneous use of zeros for array padding. This critical mistake can cause null pointer assignments which result in run-time errors if a program references the array outside of its bounds. Another notable improvement which lends the model more to real-time-simulation code development was the update method used for the control surfaces, rudder, stemplane, bowplane, and propeller revolution rate. These values are passed in a similar manner to the states (i.e., using pointers).

The model calls one external function, signum.c, which accepts a variable that has been declared as a double, and returns a double which has the value of +1.0 or -1.0 depending on the sign of the passed argument. The function model() passes the following arguments: a pointer to the address of the first value of oldstate, a pointer to the address of the first value of inputs, and the sampling interval. It returns the address of the first value of the updated state array and a integer which can been used to determine if an error has occurred during the state update.

Table 1. STATE DEFINITIONS FOR AUV MODEL

Mathematical Symbol	State Definition	Program Variable	x-y-z-axis Definition
u	surge rate	mstate[0]	x-axis velocity
v	sway rate	mstate[1]	y-axis velocity
w	heave rate	mstate[2]	z-axis velocity
p	roll rate	mstate[3]	rotation rate about x-axis
q	pitch rate	mstate[4]	rotation rate about y-axis
r	yaw rate	mstate[5]	rotation rate about z-axis
x	surge	mstate[6]	x-axis
y	sway	mstate[7]	y-axis
z	heave	mstate[8]	z-axis
ϕ	roll	mstate[9]	rotation about x-axis
θ	pitch	mstate[10]	rotation about y-axis
ψ	yaw	mstate[11]	rotation about z-axis

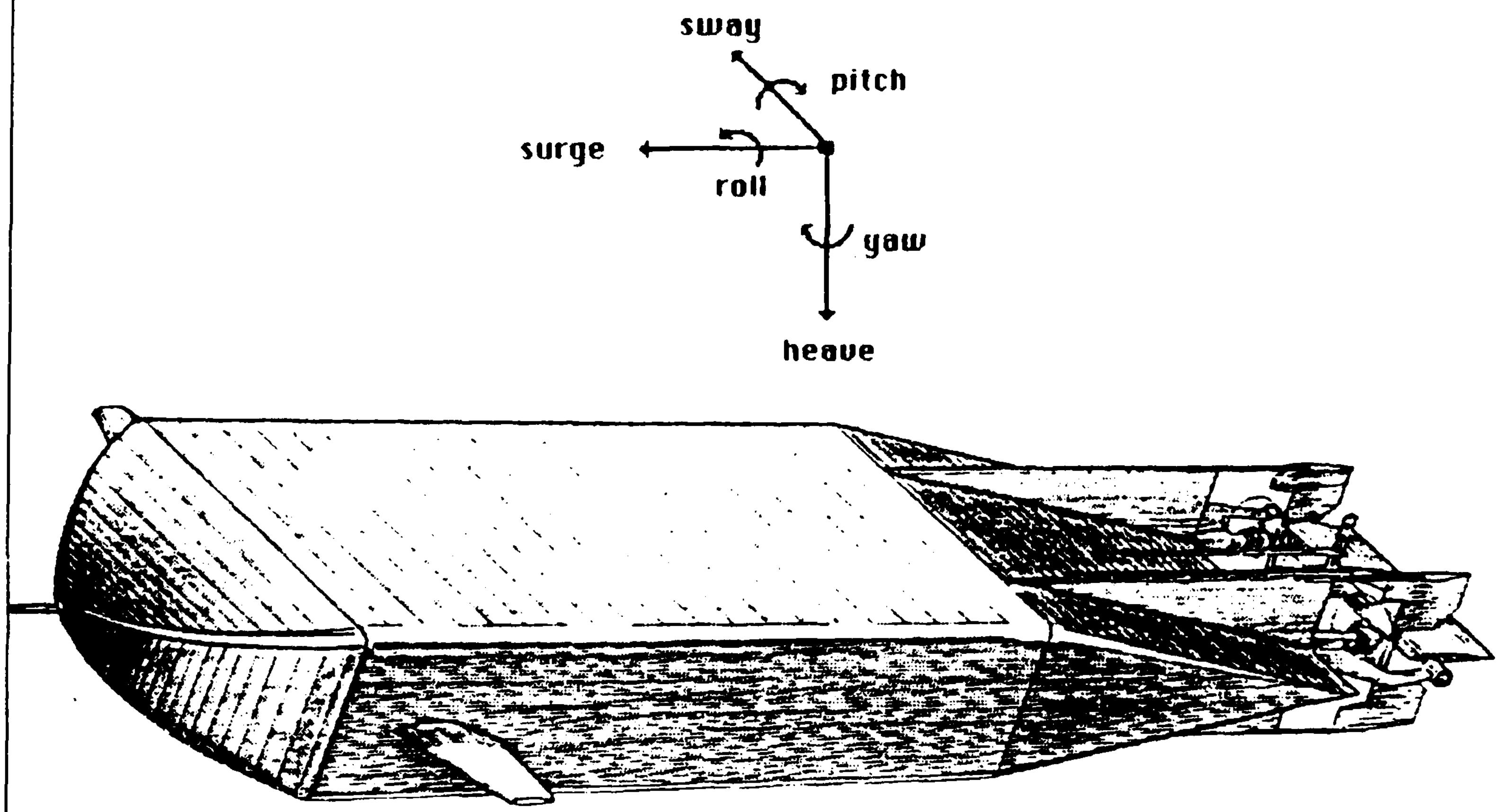


Figure 25. Sketch of the AUV with Euler Angles "From Ref. 5."

B. INTEGRATED SIMULATION ENVIRONMENT INTERFACE PROGRAM

The control code, developed in chapter 2, and the model simulation code required an interface program that would mock the actual hardware integration with the software. The program, auvs.c, performs this vital interface function, as well as demonstrating how field evaluation methods and troubleshooting with AUV run data can be accomplished. It features a scheme for calling the functions which make up the real-time software and the simulation code. It also gives one method for storing vehicle run data. This was the method used in the testing and simulation of the autopilot program which used the Doyle-Stein Observer. The suggested method for the actual AUV run data storage would be to open a data file upon completion of the autopilot update. This would allow the current position data and control surface actions to be congruent; however, this is not the order found in the interface program because the system, like all simulations, has a set of initial conditions which perturb the control system to perform a set of actions. This was the only simulation difficulty which could not be accounted for in the integrated package.

C. FIELD EVALUATION GRAPHICS ROUTINE

The data processed by the AUV during an actual deployment may have the need for immediate graphical interpretation. This case could arise due to a malfunctioning program or sensor. It was therefore essential to write a graphics routine which would take the uninterpreted data and graphically depict it with little effort by the evaluator.

1. C Functions used to Implement the Graphics Routine

The main program, plot.c, calls numerous functions to perform various tasks. The majority of these functions are system calls which are specific to Microsoft C version 5.1. Therefore, it is imperative that any future modifications made to the files composing the graphics routine be compiled using the Microsoft C 5.1 compiler. To invoke the executable file, type graph <filename>.

The data file should be set up in the same format as that produced by the interface program. In particular, the file should consist of column headings followed by the output of a particular state, sensor, or control surface in column format. The first column of the data file will always be taken as the abscissa. Therefore, to ensure that the graphs represent meaningful data, time should always be the first column of the data file. The ordinate label is the current heading of the data file. The source code for all of the graphics files are in Appendix D.

a. Main Program PLOT

This file sets up the graphics environment, as well as actually creating the graphs. Numerous system functions once again are used; the prototypes and descriptions of these functions can be found in Microsoft C version 5.1 documentation. The main program calls one user-defined function which is graphics_mode. The next section deals with this function in detail. A header file is also associated with the graphics files, grph.h. This file contains structures and prototypes common to both of the primary files.

The program begins by setting up buffers for the storage of data file headings, data, graph titles, and graph subtitles. The defaults for plot representation are white foreground, black background, and solid line representing the plotted data. The user is prompted for the use of grids on the graph. The only correct replies to this prompt are lowercase y or n; any other response will cause the prompt to reappear. Due to the nature of the program, which is to plot dynamic arrays, the program must perform some learning tasks. Therefore, it is not as rapid a plotting routine as one that knows the size of the array prior to compilation time, but the capability to plot varying array sizes is a requirement for field evaluation. The program determines the array size by reading the data file the first time through. During this first read, the program determines how many columns to expect and the length of the columns or number of rows in the array. The program does not store or act on any of the actual data until the second read.

The capability of the program to dynamically set the data storage area is a feature which is a strong point of C [Ref. 6]. The plots are displayed in succession (i.e., order is determined by the column in the data file), and are held on the screen until the user presses the return key. The graphs are autoscaled, no manual scaling is available to the user, so that the largest and smallest values are depicted. A typical output of the program is represented by Figure 26. Prior to terminating, the executable code resets the video mode of the computer system back to its previous mode.

b. Function GRAPHICS_MODE

This function is contained in the file, setvid.c, and is called by the function, plot.c. It does not pass any arguments and returns a void. The function checks the computer system for a graphics card. If a graphics card is present, the card is automatically placed in its highest resolution mode. This will cause a problem if the system that the program is running on has a Video Graphics Adapter card (VGA) which is set for enhanced mode, and the system is using an enhanced graphics adapter monitor. The program will automatically set the graphics adapter card to its highest setting which in this case is VGA; this results in the monitor receiving incorrect graphic information and will cause screen disruption or system lockup. Therefore, the graphics program should not be used in situations like this. The function tests for all common graphics cards. If no graphics card is present, the program informs the user that a graphics card is necessary and then terminates.

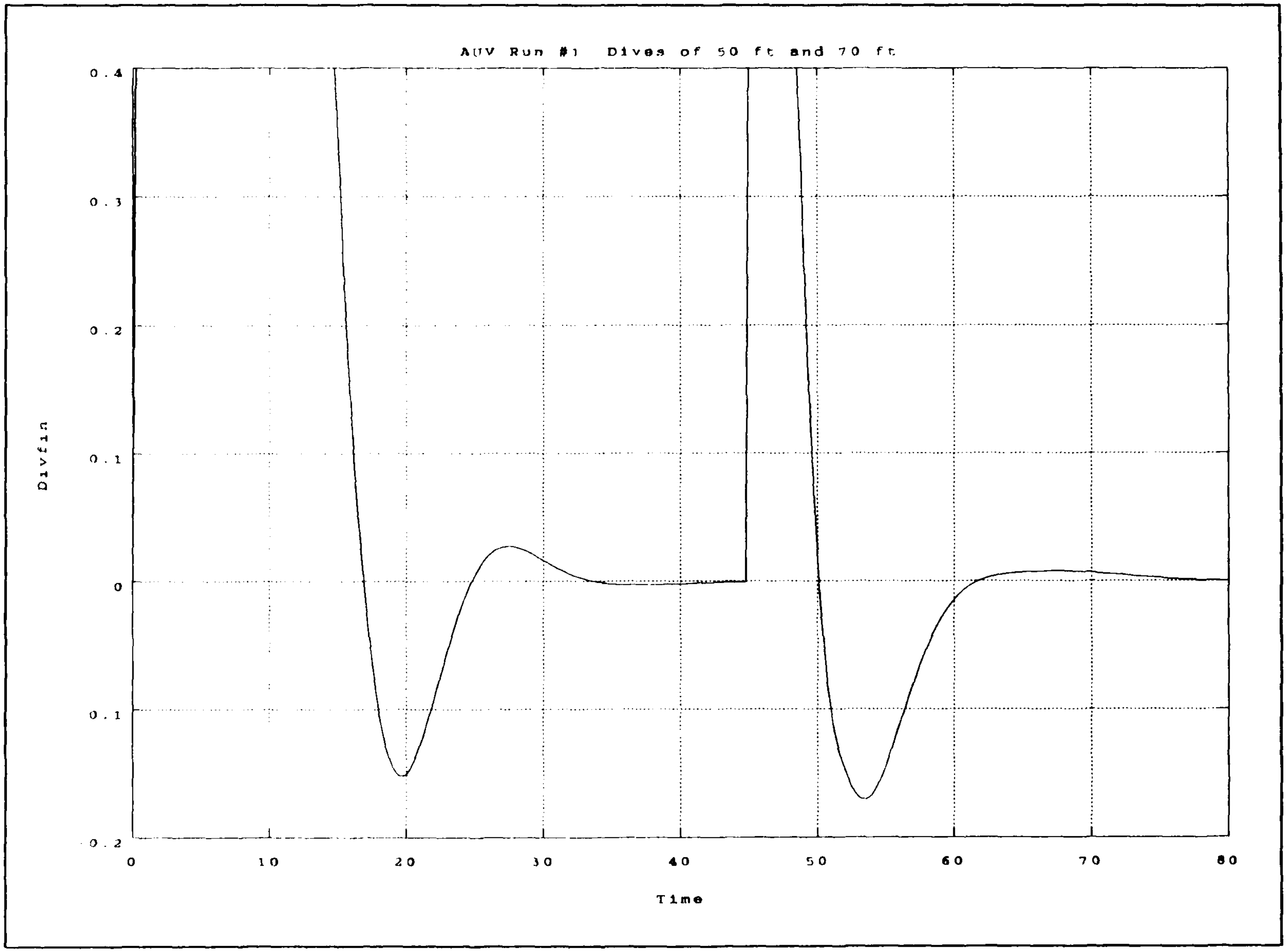


Figure 26. Typical Output of Graphics Program

IV. ASSOCIATED AUV HARDWARE

The hardware development and designs presented in this thesis are by no means all inclusive of the AUV project. The AUV project has been distributed hardware responsibilities among many individuals. The common bond has been the GRiDCASE 1535 EXP laptop computer and the associated data translation board. The initial designs of the AUV electronics called for an intricate network of sensors and control servos. These devices would be sampled and sent control signals, respectively, by the Grid interface network. This network consists of signal conditioning modules, power supplies, and sensors. A portion of the network is depicted in Figure 27.

The Grid computer was to initially contain all of the code for control, local path-planning, and mission-commander. This basic premise has grown to the current design which calls for a two computers, the Grid and a Gespack. The Gespack computer is a rack-based system which runs the OS9 operating system. It contains a separate data translation board from that of the Grid and is designed to operate both autonomously or networked. In this scheme the Grid acts as a quasi-Lisp-machine, since it runs a Lisp interpreter while not being a true Lisp-machine. It contains the Lisp code written for the global path-planner while the Gespack runs the control code for AUV mission. This modified hardware scheme still requires an elaborate interface network for signal conditioning. The following sections deal with two different aspects of the signal conditioning problem, filtering and synchro conversion of the rate gyro signals.

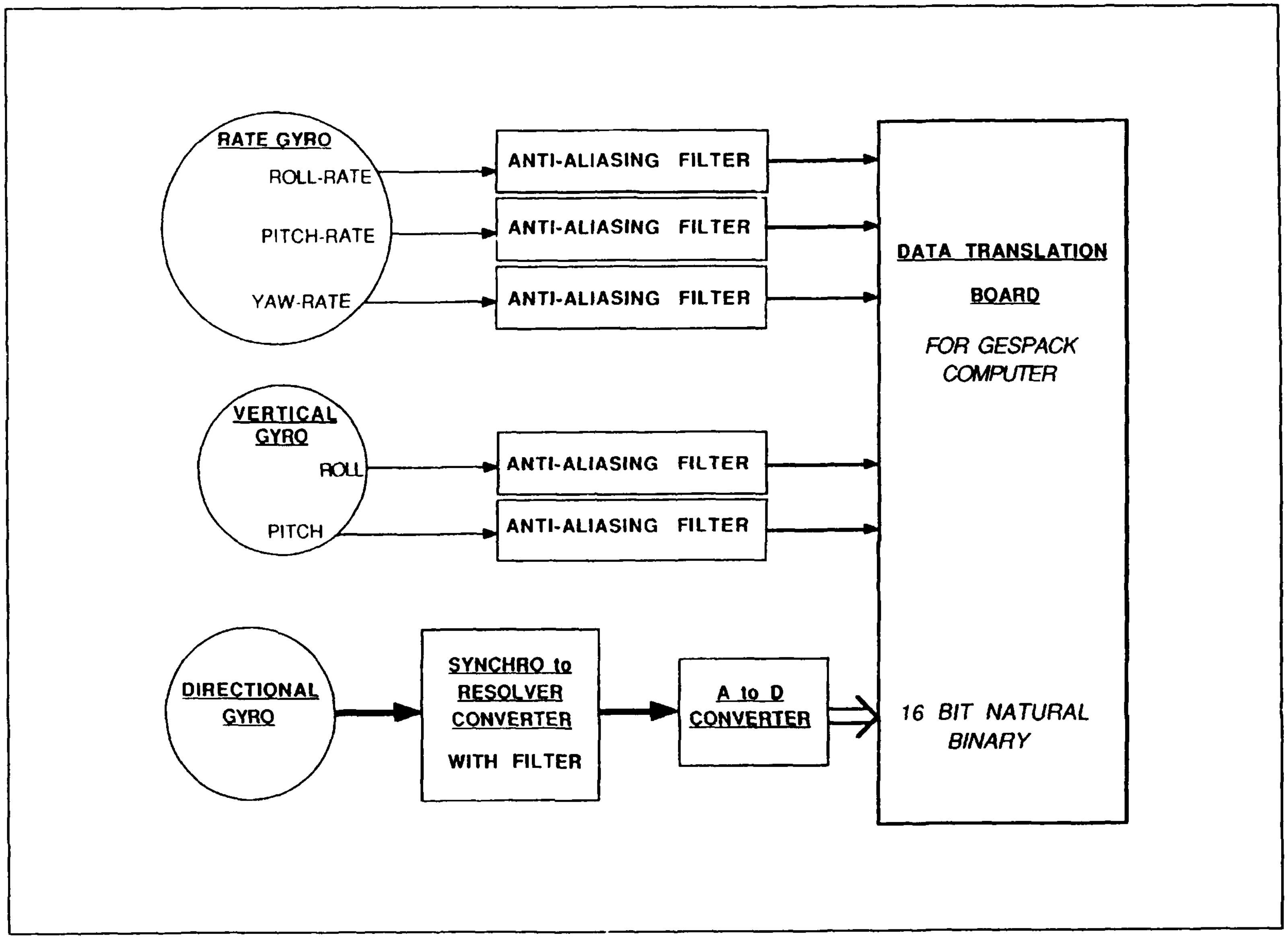


Figure 27. Signal Conditioning Network for AUV Gyros

A. LOWPASS ACTIVE ANALOG FILTER

This lowpass active filter was designed to be used as an anti-aliasing filter. Its relative position in the signal interface network is shown in Figure 27. In order to determine the filter cutoff frequency f_c, the open-loop bandwidth of the dive control system had to be determined along with the sampling frequency. The sampling frequency for the AUV prototype is proposed to be 20 Hz. This results in a significant oversampling ratio which is discussed in more detail later in this section. In order to determine the open-loop bandwidth of the dive control system, the transfer function or state space model for that system had to be derived. The open-loop Bode plot could then be plotted and the bandwidth determined.

The discrete-time transfer function for the dive system was developed by Schwartz [Ref. 3]. The corresponding continuous-time dynamics were derived and estimated by Davis [Ref. 7], and its transfer function has been estimated to be

$$H(s) = \frac{0.04}{s(s^2 + 0.7s + 0.04)} \tag{4-1}$$

which corresponds to

$$\begin{vmatrix} \dot{q} \\ \dot{\theta} \\ \dot{z} \end{vmatrix} = \begin{vmatrix} -0.7 & -0.04 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{vmatrix} \underset{\sim}{x} + \begin{vmatrix} 0.04 \\ 0 \\ 0 \end{vmatrix} u \tag{4-2}$$

in state-space format where q, θ, and z are defined in Table 1, and the input u is the dive command signal. This transfer function is based on a linearized model operating at 300 RPM. With the state space model defined, an open-loop Bode plot, magnitude only, was then produced. The code, auvbode.m, used to produce this plot is located in Appendix A. The Bode plot is depicted in Figure 28; it represents the dive system dynamics and gives us a numerical value for the system bandwidth. The Bode diagram demonstrates the low frequency nature of the AUV dive system. Because the system bandwidth is approximately 0.2 Hz and the sampling frequency is 20 Hz, the system is being significantly oversampled. This simplifies many associated sampling problems and insures that the Nyquist criterion

$$f_s > 2f_{max} \tag{4-3}$$

is satisfied [Ref. 8]. In determination of the cutoff frequency f_c needed for the anti-aliasing filter, the bandwidth was rounded up to 0.5 Hz. This fact, coupled with the 20 Hz sampling frequency f_s, meant that the lowpass filter f_c was required to be within the range of 5 Hz to 15 Hz. This is due to the fact that the next harmonic of the system will occur at a frequency equivalent to the sampling frequency f_s of 20 Hz. Based on these facts, the lowpass f_c was set at 10 Hz. Also due to the high sampling frequency with respect to the system bandwidth, a reconstruction filter associated with the anti-aliasing filter was not needed [Ref. 8].

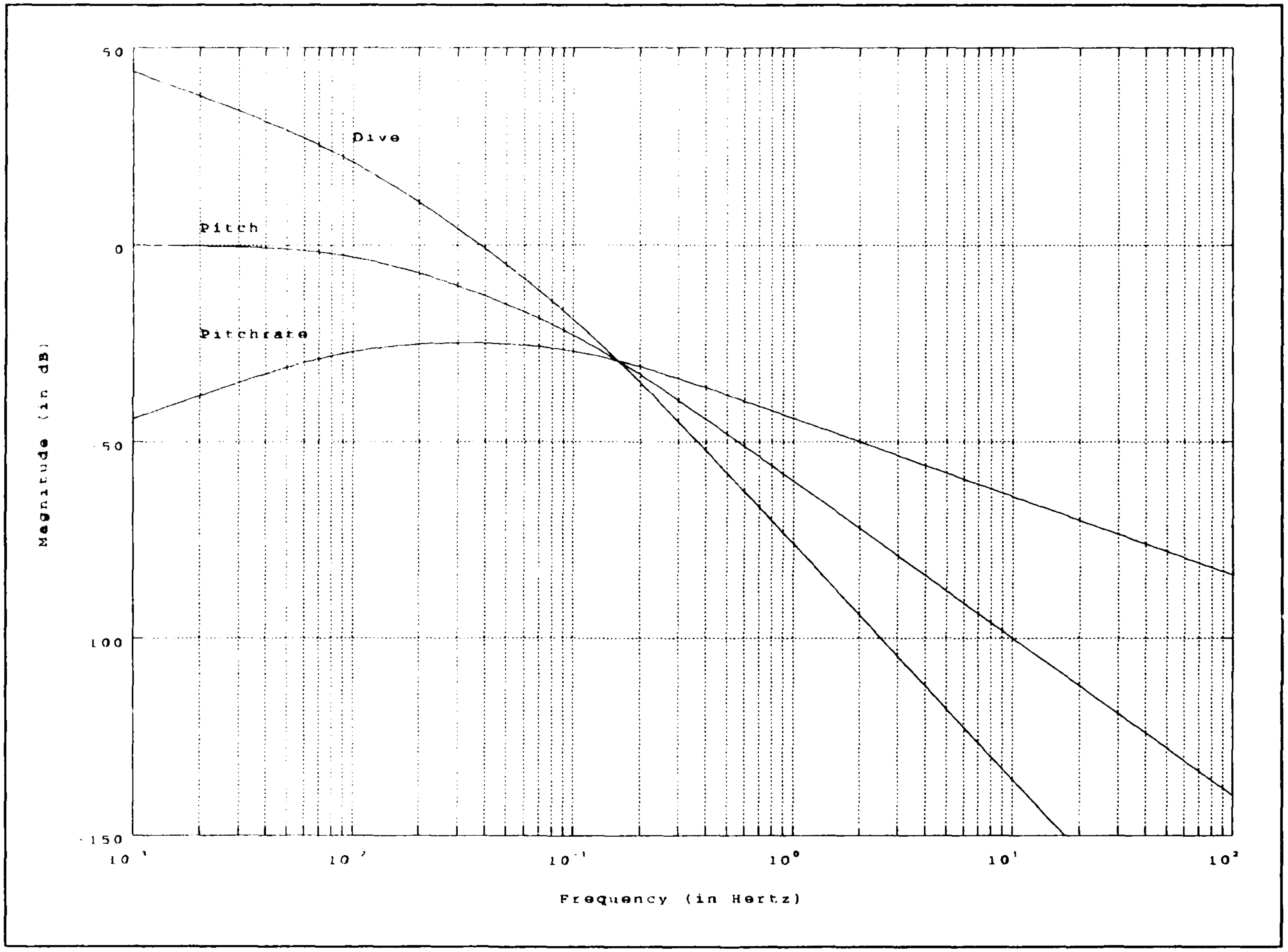


Figure 28. Open Loop Bode of the Diveplane System

1. Numerical Development of Active Lowpass Filter

The active-resistor-capacitor filter, active-RC, development is based on the generalized-immittance converter (GIC) circuit. This is due to the fact that active-RC filters can be derived directly from passive-RLC prototype networks by replacing the passive inductors with active, GIC-simulated inductors as shown by Ghausi and Laker in Ref. 9. The development of the basic transfer function for the active-RC filter using a biquadratic GIC scheme is also shown in Ref. 9. From this basic derivation, the generic lowpass filter transfer function can be derived. This is the basis of the analytical and numerical development of the lowpass active-RC filter designed. The determination of the appropriate values for the circuit components was based on

$$2\pi f_c = \frac{1}{RC} \tag{4-4}$$

where f_c is the cutoff frequency and

$$\omega_t = 2\pi \times 10^6 \tag{4-5}$$

is the gain bandwidth product. In addition to these parameters, the following design specifications were considered: maximally flat Butterworth filter (i.e., Q = 0.707), f_c = 10 Hz, and negligible phase shift in the frequency region of interest.

To limit the amount of noise introduced by this circuit, the resistor values were limited to 100 kΩ. Therefore; solving Equation (4-4) for the capacitor, a value of 0.159 μF was found. The complete schematic circuit diagram is located in Appendix E. Due to the fast sampling rate, a second-order Butterworth filter with a slow roll-off was designed. This circuit is a monatomic design; that is, for each sampled sensor a lowpass anti-aliasing filter exists as shown in Figure 27.

2. Evaluation of Numerical Results

Based on the results of the previous section, a numerical transfer function was determined. This transfer function was based on the generic lowpass filter function derived in Ref. 3. Using this transfer function the Bode diagrams representing the magnitude and phase of the designed lowpass filter were obtained and are depicted in Figures 4-2 and 4-3, respectively. The magnitude graph shows that, for an ideal op amp, the circuit's 3-dB frequency is 10 Hz, as called for in the specifications. While the phase plot shows that in the frequency region of interest (i.e., 0.2 Hz), a negligible phase shift exists, as desired.

The experimental results were well within a tolerable deviation from the ideal results. These results were also supported by simulation using Spice, a computer circuit simulation program. Due to the very low frequency involved, a GIC-circuit was a logical choice. This is due to the sensitivity of the circuit which has been proven in related experimentation [Ref. 9].

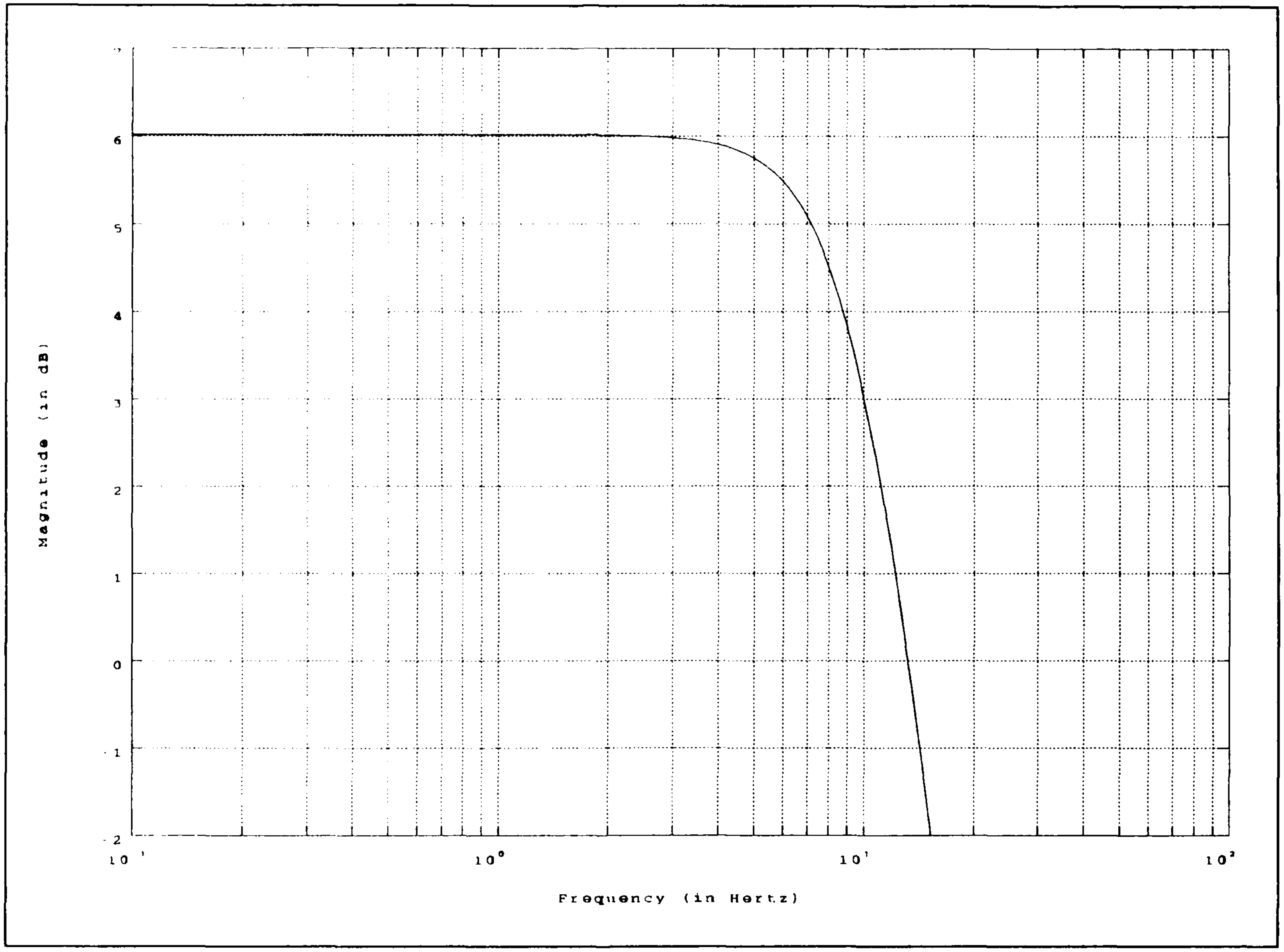


Figure 29. Magnitude Portion of Lowpass Active Filter Bode Plot

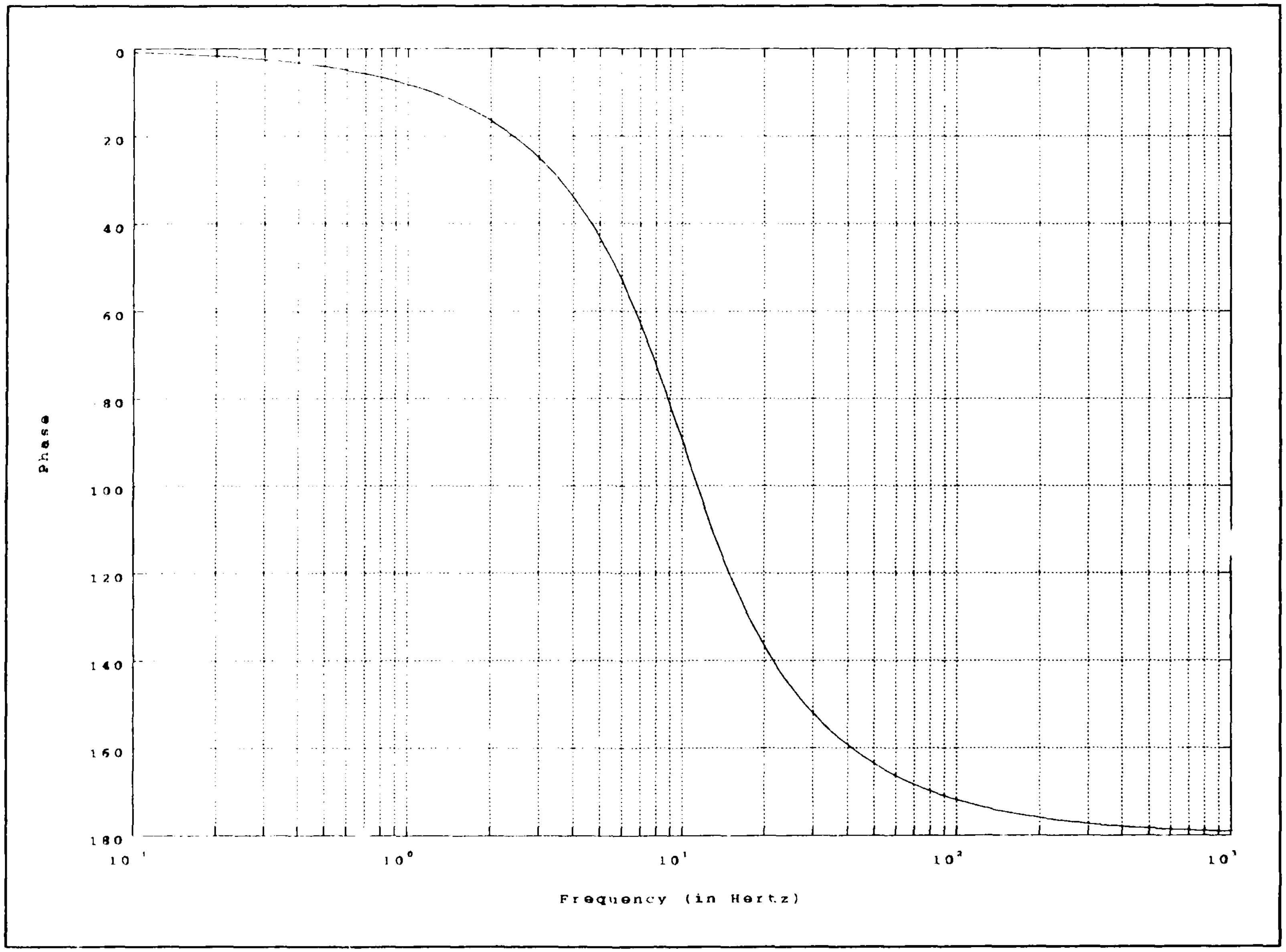


Figure 30. Phase Portion of Lowpass Active Filter Bode Plot

B. SYNCHRO-TO-RESOLVER CONVERSION

In order to determine the direction or heading of the AUV, a highly accurate Flux Gate Gyro was purchased. The gyro specifications are 400 Hz, 26 volt reference, with an 11.8 volt line signal. Due to these specifications which required higher voltages and subsequently produce higher output voltages than are capable of being handled by the system hardware, a conversion scheme had to be worked out so that the line voltages were within a specified range (i.e., 2.0 volts), but still maintained the needed accuracy which is inherent in the 11.8 volt line signal. To overcome this problem, a set of chips was purchased which included the following: reference isolation transformer, signal isolation transformer, and analog-to-digital converter. The problem that had to be resolved was the design of this circuit including all of the passive elements which are used to set various parameters on the transformer chips.

1. Directional Gyro Background

Directional gyros have two degrees of freedom. The first degree of freedom referred to as the inner gimbal axis, allows the spin axis to be maintained horizontal with respective to the fixed Earth reference coordinate system. The second degree of freedom, referred to as the azimuth or outer gimbal axis, detects and indicates movements of the vehicle in azimuth from a heading reference. This reference in the case of the AUV is along the x-axis or the surge-axis as previously defined. Therefore, it is apparent that the spin axis of the directional gyro must continue to point in a fixed direction in a horizontal plane. The term fixed direction refers to a mode, <u>slaved</u>, or type of gyro which makes corrections in order to maintain a fixed orientation with respect to the Earth reference frame. In order to do this, most gyros require the corrections to be made externally; hence, the role of the flux gate is revealed. A slaved gyro is precessed in azimuth by an external reference signal such as a magnetic transmitter. In this case the magnetic transmitter (flux gate) continuously aligns the azimuth gimbal to magnetic north. Therefore, the overall accuracy of the gyro depends in a large part on the flux gate output. We note that spurious magnetic anomalies and interference will have an adverse affect on the flux gate output and thus result in inaccurate headings. Due to this sensitive characteristic, the flux gate on the AUV has been encased in a nonmagnetic metal alloy known as mu-metal. This is intended to lessen the probability of error induced by magnetic interference.

Equipment associated with gyros are typically power supplies, and synchros, or resolvers. Usually, these pieces of equipment are internal to the overall housing of the gyro. The power supply performs an obvious task, but some discussion on the function of the synchro and resolver is warranted. The synchro and resolver perform identical missions, that of encoding the gyro output on a signal to enable other equipment to have access to this information. The primary difference in the two schemes is the method of encoding the direction information.

The rotor and stators in a resolver are oriented 90 degrees with respect to each other as in Figure 31. This electrical arrangement results in the following

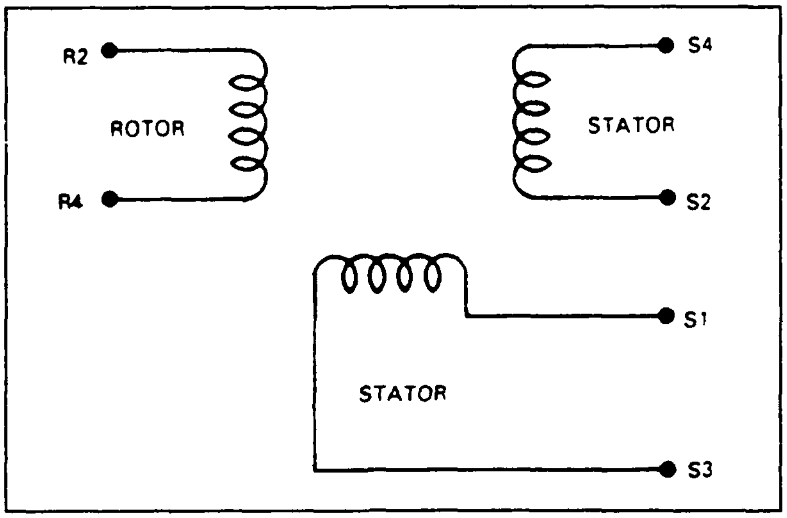


Figure 31. Simple Resolver (After Ref. 7.)

resolver shaft angle equations:

$$R = A\sin(\omega t) \qquad (4\text{-}6)$$

$$S_{1-3} = A\sin(\omega t)\sin(\theta) \qquad (4\text{-}7)$$

$$S_{4-2} = A\sin(\omega t)\cos(\theta) \qquad (4\text{-}8)$$

where R represents the rotor excitation voltage which is the ac reference voltage, and θ is the resolver shaft angle. Thus the output voltage which appears across stator terminals S_1-S_3 and S_4-S_2 contains the necessary angle information. This information must be converted into a usable format (i.e., natural binary or dc), hence the need for resolver conversion.[Ref. 10]

The synchro rotor-stator configuration differs from the above resolver format in that the orientation of the stators with respect to each other is 120 degrees. This is depicted in Figure 32. Due to this orientation of the stators, three phases of shaft angle information exist while in resolver format only two phases exist as previously shown. The resulting synchro format voltage equations are:

$$S_{1-3} = A\sin(\omega t)\sin(\theta) \tag{4-9}$$

$$S_{3-2} = A\sin(\omega t)\sin(\theta + 120^\circ) \tag{4-10}$$

$$S_{2-1} = A\sin(\omega t)\sin(\theta + 240^\circ) \tag{4-11}$$

These equations represent the line signal voltage which carries the shaft angle information. The rotor excitation voltage is the same as that of Equation (4-6); also the amplitude of the line voltage signal is the same as the reference signal. This is logical because the ac-reference signal acts as a carrier for the line signal [Ref. 10]. The problem that had to be solved was how to interpret the information contained on each line signal coming from the stators. The three-phase output produced by the synchro had to be converted to a usable format for interface with the computer.

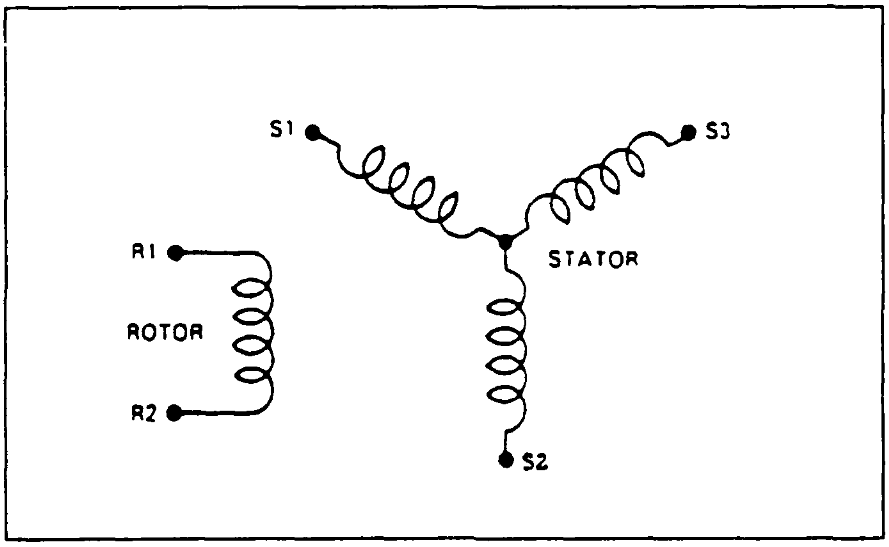


Figure 32. Simple Synchro (After Ref. 7.)

2. Synchro Conversion Circuit Design

The output format of the flux-gate gyro presented several problems to the system designers of the AUV. Some of these problems included the high line signal and reference voltages. The data translation board would only accept a maximum of +5 volts rms. The desired voltage was +2 volts rms. Another consideration was how to handle the information presented in a 3-phase output such as the one present in the flux-gate gyro, and lastly what type of information format was desired. The data translation board can accept both analog and digital inputs. This section deals with one solution to the problem.

a. Component Selection

Presented with the format problem mentioned above and the compatible output required to interface with the data translation board, the following components were selected: 5S70/411, 5S72/26V, 2S80KD, and AD767KN. These components are all manufactured by Analog Devices tc perform the task of synchro-to-resolver conversion. The additional passive components required in the design are off-the-shelf resistors and capacitors.

The 5S72/26V is a reference-signal-isolation transformer. It takes the reference signal used in the rotor excitation, Equation (4-6), as a input. The transformer then steps down this voltage to the standard 2 volts rms used by most analog-to-digital converters. The transformer also provides isolation for the electronics from the resolver.

The 5S70/411 is a line-signal-isolation transformer. It accepts the output of the synchro stators, Equations (4-9), (4-10), and (4-11), as a input. The 5S70/411 converts the three-phase 11.8 volt signals into a two-phase 2 volt rms resolver format. This is the format required by most analog-to-digital converters used in control and sensor applications. The 5S70/411 is designed specifically for synchro-to-resolver conversion of 11.8 volt line signals.

The 2S80KD is a monolithic tracking resolver-to-digital converter. This converter uses a ratiometric tracking conversion technique which provides continuous position data without conversion delay. The ratiometric conversion technique is based on the ratio of the resolver format voltages being equal to the tangent of the shaft angle. Therefore, the tracking converter is not dependent on the absolute magnitudes of the signal input [Ref. 10]. It also allows the circuit designer to select passive components to define the appropriate bandwidth and the desired resolution of the converter.

The AD767KN is a digital-to-analog converter. It accepts a 12-bit binary input and converts it to a analog output. This particular component was not used in the synchro-to-resolver conversion design, but can be implemented without the need for handshaking in the circuit design which appears in Appendix E. This chip would be needed only in the event that the binary input lines to the data translation board are required for additional purposes other than the directional gyro information. In this case the 12-bit binary output of the 2S80KD would need to be converted to a analog signal and input into the data translation board in this manner.

b. Design Considerations

The passive component selection was based on the desire to achieve the highest possible bandwidth and resolution. The components selected were all within 5% tolerance. In reference to the circuit schematic found in Appendix E, the resistors R1 and R2 alongwith the capacitors C1 and C2 makeup an HF filter. The purpose of this filter is to reduce the amount of noise present on the signal inputs to the 2S80. The values of C1 and C2 are dependent on R1 which is equal to R2. If the value of R1 is selected based on optimal value given by the manufacturer, the following equation can be solved for the values of C1 and C2:

$$C1 = C2 = \frac{1}{2\pi R1\, f_{REF}} \tag{4-12}$$

The resistor R4 is a gain scaling resistor whose value is determined by

$$R4 = \frac{DC_{err}}{100 \times 10^{-9}} \left(\frac{1}{3}\right) \Omega \qquad (4\text{-}13)$$

For resolution of 16 bits, the DC-error component is 0.0025. Components R3 and C3 act as a all-pass filter. These components introduce a phase shift in order to offset any phase shift introduced by the circuit at the reference frequency. The values for these components are $R3 = 100k\Omega$ and $C3 > \frac{1}{10^5 f_{REF}}$. The maximum tracking rate is set by the voltage-controlled-oscillator input resistor R6. This is accomplished by selecting the desired maximum tracking rate T_{max} which is not to exceed 1/16 of the reference frequency.

$$R6 = \frac{5.92 \times 10^7}{65{,}536(T_{max})} \, k\Omega \qquad (4\text{-}14)$$

The maximum tracking rate is the highest angular speed for which the converter output is able to keep track with the input. The tracking rate has a direct relationship to the reference frequency (i.e., the nigher the reference frequency, the higher the tracking rate) [Ref. 11].

The closed-loop bandwidth of the converter is selected by adjusting capacitors, C4 and C5, and resistor R5. The bandwidth of the converter is a measure of the acceptable variance of a sinusoidal input to the converter (i.e., a 80 Hz bandwidth converter will accept a varying input sinusoid < 80 Hz) [Ref. 11]. This implied that the converter for the AUV did not require a large bandwidth because the directional gyro line-signal will vary slowly. While this is, in fact, the case, the circuit was designed with a 80 Hz bandwidth to allow for any future modifications to the associated systems of the AUV. The trade-off for the increased bandwidth was an increase in the value of R5 by 20% while C4 and C5 decreased in value 30% and 50% respectively. The following equations were used in determining the appropriate values of the bandwidth components:

$$f_{REF} > 2.5(f_{BW}) \tag{4-15}$$

$$C4 = \frac{20.2 \times 10^{-3}}{R6 \times f_{BW}^{2}} \tag{4-16}$$

$$C5 = 5 \times C4 \tag{4-17}$$

$$R5 = \frac{4}{2\pi f_{BW} C5}\ \Omega \tag{4-18}$$

Offset and bias current at the integrator input of the 2S80KD can cause an additional error. This error can be lessened or completely eliminated with the use of an offset bias circuit. This offset bias circuit is realized in the design by placing a resistor, R8, at the integrator input and tying its output to a potentiometer, R9, which is coupled across the ± 12 volt dc source . The values of these components, as well as C6 and R7 are given in Reference 12.[Ref. 12]

c. *Evaluating the 2S80KD Converter Output*

Many possible methods exist for representing angular information in digital form. The most common method is natural binary coding. The most significant bit in this system represents 180 degrees, the next 90 degrees, and so on. A representation of bit and angle equivalence is listed in Table 2. This table shows the accuracy possible with the converter set in 16 bit resolution mode. In this mode the accuracy is approximately 19.8 arc seconds. The circuit can be modified to represent the output in binary coded decimal (BCD) for tests and evaluation by sending the output of the 2S80KD to a binary to BCD converter. The resulting output can then be sent directly to a seven-segment decoder for visual display purposes.

Table 2. NATURAL BINARY BIT CONVERSION SCHEME

Bit Number	Angle in Degrees
1 MSB	180.0
2	90.0
3	45.0
4	22.5
5	11.25
6	5.625
7	2.8125
8	1.40625
9	0.70313
10	0.35156
11	0.17578
12	0.08790
13	0.04395
14	0.02197
15	0.01099
16	0.00549

V. CONCLUSIONS

The Variable Structure Controller has been shown to be robust and to work under adverse operating conditions. This makes it well-suited for application in the AUV. Simulation results have shown that, despite external disturbances, the system's response converges to a desired equilibrium. In order to preserve this property, a robust observer was designed to estimate the signals in the loop. This observer design is based on the Doyle-Stein robust observer condition.

The observer signals are used in an error detection algorithm, which detects the presence of a sensor failure. The purpose of this is to eliminate the need for redundant hardware systems, while still maintaining reliable control operation in case of sensor failure. The simulation results show that the autopilot equipped with the sensor failure detector we developed can still maintain stability in the presence of sensor errors. This is conceptually again a new approach to an old problem that of system redundancy. When dealing with Autonomous Underwater Vehicles, new methods for system redundancy are required due to space limitations. This autopilot algorithm has proven to be accurate and as sensitive to deviation in the sensor errors as the user desires. The inclusion of the error detection algorithm did not result in any degradation in efficiency of the overall autopilot.

When converting the autopilot algorithm to the "C" language, an interface module was necessary to link the converted algorithm to a converted model. The model used to test the real-time autopilot had to be modified to simulate the envisaged sensor sampling of the actual hardware. This modified model behaved identically to the original "C" model which declared all variables as global.

The anti-aliasing filter design was tested in a lab environment with very satisfactory results. The very low frequency of this filter and its intended use called for a special design. The GIC active-filter was proven very well suited for the application.

The software developed for the autopilot is ready to be implemented on the onboard computer of the AUV (Gespack). Wet-bed testing of the whole system in various maneuvers is the next step in the development of the autopilot.

APPENDIX A. HIGH LEVEL DEVELOPMENT SOURCE CODE

This appendix contains the high level source code used in testing and evaluation of the autopilot algorithm. The source code is in Matlab, a "C"-based development language. This source code was used to produce all graphical representations presented in this thesis. It is not part of the real-time code intended for the AUV prototype which is written exclusively in "C", and is located in Appendix B. A copy of this source code can be obtained from Professor Roberto Cristi, Naval Postgraduate School (see Initial Distribution List for mailing address).

Matlab is a registered trademark of The Math Works, Incorporated. It does not produce stand-alone executable code, but instead is an interactive program intended for numerical problem solving. This program is not available with the source code in this appendix. Matlab is required when using the source code listed in this appendix.

```
  %  Filename    "auvobs.m"
  %  Doyle - Stein observer for continuous model of AUV
  %  This program was used to develop and evaluate the real-time
  %  autopilot code for the AUV. It calls a model that is written in
  %  "C" language to update the states using the dynamic equations
  %  motion for the submersible.
clear
ERXP=['Failure simulated in pitch at 50 ( Sensor Pitch = -35 )'];
ERXQ=['Error induced in pitch rate at 5 ( Sensor Pitch Rate = 100000)'];
ERXQP=['Sensor Pitch Rate = 100000 at 5, Sensor Pitch = -35 at 50'];
 q=input('Input your q for this run    ');
  a=.04 ;
  b=.7  ;
  A  = [  -b  -a  0
           1   0  0
           0  -1  0];   % !!!!!! notice  the minus signs
% s=[1; 0.2070;-0.0198]  % left real eigenvector of A-B*L
 s=[1; 0.7; -0.04];      % left eigenvector of A for lambda=0
  B  = [  -a;0;0];       % to get model states to track right
  C  = [ 0 0 1];
  D  = 0;
  Qe = [   .01      0      0
           0      .01      0
           0       0     .01];
  Re = [ 1 ];
  G  = 1;
  qc = [ 1   0   0
         0   1   0
         0   0   1];
  rc = [10];
  dt=.25;
  phi=eye(3)+A*dt;
 del=B*dt;
 L=lqr(A,B,qc,rc)          % returns optimal feedback gains
  Q = Qe*1 + q^2*B*B';
 Kd=lqe(A,G,C,Q,Re);     % returns optimal kalman gains
 Kd=Kd*dt
% observer dynamics:
 phi0=phi-Kd*C;
 ac=poly(phi0)

                                kmax=400;
                            ordered_depth=150;
                            previous=ordered_depth;
                            depth=100;
                            rpm=500;
                            xdot=4.0;
                            transient_flag=3;
                            thres1=.001;
                            thres2=.01;
                            thres3=.01;
  abspeed = xdot;
  pitchrate=0.0;
  pitch=0.0;
  r=[zeros(1,kmax)
    zeros(1,kmax)
    depth*ones(1,30) ordered_depth*ones(1,kmax-30)];
  state = zeros(12,kmax);
  xo = zeros(3,kmax);
  uk = zeros(1,kmax);
  error = zeros(3,kmax);
  auv_normalz = zeros(1,kmax);
  initial_state = [xdot;0;0;0;pitchrate;0;0;0;depth;0;pitch;0];
  xo(:,1) = [pitchrate;pitch;-depth/abspeed];     % initialize observer
  state(:,1) = initial_state;
%------------------------------------- main loop ------------

 for j=1:kmax-1
    if j > 500,
         ordered_depth=120;
    end;
%------Allow for Error Matrix to Change States---------%
    if previous ~= ordered_depth,
          transient_flag = j+2;
    end
    u=L(3)*ordered_depth/abspeed;
    u=u-L*[state(5,j);state(11,j);state(9,j)/abspeed];
    z=state(9,j)-ordered_depth;
    q=state(5,j);
    th=state(11,j);

% check the error to the threshold
      if abs(error(1,j)) > thres1,
         p_rate=xo(1,j);
         fprintf('pitchrate is using obs\n')
      else
         p_rate=q;
      end
      if abs(error(2,j)) > thres2,
         pit = xo(2,j);
         fprintf('pitch is using obs\n')
      else
         pit = th;
      end
      if abs(error(3,j)) > thres3,
         norm_dep_err = xo(3,j);
         fprintf('normalized depth error is using obs\n')
      else
         norm_dep_err = z/abspeed;
      end

    ex=[p_rate; pit; norm_dep_err];      % setup to compute nonlin
    ste(j)=s'*ex;
    ubar=1.0*(ste(j));
    if (abs(ubar) > 0.4)
```

```
        ubar=0.4*sign(ste(j));
    end
    u=u+ubar;
    divfin = u;
    if (abs(divfin) > .4  )
        divfin = 0.4*sign(divfin);
    end;
    inputs=[0; divfin;-divfin;rpm];
    u = divfin;
    state(:,j+1)=model(state(:,j),inputs,dt);

    uk(j+1)=u;
    xo(:,j+1)= phi*xo(:,j) + del*u ;
    xo(:,j+1)= xo(:,j+1) + Kd*(z/abspeed - C*(xo(:,j)));

    abspeed=sqrt((state(1,j+1))^2+(state(2,j+1))^2+(state(3,j+1))^2);

     % --- simulate a fault in q (pitchrate)
       if j>5;
           q=100000;
       end
     % --- end fault
     % --- simulate a fault in theta (pitch)
        if j>50;
           th=-35.0;
        end
     % --- end fault
% transient correction for the observer

    xe(:, j+1)=[q;th;z/abspeed]-xo(:,j);
    if j>transient_flag,
error(:,j+1)=ac(1)*xe(:,j+1)+ac(2)*xe(:,j)+ac(3)*xe(:,j-1)+ac(4)*xe(:,j-2);
    end
previous=ordered_depth;
end             % end of main loop
```

```
% Begin plotting routine

subplot(211),plot(xe(2,:)), title('Uncorrected Pitch Error')
subplot(212),plot(xe(1,:)), title('Uncorrected Pitchrate Error')
%meta uncorE
pause
clg
axis([0 kmax -.2 .5])
subplot(211),plot([0 kmax],[.05 .05],':')
%text(300,-.14,'Threshold');
text(300,.05,'Threshold');
hold on;
plot(abs(error(2,:))), title('Corrected Pitch Error')
%text(.13,.525,ERXP,'sc');
hold off;

axis([0 kmax -100 1000])
subplot(212),plot([0 kmax],[.005 .005],':')
text(300,.005,'Threshold');
hold on;
plot(abs(error(1,:))), title('Corrected Pitchrate Error')
text(.09,.015,ERXQ,'sc');
hold off;
meta errbigq
pause

clg
axis([0 kmax -.2 .4]);
subplot(211),plot(abs(error(3,:))), title('Depth Corrected Error')
hold on;
plot([0 kmax],[.2 .2],':')
text(300,.2,'Threshold');
hold off;
subplot(212),plot(state(5,:)), title('Pitch Rate')
text(.1,.03,ERXQP,'sc');
```

```
%meta simderr
pause

clg
axis([0 kmax 100 160]);
 subplot(211),plot(state(9,:)),title(' Model Depth')
hold on;
plot([0 225 226 kmax],[140 140 120 120],':')
text(20,140,'Ordered Depth');
hold off;
axis([0 kmax -.45 .45]);
 subplot(212),plot(uk),title('Divefin Action')
 text(.1,.03,ERXQP,'sc');
 pause
%meta simdd

clg
 subplot(211),plot(state(11,:)),title('Pitch ')
 subplot(212),plot(xo(2,:)),title(' Observer Pitch ')
 pause
clg
subplot;
axis;
plot(ste),title(' Sliding Surface'),grid
text(.1,.03,ERXQP,'sc');
```

```
  %  NON-ADAPTIVE VARIABLE STRUCTURE DIVEFIN CONTROL
  %  filename = dauv.m
  %  This program is used to determine the optimal feedback gains of
  %  the fullstate feedback sliding mode contoller. It also test the
  %  accuracy of the variable structure controller.
clear;
  a=.04 ;
  b=.7  ;
  A  = [  -b  -a  0            % State space model of Dive system
          1   0  0
          0  -1  0];   % !!!!!! notice  the minus signs
  s=[1; 0.2070;-0.0198]; % left real eigenvector of A-B*L
  B  = [ -a;0;0];      % to get model states to track right
  C  = [ 0 0 1];
  qc = [ 1   0   0
        0   1   0
        0   0   1];                % error covariance noise
  rc = [10];                       % measurement noise
  dt=.25;
  L=lqr(A,B,qc,rc);                % Optimal Feedback gains
                  kmax=400;
                  depth=0;
                  rpm=600;
                  xdot=4.0;
 am=A-B*L               ;
 [eigvec,eigval]=eig(am');
 lambda=real(eigval(1,1));
 s=eigvec(:,1)           ;       % Left eigenvector of dive system
 st=s'

  abspeed = xdot;
  pitchrate=0.0;
  pitch=0.0;
  state = zeros(12,kmax);
  uk = zeros(1,kmax);
  auv_normalz = zeros(1,kmax);
  initial_state = [xdot;0;0;0;pitchrate;0;0;0;depth;0;pitch;0];
  state(:,1) = initial_state;
%-------------------------------------- main loop -----------
 i = 0;
 for j=1:kmax-1,
    if (j < 180),
                  ordered_depth=50;
    else,
                  ordered_depth=70;
    end
    u1=L(3)*ordered_depth/abspeed;
    u=u1-(L*[state(5,j);state(11,j);state(9,j)/abspeed]);
    z=state(9,j)-ordered_depth;
    q=state(5,j);
    th=state(11,j);
    ex2=[q;th;z/abspeed];
    ste(j)=s'*ex2;
    ubar=1.0*(ste(j));
    if (abs(ubar) > 0.4)
       ubar=0.4*sign(ste(j));
    end
    u=u+ubar;
    divfin= u;
    if (abs(divfin) > .4  )          % Limit maximum deflection to .4 radians
        divfin= 0.4*sign(divfin);
    end;
    inputs=[0;divfin;0;rpm];
    u= divfin;
    state(:,j+1)=model(state(:,j),inputs,dt); % Call the dynamic equations model
    uk(j+1)=u;                          % Save u for plotting purposes
    abspeed=rpm/150;
%     abspeed=sqrt((state(1,j+1))^2+(state(2,j+1))^2+(state(3,j+1))^2);
    time(j) = dt * i;
    time1(j) = dt * i;
```

```
      i=i+1;
   end
   time(j+1) = dt * i;

% Begin plotting routine

axis([0,70,-1,51]);
plot(time,state(9,:)),xlabel('Time  (Sec)'),ylabel(' Depth  (Feet)'),grid
%meta chap2_1
pause
axis([0,80,-.2,.4]);
%plot(time,uk),xlabel('Time  (Sec)'),ylabel(' Divfin Action'),grid
plot(time,uk),title('AUV Run #1  Dives of 50 ft and 70 ft'),
xlabel('Time'),ylabel(' Divfin '),grid
meta chap3_2
pause
axis([0,70,-.08,.1]);
plot(time,state(5,:)),xlabel('Time  (Sec)'),ylabel('Pitch Rate'),grid
%meta chap2_3
pause
axis([0,70,-.8,.1]);
plot(time,state(11,:)),xlabel('Time  (Sec)'),ylabel('Pitch'),grid
%meta chap2_4
pause
axis([0,70,-.05,.25]);
 plot(time1,ste), xlabel('Time  (Sec)'),ylabel(' Sliding Surface'),grid
%meta chap2_5
axis;
```

```
  % Dive system BODE Plot
  % filename = auvbode.m
  % This program determines the Bode plot of interest for
  % the AUV dive system.

clear;
  % Set initial model conditions
                q11 =  1      ;
                q22 =  1      ;
                q33 =  1      ;
                r = [0;0;0];
                dt =  0.25    ;
                kmax =  400    ;

  a1= [  -.7  -.04  0
          1   0    0
          0   -1    0];
  b1 = [ -.04
         0
         0];
 q=[q11 0   0
    0  q22 0
    0  0   q33];

c=[0 0 1];

[num,den] = ss2tf(a1,b1,c,0,1)

w=logspace(-3,3,250);
[mag,phase]=bode(a1,b1,q,r,1,w);
axis([-3,2,-150,50]);
semilogx(w/(2*pi),20*log10(mag)),title('Magnitude Response for AUV Dive Controller')
xlabel('Frequency (in Hertz)'),ylabel('Magnitude (in dB)')
text(.002,-25,'Pitchrate')
text(.002,2,'Pitch')
text(.01,25,'Dive')
grid
  % Continuous-Time Nyquist Plot
  % filename q_cnyq.m
  % This program is used to determine the effect of varying q in the program
  % auvobs.m. This program performs the nyquist plot for a family of q's.

clear;
clg;
axis([-1.1 1.1 -1.1 1.1]);
axis('square');
dt=.25;
% Plant definitions
A=[-.7 -.04  0
   1   0    0
   0  -1    0];
B=[-.04; 0; 0];
C=[0 0 1];
s=[1; 0.7; -0.04];  %  left eigenvector of A for lambda=0
D  =[0];
Qe = [   .01      0      0
          0      .01     0
          0       0     .01];
   Re = [ 1 ];
   G  = 1;
   qc = [ 1   0   0
          0   1   0
          0   0   1];
   rc = [10];
   dt=.25;
 L=lqr(A,B,qc,rc);        % returns optimal feedback gains
 UR = zeros(3);           % upper right block of 'Ae' (A extended matrix)
 Be = [ B
       0
       0
```

```
     0];                 % B extended
 Ce = [0 0 0 L];         % C extended
w=logspace(-1,1,150);
tempq=input('Input 4 values of q as a vector (ie. [ ])   ');
% Begin loop for family of curves

plot(exp(sqrt(-1)*[0:0.05:7]),'-');
hold on;
title(['Continuous Time Nyquist Plot']),grid;
for i = 1:4,
   q=tempq(i);
   if i==1,
      QTX1 = ['--- q=',num2str(q)];
   elseif i==2,
      QTX2 = ['... q=',num2str(q)];
   elseif i==3,
      QTX3 = ['_._ q=',num2str(q)];
   else,
      QTX4 = [' _  q=',num2str(q)];
   end
   Q = Qe*1 + q^2*B*B';
   Kd=lqe(A,G,C,Q,Re);    % returns optimal kalman gains
   LL = Kd*C;             % lower left block of Ae
   LR = A-LL-B*L;
   Ae = [ A  UR
          LL LR];         % A extended
   [re,im] = nyquist(Ae,Be,Ce,D,1,w);

   if i==1,
      plot(re,im,'--')
   elseif i==2,
      plot(re,im,':')
   elseif i==3,
      plot(re,im,'-.')
   else,
        plot(re,im,'-')
     end
end
hold off;
xlabel('Real'),ylabel('Imaginary'); text(.05,.35,QTX1,'sc');
text(.05,.25,QTX2,'sc');text(.05,.15,QTX3,'sc');text(.05,.05,QTX4,'sc');
  % Discrete Nyquist Plot Program for AUV Dive System
  % filename = q_nyq.m
  % This program is used to determine the effect of varying q in the program
  % auvobs.m. This program performs the discrete Nyquist plot for a family of q's.

clear;
clg;
axis([-1.1 1.1 -1.1 1.1]);
axis('square');
dt=.25;
% Plant definitions
A=[-.7 -.04  0
    1   0    0
    0  -1    0];
B=[-.04; 0; 0];
C=[0 0 1];
s=[1; 0.7; -0.04];        % left eigenvector of A for lambda=0
D =[0];
Qe = [  .01     0     0
         0     .01    0
         0      0    .01];
   Re = [ 1 ];
   G  = 1;
   qc = [ 1   0   0
          0   1   0
          0   0   1];
   rc = [10];
   dt=.25;
 L=lqr(A,B,qc,rc);        % returns optimal feedback gains
 UR = zeros(3);           % upper right block of 'Ae' (A extended matrix)
 Be = [ B
       0
       0
       0];              % B extended
 Ce = [0 0 0 L];         % C extended

tempq=input('Input 4 values of q as a vector (ie. [ ])   ');
% Begin loop for family of curves

plot(exp(sqrt(-1)*[0:0.05:7]),'-');
hold on;
plot([-1 1],[0 0],'-');
for i = 1:4,
   q=tempq(i);
   if i==1,
     QTX1 = ['--- q=',num2str(q)];
   elseif i==2,
     QTX2 = ['... q=',num2str(q)];
   elseif i==3,
     QTX3 = ['_._ q=',num2str(q)];
   else,
     QTX4 = [' _  q=',num2str(q)];
   end
   Q = Qe*1 + q^2*B*B';
   Kd=lqe(A,G,C,Q,Re);    % returns optimal kalman gains
   LL = Kd*C;              % lower left block of Ae
   LR = A-LL-B*L;
   Ae = [ A  UR
         LL LR];          % A extended

% Begin discrete portion of program (ie. discretize Ae and Be)

   [PHIe,GAMe] = c2d(Ae,Be,dt);
   wr = logspace(-1.57,pi,250);
```

```
    [mag,phase] = dbode(PHIe,GAMe,Ce,D,1,wr);
    for k=1:length(mag),
        REAL(k)=mag(k)*cos(phase(k)*pi/180);
        IMAG(k)=mag(k)*sin(phase(k)*pi/180);
    end
    if i==1,
        plot(REAL,IMAG,'--')
    elseif i==2,
        plot(REAL,IMAG,':')
    elseif i==3,
        plot(REAL,IMAG,'-.')
    else,
        plot(REAL,IMAG,'-')
    end
end
%title('Discrete Nyquist Plot')
xlabel('Real'),ylabel('Imaginary');
grid;
text(.05,.35,QTX1,'sc');
text(.05,.25,QTX2,'sc');
text(.05,.15,QTX3,'sc');
text(.05,.05,QTX4,'sc');
hold off;
axis('normal');
axis;
```

```
  % State Feedback Steering Algorithm
  % filename = auvrud.m
  % AUV steering algorithim based on the feedback of current direction
  % and desired heading.
clear
%K = input('Input gain    ');
desired_head=input('Input desired heading    ');
                                K=.3;
                                kmax=300;
                                depth=0.0;
                                head=0;
                                rpm=500;
                                xdot=4.0;
   dt = .25;
   abspeed = xdot;
   pitchrate=0.0;
   pitch=0.0;
   yawrate=0.0;
   yaw=0.0;
   state = zeros(12,kmax);
   ruk = zeros(1,kmax);
   initial_state=[xdot;0;0;0;pitchrate;yawrate;0;head;depth;0;pitch;yaw];
% states are        1   2 3 4     5         6     7    8     9    10  11    12

   state(:,1) = initial_state;
%-------------------------------------- main loop -----------
  turn = desired_head-state(12,1);
  if turn > pi,
     fprintf('Turn is more than 180 degrees')
     desired_head = desired_head-(2*pi);
  end
  for j=1:kmax
      rud = -K*(desired_head-state(12,j));
      rudder = rud;
      if (abs(rudder) > .4  )
```

```
            rudder = 0.4*sign(rudder);
        end;
        divfin = 0;
        inputs=[rudder; divfin;-divfin;rpm];
        rud = rudder;
        state(:,j+1)=model(state(:,j),inputs,dt);

        ruk(j+1)=rud;

        abspeed=sqrt((state(1,j+1))^2+(state(2,j+1))^2+(state(3,j+1))^2);
        desired_head;
        state(12,j);
end             % end of main loop
 clg
time=0:kmax;
 subplot(211),plot(time,state(12,:)),title(' Model Head')
 subplot(212),plot(time,ruk),title(' Rudder Action')
text(.01,.03,'AUVRUD program with gain=.3','sc')
%meta pres
```

```
  %  Steering and Dive Simulation Program for the AUV
  %  filename = auv.m
  %  This program is the final phase test of the autopilot algorithm.
  %  It links the VSC, Doyle - Stein Observer, state feedback steering, and
  %  error detector components of the controller for full simulation.

clear
ERX=['Error induced in Pitch Rate at time intervals of 15 and 130 (p = 0.0)'];
desired_head=input('Input desired heading    ');
ordered_depth=input('Input ordered depth    ');
  q = 5 ;
  a=.04 ;
  b=.7  ;
  A  = [  -b  -a  0
          1   0  0
          0  -1  0];   % !!!!!! notice  the minus signs
 s=[1; 0.7; -0.04];  %  left eigenvector of A for lambda=0
  B  = [  -a;0;0];      % to get model states to track right
  C  = [ 0 0 1];
  D  = 0;
  Qe = [   .01      0     0
          0      .01     0
          0       0     .01];
  Re = [ 1 ];
  G  = 1;
  qc = [ 1    0    0
        0    1    0
        0    0    1];
  rc = [10];
  dt=.25;
  phi=eye(3)+A*dt
 del=B*dt
 L=lqr(A,B,qc,rc);        % returns optimal feedback gains
  Q = Qe*1 + q^2*B*B';
 Kd=lqe(A,G,C,Q,Re);      % returns optimal kalman gains
 Kd=Kd*dt;
% observer dynamics:
 phi0=phi-Kd*C;
 ac=poly(phi0);
                                        kmax=400;
                                       depth=100;
                                         rpm=500;
                                        xdot=4.0;
                                      thres1=.001;
                                      thres2=.01;
                                      thres3=.01;
                                           K=.3;
  abspeed = xdot;
  pitchrate=0.0;
  pitch=0.0;
  yawrate=0.0;
  yaw=0.0;
  r=[zeros(1,kmax)
     zeros(1,kmax)
     depth*ones(1,30) ordered_depth*ones(1,kmax-30)];
  state = zeros(12,kmax);
  xo = zeros(3,kmax);
  uk = zeros(1,kmax);
  ruk = zeros(1,kmax);
  error = zeros(3,kmax);
  auv_normalz = zeros(1,kmax);
  initial_state=[xdot;0;0;0;pitchrate;yawrate;0;0;depth;0;pitch;yaw];
% states are        1  2 3 4     5        6    7 8   9   10  11   12
  xo(:,1) = [pitchrate;pitch;depth/abspeed];      % initialize observer
  state(:,1) = initial_state;
  turn = desired_head-state(12,1);
  if turn > pi,
     fprintf('Turn is more than 180 degrees\n')
     desired_head = desired_head-(2*pi);
  end
%------------------------------------- main loop -----------

 for j=1:kmax-1

        u=L(3)*ordered_depth/abspeed;
        u=u-L*[state(5,j);state(11,j);state(9,j)/abspeed];
    z=state(9,j)-ordered_depth;
    q=state(5,j);
    th=state(11,j);

% check the error to the threshold
      if abs(error(1,j)) > thres1,
         p_rate=xo(1,j);
         fprintf('pitchrate is using obs\n')
      else
         p_rate=q;
      end
      if abs(error(2,j)) > thres2,
         pit = xo(2,j);
         fprintf('pitch is using obs\n')
      else
         pit = th;
      end
      if abs(error(3,j)) > thres3,
         norm_dep_err = xo(3,j);
         fprintf('normalized depth error is using obs\n')
      else
         norm_dep_err = z/abspeed;
      end

    ex=[p_rate; pit; norm_dep_err];       % setup to compute nonlin
    ste(j)=s'*ex;
    ubar=1.0*(ste(j));
    if (abs(ubar) > 0.4)
```

```
          ubar=0.4*sign(ste(j));
      end
      u=u+ubar;
      divfin = u;
      if (abs(divfin) > .4   )
          divfin = 0.4*sign(divfin);
      end;
      u = divfin;
      xo(:,j+1)= phi*xo(:,j) + del*u ;
      xo(:,j+1)= xo(:,j+1) + Kd*(z/abspeed - C*(xo(:,j)));

        % simulate a fault in q
          if j>15 & j<130,
               q=0.0;
          end
        %  end fault

% transient correction for the observer

      xe(:, j+1)=[q;th;z/abspeed]-xo(:,j);
      if j>3,
error(:,j+1)=ac(1)*xe(:,j+1)+ac(2)*xe(:,j)+ac(3)*xe(:,j-1)+ac(4)*xe(:,j-2);
      end
% Simple AUV steering algorithim
      rud = -K*(desired_head-state(12,j));
      rudder = rud;
      if (abs(rudder) > .4   )
          rudder = 0.4*sign(rudder);
      end;
      inputs=[rudder; divfin;-divfin;rpm];
      rud = rudder;
      state(:,j+1)=model(state(:,j),inputs,dt);

      ruk(j+1)=rud;
      uk(j+1)=u;
```

```
     abspeed=sqrt((state(1,j+1))^2+(state(2,j+1))^2+(state(3,j+1))^2);
end            % end of main loop

% Begin plotting portion
clg
subplot(211),plot(error(2,:)), title('Pitch Corrected Error')
subplot(212),plot(error(1,:)), title('Pitch Rate Corrected Error')
text(.03,.04,ERX,'sc');
%meta pres
pause
clg
subplot(211),plot(error(3,:)), title('Depth Corrected Error')
subplot(212),plot(state(5,:)),title('Pitch Rate')
text(.03,.04,ERX,'sc');
%meta pres
pause
clg
 subplot(211),plot(state(9,:)),title(' Model Depth')
 subplot(212),plot(uk),title(' Divefin Action')
 text(.03,.04,ERX,'sc');
%meta pres
% subplot(223),plot(state(5,:)),title('Pitch Rate')
% subplot(224),plot(xo(1,:)),title(' Observer Pitch Rate ')
 pause
clg
% subplot(221),plot(xo(3,:)),title(' Observer Depth, etc')
% subplot(222),plot(ste),title(' Sliding Surface')
% subplot(223),plot(state(11,:)),title('Pitch')
% subplot(224),plot(xo(2,:)),title(' Observer Pitch ')
subplot;
plot(ste),title(' Sliding Surface'),grid
text(.03,.04,ERX,'sc'); pause,clg
 subplot(211),plot(state(12,:)),title(' Model Head')
 subplot(212),plot(ruk),title(' Rudder Action')
text(.01,.03,'AUVRUD program with gain=.3','sc')
```

```
% filename = giclp.m
% This file plots the magnitude portion of the gic bandpass active filter.
% This was used in the determination of the required components of the
% active-RC filter discussed in Chapter 4.

% definitions
clear;
R=100e3;          % resistor values
G=1/R;
wt=2*pi*1e6;      % gain bandwidth product
%-----------------start loop-------------------
QP=.707;          % for maximally flat response
rq=G/QP;
C=.159e-6;        % Capcitor value
wc=1/(R*C);
%--------------set up transfer function---------
num1=(2*C*G^2)/w:
num2=(2*G^3+2*rq*G^2)/wt;
num3=2*G^3;
num=[num1 num2 num3];
den1=2*C^2*G/wt^2;
den2=(2*G*C^2)/wt+(4*G^2*C)/wt^2+(2*G*C*rq)/wt^2;
den3=C^2*G+2*G^3/wt^2+2*G^2*rq/wt^2+4*G^2*C/wt+2*G*C*rq/wt;
den4=2*G^3/wt+2*G^2*rq/wt+G*C*rq;
den5=G^3;
den=[den1 den2 den3 den4 den5];

w=logspace(-1,4,250);
[mag,phase]=bode(num,den,w);
% Begin plotting routine
axis([-1,2,-2,7]);
semilogx(w/(2*pi),20*log10(mag)),title('Magnitude Response (LPF) Fc=10 Hz')
xlabel('Frequency (in Hertz)'),ylabel('Magnitude (in dB)'),grid
text(.80,.03,'Q = .707','sc');
text(.05,.03,'Wt= 6.18e6','sc')
```

```
pause
axis([-1,3,-180,0]);
semilogx(w/(2*pi),phase)
title('Phase Response (LPF) Fc=10 Hz')
xlabel('Frequency (in Hertz)'),ylabel('Phase')
grid
```

APPENDIX B. REAL-TIME CONTROL CODE

This appendix contains the real-time autopilot code written in the "C" language. It has the following components: Variable-Structure Controller, Doyle-Stein Observer, error detector, and state feedback steering algorithms. This code is ready for implementation in the AUV. Also included in this appendix are two external functions called by the control routine, signum.c and matmul.c.

Particular attention should be given to the comments preceding the code and to the disclaimer. This code was written and compiled in Microsoft C 5.1, but is compiler independent (i.e., will compile on any ANSI standard C or K&R C compiler). A copy of this source code is available through Professor Roberto Cristi, Naval Postgraduate School (see Initial Distribution List).

```
/*
   AUV Autopilot Control Program          Dave Riling / Prof. Cristi
   This program makes use of a Non-Adaptive Variable Structure Algorithim
   with an associated Doyle-Stein Observer used for error checking in the
   diving mechanism. For the steering a simply Single State Feedback control
   scheme is used.
   The following values were used in the determination of the (1) feedback
   gains, (2) observer gains, (3) eigenvalues and (4) transient response
   coefficents :

   A = [ -.7  -.04  0     A matrix          B = [ -.04    B matrix
           1    0   0                              0
           0   -1   0 ]                            0 ]

   C = [ 0    0    1 ]  C matrix            D = [ 0 ]     D matrix

   The associated matlab program which was used to evaluate the above
   values has the filename : " auvobs.m ".
   The following values were obtained from the above program :
   (1) L = [ -3.6308 -2.8032  0.3162 ]
   (2) K = [ -0.0036 -0.0679  0.1859 ]
   (3) S = [  1.0
                 0.7
                -0.04 ]
   (4) Ac = [ 1.0    -2.6391  2.3302  -0.6874 ]

   The threshold levels used in the sensor error detection algorithim were
   determined using a hueristically graphic approach.
   threshold_1 = .003    pitch rate threshold error
   threshold_2 = .05     pitch threshold error
   threshold_3 = .2    normalized depth threshold error

   The feedback gain used in the steering algorithim is " Ks = .175 ". This
   value was determined using a hueristic approach.
```

```
DISCLAIMER: Once the actual AUV model has been determined the matlab
file " auvobs.m " should be changed and run to obtain the true values
required for the actual AUV. Note this algorithim is designed to be
a Real Time Implementation and is very Robust; as such, it should not
require any changes other than the above parmeters if that.

                    State Definitions
                    mstate[0] = xdot
                    mstate[1] = ydot
                    mstate[2] = zdot
                    mstate[3] = rollrate
                    mstate[4] = pitchrate
                    mstate[5] = yawrate
                    mstate[6] = x
                    mstate[7] = y
                    mstate[8] = z
                    mstate[9] = roll
                    mstate[10] = pitch
                    mstate[11] = yaw

*/

#include <math.h>
#include <stdio.h>
#include "auv.h"


#define RUD_STOPS  0.4
#define PI         3.141593
#define TWO_PI     6.283186
#define YES        1
#define NO         0

static double
          ex[]   = { 0.0, 0.0, 0.0 },
                    L[]     = { -3.6308, -2.8032, 0.3162 },
                    state[] = { 0.0, 0.0, 0.0 };

void  control8( double *mstate, double *inputs,
                      double ordered_depth, double desired_head, double rpm )
{
  int      order_change;
           jump_over = 0;
  double   bowplane,
           sternplane,
           rudder;
  double   q,
           theta,
           divres,
           obsres,
           abspeed,
            errz,
           divfin1,
           divfin,
           required_turn,
           Ks = 0.175;
  static double
           old_ordered_depth = 0.0;

  /* Now read speed sensor     */
  abspeed = mstate[0];
  q = state[0] = mstate[4];
  theta = state[1] = mstate[10];
  state[2] = mstate[8]/abspeed;
  divfin1 = L[2] * ordered_depth/abspeed;
  mat_mul(L, 1, 3, state, 3, 1, (Matrix)&divfin);
  divres = divfin1 - divfin;
  if (ordered_depth != old_ordered_depth) {
     ordered_change = YES;  /* set flag to indicate a change in ordered_depth) */
  }
```

```
else {
   ordered_change = NO;
}
old_ordered_depth = ordered_depth;
errz = (mstate[8] - ordered_depth)/abspeed;
ex[0] = mstate[4];
ex[1] = mstate[10];
ex[2] = errz;

/*---- Call Diveplane Function ----*/
diveplane(ex, &divres);
if (observer(q, theta, errz, divres, order_change, &obsres)) {
      sternplane = obsres;
      bowplane = -sternplane;
}
else {
      sternplane = divres;
      bowplane = -divres;
}

/*-----------start steering algorithim-------------------------*/
if (desired_head > TWO_PI) {
      printf(" Error in desired heading coming from mission commander\n");
      printf(" AUV surfacing and slowing to 150 RPM, rudder 0 deflection\n");
      sternplane = RUD_STOPS;
      bowplane = -RUD_STOPS;
      rpm = 150;
      rudder = 0;
      jump_over = 1;
}
if (mstate[11] > TWO_PI) {
      printf(" Error in directional gyro\n");
      printf(" AUV surfacing and slowing to 150 RPM, rudder 0 deflection\n");
      sternplane = RUD_STOPS;
      bowplane = -RUD_STOPS;
```

```
        rpm = 150;
        rudder = 0;
        jump_over = 1;
    }
    if (jump_over != 1) {
        required_turn = desired_head - mstate[11];   /* Make the shortest turn */
        if (required_turn > PI) {
            desired_head = desired_head - (2*PI);
        }
        rudder = -Ks * (desired_head - mstate[11]);
        if (fabs(rudder) > RUD_STOPS)
            rudder = RUD_STOPS * sign(rudder);
    }
    /*--movedata ???????????????--------------------------*/
    inputs[0] = rudder;
    inputs[1] = sternplane;
    inputs[2] = bowplane;
    inputs[3] = rpm;
}

/**********************************************************************
****
        This function determines the observer states and also performs
        error & threshold comparisons to check sensor validity. The arguments
        are : pitchrate, pitch, normalized error in depth, current divefin
        value, state of order_change, and observer divefin value
*/
int     observer(double q, double theta, double errz,
                        double divres, int order_change, double *obsres)
{
#define No_Error        0
#define Sensor_Error    1
#define Major_Error     5

        int i, flag;
    static int
          transient_hold = 3;
          j = 0;

    double
         PHI[] =
              { 0.8250,   -0.01,   0.0,
                0.2500,    1.00,   0.0,
                0.0,      -0.25,   1.0 };
    double
         DEL[] = { -0.01,    0.0,     0.0 },
         C[]   = {  0.0,     0.0,     1.0 },
         K[]   = { -0.0036, -0.0679,  0.1859 },
         Error[] = { 0.0,   0.0,   0.0 };

    static double
         obserr[] = { 0.0,   0.0,   0.0 },
         obs1err[] = { 0.0,   0.0,   0.0 },
         obs2err[] = { 0.0,   0.0,   0.0 },
         obs3err[] = { 0.0,   0.0,   0.0 },
         old_obst[] = { 0.0,   0.0,   0.0 },
         obstate[] = { 0.0,   0.0,    0.0 };
    double  cobs,
              comp,
              threshold_1 = .003,    /* pitch rate threshold error */
              threshold_2 = .05,    /* pitch threshold error */
              threshold_3 = .2,     /* normalized depth threshold error */
              A1 = 1.0,            /* transient error coefficents */
              A2 = -2.6391,
              A3 = 2.3302,
              A4 = -0.6874;

mat_mul(PHI, 3, 3, old_obst, 3, 1, obstate);
DEL[0] *= divres;
DEL[1] *= divres;
```

```
DEL[2] *= divres;

mat_mul(C, 1, 3, old_obst, 3, 1, (Matrix)&cobs);
comp = errz - cobs;
for (i = 0; i < 3; i++) {
        obstate[i] += DEL[i];
        K[i] *= comp;
        obstate[i] += K[i];
        old_obst[i] = obstate[i];
}
/*---Correct for Transient Response---*/
obserr[0] = q - obstate[0];
obserr[1] = theta - obstate[1];
obserr[2] = errz - obstate[3];
if (order_change == YES)
     transient_hold=j+3;
if (j > transient_hold) {
     for (i = 0; i < 3; i++) {
          Error[i] = A1*obserr[i] + A2*obs1err[i] + A3*obs2err[i] + A4*obs3err[i];
     }
}
j++;
for (i = 0; i < 3; i++) {
      obs3err[i] = obs2err[i];
      obs2err[i] = obs1err[i];
      obs1err[i] = obserr[i];
}

/*---Compare the error to the Threshold values---*/
flag = 0;
if (fabs(Error[0]) > threshold_1) {
     printf(" Error[0] = %lf \n",Error[0]);
     ex[0] = obstate[0];
     flag = Sensor_Error;
}
```

```
    if (fabs(Error[1]) > threshold_2) {
        printf(" Error[1] = %lf \n",Error[1]);
        ex[1] = obstate[1];
        flag = Sensor_Error;
    }
    if (fabs(Error[2]) > threshold_3) {
        /* Big Problem depth sensor is  malfunctioning  SURFACE SURFACE !! */
        printf(" Error[2] = %lf \n",Error[2]);
        *obsres = .4;               /* Full diveplane deflection */
        return Major_Error;         /* Depth Sensor error code    */
    }
    if (flag == 0) {
        return No_Error;   /* Sensor readings appear normal do not use observer */
    }
    else {
        diveplane(ex, &divres);
        *obsres = divres;
        return Sensor_Error;
    }
}

/************************************************************************
****
    Dive plane function determines the appropriate radian angle for the
    dive fins. The arguments are : ex = state error vector, divefin value
*/
void    diveplane(double *ex, double *divres)
{
    static double  S[] = { 1.0, 0.7, -0.04 };
    double   nonlin,
                   ste,
                   stesign,
                   diver,
                   dive;
    diver = *divres;
    mat_mul(S, 1, 3, ex, 3, 1, (Matrix)&ste);
    nonlin = ste;
    stesign = sign(ste);
    /*--Saturation Switch for Nonlinearity------------------*/
    if (fabs(nonlin) > 0.4) {
        nonlin = 0.4 * stesign;
    }
    diver += nonlin;
    dive = sign(diver);              /*--Utilize Signum Function--*/
    if (fabs(diver) > RUD_STOPS) {
        diver = RUD_STOPS * dive;
    }
    *divres = diver;
}
```

```
/*
 *   AUV.H
 *
 *      Header file for the AUV project.
 *      Associated Files : 'auvs.c' 'autplt.c' 'matmul.c' 'signum.c'
 *                         'model.c' 'modelprm.h'
 */

typedef double   *Matrix;

void  mat_mul(Matrix M1, int r1, int c1, Matrix M2, int r2, int c2, Matrix M3);
void  control8( double *mstate, double *inputs,
                       double ordered_depth, double desired_head, double rpm );
int   model( double *oldstate, double *inputs, double *dt, double *mstate );
double  sign( double argument );
int   observer(double q, double theta, double errz,
                       double divres, int order_change, double *obsres);
void    diveplane(double *ex, double *divres);
```

```
/*
 *                  signum.c
 *   This is a generic signum function for use in the A.U.V. project.
 *   The function accepts an argument, determines the sign of the argument,
 *   and returns the sign (+1 or -1).
 */

double  sign( double argument )
{
  return  (argument > 0.0 ? 1.0 : -1.0);
}
```

```
/*
 *                mat_mul.c
 *  This function multiples two matrices or vectors (or scalar, matrix mult.)
 *  and returns the product, typical usages:
 *  A=[1 2 3] B=[4; 5; 6] then mat_mul(A,1,3,B,3,1,(Matrix)&scalar_ans)
 *    A=[1 2 3  B=[1 2 3
 *       4 5 6     4 5 6    then mat_mul(A,3,3,B,3,3,matrix_ans)
 *       7 8 9]    7 8 9]
 *  This function is called by autplt.c, the autopilot program for the AUV
 *  project.                   Dave Riling    16 Jan 90
 */

#include <stdio.h>

typedef double *Matrix;
/* Row major access macro */
#define     Xrm(M,row,col,col_len)     (*(M +(row * col_len) + col))
void mat_mul(Matrix M1, int r1, int c1, Matrix M2, int r2, int c2, Matrix M3)
{
	int     i,j,k;
	double  sum;
/* Take this out when code is implemented in the auv */
	if (c1 != r2) {
		fprintf(stderr,"mat_mul: matrices rows/cols not compatible\n");
		exit(1);
	}
	for (i=0; i < r1; i++)
		for (j=0; j < c2; j++) {
			sum = 0.0;
			for(k=0; k < c1; k++) {
				sum += (Xrm(M1,i,k,c1) * Xrm(M2,k,j,c2));
			}
		Xrm(M3,i,j,r1) = sum;
		}
}
```

APPENDIX C. INTERFACE AND MODEL PROGRAMS

Included in this appendix are the interface module and the dynamic equations of motion model both written in "C" for the AUV. Also included is the header file for the model. The code in this appendix is not complier specific, and a copy can be obtained by contacting Professor Roberto Cristi, Naval Postgraduate School (see Initial Distribution List).

The model in "C" calls one external function, signum.c, which is located in Appendix B. This model updates the 12 states depicted in Table 2. of this thesis using the dynamic equations of motion modeled after the U.S. Navy's Swimmer Delivery Vehicle. The interface program calls th ıodel after the controller modifies the necessary control surface commands (i.e., Dive plane, rudder, and RPM).

```
/*
 *   MODEL.C
 *           function  state = model(oldstate,inputs,dt)
 *
 */


#include <math.h>
#include <stdio.h>
#include "modelprm.h"

double  sign(double);

int  model(double *oldstate, double *inputs, double *dt, double *mstate)
{
   int       j, k;
   double   u, v, w, p, q, r, phi, theta, psi;
   double   dr, ds, db, rpm, delt;
   double   mass, latyaw, norpit, re, term0;
   double   signu, signn, eta, cd0, ct, ct1, eps, xprop;
   double   ucf[4], fp[6], f[12];
   double   tmp1, tmp2, tmp3, tmp4;
   double   cos_theta, sin_theta, tan_theta;
   double   cos_phi, sin_phi, cos_psi, sin_psi;

   u      = oldstate[0];
   v      = oldstate[1];
   w      = oldstate[2];
   p      = oldstate[3];
   q      = oldstate[4];
   r      = oldstate[5];
   phi    = oldstate[9];
   theta = oldstate[10];
   psi    = oldstate[11];
```

```
  dr  = inputs[0];
  ds  = inputs[1];
  db  = inputs[2];
  rpm = inputs[3];

  delt = *dt;

  latyaw = norpit = 0.0;
  mass = weight/g;
  re = u*l/nu;

  signu = sign(u);
  signn = sign(rpm);
  if (fabs(u) < x1test)
     u = x1test;
  eta = 0.012*rpm/u;
  re = u*l/nu;
  cd0 = 0.00385 + 1.296e-17 * (re - 1.2e7)*(re - 1.2e7);
  ct1 = 0.008*l*l/a0;
  ct  = ct1*eta*fabs(eta);
  eps = -1.0+signn/signu*(sqrt(ct+1.0)-1.0)/(sqrt(ct1+1.0)-1.0);
  xprop = cd0*(eta*fabs(eta) - 1.0);

/*
 *   calculate the drag force, integrate the drag over the vehicle
 *   integrate using a 4 term gauss quadrature
 */

   for (k=0; k<4; ++k) {
      tmp1 = v+g4[k]*r*l;
      tmp2 = w-g4[k]*q*l;
      ucf[k] = sqrt(tmp1*tmp1 + tmp2*tmp2);
      if(1.0e-10 <= ucf[k]) {
         term0  = ((rho/2.0)*(cdy*hh[k]*tmp1*tmp1
                   + cdz*br[k]*tmp2*tmp2)) *gk4[4]*l/ucf[k];
```

```
            latyaw += term0*tmp1;
            norpit += term0*tmp2;
          }
        }

/*
 *   force equations
 */

/*
 *   common sub-expressions
 */

    tmp1 = (rho/2.0)*l*l;
    tmp2 = tmp1*l;
    tmp3 = tmp2*l;
    tmp4 = tmp3*l;
    cos_theta = cos(theta);
    sin_theta = sin(theta);
    tan_theta = sin_theta/cos_theta;
    cos_phi = cos(phi);
    sin_phi = sin(phi);
    cos_psi = cos(psi);
    sin_psi = sin(psi);


/*
 *   longitudinal force
 */

    fp[0] = mass*v*r - mass*w*q + mass*xg*q*q
       + mass*xg*r*r - mass*yg*p*q - mass*zg*p*r
       + tmp3*(xpp*p*p+xqq*q*q + xrr*r*r+xpr*p*r)
       + tmp2*(xwq*w*q+xvp*v*p+xvr*v*r+u*q*(xqds*ds+xqdb*db)+xrdr*u*r*dr)
       + tmp1*(xvv*v*v+xww*w*w + xvdr*u*v*dr+u*w*(xwds*ds+xwdb*db)
        + u*u*(xdsds*ds*ds+xdbdb*db*db+xdrdr*dr*dr))
        - (weight -boy)*sin_theta
        + tmp2*xqdsn*u*q*ds*eps
        + tmp1*(xwdsn*u*w*ds+xdsdsn*u*u*ds*ds)*eps
        + tmp1*u*u*xprop;

/*
 *   lateral force
 */

    fp[1] = -mass*u*r - mass*xg*p*q + mass*yg*r*r - mass*zg*q*r +
        tmp3*(ypq*p*q + yqr*q*r)+tmp2*(yp*u*p +
        yr*u*r + yvq*v*q + ywp*w*p + ywr*w*r) + tmp1*
        (yv*u*v + yvw*v*w +ydr*u*u*dr) -latyaw +(weight-boy)*
        cos_theta*sin_phi+mass*w*p+mass*yg*p*p;

/*
 *   normal force
 */

    fp[2] = mass*u*q - mass*v*p - mass*xg*p*r - mass*yg*q*r +
        mass*zg*p*p + mass*zg*q*q + tmp3*
        (zpp*p*p+zpr*p*r + zrr*r*r) + tmp2*(zq*u
        *q+zvp*v*p + zvr*v*r) +tmp1*(zw*u*w + zvv*v*v
        +u*u*(zds*ds+zdb*db))-norpit+(weight-boy)*cos_theta*cos_phi
        +tmp2*zqn*u*q*eps +tmp1*(zwn*u*w +zdsn*
        u*u*ds)*eps;
```

```
/*
 *   roll force
 */

   fp[3] = -iz*q*r +iy*q*r -ixy*p*r +iyz*q*q -iyz*r*r +ixz*p*q+
      mass*yg*u*q -mass*yg*v*p -mass*zg*w*p+tmp4*(kpq*
      p*q + kqr*q*r) +tmp3*(kp*u*p +kr*u*r + kvq*v*q +
      kwp*w*p + kwr*w*r) +tmp2*(kv*u*v + kvw*v*w) +
      (yg*weight - yb*boy)*cos_theta*cos_phi - (zg*weight -
      zb*boy)*cos_theta*sin_phi + tmp3*kpn*u*p*eps+
      tmp2*u*u*kprop +mass*zg*u*r;

/*
 *   pitch force
 */

   fp[4] = -ix*p*r +iz*p*r +ixy*q*r -iyz*p*q -ixz*p*p +ixz*r*r-
      mass*xg*u*q + mass*xg*v*p + mass*zg*v*r - mass*zg*w*q +
      tmp4*(mpp*p*p +mpr*p*r +mrr*r*r)+tmp3*(mq*u*q + mvp*v*p + mvr*v*r) +
      tmp2*(mw*u*w+mvv*v*v+u*u*(mds*ds+mdb*db))+ norpit -(xg*weight-
      xb*boy)*cos_theta*cos_phi+tmp3*mqn*u*q*eps +
      tmp2*(mwn*u*w+mdsn*u*u*ds)*eps-
      (zg*weight-zb*boy)*sin_theta;

/*
 *   yaw force
 */

   fp[5] = -iy*p*q +ix*p*q +ixy*p*p -ixy*q*q +iyz*p*r -ixz*q*r-
      mass*xg*u*r + mass*xg*w*p - mass*yg*v*r + mass*yg*w*q +
      tmp4*(npq*p*q + nqr*q*r) +tmp3*(np*u*p+
      nr*u*r + nvq*v*q +nwp*w*p + nwr*w*r) +tmp2*(nv*
      u*v + nvw*v*w + ndr*u*u*dr) - latyaw + (xg*weight -
      xb*boy)*cos_theta*sin_phi+(yg*weight)*sin_theta
      +tmp2*u*u*nprop-yb*boy*sin_theta;
```

```
/*
 *   now compute the f(0-5) functions
 */

  for (j=0; j<6; ++j)
     for (f[j]=0.0,k=0; k<6; ++k)
        f[j] += xmminv[j][k]*fp[k];

/*
 *   the last six equations come from the kinematic relations
 */

/*
 *   inertial position rates f(6-8)
 */

  f[6] =  u*cos_psi*cos_theta + v*(cos_psi*sin_theta*
     sin_phi - sin_psi*cos_phi) + w*(cos_psi*sin_theta*
     cos_phi + sin_psi*sin_phi);

  f[7] =  u*sin_psi*cos_theta + v*(sin_psi*sin_theta*
     sin_phi + cos_psi*cos_phi) + w*(sin_psi*sin_theta*
     cos_phi - cos_psi*sin_phi);

  f[8] = -u*sin_theta +v*cos_theta*sin_phi +w*cos_theta*cos_phi;

/*
 *   euler angle rates f(9-11)
 */

  f[9] = p + q*sin_phi*tan_theta + r*cos_phi*tan_theta;
  f[10] = q*cos_phi - r*sin_phi;
  f[11] = q*sin_phi/cos_theta + r*cos_phi/cos_theta;
```

```
/*
 *   first order integration
 */

  for (j=0; j<12; j++)
       mstate[j] = oldstate[j] + delt * f[j];

  return  0;
}
```

```
/*
 *   modelprm.h
 *
 *      This file contains all of the parameter coefficients
 *   used by the file MODEL.C.
 *
 */


/*
 *   longitudinal hydrodynamic coefficients
 */

const double
   xpp =    7.0e-3,  xqq =   -1.5e-2,  xrr =    4.0e-3,  xpr =    7.5e-4,
   xudot = -7.6e-3,  xwq =   -2.0e-1,  xvp =   -3.0e-3,  xvr =    2.0e-2,
   xqds =   2.5e-2,  xqdb =  -2.6e-3,  xrdr =  -1.0e-3,  xvv =    5.3e-2,
   xww =    1.7e-1,  xvdr =   1.7e-3,  xwds =   4.6e-2,  xwdb =   1.0e-2,
   xdsds = -1.0e-2,  xdbdb = -8.0e-3,  xdrdr = -1.0e-2,  xqdsn =  2.0e-3,
   xwdsn =  3.5e-3,  xdsdsn = -1.6e-3;


/*
 *   lateral hydrodynamic coefficients
 */

const double
   ypdot =  1.2e-4,  yrdot =  1.2e-3,  ypq =    4.0e-3,  yqr =   -6.5e-3,
   yvdot = -5.5e-2,  yp =     3.0e-3,  yr =     3.0e-2,  yvq =    2.4e-2,
   ywp =    2.3e-1,  ywr =   -1.9e-2,  yv =    -1.0e-1,  yvw =    6.8e-2,
   ydr =    2.7e-2,  cdy =    3.5e-1;
```

```
/*
 *   normal hydrodynamic coefficients
 */

const double
   zqdot = -6.8e-3,  zpp =   1.3e-4,  zpr =   6.7e-3,  zrr =  -7.4e-3,
   zwdot = -2.4e-1,  zq =   -1.4e-1,  zvp =  -4.8e-2,  zvr =   4.5e-2,
   zw =    -3.0e-1,  zvv =  -6.8e-2,  zds =  -7.3e-2,  zdb =  -2.6e-2,
   zqn =   -2.9e-3,  zwn =  -5.1e-3,  zdsn = -1.0e-2,  cdz =   1.0;


/*
 *   roll hydrodynamic coefficients
 */

const double
   kpdot = -1.0e-3,  krdot = -3.4e-5,  kpq =  -6.9e-5,  kqr =   1.7e-2,
   kvdot =  1.3e-4,  kp =   -1.1e-2,  kr =   -8.4e-4,  kvq =  -5.1e-3,
   kwp =   -1.3e-4,  kwr =   1.4e-2,  kv =    3.1e-3,  kvw =  -1.9e-1,
   kpn =   -5.7e-4,  kdb =    0.0;


/*
 *   pitch hydrodynamic coefficients
 */

const double
   mqdot = -1.7e-2,  mpp =   5.3e-5,  mpr =   5.0e-3,  mrr =  -2.9e-3,
   mwdot = -6.8e-3,  mq =   -6.8e-2,  mvp =   1.2e-3,  mvr =   1.7e-2,
   mw =     1.0e-1,  mvv =  -2.6e-2,  mds =  -4.1e-2,  mdb =   6.9e-3,
   mqn =   -1.6e-3,  mwn =  -2.9e-3,  mdsn = -5.2e-3;
```

```
/*
 *   yaw hydrodynamic coefficients
 */

const double
  npdot = -3.4e-5,  nrdot = -3.4e-3,  npq =   -2.1e-2,  nqr =    2.7e-3,
  nvdot =  1.2e-3,  np =    -8.4e-4,  nr =    -1.6e-2,  nvq =   -1.0e-2,
  nwp =   -1.7e-2,  nwr =    7.4e-3,  nv =    -7.4e-3,  nvw =   -2.7e-2,
  ndr =   -1.3e-2;



/*
 *   mass characteristics of the flooded vehicle
 */

const double
  weight = 12000.0, boy =    12000.0, vol =    200.0,  xg =     0.0,
  yg =      0.0,    zg =     0.20,    xb =     0.0,    zb =     0.0,
  ix =     1500.0,  iy =     10000.0, iz =     10000.0, ixz =  -10.0,
  iyz =   -10.0,    ixy =   -10.0,    yb =     0.0,
  l =      17.4,    rho =    1.94,    g =      32.2,    nu =     8.47e-4,
  a0 =      2.0,    kprop =  0.0,     nprop =  0.0,     x1test = 0.1,
  degrud = 0.0,     degstn = 0.0;


/*
 *   define length fractions for gauss quadrature terms
 */

const double
  g4[]  = { 0.069431844, 0.330009478, 0.669990521, 0.930568155 },
  gk4[] = { 0.1739274225687, 0.3260725774312, 0.3260725774312,
            0.1739274225687 };
```

```
/*
 *   define the breadth bb and height hh terms for the integration
 */

const double
   br[] = { 75.7/12.0, 75.7/12.0, 75.7/12.0, 55.08/12.0 },
   hh[] = { 16.38/12.0, 31.85/12.0, 31.85/12.0, 23.76/12.0 };




/*
 *   assemble inverted mass matrix
 */

const double
   xmminv[6][6] = {
      { 0.2431e-2, 0.2701e-8, 0.1899e-5, 0.1649e-7, -0.5023e-5,  0.3243e-8 },
      { 0.2679e-8, 0.1537e-2, 0.5593e-8, 0.4276e-4, -0.1479e-7,  0.1057e-4 },
      { 0.1899e-5, 0.5639e-8, 0.6293e-3, 0.3443e-7, -0.1049e-4,  0.6770e-8 },
      { 0.1649e-7, 0.4321e-4, 0.3443e-7, 0.3294e-3, -0.9106e-7, -0.1049e-5 },
      { -.5023e-5, -.1491e-7, -.1049e-4, -.9106e-7,  0.2773e-4, -0.1790e-7 },
      { 0.3243e-8, 0.1057e-4, 0.6769e-8, -.1052e-5, -0.1790e-7,  0.6561e-4 }
   };
```

APPENDIX D. FIELD EVALUATION GRAPHICS ROUTINES

This appendix contains the routines written for field graphical analysis of AUV run data. These programs are not intended to produce hardcopy output, but are merely for on-the-spot interpretation of AUV run data. They are user friendly and will operate on many different graphic systems.

Particular attention should be paid to the disclaimer in the comment sections of the source code. These programs <u>are compiler dependent</u>. They call many nonstandard C functions in Libraries specific to <u>MICROSOFT C 5.1</u> or <u>QUICKC 2.0</u>. These functions are necessary for the graphics routines. This code must be compiled using either of the above compilers.

A copy of this source code is available through Professor Roberto Cristi, Naval Postgraduate School (see Initial Distribution List). <u>MICROSOFT C 5.1</u> and <u>QUICKC 2.0</u> are registered trademarks of the Microsoft Corporation. The code contained in the libraries called by this source code is available through Microsoft Corporation and its affiliated vendors. The U.S. Navy does not support the use of any particular C compiler; C compiler used was solely the preference of the author.

```
/*
                          plot.c
   This program takes a columnized data file (x, y) and plots each column
   with respect to discrete time. It autoscales and labels the graph
   appropriately using the tabular headings created in the "auvs.c" program.
   This file is intended for field use during AUV test. Usage:
               plot <filename>   ie. plot auvrun.dat
   This program is composed of : grph.h  setvid.c.
   The setvid.c function sets the program for the correct graphics adapter
   installed (note: please see setvid.c for more information.
   IMPORTANT: The first column of the data file must always be time. This
   column is always the x-axis on the graphs. For a hardcopy output the only
   facility is the use of print screen, since this program was note intended
   for hardcopy output. There is no limitation on the size of the file or the
   number of columns of data it may contain. The larger the file the longer
   the required run time.
   Final note: These files have library functions particular to Microsoft 5.1
   or Quick C  2.0 and as such must be compiled using either of these
   compilers.
*/

#include "grph.h"
#include <malloc.h>


#define  BUFFER  256


float      **bigarray;

struct text
{
     char title[30],                /* structure used for labeling graph */
     subtitle[30];
} label;
```

```
main( int argc, char **argv )
{
    chartenv    env;        /*  setup chart enviroment structure */
    int         i,j,k,m,n;
    FILE          *fptr;
    char        headingbuf[BUFFER+1],
                    databuf[BUFFER+1],
                    gridquest[3],
                    *label_axis[15];
    char        *tptr;

    if ((fptr = fopen(argv[1],"rt")) == NULL) {
        perror(argv[1]);
        exit(1);
    }
    else  {
        fgets (headingbuf,BUFFER,fptr);             /* get the heading */
        n=0;
    tptr = headingbuf;
        while ((strtok (tptr," ")) != NULL) {
            n++;
            tptr = NULL;
        }
        m=0;
        while (fgets (databuf,BUFFER,fptr) != NULL)  /* get the data */
            m++;
        rewind(fptr);
        for (k=0; k < n; k++) {
            fscanf(fptr,"%s",label_axis[k]);      /* discard the headings */
        }
    }

    if ((bigarray = calloc (n,sizeof(float*))) == NULL) {
        printf (" Something is wrong with memory cannot initialize array\n");
        exit(1);
    }
    for (i=0; i < n; i++) {
        if ((bigarray[i] = calloc(m,sizeof(float))) == NULL) {
            printf (" Check memory cannot load data for array\n");
            exit(1);
        }
    }

    /* Begin graphics set up */
    for (i=0; i < m; i++) {
        for (j=0; j < n; j++) {
            fscanf (fptr,"%f",&bigarray[j][i]);
        }
    }
    puts("Enter title of the graph ( limit 30 characters )");
    gets(label.title);
    puts("Enter the subtitle of the graph ( limit 30 characters )");
    gets(label.subtitle);

    /* Begin graphics hardware interface */
    graphics_mode();                /* call the setvid.c file */
    _clearscreen( _GCLEARSCREEN );

    _pg_initchart();
    _pg_defaultchart( &env, _PG_SCATTERCHART, _PG_POINTANDLINE );
    strcpy( env.maintitle.title, label.title );
    env.maintitle.titlecolor = 1;        /* color 1 = white,text color */
    env.maintitle.justify = _PG_CENTER;
    strcpy( env.subtitle.title, label.subtitle),
    env.subtitle.titlecolor = 1;         /* color 1 = white,text color */
    env.subtitle.justify = _PG_CENTER;
    strcpy( env.xaxis.axistitle.title, "Time");
    env.chartwindow.border = TRUE;
```

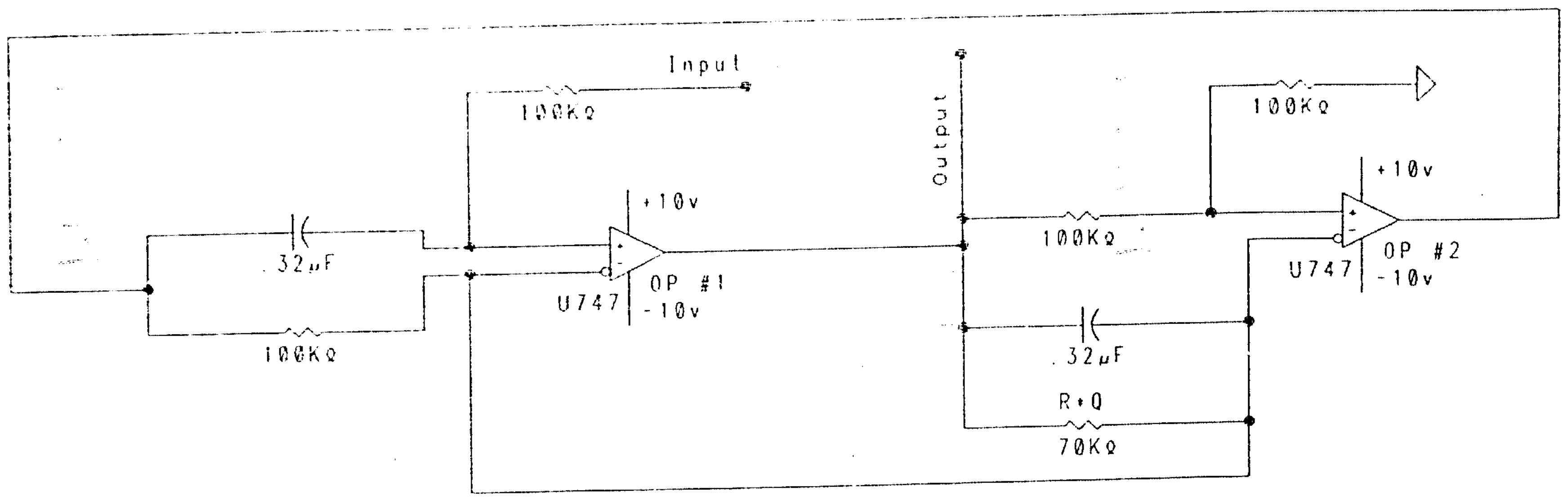
```
        _pg_getpalette( pal );          /* Set chart point value to = blank */
        pal[1].plotchar = 32;
        _pg_setpalette( pal );
jmpbak:
        puts("Do you want grids on your graph (y or n)?");
        gets (gridquest);
        switch (*gridquest) {
             case 'y':
                  env.yaxis.grid = TRUE;
                  env.xaxis.grid = TRUE;
                  env.yaxis.gridstyle = 4;
                  env.xaxis.gridstyle = 5;
             break;
             case 'n':
                  env.yaxis.grid = FALSE;
                  env.xaxis.grid = FALSE;
             break;
             default:
                  printf("Try again. Input lowercase y or n only! \n");
                  goto  jmpbak;
        }

        /* Begin plotting values */
        for (i=1; i < n; i++) {
             strcpy( env.yaxis.axistitle.title, label_axis[i]);
             _pg_chartscatter( &env, bigarray[0], bigarray[i], m);
             getch();
        }
        _setvideomode( _DEFAULTMODE );       /* reset the video back to original */
                                   /* setting prior to running plot.exe*/
        for (i=0; i < n; i++)
            free(bigarray[i]);
        free(bigarray);
        fclose(fptr);
}
```

```
/*
                         setvid.c
  This function sets the video graphics mode for the graphing routine:
  plot. It checks for the graphics-adapter card that is present in the
  respective computer and sets it automatically to its highest resolution.
  This function is called by "plot.c". It returns a void and accepts no
  arguments.Graphics cards that are accepted by this program are:
       1) VGA  2) EGA  3) CGA 4) HERCULES.
  DISCLAIMER: Due to the configuration of the standard Microsoft 'C'
  libraries used, this function will always set the graphics card present
  to its highest resolution available regardless of the monitor in use.
  Therefore, if you have a VGA graphics card installed and are using a
  EGA monitor the "plot.exe" program will not work.
*/
#include "grph.h"
void graphics_mode( void )
{
      _getvideoconfig( &myscreen );
      switch( myscreen.adapter ) {
            case _CGA:
            case _OCGA:
                  _setvideomode( _HRESBW );
                  break;
            case _EGA:
            case _OEGA:
                  _setvideomode( _ERESCOLOR );
                  break;
            case _VGA:
            case _OVGA:
            case _MCGA:
                  _setvideomode( _VRES16COLOR );
                  break;
            case _HGC:
                  _setvideomode( _HERCMONO );
                  break;
            default:
                  printf( "This program requires a graphics card.");
                  exit(0);
      }
      _getvideoconfig( &myscreen );
      maxx = myscreen.numxpixels - 1;      /* this information is not used */
      maxy = myscreen.numypixels - 1;      /* currently, it is used for    */
}                                    /* manual axis scaling         */




/*
                        grph.h
      This is the header file for the program "graph.exe".
      Associated files are : 'plot.c' and 'setvid.c'.
*/


#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <math.h>

typedef enum {FALSE, TRUE} boolean;
void graphics_mode( void );
struct videoconfig myscreen;
int maxx, maxy;
palettetype pal;
```

APPENDIX E. HARDWARE DESIGN SCHEMATICS

The diagrams in this appendix pertain to the designs discussed in Chapter IV. of this thesis. These diagrams are of the active-RC filter circuit and the Synchro-to-Resolver circuit. For explanation and positioning of these curcuits, please refer to Chapter IV. These circuits are intended for use in the signal conditioning network of the AUV. For further assistance refer to the technical notes listed in the references. These notes are available in the AUV library.

A.U.V. ACTIVE FILTER SIGNAL CONDITIONING MODULE

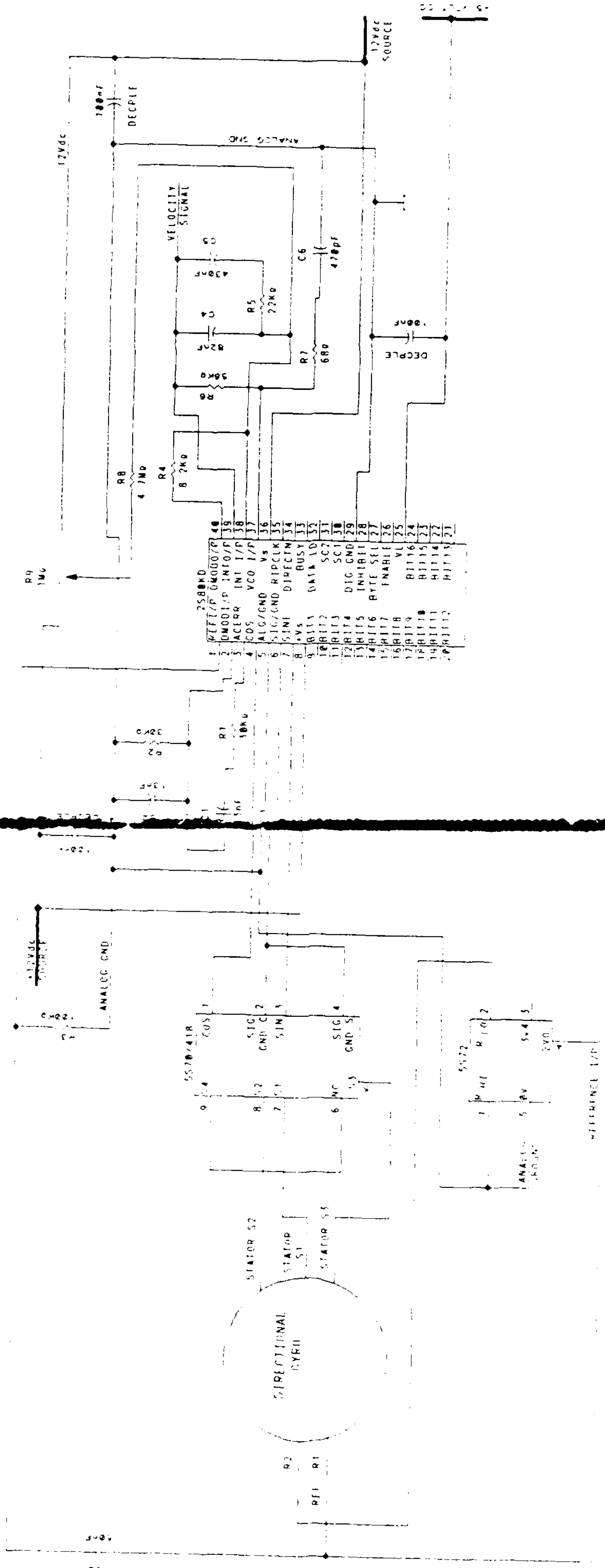


GENERALIZED IMPEDANCE
CONVERTER

MAXIMALLY FLAT ANALOG FILTER

NOTE: THIS IS AN ENLARGEMENT OF A FILTER ON THE SIGNAL CONDITIONING MODULE CIRCUIT BOARD. THE CIRCUIT IS DESIGNED TO HAVE A CUTOFF FREQUENCY OF 5Hz AND A 'Q' OF .707 .

RESOLVER CONVERSION TO NATURAL BINARY FORMAT



LIST OF REFERENCES

1. Cristi, R., Papoulias, F.A., and Healey, A.J., "Adaptive Sliding Mode Control of Autonomous Underwater Vehicles in the Dive Plane," Naval Postgraduate School (unpublished), revised March 1990.

2. Cristi, R., Healey, A.J., and Papoulias, F.A., "Dynamic Output Feedback by Robust Observer and Variable Structure Control," in review for Proceedings of the 1990 American Control Conference, San Diego, CA, May 1990.

3. Schwartz, M.A., *Kalman Filtering for Adaptive Depth, Steering and Roll Control of an Autonomous Underwater Vehicle (AUV)*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1989.

4. Boncal, Richard J., *A Study of Model Based Maneuvering Controls for Autonomous Underwater Vehicles*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1987.

5. Doyle, J. and Stein, G., "Robustness with Observers," *IEEE Transactions on Automatic Control*, v. AC-24, no. 4, pp. 607-611, August 1979.

6. Gehani, N., *C: An Advanced Introduction*, pp. 36-134, Computer Science Press, Inc., 1988.

7. Davis, M.H., *Real Time Adaptive Control of an Autonomous Underwater Vehicle (AUV)*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1989.

8. Strum, R.D., and Kirk, D.E., *First Principles of Discrete Systems and Digital Signal Processing*, pp. 54-226, Addison-Wesley, Inc., 1988.

9. Ghausi, M.S., and Laker, K.R., *Modern Filter Design Active RC and Switched Capacitor*, pp. 102-191, Prentice-Hall, Inc., 1981.

10. Analog Devices Inc., *Synchro and Resolver Conversion*, 1980.

11. Analog Devices Application Note, "Dynamic Characteristics of Tracking Converters," by M. Thomas, 1980.

12. Analog Devices Technical Note, "Variable Resolution, Monolithic Resolver-to-Digital Converter," 1987.

BIBLIOGRAPHY

Thaler, G.J., *Automatic Control Systems*, West Publishing Company, 1989.

Kraus, A.D., *Matrices For Engineers*, Hemisphere Publishing Corporation, 1987.

Friedland, B., *Control System Design*, McGraw-Hill, 1986.

Barkakati, N., *Microsoft C Bible*, Howard W. Sams & Company, 1988.

Kirk, D.E., *Optimal Control Theory*, Prentice-Hall, Inc., 1970.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5004	1
4.	Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5004 ATTN: Professor R. Cristi, Code EC/Cx	8
5.	Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5004 ATTN: LCDR J. Yurchak, Code CS/Yr	1
6.	Chairman, Code ME Department of Mechanical Engineering Naval Postgraduate School Monterey, CA 93943-5004	2
7.	Chairman, Code CS Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5004	1

8.	Curricular Officer, Code 32 Naval Postgraduate School Monterey, CA 93943-5000	1
9.	Commander, Naval Surface Weapons Center White Oak, MD 20910 ATTN: H. Cook, Code U25	1
10.	Head, Undersea AI and Robotics Branch Naval Ocean System Center San Diego, CA 92152 ATTN: P. Heckman, Code 943	1
11.	Commander, Naval Coastal Systems Center Panama City, FL 32407-5000 ATTN: Dr. G. Dobeck	1
12.	RADM Evans, SEA-92R Naval Sea Systems Command Washington, DC 20362-5101 ATTN: Ms. Judy Rumsey	1
13.	David Taylor Naval Ship Research and Development Center Carderock Laboratory Bethesda, MD 20084-5000 ATTN: Dr. D. Milne, Code 1563	1
14.	Department of Mechanical Engineering Naval Postgraduate School Monterey, CA 93943-5004 ATTN: Professor F. Papoulias, Code ME/Pa	1
15.	Naval Research Laboratory Marine Systems Division Washington, DC 20375 ATTN: Dr. D. Steiger	1

16. Commanding Officer Naval Electronics System Engineering Center Vallejo, CA 94592 ATTN: LT William D. Riling	2