1996-09

Uniform system for the rapid prototyping and testing of controllers for unmanned aerial vehicles

Zanino, James A.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

UNIFORM SYSTEM FOR THE RAPID PROTOYPING AND TESTING OF CONTROLLERS FOR UNMANNED AERIAL VEHICLES

by

James A. Zanino

September, 1996

Thesis Advisor: I.I. Kaminer

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September, 1996	Master's Thesis

4. TITLE AND SUBTITLE Uniform System for the Rapid Prototyping and Testing of Controllers for Unmanned Aerial Vehicles	5. FUNDING NUMBERS
6. AUTHOR(S) Zanino, James A.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The field of control systems has witnessed an explosion in state-space techniques addressing a variety of critical design issues facing control engineers today. Modern computational tools, such as the MATRIX$_x$ Product Family developed by Integrated Systems Incorporated, allow the designer to quickly design, test and implement control systems based on these state-space techniques. These new computing advances shorten the time required to complete a control design from a few years to a few months. However, as the design process progressed new inputs and outputs were required, which usually resulted in a confusing mess of connections that were hard to follow. Therefore, a universal system was needed that could be used on any controller design to aid in the understanding and tracking of the controller's inputs and outputs. A description of this system is given along with a detailed step by step process on how it was implemented on a Unmanned Air Vehicle (UAV).

14. SUBJECT TERMS Rapid Prototying, Unmanned Air Vehicles, Controller Design	15. NUMBER OF PAGES 98
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

UNIFORM SYSTEM FOR THE RAPID PROTOTYPING AND TESTING OF CONTROLLERS FOR UNMANNED AERIAL VEHICLES

James A. Zanino
Lieutanant, United States Navy
B.S. Boston University, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September, 1996

ABSTRACT

The field of control systems has witnessed an explosion in state–space techniques addressing a variety of critical design issues facing control engineers today. Modern computational tools, such as the MATRIX$_X$ Product Family developed by Integrated Systems Incorporated, allow the designer to quickly design, test and implement control systems based on these state–space techniques. These new computing advances shorten the time required to complete a control design from a few years to a few months. However, as the design process progressed new inputs and ouputs were required, which usually resulted in a confusing mess of connections that were hard to follow. Therefore, a universal system was needed that could be used on any controller design to aid in the understanding and tracking of the controller's inputs and outputs. A desription of this system is given along with a detailed step by step process on how it was implemented on an Unmanned Air Vehicle (UAV).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank the people who contributed to this thesis. Dr. I. I. Kaminer for his guidance, teaching, and forethought in developing an outstanding avionics laboratory. LCDR Eric Hallberg for his endless support in helping to solve the technical issues that arose. Don Meeks for his technical expertise wiring Bluebird and keeping Bluebird flying. Most of all, I would like to thank my wife Ellen for her constant support and understanding.

I. INTRODUCTION

The field of control systems has witnessed an explosion in state-space techniques developed to address a variety of critical design issues. These techniques include $\mathcal{H}_2$, $\mathcal{H}_\infty$, l_1, feedback linearization theory, and Linear Matrix Inequalities to name a few. To assess their usefulness, some of these techniques have been applied to real-life problems with some success. Nevertheless, despite all this effort, the user community remains skeptical of the utility of modern control techniques. This is particularly true in the aerospace community, where most of the control systems in service today were developed using classical control design techniques.

At the Naval Postgraduate School we were fortunate to have obtained several unmanned aircraft from various agencies for testing purposes. Therefore, a goal was set to design and flight test controllers developed with the aid of modern control tools. Early in our research we realized that a completely integrated hardware/software system was needed. This system would allow us to:

- Build and analyze controllers using a high level development tool
- Automatically generate computer code once a satisfactory controller has been obtained
- Download the code into a hardware system capable of flying the aircraft.

We have been able to develop such a system using the MATRIX$_X$Product Family of rapid prototyping software available from Integrated Systems Incorporated (ISI) [Ref. 1]. This software incorporates a program called RealSim that uses a Graphical User Interface (GUI) to step the engineer through the design process. This rapid prototyping process greatly reduces the time required to design, test and implement a controller when compared with the conventional design process. A comparison of these two techniques, conventional vs. rapid prototyping is given next.

A. CONVENTIONAL DESIGN

Conventional design of control systems takes place in several stages, using several different tools for control design, software engineering, data acquisition, and testing. A typical procedure for the design process is:

- The engineer creates an accurate plant model.
- The model is simulated to see how it compares with reality. Data measuring the behavior is collected, and the model is changed, if necessary.
- An engineer builds a control system for the plant. The control system, including the plant, is tested.
- The model is implemented in hardware and tested. Modifications are made, if necessary.
- After more testing, a functional prototype is obtained.

This method of design has two major shortcomings:

- It is expensive, because the prototype or the controller must be modified at each stage.
- It is very time-consuming from conception to finished prototype.

B. RAPID PROTOTYPING

In contrast, the rapid prototyping design process uses one tool and two steps to:

- Integrate the tools for each stage of the system development into a single environment.
- Allow the design progress to easily flow through the development stages.
- Create a working prototype early in the design process.

A flow diagram comparing the conventional and rapid prototyping techniques is given in Figure 1 [Ref. 1].

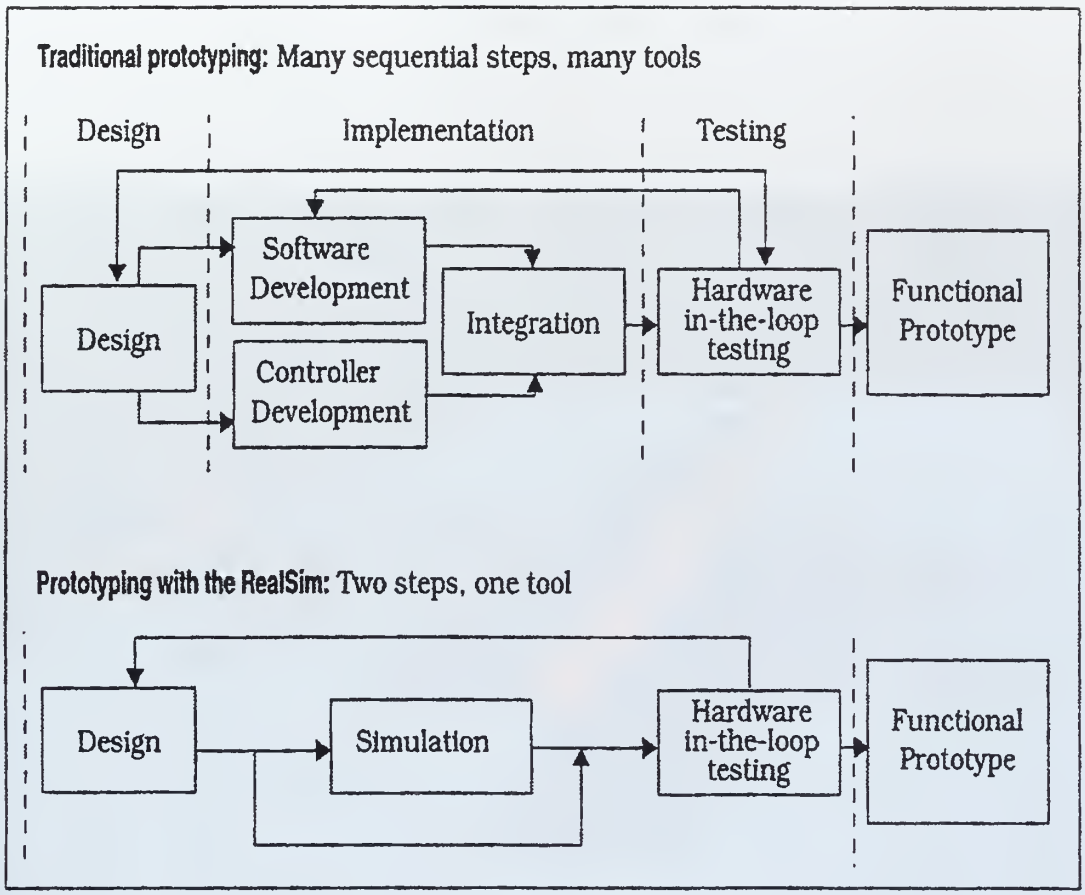

Figure 1. The Rapid Prototyping Concept [Ref. 1]

This thesis describes how the rapid prototyping software RealSim was used to develop a universal system that can be used for any controller design. This system was used to design a controller using $\mathcal{H}_\infty$ synthesis which was then tested on an Unmanned Air Vehicle (UAV) called Bluebird.

Bluebird was donated to the Unmanned Air Vehicle Lab by the Naval Ocean System Center, San Diego. It has a 12.5 foot wingspan, a 25 pound payload capability, and is equipped with a full avionics suite, including IMU, GPS and air data sensors. It is controlled through the use of a RF link that sends a Pulse Width Modulated (PWM) signal that drives the aircraft's actuators. One of these links was modified

and connected to the controller on the ground so the aircraft could be flown by the AC100 system. A photograph of Bluebird is given in Figure 2.

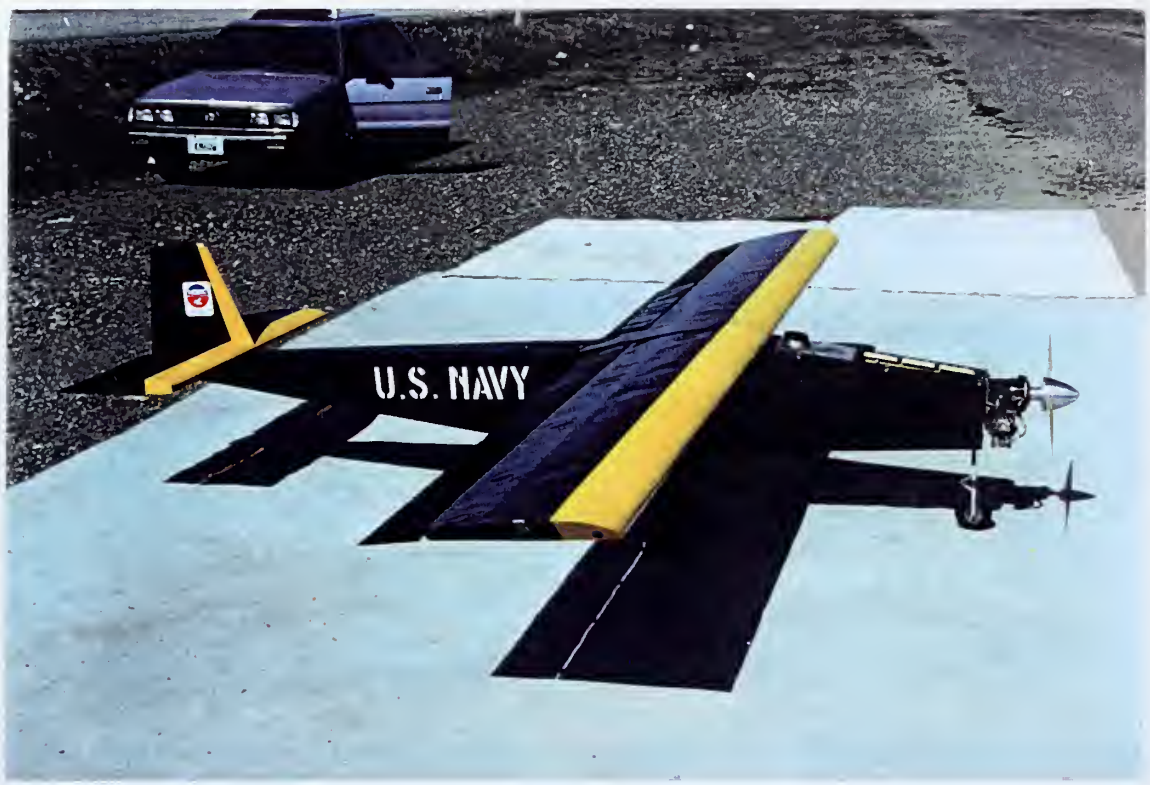

Figure 2. Bluebird

II. EQUATIONS OF MOTION & CONTROLLER DESIGN

Before discussing the specifics of the uniform system development, we present the Equations of Motion (EOM) and the $\mathcal{H}_\infty$controller designed for Bluebird, since these were used to test the utility of the system.

First, we introduce the following notation suggested by [Ref. 2]:

P - position of the origin of $\{B\}$ expressed in $\{I\}$;

$V = (u,\ v,\ w)'$ -linear velocity of the origin of $\{B\}$ relative to $\{I\}$, expressed in $\{B\}$;

$\Lambda = (\phi,\ \theta,\ \psi)'$ - vector of Euler angles which describe the orientation of frame $\{B\}$ with respect to $\{I\}$

$\Omega = (p,\ q,\ r)'$ - angular velocity of $\{B\}$ relative to $\{I\}$, expressed in $\{B\}$;

${}^I_B\mathcal{R} = {}^I_B\mathcal{R}(\Lambda)$ - rotation matrix from $\{B\}$ to $\{I\}$.

$\mathcal{Q} = \mathcal{Q}(\Lambda)$ - matrix that relates $\frac{d}{dt}\Lambda$ to Ω and satisfies the relationships $\frac{d}{dt}\Lambda = \mathcal{Q}\Omega$ and $\mathcal{Q}(0) = I$;

G - vector of gravitational acceleration expressed in $\{I\}$. We assume a constant gravitational field.

Furthermore, let U denote the vector of control inputs acting on the vehicle. For Bluebird, U consists of elevator el, thrust th, ailerons ail and rudder r.

Let $\{W\}$ denote a wind axis. It is usually attached to the aircraft's center of gravity and is defined using the right hand rule, with the x-axis pointing in the direction of the apparent wind. For example, in the absence of wind the aircraft's inertial velocity resolved in $\{W\}$ has the following form: $[\|V\|\ \ 0\ \ 0]'$. Now let ${}^B_W\mathcal{R}$ denote the transformation from $\{W\}$ to $\{B\}$. Notice, ${}^B_W\mathcal{R}$ can be computed using the angle of attack α and the sideslip angle β, where $\beta = \sin^{-1}\frac{v}{\|V\|}$ and $\alpha = \sin^{-1}\frac{w}{\|V\|}$.

Now, using the above notation, the Bluebird dynamics have the following form:

$$\mathcal{G} = \begin{cases} \frac{d}{dt} V &= \mathcal{F}_V(V,\Omega,\Lambda) + \mathcal{I}_V(V,\Omega)\mathcal{H}(V,\Omega,U), \\ \frac{d}{dt} \Omega &= \mathcal{F}_\Omega(V,\Omega,\Lambda) + \mathcal{I}_\Omega(V,\Omega)\mathcal{H}(V,\Omega,U), \\ \frac{d}{dt} P &= {}^{I}_{B}\mathcal{R} V, \\ \frac{d}{dt} \Lambda &= \mathcal{Q}\,\Omega, \end{cases} \tag{II.1}$$

where

$$\mathcal{F}_V(V,\Omega,\Lambda) = -\mathcal{S}(\Omega)V + {}^{B}_{I}\mathcal{R}G + \frac{1}{2m}\rho\|V\|^2 \mathrm{s}^{B}_{W}\mathcal{R}\left\{\begin{bmatrix} c_{d0} \\ c_{y0} \\ c_{l0} \end{bmatrix} + \begin{bmatrix} c_{d_u} & 0 & c_{d_\alpha} \\ 0 & c_{y_\beta} & 0 \\ c_{l_u} & 0 & c_{l_\alpha} \end{bmatrix} \begin{bmatrix} \frac{u-u_c}{\|V\|} \\ \beta \\ \alpha \end{bmatrix} + \begin{bmatrix} 0 & c_{d_q} & 0 \\ c_{y_p} & 0 & c_{y_r} \\ 0 & c_{l_q} & 0 \end{bmatrix} \Omega \right\},$$

$$\mathcal{I}_V(V,\Omega,\Lambda) = \frac{1}{2m}\rho\|V\|^2 \mathrm{s}^{B}_{W}\mathcal{R} \begin{bmatrix} c_{d_{th}} & c_{d_{el}} & 0 & 0 \\ 0 & 0 & c_{y_{ail}} & c_{y_{rud}} \\ c_{l_{th}} & c_{l_{el}} & 0 & 0 \end{bmatrix}, \quad \mathcal{H}(V,\Omega,U) = U$$

$$\mathcal{F}_\Omega(V,\Omega,\Lambda) = \mathcal{J}_B^{-1}\left\{-\mathcal{S}(\Omega)\mathcal{I}_B\Omega + \frac{1}{2}\rho\|V\|^2 \begin{bmatrix} \mathrm{s}b & 0 & 0 \\ 0 & \mathrm{s}\bar{c} & 0 \\ 0 & 0 & \mathrm{s}b \end{bmatrix} {}^{B}_{W}\mathcal{R}\left\{\begin{bmatrix} c_{l0} \\ c_{m0} \\ c_{n0} \end{bmatrix} + \begin{bmatrix} 0 & c_{l_\beta} & 0 \\ c_{m_u} & 0 & c_{m_\alpha} \\ 0 & c_{l_\beta} & 0 \end{bmatrix} + \begin{bmatrix} c_{l_p} & 0 & c_{l_r} \\ 0 & c_{m_q} & 0 \\ c_{n_p} & 0 & c_{n_r} \end{bmatrix} \begin{bmatrix} \frac{b}{2\|V\|} & 0 & 0 \\ 0 & \frac{\bar{c}}{2\|V\|} & 0 \\ 0 & 0 & \frac{b}{2\|V\|} \end{bmatrix} \Omega \right\}\right\},$$

$$\mathcal{I}_\Omega(V,\Omega,\Lambda) = \mathcal{J}_B^{-1}\frac{1}{2}\rho\|V\|^2 \mathrm{s}^{B}_{W}\mathcal{R} \begin{bmatrix} 0 & 0 & c_{l_{ail}} & c_{l_{rud}} \\ c_{m_{th}} & c_{m_{el}} & 0 & 0 \\ 0 & 0 & c_{n_{ail}} & c_{n_{rud}} \end{bmatrix},$$

$$
\begin{array}{rcl}
\mathrm{s} & := & \text{wing reference area} \\
\rho & := & \text{air density} \\
\bar{c} & := & \text{wing mean chord} \\
m & := & \text{mass} \\
\mathcal{J}_B & := & \text{aircraft's inertia tensor resolved in } \{B\} \\
\rho & := & \text{air density} \\
b & := & \text{wing span} \\
d,\ y,\ l & := & \text{normalized drag, lateral force and lift} \\
\mathrm{l},\ m,\ n & := & \text{normalized roll pitch and yaw moments} \\
c_{x_0} & := & \text{normalized nominal force or moment coefficient} \\
c_{x_y} & := & \text{stability derivative: } \frac{\partial x}{\partial y} \\
u_c & := & x\text{-component of } \textit{vehicle'strimairspeed.}
\end{array}
$$

Notice, in equations (II.1) we did not include ρ dynamics.

The set of trimming trajectories $\mathcal{E}$ for Bluebird is defined as follows:

$$
\mathcal{E} := \left\{ \begin{bmatrix} V_C \\ \Omega_C \\ \Lambda_C \end{bmatrix} \quad : \quad \begin{array}{l} \frac{d}{dt}\,\Lambda_C = \mathcal{Q}_C\,\Omega_C, \\ \\ \mathcal{F}_V(V_C,\Omega_C,\Lambda_C) \;+\; \mathcal{I}_V(V_C,\Omega_C)\mathcal{H}(V_C,\Omega_C,U_c) = 0 \\ \mathcal{F}_\Omega(V_C,\Omega_C,\Lambda_C) \;+\; \mathcal{I}_\Omega(V_C,\Omega_C)\mathcal{H}(V_C,\Omega_C,U_c) = 0, \end{array} \right\} \tag{II.2}
$$

where V_C, Ω_C and Λ_C can be computed using the desired airspeed v_{t_c}, desired flight path angle γ_c and the desired turning rate $\dot{\psi}_c$. Now, given $[v_{t_c}\ \ \gamma_c\ \ \dot{\psi}_c]'$ and $\beta_c = 0$ we can solve for V_C, Ω_C, U_c and Λ_C:

$$
\begin{aligned}
&\mathcal{F}_V(V_C,\Omega_C,\Lambda_C) \;+\; \mathcal{I}_V(V_C,\Omega_C)\mathcal{H}(V_C,\Omega_C,U_c) = 0 \\
&\mathcal{F}_\Omega(V_C,\Omega_C,\Lambda_C) \;+\; \mathcal{I}_\Omega(V_C,\Omega_C)\mathcal{H}(V_C,\Omega_C,U_c) = 0 \\
&\mathcal{Q}_C^{-1}\Omega_C - \dot{\Lambda}_C = 0 \\
&\|V_C\| - v_{t_c} = 0 \\
&\beta_c - \sin^{-1}\frac{v_c}{\|V_C\|} = 0
\end{aligned}
$$

$$\gamma_c - [0 \;\; 1 \;\; 0] \, arg({}^W_B\mathcal{R}^B_I\mathcal{R}) = 0, \tag{II.3}$$

where $V_C = [u_c \; v_c \; w_c]'$ and the arg function extract the angles X from the rotation matrix $\mathcal{R}(X)$: $X = arg(\mathcal{R}(X))$ (for example, for straight line flight the last expression in equations (II.3) reduces to $\gamma_c = \theta_c - \alpha_c$). Equations (II.3) consist of 12 equations in 12 unknowns, since the trimming value of the heading angle ψ is arbitrary and can be solved using analytical or numerical methods. For an example of an interesting analytic solution see [Ref. 3, 4].

Using the solution to equations (II.3) the linear model for Bluebird was obtained along a straight line trajectory characterized by the velocity v_c of 73 fps, γ_c of zero and wings level ($\dot{\psi}_c = 0$) (a typical cruise condition for Bluebird). This model was used to design a linear feedback controller for the vehicle.

A. FEEDBACK CONTROLLER DESIGN

1. Open Loop Analysis

The linear model of Bluebird in cruise is typical for a fixed wing aircraft, i.e. it naturally decouples into lateral and longitudinal dynamics. The longitudinal dynamics are characterized by a short period mode with a natural frequency of .5 radians/second, a damping ratio of .7 and a lightly damped, stable phugoid mode. Lateral dynamics include a lightly damped dutch roll mode with a damping ratio of .03, a roll mode, and an unstable spiral mode [Ref. 9]. Bluebird utilizes standard elevators, rudder, and differential ailerons for control and a single gas engine driven propeller in the nose for thrust.

To ensure appropriate dutch roll response the RC pilot's flight technique of tying positive rudder and negative aileron together was mimicked.

2. Design Requirements

The basic control strategy for the feedback controller is to emulate the approach used by the RC pilot. Classically, the pilot uses elevator and thrust to control altitude and airspeed in steady state. These considerations motivate the following

design requirements for the feedback controller.

1. **Zero Steady State Error**

   - Achieve zero steady state tracking errors in airspeed, altitude and bank angle commands in the presence of constant and light variable winds.

2. **Bandwidth Requirements**

   - The command-loop bandwidth for each command channel should be no greater than 1 radian per second and no less than 1/10 radian per second.
   - The control-loop bandwidth should not exceed 12 radians per second for the elevator, aileron and rudder loops, and 5 radians per second for the throttle loop. These numbers represent 50 % of the corresponding actuator bandwidths and shall ensure the actuators are not driven beyond their linear operating range.

3. **Closed Loop Damping and Stability Margins**

   - The dominant closed loop eigenvalues should have a damping ratio of at least 0.5. Simultaneous gain and phase margins of 6db and 45 deg in each control loop must be achieved.

3. Linear Design: $\mathcal{H}_\infty$ Synthesis

The methodology selected for linear control system design was $\mathcal{H}_\infty$ synthesis [Ref. 6]. This method rests on a firm theoretical basis, and leads naturally to an interpretation of control design specifications in the frequency domain. Furthermore, it provides clear guidelines for the design of controllers to achieve robust performance in the presence of plant uncertainty. The basic steps in the controller-design procedure, including the development of the synthesis model, were done using the approach described in [Ref. 7]. This approach provides an intuitive and straightforward way for converting the design requirements into the weights for the $\mathcal{H}_\infty$synthesis model. Consider Figure 3. Here $\mathcal{C}_l$ is the controller to be designed, and $\mathcal{G}_l$ is the linear model of Bluebird.

In Figure 3 the vector of exogenous inputs w represents the commanded inputs. The vector y_1 represents vehicle's airspeed, altitude and bank angle. The regulated output z includes the outputs of the weighting matrices W_1 and W_2. These matrices

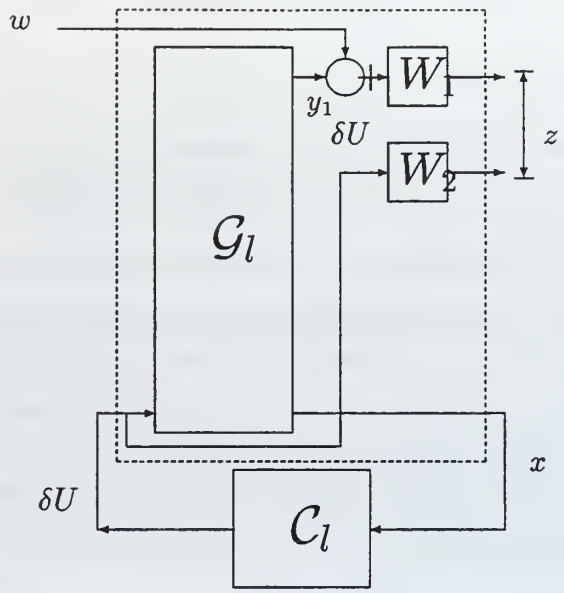

Figure 3. Synthesis and Analysis Model

had the following form:

$$W_1 = \begin{bmatrix} \frac{c_1}{s} & 0 & 0 \\ 0 & \frac{c_2}{s} & 0 \\ 0 & 0 & \frac{c_3}{s} \end{bmatrix}$$

$$W_2 = \begin{bmatrix} c_4 & 0 & 0 \\ 0 & c_5 & 0 \\ 0 & 0 & c_6 \end{bmatrix},$$

where the constants $c_i, i = 1,6$ were used as the design knobs adjusted to meet the closed loop tracking, damping, control and command loop bandwidth requirements. Notice that the structure of W_1 ensures the steady state tracking of constant commands in all three channels. The resulting linear controller has the following form:

$$\mathcal{C}_l := \begin{cases} \delta E & = [\delta v_t \;\; \delta z \;\; \delta\phi]' - [\delta v_{t_c} \;\; \delta z_c \;\; \delta\phi_c]' \\ \frac{d}{dt}\delta X_c & = \delta E \\ \delta U & = \mathcal{C}_{c1}\delta X_c + \mathcal{D}_{c1}[\delta V' \; \delta\Omega' \; \delta\Lambda']', \end{cases}$$

where $v_t = \|V\|$, the $\mathcal{H}_\infty$state feedback gain is $\mathcal{K} = [\mathcal{C}_{c1}\ \mathcal{D}_{c1}]$ and δX_c represents the state of the integral errors. The feedback system consisting of the linear plant $\mathcal{G}_l$ and the controller $\mathcal{C}_l$ was found to meet all the design specifications given in Section 2.

Since the effectiveness of the aerodynamic control surfaces (these include elevator, rudder and ailerons) is proportional to the dynamic pressure $\bar{q} = 0.5\rho\|V\|^2$ the controller $\mathcal{C}_l$ was gain-scheduled on $\bar{q}$:

$$\mathcal{C}_l(\bar{q}) := \begin{cases} \delta E & = [\delta v_t\ \ \delta z\ \ \delta\phi]' - [\delta v_{t_c}\ \ \delta z_c\ \ \delta\phi_c]' \\ \frac{d}{dt}\delta X_c & = \delta E \\ \delta U & = \mathrm{diag}(\frac{\bar{q}_0}{\bar{q}}, \frac{\bar{q}_0}{\bar{q}}, 1)\{\mathcal{C}_{c1}\delta X_c + \mathcal{D}_{c1}[\delta V'\ \delta\Omega'\ \delta\Lambda']'\}, \end{cases}$$

where $\bar{q}_0$ represents the nominal value of $\bar{q}$.

4. Implementation of the Linear Controller

Using the $\mathcal{D}$ technique for implementing the gain-scheduled controllers [Ref. 5] the family of linear gain-scheduled controllers $\mathcal{C}_l(\bar{q})$ was implemented on the nonlinear plant $\mathcal{G}$ as follows:

$$\mathcal{C} := \begin{cases} E & = [v_t - v_{t_c}\ \ z - z_c\ \ \phi - \phi_c] \\ \dot{X}_c & = \frac{\bar{q}_0}{\bar{q}}\{\mathcal{C}_{c1}X_c + \mathcal{D}_{c1}[\frac{d}{dt}V'\ \ \frac{d}{dt}\Omega'\ \ \frac{d}{dt}\Lambda]'\} \\ U & = X_c, \end{cases}$$

where the differentiation operator $\frac{d}{dt}$ was replaced by a causal operator with the transfer function $\frac{s}{\tau s+1}$ [Ref. 5].

III. HARDWARE/SOFTWARE DESCRIPTION

A. BACKGROUND

The primary tool for the research conducted in the Avionics lab is the hardware/software interface provided by the MATRIX$_X$ Product Family developed by Integrated Systems Inc (ISI). This tool set greatly enhances the control engineer's ability to test and evaluate a design concept. The primary software, called Realsim, consists of an easy to use graphical user interface (GUI) that can be run on either a workstation or PC. The interface interacts with a high speed digital signal processing board developed by Texas Instruments, called the C30, that uses parallel processing techniques to handle many tasks simultaneously. One of the unique features of this software is the ability to automatically program higher-language code such as C or ADA for the designed controller. This greatly reduces the time required for the designer to test, modify and implement a designed controller.

B. HARDWARE

The hardware system in the avionics lab consists of two ISA bus PC adapter boards, which can be placed in any IBM compatible PC with an ISA bus architecture and available ISA slots. One of the boards is installed in a luggable PC called *AC*100 and the second is installed on the Pentium tower PC called *America*. *America* is permanently connected to the AA department network, while *AC*100 can be connected to the network via a TCP/IP connection. Using this TCP/IP connection these computers can communicate with the Realsim software installed on the UNIX workstations. This software can be run on any Sun workstation in the AA department.

The two hardware boards included in the PC portion of the Realsim Series AC-100 Model C30 system are the following: a board which acts as a motherboard for the C30 digital signal processor (DSP), and a "DSP_FLEX" board, which can hold

up to 4 "IP" modules. The "IP" modules are compact input/output (I/O) devices that perform a particular type of I/O. The PC *America* has 4 IP modules, consisting of one serial communications (IP_Serial) module, one Digital to Analog (IP_DAC), and two Pulse Width Modulation (IP_68332) modules. The luggable PC *AC*100 has 3 IP modules, consisting of one serial communications (IP_Serial) module, one,one Digital to Analog (IP_DAC) and one Pulse Width Modulation (IP_68332) module. The two boards are connected by a seperate comm bus via a ribbon cable. The I/O configuration for *AC*100 and *America* are given in Table I. These values are needed when connecting the hardware to the software using the Hardware Connection Editor (HCE), which will be explained later in the thesis.

1. IP_Serial Module

The IP_Serial module provides two channels of high performance multi-mode serial communications with RS–232–C and RS–422 capability. The module can be programmed to baud rates of 2 Mbit/sec supporting both asynchronous and synchronous protocols. The PC *AC*100 has two serial modules labelled 1 and 2. As stated in Table I each of these modules has two channels and they are defined as channels A and B. To create a connection from a SystemBuild model to the IP_Serial module, the Hardware Connection Editor (HCE) must be used. For the specifics on this procedure please see [Ref. 1].

Table I. I/O Configuration

Module	*America*	*AC*100
IP_Serial 1	–	–
IP_Serial 2	3	3
IP_HiADC	1	–
IP_DAC	2	2
IP_68322	4	4

2. IP_HiADC Module

The HiADC module provides 16 input analog channels with 12–bit resolution and synchronous sampling of all inputs. The module can convert one analog channel in 1.2 μsec or approximately 800 K samples/second. Conversion of all 16 channels takes approximately 20 μsec. Each channel has a fixed voltage range of $\pm$ 5V with an input impedance of 1 MOhm. There is no anti–aliasing filtering provided on the HiADC module so inputs should be band–limited to 1/2 the sampling frequency of the system. To create a connection from a SystemBuild model to the IP_HiADC module, the HCE must be used. For the specifics on this procedure please see [Ref. 1].

3. IP_DAC Module

The IP_DAC module provides six channels of 12–bit digital to analog conversion. Each channel may be configured to either $\pm$5V or 0–10V output ranges. To create a connection from a SystemBuild model to the IP_DAC module, the HCE must be used. For the specifics on this procedure please see [Ref. 1].

4. IP_68332 Module

The IP_68322 module is a time processing unit produced by Motorola, that can perform one or more hardware I/O functions.

The software drivers for each of the IP modules are located in the C30_SP subdirectory under the AC100DSP directory. These drivers will be merged into the input and output fields in the c_c30.hce file, which is used by the Hardware Connection Editor (HCE) to determine the allowable I/O devices. A copy of the c_c30.hce file must be in the working directory of the project of interest to ensure the targeted hardware configuration is used.

C. SOFTWARE

The Realsim software installed on the UNIX workstations gives the controls engineer a vast array of tools that make the designing and testing of a control system

much quicker and simpler than before. Through the use of a GUI (see Figure 4) the designer can follow a flow diagram that steps the user through the various software components to fully implement a given design. These software tools consists of Xmath/ SystemBuild, Autocode, Interactive Animation (IA) Builder, Hardware Connection Editor (HCE), and two utilities that compile, link, download and run the control design. A brief description of these applications is given next.

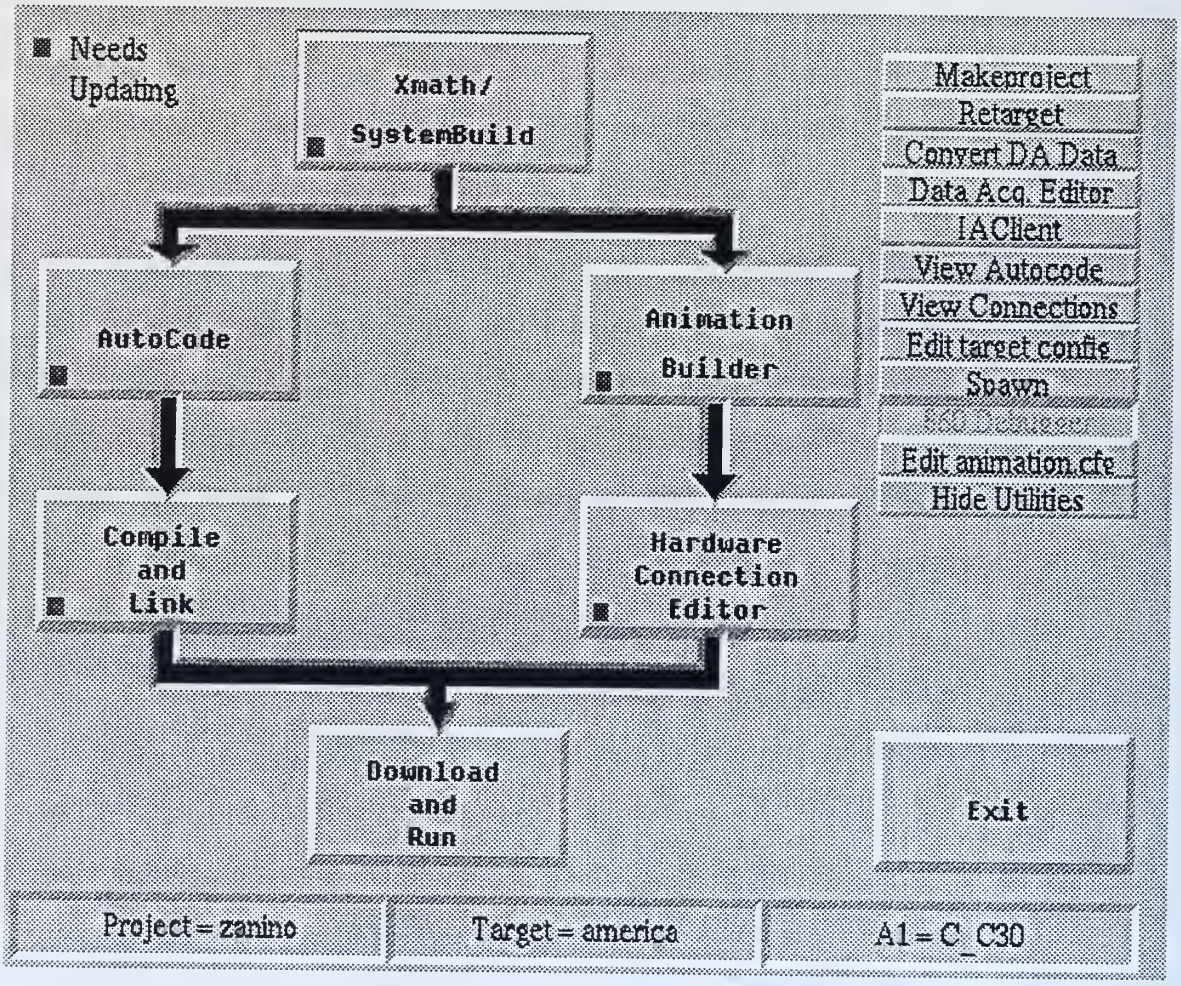

Figure 4. Realsim Graphical User Interface (GUI)

1. Xmath/SystemBuild

Xmath/SystemBuild is a software program similar to the Matlab/Simulink software program developed by Math Works Inc. Xmath/Systembuild was designed

to include an extensive set of design and analysis functions for both the classical input/output control techniques and the modern state-space control techniques. Specifics on how to design and analyze a model can be found in the Xmath and SystemBuild Core Manuals. The key requirement to test a model with external inputs/outputs, and to eventually Autocode the model, is to have the external inputs/outputs in the highest level Superblock of the model.

The SystemBuild program uses a hierarchical method of organization, based on the SuperBlock concept. SuperBlocks provide a way of organizing a group of blocks that define a function into a compact form for display and understanding, for example "Sensor Calibration_Display" or "Control_Block". Through the use of this hierarchy, variable names can be passed up and down the hierarchical structure allowing the engineer to easily track and understand what variables are and where they interact with the model. A diagram showing the interaction of Xmath and SystemBuild is given in Figure 5.

Once the model is drawn and labeled, there are several ways to test it. It can be tested within Xmath/SysyemBuild using the "SIM" function (see Core Manuals) or by generating realtime code. The second method, generating realtime code, is the preferred method since it allows for the gerenation of a higher-level language to conduct hardware-in-the-loop testing. To generate real time code the user just uses a pull down menu on the SystemBuild GUI and selects "Generate Real Time Code". This produces a file with the models name followed by a **.rtf** extension. This Real Time Code is a top level Input/Output code that is used by the AutoCode program to produce a higher-level code such as C.

2. AutoCode

An integral part of the quick design and testing of a controller is the ability to generate high-level code such as C or ADA automatically. AutoCode has this capability, and generates optimized code from a library of standard functions and calls. The Realsim package in the Avionics lab currently has a C code module.

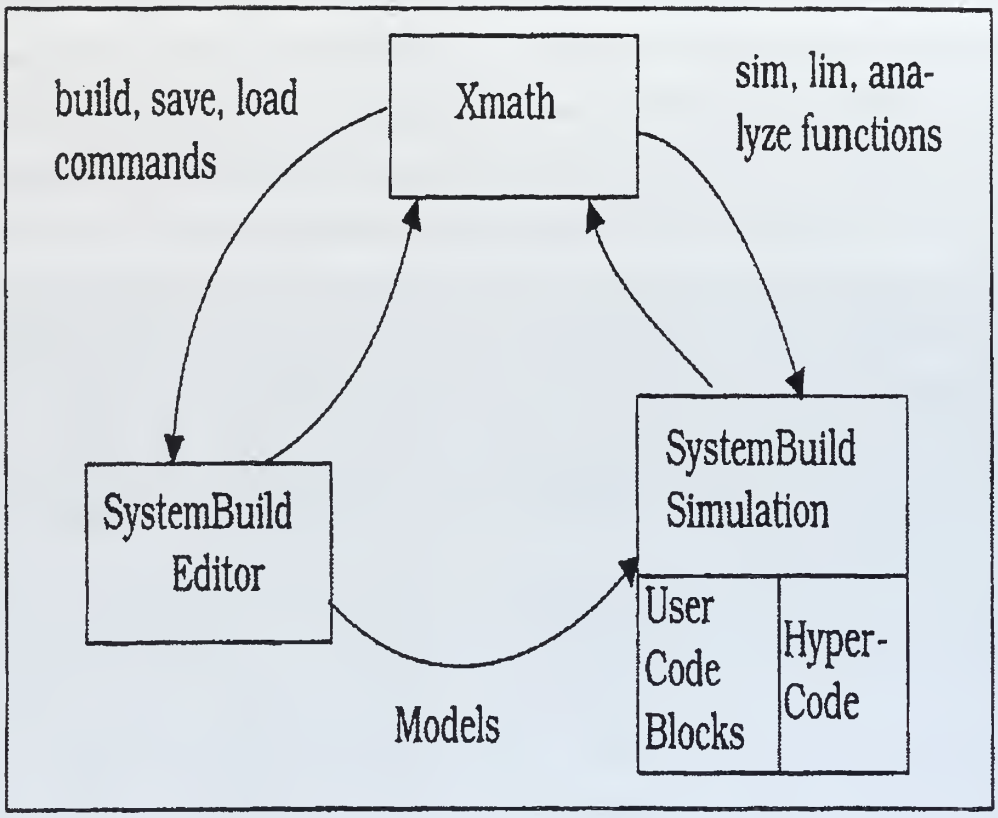

Figure 5. Xmath/SystemBuild Integration [Ref. 1]

To generate C code two tasks must be accomplished:

- Ensure settings in **target_config.cfg** are correct.
- Click on the AutoCode icon on main GUI.

The **target_config.cfg** file is created by using the Realsim utility **retarget.** This utility can be run at the command prompt in the working directory or by clicking on the Retarget icon on the utilities menu. This procedure will ask several questions. First it will ask for the computer's host name, which will be either *AC*100 or *America.* This host name designates the computer on the network where the application will be

compiled, downloaded and executed. For the rest of the questions the default setting should be accepted by hitting return.

Once the two above steps are completed a new file with a .c extension will be created in the working directory. This file will be used later to compile, link, download and run the design.

3. Interactive Animation Editor (IA)

This program is used to build a graphical animation module to display desired system outputs in various forms during testing. The animation diagrams are made by a drag and drop process from a given library of predrawn gauges, strip charts, dials, switches, and other various input and output devices. If a specific dial or gauge for the user needs can't be found in the library, custom pictures can also be created. The "RTF Names" button loads the I/O names from the model .rtf file, to help associate given inputs and outputs to their display icons. A RTF file must be loaded prior to making any connections. Using a connection editor similar to the one used in SystemBuild the appropriate inputs and outputs are connected to the appropriate devices.

Once a picture is completed the user selects "Save Picture" and a file with a .pic extension is created in the working directory. This file will be used later in the Hardware Connection Editor (HCE) and link process. Note: If the SystemBuild I/O is changed, the I/A Editor must be run again and connections changed to reflect the changes to the model. A sample IA picture is shown in Figure 6.

4. Hardware Connection Editor (HCE)

The hardware connection editor is used to associate external inputs and outputs in the SystemBuild model with either external I/O devices or the I/A module. This is accomplished using two screens that have the same basic layout. The first screen is for SuperBlock external inputs and the second for external outputs. (see Figure 7).

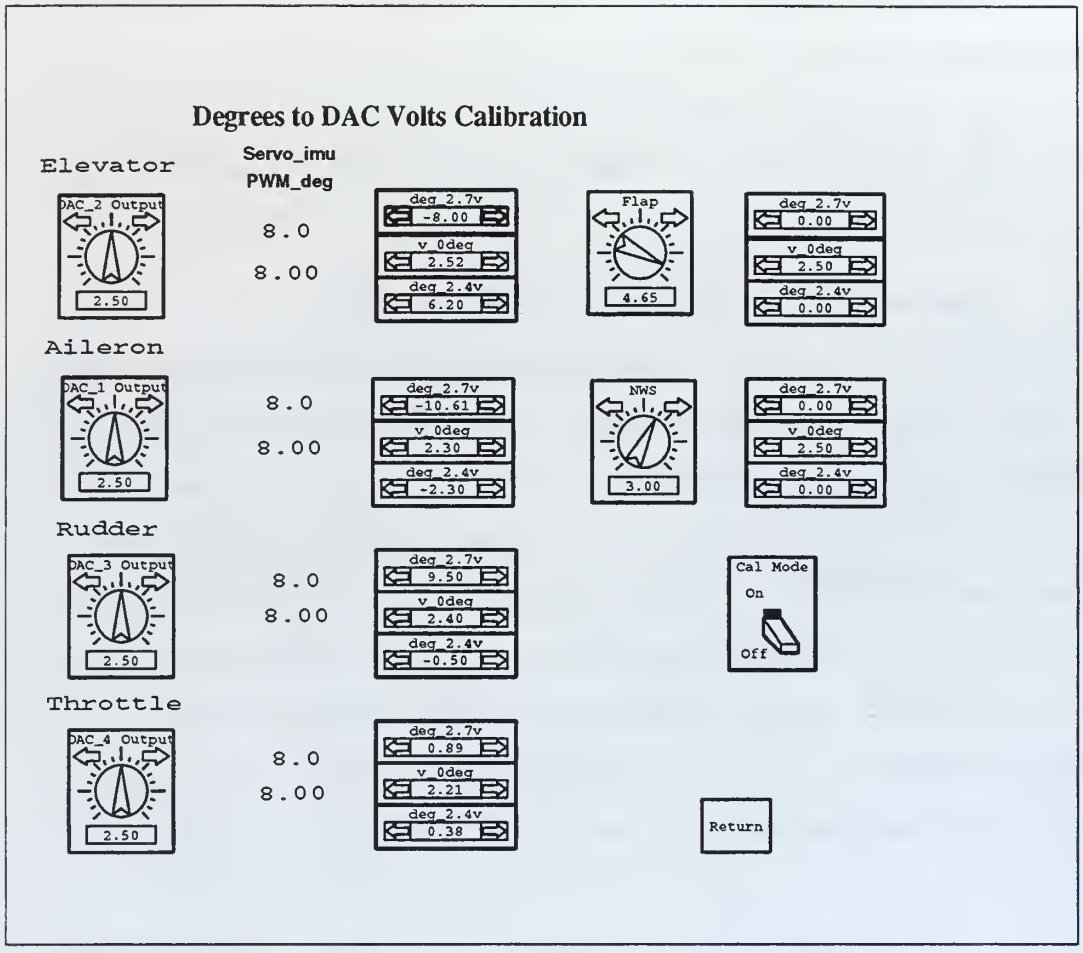

Figure 6. Sample of Interactive Animation Picture

The HCE reads the external inputs/outputs from the .rtf file with the same name as the current target. The HCE automatically checks to see if a local copy of the c_c30.hce is present, and if so reads it. If not, it reads one out of the default AC100 directory. This file informs the HCE what type of external I/O devices are present in the target AC100 computer, and the HCE will only allow these device types to be selected. A detailed explanation of the columns on the screens and their uses can be found in [Ref. 1].

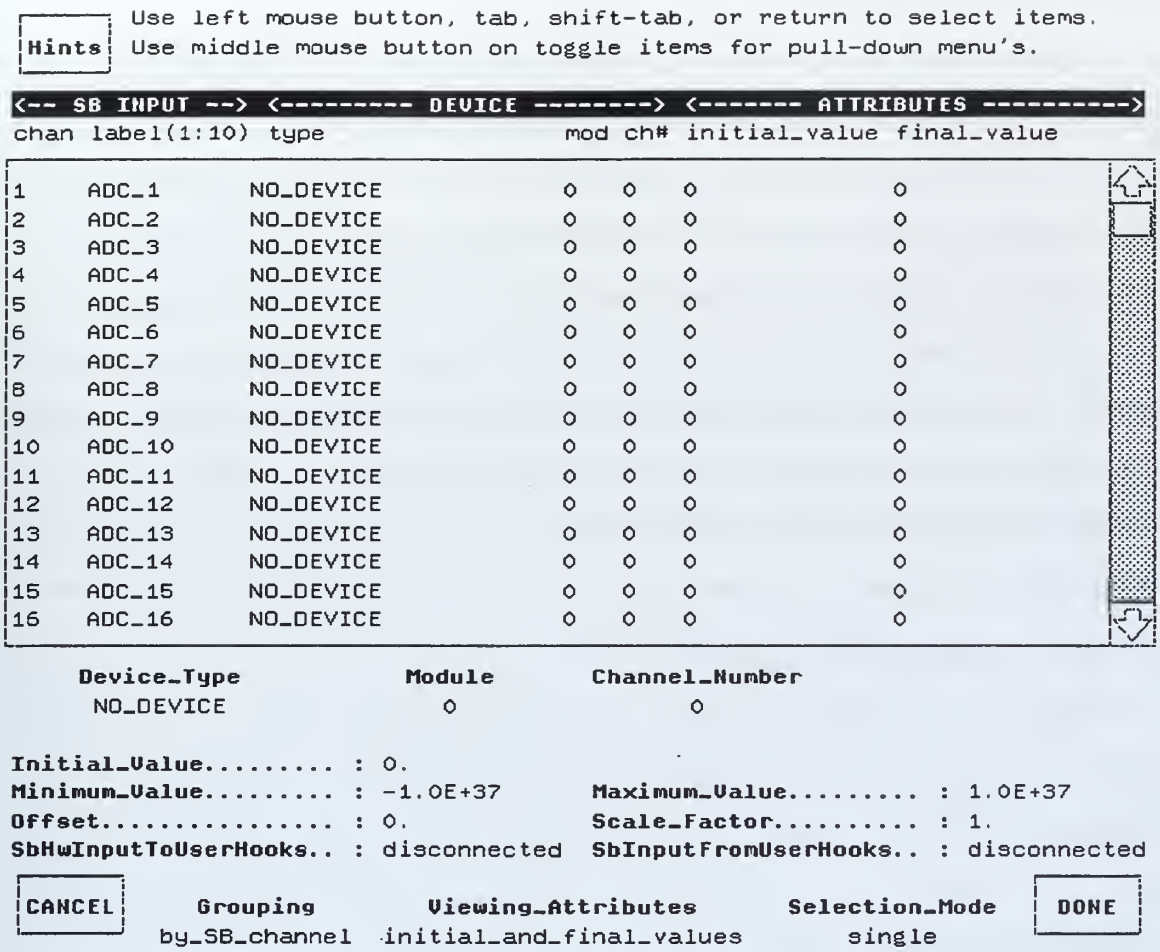

Figure 7. Sample of Hardware Connection Editor

5. Compile and Link

Once the desired inputs and outputs are connected the design needs to be complied and linked to the C30. The Realsim software will attempt to connect via ftp with the AC100 target computer. For the PC *America* the user must first type "ac100svr" at the DOS prompt to enable the computer for ftp transfer. For the portable PC *AC*100 the user repeats this same command, or if Windows is running, double clicks on "ac100 terminal". Once the connection is made, the C source files required will be transferred to the target computer for remote compiling and linking. The compilier generates object code from the .c file, and the link creates a C30 DSP

executable from the object code.

If there are other C files required for the project for compiling and linking, there should also be a file in the working directory named "sa_user.cmd". In this file, for each C file to be included, there should be a line that says **COMPILE ≪filename≫ .c** and a line that says **LINKWITH ≪filename≫** , where filename is the source file name to be included, without extension. This only applies to C source files. If there are header files referred to in the C source files, they must be manually copied to the project directory on the AC100 computer using ftp. If there is more than one file included, and there are dependencies in one file from another (i.e. function calls to functions defined in another source file), they should be listed in sa_user.cmd in the order of dependency.

6. Download and Run

When this program is selected on the GUI, Realsim software will attempt to connect to the target AC100 computer through ftp. Once the connection is made, it will load the C30 executable into the C30 memory and prepare it to run. It will bring up the IA module on the workstation screen, and place a control panel called IA Client, with 6 buttons below the IA module. See Figure 8.

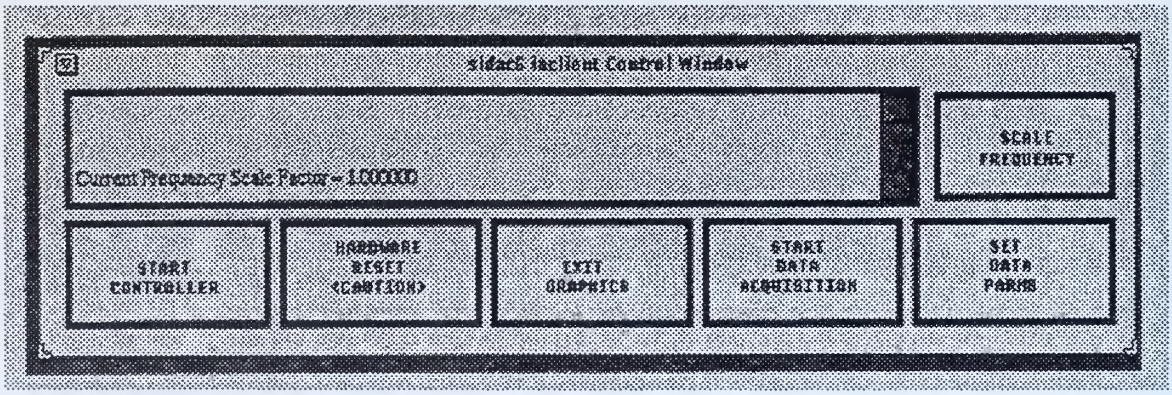

Figure 8. IA Client Control Window

The first button, **START CONTROLLER** , will start the model if the model has just been loaded and not run yet. It will stop the model if it is currently

running, and it will restart the model if it has been previously stopped. The second button, **HARDWARE RESET** , will cause the Realsim controller to immediately reset and causes IA Client to exit. This is the best way to exit after a run, as it will clear the C30 memory and return the AC100 computer to a ready status. The third button, **EXIT GRAPHICS** , exits IA Client without rebooting the controller. This is a software reboot only, which stops the model and terminates the ftp connection. This button is not recommended for use, as it will not stop the model from running on C30. If download and run is selected again by the original client which started the model, it will ask if you wish to reconnect the model. If a different client attempts to log on to run the same or a different model it will ask if the current model should first be terminated. Therfore, it is always best to terminate the model by pressing the first button. The fourth button, **START DATA ACQUISITION** , starts/stops data acquisition. Data acquisition should always be stopped before rebooting the C30, or the acquired data will not be saved to a file. The last button allows you to set certain data acquisition parameters. The Scale Frequency button in the upper right corner allows the user to slow down or speed up the animation.

In addition to the six main buttons on the Realsim GUI, there are additional features that can be invoked from the menu located in the upper right corner of the RealSim GUI. These utilities are hidden when Realsim is first started but can be displayed by clicking on the **Show Utilities** button. These utilities give the user the ability to accomplish such tasks as data collection, conversion of data for use in Xmath, IA Client, and numerous other features. More information can be found in [Ref. 1].

IV. RAPID PROTOTYPING SYSTEM (RPS)

A. RPS ARCHITECTURE

In order to conduct flight test a Rapid Prototyping System (RPS) was developed with the following features:

- synthesis, analysis and simulation of flight management algorithms using a high level developmental tool.
- automatic generation of computer code once a satisfactory flight management system has been obtained.
- code download into a hardware system capable of flying a UAV.
- display and collection of appropriate flight data.

A diagram of the RPS architecture is given in Figure 9.

The hardware architecture is divided into the following two major components: a ground station and an unmanned aircraft.

1. Ground Station

The ground station handles all the flight management functions and data collection. It consists of the following three components:

- Sparc II Workstation: this computer contains the software package RealSim that is used to design, code, and implement a control algorithm.
- Luggable PC: this computer contains the TI C30 digital processor, and the DSP_FLEX board that holds the IP_Modules explained in Chapter II.
- Communications Box: this box houses various devices used to communicate with the unmanned aircraft.

The devices that are used for communication with the aircraft are the following: two DGR-115 spread spectrum RF modems from Freewave Inc., capable of transmission rates of 116 Kbaud at 20 miles, which are used to transmit and receive

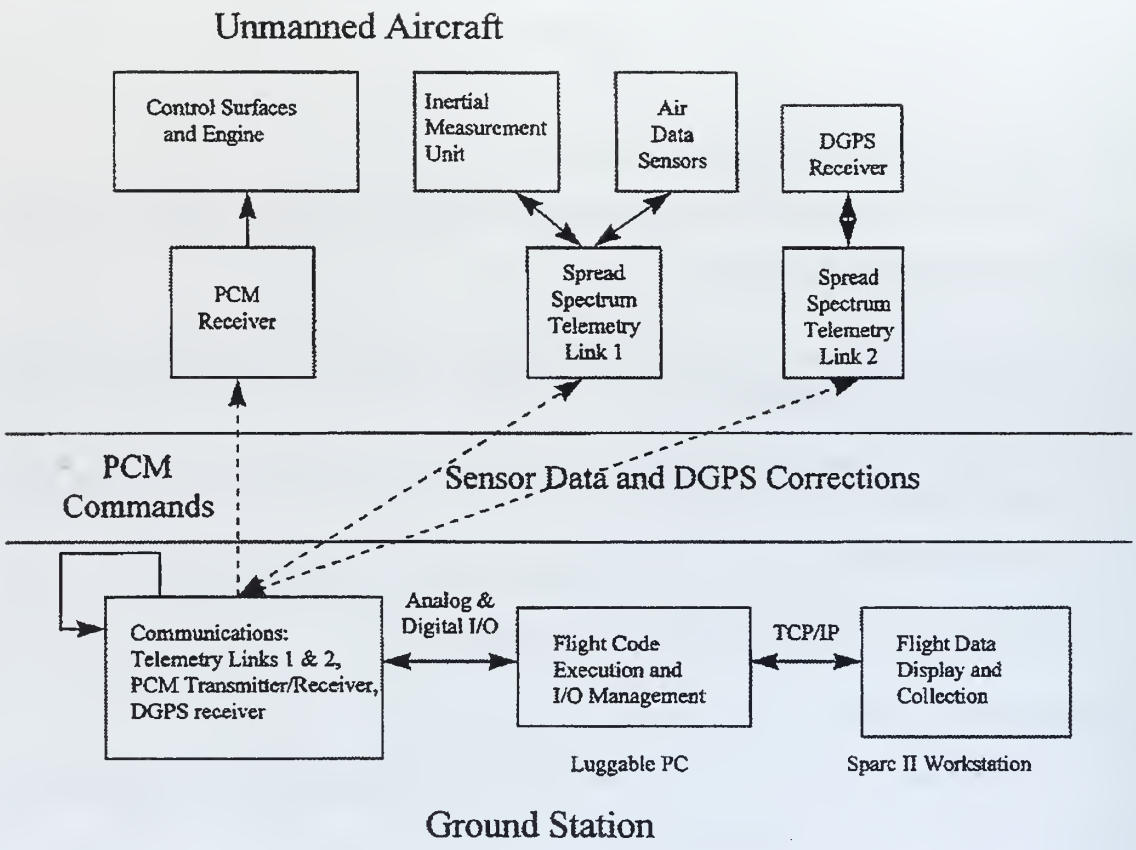

Figure 9. RPS Hardware Architecture

telemetry information, a Futaba pulse control modulated (PCM) transmitter which was modifed to give the computer the ability to control the aircraft, and a PVT-6 differential GPS (DPGS) receiver from Motorola.

The Sun workstation is connected to the luggable PC by a standard TCP/IP connection. The luggable PC is connected to the communications box by four ribbon cables from the four IP_modules.

2. Unmanned Aircraft

The unmanned aircraft is equipped with a complete avionics suite that is necessary for autonomous flight. The avionics suite consists of the following components:

an Inertial Measurement Unit (IMU) from Watson Inc., air data sensors, and a PVT-6 DGPS receiver. The data from these devices are transmitted to the ground station via two additional spread spectrum RF links from Freewave Inc. The aircraft also has a PCM receiver that is used to drive the actuators on the control surfaces and throttle. An important feature of this avionics suite is its portability; it can be easily duplicated or moved to a different platform.

B. FLIGHT TEST

Flight tests were conducted using the UAV Bluebird at an outlying field near the Naval Postgraduate School. These tests involved transporting the Sun Sparc 2 workstation Intrepid, the luggable PC AC100, the communication box, the RF antenna, the portable generator, and Bluebird to the field. To connect these components the following steps should be accomplished:

- Ensure all computer connections are made (i.e. external hard drive, monitor, etc.)
- Make all power connections to the portable generator.
- Connect TCP/IP cable from Sun to luggable PC.
- Connect the four IP_module ribbon cables to the communication box. Both the ribbon cables and the box are label to ensure proper match-up.
- Connect the RF antenna cable to communication box.

Once all the connections are made the system should be booted up. Once the Sun finishes its boot up sequence it will ask for a user ID and password. Once the system is running change to the working directory, which is /mnt/home/ archytas/realsim/baseline/flight_test_current. This is the latest version of the test setup. Start the RealSim software by typing $\ll$ *realsim* $\gg$ at the command prompt, this will bring up the RealSim GUI mentioned earlier. See Figure 10.

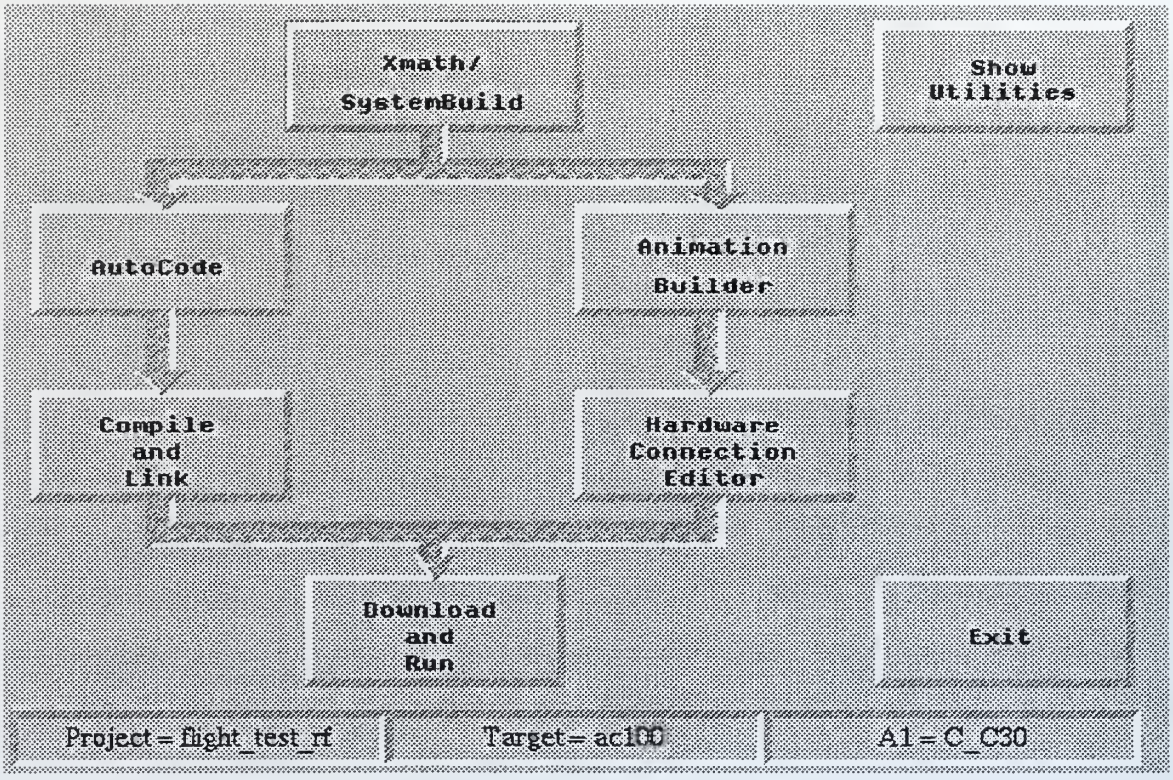

Figure 10. RealSim GUI

This GUI is slightly different than the one depicted in Figure 4 because the system has been updated and the black lines connecting the software components are now gray. To initiate a flight test complete the following two steps:

- On AC100 type $\ll ac100svr \gg$. This initiates the ftp program on AC100.
- Click on the download and run button on the RealSim GUI.

Once the download and run button has been pressed the generated C code is downloaded to the C30 DSP chip. After reaching the DSP chip the code starts to run and service the IP_modules loaded on the DSP_Flex board via the ribbon cable. The code also controls the animation screens via a TCP/IP connection to the Sun workstation.

The sytem will now display the following 2 screens, (see Figures 11, 12). The Master page allows the tester to scroll through a series of Interactive Animation

Screens that were developed using the Interactive Animation software explained in Chapter II. By using the left mouse button and clicking on a displayed icon the system will transfer to that IA screen. This allows the test engineer to accomplish such tasks as conducting calibrations, monitoring the IMU, GPS data, or viewing flight parameters during the test process. These screens are tied to the inputs and ouputs in the SystemBuild diagrams. When inputs are entered on these screens the data is incorporated in the running model and calibration equations located in the SystemBuild diagrams. Below this picture is the iaclient window that starts/stops the model. Next the steps necessary to conduct an actual flight test will be outlined.

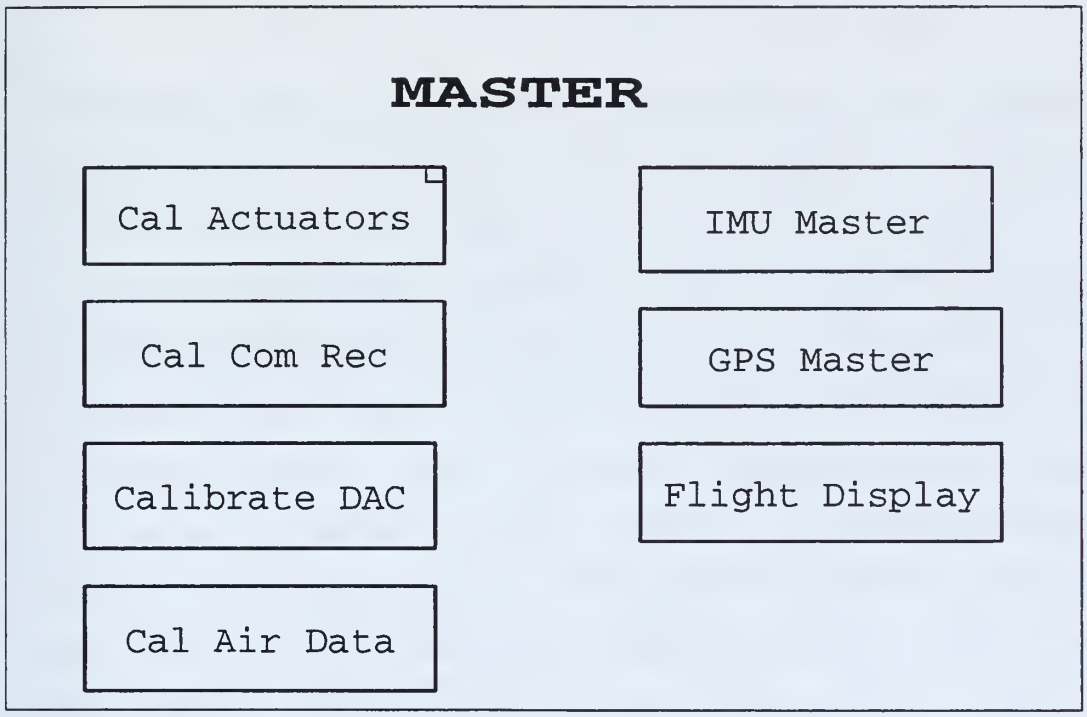

Figure 11. Master Flight Test IA Screen

C. CALIBRATIONS

Before any flight test can be conducted, it is necessary to calibrate the actuators and sensors on Bluebird to ensure accurate data transfer. The calibration

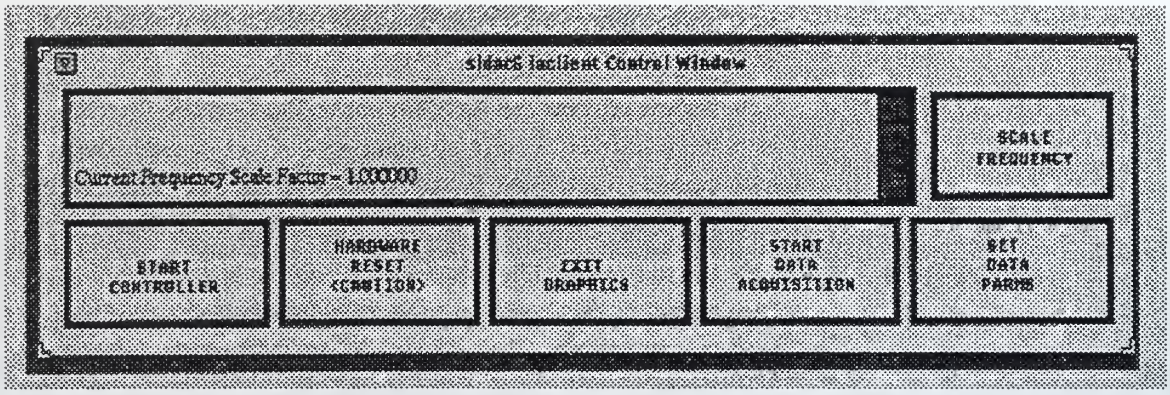

Figure 12. IAClient Screen

procedure consists of taking various measurements from the following sources: actuator pots, command receiver, and command transmitter, then comparing this data to ensure the same values are being recorded.

This procedure is accomplished by entering data into various windows of the four IA screens: Cal Actuators, Cal Com Rec, Calibrate DAC, and Cal Air Data. The data that is entered is assigned to variables that are associated with equations located in the SystemBuild diagrams. The various SystemBuild diagrams and a complete description of their use is given in Chapter V. The following subsections will descibe in detail the use of the four IA screens.

1. Cal Actuators

This screen calibrates the actuators on Bluebird. The flow of variables for the actuator calibration is outlined in Figure 13. First the voltages from the actuators are measured, passed through the IMU, and transmitted down to the ground station via the RF links. The data variables are: word1_imu, word2_imu, word3_imu and word4_imu. These four variables contain each actuator's position in volts. These variables are input into block 2 of the A_D_IMU superblock (see Figure 33) which converts them to degrees. The trim values are then added to these variables, which give the degrees of actuator deflection for the chosen flight condition. The variables that are associated with this screen are given in Table II.

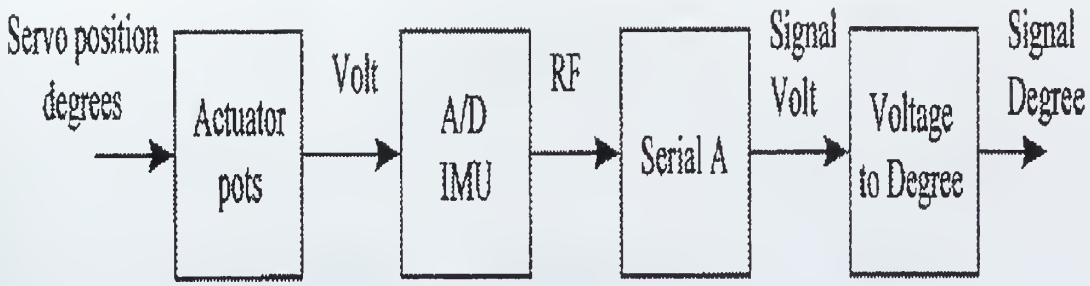

Figure 13. Variable Flow for Actuator Calibration

The first four variables are connected in the A_D_IMU Superblock to block number 97, which is a summer. The remaining four variables are outputs from the A_D_IMU Superblock and are displayed on the four dials. See Figure 14.

In order to calibrate the aircraft's actuators a zero had to be defined. This is an arbitrary location for the actuator position and was decided to be a cruise condition of 70 kts at 1000 ft. The RC pilot flew the aircraft in this flight condition and obtained trim settings. These trim settings were then read off the actuator pots

Table II. Actuator Calibration Screen Variables

VariableName	*VariableNumber*
elev_servo_trim	238
rud_servo_trim	239
ail_servo_trim	240
trt_servo_trim	241
elev_servo_deg	301
rud_servo_deg	302
ail_servo_deg	303
trt_servo_deg	304

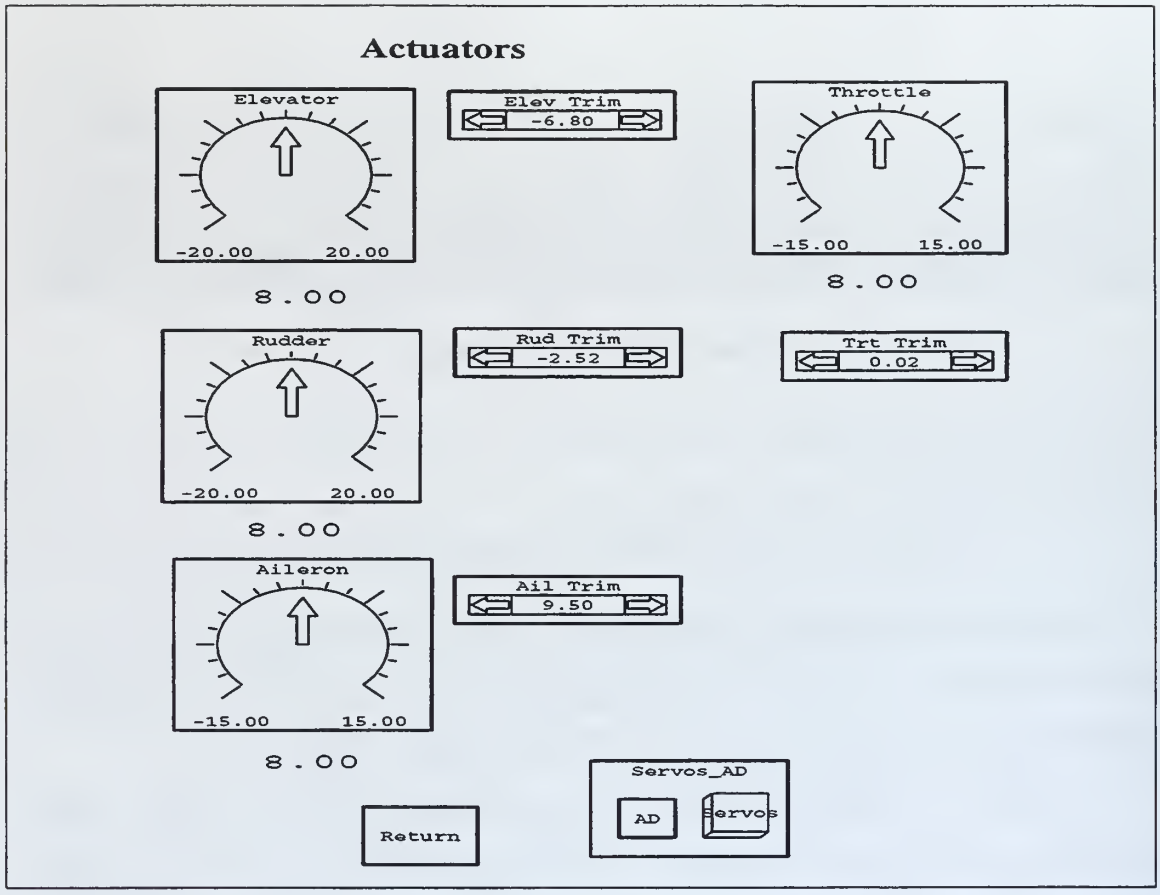

Figure 14. Actuators Calibration Screen

and were converted to degrees for the three control surfaces and percent rpm for the throttle. This data is given in Table III. This data should be entered into the Elev, Rud, Ail, and Trt Trim boxes. See Figure 14. This data is for Bluebird only, and would have to be recalculated for another aircraft or a different flight condition. Once the user has entered the data the **Return** button should be pressed to return to the previous screen.

2. Cal Com Rec

This screen calibrates the command receiver. The flow of variables for this calibration is given in Figure 15.

Table III. Actuator Calibration Data

Actuator	*Deflection*
Elevator	-6.8
Aileron	-9.5
Rudder	-2.52
Throttle	-0.02

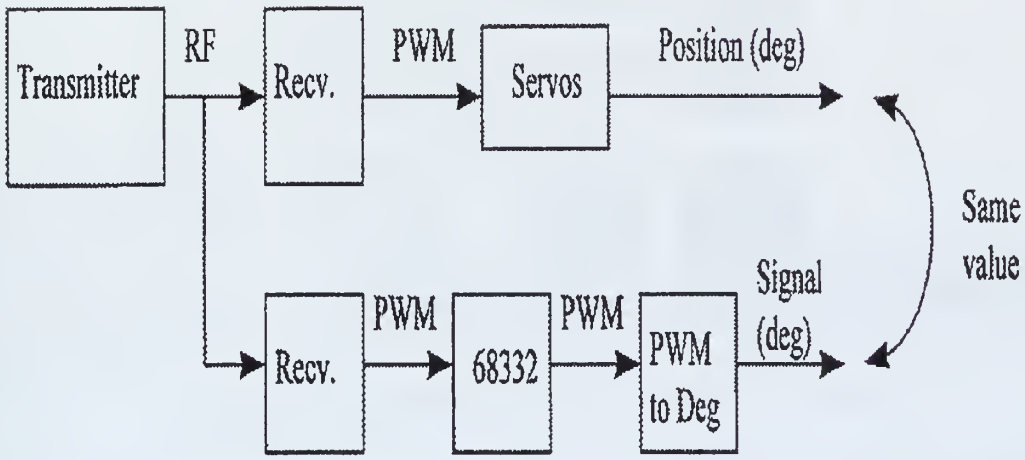

Figure 15. Variable Flow for Command Recevier Calibration

A picture of the Calibrate Command Receiver screen is given in Figure 16.

There are numerous variables associated with this screen and they are listed in Chapter V under the Command_Tx superblock. The calibration of the three control surfaces are similar, so only the steps for the elevator will be explained in detail. To calibrate the other two surfaces the user must repeat the same steps. The calibration of the throttle is slightly different, so it will also be explained.

First, the joystick for the elevator on the command receiver is moved until the number under the **Servo_Imu** column is reading -5. The variable associated with this column is, elev_servo_deg, which comes from the A_D_IMU superblock. The value displayed in the **PW(usec)** column is read off and entered into the **el_pwm_back5**

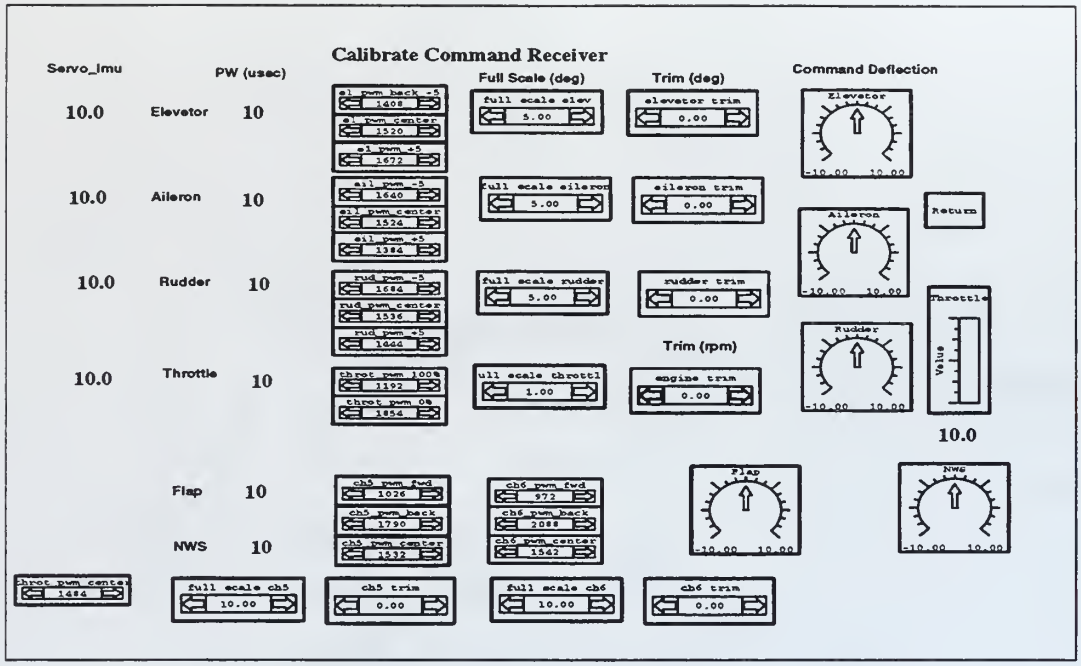

Figure 16. Calibrate Command Receiver Screen

box in the next column. This value represents the pulse width in μsec of the PWM signal coming from the receiver and is read through the PWM IP_module. The value is assigned the variable name **tp7** and is input 4 in the Command_Tx superblock (Figure 34). This variable is passed through a quantizer, (block 17), that rounds the variable off and then is output as the **elev_cmd_usec** variable which is displayed in the PW(usec) column. This procedure is repeated for 0 and +5 degrees. Once these steps are completed for the elevator calibration, they must be repeated for the other two control surfaces.

To calibrate the throttle only two settings must be entered. First move the throttle joystick to 100%. Read the PWM signal from column two and enter it into the **throt_pwm_100%** box. Then repeat the procedure for 0%. The throttle PWM signal also comes from the PWM IP_module. It is assigned the variable name **tp13** , which is input 15 in the Command_Tx superblock (Figure 34). This variable is passed

through a quantizer, (block 17), that rounds the variable off and then is output as the **trt_cmd_usec** variable which is displayed in the PW(usec) column.

The data in Table IV was collected during flight test and represents initial estimates of the PWM signals for the elevator, aileron, rudder and throttle. This data is for the UAV Bluebird and will require recalculation for a different aircraft.

Table IV. Calibrate Command Receiver Data

Actuator	−5	*center*	+5
Elevator	1270	1398	1550
Aileron	1564	1434	1292
Rudder	1702	1574	1438
–	0%	100%	–
Throttle	1794	1164	–

Once the data has been entered press the **Return** button to return to the Master display.

3. Calibrate DAC

This screen is used to calibrate the DAC signals. The flow of variables for this calibration is given in Figure 17.

A picture of this screen is given in Figure 18. There are numerous variables associated with this screen and they are listed in Chapter V under the calibrate_rf_uplink superblock. The calibration for the three control surfaces and the throttle are the same, so only the elevator will be explained in detail.

To calibrate the elevator, first flip the Cal Mode switch to the on position. Then move the elevator joystick on the command transmitter until 0 degrees is displayed in the Servo_imu and PWM_deg column. This column displays the elev_servo_deg variable from the A_D_IMU superblock and the elev_cmd_tx variable from the Command_Tx superblock respectively. Read the voltage off the dial to the left of this column which displays the voltage from the DAC IP_module and enter it into the **v_0deg** box to the right of the column. Once this value has been entered, vary the

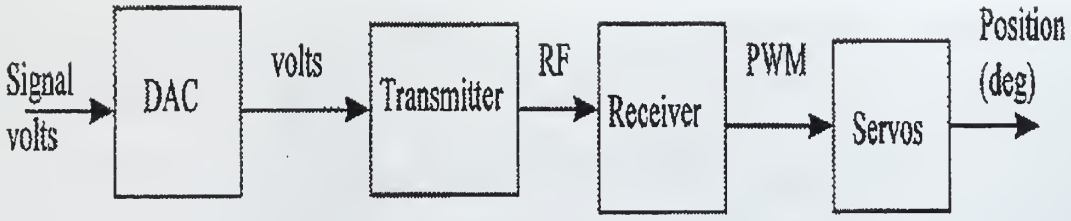

Figure 17. Variable Flow for DAC Calibration

voltage on the dial to 2.7 volts (by highlighting the number and entering 2.7) and enter the degrees from the center column into the **deg_2.7v** box. Repeat this procedure for 2.4 volts. To calibrate the other two surfaces and the throttle, repeat the same procedure.

The data in Table V was collected from test flights and represents initial estimates for the DAC Calibration. Once again these values are for Bluebird and require recalculation for another aircraft.

Table V. Calibrate DAC Calibration Data

Actuator	2.7	0	2.4
Elevator	-8.2	2.37	0.89
Aileron	-9.35	2.35	-1.3
Rudder	5.57	2.52	-3.52
Throttle	0.66	2.25	0.21

Once the data has been entered press the **Return** button to return to the previous display.

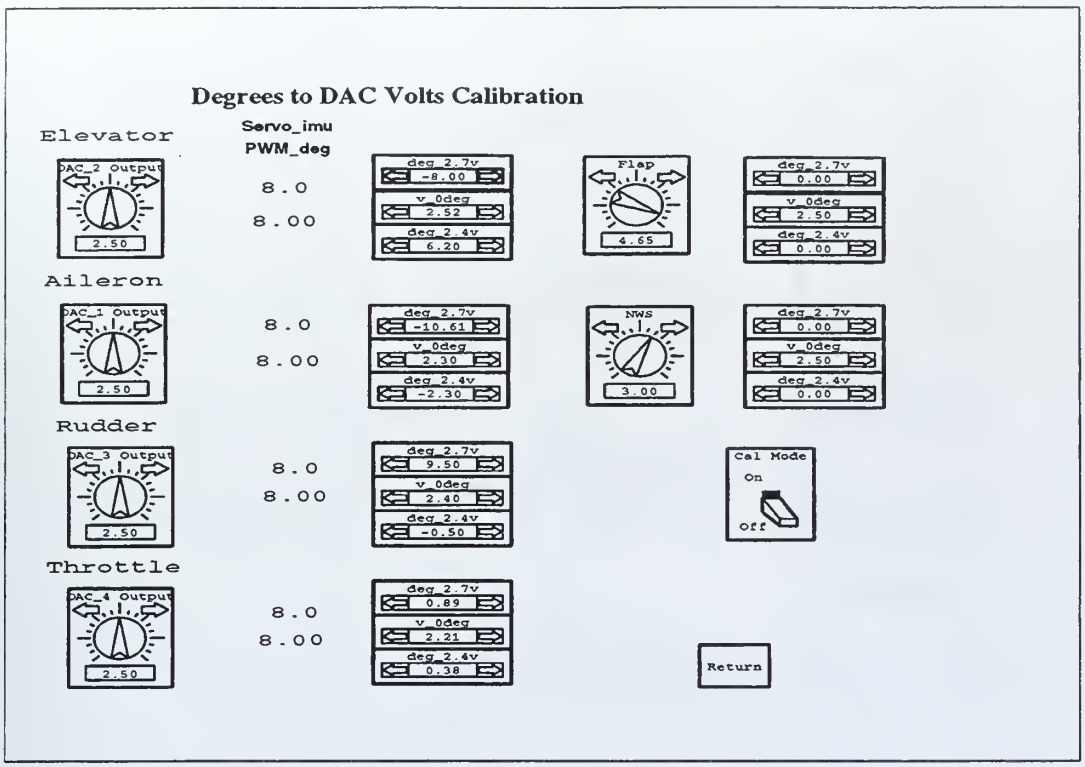

Figure 18. Calibrate DAC Screen

4. Cal Air Data

This screen calibrates the four air data sensors: alpha, beta, velocity(vt), and altitude. A figure of this screen is given in Figure 19. The variables associated with this screen are connected to the A_D_IMU superblock which is explained in detail in Chapter V. Since the IMU has only a five word capacity for data transmission these calibrations are not connected. The equations for the sensors are located in the A_D_IMU superblock and can be connected after calibrating the actuators. To receive this data from the aircraft wires connecting the actuators to the pins must be switched to the air data sensors.

D. DATA DISPLAYS

The three buttons located on the right of the Master IA screen (Figure 11) connect the user to addition screens that give data on the IMU, GPS, and Flight. A

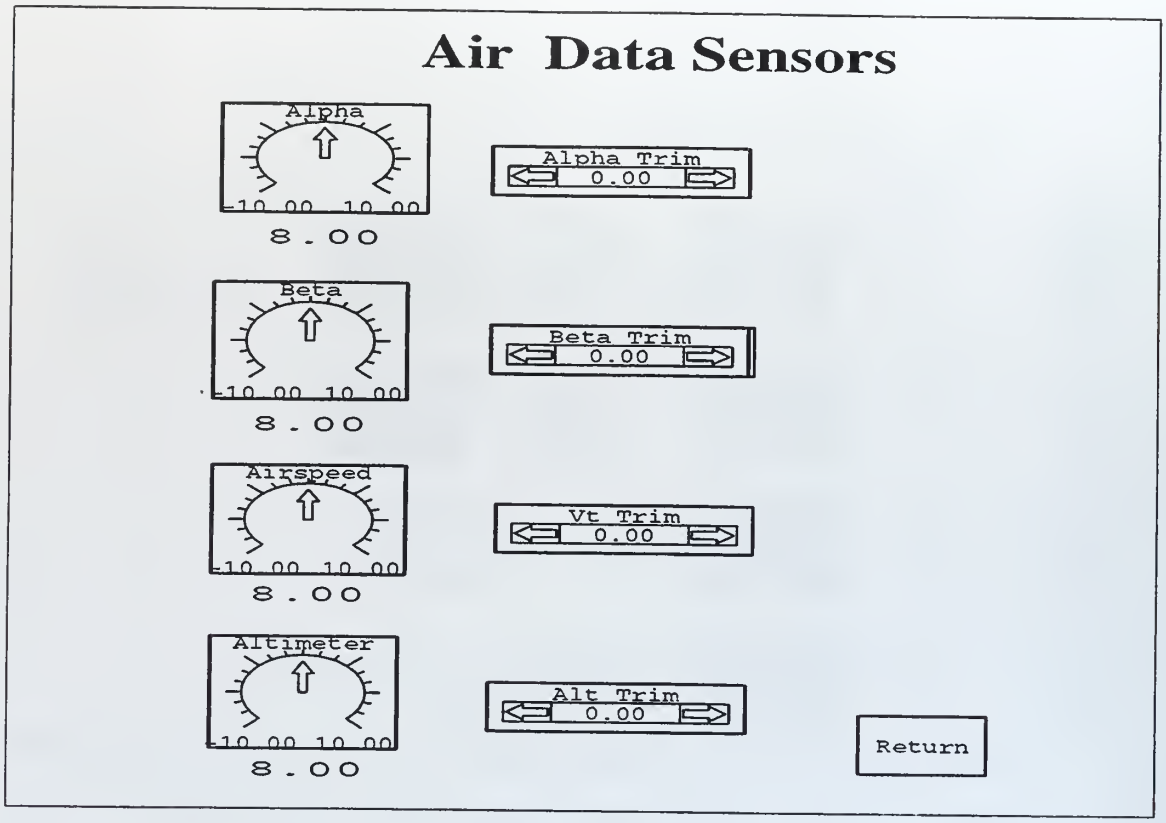

Figure 19. Air Data Sensor Calibration Screen

short description of these displays is given below.

1. IMU Master

This screen displays information concerning IMU data. See Figure 20. The IMU screen displays information for the nine states coming from the IMU: ax,ay,az,p,q,r,phi,theta, psi. In addition to the states the five words being sent from the IMU to the controller are also displayed. Currently these five words are assigned to: elevator, aileron, rudder, and throttle voltages. The fifth word is not connected. These words can be changed by connecting the wires from the different sensors to the five pins located on Bluebird. The variables associated with this screen are connected to the **imu_logic** superblock located one level down from the Sensor calibration_Display superblock.

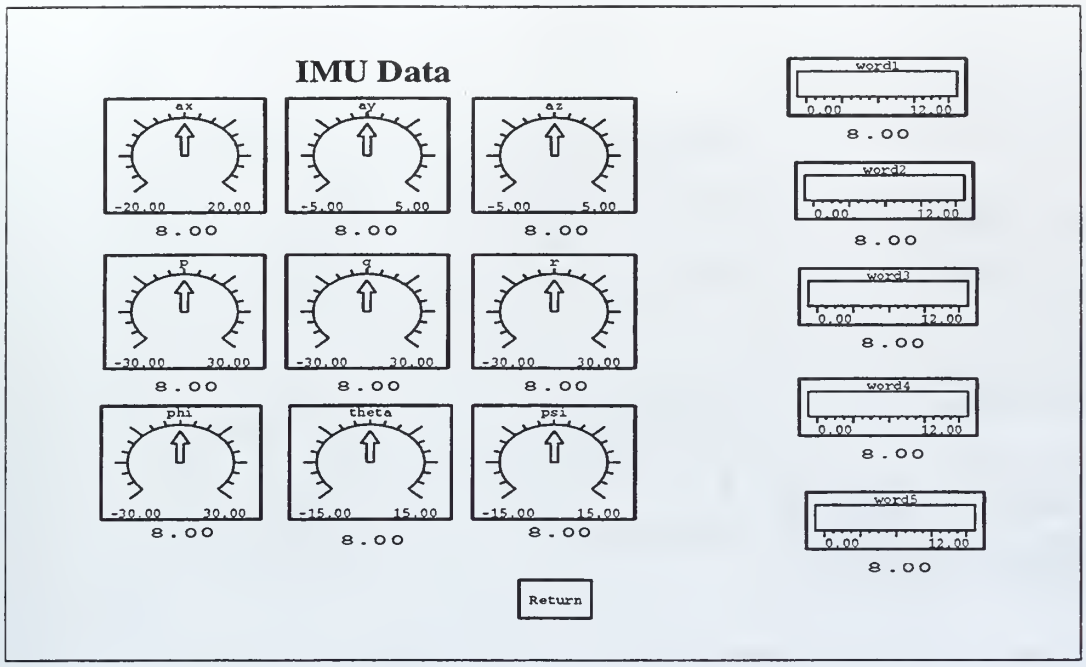

Figure 20. IMU IA Screen

2. GPS Master

The GPS screen displays information relating to the differential GPS system loaded on Bluebird. There is one receiver and antenna located on Bluebird and the second receiver and antenna are located at the ground station. This screen provides information from the GPS receiver such as, latitude, longitude, velocity, heading, and a comparison of GPS height and pressure altitude. See Figure 21.

There are also two other GPS IA screens that provide information on differential GPS operation and Earth Centered Earth Fixed(ECEF) data. See Figures 22 and 23. The variables associated with these screens are connected to the **gps** superblock located one level down from the Sensor calibration_Display superblock.

3. Flight Display

This screen is the main flight test page. See Figure 24. This screen contains switches, dials and displays to monitor flight test data. It is divided into five main sections, each concerning a certain set of data or switchology. The first section in

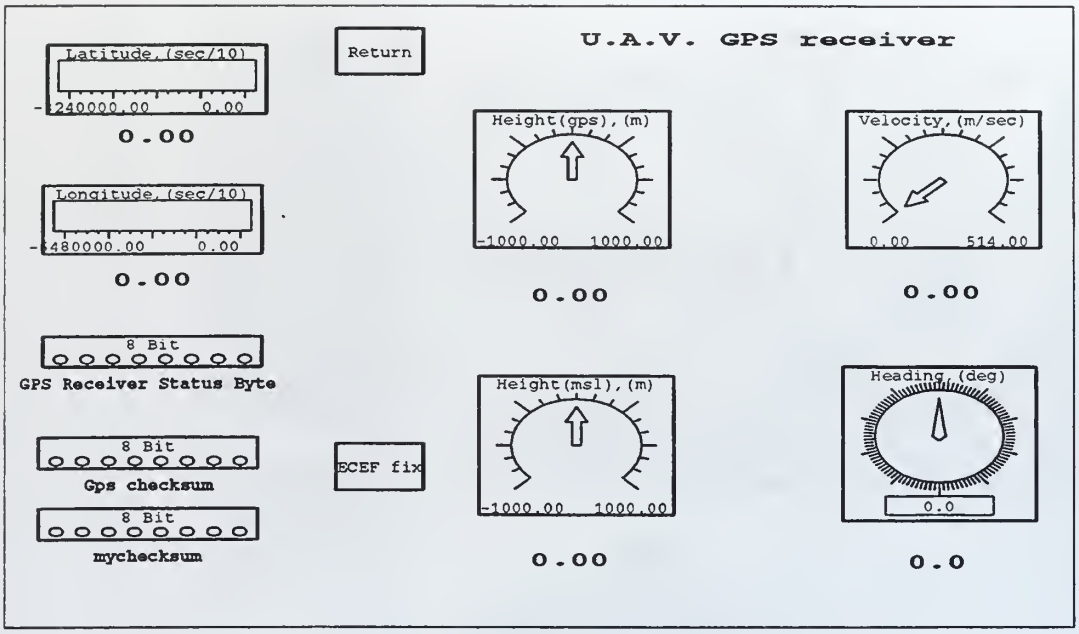

Figure 21. GPS Master IA Screen

the upper left corner contains the following three switches to toggle among: an open or closed loop controller, external comands, and Hardware-in-the-Loop testing. The next section displays the actuator commands for each actuator. The variables associated with the displayed numbers come from the Control_block, Command_Tx and A_D_IMU superblocks. The States section gives either IMU data or model states depending on the selection of the Display/Use switch located below the display. The Commands section compares the gauge reading from the aircraft to what the computer is commanding for aircraft altitude, airspeed or bank angle. The three windows in the center allow the test engineer to increase altitude, airspeed or bank angle in increments by depressing the arrows.

E. ACTUAL FLIGHT TEST

The flight test is commenced by pressing the **START CONTROLLER** button on the iaclient window with the left mouse button (Figure 12). This starts the

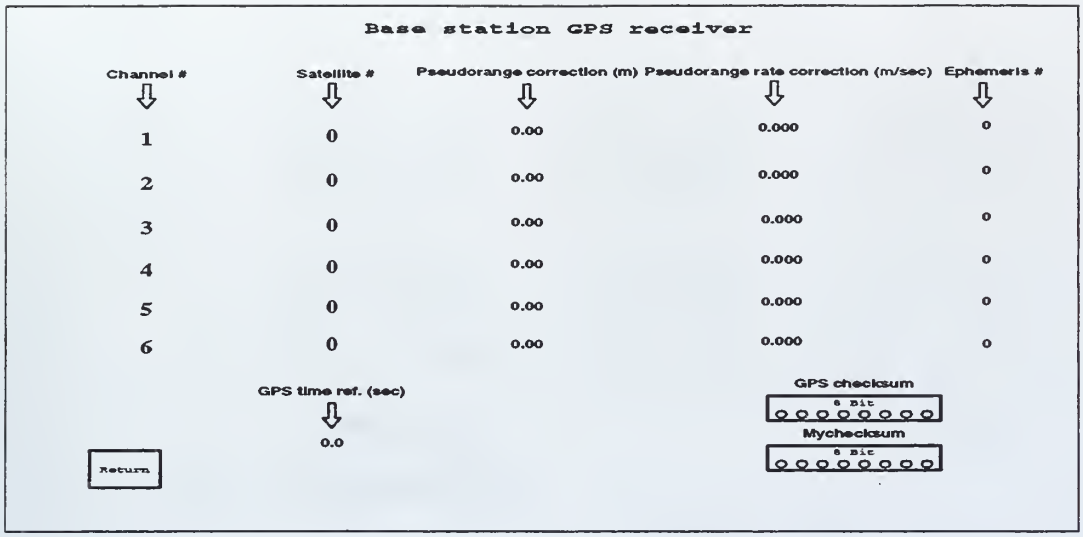

Figure 22. Differential GPS IA Screen

controller and model. If data is to be collected the data parameters must be entered by first depressing the **DATA PARAMETERS** button. This gives a screen similar to the HCE and by turning on or off the variables, data in collected. See Figure 25.

Once the parameters have been set, click on the **DATA ACQUISITION** button and data collection commences. At this point the RC pilot taxies the aircraft and takes off. Once the aircraft is established in a cruise configuration, control is handed off to the computer controller by turning on the computer's transmitter and turning off the RC pilot's transmitter. If there is a problem with the controller, control can be returned to the RC pilot by reversing this process. Data is collected until the DATA ACQUISITION is again depressed. This must be accomplished before the contoller is stopped or data will be lost. At the termination of the flight the RC pilot lands the aircraft and the controller is stopped.

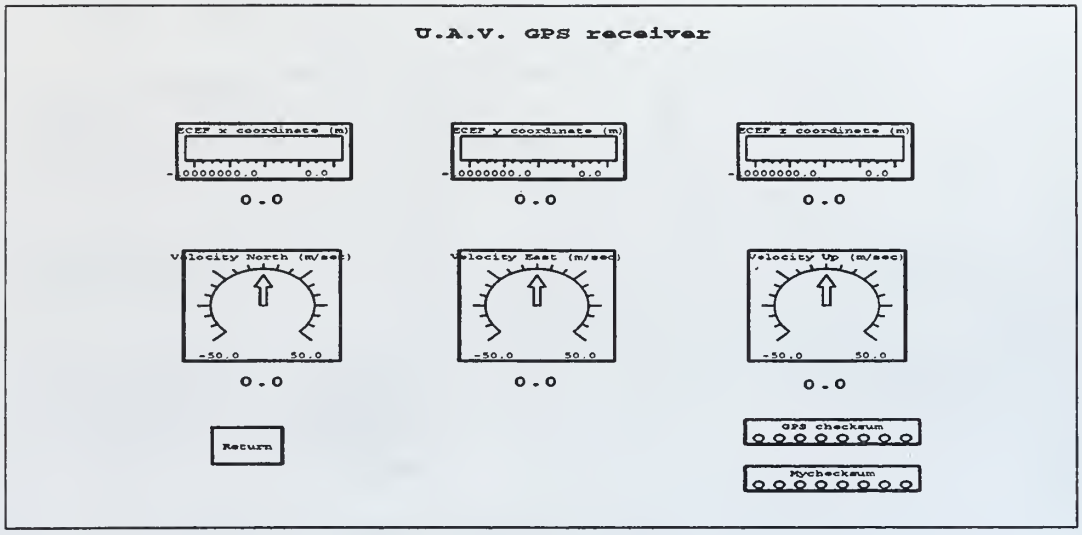

Figure 23. ECEF GPS IA Screen

F. DATA ANALYSIS

A series of flight tests were conducted using the unmanned air vehicle Bluebird to validate the calibration equations and the capability of the RF uplinks/downlinks. Data was collected using the data acqusition procedure described in Chapter IV. Once the **Stop Data Acquisition** is pressed the data is saved in a file using the model name followed by a **.raw** extension. This data must be converted to a format that can be used by Xmath. This is done by depressing the **Convert DA Data** button on the RealSim GUI (Figure 4). When the conversion is complete the data files will have a **.dat** extension.

1. Test Data

The data collected for the Bluebird test showed a 0.2 sec delay from the time the control input was sent from the computer to the time it reached an actuator. This delay is shown in Figure 26 for the elevator actuator.

The flight test data plotted is for the variables dac1_volt, tp7 and word2_imu.

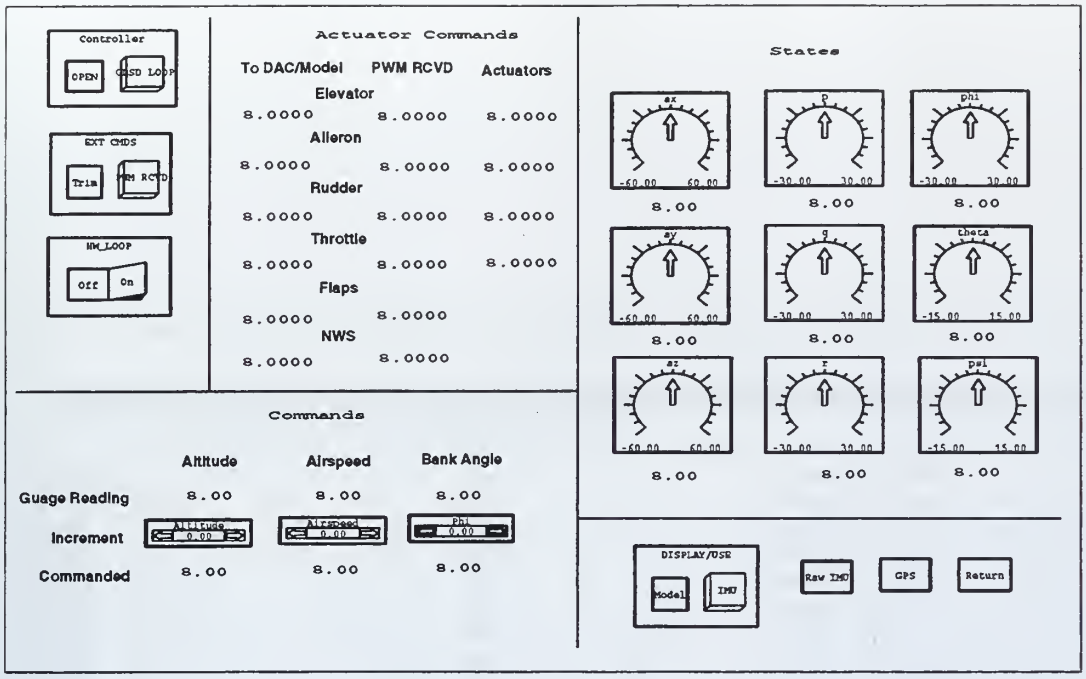

Figure 24. Master Flight Test IA Screen

These three variables measure commanded DAC voltage sent to the transmitter, the PWM signal received by the ground station, and the voltage measured at the actuator pot. This data has been normalized for ease of comparison. Studying the graph, one can see that a 0.1 second delay occurs from the time the voltage is sent from the DAC to the tramnsmitter and a second 0.1 second delay occurs from the time time the transmitter sends the PWM signal until the actuator registers a movement. The cause of this delay is most likely due to the time it takes the Futaba Transmitter to convert a voltage to a PCM signal on the ground, and for the the Futaba receiver to convert the PWM signal into an actuator command.

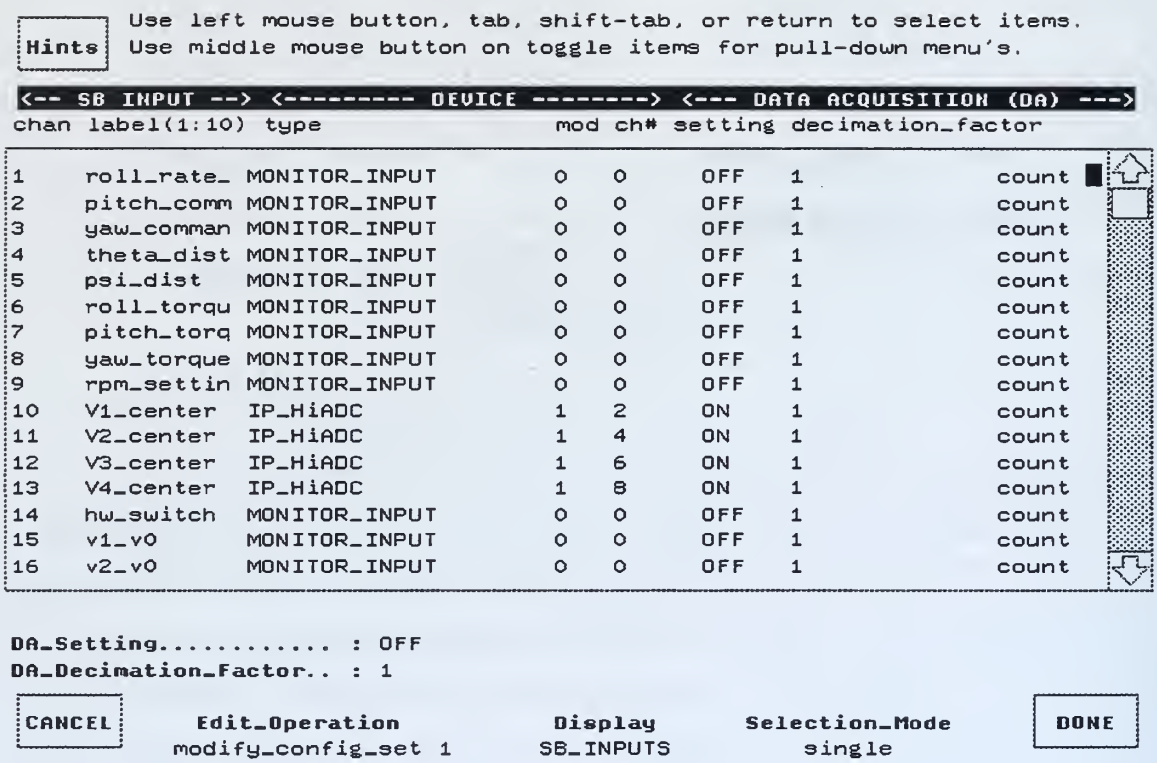

Figure 25. Data Acquisition Screen

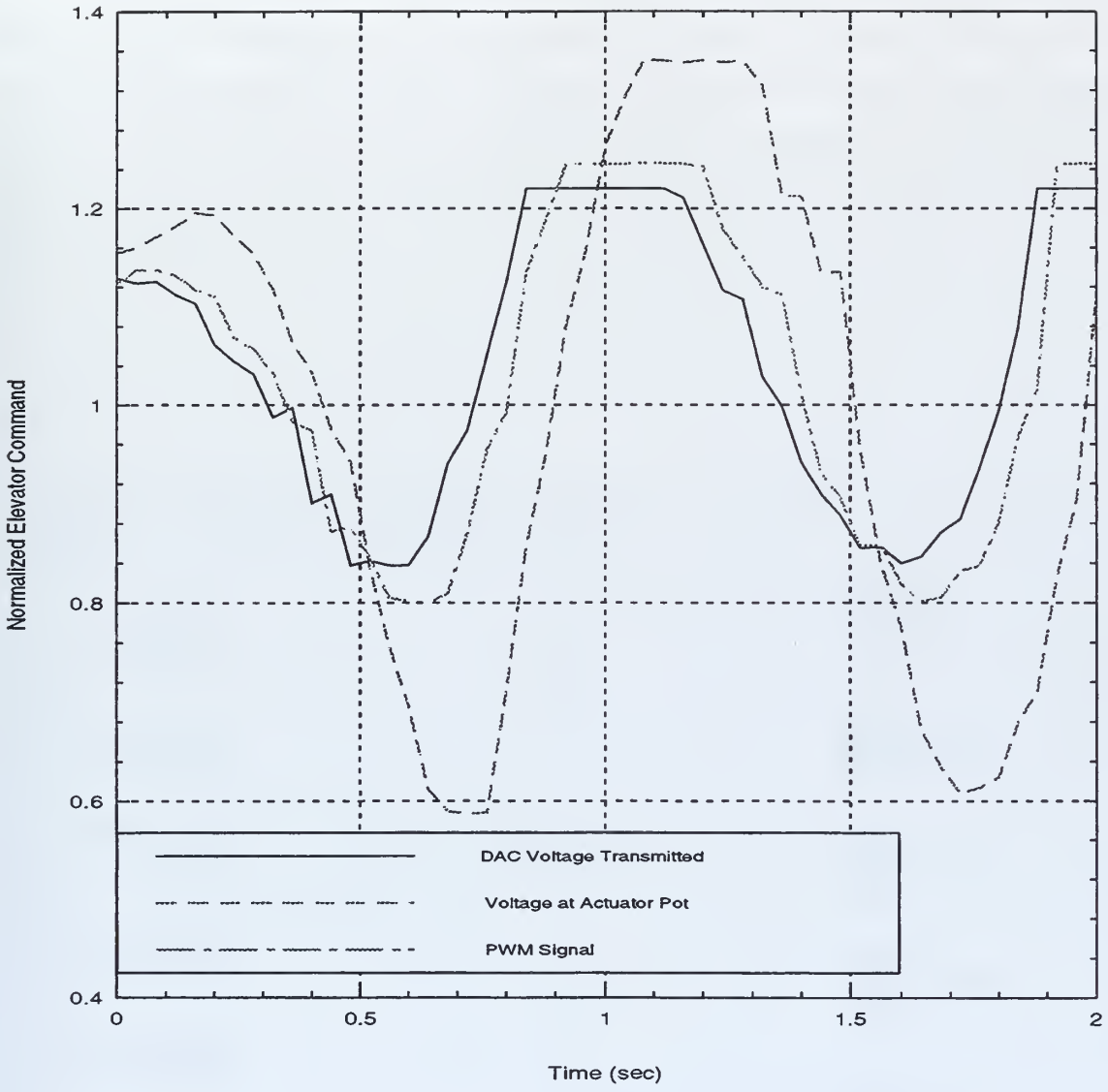

Figure 26. Control Delay for Elevator Actuator

V. IMPLEMENTATION

A. BACKGROUND

Numerous attempts were made to develop a universal outer shell using the hardware and software explained in the previous chapter. Problems arose when the number of inputs/outputs changed, additional switches were needed, or additional monitor commands were needed to expand the model. Therefore a universal outer shell was developed which is capable of growth and expansion. This outer shell is shown in Figure 27.

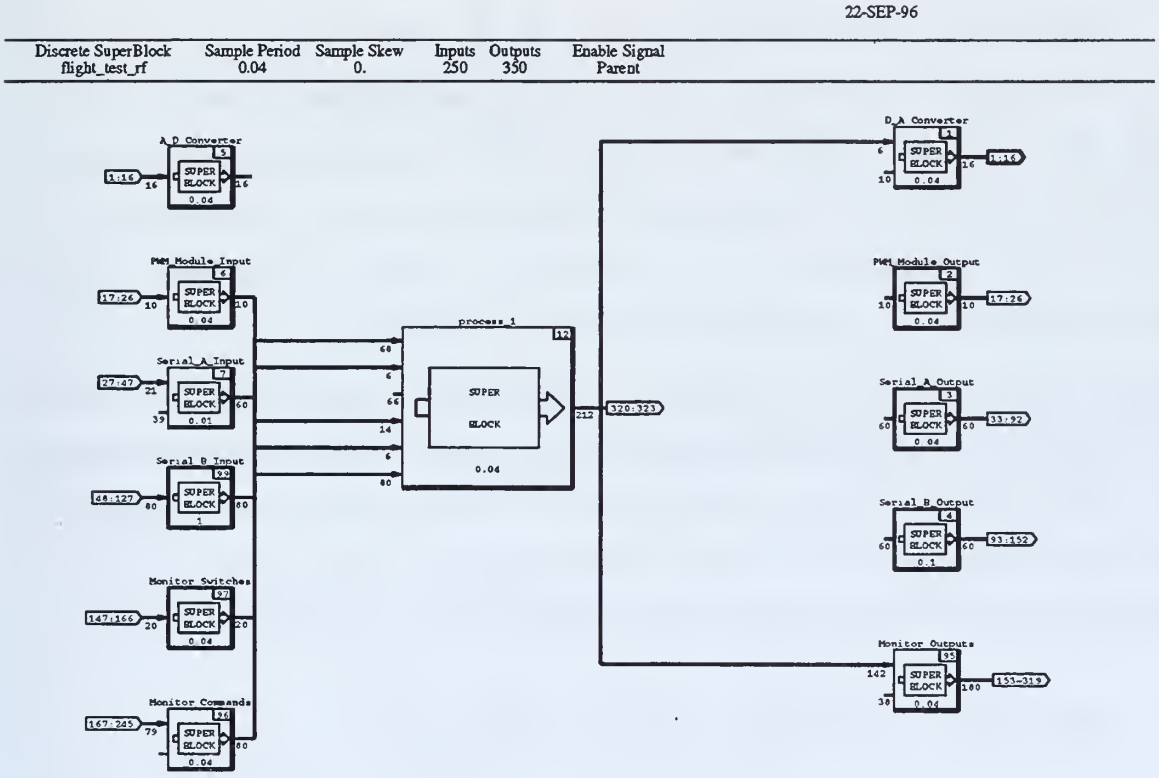

Figure 27. OuterShell

This shell has the capability to handle 250 inputs and 350 outputs. The inputs and outputs are divided among the IP_modules and monitor commands which are given in Table VI. Currently only the input/output variables needed are connected. This saves time in the compiling process and keeps the variables in a logical order.

Table VI. Input/Output Configuration

Inputs	*Outputs*
16 A/D	16 D/A
10 PWM	10 PWM
60 Serial A	60 Serial A
80 Serial B	60 Serial B
20 Switch	–
80 Monitor	180 Monitor

These inputs/outputs are connected to the Process_1 superblock, which contains four additional superblocks: **Sensor Clibration_Display, Control_block, Sensor Filtering, and calibrate_rf_uplink.** Each of these blocks performs a specific function and each will be explained in the following sections. See Figure 28.

One of the nice features of this structure is its flexibility. If the engineer designs a new controller for the aircraft all he has to do is disconnect the old controller and connect the new controller. This usually involves only a few mouse clicks. Then using a pull-down menu in SystemBuild, real time code is generated and using AutoCode the C code to drive the hardware is produced. This whole process can be accomplished in a matter of minutes, where it used to take months for a group of programmers to develop the C code for the hardware before testing can be accomplished.

B. SENSOR CALIBRATION_DISPLAY

It is necessary to calibrate the actuators and sensors on Bluebird in order to obtain accurate readings for the controller and to place the signals in the proper format. The actuator pick-offs on Bluebird use voltages to sense control surface

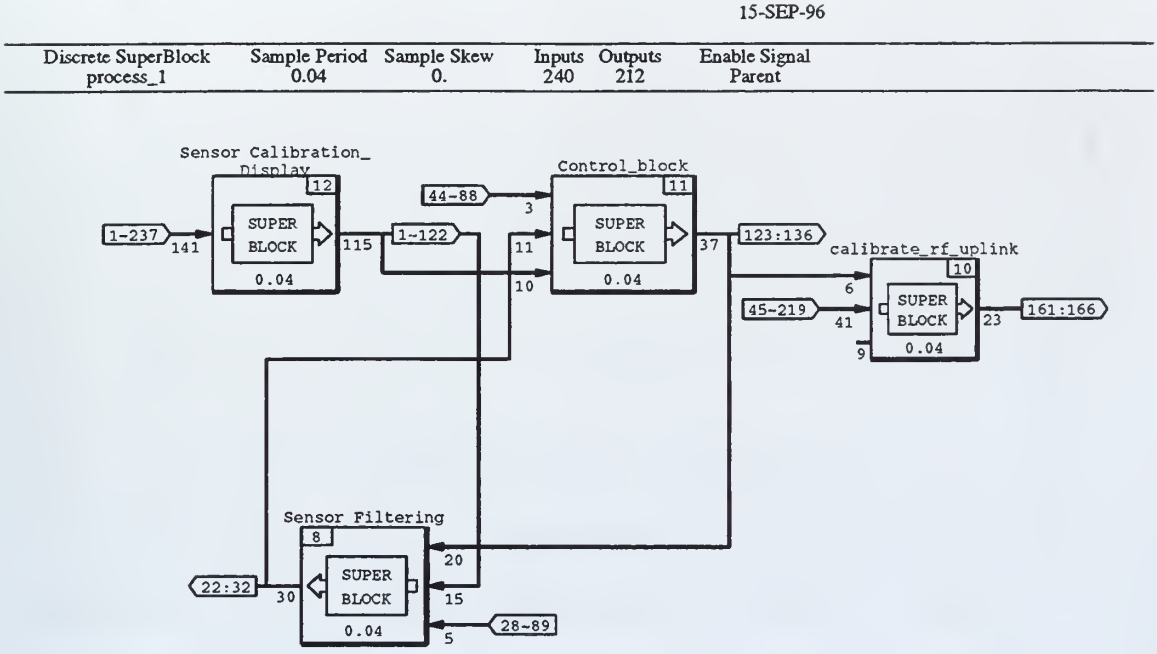

Figure 28. Process_1 SuperBlock

motion. Therefore a mapping was needed to match a certain voltage to a control surface deflection in degrees. To obtain a calibration the following steps must be accomplished.

- Using a voltmeter and a protractor measure the control deflection vs. voltage data.
- Plot this data and obtain equations for the slopes of the lines.
- Type equations into the appropiate superblocks.
- Fly the aircraft and obtain trim settings.
- Type these trim settings into the Cal Actuators, Cal Com Rec, Calibrate DAC and Cal Air Data IA screens.

Data was collected for the elevator, rudder and aileron, which was then plotted using Matlab. See Figures 29, 30, and 31.

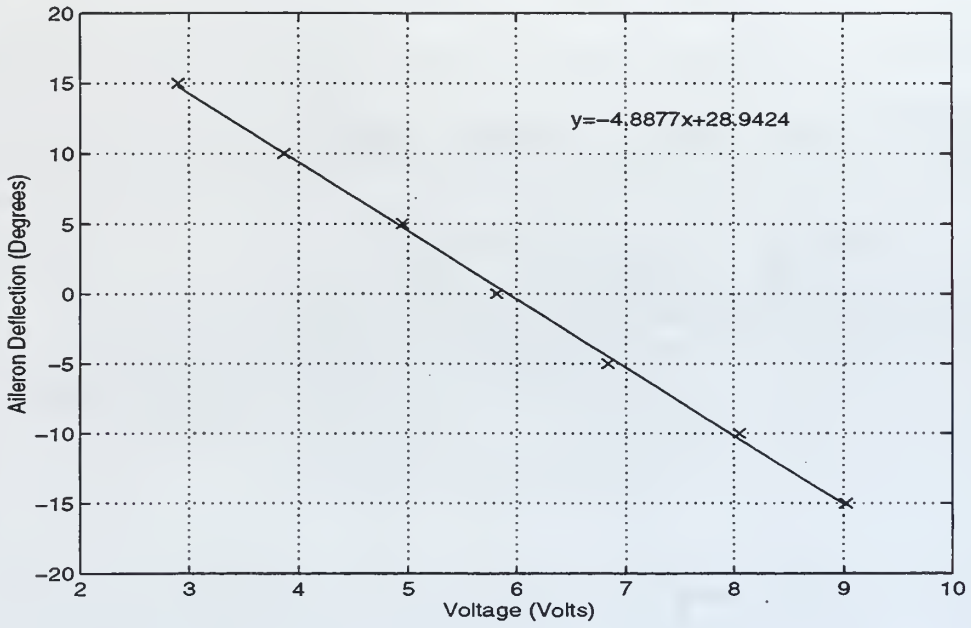

Figure 29. Aileron Calibration

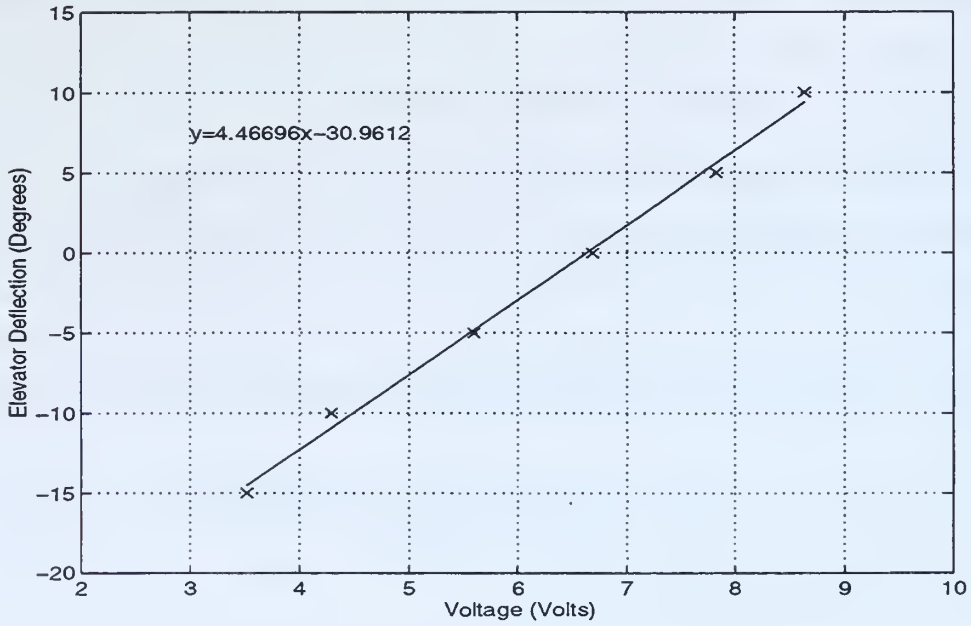

Figure 30. Elevator Calibration

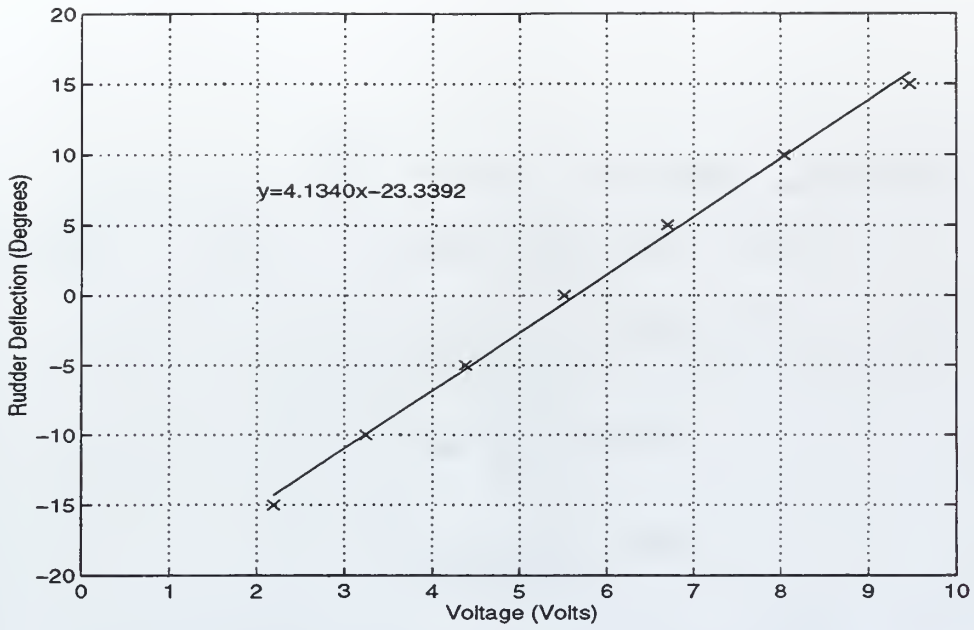

Figure 31. Rudder Calibration

The data was interpolated using a least squares fit method to obtain a linear approximation that could be used in the model. The linear approximation was chosen so the data could later be scaled for trim settings in flight. If a non-linear curve fit was used every time the trim settings were changed a new set of curves would be required. Using a linear fit a new trim setting will simply shift the curve up or down along the y-axis.

All the sensor and actuator calibrations are located in the **Sensor Calibration_Display SuperBlock** . Located in this superblock are the calibrations for the GPS, IMU logic, Command receiver, and air data sensors (see Figure 32).

By opening these superblocks the equations calculated by the calibration process are entered into algebraic blocks. For example, once the equations are calculated for the actuators they are entered into the **A_D_IMU** SuperBlock located two levels down from the **Sensor Calibration_Display** SuperBlock. This is reached by double clicking on the number in the upper right corner of the **Sensor Calibration_Display** SuperBlock and repeating the procedure on the **A_D_IMU** SuperBlock. A brief de-

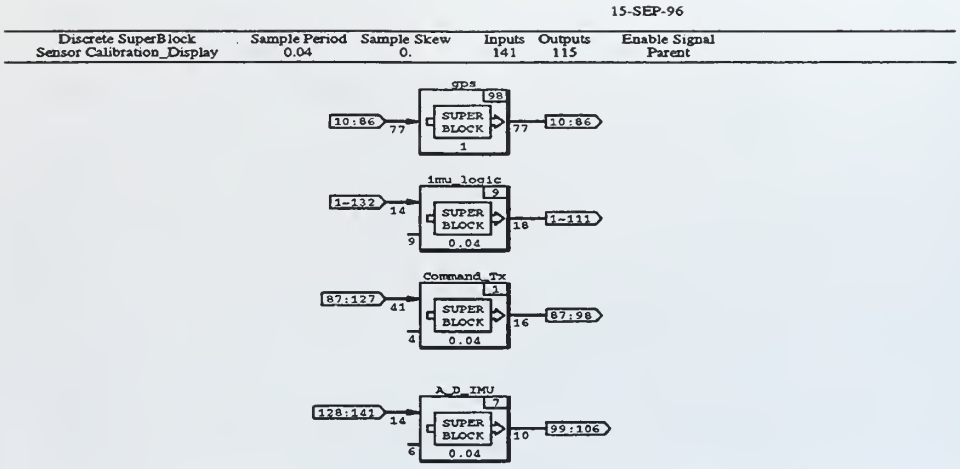

Figure 32. Sensor Calibration_Display Superblock

scription of the A_D_IMU superblock and the Command_Tx superblock are given next.

1. A_D_IMU Superblock

The A_D_IMU Superblock is used to calibrate the four actuator pots and air data sensors. See Figure 33.

The superblock has 20 input and 10 output variables and these are given in Table VII. Currently, only the first 14 inputs have variable names assigned. The remaining six inputs are intended for growth.

The calibration equations given in Figures 29, 30, and 31 are entered into block number 2 which is an algebraic block. The input variables in this block are the four IMU words, in volts, and the output variables are these inputs converted to degrees. These outputs are then sent through a switch and into a summer. At the summer, the trim values entered in the Actuator Calibration IA screen, (Figure 14), are added, and then the values are output to the Sensor_Calibration superblock.

The top half of the A_D_IMU superblock is for the calibration of the air data

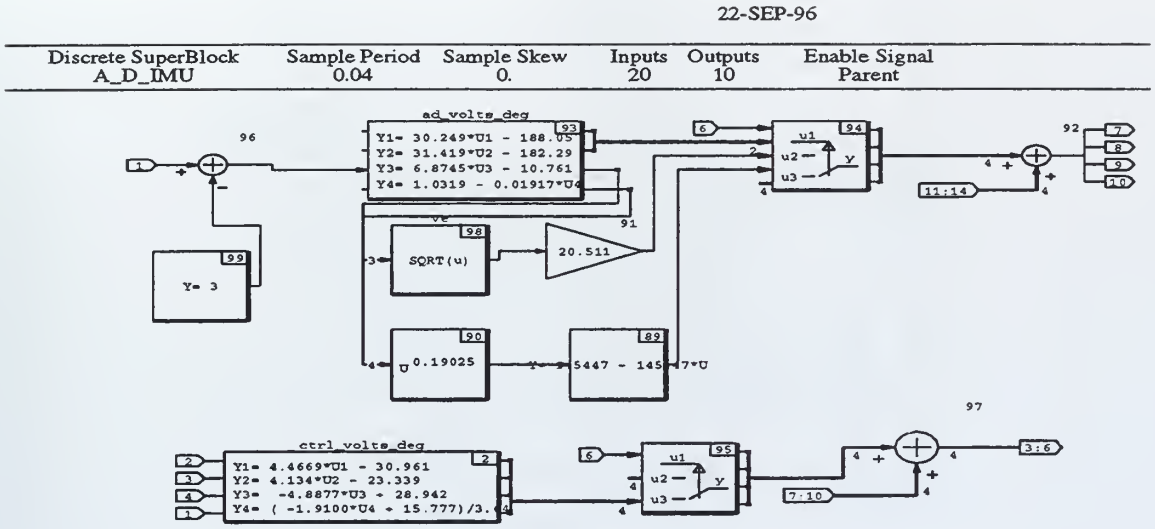

Figure 33. A_D_IMU Superblock

sensors. This section is not connected due to the limitations of the IMU which only allows five words to be transmitted to the ground station.

2. Command_Tx Superblock

The Command_Tx superblock is used in conjuction with the A_D_IMU superblock for actuator calibrations. This block has 45 input and 16 output variables. See Figure 34. Inputs 4, 8, 12, and 15 are the four PWM signals for the elevator, aileron, rudder, and throttle respectively. These four variables are input into a quantizer which rounds off the values. The variables are then output to two locations: the Calibrate Command Recevier IA screen (see explaination in Chapter IVchap5.tex) and the elevator, aileron, rudder and throttle shift blocks.

The shift blocks take the PWM values from the receiver and subtract off the PWM value for the center location of the joystick. This is necessary to put the PWM

Table VII. A_D_IMU Input/output Variables

Number	Input name	Output name
1	word1_imu	–
2	word2_imu	–
3	word3_imu	elev_servo_deg
4	word4_imu	rud_servo_deg
5	word5_imu	ail_servo_deg
6	imu_servo_sw	trt_servo_deg
7	elev_servo_trim	alpha_imu_deg
8	rud_servo_trim	beta_imu_deg
9	ail_servo_trim	vt_imu_fps
10	trt_servo_trim	alt_imu_ft
11	alpha_imu_trim	–
12	beta_imu_trim	–
13	vt_imu_trim	–
14	alt_imu_trim	–

signals in the proper format for the scaler blocks. The scaler blocks take the PWM signals in microseconds and convert them to degrees. In addition to the shifted PWM signal, the scaler blocks also has inputs for the **PWM max up, PWM max down** variables in microseconds and **full scale deflection** variable in degrees. The values for these inputs are obtained from the Calibrate Command Receiver IA screen and are used in a formula developed in [Ref. 8] to convert PWM μseconds into degrees. After leaving the scaler blocks the PWM signals, now converted to degrees, are added to the trim values from the Actuator Calibration IA screen. These variables are then output to the Sensor_Calibration superblock.

The last two sets of blocks labelled channel5 and channel6 are used for the flap and nose wheel steering. These will be used when the aircraft is fully autonomous, i.e. computer controlled from takeoff to landing.

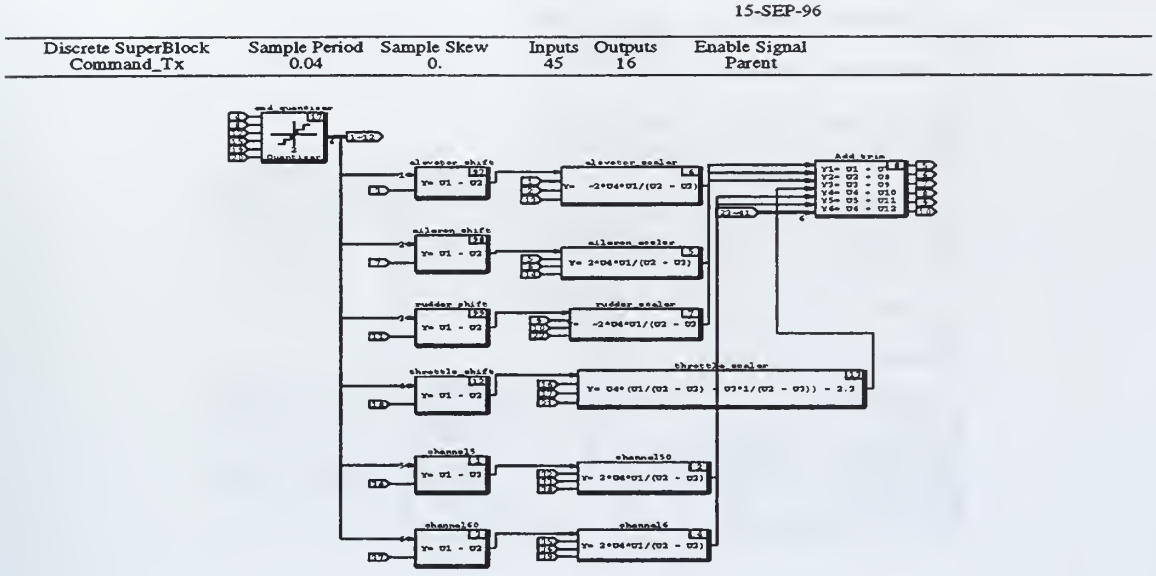

Figure 34. Command_Tx Superblock

C. CALIBRATE_RF_UPLINK

This superblock calibrates the command transmitter for the RF uplink to the aircraft. These calibrations are similar operations that are performed by the Sensor Calibration_Display superblock, except in the reverse order. It takes controller outputs, in degrees, and converts them to DAC voltages to be transmitted to Bluebird via an RF link. This superblock has 56 input and 23 output variables which are tied to the Calibrate DAC IA screen. (See Chapter IV for a complete explanation of this screen.) Expanding the Calibrate_rf_uplink superblock gives a new set of blocks which are described in Figure 35).

The first input to the **degrees to DAC voltage** algebraic blocks comes from the Control_block superblock and the remaining three inputs come from the Calibrate DAC IA screen. These inputs are used in an equation that converts degrees to DAC voltage. Once the variables are converted to degrees they are sent into a switch which

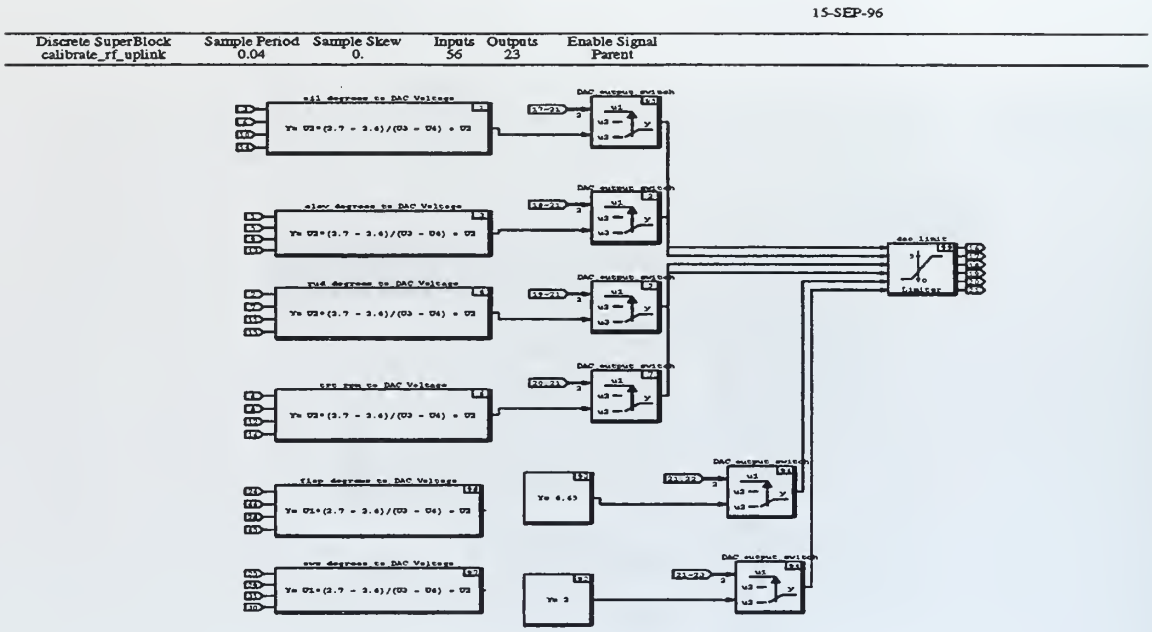

Figure 35. Expanded Calibrate_rf_uplink Superblock

is connected to the Calibrate DAC IA screen. If the switch is on the calibration mode is activated, the calibration data is displayed on the screen. If the switch is off the DAC voltages are passed through the switch into a DAC limiter that limits the DAC voltage to 5 volts. From this block the DAC voltages are outputed to the DAC superblock, which in turn sends them to the DAC IP_module.

D. CONTROL_BLOCK

This SuperBlock contains the controller, dynamics and kinematics for Bluebird. The equations that were derived in Chapter II are symbolically displayed in this superblock. Inputs are fed from the Sensor Calibration_Display superblock, Sensor Filtering superblock, and the Monitor Commands and Monitor Switches superblocks located in the outer shell to this superblock. Outputs are fed to the Calibrate_rf_uplink, where the outputs are converted back to DAC voltages for RF trans-

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
Control_block	0.04	0.	24	37	Parent

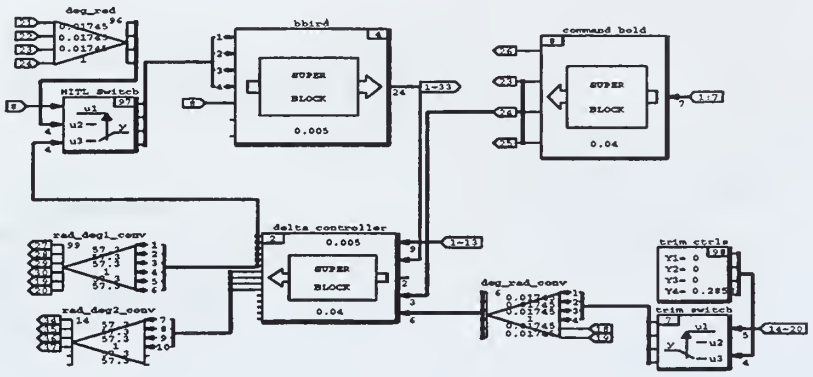

Figure 36. Expanded Control_block Superblock

mission to Bluebird, and the Sensor Filtering superblock where the signals are filtered.

Expanding the **Control_block** a new set of superblocks is given (see Figure 36).

Looking at Figure 36, some Superblocks and blocks of interest are:

- **bbird** - this superblock contains the EOM of Bluebird.
- **delta controller** - this superblock contains the delta controller developed to fly Bluebird in cruise flight.
- **command hold** - this block holds the RC pilot's last commands until control is handed over to the computer.

The remaining blocks, 96, 97, 98, 99, 14, 6, and 7 perform functions such as conversions and monitor command switchology used for flight test. A complete listing of the superblocks are given in the Appendix.

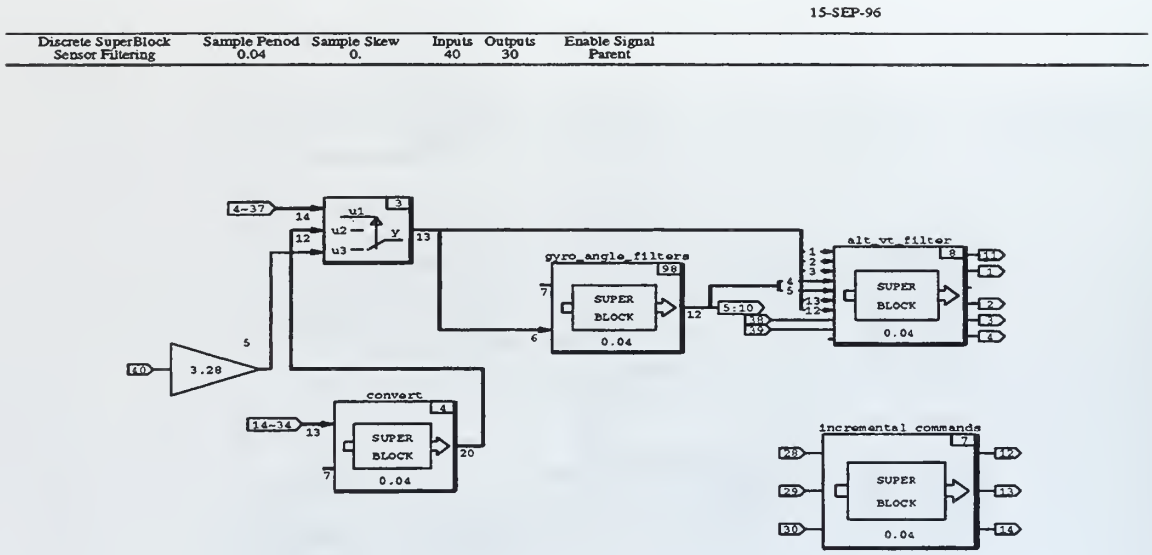

Figure 37. Expanded Sensor Filtering Superblock

E. SENSOR FILTERING

This superblock contains various filters needed for the controller. Inputs are fed from the Sensor Calibration_Display superblock, Control_block superblock, and the Monitor Commands and Monitor Switches superblocks located in the outer shell. The filtered outputs are fed to the Contol_block.

Expanding the **Sensor Filtering** superblock a new set of superblocks are given (see Figure 37).

Looking at Figure 37, some Superblocks and blocks of interest are:

- **gyro_angle_filters** - this superblock contains kalman filters that damp out the vibrations in the IMU.
- **alt_vt_filter** - this superblock contains filters to correct for errors in the pitot static system.

- **incremental commands** - this superblock contains rate limiters for altitude, airspeed and phi.

  A complete listing of the superblocks are given in the Appendix.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The MATRIX$_X$Product Family of hardware/software developed by Integrated System Inc.(ISI) is an excellent tool for the design, test and implementation of controllers. By stepping the engineer through the rapid prototyping design process the time required to implement a design concept is significantly reduced. Where it used to take years to develop a working prototype from a initial design concept, by using ISI's product the time is reduced to months.

One of the great benefits of this product is the ability to automatically program in a higher order language code such as C. This eliminates the necessity of having dedicated software programmers to develop code for a working model. This is especially true when changes are made in the controller and new code must to be generated. Now, the 10,000 lines of C code are generated in minutes.

Another benefit of this product is the ability to collect and analyze data during the flight testing process. During flight test we were able to make changes in the field without having to dismantle the entire setup and transporting it back to the lab. Using the data acquisition editor and Xmath, graphs were produced to analyze the success of the controller and filter designs. With a conventional test setup this would not have been possible.

The outershell developed in SystemBuild greatly increases the understanding and tracking of the model variables. Prior to this setup every time the model was changed or variables added the outer connections had to be modified. Now, using the outershell with its built in growth capability, it is easy to add variables and monitor commands.

B. RECOMMENDATIONS

Based on the conclusions presented above and the experience of developing the uniform system presented in this thesis, the following recommendations are forwarded.

- Test the uniform system with another aircraft model. Due to time constraints and aircraft availability, Bluebird was the only UAV that was tested using the system. Another UAV, called the FROG, is being set up for testing and could be used for this purpose.
- Provide a realtime animation of the test aircraft in flight. When flight test are being conducted it is very difficult to see the aircraft due to its small size. Using a 3D simulation, such as that provided by Designer's Workbench, would give the test engineer the ability to better monitor the aircraft's flight. This will involve significant expertise in C programming and TCP/IP networking.
- Purchase a portable Unix based workstation. Currently it is necessary to transport a full size Sun workstation to the test site. This is difficult due to the large size and sensitivity of the hardware. We have already burnt out a monitor and frayed a SCSI cable during the transport process.

APPENDIX. ADDITIONAL SYSTEMBUILD DIAGRAMS

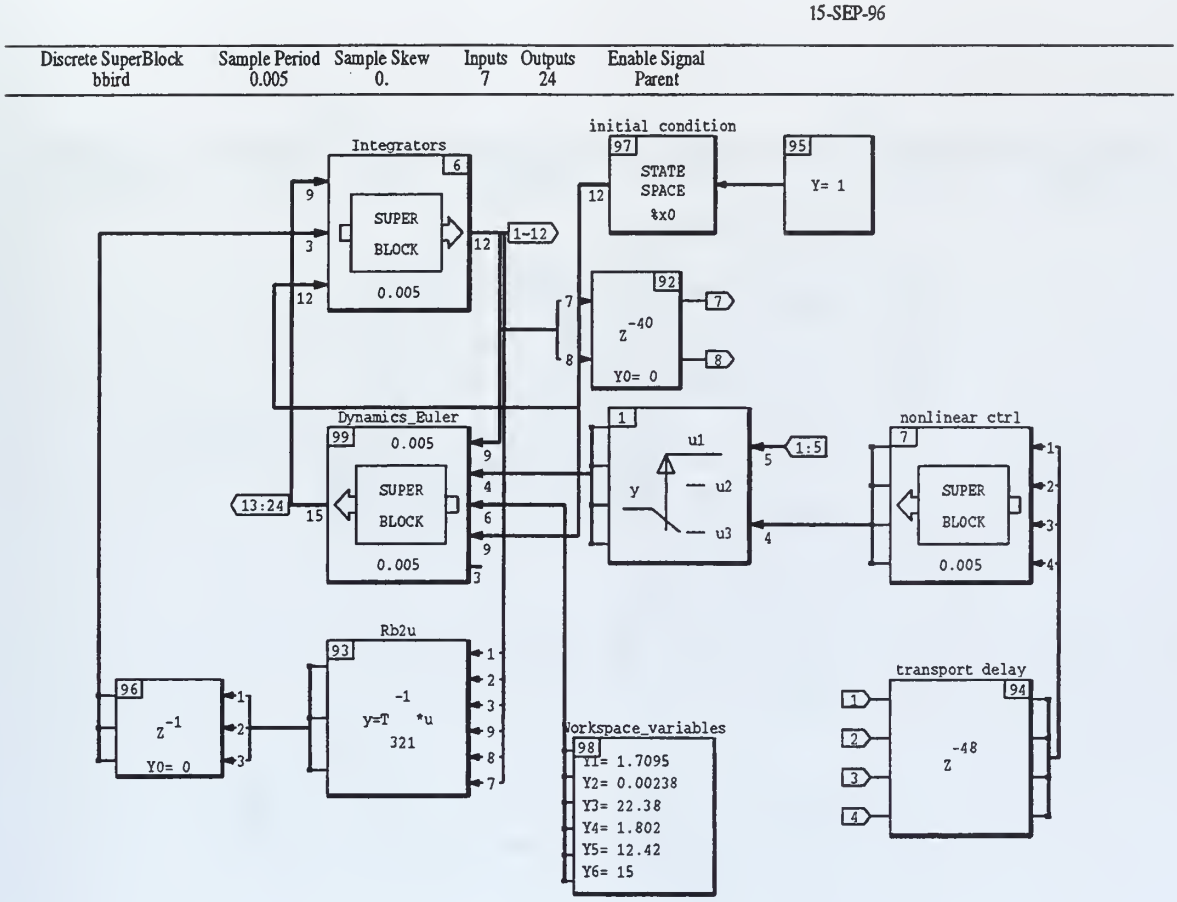

Figure 38. bbird

S

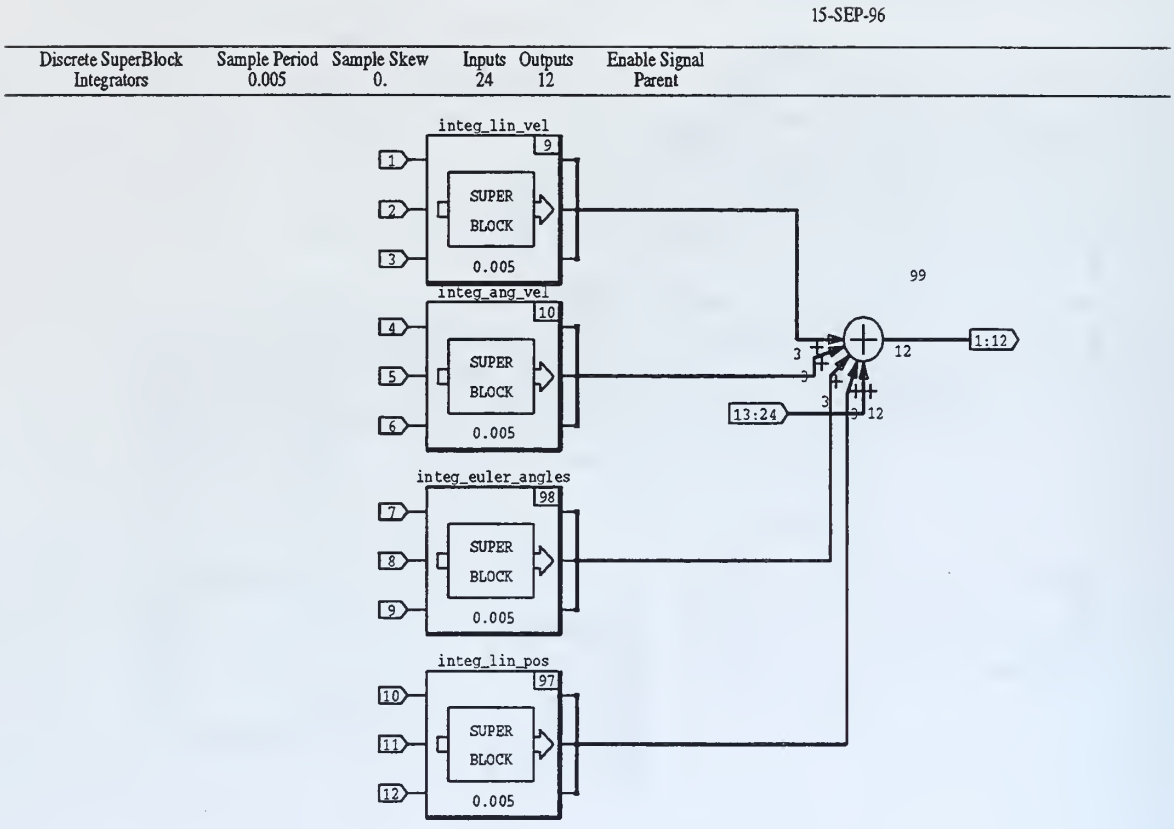

Figure 39. Integrators

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
Dynamics_Euler	0.005	0.005	31	15	Parent

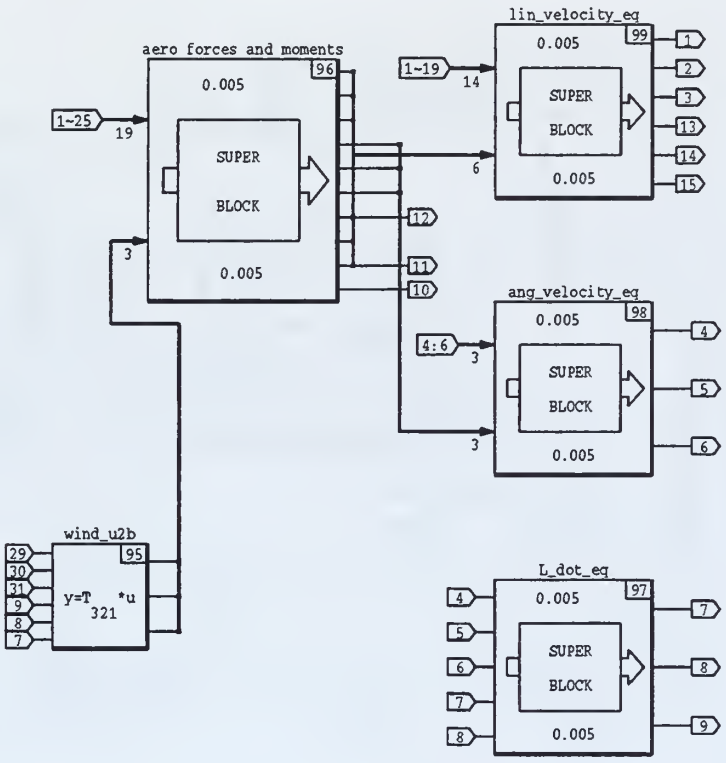

Figure 40. Dynamics_Euler

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
aero forces and moments	0.005	0.005	22	10	Parent

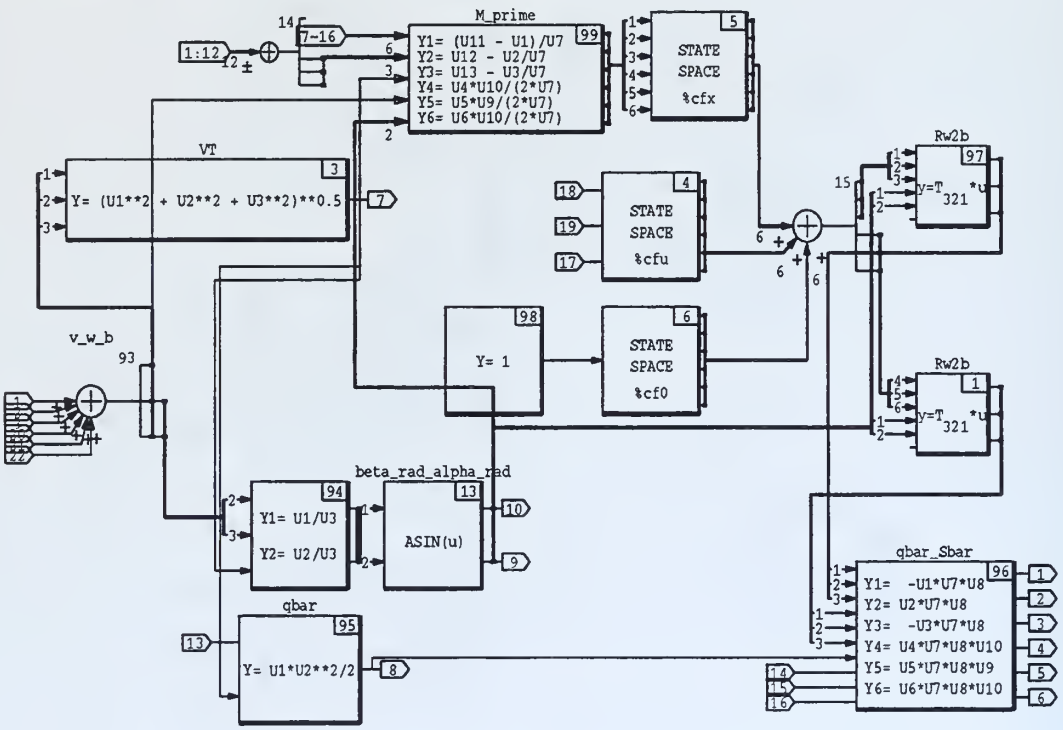

Figure 41. aero_forces_and_moments

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
lin_velocity_eq	0.005	0.005	20	6	Parent

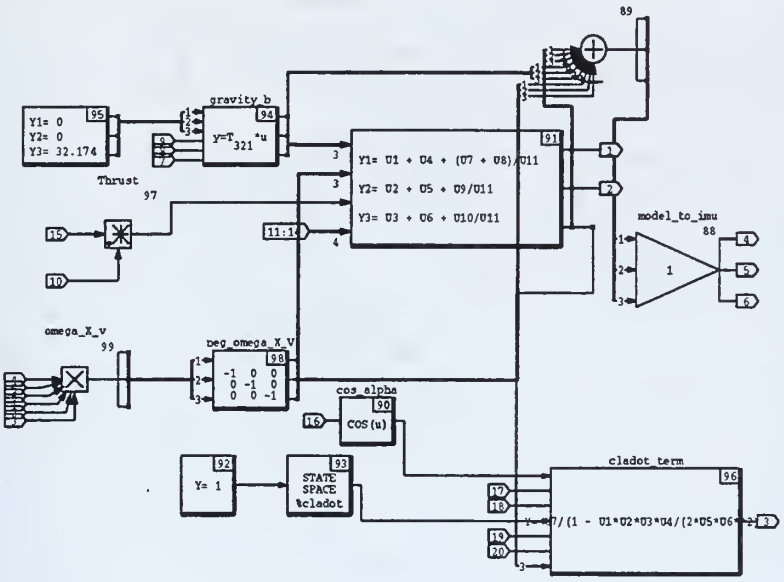

Figure 42. lin_velocity_eq

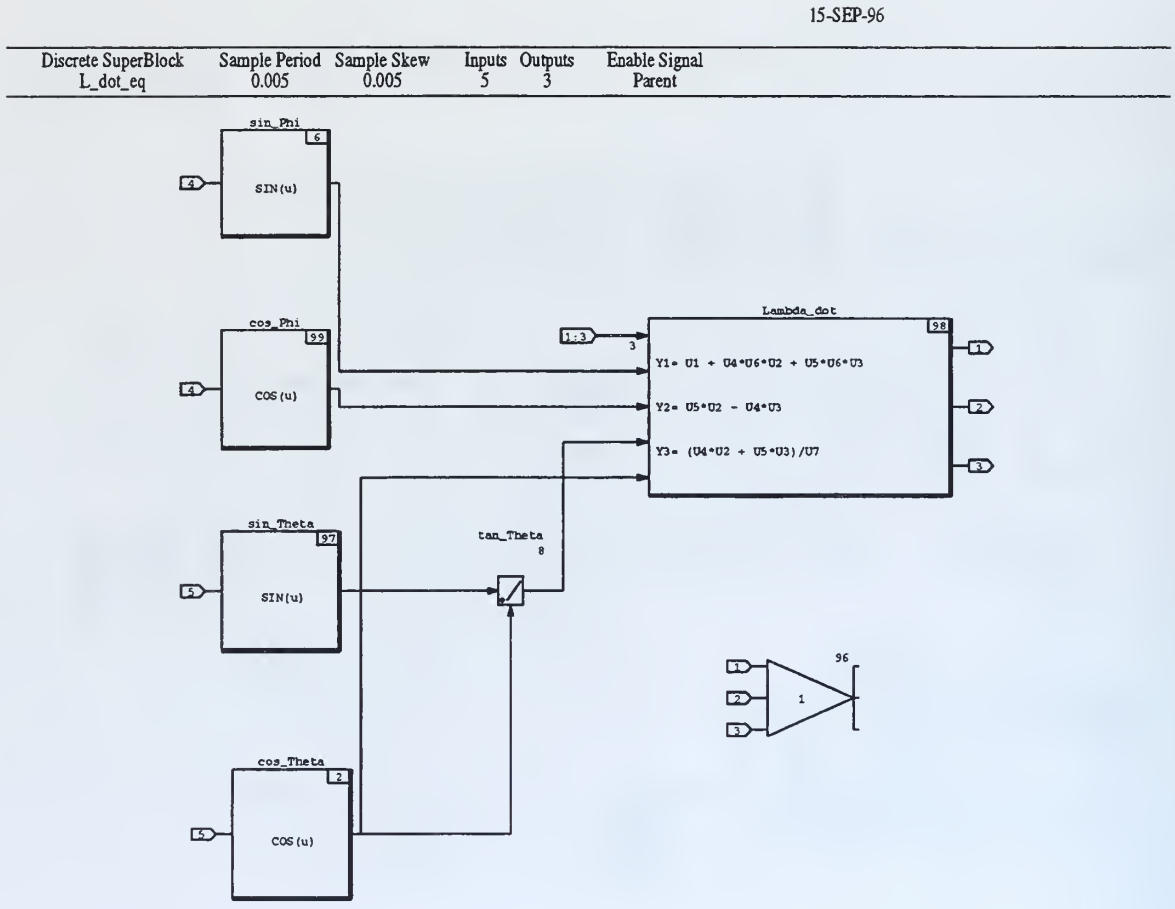

Figure 43. L_dot_eq

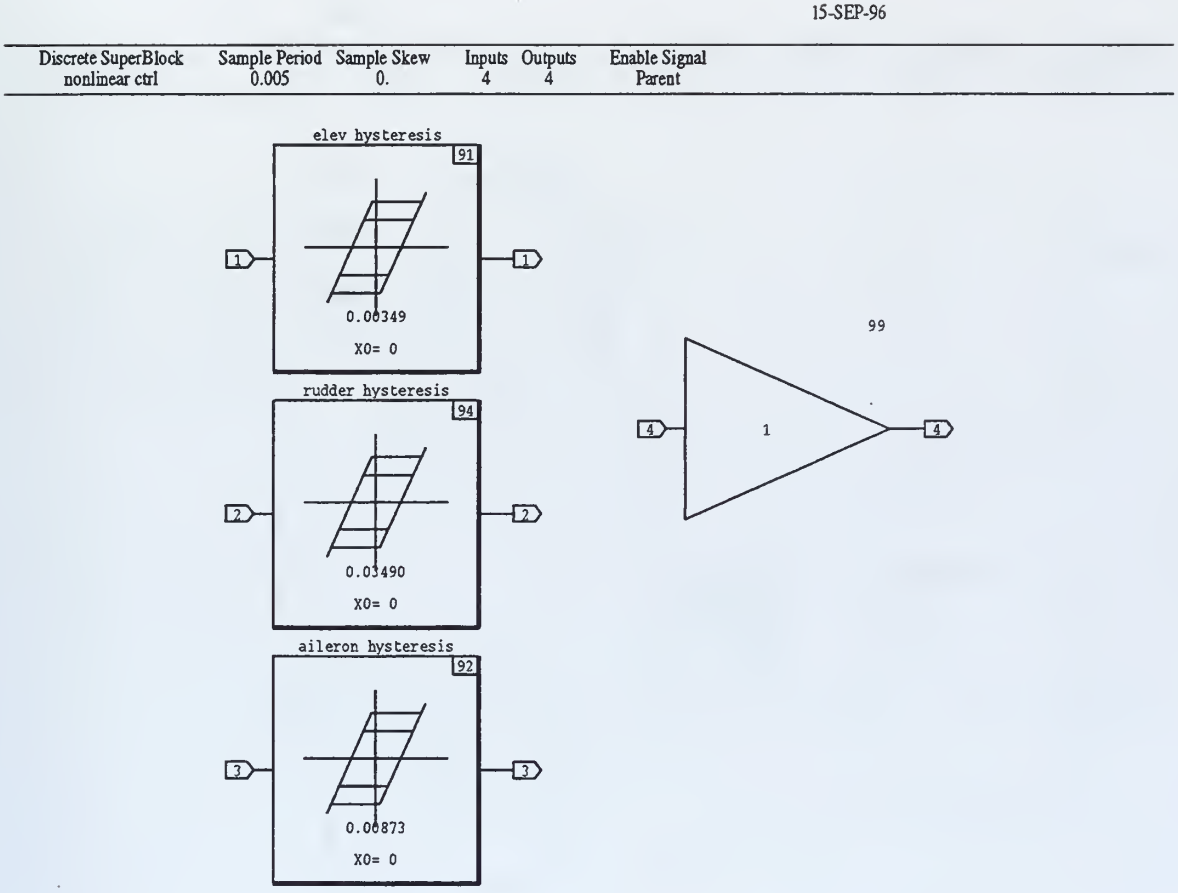

Figure 44. nonlinear_ctrl

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
command hold	0.04	0.	7	4	Parent

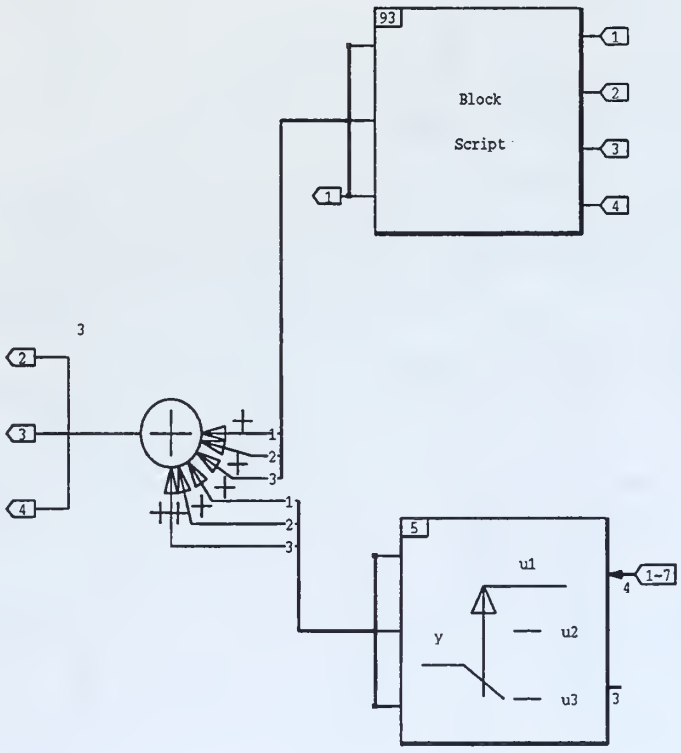

Figure 45. command_hold

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
delta controller	0.04	0.005	21	15	Parent

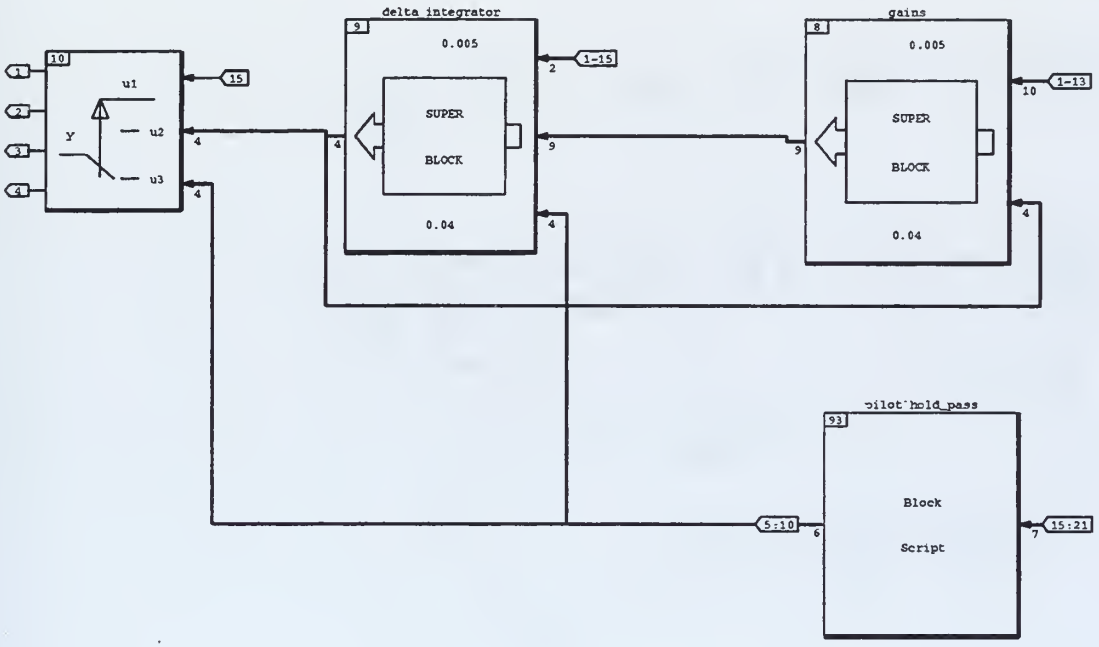

Figure 46. delta_controller

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
delta integrator	0.04	0.005	15	4	Parent

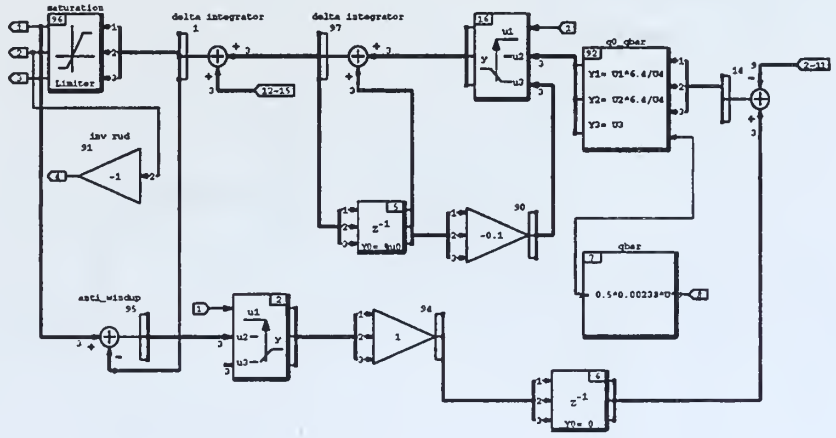

Figure 47. delta_integrator

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
act model	0.04	0.005	4	4	Parent

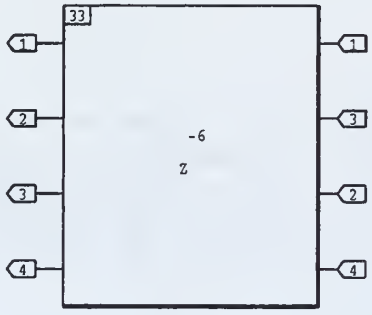

Figure 48. act_model

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
Sensor Filtering	0.04	0.	40	30	Parent

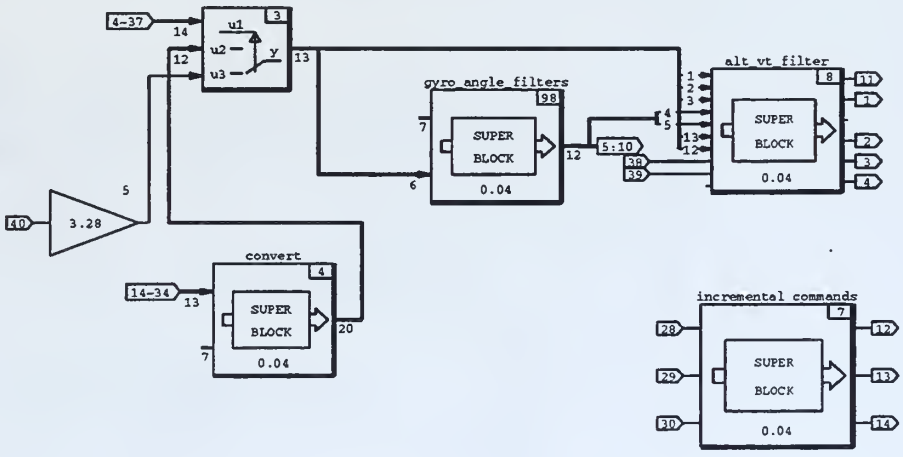

Figure 49. Sensor_Filtering

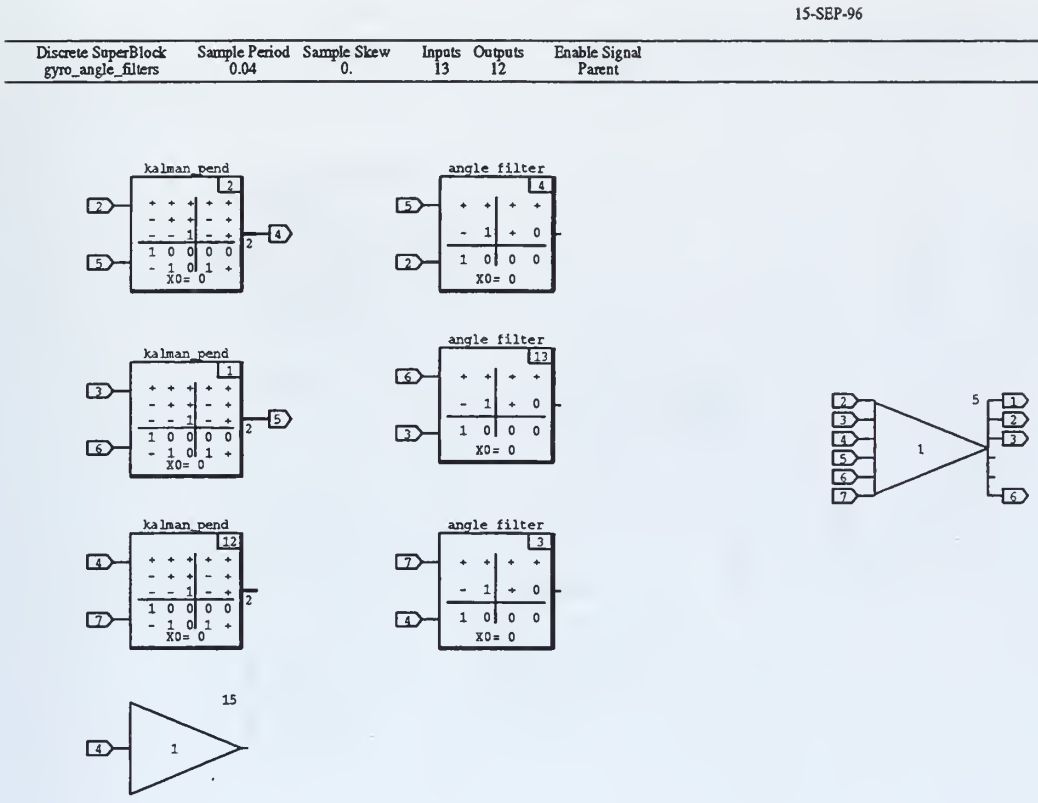

Figure 50. gyro_angle_filters

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
alt_vt_filter	0.04	0.	10	6	Parent

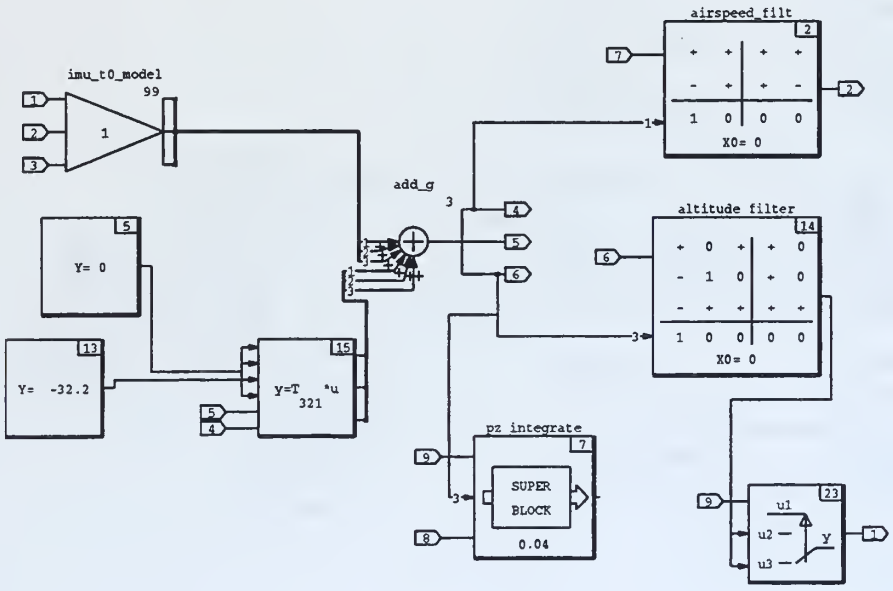

Figure 51. alt_vt_filter

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
convert	0.04	0.	20	20	Parent

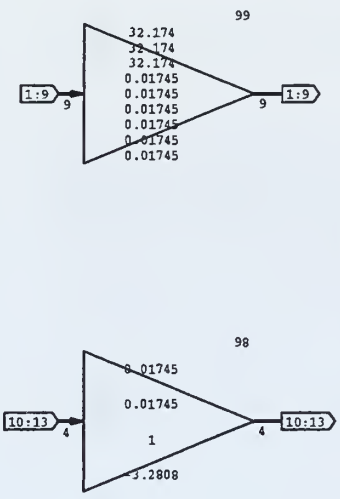

Figure 52. convert

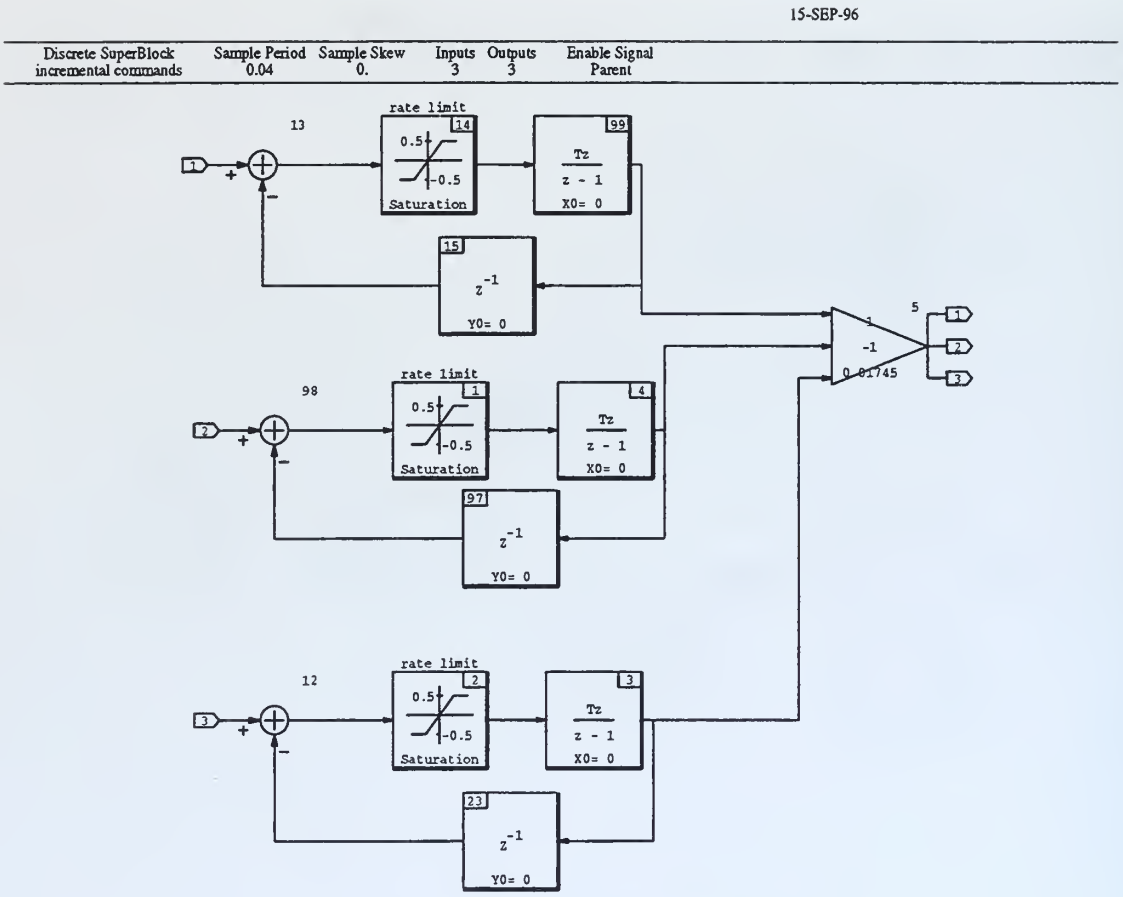

Figure 53. incremental_commands

15-SEP-96

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal
calibrate_rf_uplink	0.04	0.	56	23	Parent

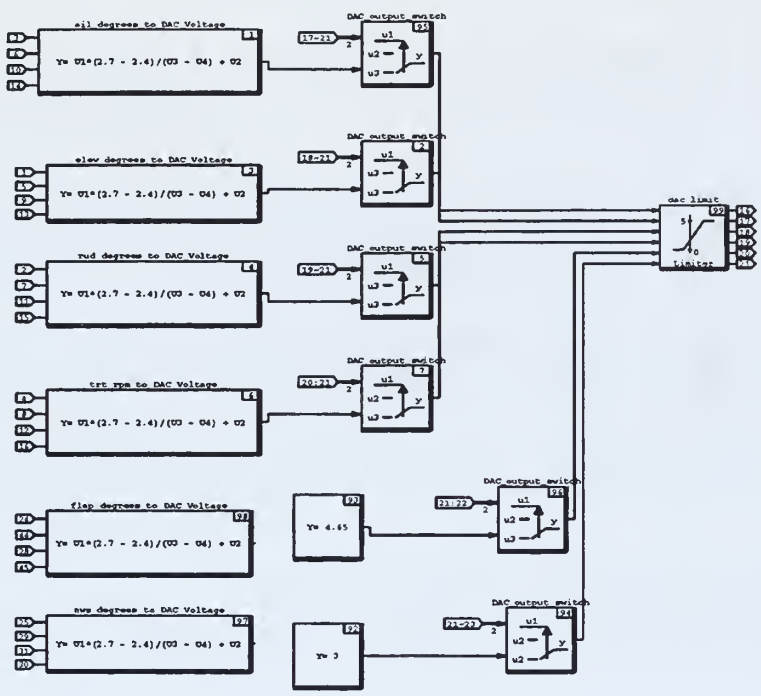

Figure 54. calibrate_rf_uplink

LIST OF REFERENCES

[1] Integrated Systems, Inc. *$MATRIX_X$ Product Family System Manuals*, Version 5.0, 1996.

[2] J. J. Craig, *Robotics.* Addison-Wesley, 1989.

[3] M. Elgersma, *Control of Nonlinear Systems. Using Partial Dynamic Inversion*, Ph.D. Thesis, University of Minnesota, Minneapolis, MN, 1988.

[4] Silvestre, C., Pascoal, A., D. Fryxell, and Kaminer, I., "Design and Implementation of a Trajectory Tracking Controller for an Autonomous Underwater Vehicle (AUV)," *Proc.1995 American Control Conference*, Seattle, June, 1995.

[5] Kaminer, I., Pascoal, A., Khargonekar, P., and Coleman, E., "A Velocity Algorithm for the Implementation of Gain-Scheduled Controllers", *Automatica.* **Vol. 31**, pp. 1185–1191, 1995.

[6] Doyle, J., Glover, K., Khargonekar, P. and Francis, B., State space solutions to standard $\mathcal{H}_2$ and $\mathcal{H}_\infty$ control problems. *IEEE Transactions on Automatic Control.* **Vol. AC- 34(8)**, pp. 831–847. 1989.

[7] Kaminer, I., *Motion Control of Rigid Bodies Using $\mathcal{H}_\infty$Synthesis and Related Theory* Ph.D. Thesis, University of Michigan, Ann Arbor, MI, 1992.

[8] Moats, M. L., *Automation of Hardware-In-The-Loop Testing and Implementation of Controllers for Umanned Air Vehicles* Masters Thesis, Naval Postgraduate School, Monterey, CA, 1994.

[9] Hallberg, E., *Design of a GPS Aided Guidance, Navigation, and Control System for Trajectory Control of an Air Vehicle* Masters Thesis. Naval Postgraduate School, Monterey, CA, 1994.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   8725 John J. Kingman Road.,Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library — 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Dr. Isaac I. Kaminer — 5
   Department of Aeronautics and
   Astronautics, Code AA/KA
   Naval Postgraduate School
   Monterey, CA 93943-5101

4. Dr. Richard M. Howard — 1
   Department of Aeronautics and
   Astronautics, Code AA/Ho
   Naval Postgraduate School
   Monterey, CA 93943-5101

5. Chairman — 2
   Department of Aeronautics and
   Astronautics, Code AA
   Naval Postgraduate School
   Monterey, CA 93943-5101

6. James A. Zanino — 2
   School of Aviation Safety
   Code 10
   Naval Postgraduate School
   Monterey, CA 93943-5101