



**Calhoun: The NPS Institutional Archive**

---

Theses and Dissertations

Thesis Collection

---

1994-09

# Design and evaluation of a LQR controller for the Bluebird Unmanned Air Vehicle

Foley, Brian T.

Monterey, California. Naval Postgraduate School

---

<http://hdl.handle.net/10945/30934>



Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

**Dudley Knox Library / Naval Postgraduate School**  
**411 Dyer Road / 1 University Circle**  
**Monterey, California USA 93943**

<http://www.nps.edu/library>

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California



## THESIS

DESIGN AND EVALUATION OF  
A LQR CONTROLLER FOR THE  
BLUEBIRD UNMANNED AIR VEHICLE

by

Brian T. Foley  
September, 1994

Thesis Advisor:

Isaac I. Kaminer

Approved for public release; distribution is unlimited.

Thesis  
P5363

BUGLEY KNOX LIBRARY  
NAVAL POSTGRADUATE SCHOOL  
MONTEREY CA 93943-5101

## REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding the burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September, 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE DESIGN AND EVALUATION OF A LQR CONTROLLER FOR THE BLUE-BIRD UNMANNED AIR VEHICLE			5. FUNDING NUMBERS	
6. AUTHOR(S) Brian T. Foley			8. PERFORMING ORGANIZATION REPORT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
1. 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 United States Government.				
2a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
3. ABSTRACT (Maximum 200 words) The modern aerospace controls engineer is provided with a variety of powerful tools to aid in the design and testing of digital flight control systems. The current fiscal environment requires extensive validation of all aerospace based systems through simulation and hardware-in-the-loop testing prior to implementation. This work explores the design and evaluation of an Automatic Flight Control System (AFCS) for the Bluebird Unmanned Aerial Vehicle (UAV). Software tools such as MATLAB and MATRIXx are used to evaluate the dynamic stability of the aircraft model and linear Quadratic Gaussian algorithms are used to obtain the appropriate controller. Graphical design applications such as SIMULINK and SystemBuild are then used to build a visual block diagram model of the aircraft dynamics and link it with the designed controller. Using this model, the control system response to commanded inputs and external disturbances was evaluated.				
4. SUBJECT TERMS MATLAB, Simulink, MATRIXx, SystemBuild, LQR, Bluebird, UAV			15. NUMBER OF PAGES 87	
			16. PRICE CODE	
7. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

N 7540-01-280-5500

Standard Form 298 (Rev. 2-89)  
Prescribed by ANSI Std. Z39-18  
298-102

Approved for public release; distribution is unlimited

**Design and Evaluation of  
a LQR Controller for the  
Bluebird Unmanned Air Vehicle**

by

Brian T. Foley  
Lieutenant, United States Navy  
B.A. College of the Holy Cross, 1986

Submitted in partial fulfillment of the  
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the


NAVAL POSTGRADUATE SCHOOL

September 1994


Author:

  
\_\_\_\_\_  
Brian T. Foley

Approved by:

  
\_\_\_\_\_  
Isaac I. Kammer, Thesis Advisor

  
\_\_\_\_\_  
Richard M. Howard, Second Reader

  
\_\_\_\_\_  
Daniel J. Collins, Chairman  
Department of Aeronautics and Astronautics

## ABSTRACT

The modern aerospace controls engineer is provided with a variety of powerful tools to aid in the design and testing of digital flight control systems. The current fiscal environment requires extensive validation of all aerospace based systems through simulation and hardware-in-the-loop testing prior to implementation. This work explores the design and evaluation of an Automatic Flight Control System (AFCS) for the Bluebird Unmanned Aerial Vehicle (UAV). Software tools such as MATLAB and MATRIX<sub>X</sub> are used to evaluate the dynamic stability of the aircraft model and Linear Quadratic Gaussian algorithms are used to obtain the appropriate controller. Graphical design applications such as SIMULINK and SystemBuild are then used to build a visual block diagram model of the aircraft dynamics and link it with the designed controller. Using this model, the control system response to commanded inputs and external disturbances was evaluated.

11/05  
F5363  
C.J.

## TABLE OF CONTENTS

<b>I. INTRODUCTION</b> . . . . .	1
A. BACKGROUND FOR UAV RESEARCH . . . . .	1
B. OVERVIEW . . . . .	2
<b>II. BACKGROUND: EOM DEVELOPMENT</b> . . . . .	6
A. FORCE EQUATION . . . . .	6
B. MOMENT EQUATION . . . . .	7
C. EXTERNAL FORCES AND MOMENTS . . . . .	8
D. STATE SPACE REPRESENTATION . . . . .	9
<b>III. OPEN-LOOP ANALYSIS</b> . . . . .	12
A. PLANT MODELING AND VALIDATION . . . . .	13
B. CONTROLLER DESIGN REQUIREMENTS . . . . .	17
<b>IV. LQR THEORY AND CONTROLLER DESIGN</b> . . . . .	19
A. OPTIMAL CONTROL . . . . .	19
B. CONTROLLER DESIGN AND ANALYSIS . . . . .	20
<b>V. CONTROLLER IMPLEMENTATION AND TESTING</b> . . . . .	25
A. CONTROLLER STRUCTURE . . . . .	26
B. LINEAR CONTROLLER, LINEAR PLANT MODEL . . . . .	27
C. DELTA CONTROLLER, LINEAR PLANT MODEL . . . . .	28
D. DELTA CONTROLLER, NONLINEAR PLANT MODEL . . . . .	29
E. DISCRETE CONTROLLER IN SIMULINK AND SystemBuild . . . . .	30
F. MODEL VERIFICATION . . . . .	34
<b>VI. CONCLUSIONS AND RECOMMENDATIONS</b> . . . . .	38

A. CONCLUSIONS . . . . .	38
B. RECOMMENDATIONS . . . . .	39
<b>APPENDIX A. CODED EQUATIONS OF MOTION . . . . .</b>	<b>40</b>
A. STATE.DERIV.M. . . . .	40
B. EOM2.WIND.C . . . . .	42
<b>APPENDIX B. OPEN-LOOP PLANT . . . . .</b>	<b>52</b>
A. OPEN-LOOP PLANT . . . . .	52
B. OPEN-LOOP STEP RESPONSE PLOTS . . . . .	53
<b>APPENDIX C. CONTROLLER CALCULATION/VERIFICATION . . . . .</b>	<b>57</b>
A. BLUE.PLOT.M . . . . .	57
B. OPTIMAL GAINS . . . . .	58
C. CLOSED-LOOP FREQUENCY AND STEP RESPONSES . . . . .	59
<b>APPENDIX D. CLOSED-LOOP PERFORMANCE OF DISCRETE                   CONTROLLER ON THE NONLINEAR PLANT . . . . .</b>	<b>69</b>
LIST OF REFERENCES . . . . .	74
INITIAL DISTRIBUTION LIST . . . . .	76



## LIST OF TABLES

1.1	PHYSICAL CHARACTERISTICS OF Bluebird . . . . .	2
3.1	CESSNA 172 EIGENVALUES . . . . .	16
3.2	Bluebird EIGENVALUES . . . . .	16
4.1	Bluebird CLOSED-LOOP EIGENVALUES . . . . .	24

## LIST OF FIGURES

3.1	Bluebird Open-loop Plant Model – SIMULINK . . . . .	13
3.2	SIMULINK Implementation of the EOM . . . . .	14
3.3	SystemBuild Open-Loop Plant Model . . . . .	15
4.1	SIMULINK Longitudinal Control Synthesis Model . . . . .	21
4.2	SIMULINK Lateral Control Synthesis Model . . . . .	22
5.1	SIMULINK Linear Control Model . . . . .	27
5.2	SIMULINK Closed-Loop Linear Control Model . . . . .	28
5.3	Delta Controller Model . . . . .	29
5.4	Delta Control, Nonlinear Plant . . . . .	31
5.5	Discrete Differentiator & Integrator . . . . .	32
5.6	SIMULINK Discrete Control Model . . . . .	33
5.7	SystemBuild Discrete Longitudinal Control Model . . . . .	34
5.8	SystemBuild Discrete Lateral Control Model . . . . .	35
5.9	SystemBuild Discrete Control Model . . . . .	36
B.1	Open-loop Inertial Velocity, $u$ $v$ $w$ . . . . .	54
B.2	Open-loop Inertial Rates, $p$ $q$ $r$ . . . . .	55
B.3	Open-loop Euler Angles, $\phi$ $\theta$ $\psi$ . . . . .	56
C.1	Elevator Control Loop . . . . .	60
C.2	Throttle Control Loop . . . . .	61
C.3	Step Altitude Command . . . . .	62
C.4	Altitude Command Bode Diagram . . . . .	63

C.5	Step Airspeed Command . . . . .	64
C.6	Airspeed Command Bode Diagram . . . . .	65
C.7	Aileron Control Loop . . . . .	66
C.8	Step Heading Command . . . . .	67
C.9	Heading Command Bode Diagram . . . . .	68
D.1	Inertial Velocity Time History . . . . .	70
D.2	Inertial Rate Time History . . . . .	71
D.3	Euler Angle Time History . . . . .	72
D.4	Altitude and TAS Time History . . . . .	73

## ACKNOWLEDGMENT

I would like to take this opportunity to thank Dr. Isaac Kaminer for his support and guidance during my work on this project. His expertise was invaluable in the completion of a body of work that I hope will complement the outstanding research being done at the NPS Avionics Lab. Finally I would like to thank Jeanmarie and Matthew for their understanding and support. You were my sounding board and sanity check. You participated in my research more than you will ever realize. Thank You.



# I. INTRODUCTION

## A. BACKGROUND FOR UAV RESEARCH

The modern battlefield has increasingly progressed towards the use of automated systems and remotely controlled devices to perform a variety of missions. From surveillance to weapons delivery and bomb damage assessment, the human operator is being removed from the direct danger of a hostile environment and placed in a position of evaluating data received via RF or fiber optic link. The direct and obvious benefits of such an arrangement are the reduced risk to the operator and the reduced cost of the unmanned sensor platform as compared to traditional manned platforms. The state-of-the-art technology in unmanned aerial vehicle development has demonstrated the capability of flight out to ranges of 500 nm and endurances exceeding 24 hours. Combined with the ability to carry a variety of sensor suites, these platforms represent the future in airborne data acquisition for both military and civilian applications.

In support of these technological developments the Unmanned Air Vehicle Flight Research Lab (UAV FRL) at the Naval Postgraduate School has been investigating several unmanned aerial vehicles as technology demonstrators. The AROD UAV is a vertical take-off and landing platform. Vertical flight is accomplished with a powerful ducted fan producing enough thrust to lift the aircraft. Current proposals have the AROD working in conjunction with unmanned surface and subsurface vehicles providing additional remote sensing capabilities and data link services between the operator and the surface vehicles. The AROD is an inherently unstable platform and is subject to gyroscopic coupling and torque effects during the production of

TABLE 1.1: PHYSICAL CHARACTERISTICS OF Bluebird

Weight	55 lbs
Average Wing Chord, $\bar{c}$	1.802 $f$
Wing Span, $b$	12.42 $f$
Planform Surface Area, $S$	22.380 $f^2$
Engine Power	4.0 HP
Mass Moment of Inertia, $I_x$	10.0 $slug - f^2$
Mass Moment of Inertia, $I_y$	16.12 $slug - f^2$
Mass Moment of Inertia, $I_z$	7.97 $slug - f^2$

lifting thrust. Extensive modeling and simulation of this vehicle was previously accomplished by Sivashankar and Moats [Ref. 1, 2] in separate work at the UAV FRL. This work validated the dynamically unstable nature of the AROD and provided the motivation for the second UAV project currently under development at the UAV FRL.

The Bluebird aircraft was acquired as a test bed for guidance and navigation systems. It is similar in appearance to a scaled down Cessna 172. Its physical characteristics are given in Table 1.1. Its conventional high-wing configuration makes for a stable aircraft. This provides the ideal platform for testing guidance, navigation, and control software and hardware before installation on the AROD. As with the AROD, the Bluebird has been extensively modeled [Ref. 3, 4], the results of which will be covered in Chapter III.

## B. OVERVIEW

This thesis fulfills a twofold purpose. First, to provide for the autonomous control of the Bluebird UAV, a controller is designed based on Linear Quadratic Regulator Theory and using the 'lqr' and 'regulator' functions of MATLAB and MATRIX\_X. This design will allow the remote operator to control the vehicle's altitude, true air-

speed, and heading, while limiting the response to commanded inputs to within the vehicles dynamic operating envelope. This stable control provides the capability to test a variety of avionics systems through a range of dynamic maneuvers that would not be possible in tethered flight. It also provides for a more stable control than in the case of direct RF control by a human operator. Second, this work will provide a link from the courses in classical and modern control theory at the Naval Postgraduate School to the implementation of these concepts using state-of-the-art software tools such as MATLAB and MATRIX<sub>Y</sub>. The ultimate goal is the integration of the modeling of the airborne platform and sensors, controller design, and hardware-in-the-loop testing of the design on the chosen platform.

These objectives were achieved in a multi-step process described in this thesis. This description begins with a summary of the development of the nonlinear equations of motion of a rigid body in space that is subjected to external forces and moments. The formulation of these equations has been the subject of much study at the UAV FRL [Ref. 3, 4]. For this reason only the significant results will be examined as an aid in understanding the development of the aircraft models (Chapter II).

Following this step, the equations of motion are encoded to form the core of a high fidelity nonlinear block diagram model of the aircraft dynamics in SIMULINK and SystemBuild. These computer codes have been previously developed and validated independently [Ref. 4, 5] in the .m file format of MATLAB-SIMULINK and as C-code. To model disturbances induced during flight through a moving air mass and to calculate the aircraft true airspeed, this work modified these computer codes to include wind inputs in the inertial  $x$ ,  $y$ , and  $z$  coordinate directions.



To better determine control system requirements an open-loop analysis of the aircraft model was done as follows:

- The nonlinear model was trimmed about a nominal operating point around which the dynamic response to small perturbations could be analyzed.
- The model was then linearized around the trim point to obtain a linear model.
- The eigenvalues of the linear plant were determined and the natural frequency and damping of the different modes were analyzed.

. From this data controller requirements were established, determining desired bandwidths for response to command and control inputs versus the actual open-loop plant responses (Chapter III).

These requirements provide the basis evaluating the feedback controller obtained using the linear quadratic regulator algorithms of MATLAB and MATRIX<sub>X</sub>. The controller design was based upon linear quadratic regulator theory. To allow for a better understanding of the algorithms used to calculate the controller, the main points of this theory are reviewed. The controller design proceeds using the following steps:

- The control synthesis model is developed. In this model the states to be controlled and actuators to accomplish this control are included.
- The control gains are calculated using the appropriate MATLAB and MATRIX<sub>X</sub> algorithms. In these calculations a *cost function*, which includes weighting factors, is used to modify the energy penalty incurred in responding to the various control and command inputs.

- The time and frequency domain response of the closed-loop controller and plant are evaluated. If the step response or bandwidth of the system does not meet the design requirements, the weighting factor is changed for the particular command or control input and the process repeated.

The entire design process may take several iterations and may also involve the manipulations of zeros added to the synthesis model. In theory, as the weights on the controls go to zero, the closed-loop poles will go to the open-loop zeros that were added to the model. This ensures that the closed-loop poles have the desired damping ratio and natural frequency to obtain the desired system response (Chapter IV).

Once the controller meets design requirements, it is implemented on the nonlinear plant. This implementation is accomplished in four steps:

- Linear controller with the linear plant, initial conditions equal to zero.
- Differential controller with the linear plant, initial conditions equal to zero.
- Differential controller with the nonlinear plant, initial conditions equal to the trim states.
- Differential controller and nonlinear plant are discretized to run on a digital computer.

At the completion of each stage of this integration process the closed-loop controller/plant model is tested to ensure the appropriate response to a variety of commanded inputs and external disturbances (Chapter V).

## II. BACKGROUND: EOM DEVELOPMENT

The development of the equations of motion for a generic aircraft with six degree freedom of motion has been the subject of much study at the UAV FRL . High fidelity nonlinear models of both the AROD and Bluebird have been developed and verified through extensive simulation [Ref. 3, 4]. To rederive these results would be an unnecessary repetition of effort. However, to provide a sound basis for understanding the models developed in this thesis, a summary of the significant results of these previous works will be provided.

### A. FORCE EQUATION

The derivation of equations of motion for a general six degree of freedom airplane model can be divided into two parts. The first part is simply the determination of the equations of motion for any rigid body in space. It is dependent only on the linear and angular momenta of the body. The equations of linear motion for the aircraft center of gravity are derived by a direct application of Newton's Law,  $F = m a$ . The final result for the forces acting on the aircraft body in a body referenced coordinate system is,

$$\begin{aligned} {}^B F &= m \left( \frac{d}{dt} {}^B v_{BO} + {}^B \omega_B \times {}^B v_{BO} \right) \\ &= m \frac{d}{dt} {}^B v_{BO} + m {}^B \omega_B \times {}^B v_{BO}. \end{aligned} \quad (2.1)$$

where,

- ${}^B F$  represents the forces on the aircraft in the body coordinate system.

- $\frac{d}{dt} {}^B v_{BO}$  represents the derivative with respect to time in the body coordinate system of the velocities of the aircraft body origin.
- $\frac{d}{dt} {}^B v_{BO}$  are the velocities of the body origin resolved in the body coordinate system.
- ${}^B \omega_B$  are the body angular rates resolved in the body coordinate system.

## B. MOMENT EQUATION

The equations for angular acceleration are derived from Euler's Law for the conservation of angular momentum,

$$\dot{L} = N, \quad (2.2)$$

The final result, shown in Equation 2.3, expresses the total external moment applied to the aircraft center of gravity and is given as:

$${}^B N_{BO} = I_B {}^B \dot{\omega}_B + {}^B \omega_B \times (I_B {}^B \omega_B + I_R {}^B \omega_R). \quad (2.3)$$

where,

- ${}^B N_{BO}$  represents the moments acting on the aircraft body origin resolved in the body coordinate system.
- $I_B$  is the inertia tensor for the aircraft.
- $I_R$  is the inertia tensor for any significant rotating object located at the center of gravity of the aircraft, such as a propeller or a turbine. It should be noted that the term  $I_R {}^B \omega_R$  can be disregarded if it is insignificant compared to  $I_B$  and  ${}^B \omega_B$  [Ref. 6].

## C. EXTERNAL FORCES AND MOMENTS

The second part is the calculation of aerodynamic, gravitational, and thrust forces and moments on the airplane. The total forces and moments acting on the aircraft are determined by summing these components and can be expressed as:

$$\begin{bmatrix} {}^B F \\ {}^B N \end{bmatrix} = \begin{bmatrix} {}^B F_{GRAV} + {}^B F_{PROP} + {}^B F_{AERO} \\ {}^B N_{PROP} + {}^B N_{AERO} \end{bmatrix} \quad (2.4)$$

The aerodynamic forces are specific to the aircraft platform and are calculated using nondimensional stability and control derivatives. The stability and control derivatives are obtained by approximating the aerodynamic forces and moments acting on the aircraft using a first order Taylor Series expansion about a given flight trim point [Ref. 7]. Since these derivatives are generally computed in the wind referenced coordinate system it is necessary to resolve the resulting expression for the aerodynamic forces and moments in a body referenced coordinate system. The final result for the aerodynamic forces is given by [Ref. 8]:

$$\begin{bmatrix} {}^B F_{AERO} \\ {}^B N_{AERO} \end{bmatrix} = \bar{q} \bar{S} \begin{bmatrix} {}^B_W R & 0 \\ 0 & {}^B_W R \end{bmatrix} \left\{ C_{F_0} + \frac{\partial C}{\partial x'} M' x + \frac{\partial C}{\partial \dot{x}'} M' \dot{x} + \frac{\partial C}{\partial \Delta} \Delta \right\} \quad (2.5)$$

where,

- $\bar{q}$  is the dynamic pressure ( $0.5\rho V^2$ ).
- $\bar{S} = \text{diag}\{S, S, S, Sb, Sc, Sb\}$ .
- $\Delta = [\delta_{elevator}, \delta_{rudder}, \delta_{aileron}]$ , is the vector of control inputs.
- ${}^B_W R$  is the rotation matrix from wind to body coordinates.
- $M'$  is the scaling matrix given by  $\text{diag}\{1/V_T, 1/V_T, 1/V_T, b/2V_T, c/2V_T, b/2V_T\}$ .
- $C_{F_0}$  is the vector of steady state coefficients representing trimmed flight.

- $\frac{\partial C}{\partial z'}$  is the matrix of stability and control derivatives and is given by,

$$\begin{bmatrix} C_{L_U} & C_{L_\beta} & C_{L_\alpha} & C_{L_p} & C_{L_q} & C_{L_r} \\ C_{Y_U} & C_{Y_\beta} & C_{Y_\alpha} & C_{Y_p} & C_{Y_q} & C_{Y_r} \\ C_{D_U} & C_{D_\beta} & C_{D_\alpha} & C_{D_p} & C_{D_q} & C_{D_r} \\ C_{l_U} & C_{l_\beta} & C_{l_\alpha} & C_{l_p} & C_{l_q} & C_{l_r} \\ C_{m_U} & C_{m_\beta} & C_{m_\alpha} & C_{m_p} & C_{m_q} & C_{m_r} \\ C_{n_U} & C_{n_\beta} & C_{n_\alpha} & C_{n_p} & C_{n_q} & C_{n_r} \end{bmatrix} \quad (2.6)$$

$\frac{\partial C}{\partial z'}$  is very similar to  $\frac{\partial C}{\partial z}$ , except that only the  $\dot{\alpha}$  and  $\dot{\beta}$  terms are the only nonzero terms.

- $x$  is the state vector composed of body inertial velocities and rates.

The calculation of the forces and moments due to gravitational and propulsive effects are carried out in a more straight forward manner. The final results are given by.

$${}^B F_{GRAV} = {}^B_U R \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \quad (2.7)$$

and,

$${}^B F_{PROP} = \begin{bmatrix} T_X \\ T_Y \\ T_Z \end{bmatrix} \quad (2.8)$$

and

$${}^B N_{PROP} = \begin{bmatrix} T_l \\ T_m \\ T_n \end{bmatrix}, \quad (2.9)$$

where the  $T_i$  terms represent the forces and moments due to the generation of thrust.

## D. STATE SPACE REPRESENTATION

From the expressions derived for the forces and moments acting on the aircraft it is possible to derive a state space representation of the equations of motion. We

can write equations 2.1 and 2.3 in the following manner,

$$\begin{bmatrix} {}^B F \\ {}^B N \end{bmatrix} = \begin{bmatrix} m \frac{d}{}^B v_{BO} + m ({}^B \omega_B \times {}^B v_{BO}) \\ I_B {}^B \dot{\omega}_B + {}^B \omega_B \times (I_B {}^B \omega_B + I_R {}^B \omega_R) \end{bmatrix}. \quad (2.10)$$

By rearranging this expression and normalizing by  $1/m$  and  $I_B^{-1}$  we obtain the resulting equation in state space form,

$$\frac{d}{dt} \begin{bmatrix} {}^B v_{BO} \\ {}^B \omega_B \end{bmatrix} = \begin{bmatrix} -{}^B \omega_B \times {}^B v_{BO} & + \frac{{}^B F}{I_B^{-1} {}^B N} \\ -I_B^{-1} {}^B \omega_B \times (I_B {}^B \omega_B + I_R {}^B \omega_R) & \end{bmatrix}. \quad (2.11)$$

Substituting the expressions for the aerodynamic, gravitational, and thrust forces and moments into equation 2.4 and then substituting this into equation 2.11 the complete state space representation of the nonlinear equations of motion becomes,

$$\begin{aligned} \frac{d}{dt} \begin{bmatrix} {}^B v_{BO} \\ {}^B \omega_B \end{bmatrix} = \chi^{-1} \left\{ \begin{bmatrix} -{}^B \omega_B \times & 0 \\ 0 & -{}^B I_B^{-1} ({}^B \omega_B \times ({}^B I_B {}^B \omega_B + I_R {}^B \omega_R)) \end{bmatrix} + \right. \\ \left. M_I^{-1} {}^B T \bar{q} \bar{S} \frac{\partial C_F}{\partial x^i} M^i \begin{bmatrix} {}^B v_{BO} \\ {}^B \omega_B \end{bmatrix} + M_I^{-1} \left\{ \begin{bmatrix} {}^B F_{GRAV} \\ 0 \end{bmatrix} + \right. \right. \\ \left. \left. \begin{bmatrix} {}^B F_{PROP} \\ {}^B N_{PROP} \end{bmatrix} \delta_T + {}^B T \bar{q} \bar{S} (C_{F_0} + \frac{\partial C_F}{\partial \Delta} \Delta) \right\} \right\}, \quad (2.12) \end{aligned}$$

where.

- $\chi = I_6 - M_I^{-1} {}^B T \bar{q} \bar{S} \frac{\partial C_F}{\partial x^i} M^i$
- $M_I = \begin{bmatrix} m & 0 \\ 0 & {}^B I_B \end{bmatrix}$
- ${}^B T = \begin{bmatrix} {}^B R & 0 \\ 0 & {}^B R \end{bmatrix}$

To complete the equations of motion, the following two differential equations are used to calculate the body positions and Euler angles,

$${}^U \dot{P}_{BO} = {}^U B {}^B v_{BO}, \quad (2.13)$$

and

$$\dot{\Lambda} = S(\Lambda) {}^B \omega_B. \quad (2.14)$$

where,

- ${}^U_B R$  is the rotation matrix from the body to the inertial coordinate system.
- $S(\Lambda)$  is the rotation matrix that takes the body angular rates to the Euler angular rates.



### III. OPEN-LOOP ANALYSIS

In order to design a controller which stabilizes the feedback system for the Bluebird UAV it is necessary to analyze the open-loop characteristics of the aircraft model. Before this analysis takes place, the differential equations of motion must be modeled using such tools as SIMULINK or SystemBuild or encoded in a form that graphical software applications such as these can use. Work by Halberg [Ref. 4] and Byerly [Ref. 5] have developed such codes in the *.m* file format of MATLAB and as a *C - code* file in the User Code Block format of MATRIX<sub>X</sub>. To account for the motion of the aerodynamic body through a moving airmass, the *C - code* was modified to allow for the input of wind disturbances in the three inertial coordinate directions. The *.m* file allows for airmass disturbances and no modifications were necessary. The codes are shown in Appendix A. Using these codes as the core, both SIMULINK and SystemBuild block diagram models were developed to represent the dynamic aircraft model. Using data for the Cessna 172 aircraft from Roskam [Ref. 6] both the models were validated by comparing eigenvalues for the open-loop plant. Using these models the open-loop analysis proceeded. This analysis provided data on the damping and frequencies of the different aircraft modes. Additionally, the time history of the aircraft states was obtained and plotted to provide a visual clue to the open-loop aircraft performance. Using this data, control requirements were derived and controller design accomplished.

## A. PLANT MODELING AND VALIDATION

The computer codes modeling the aircraft equations of motion have been previously developed. The validation of these codes using SIMULINK or SystemBuild requires the generation of appropriate block diagrams. Figure 3.1 and shows the structure of the block

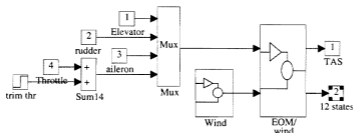


Figure 3.1: BlueBird Open-loop Plant Model – SIMULINK [Ref. 4]

named EOM/wind, Figure 3.2, shows the actual implementation of the nonlinear differential equations of motion. The MATLAB Function block named 'state\_deriv' is the calling block for the .m file that calculates the derivative states of the body velocities and angular rates.

The  $MATRIX_X$  implementation of the equations of motion was less complex since the calculation of the position and Euler angle derivative states was internal to the *C-code* routine that is called in the SuperBlock USR003. Figure 3.3 details the SystemBuild open-loop model. Note that the only external routine block required is for the calculation of the vehicle true airspeed.

To determine the vehicle open-loop characteristics and performance, so that a

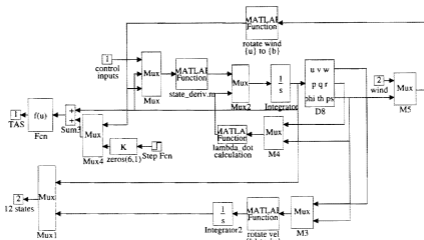
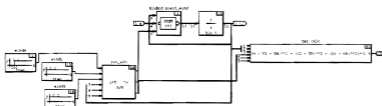


Figure 3.2: SIMULINK Implementation of the EOM, [Ref. 4]

linear controller can be designed, both models were trimmed using the appropriate MATLAB and MATRIX<sub>X</sub> functions and then linearized about the trim state. The open-loop plant is shown in Appendix B. According to small perturbation theory, it can then be assumed that the resulting aircraft plant will respond linearly to small magnitude disturbances about the trim state.

To validate the model dynamics, stability and control derivatives from Roskam [Ref. 6] were substituted into the coded equations of motion. The models were then trimmed and linearized. The eigenvalues of the resulting linear state space model were

Continuous Superlock	Ext. Inputs	Ext. Outputs
blatrd_wind_openloop	4	33



**Figure 3.3: SystemBuild Open-Loop Plant Model**

then determined and compared with the values from Roskam. The results are shown in Table 3.1. The data in Table 3.1 indicates that the models are highly accurate. The only significant difference occurs in the real part of the Phugoid mode poles. However, when the magnitude of these eigenvalues are compared, the difference is less than 0.2%.

With the models validated, the open-loop analysis of the Bluebird plant model followed the identical procedure. After substituting the aerodynamic stability and control derivatives in the encoded equations of motion, the model was then trimmed

Mode	Roskam	MATLAB	MATRIX <sub>x</sub>
Longitudinal			
Short Period	$-4.130 + 4.390j$	$-4.1303 + 4.3895j$	$-4.1303 + 4.3895j$
	$-4.130 - 4.390j$	$-4.1303 - 4.3895j$	$-4.1303 - 4.3895j$
Phugoid	$-.02092 + .1797j$	$-.0135 + .1801j$	$-.0135 + .1801j$
	$-.02092 - .1797j$	$-.0135 - .1801j$	$-.0135 - .1801j$
Lateral-Directional			
Dutch Roll	$-.6858 + 3.306j$	$-.6930 + 3.3077j$	$-.6930 + 3.3077j$
	$-.6858 - 3.306j$	$-.6930 - 3.3077j$	$-.6930 - 3.3077j$
Roll	$-12.43$	$-12.4343$	$-12.4343$
Spiral	$-.01095$	$-.0109$	$-.0109$

TABLE 3.1: CESSNA 172 EIGENVALUES

and linearized. The results were identical for the SIMULINK and SystemBuild models. These results are summarized in Table 3.2 along with the associated damping ratio and frequency of each mode.

Mode	Eigenvalue	Damping, $\zeta$	Frequency(rad/sec), $\omega$
Longitudinal			
Short Period	$-4.3290 + 3.9939j$	.735	5.8899
	$-4.3290 - 3.9939j$	.735	5.8899
Phugoid	$-.0171 + .4970j$	.0344	.4973
	$-.0171 - .4970j$	.0344	.4973
Lateral-Directional			
Dutch Roll	$-.2665 + 2.3861j$	.1110	2.4009
	$-.2665 - 2.3861j$	.1110	2.4009
Roll	$-4.5722$	1.0	4.5722
Spiral	.0384	-1.0	.0384

TABLE 3.2: Bluebird EIGENVALUES

To provide further data for the open-loop analysis, the model simulations were run over a period of 300 secs and time history plots of the output states were obtained. (Appendix B)

## B. CONTROLLER DESIGN REQUIREMENTS

The information provided by the open-loop analysis lead to several conclusions about the uncontrolled aircraft dynamics:

- The short period is critically damped with stable left plane poles.
- The phugoid mode is extremely underdamped. To improve the altitude and airspeed tracking performance of the vehicle, the damping of this mode must be increased.
- The dutch roll mode damping is inadequate for stable heading control.
- The unstable spiral mode has a destabilizing effect on performance. Even though this mode is extremely slow, to adequately control aircraft heading this eigenvalue must be moved to the left half plane.

Additional considerations in establishing controller requirements for the Bluebird UAV were:

- The Bluebird test vehicle is designed for use as an avionics test bed. The platform flight regime should thus include only relatively benign maneuvers. This requirement leads to the implementation of limiters on both body coordinate  $x$ - and  $z$ - accelerations and bank angle,  $\phi$ .
- The Futaba SB-34 servomotors that serve as control actuators have been shown to respond accurately up to  $4 \text{ rad/sec}$  [Ref. 2].

The preceding factors lead to the formulation of the following control system requirements:

- Feedback system must be stable.

- Minimum phugoid mode damping of 0.50.
- Yaw rate commanded continuously to zero.
- Control bandwidths less than or equal to  $4rad/sec$  to conform with the servomotor limitations.
- Maximum overshoot of 10% to step commands in airspeed, altitude, and heading.
- Rise time to step commands in airspeed, altitude, and heading of approximately  $4sec$ . Rise time is defined as the time to go from 10% to 90% of the commanded input.
- Maximum angle of bank of 30 degrees.
- Accelerations in the  $x$ - and  $z$ - body coordinates limited to less than  $1.0g$ .

## IV. LQR THEORY AND CONTROLLER DESIGN

### A. OPTIMAL CONTROL

In designing control systems we are often concerned with choosing the control  $u(t)$  such that a given performance index is minimized. Such an index is given by:

$$J = \int_0^{\infty} (x'Qx + u'Ru) dx \quad (4.1)$$

Now suppose a system is described by the following equations:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= x \end{aligned} \quad (4.2)$$

assuming that  $(A, B)$  is stabilizable,  $(Q, A)$  is detectable,  $Q \geq 0$ , and  $R > 0$ .

From theory [Ref. 9], given the performance index shown in Equation 4.1 and the system given by Equations 4.2, there exists a minimizing control,

$$u = Kx \quad (4.3)$$

which stabilizes  $(A - BK)$ , where:

$$K = R^{-1}B'P \quad (4.4)$$

In Equation 4.4,  $P$  is a unique, positive definite solution to the Algebraic Riccati Equation shown in Equation 4.5.

$$A'P + PA - PBR^{-1}B'P + Q = 0 \quad (4.5)$$



By solving this equation for the matrix  $P$ , for the chosen weighting matrices  $Q$  and  $R$ , the optimal gain  $K$  can be determined using equation 4.4.

The entire control design process reduces to an iterative procedure that follows these basic steps:

- Choose weighting matrices  $Q$  and  $R$ .
- Calculate the solution to the Ricatti Equation,  $P$ , equation 4.5.
- Calculate the optimal gain,  $K$ , equation 4.4.
- Examine the closed-loop system for the desired frequency and step responses.
- If the bandwidth or time domain characteristics are not as desired, choose new  $Q$  or  $R$  and begin the design procedure again.

In this manner the designer is able to iterate through the family of stabilizing controllers until obtaining the desired frequency and time domain characteristics.

## B. CONTROLLER DESIGN AND ANALYSIS

The control system was designed assuming zero coupling between the lateral and longitudinal states. Examination of the open-loop plant in Appendix B shows the absence of cross-coupling terms in the state matrix,  $A$ , and minimal cross coupling in the control matrix,  $B$ . This analysis supports the choice of developing separate controllers for the lateral and longitudinal states.

The previous section showed that the solution to the Ricatti Equation leads to the calculation of the optimal control based upon the choice of weighting matrices. To aid the modern controls designers, software packages such as MATLAB and MATRIX<sub>X</sub> have provided algorithms that automate this process. These routines take as inputs

the plant and weighting matrices and output the optimal gain matrix, solution to the Riccati equation, and the eigenvalues of the closed-loop plant.

In the first step of utilizing these algorithms the open-loop plant is separated into longitudinal and lateral parts. The longitudinal plant is composed of the  $u$ ,  $w$ ,  $q$ ,  $\theta$ , and  $z$  states. The lateral plant is composed of the  $v$ ,  $p$ ,  $r$ ,  $\phi$ , and  $\psi$  states. The control synthesis models for the longitudinal and lateral plants were then developed. In these models the states to be controlled and the actuators to achieve this control were chosen. The longitudinal synthesis model is shown in Figure 4.1. The states to be controlled in the steady state were the inertial velocity,  $u$ , and the altitude,  $z$ . The actuators used to achieve this control were the elevator and throttle. The lateral

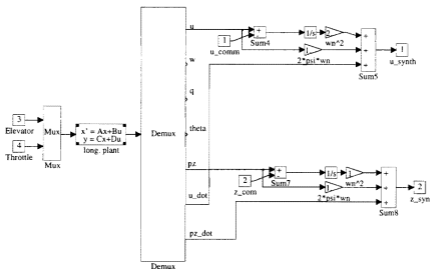


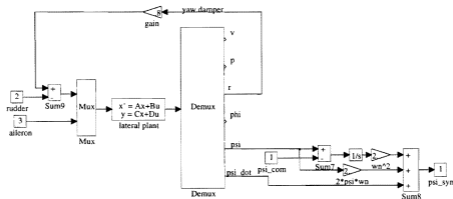
Figure 4.1: SIMULINK Longitudinal Control Synthesis Model

state to be controlled in the steady state was heading,  $\psi$ . The actuators recruited to

control  $\psi$  were the ailerons. Additionally, to increase lateral stability, a yaw damper was implemented by feeding back yaw rate through a gain block directly to rudder input. The choice of gain is arbitrary and the final value was obtained via a trial and error process by examining the step response of the lateral states over a range of gains. The lateral synthesis model is shown in Figure 4.2. From an examination of the closed-loop from rudder input to yaw rate it can be shown that the time response of the yaw rate is a decreasing exponential function of the form:

$$\frac{r}{\delta_r} = C_1 e^{-C_2 t} \quad (4.6)$$

where the constants,  $c_1$  and  $c_2$ , are elements of the open-loop  $A$  and  $B$  matrices. Therefore this implementation of the yaw damper continuously commands yaw rate to zero.



**Figure 4.2: SIMULINK Lateral Control Synthesis Model**

Note the addition of zeros to all the states being controlled. In theory, as the weights on the controls go to zero, the closed-loop poles will go to the open-loop

zeros of the synthesis model. This ensures that the closed-loop poles have the desired damping ratio and natural frequency to obtain the desired system response.

Once the control synthesis models were built, they were linearized using the 'linmod' function of MATLAB. Along with the choices for the weighting matrices, these linear plant models provided the inputs to the 'lqr' function of MATLAB. This routine calculates and outputs the optimal gain along with the solution to the Riccati equation and the closed-loop eigenvalues. An .m file was written to compute the control gains for both the longitudinal and lateral controllers and is detailed in Appendix C. Utilizing a design technique presented in [Ref. 8], the optimal gains were multiplied by a factor of two. This had the effect of suppressing the magnitude of the frequency response at the resonant frequency, while slightly increasing the bandwidth of the response. The exact mechanics of this effect are unclear and are recommended for further study. This program also computes and plots the frequency response for the command and control loops of the controller as well as the time domain response to step command inputs. These plots are also detailed in Appendix C.

Table 4.1 lists the closed-loop eigenvalues. All modes are stable and minimum longitudinal damping is only slightly less than the desired value of 0.5. Figures C.3, C.5, C.8 in Appendix C detail the responses to step altitude, velocity, and heading commands. Both the altitude and velocity responses, Figures C.3, C.5, easily meet the 4.0 seconds rise time and show no overshoot to the step commands. The heading response overshoot, Figure C.8, is well under the established requirement at less than 1.0

Finally, Figures C.1, C.2, C.7 of Appendix C indicate that the controller is keeping the actuator responses at or below the desired 4 rad/sec bandwidth, with minimal magnitude disturbances at the resonant frequencies.

Mode	Eigenvalue	Damping, $\zeta$	Frequency(rad/sec), $\omega$
Longitudinal			
	$-4.3144 + 4.0187j$	.7317	5.8961
	$-4.3144 - 4.0187j$	.7317	5.8961
	-1.0933	1.0	1.0933
	$-.4846 + .8640j$	.4892	.9906
	$-.4846 - .8640j$	.4892	.9906
	$-.2692 + .2580j$	.7219	.3728
	$-.2692 - .2580j$	.7219	.3728
Lateral-Directional			
	$-1.2874 + 1.8991j$	.5611	2.2943
	$-1.2874 - 1.8991j$	.5611	2.2943
	$-.4347 + .5657j$	.6093	.7134
	$-.4347 - .5657j$	.6093	.7134
	-4.5604	1.0	4.604
	-.6556	1.0	.6556

TABLE 4.1: Bluebird CLOSED-LOOP EIGENVALUES

MATRIX<sub>X</sub> also has the appropriate algorithms to calculate an optimal controller using the solution to the Ricatti Equation. The above results based on MATLAB and SIMULINK were verified by Byerly [Ref. 5] using the the MATRIX<sub>X</sub> codes with SystemBuild block diagrams. These results will not be repeated here. However, to allow for hardware-in-the-loop testing with the Futaba actuators, the controller will be implemented in SystemBuild as well as SIMULINK. This work is discussed next.

## V. CONTROLLER IMPLEMENTATION AND TESTING

The previous chapter detailed the design and analysis of the linear controller. The next step is to implement the controller with the aircraft block diagram model and subject the feedback system to external commands and disturbances. The software packages of MATLAB and MATRIX<sub>X</sub> each offer graphical design tools, SIMULINK and SystemBuild respectively, in which to implement the controller. The MATLAB / SIMULINK combination offers the advantage of familiarity. It is the standard engineering software for the Aeronautics and Astronautics Department at the Naval Postgraduate School. The nonlinear equations of motion for a vehicle moving through three-dimensional space have been well developed in the .m file format of MATLAB [Ref. 3, 4] and SIMULINK allows for the easy implementation of these user defined functions into the block diagrams representing the aircraft plant. The major disadvantage is the inability to generate autocode. The MATRIX<sub>X</sub> / SystemBuild combination allows the user to automatically generate C-code from the system block diagram. This computer code can then be run on any number of digital computers that are currently available. The disadvantage of using this software is that it is not as widely used and therefore not as familiar as MATLAB.

As noted above, each software package and its associated graphical design application have their advantages and disadvantages. For this reason the controller was first implemented using SIMULINK. This implementation was accomplished in four steps:

- Linear controller with the linear plant, initial conditions equal to zero.

- Differential controller with the linear plant, initial conditions equal to zero.
- Differential controller with the nonlinear plant, initial conditions equal to the trim states.
- Discretize the differential controller and nonlinear plant to be compatible with a digital control computer.

Once the aircraft/controller model was verified, the complementary model was developed using SystemBuild and verified once again. Then the SystemBuild model was discretized and autocode generated.

## A. CONTROLLER STRUCTURE

The optimal control gain,  $K$ , was obtained using the 'lqr' algorithm of MATLAB. This algorithm used a linearized model of the synthesis model as part of its input variables. These synthesis models used integrators to develop the integral error between the actual state and the commanded state. Thus the control gain can be separated into proportional and integral gain matrices and the controller must have both proportional and integral error portions. The proportional part assumes all the state variables are measurable. These variables are then used for state feedback. The integral portion consists of integrators on the error signal generated between the actual state and the commanded state. The state space equations of the controller take the following form:

$$\begin{aligned}\dot{x}_c &= B_c(y - y_c) \\ u &= K_i x_c + K_p x\end{aligned}\tag{5.1}$$

where,

- $y = \begin{bmatrix} u \\ \psi \\ z \end{bmatrix}$ , is the vector of actual outputs.
- $y_c = \begin{bmatrix} u_c \\ \psi_c \\ z_c \end{bmatrix}$ , is the vector of commanded outputs.
- $K = [K_p, K_i]$

It is from this form that the SIMULINK implementation of the controller was developed. The details of this development are covered in the following sections.

## B. LINEAR CONTROLLER, LINEAR PLANT MODEL

The linear controller is shown in Figure 5.1. The controller takes the optimal

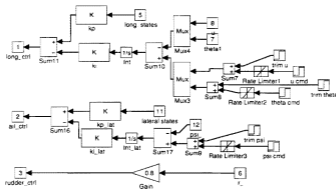


Figure 5.1: SIMULINK Linear Control Model

gain matrix calculated using the 'lqr' of MATLAB and breaks it into a matrix of proportional gains corresponding to the states and a matrix of integral gains corresponding to the integral error states. The control outputs were the commands for the longitudinal controls (elevator and throttle), the lateral control (ailerons), and the yaw damper (rudder). The complete closed-loop linear control/linear plant model



is shown in Figure 5.2. The block named *Long-Lat.Ctrl* contains the linear control detailed in Figure 5.1.

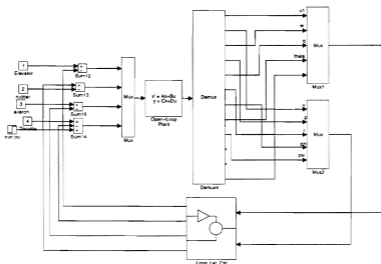


Figure 5.2: SIMULINK Closed-Loop Linear Control Model

### C. DELTA CONTROLLER, LINEAR PLANT MODEL

The linear controller model of the previous section was designed for the linearized model of the aircraft dynamics. This linear plant model was obtained by trimming the nonlinear model about some trim flight state and then linearizing the model at that state. This entire process is performed under the assumption of small perturbations about the trim state. Thus the linear controller on the linear plant is valid only for minor deviations about the trim position. It is then necessary to obtain small perturbations of the full states. This is achieved by differentiating the states prior to multiplication by the proportional gains and then integrating at the controller

output.

The complete delta controller is implemented as shown in Figure 5.3. The integrator has been removed from the integral error states and a differentiator has been added to the full states prior to the proportional gains. Once these signals have been summed the result is fed through an integrator, whose initial conditions equal the trim states. The resulting outputs are the control commands.

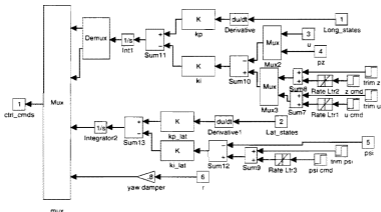


Figure 5.3: Delta Controller Model

## D. DELTA CONTROLLER, NONLINEAR PLANT MODEL

The linear controller used a combination of throttle and elevator to control inertial velocity,  $u$ . This control is extremely accurate for the case of an aerial vehicle moving through a still air mass, as the vehicle true airspeed (TAS) and inertial velocity (ground speed) are identical. However, for motion in a moving air mass it is desired to control the vehicle TAS. This is necessary to prevent stall. TAS is computed in two steps:

- Wind vectors in the three inertial coordinate axis are used as inputs to the nonlinear equations of motion. This includes the effects of air mass velocities in the calculations of the vehicle states.
- The inertial velocities are then summed with the wind velocity components and the magnitude of their vector sum is determined by

$$TAS = \sqrt{(u + w_x)^2 + (v + w_y)^2 + (w + w_z)^2} \quad (5.2)$$

where,  $w_x$ ,  $w_y$ , and  $w_z$  are the wind components in the three inertial directions.

This calculation is shown in the 'Fcn' block of Figure 3.2 and in the block 'TAS CALC' of Figure 3.3. The inertial velocity,  $u$ , is then replaced by TAS to determine the velocity error. The actual controller implementation remains the same as shown in Figure 5.3. The closed-loop plant takes the form as shown in Figure 5.4, where the block 'EOM/wind' contains the implementation of the nonlinear equations of motion shown in Figure 3.2.

## E. DISCRETE CONTROLLER IN SIMULINK AND System-Build

From the outset, the controller designed in this thesis was intended to operate on a digital flight control computer. For this reason the delta analog controller from the preceding section must be discretized. This process involves implementing the controller in the discrete state-space form given by:

$$\begin{aligned} (x_c)_{k+1} &= x_k + \Delta T * K_i(y_1 - y_c)_k \\ u_k &= x_{ck} + K_p y_{2k} \end{aligned} \quad (5.3)$$

where,

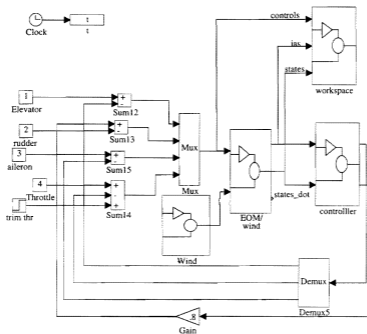


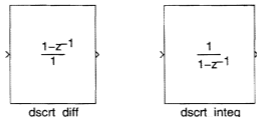
Figure 5.4: Delta Control Nonlinear Plant

- $\Delta T$  is the sampling interval.
- $y_1 = \begin{bmatrix} TAS \\ \psi \\ z \end{bmatrix}$ , is the vector of actual outputs.
- $y_c = \begin{bmatrix} TAS_c \\ \psi_c \\ z_c \end{bmatrix}$ , is the vector of commanded outputs.
- $y_{2k}$  the complete vector of state outputs.

The sampling interval of  $0.1sec$  was chosen to be greater than twice the highest modal frequency of the uncontrolled plant as shown in Table 4.1. This rate will

provide for complete recovery of the analog signal, as per the Nyquist Sampling Theorem ([Ref. 11]), and is well within the capabilities of the available digital computers.

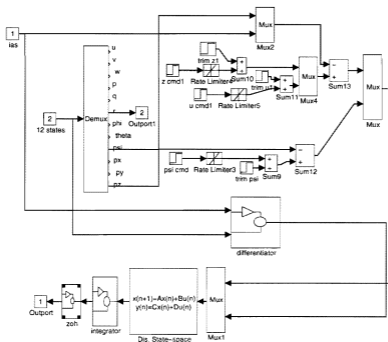
Additional modifications included discretizing the differentiators on each state and the integrators on each control signal just prior to the controller output. The discrete time implementation of these system elements is shown in Figure 5.5



**Figure 5.5: Discrete Differentiator & Integrator**

The final modification results from the fact that the use of a digital control induces some time delay in the system. To model this effect a zero-order hold is added to the output of the discrete integrator. This block effectively holds the control signal for the length of the sampling interval until the next update of the control signal is transmitted. Therefore a piecewise continuous control signal is fed to the continuous time nonlinear model of the aircraft dynamics.

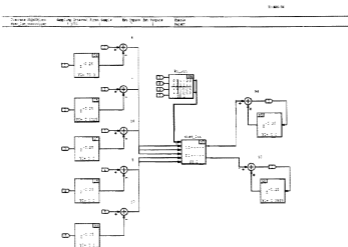
The complete discrete controller is detailed in Figure 5.6. It is implemented with the nonlinear model in the same fashion as was the differential controller, shown in Figure 5.4.



**Figure 5.6: SIMULINK Discrete Control Model**

Once the SIMULINK modeled was validated, the discrete controller–nonlinear plant system was developed in SystemBuild. The discrete longitudinal and lateral controllers are shown in Figures 5.7,5.8. The differentiators and integrators are implemented as feedforward and feedback loops respectively. The result is the same as for the SIMULINK implementation shown in Figure 5.5. The blocks incorporating the integral gains, also contain the sampling interval. The resulting output executes the algebraic expression,

$$(x_c)_{k+1} = x_k + \Delta T * K_i(y_1 - y_c)_k$$



**Figure 5.7: SystemBuild Discrete Longitudinal Control Model**

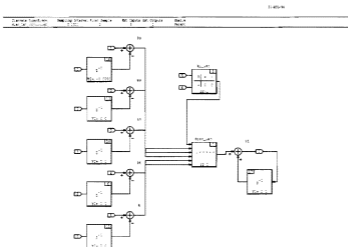
The blocks entitled 'disc\_lon' and 'disc\_lat' complete the discrete controller calculation by executing the expression,

$$u_k = x_{ck} + K_p y_{2k}$$

The complete closed-loop model implemented in SystemBuild is shown in Figure 5.9.

## F. MODEL VERIFICATION

To verify the stability and performance of the closed-loop system, the model was subjected to a variety of command and control inputs and external disturbances. The state outputs were then evaluated for stability, steady state error, rise time, and overshoot. This process was accomplished at each step of the control implementation



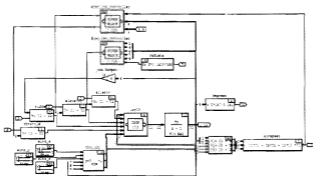
**Figure 5.8: SystemBuild Discrete Lateral Control Model**

process and was the prerequisite for continuation to the next step. Results of the final closed-loop time history performance are shown in Appendix C.

The initialization period of about 10 secs was a result of being unable to set initial conditions in the discrete integrators in the SIMULINK model. There was no such problem when implementing the integrators in SystemBuild. Commanded inputs and disturbances were introduced at the following intervals and magnitudes,

- Commanded TAS increase of 10 ft/sec at time 20 secs.
- Commanded altitude decrease of 50 feet at time 40 secs.
- Commanded heading change of 90 degrees at time 60 secs.
- Input a headwind of 10 ft/sec at time 80 secs and removed it at time 100 secs.





**Figure 5.9: SystemBuild Discrete Control Model**

As the plots of Appendix D show, the controller tracks heading (bottom plot of Figure D.3), altitude (top plot of Figure D.4), and TAS (bottom plot of Figure D.4) with zero steady state error. The feedback system is stable and handles the large magnitude heading change by limiting the angle of bank to a maximum of 30 degrees. This limiting was achieved setting the rate limiter on the heading command to  $\pm 0.2536$  rad/sec (approximately 14 degrees/sec). This value was determined by examining the dynamics of an aircraft in steady, level, turning flight. The following relationships are obtained,

$$L \cdot \cos \phi = mg$$

$$L \cdot \sin \phi = \frac{V^2}{Rg}$$

and.

- $L = \text{lift}$
- $W = mg = \text{weight}$
- $\phi = \text{bank angle}$
- $R = \text{turn radius}$
- $V = \text{TAS}$

Dividing the above two equations and solving for the turn radius results in,

$$R = \frac{V^2}{g * \tan \phi} \quad (5.4)$$

From the kinematics of a rotating body, the expression for the angular rate (which is also the turn rate) is given by  $\dot{\psi} = V/R$ . Substituting for  $R$  in this expression,

$$\dot{\psi} = \frac{g * \tan \phi}{V} \quad (5.5)$$

To obtain the maximum turn rate, and thus the bounds for the rate limiter, it is necessary only to choose the maximum bank angle and substitute into Equation 5.5.

## VI. CONCLUSIONS AND RECOMMENDATIONS

### A. CONCLUSIONS

The design process and tools investigated in this thesis allow the designer to obtain the optimal controller, model the closed-loop plant/controller system, and evaluate the controller's performance using a computer workstation. The benefits of such a process are numerous:

- Optimal controllers can be designed, evaluated, and modified quickly and easily. Simply by adjusting the weighting matrices of the cost function, the designer can obtain any desired system response.
- Incorporated with a high fidelity model of the aircraft dynamics, the controller can be validated throughout a variety of flight regimes. This process proves to be cost effective and obtains a level of safety not achievable in actual flight test.
- Software tools such as MATRIX<sub>X</sub> and its graphical design application, SystemBuild, provide the added benefit of automatically generating the computer code required for operational digital flight control systems. The costly and time consuming process of generating the thousands of lines of codes required by these systems is reduced to development of a block diagram and the click of a mouse button.

## B. RECOMMENDATIONS

Based upon the conclusions and experience gained during this research project, the following recommendations are made:

- The `MATRIXX` / SystemBuild software design package should become an integral element of the Avionics curriculum at the Naval Postgraduate School. The ability to generate real-time *C*-code makes this combination second to none in the area of system simulation and control design.
- Using the autocode generated from the discrete SystemBuild closed-loop system, hardware-in-the-loop testing of the Bluebird controller should be accomplished.
- Incorporate sensor and Kalman Filter models into the SystemBuild block diagram. These models have been developed using `SIMULINK`. [Ref. 4], and must be developed for the SystemBuild model to provide a complete model.
- Currently SystemBuild has no block diagram available to model rate limiters for the commanded state inputs. These are necessary because the commands are usually modeled as step inputs. For large step changes in the commanded states, the controller will employ large control deflections to reach the desired state. In some instances, this response can be outside the vehicle's dynamic operating envelope. Rate limiters employed in the `SIMULINK` models have shown the capability to limit roll angle, longitudinal acceleration, and vertical acceleration to within vehicle operating limits. The same must be done for the SystemBuild model.

## APPENDIX A

### CODED EQUATIONS OF MOTION

#### A. STATE\_DERIV.M

```

function accel = state.deriv(x)
% calculates the accelerations (angular and linear) due to
% aero forces; w X v; gravity.
% Variables brought from workspace:
% x = combination vector [ contrl inputs, state variables, euler angles]
% (da, de, dr, dtrt, u, v, w, p, q, r, phi, theta, psi)
% Variables called from function "blue_data"
% rho = air density
% b = wing span
% c = wing cord
% s = wing area
% Cfo = Steady state force term
% Cfu = Stability derivative for control inputs
% m = airplane mass
% Ib = inertia tensor matrix (body frame)
% To = Thrust scale term
% Pe = Engine postion matrix
% get the aircraft data
[ u0,w0,rho,Cfx,Cfo,Cfu,Cfxdot,s,b,c,m,Pe,To,Ib] = blue_data;
% seperate the combined vector into seperate elements
u = [ x(1); x(2); x(3) ] ;
dtrt = x(4);
state = [ x(5); x(6); x(7); x(8); x(9); x(10) ] ;
lambda = [ x(11); x(12); x(13) ] ;
wind = [ x(14); x(15); x(16) ] ;
%%%%%%%%% calculate velocity wrt the airmass and form state vector
%%%%%%%%% that will be used to calculate the aerodynamic forces/moments
ias = [ state(1); state(2); state(3) ] + wind;
state1 = [ ias(1)-u0; ias(2); ias(3)-w0; x(8); x(9); x(10) ] ;
%%%%%%%%% calculate total velocity, vt
vt = (ias(1)^2 + ias(2)^2 + ias(3)^2)^.5;

```

```

% calculate qbar
qbar = .5* rho* (vt^ 2);

% calculate M1
M1 = diag([ 1/vt, 1/vt, 1/vt, (.5* b)/vt, (.5* c)/vt, (.5* b)/vt ]);

% calculate M2
M2 = diag([ 0, 0, (.5* c)/(vt^ 2), 0, 0, 0 ]);

% calculate Sprime
Sprime = diag([ -1, 1, -1, b, c, b ] * s);

% calculate Mu
Mu = inv([ eye(3)* m,zeros(3);zeros(3),Ib ]);

% calculate Tw2b
alpha = state(3)/vt;
beta = state(2)/vt;
T1 = [ cos(alpha), 0, -sin(alpha); 0,1,0; sin(alpha), 0, cos(alpha) ] ;
T2 = [ cos(beta), -sin(beta), 0; sin(beta), cos(beta), 0; 0,0,1 ] ;
Tw2b = [ T1* T2, zeros(3); zeros(3), T1* T2 ] ;

% calculate Chi
Chi = eye(6) - Mu* Tw2b* qbar* Sprime* Cfxdot* M2;

% calculate Propulsion matrix
Prop = [ eye(3);
0,-Pe(3),Pe(2);
Pe(3),0,-Pe(1);
-Pe(2),Pe(1),0] ;

% calculate gravity vector and rotation matrix
Rot = [ 1, 0, -sin(lambda(2));
0, cos(lambda(1)), cos(lambda(2))* sin(lambda(1));
0, -sin(lambda(1)), cos(lambda(2))* cos(lambda(1)) ] ;
Ru2b = [ Rot;zeros(3) ] ;
g = [ 0; 0; 32.174 ] ;
FgU = m* g;

% put the components due to gravity; thrust; and control surface deflections
% together for their contribution to x-dot
thrust = Prop* To* dtrt;
gravity = Ru2b* FgU;
ctrl=qbar* (Tw2b* (Sprime* (Cfo + (Cfu* u))));

xdotu=(Mu* (ctrl+thrust+gravity));

```

```

% calculate rotation matrix
omegax = [ 0,-state(6),state(5);state(6),0,-state(4);-state(5),state(4),0] ;
wxIb = (-inv(Ib))* (omegax* Ib);
Rot = [ -omegax, zeros(3); zeros(3), wxIb] ;
% rotation component of xdot (w X v)
xdotrot = Rot* state;
% state vector feedback component xdot
xdotcfx =qbar* (Mu* (Tw2b* (Sprime* (Cfx* (M1* statel)))));
% add three components of xdot together and premult by inv(Chi)
xdot= inv(Chi)* (xdotrot+xdotcfx+xdotu);
% calc accel that a strap-down IMU would measure
xdotb=xdot-xdotrot-Ru2b* g;
% put together for the return vector
%accel=[ xdotb(1);xdotb(2);xdotb(3);xdot] ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% return xdot only
accel=xdot;

```

## B. EOM2\_WIND.C

```

/* -----
   Aircraft Equations of Motion:   by Jim Byerly
   Revision 1.1:                   by Brian T. Foley
   ----->*/

```

```

/* -----
   MATRIXx/WS TM Software V2.4   (C) Copyright 1990
   INTEGRATED SYSTEMS INC., Santa Clara, California
   Unpublished Work. All Rights Reserved Under The U.S. Copyright Act.

   RESTRICTED RIGHTS NOTICE : Use, Reproduction Or Disclosure Is
   Subject To Restrictions Set Forth In The Rights In The
   Technical Data And Computer Clause At 252.227-7013 And The
   Equivalent Provisions In Other Agency's Regulations.
   ----->*/

```

```

/*
=====
This template is provided to let users write their own C
Code blocks to implement User Code Blocks. See the UCB chapter
in your SystemBuild manual for further information.
=====
*/

```

If you are writing more than one UCB, you will need a separate copy of this template for each UCB, and you will have to give each one a different name: `usr01`, `usr02`, ... `usrnn` are conventional names.

Certain lines of code are required for definition of the code block and its calling sequence. Do not modify them except as described in the preceding paragraph. The following line is the first such.

```
=====
/
#include { stdlib.h}
#include { math.h}
#include "matsrc.h"

#if (FC-)
#define usr03 usr03_
#endif

/* Aircraft Equations of Motion */
/* Subroutines */

void inverse(int n);
void cross_prod(double(* px1), double(* py2), double(* pz3));
void rot_u2b(double phi, double theta, double psi);
void rot_w2b(double alpha, double beta);
void m_mult_331(double(* pa1)[ 3], double(* pa2), double(* pa3));
void m_mult_333(double(* pa1)[ 3], double(* pa2)[ 3], double(* pa3)[ 3] );
void m_mult_631(double(* pa1)[ 3], double(* pa2), double(* pa3));
void m_mult_661(double(* pa1)[ 6], double(* pa2), double(* pa3));
void m_mult_666(double(* pa1)[ 6], double(* pa2)[ 6], double(* pa3)[ 6] );

/* External matrices */

double ainv[ 6][ 12];
double tu2b[ 6][ 3], tw2b[ 6][ 6];

void USR03( iinfo,rinfo, u,nu, x,xdot,nx, y,ny, rpar,ipar )

double rinfo[ ], rpar[ ], u[ ], x[ ], xdot[ ], y[ ];
int iinfo[ ], * nu, * nx, * ny;
long ipar[ ];

/*
=====

```

SystemBuild User Code Function Block Template

```
iinfo[ 0] { 0 : Error Occurred : { -10* Message Index + Status }
          Return Status : { 0:Normal | -1:Warning | -2:Error }
```

```
Index Error Message
```



- 0 Provide your own message via MATWR.
- 1 Division by 0.0 produces infinity.
- 2 Raise 0.0 to a non-positive power.
- 3 Both arguments to ATAN2 are zero.
- 4 ASIN or ACOS argument out of range.
- 5 Natural log of zero or negative number.
- 6 Square root of negative number.
- 7 Incoming data not in range of table.
- 8 Raise negative number to non-integer.
- 9 Overflow in  $y = \text{EXP}(u)$  function.

```

+-----+
iinfo[ 1] =1 : Initialize. (First call in SIM).
iinfo[ 2] =1 : Compute the state derivatives.
iinfo[ 3] =1 : Compute the outputs.
iinfo[ 4]      : Reserved.
iinfo[ 5]      : Reserved.
iinfo[ 6]      : Reserved.
iinfo[ 7]      : Reserved.
iinfo[ 8] =i : Number of Integer Parameter Values.
iinfo[ 9] =r : Number of Real      Parameter Values.
iinfo[10] =1 : Integration Algorithm (IALGORITHM).
iinfo[11] =1 : Inside Linearization Process
              (Jacobian Computations).
iinfo[12] =1 : Inside SIM Initialization Process (Time=0).
iinfo[13]      : Update with converged state values.
+-----+

```

Rinfo is a real array of dimension 4 with timing and related information for the called routine. User code may not modify any values in rinfo.

	Continuous	Discrete	Triggered
rinfo[ 0]	: Current Time	Current Time	Current Time
rinfo[ 1]	: 0.0	Sample Interval	1.0
rinfo[ 2]	: 0.0	Initial Time Skew	Timing Requirement
rinfo[ 3]	: reltol	reltol	reltol

The input arguments are defined as follows. Note that initial conditions defined in the UCB block form are passed in through  $x$ , the state vector. The state vector can be modified only on the first call to the UCB, when  $\text{init} = 1$ . LIN UCB's are never called with  $\text{init} = 1$ . Hence, the operating point can only be changed from  $\text{usr01}$ .

```

u[ nu] = input vector dimensioned number of inputs (nu).
x[ nx] = state vector dimensioned number of states (nx).
xdot[ nx] = first derivative with respect to time of the

```

```

state vector. If the UCB is in a discrete Super-
block, xdot represents xnext, the value of x at
the next sample time.
y[ ny]      = output vector, dimensioned number of outputs (ny)
rpar[ nr]   = general vector of real double precision parameter
              (double precision R* 8), to be initialized in
              MATRIXx and passed to the UCB. Nr is requested in
              the SystemBuild Block form of the UCB as 'dimension
              of RPAR'and passed to the UCB as iinfo[ 9] .
ipar[ ni]   = general vector of integer (I* 4) parameters to be
              initialized in MATRIXx and passed to the UCB. Ni
              is requested as 'dimension of IPAR'in the block
              form of the UCB in SystemBuild, and passed to the
              UCB as iinfo[ 8] .
+-----+
| The following two lines of code are not to be modified. |
+-----+
/
{
    int    init, state, output, messg;
    double time, tsamp, tskev;

+-----+
/* Enter user variable declarations
+-----+
/

double vbo[ 3] , wbo[ 3] , lambda[ 3] , delu[ 3] , delt, phi, theta, psi;
double vt, alpha, beta, qbar, mass, s[ 6] , sbar[ 6] [ 6] , m_prime[ 6] [ 6] ;
double m_dot_prime[ 6] [ 6] , mi[ 6] [ 6] , miinv[ 6] [ 6] , chi[ 6] [ 6] ;
double chiinv[ 6] [ 6] , inv_inertia[ 3] [ 3] , wcv[ 3] , iwci[ 3] , fwvob[ 6] ;
double wght[ 3] , fgrav[ 6] , del_vw[ 6] , faero[ 6] , fcntl[ 6] , vwnew[ 6] ;
double slam[ 3] [ 3] , euang[ 3] , rb2u[ 3] [ 3] , dpos[ 3] ;
double tmp1[ 6] , tmp3[ 6] [ 6] , tmp5[ 3] , windx, windy, windz;
double a, b, c;
double wt = 55.0, rho = .0023769, warea = 22.38, span = 12.42;
double chord = 1.802, grav = 32.174, T0 = 15.0, V0 = 73.3, alph0 = 0.0;
double cfx[ 6] [ 6] , cfu[ 6] [ 3] , cfx_dot[ 6] [ 6] , cf0[ 6] , inertia[ 3] [ 3] ;
int i, j, n;

+-----+
/* The following six lines of code are not to be modified.
+-----+
/

init    = (iinfo[ 1] !=0);
state  = (iinfo[ 2] !=0);
output = (iinfo[ 3] !=0);
time   = rinfo[ 0] ;

```

```

    tsamp = rinfo[ 1 ] ;
    tskev  = rinfo[ 2 ] ;
/* -----+
| Replace the following with user code.
|=====+
/
/* define aircraft data */
/* Zeroize Arrays */
for (i=0; i< 6; i++)
{
    for (j=0; j< 6; j++)
    {
        cfx[ i ] [ j] = 0.0;
        cfx_dot[ i ] [ j] = 0.0;
    }
    cf0[ i] = 0.0;
}
for (i=0; i< 6; i++)
    for (j=0; j< 3; j++)
        cfu[ i ] [ j] = 0.0;
for (i=0; i< 3; i++)
    for (j=0; j< 3; j++)
        inertia[ i ] [ j] = 0.0;
/* define elements of arrays */
/* *** COLUMNS: U, ALP, BETA, P, Q, R */
/* *** ROWS: CD, CY, CL, Cl, Cm, Cn */
cfx[ 0 ] [ 2] = 0.188;
cfx[ 1 ] [ 1] = -0.31;
cfx[ 1 ] [ 5] = 0.0973;
cfx[ 2 ] [ 2] = 4.22;
cfx[ 2 ] [ 4] = 3.94;
cfx[ 3 ] [ 1] = -0.0597;
cfx[ 3 ] [ 3] = -0.363;
cfx[ 3 ] [ 5] = 0.1;
cfx[ 4 ] [ 2] = -1.163;
cfx[ 4 ] [ 4] = -11.77;
cfx[ 5 ] [ 1] = 0.0487;
cfx[ 5 ] [ 3] = -0.0481;
cfx[ 5 ] [ 5] = -0.0452;
/* *** COLUMNS: da, de, dr */
/* *** ROWS: CD, CY, CL, Cl, Cm, Cn */
cfu[ 0 ] [ 1] = 0.065;
cfu[ 1 ] [ 2] = 0.0697;
cfu[ 2 ] [ 1] = 0.472;

```

```

cfu[ 2] [ 2] = 0.0147;
cfu[ 3] [ 0] = 0.265;
cfu[ 3] [ 2] = 0.0028;
cfu[ 4] [ 1] = -1.41;
cfu[ 5] [ 0] = -0.0347;
cfu[ 5] [ 2] = -0.0329;

/* *** COLUMNS: U, ALP, BETA, P, Q, R */
/* *** ROWS: CD, CY, CL, Cl, Cm, Cn */

cfx.dot[ 2] [ 2] = 1.32;
cfx.dot[ 4] [ 2] = -4.7;

/* *** ROWS: CD0, CY0, CL0, Cl0, Cm0, Cn0 */

cf0[ 0] = 0.03;
cf0[ 2] = 0.385;

/* Inertial elements */

inertia[ 0] [ 0] = 12.58;
inertia[ 1] [ 1] = 13.21;
inertia[ 2] [ 2] = 19.99;

/* Split up input variables */
for (i=0; i< 3; i++)
{
    vbo[ i] = u[ i] ;
    wbo[ i] = u[ i+3] ;
    lambda[ i] = u[ i+6] ;
    delu[ i] = u[ i+9] ;
}
delt = u[ 12] ;
windx = u[ 13] ;
windy = u[ 14] ;
windz = u[ 15] ;

phi = lambda[ 0] ;
theta = lambda[ 1] ;
psi = lambda[ 2] ;

a = vbo[ 0] - windy;
b = vbo[ 1] - windz;
c = vbo[ 2] - windz;
vt = sqrt(a*a + b*b + c*c);
alpha = asin(vbo[ 2] /vt);
beta = asin(vbo[ 1] /vt);

qbar = .5 * rho * vt * vt;
mass = wt / grav;

/* Build S Bar Matrix */
s[ 0] = -warea;

```

```

s[1] = warea;
s[2] = -warea;
s[3] = warea * span;
s[4] = warea * chord;
s[5] = warea * span;

for (i=0;i< 6;i++)
{
    for (j=0; j< 6; j++)
        sbar[i][j] = 0.0;
    sbar[i][i] = s[i];
}

/* Build m-prime matrix */
s[0] = 1./vt;
s[1] = 1./vt;
s[2] = 1./vt;
s[3] = span / (2.* vt);
s[4] = chord / (2.* vt);
s[5] = span / (2.* vt);

for (i=0;i< 6;i++)
{
    for (j=0; j< 6; j++)
        m_prime[i][j] = 0.0;
    m_prime[i][i] = s[i];
}

/* Build m_dot_prime matrix */
s[0] = 0.0;
s[1] = 0.0;
s[2] = chord / (2.* vt * vt);
s[3] = 0.0;
s[4] = 0.0;
s[5] = 0.0;

for (i=0;i< 6;i++)
{
    for (j=0; j< 6; j++)
        m_dot_prime[i][j] = 0.0;
    m_dot_prime[i][i] = s[i];
}

/* Build mass_inertia matrix */
s[0] = mass;
s[1] = mass;
s[2] = mass;
s[3] = 1.0;
s[4] = 1.0;
s[5] = 1.0;

```

```

for (i=0; i< 6; i++)
{
    for (j=0; j< 6; j++)
        mi[i][j] = 0.0;
    mi[i][i] = s[i];
}
for (i=3; i< 6; i++)
    for (j=3; j< 6; j++)
        mi[i][j] = inertia[i-3][j-3];

/* Compute the inverse of the matrix */
n=6;
for (i=0; i< n; i++)
    for (j=0; j< n; j++)
        ainv[i][j] = mi[i][j];
inverse(n);
for (i=0; i< n; i++)
    for (j=0; j< n; j++)
        miinv[i][j] = ainv[i][j+n];

/* compute rotation matrices:  wind-to-body; inertial-to-body */
rot_w2b(phi, theta, psi);
rot_w2b(alpha, beta);

/* Compute Chi */
m_mult_666(cfx_dot, m_dot_prime, tmp3);
m_mult_666(sbar, tmp3, chi);
m_mult_666(tw2b, chi, tmp3);
m_mult_666(miinv, tmp3, chi);
for (i=0; i< 6; i++)
{
    for (j=0; j< 6; j++)
        chi[i][j] = -qbar * chi[i][j];
    chi[i][i] = 1. + chi[i][i];
}

/* Compute the inverse of the matrix */
n=6;
for (i=0; i< n; i++)
    for (j=0; j< n; j++)
        ainv[i][j] = chi[i][j];
inverse(n);
for (i=0; i< n; i++)
    for (j=0; j< n; j++)
        chiinv[i][j] = ainv[i][j+n];

```

```

/* Compute w x v */
cross_prod(wbo, vbo, wcv);
/* Compute inverse of inertia matrix */
n=3;
for (i=0; i< n; i++)
    for (j=0; j< n; j++)
        ainv[i][j] = inertia[i][j];
inverse(n);
for (i=0; i< n; i++)
    for (j=0; j< n; j++)
        inv_inertia[i][j] = ainv[i][j+3];
/* Compute I \ (w x Iw) */
m_mult_331(inertia, wbo, iwci);
cross_prod(wbo, iwci, tmp5);
m_mult_331(inv_inertia, tmp5, iwci);
for (i=0; i< 3; i++)
{
    fvwob[i] = -wcv[i];
    fvwob[i+3] = -iwci[i];
}
/* Compute forces due to gravity */
wght[0] = 0.0;
wght[1] = 0.0;
wght[2] = wt;
m_mult_631(tu2b, wght, fgrav);
/* Compute aero forces and moments */
del_vw[0] = vbo[0] - V0 * cos(alph0);
del_vw[1] = vbo[1];
del_vw[2] = vbo[2] - V0 * sin(alph0);
del_vw[3] = wbo[0];
del_vw[4] = wbo[1];
del_vw[5] = wbo[2];
m_mult_661(m_prime, del_vw, tmp1);
m_mult_661(cfx, tmp1, faero);
m_mult_631(cfu, delu, fcntl);
for (i=0; i< 6; i++)
    faero[i] = (faero[i] + cf0[i] + fcntl[i]) * qbar;
m_mult_661(sbar, faero, tmp1);
m_mult_661(tw2b, tmp1, faero);

```

```

/* Compute linear (u,v,w) and angular (p,q,r) accelerations */
for (i=0; i( 6; i++)
    tmp1[i] = fgrav[ i] + faero[ i] ;
tmp1[ 0] = tmp1[ 0] + T0 * delt;
m_mult.661(miinv, tmp1, vwnew);
for (i=0; i( 6; i++)
    tmp1[i] = fwob[ i] + vwnew[ i] ;
m_mult.661(chiinv, tmp1, vwnew);
/* Compute euler angle derivatives */
slam [ 0] [ 0] = 1.0;
slam [ 0] [ 1] = sin(phi) * tan(theta);
slam [ 0] [ 2] = cos(phi) * tan(theta);
slam [ 1] [ 0] = 0.0;
slam [ 1] [ 1] = cos(phi);
slam [ 1] [ 2] = -sin(phi);
slam [ 2] [ 0] = 0.0;
slam [ 2] [ 1] = sin(phi) / cos(theta);
slam [ 2] [ 2] = cos(phi) / cos(theta);
m_mult.331(slam, wbo, euang);
/* Compute position derivatives */
for (i=0; i( 3; i++)
    for (j=0; j( 3; j++)
        rb2u[ i] [ j] = tu2b[ j] [ i] ;
m_mult.331(rb2u, vbo, dpos);
/* Set up output vector */
for (i=0; i( 6; i++)
    y[ i] = vwnew[ i] ;
for (i=0; i( 3; i++)
    y[ i+6] = euang[ i] ;
for (i=0; i( 3; i++)
    y[ i+9] = dpos[ i] ;
} /* End of main program */
/*
=====
/

```



## APPENDIX B

### OPEN-LOOP PLANT

#### A. OPEN-LOOP PLANT

a =

Columns 1 through 7

-0.0684	0.0000	0.2244	0	0.0021	0	0.0000
-0.0000	-0.3877	0.0000	-0.0023	0	-72.6109	32.1740
-0.8621	-0.0000	-4.7588	0	67.9933	0	-0.0002
0.0000	-0.1149	-0.0000	-4.3390	-0.0000	1.1953	0.0000
0.0132	-0.0000	-0.2362	0	-3.8650	0	0.0000
-0.0000	0.0590	0.0000	-0.3617	-0.0000	-0.3400	-0.0000
0	0	0	1.0000	-0.0000	-0.0000	-0.0000
0	0	0	0	1.0000	-0.0000	-0.0000
0	0	0	0	0.0000	1.0000	0.0000
1.0000	0	-0.0000	0	0	0	0
-0.0000	1.0000	-0.0000	0	0	0	0.0023
0.0000	0	1.0000	0	0	0	0.0000

Columns 8 through 12

-32.1740	0	0	0	0
0.0000	0	0	0	0
0.0008	0	0	0	0
-0.0000	0	0	0	0
-0.0000	0	0	0	0
0.0000	0	0	0	0
0.0000	0	0	0	0
0	0	0	0	0
-0.0000	0	0	0	0
-0.0004	-0.0004	0	0	0
0.0000	73.3000	0	0	0
-73.3000	0	0	0	0

b =			
-5.4349	-0.0000	0	8.7745
0.0000	5.8266	0	0
-38.7398	-1.2065	0	0
-0.0000	0.3949	37.3882	0
-26.8913	0.0185	0	0
0.0000	-2.9212	-3.0817	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

## B. OPEN-LOOP STEP RESPONSE PLOTS

The open-loop SystemBuild model was subjected to a step elevator input of approximately 3 degrees at a time of 30 seconds. The following plots detail the open-loop response of the plant model. The sinusoidal inertial velocity,  $u$ , shown in the top diagram of Figure B.1 demonstrates the underdamped phugoid mode. Also, the unstable spiral mode is evident from the linearly increasing plot of heading,  $\psi$ , in the bottom diagram of Figure B.3.

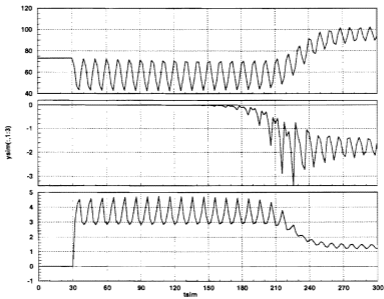


Figure B.1: Open-loop Inertial Velocity,  $u$   $v$   $w$

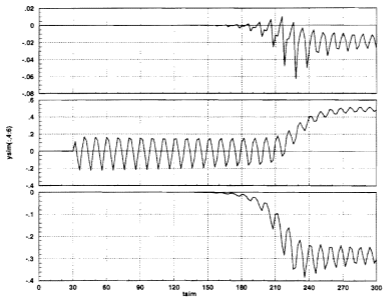


Figure B.2: Open-loop Inertial Rates,  $p$   $q$   $r$

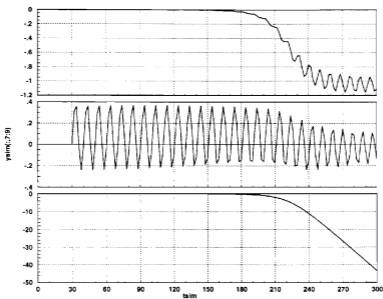


Figure B.3: Open-loop Euler Angles,  $\phi$   $\theta$   $\psi$

# APPENDIX C

## CONTROLLER CALCULATION/VERIFICATION

### A. BLUE.PLOT.M

```
% This file plots bode diagrams and step responses for the lqr  
%controller calculated .
```

```
load lin16
```

```
[ alat,blat,alon,bion] =latlon1(a);
```

```
%%%Compute longitudinal controller%%%
```

```
[ as,bs,cs,ds] =linmod('blue_ syn_ lon');
```

```
r=[ 110000 0:0 10000] ;
```

```
q=[ 1 0:0 1] ;
```

```
b1=bs(:,1:2);
```

```
b2=bs(:,3:4);
```

```
c1=as(6:7,:);
```

```
[ k,s,e] =lqr(as,b2,cs'* q* cs,r);
```

```
damp(e)
```

```
k=2* k;
```

```
kp=k(:,1:5);
```

```
ki=k(:,6:7);
```

```
pause
```

```
%compute control bode diagrams
```

```
bode(as-b2* k,b2,k(1,:),[ 0 0] ,1)
```

```
title('ele cmd loop')
```

```
pause
```

```
bode(as-b2* k,b2,k(2,:),[ 0 0] ,2)
```

```
title('thr cmd loop')
```

```
pause
```

```
%compute command bode diagrams and step responses.
```

```
step(as-b2* k,b1,c1(1,:),[ 0 0] ,2)
```

```
title('step z response')
```

```
pause
```

```
bode(as-b2* k,b1,c1(1,:),[ 0 0] ,2)
```

```
title('z cmd loop')
```

```

pause
step(as-b2* k,b1,c1(2,:),[ 0 0] ,1)
title('step u response')
pause
bode(as-b2* k,b1,c1(2,:),[ 0 0] ,1)
title('u cmd loop')
pause
%%Compute lateral controller%%
[ as_ lat,bs_ lat,cs_ lat,ds_ lat] =linmod('blue_ syn_ lat');
r_ lat=7000;
q_ lat=25;
b1_ lat=bs_ lat(:,1);
b2_ lat=bs_ lat(:,3);
c1_ lat=as_ lat(1,:);
[ k_ lat,s_ lat,e_ lat] =lqr(as_ lat,b2_ lat,cs_ lat* q_ lat* cs_ lat,r_ lat);
damp(e_ lat)
k_ lat=2* k_ lat;
kp_ lat=k_ lat(:,2:6);
ki_ lat=k_ lat(:,1);
pause
%compute control bode diagrams
bode(as_ lat-b2_ lat* k_ lat,b2_ lat,k_ lat,0)
title('aileron cmd loop')
pause
%compute command bode diagrams and step responses.
step(as_ lat-b2_ lat* k_ lat,b1_ lat,c1_ lat,0)
title('step heading response')
pause
bode(as_ lat-b2_ lat* k_ lat,b1_ lat,c1_ lat,0)
title('heading cmd loop')
pause

```

## B. OPTIMAL GAINS

The *.m* file shown in the previous section calculated the optimal gains under the assumption of zero coupling between the longitudinal and lateral states. The results included a  $2 \times 7$  matrix for the longitudinal gains and a  $1 \times 7$  vector for the lateral gains. These gains were then split into proportional and integral error parts for use in the

controller gain blocks as detailed in Figure 5.1. The resulting matrices are shown below.

$$ki\_long =$$

$$\begin{bmatrix} 0.0051 & 0.0073 \\ -0.0107 & 0.0380 \end{bmatrix}$$

$$ki\_lat =$$

$$0.2689$$

$$kp\_long =$$

$$\begin{bmatrix} 0.0037 & 0.0095 & -0.1632 & -1.4319 & 0.0131 \\ 0.1223 & -0.0032 & -0.0332 & 0.0368 & -0.0494 \end{bmatrix}$$

$$kp\_lat =$$

$$0.0077 \quad 0.0861 \quad 0.2293 \quad 0.5062 \quad 1.1072$$

### C. CLOSED-LOOP FREQUENCY AND STEP RESPONSES



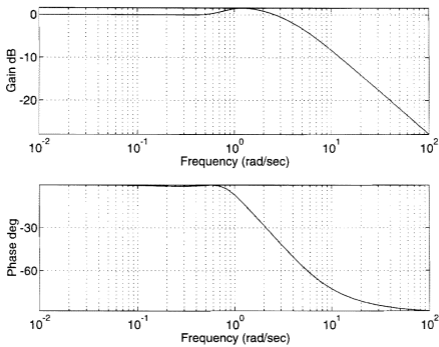


Figure C.1: Elevator Control Loop

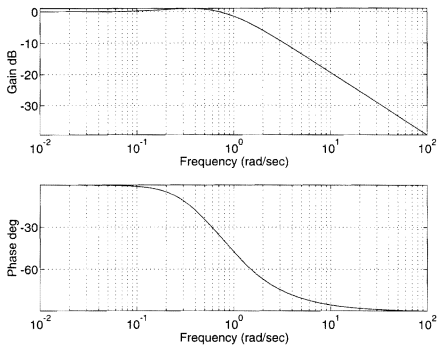


Figure C.2: Throttle Control Loop

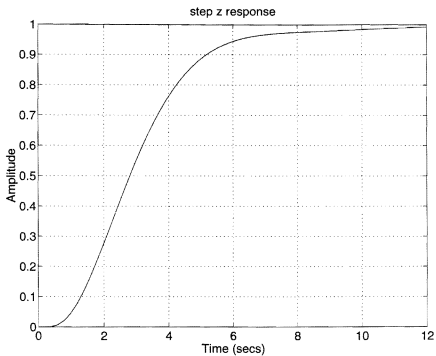
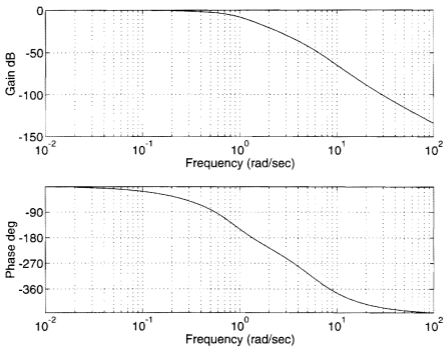


Figure C.3: Step Altitude Command



**Figure C.4: Altitude Command Bode Diagram**

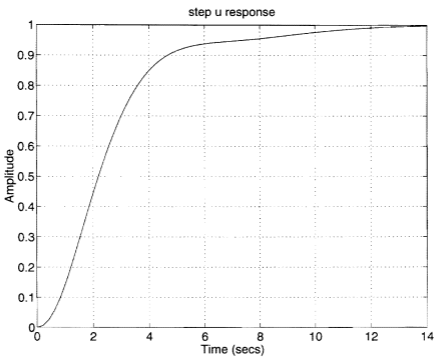


Figure C.5: Step Airspeed Command

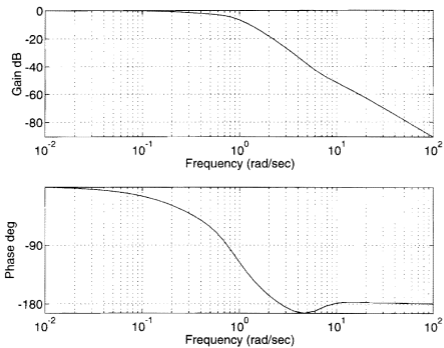


Figure C.6: Airspeed Command Bode Diagram

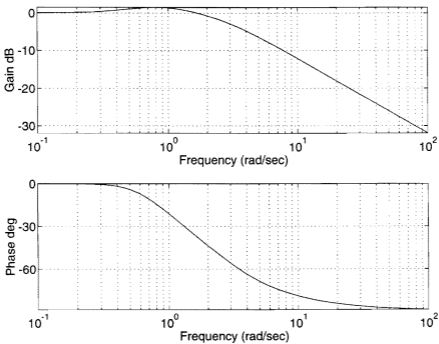


Figure C.7: Aileron Control Loop

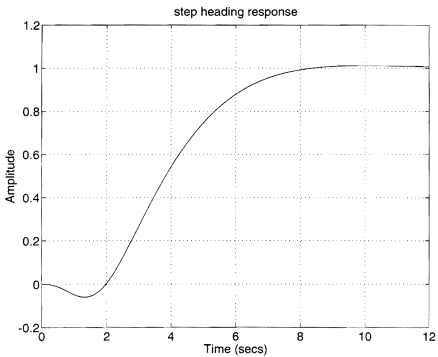


Figure C.8: Step Heading Command



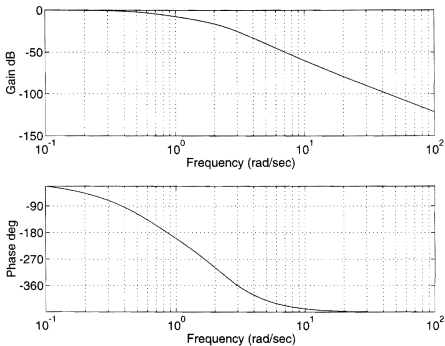


Figure C.9: Heading Command Bode Diagram

**APPENDIX D**  
**CLOSED-LOOP PERFORMANCE OF**  
**DISCRETE CONTROLLER ON THE**  
**NONLINEAR PLANT**

The following plots chronicle the time history of the closed-loop system response to the commanded inputs and airmass disturbances detailed in Chapter V, Section E.

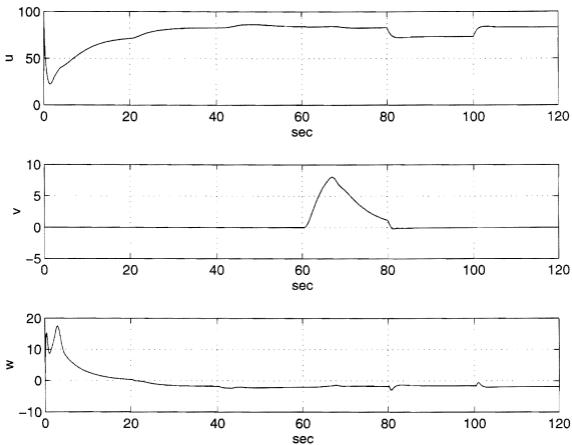


Figure D.1: Inertial Velocity Time History

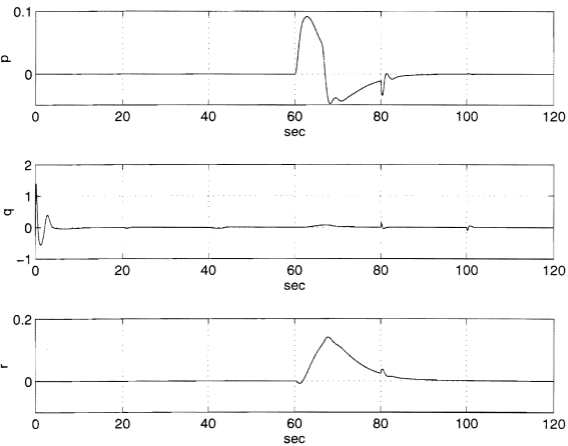


Figure D.2: Inertial Rate Time History

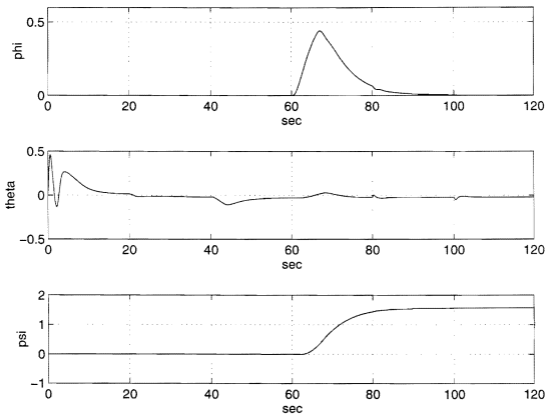


Figure D.3: Euler Angle Time History

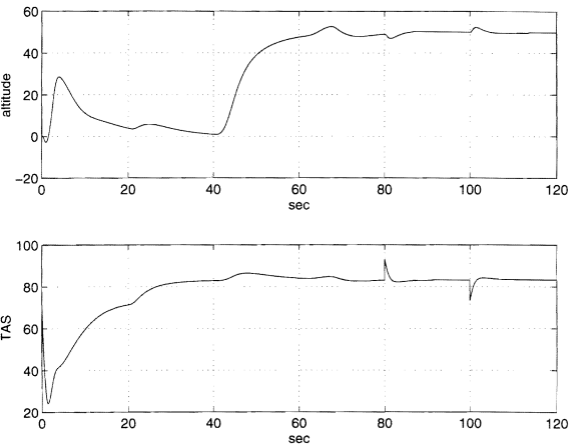


Figure D.4: Altitude and TAS Time History

## LIST OF REFERENCES

- [1] Sivashankar, N., "Design, Analysis, and Hardware-In-The-Loop Testing of a Controller for the Unmanned Aerial Vehicle ARCHYTAS," work done while a visiting scholar, Department of Aeronautics, Naval Postgraduate School, Monterey, CA. August 1993.
- [2] Moats, Michael L., "Automation of Hardware-In-The-Loop Testing and Implementation of Controllers for Unmanned Air Vehicles," Master's Thesis, Department of Aeronautics, Naval Postgraduate School, Monterey, CA. September 1994.
- [3] Kuechenmeister, David R., "A Non-Linear Simulation For An Autonomous Unmanned Aerial Vehicle," Master's Thesis, Department Of Aeronautics, Naval Postgraduate School, Monterey, CA, September 1993.
- [4] Halberg, Eric N., "Design of a GPS Aided Guidance, Navigation, and Control System for Trajectory Control of an Air Vehicle," Master's Thesis, Department of Aeronautics, Naval Postgraduate School, Monterey, CA, March 1994.
- [5] Byerly, James W., "Development of Equations-of-Motion in MATRIX<sub>X</sub> and SystemBuild" Directed Study Report, Department of Aeronautics, Naval Postgraduate School, Monterey, CA, March 1994.
- [6] Roskam, J., *Airplane Flight Dynamics and Automatic Flight Controls*, Roskam Aviation and Engineering corp, Ottawa, KS, 1979
- [7] Howard, R., *Class Notes for AE3340*, U.S. Naval Postgraduate School, Monterey, CA. 1993.

- [8] Kaminer, I. I., *Class Notes for AE4276*, U.S. Naval Postgraduate School, Monterey, CA. 1994
- [9] Kaminer, I. I., *Class Notes for AE4341*, U.S. Naval Postgraduate School, Monterey, CA. 1994
- [10] Ogata, Katsuhiko, *Modern Control Engineering*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.
- [11] Anderson, B.O., *Optimal Control: Linear Quadratic Methods*. Prentice Hall, Englewood Cliffs. New Jersey, 1990.



## INITIAL DISTRIBUTION LIST

	No. of Copies
1. Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3. Dr. Isaac I. Kaminer Department of Aeronautics and Astronautics, Code AA/KA Naval Postgraduate School Monterey, CA 93943-5000	5
4. Dr. Richard M. Howard Department of Aeronautics and Astronautics, Code AA/Ho Naval Postgraduate School Monterey, CA 93943-5000	1
5. Chairman Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, CA 93943-5000	2
6. John J Foley Jr 3 Walnut Grove Dr Fredricksburg, VA 22406	1
7. Brian T Foley 3 Walnut Grove Dr Fredricksburg, VA 22406	1

DUDLEY KNOX LIBRARY  
NAVAL POSTGRADUATE SCHOOL  
MONTEREY CA 93943-5101



GAYLORD 1

DUDLEY KNOX LIBRARY



3 2768 00304325 8