Subhabrata Ganguli

Control System Design with an Industrial Perspective:

With MATLAB Examples

Table of Contents

[Acknowledgement 1](#_Toc164947306)

[Introduction 2](#_Toc164947307)

[1.0 PID Control Implementation 5](#_Toc164947308)

[1.1 Overview 5](#_Toc164947309)

[1.2 Implementation of PI Controller 6](#_Toc164947310)

[1.2.1 Setpoint Weights 8](#_Toc164947311)

[1.2.2 Implementation of the integral term 9](#_Toc164947312)

[1.2.3 Anti-windup mechanism 1: Limiting the min/max value of the integrator state 12](#_Toc164947313)

[1.2.4 Anti-windup mechanism 2: Disabling PI control integrator during plant input saturation 13](#_Toc164947314)

[1.2.5 Anti-windup mechanism 3: Implementing of anti-windup filter 14](#_Toc164947315)

[1.3 Usage of derivative term for a PID controller 15](#_Toc164947316)

[1.4 Position and Velocity Control Using Acceleration Input 16](#_Toc164947317)

[2 F-16 Model and Simulations 20](#_Toc164947318)

[2.1 Equations of Motion 20](#_Toc164947319)

[2.2 MATLAB/Simulink Model – Trim and Linearization 22](#_Toc164947320)

[2.3 Pitch Rate Control for Longitudinal Dynamics 30](#_Toc164947321)

[2.4 Roll, Pitch and Yaw Rate Control 33](#_Toc164947322)

[Appendix A 39](#_Toc164947323)

[Appendix B 40](#_Toc164947324)

[References 62](#_Toc164947325)

# Acknowledgement

Some text

# Introduction

Some text

Part 1:

Control Design and Analysis

In this part of the book, we will concentrate on Linear Time Invariant (LTI) plant which does not have any modeling uncertainty. Our goal is to gain insight into some of the key aspect of control design in a simplistic setup.

We will look into a simple but fundamental plant model called a *double integrator*. This type of plant appears in various industrial problems. One of them is related to velocity and position control of Adaptive Cruise Control (ACC) system in automotive vehicles. In ACC, velocity control is required to maintain a set speed during normal operations. In traffic, the vehicle may not be able to operate at the set speed and may need to drive at a lower speed by maintaining a fixed distance from the lead traffic vehicle.

We will use various control design techniques to obtain position and velocity tracking. We will start with a classical control method called Proportional-Integral-Derivative (PID) control. We will also show how a Lead Compensation can be used as an alternate technique. Finally, we will delve into various control analysis techniques to check the stability and tracking performance of the closed loop system.

# PID Control Implementation

## Overview

PID control, which stands for Proportional-Integral-Derivative control, is a widely used feedback control mechanism in engineering and industrial applications.

PID Control is a type of closed-loop control system that continuously calculates an error value as the difference between a desired setpoint and a measured process variable. The PID controller then adjusts the control input to the system based on three main components: proportional, integral, and derivative terms.

Here is a brief overview of the different components of the PID controller:

**1. Proportional (P) Term**:

* The proportional term is directly proportional to the current error. It contributes to the control output in proportion to the magnitude of the error.
* The goal of the proportional term is to reduce the steady-state error, which is the difference between the desired setpoint and the actual process variable when the system has stabilized.

**2. Integral (I) Term**:

* The integral term is proportional to both the magnitude and the duration of the error. It accumulates the error over time.
* The integral term helps eliminate any residual steady-state error that may be present after the proportional control has brought the system close to the setpoint.

3. **Derivative (D) Term:**

* The derivative term is proportional to the rate of change of the error. It anticipates future behavior based on the current rate of change of the error.
* The derivative term helps to dampen the system's response, preventing overshooting and oscillations.

The overall control output (u) of the PID controller is calculated as the sum of the three terms:

,

where *u*(*t*) and *e*(*t*) are the control and error signals, and *KP*, *KI* and *K­D* are the PID gains.

## Implementation of PI Controller

Note that although all the above terms on the right-hand side of the equation uses the error term *e*(*t*) as the feedback signal, it is not an ideal approach to implement the controller as such. Using the error term in the closed loop system adds a zero in the transfer function between the controlled signal and the reference signal which can cause an unnecessary fast response in the beginning and can even cause an overshoot. Hence, an idea called *setpoint weights* is typically used in the actual implementation.

Let us try understanding the problem of implementation of a PI controller with an illustrative example. Consider a first order plant:

A diagram of a block diagram

Description automatically generated

Figure 1: Closed loop system with PI controller.

The closed loop system with the PI controller is shown in Figure 1. The closed loop transfer function is given by:

Thus, we see that the closed loop transfer function has a zero at . If we want the closed loop system to behave like a second order system with a corner frequency of and damping ratio of , then we can write:

Equating the coefficients of *s* in the denominator, we have:

|  |  |
| --- | --- |
|  | (1) |
|  | (2) |

Let us consider a specific example with rad/sec and . Using equations (1) and (2), the PI gains can be calculated as *KP* = 0.35 and *KI* = 0.8. The implementation in Simulink is shown in Figure 2. If we look at the step response in Figure 3, we see that it has an overshoot before the output settles down to one. This is due the presence of a zero of rad/sec rad/sec in the transfer function of the closed loop system.

A diagram of a computer

Description automatically generated

Figure 2: PI Controller with error term as feedback for both K­P and K­­I.

A graph with a line

Description automatically generated

Figure 3: Step response of closed loop for PI controller with error signal feedback for both K­P and K­­I. (Time in x-axis is in sec).

### Setpoint Weights

A better way to implement a PID controller is to weight the set-point/reference signal. We scale the proportional term feedback as and derivative term feedback as where *r*(*t*) and *y*(*t*) are the reference and the feedback signals respectively and and are positive scalars. These scalars terms are designer’s choices. The Simulink implementation for our PI controller example is shown in Figure 4. For example, if we set in our PI controller example, we get no overshoot as shown in Figure 5.

A diagram of a machine

Description automatically generated

Figure 4: PI controller with set point weight for KP.

A graph with a line

Description automatically generated

Figure 5: Step response of closed loop for PI controller with set point weight for KP. (Time in x-axis is in sec).

### Implementation of the integral term

The integral term in a PID controller is rarely implemented as is. This is due to the integrator windup phenomenon. Integrator windup occurs when the error term keeps accumulating over time and the system is unable to implement the control signal due to physical limitation or saturation. This can lead to a delayed response or an overshoot from the setpoint.

Consider the PI control problem discussed above, but now we add a saturation block before the plant as shown in Figure 6. We set the saturation limits of . (We also change the proportional gain setpoint to 1 for reason explained below). If we input a step of 1 at 1 sec followed by another step of -1 at 5 sec, we observe that the output responds to the change in second step at 8.2 sec instead of 5 sec, as shown in Figure 7 (a). This is due to integrator windup. The control signal before and after the saturation block is shown in Figure 7 (b). We observe that the control signal before the saturation block continuously builds up until 5 sec. After the second step input at 5 sec, the control signal starts decreasing and matches the signal after the saturation block only after 8.2 sec.

A diagram of a machine

Description automatically generated

Figure 6: Closed loop system with plant input saturation.

A graph with a line

Description automatically generated

(a)

A graph with a blue line

Description automatically generated

(b)

Figure 7: (a) Step response for plant with input saturation (b) Control signal before and after plant input saturation block.

There are several strategies which can be implemented to prevent integrator windup. These are listed below:

* 1. Limiting the min/max value of the integrator state.
  2. Disabling PI control integrator during plant input saturation.
  3. Implementing an anti-windup filter.

### Anti-windup mechanism 1: Limiting the min/max value of the integrator state

One of the strategies to avoid the integrator windup is to add a limit to the min/max value of the output of the integrator. The Simulink implementation is shown in Figure 8.

In this example we limit the output of the integrator to be . With this we see that the output signal immediately responds to the change in the step input at 5 sec as shown in Figure 9 (a). The control signal is shown in Figure 9 (b). However, the steady-state does not reach of 1. This is because of the input saturation of the plant.

A diagram of a machine

Description automatically generated

Figure 8: Closed loop system with saturation limit on input to KI. The integral gain limiting is shown by the shaded area.

|  |  |
| --- | --- |
| (a) | (b) |

Figure 9: (a) Step response for plant with integral limiting (b) Control signal showing input saturation.

### Anti-windup mechanism 2: Disabling PI control integrator during plant input saturation

The second strategy is to disable the integrator when the difference between the input and output of the plant saturation block is greater than a threshold. This will help in stopping the buildup of the integrator signal when the error is high and will prevent windup.

The implementation of this mechanism is shown in Figure 10. Here the threshold is chosen as 0.01. The integrator for the PI controller is modified into an enabled system. This block is enabled only when the saturation error is lower than the threshold.

The plant and the controller responses are shown in Figure 11. Note that the plant response for the second step changes almost immediately. We observe an oscillation in the control signal. This is because of a high frequency switching of the enabling signal. This occurs because when the integrator is enabled, the control signal increases and creates an error which when greater than the threshold causes the integrator path of the PI controller to disable. Hence, this scheme as implemented may not be ideal for anti-windup.

A diagram of a machine

Description automatically generated

Figure 10: Closed loop system with mechanism for disabling the integrator gain during plant input saturation.

|  |  |
| --- | --- |
| (a) | (b) |

Figure 11: (a) Step response of the plant with anti-windup scheme involving disabling of the integrator (b) Corresponding control signal.

### Anti-windup mechanism 3: Implementing of anti-windup filter

A third strategy is to use an anti-windup filter. In this case, the difference between the input and output of the plant input saturation block is subtracted from the input to the integrator with an anti-windup gain.

The implementation in Simulink is shown in Figure 10. The anti-windup gain is chosen as 3. The step response of the plant and the corresponding control signal are show in Figure 11. Note that the plant responds immediately to the second step command. The steady-state response of the plant does not reach 1 due to the input saturation.

A diagram of a machine

Description automatically generated

Figure 12: Anti-windup filter.

|  |  |
| --- | --- |
| (a) | (b) |

Figure 13: (a) Step response for the plant with anti-windup filter. (b) Control signal showing input saturation.

## Implementation of a PID controller

## Implementation of the derivative term.

The derivative term of a PID controller is a non-causal when implemented as a pure derivative. In control theory, a non-causal system refers to a system whose output depends on future input values, which is not physically realizable. In practical implementations, it's not possible to directly measure the future values of the error signal, making a pure derivative term non-causal. To address this issue, practical implementations of derivative action often use an approximation of the derivative term that depends only on current and past values of the error signal. Common approaches include the backward difference method, central difference method, or a low-pass filtered derivative. These methods make the derivative term causal and prevent it from causing instability or oscillations in the control system.

For continuous-time systems, the derivative term is implemented as a low-pass derivative filter. This can be a system such as , where is chosen as a small positive value. An implementation of the derivative term in a PID controller as a Simulink model is shown in Figure 14.

A diagram of a machine

Description automatically generated

Figure 14: Simulink implementation of the derivative term in a PID controller.

## Usage of derivative term for a PID controller

In general, the derivative term of a PID controller is used to improve the closed loop damping. One way to reduce oscillations in a low damped closed loop system is to reduce the closed loop bandwidth which is equivalent to reducing the controlled gains. But this may be ideal for the closed-loop design. A derivative term can help in such cases.

A closed loop implementation of a PI and a PID controller is shown in Figure 14. Here the plant dynamic is a low damped second order system given by . The gains for the PI system are KP = 0.1 and KI = 0.2. The gains for the PID controllers are KP = 4, KI=1 and KD = 1. Thus, we see that the gains of the PID controller are higher than the PI controller which indicates than the closed loop system with the PID controller has a higher bandwidth.

The step responses of both closed loop systems are shown in Figure 15. We observe that the oscillations are reduced in the PID controller. Also, the step response rise time is significantly higher for the PID controller. This indicates than we can design a higher bandwidth closed loop system with less damping if we decide to implement a PID controller.

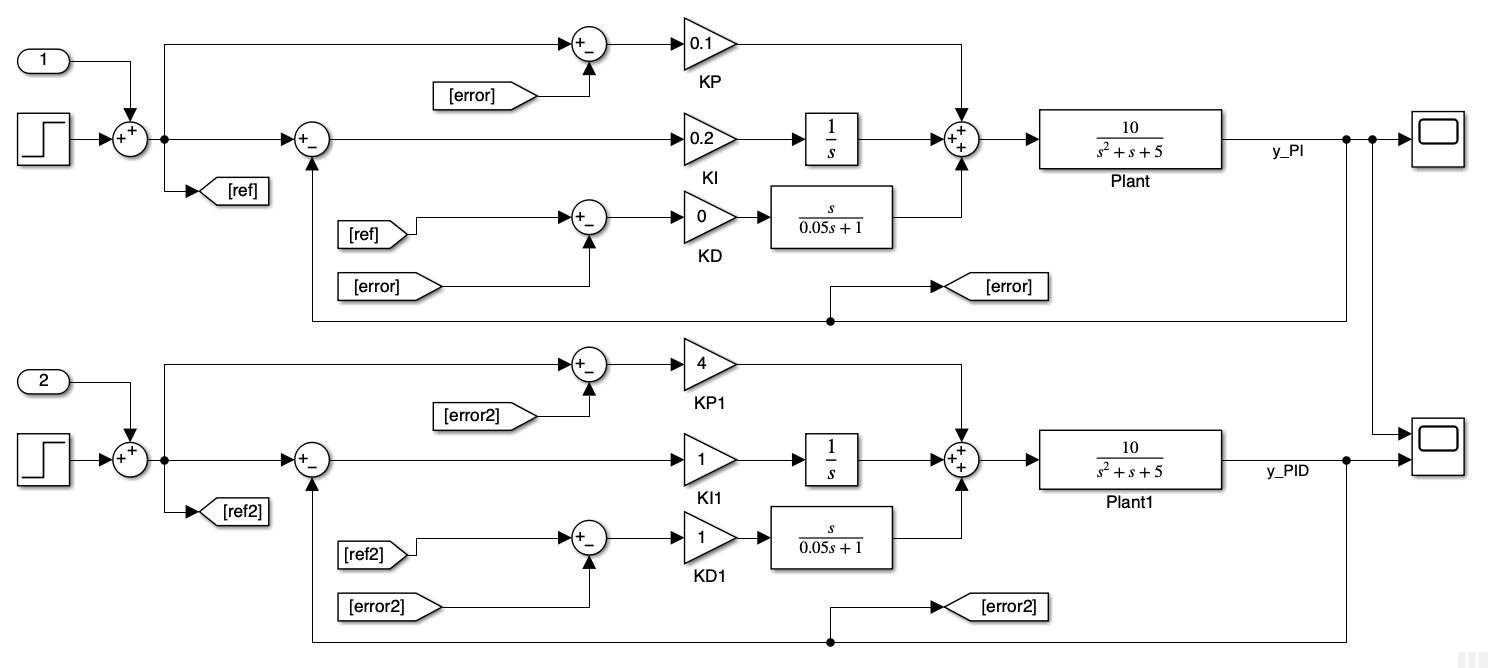


Figure 15: Simulink implementation of PI and PID controllers.

A graph with lines and numbers

Description automatically generated

Figure 16: Step responses of closed loop system with PID controller and comparison with that of a PI controller.

## Position and Velocity Control Using Acceleration Input

Let us look at a classic problem of position control using acceleration input. The dynamics of a position control system using acceleration can be formulated as:

where *x* is the position and *u* is the control input. We will consider two cases (a) where both position and velocity can be measured, and (b) where we only have position measurement available.

Using Laplace transform, the above equation can be written as

In other words, there are two *pure integrators* in the plant. We call this the *double integrator* problem.

The control solution to the double integration is not straight forward.

Part 2:

Control Applications

F-16 Aircraft Model

# F-16 Model and Simulations

The F-16 aircraft is a multirole fighter jet that has been widely used by various air forces around the world. Developed by General Dynamics (now Lockheed Martin), the F-16 is known for its agility, versatility, and advanced avionics. The F-16 was designed in the 1970s as a lightweight, cost-effective fighter aircraft with a focus on air-to-air combat. It is renowned for its exceptional maneuverability, often attributed to its relaxed static stability and fly-by-wire control system. It has a top speed of over Mach 2 (twice the speed of sound) and can climb at a rate of 50,000 feet per minute.



Figure 17: F-16 Aircraft

This section presents a MATLAB/Simulink F-16 aircraft model as developed by (Stevens et al., 2003). First, the aircraft equations of motion are introduced. Next, a Simulink model is developed for the F-16 aircraft. This is followed by simulations of the Simulink model of the F-16 aircraft.

## Equations of Motion

Let us begin with the equations of motion (EOM) of a 6 degrees of freedom (DOF) of an aircraft. We assume that the reader is somewhat familiar with aircraft literature – so these are not derived here. The following notations are used in this text:

*V*: aircraft velocity

: angle of attack

: side slip angle

: flight path angle

body axis *x* velocity

body axis *y* velocity

body axis *z* velocity

: roll angle

: pitch angle

: yaw angle

*p*: roll rate

*q*: pitch rate

*r*: yaw rate

: inertial *x* displacement

: inertial *y* displacement

: inertial *z* displacement

*h*: altitude

*g*: acceleration due to gravity

*m*: aircraft mass

*J*:aircraft inertia (subscripts *x*, *y*, *z* refers to the body axis coordinates)

*F*: aircraft force (subscripts *x*, *y*, *z* refers to the body axis coordinates)

*M*: aircraft moment (subscripts *x*, *y*, *z* refers to the body axis coordinates)

Given the above notations, the aircraft 6 DOF equations of motion are:

|  |  |
| --- | --- |
| Force Equations:  Kinematic Equations:  Moment Equations:  Navigation Equations: | (3) |

The body axis velocities can be converted to aircraft velocity, angle of attack and side slip angle using the following equations:

|  |  |
| --- | --- |
|  | (4) |

The flight path angle is given by:

|  |  |
| --- | --- |
|  | (5) |

## MATLAB/Simulink Model – Trim and Linearization

The code for the F-16 aircraft dynamics is presented in Appendix B. The code given in (Stevens et al, 2003) is written in Fortran and these are converted into MATLAB in this book.

In order to simulate the model at an equilibrium point, we first need to trim the aircraft. This is done by reducing a cost function as given by the code in F16COST.m. For straight and level flight, the trim condition is specified by the equilibrium velocity , altitude and cg location the first 9 states from the state vector *x* are passed to F16COST.m from DOTRIM.m. These are Inside F16COST.m, all of the 13 states are initialized as zero, except for velocity (state 1) , altitude (state 12) and power (state 13). The power is given by TGEAR(throttle). The control vector (throttle, elevator, aileron and rudder) is initialed as [0 0 0 0]*T.* The input to F16COST.m is .

The cost is given by the quadratic term where *Q* is a constraint term and is a vector given the following vector:

Here, is the flight path angle which is zero for straight and level flight. The term *H* is set as 1000 x *I*13. When the cost is minimized, the term *Q* tends to zero.

The Simulink model F16sim\_lin.slx shown in Figure 10 is used to trim the aircraft. The corresponding code is given in DOTRIM.m in Appendix B. The MATLAB function for the aircraft dynamics is given in F16DYN.m

For the trim condition ft/sec, altitude ft and cg location , the cost term is 1.2319 x 10-4. The trim condition *z*\* is calculated as:

Where the unit for z are [ft/sec rad rad rad rad rad rad/sec rad/sec rad/sec % rad rad rad]. Once the aircraft trim values are calculated, we can linearize the aircraft model. The Simulink model for linearization is shown in Figure 12. After linearization, the state space matrices are separated for the longitudinal and lateral dynamics.



Figure 18: Simulink model for trimming the aircraft model.

|  |  |
| --- | --- |
| (a) | (b) |
| (c) | (d) |

Figure 19: Actuator models – (a) Throttle (b) Elevator (c) Aileron (d) Rudder. The rate and control saturation values are given in Appendix A.

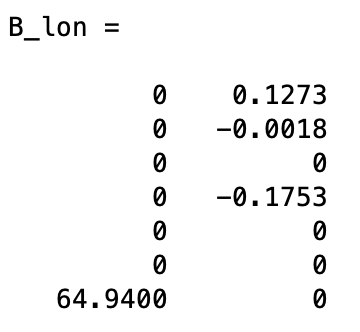


Figure 20: Simulink model for linearizing the aircraft model.

The *A* and *B* matrices for the longitudinal dynamics is given by:

A number of numbers and digits

Description automatically generated with medium confidence



The state vector for the longitudinal dynamics is (where *pow* is power) and the control vector is [.

The *A* and *B* matrices for the lateral dynamics is given by:

A number of numbers and a line of numbers

Description automatically generated with medium confidence

A close-up of numbers

Description automatically generated

The state vector for the longitudinal dynamics is (where *p* is power) and the control vector is [.

After linearization, we simulate the models with step inputs in . The corresponding plots are shown in Figure 13 to Figure 20. We observe that except for the states which are directly affected by the inputs (*V*, *pow* for ; for ; , *p* for ; and *r* for ), the deviations for the rest of the states are minimal. This is expected for the trim condition simulations.



Figure 21: V and for linearized model with 10% step input in throttle.



Figure 22: , q, and pow for linearized model with 10% step input in throttle.



Figure 23: V and for linearized model with 1 deg step input in elevator..



Figure 24: , q, and pow for linearized model with 1 deg step input in elevator.



Figure 25: and for linearized model with 1 deg step input in aileron..



Figure 26: , p and r for linearized model with 1 deg step input in aileron..



Figure 27: and for linearized model with 1 deg step input in rudder.

.

Figure 28: , p and r for linearized model with 1 deg step input in rudder..

## Pitch Rate Control for Longitudinal Dynamics

In this section, we present a pitch rate controller for the F16 plant using dynamic inversion. The aircraft is straight and level and is linearized at ft/sec, altitude ft and cg location . We will simulate the linear and nonlinear controllers and compare the step responses.

The Simulink diagram in Figure 21 shows the closed loop system with the linearized plant. The plant is modeled using the longitudinal state space matrices mentioned earlier. Please refer to the chapter TBD to understand the controller structure. The controller gains are selected as *Kq = 4*, *fc =* 0.5, and *fi* = 0.25. The *C* matrix is [0, 0, 0, 1, 0, 0 0] which corresponds to pitch rate feedback.

The Simulink diagram in shows the closed loop system Figure 22 with the nonlinear plant. (The dynamics in modeled in F16DYN.m). The key difference in the setup is that the trim values of the control signals is added to the controller output and the trim values of the plant states is subtracted.

The pitch rate and elevator responses for the linear and nonlinear closed-loop systems are plotted in Figure 23. It is observed that the responses match very closely.



(a)

****

(b)



(c)

Figure 29: Pitch rate controller for linear plant: (a) Closed loop system (b) Plant (c) Controller

A diagram of a computer

Description automatically generated

(a)

A diagram of a machine

Description automatically generated

(b)

A diagram of a graph

Description automatically generated

(c)

A diagram of a machine

Description automatically generated

(d)

Figure 30: Pitch rate controller for linear plant: (a) Closed loop system (b) Plant (c) Throttle (d) Controller



Figure 31: Pitch rate and elevator responses for linear (solid) and nonlinear (dashed) closed-loop systems. The responses match very closely.

## Roll, Pitch and Yaw Rate Control

In this section, we present the aircraft rotation rate control for all the three axis. The roll rate is controlled by the aileron, the pitch rate is controlled by the elevator, while the yaw rate is controlled by the rudder.

As we know from before, the goal of the dynamic inversion controller is to invert the plant dynamics and replace it with the desired dynamics. However, for the nonlinear F16 aircraft model, the dynamics is not invertible. We can observe this from the function PDOT.m which is shown below:

function PDOT\_OUT = PDOT(P3,P1)

% PDOT\_OUT = rate of change of power

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

% P3 = actual power

% P1 = power command

if (P1 >= 50)

if (P3 >= 50)

T = 5;

P2 = P1;

else

P2 = 60;

T = RTAU(P2-P3);

end

else

if (P3 >= 50)

T = 5;

P2 = 40;

else

P2 = P1;

T = RTAU(P2-P3);

end

end

PDOT\_OUT = T\*(P2-P3);

end

Note that for P150 and P350, P2 is always equal to 60. This means that we will get the same PDOT\_OUT for all values of P150 and P350 since PDOT = RTAU(P2-P3). Thus, if we want to invert the function to get P1 from PDOT\_OUT and P3, we will not get a unique value. In other words, the function is not invertible. A similar argument holds for the function RTAU.m.

Hence, we make a slight alteration to the function to get invertibility. We add a perturbation term to PDOTINV\_K.m and RTAU\_K.m. In PDOTINV\_K.m we add the term “0.1\*P1” as shown below:

function [P1\_OUT] = PDOTINV\_K(P3,PDOT\_IN)

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

% P3 = actual power

% P1 = power command

% PDOT\_IN = power rate

if isempty(P3)

disp('empty')

end

ctr = 1;

n = length(0:0.01:100);

P1\_OUT\_vec = inf\*ones(n,1);

err = inf\*ones(n,1);

for P1 = 0:0.01:100

if (P1 >= 50)

if (P3 >= 50)

T = 5;

P2 = P1;

else

P2 = 60 + 0.1\*P1;

T = RTAU\_K(P2-P3);

end

else

if (P3 >= 50)

T = 5;

P2 = 40 + 0.1\*P1;

else

P2 = P1;

T = RTAU\_K(P2-P3);

end

end

PDOT\_OUT = T\*(P2-P3);

P1\_OUT\_vec(ctr) = P1;

err(ctr) = abs(PDOT\_OUT - PDOT\_IN);

ctr = ctr + 1;

end

[~,idx] = min(err);

P1\_OUT = P1\_OUT\_vec(idx);

end

For RTAU\_K.m, we add a similar perturbation term “0.01\*DP” as shown below to make it invertible.

function RTAU\_OUT = RTAU\_K(DP)

% function used by PDOT

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

if (DP < 25)

RTAU\_OUT = 1 + 0.01\*DP;

elseif (DP > 50)

RTAU\_OUT = 0.1 + 0.01\*DP;

else

RTAU\_OUT = 1.9 -.036 \* DP;

end

Note that the functions PDOTINV\_K.m and RTAU\_K.m are only called by the dynamic inversion function F16DYNINV.m in the controller. Hence the suffix ‘\_K’ is added. The plant model represented by the actual aircraft dynamics is not affected with these changes.

The Simulink model for the roll-pitch-yaw rate control is shown in Figure 24. In Figure 25, we plot the roll-pitch-yaw rate responses for doublet commands. In each row, we plot the responses for each individual axis which are simulated separately. On the left-hand side, the rate responses are plotted. On the right-hand side, the corresponding control responses are plotted. The doublet responses for the rate responses match well with the desired dynamics.

A diagram of a computer

Description automatically generated

(a)

A diagram of a computer

Description automatically generated

(b)

A diagram of a computer system

Description automatically generated

(c)

Figure 32: (a) F16 closed-loop for roll-pitch-yaw rate control (b) F16 controller (c) F16 plant.



Figure 33: Roll-pitch-yaw rate responses for doublet commands.

# Appendix A

The F-16 aircraft modeling data is presented in this appendix. These are obtained from (Stevens et. al., 2003). The code in Appendix B uses these data.

**Mass Properties**

|  |  |
| --- | --- |
| Weight (lb) |  |
| Moment of Inertia (slugs-ft2) |  |

**Wing Dimensions**

|  |  |
| --- | --- |
| Span (ft) |  |
| Area (ft2)  Mean aerodynamic chord () |  |

**Reference CG Location**

|  |  |
| --- | --- |
| *XCG* |  |
|  |  |

**Control Surface Actuator Models**

|  |  |  |  |
| --- | --- | --- | --- |
|  | deflection limit | rate limit | time constant |
| Elevator | deg |  | sec lag |
| Aileron | deg |  | sec lag |
| Rudder | deg |  | sec lag |

**Engine Angular Momentum**

Assumed fixed at 160 slug-ft2/sec

# Appendix B

function XD = F16DYN(X,U,XCG)

% f16 dynamics

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

%% Parameters

AXX = 9496.0;

AYY = 55814.0;

AZZ = 63100.0;

AXZ = 982.0;

AXZS = AXZ^2;

XPQ = AXZ \* (AXX - AYY + AZZ);

GAM = AXX \* AZZ - AXZ^2;

XQR = AZZ \* (AZZ - AYY) + AXZS;

ZPQ = (AXX-AYY) \* AXX + AXZS;

YPR = AZZ - AXX;

WEIGHT = 20500.0;

GD = 32.174;

MASS = WEIGHT/GD;

S = 300;

B = 30;

CBAR = 11.32;

XCGR = 0.35;

HX = 160;

RTOD = 180/pi;

%% Assign State and Control Variables

VT = X(1);

ALPHA = X(2)\*RTOD;

BETA = X(3)\*RTOD;

PHI = X(4);

THETA = X(5);

PSI = X(6);

P = X(7);

Q = X(8);

R = X(9);

ALT = X(12);

POW = X(13);

THTL = U(1);

EL = U(2);

AIL = U(3);

RDR = U(4);

XD = zeros(13,1);

%% Air data computer and engine model

[TFAC, T, RHO, AMACH, QBAR, PS ] = ADC(VT, ALT);

CPOW = TGEAR(THTL);

XD(13) = PDOT(POW,CPOW);

T = THRUST(POW,ALT,AMACH);

%% Look-up tables and component buildup

CXT = CX (ALPHA,EL);

CYT = CY (BETA,AIL,RDR);

CZT = CZ (ALPHA,BETA,EL);

DAIL= AIL/20.0; DRDR= RDR/30.0;

DLDA\_OUT = DLDA(ALPHA,BETA);

DLDR\_OUT = DLDR(ALPHA,BETA);

DNDA\_OUT = DNDA(ALPHA,BETA);

DNDR\_OUT = DNDR(ALPHA,BETA);

CLT = CL(ALPHA,BETA) + DLDA\_OUT\*DAIL + DLDR\_OUT\*DRDR;

CMT = CM(ALPHA,EL);

CNT = CN(ALPHA,BETA) + DNDA\_OUT\*DAIL ...

+ DNDR\_OUT\*DRDR;

%% Add damping derivatives

TVT = 0.5/VT;

B2V = B\*TVT;

CQ = CBAR\*Q\*TVT;

D = DAMP(ALPHA);

CXT = CXT + CQ \* D(1);

CYT = CYT + B2V \* ( D(2)\*R + D(3)\*P );

CZT = CZT + CQ \* D(4);

CLT = CLT + B2V \* ( D(5)\*R + D(6)\*P );

CMT = CMT + CQ \* D(7) + CZT \* (XCGR-XCG);

CNT = CNT + B2V\*(D(8)\*R + D(9)\*P) - CYT\*(XCGR-XCG) \* CBAR/B;

%% Get ready for state equations

CBTA = cos(X(3));

U = VT\*cos(X(2))\*CBTA;

V = VT \* sin(X(3));

W = VT\*sin(X(2))\*CBTA;

STH = sin(THETA);

CTH = cos(THETA);

SPH = sin(PHI);

CPH = cos(PHI);

SPSI = sin(PSI);

CPSI = cos(PSI);

QS = QBAR \* S;

QSB = QS \* B;

RMQS = QS/MASS;

GCTH = GD \* CTH;

QSPH = Q \* SPH;

AY = RMQS\*CYT;

AZ = RMQS \* CZT;

%% Force Equations

UDOT = R\*V - Q\*W - GD\*STH + (QS \* CXT + T)/MASS;

VDOT = P\*W - R\*U + GCTH \* SPH + AY;

WDOT = Q\*U - P\*V + GCTH \* CPH + AZ;

DUM = (U\*U + W\*W);

XD(1) = (U\*UDOT + V\*VDOT + W\*WDOT)/VT;

XD(2) = (U\*WDOT - W\*UDOT) / DUM;

XD(3) = (VT\*VDOT- V\*XD(1)) \* CBTA / DUM;

%% Kinematics

XD(4) = P + (STH/CTH)\*(QSPH + R\*CPH);

XD(5) = Q\*CPH - R\*SPH;

XD(6) = (QSPH + R\*CPH)/CTH;

%% Moments

ROLL = QSB\*CLT;

PITCH = QS \*CBAR\*CMT;

YAW = QSB\*CNT;

PQ = P\*Q;

QR = P\*R;

QHX = Q\*HX;

XD(7) = ( XPQ\*PQ - XQR\*QR + AZZ\*ROLL + AXZ\*(YAW + QHX) )/GAM;

XD(8) = ( YPR\*P\*R - AXZ\*(P^2 - R^2) + PITCH - R\*HX )/AYY;

XD(9) = ( ZPQ\*PQ - XPQ\*QR + AXZ\*ROLL + AXX\*(YAW + QHX) )/GAM;

%% Navigation

T1 = SPH \* CPSI;

T2 = CPH \* STH;

T3 = SPH \* SPSI;

S1 = CTH \* CPSI;

S2 = CTH \* SPSI;

S3 = T1 \* STH - CPH \* SPSI;

S4 = T3 \* STH + CPH \* CPSI;

S5 = SPH \* CTH;

S6 = T2 \* CPSI + T3;

S7 = T2 \* SPSI - T1;

S8 = CPH \* CTH;

XD(10) = U \* S1 + V \* S3 + W \* S6; % North speed

XD(11) = U \* S2 + V \* S4 + W \* S7; % East speed

XD(12) = U \* STH -V \* S5 - W \* S8; % Vertical speed

end

function [TFAC, T, RHO, AMACH, QBAR, PS ] = ADC (VT, ALT)

% air data computer

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

% VT = true velocity (fps)

% ALT = altitude (ft)

% TFAC = temperature factor

% T = temperature (deg R), R = Rankine scale

% RHO = density (slug/ft^3)

% AMACH = Mach

% QBAR = dynamic pressure (psf)

% PS = static pressure (psf)

R0 = 2.37764e-3; %2.377E-3;

TFAC = 1.0 - 0.703E-5 \* ALT;

T = 519.0 \* TFAC;

if (ALT >= 35000.0)

T= 390.0;

end

RHO = R0 \* (TFAC^4.14);

AMACH = VT/sqrt(1.4 \* 1716.3 \* T);

QBAR = 0.5 \* RHO \* VT \* VT;

PS = 1715.0 \* RHO \* T;

end

function CL\_OUT = CL(ALPHA,BETA)

% rolling moment coeff.

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

A = [.0 .0 .0 .0 .0 .0 .0 .0 .0 .0 .0 .0 ;

-.001 -.004 -.008 -.012 -.016 -.022 -.022 -.021 -.015 -.008 -.013 -.015;

-.003 -.009 -.017 -.024 -.030 -.041 -.045 -.040 -.016 -.002 -.010 -.019;

-.001 -.010 -.020 -.030 -.039 -.054 -.057 -.054 -.023 -.006 -.014 -.027;

.000 -.010 -.022 -.034 -.047 -.060 -.069 -.067 -.033 -.036 -.035 -.035;

.007 -.010 -.023 -.034 -.049 -.063 -.081 -.079 -.060 -.058 -.062 -.059;

.009 -.011 -.023 -.037 -.050 -.068 -.089 -.088 -.091 -.076 -.077 -.076]';

S = 0.2 \* ALPHA;

K = fix(S);

if(K <= -2)

K= -1;

end

if(K >= 9)

K= 8;

end

DA = S - double(K);

L = K + fix( sign(DA) \* 1.1 );

S = .2\* abs(BETA);

M = fix(S);

if(M == 0)

M= 1;

end

if(M >= 6)

M= 5;

end

DB = S - double(M);

IDX\_SHIFT\_ALPHA = 3;

IDX\_SHIFT\_BETA = 1;

N = M + fix( sign(DB) \* 1.1 );

T = A(K + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA);

U = A(K + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA);

V = T + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA) - T);

W = U + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA) - U);

DUM = V + (W-V) \* abs(DB);

CL\_OUT = DUM \* sign(BETA) \* 1.0;

end

function CM\_OUT = CM(ALPHA,EL)

% pitching moment coeff.

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

A = [.205 .168 .186 .196 .213 .251 .245 .238 .252 .231 .198 .192;

.081 .077 .107 .110 .110 .141 .127 .119 .133 .108 .081 .093;

-.046 -.020 -.009 -.005 -.006 .010 .006 -.001 .014 .000 -.013 .032;

-.174 -.145 -.121 -.127 -.129 -.102 -.097 -.113 -.087 -.084 -.069 -.006;

-.259 -.202 -.184 -.193 -.199 -.150 -.160 -.167 -.104 -.076 -.041 -.005]';

S = 0.2 \* ALPHA;

K = fix(S);

if (K <= -2)

K = -1;

end

if(K >= 9)

K = 8;

end

DA = S - double(K);

L = K + fix( sign(DA) \* 1.1 );

S = EL/12.0;

M = fix(S);

if(M <= -2)

M = -1;

end

if(M >= 2)

M = 1;

end

DE = S - double(M);

N = M + fix( sign(DE) \* 1.1 );

IDX\_SHIFT\_ALPHA = 3; % Fortran to Matlab

IDX\_SHIFT\_EL = 3; % Fortran to Matlab

T = A(K + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_EL);

U = A(K + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_EL);

V = T + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_EL) - T);

W = U + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_EL) - U);

CM\_OUT = V + (W-V) \* abs(DE);

end

function CN\_OUT = CN(ALPHA,BETA)

% yawing moment coeff.

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

A = [.0 .0 .0 .0 .0 .0 .0 .0 .0 .0 .0 .0 ;

.018 .019 .018 .019 .019 .018 .013 .007 .004 -.014 -.017 -.033;

.038 .042 .042 .042 .043 .039 .030 .017 .004 -.035 -.047 -.057;

.056 .057 .059 .058 .058 .053 .032 .012 .002 -.046 -.071 -.073;

.064 .077 .076 .074 .073 .057 .029 .007 .012 -.034 -.065 -.041;

.074 .086 .093 .089 .080 .062 .049 .022 .028 -.012 -.002 -.013;

.079 .090 .106 .106 .096 .080 .068 .030 .064 .015 .011 -.001]';

S = 0.2 \* ALPHA;

K = fix(S);

if (K <= -2)

K = -1;

end

if (K >= 9)

K = 8;

end

DA = S - double(K);

L = K + fix( sign(DA) \* 1.1 );

S = .2\* abs(BETA);

M = fix(S);

if(M == 0)

M= 1;

end

if(M >= 6)

M= 5;

end

DB = S - double(M);

IDX\_SHIFT\_ALPHA = 3;

IDX\_SHIFT\_BETA = 1;

N = M + fix( sign(DB) \* 1.1 );

T = A(K + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA);

U = A(K + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA);

V = T + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA) - T);

W = U + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA) - U);

DUM = V + (W-V) \* abs(DB);

CN\_OUT = DUM \* sign(BETA) \* 1.0;

end

function CX\_OUT = CX(ALPHA,EL)

% x-axis aerodynamic force coeff.

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

A = [-0.099 -0.081 -0.081 -0.063 -0.025 0.044 0.097 0.113 0.145 0.167 0.174 0.166;

-0.048 -0.038 -0.040 -0.021 0.016 0.083 0.127 0.137 0.162 0.177 0.179 0.167;

-0.022 -0.020 -0.021 -0.004 0.032 0.094 0.128 0.130 0.154 0.161 0.155 0.138;

-0.040 -0.038 -0.039 -0.025 0.006 0.062 0.087 0.085 0.100 0.110 0.104 0.091;

-0.083 -0.073 -0.076 -0.072 -0.046 0.012 0.024 0.025 0.043 0.053 0.047 0.040]';

S = 0.2 \* ALPHA;

K = fix(S);

if (K <= -2)

K = -1;

end

if(K >= 9)

K = 8;

end

DA = S - double(K);

L = K + fix( sign(DA) \* 1.1 );

S = EL/12.0;

M = fix(S);

if(M <= -2)

M = -1;

end

if(M >= 2)

M = 1;

end

DE = S - double(M);

N = M + fix( sign(DE) \* 1.1 );

IDX\_SHIFT\_ALPHA = 3; % Fortran to Matlab

IDX\_SHIFT\_EL = 3; % Fortran to Matlab

T = A(K + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_EL);

U = A(K + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_EL);

V = T + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_EL) - T);

W = U + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_EL) - U);

CX\_OUT = V + (W-V) \* abs(DE);

end

function CY\_OUT = CY(BETA,AIL,RDR)

% sideforce coefficient

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

CY\_OUT = -.02\*BETA + .021\*(AIL/20.0) + .086\*(RDR/30.0);

end

function CZ\_OUT = CZ(ALPHA,BETA,EL)

% z-axis force coeff.

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

A = [0.770 0.241 -0.100 -0.416 -0.731 ...

-1.053 -1.366 -1.646 -1.917 -2.120 ...

-2.248 -2.229]';

IDX\_SHIFT = 3; % Fortran to Matlab

S = 0.2 \* ALPHA;

K = fix(S);

if (K <= -2)

K = -1;

end

if (K >= 9)

K = 8;

end

DA = S - double(K);

L = K + fix( sign(DA) \* 1.1 );

S = A(K + IDX\_SHIFT) + abs(DA) \* ...

(A(L + IDX\_SHIFT) - A(K + IDX\_SHIFT));

CZ\_OUT = S\*(1-(BETA/57.3)^2) - .19\*(EL/25.0);

end

function D = DAMP(ALPHA)

% various damping coefficients

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

% D1 = CXq;

% D2 = CYr;

% D3 = CYp;

% D4 = CZq;

% D5 = Clr;

% D6 = Clp

% D7 = Cmq;

% D8 = Cnr;

% D9 = Cnp;

A = [-0.267 -0.110 0.308 1.34 2.08 2.91 2.76 2.05 1.50 1.49 1.83 1.21;

0.882 0.852 0.876 0.958 0.962 0.974 0.819 0.483 0.590 1.21 -0.493 -1.04;

-0.108 -0.108 -0.188 0.110 0.258 0.226 0.344 0.362 0.611 0.529 0.298 -0.227;

-8.80 -25.8 -28.9 -31.4 -31.2 -30.7 -27.7 -28.2 -29.0 -29.8 -38.3 -35.3;

-0.126 -0.026 0.063 0.113 0.208 0.230 0.319 0.437 0.680 0.100 0.447 -0.330;

-0.360 -0.359 -0.443 -0.420 -0.383 -0.375 -0.329 -0.294 -0.230 -0.210 -0.120 -0.100;

-7.21 -5.40 -5.23 -5.26 -6.11 -6.64 -5.69 -6.00 -6.20 -6.40 -6.60 -6.00;

-0.380 -0.363 -0.378 -0.386 -0.370 -0.453 -0.550 -0.582 -0.595 -0.637 -1.02 -0.840;

0.061 0.052 0.052 -0.012 -0.013 -0.024 0.050 0.150 0.130 0.158 0.240 0.150]';

S = 0.2 \* ALPHA;

K = fix(S);

if(K <= -2)

K= -1;

end

if(K >= 9)

K= 8;

end

DA = S - double(K);

L = K + fix( sign(DA)\*1.1 );

IDX\_SHIFT\_ALPHA = 3;

D = zeros(9,1);

for I = 1:9

D(I) = A(K+IDX\_SHIFT\_ALPHA,I) + ...

abs(DA) \* (A(L+IDX\_SHIFT\_ALPHA,I) - A(K+IDX\_SHIFT\_ALPHA,I));

end

end

function DLDA\_OUT = DLDA(ALPHA,BETA)

% rolling mom. due to ailerons

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

A = [-.041 -.052 -.053 -.056 -.050 -.056 -.082 -.059 -.042 -.038 -.027 -.017;

-.041 -.053 -.053 -.053 -.050 -.051 -.066 -.043 -.038 -.027 -.023 -.016;

-.042 -.053 -.052 -.051 -.049 -.049 -.043 -.035 -.026 -.016 -.018 -.014;

-.040 -.052 -.051 -.052 -.048 -.048 -.042 -.037 -.031 -.026 -.017 -.012;

-.043 -.049 -.048 -.049 -.043 -.042 -.042 -.036 -.025 -.021 -.016 -.011;

-.044 -.048 -.048 -.047 -.042 -.041 -.020 -.028 -.013 -.014 -.011 -.010;

-.043 -.049 -.047 -.045 -.042 -.037 -.003 -.013 -.010 -.003 -.007 -.008]';

S = 0.2 \* ALPHA;

K = fix(S);

if (K <= -2)

K = -1;

end

if (K >= 9)

K = 8;

end

DA = S - double(K);

L = K + fix( sign(DA)\*1.1 );

S = 0.1 \* BETA;

M = fix(S);

if (M == -3)

M = -2;

end

if (M >= 3)

M = 2;

end

DB = S - double(M);

N = M + fix( sign(DB)\*1.11 );

IDX\_SHIFT\_ALPHA = 3;

IDX\_SHIFT\_BETA = 4;

T = A(K + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA);

U = A(K + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA);

V = T + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA) - T);

W = U + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA) - U);

DLDA\_OUT = V + (W-V) \* abs(DB);

end

function DLDR\_OUT = DLDR(ALPHA,BETA)

% rolling mom. due to rudder

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

A = [.005 .017 .014 .010 -.005 .009 .019 .005 .000 -.005 -.011 .008;

.007 .016 .014 .014 .013 .009 .012 .005 .000 .004 .009 .007;

.013 .013 .011 .012 .011 .009 .008 .005 .000 .005 .003 .005;

.018 .015 .015 .014 .014 .014 .014 .015 .013 .011 .006 .001;

.015 .014 .013 .013 .012 .011 .011 .010 .008 .008 .007 .003;

.021 .011 .010 .011 .010 .009 .008 .010 .006 .005 .000 .001;

.023 .010 .011 .011 .011 .010 .008 .010 .006 .014 .020 .000]';

S = 0.2 \* ALPHA;

K = fix(S);

if (K <= -2)

K = -1;

end

if (K >= 9)

K = 8;

end

DA = S - double(K);

L = K + fix( sign(DA)\*1.1 );

S = 0.1 \* BETA;

M = fix(S);

if (M == -3)

M = -2;

end

if (M >= 3)

M = 2;

end

DB = S - double(M);

N = M + fix( sign(DB)\*1.11 );

IDX\_SHIFT\_ALPHA = 3;

IDX\_SHIFT\_BETA = 4;

T = A(K + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA);

U = A(K + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA);

V = T + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA) - T);

W = U + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA) - U);

DLDR\_OUT = V + (W-V) \* abs(DB);

end

function DNDA\_OUT = DNDA(ALPHA,BETA)

% yawing mom. due to ailerons

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

A = [.001 -.027 -.017 -.013 -.012 -.016 .001 .017 .011 .017 .008 .016

.002 -.014 -.016 -.016 -.014 -.019 -.021 .002 .012 .016 .015 .011

-.006 -.008 -.006 -.006 -.005 -.008 -.005 .007 .004 .007 .006 .006

-.011 -.011 -.010 -.009 -.008 -.006 .000 .004 .007 .010 .004 .010

-.015 -.015 -.014 -.012 -.011 -.008 -.002 .002 .006 .012 .011 .011

-.024 -.010 -.004 -.002 -.001 .003 .014 .006 -.001 .004 .004 .006

-.022 .002 -.003 -.005 -.003 -.001 -.009 -.009 -.001 .003 -.002 .001]';

S = 0.2 \* ALPHA;

K = fix(S);

if (K <= -2)

K = -1;

end

if (K >= 9)

K = 8;

end

DA = S - double(K);

L = K + fix( sign(DA)\*1.1 );

S = 0.1 \* BETA;

M = fix(S);

if (M == -3)

M = -2;

end

if (M >= 3)

M = 2;

end

DB = S - double(M);

N = M + fix( sign(DB)\*1.11 );

IDX\_SHIFT\_ALPHA = 3;

IDX\_SHIFT\_BETA = 4;

T = A(K + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA);

U = A(K + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA);

V = T + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA) - T);

W = U + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA) - U);

DNDA\_OUT = V + (W-V) \* abs(DB);

end

function DNDR\_OUT = DNDR(ALPHA,BETA)

% yawing mom. due to rudder

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

A = [-.018 -.052 -.052 -.052 -.054 -.049 -.059 -.051 -.030 -.037 -.026 -.013;

-.028 -.051 -.043 -.046 -.045 -.049 -.057 -.052 -.030 -.033 -.030 -.008;

-.037 -.041 -.038 -.040 -.040 -.038 -.037 -.030 -.027 -.024 -.019 -.013;

-.048 -.045 -.045 -.045 -.044 -.045 -.047 -.048 -.049 -.045 -.033 -.016;

-.043 -.044 -.041 -.041 -.040 -.038 -.034 -.035 -.035 -.029 -.022 -.009;

-.052 -.034 -.036 -.036 -.035 -.028 -.024 -.023 -.020 -.016 -.010 -.014;

-.062 -.034 -.027 -.028 -.027 -.027 -.023 -.023 -.019 -.009 -.025 -.010]';

S = 0.2 \* ALPHA;

K = fix(S);

if (K <= -2)

K = -1;

end

if (K >= 9)

K = 8;

end

DA = S - double(K);

L = K + fix( sign(DA)\*1.1 );

S = 0.1 \* BETA;

M = fix(S);

if (M == -3)

M = -2;

end

if (M >= 3)

M = 2;

end

DB = S - double(M);

N = M + fix( sign(DB)\*1.11 );

IDX\_SHIFT\_ALPHA = 3;

IDX\_SHIFT\_BETA = 4;

T = A(K + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA);

U = A(K + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA);

V = T + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, M + IDX\_SHIFT\_BETA) - T);

W = U + abs(DA) \* (A(L + IDX\_SHIFT\_ALPHA, N + IDX\_SHIFT\_BETA) - U);

DNDR\_OUT = V + (W-V) \* abs(DB);

end

% DOLIN.m

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

%% Linmod

load trimcond.mat

[A,B,C,D] = linmod('F16sim\_lin',X\_trim,U\_trim);

% 1 - V

% 2 - alpha

% 3 - beta

% 4 - phi

% 5 - theta

% 6 - psi

% 7 - p

% 8 - q

% 9 - r

% 10 - Xe

% 11 - Ye

% 12 - h

%% Longitudinal

idx\_lon = [1 2 5 8 10 12 13];

A\_lon = A(idx\_lon,idx\_lon);

B\_lon = B(idx\_lon,[1 2]);

C\_lon = C(idx\_lon,idx\_lon);

D\_lon = D(idx\_lon,[1 2]);

sys\_lon = ss(A\_lon,B\_lon,C\_lon,D\_lon);

disp('Longitudinal Poles');

rifd(eig(A\_lon));

%% Lateral

idx\_lat = [3 4 6 7 9 11];

A\_lat = A(idx\_lat,idx\_lat);

B\_lat = B(idx\_lat,[3 4]);

C\_lat = C(idx\_lat,idx\_lat);

D\_lat = D(idx\_lat,[3 4]);

sys\_lat = ss(A\_lat,B\_lat,C\_lat,D\_lat);

disp(' ')

disp('Lateral Poles');

rifd(eig(A\_lat));

%% Step responses - longitudinal

% throttle

t = 0:0.01:1;

n = length(t);

u = [ones(n,1) zeros(n,1)];

[Y,T] = lsim(sys\_lon, u, t);

figure(1);

subplot(211);

plot(T,Y(:,1))

ylabel('V (ft/sec)')

subplot(212);

plot(T,Y(:,2)\*180/pi)

ylabel('alpha (deg)')

xlabel('time (sec)')

figure(2);

subplot(311);

plot(T,Y(:,3)\*180/pi)

ylabel('theta (deg)')

subplot(312);

plot(T,Y(:,4)\*180/pi)

ylabel('q (deg/sec)')

subplot(313);

plot(T,Y(:,7))

ylabel('power (ft-lb/sec)')

xlabel('time (sec)')

% elevator

t = 0:0.01:1;

n = length(t);

u = [zeros(n,1) ones(n,1)];

[Y,T] = lsim(sys\_lon, u, t);

figure(3);

subplot(211);

plot(T,Y(:,1))

ylabel('V (ft/sec)')

subplot(212);

plot(T,Y(:,2)\*180/pi)

ylabel('alpha (deg)')

xlabel('time (sec)')

figure(4);

subplot(311);

plot(T,Y(:,3)\*180/pi)

ylabel('theta (deg)')

subplot(312);

plot(T,Y(:,4)\*180/pi)

ylabel('q (deg/sec)')

subplot(313);

plot(T,Y(:,7))

ylabel('power (ft-lb/sec)')

xlabel('time (sec)')

%% Step responses - lateral

% aileron

t = 0:0.01:1;

n = length(t);

u = [ones(n,1) zeros(n,1)];

[Y,T] = lsim(sys\_lat, u, t);

figure(5);

subplot(211);

plot(T,Y(:,1)\*180/pi)

ylabel('beta (deg)')

subplot(212);

plot(T,Y(:,2)\*180/pi)

ylabel('phi (deg)')

xlabel('time (sec)')

figure(6);

subplot(311);

plot(T,Y(:,3)\*180/pi)

ylabel('psi (deg)')

subplot(312);

plot(T,Y(:,4)\*180/pi)

ylabel('p (deg/sec)')

subplot(313);

plot(T,Y(:,5)\*180/pi)

ylabel('r (deg/sec)')

xlabel('time (sec)')

% rudder

t = 0:0.01:1;

n = length(t);

u = [zeros(n,1) ones(n,1)];

[Y,T] = lsim(sys\_lon, u, t);

figure(7);

subplot(211);

plot(T,Y(:,1)\*180/pi)

ylabel('beta (deg)')

subplot(212);

plot(T,Y(:,2)\*180/pi)

ylabel('phi (deg)')

xlabel('time (sec)')

figure(8);

subplot(311);

plot(T,Y(:,3)\*180/pi)

ylabel('psi (deg)')

subplot(312);

plot(T,Y(:,4)\*180/pi)

ylabel('p (deg/sec)')

subplot(313);

plot(T,Y(:,5)\*180/pi)

ylabel('r (deg/sec)')

xlabel('time (sec)')

% DOTRIM.m

% Trim Function

% Author: Subhabrata Ganguli

% V0 = input

% H0 = input

% Straight Level Flight

% theta = alpha (flight path angle = 0)

%% Inputs

V0 = 502; % ft/sec

h0 = 0; % ft

xcg = 0.35;

%% Set trim function input

Z0 = [zeros(13,1)];

Z0(1) = V0;

[Zstar,f0,exitflag] = fminunc('F16COST',Z0,...

optimset('TolFun',1e-10,...

'TolX',1e-10,'MaxFunEval',1e5,'MaxIter',1e5),...

xcg,V0,h0)

x13 = TGEAR(Zstar(10));

X\_trim = [Zstar(1:9); 0; 0; 0; x13];

U\_trim = Zstar(10:13);

save trimcond X\_trim U\_trim xcg

function PDOT\_OUT = PDOT(P3,P1)

% PDOT\_OUT = rate of change of power

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

% P3 = actual power

% P1 = power command

if (P1 >= 50)

if (P3 >= 50)

T = 5;

P2 = P1;

else

P2 = 60;

T = RTAU(P2-P3);

end

else

if (P3 >= 50)

T = 5;

P2 = 40;

else

P2 = P1;

T = RTAU(P2-P3);

end

end

PDOT\_OUT = T\*(P2-P3);

end

function [P1\_OUT] = PDOTINV\_K(P3,PDOT\_IN)

% P1\_OUT is power command for P3 (actual power) and PDOT\_IN

% (power rate)

%

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

% P3 = actual power

% P1 = power command

­­­

if isempty(P3)

disp('empty')

end

ctr = 1;

k = 1;

P1\_OUT\_vec = 100\*ones(100,1);

for P1 = 0:0.01:100

if (P1 >= 50)

if (P3 >= 50)

T = 5;

P2 = P1;

else

P2 = 60 + 0.1\*P1;

T = RTAU\_K(P2-P3);

end

else

if (P3 >= 50)

T = 5;

P2 = 40 + 0.1\*P1;

else

P2 = P1;

T = RTAU\_K(P2-P3);

end

end

PDOT\_OUT = T\*(P2-P3);

if abs(PDOT\_OUT - PDOT\_IN) < 1e-6 % higher tolerance

P1\_OUT\_vec(k) = P1;

k = k+1;

elseif abs(PDOT\_OUT - PDOT\_IN) < 1e-2 % lower tolerance

P1\_OUT\_vec(k) = P1;

k = k+1;

end

ctr = ctr + 1;

end

P1\_OUT = min(P1\_OUT\_vec);

end

function RTAU\_OUT = RTAU(DP)

% function used by PDOT

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

if (DP < 25)

RTAU\_OUT = 1;

elseif (DP > 50)

RTAU\_OUT = 0.1;

else

RTAU\_OUT = 1.9 -.036 \* DP;

end

function TGEAR\_OUT = TGEAR(THTL)

% Power command v. thtl. relationship

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

if (THTL <= 0.77)

TGEAR\_OUT = 64.94\*THTL;

else

TGEAR\_OUT = 217.38\*THTL-117.38;

end

end

function [THTL\_OUT] = TGEARINV\_K(TGEAR\_IN)

% Power command v. thtl. relationship

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

THTL\_OUT = 0;

i = 0;

for THTL = 0:0.001:1

if (THTL <= 0.77)

TGEAR = 64.94\*THTL;

else

TGEAR = 217.3816\*THTL-117.38;

end

if (abs(TGEAR - TGEAR\_IN) < 0.5) && (TGEAR ~= 0) && (THTL ~=0)

THTL\_OUT = THTL;

break

end

i = i+1;

end

end

function THRUST\_OUT = THRUST(POW,ALT,RMACH )

% thrust

% Author: Subhabrata Ganguli

%

% Ref: Stevens, Brian L.; Lewis, Frank L.; Johnson, Eric N..

% Aircraft Control and Simulation: Dynamics,

% Controls Design, and Autonomous Systems.

% Wiley.

% nominal data

A = [1060.0 670.0 880.0 1140.0 1500.0 1860.0;

635.0 425.0 690.0 1010.0 1330.0 1700.0;

60.0 25.0 345.0 755.0 1130.0 1525.0;

-1020.0 -710.0 -300.0 350.0 910.0 1360.0;

-2700.0 -1900.0 -1300.0 -247.0 600.0 1100.0;

-3600.0 -1400.0 -595.0 -342.0 -200.0 700.0]';

% military data

B = [12680.0 9150.0 6200.0 3950.0 2450.0 1400.0;

12680.0 9150.0 6313.0 4040.0 2470.0 1400.0;

12610.0 9312.0 6610.0 4290.0 2600.0 1560.0;

12640.0 9839.0 7090.0 4660.0 2840.0 1660.0;

12390.0 10176.0 7750.0 5320.0 3250.0 1930.0;

11680.0 9848.0 8050.0 6100.0 3800.0 2310.0]';

% max data

C = [20000.0 15000.0 10800.0 7000.0 4000.0 2500.0;

21420.0 15700.0 11225.0 7323.0 4435.0 2600.0;

22700.0 16860.0 12250.0 8154.0 5000.0 2835.0;

24240.0 18910.0 13760.0 9285.0 5700.0 3215.0;

26070.0 21075.0 15975.0 11115.0 6860.0 3950.0;

28886.0 23319.0 18300.0 13484.0 8642.0 5057.0]';

H = .0001\*ALT;

I = fix(H);

if (I>=5)

I=4;

end

DH= H-I;

RM= 5.0 \* RMACH;

M = fix(RM);

if (M>=5)

M=4;

end

DM= RM-M;

CDH=1.0-DH;

IDX\_SHIFT1 = 1;

IDX\_SHIFT2 = 1;

I = I + IDX\_SHIFT1;

M = M + IDX\_SHIFT2;

S = B(I,M) \*CDH + B(I+1,M) \*DH;

T = B(I,M+1)\*CDH + B(I+1,M+1)\*DH;

TMIL = S + (T-S)\*DM;

if( POW < 50.0 )

S = A(I,M) \*CDH + A(I+1,M) \*DH;

T = A(I,M+1)\*CDH + A(I+1,M+1)\*DH;

TIDL = S + (T-S)\*DM;

THRUST\_OUT = TIDL+(TMIL-TIDL)\*POW\*.02;

else

S = C(I,M) \*CDH + C(I+1,M) \*DH;

T = C(I,M+1)\*CDH + C(I+1,M+1)\*DH;

TMAX = S + (T-S)\*DM;

THRUST\_OUT = TMIL+(TMAX-TMIL)\*(POW-50.0)\*.02;

end

end

# References

* Stevens, B.L., Lewis, F.L (2003). *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*, Wiley Edition.