

Simulation of Autonomous Landing

By

Jason A. Tolvtvar

AEROSPACE ENGINEERING DEPARTMENT

California Polytechnic State University, San Luis Obispo

July 15, 2005

## DISCLAIMER

This project was completed to fulfill the College of Engineering senior project requirement for attainment of the Baccalaureate degree. It has been graded and accepted as fulfillment of the degree requirement. No implication is made of its technical accuracy or reliability. Any use of this project is done so at the sole risk of the user. California Polytechnic State University at San Luis Obispo and its staff cannot be held liable for any use or misuse of the information contained herein.

## TABLE OF CONTENTS

|   |     |
|---|-----|
| Disclaimer  | ii  |
| Table of Contents                                     | iii |
| List of Figures                                       | iv  |
| List of Symbols                                       | v   |
| Abstract  | 1   |
| Introduction  | 2   |
| Procedure   | 3   |
| Results and Discussion                                | 6   |
| Conclusions   | 21  |
| Appendix A: MatLAB Code to Produce Transfer Functions | 22  |
| Appendix B: Simulation Architecture                   | 27  |
| References  | 34  |

## LIST OF FIGURES

|  |    |
|--|----|
| Figure 1: Glide Path Geometry  | 3  |
| Figure 2: Flare Path Geometry  | 4  |
| Figure 3: Root Locus and Bode Plot for Pitch Angle Autopilot                               | 10 |
| Figure 4: Predicted and Actual Responses of the Pitch Angle Autopilot to a Step<br>Command | 11 |
| Figure 5: Actual Response of Pitch Angle Autopilot With PID Compensation                   | 12 |
| Figure 6: Actual Glideslope Response   | 14 |
| Figure 7: Glideslope Response With Updraft and Downdraft of 20 ft/s                        | 14 |
| Figure 8: Glideslope/Flare Blending Function   | 18 |
| Figure 9: Aircraft Response During Glideslope/Flare Switch and Touchdown                   | 19 |

## LIST OF SYMBOLS

| <u>Symbol</u> | <u>Description</u>                      | <u>Units</u>           |
|---------------|---|------------------------|
| 6DoF          | Six degrees of freedom                  | N/A                    |
| $d$           | Linear glide path error                 | Feet                   |
| $H$           | Height above ground level               | Feet                   |
| $H_0$         | Initial Height above ground level       | Feet                   |
| LLA           | Latitude, longitude altitude            | Degrees, degrees, feet |
| PID           | Proportional, integral, derivative      | N/A                    |
| $R$           | Range from aircraft to runway threshold | Feet                   |
| $x$           | Ground distance from aircraft to runway | Feet                   |
| $\beta$       | Glide slope angular error               | Degrees                |
| $\gamma$      | Glide path angle                        | Degrees                |
| $\delta_e$    | Elevator deflection angle               | Radians                |
| $\theta$      | Euler pitch angle                       | Radians                |
| $\tau$        | Exponential decay constant              | N/A                    |

## ABSTRACT

In this project, controllers were developed for use with the Cal Poly Flight Simulator to simulate autonomous landing of an F-4 *Phantom*. Pitch angle and airspeed autopilots were developed, followed by glide slope and flare controllers to guide the aircraft to successful landing. The flight path command simulation architecture was developed to provide the appropriated command signals for glide path, flare, and braking after touchdown. The flight path command signals reference the destination airport allowing the simulation to perform at any airport supported by the simulation graphics. Successful landings were demonstrated at a variety of simulated airports all over the earth. The glide path controller responds well to up and down drafts of up to 20 ft/s, however, the flare controller requires ideal conditions for successful landing.

## INTRODUCTION

The overall goal of this project was to achieve autonomous landing simulation for the Cal Poly Flight Simulator. One of the very long term goals of the simulation project is to achieve fully autonomous flight and before this project, autonomous takeoff and autonomous waypoint navigation simulations had been developed. The development of an autonomous landing simulation completes the requirements to begin fully autonomous flight simulation.

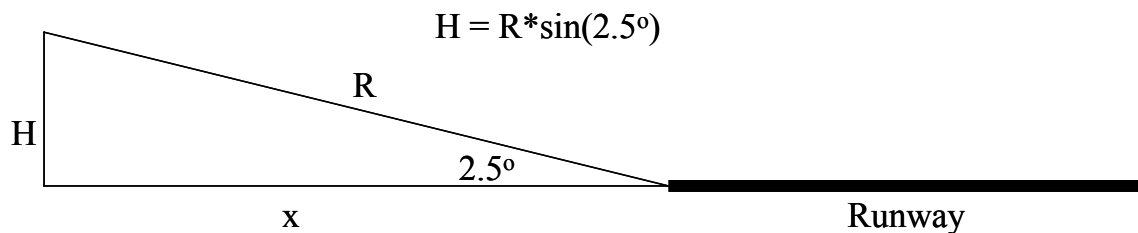
The Cal Poly Flight Simulator employs a six degree-of-freedom (6DoF) mathematical model with table lookup of aerodynamic coefficients for simulation of an F-4 *Phantom*. It utilizes commercial gaming software *X-Plane* for graphics and includes an airport selector function that allows the simulation to operate at many airports supported by *X-Plane*. The simulator uses two coordinate systems, standard xyz and latitude-longitude-altitude (LLA). *X-Plane* requires LLA for graphical output.

The landing portion of autonomous flight is accomplished with reference to the destination airport. It is also dependent on the range from the aircraft to the runway. Because the airport is the main reference for flight path geometry and *X-Plane* uses LLA coordinates, these coordinates should be used to determine the range. The requirements to successfully complete an autonomous landing are: define the glide path and flare path geometry, design autopilots for pitch, roll, and yaw, and design controllers for glide path, flare, and directional course.

## PROCEDURE

The first step in the process of simulating autonomous landing is to define the glide path and flare path geometries. The glide path is defined as a line from some starting point to the end of the runway. For this project, a glide path angle of  $-2.5^\circ$  was used, so the starting point was defined by the LLA position of the end of the runway and the desired final approach distance, four nautical miles in this case.

A real autonomous landing controller would rely on signals emitted from stations at the end of the runway to maintain the appropriate glide path. As a result, the controller becomes more sensitive as the aircraft gets closer to the runway threshold. To simulate this dependence on the range from the aircraft to the runway, the glide path command signal was defined to include the range. Figure 1 shows the glide path geometry where the commanded height above ground is a function of the range.



**Figure 1: Glide Path Geometry**

To simulate the increasing sensitivity to range, the error signal should be defined as the error in glide slope angle  $\gamma$ . For small angles, the arc length of an angular

displacement can be approximated by a straight line and thus the line distance is equal to the radius of the circle multiplied by the angle as shown in equation 1 where  $d$  is the perpendicular distance off the glide path,  $R$  is the range, and  $\beta$  is the angular error.

$$d = R\beta \quad (1)$$

Using this relationship, the angular error can be found in terms of the range where for the same linear error, the angular error becomes increasingly larger as the range gets smaller. At some point, the sensitivity of the glide slope controller exceeds its ability to keep the airplane on the glide path. At this point, the flare controller, which is not dependent on the range, must take over control of the airplane.

During the flare maneuver, pilots transition from flying a straight line to an exponential path to slow the descent rate of the airplane. This can be simulated by defining an exponentially decaying flight path and using altitude above ground to generate the error signal to the controller. Figure 2 shows the flare path geometry with the intended touchdown zone approximately 500 ft. from the runway threshold.

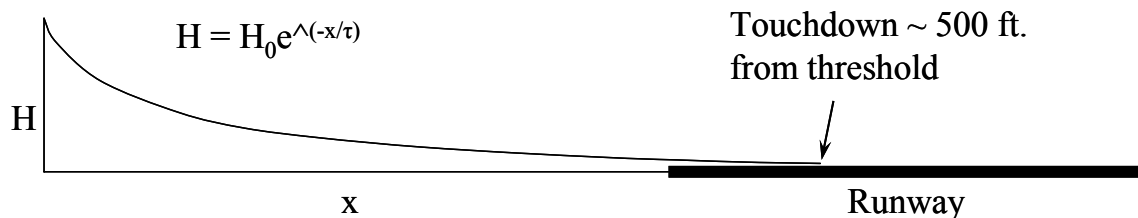


Figure 2: Flare Path Geometry

The exponential decay constant  $\tau$  is a function of the distance  $x$  and the distance to the touchdown zone from the threshold as exponential functions decay in approximately 5 decay constants.  $\tau$  must be selected such that the airplane touches down where desired and at an acceptable vertical speed.

Upon main gear touchdown, the elevator must allow the nose of the airplane to rotate downward to contact the nose gear with the ground and the brakes must be applied. Because exponentially decaying functions never actually reach zero, the elevator control command must be switched from the flare path to neutral upon main gear touchdown. The brakes must be applied smoothly after touchdown or the gear will fail. A rate limiter after a switch can be used to accomplish this.

With the flight path geometry defined, autopilots for pitch, yaw, and roll must be designed to fly the airplane autonomously. Then controllers for glide slope, flare, and directional course must be designed to keep the airplane on the desired flight path. Because of the complexity of the problem, it was determined that only pitch control should be used without wind disturbances to accomplish the landing. Upon successful landing in ideal conditions, lateral controllers can be designed for responsiveness in conditions with wind disturbances.

## RESULTS AND DISCUSSION

Because the glide slope geometry and control signal error depend on the range to the runway threshold, calculation of the range was the first step in this simulation. The Airport Selector S-function was evaluated to determine if the outputs from it were suitable for determining the range. The initial LLA and initial distance from the runway are output from the Airport Selector function and are suitable for finding the initial range and runway threshold LLA. In planning for lateral control the Airport Selector S-function was modified to output the true heading of the runway.

The simulator outputs LLA for the airplane at every time step of simulation. Knowing the LLA of the runway threshold, and the instantaneous LLA, the instantaneous range can be found. The instantaneous LLA can be subtracted from the threshold LLA to obtain the LLA difference. Assuming the earth is flat for small distances, this LLA difference represents a line in space that is found by the Pythagorean Theorem. For this simulation, the initial range was four nautical miles which is sufficiently small to consider the earth as flat.

Latitude and longitude are represented in degrees, where there are 60 minutes per degree and 60 seconds per minute, and must be converted to feet so that they have the same units as altitude. The latitude conversion to feet is relatively constant from the equator to the poles and is approximated at all points as 6076 feet per minute. The longitude conversion to feet varies from the equator to the poles. This is because the lines

of longitude become closer together toward the poles. The circle created by intersecting a plane with the earth at some line of latitude will have a radius equal to the radius of the earth times the cosine of the latitude angle. The radius of this circle is used to calculate the circumference of the earth at that particular latitude. Regardless of the circumference of the circle, it still contains 360 degrees, thus a conversion factor can be found. The website, [www.google.com](http://www.google.com)<sup>1</sup> provides commonly used constants, conversion factors, and measurements. Google provides the average radius of the earth as 36,522 feet. This then yields the conversion factor for longitude as  $36,522 * \cos(\text{latitude}^\circ)$  feet per minute.

Altitude is output from the simulation as feet above mean sea level. The runway altitude is output from the Airport Selector function and was subtracted from the instantaneous altitude to give the height above ground level (H). By the Pythagorean Theorem then, the range is given by

$$R = \sqrt{(\text{latitude ft.})^2 + (\text{longitude ft.})^2 + (\text{altitude ft.})^2} \quad (2)$$

where the latitude and longitude used in this calculation are the instantaneous latitude and longitude subtracted from the runway threshold latitude and longitude and converted to feet.

Subtracting the instantaneous LLA from the runway threshold LLA proved to be problematic for Simulink as regardless of the signs in the sum block, the range would only get larger. So, the initial range was found by using the initial ground distance and initial altitude output from Airport Selector. Then, the instantaneous LLA was subtracted

from the initial LLA to find the change in range. After conversion to feet, this change was subtracted from the initial range to get the instantaneous range. This can present a problem if the aircraft does not establish the glide path quickly because the change in range can be greater than intended if the aircraft gets far off the desired path. However, with reasonably small errors in the flight path, this approximation works well.

To begin the design of the pitch angle autopilot, a transfer function representative of the F-4 *Phantom* in landing conditions was required. The table used by the 6DoF S-function (simtable.txt) lists the weight of the aircraft at 39,000 pounds. From experience in flying the simulation, the airplane lands well at approximately 200 knots which translates to about 330 ft/s. This corresponds to a lift coefficient of 0.57. Using this value, and the desired speed, the angle of attack and Mach number could be found. Knowing the angle of attack and Mach number, the drag coefficient and pitching moment coefficient were determined from the table. With these values and the other stability and control derivatives listed in the table, a transfer function for pitch angle to elevator deflection could be determined from the short period mode literal factors approximation. A script was written in MatLAB to find transfer functions for short period, dutch roll, and spiral modes by literal factors (Appendix A) and generated the transfer function

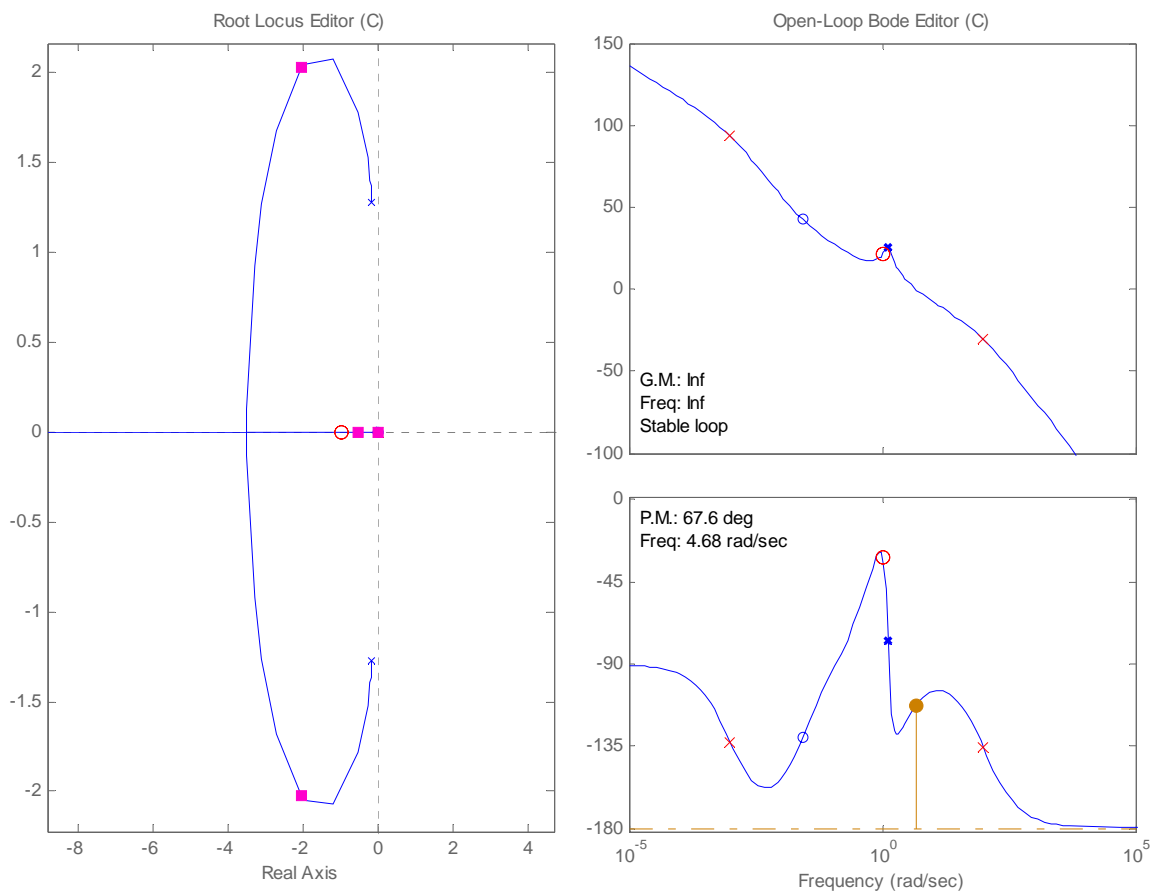
$$\frac{\theta}{\delta_e} = \frac{1728s + 46.81}{330s^3 + 116.9s^2 + 546.4s} \quad (3)$$

which was used for pitch angle autopilot design.

Using MatLAB's SISO tool, the compensator

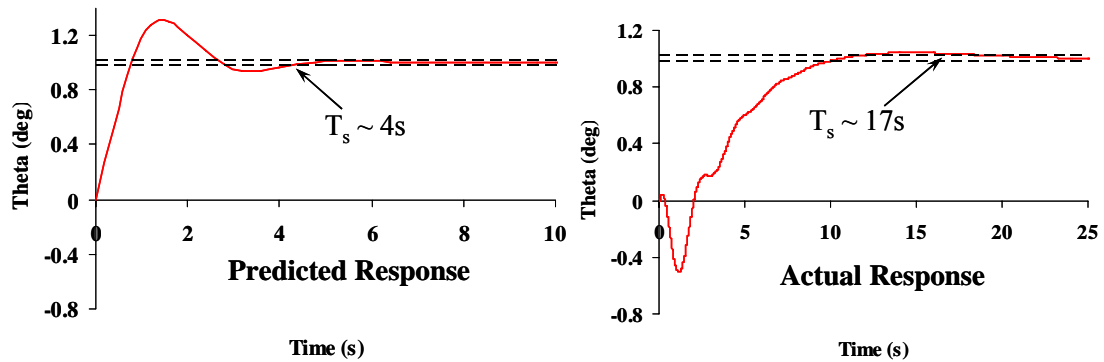
$$\frac{79.4(s+1)(s+1)}{(s+0.001)(s+100)} \quad (4)$$

was designed for pitch angle to yield a damping ratio of 0.709, a crossover frequency of 4.68 rad/s, and a phase margin of 67.6 degrees. Figure 3 shows the SISO tool display for the pitch angle autopilot. The root locus shows a stable system with damping of 0.709, so approximately three overshoots are expected in the response. The Bode plot shows a crossover frequency of 4.68 rad/s which indicates the system should respond well for the high-bandwidth task of landing, but not be responsive to high-frequency disturbances and noise.



**Figure 3: Root Locus and Bode Plot for Pitch Angle Autopilot**

Figure 4 shows the predicted and actual responses of the pitch angle autopilot to a step command. The predicted response shows three overshoots as expected and a suitable settling time of four seconds. The actual response, however, is radically different. The initial response is opposite the command, and then with some oscillation and one overshoot, the aircraft reaches the desired pitch angle with a settling time of approximately 17 seconds.

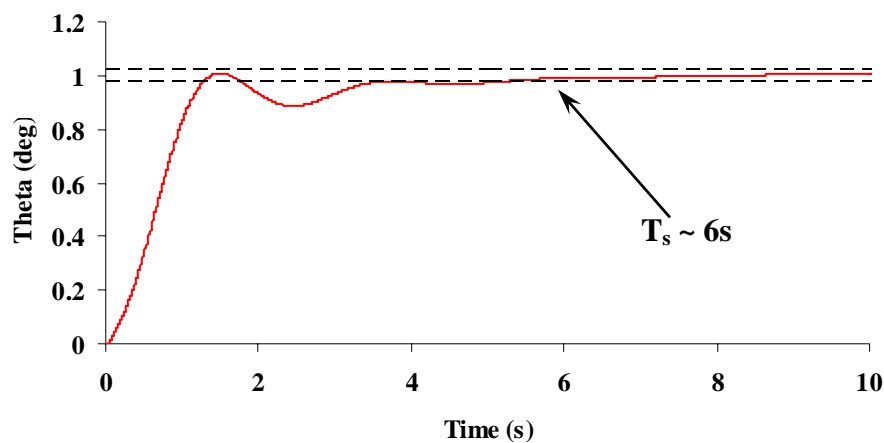


**Figure 4: Predicted and Actual Responses of the Pitch Angle Autopilot to a Step Command**

The actual settling time of 17 seconds is much too slow for an autopilot to control an aircraft on landing. More importantly, however, the plant from which the autopilot was designed is obviously much different than representative of the actual simulation. Two possibilities for explanation exist. Either the mathematical calculations within the 6DoF S-function are not correct or the literal factors approximation is not valid for this case. Regardless, another method was required to design an appropriate autopilot for pitch angle.

Without knowing the plant of the system, the only method left was trial and error. The two options available were to attempt to place poles and zeros of a lead-lag compensator to achieve the desired response or to search for PID (proportion, integral, derivative) controller gains. Pole placement for lead-lag was attempted, but abandoned in favor of using a PID controller as progress was not being made. The PID control gains were determined by setting up a simulation block with a step command and pitch angle feedback. Then, a proportional gain, integral gain, and derivative gain were added to the error signal and sent to the 6DoF. The simulation was run for short periods of time and a

scope was used to evaluate the output. The three gains were changed according to the observed response on the scope. The proportion was used to decrease the settling time, the derivative was used to decrease the overshoots and the integral was used to eliminate the steady state error. Using this method, the required gains were found to be 2.5, 5, and -1.5 for the proportion, integral, and derivative respectively. Figure 5 shows the actual time response of this system to a step command. The settling time is just under six seconds with one overshoot and some small, minor oscillation.



**Figure 5: Actual Response of Pitch Angle Autopilot with PID Compensation**

In order to design the pitch angle autopilot adequately for glide slope control, airspeed control was required. The engine model currently running in the simulation has very little time lag. In fact, it is so small, it can be considered negligible. So, it was assumed that a lag compensator would be required to maintain airspeed. The desired airspeed is 330 ft/s and the controller uses forward airspeed feedback. However, the throttle command signal must be a small number between zero and one where zero is reverse thrust, 0.1 is neutral, and one is maximum forward thrust. So, to normalize the

error signal to a maximum value of one, a gain of 1/330 was imposed. The initial assumption of lag compensation proved correct and with little effort, the compensator

$$\frac{3(s + 0.1)}{s} \quad (5)$$

was found to provide exceptional performance.

In designing the glide slope controller, again not having the plant proved to be an extreme challenge. Two methods were considered, as for the pitch autopilot, for designing an adequate controller, attempt to place lead-lag poles and zeros, or search for PID gains. On the assumption that lead-lag should work for this design, the first method was employed. For the glide slope controller, however, an understanding of the effect of pole and zero placement was gained by changing their locations individually and observing the response. Then, experimentally, the compensator

$$\frac{50(s + 0.1)(s + 0.1)}{(s + 1)s} \quad (6)$$

was determined to provide good performance. This is shown in Figure 6 where the airplane falls below glide path initially, but comes back with no overshoot while still more than 10,000 ft. from the runway.

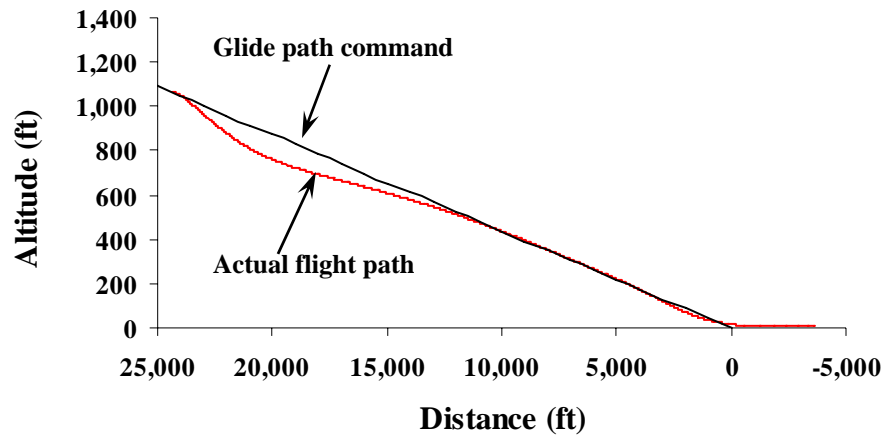


Figure 6: Actual Glide slope Response

The glide slope controller performs well in updrafts and downdrafts as shown in Figure 7. The plot on the left shows the response to an updraft of 20 ft/s and the plot on the right shows the response to a down draft of the same magnitude. In the updraft case, the aircraft virtually does not leave the desired path until the flare command and in the downdraft case, the aircraft is still established on the glide path at a range of around 10,000 ft.

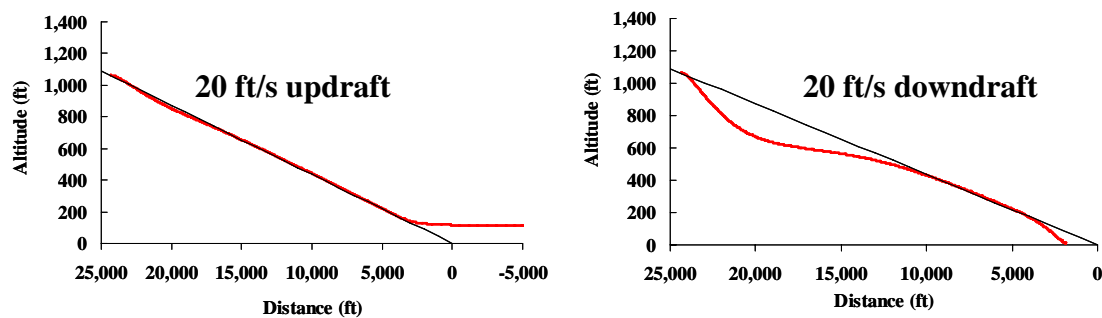


Figure 7: Glide slope Response with Updraft and Downdraft of 20 ft/s

After experiencing the difficulty of trial and error design for the glide slope controller, it was clear that the flare controller design would be similar. However, there was no way to start the simulation at the flare engagement point, so the method of

switching control from the glide slope to the flare controller had a profound effect on the ability to design the flare controller. Blakelock<sup>2</sup> suggests that the flare controller should be the same as the glide slope controller with an additional lead network. The initial attempt at the flare controller was then, the glide slope controller.

Blakelock also suggests that the sink rate should be controlled. This is because the flare path command is an exponential function for which the derivative is a constant multiplied by the instantaneous altitude. The constant is the inverse of the exponential decay time constant. Then, assuming that the aircraft will touch down in four or five time constants, the appropriate time constant for the function can be determined. The following equations show this relationship for the flare path command.

$$H = H_0 e^{-t/\tau} \quad (7)$$

$$\dot{H} = -\frac{1}{\tau} H_0 e^{-t/\tau} \quad (8)$$

$$\dot{H} = -\frac{1}{\tau} H \quad (9)$$

Considering Blakelock's suggestions, the aircraft sink rate was used for the flare command along with the glide slope controller in the initial attempt. The range at glide slope/flare switch was selected as 2,000 ft. This is because the glide slope controller had proven to perform well to a range somewhat less than 2,000 ft. and the aircraft is just

under 100 ft. altitude above ground level at a range of 2,000 ft. The initial attempt was to switch the signals at the instant the range became equal to or less than 2,000 ft. A problem was immediately evident, however as the aircraft became amazingly unstable at the moment of command signal switch.

Upon further evaluation of Blakelock's methods, it was determined that a time dependent function was not appropriate as it required that the aircraft be at the proper position for flare at a certain time step in the simulation. There is no way to guarantee this requirement, so the time dependent exponential function was abandoned. The geometry of the glide path and flare are both dependent on the position off of the runway threshold. Defining the flare path command in terms of the distance off of the runway seemed much more appropriate because the distance off the runway is always known, regardless of the simulation time step. The path was then defined as:

$$H = H_0 e^{-x/\tau} \quad (10)$$

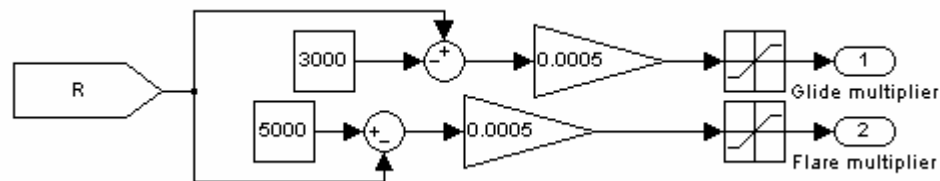
where  $x$  is the distance from the glide slope engagement point determined by subtracting the instantaneous LLA from the initial LLA, converting the latitude and longitude differences to feet and solving for  $x$  by the Pythagorean Theorem. Because the starting point of the flare command signal coincides with the glide path command signal, the value of the decay constant was found to be 5,500. This corresponds to touchdown in approximately five decay constants.

Having the flare path defined in terms of the distance from the runway, the altitude above ground level was used as feedback to generate the error signal. Again the glide slope controller was evaluated for use in the flare, but was not suitable as the command signal was no longer dependent on the sink rate. By experiment, it was observed that with a very small gain, the aircraft could be made to flare very well and touch down at any point desired. With a flare command gain of 0.0004995, the aircraft touches down approximately 600 ft. after the runway threshold. The only problem with this is that it requires ideal conditions to work as there is no integrator in the compensator.

An additional problem in the flare controller design was the method of switching from glide slope to flare command signals. With the switch at a range of 2,000 ft., the aircraft could not be made stable as having a switch in the simulation caused adverse affects on the very input signals to the switch. It was determined by experiment that constants in Simulink create problems in solving the algebraic loop and cause the simulation to produce erroneous results. To compensate, step blocks were used with the step value equal to that of the desired constant and the step time equal to the first time step of simulation (0.01 s). Even with this correction, stability problems were still evident at the switch of control commands.

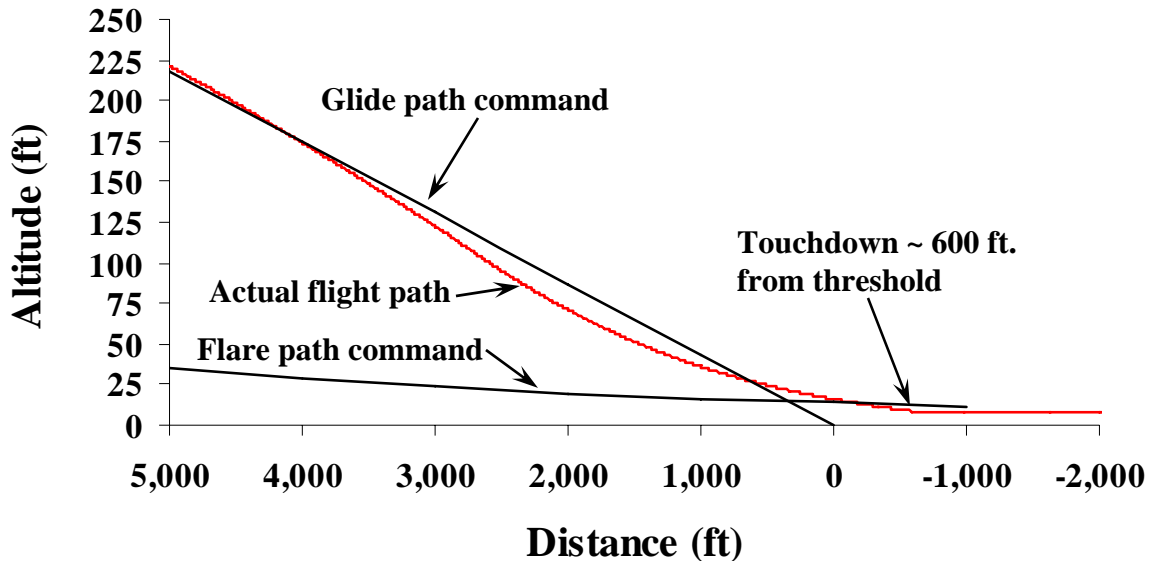
To compensate for the sudden switch in commands, a blending function was developed at the direction of Dr. Dan Biezd<sup>3</sup> to soften the effect of the switch. This function blends the signals over range values of 5,000 ft. to 3,000 ft. These values were selected to ensure that the aircraft would be established under the flare controller before

reaching the desired switch range of 2,000 ft. Figure 8 shows the Simulink architecture of the function. The saturation blocks normalize the signal multipliers to a value between zero and one. 3,000 is subtracted from the range and multiplied by a gain to generate a multiplier for the glide path command. The gain was selected such that at 5,000 ft. range, the glide multiplier is one and at 3,000 ft. range, the glide multiplier is zero. The flare multiplier is much the same, but in the reverse direction. These multipliers directly multiply the glide and flare signals such that when the range is between 5,000 ft. and 3,000 ft., both signals are active. Above 5,000 ft. only the glide path signal is active and below 3,000 ft. only the flare signal is active.



**Figure 8: Glide slope/Flare Blending Function**

The blending of signals at the switch of glide slope and flare control signals solved the problem of extreme oscillation and instability during the switch. In fact, graphically, the switch is almost unnoticeable. Figure 9 shows the response of the aircraft during the signal blend. Oscillation is not visible on this graph, but is very slightly visible in the graphics produced by *X-Plane*. Also shown in Figure 9 is the response of the aircraft during the flare. It flies an exponential path to touchdown approximately 600 ft. from the runway threshold. It also touches down at a vertical velocity of approximately four feet per second, which would be considered a good landing by any pilot.



**Figure 9: Aircraft Response During Glide slope/Flare Switch and Touchdown**

Because the flare path command is exponential, the aircraft tends to bounce if the elevator remains under control of the flare controller after main gear touchdown. To ensure the aircraft remains on the ground, the elevator is neutralized with a relay at an altitude above ground level equal to the height of the main gear (9 ft.). At the same altitude, the throttle command is also neutralized and the brake command is changed from one to zero (zero to full braking) with a rate limiter to limit the brake application time to two seconds. The rate limiter on braking prevents gear failure due to over-braking upon touchdown.

Upon completion of a successful landing in ideal conditions, the system was tested at various airports to ensure adequate representation of landing geometry with reference to LLA coordinates. Successful landings were demonstrated at San Luis Obispo, California, Kona, Hawaii, Dallas/Ft. Worth, Texas, New York (JFK), New York,

and Sydney, Australia indicating that the landing geometry definitions are in fact appropriate for use at any destination airport.

## CONCLUSIONS

The overall goal of this project was to achieve autonomous landing simulation on an F-4 *Phantom* using the Cal Poly Flight Simulator. Pitch control was used to fly the airplane on a defined glide path and then through a flare maneuver to touchdown 600 ft. after the runway threshold. The glide path controller was shown to respond well to disturbances and the flare controller was shown to require ideal conditions. Pitch angle and airspeed autopilots were required to make the landing and were designed appropriately. Successful landings were demonstrated at simulated airports around the world showing the versatility of the landing controller as it uses solely the destination airport for landing geometry and command reference.

This project does not complete the requirements for fully autonomous flight in the Cal Poly Flight Simulator, however. Lateral control was not addressed in this project as time ran short because of the many problems encountered in matching the simulation to theory. So, the simulation requires ideal conditions and a perfect starting position to work. Another problem that this project leaves unaddressed is the flare path command starts at the same point as the glide path command and is not tangent to the glide path at the switch point. For future work, the flare path command should be defined such that the flare path is tangent to the glide path in the switch range and then a new controller should be designed to respond appropriately to up and down draft disturbances. Lateral control can then be addressed followed by transition from cruise flight to the landing task.

## APPENDIX A

MATLAB CODE TO PRODUCE TRANSFER FUNCTIONS  
(Literal factors approximation equations from  
Study Guide for Flight Control Systems)<sup>4</sup>

```
% Sim_Dynamics.m
```

```
% This file generates the transfer functions based on literal factor
% approximations for the F4-E Phantom using derivatives from the file
% "sim_table.txt"
```

```
s = sym('s');
```

```
%=====
```

```
% Airplane data
```

```
%=====
```

```
U0 = 330;          %ft/s
Rho = 0.002377;   %slug/ft^3
W = 39000;        %lbs
S = 530;          %ft^2
b = 38.7;         %ft
cbar = 16;        %ft
Ixx = 25000;      %slug-ft^2
Iyy = 122000;     %slug-ft^2
Izz = 140000;     %slug-ft^2
Ixz = 2200;       %slug-ft^2
```

```
Q = 1/2*Rho*U0^2; %psf
m = W/32.2;       %slug
```

```
%=====
```

```
% Stability and control parameters
```

```
%=====
```

```
CL = W/(Q*S);
CL0 = 0.2;        %-|
CD = 0.1317;      % |
CLalpha = 3.684;  % |- Calculated estimates based on CL = 1.17
CDalpha = 0.7932; % |
Cmalpha = -0.184; %-|
```

```

CLalphadot = -1.3;
Cmalphadot = -1.3;
CLdelE = 0.4;
CDdelE = 0.1;
CD0 = 0.05;
Cm0 = -0.01;
CmdeIE = -0.58;
Cmq = -2.7;
Czalphadot = -CLalphadot;
CzdeIE = -CLdelE;

```

```

Cybeta = -0.68;
CydeIA = -0.016;
CydeIR = 0.095;
Cyr = 0;
Cyp = 0;
Clbeta = -0.08;
CldeIA = 0.042;
CldeIR = 0.006;
Clp = -0.27;
Clr = 0.07;
CndeIA = -0.001;
Cnbeta = 0.125;
CndeIR = -0.066;
Cnp = -0.036;
Cnr = -0.27;

```

```

%=====
% Longitudinal Directional Derivatives
%=====

```

```

Zwdot = -Czalphadot*cbar*Q*S/(2*U0^2*m);
Zalpha = U0*Zwdot;
Mw = Cmalpha*Q*S*cbar/(U0*Iyy);
Mwdot = Cmalphadot*Q*S*cbar^2/(2*U0^2*Iyy);
Malpha = U0*Mw;
Malphadot = U0*Mwdot;
Mq = Cmq*cbar/(2*U0);
ZdeIE = -CzdeIE*Q*S/m;
MdeIE = CmdeIE*Q*S*cbar/Iyy;

```

```

%=====
% Lateral Directional Derivatives
%=====

```

```

Ybeta = (Q*S*Cybeta)/m;
Nbeta = (Q*S*b*Cnbeta)/Izz;
Lbeta = (Q*S*b*Clbeta)/Ixx;
Yp = (Q*S*b*Cyp)/(2*m*U0);
Np = (Q*S*b^2*Cnp)/(2*Ixx*U0);
Lp = (Q*S*b^2*Clp)/(2*Ixx*U0);
Yr = (Q*S*b*Cyr)/(2*m*U0);
Nr = (Q*S*b^2*Cnr)/(2*Ixx*U0);
Lr = (Q*S*b^2*Clr)/(2*Ixx*U0);
YdelA = (Q*S*CydelA)/(m);
NdelA = (Q*S*b*CndelA)/Izz;
LdelA = (Q*S*b*CldelA)/Ixx;
YdelR = (Q*S*CydelR)/m;
NdelR = (Q*S*b*CndelR)/Izz;
LdelR = (Q*S*b*CldelR)/Ixx;

%=====
% Literal Factors Approximations
%=====

% Short Period
% Asp = [(s*U0 - Zalpha), -U0*s; -(Malphadot*s + Malpha), (s^2 - Mq*s)]
% Asp = [ Asp_1,    Asp_2,    Asp_3,    Asp_4 ]

% Bsp = [ZdelE; MdelE]
% Bsp = [Bsp_1; Bsp_2]

Asp_1 = [0, U0, -Zalpha];
Asp_2 = [0, -U0, 0];
Asp_3 = [0, -Malphadot, -Malpha];
Asp_4 = [1, -Mq, 0];

Bsp_1 = [0, 0, ZdelE];
Bsp_2 = [0, 0, MdelE];

Char_sp = conv(Asp_1, Asp_4) - conv(Asp_3, Asp_2);
Char_sp_rate = [Char_sp(1,2), Char_sp(1,3), Char_sp(1,4)]; %Reduces the order of the
denominator by 1, effectively taking the derivative.

Num_Alpha = -(conv(Bsp_1, Asp_4) - conv(Bsp_2, Asp_2));
Num_Theta = -(conv(Asp_1, Bsp_2) - conv(Asp_3, Bsp_1));

Alpha_delE = tf(Num_Alpha, Char_sp);

```

```

Alphadot_delE = tf(Num_Alpha, Char_sp_rate);
Theta_delE = tf(Num_Theta, Char_sp);
Thetadot_delE = tf(Num_Theta, Char_sp_rate);

disp('Transfer Function Alpha_delE');
Alpha_delE

disp('Transfer Function Alphadot_delE');
Alphadot_delE

disp('Transfer Function Theta_delE');
Theta_delE

disp('Transfer Function Thetadot_delE');
Thetadot_delE

% Dutch roll
% Adutch = [(s*U0 - Ybeta), s*(U0 - Yr); -Nbeta, (s^2 - s*Nr)]
% Adutch = [ Adutch_1, Adutch_2, Adutch_3, Adutch_4 ]

Adutch_1 = [0, U0, -Ybeta];
Adutch_2 = [0, (U0 - Yr), 0];
Adutch_3 = [0, 0, -Nbeta];
Adutch_4 = [1, -Nr, 0];

Char_Dutch = conv(Adutch_1, Adutch_4) - conv(Adutch_3, Adutch_2);
Char_Dutch_rate = [Char_Dutch(1,2), Char_Dutch(1,3), Char_Dutch(1,4)];

% Bdutch = [YdelR; NdelR]
% Bdutch = [Bdutch_1, Bdutch_2]

Bdutch_1 = [0, 0, YdelR];
Bdutch_2 = [0, 0, NdelR];

Num_Beta = conv(Bdutch_1, Adutch_4) - conv(Bdutch_2, Adutch_2);
Num_Psi = conv(Adutch_1, Bdutch_2) - conv(Adutch_3, Bdutch_1);

Beta_delR = tf(Num_Beta, Char_Dutch);
Betadot_delR = tf(Num_Beta, Char_Dutch_rate);
Psi_delR = tf(Num_Psi, Char_Dutch);
Psidot_delR = tf(Num_Psi, Char_Dutch_rate);

disp('Transfer Function Beta_delR');

```

```
Beta_delR

disp('Transfer Function Betadot_delR');
Betadot_delR

disp('Transfer Function Psi_delR');
Psi_delR

disp('Transfer Function Psidot_delR');
Psidot_delR

% Lateral Roll Mode
Num_Lat = [0, 0, LdelA];
Den_Lat = [1, -Lp, 0];
Den_Lat_rate = [Den_Lat(1,1), Den_Lat(1,2)];

p_delA = tf(Num_Lat, Den_Lat);
pdot_delA = tf(Num_Lat, Den_Lat_rate);

disp('Transfer Function p_delA');
p_delA

disp('Transfer Function pdot_delA');
pdot_delA
```

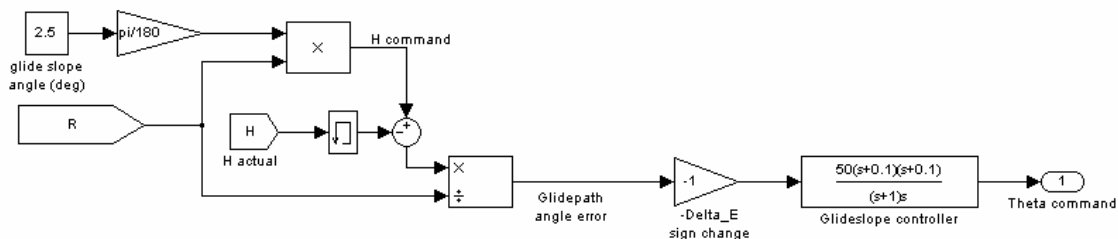
## APPENDIX B

### SIMULATION ARCHITECTURE

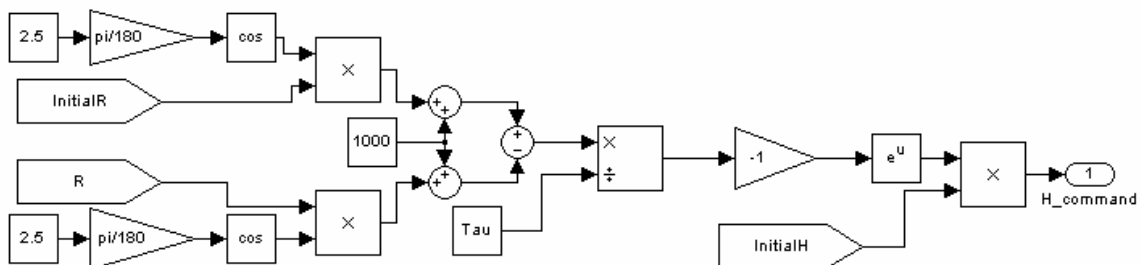
All of the simulation architecture for this project was placed in subsystems which were placed in one large subsystem that outputs and elevator control signal and a throttle control “goto” flag. The following pages show pictures of the actual block diagrams used to create the landing simulation. The pictures are in the following order: Glide path subsystem, glide path command, flare path command, glide/flare signal blending function, throttle command, braking command, and range. The range block is shown in six separate pictures.



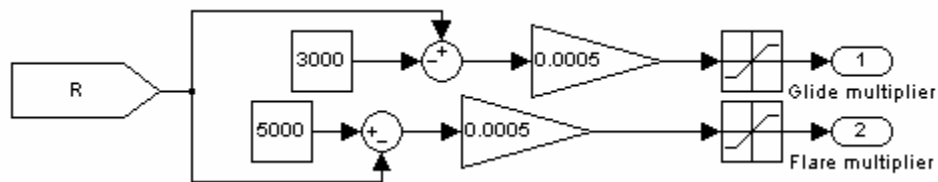
Glide path command



Flare path command

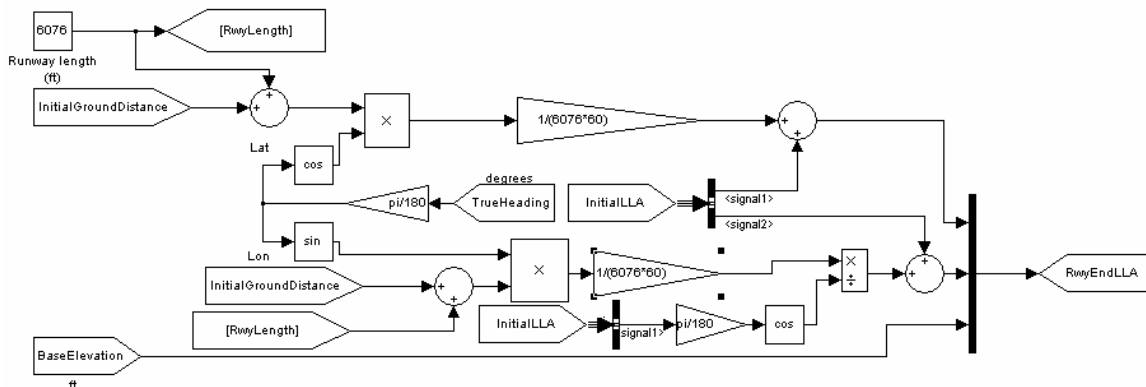


Glide/flare signal blending function

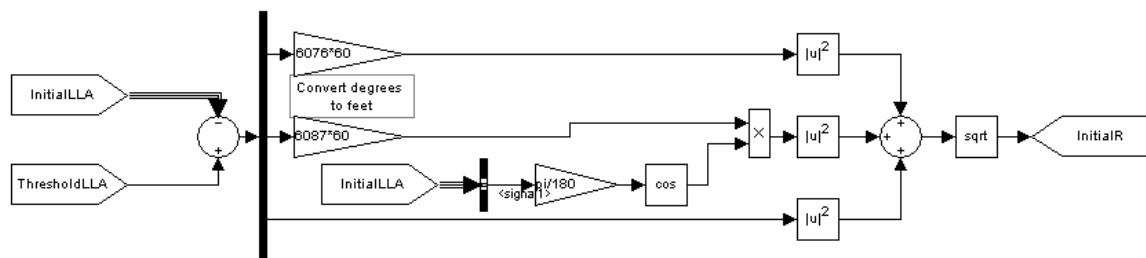




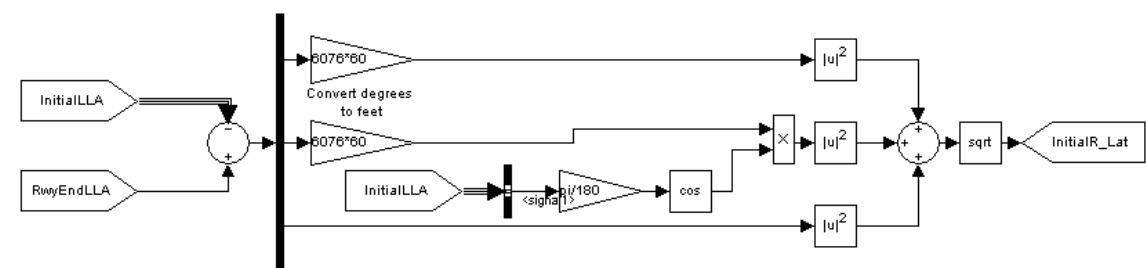
Range, calculation of runway far end LLA (for future directional course controller)



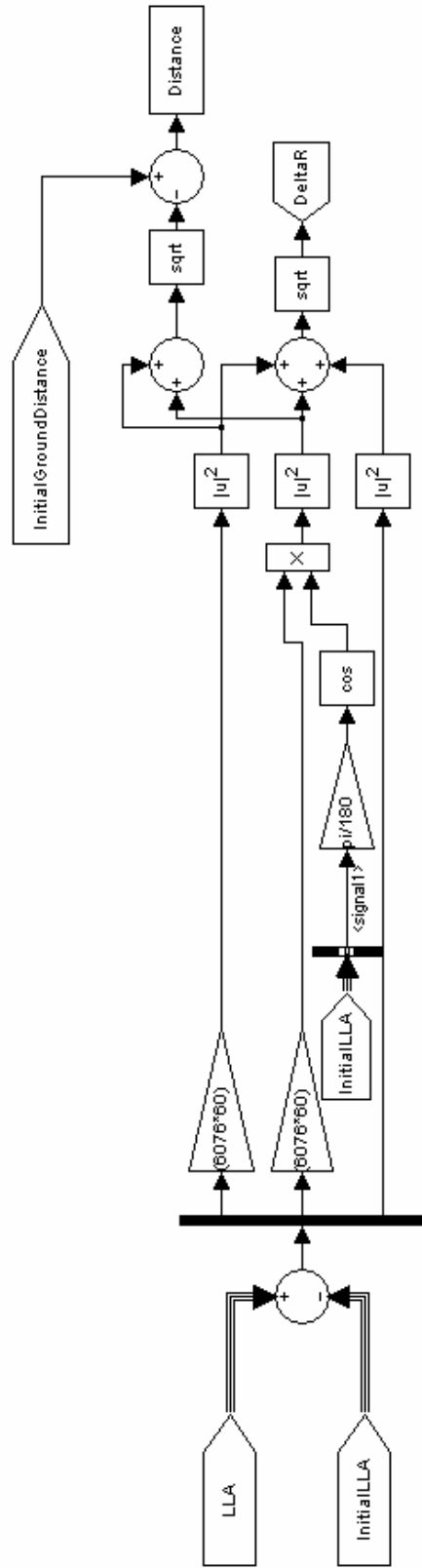
Range, calculation of initial range



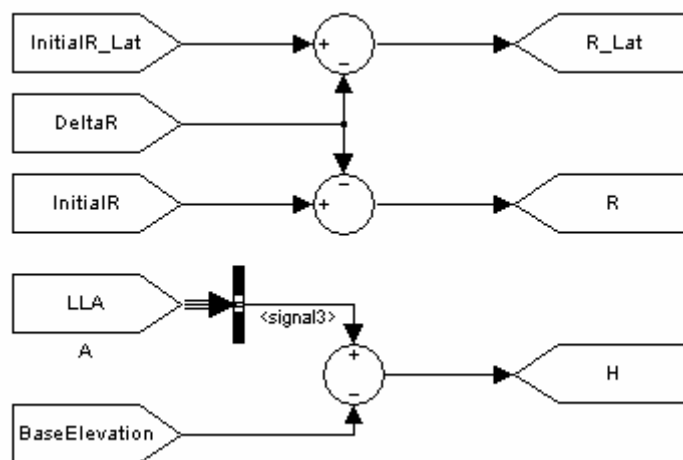
Range, calculation of initial range for directional course control



Range, calculation of change in range and ground distance traveled



Range, calculation of instantaneous range, range for directional course control (R\_Lat) and height above ground level.



## REFERENCES

---

<sup>1</sup> Google Calculator. “<http://www.google.com/help/features.html#calculator>.”

<sup>2</sup> Blakelock, John H. Automatic Control of Aircraft and Missiles. John Wiley and Sons, Inc. New York/London/Sydney 1965.

<sup>3</sup> Biezad, Daniel PhD. Project advisor and Professor, Aerospace Engineering Department, California Polytechnic State University, San Luis Obispo.

<sup>4</sup> Biezad, Daniel PhD. Study Guide for Flight Control Systems. Aerospace Engineering Department, California Polytechnic State University, San Luis Obispo. Winter quarter 2005.