**Flight Management System for Maintaining Steady Flight**

**of a Subsonic Commercial Jet**

**5 December 2014**

**Dietrich Klemm (dietdk)**

**Zach Meves (zmeves)**

***AEROSP 201: Fall 2014***

# Abstract

In today’s modern world, leading cause of aircraft accidents is loss of control. The primary reason control is lost is an attempt by the pilot to fly outside the steady flight envelope for the aircraft they are flying in. We have been tasked with developing a Flight Management System (FMS) that will keep the aircraft from flying outside of its flight envelope, which will lead to fewer accidents and greater flight safety. The FMS should be able to plan the trajectory of the aircraft given a three-dimensional point in space at a starting and ending position, as well as dictate pilot settings so that the aircraft stays within its flight envelope.

Our FMS has a fairly simple structure, with a main function that keeps track of flight pararmeters and calls on other functions when needed, so the code is compartmented and easier to understand. These other functions are what really run the FMS. They include functions that keep track of straight flight, climbing, descending, and rolling. The main function also calculates the flight evelope for each stage of the flight, and ouputs it so the pilot knows uder which conditions to fly.

We had limited time and resources for this project, so we made a few (qualified) assumptions that would speed development without greatly compromising the integrity of the FMS. The non-steady portions of flight between the steady portions are ignored because they have no great affect on airspeed, altitude, or bearing. This includes pitching and yawing. All defelctions of control surface are also considered to be instantaneous.

The FMS is based on a numerical integrator, which constantly recalculates values such as altitiude, thrust, and bearing as parameters of the aircraft change. As each of the spate steady flight, banking, rolling, climbing, and descending functions interate through, those values are being regularly updated to reflect the changing conditions of flight. These functions are explained in detail later in this report.

Combined together, all our functions form a very robust FMS. Not only able to perform up to the project specifications, it can also plot the paths of flights anywhere in the world, as long as they are within the plane’s range. The FMS we’ve developed is for a subsonic business jet, whose specifications will be detailed later.

# 

# Analysis

## Program Architecture

The FMS is a complex program, and as such it is split into several sub-sections. There is a driver script (*FMS.m*) which includes all of the aircraft parameters as well as all user inputs. This script calls multiple functions, each of which deals with a certain portion of flight, as needed. This abstraction is useful for maintaining the program’s flexibility and ease of operation.

In order to abstract the FMS functionality into sub-sections, several general maneuvers were identified. These include straight level flight, turning, climbing, and combined turns and climbs. The straight flight maneuver is represented by one function (*Straight.m*). Given that there can exist combined turns and climbs, these two maneuvers are combined into another FMS function (*Helix2.m*). There is an additional function (*BearingChange.m*) which can be used for turning – the only difference between this function and the aforementioned turn/climb function is the input array. This was done purely for software interface purposes and provides no additional functionality. In order to transition from level flight to a banked turn and vice versa, the aircraft must roll. There is a function (*Roll.m*) that provides this functionality.

The final major function is used to determine the flight envelope (*FlightEnvelope.m*). There are also several auxiliary functions which calculate useful numbers such as drag, air density, distance, and changes in latitude and longitude. All of the functions mentioned here will be discussed in further detail, and the source code is included in Appendix B.

The FMS is based upon a numerical integrator. All flight simulation is done using this integrator, allowing values to be constantly recalculated as certain variables change. In testing, this method has provided sufficient resolution (+/- a few feet altitude, +/- < 1 degree bearing) to provide useful and insightful results. Any adverse consequences of this integration method will be discussed in relevant sections.

## Initial Assumptions

Several maneuvers which aircraft undertake are not included in this FMS. These include pitching, yawing, takeoff, and landing. The inputs for the FMS include initial and final positions for which the altitude is assumed to be non-zero. Non-steady portions between steady flight can be ignored, and pitching and yawing are example of non-steady maneuvers. Pure yaw is rarely used in flight, and ignoring it will not significantly affect the FMS’s accuracy. The lack of pitch calculation will offset the true flight distances by a small amount, but will not significantly change the positions on the scale of the full flight. All control surface deflections are assumed to be instantaneous, which is a minor assumption to make. It only takes a few seconds for ailerons, rudders, or elevators to deflect, and ignoring this small time will not significantly impact the program’s accuracy. All of these assumptions comply with the scope of this project as described in the official specification.

## Variables for Analysis

For all subsequent analysis, the variables in the table below are used (unless otherwise denoted).

Table 1. Variable Descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| T | Thrust | C\_ | Coefficient of \_\_\_ (Lift, etc.) |
| D | Drag | cchord | Mean Aerodynamic Chord |
| W | Weight | K | Aircraft Property (constant) |
| L | Lift |  | Max Surface Thrust |
| φ | Bank Angle | ρ, ρs | Air Density, [at sea level] |
| γ | Flight Path Angle | Ψ | Gas Constant |
| α | Angle of Attack | τ | Temperature |
| R | Radius of Turn | m | Engine Parameter |
| v | Velocity | n | Load Factor |
| S | Wing Surface Area | Σ | Rolling Moment |
| M | Pitching Moment | N | Yawing Moment |
| B | Wing Span | c | Thrust Specific Fuel Consumption |
| δt | Throttle Setting | δe | Elevator Setting |
| δa | Aileron Setting | δr | Rudder Setting |
| λ | Longitude | µ | Latitude |
| θ | Bearing |  |  |

## Geospatial Coordinates and Preliminary Equations

The curvature of the earth affects the bearing at which a plane will fly. As it progresses along an apparent straight-line path, the bearing will actually change to keep the plane going in a circle tangent to the earth’s curvature. The initial bearing needed in order to reach a final latitude and longitude is given by the following equation1:

This equation is evaluated by *CalculateBearing.m*. In order to modify the latitude and longitude of the aircraft, a function called *CalculateLatLong.m* is invoked. Assuming small changes in the N/S and E/W directions, the Earth can be approximated as flat, and the bearing will not change by an appreciable amount. Using the arc length formula, s = rθ, the change in latitude or longitude can be calculated from a known *s* (the change in N/S or E/W direction) and *r* (the radius of the earth for latitude, or the distance from the earth’s rotational axis to the current position for longitude).

The final functions dealing with geospatial data are *CalculateDistance.m* and *CalculateDistanceNumerical.m*. These functions, which differ only in their parameter data types, calculate the distance between two coordinates. Using the spherical-Cartesian coordinate transformations below and the inner product definition, the total angle between the two positions can be found.

The distance between the two positions can be found using *s* = *R*earth ξ.

Air density is calculated using a combination of the following equations:

Once an atmospheric temperature gradient has been established, the pressure and then the density can be calculated for any given altitude. The functions *Calculate\_T\_P\_rho.m*, *Calculate\_rho.m*, and *Calculate\_rho\_vector.m* each perform these calculations (they differ only in their output data types).

## General Flight Conditions

The thrust available depends on altitude by the equation:

And so the maximum available thrust is known for all altitudes.

Lift, drag, rolling moment, pitching moment, and yawing moment are defined as follows:

(calculated by *Calc\_D.m*)

There are several conditions (and governing equations) for general steady flight. The most basic (that hold for all steady conditions) are as follows:

Assuming a small enough flight path angle that the small angle approximation holds. The coefficient of drag is given by the drag polar equation.

Using the drag polar equation as well as a general equation for thrust can be found. It is as follows:

In trimmed flight, the rolling, pitching, and yawing moments are zero, meaning their respective coefficients are zero. By setting the coefficients to zero, the following relationships are derived:

In order to change the position of the aircraft using the FMS’s numerical integrator, the displacement for a small time dt is calculated by the following equations:

The latitude and longitude are modified according to the methods previously described. The change in weight is calculated from the following equation:

Derivations of some intermediate steps are included in Appendix A.

## Flight Envelope

In order to ensure that the aircraft remains within a steady flight envelope, this envelope must be calculated for each portion of the flight. The function *FlightEnvelope.m* performs this calculation. It iterates through altitudes 0 – 80,000 feet, determining the air density at each level. For each of these altitudes, the function then iterates through the possible velocity values and determines what values are safe to fly at. The function returns the minimum and maximum safe velocity values for each altitude. In order to determine if the velocity is safe, the following equation is evaluated for each candidate velocity:

where Trequired is the general thrust equation (velocity dependent) derived previously, and Tavailable is the available thrust. If the conditional above is satisfied, then the velocity is safe to fly at. Note that the equation for Trequired includes both the climbing and turning terms (γ and φ) and so is valid for all steady flight scenarios.

## Steady Level Flight

For steady level flight, the general flight equations above are simplified greatly by noting that γ = φ = 0. The steady flight envelope is calculated for the weight at the start of the level flight section and for an empty weight. These are the boundary flight envelopes, and the FMS attempts to output settings that will satisfy both, and all in-between, envelopes.

The initial weight flight envelope is exclusively more restrictive than the empty weight envelope, and so satisfying the initial weight envelope will result in steady flight for the duration of the level flight segment. The target velocity is set to the average of the minimum and maximum safe velocities. This provides the largest margin for error (in case of unforeseen events, loss of fine control, etc.). The thrust required, and subsequently the thrust setting δt, are calculated using

For each time step, the FMS finds the velocity based on the throttle setting and current weight (using the above equation and solving numerically), and then it calculates the distance change, new latitude/longitude, and new weight for each time step *dt*. Recalculating the velocity and weight after each iteration allows for extremely accurate results. In order to simplify the program outputs, the angle of attack is assumed to be constant, even as the weight changes. The angle of attack will vary by only a small amount over the course of level flight, especially when compared to constantly varying values like velocity (which is an n2 relationship). Keeping α constant allows for a constant δe instead of a continuously varying function for the entire level flight phase.

The FMS will numerically iterate through small time steps *dt* until the target position is achieved.

## Steady Climb and Ascent/Descent

In steady climb or ascent/descent, either φ or γ or both are not equal to 0. For these maneuvers, the weight is held constant. These maneuvers are short enough that the change in weight due to fuel loss is very small compared to the total aircraft weight. The following analysis assumes a positive γ, as it is the more restrictive case. Steady flight for a negative γ will be satisfied by conditions for a positive γ (this has been confirmed by testing).

In order to ensure steady flight, the FMS calculates the required thrust to reach the target altitude at a flight path angle γ and bank angle φ. The most restrictive envelope will be at the maximum altitude, at the maximum possible φ and γ, and so satisfying this envelope will satisfy all others for this steady flight segment. The FMS calculates the thrust available to the aircraft at the target altitude using the equation stated previously for available thrust. It then calculates the thrust needed to fly at the target altitude with γ = γmax and φ = φmax using the equation below. γ­max is set to 10° and φmax is set to 33°. A γ of 10° is a reasonable flight path angle for a commercial jet, and a φ of 33° corresponds to a wing loading of ~ 1.2. A wing loading of 1.2 corresponds to a safety factor of at least 2 for standard commercial airframes.

The derivation of the T­needed equation is included in Appendix A. If T­needed is greater than Tavailable, the FMS lowers the bank angle to a minimum of 3° in 1° increments. If the bank angle reaches 3° and the thrust needed still has not been achieved, γ is lowered 1° at a time to a minimum of 1°. Once T­needed is less than Tavailable, γ and φ are set. The flight envelope for the final altitude with γ and φ is found, and the target velocity is set the midpoint between the minimum and maximum safe velocities. The δt is set using the general thrust equation and the same method as was described for level flight.

The FMS first performs a helical turn/climb, if needed, until either the target bearing or altitude is achieved. Then it performs a pure climb or pure turn until the remaining condition is satisfied. If a turn is needed, the FMS fist calls *Roll.m*, which rolls the aircraft to the requested bank angle. *Roll.m* will be called after the turn to restore φ to 0. It will be detailed in the next section.

For each time step *dt*, the FMS solves for the thrust based on δt and the current altitude, and then finds velocity using the same methods as for level flight. By recalculating the air density at each iteration, the correct thrust and velocity are calculated. The turning radius, if any, is calculated from the velocity and bank angle from the equation below. Based on the velocity, heading, γ, and φ, the FMS modifies the latitude, longitude, heading, and altitude by the appropriate values:

*dx* and *dy* (and latitude and longitude) are calculated the same way as for level flight. The FMS numerically integrates through small time steps *dt* until the target altitude or bearing is satisfied. The remaining process of a pure turn or climb is identical to the combined process, but without the already satisfied maneuver.

In order to simplify the program outputs, the angle of attack is assumed to be constant, even as the altitude changes. Similar to the level flight case, the angle of attack will vary by only a small amount over the course of a climb – velocity will differ much more as a result of the changing thrust. Keeping α constant allows for a constant δe instead of a continuously varying function for the entire level flight phase. This is a potential inaccuracy, but one that is very small and greatly simplifies the program outputs. The angle of attack is calculated at the beginning of the maneuver, and the elevator deflection is set at this point for the remainder of the maneuver (per the equation stated earlier).

## Rolling

Aileron and rudder deflections can cause a rolling or yawing moment to develop on the aircraft. For rolling, a rolling moment is desirable while the yawing moment is set to 0. This moment will cause an angular acceleration, related to the moment by the equation:

The moment of inertia *I* is approximated by treating the fuselage as a cylinder of the specified radius with a weight equal to the plane’s current weight. The aileron deflection is set to .3, simply because this is a modest deflection that will not result in any sudden unsteady rolling. It is also low enough to further justify the assumption that control surfaces deflect instantaneously. The rudder deflection is solved for via the equation:

The time it takes for the aircraft to roll to its target angle φf is:

# 

# Demonstration of Functionality

The following settings and plots were output as a result of the following inputs:

Table 2. Input Parameters

|  |  |  |  |
| --- | --- | --- | --- |
|  | Start | Waypoint | End |
| Coordinates | 42.2°N, 83.37°W, 35 ft | 41.06°N, 80.06°N, ? ft. | 40.71°N, 74.01°W, 3000 ft |
| Velocity | 1.15\*vstall @ 29° | ? | ? @ 120° |

*Table 3. Output Settings*

**Minutes Seconds Throttle Elevator Aileron Rudder Description**

0.0000 1.0000 0.2016 -0.0805 0.3000 -0.0141 *Begin Helix: Start Roll*

0.0000 1.4730 0.7556 -0.1155 0.0000 0.0000 *Begin Helix: End Roll*

0.0000 24.0730 0.7556 -0.1207 -0.3000 0.0141 *End Helix: Start Roll*

0.0000 24.5518 0.7556 -0.0847 0.0000 0.0000 *End Helix: End Roll*

8.0000 19.1518 0.3901 0.0399 0.0000 0.0000 *Cruise to Waypoint*

36.0000 30.1518 0.3827 0.0480 -0.3000 0.0141 *Begin Waypt. Turn: Start Roll*

36.0000 30.3964 0.4054 0.0386 0.0000 0.0000 *Begin Waypt. Turn: End Roll*

36.0000 44.4264 0.4054 0.0372 0.3000 -0.0141 *Exit Waypt. Turn: Start Roll*

36.0000 44.6739 0.4054 0.0468 0.0000 0.0000 *Exit Waypt. Turn: End Roll*

36.0000 45.6739 0.3663 0.0447 0.0000 0.0000 *Cruise to Final*

94.0000 18.6739 0.3592 0.0595 0.3000 -0.0141 *Begin Final Descent: Start Roll*

94.0000 18.8861 0.2691 0.0525 0.0000 0.0000 *Begin Final Descent: End Roll*

94.0000 19.8861 0.2691 0.0525 -0.3000 0.0141 *End Final Descent: Start Roll*

94.0000 20.0983 0.2691 0.0595 0.0000 0.0000 *End Final Descent: End Roll*

101.0000 42.0983 0.3523 0.0753 0.0000 0.0000 *Approach*

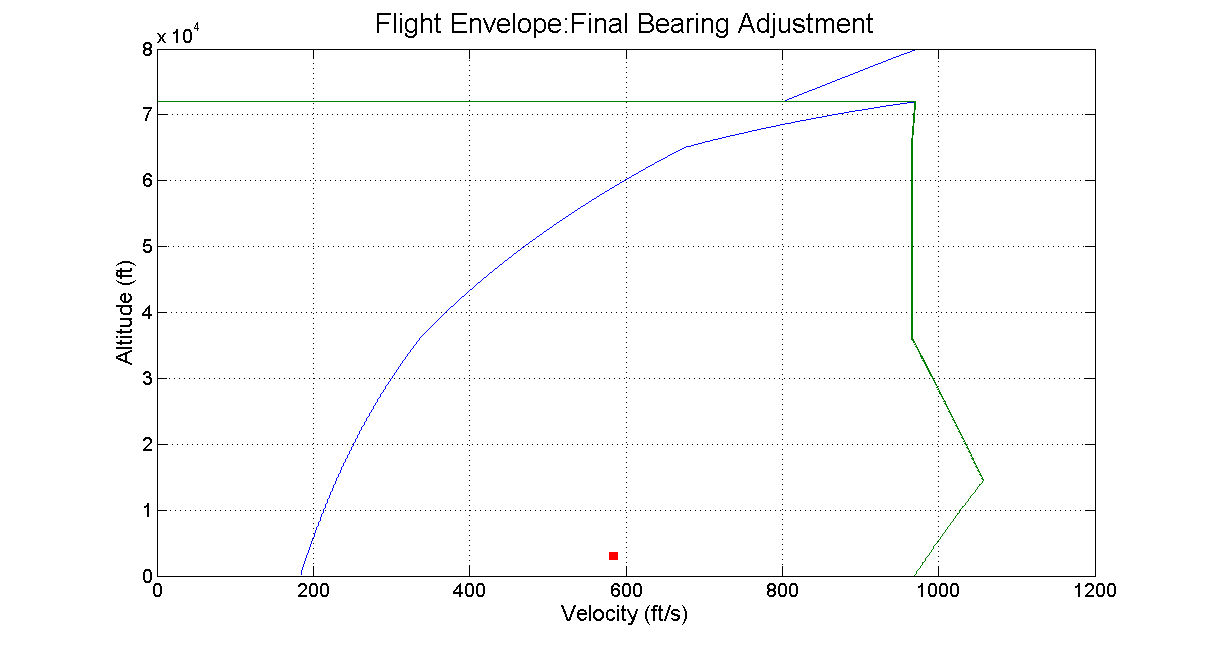
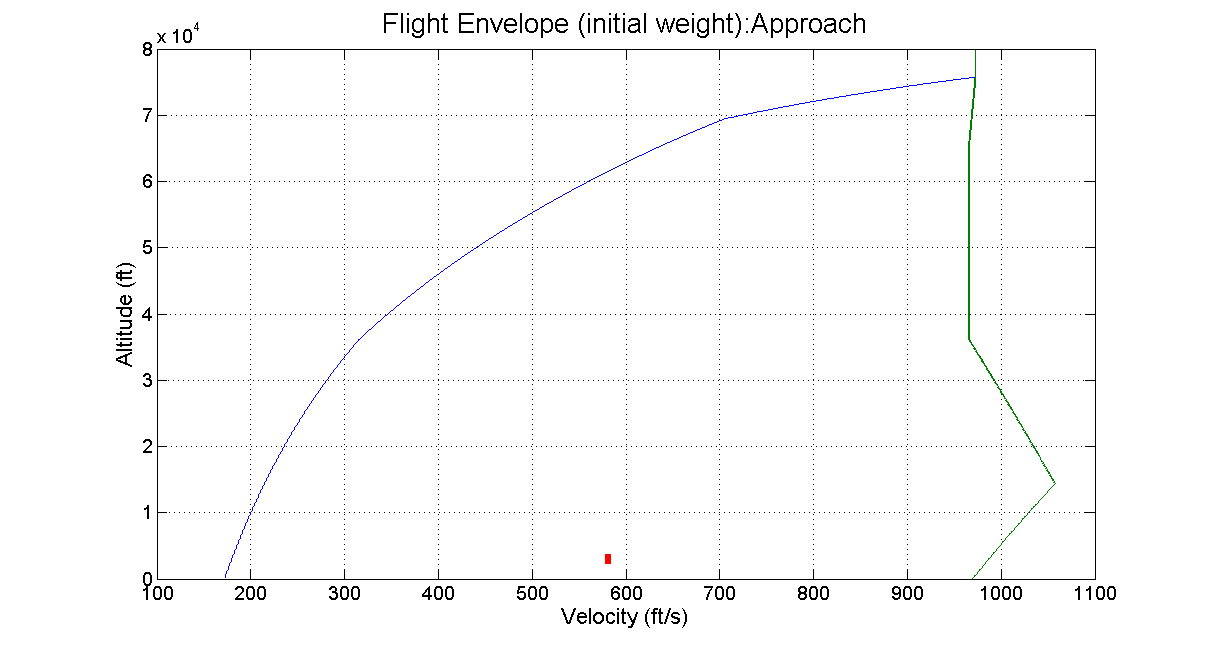
115.0000 45.0983 0.3461 0.0775 -0.3000 0.0141 *Begin ΔBearing: Start Roll*

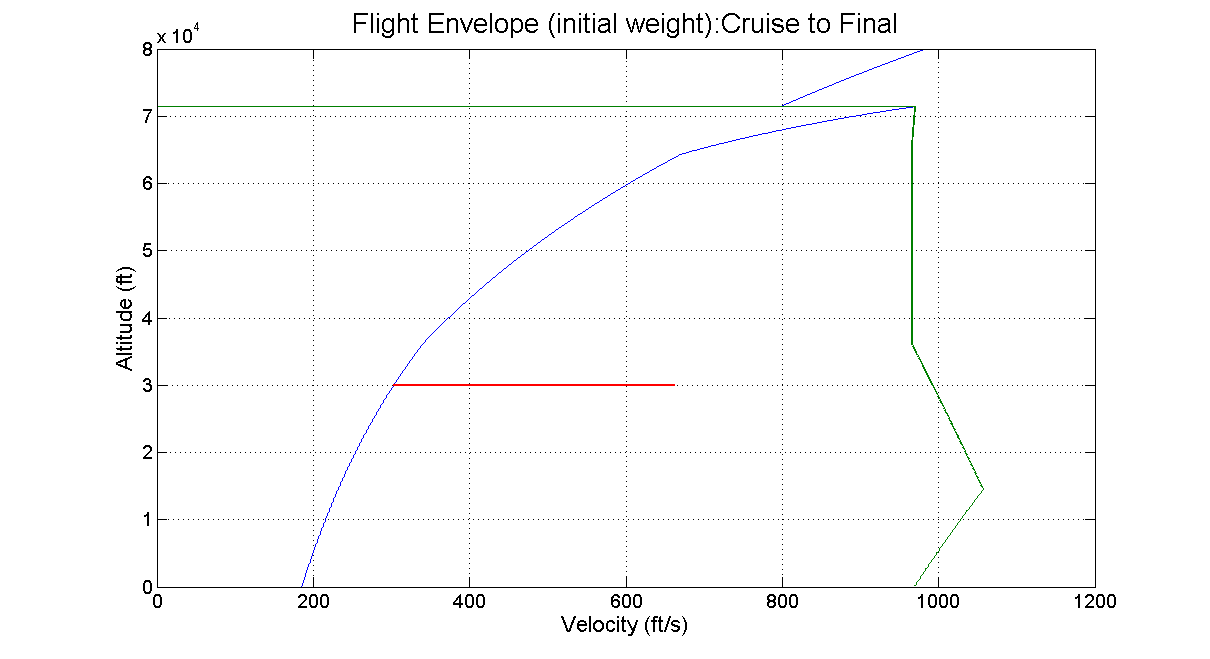
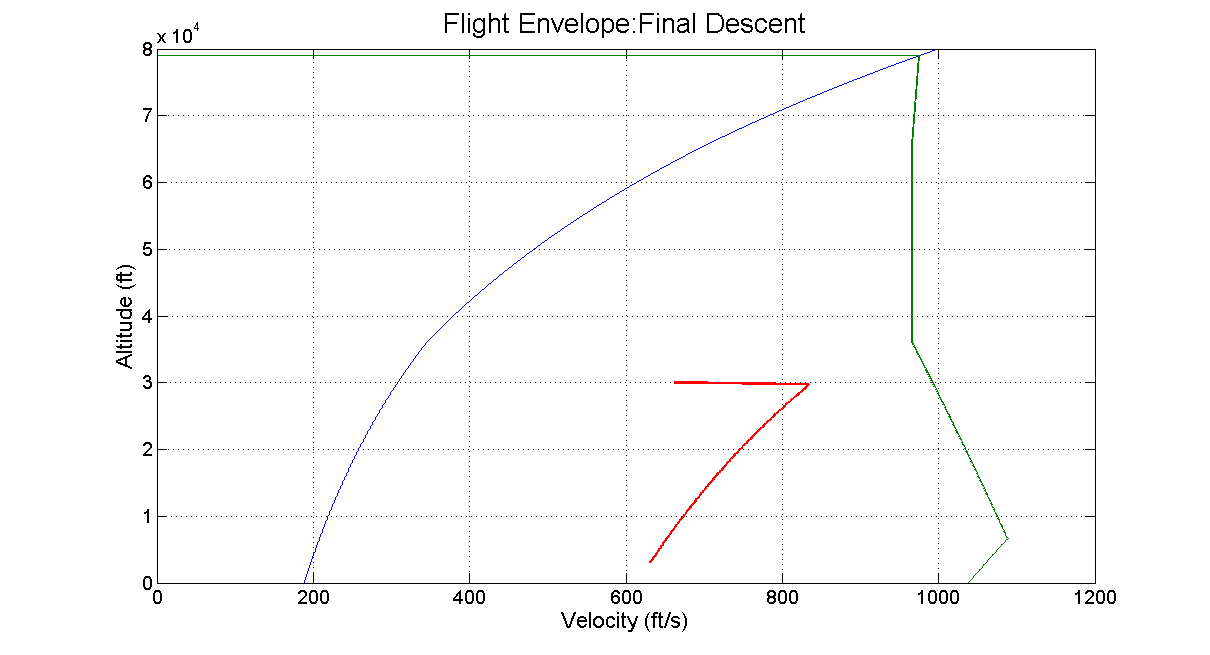
115.0000 45.2467 0.3559 0.0741 0.0000 0.0000 *Begin ΔBearing: End Roll*

115.0000 52.3717 0.3559 0.0745 0.3000 -0.0141 *End ΔBearing: Start Roll*

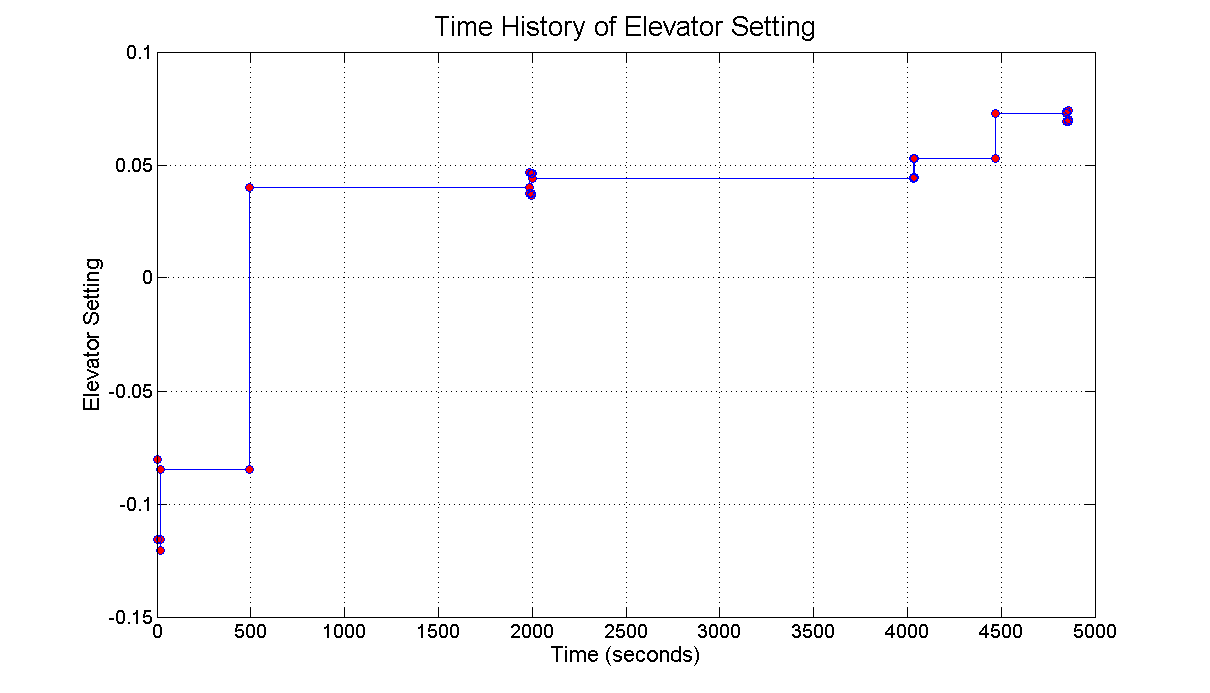
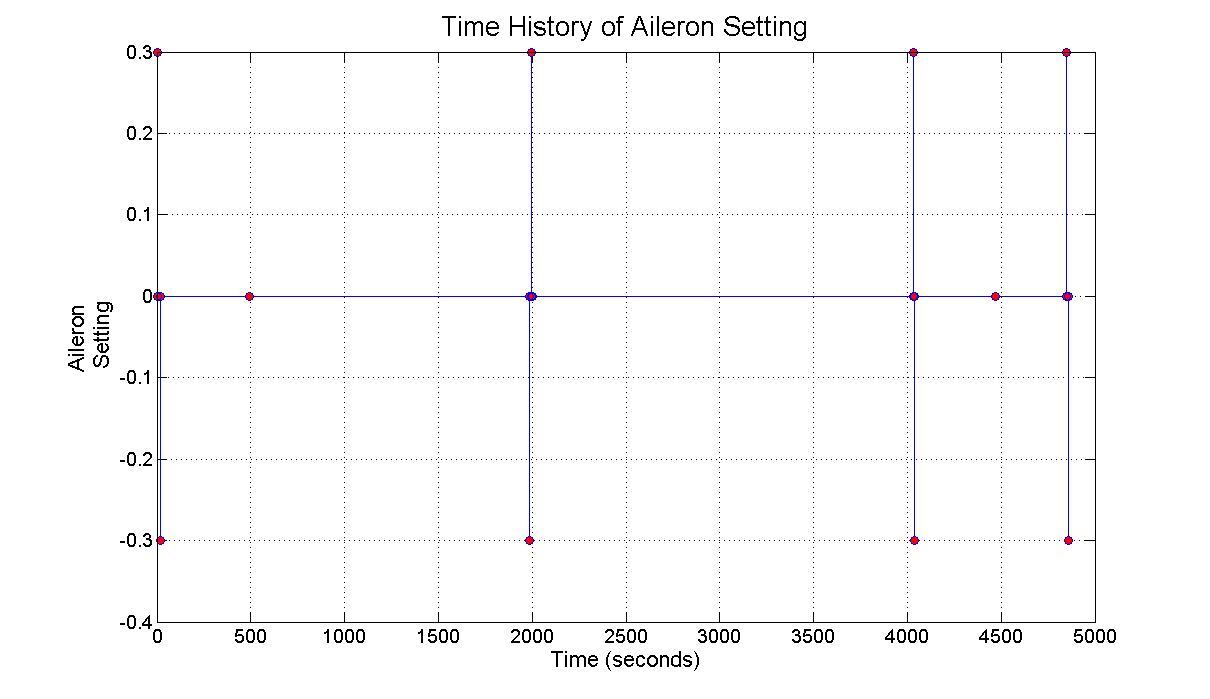
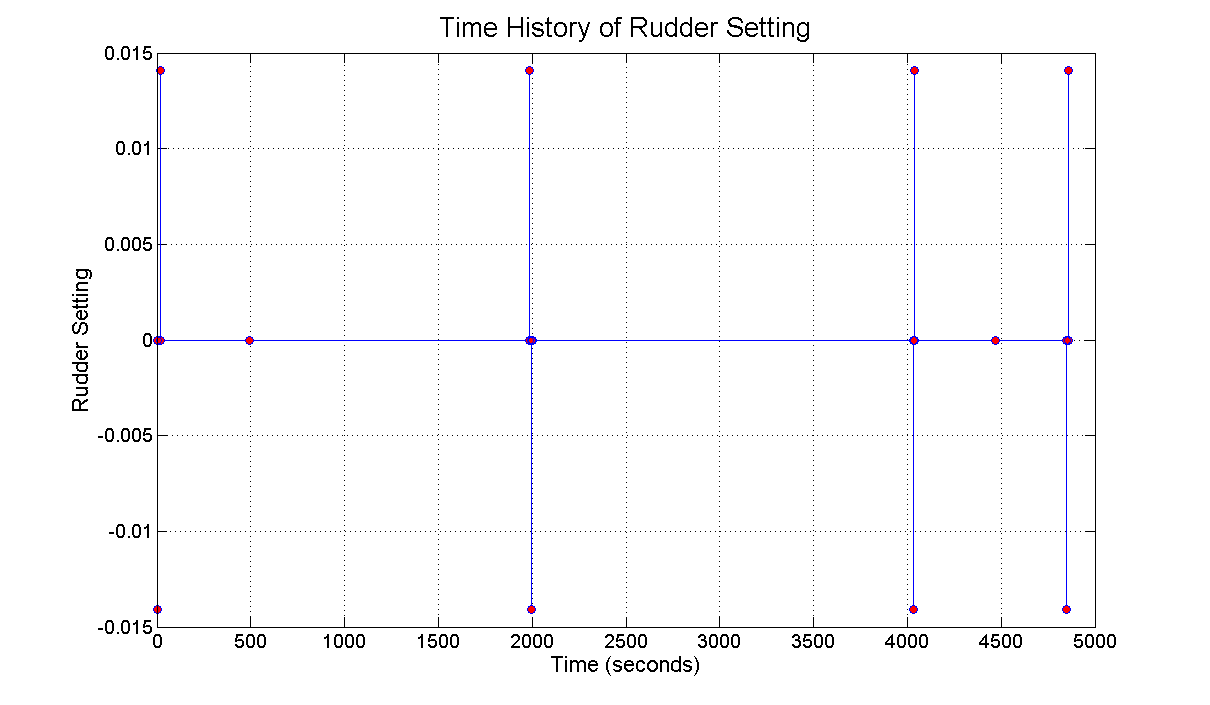
115.0000 52.5189 0.3559 0.0778 0.0000 0.0000 *End ΔBearing: End Roll*

# Flight Envelopes



In all of the steady flight phases, the aircraft (shown in red) stays within the steady flight envelope.

# Plots



# Conclusions

The FMS we’ve developed achieves the objectives detailed in the project spec, and more. The FMS is also very pilot oriented, and with more time and research it could be made even more user-friendly. As it is right now, the FMS lets the pilot keep his or her controls constant throughout a maneuver, so he or she only has to adjust the throttle and control surface deflections during the transition between maneuvers.

The FMS outputs exactly what the deflections and throttle setting should be and when, so all the pilot has to do is adjust his or her controls at the proper times to stay within the airplane’s flight evelope. The FMS also output the various flight evelopes for the various flight conditions, so if the pilot does have to deviate from the program he or she knows what conditions to stay within in order to keep the plane and passengers safe. It also outputs a flight path so the pilot knows exactly where he or she is going, along with graphs of all the deflections and throttle settings versus time so the pilot knows exaclt when to make control changes.

The FMS is also robust. It works perfectly for the given starting location, waypoint, and destination, and after further testing, it is confirmed that it will work for any given starting location, waypoint, and destination as long as they are within the airplane’s range. With more funding and attention, the FMS we’ve developed could be improved and upgraded. This will include functionality for pitching and yawing, as well as not-instananteous control surface deflections. The FMS can also potentially be adapted for other aircraft. Our future research will focus on these areas and we’ll continue you to innovate in them.

# References

1 Veness, Chris (2014). Calculate distance, bearing, and more between Latitude/Longitude points. *Moveable Type Scripts.* [www.moveable-type.co.uk/scripts/latlong.html](http://www.moveable-type.co.uk/scripts/latlong.html)

# Appendix A: Detailed Calculations

# Appendix B: FMS Code

%FMS.m

%Zach Meves and Dieter Klemm

%All units are imperial

%AIRCRAFT\_PARAMETERS

W\_0 = 13000; %lbs, including all fuel

W\_empty = 8000; %lbs

W\_fuel = 5000; %lbs

CL\_max = 1.24;

CL\_0 = .0835;

CL\_alpha = 5.16;

CL\_de = .43;

CD\_0 = .023;

K = .073;

CM\_0 = .0895;

CM\_alpha = -1.09;

CM\_de = -1.13;

T\_max\_surface = 2\*3000; %lbs

c = 2\*1.18;% 1b/hr

m = .6;

C\_roll\_da = .05;

C\_roll\_dr = -.019;

C\_yaw\_da = .007;

C\_yaw\_dr = .149;

S = 232; %ft^2

%%% USER INPUT BELOW %%%

%INPUTS

LATITUDE\_IN = '42.20\_N';

LONGITUDE\_IN = '83.37\_W';

[LATITUDE\_IN\_NUM, LONGITUDE\_IN\_NUM] = ConvertString(LATITUDE\_IN, LONGITUDE\_IN);

ALTITUDE\_IN = 35; %ft

HEADING\_IN = 29; %degrees

v\_stall = sqrt(2\*W\_0./Calculate\_rho(ALTITUDE\_IN)./CL\_max./S); %ft/s

SPEED\_IN = 1.15.\*v\_stall;

LATITUDE\_OUT = '40.71\_N';

LONGITUDE\_OUT = '74.01\_W';

[LATITUDE\_OUT\_NUM, LONGITUDE\_OUT\_NUM] = ConvertString(LATITUDE\_OUT, LONGITUDE\_OUT);

ALTITUDE\_OUT = 3000; %ft

HEADING\_OUT = 120; %degrees

LATITUDE\_WAY = '41.06\_N';

LONGITUDE\_WAY = '80.06\_W';

[LATITUDE\_WAY\_NUM, LONGITUDE\_WAY\_NUM] = ConvertString(LATITUDE\_WAY, LONGITUDE\_WAY);

DESCEND\_DIST = 50\*5280; %50 miles

%CONSTANTS

da = 0;

dr = 0;

cruise\_altitude = 30000; %ft

%%% Solve for thrust and elevator settings for flight segments %%%

W = W\_0;

heading = HEADING\_IN;

t = 0; %time elapsed in seconds

distance1 = CalculateDistance(LATITUDE\_IN, LONGITUDE\_IN, LATITUDE\_WAY, LONGITUDE\_WAY);

distance2 = CalculateDistance(LATITUDE\_WAY, LONGITUDE\_WAY, LATITUDE\_OUT, LONGITUDE\_OUT);

Total\_distance = distance1 + distance2;

t\_max\_estimate = ceil(Total\_distance/SPEED\_IN\*2); %seconds (double the shortest possible time)

t\_arr = 1:t\_max\_estimate;

stagings\_arr = ones(1, 10); %up to 10 flight sequences

FlightSegment = 1; % before or after the waypoint

% 1 = before

% 2 = after

%% Logic: reach target altitude, turn to correct path, fly over waypoint

if(FlightSegment == 1) %going to waypoint

%Helix turn from takeoff

[lat, long, alt, head, speed, time, t\_marks, s\_marks, throttle, elev, ail, rud] = Helix2(LATITUDE\_IN\_NUM, LONGITUDE\_IN\_NUM, ALTITUDE\_IN, HEADING\_IN, SPEED\_IN, ...

LATITUDE\_WAY\_NUM, LONGITUDE\_WAY\_NUM, cruise\_altitude, W, 'First Helix Up');

lat\_arr = lat;

long\_arr = long;

alt\_arr = alt;

heading = head;

speed\_arr = speed;

time\_arr = time;

timestamps = t\_marks;

stringstamps = s\_marks;

delta\_t\_arr = throttle;

delta\_a\_arr = ail;

delta\_e\_arr = elev;

delta\_r\_arr = rud;

if(lat\_arr(end) ~= LATITUDE\_WAY\_NUM || long\_arr(end) ~= LONGITUDE\_WAY\_NUM)

%Straight to Waypoint

[lat, long, alt, head, speed, time, t\_marks, s\_marks, throttle, elev, ail, rud, W\_new] = Straight(lat\_arr(end), long\_arr(end), alt\_arr(end), heading(end), speed\_arr(end), ...LATITUDE\_WAY\_NUM, LONGITUDE\_WAY\_NUM, 0, W, 'Cruise to Waypoint');

lat\_arr = horzcat(lat\_arr, lat);

long\_arr = horzcat(long\_arr, long);

alt\_arr = horzcat(alt\_arr, alt);

heading = horzcat(heading, head);

speed\_arr = horzcat(speed\_arr, speed);

tau = time\_arr(end);

time\_arr = horzcat(time\_arr, tau + time);

timestamps = horzcat(timestamps, tau + t\_marks);

stringstamps = horzcat(stringstamps, s\_marks);

delta\_t\_arr = horzcat(delta\_t\_arr, throttle);

delta\_a\_arr = horzcat(delta\_a\_arr, ail);

delta\_e\_arr = horzcat(delta\_e\_arr, elev);

delta\_r\_arr = horzcat(delta\_r\_arr, rud);

W = W\_new;

end

%Turn at waypoint

[lat, long, alt, head, speed, time, t\_marks, s\_marks, throttle, elev, ail, rud] = Helix2(lat\_arr(end), long\_arr(end), alt\_arr(end), heading(end), speed\_arr(end), ...

LATITUDE\_OUT\_NUM, LONGITUDE\_OUT\_NUM, alt\_arr(end), W, 'Waypoint Turn');

lat\_arr = horzcat(lat\_arr, lat);

long\_arr = horzcat(long\_arr, long);

alt\_arr = horzcat(alt\_arr, alt);

heading = horzcat(heading, head);

speed\_arr = horzcat(speed\_arr, speed);

tau = time\_arr(end);

time\_arr = horzcat(time\_arr, tau + time);

timestamps = horzcat(timestamps, tau + t\_marks);

stringstamps = horzcat(stringstamps, s\_marks);

delta\_t\_arr = horzcat(delta\_t\_arr, throttle);

delta\_a\_arr = horzcat(delta\_a\_arr, ail);

delta\_e\_arr = horzcat(delta\_e\_arr, elev);

delta\_r\_arr = horzcat(delta\_r\_arr, rud);

FlightSegment = 2;

end

%% Logic: correct heading to offset point, fly at set altidude, ...

if(FlightSegment == 2) %going to final destination

%Straight to final destination

[lat, long, alt, head, speed, time, t\_marks, s\_marks, throttle, elev, ail, rud, W\_new] = Straight(lat\_arr(end), long\_arr(end), alt\_arr(end), heading(end), speed\_arr(end), ...

LATITUDE\_OUT\_NUM, LONGITUDE\_OUT\_NUM, DESCEND\_DIST, W, 'Cruise to Final');

lat\_arr = horzcat(lat\_arr, lat);

long\_arr = horzcat(long\_arr, long);

alt\_arr = horzcat(alt\_arr, alt);

heading = horzcat(heading, head);

speed\_arr = horzcat(speed\_arr, speed);

tau = time\_arr(end);

time\_arr = horzcat(time\_arr, tau + time);

timestamps = horzcat(timestamps, tau + t\_marks);

stringstamps = horzcat(stringstamps, s\_marks);

delta\_t\_arr = horzcat(delta\_t\_arr, throttle);

delta\_a\_arr = horzcat(delta\_a\_arr, ail);

delta\_e\_arr = horzcat(delta\_e\_arr, elev);

delta\_r\_arr = horzcat(delta\_r\_arr, rud);

W = W\_new;

% %Descend when out 10 miles

[lat, long, alt, head, speed, time, t\_marks, s\_marks, throttle, elev, ail, rud] = Helix2(lat\_arr(end), long\_arr(end), alt\_arr(end), heading(end), speed\_arr(end), ...

LATITUDE\_OUT\_NUM, LONGITUDE\_OUT\_NUM, ALTITUDE\_OUT, W, 'Final Descent');

lat\_arr = horzcat(lat\_arr, lat);

long\_arr = horzcat(long\_arr, long);

alt\_arr = horzcat(alt\_arr, alt);

heading = horzcat(heading, head);

speed\_arr = horzcat(speed\_arr, speed);

tau = time\_arr(end);

time\_arr = horzcat(time\_arr, tau + time);

timestamps = horzcat(timestamps, tau + t\_marks);

stringstamps = horzcat(stringstamps, s\_marks);

delta\_t\_arr = horzcat(delta\_t\_arr, throttle);

delta\_a\_arr = horzcat(delta\_a\_arr, ail);

delta\_e\_arr = horzcat(delta\_e\_arr, elev);

delta\_r\_arr = horzcat(delta\_r\_arr, rud);

% Straight to final destination

[lat, long, alt, head, speed, time, t\_marks, s\_marks, throttle, elev, ail, rud, W\_new] = Straight(lat\_arr(end), long\_arr(end), alt\_arr(end), heading(end), speed\_arr(end), ...

LATITUDE\_OUT\_NUM, LONGITUDE\_OUT\_NUM, 0, W, 'Approach');

lat\_arr = horzcat(lat\_arr, lat);

long\_arr = horzcat(long\_arr, long);

alt\_arr = horzcat(alt\_arr, alt);

heading = horzcat(heading, head);

speed\_arr = horzcat(speed\_arr, speed);

tau = time\_arr(end);

time\_arr = horzcat(time\_arr, tau + time);

timestamps = horzcat(timestamps, tau + t\_marks);

stringstamps = horzcat(stringstamps, s\_marks);

delta\_t\_arr = horzcat(delta\_t\_arr, throttle);

delta\_a\_arr = horzcat(delta\_a\_arr, ail);

delta\_e\_arr = horzcat(delta\_e\_arr, elev);

delta\_r\_arr = horzcat(delta\_r\_arr, rud);

W = W\_new;

% Adjust bearing at destination

[lat, long, alt, head, speed, time, t\_marks, s\_marks, throttle, elev, ail, rud] = BearingChange(lat\_arr(end), long\_arr(end), alt\_arr(end), heading(end), speed\_arr(end), ...

HEADING\_OUT, ALTITUDE\_OUT, W, 'Final Bearing Adjustment');

lat\_arr = horzcat(lat\_arr, lat);

long\_arr = horzcat(long\_arr, long);

alt\_arr = horzcat(alt\_arr, alt);

heading = horzcat(heading, head);

speed\_arr = horzcat(speed\_arr, speed);

tau = time\_arr(end);

time\_arr = horzcat(time\_arr, tau + time);

timestamps = horzcat(timestamps, tau + t\_marks);

stringstamps = horzcat(stringstamps, s\_marks);

delta\_t\_arr = horzcat(delta\_t\_arr, throttle);

delta\_a\_arr = horzcat(delta\_a\_arr, ail);

delta\_e\_arr = horzcat(delta\_e\_arr, elev);

delta\_r\_arr = horzcat(delta\_r\_arr, rud);

end

%% Plot results:

figure

handle = usamap('conus');

states = shaperead('usastatelo', 'UseGeoCoords', true);

geoshow(handle, states)

geoshow(handle, lat\_arr, long\_arr)

figure

latlim = [min([LATITUDE\_IN\_NUM, LATITUDE\_OUT\_NUM, LATITUDE\_WAY\_NUM]) - 1, ... max([LATITUDE\_IN\_NUM, LATITUDE\_OUT\_NUM, LATITUDE\_WAY\_NUM]) + 1];

longlim = [min([LONGITUDE\_IN\_NUM, LONGITUDE\_OUT\_NUM, LONGITUDE\_WAY\_NUM]) - 2, ...

max([LONGITUDE\_IN\_NUM, LONGITUDE\_OUT\_NUM, LONGITUDE\_WAY\_NUM]) + 2];

%won't work if ]cross international date line, code below fixes it

if(abs(longlim(1) - longlim(2)) > 180)

temp = longlim;

longlim(1) = temp(2);

longlim(2) = temp(1);

end

handle2 = worldmap(latlim, longlim);

geoshow(handle2, 'landareas.shp')

geoshow(handle2, lat\_arr, long\_arr);

figure

plot3(long\_arr, lat\_arr, alt\_arr)

title('3D Projection of Flight Path')

xlabel('Longitude')

ylabel('Latitude')

zlabel('Altitude')

figure

plot(time\_arr, alt\_arr)

title('Time History of Altitude');

xlabel('Time (seconds)')

ylabel('Altitude (ft)')

timestamps\_minutes = degrees2dm(timestamps./60);

stringstamps = char(stringstamps);

fid = fopen('Outputs.txt', 'w');

if(fid ~= -1)

fprintf(fid, 'Minutes Seconds Throttle Elevator Aileron Rudder Description \n');

for i = 1:length(delta\_t\_arr)

fprintf(fid, '%-10.4f %-10.4f %-10.4f %-10.4f %-10.4f %-10.4f %-25s \n', ...

timestamps\_minutes(i,1), timestamps\_minutes(i,2), delta\_t\_arr(i), ...

delta\_e\_arr(i), delta\_a\_arr(i), delta\_r\_arr(i), stringstamps(i,:));

end

end

fclose(fid);

time\_plot = 0:length(time\_arr);

figure

stairs(timestamps, delta\_t\_arr, 'Marker', 'o', 'MarkerFaceColor', 'r')

title('Time History of Throttle Setting');

xlabel('Time (seconds)')

ylabel('Throttle Setting')

figure

stairs(timestamps, delta\_e\_arr, 'Marker', 'o', 'MarkerFaceColor', 'r')

title('Time History of Elevator Setting');

xlabel('Time (seconds)')

ylabel('Elevator Setting')

figure

stairs(timestamps, delta\_a\_arr, 'Marker', 'o', 'MarkerFaceColor', 'r')

title('Time History of Aileron Setting');

xlabel('Time (seconds)')

ylabel('Ailer Setting')

figure

stairs(timestamps, delta\_r\_arr, 'Marker', 'o', 'MarkerFaceColor', 'r')

title('Time History of Rudder Setting');

xlabel('Time (seconds)')

ylabel('Rudder Setting')

%END FMS.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%Helix.m

%Zach Meves and Dieter Klemm

%Performs steady climb, banked turn, or both between two positions

function [lat\_arr, long\_arr, alt\_arr, head\_arr, speed\_arr, t\_arr, time\_marks, string\_marks, throttle\_arr, elev\_arr, ail\_arr, rud\_arr] = Helix2(latIN, ...

longIN, altIN, headIN, speedIN, latOUT, longOUT, altOUT, W, title\_)

S = 232;

W\_0 = 13000; %lbs, including all fuel

W\_empty = 8000; %lbs

W\_fuel = 5000; %lbs

CL\_max = 1.24;

CL\_0 = .0835;

CL\_alpha = 5.16;

CL\_de = .43;

CD\_0 = .023;

K = .073;

CM\_0 = .0895;

CM\_alpha = -1.09;

CM\_de = -1.13;

T\_max\_surface = 2\*3000; %lbs

c = 2\*1.18;% 1b/hr

m = .6;

rho\_surface = .0023769; %slug/ft^3

C\_roll\_da = .05;

C\_roll\_dr = -.019;

C\_yaw\_da = .007;

C\_yaw\_dr = .149;

phi\_max = 33.557; %degrees, load factor of 1.2

rho = Calculate\_rho(altOUT) %slugs/ft^3

T\_max = T\_max\_surface.\*(rho./rho\_surface).^m %lbs

final\_head\_init = CalculateBearing(latIN, longIN, latOUT, longOUT); % ~ final heading

diff\_head\_init = final\_head\_init - headIN;

diff\_alt = altOUT - altIN;

diff\_head = diff\_head\_init;

phi = phi\_max;

turnRight = 1;

if(diff\_head\_init == 0)

R = 0;

phi = 0;

turnRight = 0;

end

if(diff\_head\_init > 0)

turnRight = 1;

if(diff\_head\_init > 180)

turnRight = -1;

end

else

turnRight = -1;

if(diff\_head\_init < -180)

turnRight = 1;

end

end

y = 5; %degrees

if(diff\_alt == 0)

y = 0;

end

if(diff\_alt < 0)

y = -1\*y;

end

T\_needed = W.\*(CD\_0./((cosd(phi)).^2).\*sqrt(K./CD\_0) + K.\*sqrt(CD\_0./K) + deg2rad(y)) %lb

ceiling\_rho = (T\_needed./T\_max\_surface).^(1./m).\*rho\_surface %slug/ft^3

while(ceiling\_rho > rho && phi >= 3) %while can't climb that high with phi and y

phi = phi - 2;

T\_needed = W.\*(CD\_0./((cosd(phi)).^2).\*sqrt(K./CD\_0) + K.\*sqrt(CD\_0./K) + deg2rad(y)) %lb

ceiling\_rho = (T\_needed/T\_max\_surface)^(1/m)\*rho\_surface; %slug/ft^3

end

while(ceiling\_rho > rho && y > 1) %while can't climb that high with phi and y

y = y - 1;

T\_needed = W.\*(CD\_0./((cosd(phi)).^2).\*sqrt(K./CD\_0) + K.\*sqrt(CD\_0./K) + deg2rad(y)) %lb

ceiling\_rho = (T\_needed/T\_max\_surface)^(1/m)\*rho\_surface; %slug/ft^3

end

T\_needed

T\_avail = T\_max\_surface.\*(rho./rho\_surface).^m

Throttle = T\_needed/T\_avail

phi

y %disp in degrees

y = deg2rad(y);

disp('Calculating Flight Envelope for Helix');

[H, v\_min, v\_max, a, stall] = FlightEnvelope(W, deg2rad(phi), y);

indexes = find(H < altOUT);

speedLOW = v\_min(indexes(end) + 1);

speedHIGH = v\_max(indexes(end) + 1);

speed\_optimal = .5\*(speedLOW + speedHIGH);

T\_needed = Calc\_D(rho, speed\_optimal, W, deg2rad(phi)) + W.\*y;

Throttle = T\_needed/T\_avail

if(diff\_head\_init ~= 0)

R = abs(speed\_optimal.^2/32.2/tand(phi)) %ft

end

head(1) = headIN;

alt\_arr(1) = altIN;

lat\_arr(1) = latIN;

long\_arr(1) = longIN;

speed\_arr(1) = speedIN;

T\_current = Calc\_D(Calculate\_rho(altIN), speedIN, W, 0);

T\_current\_avail = T\_max\_surface.\*(Calculate\_rho(altIN)./rho\_surface).^m;

time\_marks(1) = 1;

string\_marks(1) = cellstr(title\_);

throttle\_arr(1) = Throttle;

alpha\_ = (2.\*W./(cosd(phi).\*rho.\*speed\_optimal.^2.\*S) - CL\_0)./CL\_alpha;

elev\_arr(1) = (-1.\*CM\_0 - CM\_alpha.\*alpha\_)./CM\_de;

ail\_arr(1) = 0;

rud\_arr(1) = 0;

t = 1;

t\_arr(1) = 1;

%%% Roll to phi here, if needed %%%

%%% Assume rolling doesn't change lat or long by appreciable amount

if(phi ~= 0)

[t\_arr, time\_marks, string\_marks, elev\_arr, ail\_arr, rud\_arr] = Roll(alt\_arr(1), ...

speedIN, 0, deg2rad(phi), W, turnRight, strcat(title\_, ' enter'));

t = 2; %spot in t\_arr

head(1:t) = head(1);

alt\_arr(1:t) = alt\_arr(1);

lat\_arr(1:t) = lat\_arr(1);

long\_arr(1:t) = long\_arr(1);

speed\_arr(1:t) = speedIN;

throttle\_arr(1) = T\_current/T\_current\_avail;

throttle\_arr(2) = Throttle;

alpha\_ = (2.\*W./(cosd(phi).\*rho.\*speed\_optimal.^2.\*S) - CL\_0)./CL\_alpha;

end

dt = 1;

keep\_climb = 1;

keep\_turn = 1;

while(abs(diff\_alt) > 500 && abs(diff\_head) > 5 && keep\_climb == 1 && keep\_turn == 1)

% disp('first while loop');

rho = Calculate\_rho(alt\_arr(t));

% T = W.\*y + Calc\_D(rho, speed, W, deg2rad(phi));

T = Throttle.\*T\_max\_surface.\*(rho./rho\_surface).^m;

found = 0;

indexes = find(H < alt\_arr(t));

speedLOW = v\_min(indexes(end) + 1);

speedHIGH = v\_max(indexes(end) + 1);

speed\_test = speedLOW;

while(~found)

T\_test = Calc\_D(rho, speed\_test, W, phi) + W.\*y;

if(T\_test > .98\*T && T\_test < 1.02\*T)

speed = speed\_test;

found = 1;

else

speed\_test = speed\_test + 1;

end

if(speed\_test > speedHIGH)

disp('You messed up');

end

end

R = abs(speed.^2/32.2/tand(phi))

arcLen = speed\*dt; %ft

if(R ~= 0)

d\_bearing = arcLen/R\*turnRight; %radians

headIN = headIN + rad2deg(d\_bearing);

end

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

t = t + 1;

t\_arr(t) = t\_arr(t-1) + dt;

head(t) = headIN;

speed\_arr(t) = speed;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

dH = double(speed.\*y.\*dt); %ft

alt\_arr(t) = alt\_arr(t-1) + dH;

diff\_alt = altOUT - alt\_arr(t);

diff\_head = final\_head\_init - head(t);

if(sign(diff\_alt) ~= sign(y))

keep\_climb = 0;

end

if(sign(diff\_head) ~= sign(diff\_head\_init) || abs(diff\_head) < .01)

keep\_turn = 0;

end

end

dt = .005; %refined integration

while(abs(diff\_alt) > 1 && abs(diff\_head > .01) && keep\_climb == 1 && keep\_turn == 1)

rho = Calculate\_rho(alt\_arr(t));

% T = W.\*y + Calc\_D(rho, speed, W, deg2rad(phi));

T = Throttle.\*T\_max\_surface.\*(rho./rho\_surface).^m;

found = 0;

indexes = find(H < alt\_arr(t));

speedLOW = v\_min(indexes(end) + 1);

speedHIGH = v\_max(indexes(end) + 1);

speed\_test = speedLOW;

while(~found)

T\_test = Calc\_D(rho, speed\_test, W, phi) + W.\*y;

if(T\_test > .98\*T && T\_test < 1.02\*T)

speed = speed\_test;

found = 1;

else

speed\_test = speed\_test + 1;

end

if(speed\_test > speedHIGH)

disp('You messed up');

end

end

R = abs(speed.^2/32.2/tand(phi))

arcLen = speed\*dt; %ft

if(R ~= 0)

d\_bearing = arcLen/R\*turnRight; %radians

headIN = headIN + rad2deg(d\_bearing);

end

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

t = t + 1;

t\_arr(t) = t\_arr(t-1) + dt;

head(t) = headIN;

speed\_arr(t) = speed;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

dH = speed.\*y.\*dt; %ft

alt\_arr(t) = alt\_arr(t-1) + dH;

diff\_alt = altOUT - alt\_arr(t);

diff\_head = CalculateBearing(lat\_arr(t), long\_arr(t), latOUT, longOUT) - head(t);

if(sign(diff\_alt) ~= sign(y))

keep\_climb = 0;

end

if(sign(diff\_head) ~= sign(diff\_head\_init) || abs(diff\_head) < .01)

keep\_turn = 0;

end

end

R = abs(speed\_optimal.^2/32.2/tand(phi))

%% either altitude or bearing satisfied

%keep turning

if(abs(diff\_head) > 5 && keep\_turn == 1)

dt = 1;

y\_turn = 0;

rho = Calculate\_rho(alt\_arr(t));

% T = W.\*y\_turn + Calc\_D(rho, speed, W, deg2rad(phi));

T = Throttle.\*T\_max\_surface.\*(rho./rho\_surface).^m;

speed = speed\_optimal;

while(abs(diff\_head) > 2 && keep\_turn == 1)

arcLen = speed\*dt; %ft

if(R ~= 0)

d\_bearing = arcLen/R\*turnRight; %radians

headIN = headIN + rad2deg(d\_bearing);

end

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

t = t + 1;

t\_arr(t) = t\_arr(t-1) + dt;

head(t) = mod(headIN, 360);

speed\_arr(t) = speed;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

alt\_arr(t) = alt\_arr(t-1);

diff\_head = CalculateBearing(lat\_arr(t), long\_arr(t), latOUT, longOUT) - head(t)

if(sign(diff\_head) ~= sign(diff\_head\_init));

keep\_turn = 0;

end

end

end

if(abs(diff\_head) > .01 && keep\_turn == 1)

dt = .005;

y\_turn = 0;

rho = Calculate\_rho(alt\_arr(t));

% T = W.\*y\_turn+ Calc\_D(rho, speed, W, deg2rad(phi));

T = Throttle.\*T\_max\_surface.\*(rho./rho\_surface).^m;

speed = speed\_optimal;

while(abs(diff\_head) > .005 && keep\_turn == 1)

arcLen = speed\*dt; %ft

if(R ~= 0)

d\_bearing = arcLen/R\*turnRight; %radians

headIN = headIN + rad2deg(d\_bearing);

end

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

t = t + 1;

t\_arr(t) = t\_arr(t-1) + dt;

head(t) = mod(headIN, 360);

speed\_arr(t) = speed;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

alt\_arr(t) = alt\_arr(t-1);

diff\_head = CalculateBearing(lat\_arr(t), long\_arr(t), latOUT, longOUT) - head(t)

if(sign(diff\_head) ~= sign(diff\_head\_init))

keep\_turn = 0;

end

end

end

% need to set phi back to 0

if(phi ~= 0)

[t\_arr1, time\_marks1, string\_marks1, elev\_arr1, ail\_arr1, rud\_arr1] = Roll(alt\_arr(t), ...

speed\_arr(t), deg2rad(phi), 0, W, -1.\*turnRight, strcat(title\_, ' exit'));

t = t+2;

tau = t\_arr(end);

t\_arr = horzcat(t\_arr, tau + t\_arr1);

time\_marks = horzcat(time\_marks, tau + time\_marks1);

string\_marks = horzcat(string\_marks, string\_marks1);

head(t-1:t) = head(t-2);

alt\_arr(t-1:t) = alt\_arr(t-2);

lat\_arr(t-1:t) = lat\_arr(t-2);

long\_arr(t-1:t) = long\_arr(t-2);

speed\_arr(t-1:t) = speed\_arr(t-2);

throttle\_arr(length(throttle\_arr) + 1:length(throttle\_arr) + 2) = Throttle;

elev\_arr = horzcat(elev\_arr, elev\_arr1);

ail\_arr = horzcat(ail\_arr, ail\_arr1);

rud\_arr = horzcat(rud\_arr, rud\_arr1);

end

%keep climbing

if(abs(diff\_alt) > 500 && keep\_climb == 1)

dt = 3;

phi = 0;

while(abs(diff\_alt) > 500 && keep\_climb == 1)

rho = Calculate\_rho(alt\_arr(t));

% T = W.\*y + Calc\_D(rho, speed, W, deg2rad(phi));

T = Throttle.\*T\_max\_surface.\*(rho./rho\_surface).^m;

found = 0;

indexes = find(H < alt\_arr(t));

speedLOW = v\_min(indexes(end) + 1);

speedHIGH = v\_max(indexes(end) + 1);

speed\_test = speedLOW;

while(~found)

T\_test = Calc\_D(rho, speed\_test, W, phi) + W.\*y;

if(T\_test > .98\*T && T\_test < 1.02\*T)

speed = speed\_test;

found = 1;

else

speed\_test = speed\_test + 1;

end

end

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

t = t + 1;

t\_arr(t) = t\_arr(t-1) + dt;

head(t) = headIN;

speed\_arr(t) = speed;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

dH = speed.\*y.\*dt; %ft

alt\_arr(t) = alt\_arr(t-1) + dH;

diff\_alt = altOUT - alt\_arr(t)

if(sign(diff\_alt) ~= sign(y))

keep\_climb = 0;

end

end

end

if(abs(diff\_alt) > 1 && keep\_climb == 1)

dt = .1;

phi = 0;

while(abs(diff\_alt) > 1 && keep\_climb == 1)

rho = Calculate\_rho(alt\_arr(t));

% T = W.\*y + Calc\_D(rho, speed, W, deg2rad(phi));

T = Throttle.\*T\_max\_surface.\*(rho./rho\_surface).^m;

found = 0;

indexes = find(H < alt\_arr(t));

speedLOW = v\_min(indexes(end) + 1);

speedHIGH = v\_max(indexes(end) + 1);

speed\_test = speedLOW;

while(~found)

T\_test = Calc\_D(rho, speed\_test, W, phi) + W.\*y;

if(T\_test > .98\*T && T\_test < 1.02\*T)

speed = speed\_test;

found = 1;

else

speed\_test = speed\_test + 1;

end

end

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

t = t + 1;

t\_arr(t) = t\_arr(t-1) + dt;

head(t) = headIN;

speed\_arr(t) = speed;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

dH = speed.\*y.\*dt; %ft

alt\_arr(t) = alt\_arr(t-1) + dH;

diff\_alt = altOUT - alt\_arr(t)

if(sign(diff\_alt) ~= sign(y))

keep\_climb = 0;

end

end

end

% lat\_arr = lat\_arr(1:t);

% long\_arr = long\_arr(1:t);

% alt\_arr = alt\_arr(1:t);

% head = head(1:t);

head\_arr = head;

%%% Plot the flight envelope

figure

plot(v\_min, H, v\_max, H, speed\_arr, alt\_arr)

title(strcat('Flight Envelope: ', title\_))

xlabel('Velocity (ft/s)')

ylabel('Altitude (ft)')

end

%END HELIX2.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%Straight.m

%Zach Meves and Dieter Klemm

function [lat\_arr, long\_arr, alt\_arr, head\_arr, speed\_arr, t\_arr, time\_marks, string\_marks, throttle\_arr, elev\_arr, ail\_arr, rud\_arr, W\_out] = Straight(latIN, ...

longIN, altIN, headIN, speedIN, latOUT, longOUT, holdDistance, W, title\_)

S = 232;

W\_0 = 13000; %lbs, including all fuel

W\_empty = 8000; %lbs

W\_fuel = 5000; %lbs

CL\_max = 1.24;

CL\_0 = .0835;

CL\_alpha = 5.16;

CL\_de = .43;

CD\_0 = .023;

K = .073;

CM\_0 = .0895;

CM\_alpha = -1.09;

CM\_de = -1.13;

T\_max\_surface = 2\*3000; %lbs

c = 2\*1.18;% lb/(lb hr)

m = .6;

rho\_surface = .0023769; %slug/ft^3

C\_roll\_da = .05;

C\_roll\_dr = -.019;

C\_yaw\_da = .007;

C\_yaw\_dr = .149;

phi\_max = 33.557; %degrees, load factor of 1.2

rho = Calculate\_rho(altIN) %slugs/ft^3

T\_max = T\_max\_surface.\*(rho./rho\_surface).^m %lbs

final\_head\_init = CalculateBearing(latIN, longIN, latOUT, longOUT); % ~ final heading

diff\_head\_init = final\_head\_init - headIN;

headIN = final\_head\_init; %if close enough, clears numerical integration errors

dR = CalculateDistanceNumerical(latIN, longIN, latOUT, longOUT);

R = dR - holdDistance;

disp('Calculating Flight Envelope for Level Flight (Initial Weight)');

[H, v\_min, v\_max, a, stall] = FlightEnvelope(W, 0, 0);

indexes = find(H < altIN);

speedLOW = v\_min(indexes(end) + 1);

speedHIGH = v\_max(indexes(end) + 1);

speed\_optimal = .5\*(speedLOW + speedHIGH);

disp('Calculating Flight Envelope for Level Flight (Empty Weight)');

[H2, v\_min2, v\_max2, a2, stall2] = FlightEnvelope(W\_empty, 0, 0);

indexes = find(H2 < altIN);

speedLOW2 = v\_min2(indexes(end) + 1);

speedHIGH2 = v\_max2(indexes(end) + 1);

speed\_optimal2 = .5\*(speedLOW2 + speedHIGH2);

speed\_optimal\_avg = .5.\*(speed\_optimal + speed\_optimal2);

T\_avail = T\_max\_surface.\*(rho./rho\_surface).^m;

T\_needed = Calc\_D(rho, speed\_optimal, W, 0);

Throttle = T\_needed/T\_avail;

head(1) = headIN;

alt\_arr(1) = altIN;

lat\_arr(1) = latIN;

long\_arr(1) = longIN;

speed\_arr(1) = speed\_optimal;

throttle\_arr(1) = Throttle;

alpha\_ = (2.\*W./(cosd(0).\*rho.\*speed\_optimal.^2.\*S) - CL\_0)./CL\_alpha;

elev\_arr(1) = (-1.\*CM\_0 - CM\_alpha.\*alpha\_)./CM\_de;

ail\_arr(1) = 0;

rud\_arr(1) = 0;

t\_arr(1) = 1;

time\_marks(1) = 1;

string\_marks(1) = cellstr(title\_);

t = 1;

dt = 10;

while R > (10\*5280 + holdDistance)

t = t + 1;

alt\_arr(t) = altIN;

head\_arr(t) = headIN;

speed\_test = speedLOW;

T = Throttle.\*T\_avail;

found = 0;

segment\_ = 0;

while(~found)

T\_test = Calc\_D(rho, speed\_test, W, 0);

if(T\_test > .98\*T && T\_test < 1.02\*T)

speed = speed\_test;

found = 1;

if(segment\_ == 1) %% if modified Throttle

throttle\_arr(length(throttle\_arr) + 1) = Throttle;

alpha\_ = (2.\*W./(cosd(0).\*rho.\*speed.^2.\*S) - CL\_0)./CL\_alpha;

elev\_arr(length(elev\_arr) + 1) = (-1.\*CM\_0 - CM\_alpha.\*alpha\_)./CM\_de;

ail\_arr(length(ail\_arr) + 1) = 0;

rud\_arr(length(ail\_arr) + 1) = 0;

time\_marks(length(time\_marks) + 1) = t;

string\_marks(length(string\_marks) + 1) = cellstr('Modified Throttle');

end

else

speed\_test = speed\_test + 1;

end

if(speed\_test > speedHIGH)

segment\_ = 1;

Throttle = min(1, 1.1\*Throttle);

T = Throttle.\*T\_avail;

speed\_test = speedLOW;

end

if(Throttle == 1 && speed\_test > speedHIGH)

disp('You messed up');

end

end

speed\_arr(t) = speed;

t\_arr(t) = t\_arr(t-1) + dt;

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

headIN = CalculateBearing(lat\_arr(t), long\_arr(t), latOUT, longOUT);

W = W - c.\*T.\*dt./3600;

R = R - speed.\*dt;

if W < W\_empty

disp('Out of fuel');

end

end

dt = 2;

while R > holdDistance

t = t + 1;

alt\_arr(t) = altIN;

head\_arr(t) = headIN;

speed\_test = speedLOW;

T = Throttle.\*T\_avail;

found = 0;

segment\_ = 0;

while(~found)

T\_test = Calc\_D(rho, speed\_test, W, 0);

if(T\_test > .98\*T && T\_test < 1.02\*T)

speed = speed\_test;

found = 1;

if(segment\_ == 1) %% if modified Throttle

throttle\_arr(length(throttle\_arr) + 1) = Throttle;

alpha\_ = (2.\*W./(cosd(0).\*rho.\*speed.^2.\*S) - CL\_0)./CL\_alpha;

elev\_arr(length(elev\_arr) + 1) = (-1.\*CM\_0 - CM\_alpha.\*alpha\_)./CM\_de;

ail\_arr(length(ail\_arr) + 1) = 0;

rud\_arr(length(ail\_arr) + 1) = 0;

time\_marks(length(time\_marks) + 1) = t;

string\_marks(length(string\_marks) + 1) = cellstr('Modified Throttle');

end

else

speed\_test = speed\_test + 1;

end

if(speed\_test > speedHIGH)

segment\_ = 1;

Throttle = min(1, 1.1\*Throttle);

T = Throttle.\*T\_avail;

speed\_test = speedLOW;

end

if(Throttle == 1 && speed\_test > speedHIGH)

disp('You messed up');

end

end

speed\_arr(t) = speed;

t\_arr(t) = t\_arr(t-1) + dt;

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

headIN = CalculateBearing(lat\_arr(t), long\_arr(t), latOUT, longOUT);

W = W - c.\*T.\*dt./3600;

R = R - speed.\*dt;

if W < W\_empty

disp('Out of fuel');

end

end

W\_out = W;

figure

plot(v\_min, H, v\_max, H, speed\_arr, alt\_arr)

title(strcat('Flight Envelope (initial weight): ', title\_))

xlabel('Velocity (ft/s)')

ylabel('Altitude (ft)')

figure

plot(v\_min2, H2, v\_max2, H2, speed\_arr, alt\_arr)

title(strcat('Flight Envelope (empty weight): ', title\_))

xlabel('Velocity (ft/s)')

ylabel('Altitude (ft)')

end

%END STRAIGHT.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%Roll.m

%Zach Meves and Dieter Klemm

function [t\_arr, time\_marks, string\_marks, Elev, Ail, Rud] = Roll(altIN, speedIN, phiIN, phiOUT, W, right, title\_)

S = 232;

W\_0 = 13000; %lbs, including all fuel

W\_empty = 8000; %lbs

W\_fuel = 5000; %lbs

CL\_max = 1.24;

CL\_0 = .0835;

CL\_alpha = 5.16;

CL\_de = .43;

CD\_0 = .023;

K = .073;

CM\_0 = .0895;

CM\_alpha = -1.09;

CM\_de = -1.13;

T\_max\_surface = 2\*3000; %lbs

c = 2\*1.18;% 1b/hr

m = .6;

rho\_surface = .0023769; %slug/ft^3

C\_roll\_da = .05;

C\_roll\_dr = -.019;

C\_yaw\_da = .007;

C\_yaw\_dr = .149;

b = 17.2\*2; %ft

I = .5.\*W./32.2.\*(5.25/2).^2; %approximate as a cylinder

phi\_max = 33.557; %degrees, load factor of 1.2

rho = Calculate\_rho(altIN);

alpha\_ = (2.\*W./(cos(phiIN).\*rho.\*speedIN.^2.\*S) - CL\_0)./CL\_alpha;

Elev(1) = (-CM\_0 - CM\_alpha.\*alpha\_)./CM\_de;

Ail(1) = right.\*.3;

Rud(1) = -C\_yaw\_da.\*Ail./C\_yaw\_dr;

C\_roll = C\_roll\_dr.\*Rud(1) + C\_roll\_da.\*Ail(1);

a = .5.\*rho.\*speedIN.^2.\*S.\*b.\*C\_roll./I;

t\_F = sqrt(abs(abs(phiIN-phiOUT).\*2./a));

t\_arr = [1, 1 + t\_F];

time\_marks = [1, 1 + t\_F];

string\_marks = [cellstr(strcat(title\_,': Start Roll')), cellstr(strcat(title\_, ': End Roll'))];

alpha\_ = (2.\*W./(cos(phiOUT).\*rho.\*speedIN.^2.\*S) - CL\_0)./CL\_alpha;

Elev(2) = (-CM\_0 - CM\_alpha.\*alpha\_)./CM\_de;

Ail(2) = 0;

Rud(2) = 0;

%% roll right is positive aileron deflection

End

%END ROLL.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%BearingChange.m

%Zach Meves Dieter Klemm

function [lat\_arr, long\_arr, alt\_arr, head\_arr, speed\_arr, t\_arr, time\_marks, string\_marks, throttle\_arr, elev\_arr, ail\_arr, rud\_arr] = BearingChange(latIN, ...

longIN, altIN, headIN, speedIN, headOUT, altOUT, W, title\_)

S = 232;

W\_0 = 13000; %lbs, including all fuel

W\_empty = 8000; %lbs

W\_fuel = 5000; %lbs

CL\_max = 1.24;

CL\_0 = .0835;

CL\_alpha = 5.16;

CL\_de = .43;

CD\_0 = .023;

K = .073;

CM\_0 = .0895;

CM\_alpha = -1.09;

CM\_de = -1.13;

T\_max\_surface = 2\*3000; %lbs

c = 2\*1.18;% 1b/hr

m = .6;

rho\_surface = .0023769; %slug/ft^3

C\_roll\_da = .05;

C\_roll\_dr = -.019;

C\_yaw\_da = .007;

C\_yaw\_dr = .149;

phi\_max = 33.557; %degrees, load factor of 1.2

rho = Calculate\_rho(altOUT) %slugs/ft^3

T\_max = T\_max\_surface.\*(rho./rho\_surface).^m %lbs

diff\_head\_init = headOUT - headIN;

if(diff\_head\_init == 0)

lat\_arr = [];

long\_arr = [];

alt\_arr = [];

head\_arr = [];

speed\_arr = [];

t\_arr = [];

time\_marks = [];

string\_marks = [];

throttle\_arr = [];

elev\_arr = [];

ail\_arr = [];

rud\_arr = [];

return;

end

diff\_head = diff\_head\_init;

phi = phi\_max;

turnRight = 1;

if(diff\_head\_init == 0)

R = 0;

phi = 0;

turnRight = 0;

end

if(diff\_head\_init > 0)

turnRight = 1;

if(diff\_head\_init > 180)

turnRight = -1;

end

else

turnRight = -1;

if(diff\_head\_init < -180)

turnRight = 1;

end

end

T\_needed = W.\*(CD\_0./((cosd(phi)).^2).\*sqrt(K./CD\_0) + K.\*sqrt(CD\_0./K)) %lb

ceiling\_rho = (T\_needed./T\_max\_surface).^(1./m).\*rho\_surface %slug/ft^3

while(ceiling\_rho > rho && phi >= 3) %while can't turn that high with phi

phi = phi - 2;

T\_needed = W.\*(CD\_0./((cosd(phi)).^2).\*sqrt(K./CD\_0) + K.\*sqrt(CD\_0./K)) %lb

ceiling\_rho = (T\_needed/T\_max\_surface)^(1/m)\*rho\_surface; %slug/ft^3

end

T\_needed

T\_avail = T\_max\_surface.\*(rho./rho\_surface).^m

Throttle = T\_needed/T\_avail

phi

disp('Calculating Flight Envelope for Bearing Change');

[H, v\_min, v\_max, a, stall] = FlightEnvelope(W, deg2rad(phi), 0);

indexes = find(H < altOUT);

speedLOW = v\_min(indexes(end) + 1);

speedHIGH = v\_max(indexes(end) + 1);

speed\_optimal = .5\*(speedLOW + speedHIGH);

T\_needed = Calc\_D(rho, speed\_optimal, W, deg2rad(phi));

Throttle = T\_needed/T\_avail

if(diff\_head\_init ~= 0)

R = abs(speed\_optimal.^2/32.2/tand(phi)) %ft

end

head(1) = headIN;

alt\_arr(1) = altIN;

lat\_arr(1) = latIN;

long\_arr(1) = longIN;

speed\_arr(1) = speed\_optimal;

T\_current = Calc\_D(Calculate\_rho(altIN), speedIN, W, 0);

T\_current\_avail = T\_max\_surface.\*(Calculate\_rho(altIN)./rho\_surface).^m;

[t\_arr, time\_marks, string\_marks, elev\_arr, ail\_arr, rud\_arr] = Roll(alt\_arr(1), ...

speedIN, 0, deg2rad(phi), W, turnRight, strcat(title\_, ' enter'));

t = 2; %spot in t\_arr

head(1:t) = head(1);

alt\_arr(1:t) = alt\_arr(1);

lat\_arr(1:t) = lat\_arr(1);

long\_arr(1:t) = long\_arr(1);

speed\_arr(1:t) = speedIN;

throttle\_arr(1) = T\_current/T\_current\_avail;

throttle\_arr(2) = Throttle;

alpha\_ = (2.\*W./(cosd(phi).\*rho.\*speed\_optimal.^2.\*S) - CL\_0)./CL\_alpha;

keep\_turn = 1;

if(sign(diff\_head) ~= sign(diff\_head\_init) || abs(diff\_head) < .01)

keep\_turn = 0;

end

if(abs(diff\_head) > 5 && keep\_turn == 1)

dt = 1;

rho = Calculate\_rho(alt\_arr(t));

T = Throttle.\*T\_max\_surface.\*(rho./rho\_surface).^m;

speed = speed\_optimal;

while(abs(diff\_head) > 2 && keep\_turn == 1)

arcLen = speed\*dt; %ft

if(R ~= 0)

d\_bearing = arcLen/R\*turnRight; %radians

headIN = headIN + rad2deg(d\_bearing);

end

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

t = t + 1;

t\_arr(t) = t\_arr(t-1) + dt;

head(t) = headIN;

speed\_arr(t) = speed;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

alt\_arr(t) = alt\_arr(t-1);

diff\_head = headOUT - head(t)

if(sign(diff\_head) ~= sign(diff\_head\_init));

keep\_turn = 0;

end

end

end

if(abs(diff\_head) > .01 && keep\_turn == 1)

dt = .005;

rho = Calculate\_rho(alt\_arr(t));

T = Throttle.\*T\_max\_surface.\*(rho./rho\_surface).^m;

speed = speed\_optimal;

while(abs(diff\_head) > .005 && keep\_turn == 1)

arcLen = speed\*dt; %ft

if(R ~= 0)

d\_bearing = arcLen/R\*turnRight; %radians

headIN = headIN + rad2deg(d\_bearing);

end

dx = speed.\*sind(headIN).\*dt;

dy = speed.\*cosd(headIN).\*dt;

t = t + 1;

t\_arr(t) = t\_arr(t-1) + dt;

head(t) = headIN;

speed\_arr(t) = speed;

[lat\_arr(t), long\_arr(t)] = CalculateLatLong(lat\_arr(t-1), long\_arr(t-1), dx, dy);

alt\_arr(t) = alt\_arr(t-1);

diff\_head = headOUT - head(t)

if(sign(diff\_head) ~= sign(diff\_head\_init))

keep\_turn = 0;

end

end

end

% need to set phi back to 0

if(phi ~= 0)

[t\_arr1, time\_marks1, string\_marks1, elev\_arr1, ail\_arr1, rud\_arr1] = Roll(alt\_arr(t), ...

speed\_arr(t), deg2rad(phi), 0, W, -1.\*turnRight, strcat(title\_, ' exit'));

t = t+2;

tau = t\_arr(end);

t\_arr = horzcat(t\_arr, tau + t\_arr1);

time\_marks = horzcat(time\_marks, tau + time\_marks1);

string\_marks = horzcat(string\_marks, string\_marks1);

head(t-1:t) = head(t-2);

alt\_arr(t-1:t) = alt\_arr(t-2);

lat\_arr(t-1:t) = lat\_arr(t-2);

long\_arr(t-1:t) = long\_arr(t-2);

speed\_arr(t-1:t) = speed\_arr(t-2);

throttle\_arr(length(throttle\_arr) + 1:length(throttle\_arr) + 2) = Throttle;

elev\_arr = horzcat(elev\_arr, elev\_arr1);

ail\_arr = horzcat(ail\_arr, ail\_arr1);

rud\_arr = horzcat(rud\_arr, rud\_arr1);

end

head\_arr = head;

%%% Plot the flight envelope

figure

plot(v\_min, H, v\_max, H, speed\_arr, alt\_arr)

title(strcat('Flight Envelope: ', title\_))

xlabel('Velocity (ft/s)')

ylabel('Altitude (ft)')

end

%END BEARINGCHANGE.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%FlightEnvelope.m

%Zach Meves and Dieter Klemmm

%inputs: Weight, phi, gamma (phi in radians)

%outpus are in imperial units

function [H, v\_min, v\_max, a, stall] = FlightEnvelope(W, phi, y)

S = 232;

W\_0 = 13000; %lbs, including all fuel

W\_empty = 8000; %lbs

W\_fuel = 5000; %lbs

CL\_max = 1.24;

CL\_0 = .0835;

CL\_alpha = 5.16;

CL\_de = .43;

CD\_0 = .023;

K = .073;

CM\_0 = .0895;

CM\_alpha = -1.09;

CM\_de = -1.13;

T\_max\_surface = 2\*3000; %lbs

c = 2\*1.18;% 1/hr

m = .6;

rho\_surface = .0023769; %slug/ft^3

C\_roll\_da = .05;

C\_roll\_dr = -.019;

C\_yaw\_da = .007;

C\_yaw\_dr = .149;

h\_meters = 1:(80000\*.3048); %meters

[Temp, P, rho] = Calculate\_T\_P\_rho(h\_meters(end)); %kg/m^3

rho = rho(2:end);

rho = double(rho).\*.00194032033; %slug/ft^3

h\_feet = h\_meters./.3048; %feet

v\_stall = (2\*W./S./CL\_max./rho./cos(phi)).^.5; %ft/s

stall = v\_stall;

a\_seaLevel = 340/.3048; %ft/s

a = (Temp.\*1.4.\*286).^.5;

a = a./.3048;

ceiling\_thrust = W.\*(CD\_0./((cos(phi)).^2).\*sqrt(K./CD\_0) + K.\*sqrt(CD\_0./K) + y); %lb

%for project engine:

ceiling\_rho = double((ceiling\_thrust/T\_max\_surface))^double((1/m))\*double(rho\_surface); %slug/ft^3

h = h\_feet;

h\_meters\_min = h\_meters(find(rho >= .7\*ceiling\_rho));

T = (rho./rho\_surface).^m.\*T\_max\_surface; %lbs

%loop through height

for i = 1:h\_meters(end)-1

% T = (rho(i)/rho\_surface).^m.\*T\_max\_surface; %lbs

count = 0;

v = v\_stall(i):a(i);

T\_v = Calc\_D(rho(i), v, W, phi) + W.\*y;

% for v = v\_stall(i):a(i) %ft/s

if(length(find(T\_v <= T(i))) ~= 0)

count = 1;

temp\_v = v(find(T\_v <= T(i)));

end

% end

if(count == 0)

temp\_v = [0, 0];

end

if(temp\_v(1) > v\_stall(i))

min\_v(i) = temp\_v(1);

else

min\_v(i) = v\_stall(i);

end

if(temp\_v(end) < a(i))

max\_v(i) = temp\_v(end);

else

max\_v(i) = a(i);

%max\_v(i) = temp\_v(end);

end

temp\_v = 0;

end

H = h(1:end-1);

v\_min = min\_v;

v\_max = max\_v;

index = find(v\_min == max(v\_min));

if(v\_min(index) ~= v\_min(end))

v\_min(index + 1 : end) = max(v\_min);

v\_max(index + 1 : end) = max(v\_min);

end

a = a(1:end-1);

stall = v\_stall(find(rho >= ceiling\_rho));

stall = stall(1:end-1);

end

%END FLIGHTENVELOPE.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%Calculate\_T\_P\_rho.m

%Zach Meves and Dieter Klemm

%Calculates the temperature, pressure, and density at heights (metric units)

function [T,P,rho] = Calculate\_T\_P\_rho(height)

% Define constants

g = 9.81; %m/s^2

R = 287.0; %J/kgK

T0 = 273.15 + 15; %K

P0 = 101325.0; %Pa

rho0 = 1.225; %kg/m^3

% Define thermal layers

%% Troposphere (0-11000m)

H1 = 11000.0; %m

G1 = -.0065; %K/m

%% Tropopause (11000-20000m)

G2 = 0.0;

H2 = 20000.0; %m

%% Stratosphere (20000-32000m)

G3 = .001; %K/m

H3 = 32000.0; %m

T = calculateTemperatures();

P = calculatePressures();

rho = calculateDensities();

%height = ceil(height\*.3048);

% Define equations

function T = calculateTemperatures()

% Fills the T array with temperature values

for n = 0:height

T(n+1) = calculateTemperature(n);

end

end

function T = calculateTemperature(h)

% Calculates the temperature at height 'h'

if h > H2

T = T0 + H1.\*G1 + (h-H2).\*G3;

elseif h > H1

T = T0 + H1.\*G1;

else

T = T0 + h.\*G1;

end

end

function P = calculatePressures()

% Fills the pressure array

for n = 0:height

P(n+1) = calculatePressure(n);

end

end

function P = calculatePressure(h)

% Calculates the pressure at a height 'h'

if h <= H1 % Troposphere

P = P0.\*((calculateTemperature(h)./T0).^(-g/(R.\*G1)));

elseif h <= H2 %Tropopause

p0 = P0.\*((calculateTemperature(H1)./T0).^(-g/(R.\*G1)));

power = -g./(R.\*calculateTemperature(h)).\*(h - H1);

P = p0.\*exp(power);

else %Stratosphere

p1 = P0.\*((calculateTemperature(H1)./T0).^(-g/(R.\*G1)));

power = -g./(R.\*calculateTemperature(H2)).\*(H2 - H1);

p0 = p1.\*exp(power);

t0 = calculateTemperature(H2);

P = p0.\*((calculateTemperature(h)./t0).^(-g/(R.\*G3)));

end

end

function rho = calculateDensities()

% Fills the density array

for n = 0:height

rho(n+1) = calculateDensity(n);

end

end

function rho = calculateDensity(h)

p = calculatePressure(h);

t = calculateTemperature(h);

rho = p./(R.\*t);

end

end

%END CALCULATE\_T\_P\_RHO.m\_\_\_\_\_\_\_\_\_\_\_\_\_

%Calculate\_rho\_vector.m

%Zach Meves and Dieter Klemm

%Calculates the density (in slug/ft^3) at height (in feet)

function rho = Calculate\_rho\_vector(height)

% Define constants

g = 9.81; %m/s^2

R = 287.0; %J/kgK

T0 = 273.15 + 15; %K

P0 = 101325.0; %Pa

rho0 = 1.225; %kg/m^3

% Define thermal layers

%% Troposphere (0-11000m)

H1 = 11000.0; %m

G1 = -.0065; %K/m

%% Tropopause (11000-20000m)

G2 = 0.0;

H2 = 20000.0; %m

%% Stratosphere (20000-32000m)

G3 = .001; %K/m

H3 = 32000.0; %m

height = ceil(height.\*.3048); % to metric

T = calculateTemperatures();

P = calculatePressures();

rho = calculateDensities();

rho = rho.\*.00194032033;

% Define equations

function T = calculateTemperatures()

% Fills the T array with temperature values

for n = 0:height

T(n+1) = calculateTemperature(n);

end

end

function T = calculateTemperature(h)

% Calculates the temperature at height 'h'

if h > H2

T = T0 + H1.\*G1 + (h-H2).\*G3;

elseif h > H1

T = T0 + H1.\*G1;

else

T = T0 + h.\*G1;

end

end

function P = calculatePressures()

% Fills the pressure array

for n = 0:height

P(n+1) = calculatePressure(n);

end

end

function P = calculatePressure(h)

% Calculates the pressure at a height 'h'

if h <= H1 % Troposphere

P = P0.\*((calculateTemperature(h)./T0).^(-g/(R.\*G1)));

elseif h <= H2 %Tropopause

p0 = P0.\*((calculateTemperature(H1)./T0).^(-g/(R.\*G1)));

power = -g./(R.\*calculateTemperature(h)).\*(h - H1);

P = p0.\*exp(power);

else %Stratosphere

p1 = P0.\*((calculateTemperature(H1)./T0).^(-g/(R.\*G1)));

power = -g./(R.\*calculateTemperature(H2)).\*(H2 - H1);

p0 = p1.\*exp(power);

t0 = calculateTemperature(H2);

P = p0.\*((calculateTemperature(h)./t0).^(-g/(R.\*G3)));

end

end

function rho = calculateDensities()

% Fills the density array

for n = 0:height

rho(n+1) = calculateDensity(n);

end

end

function rho = calculateDensity(h)

p = calculatePressure(h);

t = calculateTemperature(h);

rho = p./(R.\*t);

end

end

%END CALCULATE\_RHO\_VECTOR.m\_\_\_\_\_\_\_\_\_\_\_\_

%Calculate\_rho.m

%Zach Meves and Dieter Klemm

%Calculates the density (in slug/ft^3) at height (in feet)

function rho = Calculate\_rho(height)

rho = Calculate\_rho\_vector(height)

rho = rho(end);

end

%END CALCULATE\_RHO.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%ConvertString

%Zach Meves and Dieter Klemm

%converts latitude and longitude strings to numerical values

function [lat, long] = ConvertString(latIN, longIN)

lat1 = strsplit(latIN, '\_');

long1 = strsplit(longIN, '\_');

%filling angle values

if(strcmp(char(long1(2)),'W') == 1)

theta1 = -1\*str2double(long1(1));

else

theta1 = str2double(long1(1));

end

if(strcmp(char(lat1(2)),'N') == 1)

phi1 = str2double(lat1(1));

else

phi1 = -str2double(lat1(1));

end

lat = phi1;

long= theta1;

end

%END CONVERT\_STRING.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%CalculateBearing.m

%Zach Meves and Dieter Klemm

%input the initial and final lat and long, in degrees

%outputs the initial bearing needed to hit that final destination, in degress from N

%source equation: www.moveable-type.co.uk/scripts/latlong.html

function bearing = CalculateBearing(lat1, long1, lat2, long2)

bearing = mod(360 + atan2d(sind(long2-long1).\*cosd(lat2), cosd(lat1).\*sind(lat2) ...

- sind(lat1).\*cosd(lat2).\*cosd(long2 - long1)), 360);

End

%END CALCULATEBEARING.m

%Calc\_D

%Zach Meves and Dieter Klemm

%Calculates the drag at the given conditions (enter phi in radians)

function D = Calc\_D(rho, v, W, phi)

W\_0 = 13000; %lbs, including all fuel

W\_empty = 8000; %lbs

W\_fuel = 5000; %lbs

CL\_max = 1.24;

CL\_0 = .0835;

CL\_alpha = 5.16;

CL\_de = .43;

CD\_0 = .023;

K = .073;

CM\_0 = .0895;

CM\_alpha = -1.09;

CM\_de = -1.13;

T\_max\_surface = 2\*3000; %lbs

c = 2\*1.18;% 1b/hr

m = .6;

C\_roll\_da = .05;

C\_roll\_dr = -.019;

S = 232;

C\_yaw\_da = .007;

C\_yaw\_dr = .149;

D = .5.\*rho.\*v.^2.\*S.\*CD\_0 + 2.\*K.\*W.^2./(rho.\*v.^2.\*S.\*(cos(phi)).^2);

End

%END CALC\_D.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%CalculateLatLong

%Zach Meves and Dieter Klemm

%input current latIn, longIN in degrees

%input delta x and delta y in feet

%output final lat and long in degrees

%accurate for small dx and dy

function [lat, long] = CalculateLatLong(latIN, longIN, dx, dy)

rad = 3959\*5280; %ft

lat = latIN + rad2deg(dy/rad);

phi = 90 - latIN;

x = rad\*sind(phi);

dlong = dx./x;

long = longIN + rad2deg(dlong);

end

%END CALCULATELATLONG.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%CalculateDistanceNumerical.m

%Zach Meves and Dieter Klemm

% Calculates the distance (in ft) between two coordinates,

% Input format: numerical value of latitudes and longitudes

function distance = CalculateDistanceNumerical(lat1, long1, lat2, long2)

p = 3959\*5280; %Radius of Earth in feet

phi1 = 90 - lat1;

phi2 = 90 - lat2;

%convert to x,y,z coordinates

x1 = p.\*sind(phi1)\*cosd(long1);

x2 = p.\*sind(phi2)\*cosd(long2);

y1 = p.\*sind(phi1)\*sind(long1);

y2 = p.\*sind(phi2)\*sind(long2);

z1 = p.\*cosd(phi1);

z2 = p.\*cosd(phi2);

angle = acosd((x1.\*x2+y1.\*y2+z1.\*z2)/p^2);

distance = degtorad(angle)\*p; %distance in feet

end

%END CALCULATEDISTANCENUMERICAL.m\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%CalculateDistance.m

%Zach Meves and Dieter Klemm

% Calculates the distance (in km) between two coordinates,

% Input format: ###\_[N,S,E,W] (number followed by '\_' then N,S,E,or W)

function distance = CalculateDistance(lat1, long1, lat2, long2)

p = 3959\*5280; %Radius of Earth in feet

%format input correctly

lat1 = strsplit(lat1, '\_');

long1 = strsplit(long1, '\_');

lat2 = strsplit(lat2, '\_');

long2 = strsplit(long2, '\_');

%filling angle values

if(strcmp(char(long1(2)),'W') == 1)

theta1 = str2double(long1(1));

else

theta1 = 360 - str2double(long1(1));

end

if(strcmp(char(long2(2)), 'W') == 1)

theta2 = str2double(long2(1));

else

theta2 = 360 - str2double(long2(1));

end

if(strcmp(char(lat1(2)),'N') == 1)

phi1 = 90 - str2double(lat1(1));

else

phi1 = 90 + str2double(lat1(1));

end

if(strcmp(char(lat2(2)),'N') == 1)

phi2 = 90 - str2double(lat2(1));

else

phi2 = 90 + str2double(lat2(1));

end

%convert to x,y,z coordinates

x1 = p.\*sind(phi1)\*cosd(theta1);

x2 = p.\*sind(phi2)\*cosd(theta2);

y1 = p.\*sind(phi1)\*sind(theta1);

y2 = p.\*sind(phi2)\*sind(theta2);

z1 = p.\*cosd(phi1);

z2 = p.\*cosd(phi2);

angle = acosd((x1.\*x2+y1.\*y2+z1.\*z2)/p^2);

distance = degtorad(angle)\*p; %distance in feet

end

%END CALCULATEDISTANCE.m