UAV Simulation File Information

Austin Murch

January 25, 2012

Contents

1 Introduction

This document is a collection of the embedded README blocks and m-file help comments for the UMN UAV simulation, developed by the UAV Research Group at the University of Minnesota. The UAV simulation model is written in the Matlab/Simulink environment using the Aerospace Blockset. Three simulation environments are maintained: a basic nonlinear simulation, a Software-In-the-Loop simulation, and a Processor-In-the-Loop simulation. All three simulations share the same plant dynamics, actuator, sensor, and environmental models via Simulink Libraries. Aircraft and environmental parameters are set in m-files and shared between the simulations. Two aircraft models are maintained, one for the Ultra Stick 25e and one for the FASER aircraft.

1.1 MATLAB Version

The UMN UAV simulation was developed with 32-bit MATLAB R2010a. Users have reported successfully using R2009b; however, R2010a or later is recommended. R2009a is known to fail with the SIL simulation.

2 Nonlinear Simulation: UAV_NL

2.1 M-Files

2.1.1 example1.m

```
example1.m

-------------- Doublet response, NonLinear and Linear Models -------------

Script trims the model to a level flight condition and linearizes.
It compares doublet responses between full nonlinear sim and
the full and decoupled linearized models

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: example1.m 559 2011-09-01 19:44:55Z murch $
```

2.1.2 example2.m

```
example2.m

---------- Trim & linearize over a range of flight conditions ------------

Script calculates a set of level flight trim condtions and linear models
for different airspeeds. Plots trim conditions and dynamic mode
characteristics as a function of airspeed.

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.
```

```
SVN Info: $Id: example2.m 312 2011-04-01 16:15:59Z murch $
```

2.1.3 linear_aero.m

```
#eml
This function uses the linear derivatives to compute the 6 aerodynamic coefficients
```

2.1.4 linearize_tutorial.m

```
UMN UAV Simulation: Linearize Tutorial
This tutorial walks through the steps of linearizing the UMN UAV
Simulation model. Most of these steps are handled in the "setup.m" and
"linearize_UAV.m" functions provided with the UMN UAV sim. However, this
tutorial will give you an in-depth understanding of how these functions
work.

  Published output in the Help browser
     showdemo linearize_tutorial
```

2.1.5 make_faser_aero_symmetric.m

```
This script makes the FASER baseline aerodynamic data symmetric, and
saves the offsets in a new data structure so they can be added in if
desired.
```

2.1.6 model_check.m

```
model_check.m
UAV_NL Model Verification

Compares the linear/nonlinear doublet response of the current simulation
model (blue/green lines) with the checkcase data (red/black).

University of Minnesota
Aerospace Engineering and Mechanics
```

```
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: model_check.m 314 2011-04-05 16:53:30Z murch $
```

2.1.7 setup.m

```
setup.m
UAV Nonlinear Simulation setup

This script will setup the nonlinear simulation (UAV_NL.mdl) and call
trim and linearization routines. Select the desired aircraft here in this
script, via the "UAV_config()" function call.

Note: the UAV_NL.mdl model is not opened by default. This is not
necessary to trim, linearize, and simulate via command line inputs.

Calls: UAV_config.m
       trim_UAV.m
       linearize_UAV.m

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: setup.m 720 2011-11-23 18:30:24Z murch $
```

2.1.8 trim_tutorial.m

```
UMN UAV Simulation: Trim Tutorial
This tutorial walks through the steps of trimming the UMN UAV Simulation
model. Most of these steps are handled in the "setup.m" and "trim_UAV.m"
functions provided with the UMN UAV sim. However, this tutorial will give
you an in-depth understanding of how these functions work.

  Published output in the Help browser
     showdemo trim_tutorial
```

2.2 Simulink Blocks

2.2.1 UAV_NL

Nonlinear UAV Simulation

The nonlinear simulation has the Nonlinear UAV Model only (no actuators or sensor models). Top level inputs and outputs are used for generating and storing trim conditions and linear models. The trim condition generated with this model is used for the other simulations. The aircraft configuration, trim condition, and linear models are stored in the Libraries directory.

Notes:

The model automatically sets the wind and magnetic models to a "Bypass" option. This is done using the model's InitFcn callback. To view or edit this function, use the Model Explorer -> UAV_NL, Callbacks tab, then go to "InitFcn".

The UAV_NL.mdl model is not opened by default. This is not necessary to trim, linearize, and simulate via command line inputs.

Light blue blocks are a UMN library link; orange blocks are a Simulink Aerospace Blockset library link. README blocks are green.

The root level inport/outport blocks are require for trimming the model. DO NOT delete or rename these blocks.

Control Inputs Sign Convention
Elevator: +TED
Rudder: +TEL
Aileron: +TED, da = (da_R-da_L)/2
Flap: +TED
Throttle: always positive

2.2.2 UAV_NL/Nonlinear UAV Model

Nonlinear UAV Model

This block implements the nonlinear UAV dynamics model. Beginning on the left, the Forces and Moments block models all of the relevant external forces and moments acting on the aircraft.

The 6DoF EOM block implements the six degree of freedom, fixed mass, flat, non-rotating Earth, rigid body equations of motion. This block has been modified from the original Aerospace Blockset implementation. The inertial position vector (Xe) is no longer computed. The original block did not have a way to add steady state winds to the inertial velocities (Ve) prior to integrating to get Xe. Therefore, this step is performed in the Auxiliary Equations block.

The Auxiliary Equations computes other relevant variables (such as angle of attack, indicated airspeed, etc) from state data. In this block are the Navigation equations.

Finally, the Environment block has Aerospace Blockset models for Earth's atmosphere, gravity, and magnetic fields. Winds are modeled in two portions: steady and unsteady. The steady portion is defined by a speed and direction, and is horizontal only. The unsteady portion is made up of a wind shear model and a turbulence model.

2.2.3 UAV_NL/Nonlinear UAV Model/Environment

Environment Model

This block uses the Aerospace Blockset models for Earth's atmosphere, gravity, magnetic field, and wind.

The atmosphere is modeled using the 1976 Standard Atmosphere block.

Winds are modeled in two portions: steady and unsteady. The steady portion is defined by a speed and direction, and is horizontal only. The unsteady portion is made up of a wind gust model and a turbulence model.

The Dryden Wind Turbulence Model is used to model turbulence. The intensity of the turbulence at low altitude (<1000ft) is determined by the wind speed and direction; this is set as separate variables from the steady wind speed and direction. The turbulence is off by default, and can be enabled by setting the Env.Winds.TurbulenceOn boolean in UAV_config.m.

The Discrete Gust Model is used to model wind gusts. Parameters are time on, duration (length), and amplitude, in three axes (u,v,w). These parameters are set in Env.Winds, in UAV_config.m

Note the booleans in the Env.Winds stucture to turn on each wind component: >> Env.Winds

```
ans =

     TurbulenceOn: 0
    TurbWindSpeed: 0
      TurbWindDir: 0
            GustOn: 0
    GustStartTime: 0
       GustLength: [1 1 1]
    GustAmplitude: [1 1 1]
     SteadyWindOn: 0
        WindSpeed: 0
          WindDir: 0
```

The WGS84 Gravity Model is used to model Earth's gravity as a function of latitude, longitude, and altitude.

The World Magnetic Model 2005 is used to model Earth's magnetic field as a function of latitude, longitude, and altitude. The current decimal year is input.

*Note for trim/linearizing: the Dryden turbulence model and the World Magnetic Model 2005 have many internal states, which makes trimming and linearizing difficult. Thus "Bypass" blocks are substituted for the Winds and Magnetic Model blocks using Configurable Subsystems. For the UAV_NL.mdl, the block choices are automatically set to "Bypass" using the models "InitFcn" callback, which can be viewed by using the Model Explorer, selecting UAV_NL, and going to the "Callbacks" tab. The Nonlinear UAV Model block will then show up as a Parameterized Link (red arrow in the lower left corner). The SIL and PIL sims use the default blocks, and should have a normal library link (black arrow).

2.2.4 UAV_NL/Nonlinear UAV Model/Forces and Moments

```
Forces and Moments

This block models all of the relevant external forces and
moments acting on the aircraft. The Aerodynamic Forces and Moments block
contains the aero models, and is a masked subsystem. The input parameter to
this block is a boolean which controls which aero model (either FASER or
the Ultrastick) is used.

The Gravitational Force block models the effect of Earth's graviy field on
the aircraft.

The Electric Propulsion Forces and Moments block contains the electric
motor model.

Note the non-gravitational forces (nonGravForces) are output- this is what
an accelerometer on the aircraft would measure, and is used in the sensor
models.
```

2.2.5 UAV_NL/Nonlinear UAV Model/Auxiliary Equations

```
Auxiliary Equations

This block computes other relevant variables (such as angle of attack,
indicated airspeed, etc) from state data. Part of this block is simply to
assign signals name and create the States bus.

Working from the top of the block downwards:

The inertial velocity, V_e, is computed by using the Direction Cosine
Matrix (DCM) to transform the body-axis velocities (u,v,w) to the
inertial frame. The steady state winds are then added, and the result is
V_e. This is integrated to obtain the inertial position, X_e.  The
initial condition of the X_e integrator is set by
TrimCondition.InertialIni. V_e and X_e are inputs to the Navigation
block- see that blocks README for details.

Euler angles are bounded by +\- [pi/2, pi, 2*pi] respectively. The time
derivatives of the Euler angles are computed using the body-axis rates
```

and the Euler angles.

The DCM is included in the States bus as "R_be [3x3]".

Body axis velocites and rates, and their derivatives are included. The unsteady (turbulence and gusts) winds are added to the body axis velocities and rates.

Inertial accelerations are computed by transforming the body-axis accelerations with the DCM. This step is simplified since we are using a non-rotating Earth.

WindAxesParam is the true airspeed, angle of attack, and sideslip angle. The derivatives of alpha and beta are computed using a deriative block. Mach number is simply the ratio between true airspeed and the speed of sound.

Accels [m/s^2] is what an accelerometer would read onboard the aircraft.

*Note on winds: dividing the wind components into steady and unsteady components is necessary because we do not use an intermediate atmospheric reference frame to account for the motion of the air mass relative to the Earth. We can approximate the physical effects of wind without an additional reference frame by splitting the wind into steady and unsteady components, where the steady component is added to the inertial velocities and the unsteady component is added to the body-axis velocities allows us to

2.2.6 UAV_NL/Nonlinear UAV Model/Auxiliary Equations/Navigation

Navigation

This block is library link that contains the navigational model and equations. Included are equations relating the flat Earth position to latitude/longitude, a simplified 2D table lookup version of the EGM-96 Geoid model for computation of MSL/AGL altitudes, and computation of flight path and ground track angles.

2.2.7 UAV_NL/Nonlinear UAV Model/Forces and Moments/Electric Propulsion Forces and Moments

```
Electric Propulsion Forces and Moments

This block models the electric motor, propeller, and the resulting forces
and moments.

The electric motor is modeled by using a table lookup to relate throttle
position to power output in Watts; power is coverted to torque by
dividing by the current motor angular velocity, omega (rad/s). This
torque is then summed with the required torque from the propeller. The
resulting net torque is divided by the combined motor/propeller inertia,
yielding omega_dot, which is integrated to get the current motor speed.
The initial condition of the Engine speed integrator is set in the
TrimCondition data structure.

The propeller is modelled using lookup tables of Thrust and Power
Coefficients, CT and CP, as a function of advance ratio J. These are
defined as:

CT = Thrust / (R^4 * omega^2 * 4/pi^2 * rho)
CP = Power  / (R^5 * omega^3 * 4/pi^3 * rho)
J  = V * pi / (omega * R)

where R is the propeller radius, omega is the angular velocity in
radians per second, and rho is the density of air. Note that the
torque coefficient for the propeller can be calculated from CP by
multiplying by n; thus the torque coefficient is not explicitly modeled.

The total forces due to the propeller is simply the thrust, which is
not directly aligned with the body axes, so it must be rotated. These
angles are set in the aircraft configuration m-file.

The moments due to the propeller are due to the derivative of the angular
momemtum (ie gyroscopic moments) and moments due to the position of the
thrustline relative to the center of gravity. In addition, the torque from
the motor appears as an applied moment in the equations of motion. The
moments due to the propeller are as follows:

Mp = d/dt(Jmp*omega) + M_motor

where Jmp is the moment of inertia of the rotating portion of the motor
```

```
and propeller. Taking the derivative in the body frame, which is
non-inertial, results in:

Mp = Jmp*omega_dot + [p;q;r] X omega*Jmp + M_motor

where [p;q;r] is the body axis angular velocity. Since the
rotation axis is NOT aligned with the body x-axis, we must rotate these the
angular moment terms by a rotation matrix (L_mb) before computing the cross
product:

Mp = L_mb*(Jmp*omega_dot + M_motor) + [p;q;r] X L_mb*omega*Jmp

The data for the lookup tables and all of the needed aircraft parameters
are stored in the aircraft configuration data structure (AC).

>> AC.Prop

ans =

                    CT: [-0.4314 1.0800 -0.8960 0.1089 0.0604]
                    CP: [0.5054 -0.5304 0.0412 0.0166 0.0223]
                Radius: 0.1524
                 Power: [174.4600 70.1350 -4.3900]
    ThrottleOutputLimit: [1x1 struct]
        OmegaSaturation: [1x1 struct]
                   Jmp: 1.2991e-004
                Angles: [0 0 0.0524]
```

2.2.8 UAV_NL/Nonlinear UAV Model/Forces and Moments/Aerodynamic Forces and Moments/Aero Model/Ultrastick

```
Ultrastick Aerodynamic Model

This block models the Ultrastick aerodynamics using a linear derivatives.
The force coefficients are in the wind axes, but the moment coefficients
are in body axes, so the six coefficients are CL, CDw, CYw, Cl, Cm, Cn.
See Klein, Morelli, Aircraft System Identification, pg 41. Thus in the
Aero lib, the force and moment transformation block must be set as though
outgoing moment coefficients are in wind axes so no transformation is
applied.

The derivatives are stored in the aircraft configuration data structure
```

(AC). They are loaded into AC.Aero when 'Ulstrastick' is selected with UAV_config.m. The derivatives themselves are stored in Ultrastick_config.m. An example is the derivatives for CL:

```
>>  AC.Aero.CL

ans =

        zero: 0.1086
       alpha: 4.5800
       dflap: 0.7400
       delev: 0.0983
    alphadot: 1.9724
           q: 6.1639
        minD: 0.2300
```

The majority of the derivatives have been indentified from flight test data. The drag model was derived from first principles.

3 SIL Simulation: UAV_SIL

3.1 M-Files

3.1.1 SIL_montecarlo.m

```
SIL Simulation Monte Carlo Setup
 The commands in this script will set conditions for the SIL simulation
 for a range of environmental conditions and model uncertainties, run and
 save the simulation data, and compare the results.

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: SIL_montecarlo.m 337 2011-04-15 15:20:27Z murch $
```

3.1.2 compare_sim2flight.m

```
compare_sim2flight.m
```

```
This script will compare flight data to SIL simulation data. Change the
flight data file and simulation data file in the section below titled
"User Input".

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: compare_sim2flight.m 743 2011-12-09 19:19:12Z murch $
```

3.1.3 mkosswp.m

3.1.4 model_check.m

```
model_check.m
UAV_SIL Model Verification

This script runs "plot_and_save.m", which plots the pitch and roll angle
doublet response using the current simulation model and controller. These
results are compared to a stored simulation run of the same inputs using
the baseline controller. Users can use this script to evaluate and
compare the performance of their controller or model relative to a flight
tested baseline.

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: model_check.m 337 2011-04-15 15:20:27Z murch $
```

3.1.5 plot_SIL.m

```
plot_SIL.m
```

```
UAV_SIL sim plot and comparison tool

Input file names of saved simulation results (simData structure) and this
function will co-plot the results. If no file name is input, the file
"simData.mat" will be used.

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: plot_SIL.m 559 2011-09-01 19:44:55Z murch $
```

3.1.6 setup.m

```
setup.m
UAV Software-in-the-Loop Simulation setup

This script will setup the SIL simulation. Stored aircraf configuration
and trim conditions are used.

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: setup.m 725 2011-11-23 21:29:55Z murch $
```

3.1.7 simulate_and_save.m

```
simulate_and_save.m
UAV Software-in-the-Loop simulate_and_save function

This function runs the SIL sim and saves the results to a file with the
given name. If no file name is given, "simData" is used. A simulation
time can also be input; the default is 45 seconds.

University of Minnesota
Aerospace Engineering and Mechanics
```

```
  Copyright 2011 Regents of the University of Minnesota.
  All rights reserved.

  SVN Info: $Id: simulate_and_save.m 638 2011-09-30 20:27:53Z murch $
```

3.1.8 write_sysid_signal_header.m

3.2 Simulink Blocks

3.2.1 UAV_SIL

```
UAV Software-in-the-Loop Simulation

This Simulink model contains a nonlinear UAV model with closed-loop
feedback control provided by a mex-function written in C. Actuator
dynamics and sensor noise are modeled, and the simulation data is
exported to the workspace via the "Flight Data Display" block.

Light blue blocks are a UMN library link; orange blocks are a Simulink
Aerospace Blockset library link. README blocks are green.

1) Make sure that you have a C compiler installed that can interface with
MATLAB. You can setup the default lcc compiler of MATLAB by typing "mex
-setup"

2) Run the file 'setup.m'. The default trim condition will be used.
Change the string variable "control_code_path" to specify the path and name
of the controller source code, or specify a different Variant with the
controller_mode variable.

3) Run the file 'plot_SIL' which will generate plots of the simulation
response. Run 'model_check.m' to compare the current sim response to
stored checkcase data. If the file runs successfully and the plots
corresponded with the provided results, the system should be working
properly.  If you encounter problems with this setup, please contact
descobar@aem.umn.edu.

4) Run the file 'SIL_montecarlo.m' to run and save simulation runs with
```

varying enviromental conditions and aerodynamic model parameters.

5) Run the file 'compare_sim2flight.m' to compare flight data to SIL simulation data.

You can change the trim conditon without changing directories to NL_Sim; simply call trim_UAV normally. Note that the UAV_NL model will be loaded invisibly and will use your current workspace variables.

3.2.2 UAV_SIL/Control Software

Control Law Block

This block uses Model Referencing to allow the user to easily switch out control law implementations. An example usage would be first a user develops a control law using simulink blocks. This controller is referred to as the "simulink controller" and would be selected by setting the variable "controller_mode" to 2. Once the development is completed, the user then implements the Simulink controller into C-code suitable for integration with the UAV software. This controller is referred to as the "flight code" and would be selected by setting "controller_mode" to 1.

This is done using the Simulink.Variant object. See the documentation for more detail. The user can specify any number of variants for the Control Software block; edit the ModelReferenceParameters by right-clicking this block to set which Simulink model is reference for each Variant object. Note that these objects must be present in the base workspace. It is strongly recommended that you use the "simulink_controller.mdl" file as a starting place and "Save As" to a different file name.

In summary, to switch the Control Software, modify the "contoller_mode" variable:
1 = flight code controller (C implementation)
2 = simulink controller (empty simulink model)

See "Software\Documentation\UAV_controllaw_ICD.pdf" for details on the input/output signals.

The control_cmd signal must have the following order:

```
throttle
elevator
rudder
l_aileron
r_aileron
l_flap
r_flap

Control Inputs Sign Convention
TED = Trailing Edge Down
TEL = Trailing Edge Left
------------------------
Elevator: +TED
Rudder: +TEL
Aileron: +TED, da = (da_R-da_L)/2
Throttle: always positive
Flap: +TED


The refrence command signal must have the following order:
phi_ref
theta_ref
Note the trimmed value of pitch angle theta is included in the reference
command. Use the Doublet Generator block (or other signal generating block)
to input reference commands to the Control Software.
```

4 PIL Simulation: UAV_PIL

4.1 M-Files

4.1.1 plot_pil.m

```
plot_pil.m
UAV_PIL sim plot

Input file names of saved simulation results (pilSimData structure) and this
function will co-plot the results. If no file name is input, the file
"pilSimData.mat" will be used.

University of Minnesota
Aerospace Engineering and Mechanics
```

```
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: plot_pil.m 559 2011-09-01 19:44:55Z murch $
```

4.1.2 publish_plots.m

```
Execute this code to publish the plots to a pdf file
```

4.1.3 save_pil_data.m

```
save_pil_data.m
 UAV Processor-in-the-Loop save_pil_data script

This script saves the results to a file with the
name specified in variable savename.

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: save_pil_data.m 722 2011-11-23 19:47:03Z murch $
```

4.1.4 setup.m

```
setup.m
UAV Processor-in-the-Loop Simulation setup

IMPORTANT: Mathworks Real Time Windows Target is only supported for
32-bit machines. http://www.mathworks.com/products/rtwt/requirements.html

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.
```

```
SVN Info: $Id: setup.m 722 2011-11-23 19:47:03Z murch $
```

4.2 Simulink Blocks

4.2.1 UAV_PIL

```
UAV Processor-in-the-Loop Simulation

This Simulink model contains a nonlinear UAV model with hardware interfaces
to connect to the UAV flight computer hardware. Aircraft state data is
visualized via FlightGear and the UAV Ground Control Station software.

Light blue blocks are a UMN library link; orange blocks are a Simulink
Aerospace Blockset library link. README blocks are green. Yellow blocks
indicate an external interface.

You can change the trim conditon without changing directories to NL_Sim;
simply call trim_UAV normally. Note that the UAV_NL model will be loaded
invisibly and will use your current workspace variables.

For the first 10 seconds, trim settings are applied so the operator has a
chance to manually control the aircraft.
```

4.2.2 UAV_PIL/To FlightGear

```
To FlightGear

This block uses the Aerospace Blockset tools to send simulation data to
FlightGear for visualization. See the Simulink documentation for more
details these blocks. The latest FlightGear version supported is v1.9.1.
FlightGear can be running on the same computer as the PIL sim or can be on
another computer connected via LAN (note the destination IP address and
port setting must be updated in this case).

FlightGear is started automatically in the setup.m script, using
"StartFlightGear.bat". Edit this file to set the correct path to your
FlightGear installation.
```

4.2.3 UAV_PIL/MPC5200/Sensor Data to MPC5200 via Serial Port

To MPC5200 (via Serial)

This block streams a binary data packet to the MPC5200B. The packet header was chosen to be the same as the SiRF GPS Binary Protocol for familiarity. The data values are representative of the primary sensor data onboard the aircraft. The control mode (ie manual or auto) is also an input.

Data are sent as double precision values, and are decoded in L_pil_daq.c.

4.2.4 UAV_PIL/MPC5200/Control Inputs from MPC5200 via Serial Port

Control Inputs

This block contains the serial interface to the MPC5200B that recieves the actuator commands. The commands are in the following order, and are sent as double precision (8 bytes each). Units are radians.

Throttle (0-1)
Elevator
Rudder
Left Aileron
Right Aileron
Left Flap
Right Flap

5 Common M-Files

5.1 FASER_config.m

```
function [AC] = FASER_config()


FASER configuration file. Sets aircraft parameters.
Called from: UAV_config.m

University of Minnesota
```

```
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: FASER_config.m 695 2011-11-15 16:17:42Z murch $
```

5.2 UAV_config.m

```
function [AC,Env] = UAV_config(aircraft,savefile)

Defines aircraft parameters. Input desired aircraft and savefile boolean.
Sets Env data structure.

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: UAV_config.m 314 2011-04-05 16:53:30Z murch $
```

5.3 Ultrastick_config.m

```
function [AC] = Ultrastick_config()

Ultra Stick 25e configuration file. Sets aircraft parameters.
Called from: UAV_config.m

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota. All rights reserved.

SVN Info: $Id: Ultrastick_config.m 695 2011-11-15 16:17:42Z murch $
```

5.4 busnames2excel.m

```
function busnames2excel(savename)
```

```
Takes input/output bus signal names from UAV_NL.mdl and stores them in a
Excel file. The default name for this file is 'UAV_sim_ICD.xlsx'.

Output signal names are taken from the "States" and "EnvData" bus
selectors. Input signal names are taken from the "Control Inputs" bus
creator.

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: busnames2excel.m 284 2011-03-03 15:07:19Z murch $
```

5.5 eigpara.m

```
function [wd, T, wn, zeta] = eigpara(lambda)

[wd, T, wn, zeta] = eigparam(lambda)

Return the parameters of a complex eigenvalue
Inputs:
  lambda = a complex eigenvalue
Outputs:
  wd = the damped natural frequency
  T = the period
  wn = the natural frequency
  zeta = the damping

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: eigpara.m 284 2011-03-03 15:07:19Z murch $
```

5.6 linearize_UAV.m

```
[longmod,spmod,latmod,linmodel]=linearize_UAV(OperatingPoint,verbose)
```

```
Linearizes the UAV model about a given operating point using
../NL_Sim/UAV_NL.mdl. This function can be called from any of the three
sim directories. However, this function will use your workspace
variables. Requires the Control System Toolbox and Simulink Control
Design.

Inputs:
  OperatingPoint - Operating point object of a trim condition
  use_uvw        - boolean flag to use u,v,w as linear model outputs
                   instead of V, alpha, beta; defaults to "false"
  verbose        - boolean flag to suppress output; default "true"

Outputs:
  longmod  - longitudinal linear model
  spmod    - short period approximation
  latmod   - lateral directional linear model
  linmodel - full linear model


University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: linearize_UAV.m 648 2011-10-07 17:14:56Z murch $
```

5.7 trim_UAV.m

```
[TrimCondition,OperatingPoint]=trim_UAV(TrimCondition,AC,savefile,verbose)

Trims the UAV simulation to target conditions using ..\NL_Sim\UAV_NL.mdl.
This function can be called from any of the three sim directories.
However, this function will use your workspace variables. Requires
Simulink Control Design.

Set the trim target as shown below.

Inputs:
    TrimCondition - Initial aircraft state, with a structure called
    "target", which has some subset of the following fields:
```

```
    V_s          - True airspeed (m/s)
    alpha        - Angle of attack (rad)
    beta         - Sideslip (rad), defaults to zero
    gamma        - Flight path angle (rad), defaults to zero
    phi          - roll angle (rad)
    theta        - pitch angle (rad)
    psi          - Heading angle (0-360)
    phidot       - d/dt(phi) (rad/sec), defaults to zero
    thetadot     - d/dt(theta) (rad/sec), defaults to zero
    psidot       - d/dt(psi) (rad/sec), defaults to zero
    p            - Angular velocity (rad/sec)
    q            - Angular velocity (rad/sec)
    r            - Angular velocity (rad/sec)
    h            - Altitude above ground level (AGL) (m)
    elevator     - elevator control input, rad.
    aileron      - combined aileron control input, rad. (da_r - da_l)/2
    l_aileron    - left aileron control input, rad
    r_aileron    - right aileron control input, rad
    rudder       - rudder control input, rad.
    throttle     - throttle control input, nd.
    flap         - flap control input, rad. Defaults to fixed at zero.

    AC       - Aircraft configuration structure, from UAV_config.m
    savefile - boolean flag to save trim condition; default "true"
    verbose  - boolean flag to suppress output; default "true"

Outputs:
  TrimCondition  - values of state and control surfaces at trim.
  OperatingPoint - Simulink OperatingPoint object to use with
  linearization

Unspecified variables are free, or defaulted to the values shown above.
To force a defaulted variable to be free define it with an empty matrix.
For example, by default beta=0  but "target.beta=[];" will allow beta
to be free in searching for a trim condition.

Examples:
TrimCondition.target = struct('V_s',17,'gamma',0); % straight and level
TrimCondition.target = struct('V_s',17,'gamma',5/180*pi); % level climb
TrimCondition.target = struct('V_s',17,'gamma',0,...
                    'psidot',20/180*pi); % level turn
TrimCondition.target = struct('V_s',17,'gamma',5/180*pi,...
```

```
                    'psidot',20/180*pi); % climbing turn
TrimCondition.target = struct('V_s',17,'gamma',0,...
                     'beta',5/180*pi); % level steady heading sideslip

Based in part on the trimgtm.m script by David Cox, NASA LaRC
(David.E.Cox@.nasa.gov)

University of Minnesota
Aerospace Engineering and Mechanics
Copyright 2011 Regents of the University of Minnesota.
All rights reserved.

SVN Info: $Id: trim_UAV.m 684 2011-11-09 21:06:52Z murch $
```