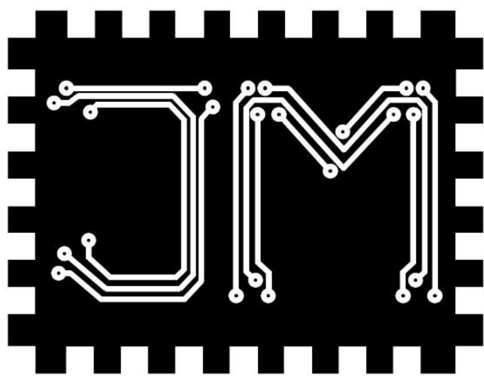

Jeff’s Laboratory

NMSW03 – Flight Manager Estimator

Comments	Revision	Date	Author
Initial Release	A	February 2, 2025	J. Mays

1. References

1. NMSW01 - Flight Manager Sequencer

2. Purpose

This document describes the ESTIMATOR software capability as illustrated in Simulink, and then further discussed in C/C++ software. The ESTIMATOR uses states from the SEQUENCER [1] and the sensor suite to derive flight estimated states such as tilt angle, thrust, and altitude. The downstream logic then uses these states for control and event logic.

Original implementation was written in Simulink and then derived into C/C++ software for compilation onto hardware. Custom Simulink and C/C++ libraries were created to align the two languages. The Mathworks autocoder was not used as this was developed on the home license as a home project.

3. Design Description

The purpose of the Flight Manager (FM) Estimator (Est) software is to read sensor data and convert it into signals that are required for downstream software. This includes information like the attitude of the vehicle during boost, or the rate of ascent. This document will discuss the algorithms and software that make up the ESTIMATOR software.

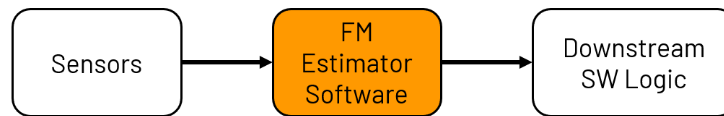

Figure 1: Estimator

4. Interface Control Document

The ESTIMATOR SWC output bus is shown below. This SWC is called by the FLIGHT_MANAGER, and its elements are populated to the vehicle state vector every cycle count.

Table 1: ESTIMATOR input bus

App	Direction	Hierarchy	Element	DataType	Unit	Size	Comment
fm	out	seq	segment	uint16_t	–	1	FM sequencer state
fm	out	seq	state	uint16_t	–	1	FM state
fm	in	sns.imu	gyro_iv_v	float	rad/s	3	IMU measurement of angular velocity in the NAV frame
fm	in	sns.imu	f_iv_v	float	m/s/s	3	IMU measurement of sensed acceleration in the NAV frame, m/s/s
fm	in	sns.baro	press_pa	float	Pa	1	Barometric pressure, Pa
fm	in	sns.baro	temp_c	float	C	1	Measured temperature, C
fm	in	sns.volt_reg	battery_analog	int16_t	–	1	ADC of the voltage divider on the battery
fm	in	sns.volt_reg	bec_analog	int16_t	–	1	ADC of the voltage divider on the battery eliminator circuit
fm	in	sns.volt_reg	servoA_analog	int16_t	–	1	ADC of the voltage divider on the A servo bus
fm	in	sns.volt_reg	servoB_analog	int16_t	–	1	ADC of the voltage divider on the B servo bus
fm	in	sns.volt_reg	nav_analog	int16_t	–	1	ADC of the voltage divider on the nav bus
fm	in	sns.volt_reg	igniter_analog	int16_t	–	1	ADC of the voltage divider on the pyro igniter bus
fm	in	sns.volt_reg	chuteA_analog	int16_t	–	1	ADC of the voltage divider on the A chute pyro
fm	in	sns.volt_reg	chuteB_analog	int16_t	–	1	ADC of the voltage divider on the B chute pyro

Table 2: ESTIMATOR output bus

App	Direction	Hierarchy	Element	DataType	Unit	Size	Comment
fm	out	est.power	battery_voltage	float	V	1	Measured battery voltage
fm	out	est.power	bec_voltage	float	V	1	Measured BEC voltage
fm	out	est.power	servoA_voltage	float	V	1	Measured A servo bus boltage
fm	out	est.power	servoB_voltage	float	V	1	Measured B servo bus boltage
fm	out	est.power	nav_voltage	float	V	1	Nav bus voltage
fm	out	est.power	igniter_voltage	float	V	1	SRM pyro igniter power bus voltage
fm	out	est.power	chuteA_voltage	float	V	1	Chute A pyro voltage
fm	out	est.power	chuteB_voltage	float	V	1	Chute B pyro voltage
fm	out	est.state	quat_uf	float	–	4	Attitude quaternion from UEN to FAB
fm	out	est.state	eul_uf	float	rad	3	Euler attitude from UEN to FAB
fm	out	est.state	tilt	float	rad	1	Vehicle tilt with the vertical axis
fm	out	est.state	heading	float	rad	1	Heading of tilt
fm	out	est.state	accel_sensed_up	float	m/s/s	1	Sensed accel in the up direction with the UEN frame
fm	out	est.state	alt_AGL	float	m	1	Above ground altitude
fm	out	est.state	alt_dot	float	m/s	1	Vertical velocity
fm	out	est.state	mach	float	–	1	Mach number
fm	out	est.state	rho_air	float	kg/ft^3	1	Density of air
fm	out	est.state	qbar	float	Nm^2	1	Dynamic pressure
fm	out	est.nav	omega_ic_c	float	rad/s	1	Inertial sensed angular velocity in the body frame
fm	out	est.nav	accs_ic_c	float	m/s/s	1	Inertial sensed acceleration in the body frame
fm	out	est.nav	bit_complete	bool	–	1	Sensor BIT complete
fm	out	est.mass	mass	float	kg	1	Estimated mass
fm	out	est.mass	com	float	m	3	Estimated center of mass
fm	out	est.mass	inertia	float	kg-ft^2	6	Estimated inertia tensor
fm	out	est.thrust	thrust	float	N	1	Estimated thrust

5. Pre-Configured Gains

Each SWC has an initialization subroutine that initializes parameterized gains. The gains are first defined in Matlab, and then are autocoded into a Cpp file for reference during compilation of the Cpp version of this SWC. The following code snip shows the gain subroutine for `ESTIMATOR`.

```
function [gains] = init_gains_est(config)

% Load common gains
common = init_gains_common([]);

%% Power
gains.power.R1 = 160000 * C_OHM;
gains.power.R2 = 38300 * C_OHM;
gains.power.uC_voltage = 3.3 * C_V;
gains.power.adc_bits = 12;

%% SNS BIT

gains.bit.sns_time = 3.0 * C_SEC;
gains.bit.sns_window_size = 200;
gains.bit.accel_mag = 9.80665 * C_M/C_SEC/C_SEC;

%% Attitude Accel-Gyro Complementary filter

gains.attitude.alpha       = 0.98;
gains.attitude.acc_dyn_env = 0.05 * C_M/C_SEC/C_SEC;

%% Altitude Baro_INS Complementary filter

gains.altitude.alpha = 0.98;
gains.altitude.accel_filter.wn   = 10*C_HZ;
gains.altitude.accel_filter.zeta = sqrt(2)/2;

gains.altitude.hdot_filter.wn    = 10*C_HZ;
```

```
gains.altitude.hdot_filter.zeta  = sqrt(2)/2;

%% Frame locations

% Load our models used to parameterize the sim. And use nominal MC index
mc = SimCore.MonteCarlo(0);
[cnst.vehicle, tnbl.vehicle_tunable] = loadJ2Parameters(mc);
% Load battery with config
% Load SRM with config

% Run the multibody script we use in the simulation to pack mass parameters
% together

% Frame rotations
gains.frame.quat_vf = cnst.vehicle.frame.quat_vf;
gains.frame.quat_fc = [1;0;0;0]; % COM and FAB are aligned always

% NAV frame in the vehicle about FAB
gains.frame.r_fv_f = cnst.vehicle.frame.r_fv_f;
gains.frame.v_fv_f = [0;0;0];
gains.frame.a_fv_f = [0;0;0];

% TVC frame
gains.frame.r_ft_f = cnst.vehicle.gimbal.r_ft_f;

% COM frame in the vehicle about FAB

% FIX ME!!! THIS IS NOT CORRECT!!! See comment above
gains.frame.r_fc_f = single(cnst.vehicle.frame.r_fc_f);
gains.frame.v_fc_f = [0;0;0];
gains.frame.a_fc_f = [0;0;0];

%% Mass
gains.mass.mass = single(tnbl.vehicle_tunable.mass.mass);
% COM defined in frame
gains.mass.inertia = single(tnbl.vehicle_tunable.mass.inertia);

%% Thrust
% Load our models used to parameterize the sim. And use nominal MC index
mc = SimCore.MonteCarlo(0);
[~, srm_tunable] = loadSRMParameters(mc, config);

% We are going to average the thrust vector, and assume thrust is constant
% over the BOOST segment. This way, if our estimate of thrust is incorrect,
% it wont mess up the stability margins

% Equally spaced thrust lookup
thrust = interp1(srm_tunable.time_lookup, srm_tunable.thrust_lookup, linspace(0, max(srm_tunable.time_lookup),
10000));

% Find thrust average for all non-zero datapoints
thrust_nonzero = thrust(thrust>0);
thrust_average = mean(thrust_nonzero);
gains.thrust.thrust_average = single(thrust_average);

% figure; hold on;
% set(gca,'fontname','Consolas')
% plot(srm_tunable.time_lookup, srm_tunable.thrust_lookup, 'LineWidth', 2)
% yline(thrust_average, 'r', 'LineWidth', 2)
% grid on; grid minor;
% xlabel('Time [sec]')
% ylabel('Thrust [N]')
% legend('Thrust Lookup',['Thrust Average (' num2str(thrust_average,3) ' N)'])
% xlim([0 max(srm_tunable.time_lookup)+1])

%% State
% Need speed of sound to derive mach number
gains.state.speed_of_sound = 343.0 * C_M / C_SEC;

end
```

6. Software Logic

A high-level capture of the `FLIGHT_MANAGER` software logic is shown in Figure 2. The Simulink `ESTIMATOR` software capability is highlighted.

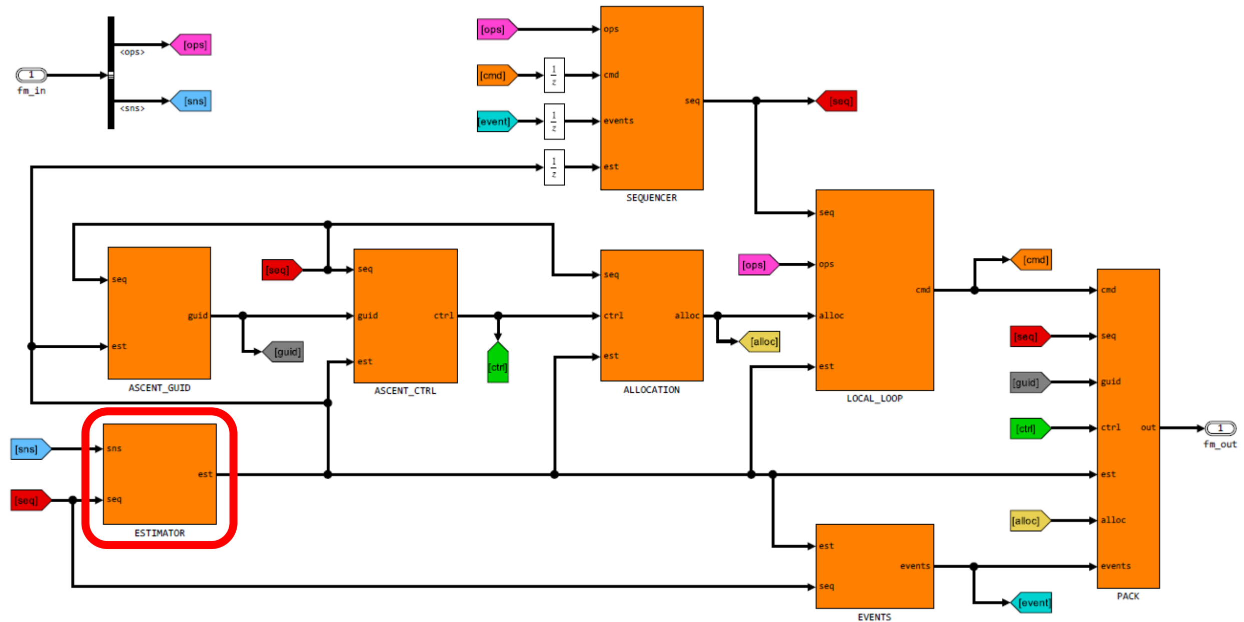

Figure 2: Flight Manager

The ESTIMATOR software capability is shown in Figure 3. It is broken up into `NAV`, `STATE`, `THRUST`, and `POWER` subsystems. Each of these subsystems will be discussed in the following section.

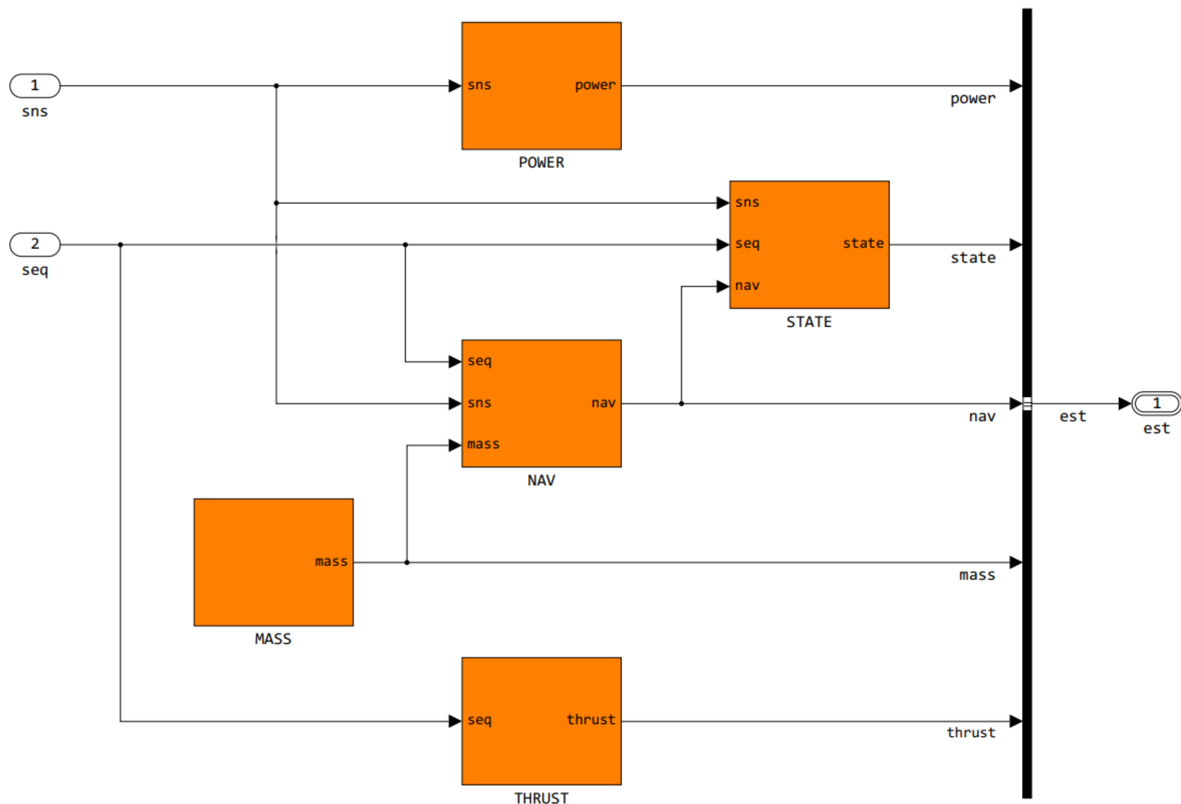

Figure 3: EVENTS software capability

6.1. NAV

The navigation subsystem is shown in Figure 4. This system has two functionalities

1. Performs the gyroscope and accelerometer calibration. This is intended to be performed when the vehicle is not moving and completely upright with the body axial axis aligned with the up local geographic frame.
2. Kinematically transitions the sensed states, which are sourced from the navigation (NAV) frame to the center of mass (COM) frame.

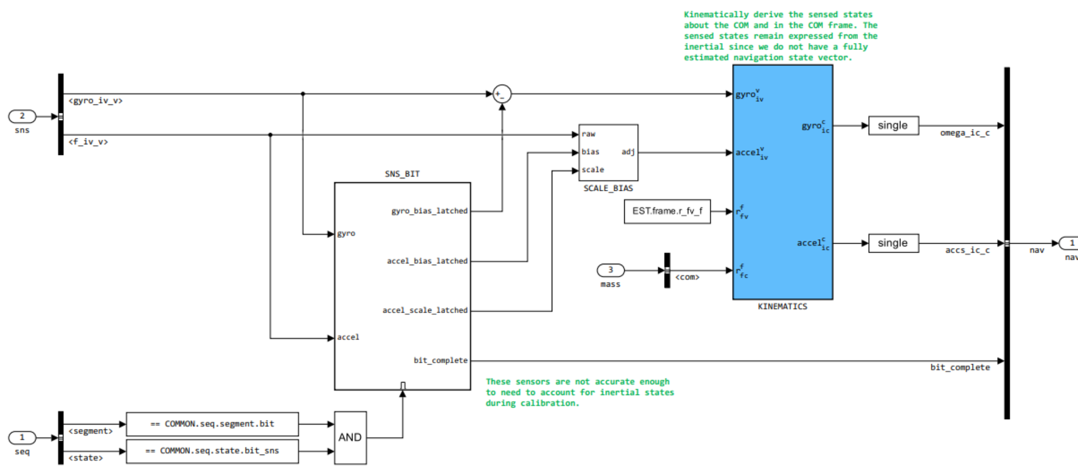

Figure 4:ESTIMATOR/NAV

The SNS_BIT and the ACCEL_BIAS_SCALE blocks are shown in Figure 5 and Figure 6. The bias and scale of the gyroscope and accelerometer are found through the following equations

$$\boldsymbol{\omega}_{bias} = \widehat{\boldsymbol{\omega}}_{sensed} \tag{1}$$

$$a_{i,scale} = \left[g_{mag} * \frac{\widehat{\boldsymbol{a}}_{sensed}}{\|\widehat{\boldsymbol{a}}_{sensed}\|} \right]_i / \hat{a}_{i,sensed} \tag{2}$$

$$\boldsymbol{a}_{bias} = \widehat{\boldsymbol{a}}_{sensed} * a_{i,scale} - \widehat{\boldsymbol{a}}_{pad} \tag{3}$$

where $\widehat{\boldsymbol{\omega}}_{sensed}$ and $\widehat{\boldsymbol{a}}_{sensed}$ are moving average vectors of the gyroscope and accelerometer measurements, g_{mag} is the desired magnitude of the sensed acceleration vector, $a_{i,scale}$ is the estimated accelerometer scape factor, and $\widehat{\boldsymbol{a}}_{pad}$ is the expected accelerometer reading while on the pad (using a simplified assumption that is sufficient for this application). We also output a BIT complete flag to indicate when the bias has been found. This flag is set to true after a certain time passes. Should the SNS BIT be re-performed, the states of the system reset, allowing the process to occur again.

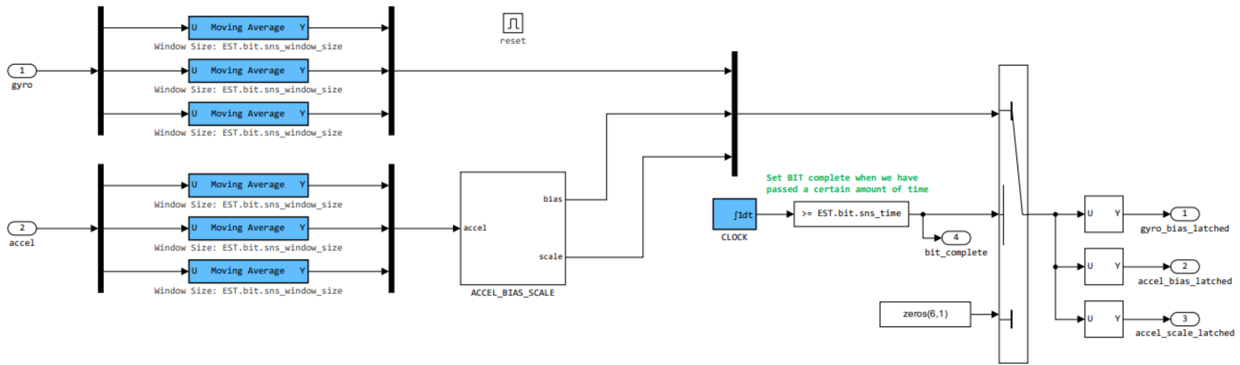

Figure 5: ESTIMATOR/NAV/SNS_BIT

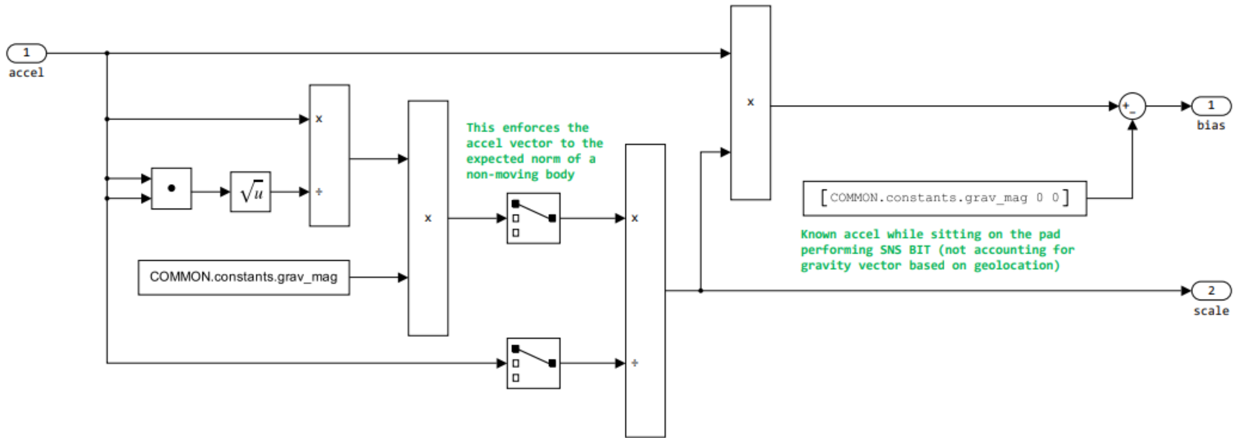

Figure 6: ESTIMATOR/NAV/SNS_BIT/ACCEL_BIAS_SCALE

The KINEMATICS library block is used to move the sensed angular velocity and linear acceleration from the NAV frame to the COM frame. We can use the DCM, C_v^f, from the NAV frame, v, to the FAB frame, f to rotate the sensed states into the COM frame. If we assume the body is rigid, then we can simply multiply the sensed angular velocity by the DCM as shown in the following equation. Note that C_f^c is an identity matrix that describes the FAB frame to the COM frame.

$$\boldsymbol{\omega}_{ic}^{c} = C_f^c C_v^f \boldsymbol{\omega}_{iv}^{v} \tag{4}$$

Given we know the rigid parameters of the vehicle, we can also derive the sensed acceleration at the center of mass as a function of the sensed acceleration at the NAV frame minus the acceleration effects from off-axis rotations.

$$\boldsymbol{a}_{ic}^{c} = C_f^c C_v^f (\boldsymbol{a}_{iv}^{v}) - C_f^c (\boldsymbol{\alpha}_{iv}^{f} \times \boldsymbol{r}_{cv}^{f} - \boldsymbol{\omega}_{iv}^{f} \times \boldsymbol{\omega}_{iv}^{f} \times \boldsymbol{r}_{cv}^{f}) \tag{5}$$

Figure 7 illustrates the Simulink code that contains the above formulations.

```
function [gyro_ic_c, accel_ic_c] = kinematics_nav_to_com(gyro_iv_v, accel_iv_v, r_fv_f, r_fc_f, a_fv_f, v_fv_f,
quat_vf, v_fc_f, a_fc_f, quat_fc)
% Simple kinematic equation to move the sensed states from a position
% on a rigid body to another position within that rigid body.
%
% Inputs:
%   gyro_iv_v   Inertial angular velocity sensed at NAV in NAV
%   accel_iv_v  Inertial accelerometer sensed at NAV in NAV
%   quat_vf     Quaternion from NAV to FAB
%   r_fv_f      Position vector from FAB to NAV, in m
%   v_fv_f      Velocity vector from FAB to NAV, in m/s
%   a_fv_f      Acceleration vector from FAB to NAV, in m/s
%   quat_fc     Quaternion from COM to FAB
%   r_fc_f      Position vector from FAB to COM, in m
%   v_fc_f      Velocity vector from FAB to COM, in m/s
%   a_fc_f      Acceleration vector from FAB to COM, in m/s
%
% Outputs:
%   gyro_ic_c   Inertial angular velocity sensed at COM in COM
%   accel_ic_c  Inertial accelerometer sensed at COM in COM
%
% Note 1: Differentiating the gyroscope adds a lot of noise. We could
% filter this, but for now just ignore it because it shouldn't really matter
% for this application. To-do.
%
% Note 2: We can't take out the inertial components because we do not know
% the state of the vehicle within the inertial frame
%
% Author: Jeff Mays

% Ensure shape
quat_vf     = reshape(quat_vf,    [4,1]);
quat_fc     = reshape(quat_fc,    [4,1]);
gyro_iv_v   = reshape(gyro_iv_v,  [3,1]);
accel_iv_v  = reshape(accel_iv_v, [3,1]);
r_fv_f      = reshape(r_fv_f,     [3,1]);
r_fc_f      = reshape(r_fc_f,     [3,1]);
a_fv_f      = reshape(a_fv_f,     [3,1]);
v_fv_f      = reshape(v_fv_f,     [3,1]);
v_fc_f      = reshape(v_fc_f,     [3,1]);
a_fc_f      = reshape(a_fc_f,     [3,1]);

% Find relative motion
r_cv_f = r_fv_f - r_fc_f;
v_cv_f = v_fv_f - v_fc_f;
a_cv_f = a_fv_f - a_fc_f;

% Find frame relationships
C_v_f = quat_ab_to_C_a_b(quat_vf);
C_f_c = quat_ab_to_C_a_b(quat_fc);

% Move inputs from NAV to FAB
gyro_iv_f  = C_v_f * gyro_iv_v;
accel_iv_f = C_v_f * accel_iv_v;

% Assume alpha_iv_f is zero
alpha_iv_f = zeros(3,1);

% Move from NAV to COM
rel_frame_accel    = a_cv_f;
sensed_accel_total = accel_iv_f;
eul_accel          = cross(alpha_iv_f, r_cv_f);
centripetal_accel  = cross(gyro_iv_f, cross(gyro_iv_f, r_cv_f));
coriolis_accel     = 2 * cross(gyro_iv_f, v_cv_f);
```

```
% Remove body frame effects
accel_ic_f = sensed_accel_total - rel_frame_accel - eul_accel - centripetal_accel - coriolis_accel;
accel_ic_c = reshape(C_f_c * accel_ic_f, [3,1]);

% Assume vehicle is rigid
gyro_ic_c = reshape(C_f_c * gyro_iv_f, [3,1]);

end
```

Figure 7: ESTIMATOR/NAV/KINEMATICS

6.2. STATE

The state submodule estimates the states of the vehicle, providing estimates of parameters such as dynamic pressure, tilt off vertical, and altitude.

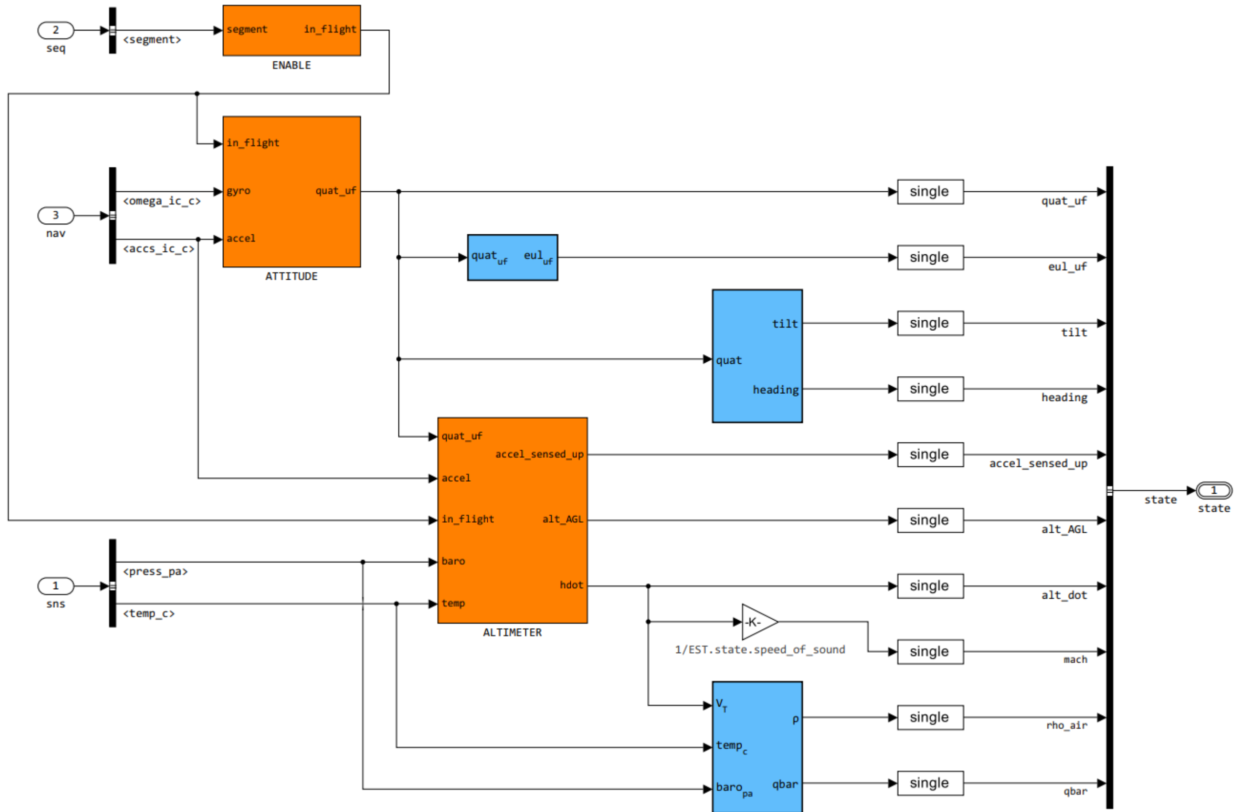

Figure 8: ESTIMATOR/STATE

The enable submodule is used to tell the system when it is in a flight like state.

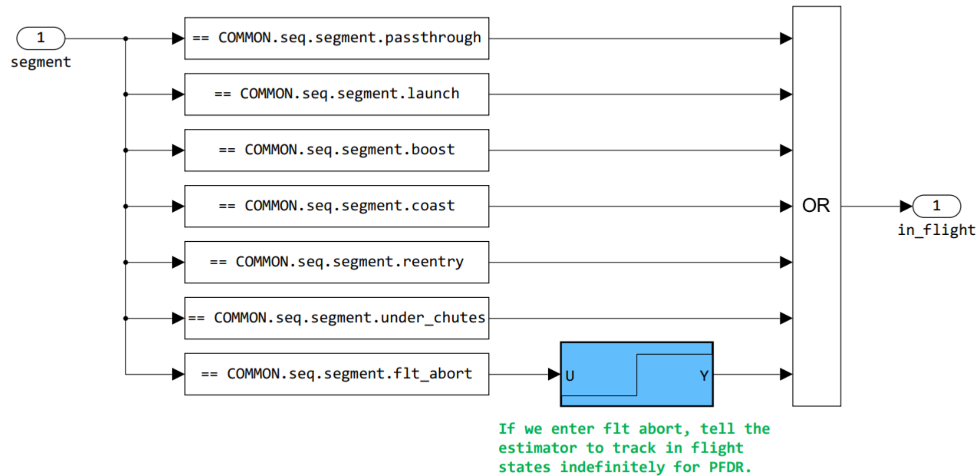

Figure 9: ESTIMATOR/STATE/ENABLE

The attitude submodule is used to estimate the attitude of the vehicle in the vertical local geographic frame. It does this by utilizing a complementary attitude filter as well as a simple gyroscope integrator to derive an attitude quaternion from the UEN frame to the FAB frame.

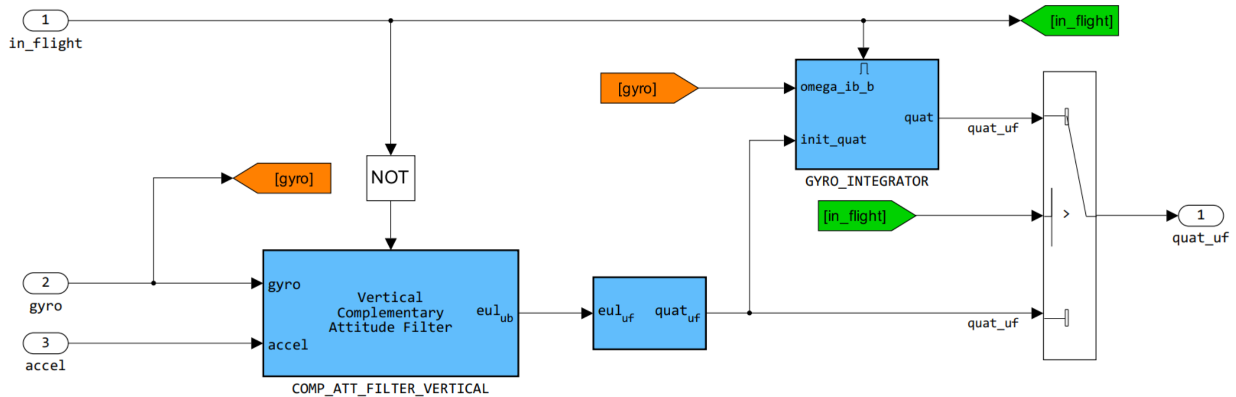

Figure 10: ESTIMATOR/STATE/ATTITUDE

```
function q = eulR_to_quat(eulR)
    % RPY Euler in radians to quaternion
    %
    % Author: Jeff Mays
    % Euler to Quaternion (Page 52 of Aircraft Control and Simulation by Stevens, Lewis, and Johnson 3rd. Edition

    cy = cos(eulR(3) * 0.5);
    sy = sin(eulR(3) * 0.5);
    cp = cos(eulR(2) * 0.5);
    sp = sin(eulR(2) * 0.5);
    cr = cos(eulR(1) * 0.5);
    sr = sin(eulR(1) * 0.5);

    q = zeros(4,1);
    q(1) = cr * cp * cy + sr * sp * sy;
    q(2) = sr * cp * cy - cr * sp * sy;
    q(3) = cr * sp * cy + sr * cp * sy;
    q(4) = cr * cp * sy - sr * sp * cy;

    % Normalize
    q = q / sqrt(q'*q);
end
```

Figure 11: ESTIMATOR/STATE/ ATTITUDE/eulR_to_quat

Below is the complementary filter used for estimating attitude. Internally, it can also simply integrate the gyro or use both the accelerometer and gyroscope to derive attitude.

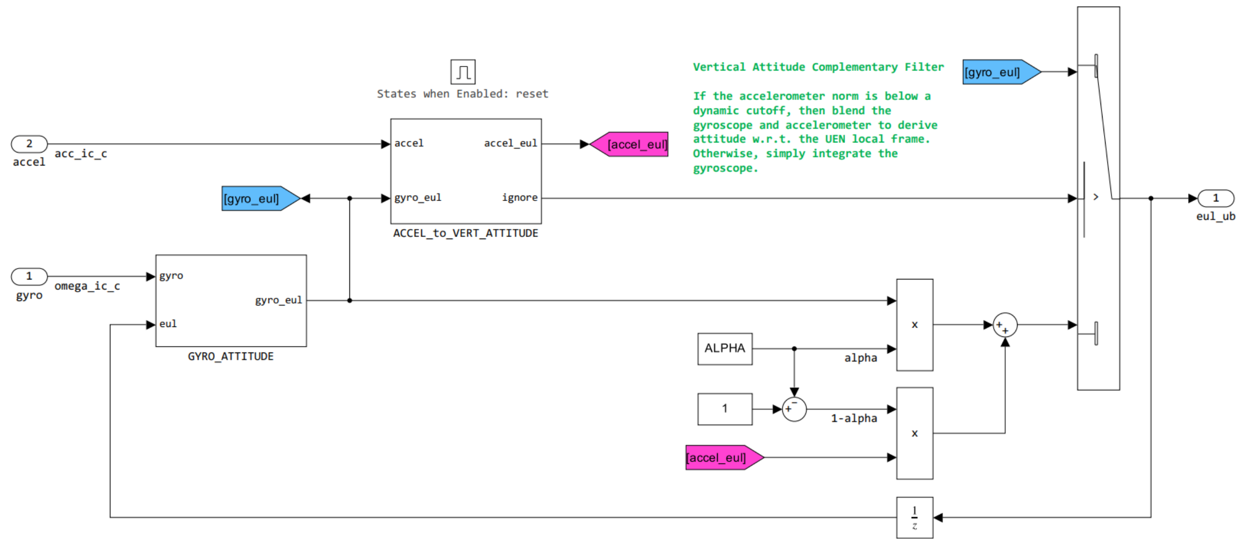

Figure 12: ESTIMATOR/STATE/ ATTITUDE/COMP_ATT_FILTER_VERTICAL

Should the gyro be integrated, we use simple forward Euler integration as shown below.

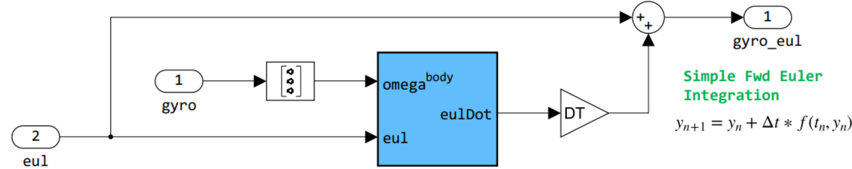

Figure 13: ESTIMATOR/STATE/ ATTITUDE/GYRO_ATTITUDE

```
function eulDot = gyro_to_eulDot(omega, eul)

% Gyro to Euler rate
phi   = eul(1);
theta = eul(2);
psi   = eul(3); %#ok<NASGU>

PHI = [1    sin(phi)*tan(theta)     cos(phi)*tan(theta); ...
       0    cos(phi)               -sin(phi); ...
       0    sin(phi)/cos(theta)     cos(phi)/cos(theta)];

eulDot = PHI * omega;

end
```

Figure 14: ESTIMATOR/STATE/ ATTITUDE/gyro_to_eulDot

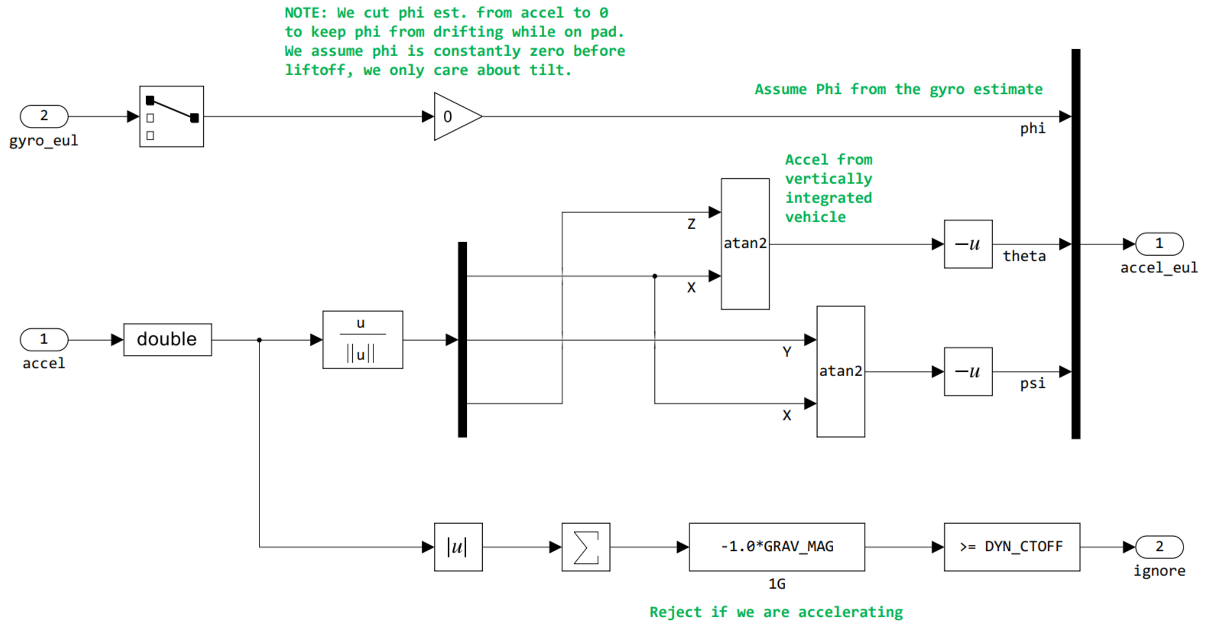

Figure 15: ESTIMATOR/STATE/ ATTITUDE/ACCEL_TO_VERT_ATTITUDE

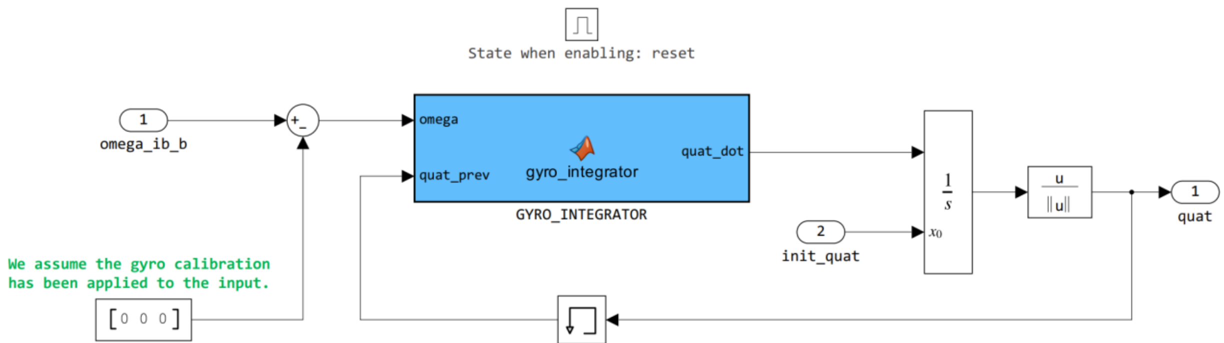

Figure 16: ESTIMATOR/STATE/ATTITUDE/GYRO_INTEGRATOR

```
function quat_dot = gyro_integrator(omega, quat_prev)

% Pre-load
quat_dot = [1; 0; 0; 0]; %#ok<NASGU>

% Reshape
omegap = reshape(omega, [3,1]);

% Quat matrix
Aq = [-quat_prev(2)     -quat_prev(3)    -quat_prev(4); ...
       quat_prev(1)     -quat_prev(4)     quat_prev(3); ...
       quat_prev(4)      quat_prev(1)    -quat_prev(2); ...
      -quat_prev(3)      quat_prev(2)     quat_prev(1)];

quat_dot = 0.5 * Aq * omegap;
end
```

Figure 17: ESTIMATOR/STATE/ATTITUDE/GYRO_INTEGRATOR/gyro_integrator

We similarly have a complementary filter on the vertical altitude estimate where we blend the low frequency barometer measurements with the high frequency acceleration measurements.

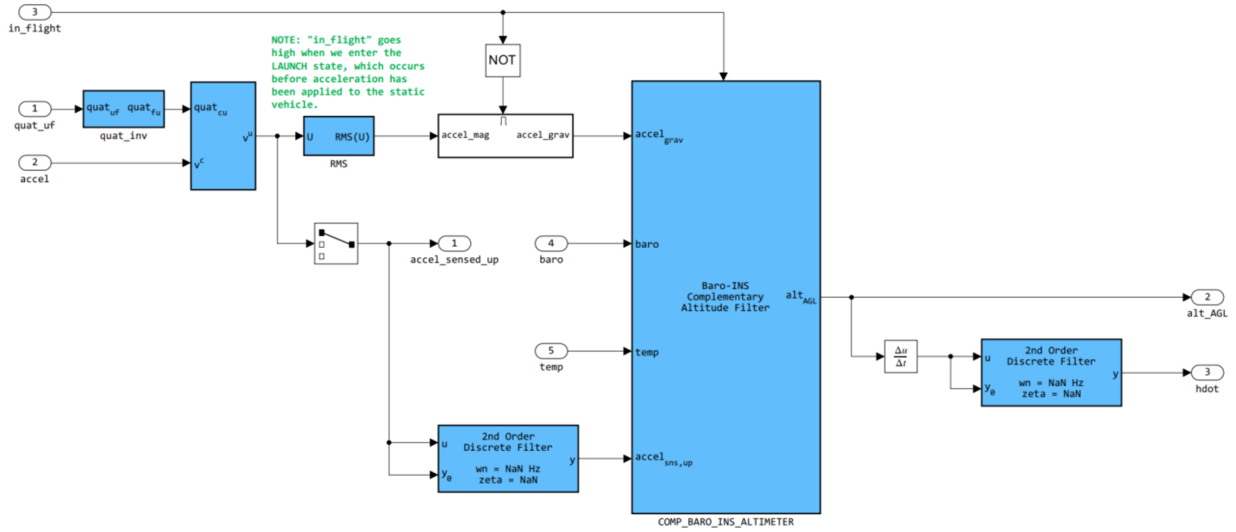

Figure 18: ESTIMATOR/STATE/ATLTITUDE

Within this submodule, we need to be able to filter the system accordingly.

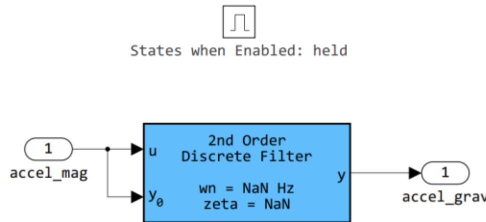

Figure 19: ESTIMATOR/STATE/ALTITUDE/enable

The hypsometric equation is used to derive the altitude we are at given a datum of pressure from the barometer.

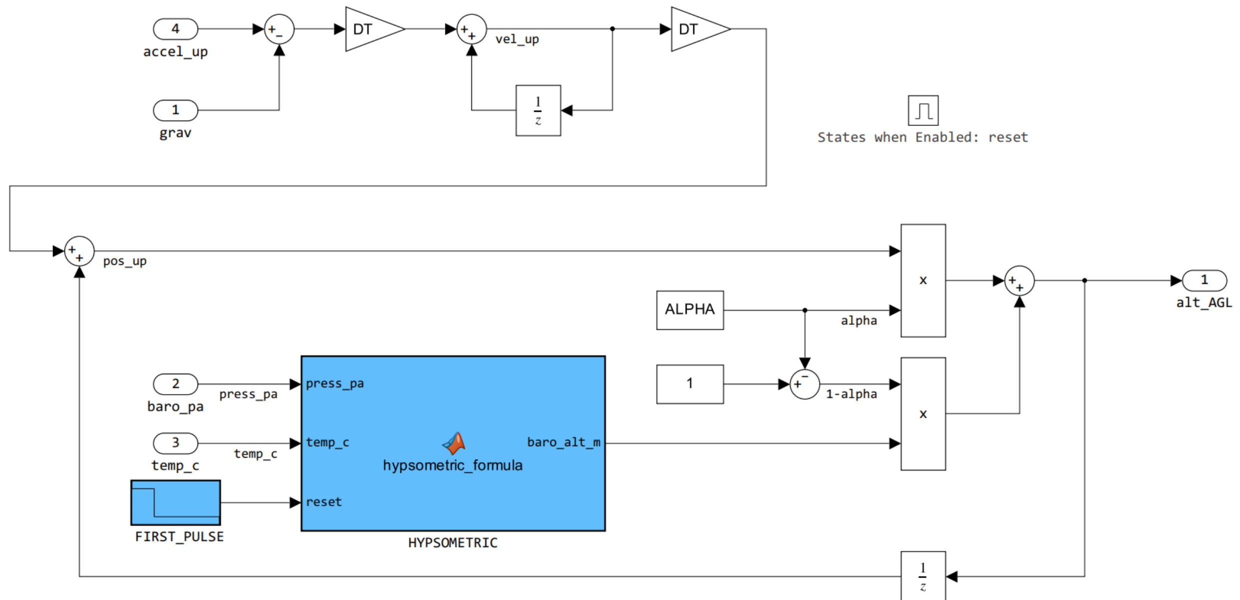

Figure 20: ESTIMATOR/STATE/ALTITUDE/COMP_BARO_INS_ALTIMETER

```
function baro_alt_m = hypsometric_formula(press_pa, temp_c, reset)
% Hypsometric formula
%
% Author: Jeff Mays

% Force datatypes
press_pa = single(press_pa);
temp_c   = single(temp_c);
reset    = logical(reset);

% Const
C_to_K = 273.15;

persistent base_pa
if reset || isempty(base_pa)
    base_pa = press_pa;
end

% Hypsometric
baro_alt_m = ((base_pa / press_pa) ^ (1/5.257) - 1) * (temp_c + C_to_K) / (0.0065);

end
```

Figure 21: ESTIMATOR/STATE/ALTITUDE/COMP_BARO_INS_ALTIMETER/hypsometric_formula

```
function [tilt, heading] = quat_to_tilt_heading(quat_ab)
% Tilt and Heading off a resolved quaternion

% Get DCM
C_a_b = quat_ab_to_C_a_b(quat_ab);
vb = C_a_b * [1; 0; 0];

% tilt
tilt = acos(vb(1) / sqrt(dot(vb, vb)));

% heading
heading = atan2(vb(2), vb(3));

end
```

Figure 22: ESTIMATOR/STATE/quat_to_tilt_heading

```
function [rho, qbar] = derive_qbar(vel, temp_c, baro_pa)
% This function attempts to determine the air density and dynamic pressure
% of the vehicle.

% Const
Rd = 287.058; % J/(kg*K), Specific gas constant for dry air
C_to_K = 273.15; % C to K

% rho = P / (R * T)
temp_k = temp_c + C_to_K;
rho = baro_pa / (Rd * temp_k);

% qbar = 1/2 * rho * v ^ 2
qbar = 0.5 * rho * vel ^ 2;

end
```

Figure 23: ESTIMATOR/STATE /derive_qbar

6.3. THRUST

Thrust is first considered non-zero when the pyro ignitor is lit during the launch segment. Thrust is then non-zero until the end of Boost where the EVENTS SWC detects SRM burnout. The thrust estimate is directly used downstream in the ALLOCATOR SWC to represent TVC effectiveness in the effector mixing routine. We use a constant average of the expected thrust so that we do not need to consider the change in effectiveness of the TVC during the burn. It also aids in robustness should the expected thrust profile deviate from the realized profile. The controller uses gain scheduling during flight; therefore, this simplification should bake into the entire closed loop system.

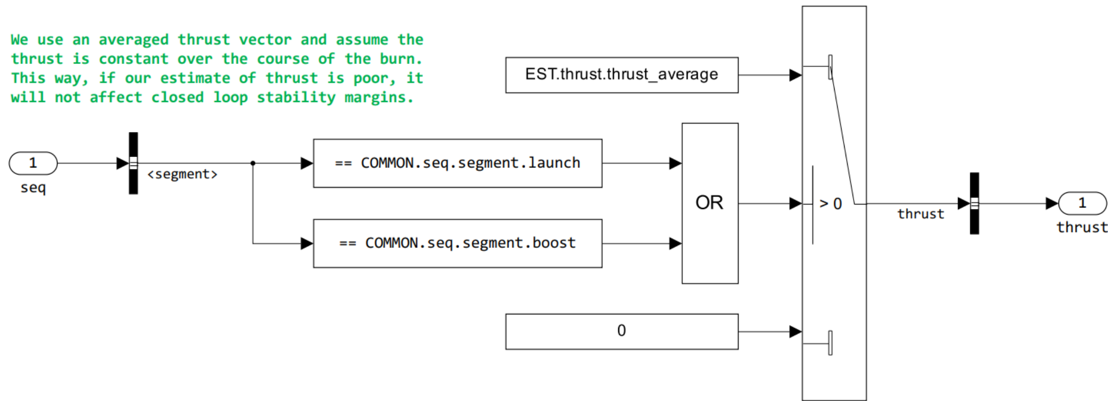

Figure 24: ESTIMATOR/THRUST

6.4. MASS

The mass estimation software is the simplest of the software inside the estimator. Even though there are elements of the vehicle that are changing mass during the flight, for this application, we only need to know a representative set of characteristics. These values are used downstream in the ALLOCATOR and keeping these values constant makes the controller loop-gain easier to manage.

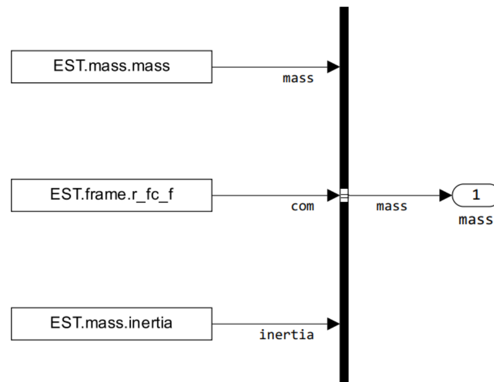

Figure 25: ESTIMATOR/MASS

6.5. POWER

The power subsystem consumes the analog voltage dividers that are connected to the various power systems and calculates the voltage that exists on that system. These include the battery, battery eliminator circuit (BEC), A and B servo buses, navigator power, ignitor power, and A and B chute pyro power. We can use the voltages on these lines to infer if a power switching mosphet worked, or if a particular subsystem has sufficient power to work.

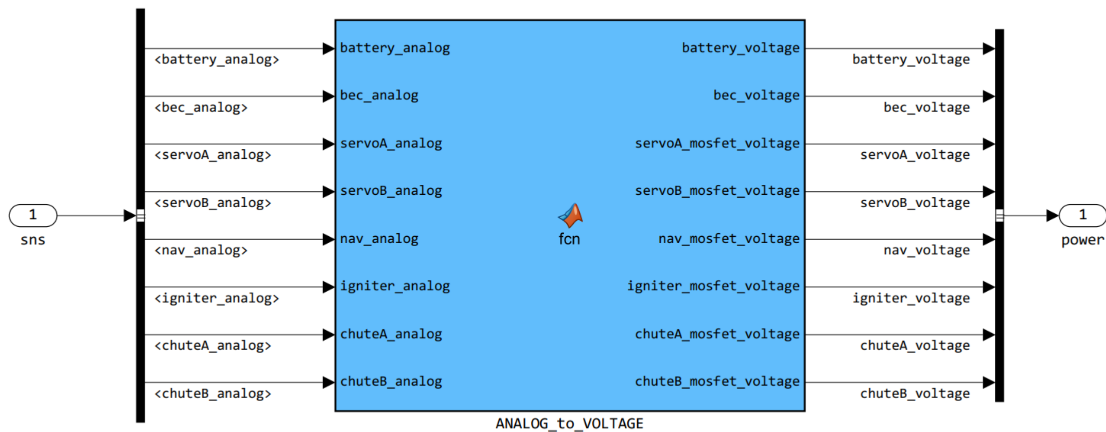

Figure 26: ESTIMATOR/POWER

```
function [battery_voltage, bec_voltage, servoA_mosfet_voltage, servoB_mosfet_voltage, ...
          nav_mosfet_voltage, igniter_mosfet_voltage, chuteA_mosfet_voltage, ...
          chuteB_mosfet_voltage] = fcn_analog_to_voltage(battery_analog, bec_analog, ...
          servoA_analog, servoB_analog, nav_analog, igniter_analog, chuteA_analog, ...
          chuteB_analog, adc_bits, uC_voltage, R2, R1)

% Adc range
adc_range = 2^adc_bits-1;

% Determine voltages
battery_voltage        = single(battery_analog * (uC_voltage / adc_range) / (R2 / (R1 + R2)));
bec_voltage            = single(bec_analog     * (uC_voltage / adc_range) / (R2 / (R1 + R2)));
servoA_mosfet_voltage  = single(servoA_analog  * (uC_voltage / adc_range) / (R2 / (R1 + R2)));
servoB_mosfet_voltage  = single(servoB_analog  * (uC_voltage / adc_range) / (R2 / (R1 + R2)));
nav_mosfet_voltage     = single(nav_analog     * (uC_voltage / adc_range) / (R2 / (R1 + R2)));
igniter_mosfet_voltage = single(igniter_analog * (uC_voltage / adc_range) / (R2 / (R1 + R2)));
chuteA_mosfet_voltage  = single(chuteA_analog  * (uC_voltage / adc_range) / (R2 / (R1 + R2)));
chuteB_mosfet_voltage  = single(chuteB_analog  * (uC_voltage / adc_range) / (R2 / (R1 + R2)));

end
```

Figure 27: ESTIMATOR/POWER /ANALOG_to_VOLTAGE