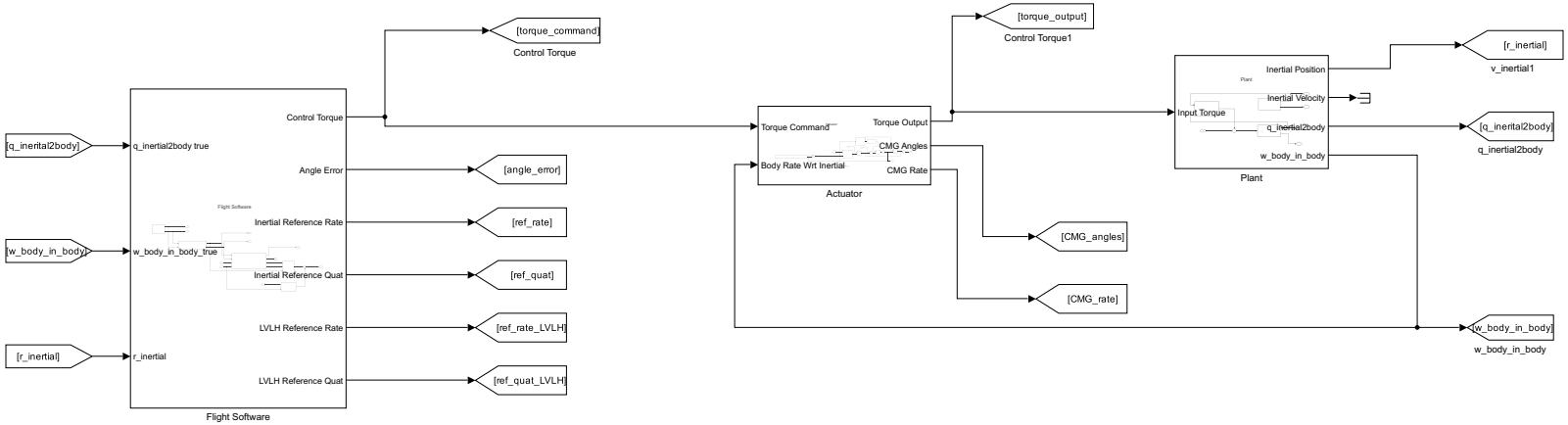
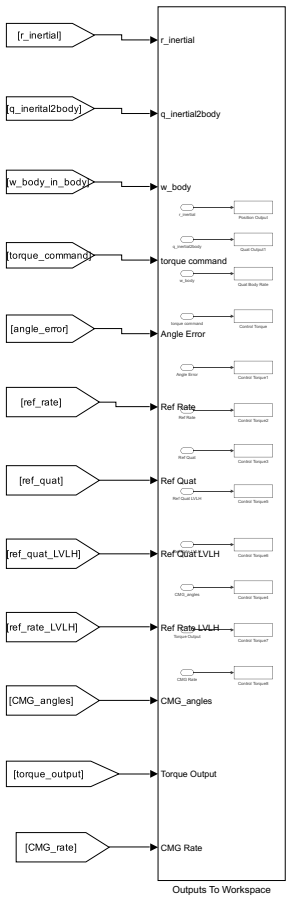
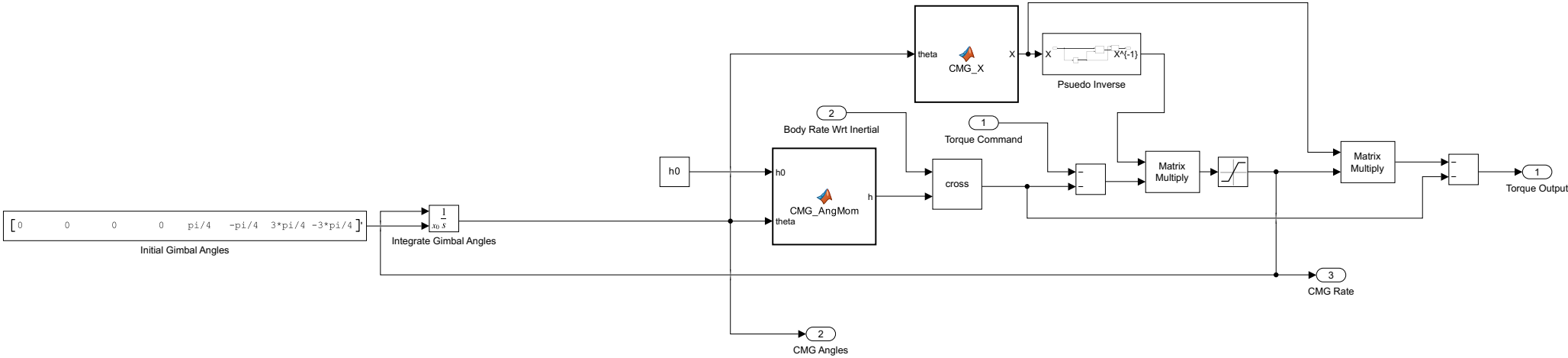

[torque_command]
Control Torque
[q_inertial2body]
q_inertial2body true
Control Torque
Angle Error
[angle_error]
Inertial Reference Rate
[ref_rate]
[w_body_in_body]
w_body_in_body_true
Inertial Reference Quat
[ref_quat]
LVLH Reference Rate
[ref_rate_LVLH]
[r_inertial]
r_inertial
LVLH Reference Quat
[ref_quat_LVLH]
Flight Software
[torque_output]
Control Torque1
Torque Command
Torque Output
CMG Angles
Body Rate Wrt Inertial
CMG Rate
Actuator
[CMG_angles]
[CMG_rate]
Inertial Position
Inertial Velocity
Input Torque
q_inertial2body
w_body_in_body
Plant
[r_inertial]
v_inertial1
[q_inertial2body]
q_inertial2body
[w_body_in_body]
w_body_in_body

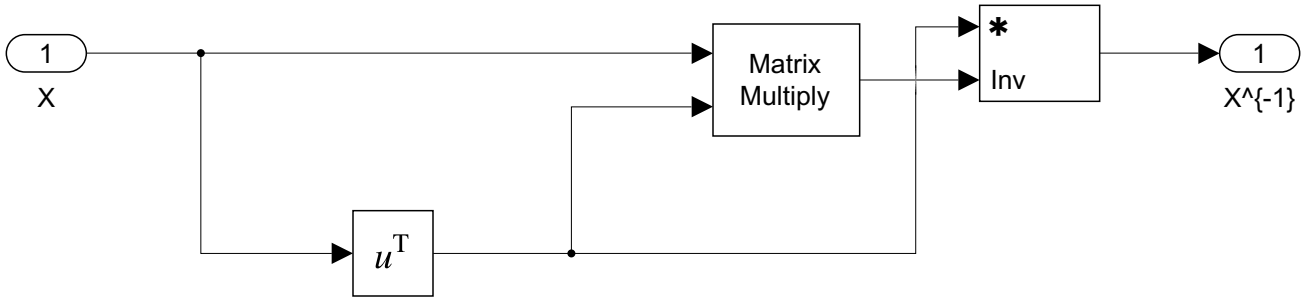

[r_inertial]
r_inertial
[q_inertial2body]
q_inertial2body
[w_body_in_body]
w_body
[torque_command]
torque command
[angle_error]
Angle Error
[ref_rate]
Ref Rate
[ref_quat]
Ref Quat
[ref_quat_LVLH]
Ref Quat LVLH
[ref_rate_LVLH]
Ref Rate LVLH
[CMG_angles]
CMG_angles
[torque_output]
Torque Output
[CMG_rate]
CMG Rate
Outputs To Workspace

Actuator

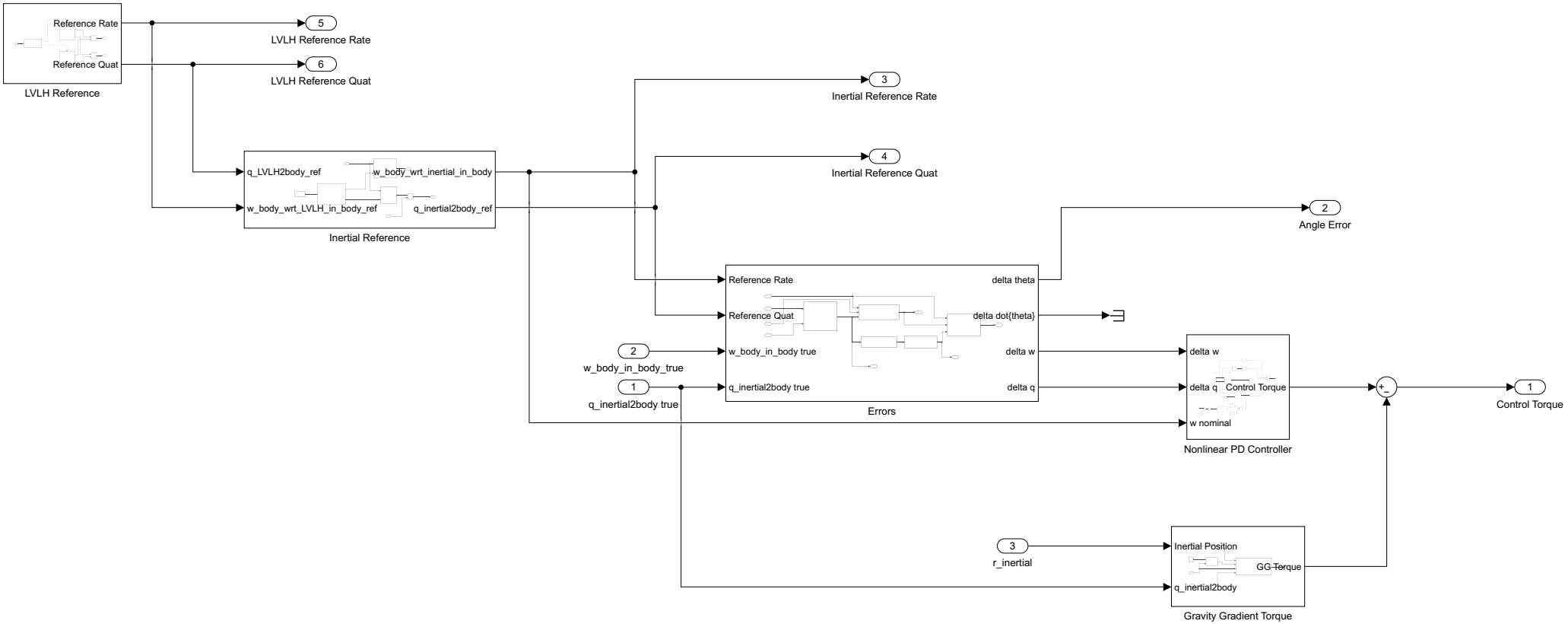

[0 0 0 0 pi/4 -pi/4 3*pi/4 -3*pi/4]'
Initial Gimbal Angles
1/s
Integrate Gimbal Angles
h0
theta
CMG_X
X
X
X^{-1}
Psuedo Inverse
2
Body Rate Wrt Inertial
1
Torque Command
h0
theta
CMG_AngMom
h
cross
Matrix Multiply
Matrix Multiply
1
Torque Output
3
CMG Rate
2
CMG Angles

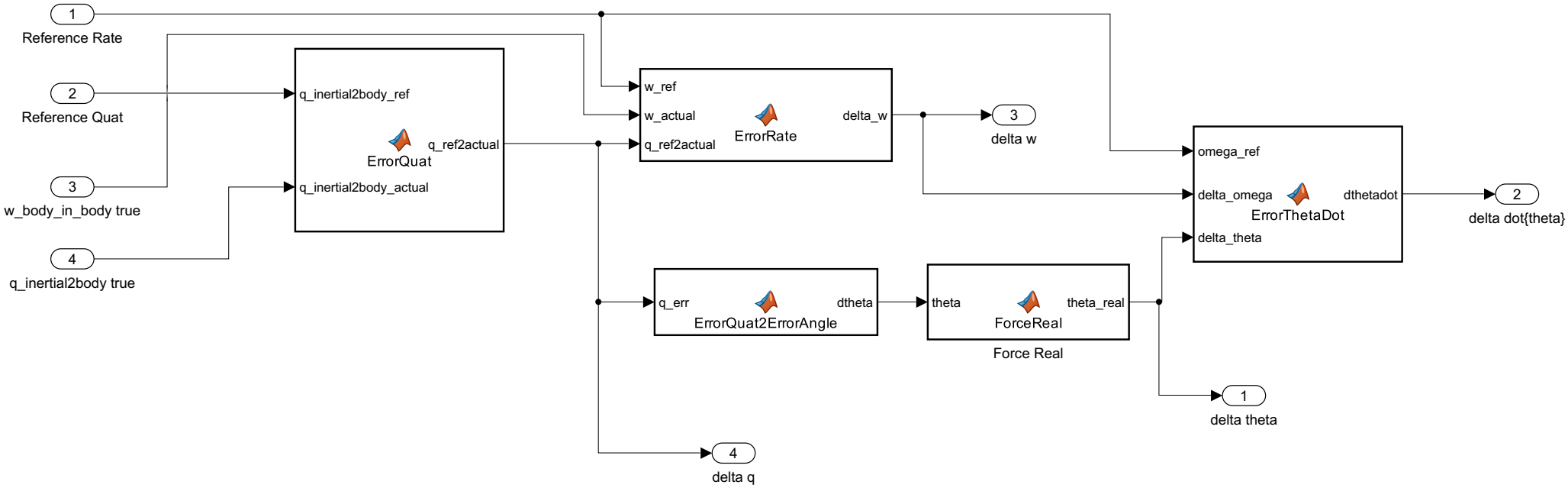
```
function h = CMG_AngMom(h0, theta)
% Produces the angular momentum vector for the CMGs

% Extract alpha and betas
alphas = theta(1:4);
betas = theta(5:8);

h = zeros(3,1);
for ii = 1:4
    h = h + h0*[sin(alphas(ii)); cos(alphas(ii))*cos(betas(ii)); cos(alphas(ii))*sin(betas(ii))];
end
```

```
function X = CMG_X(theta)
% Produces the matrix of CMG spin axes

% Extract alpha and betas
alphas = theta(1:4);
betas = theta(5:8);

X = zeros(3,8);
for ii = 1:4
    X(:,2*ii-1:2*ii) = [cos(alphas(ii)), 0;
        -sin(alphas(ii))*cos(betas(ii)), -cos(alphas(ii))*sin(betas(ii));
        -sin(alphas(ii))*sin(betas(ii)), cos(alphas(ii))*cos(betas(ii))];
end
```


1
X
Matrix
Multiply
*
Inv
1
X^{-1}
u^{T}

Flight Software




1
Reference Rate
2
Reference Quat
3
w_body_in_body true
4
q_inertial2body true
q_inertial2body_ref
q_inertial2body_actual
ErrorQuat
q_ref2actual
w_ref
w_actual
q_ref2actual
ErrorRate
delta_w
3
delta w
omega_ref
delta_omega
delta_theta
ErrorThetaDot
dthetadot
2
delta dot{theta}
q_err
ErrorQuat2ErrorAngle
dtheta
theta
ForceReal
theta_real
Force Real
1
delta theta
4
delta q

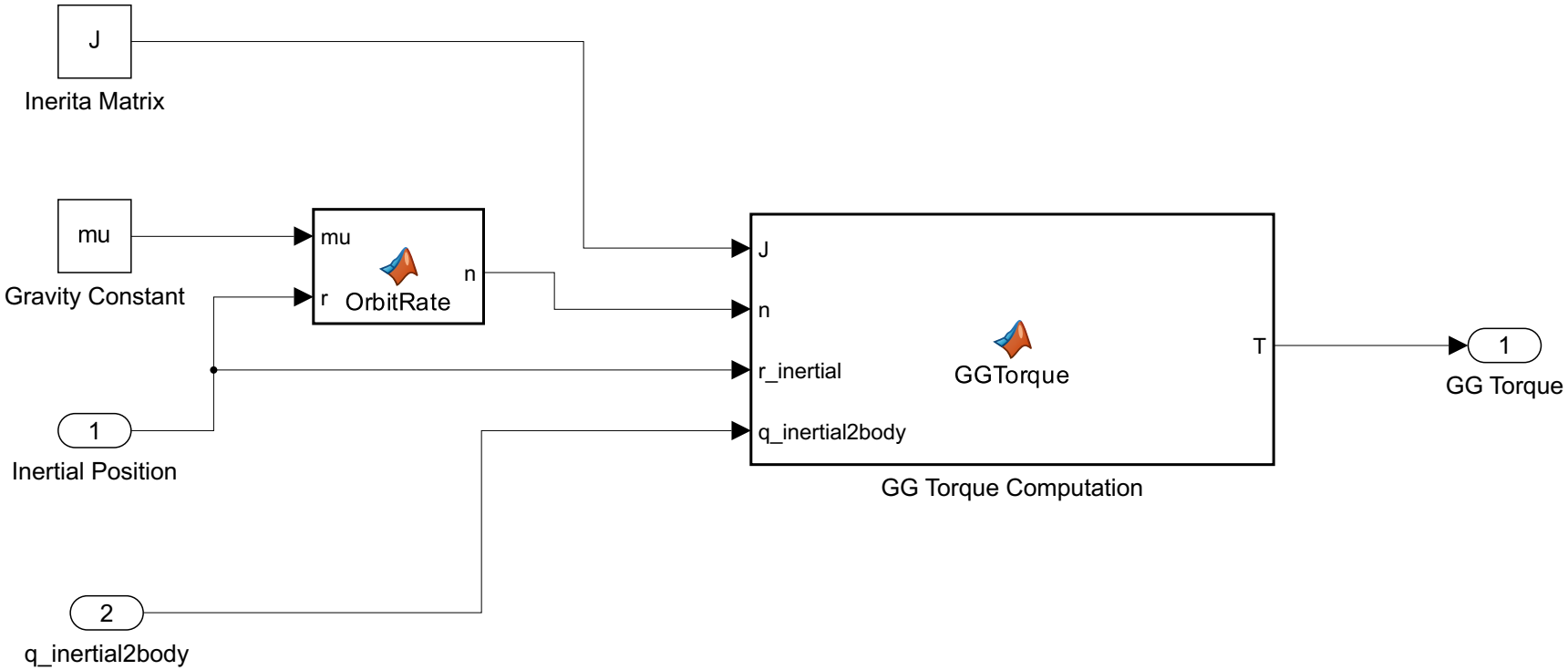
```
function theta_real = ForceReal(theta)

theta_real = real(theta);
```

```
function q_ref2actual = ErrorQuat(q_inertial2body_ref, q_inertial2body_actual)

q_ref2actual = QuatProduct(q_inertial2body_actual,QuatInv(q_inertial2body_ref));

% Force normalization
q_ref2actual = q_ref2actual/norm(q_ref2actual);
```

```
function delta_w = ErrorRate(w_ref, w_actual, q_ref2actual)

delta_w = w_actual - QuatTransform(q_ref2actual,w_ref);
```

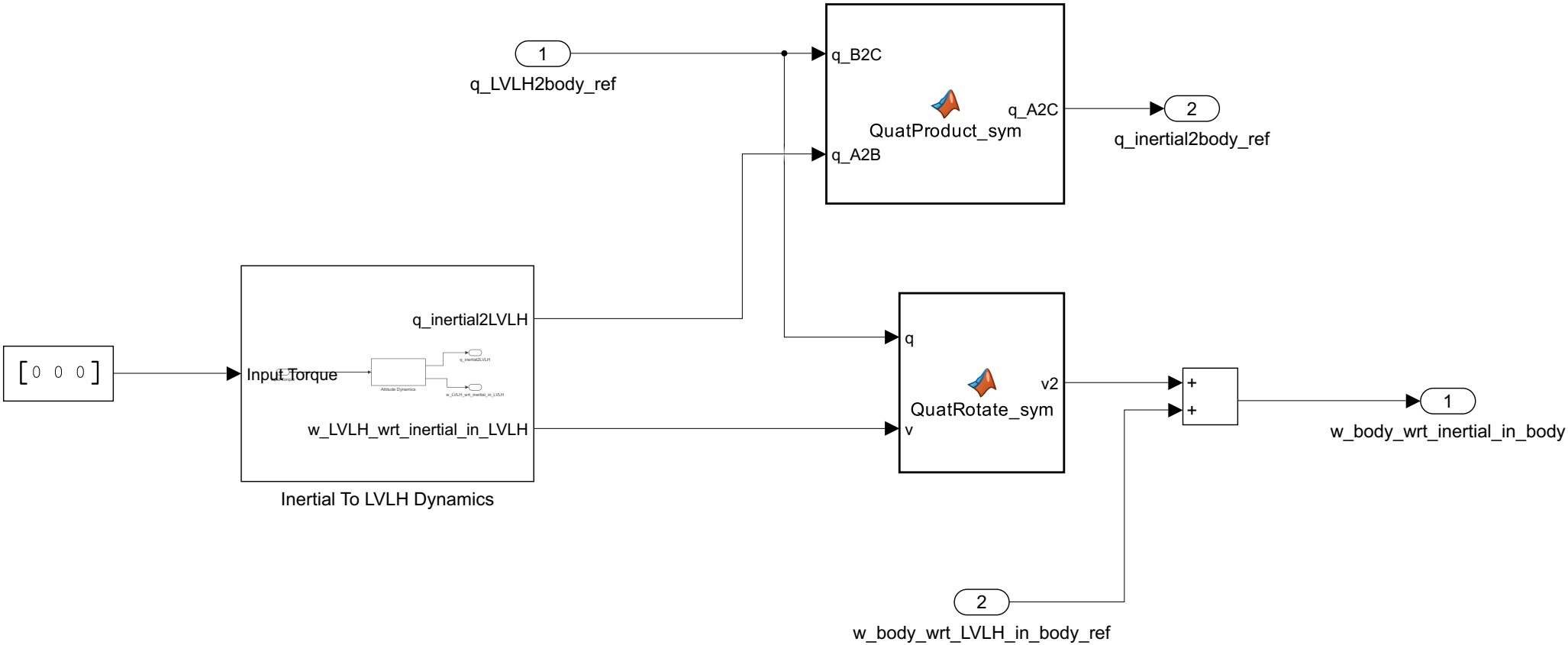
```
function dtheta = ErrorQuat2ErrorAngle(q_err)

% Quaternion components
q_v = real(q_err(1:3));
q_s = real(q_err(4));

dtheta = [atan2(q_v(1),q_s);
    atan2(q_v(2),q_s);
    atan2(q_v(3),q_s)];
```

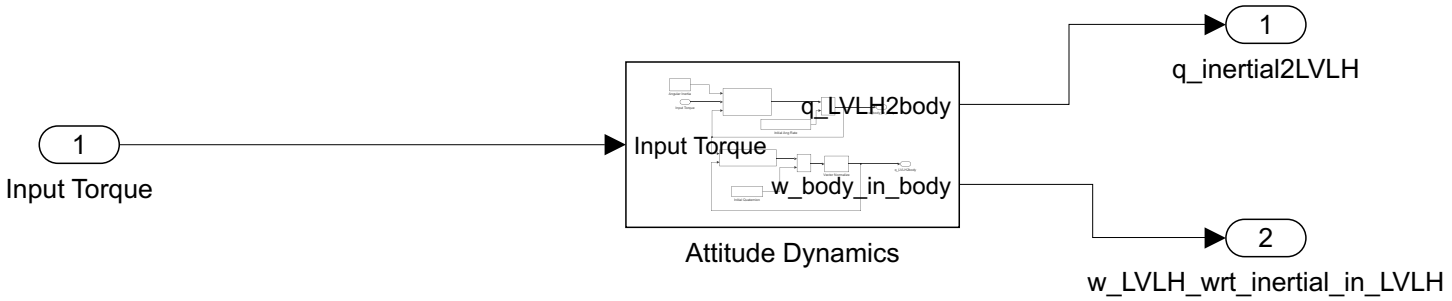
```
function dthetadot = ErrorThetaDot(omega_ref, delta_omega, delta_theta)

dthetadot = delta_omega - cross(omega_ref,delta_theta);
```

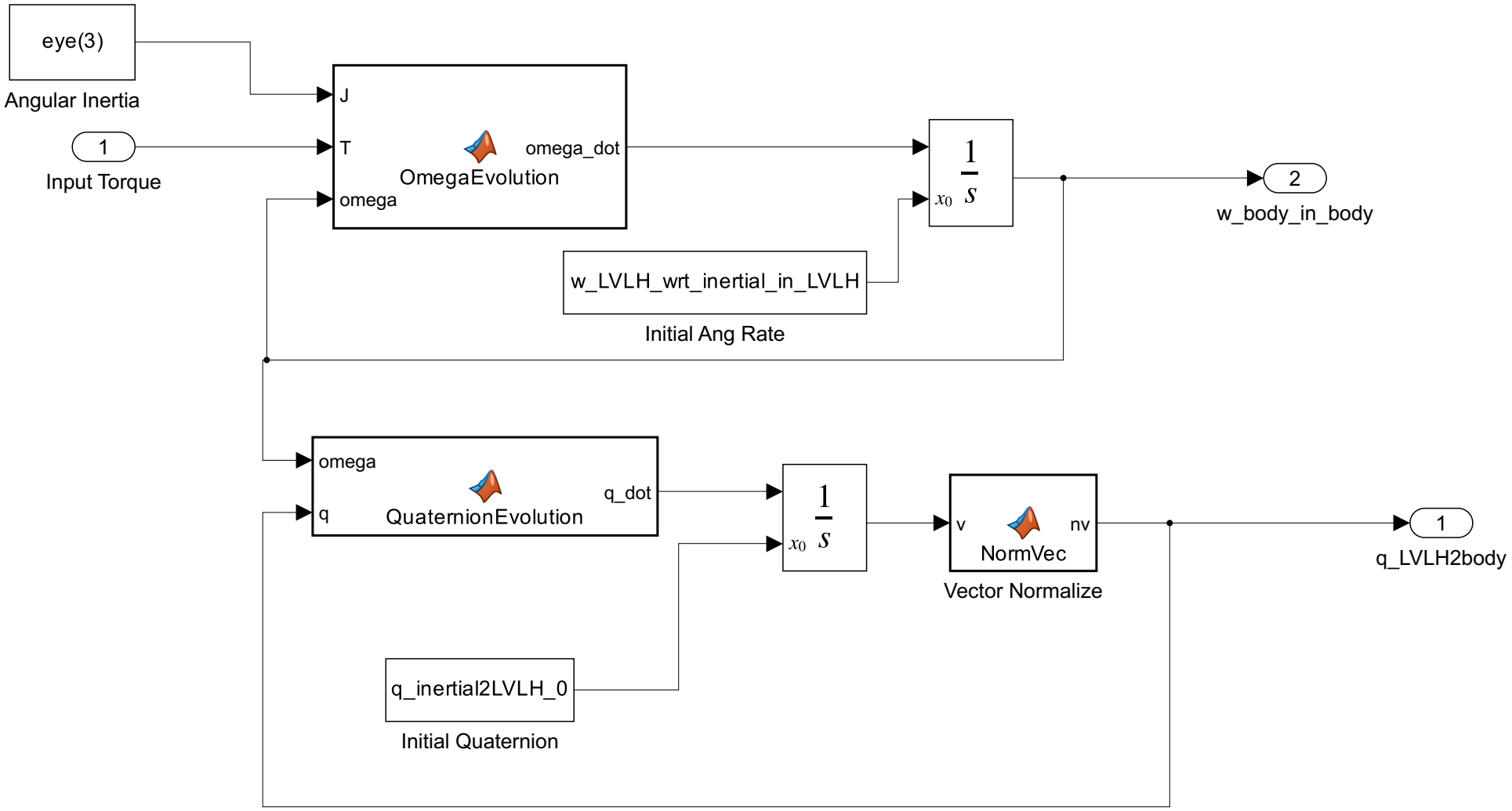

J
Inerita Matrix
mu
Gravity Constant
mu
r
OrbitRate
n
1
Inertial Position
2
q_inertial2body
J
n
r_inertial
q_inertial2body
GGTorque
T
GG Torque Computation
1
GG Torque

```
function T = GGTorque(J, n, r_inertial, q_inertial2body)

% Down in body frame
down_inertial = -r_inertial/norm(r_inertial);
down_body = QuatTransform(q_inertial2body,down_inertial);

% Gravity gradient torque
T = 3*n^2*cross(down_body,J*down_body);
```

```
function n = OrbitRate(mu,r)

n = sqrt(mu/norm(r)^3);
```


1
q_LVLH2body_ref
q_B2C
q_A2B
QuatProduct_sym
q_A2C
2
q_inertial2body_ref
[0 0 0]
Input Torque
q_inertial2LVLH
w_LVLH_wrt_inertial_in_LVLH
Inertial To LVLH Dynamics
q
v
QuatRotate_sym
v2
+
+
1
w_body_wrt_inertial_in_body
2
w_body_wrt_LVLH_in_body_ref


1
Input Torque
Input Torque
q_LVLH2body
w_body_in_body
Attitude Dynamics
1
q_inertial2LVLH
2
w_LVLH_wrt_inertial_in_LVLH


eye(3)
Angular Inertia
1
Input Torque
J
T
omega
omega_dot
OmegaEvolution
1
s
x0
2
w_body_in_body
w_LVLH_wrt_inertial_in_LVLH
Initial Ang Rate
omega
q
q_dot
QuaternionEvolution
1
s
x0
v
nv
NormVec
Vector Normalize
1
q_LVLH2body
q_inertial2LVLH_0
Initial Quaternion

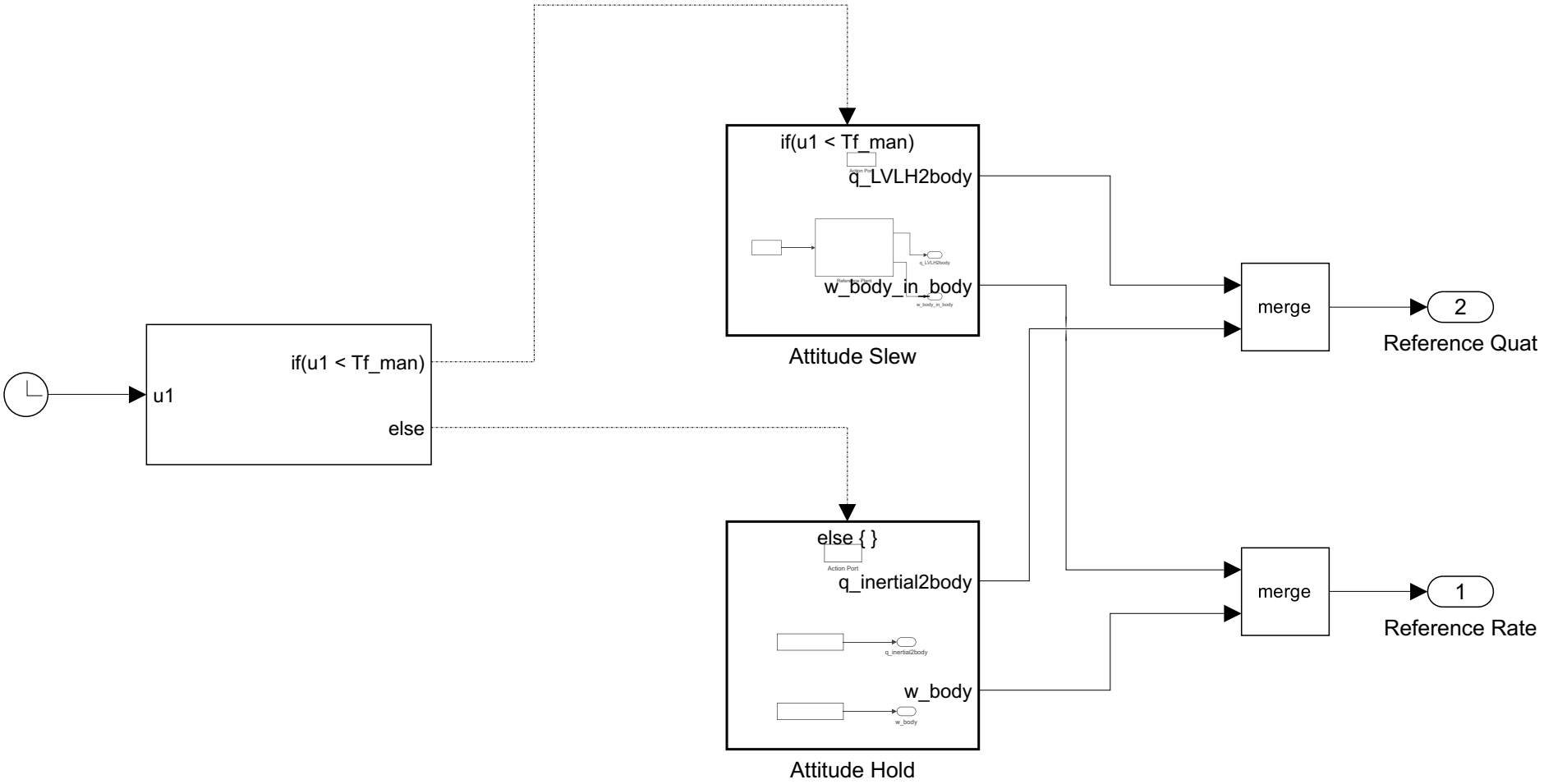
```
function q_dot = QuaternionEvolution(omega, q)
intermed = [omega; 0];
q_dot = 0.5*QuatProduct(intermed,q);
```

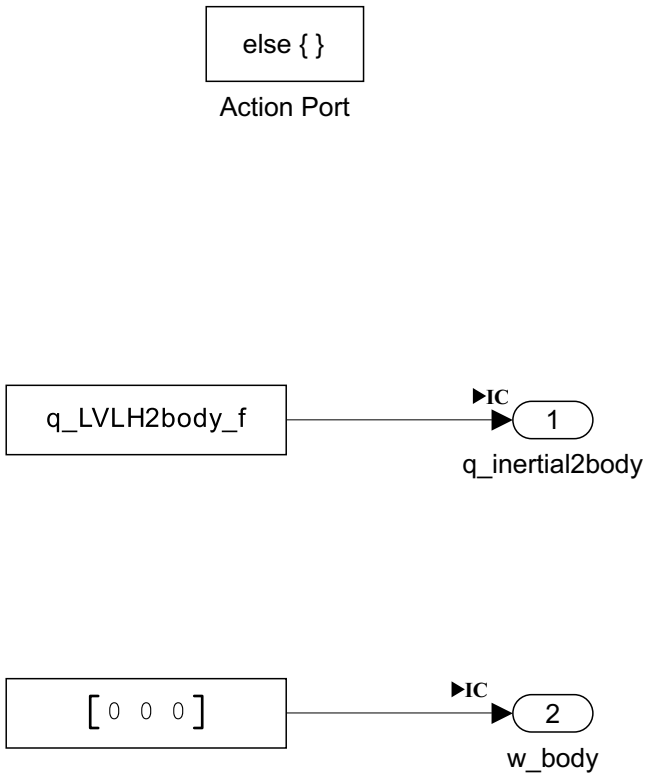
```
function omega_dot = OmegaEvolution(J, T, omega)
omega_dot = J\(T - cross(omega, J*omega));
```

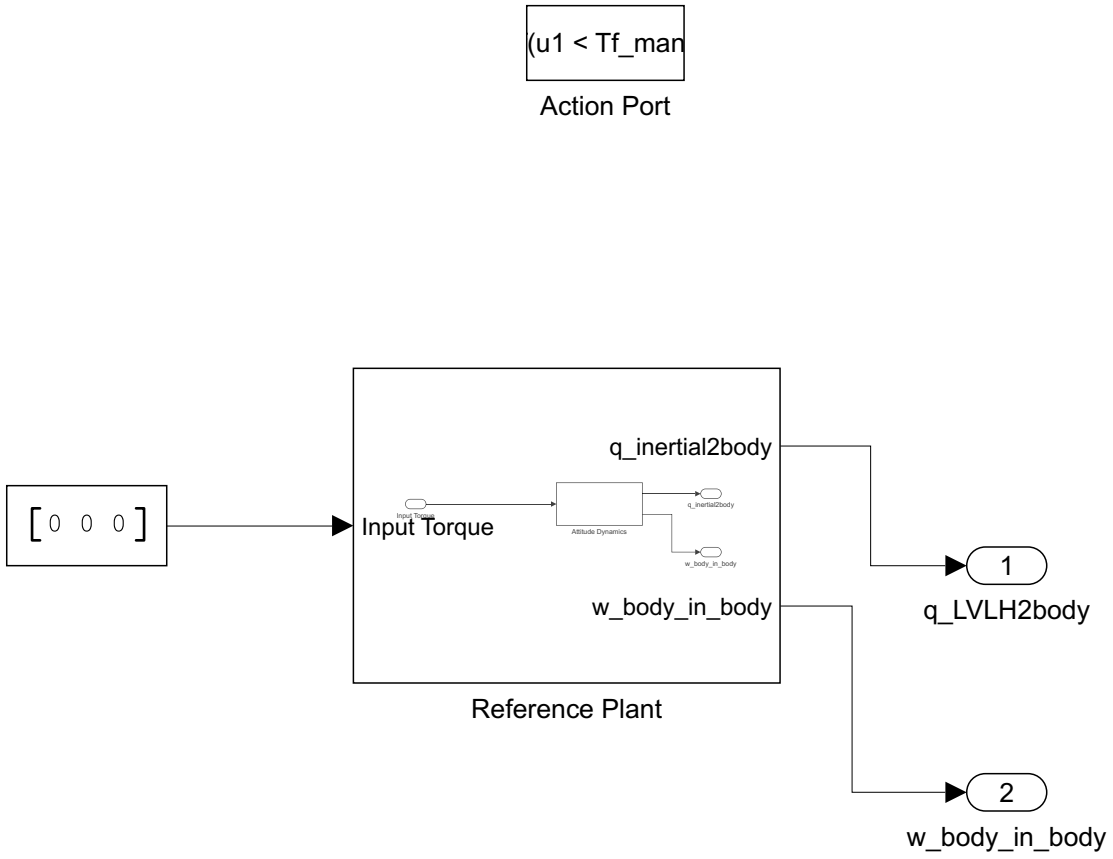
```
function nv = NormVec(v)

nv = v/norm(v);
```

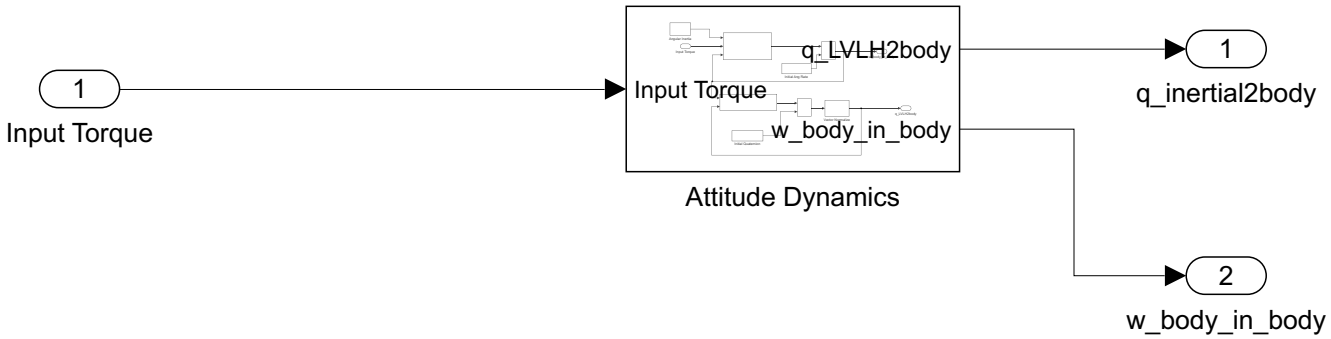
```
function q_A2C = QuatProduct_sym(q_B2C,q_A2B)

q_A2C = QuatProduct(q_B2C,q_A2B);
```

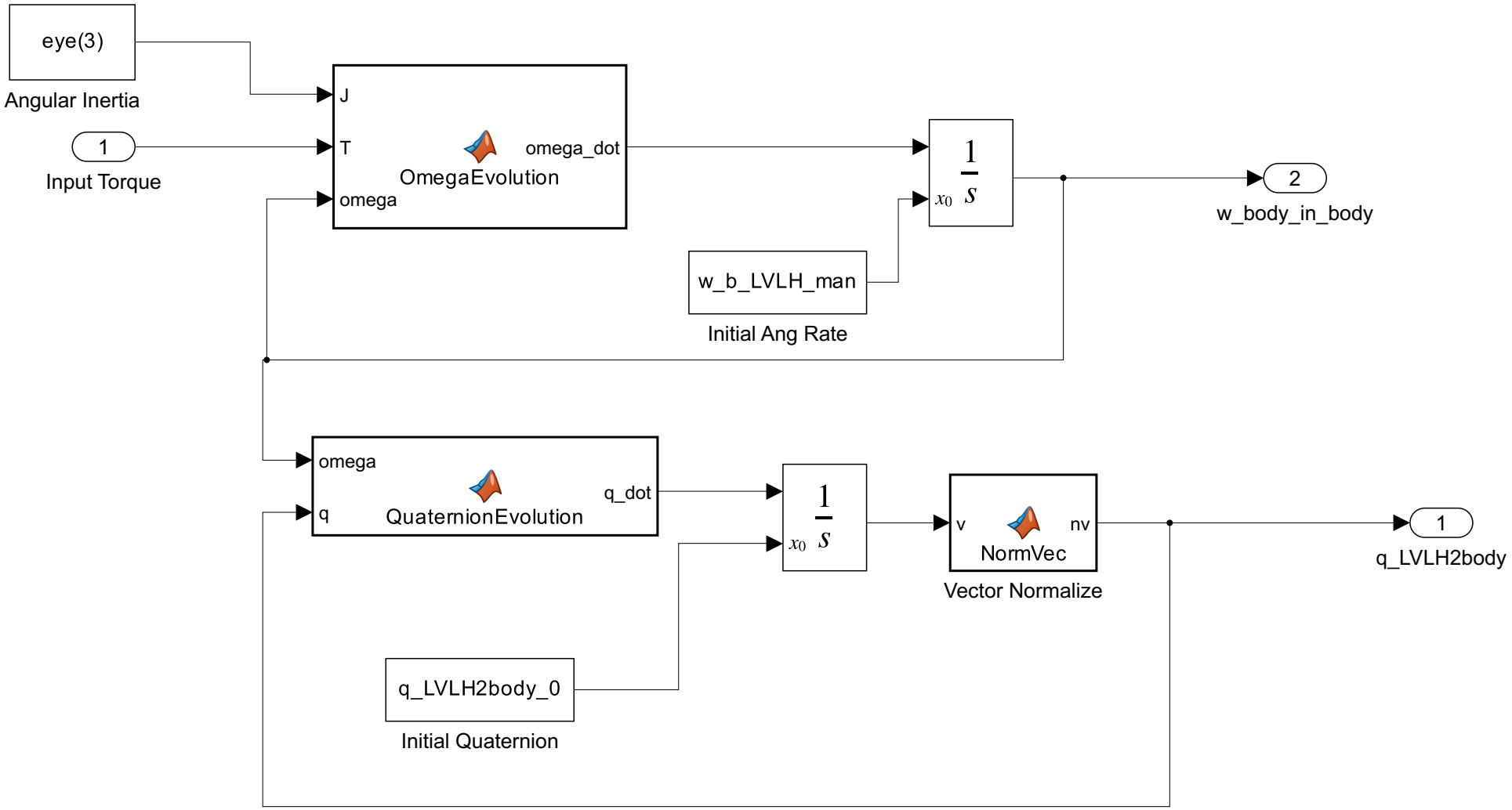
```
function v2 = QuatRotate_sym(q,v)

v2 = QuatTransform(q,v);
```


if(u1 < Tf_man)
u1
else
if(u1 < Tf_man)
q_LVLH2body
w_body_in_body
Attitude Slew
else { }
Action Port
q_inertial2body
w_body
Attitude Hold
merge
merge
2
Reference Quat
1
Reference Rate


else { }
Action Port
q_LVLH2body_f
IC
1
q_inertial2body
[0 0 0]
IC
2
w_body


(u1 < Tf_man
Action Port
[0 0 0]
Input Torque
q_inertial2body
w_body_in_body
Reference Plant
1
q_LVLH2body
2
w_body_in_body

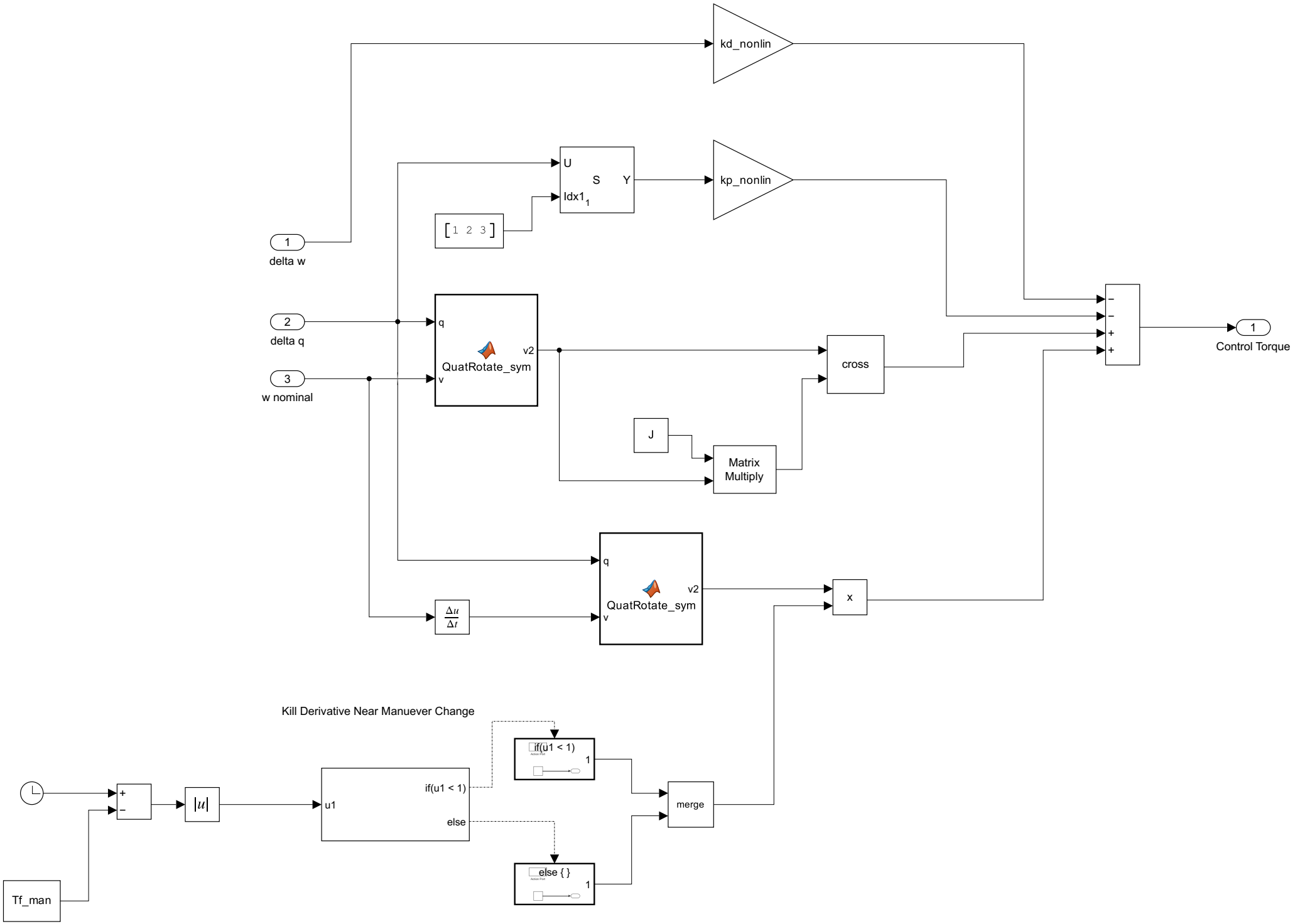

1
Input Torque
Input Torque
q_LVLH2body
w_body_in_body
Attitude Dynamics
1
q_inertial2body
2
w_body_in_body

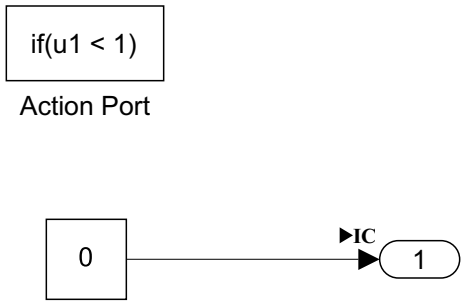

eye(3)
Angular Inertia
1
Input Torque
J
T
omega
omega_dot
OmegaEvolution
1
s
x0
2
w_body_in_body
w_b_LVLH_man
Initial Ang Rate
omega
q
q_dot
QuaternionEvolution
1
s
x0
v
nv
NormVec
Vector Normalize
1
q_LVLH2body
q_LVLH2body_0
Initial Quaternion

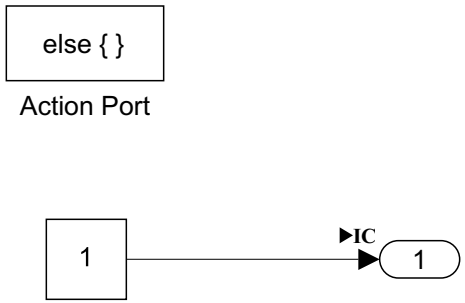
```
function q_dot = QuaternionEvolution(omega, q)
intermed = [omega; 0];
q_dot = 0.5*QuatProduct(intermed,q);
```

```
function omega_dot = OmegaEvolution(J, T, omega)
omega_dot = J\(T - cross(omega, J*omega));
```

```
function nv = NormVec(v)

nv = v/norm(v);
```

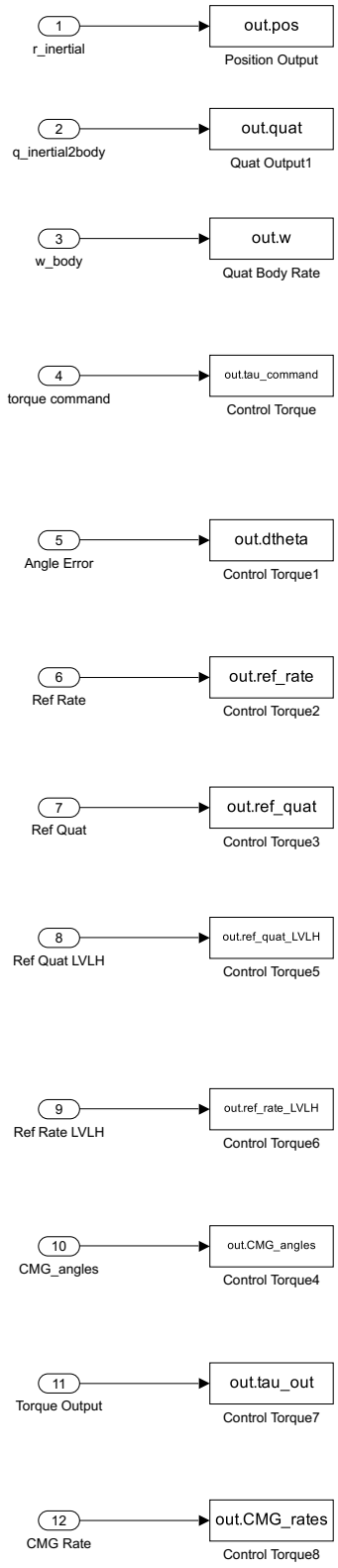

kd_nonlin
U
S
Y
Idx1
1
kp_nonlin
[1 2 3]
1
delta w
2
delta q
3
w nominal
q
v
QuatRotate_sym
v2
cross
J
Matrix Multiply
1
Control Torque
Δu/Δt
x
Kill Derivative Near Manuever Change
if(u1 < 1)
else
u1
|u|
merge
else { }
Tf_man


if(u1 < 1)
Action Port
0
IC
1

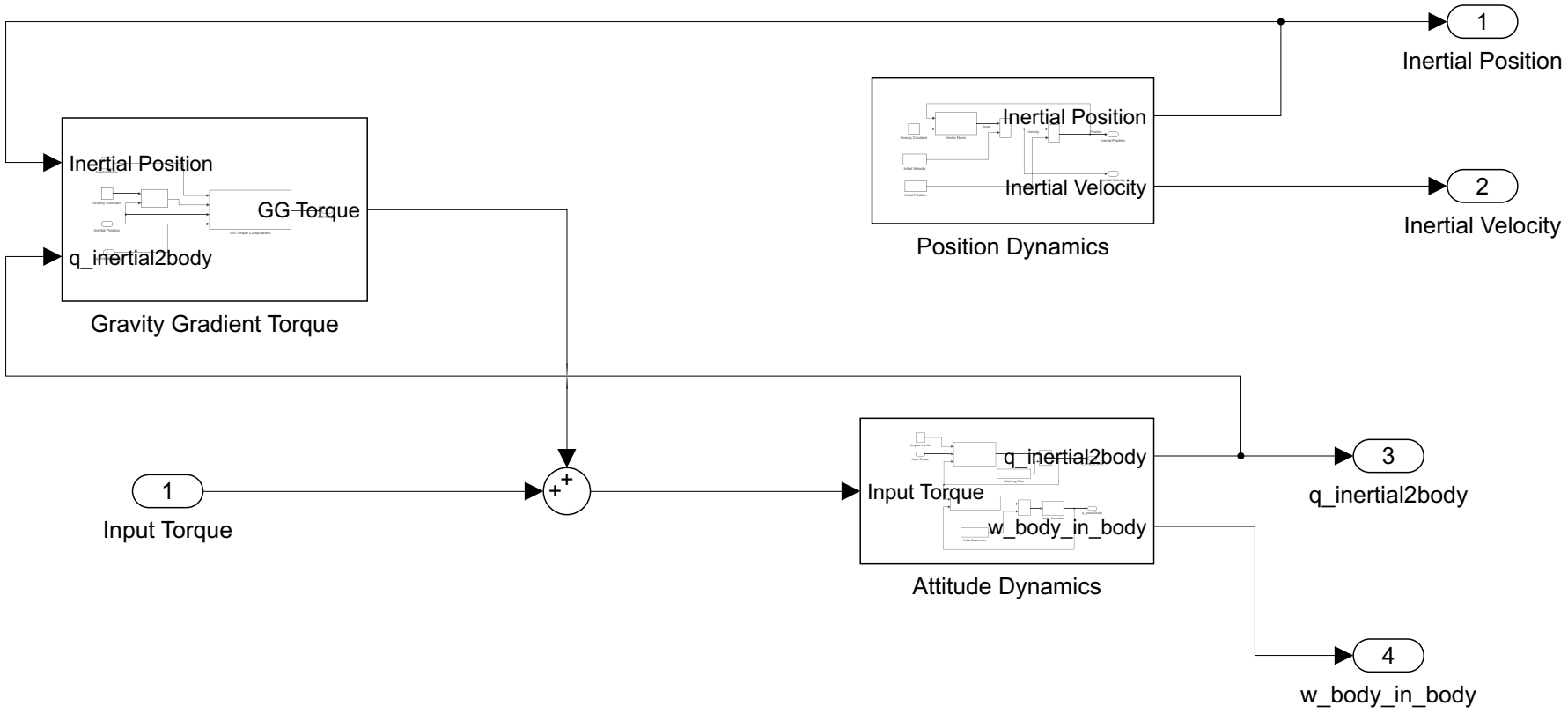

else { }
Action Port
1
IC
1

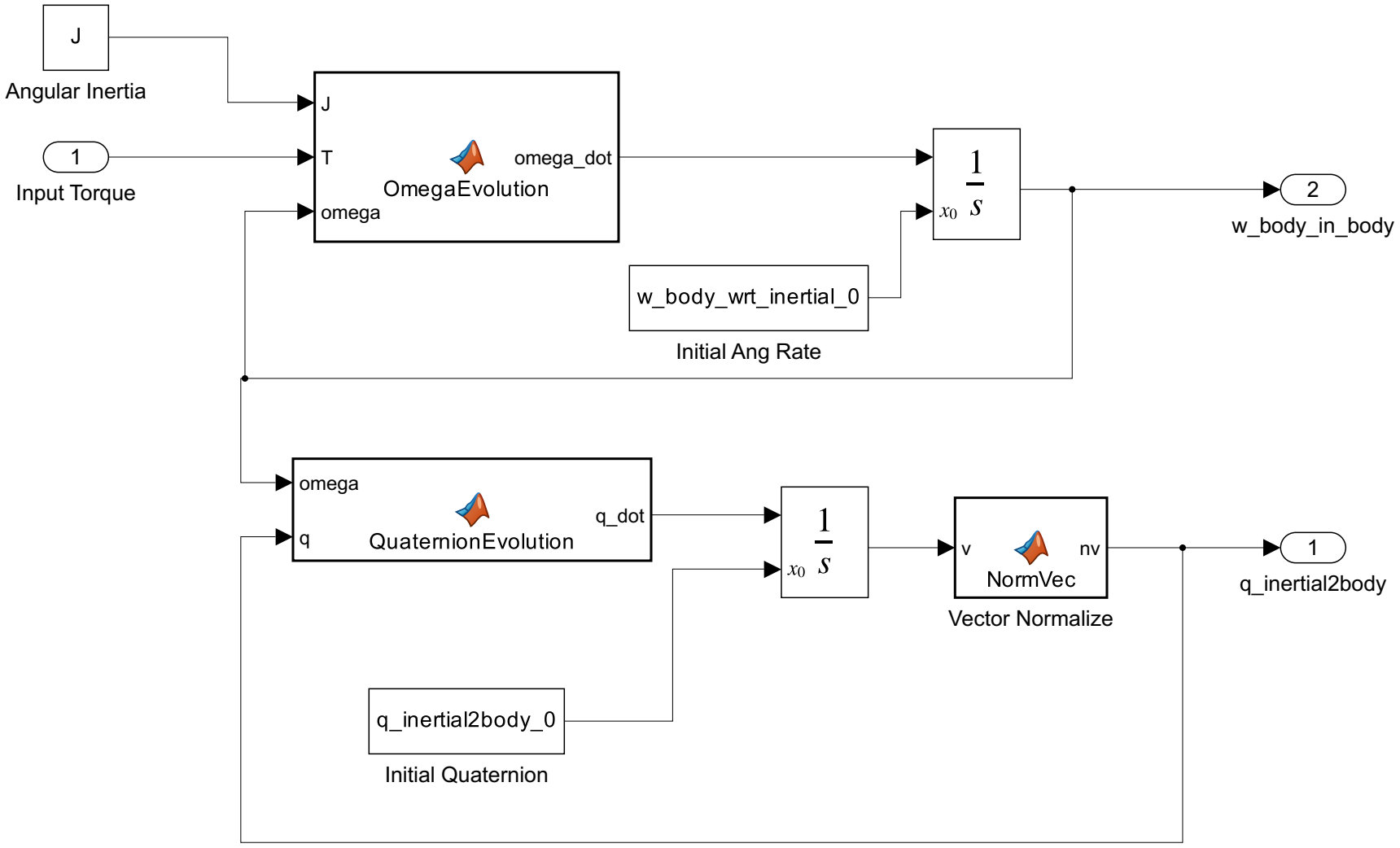
```
function v2 = QuatRotate_sym(q,v)

v2 = QuatTransform(q,v);
```

```
function v2 = QuatRotate_sym(q,v)

v2 = QuatTransform(q,v);
```


1
r_inertial
out.pos
Position Output
2
q_inertial2body
out.quat
Quat Output1
3
w_body
out.w
Quat Body Rate
4
torque command
out.tau_command
Control Torque
5
Angle Error
out.dtheta
Control Torque1
6
Ref Rate
out.ref_rate
Control Torque2
7
Ref Quat
out.ref_quat
Control Torque3
8
Ref Quat LVLH
out.ref_quat_LVLH
Control Torque5
9
Ref Rate LVLH
out.ref_rate_LVLH
Control Torque6
10
CMG_angles
out.CMG_angles
Control Torque4
11
Torque Output
out.tau_out
Control Torque7
12
CMG Rate
out.CMG_rates
Control Torque8

Plant

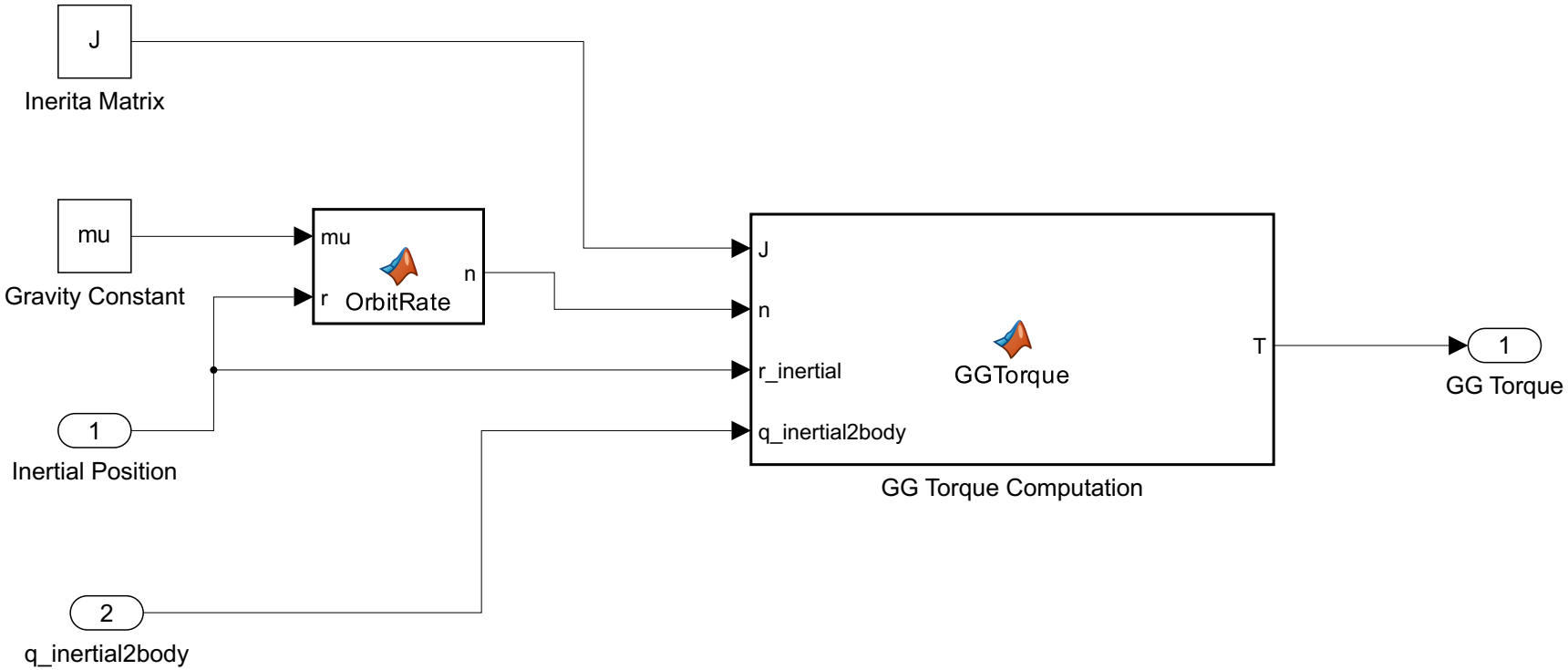

1
Inertial Position
Inertial Position
Inertial Velocity
Position Dynamics
2
Inertial Velocity
Inertial Position
GG Torque
q_inertial2body
Gravity Gradient Torque
1
Input Torque
+
+
q_inertial2body
Input Torque
w_body_in_body
Attitude Dynamics
3
q_inertial2body
4
w_body_in_body


J
Angular Inertia
1
Input Torque
J
T
omega
omega_dot
OmegaEvolution
1
s
x0
w_body_wrt_inertial_0
Initial Ang Rate
2
w_body_in_body
omega
q
q_dot
QuaternionEvolution
1
s
x0
v
nv
NormVec
Vector Normalize
1
q_inertial2body
q_inertial2body_0
Initial Quaternion

```
function q_dot = QuaternionEvolution(omega, q)
intermed = [omega; 0];
q_dot = 0.5*QuatProduct(intermed,q);
```

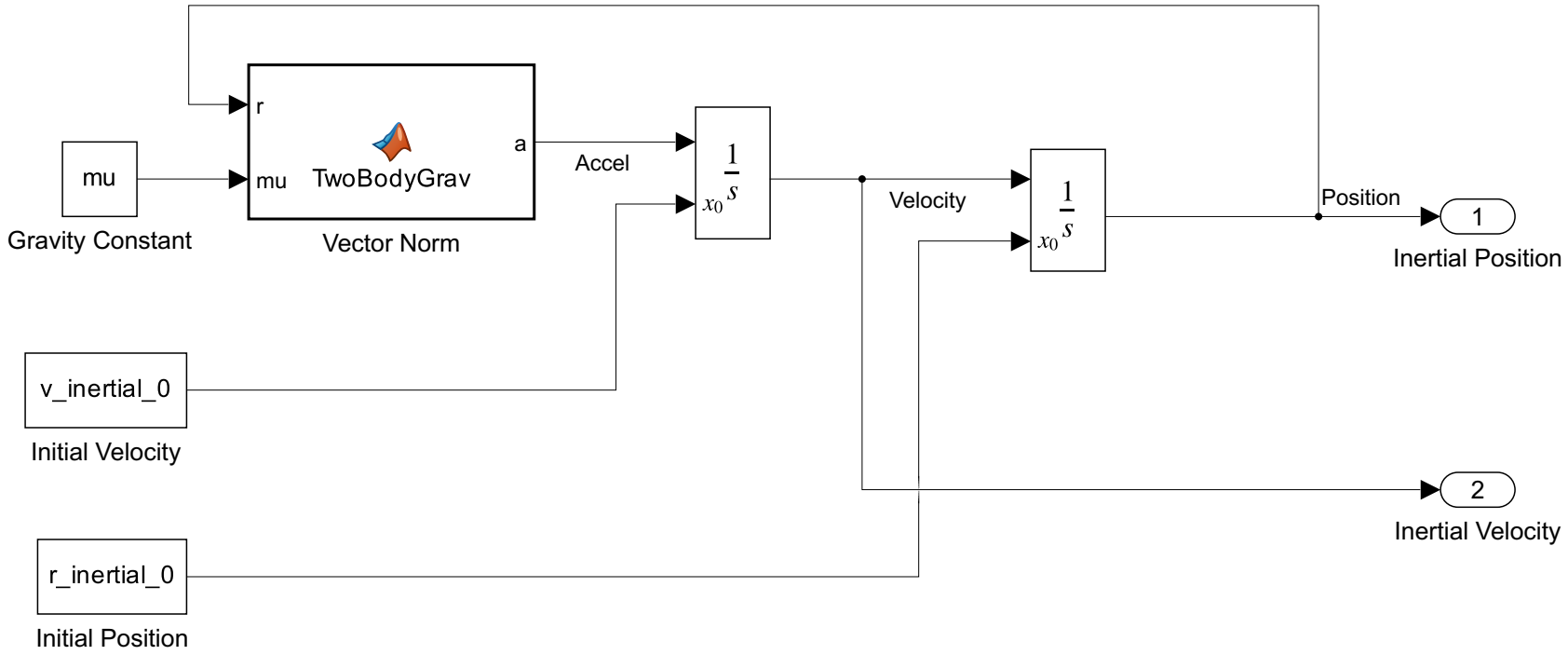
```
function omega_dot = OmegaEvolution(J, T, omega)
omega_dot = J\(T - cross(omega, J*omega));
```

```
function nv = NormVec(v)

nv = v/norm(v);
```


J
Inerita Matrix
mu
Gravity Constant
mu
r
OrbitRate
n
1
Inertial Position
2
q_inertial2body
J
n
r_inertial
q_inertial2body
GGTorque
T
GG Torque Computation
1
GG Torque

```
function T = GGTorque(J, n, r_inertial, q_inertial2body)

% Down in body frame
down_inertial = -r_inertial/norm(r_inertial);
down_body = QuatTransform(q_inertial2body,down_inertial);

% Gravity gradient torque
T = 3*n^2*cross(down_body,J*down_body);
```

```
function n = OrbitRate(mu,r)

n = sqrt(mu/norm(r)^3);
```


r
mu
TwoBodyGrav
a
Vector Norm
mu
Gravity Constant
Accel
1
s
x0
Velocity
1
s
x0
Position
1
Inertial Position
v_inertial_0
Initial Velocity
2
Inertial Velocity
r_inertial_0
Initial Position

```
function a = TwoBodyGrav(r,mu)

nr = norm(r);
a = -mu*r/(nr^3);
```