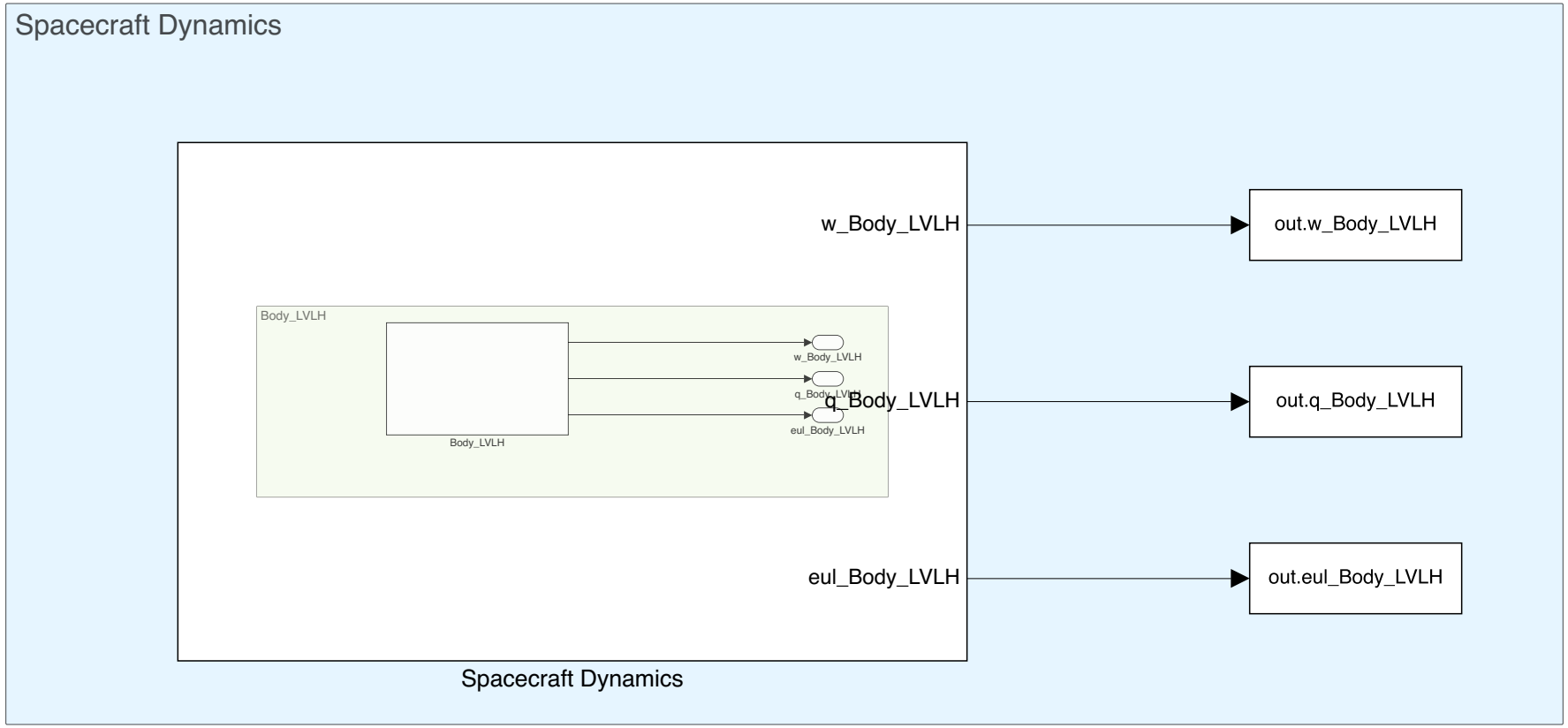
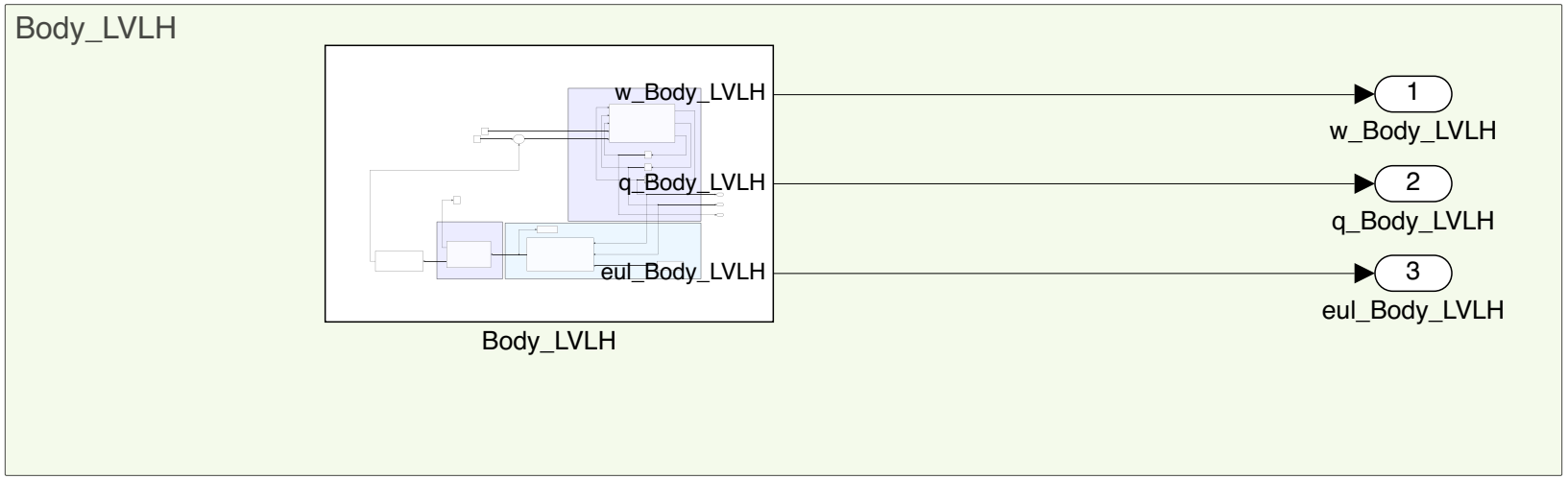
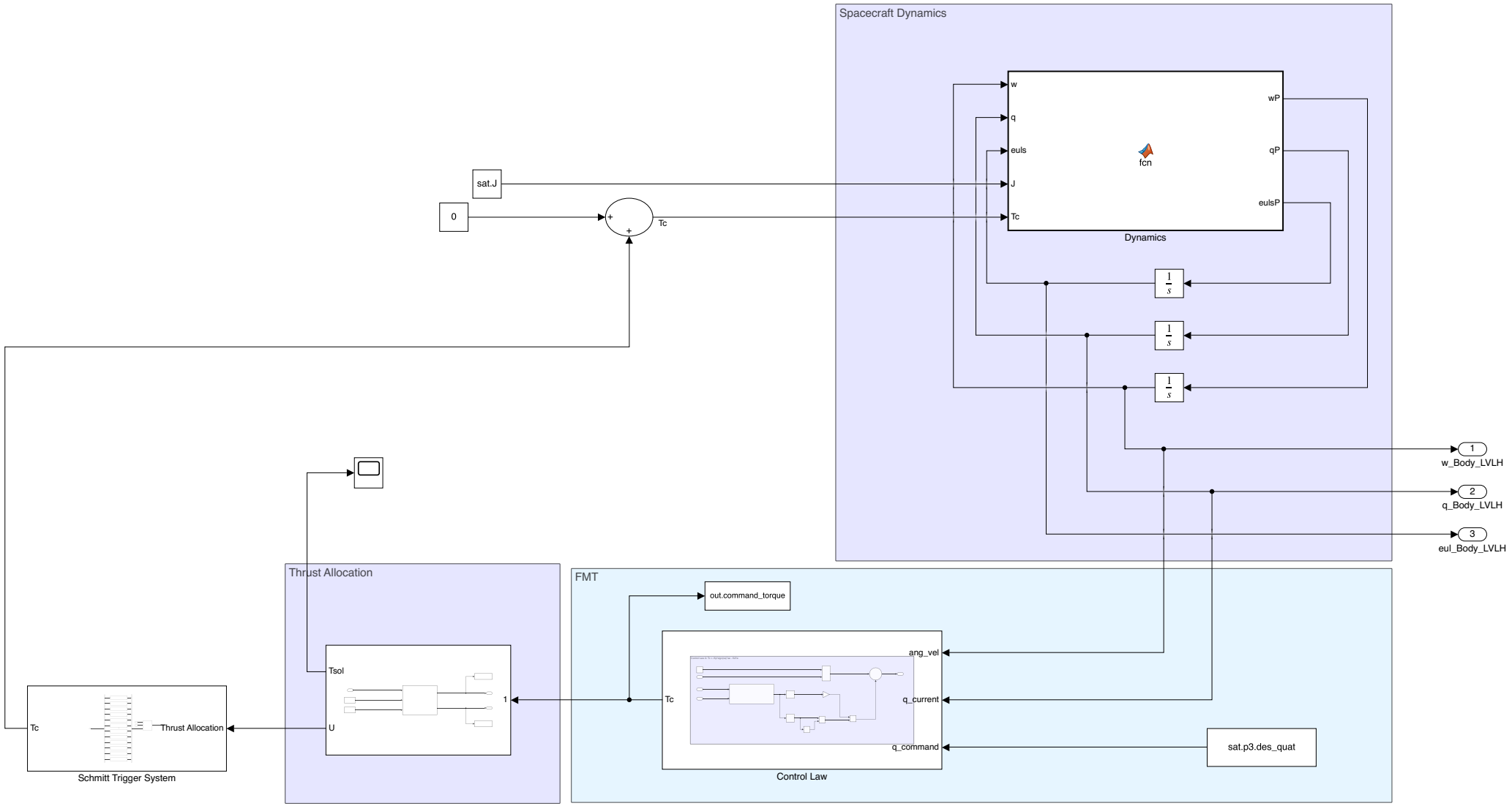
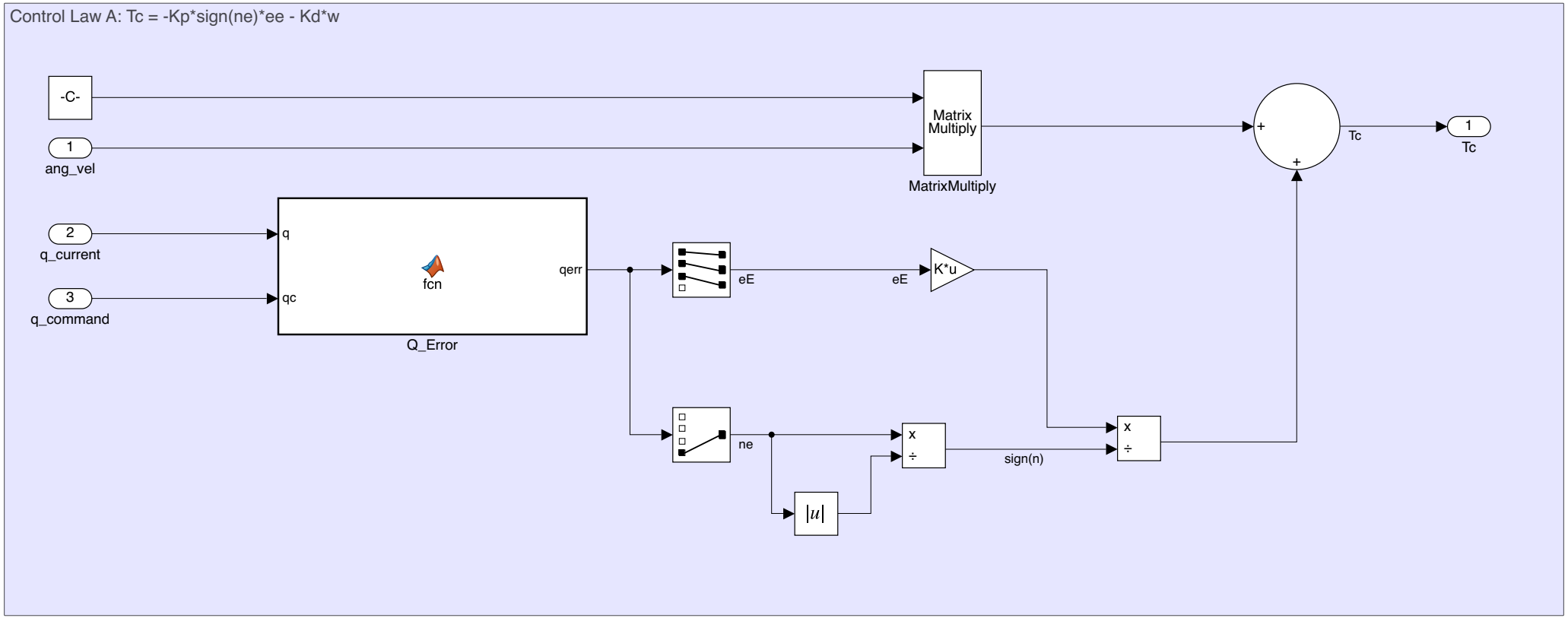

Spacecraft Dynamics
w_Body_LVLH
q_Body_LVLH
eul_Body_LVLH
out.w_Body_LVLH
out.q_Body_LVLH
out.eul_Body_LVLH
Body_LVLH
w_Body_LVLH
q_Body_LVLH
eul_Body_LVLH
Body_LVLH
Spacecraft Dynamics

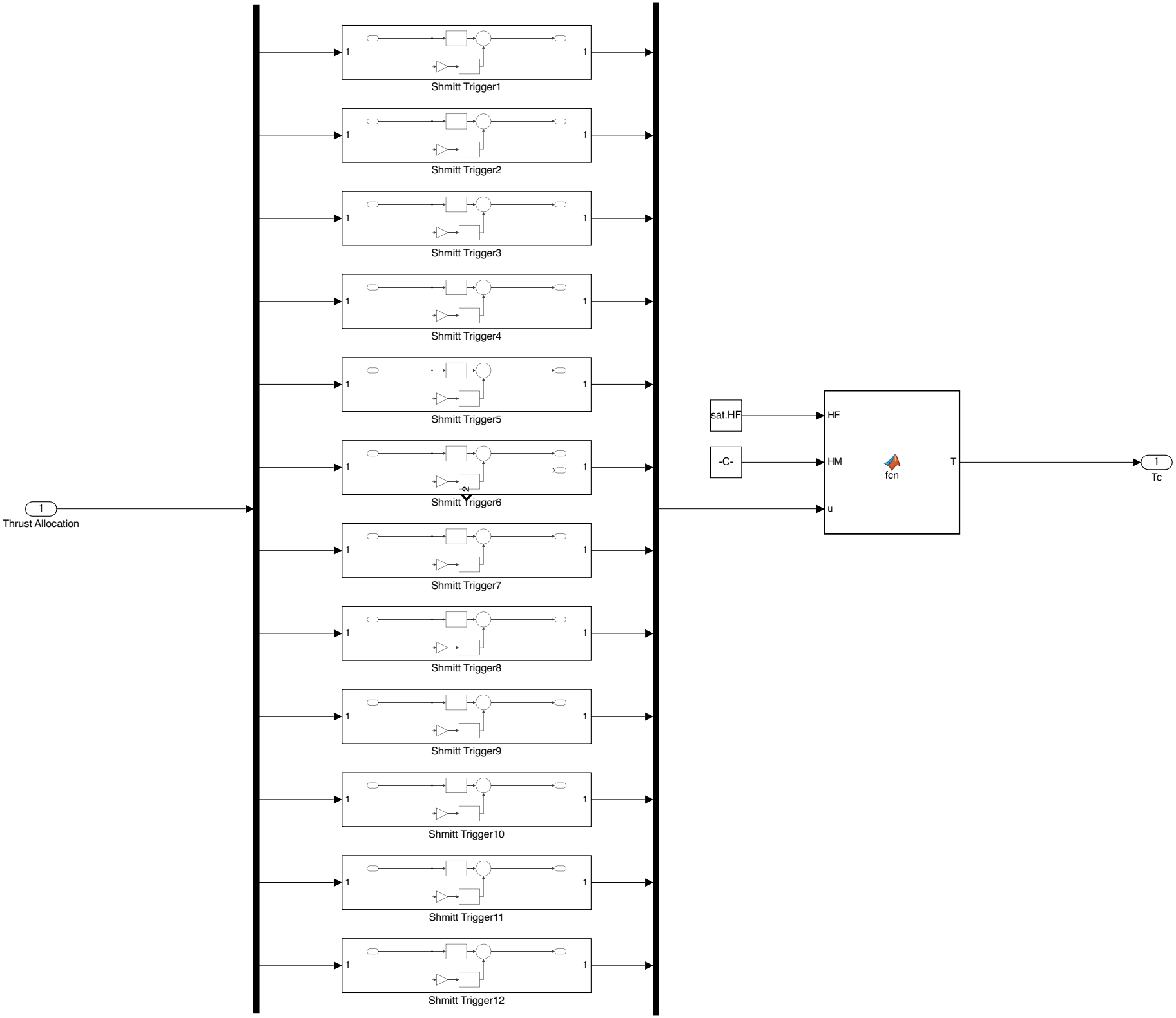

Body_LVLH
w_Body_LVLH
q_Body_LVLH
eul_Body_LVLH
Body_LVLH
1
w_Body_LVLH
2
q_Body_LVLH
3
eul_Body_LVLH


Spacecraft Dynamics
w
q
euls
J
Tc
wP
qP
eulsP
fcn
Dynamics
sat.J
0
Tc
1
s
1
s
1
s
1
w_Body_LVLH
2
q_Body_LVLH
3
eul_Body_LVLH
Thrust Allocation
Tsol
U
1
FMT
out.command_torque
Tc
ang_vel
q_current
q_command
Control Law
sat.p3.des_quat
Tc
Thrust Allocation
Schmitt Trigger System

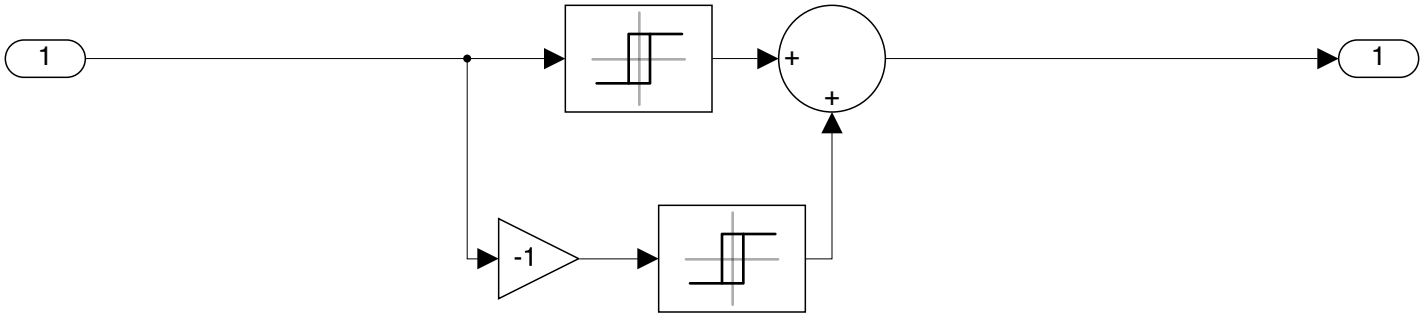

Control Law A: Tc = -Kp*sign(ne)*ee - Kd*w
-C-
1
ang_vel
Matrix Multiply
MatrixMultiply
+
+
Tc
1
Tc
2
q_current
3
q_command
q
qc
fcn
Q_Error
qerr
eE
eE
K*u
ne
x
÷
sign(n)
x
÷
|u|

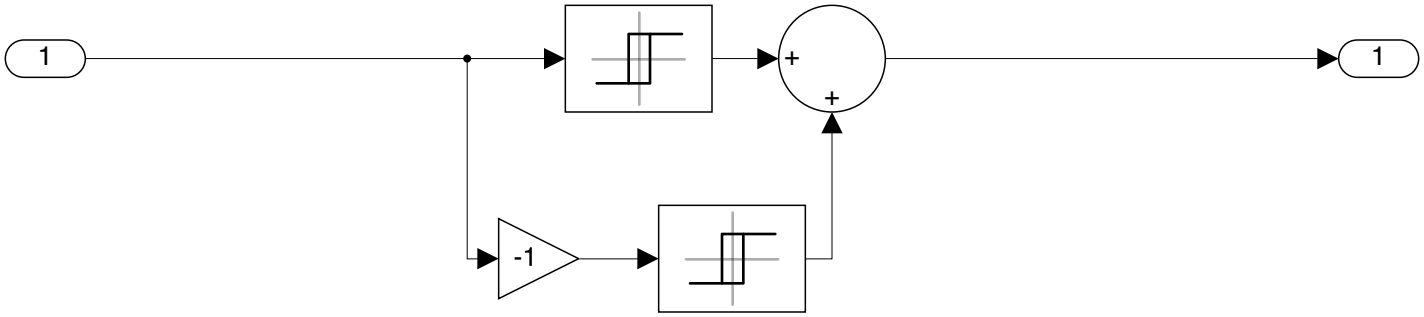
```
function qerr = fcn(q, qc)

%     function wx = skewSymmetric(w)
%         wx = [0, -1*w(3), w(2);
%               w(3), 0, -1*w(1);
%               -1*w(2), w(1), 0];
%     end

    function qp = quatmult(q, p)

        function wx = skewSymmetric(w)
            wx = [0, -1*w(3), w(2);
                  w(3), 0, -1*w(1);
                  -1*w(2), w(1), 0];
        end

        qn = q(4);
        qe = q(1:3);

        pn = p(4);
        pe = p(1:3);

        n = pn * qn - pe'*qe;
        e = pn * qe + qn*pe + skewSymmetric(pe)*qe;

        qp = [e(1);e(2);e(3);n];

    end

qc(1:3) = -1*qc(1:3);
qerr = quatmult(qc, q);

end
```

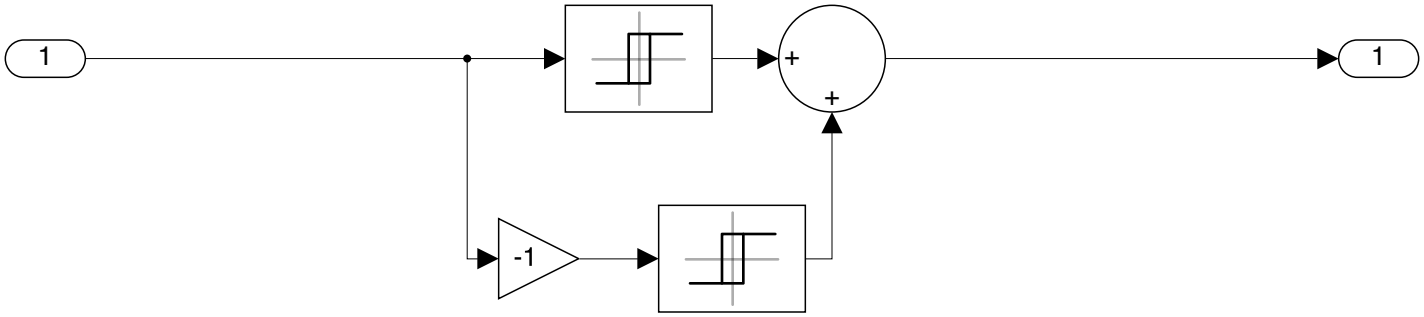
```
function [wP, qP, eulsP] = fcn(w, q, euls, J, Tc)

    function wx = skewSymmetric(w)
        wx = [0, -1*w(3), w(2);
              w(3), 0, -1*w(1);
              -1*w(2), w(1), 0];
    end

    T = [Tc(1);Tc(2);Tc(3)];

    wP = J\(T - skewSymmetric(w)*J*w);

    e = q(1:3);
    n = q(4);
    eP = 0.5*(n*eye(3) + skewSymmetric(e))*w;
    nP = -0.5*e'*w;
    qP = [eP;nP];

    phi = euls(1);
    theta = euls(2);
    psi = euls(3);
    eulsP = 1/(cos(theta)) * [cos(theta), sin(phi)*sin(theta), cos(phi)*sin(theta);
                              0, cos(phi)*cos(theta), -1*sin(phi)*cos(theta);
                              0, sin(phi), cos(phi)] * w;


end
```

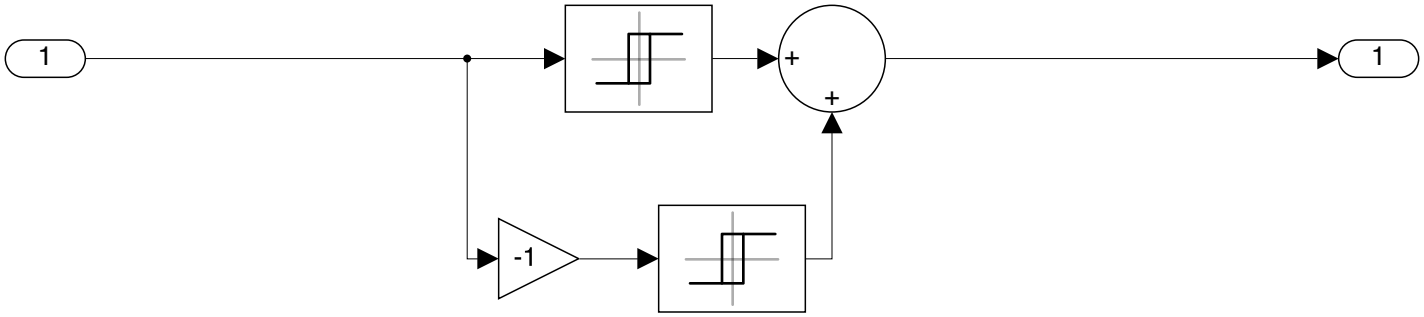

1
Thrust Allocation
Shmitt Trigger1
Shmitt Trigger2
Shmitt Trigger3
Shmitt Trigger4
Shmitt Trigger5
Shmitt Trigger6
Shmitt Trigger7
Shmitt Trigger8
Shmitt Trigger9
Shmitt Trigger10
Shmitt Trigger11
Shmitt Trigger12
sat.HF
-C-
HF
HM
u
fcn
T
1
Tc

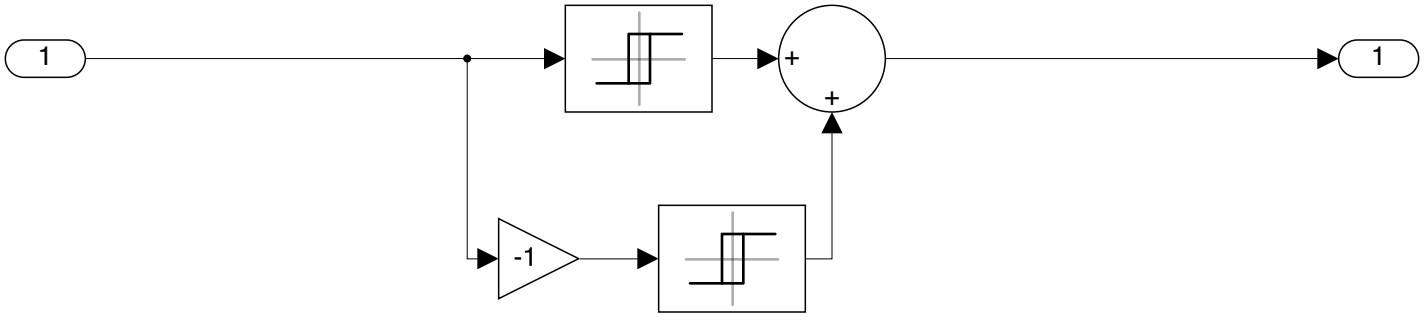
```
function T = fcn(HF, HM, u)

H = [HF;HM];
T = HM*u;
```

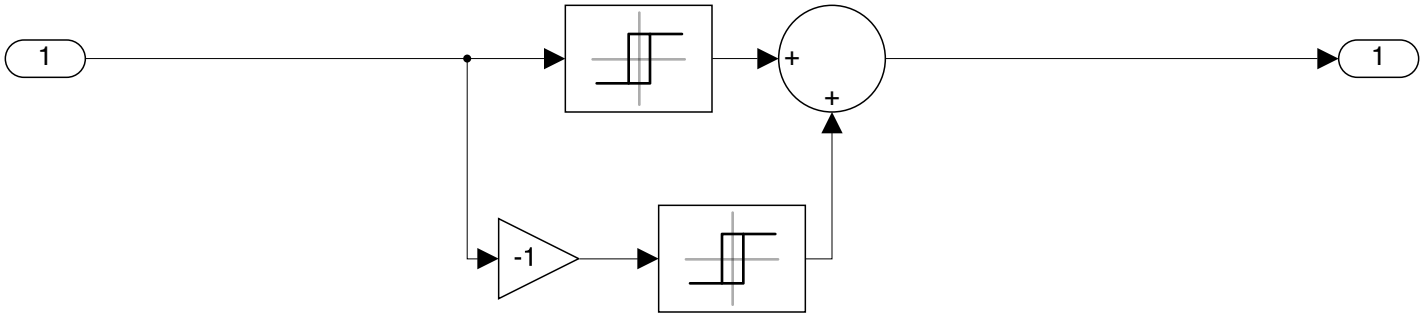

1
+
+
1
-1

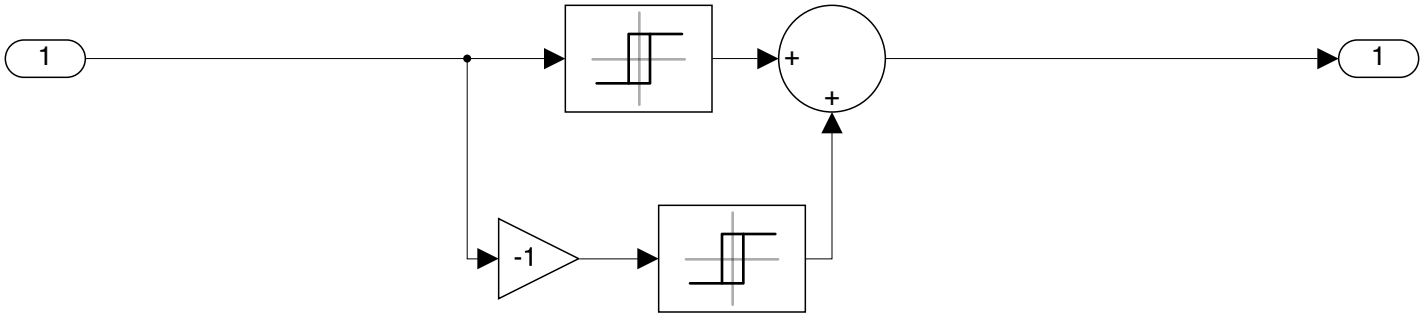

1
+
+
1
-1

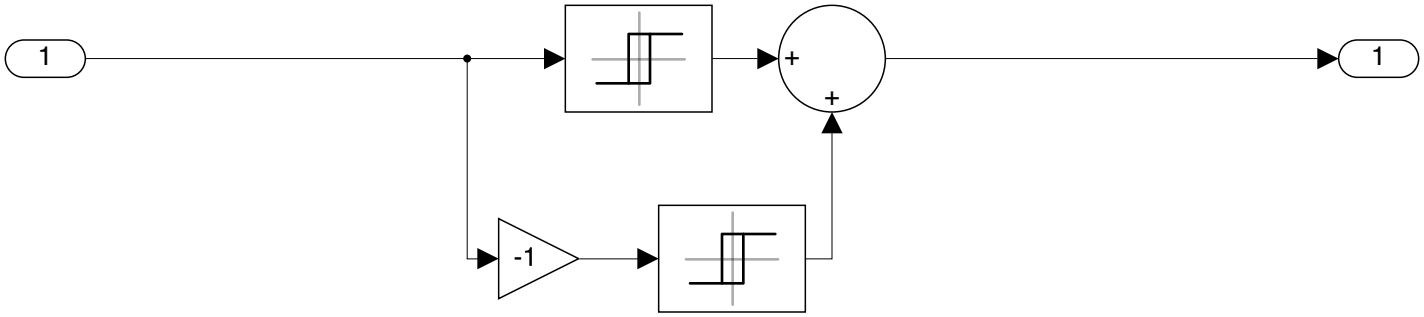

1
+
+
1
-1

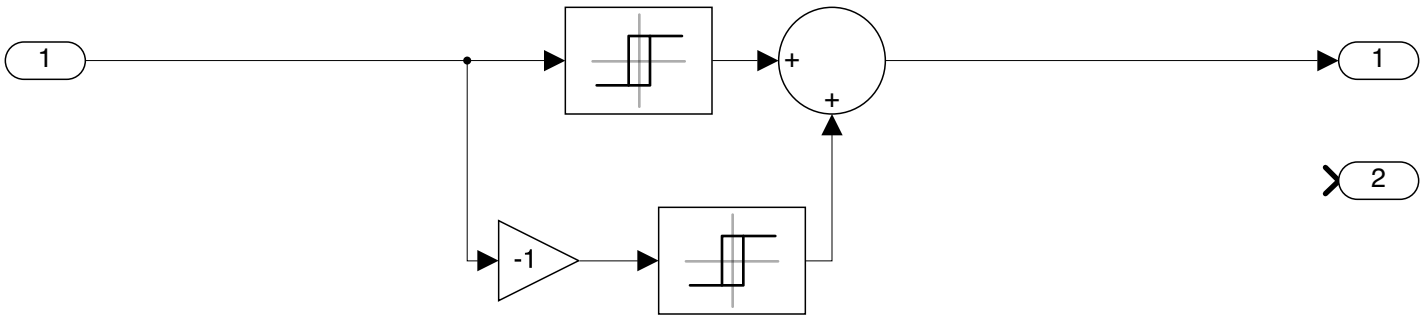

1
+
+
1
-1

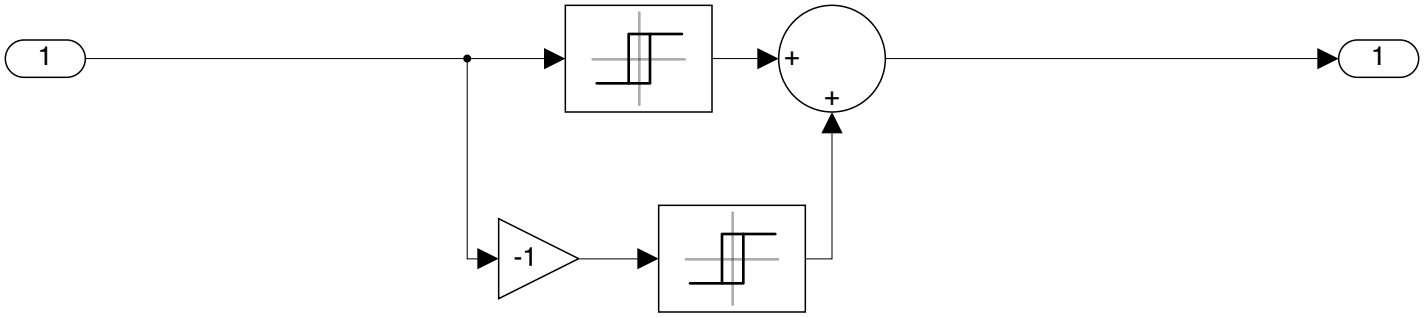

1
+
+
1
-1

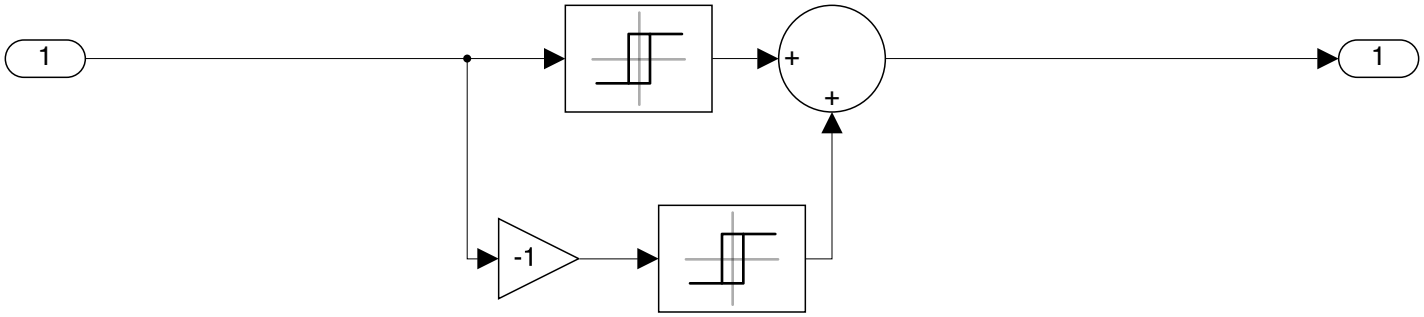

1
+
+
1
-1

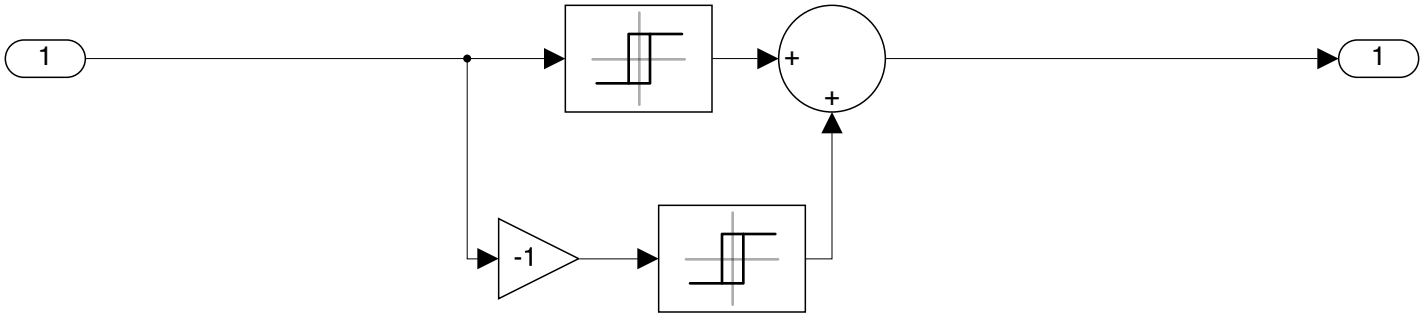

1
+
+
1
-1

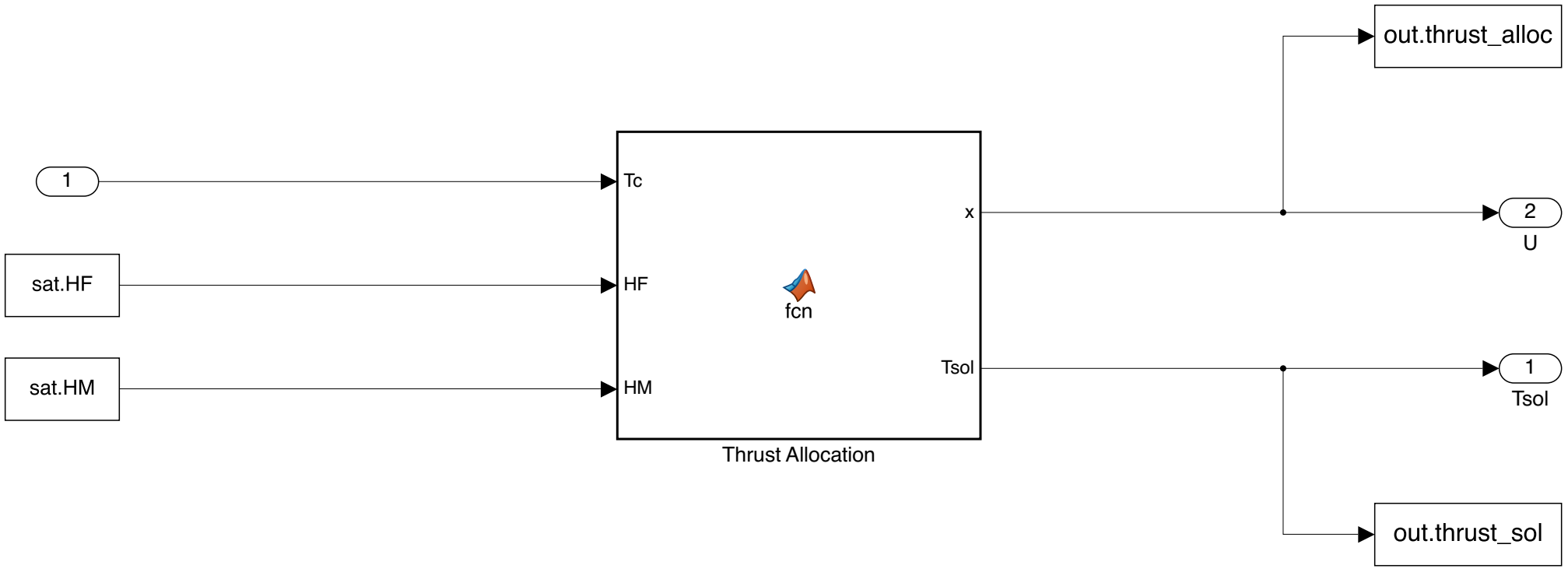

1
+
+
1
-1


1
+
+
1
2
-1


1
+
+
1
-1


1
+
+
1
-1


1
+
+
1
-1


1
sat.HF
sat.HM
Tc
HF
HM
fcn
x
Tsol
Thrust Allocation
out.thrust_alloc
2
U
1
Tsol
out.thrust_sol

```
function [x, Tsol] = fcn(Tc, HF, HM)

coder.extrinsic('linprog');
coder.extrinsic('intlinprog');
coder.extrinsic('optimoptions')

intlinprogflag = 0;

x = zeros(12,1);
lb = zeros(12,1);   % lower bound
ub = 50 * ones(12,1);   % upper bound
f = ones(1,12); % cost function to minimize is the sum of all thrusters

translate_sol = zeros(3,1); % want 0 translation
moment_sol = Tc;    % need to reach ideal Tc using thruster alloc

H = [HF;HM];    % combine translate/moment matrices

%Aeq = H;    % allocation matrix
Aeq = HM;

%Beq = [translate_sol;moment_sol]; % solution matrix
Beq = moment_sol;



if intlinprogflag == 1
    intcon = 1:12;
    options = optimoptions('intlinprog', 'Display', 'off');
    x = intlinprog(f, intcon, [], [], Aeq, Beq,  lb, ub, [], options);   % solves linprog problem

else
    options = optimoptions('linprog', 'Display', 'off');
    x = linprog(f, [], [], Aeq, Beq, lb, ub, options);
end

Tsol = Aeq * x;









Tsol = HM*x;    % gets torque from thruster allocation
```