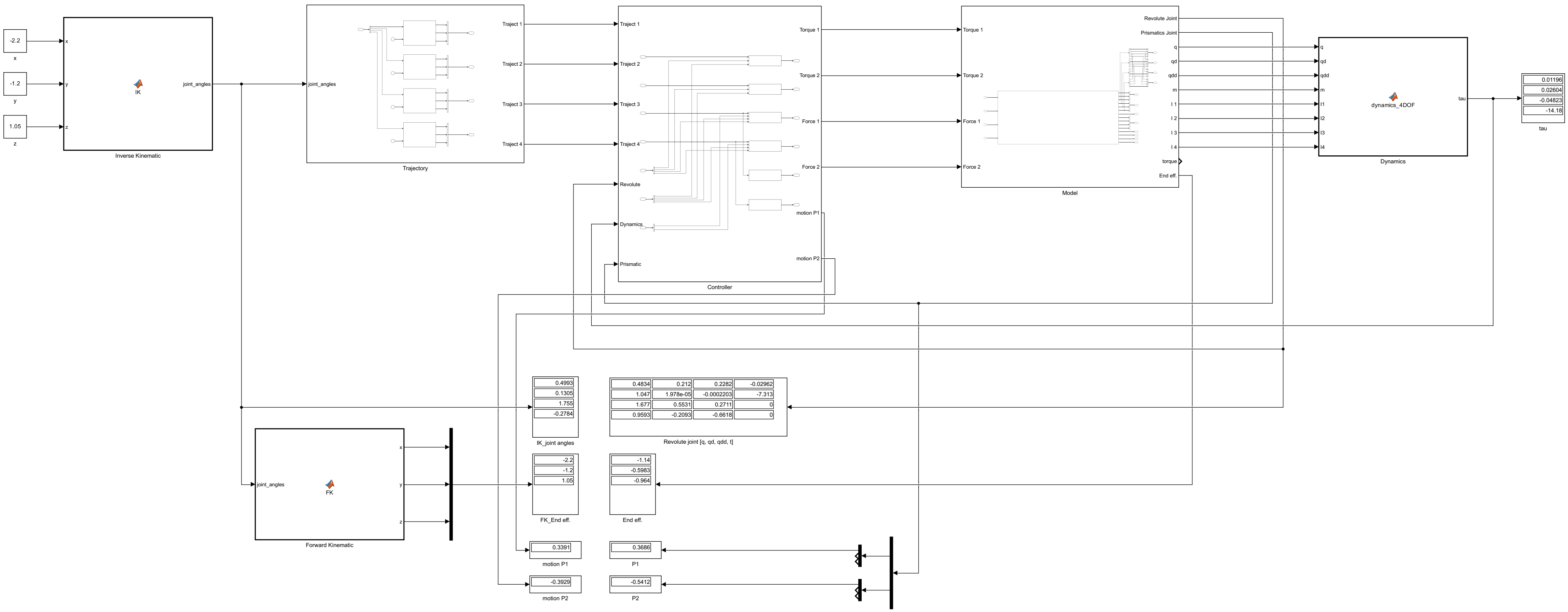
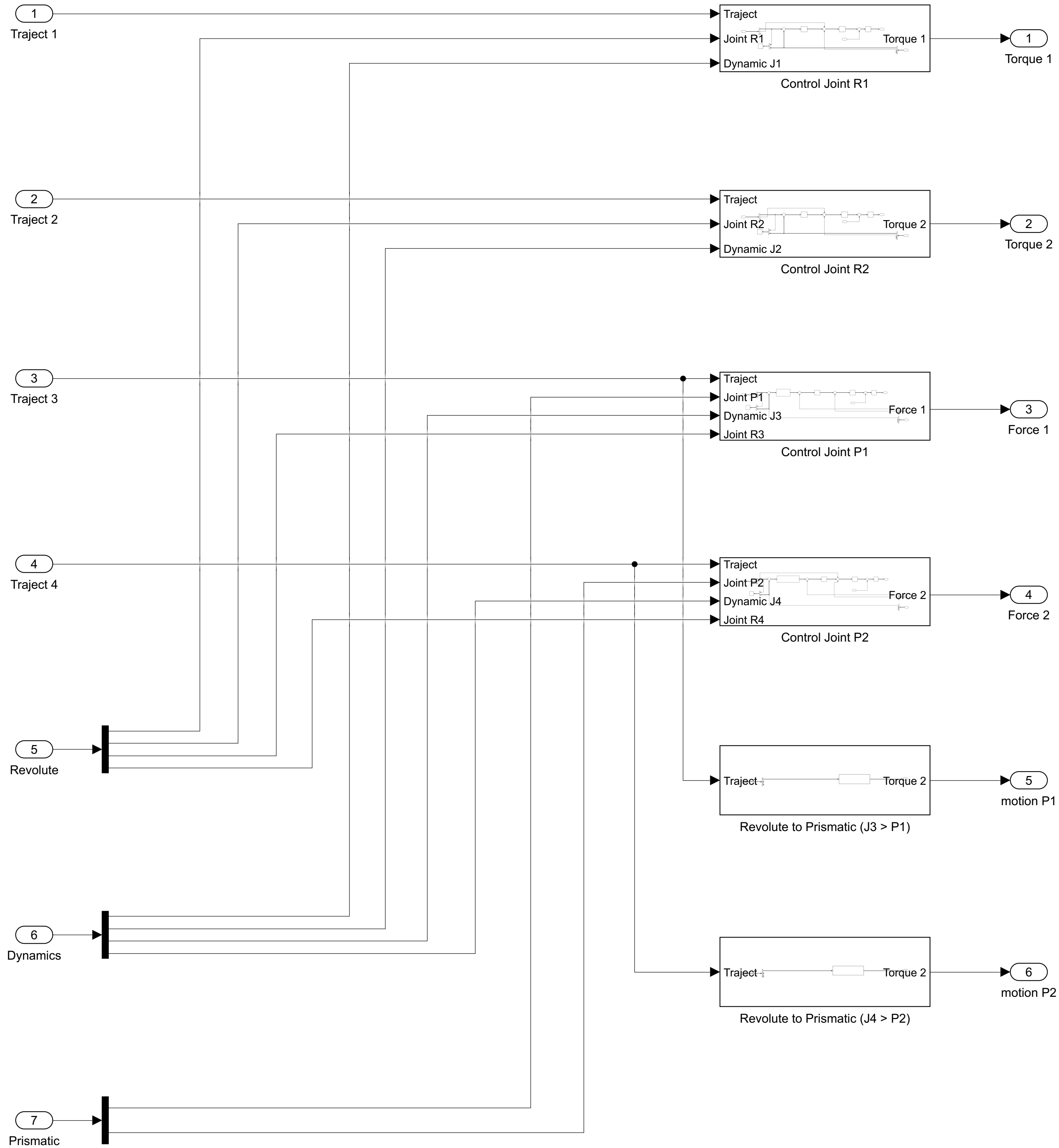
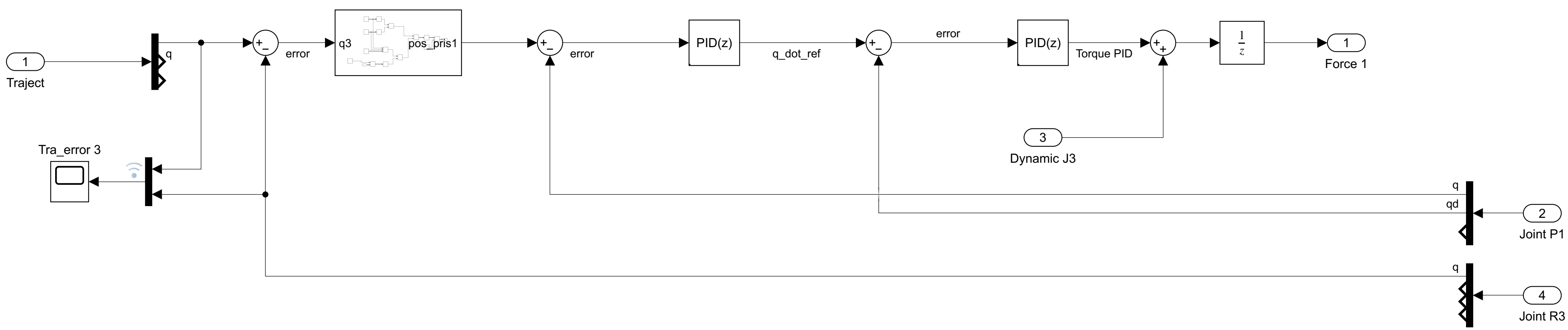
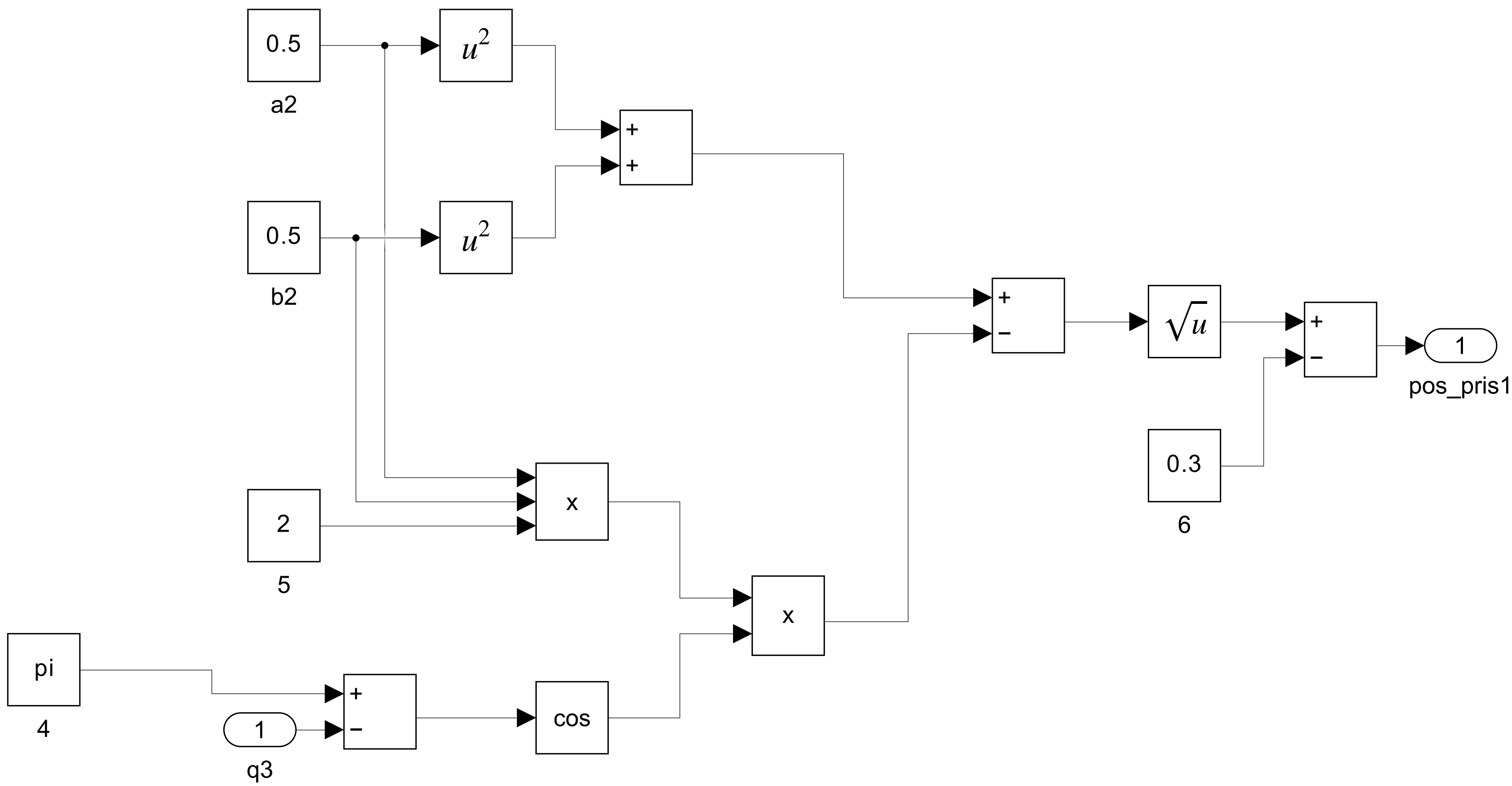
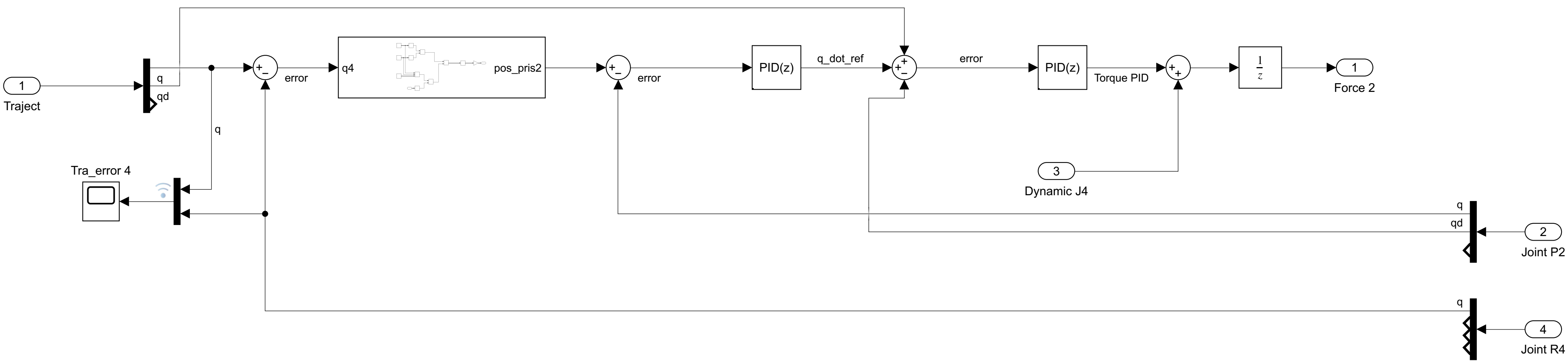
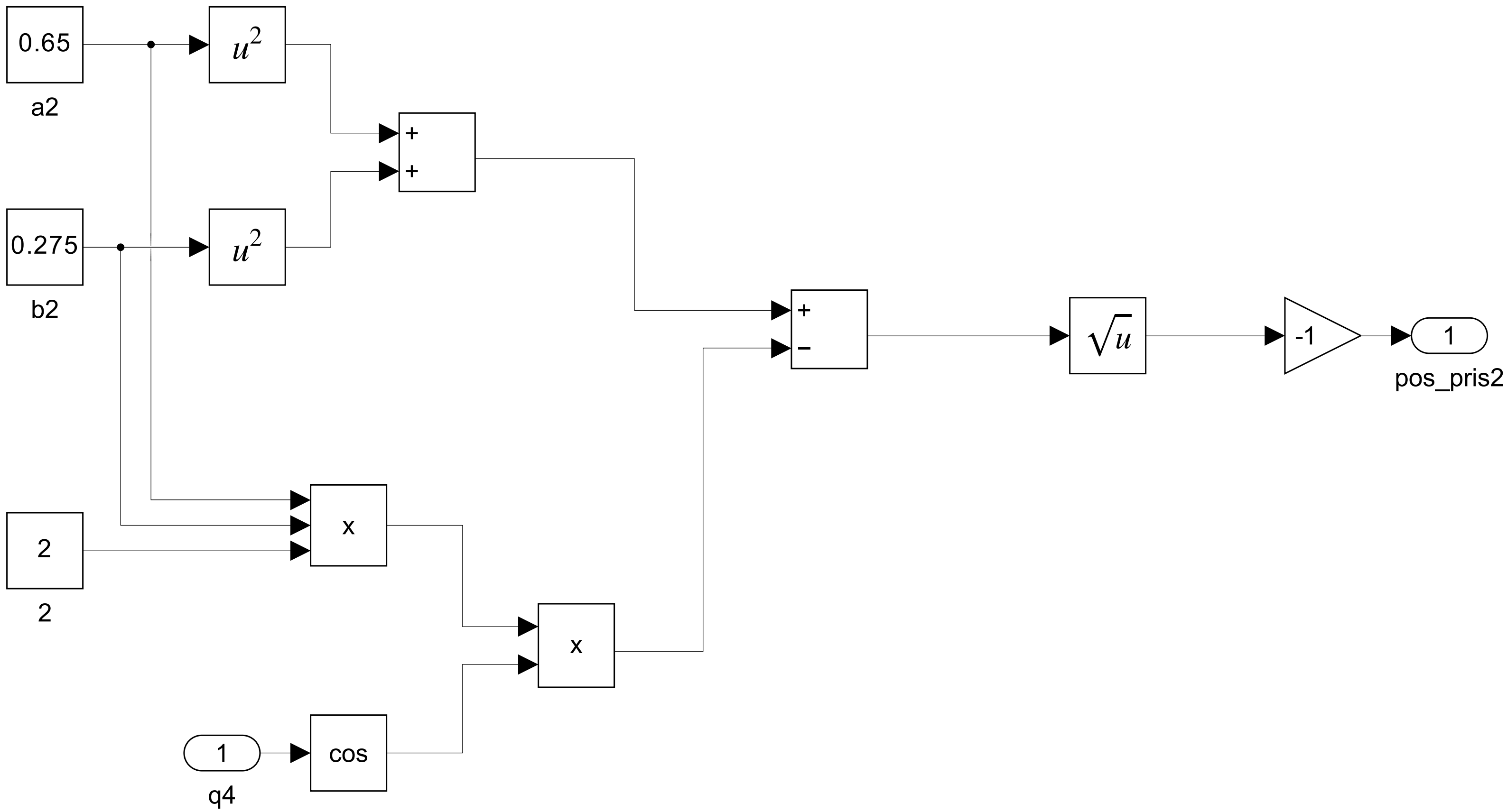

-2.2
x
-1.2
y
1.05
z
Inverse Kinematic
IK
joint_angles
Trajectory
Traject 1
Traject 2
Traject 3
Traject 4
Controller
Revolute
Dynamics
Prismatic
Torque 1
Torque 2
Force 1
Force 2
motion P1
motion P2
Model
Revolute Joint
Prismatics Joint
q
qd
qdd
m
I 1
I 2
I 3
I 4
torque
End eff.
Dynamics
dynamics_4DOF
tau
0.01196
0.02604
-0.04823
-14.18
tau
0.4993
0.1305
1.755
-0.2784
IK_joint angles
0.4834 0.212 0.2282 -0.02962
1.047 1.978e-05 -0.0002203 -7.313
1.677 0.5531 0.2711 0
0.9593 -0.2093 -0.6618 0
Revolute joint [q, qd, qdd, t]
Forward Kinematic
FK
-2.2
-1.2
1.05
FK_End eff.
-1.14
-0.5983
-0.964
End eff.
0.3391
motion P1
0.3686
P1
-0.3929
motion P2
-0.5412
P2

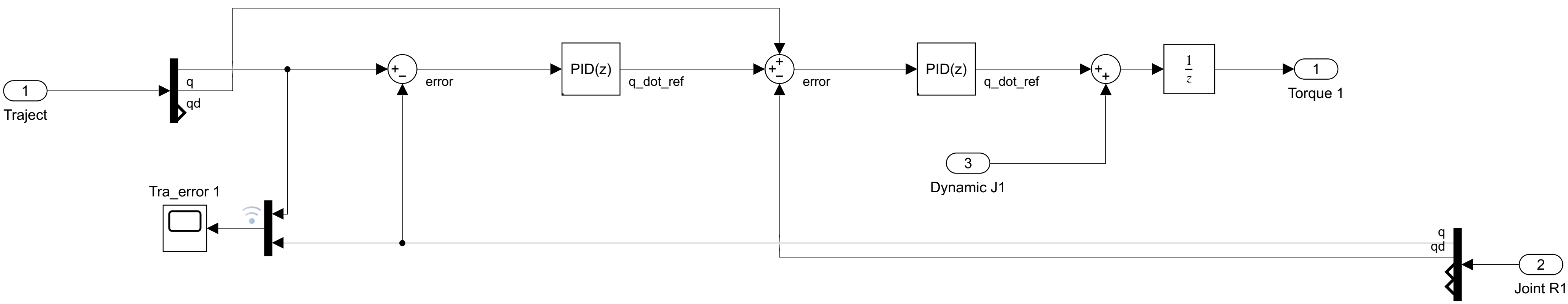

1
Traject 1
2
Traject 2
3
Traject 3
4
Traject 4
5
Revolute
6
Dynamics
7
Prismatic
Traject
Joint R1
Dynamic J1
Torque 1
Control Joint R1
Traject
Joint R2
Dynamic J2
Torque 2
Control Joint R2
Traject
Joint P1
Dynamic J3
Joint R3
Force 1
Control Joint P1
Traject
Joint P2
Dynamic J4
Joint R4
Force 2
Control Joint P2
Traject
Torque 2
Revolute to Prismatic (J3 > P1)
Traject
Torque 2
Revolute to Prismatic (J4 > P2)
1
Torque 1
2
Torque 2
3
Force 1
4
Force 2
5
motion P1
6
motion P2

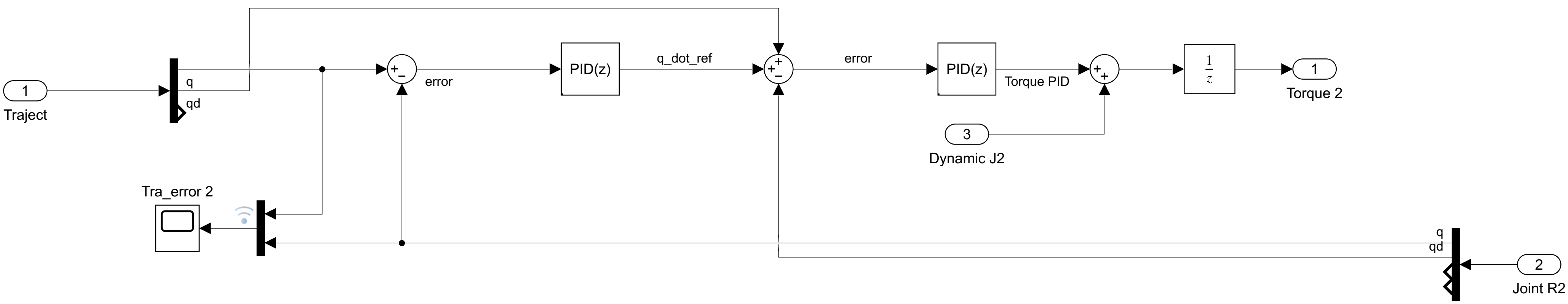

1
Traject
q
error
q3
pos_pris1
error
PID(z)
q_dot_ref
error
PID(z)
Torque PID
1
Force 1
3
Dynamic J3
Tra_error 3
q
qd
2
Joint P1
q
4
Joint R3


0.5
a2
0.5
b2
2
5
pi
4
1
q3
cos
x
x
0.3
6
1
pos_pris1

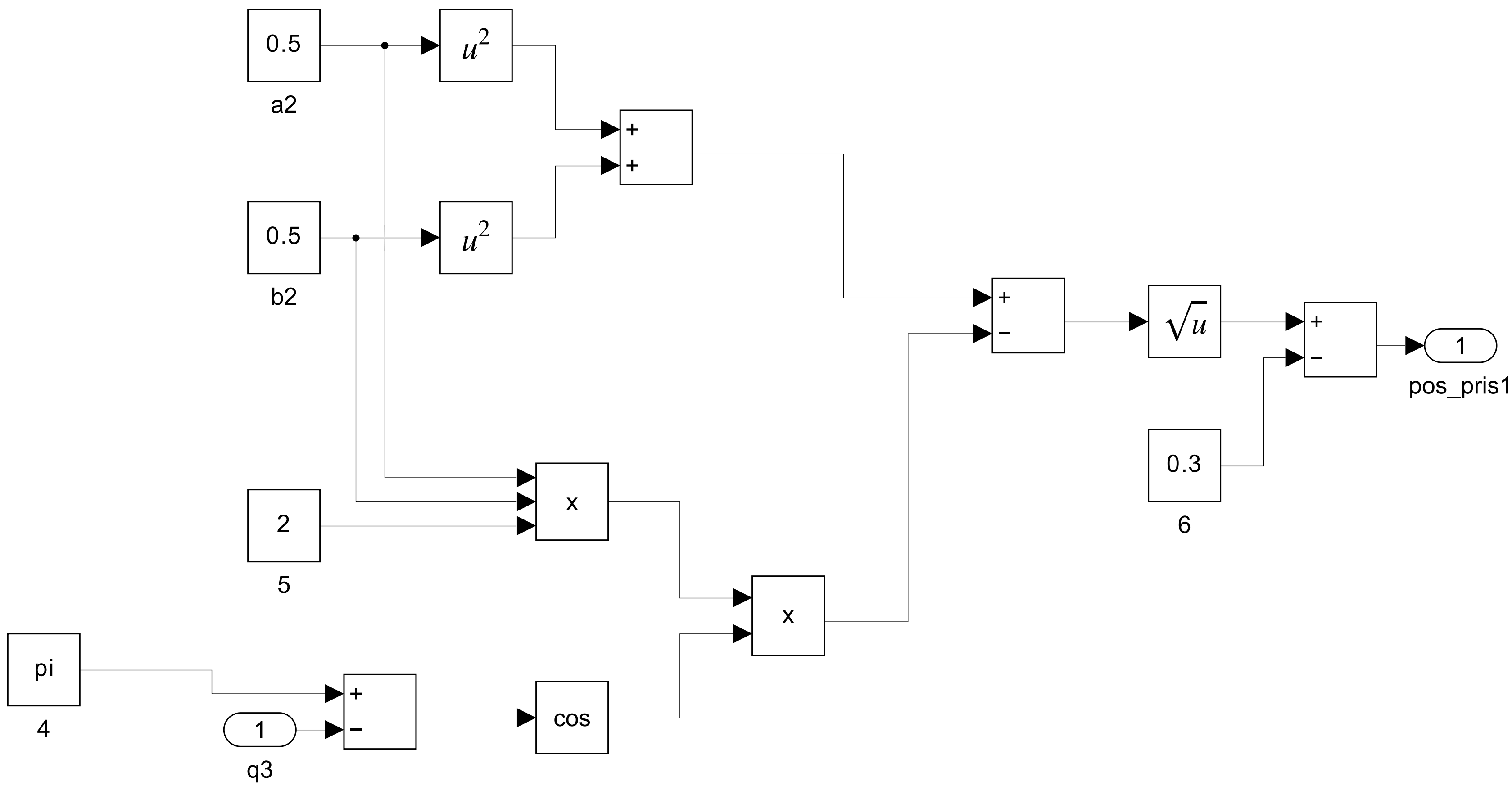

1
Traject
q
qd
q
Tra_error 4
error
q4
pos_pris2
error
PID(z)
q_dot_ref
error
PID(z)
Torque PID
1
z
1
Force 2
3
Dynamic J4
q
qd
2
Joint P2
q
4
Joint R4


0.65
a2
0.275
b2
2
2
1
q4
cos
x
x
+
+
+
−
-1
1
pos_pris2

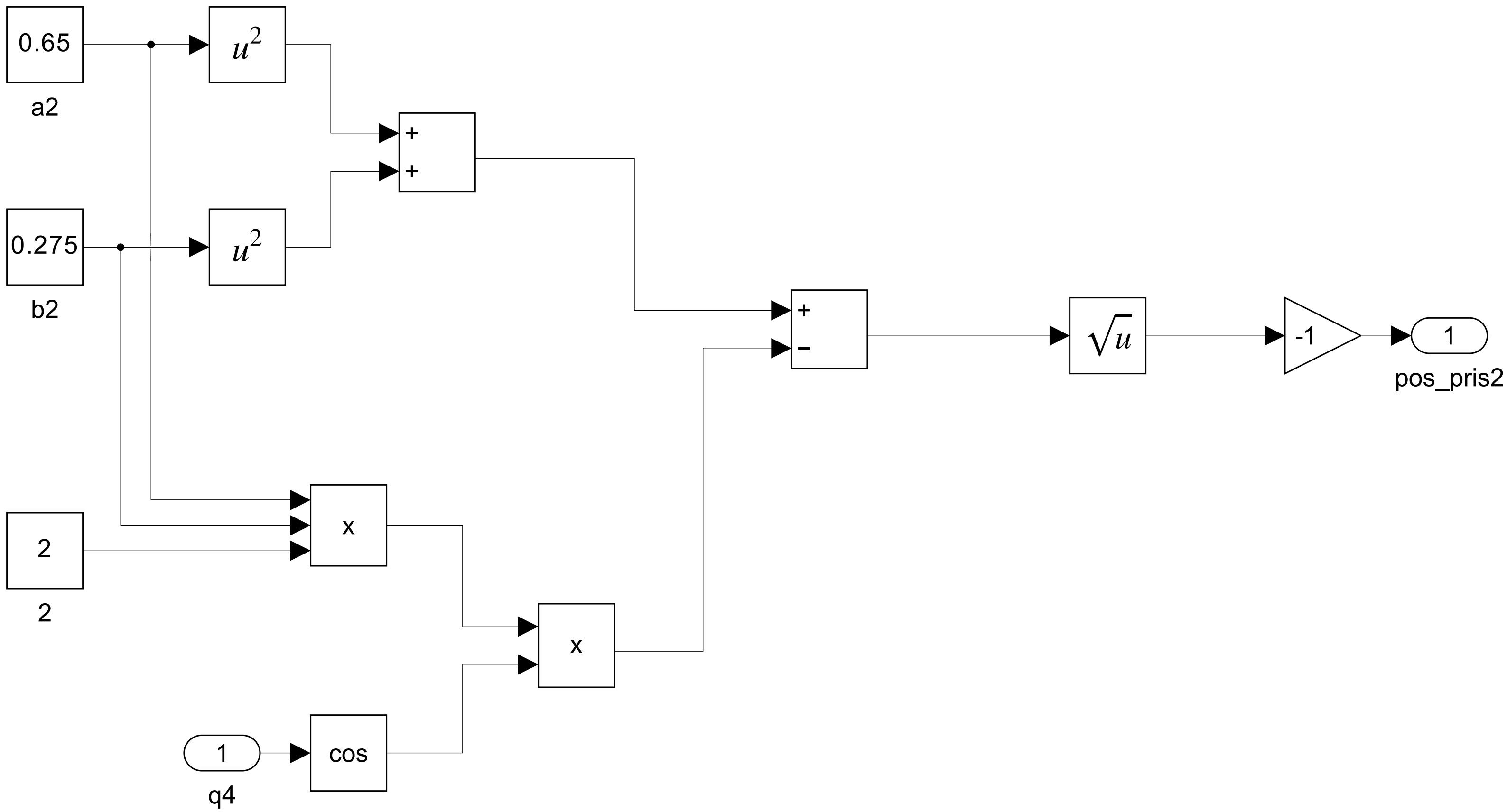

1
Traject
q
qd
PID(z)
error
q_dot_ref
PID(z)
error
q_dot_ref
1/z
1
Torque 1
3
Dynamic J1
Tra_error 1
q
qd
2
Joint R1


1
Traject
q
qd
error
PID(z)
q_dot_ref
error
PID(z)
Torque PID
1
z
1
Torque 2
3
Dynamic J2
Tra_error 2
q
qd
2
Joint R2


1
Traject
q
q
q3
pos_pris1
1
Torque 2


0.5
a2
0.5
b2
u^2
u^2
+
+
2
5
x
x
pi
4
1
q3
+
−
cos
+
−
$\sqrt{u}$
+
−
0.3
6
1
pos_pris1


1
Traject
q
q
q4
pos_pris2
1
Torque 2

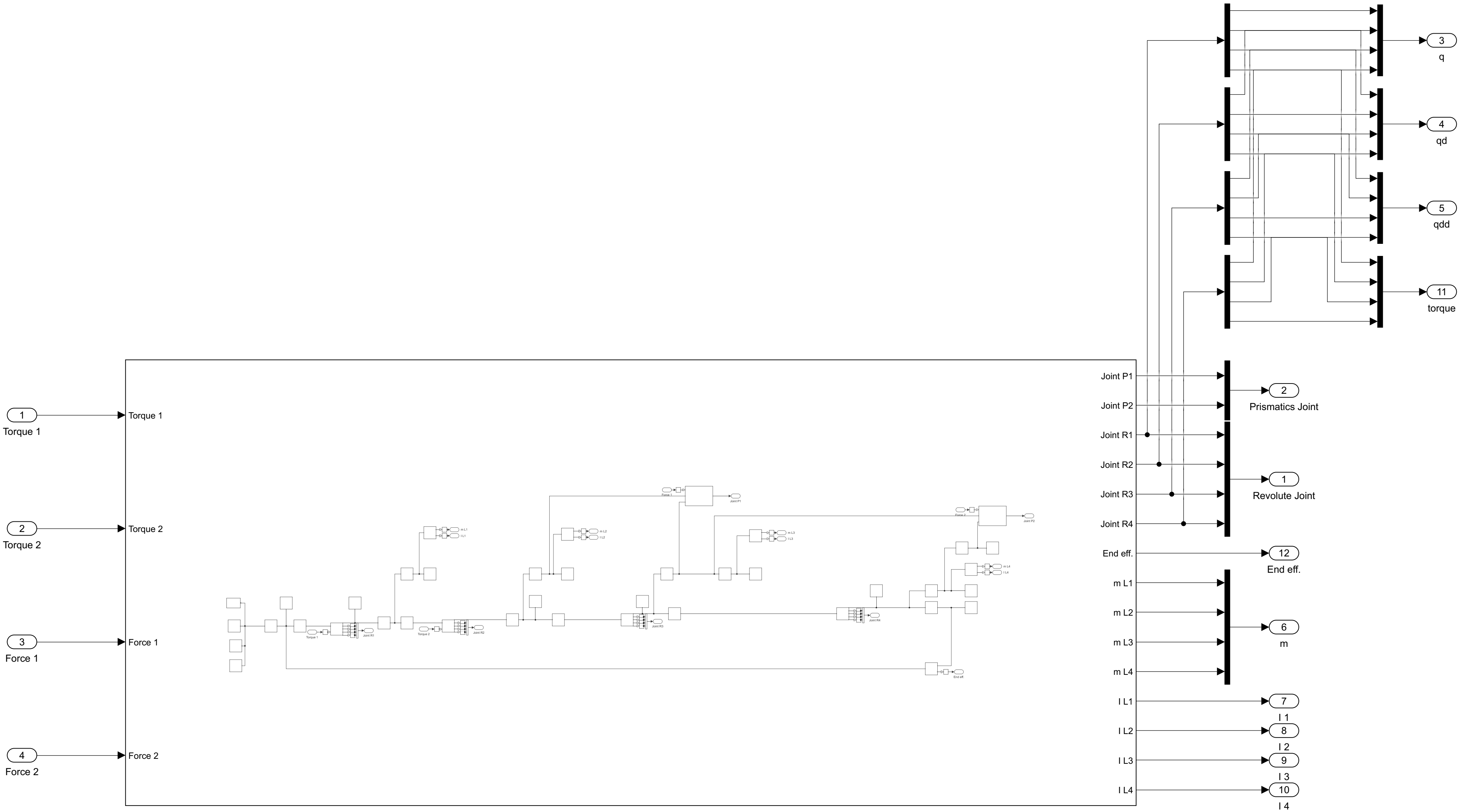

0.65
a2
0.275
b2
2
2
1
q4
cos
+
+
+
−
x
x
-1
1
pos_pris2

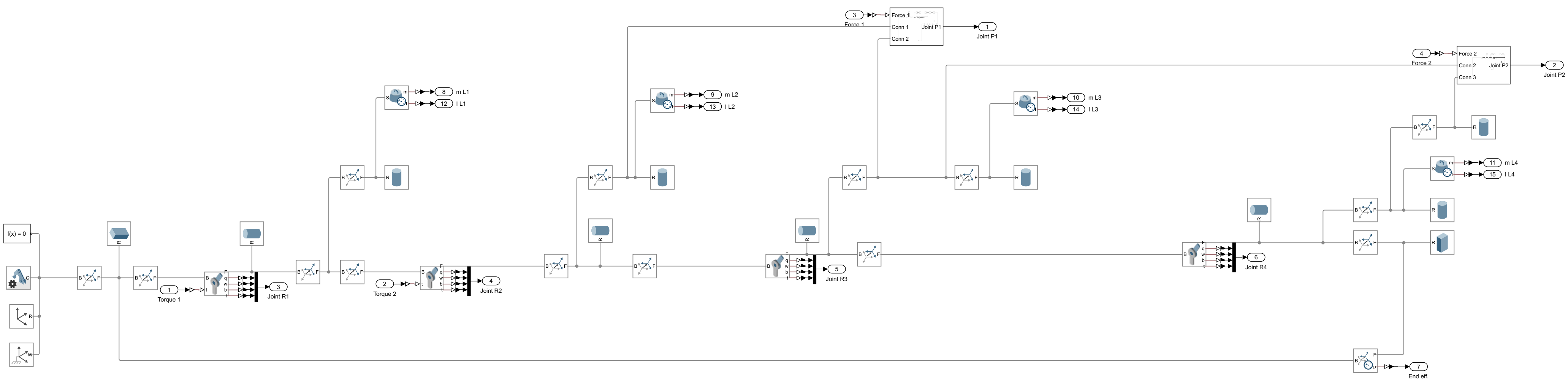
```
function tau = dynamics_4DOF(q, qd, qdd, m, I1, I2, I3, I4)
    % dynamics_4DOF: Calculates joint torques for a 4-DOF manipulator
    % Input:
    %   q: Joint angles [q1; q2; q3; q4]
    %   qd: Joint velocities [qd1; qd2; qd3; qd4]
    %   qdd: Joint accelerations [qdd1; qdd2; qdd3; qdd4]
    %   params: Struct with manipulator parameters
    %       params.m: Mass of links [m1, m2, m3, m4]
    %       params.L: Lengths of links [L1, L2, L3, L4]
    %       params.I: Inertia matrices for links (cell array of 3x3 matrices)
    %       params.g: Gravity (e.g., 9.81 m/s^2)
    % Output:
    %   tau: Joint torques [tau1; tau2; tau3; tau4]

    % กำหนดขนาดของเอาต์พุต
    tau = zeros(4, 1); % กำหนดให้ tau เป็นเวกเตอร์ขนาด 4x1

    assert(isequal(size(q), [4, 1]));      % กำหนดว่า q ต้องมีขนาด 4x1
    assert(isequal(size(qd), [4, 1]));     % qd ต้องมีขนาด 4x1
    assert(isequal(size(qdd), [4, 1]));    % qdd ต้องมีขนาด 4x1
    assert(isequal(size(I1), [3, 3])); % inertiaMatrix ต้องเป็น 3x3
    assert(isequal(size(I2), [3, 3])); % inertiaMatrix ต้องเป็น 3x3
    assert(isequal(size(I3), [3, 3])); % inertiaMatrix ต้องเป็น 3x3
    assert(isequal(size(I4), [3, 3])); % inertiaMatrix ต้องเป็น 3x3

    % Unpack parameters
    % m = params.m;    % Masses
    L = [0.25, 1, 1.5, 0.75];    % Link lengths
    I = [I1, I2, I3, I4];    % Cell array of inertia matrices (3x3)
    g = 9.81;    % Gravity acceleration

    % Joint variables
    q1 = q(1); q2 = q(2); q3 = q(3); q4 = q(4);
    qd1 = qd(1); qd2 = qd(2); qd3 = qd(3); qd4 = qd(4);
    qdd1 = qdd(1); qdd2 = qdd(2); qdd3 = qdd(3); qdd4 = qdd(4);

    % Gravitational forces
    G1 = m(1) * g * L(1) / 2 * cos(q1);
    G2 = m(2) * g * (L(1) * cos(q1) + L(2) / 2 * cos(q1 + q2));
    G3 = m(3) * g * (L(1) * cos(q1) + L(2) * cos(q1 + q2) + L(3) / 2 * cos(q1 + q2 + q3));
    G4 = m(4) * g * (L(1) * cos(q1) + L(2) * cos(q1 + q2) + L(3) * cos(q1 + q2 + q3) + L(4) / 2 * cos(q1 + q2 + q3 + q4));
    G = [G1; G2; G3; G4];

    % Inertia matrix
    M = zeros(4, 4);
    for i = 1:4
        M(i, i) = I(3, 3); % Use principal moment of inertia along z-axis
    end

    % Coriolis and Centrifugal forces (simplified)
    C = zeros(4, 4);

    % Compute torques
    tau = M * qdd + C * qd + G;
end
```

```
function [x, y, z] = FK(joint_angles)
    % FK_Simulink: Calculates the position and orientation of the end effector for a 4-DOF manipulator

    % Lengths of the links
    d1 = 0.3;  % {0} -> {1}
    a1 = 0.2;  % {1} -> {2}
    L1 = 0.25; % Link 1
    L2 = 1;    % Link 2
    L3 = 1.5;  % Link 3
    L4 = 0.75; % Link 4 (End Effector)

    % Extract joint angles
    q1 = joint_angles(1); % Joint 1 (base rotation)
    q2 = joint_angles(2); % Joint 2 (shoulder rotation)
    q3 = joint_angles(3); % Joint 3 (elbow rotation)
    q4 = joint_angles(4); % Joint 4 (end effector orientation)

    x12 = pi/2;
    z23 = pi/2;

    % Transformation Matrices
    % Transform from base frame to end effector
    % Transformation for Joint 0 -> 1 (Rotation around the base)
    T01_T1 = [  1, 0, 0,  0;
                0, 1, 0,  0;
                0, 0, 1, d1;
                0, 0, 0,  1];

    T01_Rz = [  cos(q1), -sin(q1), 0, 0;
                sin(q1),  cos(q1), 0, 0;
                      0,        0, 1, 0;
                      0,        0, 0, 1];

    T01 = T01_T1 * T01_Rz;

    % Transformation for Joint 1 -> 2 (shoulder)
    T12_T1 = [  1, 0, 0, -a1;
                0, 1, 0,   0;
                0, 0, 1,  L1;
                0, 0, 0,   1];

    T12_Rx = [  1,         0,          0, 0;
                0, cos(x12), -sin(x12), 0;
                0, sin(x12),  cos(x12), 0;
                0,         0,          0, 1];

    T12_Rz = [  cos(q2), -sin(q2), 0, 0;
                sin(q2),  cos(q2), 0, 0;
                      0,        0, 1, 0;
                      0,        0, 0, 1];

    T12 = T12_T1 * T12_Rx * T12_Rz;

    % Transformation for Joint 2 -> 3 (elbow)
    T23_Rz1 = [  cos(z23), -sin(z23), 0, 0;
                 sin(z23),  cos(z23), 0, 0;
                        0,         0, 1, 0;
                        0,         0, 0, 1];
```

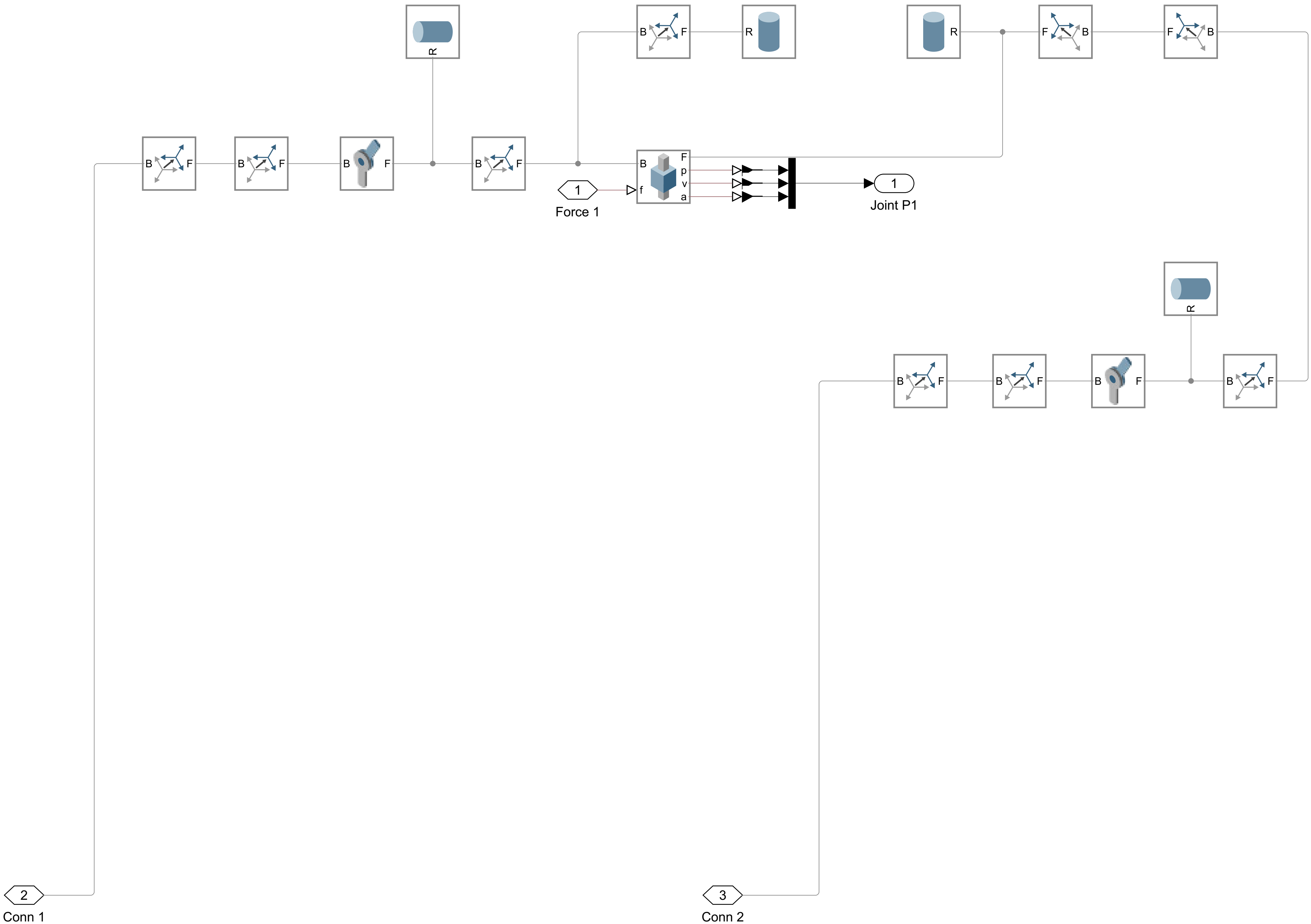
```
    T23_T1 = [  1, 0, 0, L2;
                0, 1, 0,  0;
                0, 0, 1,  0;
                0, 0, 0,  1];

    T23_Rz = [  cos(q3), -sin(q3), 0, 0;
                sin(q3),  cos(q3), 0, 0;
                      0,        0, 1, 0;
                      0,        0, 0, 1];

    T23 = T23_Rz1 * T23_T1 * T23_Rz;

    % Transformation for Joint 3 -> 4
    T34_T1 = [  1, 0, 0, L3;
                0, 1, 0,  0;
                0, 0, 1,  0;
                0, 0, 0,  1];

    T34_Rz = [  cos(q4), -sin(q4), 0, 0;
                sin(q4),  cos(q4), 0, 0;
                      0,        0, 1, 0;
                      0,        0, 0, 1];

    T34 = T34_T1 * T34_Rz;

    % Transformation for Joint 4 -> e

    T4e = [ 1, 0, 0, L4;
            0, 1, 0,  0;
            0, 0, 1,  0;
            0, 0, 0,  1];

    % Final transformation from the base to end effector
    T = T01 * T12 * T23 * T34 * T4e;

    % Extract position from the final transformation matrix
    x = T(1, 4); % x-position of the end effector
    y = T(2, 4); % y-position of the end effector
    z = T(3, 4); % z-position of the end effector


end

% Tranformation matrix
% ([  [-sin(q2 + q3 + q4)*cos(q1), -cos(q1)*cos(q2 + q3 + q4),  sin(q1), -L2*sin(q2)*cos(q1) - L3*sin(q2 + q3)*cos(q1) - L4*sin(q2
%     [-sin(q1)*sin(q2 + q3 + q4), -sin(q1)*cos(q2 + q3 + q4), -cos(q1), -L2*sin(q1)*sin(q2) - L3*sin(q1)*sin(q2 + q3) - L4*sin(q1
%     [          cos(q2 + q3 + q4),          -sin(q2 + q3 + q4),        0,                      L1 + L2*cos(q2) + L3*(1.0*cos(q2 + q3
%     [                          0,                           0,        0,
```

```matlab
%#codegen
function joint_angles = IK(x, y, z)
    % กำหนดขนาดคงที่สำหรับ output solutions
    joint_angles = zeros(1, 4); % solutions จะมีขนาด [1x4] เสมอ

    % Lengths of the links
    d1 = 0.3;
    a1 = 0.2;
    L1 = 0.25;
    L2 = 1;
    L3 = 1.5;
    L4 = 0.75;

    % Initial guess for q1, q2, q3, q4
    initial_guess = [0, 0, 0, 0];

    % Define the function to solve
    function F = equations(vars)
        q1 = vars(1);
        q2 = vars(2);
        q3 = vars(3);
        q4 = vars(4);
        F = [
            -L2*sin(q2)*cos(q1) - L3*sin(q2 + q3)*cos(q1) - L4*sin(q2 + q3 + q4)*cos(q1) - a1*cos(q1) - x;
            -L2*sin(q1)*sin(q2) - L3*sin(q1)*sin(q2 + q3) - L4*sin(q1)*sin(q2 + q3 + q4) - a1*sin(q1) - y;
            L1 + L2*cos(q2) + L3*cos(q2 + q3) + L4*cos(q2 + q3 + q4) + d1 - z;
        ];
    end

    % Solve the equations numerically
    options = optimoptions('fsolve', 'Algorithm', 'levenberg-marquardt', 'Display', 'none');
    joint_angles = fsolve(@equations, initial_guess, options);
end
```

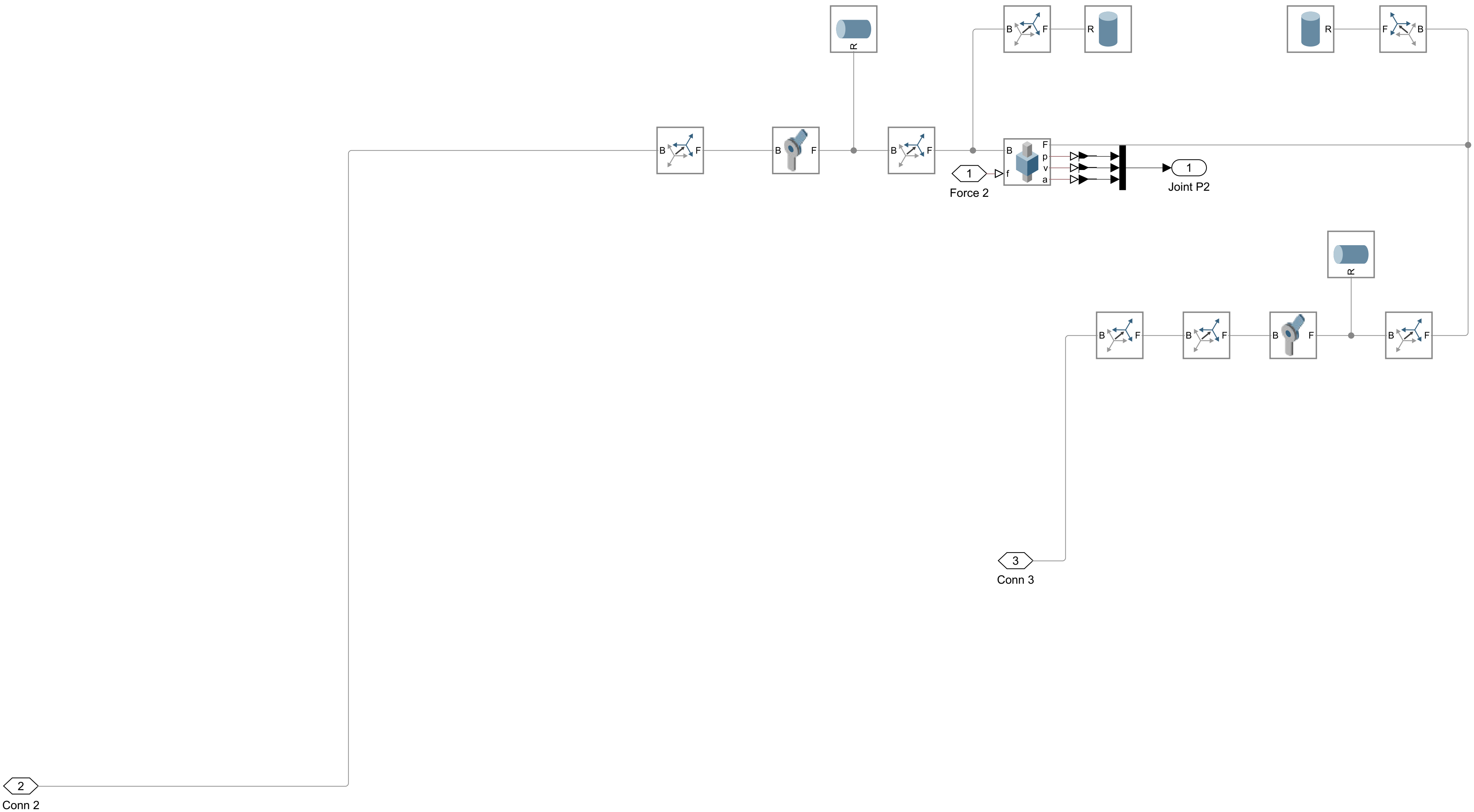

1
Torque 1
2
Torque 2
3
Force 1
4
Force 2
Torque 1
Torque 2
Force 1
Force 2
Joint P1
Joint P2
Joint R1
Joint R2
Joint R3
Joint R4
End eff.
m L1
m L2
m L3
m L4
I L1
I L2
I L3
I L4
3
q
4
qd
5
qdd
11
torque
2
Prismatics Joint
1
Revolute Joint
12
End eff.
6
m
7
I 1
8
I 2
9
I 3
10
I 4



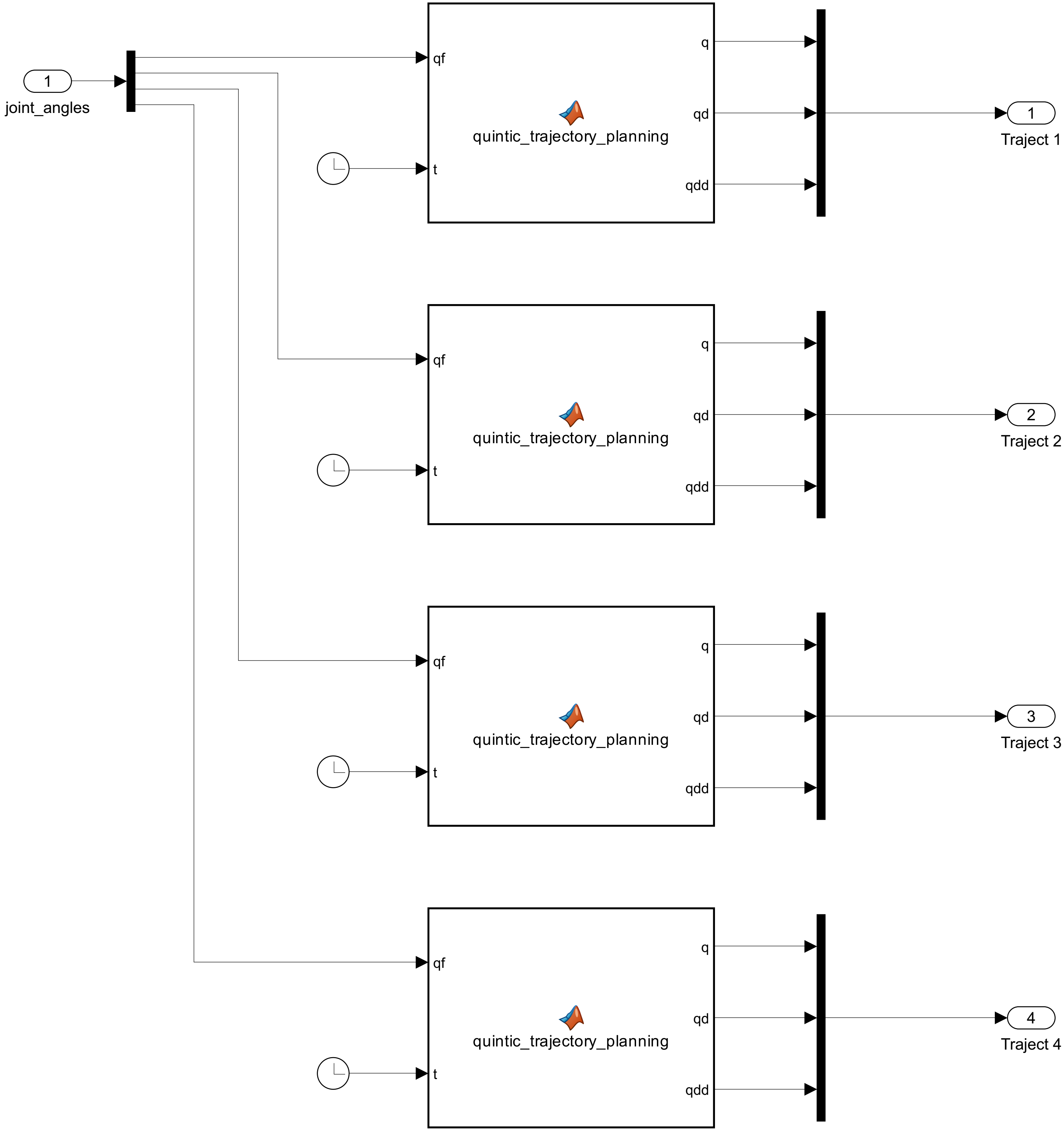
f(x) = 0
Torque 1
Joint R1
Torque 2
Joint R2
Joint R3
Joint R4
Force 1
Conn 1
Conn 2
Joint P1
Force 2
Conn 2
Conn 3
Joint P2
m L1
I L1
m L2
I L2
m L3
I L3
m L4
I L4
End eff.



Force 1
Joint P1
Conn 1
Conn 2



Force 2
Joint P2
Conn 3
Conn 2


1
joint_angles
qf
t
q
qd
qdd
quintic_trajectory_planning
1
Traject 1
2
Traject 2
3
Traject 3
4
Traject 4

```matlab
function [q, qd, qdd] = quintic_trajectory_planning(qf, t)
    % Quintic Trajectory Planning with reference to Simulink Clock for 3-second duration

    % Initial conditions (set to zero)
    q0 = 0;      % Initial Position
    qd0 = 0;     % Initial Velocity
    qdd0 = 0;    % Initial Acceleration
    qdf = 0;     % Final Velocity
    qddf = 0;    % Final Acceleration

    % Total time for trajectory
    T = 0.5;  % 3 seconds duration

    % Clamp time to ensure it does not exceed 3 seconds
    t = min(t, T);

    % Polynomial coefficients calculation based on boundary conditions
    a0 = q0;
    a1 = qd0;
    a2 = qdd0 / 2;
    a3 = (20 * qf - 20 * q0 - 8 * qdf - 12 * qd0 - 3 * qdd0 + qddf) / (2 * T^3);
    a4 = (30 * q0 - 30 * qf + 14 * qdf + 16 * qd0 + 3 * qdd0 - 2 * qddf) / (2 * T^4);
    a5 = (12 * qf - 12 * q0 - 6 * qdf - 6 * qd0 - qdd0 + qddf) / (2 * T^5);

    % Calculate position, velocity, and acceleration based on time t
    q = a0 + a1 * t + a2 * t^2 + a3 * t^3 + a4 * t^4 + a5 * t^5;
    qd = a1 + 2 * a2 * t + 3 * a3 * t^2 + 4 * a4 * t^3 + 5 * a5 * t^4;
    qdd = 2 * a2 + 6 * a3 * t + 12 * a4 * t^2 + 20 * a5 * t^3;

    % Ensure the final position q reaches qf exactly after 3 seconds
    if t == T
        q = qf;
        qd = 0;  % Final velocity is zero if not specified
        qdd = 0; % Final acceleration is zero if not specified
    end
end
```

```
function [q, qd, qdd] = quintic_trajectory_planning(qf, t)
    % Quintic Trajectory Planning with reference to Simulink Clock for 3-second duration

    % Initial conditions (set to zero)
    q0 = 0;      % Initial Position
    qd0 = 0;     % Initial Velocity
    qdd0 = 0;    % Initial Acceleration
    qdf = 0;     % Final Velocity
    qddf = 0;    % Final Acceleration

    % Total time for trajectory
    T = 0.5;  % 3 seconds duration

    % Clamp time to ensure it does not exceed 3 seconds
    t = min(t, T);

    % Polynomial coefficients calculation based on boundary conditions
    a0 = q0;
    a1 = qd0;
    a2 = qdd0 / 2;
    a3 = (20 * qf - 20 * q0 - 8 * qdf - 12 * qd0 - 3 * qdd0 + qddf) / (2 * T^3);
    a4 = (30 * q0 - 30 * qf + 14 * qdf + 16 * qd0 + 3 * qdd0 - 2 * qddf) / (2 * T^4);
    a5 = (12 * qf - 12 * q0 - 6 * qdf - 6 * qd0 - qdd0 + qddf) / (2 * T^5);

    % Calculate position, velocity, and acceleration based on time t
    q = a0 + a1 * t + a2 * t^2 + a3 * t^3 + a4 * t^4 + a5 * t^5;
    qd = a1 + 2 * a2 * t + 3 * a3 * t^2 + 4 * a4 * t^3 + 5 * a5 * t^4;
    qdd = 2 * a2 + 6 * a3 * t + 12 * a4 * t^2 + 20 * a5 * t^3;

    % Ensure the final position q reaches qf exactly after 3 seconds
    if t == T
        q = qf;
        qd = 0;  % Final velocity is zero if not specified
        qdd = 0; % Final acceleration is zero if not specified
    end
end
```

```
function [q, qd, qdd] = quintic_trajectory_planning(qf, t)
    % Quintic Trajectory Planning with reference to Simulink Clock for 3-second duration

    % Initial conditions (set to zero)
    q0 = 0;       % Initial Position
    qd0 = 0;      % Initial Velocity
    qdd0 = 0;     % Initial Acceleration
    qdf = 0;      % Final Velocity
    qddf = 0;     % Final Acceleration

    % Total time for trajectory
    T = 0.5;  % 3 seconds duration

    % Clamp time to ensure it does not exceed 3 seconds
    t = min(t, T);

    % Polynomial coefficients calculation based on boundary conditions
    a0 = q0;
    a1 = qd0;
    a2 = qdd0 / 2;
    a3 = (20 * qf - 20 * q0 - 8 * qdf - 12 * qd0 - 3 * qdd0 + qddf) / (2 * T^3);
    a4 = (30 * q0 - 30 * qf + 14 * qdf + 16 * qd0 + 3 * qdd0 - 2 * qddf) / (2 * T^4);
    a5 = (12 * qf - 12 * q0 - 6 * qdf - 6 * qd0 - qdd0 + qddf) / (2 * T^5);

    % Calculate position, velocity, and acceleration based on time t
    q = a0 + a1 * t + a2 * t^2 + a3 * t^3 + a4 * t^4 + a5 * t^5;
    qd = a1 + 2 * a2 * t + 3 * a3 * t^2 + 4 * a4 * t^3 + 5 * a5 * t^4;
    qdd = 2 * a2 + 6 * a3 * t + 12 * a4 * t^2 + 20 * a5 * t^3;

    % Ensure the final position q reaches qf exactly after 3 seconds
    if t == T
        q = qf;
        qd = 0;  % Final velocity is zero if not specified
        qdd = 0; % Final acceleration is zero if not specified
    end
end
```

```
function [q, qd, qdd] = quintic_trajectory_planning(qf, t)
    % Quintic Trajectory Planning with reference to Simulink Clock for 3-second duration

    % Initial conditions (set to zero)
    q0 = 0;      % Initial Position
    qd0 = 0;     % Initial Velocity
    qdd0 = 0;    % Initial Acceleration
    qdf = 0;     % Final Velocity
    qddf = 0;    % Final Acceleration

    % Total time for trajectory
    T = 0.5;  % 3 seconds duration

    % Clamp time to ensure it does not exceed 3 seconds
    t = min(t, T);

    % Polynomial coefficients calculation based on boundary conditions
    a0 = q0;
    a1 = qd0;
    a2 = qdd0 / 2;
    a3 = (20 * qf - 20 * q0 - 8 * qdf - 12 * qd0 - 3 * qdd0 + qddf) / (2 * T^3);
    a4 = (30 * q0 - 30 * qf + 14 * qdf + 16 * qd0 + 3 * qdd0 - 2 * qddf) / (2 * T^4);
    a5 = (12 * qf - 12 * q0 - 6 * qdf - 6 * qd0 - qdd0 + qddf) / (2 * T^5);

    % Calculate position, velocity, and acceleration based on time t
    q = a0 + a1 * t + a2 * t^2 + a3 * t^3 + a4 * t^4 + a5 * t^5;
    qd = a1 + 2 * a2 * t + 3 * a3 * t^2 + 4 * a4 * t^3 + 5 * a5 * t^4;
    qdd = 2 * a2 + 6 * a3 * t + 12 * a4 * t^2 + 20 * a5 * t^3;

    % Ensure the final position q reaches qf exactly after 3 seconds
    if t == T
        q = qf;
        qd = 0;  % Final velocity is zero if not specified
        qdd = 0; % Final acceleration is zero if not specified
    end
end
```