```cpp
#pragma once

#include <cmath>
#include "Math.hpp"
#include "RocketState.hpp"

// from body frame to inertial frame
inline Matrix3 eulerToR_IB(const Vector3 &euler)
{
    double phi   = euler.x;  // roll
    double theta = euler.y;  // pitch
    double psi   = euler.z;  // yaw

    double cphi = std::cos(phi); // save cosines and sines for later use
    double sphi = std::sin(phi);
    double cth  = std::cos(theta);
    double sth  = std::sin(theta);
    double cpsi = std::cos(psi);
    double spsi = std::sin(psi);

    // Z Y X sequence yaw then pitch then roll
    Matrix3 R;

    R.m11 =  cpsi * cth;
    R.m12 =  cpsi * sth * sphi - spsi * cphi;
    R.m13 =  cpsi * sth * cphi + spsi * sphi;

    R.m21 =  spsi * cth;
    R.m22 =  spsi * sth * sphi + cpsi * cphi;
    R.m23 =  spsi * sth * cphi - cpsi * sphi;

    R.m31 = -sth;
    R.m32 =  cth * sphi;
    R.m33 =  cth * cphi;

    return R;
}

// Euler angle rates from body rates [p q r] to [phi_dot; theta_dot; psi_dot]
// euler_dot = T(euler) * omega_B
inline Vector3 computeEulerDot(const Vector3 &euler,
                               const Vector3 &omega_B)
{
    double phi   = euler.x;
    double theta = euler.y;

    double cphi = std::cos(phi); // again precompute cosines and sines
    double sphi = std::sin(phi);
    double cth  = std::cos(theta);
    double sth  = std::sin(theta);

    double p = omega_B.x; // roll
    double q = omega_B.y; // pitch
    double r = omega_B.z; // yaw

    Vector3 euler_dot; // transformation

    euler_dot.x = p + sphi * std::tan(theta) * q + cphi * std::tan(theta) * r;
    euler_dot.y = cphi * q - sphi * r;
    euler_dot.z = sphi * q / cth + cphi * r / cth;

    return euler_dot;
}
```