Natural and Applied Sciences International Journal (NASIJ)
eISSN: 2788-4619 (online)
https://ideapublishers.org/index.php/nasij
Vol. 7, No. 1 (January-December), 2026, pp. 140-165
https://doi.org/10.47264/idea.nasij/7.1.8
Original Research Article

Rigid-body modeling and nonlinear state estimation of an X-configuration quadrotor UAV based on the AscTec hummingbird platform

Mamoon Amir* | Rumman Yousaf Abbasi

Department of Aeronautics and Astronautics, Institute of Space Technology, Islamabad, Pakistan.

* Corresponding Author Email: mamoonaamir08@gmail.com

Abstract:

In this paper, the mathematical modeling and state estimation of a nonlinear quadrotor UAV are presented by means of MATLAB/Simulink. A 12-state nonlinear dynamic model of the quadrotor was built up using the Newton–Euler equations. This model consists of the translational dynamics, rotational dynamics, and Euler-angle kinematics. The control inputs of the quadrotor are defined as total thrust, roll moment, pitch moment, and yaw moment. Open-loop simulation was carried out in order to investigate the dynamic characteristics of the UAV. It was observed that the dynamic behavior of the quadrotor system is inherently unstable without any feedback control mechanism. Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) approaches were considered for nonlinear state estimation and sensor fusion from measurements with noise. Results indicate that the UKF provides more accurate state tracking than the EKF, and that sensor fusion improves position and attitude estimation compared with individual, unfused sensor signals.

Article History

Received:
03-Jul-2026

Revised:
01-Aug-2026

Re-revised:
05-Sep-2026

Accepted:
08-Sep-2026

Published:
15-Sep-2026

Keywords: Quadrotor UAV, Newton–Euler equations, Nonlinear state-space modeling, Extended Kalman Filter, Unscented Kalman Filter, Jacobian linearization.

How to Cite:
Amir, M. & Abbasi, R. Y. (2026). Rigid-body modeling and nonlinear state estimation of an X-configuration quadrotor UAV based on the AscTec hummingbird platform. *Natural and Applied Sciences International Journal (NASIJ), 7*(1), 140-165. https://doi.org/10.47264/idea.nasij/7.1.8

1. Introduction

Quadrotor UAVs have emerged as one of the most significant platforms in modern robotics and autonomy. Due to their vertical take-off and landing (VTOL), hovering, and agile maneuverability capabilities, quadrotors can be used for surveillance, mapping, inspection, delivery, and research purposes (Senelt, 2010; Mahony, 2012). However, quadrotors are inherently unstable systems because of their nonlinear and coupled dynamics (Mahony, 2012; Bouabdallah, 2007). Hence, mathematical modeling and state estimation are crucial for stabilization and control design.

Fig. 1. Quadrotor — AscTec Hummingbird X-configuration platform.

1.1. Problem statement

The quadrotor system is highly nonlinear and disturbance-sensitive. Precise model building and state estimation are necessary for stability and autonomous navigation. In the absence of precise state estimation and feedback control, the quadrotor will be unstable in an open-loop configuration.

1.2. Objectives

- Development of a non-linear 12-state quadrotor model
- Implementation of the model in the MATLAB/Simulink environment
- Open-loop dynamic analysis
- Study of the Extended Kalman Filter (EKF)
- Study of the Unscented Kalman Filter (UKF)
- Nonlinear state estimation
- Stability analysis of open-loop quadrotor dynamics

1.3. Scope of this research

The scope of this work includes non-linear mathematical modeling, open-loop simulation, state-space model derivation, EKF and UKF estimation techniques, and MATLAB/Simulink programming. The design of a feedback controller is left for a future phase of this work and is not addressed here.

1.4. Related work

The nonlinear model of the quadrotor, the state-space equations, and the simulation approach used in this paper are adopted from Khalilov (2016). That study develops the modeling of a quadrotor UAV using the Newton–Euler equations, together with MATLAB/Simulink simulation, control techniques, and state-estimation methods such as the EKF and UKF. The present work reproduces and extends that modeling and estimation framework for the AscTec Hummingbird platform in an X-configuration.

2. Quadrotor System Modeling

2.1. Quadrotor Configuration

The quadrotor has four rotors arranged in a cross shape, with the opposite rotor pairs rotating in opposite directions to cancel reactive yaw torque under nominal hover. The four control inputs are defined in terms of the individual rotor thrusts T1–T4 and reaction torques Q1–Q4 as:

$$\begin{aligned} U_1 &= T_1 + T_2 + T_3 + T_4 \\ U_2 &= T_4 - T_2 \\ U_3 &= T_3 - T_1 \\ U_4 &= Q_2 + Q_4 - Q_1 - Q_3 \end{aligned}$$

where U_1 is the total thrust, U_2 the roll torque, U_3 the pitch torque, and U_4 the yaw torque.

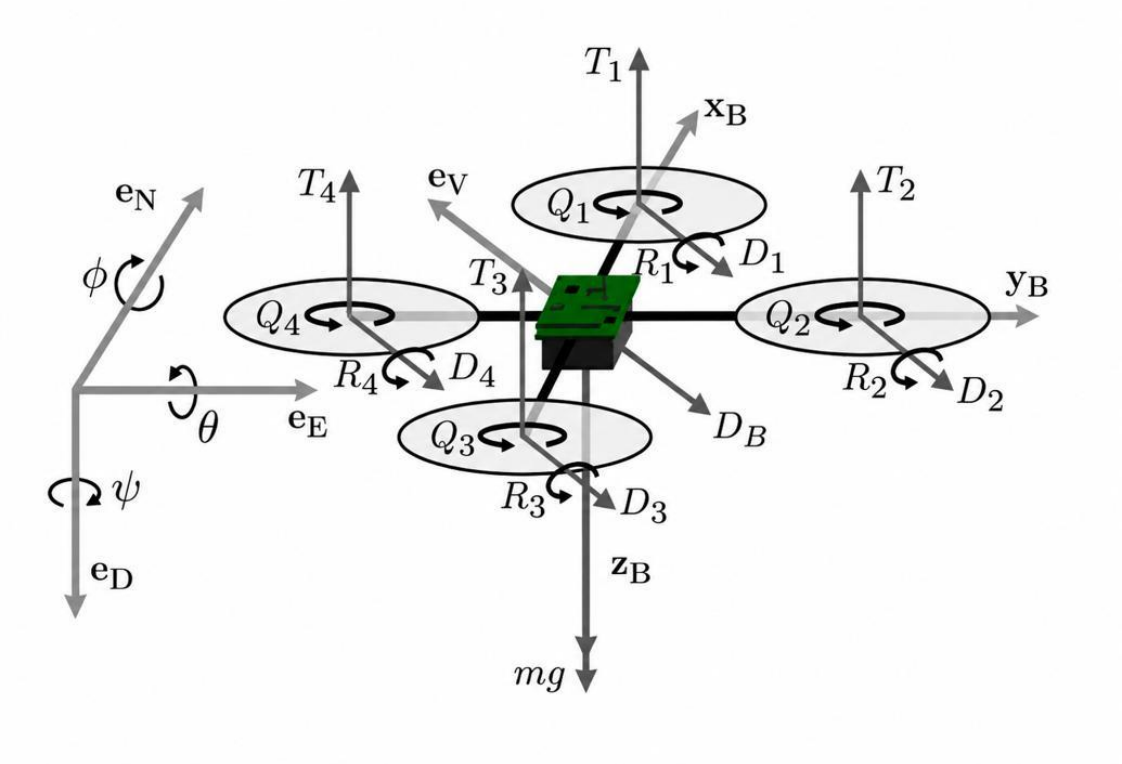

Fig. 2. Free-body diagram of the quadrotor showing rotor thrusts, reaction torques, and the body/inertial axes.

2.2. Coordinate Frames

Two coordinate frames are used in the formulation. The inertial frame is fixed to the earth and is used for position tracking, while the body frame is attached to the quadrotor and is used to express velocities and angular rates.

2.3. State Variables

The quadrotor is described by a 12-element state vector:

$$X = [\mathrm{x}, \mathrm{y}, \mathrm{z}, \mathrm{u}, \mathrm{v}, \mathrm{w}, \varphi, \theta, \psi, \mathrm{p}, \mathrm{q}, \mathrm{r}]^{\mathrm{T}}$$

Variable	Description
x, y, z	Position in the inertial frame
u, v, w	Linear velocities in the body frame
φ, θ, ψ	Euler angles (roll, pitch, yaw)
p, q, r	Angular rates (roll, pitch, yaw)

2.4. System Parameters

The physical parameters used for the AscTec Hummingbird platform are summarized in Table I.

Table I: Quadrotor system parameters

Parameter	Symbol	Value
Mass	m	0.48 kg
Gravitational acceleration	g	9.81 m/s²
Moment of inertia (x-axis)	I_x	5.6×10^{-3} kg·m²
Moment of inertia (y-axis)	I_y	5.6×10^{-3} kg·m²
Moment of inertia (z-axis)	I_z	8.1×10^{-3} kg·m²

3. Nonlinear State-Space Representation

The full nonlinear quadrotor model can be written compactly in state-space form as $\dot{\mathrm{X}} = f(\mathrm{X}, \mathrm{U})$, where the state vector $X = [\mathrm{x}, \mathrm{y}, \mathrm{z}, \mathrm{u}, \mathrm{v}, \mathrm{w}, \varphi, \theta, \psi, \mathrm{p}, \mathrm{q}, \mathrm{r}]^{\mathrm{T}}$ and the input vector $U = [\mathrm{U}_1, \mathrm{U}_2, \mathrm{U}_3, \mathrm{U}_4]^{\mathrm{T}}$. The complete set of state equations is:

Position equations:

$$\dot{\mathrm{x}} = u$$
$$\dot{\mathrm{y}} = v$$
$$\dot{\mathrm{z}} = w$$

3.1. Translational Dynamics

$$\ddot{\mathrm{x}} = \frac{\mathrm{U}_1}{\mathrm{m}}(\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)$$
$$\ddot{\mathrm{y}} = \frac{\mathrm{U}_1}{\mathrm{m}}(\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi)$$

$$\ddot{z} = \frac{U_1}{m}(\cos\phi\cos\theta) - g$$

3.2. Rotational Dynamics

$$\dot{p} = \frac{I_y - I_z}{I_x} qr + \frac{U_2}{I_x}$$

$$\dot{q} = \frac{I_z - I_x}{I_y} pr + \frac{U_3}{I_y}$$

$$\dot{r} = \frac{I_x - I_y}{I_z} pq + \frac{U_4}{I_z}$$

3.3. Euler Angle Kinematics

$$\dot{\phi} = p + q\sin\phi\tan\theta + r\cos\phi\tan\theta$$

$$\dot{\theta} = q\cos\phi - r\sin\phi$$

$$\dot{\psi} = \frac{q\sin\phi + r\cos\phi}{\cos\theta}$$

These twelve first-order differential equations constitute the nonlinear state-space model of the quadrotor, following the thesis-based Newton–Euler formulation used in the accompanying Simulink implementation.

4. MATLAB/Simulink Implementation

4.1. Simulink Model

The nonlinear equations of motion were implemented using a MATLAB Function block embedded in a Simulink model. The model consists of constant input blocks representing U1–U4, the MATLAB Function block encoding f(X, U), a bank of integrator blocks that propagate the twelve states, and scope blocks used for visualization of the state trajectories.

4.2. Open-Loop Simulation

An open-loop simulation was performed to examine the dynamic characteristics of the quadrotor in the absence of any feedback control law. The twelve integrator outputs were fed back into the MATLAB Function block to close the numerical integration loop, and their outputs were routed to a multi-axis scope for inspection.

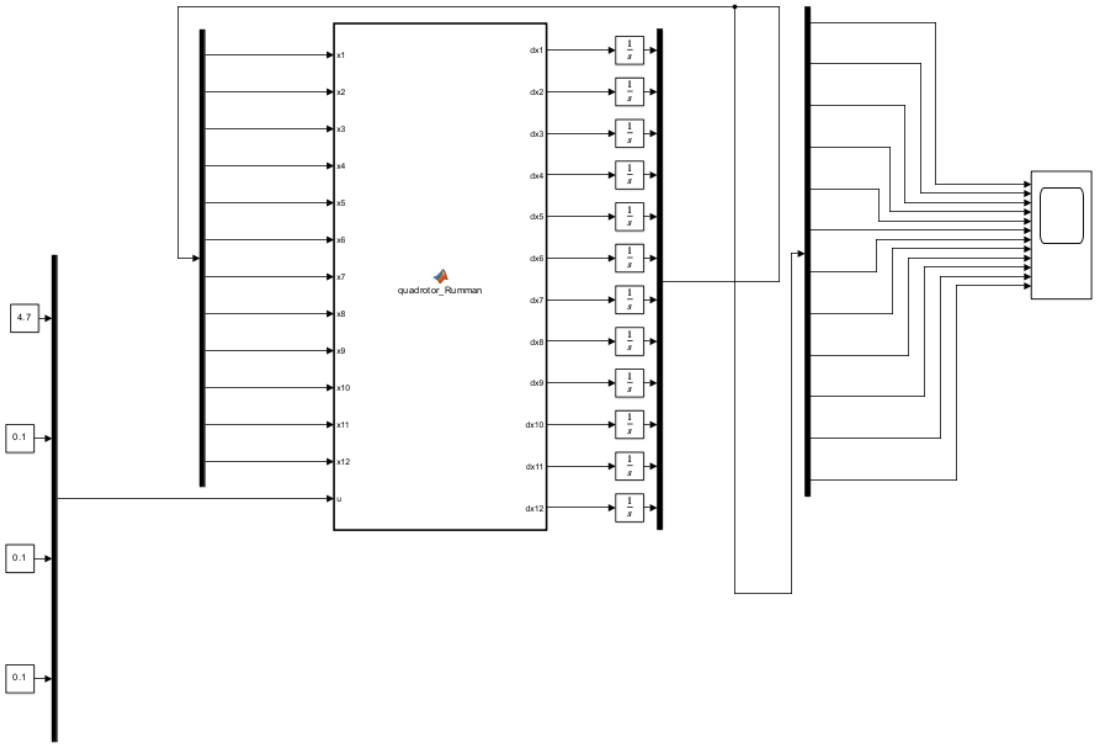

Fig. 3. Open-loop Simulink implementation of the 12-state quadrotor model (MATLAB Function block, integrator bank, and scope).

4.3. MATLAB Function Block

The MATLAB Function block — named quadrotor_Rumman — takes the twelve states x1–x12 (position, velocity, Euler angle, and angular-rate components, respectively) and the input vector u = [U1, U2, U3, U4] and returns the twelve state derivatives dx1–dx12. Internally, the block evaluates the translational dynamics (Xdd, Ydd, Zdd), the rotational dynamics (pdot, qdot, rdot), and the Euler-angle kinematics (phi_dot, theta_dot, psi_dot) using the parameters m, Ix, Iy, Iz, and g listed in Table I, before assigning the corresponding derivative outputs. The complete listing is given below.

Listing 1. Nonlinear quadrotor dynamics — quadrotor_Rumman.m (MATLAB Function block).

```
function [dx1,dx2,dx3,dx4,dx5,dx6,...
    dx7,dx8,dx9,dx10,dx11,dx12] ...
    = quadrotor_Rumman(...
    x1,x2,x3,x4,x5,x6,...
    x7,x8,x9,x10,x11,x12,u)

% STATE DEFINITIONS
% x1  = X position (m)
% x2  = Y position (m)
% x3  = Z position (m)
% x4  = X velocity u (m/s)
% x5  = Y velocity v (m/s)
% x6  = Z velocity w (m/s)
```

```
% x7  = phi   (roll angle)
% x8  = theta (pitch angle)
% x9  = psi   (yaw angle)
% x10 = p (roll rate)
% x11 = q (pitch rate)
% x12 = r (yaw rate)

%% ============================================================
% INPUT DEFINITIONS
%% ============================================================
% u(1) = U1 = Total thrust
% u(2) = U2 = Roll torque
% u(3) = U3 = Pitch torque
% u(4) = U4 = Yaw torque

% PARAMETERS (FROM THESIS)
%% ============================================================
m  = 0.48;   % Mass (kg)
Ix = 5.6e-3; % Inertia about X-axis
Iy = 5.6e-3; % Inertia about Y-axis
Iz = 8.1e-3; % Inertia about Z-axis
g  = 9.81;   % Gravity

% INPUTS
%% ============================================================
U1 = u(1);
U2 = u(2);
U3 = u(3);
U4 = u(4);

%% ============================================================
% TRANSLATIONAL DYNAMICS
%% ============================================================
Xdd = (U1/m)*( cos(x7)*sin(x8)*cos(x9) ...
    + sin(x7)*sin(x9) );
Ydd = (U1/m)*( cos(x7)*sin(x8)*sin(x9) ...
    - sin(x7)*cos(x9) );
Zdd = (U1/m)*( cos(x7)*cos(x8) ) ...
    - g;

%% ============================================================
% ROTATIONAL DYNAMICS
%% ============================================================
pdot = ((Iy-Iz)/Ix)*x11*x12 ...
    + (U2/Ix);
qdot = ((Iz-Ix)/Iy)*x10*x12 ...
    + (U3/Iy);
rdot = ((Ix-Iy)/Iz)*x10*x11 ...
```

```
    + (U4/Iz);

%% ============================================================
% EULER ANGLE KINEMATICS
%% ============================================================
phi_dot   = x10 ...
    + x11*sin(x7)*tan(x8) ...
    + x12*cos(x7)*tan(x8);
theta_dot = x11*cos(x7) ...
    - x12*sin(x7);
psi_dot   = x11*sin(x7)/cos(x8) ...
    + x12*cos(x7)/cos(x8);

%% ============================================================
% STATE DERIVATIVES
%% ============================================================
dx1  = x4;
dx2  = x5;
dx3  = x6;
dx4  = Xdd;
dx5  = Ydd;
dx6  = Zdd;
dx7  = phi_dot;
dx8  = theta_dot;
dx9  = psi_dot;
dx10 = pdot;
dx11 = qdot;
dx12 = rdot;
end
```

4.4. Results

Two open-loop cases were simulated to study the sensitivity of the system to the magnitude of the moment inputs.

Case 1:

Constant inputs $\mathbf{U_1 = 4.7, U_2 = 0.1, U_3 = 0.1, U_4 = 0.1}$ were applied. As shown in Fig. 4, the position states (x, y) grow without bound, the altitude z decreases monotonically, and the roll and pitch rates exhibit sustained, growing oscillations, confirming the inherently unstable and diverging behavior of the uncontrolled quadrotor.

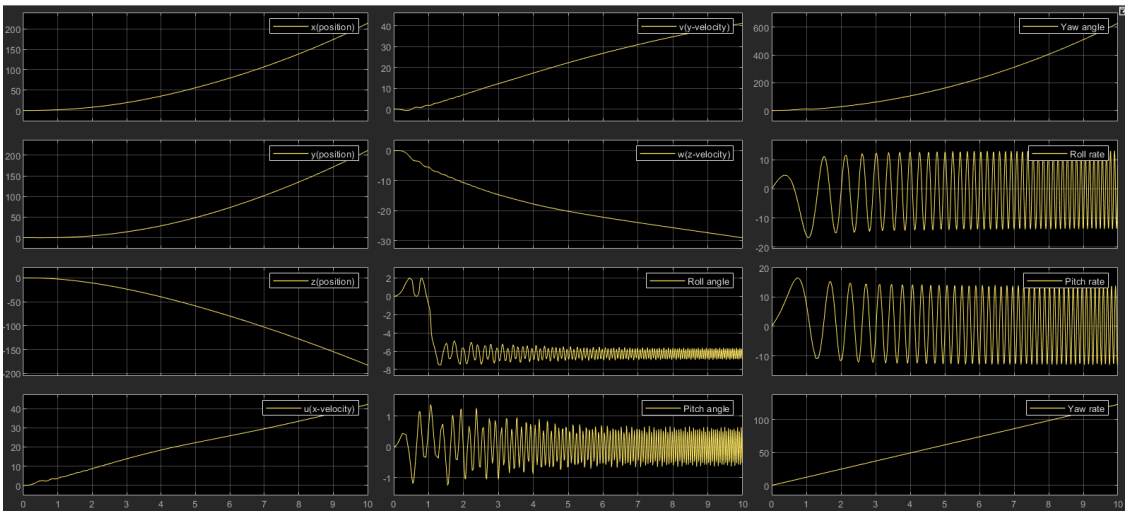

Fig. 4. Open-loop results — Case 1 ($U_1 = 4.7, U_2 = U_3 = U_4 = 0.1$).

Case 2:

Constant inputs $\mathbf{U_1 = 4.7, U_2 = 0.0001, U_3 = 0.0001, U_4 = 0.0001}$ were applied. As shown in Fig. 5, reducing the moment inputs by three orders of magnitude slows the rate of divergence but does not eliminate it — the position states still diverge over a longer time horizon (100 s), further reinforcing that the open-loop quadrotor is unstable for any non-zero moment input and requires feedback stabilization.

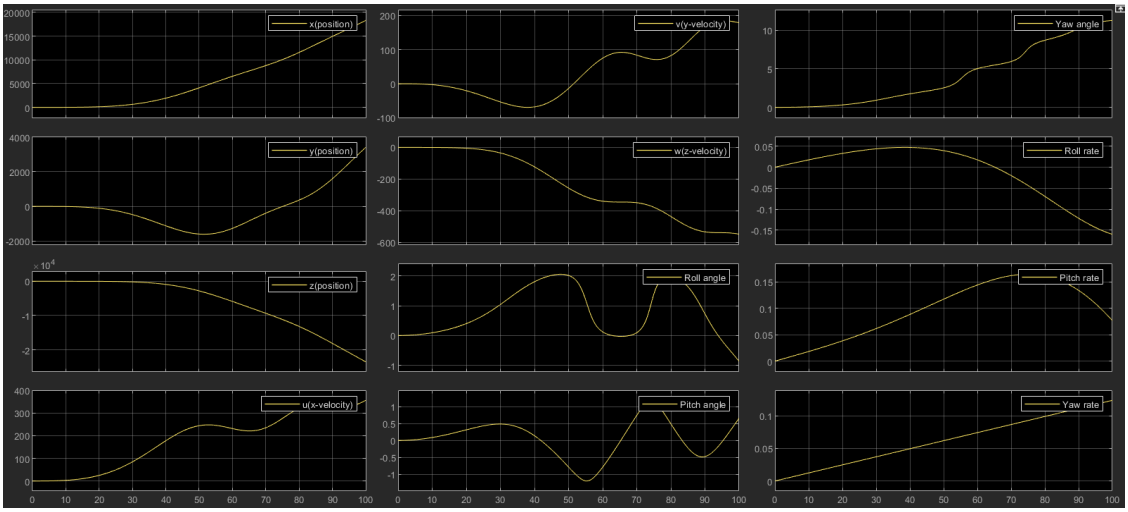

Fig. 5. Open-loop results — Case 2 ($U_1 = 4.7, U_2 = U_3 = U_4 = 0.0001$).

5. Model Linearization and Jacobian Analysis

To support the design of the EKF and to characterize the local dynamics of the system, the Jacobian $A = \partial f/\partial X$ of the nonlinear state equations was derived symbolically and implemented as the function QuadrotorStateJacobianFcnrumman(x, u). Analytical partial derivatives were computed for each block of the state equations — the position, translational-dynamics, Euler-kinematics, and rotational-dynamics rows — in terms of the trigonometric functions of φ, θ, and ψ and the angular rates p, q, r. The complete listing is given below.

Listing 2. State Jacobian — QuadrotorStateJacobianFcnrumman.m.

```
function A = QuadrotorStateJacobianFcnrumman(x,u)
%#codegen
% ========================================================
% JACOBIAN MATRIX OF NONLINEAR QUADROTOR MODEL
% STATES:
% x = [x y z u v w phi theta psi p q r]'
% INPUTS:
% u = [U1 U2 U3 U4]'
%
% OUTPUT:
% A = df/dx
% ========================================================

%% ========================================================
% PARAMETERS (THESIS VALUES)
%% ========================================================
m  = 0.48;
Ix = 5.6e-3;
Iy = 5.6e-3;
Iz = 8.1e-3;
g  = 9.81;

%% ========================================================
% STATES
%% ========================================================
phi   = x(7);
theta = x(8);
psi   = x(9);
p = x(10);
q = x(11);
r = x(12);

%% ========================================================
% INPUTS
%% ========================================================
U1 = u(1);

%% ========================================================
% TRIGONOMETRIC TERMS
%% ========================================================
cphi = cos(phi);
sphi = sin(phi);
cth  = cos(theta);
sth  = sin(theta);
cpsi = cos(psi);
spsi = sin(psi);
tth  = tan(theta);
```

```
secth = 1/cth;
sec2  = secth^2;

%% ======================================================
% INITIALIZE JACOBIAN MATRIX
%% ======================================================
A = zeros(12,12);

%% ======================================================
% POSITION DYNAMICS
%% ======================================================
A(1,4) = 1;
A(2,5) = 1;
A(3,6) = 1;

%% ======================================================
% TRANSLATIONAL DYNAMICS
%% ======================================================
% du/dx
A(4,7) = (U1/m)*( -sphi*sth*cpsi + cphi*spsi );
A(4,8) = (U1/m)*(  cphi*cth*cpsi );
A(4,9) = (U1/m)*( -cphi*sth*spsi + sphi*cpsi );

% dv/dx
A(5,7) = (U1/m)*( -sphi*sth*spsi - cphi*cpsi );
A(5,8) = (U1/m)*(  cphi*cth*spsi );
A(5,9) = (U1/m)*(  cphi*sth*cpsi + sphi*spsi );

% dw/dx
A(6,7) = -(U1/m)*(sphi*cth);
A(6,8) = -(U1/m)*(cphi*sth);

%% ======================================================
% EULER ANGLE KINEMATICS
%% ======================================================
% d(phi_dot)/dx
A(7,7)  = q*cphi*tth - r*sphi*tth;
A(7,8)  = q*sphi*sec2 + r*cphi*sec2;
A(7,10) = 1;
A(7,11) = sphi*tth;
A(7,12) = cphi*tth;

% d(theta_dot)/dx
A(8,7)  = -q*sphi - r*cphi;
A(8,11) = cphi;
A(8,12) = -sphi;

% d(psi_dot)/dx
```

```
A(9,7)  = q*cphi*secth - r*sphi*secth;
A(9,8)  = (q*sphi + r*cphi)*secth*tth;
A(9,11) = sphi*secth;
A(9,12) = cphi*secth;

%% =======================================================
% ROTATIONAL DYNAMICS
%% =======================================================
% dp/dx
A(10,11) = ((Iy-Iz)/Ix)*r;
A(10,12) = ((Iy-Iz)/Ix)*q;

% dq/dx
A(11,10) = ((Iz-Ix)/Iy)*r;
A(11,12) = ((Iz-Ix)/Iy)*p;

% dr/dx
A(12,10) = ((Ix-Iy)/Iz)*q;
A(12,11) = ((Ix-Iy)/Iz)*p;
end
```

A symbolic output-Jacobian function, model_jacobian_quad, was additionally generated using the MATLAB Symbolic Math Toolbox to independently validate the analytical derivation above; its listing is given below.

Listing 3. Symbolically generated output Jacobian — model_jacobian_quad.m.

```
function Jac = model_jacobian_quad(U1,x7,x8,x9,x10,x11,x12)
%MODEL_JACOBIAN_QUAD
%   Jac = MODEL_JACOBIAN_QUAD(U1,X7,X8,X9,X10,X11,X12)
%   This function was generated by the Symbolic Math Toolbox version 24.2.
%   18-May-2026 23:06:27

t2  = cos(x7);
t3  = cos(x8);
t4  = cos(x9);
t5  = sin(x7);
t6  = sin(x8);
t7  = sin(x9);
t8  = tan(x8);
t9  = x12.*8.928571428583609e-4;
t10 = t8.^2;
t11 = 1.0./t3;
t12 = t11.^2;
t13 = t10+1.0;

mt1 = [1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, ...
       0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, ...
```

```
    0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, ...
    0.0,0.0,0.0,2.000000000095e-3,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0, ...
    0.0,0.0,0.0,2.00000000000955e3,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0, ...
    0.0,0.0,0.0,2.00000000000955e3,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0, ...
    U1.*(t2.*t7-t4.*t5.*t6.*1.0).*4.166666666719721e-3, ...
    U1.*(t2.*t4+t5.*t6.*t7).*-4.166666666719721e-3, ...
    U1.*t3.*t5.*-4.166666666719721e3, ...
    t2.*t8.*x11.*2.00000000000955e-3-t5.*t8.*x12.*2.00000000000955e3+1.0, ...
    t2.*x12.*-2.00000000000955e-3-t5.*x11.*2.00000000000955e3, ...
    t2.*t11.*x11.*2.00000000000955e-3-t5.*t11.*x12.*2.00000000000955e3, ...
    0.0,0.0,0.0,0.0,0.0,0.0, ...
    U1.*t2.*t3.*t4.*4.166666666719721e3, ...
    U1.*t2.*t3.*t7.*4.166666666719721e-3, ...
    U1.*t2.*t6.*-4.166666666719721e3, ...
    t2.*t13.*x12.*2.00000000000955e-3+t5.*t13.*x11.*2.00000000000955e3, ...
    1.0, ...
    t2.*t6.*t12.*x12.*2.00000000000955e-3+t5.*t6.*t12.*x11.*2.00000000000955e3, ...
    0.0,0.0,0.0,0.0,0.0,0.0];

mt2 = [U1.*(t4.*t5-t2.*t6.*t7.*1.0).*4.166666666719721e3, ...
    U1.*(t5.*t7+t2.*t4.*t6).*4.166666666719721e3, ...
    0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, ...
    2.00000000000955e3,0.0,0.0,1.0,t9,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, ...
    t5.*t8.*2.00000000000955e3,t2.*2.00000000000955e-3, ...
    t5.*t11.*2.00000000000955e-3,-t9,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, ...
    t2.*t8.*2.00000000000955e-3,t5.*-2.00000000000955e-3, ...
    t2.*t11.*2.00000000000955e-3, ...
    x11.*-8.928571428583609e4,x10.*8.928571428583609e-4,1.0];

Jac = reshape([mt1,mt2],12,12);
End
```

Evaluating the Jacobian at the equilibrium state X = 0 with inputs $U_1 = 4.7, U_2 = U_3 = U_4 = 0.0001$ yields the linearized state matrix shown below. The unit entries on the super-diagonal blocks reflect the direct kinematic coupling between position and velocity, and between Euler angles and angular rates, while the terms A(4,8) = 9.7917 and A(5,7) = −9.7917 (≈ U1/m) capture the linearized coupling between pitch/roll angle and horizontal acceleration about the hover condition. This linear model was subsequently used as the state-transition Jacobian required by the Extended Kalman Filter.

$$A = \partial f / \partial X \quad (\text{evaluated at } X = 0, U_1 = 4.7, U_2 = U_3 = U_4 = 1 \times 10^{-4})$$

6. Nonlinear State Estimation: EKF and UKF

The overall estimator was implemented in Simulink as a closed-loop model in which the plant, sensor models, and Kalman filter blocks are interconnected, as shown in Fig. 6, with detailed zoomed views of the filter wiring shown in Figs. 7–9.

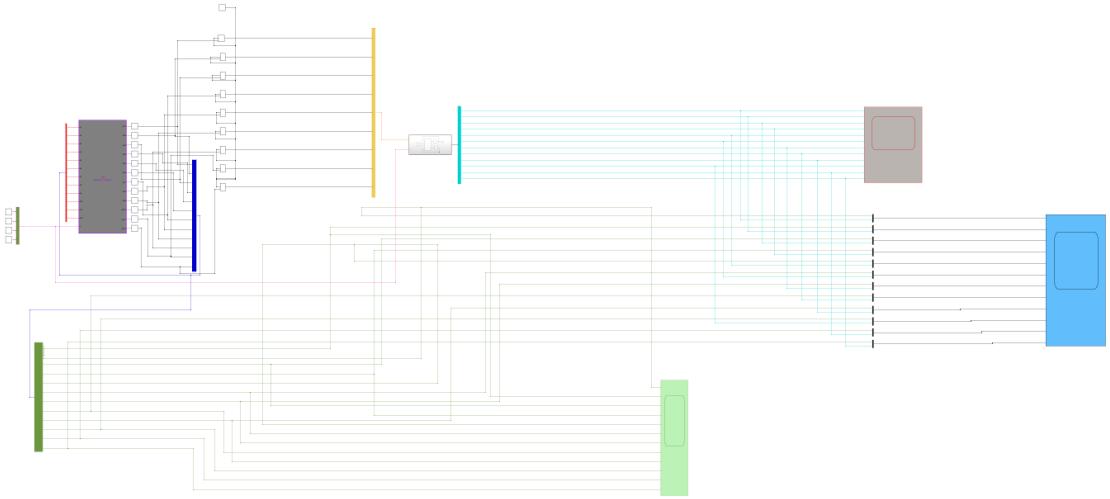

Fig. 6. Simulink model of the quadrotor with an embedded Kalman filter for state estimation.

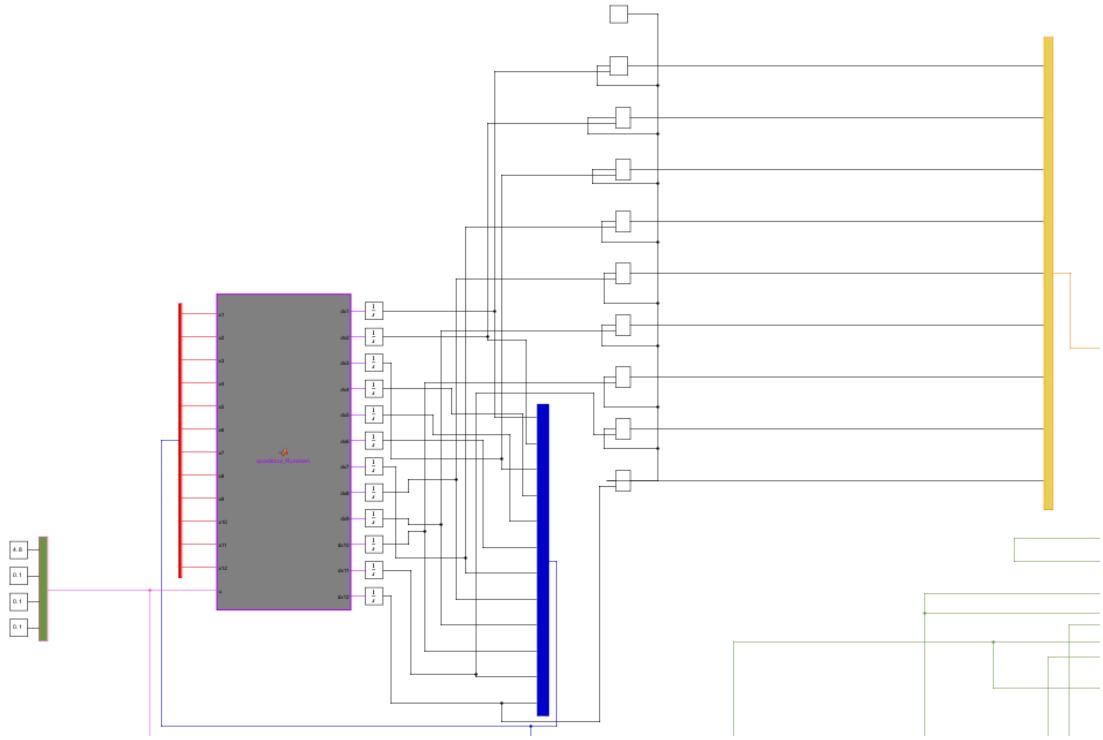

Fig. 7. Zoom view 1 of the Kalman filter subsystem.

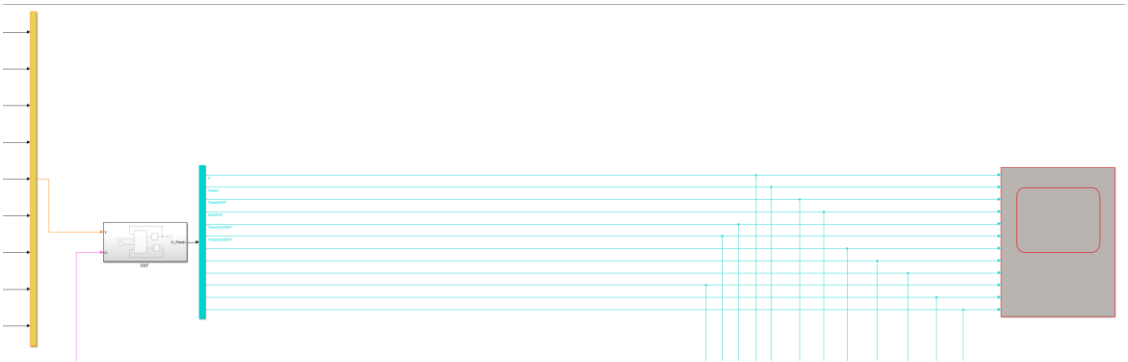

Fig. 8. Zoom view 2 of the Kalman filter subsystem.

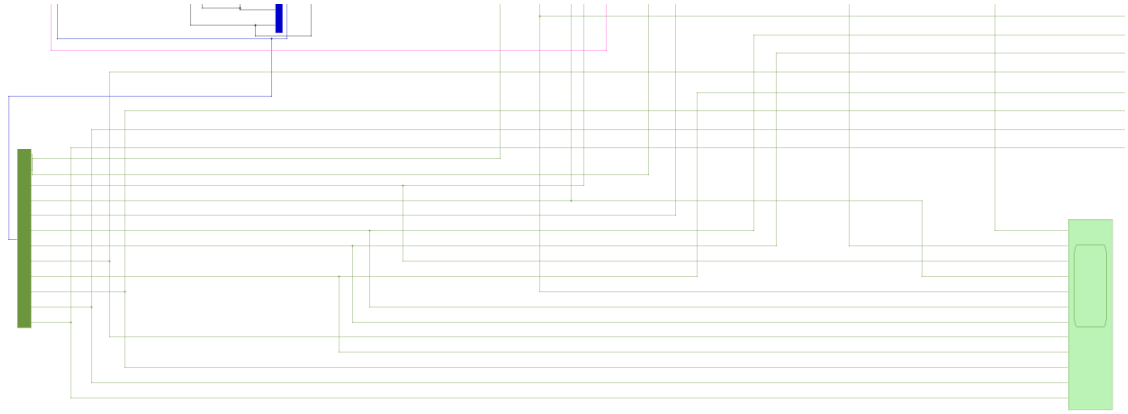

Fig. 9. Zoom view 3 of the Kalman filter subsystem.

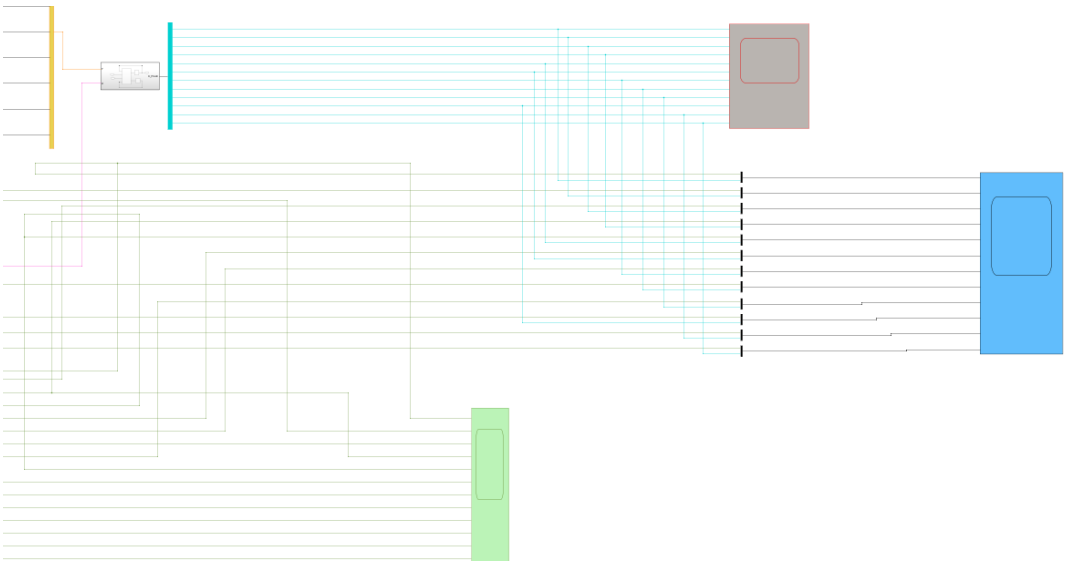

Fig. 9 (cont.). Zoom view 3 of the Kalman filter subsystem — expanded detail.

6.1. Extended Kalman Filter

The EKF linearizes the nonlinear system about the current operating point using the Jacobian matrices derived in Section VII (Goodarzi & Lee, 2017; Ghanai et al., 2018). The filter was implemented in Simulink using MATLAB's extended Kalman Filter object, configured with the nonlinear state-transition function QuadrotorStateFcn, the measurement function QuadrotorMeasurementFcn, and their respective Jacobians, as shown in the EKF function block of Fig. 10.

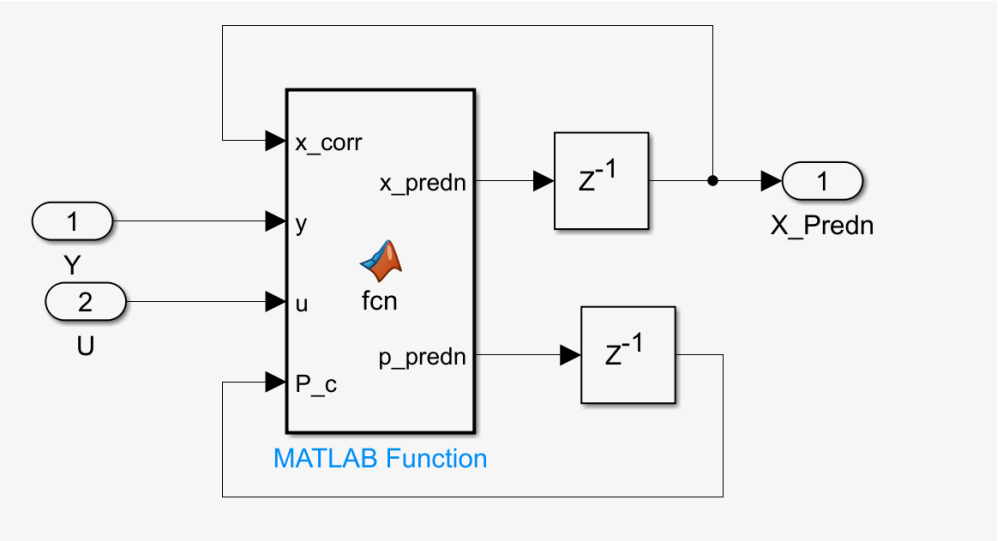

Fig. 10. EKF MATLAB Function block — inputs Y (measurement), U (control input), and P_c (covariance); outputs the predicted state X_Predn.

The measurement vector was defined as $y = [x, y, z, \varphi, \theta, \psi, p, q, r]^T$. Process noise and measurement noise covariances were set to $Qv = 1\times10^{-6}\times I_{12}$ and $Rv = 1\times10^{-4}\times I_9$, respectively. Each estimation cycle consists of a prediction step, in which the filter propagates the state and covariance forward using the control input u, followed by a correction step, in which the predicted state is updated using the noisy measurement y. The complete EKF listing is given below.

Listing 4. Extended Kalman Filter — EKF_Quadrotor.m.

```
function [x_est,P_est] = EKF_Quadrotor(x_prev,y,u,P_prev)
%#codegen
%% ============================================================
% EXTENDED KALMAN FILTER FOR NONLINEAR QUADROTOR
% STATES:
% x = [x y z u v w phi theta psi p q r]'
% INPUTS:
% u = [U1 U2 U3 U4]'
% MEASUREMENTS:
% y = [x y z phi theta psi p q r]'
%% ============================================================

%% ============================================================
% PROCESS NOISE COVARIANCE MATRIX
%% ============================================================
Qv = 1e-6*eye(12);

%% ============================================================
% MEASUREMENT NOISE COVARIANCE MATRIX
%% ============================================================
Rv = 1e-4*eye(9);

%% ============================================================
```

```
% CREATE EKF OBJECT
%% ============================================================
ekf = extendedKalmanFilter( ...
    @QuadrotorStateFcn, ...
    @QuadrotorMeasurementFcn, ...
    x_prev, ...
    'StateCovariance', P_prev, ...
    'ProcessNoise', Qv, ...
    'MeasurementNoise', Rv, ...
    'StateTransitionJacobianFcn', ...
    @QuadrotorStateJacobianFcn, ...
    'MeasurementJacobianFcn', ...
    @QuadrotorMeasurementJacobianFcn);

%% ============================================================
% PREDICTION STEP
%% ============================================================
predict(ekf,u);

%% ============================================================
% CORRECTION STEP
%% ============================================================
correct(ekf,y,u);

x_est = ekf.State;
P_est = ekf.StateCovariance;
End
```

With inputs $U_1 = 4.7\ and\ U_2 = U_3 = U_4 = 0.0001$, the EKF produced smooth, converging estimates of position, velocity, and attitude that closely track the underlying nonlinear trajectory, though with small residual errors attributable to the linearization inherent in the EKF (Figs. 11–12).

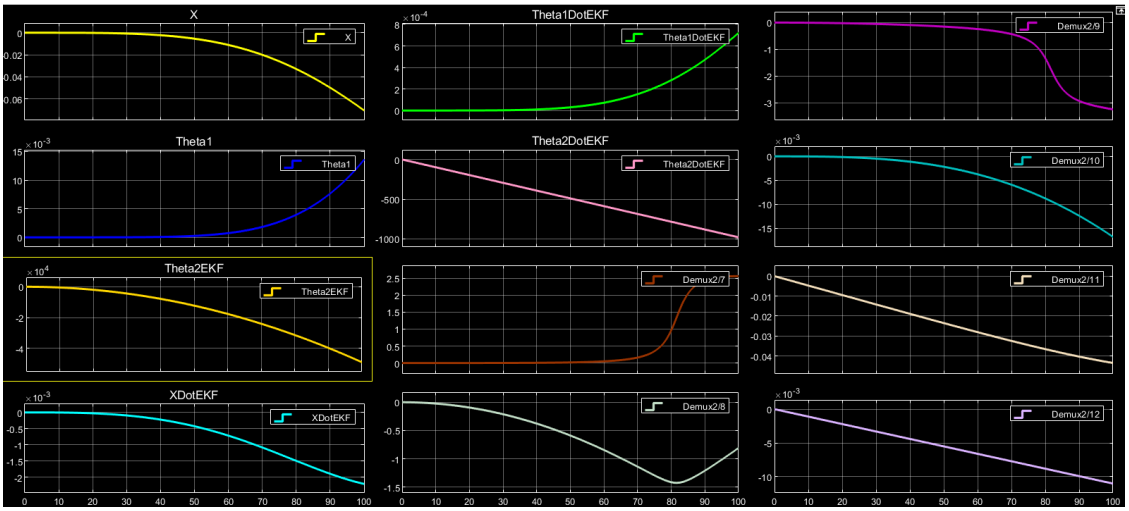

Fig. 11. EKF state-estimation results (position, Euler angles, and their rates) for $U_1 = 4.7, U_2 = U_3 = U_4 = 0.0001$.

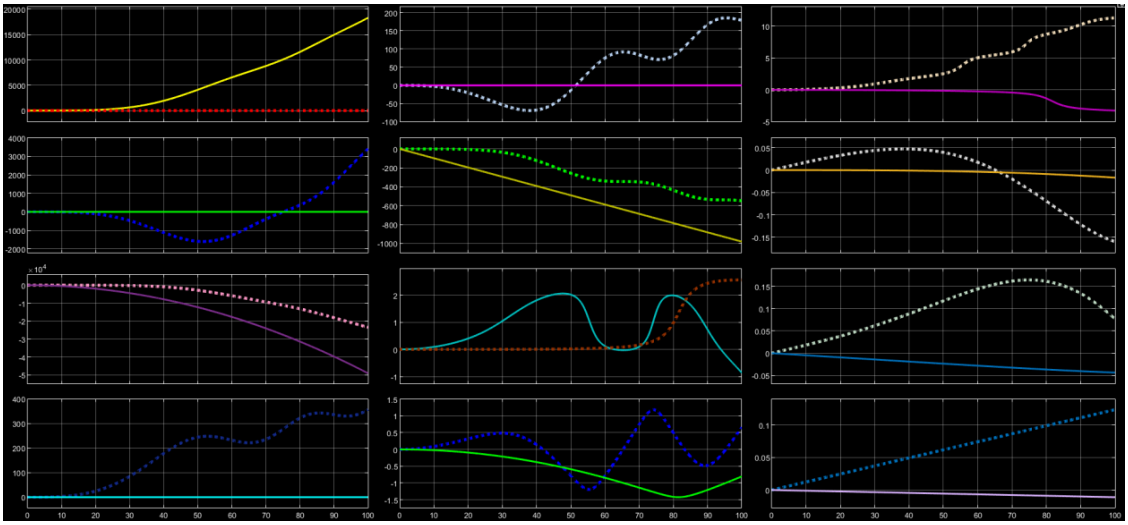

Fig. 12. EKF results, Scope 2 — extended view of position, velocity, and attitude estimates.

6.2. Unscented Kalman Filter

Unlike the EKF, the UKF avoids explicit linearization by propagating a deterministic set of sigma points through the true nonlinear state and measurement functions to approximate the mean and covariance of the transformed distribution (Julier & Uhlmann, 2004; Wan & van der Merwe, 2000) as illustrated by the UKF function block of Fig. 13.

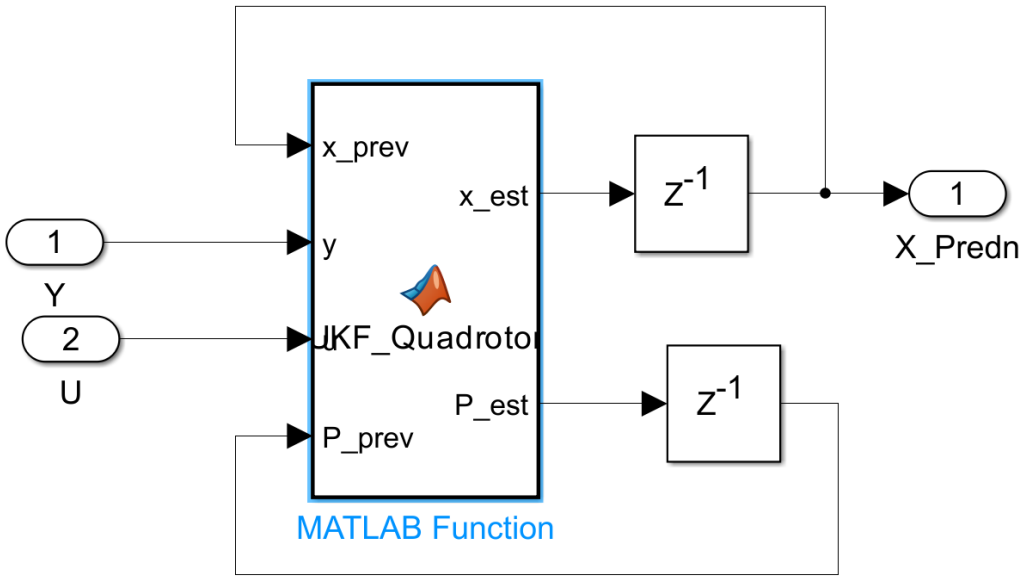

Fig. 13. UKF MATLAB Function block — inputs x_prev, Y, U, and P_prev; outputs the predicted state X_Predn.

The UKF was implemented using MATLAB's unscentedKalmanFilter object with the same nonlinear state and measurement functions, process noise Qv, and measurement noise Rv as the EKF, so that the two filters could be compared directly under identical conditions. Its principal advantages are improved nonlinear estimation accuracy and the elimination of the need to derive or evaluate a Jacobian. The complete UKF listing is given below.

Listing 5. Unscented Kalman Filter — UKF_Quadrotor.m.

```
function [x_est,P_est] = UKF_Quadrotor(x_prev,y,u,P_prev)
%#codegen
%% ========================================================
% UNSCENTED KALMAN FILTER FOR NONLINEAR QUADROTOR
% STATES:
% x = [x y z u v w phi theta psi p q r]'
% INPUTS:
% u = [U1 U2 U3 U4]'
% MEASUREMENTS:
% y = [x y z phi theta psi p q r]'
%% ========================================================

%% ========================================================
% PROCESS NOISE COVARIANCE MATRIX
%% ========================================================
Qv = 1e-6*eye(12);

%% ========================================================
% MEASUREMENT NOISE COVARIANCE MATRIX
%% ========================================================
Rv = 1e-4*eye(9);

%% ========================================================
% CREATE UKF OBJECT
%% ========================================================
ukf = unscentedKalmanFilter( ...
   @QuadrotorStateFcn, ...
   @QuadrotorMeasurementFcn, ...
   x_prev);

%% ========================================================
% INITIALIZE COVARIANCES
%% ========================================================
ukf.StateCovariance = P_prev;
ukf.ProcessNoise    = Qv;
ukf.MeasurementNoise = Rv;

%% ========================================================
% PREDICTION STEP
%% ========================================================
predict(ukf,u);

%% ========================================================
% CORRECTION STEP
%% ========================================================
correct(ukf,y,u);
```

```
%% ========================================================
% OUTPUTS
%% ========================================================
x_est = ukf.State;
P_est = ukf.StateCovariance;
End
```

Under the same input conditions ($U_1 = 4.7, U_2 = U_3 = U_4 = 0.0001$), the UKF produced estimates that tracked the reference states more closely than the EKF, with visibly smaller deviation in the transient region (Figs. 12–13).

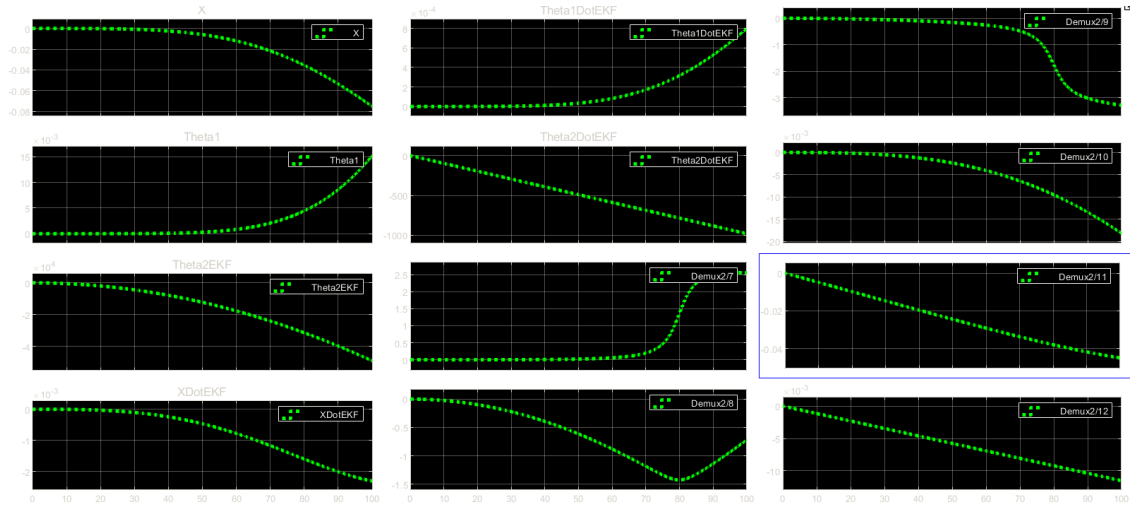

Fig. 12. UKF state-estimation results 1 — position, Euler angles, and their rates.

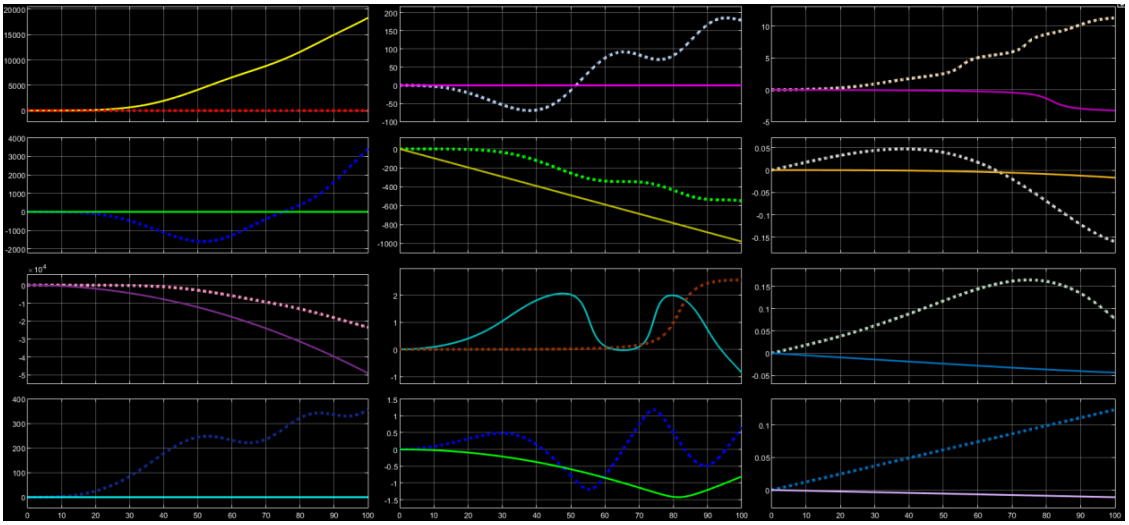

Fig. 13. UKF state-estimation results 2 — extended view of position, velocity, and attitude estimates.

7. Sensor Fusion

Sensor fusion involves the integration of data collected from multiple, independent sensors to increase the precision of state estimation in the quadrotor (Kaba, 2021). Its primary objective is minimizing the effect of measurement noise and individual sensor errors; both the EKF and

the UKF employ sensor fusion in estimating the quadrotor's states, which leads to smoother and more consistent results and enhances the precision of the estimated position and attitude.

7.1. Simulink Model

A dedicated Simulink model incorporating sensor fusion was constructed, as shown in Fig. 14, and simulated with constant inputs $U_1 = 4.7, U_2 = 0.1, U_3 = 0.1, U_4 = 0.1$.

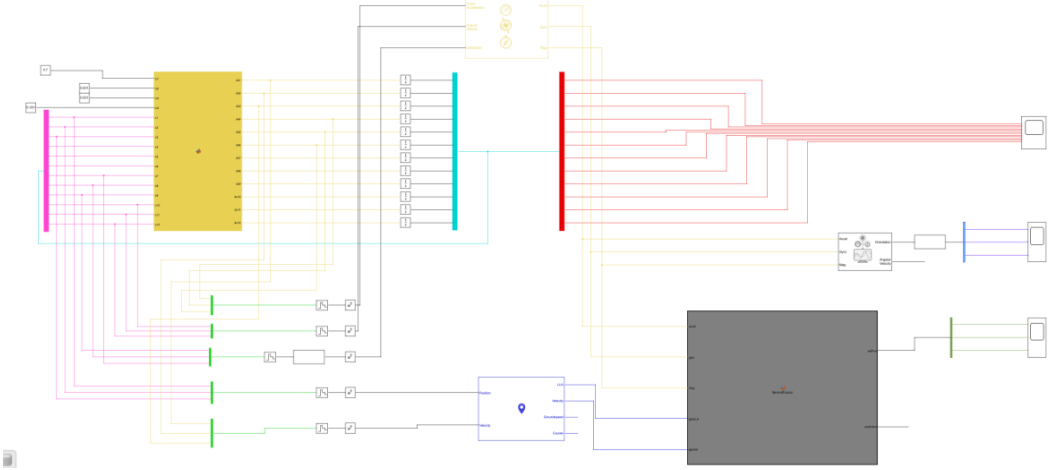

Fig. 14. Simulink model incorporating Kalman-filter-based sensor fusion.

7.2. Results

The results (Figs. 15–16) show that, in the absence of sensor fusion, the raw attitude signals (φ, θ, ψ) are highly oscillatory and noisy, whereas the fused position estimates (x, y, z) are considerably smoother and closer to the underlying trajectory, consistent with the expected noise-reduction benefit of fusing multiple sensor sources.

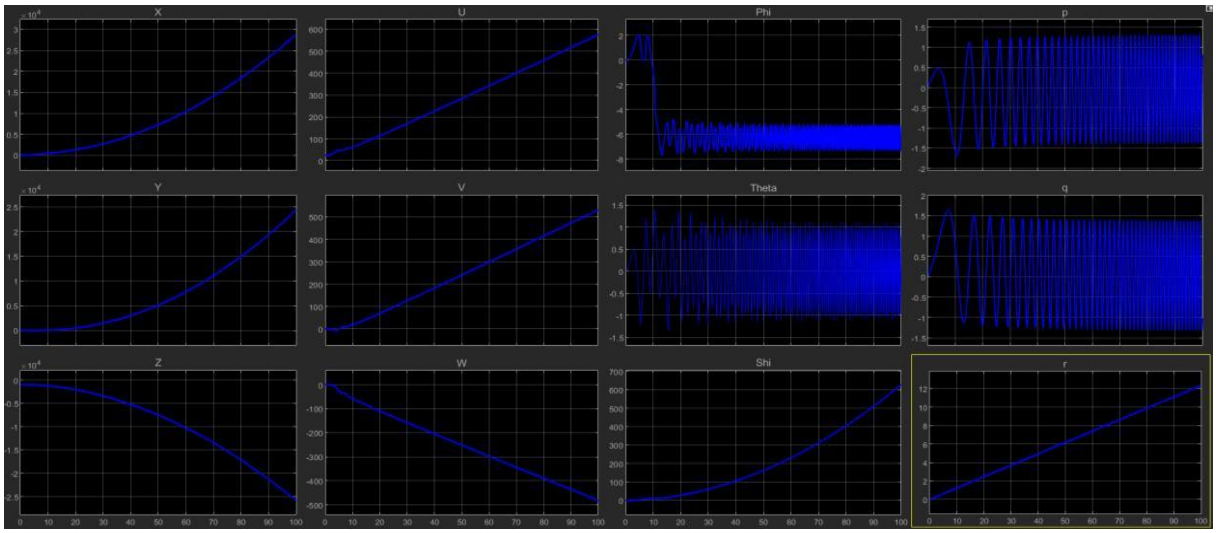

Fig. 15. Results 1 with sensor fusion — fused position, velocity, and attitude estimates.

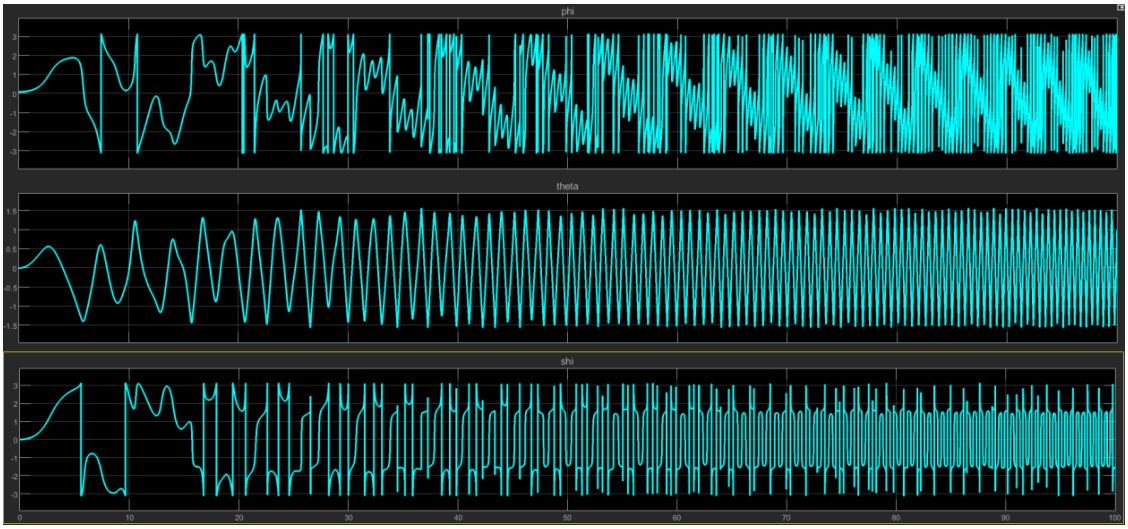

Fig. 15 (cont.). Results 2 with sensor fusion — unfused attitude signals (φ, θ, ψ) showing pronounced noise.

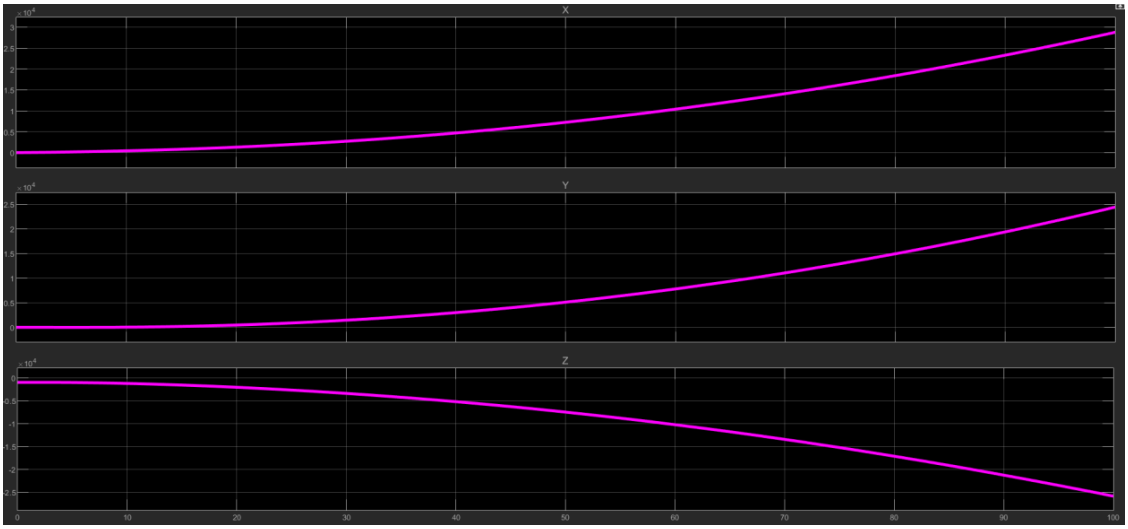

Fig. 16. Results 3 with sensor fusion — fused x, y, z position estimates.

8. Results and Discussion

8.1. Basic (Open-Loop) Model

The basic open-loop model gives very smooth and simple results but is inaccurate, as it does not take into account practical problems such as measurement noise. It is therefore a useful tool for understanding the dynamics of the system, but it cannot be relied upon entirely in practice.

8.2. EKF and UKF Implementation

Implementing the system through the EKF/UKF framework gives better results, since both filters account for process and measurement uncertainty. The downside is that this considerably increases the complexity of the overall system. The EKF estimates the quadrotor states based on noisy measurements with good overall accuracy; however, small errors are introduced by the linearization step inherent to the EKF. The UKF produces more accurate results than the EKF, offering better tracking of the quadrotor states and smaller estimation errors, since it is

inherently better suited to nonlinear systems. Both algorithms estimate the quadrotor states effectively, but the UKF provides superior estimation accuracy at the cost of higher computational demand, while the EKF remains easier and computationally cheaper to implement.

8.3. Sensor Fusion

Kalman-filter-based sensor fusion was found to improve the position estimation of the quadrotor. By fusing data from multiple sensors, the resulting position estimates were smoother and more accurate than those obtained from individual, unfused sensor signals.

8.4. Summary

Overall, it can be concluded that the basic open-loop model is easier to use but less reliable, whereas the EKF/UKF-based estimation model is more accurate but harder to implement. Between the two filters, the UKF is the more accurate but more computationally demanding option, while the EKF offers a reasonable trade-off between accuracy and implementation simplicity.

9. Conclusion

This paper successfully developed a nonlinear 12-state quadrotor model in the X-configuration using MATLAB/Simulink, based on the Newton–Euler formulation. The open-loop simulation demonstrated the inherently unstable nature of quadrotor systems in the absence of feedback control. The paper further studied the EKF and UKF methods for nonlinear state estimation, and examined the benefit of sensor fusion in improving the accuracy of position and attitude estimates under measurement noise. The developed model and estimation framework can be extended in future work to incorporate closed-loop controller design, autonomous navigation, and more advanced sensor-fusion architectures.

Declaration of conflict of interest:

The author(s) declared no potential conflicts of interest(s) with respect to the research, authorship, and/or publication of this article.

Funding:

The author(s) received no financial support for the research, authorship and/or publication of this article.

Publisher's Note:

IDEA PUBLISHERS (NASIJ) stands neutral with regard to jurisdictional claims in the published maps and institutional affiliations.

References

Bouabdallah, S. (2007). *Design and control of quadrotors with application to autonomous flying* [Doctoral dissertation, École Polytechnique Fédérale de Lausanne]. Infoscience. https://doi.org/10.5075/epfl-thesis-3727

Ghanai, M., Medjghou, A., & Chafaa, K. (2018). Extended Kalman filter based states estimation of unmanned quadrotors for altitude-attitude tracking control. *Advances in Electrical and Electronic Engineering, 16*(4), 446–458. https://doi.org/10.15598/aeee.v16i4.2911

Goodarzi, F. A., & Lee, T. (2017). Global formulation of an extended Kalman filter on SE(3) for geometric control of a quadrotor UAV. *Journal of Intelligent & Robotic Systems, 88*, 395–413.

Julier, S. J., & Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE, 92*(3), 401–422. https://doi.org/10.1109/JPROC.2003.823141

Kaba, A. (2021). Unscented Kalman filter based attitude estimation of a quadrotor. *Journal of Aeronautics and Space Technologies, 14*(1), 79–88. https://jast.msu.edu.tr/index.php/JAST/article/view/442

Khalilov, J. (2016). *Interfacing SIMULINK/MATLAB with V-REP for analysis and control synthesis of a quadrotor* [Master's thesis, Middle East Technical University]. METU Open Research Repository. https://hdl.handle.net/11511/25668

Mahony, R., Kumar, V., & Corke, P. (2012). Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics & Automation Magazine, 19*(3), 20–32. https://doi.org/10.1109/MRA.2012.2206474

Şenelt, E. (2010). *Design and manufacturing of a tactical unmanned air vehicle* [Master's thesis, Middle East Technical University]. METU Open Research Repository. https://hdl.handle.net/11511/19938

Wan, E. A., & van der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (AS-SPCC)* (pp. 153–158). IEEE. https://doi.org/10.1109/ASSPCC.2000.882463

Appendix:

Nomenclature

Symbol	Description
x, y, z	Position coordinates in the inertial frame (m)
u, v, w	Linear velocity components in the body frame (m/s)
p, q, r	Angular rates about the body x-, y-, z-axes (rad/s)
φ	Roll angle (rad)
θ	Pitch angle (rad)
ψ	Yaw angle (rad)
m	Mass of the quadrotor (kg)
g	Gravitational acceleration (m/s²)
I_x, I_y, I_z	Moments of inertia about the body x-, y-, z-axes (kg·m²)
$T_1 - T_4$	Individual rotor thrust forces (N)
$Q_1 - Q_4$	Individual rotor reaction (drag) torques (N·m)
U_1	Total thrust control input (N)
U_2	Roll moment control input (N·m)
U_3	Pitch moment control input (N·m)
U_4	Yaw moment control input (N·m)
X	State vector, X = [x, y, z, u, v, w, φ, θ, ψ, p, q, r]^T
U	Input vector, U = [U1, U2, U3, U4]^T
$f(X, U)$	Nonlinear state-transition function
A	State Jacobian matrix, A = ∂f/∂X
y	Measurement vector, y = [x, y, z, φ, θ, ψ, p, q, r]^T
P	State error covariance matrix
Qv	Process noise covariance matrix
Rv	Measurement noise covariance matrix