



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده هواشناسی

ناوبری تلفیقی

پروژه نهایی : ناوبری با استفاده از GPS
کامل فیلتر مستقیم و غیر مستقیم

نگارش
امیرحسین آقامحمدی
403129911

استاد درس
دکتر محمد کریمی

1. سینماتیک سیستم

در گام نخست باید معادلات سینماتیکی حاکم بر مسئله را استخراج کنیم. معادلات سینماتیک شامل حل معادلات در دستگاه ناوبری است که این معادلات به شکل زیر استخراج میشوند.

✓ معادلات بروزرسانی موقعیت

$$\begin{aligned} [S_{BE}]^N &= \begin{bmatrix} 0 \\ 0 \\ -(R_e + h) \end{bmatrix} \\ \rightarrow [D^N S_{BE}]^N &= \left[\frac{d}{dt} S_{BE} \right]^N = \begin{bmatrix} 0 \\ 0 \\ -\dot{h} \end{bmatrix} \\ [v_B^E]^N &= [D^N S_{BE}]^N + [\Omega^{NE}]^N [S_{BE}]^N \\ \rightarrow \begin{bmatrix} v_n \\ v_e \\ v_d \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \\ -\dot{h} \end{bmatrix} + \begin{bmatrix} 0 & \dot{l} \sin \lambda & -\dot{\lambda} \\ -\dot{l} \sin \lambda & 0 & -\dot{l} \cos \lambda \\ -\dot{\lambda} & \dot{l} \cos \lambda & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -(R_e + h) \end{bmatrix} \\ \rightarrow \begin{cases} \dot{\lambda} = \frac{v_n}{R_e + h} \\ \dot{l} = \frac{v_e}{(R_e + h) \cos \lambda} \\ \dot{h} = -v_d \end{cases} &\xrightarrow{\text{effect of } R_n, R_m} \begin{cases} \dot{\lambda} = \frac{v_n}{R_n + h} \\ \dot{l} = \frac{v_e}{(R_m + h) \cos \lambda} \\ \dot{h} = -v_d \end{cases} \end{aligned}$$

که در معادلات فوق $[\Omega^{NE}]^N$ از ترتیب چرخش به صورت زیر بدست آمده.

$$\begin{aligned} E \left[\begin{array}{c} l \\ \xrightarrow[3]{} X \end{array} \right] &\xrightarrow{270-\lambda} N \left[\begin{array}{c} 270-\lambda \\ \xrightarrow[2]{} \end{array} \right] \\ \rightarrow \omega^{NE} = -\dot{\lambda} 2^N + \dot{l} 3^X &\rightarrow [\omega^{NE}]^N = \begin{bmatrix} \dot{l} \cos \lambda \\ -\dot{\lambda} \\ -\dot{l} \sin \lambda \end{bmatrix} \end{aligned}$$

✓ معادلات بروزرسانی سرعت

برای بروزرسانی سرعت ابتدا از قانون نیوتون در دستگاه اینرسی شروع کرده و آن را در دستگاه ناوبری بازنویسی میکنیم.

$$\begin{aligned}
 v_B^I &= D^I s_{BI} = D^E s_{BI} + \Omega^{EI} s_{BI} = v_B^E + \Omega^{EI} s_{BE} \\
 D^I v_B^I &= f_B^I + G \\
 \rightarrow D^I v_B^I &= D^E v_B^I + \Omega^{EI} v_B^I \\
 &= D^E v_B^E + D^E \{\Omega^{EI} s_{BE}\} + \Omega^{EI} v_B^E + \Omega^{EI} \Omega^{EI} s_{BE} \\
 &= D^E v_B^E + 2\Omega^{EI} v_B^E + \Omega^{EI} \Omega^{EI} s_{BE} = f_B^I + G \\
 \rightarrow D^E v_B^E + 2\Omega^{EI} v_B^E &= f_B^I + G - \Omega^{EI} \Omega^{EI} s_{BE} = f_B^I + g
 \end{aligned}$$

حال معادلات فوق را در دستگاه ناوبری بیان میکنیم.

$$\begin{aligned}
 D^E v_B^E &= D^N v_B^E + \Omega^{NE} v_B^E \\
 D^N v_B^E &= f_B^I + g - 2\Omega^{EI} v_B^E - \Omega^{NE} v_B^E \\
 [D^N v_B^E]^N &= [T]^{NB} [f_B^I]^B + [g]^N - 2[T]^{NE} [\Omega^{EI}]^E [\bar{T}]^{NE} [v_B^E]^N - [\Omega^{NE}]^N [v_B^E]^N \\
 [D^N v_B^E]^N &= \begin{bmatrix} \dot{v}_n \\ \dot{v}_e \\ \dot{v}_d \end{bmatrix}
 \end{aligned}$$

در حل معادلات فوق مقادیر زیر را جاگذاری میکنیم.

$$\begin{aligned}
 [\omega^{EI}]^E &= \begin{bmatrix} 0 \\ 0 \\ \omega_{ei} \end{bmatrix} \\
 [\omega^{NE}]^N &= \begin{bmatrix} l \cos \lambda \\ -\dot{\lambda} \\ -l \sin \lambda \end{bmatrix} = \begin{bmatrix} \frac{v_e}{R_n + h} \\ -\frac{v_n}{R_m + h} \\ -\frac{v_e}{R_n + h} \tan \lambda \end{bmatrix}
 \end{aligned}$$

$[T]^{NB} \rightarrow$ from attitude update

$$[T]^{NE} = \begin{bmatrix} -\sin \lambda \cos l & -\sin \lambda \sin l & \cos \lambda \\ -\sin l & \cos l & 0 \\ -\cos \lambda \cos l & -\cos \lambda \sin l & -\sin \lambda \end{bmatrix}$$

در معادلات قبل از روابط ارائه زیر نیز استفاده میگردد.

$$[f_B^I]^B = [\tilde{f}_B^I]^B - [b_{accel}]^B$$

$$[g]^N = \begin{bmatrix} -8.08 \times 10^{-9} h \sin(2\lambda) \\ 0 \\ pg_0 \end{bmatrix}$$

$$p = 1 - \frac{2qh}{R_0} + 3 \frac{h^2}{R_0^2}$$

$$q = 1 + \frac{\omega_{ei}^2 R_0^2 R_p}{\mu} + (1 - 2 \sin^2 \lambda)$$

$$g_0 = \frac{9.7803253359 [1 + 0.001931853 \sin^2 \lambda]}{\sqrt{1 - e^2 \sin^2 \lambda}}$$

$$R_0 = 63781337$$

$$R_p = 6356752.31425$$

$$\omega_{ei} = \frac{2\pi}{24 \times 3600}$$

$$e = 0.0818191908425$$

$$f = \frac{1}{298.257223563}$$

$$\mu = 3.986004418 \times 10^{14}$$

$$R_n = \frac{R_0}{\sqrt{1 - e^2 \sin^2 \lambda}}$$

$$R_m = \frac{R_0 (1 - e^2)}{(1 - e^2 \sin^2 \lambda)^{3/2}}$$

✓ معادلات بروزرسانی وضعیت

در این قسمت هدف محاسبه نرخ تغییرات زوایای اویلر میباشد و پس از آن محاسبه $[T]^{NB}$ برای استفاده در معادلات قسمت قبل. در اینجا دو روش بیان میوشد و در شبیه سازی از روش ساده تر که بروزرسانی مستقیم زوایای اویلر است محاسبه استفاده خواهد شد.

روش اول شامل آپدیت مستقیم ماتریس DCM است.

$$\begin{aligned} \left[\frac{dT}{dt} \right]^{BN} &= -\left[\Omega^{BN} \right]^B [T]^{BN} \\ \left[\omega^{BN} \right]^B &= \left[\omega^{BI} \right]^B - [T]^{BN} \left[\omega^{NE} \right]^N - [T]^{BN} [T]^{NE} \left[\omega^{EI} \right]^E \\ \left[\omega^{BI} \right]^B &= \left[\tilde{\omega}^{BI} \right]^B - \left[b_g \right]^B \end{aligned}$$

روش دوم بروزرسانی مستقیم زوایای اویلر است.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \frac{\sin \varphi \sin \theta}{\cos \theta} & \frac{\cos \varphi \sin \theta}{\cos \theta} \\ 0 & \cos \varphi & -\sin \varphi \\ 0 & \frac{\sin \varphi}{\cos \theta} & \frac{\cos \varphi}{\cos \theta} \end{bmatrix} \left[\omega^{BN} \right]^B$$

$$-\pi < \varphi < \pi$$

$$-\pi < \psi < \pi$$

$$-\frac{\pi}{2} < \theta < \frac{\pi}{2}$$

در این روش باید به نقاط تکینگی دقت شود. همچنین محدوده زوایا نیز باید همواره در محدوده منطقی باقی بمانند.

✓ معادلات بروزرسانی بایاس

در اینجا فرض میشود که مقادیر بایاس

✓ شرایط اولیه معادلات

برای حل معادلات نیاز به شرایط اولیه داریم که بنابر صورت مسئله آن را به صورت زیر در نظر میگیریم.

$$l=0.900269181102793$$

$$\lambda=0.591553645549858$$

$$h=1.078969566106361e+03$$

$$v_n = 0$$

$$v_e = 0$$

$$v_d = 0$$

$$\varphi = 0$$

$$\theta = 0$$

$$\psi = 0$$

$$b_{accel} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$b_{gyro} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$x = \begin{bmatrix} l \\ \lambda \\ h \\ v_n \\ v_e \\ v_d \\ \varphi \\ \theta \\ \psi \\ b_{accel} \\ b_{gyro} \end{bmatrix}$$

در معادلات \mathbf{X} بردار حالت است که مجموعاً شامل 15 حالت از سیستم میباشد.

برای معادلات سینماتیکیتابع state_derivative در مطلب پیاده سازی شده است. در این تابع ورودی شامل بردار حالت و سنسور imu میباشد و پیاده سازی آن به شکل زیر بود است.

```
function state_derivative = state_derivative(x_n, Wbi_b, Fbi_b)
    lon = x_n(1,1);
    lat = x_n(2,1);
    h = x_n(3,1);
    vn = x_n(4,1);
    ve = x_n(5,1);
    vd = x_n(6,1);
    Vbe_n = [vn;
              ve;
              vd];
    phi = x_n(7,1);
    theta = x_n(8,1);
    si = x_n(9,1);
    b_accel = x_n(10:12,1);
    b_gyro = x_n(13:15,1);

    R0 = 6378137;
    Rp = 6356752.31425;
    wei = 2 * pi / (24 * 3600);
    e = 0.0818191908425;
    f = 1 / (298.257223563);
    mu = 3.986004418 * 1e14;
    Rn = R0 / sqrt( 1 - e^2 * sin(lat)^2);
    Rm = R0 * ( 1 - e^2) / ( 1 - e^2 * sin(lat)^2)^(3/2);
    g0 = 9.7803253359 * (1 + 0.001931853 * sin(lat)^2) / sqrt( 1 - e^2 * sin(lat)^2);
    q = 1 + (wei^2 * R0^2 * Rp / mu) + (1 - 2 * sin(lat)^2) * f;
    p = 1 - (2 * q * h / R0) + (3 * h^2 / R0^2);
    g = [-8.08 * 1e-9 * h * sin(2 * lat);
          0;
          p * g0];

    Wei_e = [0;
              0;
              wei];
    Wne_n = [ve / (Rn + h);
              -vn / (Rm + h);
              -ve / (Rn + h) * tan(lat)];
    T_ne = [-sin(lat) * cos(lon), -sin(lat) * sin(lon), cos(lat);
              -sin(lon), cos(lon), 0;
              -cos(lat) * cos(lon), -cos(lat) * sin(lon), -sin(lat)];
    T_bn = [cos(si) * cos(theta), sin(si) * cos(theta),
              -sin(theta);
              cos(si) * sin(theta) * sin(phi) - sin(si) * cos(phi), sin(si) * sin(theta) * sin(phi) + cos(si) * cos(phi), cos(theta) * sin(phi);
              cos(si) * sin(theta) * cos(phi) + sin(si) * sin(phi), sin(si) * sin(theta) * cos(phi) - cos(si) * sin(phi), cos(theta) * cos(phi)];
    Wbn_b = Wbi_b - T_bn * Wne_n - T_bn * T_ne * Wei_e;
```

```

Dn_Vbe_n = T_bn' * (Fbi_b - b_accel) + g - 2 * T_ne * skew_symmetric(Wei_e) *
T_ne' * Vbe_n - skew_symmetric(Wne_n) * Vbe_n;
state_derivative = [ve / ((Rn + h) * cos(lat));
vn / (Rm + h);
-vd;
Dn_Vbe_n;
[1, sin(phi) * tan(theta), cos(phi) * tan(theta);
0, cos(phi), -sin(phi);
0, sin(phi) / cos(theta), cos(phi) / cos(theta)] *
(Wbn_b - b_gyro)
zeros(6, 1)];

```

end

2. حل معادلات RK4

پس از نوشتتن معادلات به فرم $\dot{x} = g(x, u)$ میبایست این معادلات توسط روش های عددی حل شوند.

برای حل انتگرال گیری از روش RK4 استفاده میشود.

$$\Delta x_1 = g(x_k, u_k) \times T_s$$

$$\Delta x_2 = g\left(x_k + \frac{\Delta x_1}{2}, u_k\right) \times T_s$$

$$\Delta x_3 = g\left(x_k + \frac{\Delta x_2}{2}, u_k\right) \times T_s$$

$$\Delta x_4 = g(x_k + \Delta x_3, u_k) \times T_s$$

$$x_{k+1} = x_k + \frac{1}{6}(\Delta x_1 + \Delta x_2 + \Delta x_3 + \Delta x_4) = f(x_k, u_k)$$

که در آن منظور از $g(x, u)$ همان تابع `state_derivative` پیاده سازی شده است.

3. مدل اندازه گیری

در قسمت پیش معادله $x_{k+1} = f(x_k, u_k)$ استخراج شد که از این معادله به عنوان مدل سیستم در فیلتر کالمن استفاده خواهیم کرد. اکنون نیاز است تا مدل اندازه گیری برای gps نیز طراحی شود. خروجی gps شامل موقعیت و سرعت میباشد بنابراین مدل اندازه گیری به شکل زیر تعریف میشود.

$$y = h(x)$$

$$y = Hx = \begin{bmatrix} l \\ \lambda \\ h \\ v_n \\ v_e \\ v_d \\ \psi \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4. طراحی و پیاده سازی فیلتر کالمن مستقیم

در این قسمت معادلات کامل فیلتر توسعه یافته یا همان EKF برای سیستم پیاده سازی شده و گام های prediction و نیز correction در این قسمت بحث میشود. همان طور که در قسمت قبل بیان شد تا کنون مدل سیستم و مدل اندازه گیری بیان شده است. ابتدا معادلات عمومی فیلتر کالمن را بیان میکنیم و سپس آن را برای معادلات سیستم استفاده خواهیم کرد.

✓ گام prediction

در این گام با استفاده از حل معادلات به روش RK4 ابتدا بردار حالت سیستم را با استفاده از سنسور IMU گام جلو میبریم و همزمان ماتریس کوواریانس سیستم را نیز حل خواهیم کرد. معادلات به شکل زیر خواهد بود.

$$x_{k+1} = x_k + \frac{1}{6}(\Delta x_1 + \Delta dx_2 + \Delta dx_3 + \Delta x_4) = f(x_k, u_k)$$

$$P_{k+1} = F_k P_k F_k^T + Q_k$$

که در معادله فوق نیاز است ماتریس F_k به شکل زیر محاسبه شود.

$$F_k = \frac{\partial f(x_k, u_k)}{\partial x_k} \Big|_{x_k, u_k}$$

برای محاسبه این ماتریس کد زیر در متلب استفاده شده است.

```
syms lon lat h vn ve vd phi theta si real
syms wx wy wz fx fy fz real
syms bax bay baz
syms bgx bgy bgz
syms Ts

x_sym = [lon; lat; h; vn; ve; vd; phi; theta; si; bax; bay; baz; bgx; bgy; bgz];
Wbi_b_sym = [wx; wy; wz];
Fbi_b_sym = [fx; fy; fz];

dx1 = state_derivative(x_sym, Wbi_b_sym, Fbi_b_sym) * Ts;
dx2 = state_derivative(x_sym + dx1 / 2, Wbi_b_sym, Fbi_b_sym) * Ts;
dx3 = state_derivative(x_sym + dx2 / 2, Wbi_b_sym, Fbi_b_sym) * Ts;
dx4 = state_derivative(x_sym + dx3, Wbi_b_sym, Fbi_b_sym) * Ts;
x_next = x_sym + (1/6) * (dx1 + 2 * dx2 + 2 * dx3 + dx4);

Fk_sym = jacobian(x_next, x_sym);

% Create function file for numeric Jacobian evaluation
matlabFunction(Fk_sym, 'File', 'Fk_numeric_simple', ...
    'Vars', {x_sym, Wbi_b_sym, Fbi_b_sym, Ts});
```

متاسفانه بدليل آنکه محاسبات این کد دچار انفجار پيچيدگی ميشود اين کد در حين اجرا بسيار زمان بر ميگردد به همين علت از کد زير برای تقریب این ماتریس استفاده خواهد شد.

```
syms lon lat h vn ve vd phi theta si real
syms wx wy wz fx fy fz real
syms bax bay baz
syms bgx bgy bgz
syms Ts real

x_sym = [lon; lat; h; vn; ve; vd; phi; theta; si; bax; bay; baz; bgx; bgy; bgz];
Wbi_b_sym = [wx; wy; wz];
Fbi_b_sym = [fx; fy; fz];

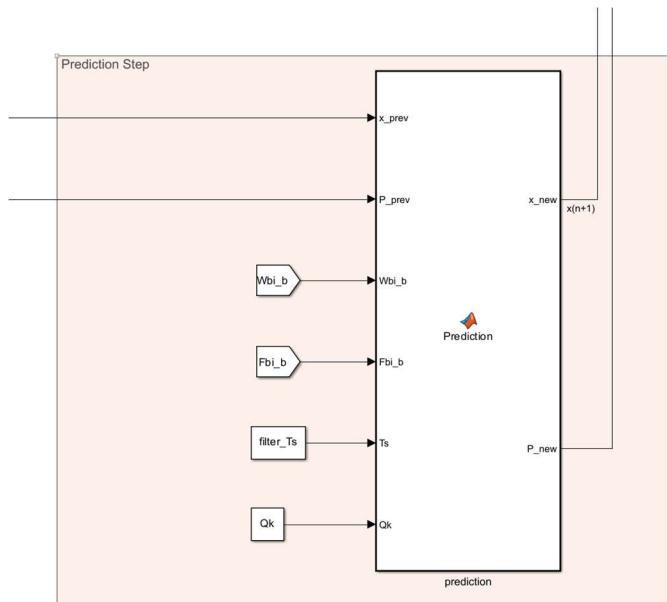
dx1 = state_derivative(x_sym, Wbi_b_sym, Fbi_b_sym) * Ts;
x_next = x_sym + dx1;
```

```
% state Jacobian
Fk_sym = jacobian(x_next, x_sym);

% Create function file for numeric Jacobian evaluation
matlabFunction(Fk_sym, 'File', 'Fk', ...
    'Vars', {x_sym, Wbi_b_sym, Fbi_b_sym, Ts});
```

خروجی این کد تابع 'Fk' ذخیره خواهد شد و در حین شبیه سازی از آن استفاده خواهد شد.

حال میتوان بلوک مربوط به گام prediction را در سیمولینک شبیه سازی نمود.



این بلوک محاسبات مربوط به گام prediction را انجام داده و در خروجی مقدار پیش بینی گام بعد را به همراه ماتریس کوواریانس تحويل میدهد. کد این بلوک به شکل زیر میباشد.

```
function [x_new, P_new] = Prediction(x_prev, P_prev, Wbi_b, Fbi_b, Ts, Qk)

dx1 = state_derivative(x_prev, Wbi_b, Fbi_b) * Ts;
dx2 = state_derivative(x_prev + dx1/2, Wbi_b, Fbi_b) * Ts;
dx3 = state_derivative(x_prev + dx2/2, Wbi_b, Fbi_b) * Ts;
dx4 = state_derivative(x_prev + dx3, Wbi_b, Fbi_b) * Ts;

x_new = x_prev + (dx1 + 2*dx2 + 2*dx3 + dx4)/6;

F = Fk(x_prev, Wbi_b, Fbi_b, Ts);
P_new = F * P_prev * F.' + Qk;
end
```

correction گام ✓

در لحظاتی که داده GPS جدیدی میرسد باید تخمین حالت ها اصلاح شود. بنابراین در این قسمت به شکل زیر
حالت های سیستم را اصلاح میکنیم.

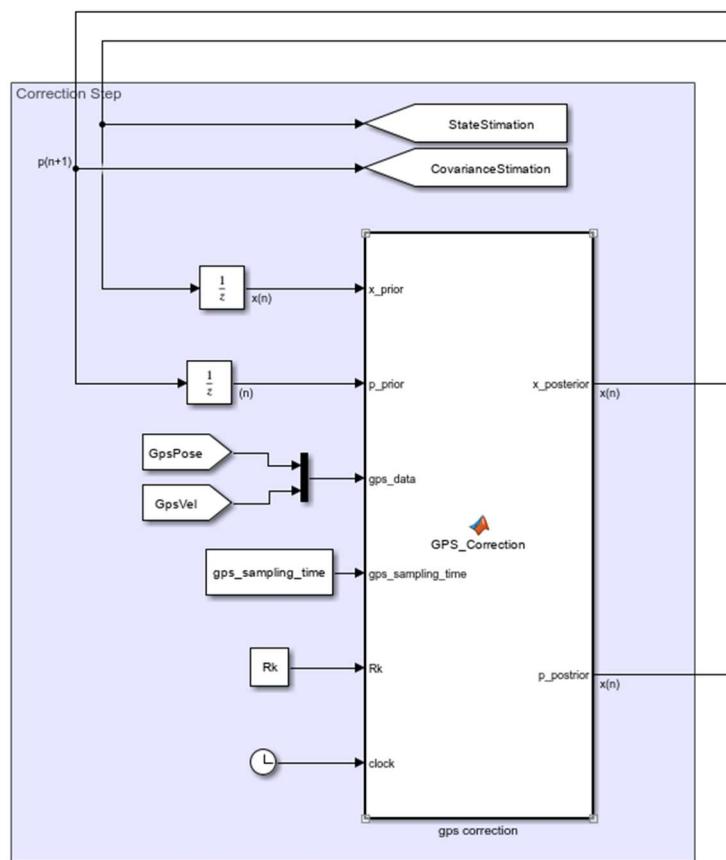
$$S_k = H P_k^- H^T + R_k$$

$$K_k = P_k^- H^T S_k^{-1}$$

$$x_k^+ = x_k^- + K_k (y_k - H x_k^-)$$

$$P_k^+ = (I - K_k H) P_k^- (I - K_k H)^T + K_k R_k K_k^T$$

بلوک دیاگرام مربوط به این قسمت به شکل زیر خواهد بود.



بنابراین این بلوک تصحیحات را بر روی حالت و ماتریس کوواریانس انجام داده و آن را برای گام prediction ارسال میکند. لازم به ذکر است که در لحظاتی که سنسور اطلاعاتی ندارد همان حالت ها و کوواریانس های دریافت شده به عنوان خروجی گزارش خواهند شد. کد این بلوک به شکل زیر خواهد بود.

```

function [x_posterior, p_posterior] = GPS_Correction(x_prior, p_prior, gps_data,
gps_sampling_time, Rk, clock)
    % Default outputs (unchanged)
    coder.extrinsic('disp')
    x_posterior = x_prior;
    p_posterior = p_prior;
    K = zeros(15, 6);
    innovation = zeros(6, 1);
    S = zeros(6, 6);

    if mod(clock, gps_sampling_time) == 0

        H = [eye(6), zeros(6,9)];

        innovation = gps_data - H * x_prior;
        S = H * p_prior * H' + Rk;

        y_tilde = innovation;
        K = p_prior * H' / S;
        x_posterior = x_prior + K * y_tilde;
        I = eye(15);
        p_posterior = (I - K * H) * p_prior * (I - K * H)' + K * Rk * K';

    end
end

```

5. طراحی و پیاده سازی فیلتر کالمن غیرمستقیم

در این قسمت معادلات کامل فیلتر غیرمستقیم برای سیستم پیاده سازی شده و گام های prediction و نیز correction در این قسمت بحث میشود. ابتدا معادلات عمومی فیلتر کالمن را بیان میکنیم و سپس آن را برای معادلات سیستم استفاده خواهیم کرد.

✓ خطی سازی معادلات

برخلاف EKF که در آن از معادلات غیرخطی حکم بر سیستم در گام prediction استفاده میشد. در این قسمت خطای سیستم به صورت خطی تخمین زده میشود. برای خطی سازی نیر از بسط تیلور استفاده میکنیم.

$$x_{k+1} = f(x_k, u_k) \approx f(x_{ins|k}, u_{ins|k}) + \left. \frac{\partial f(x_k, u_k)}{\partial x_k} \right|_{x_{ins|k}, u_{ins|k}} (x_k - x_{ins|k}) + \left. \frac{\partial f(x_k, u_k)}{\partial u_k} \right|_{x_{ins|k}, u_{ins|k}} (u_k - u_{ins|k})$$

مقدار $f(x_{ins|k}, u_{ins|k})$ همان مقدار $x_{ins|k+1}$ یعنی خروجی FreeINS در گام آینده است. بنابراین دینامیک خطای حالت ها را میتوان به صورت زیر نوشت. لازم به ذکر است که تفاصل بین حالت واقعی سیستم و حالتی که از طریق انتگرال گیری گستته از FreeINS بدست می آید به عنوان خطای سیستم در نظر گرفته میشود.

$$\delta x_{k+1} = x_{k+1} - x_{ins|k+1} = \frac{\partial f(x_k, u_k)}{\partial x_k} \Bigg|_{x_{ins|k}, u_{ins|k}} (x_k - x_{ins|k}) + \frac{\partial f(x_k, u_k)}{\partial u_k} \Bigg|_{x_{ins|k}, u_{ins|k}} (u_k - u_{ins|k})$$

$$\delta x_{k+1} = x_{k+1} - x_{ins|k+1} = \frac{\partial f(x_k, u_k)}{\partial x_k} \Bigg|_{x_{ins|k}, u_{ins|k}} \delta x_k + \frac{\partial f(x_k, u_k)}{\partial u_k} \Bigg|_{x_{ins|k}, u_{ins|k}} \delta u_k$$

$$\delta u_k \rightarrow gaussian noise wrt Q_k$$

precition گام ✓

در این گام مقدار خطای سیستم را گام به گام پیش‌بینی می‌کنیم و در این گام از معادلات خطی سازی شده برای خطای سیستم استفاده خواهیم نمود. معادلات این قسمت به شکل زیر خواهد بود.

$$F_{ins} = \frac{\partial f(x_k, u_k)}{\partial x_k} \Bigg|_{x_{ins|k}, u_{ins|k}}$$

$$B_{ins} = \frac{\partial f(x_k, u_k)}{\partial u_k} \Bigg|_{x_{ins|k}, u_{ins|k}}$$

$$\delta x_{k+1} = F_{ins} \delta x_k + B_{ins} \delta u_k$$

$$P_{k+1} = F_{ins} P_k F_{ins}^T + Q_k$$

ماتریس‌های F_{ins} و B_{ins} فقط در لحظاتی که در آن شرایط **reseting** رخ میدهد آپدیت خواهند شد. شرایط قابل تنظیم بوده و در نتایج مقدار 0.1 در نظر گرفته شده است تا نتایج مناسب باشند.

همچنین لازم به ذکر است که B_{ins} و F_{ins} در مطلب از طریق کد زیر تبدیل به توابع آماده شده اند.

```
syms lon lat h vn ve vd phi theta si real
syms wx wy wz fx fy fz real
syms bax bay baz
syms bgx bgy bgz
syms Ts real

x_sym = [lon; lat; h; vn; ve; vd; phi; theta; si; bax; bay; baz; bgx; bgy; bgz];
Wbi_b_sym = [wx; wy; wz];
Fbi_b_sym = [fx; fy; fz];

dx1 = state_derivative(x_sym, Wbi_b_sym, Fbi_b_sym) * Ts;
```

```

x_next = x_sym + dx1;

% state Jacobian
Fk_sym = jacobian(x_next, x_sym);

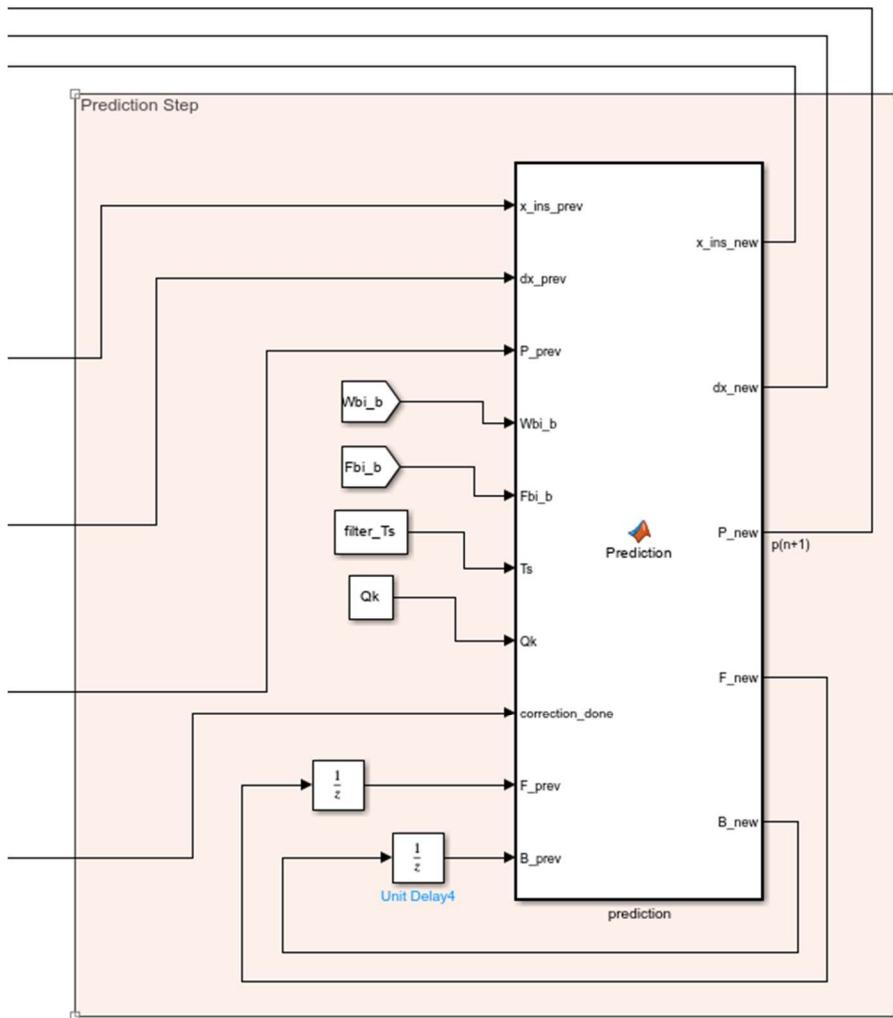
Bk_sym = jacobian(x_next, [Wbi_b_sym; Fbi_b_sym]);

% Create function file for numeric Jacobian evaluation
matlabFunction(Fk_sym, 'File', 'Fk', ...
    'Vars', {x_sym, Wbi_b_sym, Fbi_b_sym, Ts});

matlabFunction(Bk_sym, 'File', 'Bk', ...
    'Vars', {x_sym, Wbi_b_sym, Fbi_b_sym, Ts});

```

بلوک سیمولینک این قسمت به شکل زیر میباشد.



کد مربوط به این قسمت نیز به شکل زیر میباشد.

```

function [x_ins_new, dx_new, P_new, F_new, B_new] = Prediction(x_ins_prev, dx_prev,
P_prev, Wbi_b, Fbi_b, Ts, Qk, correction_done, F_prev, B_prev)

    % FreeINS propagation
    dx1 = state_derivative(x_ins_prev , Wbi_b, Fbi_b) * Ts;
    dx2 = state_derivative(x_ins_prev + dx1/2 , Wbi_b, Fbi_b) * Ts;
    dx3 = state_derivative(x_ins_prev + dx2/2 , Wbi_b, Fbi_b) * Ts;
    dx4 = state_derivative(x_ins_prev + dx3 , Wbi_b, Fbi_b) * Ts;

    x_ins_new = x_ins_prev + (dx1 + 2*dx2 + 2*dx3 + dx4)/6;

    % resetting if needed
    F_new = F_prev;
    B_new = B_prev;
    if correction_done
        u_ins_new = [Wbi_b; Fbi_b];
        F_new = Fk(x_ins_new, u_ins_new(1:3), u_ins_new(4:6), Ts);
        B_new = Bk(x_ins_new, u_ins_new(1:3), u_ins_new(4:6), Ts);
    end

    % Error State Propagation
    imu_noise_var = diag(Qk(4:9, 4:9));
    dx_new = F_new * dx_prev + B_new * (randn(6, 1) .* sqrt(imu_noise_var));
    P_new = F_new * P_prev * F_new.' + Qk;

end

```

گام ✓

در این گام از اندازه گیری رسیده از GPS استفاده میشود تا تخمین خطای حالت ها اصلاح شود. در این گام معادلات زیر پیاده سازی میشوند.

$$S_k = H P_k^- H^T + R_k$$

$$K_k = P_k^- H^T S_k^{-1}$$

$$\delta y_k = y_k - H x_{ins|k}^-$$

$$\delta x_k^+ = \delta x_k^- + K_k (\delta y_k - H \delta x_k^-)$$

$$P_k^+ = (I - K_k H) P_k^- (I - K_k H)^T + K_k R_k K_k^T$$

همچنین در این قسمت به توجه به پارامتر `reseting_rate` که به صورت پیش فرض 0.1 ثانیه است به شکل زیر ریست انجام میگردد.

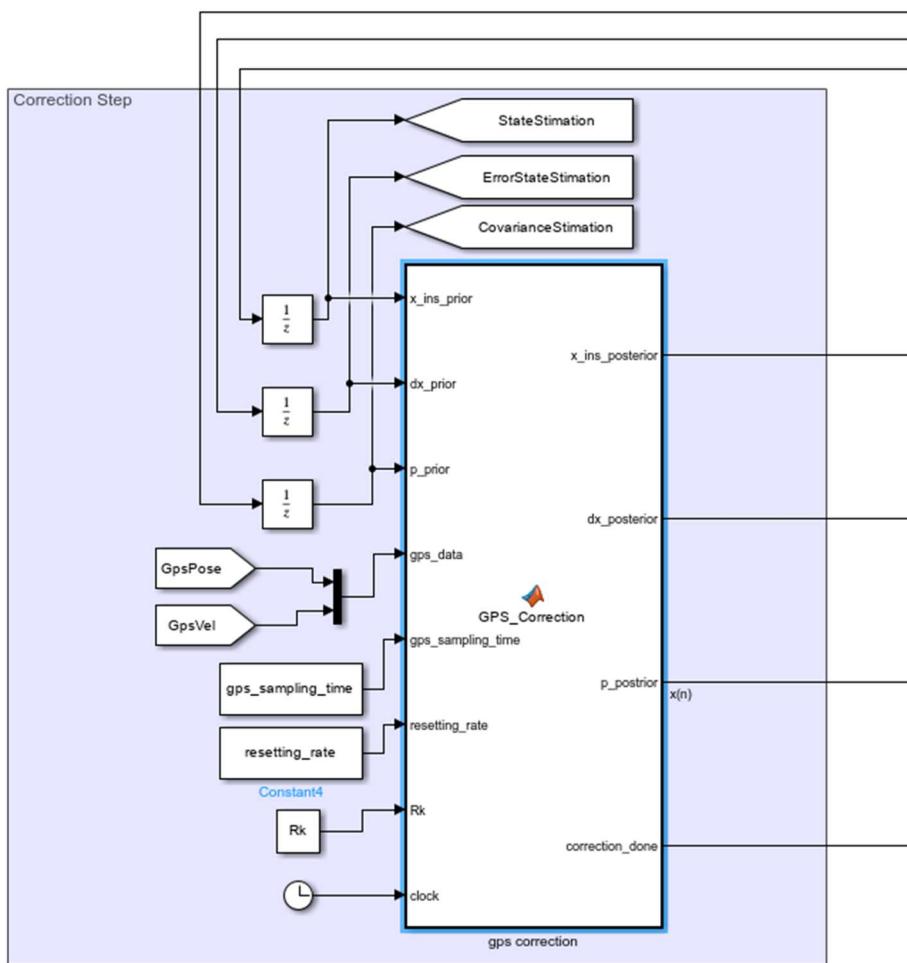
$$x_{ins|k} = x_{ins|k} + \delta x_k^+$$

$$\delta x_k^+ = 0$$

$$F_{ins} = \left. \frac{\partial f(x_k, u_k)}{\partial x_k} \right|_{x_{ins|k}, u_{ins|k}}$$

$$B_{ins} = \left. \frac{\partial f(x_k, u_k)}{\partial u_k} \right|_{x_{ins|k}, u_{ins|k}}$$

بلوک مربوط به این قسمت به شکل زیر میباشد.



کد پیاده سازی شده در این بلوک به شکل زیر است.

```

function [x_ins_posterior, dx_posterior, p_posterior, correction_done] =
GPS_Correction(x_ins_prior, dx_prior, p_prior, gps_data, gps_sampling_time,
resetting_rate, Rk, clock)
coder.extrinsic('disp')
p_posterior = p_prior;
dx_posterior = dx_prior;
x_ins_posterior = x_ins_prior;
K = zeros(15, 6);
innovation = zeros(6, 1);
S = zeros(6, 6);
correction_done = 0;

if mod(clock, gps_sampling_time) == 0

H = [eye(6), zeros(6,9)];

S = H * p_prior * H' + Rk;
dy = gps_data - H * x_ins_prior;
K = p_prior * H' / S;
innovation = dy - H * dx_prior;

dx_posterior = dx_prior + K * innovation;

I = eye(15);
p_posterior = (I - K * H) * p_prior * (I - K * H)' + K * Rk * K';

end

if mod(clock, resetting_rate) == 0
correction_done = 1;
x_ins_posterior = x_ins_prior + dx_posterior;
dx_posterior = zeros(length(dx_prior), 1);
end
end

```

6. نتایج شبیه سازی

در این قسمت شبیه سازی های انجام شده توسط دو روش کالمن فیلتر مستقیم و غیر مستقیم را ارزیابی خواهیم کرد. قبل از این کار باید مقادیر ماتریس های کوواریانس مربوط به سنسور های اندازه گیری شود تا در ماتریس های R_k و Q_k مشخص گردد.

باتوجه به اینکه مقادیر درست بردار حالت در دسترس است میتوان نویز مربوط به سنسور GPS را به طور مستقیم با مقایسه کردن خروجی آن با مقادیر صحیح بدست آورد. برای اینکار از کد زیر برای محاسبه ماتریس کوواریانس استفاده میشود.

```

function R = computeGpsMeasurementNoise(gpsData10Hz, trueData100Hz)
% Ensure gpsData10Hz has the necessary fields
requiredFields = {'lon', 'lat', 'alt', 've', 'vn', 'vd'};

```

```

for i = 1:length(requiredFields)
    if ~isfield(gpsData10Hz, requiredFields{i})
        error(['Missing field: ', requiredFields{i}]);
    end
end

% Extract data
lat_gps = rad2deg(gpsData10Hz.lat);
lon_gps = rad2deg(gpsData10Hz.lon);
alt_gps = gpsData10Hz.alt;
ve_gps = gpsData10Hz.ve;
vn_gps = gpsData10Hz.vn;
vd_gps = gpsData10Hz.vd;

lat_gt = rad2deg(trueData100Hz.lat(1:10:end));
lon_gt = rad2deg(trueData100Hz.lon(1:10:end));
alt_gt = trueData100Hz.alt(1:10:end);
ve_gt = trueData100Hz.ve(1:10:end);
vn_gt = trueData100Hz.vn(1:10:end);
vd_gt = trueData100Hz.vd(1:10:end);

% Compute zero-mean position and velocity
pos_demeaned = [lat_gps, lon_gps, alt_gps] - [lat_gt, lon_gt, alt_gt];
vel_demeaned = [ve_gps, vn_gps, vd_gps] - [ve_gt, vn_gt, vd_gt];

pos_var = var(pos_demeaned); % [σ²_east, σ²_north, σ²_up]
vel_var = var(vel_demeaned); % [σ²_ve, σ²_vn, σ²_vd]

% Construct 6x6 measurement noise covariance matrix
R = diag([pos_var, vel_var]);
end

```

خروجی این کد به شکل زیر بوده است.

$$R = \begin{bmatrix} 9.07252092613911e-11 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9.07204808274199e-11 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.991470480960317 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.00998692051771264 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.00989027264124697 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.0100334320982840 \end{bmatrix}$$

لازم به ذکر است که این مقدار برابر مقدار پیوسته ماتریس R_k است و در نتیجه برای استفاده از آن مطابق صورت مثال بر T_s تقسیم خواهد شد. بنابراین مقدار R_k واقعی که در correction استفاده خواهد شد به شکل زیر میباشد.

$$R_k = \begin{bmatrix} 9.07252092613911e-09 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9.07204808274199e-09 & 0 & 0 & 0 & 0 \\ 0 & 0 & 99.1470480960317 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.998692051771264 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.989027264124697 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.00334320982840 \end{bmatrix}$$

اکنون باید مقدار ماتریس کوواریانس نویز فرآیند را برای گام prediction مشخص نماییم. برای اینکار از 100,000 داده نخست سنسور IMU برای تعیین مقادیر کوواریانس استفاده میشود (با توجه به اینکه در 100000 داده اول گویا سیستم ساکن بوده است). برای اینکار از کد زیر استفاده میشود.

```
function IMU_NOISE = computeIMUNoise(imuData100Hz, N)
    % Default N to 100000 if not provided
    if nargin < 4
        N = 100000;
    end

    % === Extract IMU data ===
    gyro_data = [imuData100Hz.wx(1:N), imuData100Hz.wy(1:N), imuData100Hz.wz(1:N)];
    accel_data = [imuData100Hz.ax(1:N), imuData100Hz.ay(1:N), imuData100Hz.az(1:N)];

    % === Zero-mean signals ===
    gyro_zero_mean = gyro_data - mean(gyro_data, 1);
    accel_zero_mean = accel_data - mean(accel_data, 1);

    % === Standard deviation and continuous-time variance ===
    gyro_std = std(gyro_zero_mean); % rad/s
    accel_std = std(accel_zero_mean); % m/s^2

    gyro_var_continuous = (gyro_std .^ 2); % rad^2/s^3
    accel_var_continuous = (accel_std .^ 2); % (m/s^2)^2/s

    % === Build diagonal Qk matrix ===
    % For example: 3 accel + 3 gyro
    IMU_NOISE = diag([accel_var_continuous, gyro_var_continuous]);

end
```

خروجی این کد به شکل بوده است.

$$\begin{bmatrix} 0.00562580189672897 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.00341978401695017 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0468847974899124 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3.67873945578278e-06 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.30139029020027e-06 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4.36246618883438e-07 \end{bmatrix}$$

حال با توجه به صورت سوال این مقدار را در T_s ضرب کرده و به عنوان Q_k از آن استفاده خواهیم کرد. لازم به ذکر است که Q_k در قسمت های مربوط به حالت های موقعیت و نیز بایاس مقدار صفر و یا مقدار نزدیک به صفر قرار خواهیم داد.

$$Q_k = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5.62580189672900e-05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.41978401695000e-05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.000468847974899120 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3.67873945600000e-08 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.30139029000000e-08 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4.36246619000000e-09 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

لازم به ذکر است مقادیر اولیه بردار حالت و نیز مقدار اولیه ماتریس کوواریانس به شکل زیر در نظر گرفته شده اند.

$$P_o = \begin{bmatrix} 1.00000000000000e-08 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.00000000000000e-08 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0050000000000000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0050000000000000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0030000000000000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0030000000000000 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$x_o = \begin{bmatrix} 0.900269181102793 \\ 0.591553645549858 \\ 1078.96956610636 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

مورد دیگر که لازم است به آن اشاره شود نحوه بررسی و ارزیابی مقدار خطأ در جایه جایی میباشد. بدین صورت عمل خواهد شد که ابتدا زاویای طول و عرض جغرافیای به همراه ارتفاع به همراه این مقادیر که در داده های صحیح گزارش شده اند ابتدا هر دو از طریق استاندارد ECEF به دستگاه WGS84 انتقال یافته سپس مقدار تفاضل دو بردار حاصله محاسبه شده و سپس این بردار خطأ در دستگاه ناوبری گزارش میشود که برای این کار از کد زیر استفاده شده است.

```
function ned = ned_distance(trueData100Hz, out)
    ned = [0, 0, 0];

    for i = 1:length(trueData100Hz.lat)

        true_lls = [trueData100Hz.lat(i), trueData100Hz.lon(i),
        trueData100Hz.alt(i)];
```

```

estimated_lls = [out.lat(i), out.lon(i), out.alt(i)];

% Convert to radians
lat0 = trueData100Hz.lat(i);
lon0 = trueData100Hz.lon(i);

% Convert both to ECEF
ref_ecef = lla_to_ecef(true_lls);
est_ecef = lla_to_ecef(estimated_lls);

% Delta vector in ECEF
d = est_ecef - ref_ecef;

% Rotation matrix from ECEF to ENU
T_NE = [-sin(lat0)*cos(lon0), -sin(lat0)*sin(lon0), cos(lat0);
         -sin(lon0),           cos(lon0),          0;
         -cos(lat0)*cos(lon0), -cos(lat0)*sin(lon0), -sin(lat0)];

% Convert ECEF delta to ENU
ned(i, :) = T_NE * d;
end
end

function ecef = lla_to_ecef(lls)
% Convert [lat; lon; alt] to ECEF
lat = lls(1);
lon = lls(2);
alt = lls(3);

% WGS84 ellipsoid parameters
a = 6378137.0; % semi-major axis
f = 1 / 298.257223563; % flattening
e2 = f * (2 - f); % eccentricity squared

sin_lat = sin(lat);
cos_lat = cos(lat);
sin_lon = sin(lon);
cos_lon = cos(lon);

N = a / sqrt(1 - e2 * sin_lat^2); % prime vertical radius

x = (N + alt) * cos_lat * cos_lon;
y = (N + alt) * cos_lat * sin_lon;
z = (N * (1 - e2) + alt) * sin_lat;

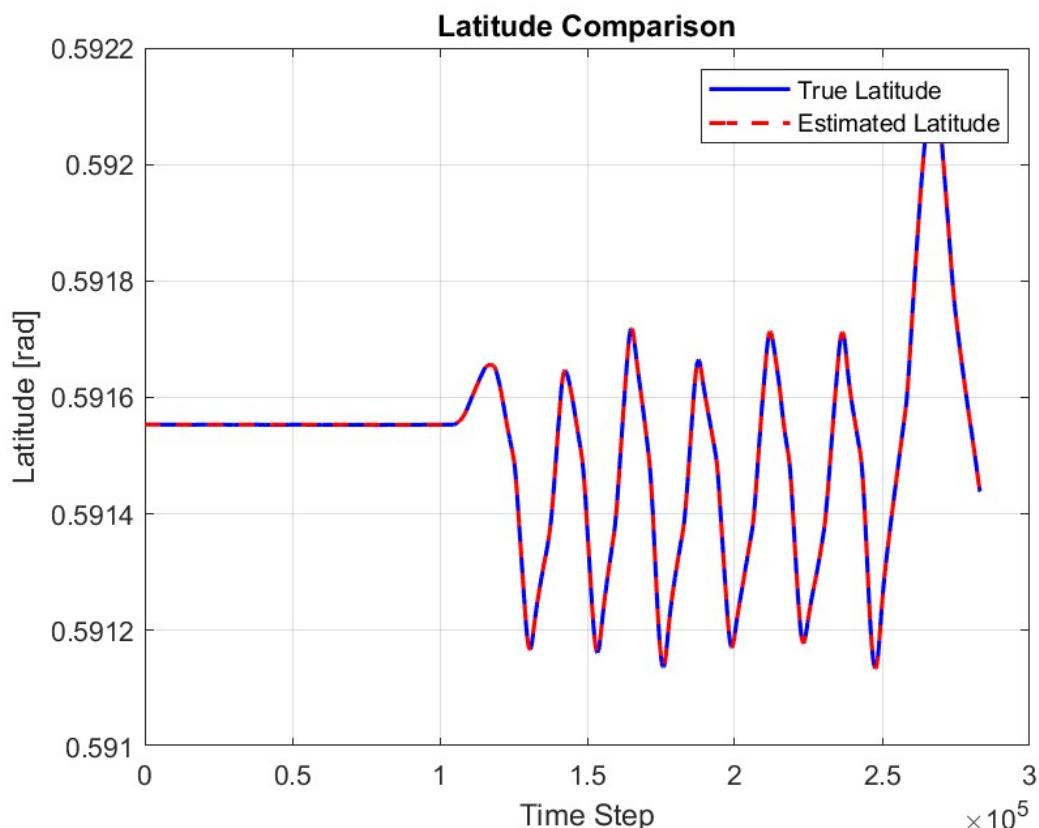
ecef = [x; y; z];
end

```

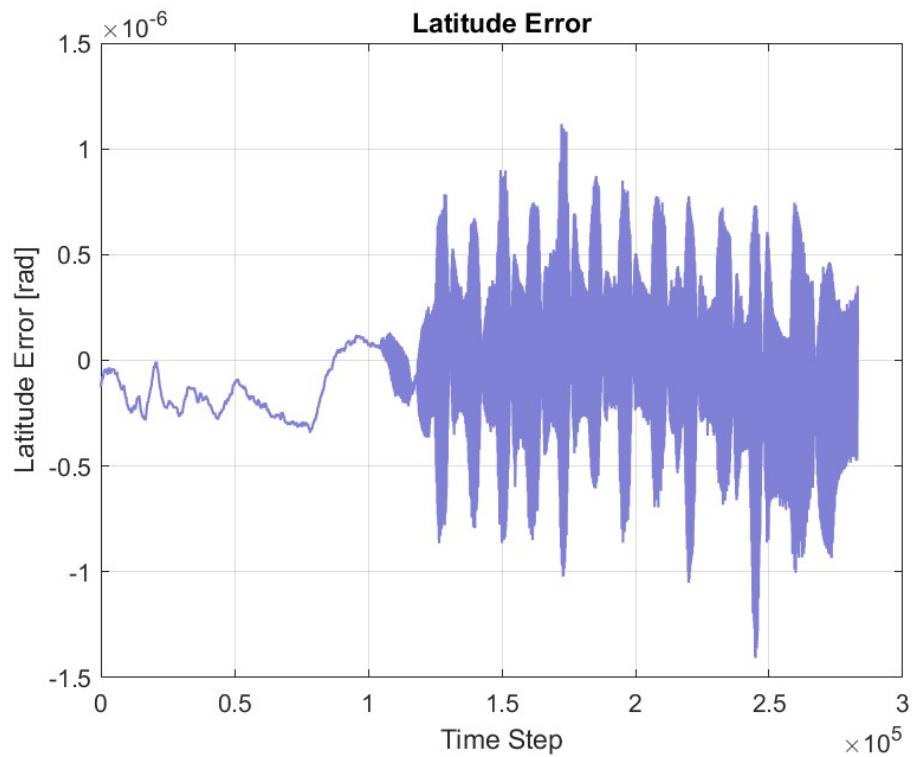
✓ نتایج شبیه سازی فیلتر کالمن مستقیم

نتایج شبیه سازی برای فیلتر کالمن مستقیم به صورت زیر بوده است.

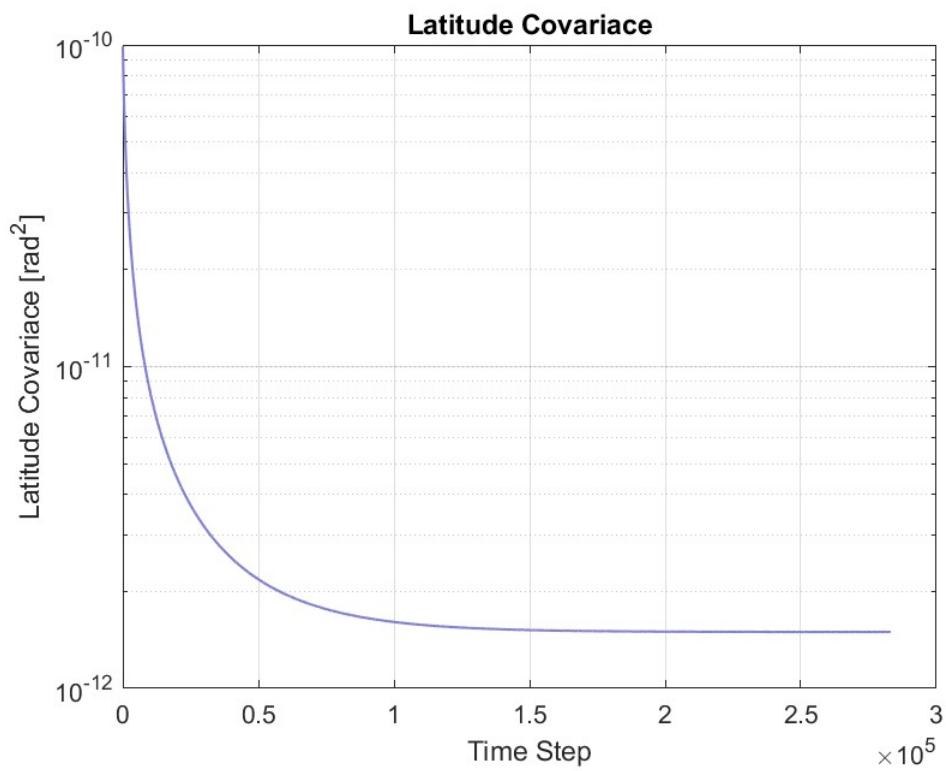
✓ نتایج زاویه Latitude



تصویر ۱ مقایسه زاویه Latitude

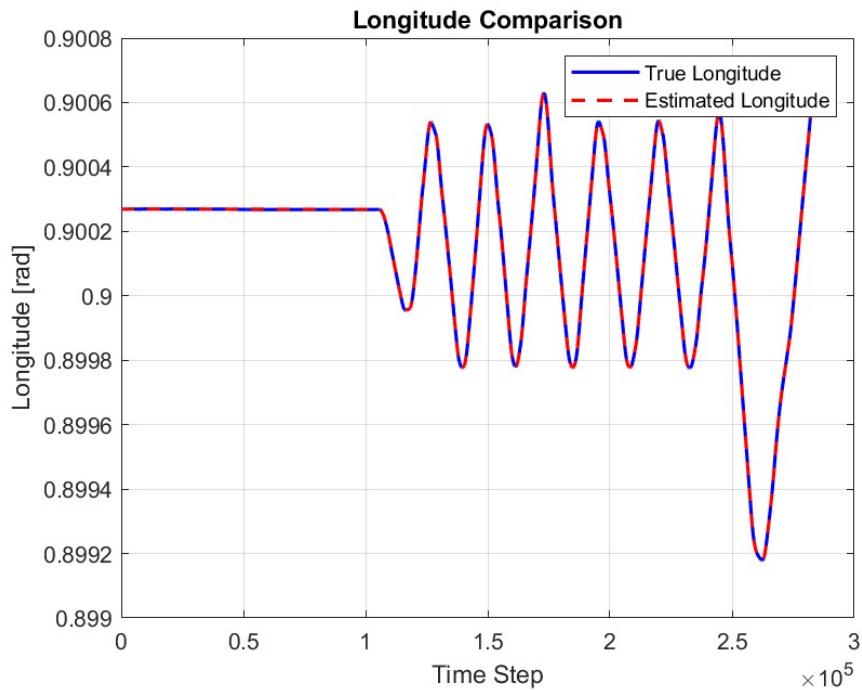


تصویر ۲ خطای تخمین زاویه Latitude

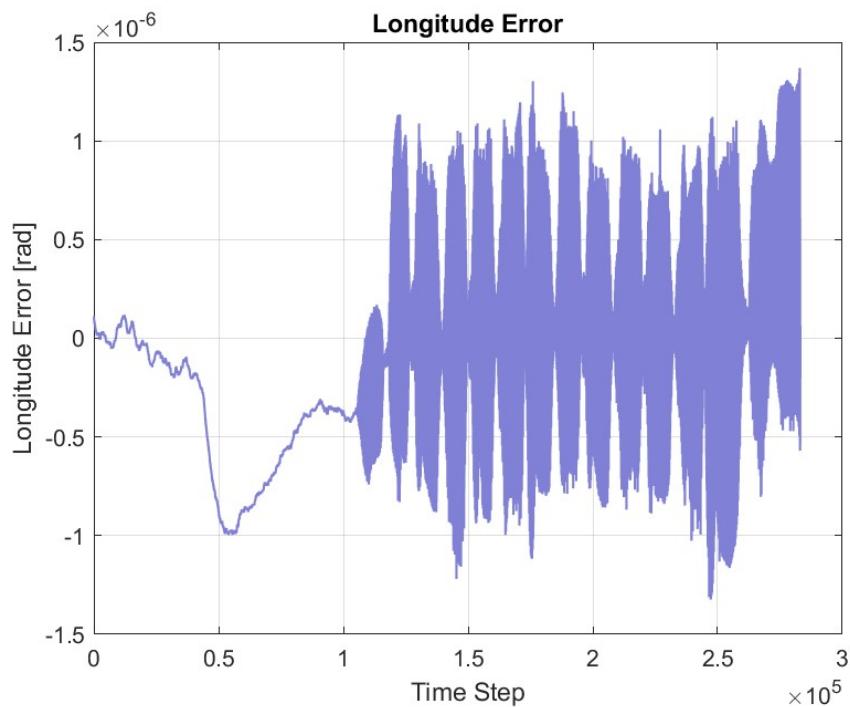


تصویر ۳ کوواریانس زاویه Latitude

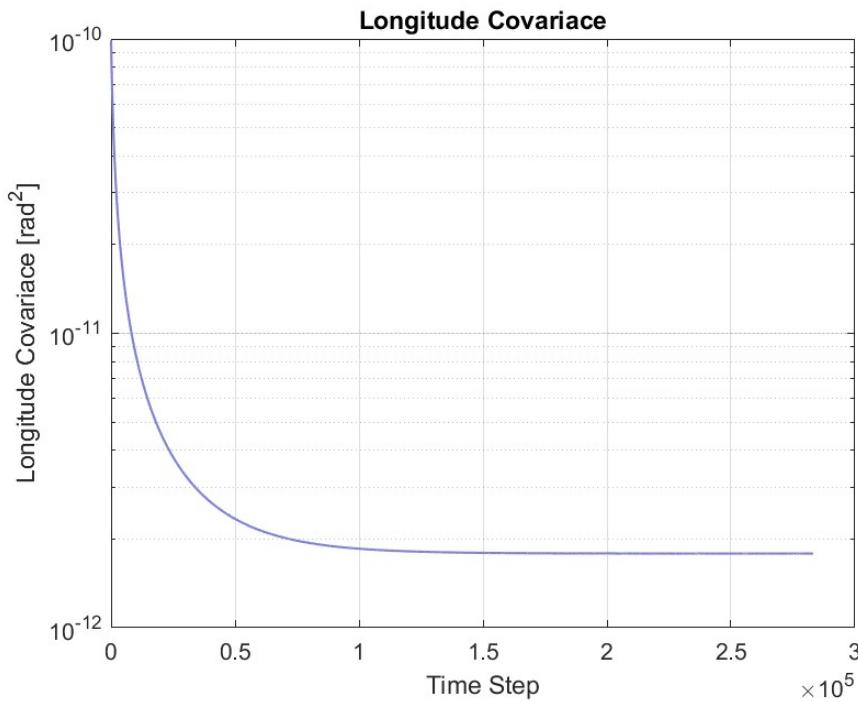
✓ نتایج زاویه Longitude



تصویر 4 مقایسه زاویه Longitude

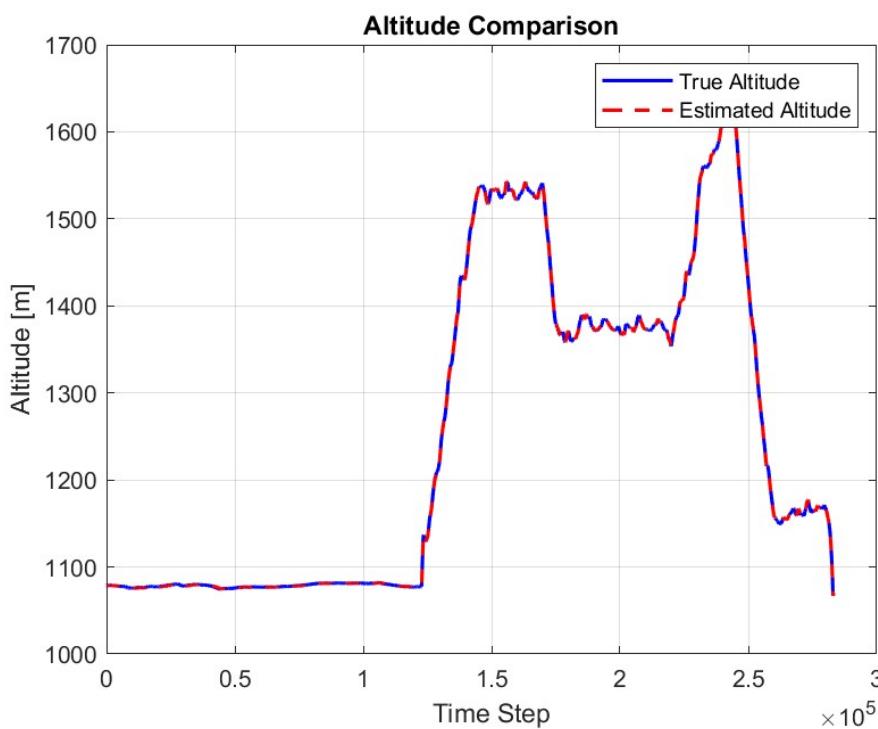


تصویر 5 خطای تخمین زاویه Longitude

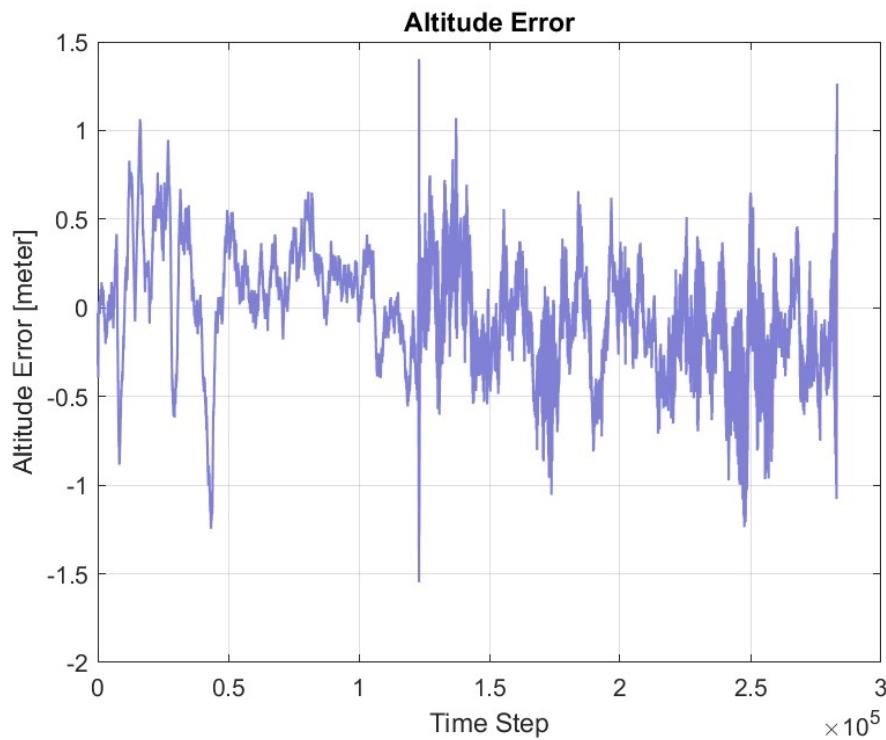


تصویر 6 کوواریانس زاویه Longitude

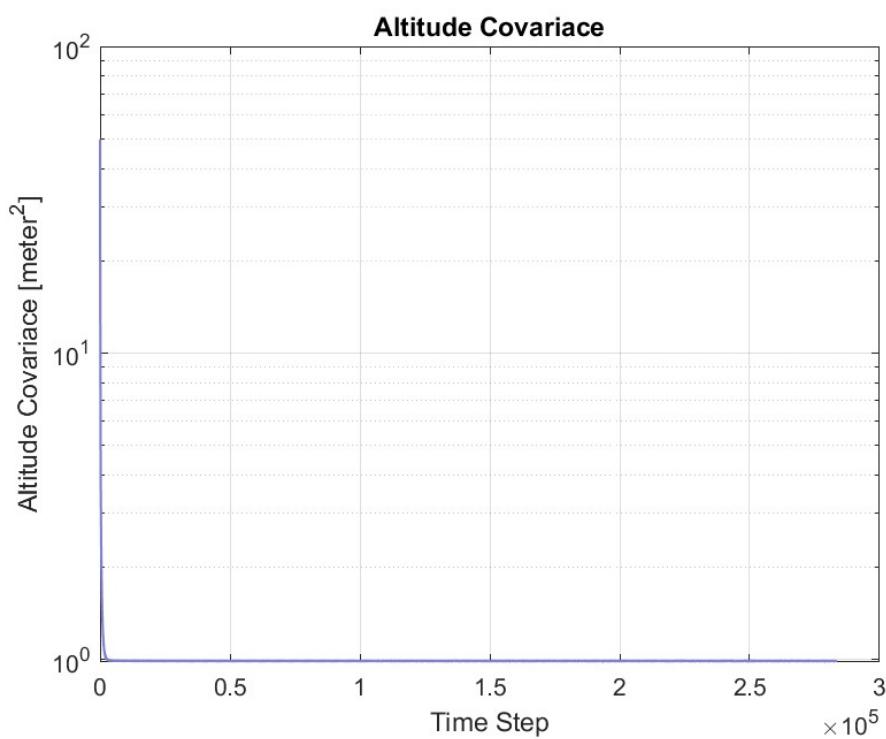
✓ نتایج تخمین ارتفاع



تصویر 7 مقایسه ارتفاع



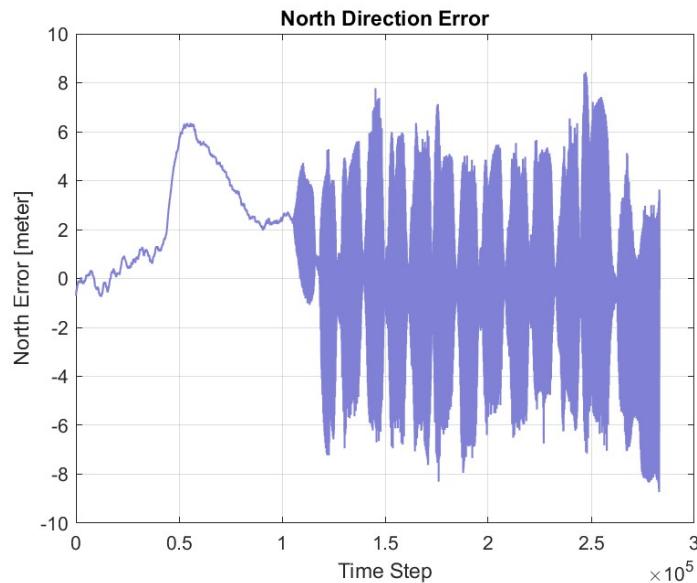
تصویر ۸ خطای تخمین ارتفاع



تصویر ۹ کوواریانس ارتفاع

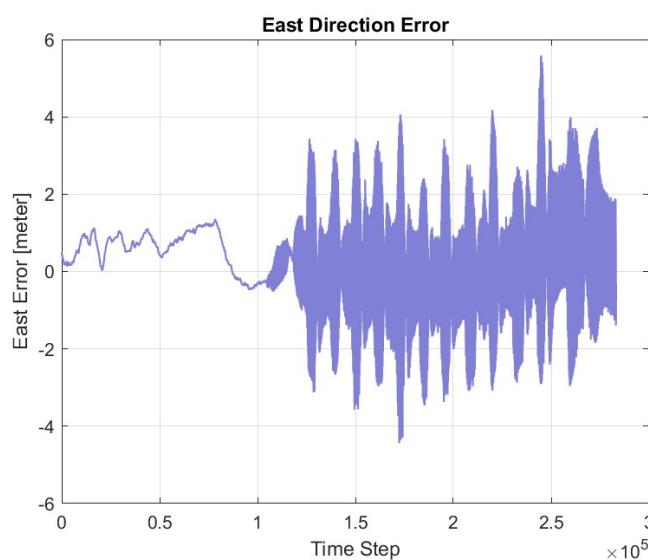
✓ نتایج خطای تخمین در جهت شمال

همانطور که مشاهده میشود علارغم تنظیم ضرایب فیلتر کالمن به علت خطای بالای موجود در راستای طول جغرافیایی سنسور GPS متاسفانه فیلتر خطای زیادی در تخمین موقعیت در راستای شمالی داشته است.



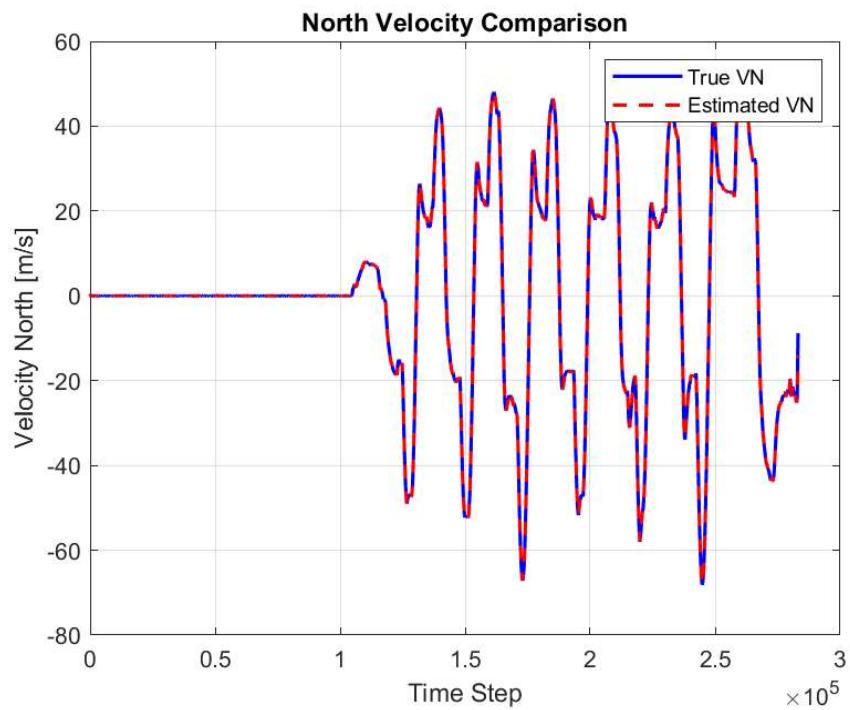
تصویر 10 خطای تخمین موقعیت در راستای شمال

✓ نتایج خطای تخمین در جهت شرق

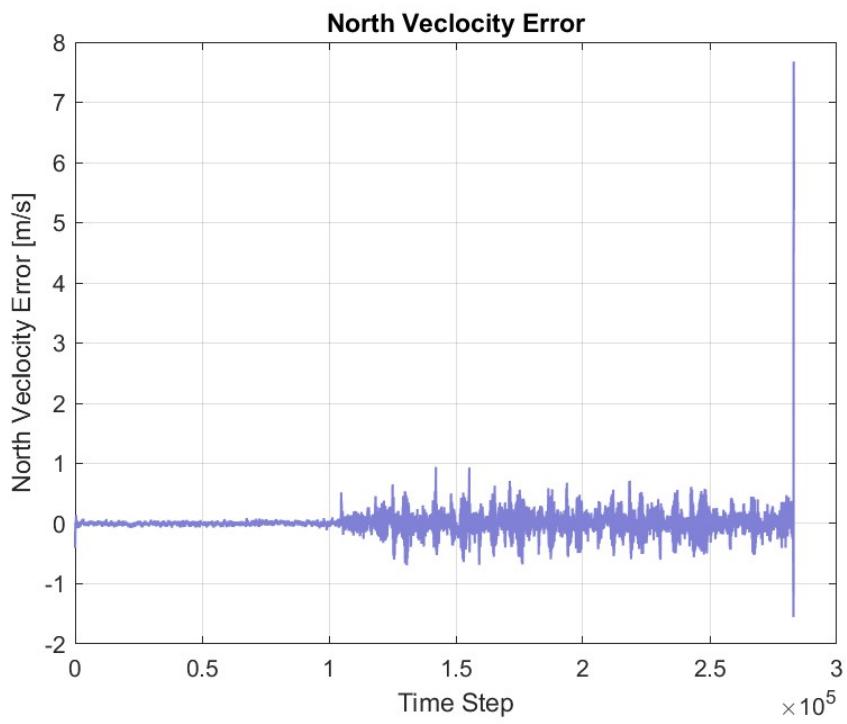


تصویر 11 خطای تخمین موقعیت در راستای شرق

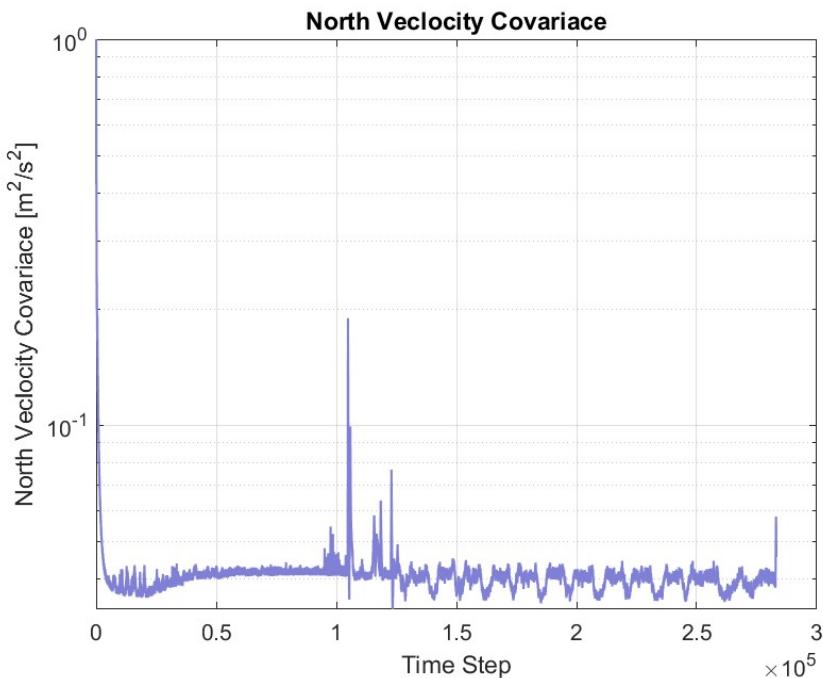
✓ نتایج تخمین سرعت V_n



تصویر ۱۲ مقایسه سرعت V_n

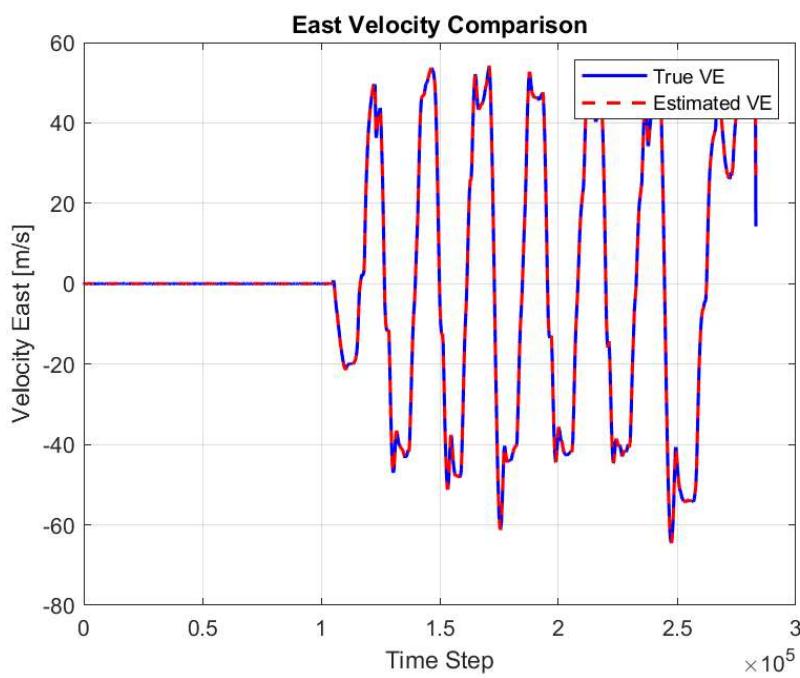


تصویر ۱۳ خطای تخمین سرعت V_n

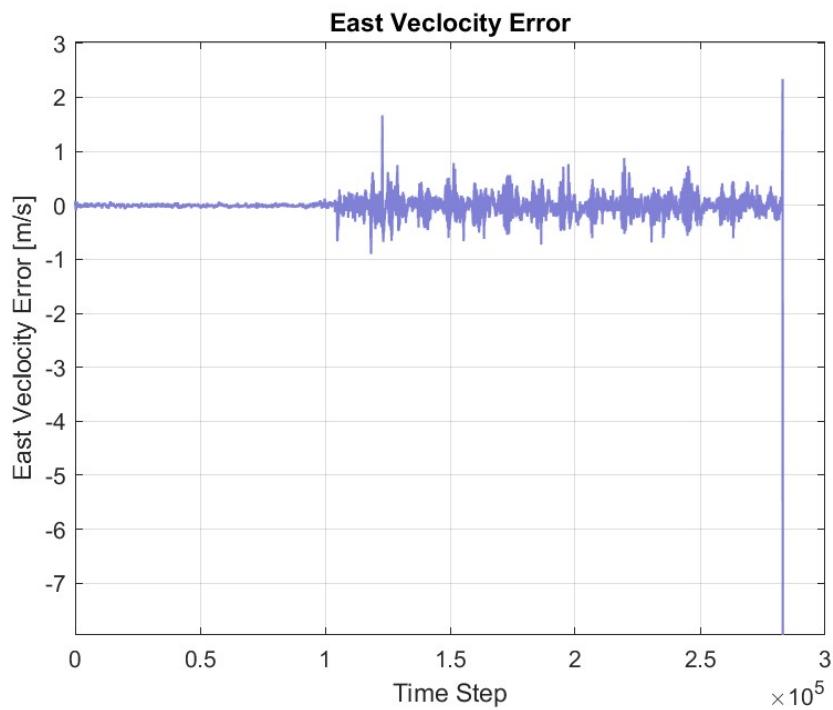


تصویر ۱۴ کواریانس سرعت V_n

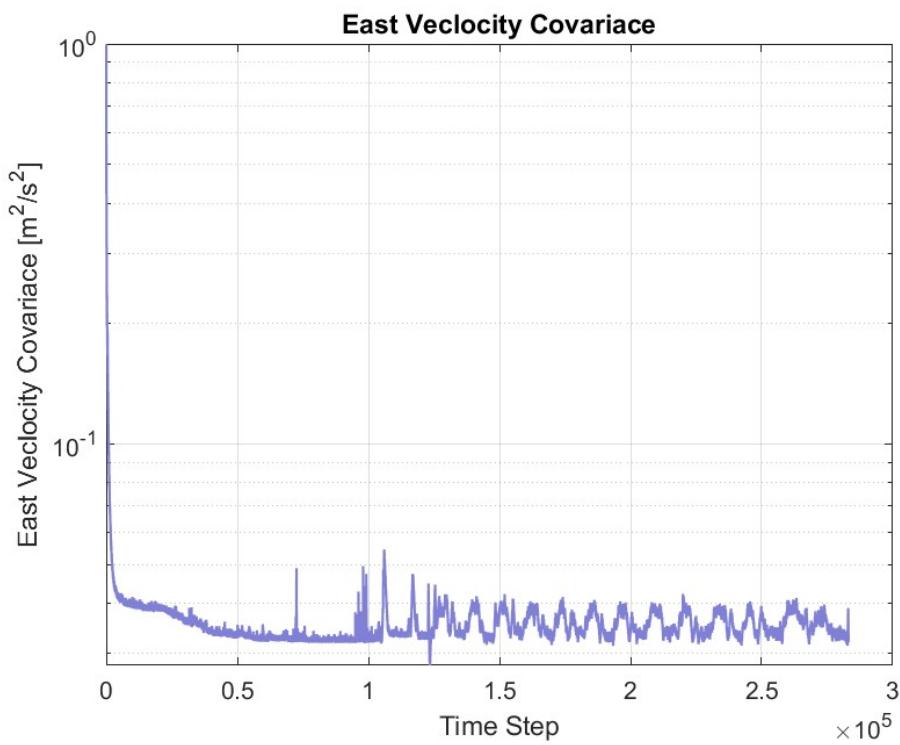
✓ نتایج تخمین سرعت V_e



تصویر ۱۵ مقایسه سرعت V_e

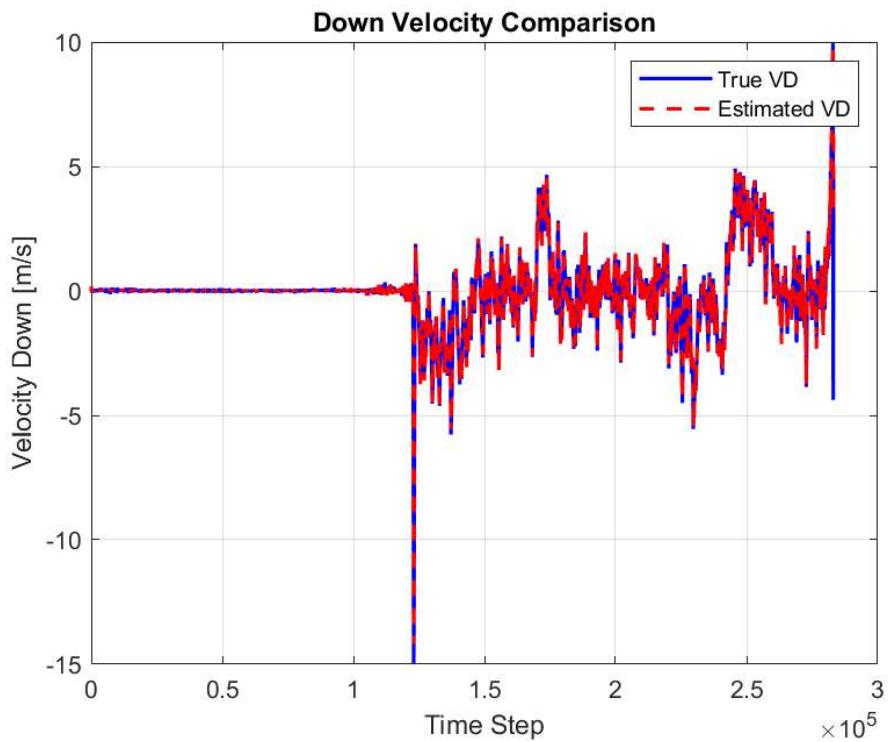


تصویر 16 خطای تخمین سرعت V_e

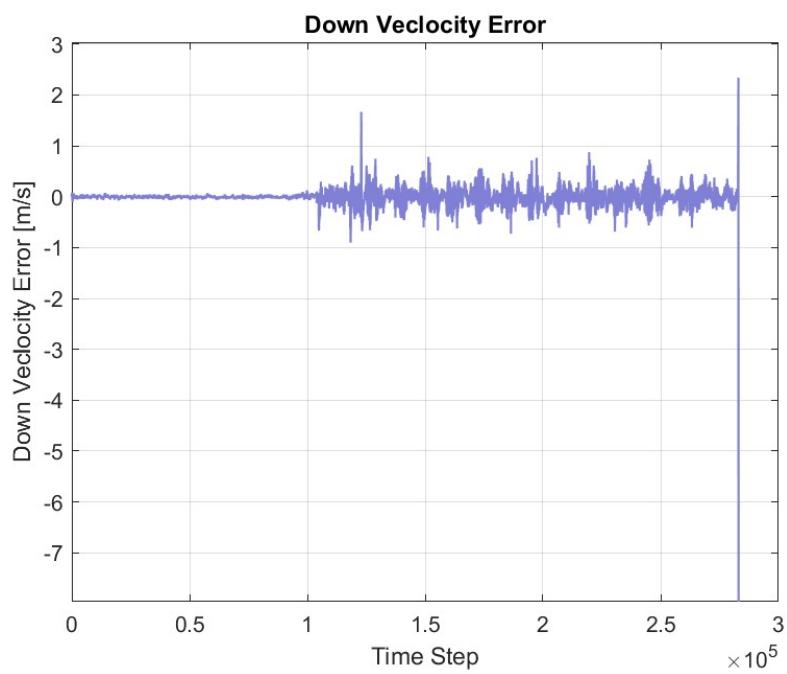


تصویر 17 کوواریانس سرعت V_e

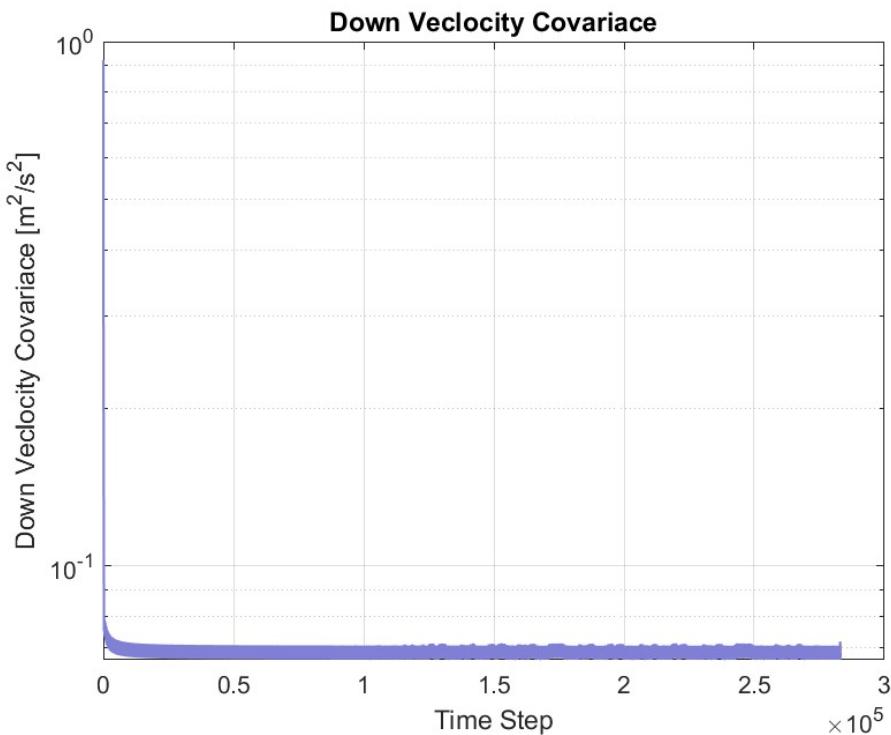
✓ نتایج تخمین سرعت V_d



تصویر 18 مقایسه سرعت V_d

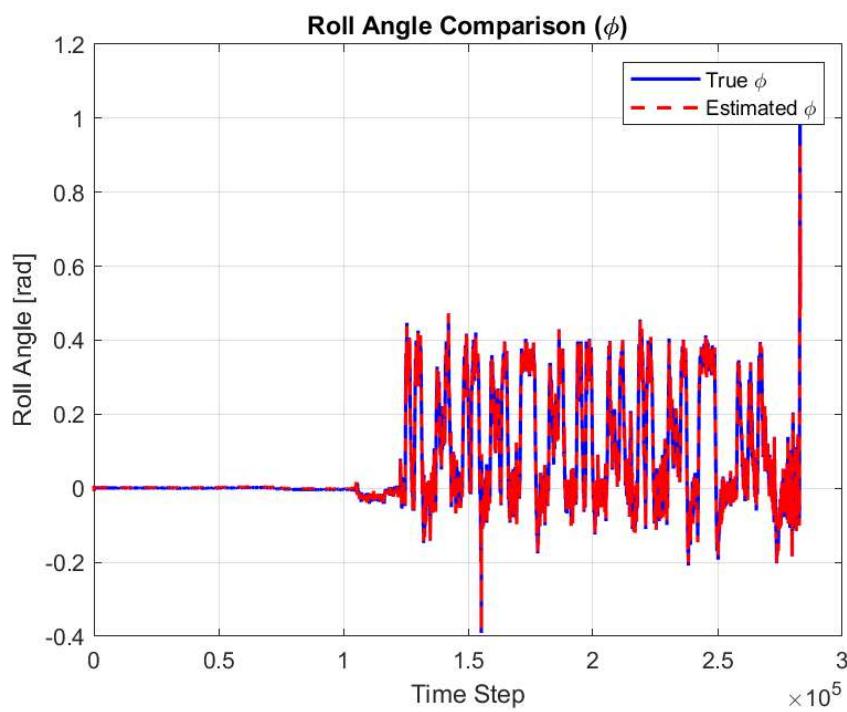


تصویر 19 خطای تخمین سرعت V_d

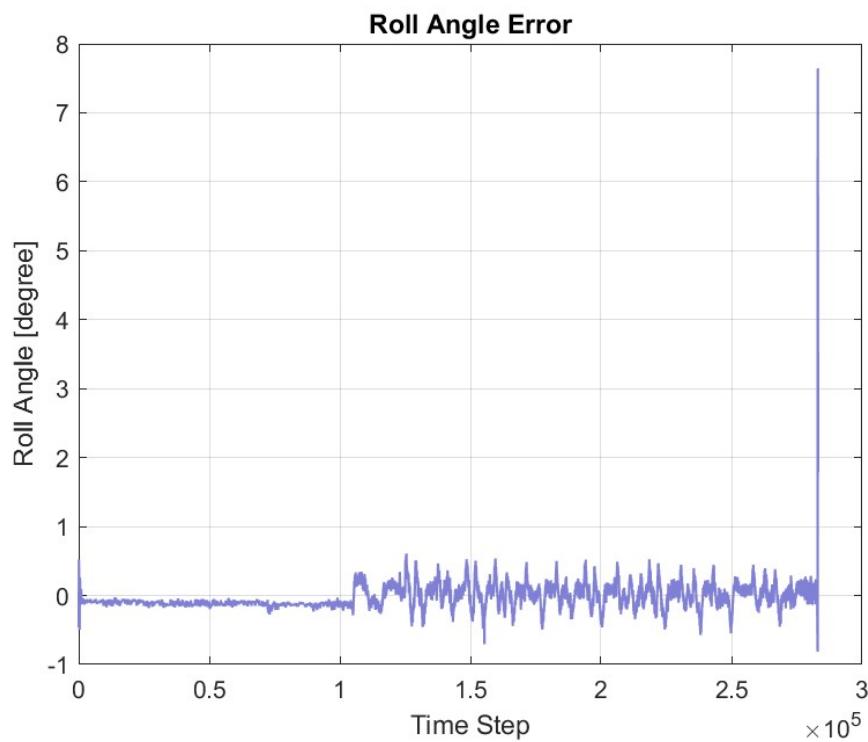


تصویر 20 کواریانس سرعت V_d

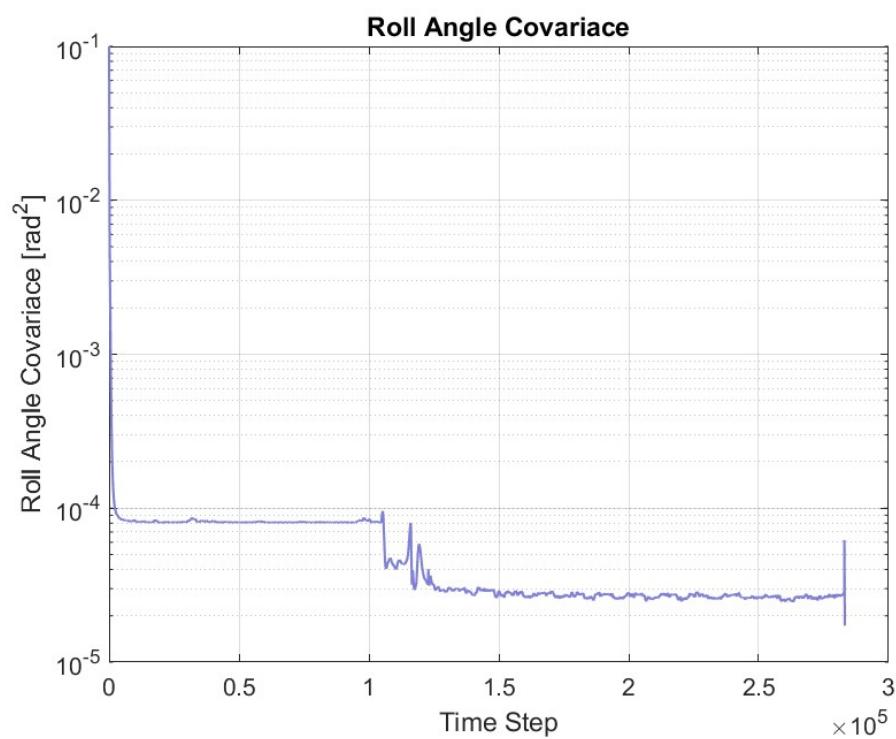
✓ نتایج شبیه سازی زاویه Roll



تصویر 21 مقایسه زوایه Roll

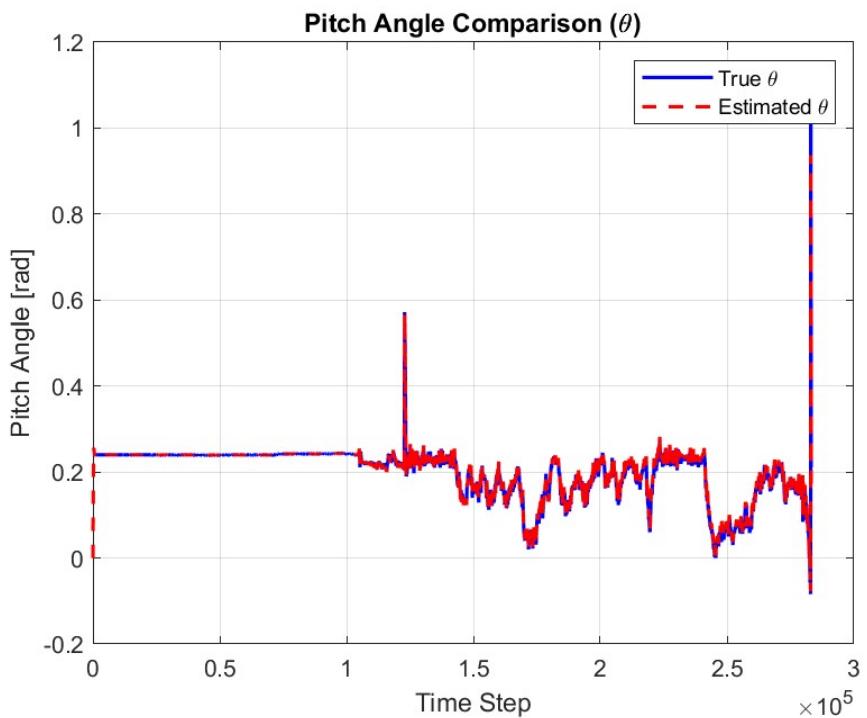


تصویر 22 خطای تخمین زاویه Roll

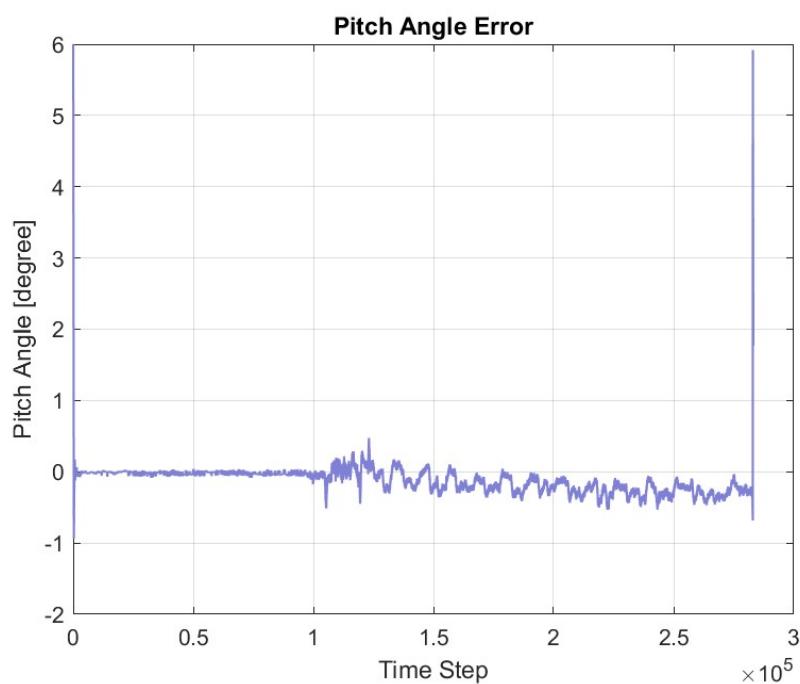


تصویر 23 کوواریانس زاویه Roll

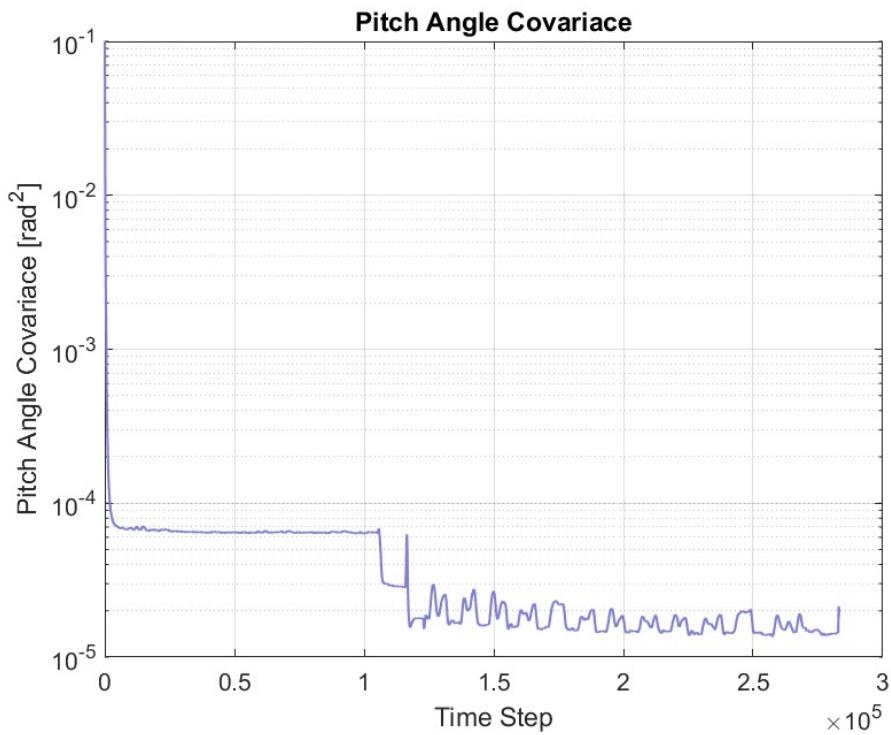
✓ نتایج شبیه سازی زاویه Pitch



تصویر 24 مقایسه زاویه Pitch

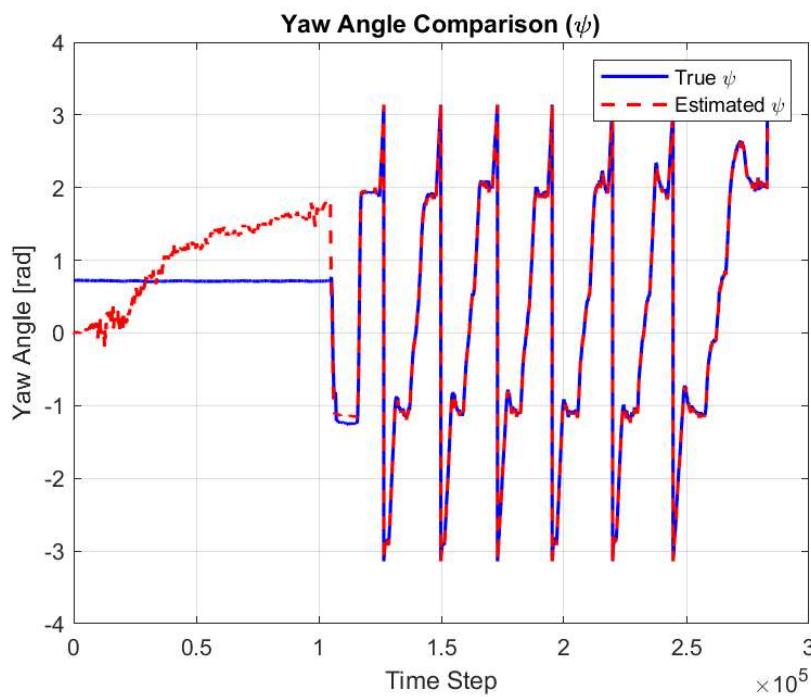


تصویر 25 خطای تخمین زاویه Pitch

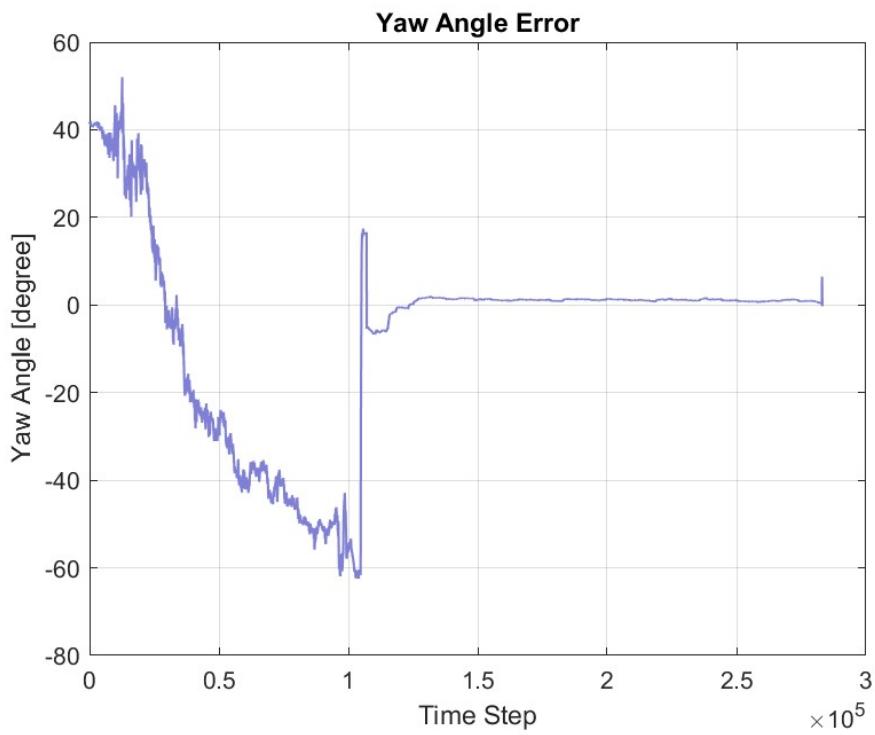


تصویر 26 کوواریانس زاویه Pitch

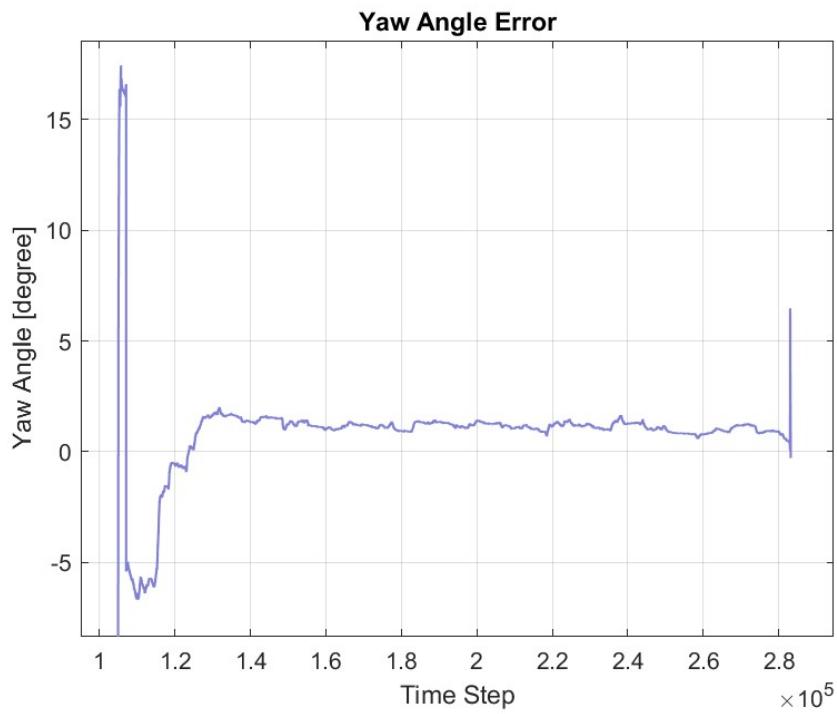
✓ نتایج شبیه سازی زاویه Yaw



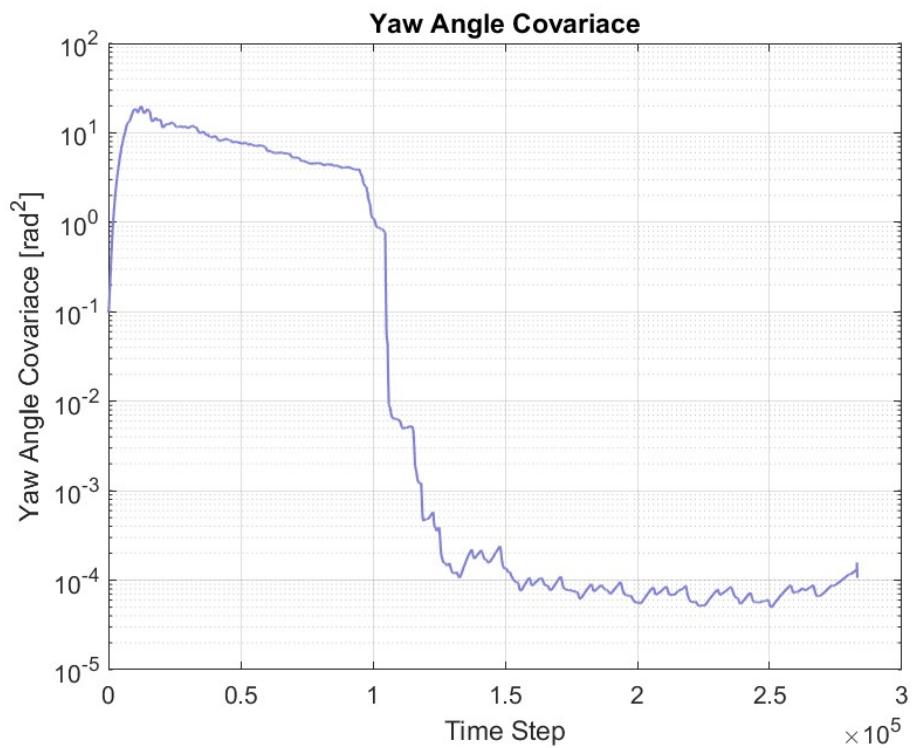
تصویر 27 مقایسه زاویه Yaw



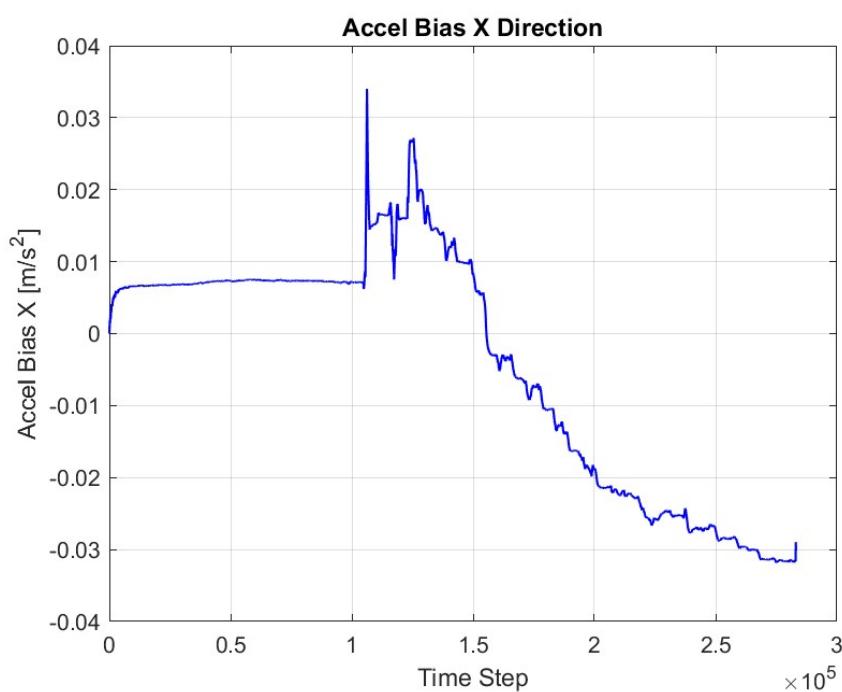
تصویر 28 خطای تخمین زاویه yaw



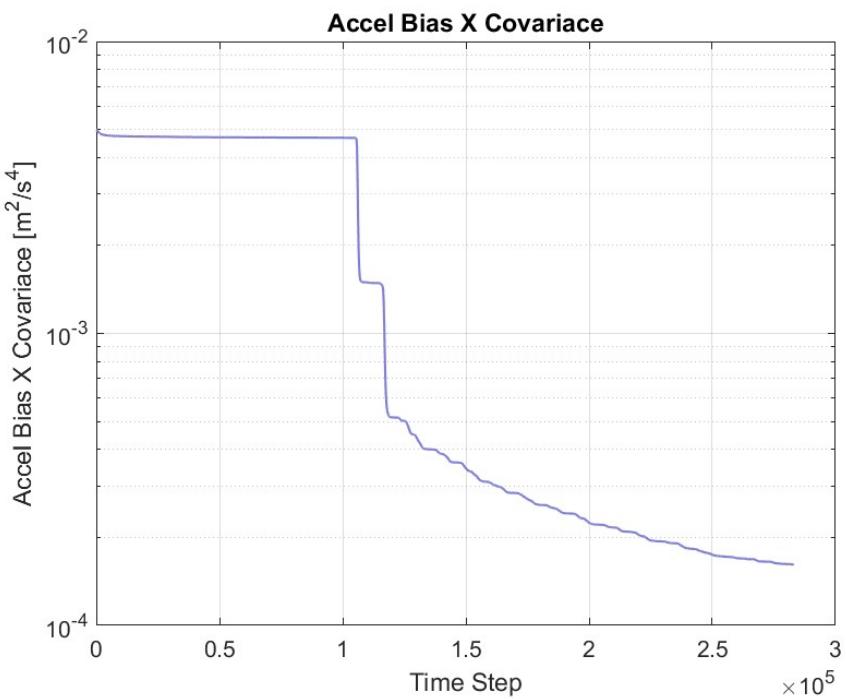
تصویر 29 نمای نزدیک خطای زاویه yaw که پس از همگرایی با یاس زایرو در محدوده منطقی قرار گرفته



✓ نتایج شبیه سازی بایاس شتاب سنج محور X

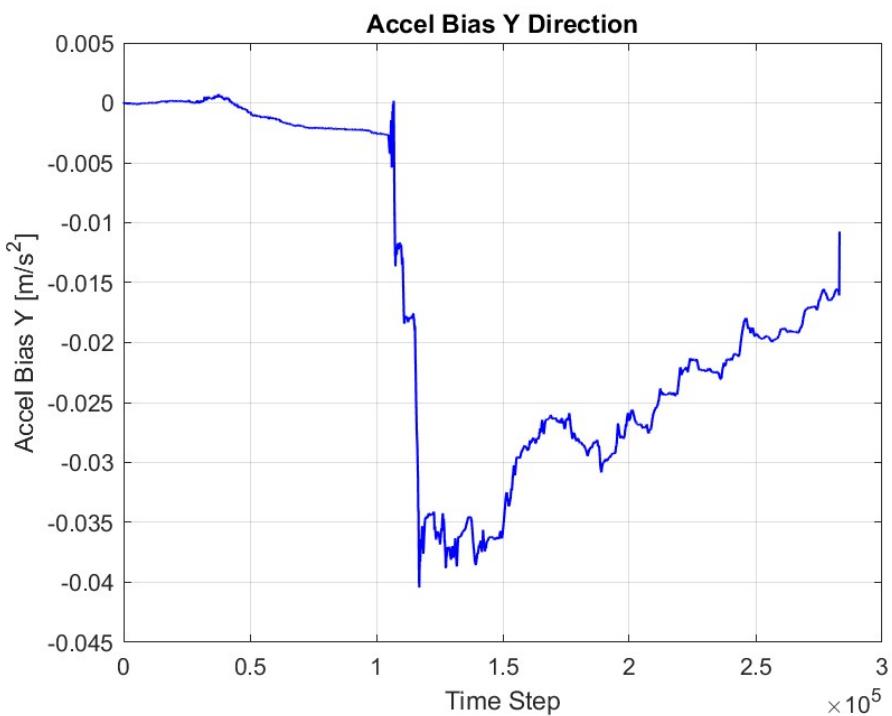


تصویر 31 بایاس شتاب سنج محور X

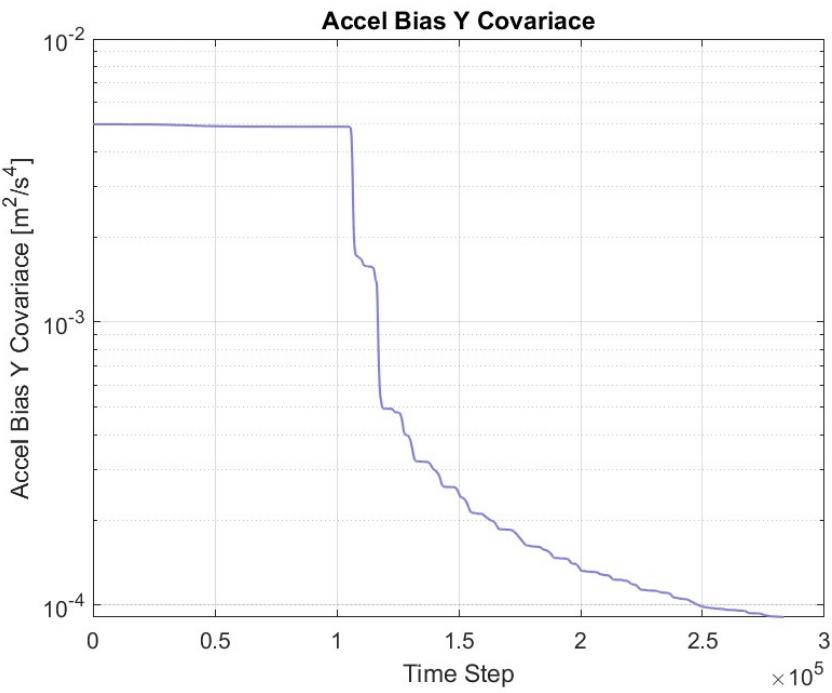


تصویر 32 کوواریانس بایاس شتاب سنج محور X

✓ نتایج شبیه سازی بایاس شتاب سنج محور Y

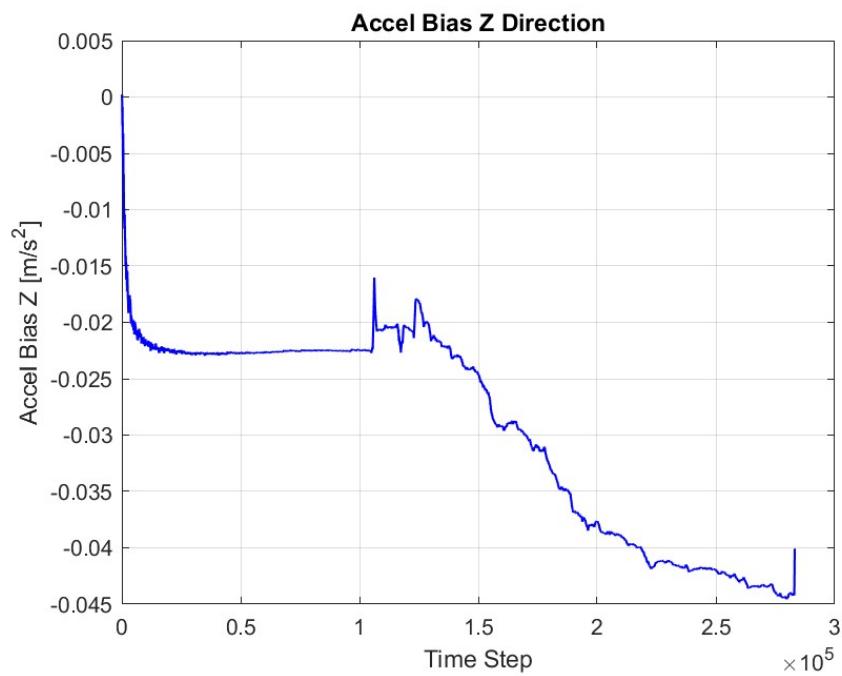


تصویر 33 بایاس شتاب سنج محور Y

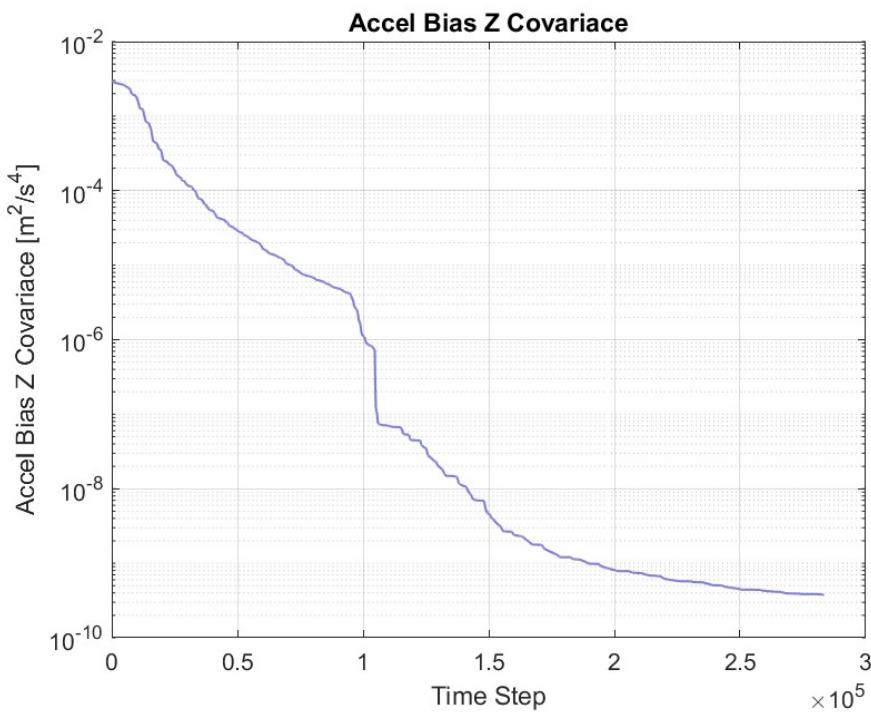


تصویر 34 کوواریانس بایاس شتاب سنج محور Y

✓ نتایج شبیه سازی بایاس شتاب سنج محور Z

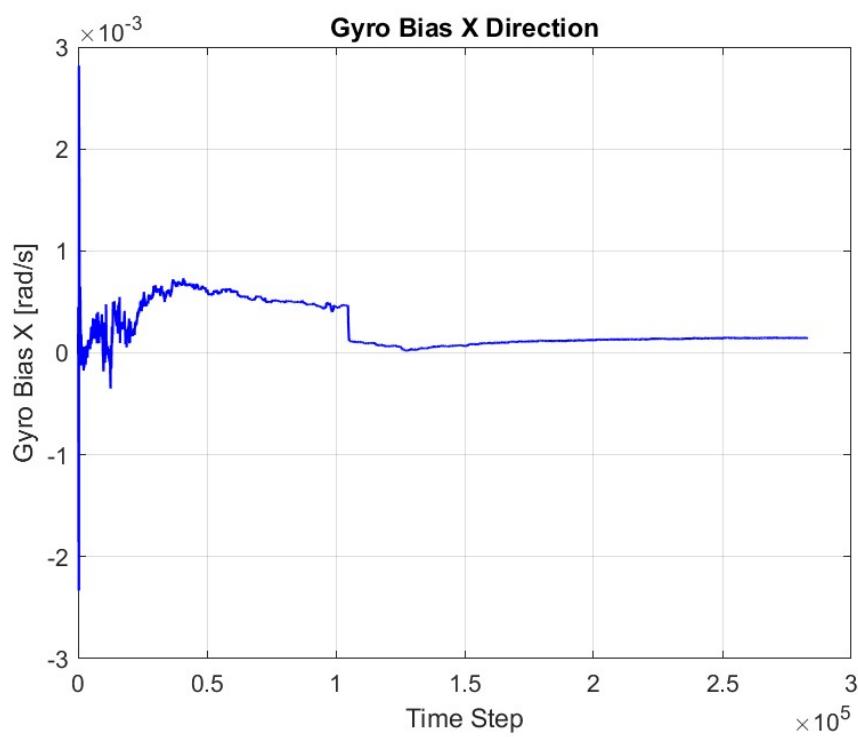


تصویر 35 بایاس شتاب سنج محور Z

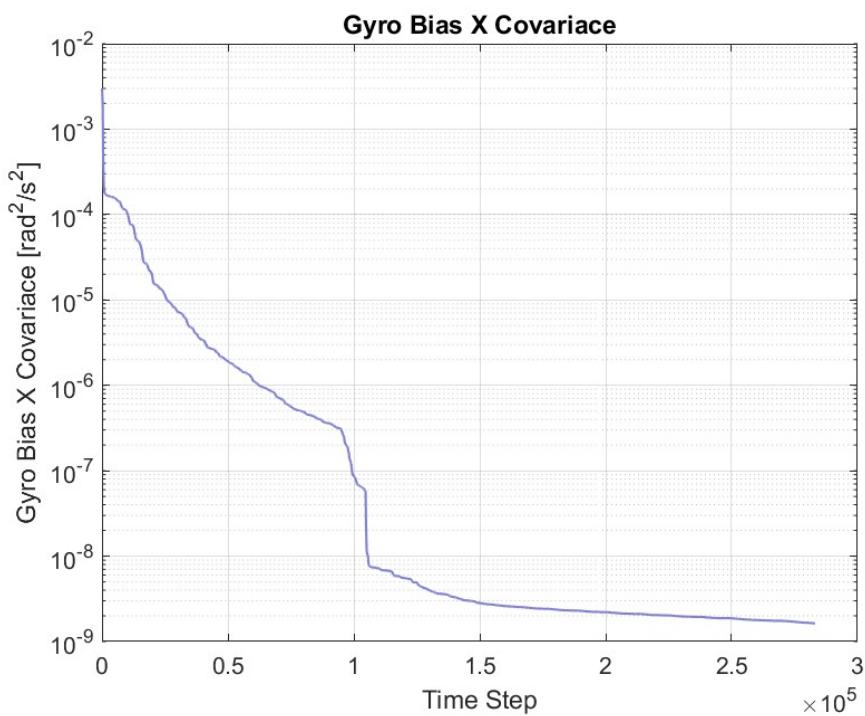


تصویر 36 کواریانس بایاس شتاب سنج محور Z

✓ نتایج شبیه سازی بایاس ژایرو محور X

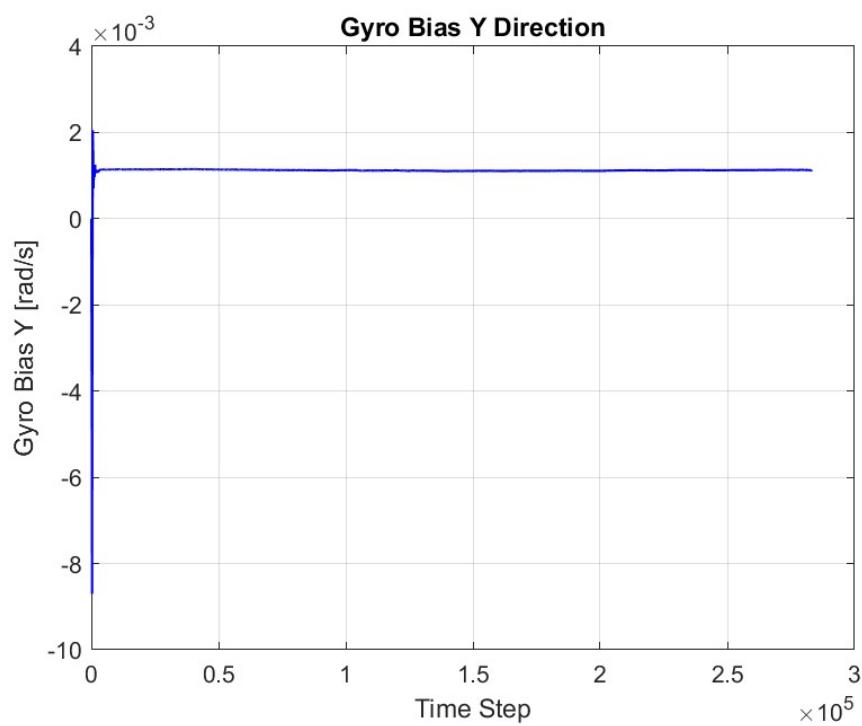


تصویر 37 بایاس ژایرو محور X

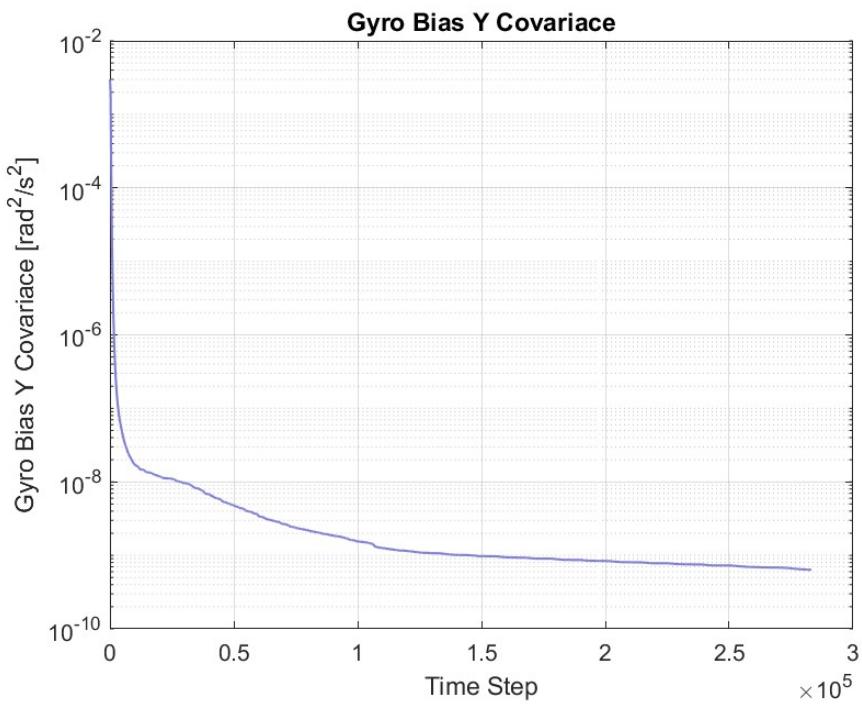


تصویر 38 کوواریانس بایاس ژایرو محور X

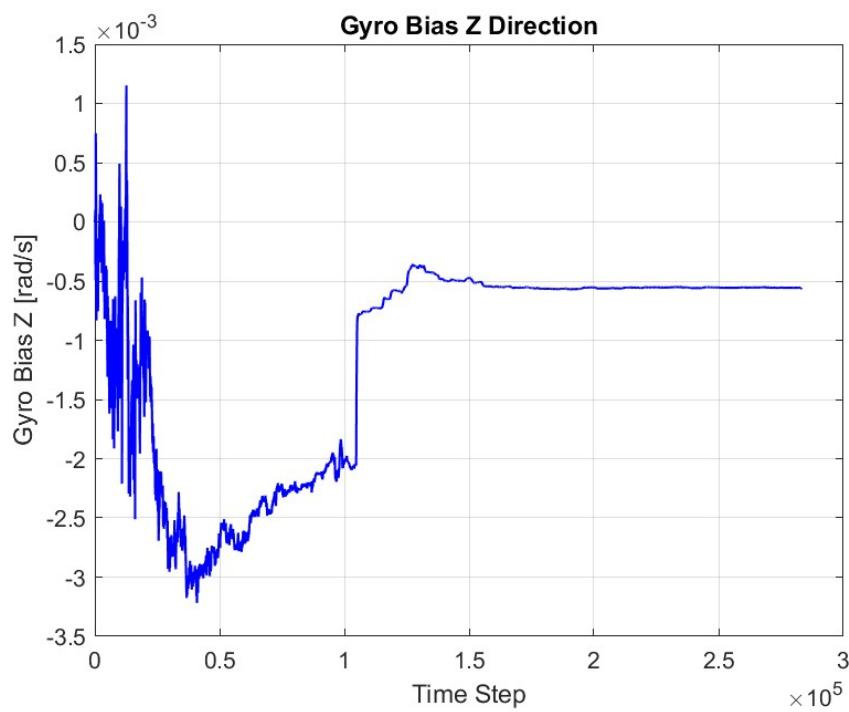
✓ نتایج شبیه سازی بایاس ژایرو محور Y

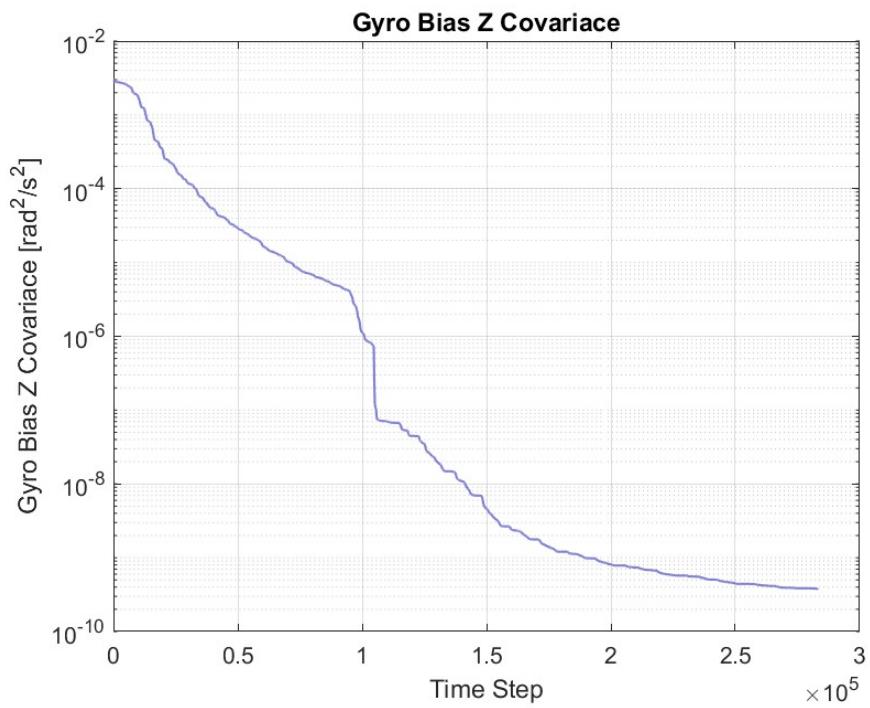


تصویر 39 بایاس ژایرو محور Y



✓ نتایج شبیه سازی باياس ژايو محور Z



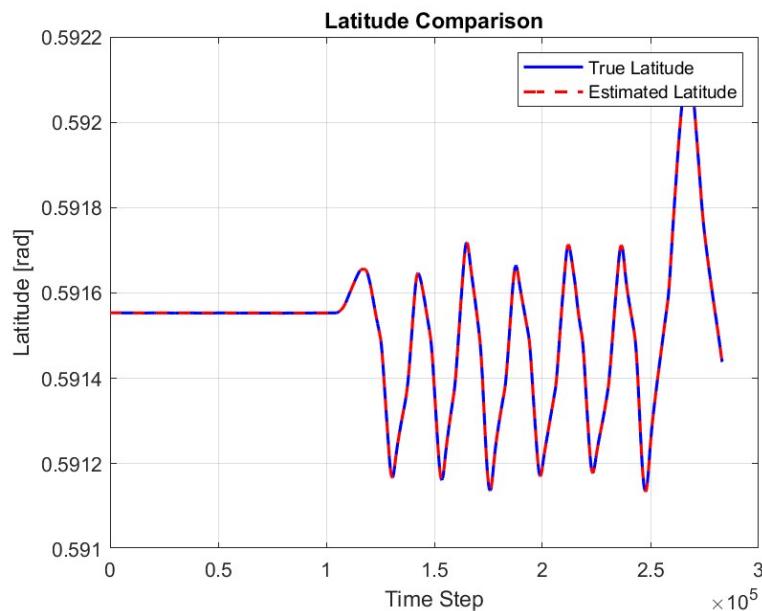


تصویر 42 کوواریانس بایاس زاویه محور Z

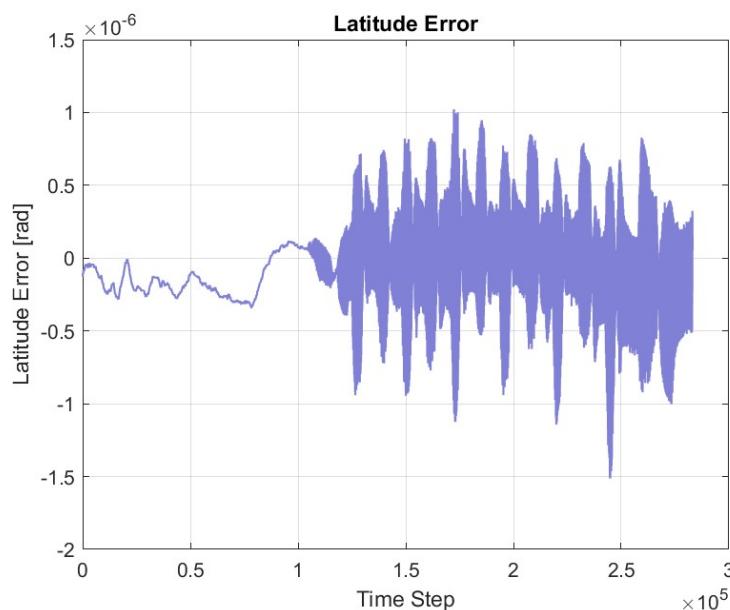
✓ نتایج شبیه سازی فیلتر کالمن غیر مستقیم

نتایج شبیه سازی برای فیلتر کالمن غیر مستقیم به صورت زیر بوده است. لازم به ذکر است که در این قسمت نتایج 0.1 ثانیه (معادل هر 10 گام گسسته) انجام شده است.

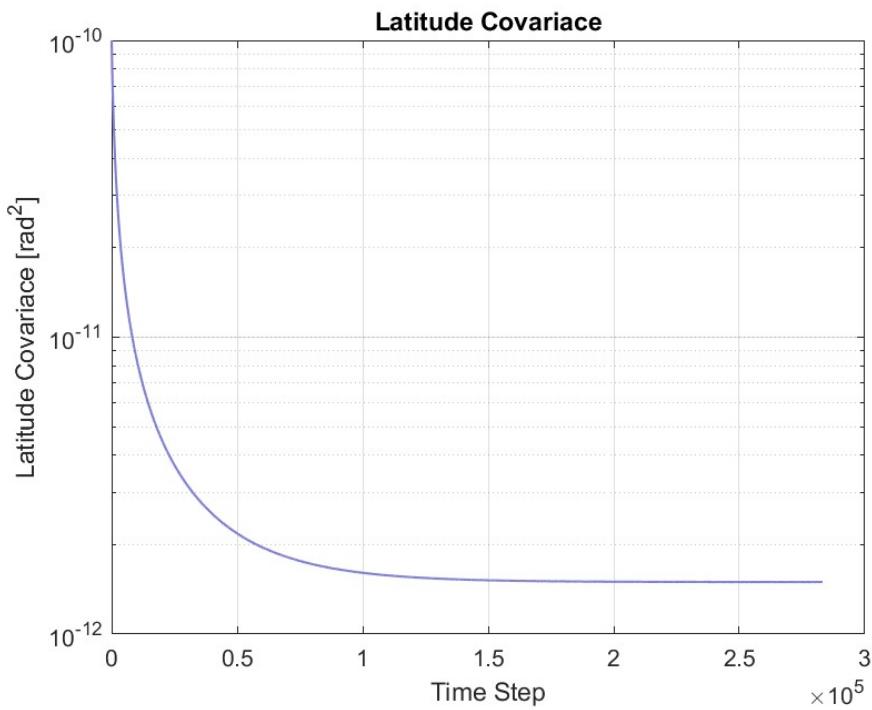
✓ نتایج زاویه Latitude



تصویر 43 مقایسه زوایه Latitude

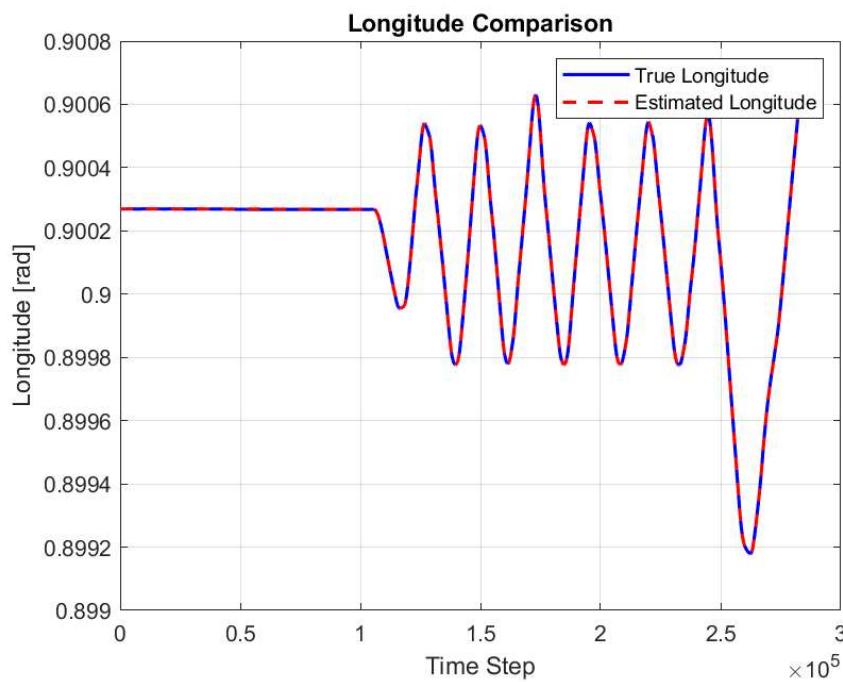


تصویر 44 خطای تخمین زوایه Latitude

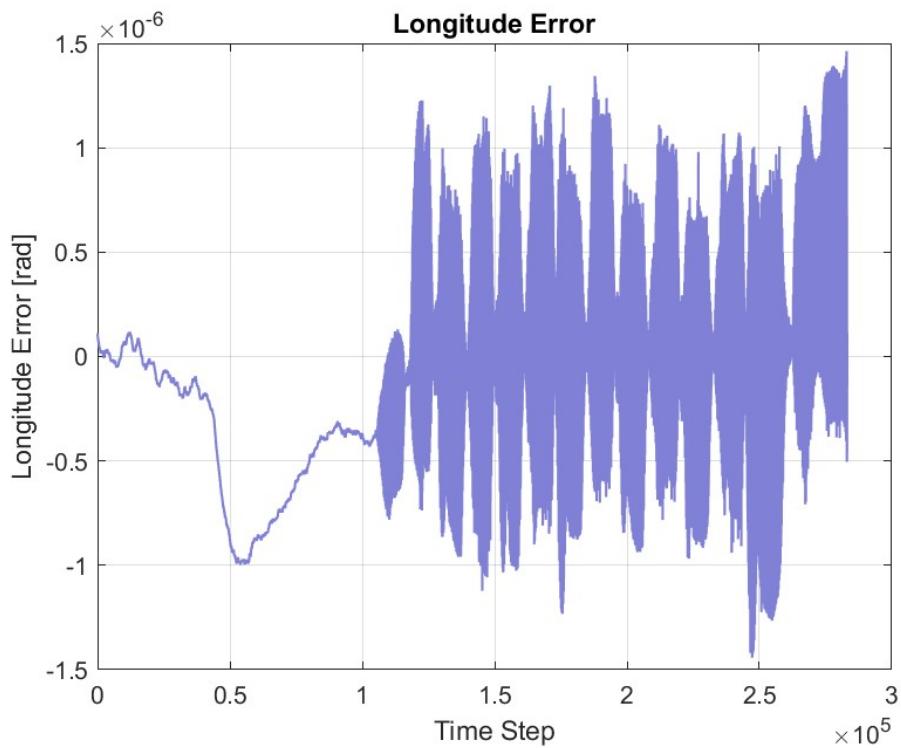


تصویر 45 کوواریانس زاویه Latitude

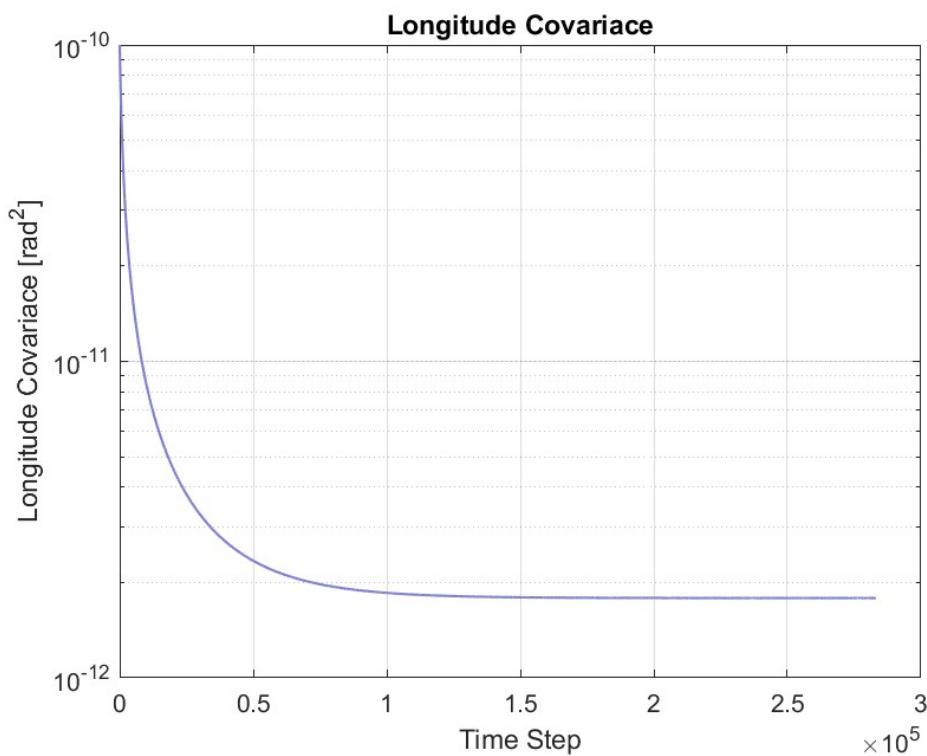
✓ نتایج زاویه Longitude



تصویر 46 مقایسه زاویه Longitude

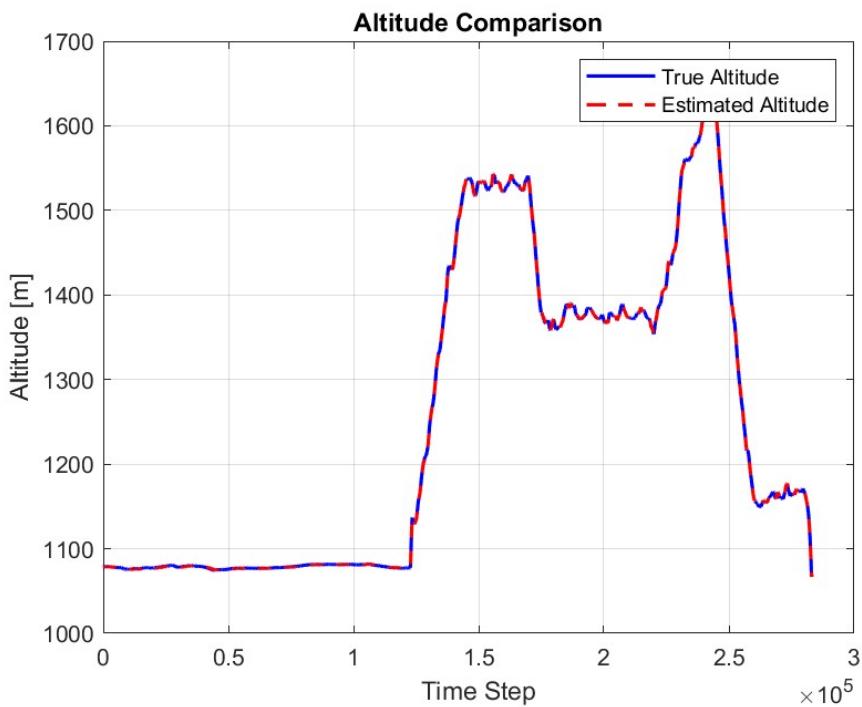


تصویر 47 خطای تخمین زاویه Longitude

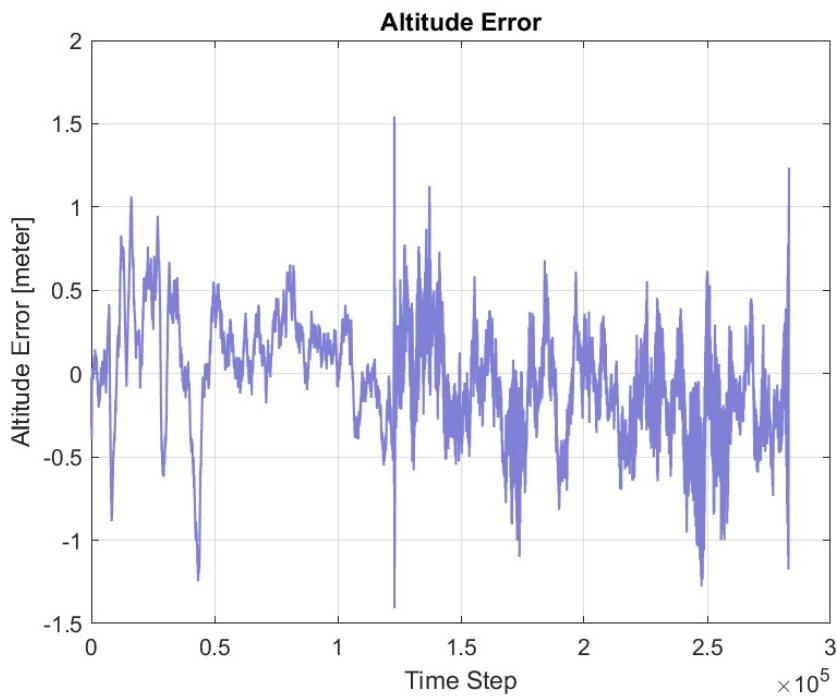


تصویر 48 کوواریانس زاویه Longitude

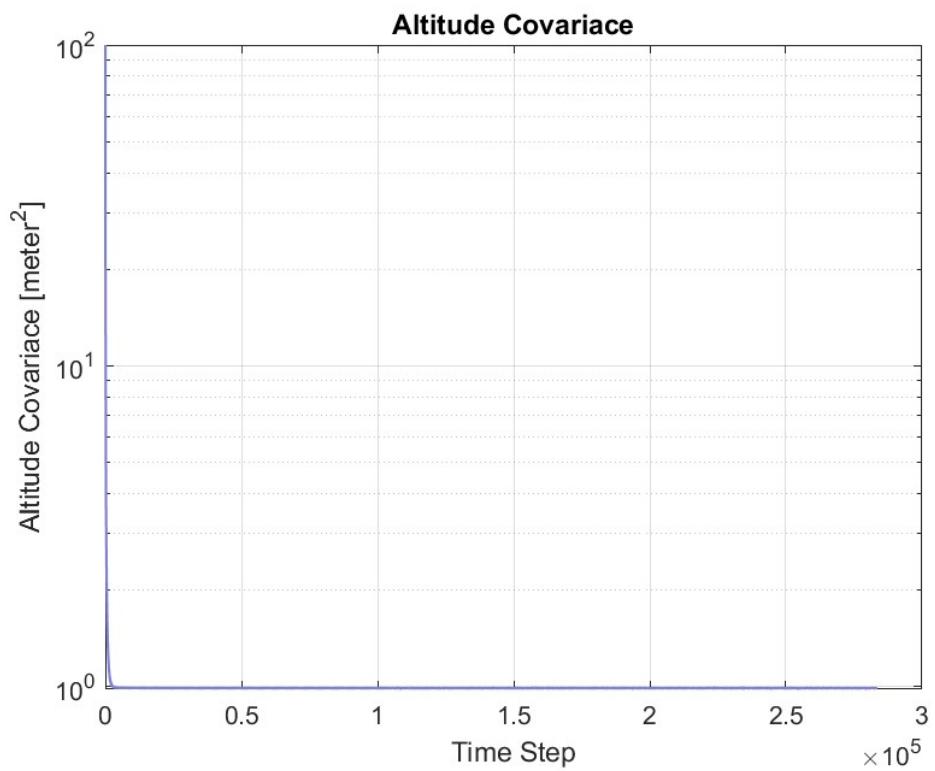
✓ نتایج تخمین ارتفاع



تصویر 49 مقایسه ارتفاع



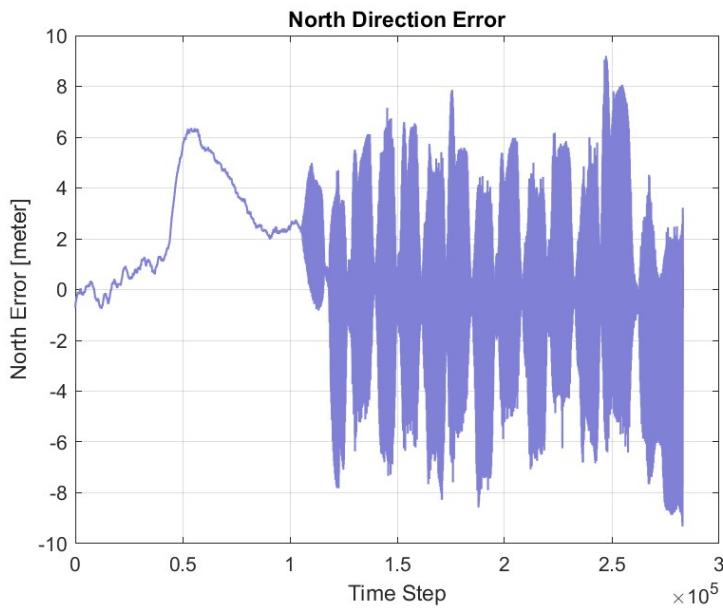
تصویر 50 خطای تخمین ارتفاع



تصویر 51 کوواریانس ارتفاع

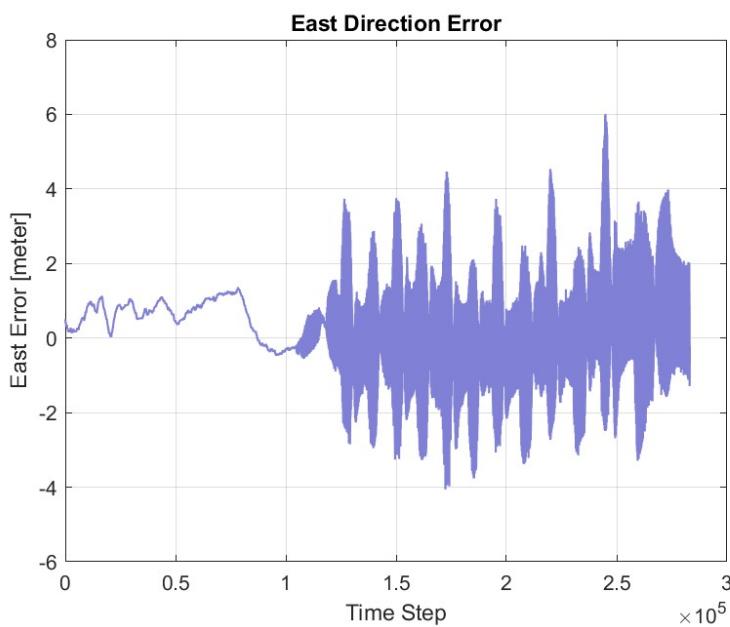
✓ نتایج خطای تخمین در جهت شمال

همانطور که مشاهده میشود علارغم تنظیم ضرایب فیلتر کالمن به علت خطای بالای موجود در راستای طول جغرافیایی سنسور GPS متاسفانه فیلتر خطای زیادی در تخمین موقعیت در راستای شمالی داشته است.



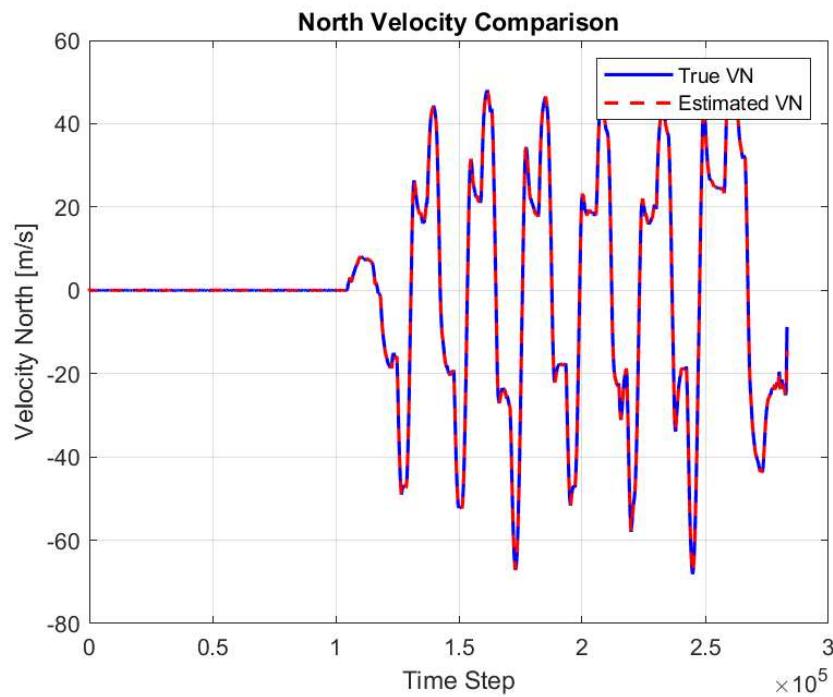
تصویر 52 خطای تخمین موقعیت در راستای شمال

✓ نتایج خطای تخمین در جهت شرق

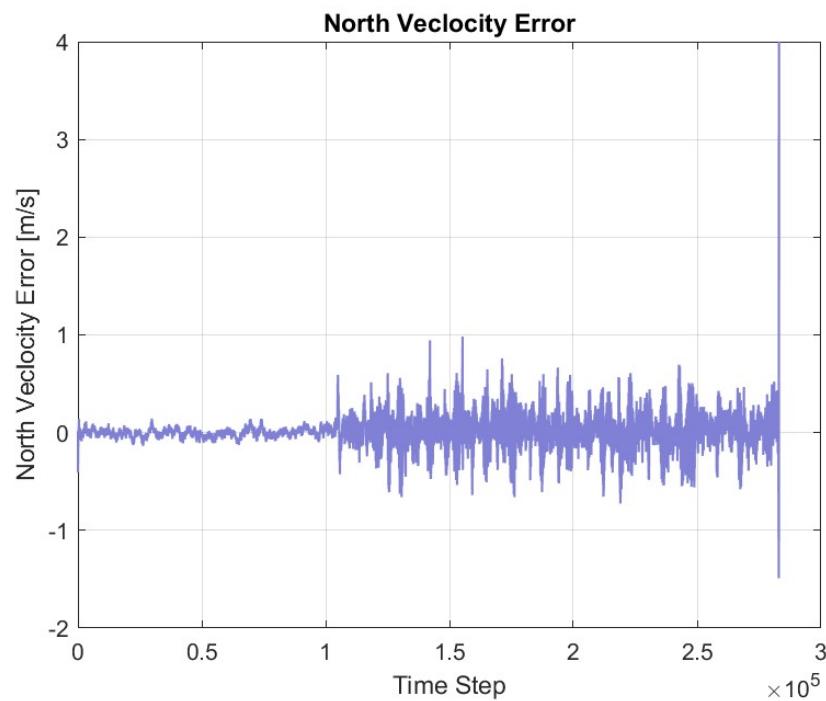


تصویر 53 خطای تخمین موقعیت در راستای شرق

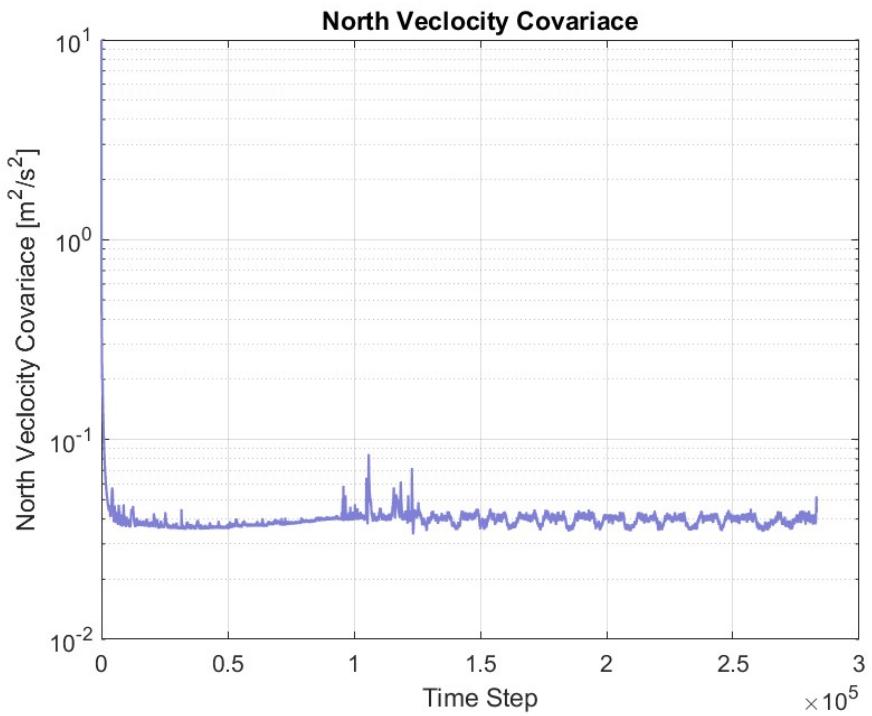
✓ نتایج تخمین سرعت V_n



تصویر 54 مقایسه سرعت V_n

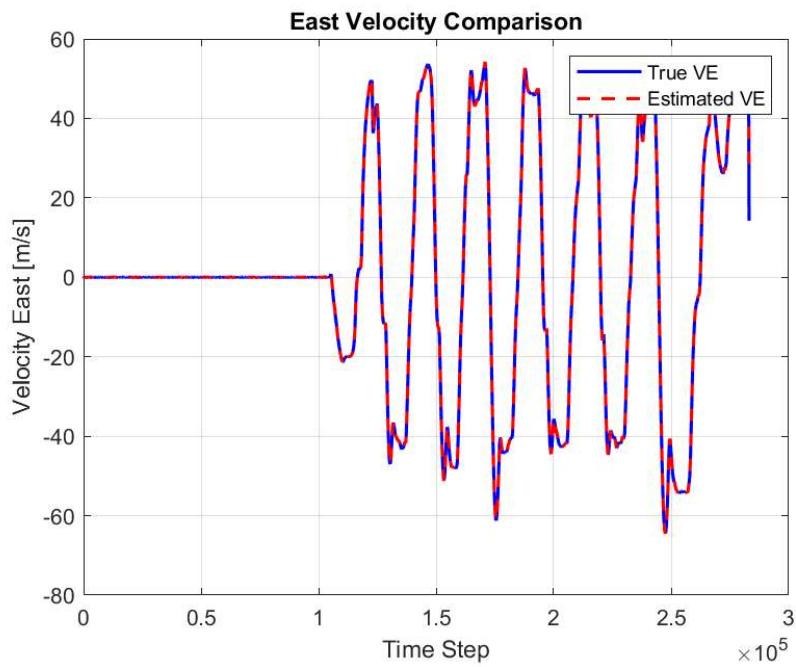


تصویر 55 خطای تخمین سرعت V_n

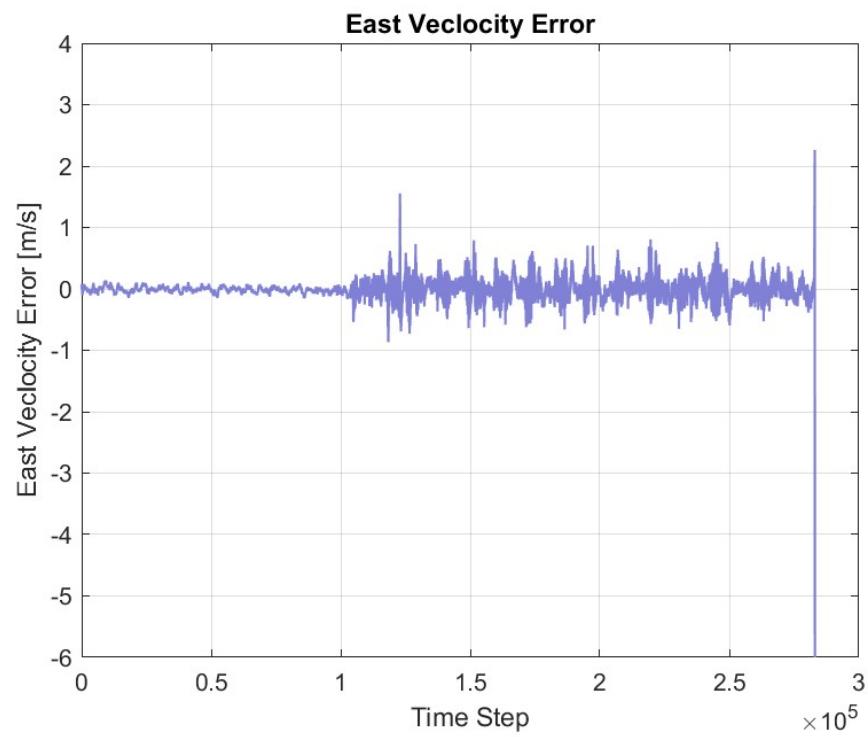


تصویر 56 کواریانس سرعت V_n

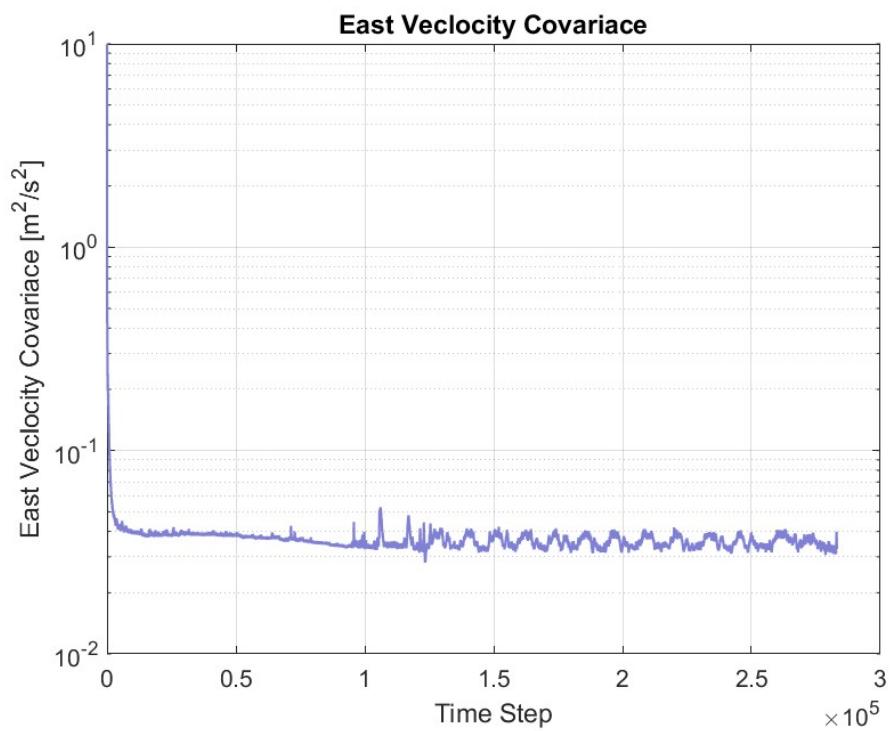
✓ نتایج تخمین سرعت V_e



تصویر 57 مقایسه سرعت V_e

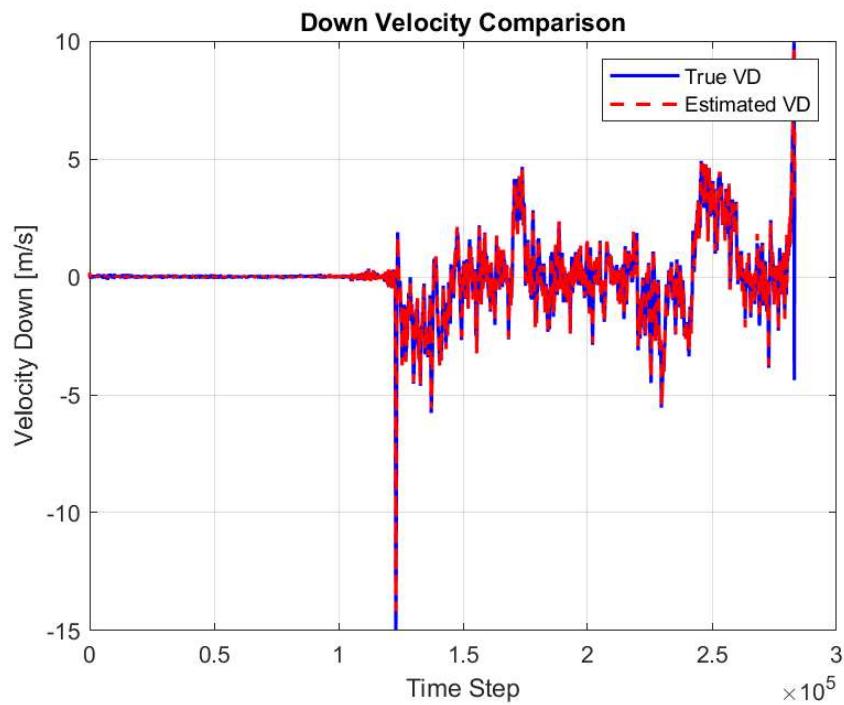


تصویر 58 خطای تخمین سرعت V_e

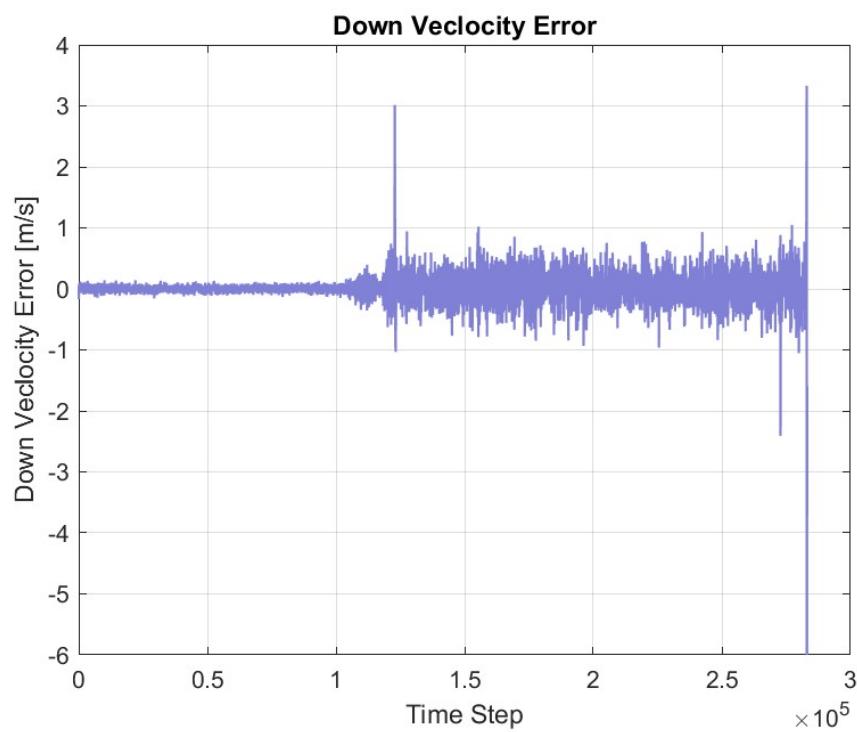


تصویر 59 کوواریانس سرعت V_e

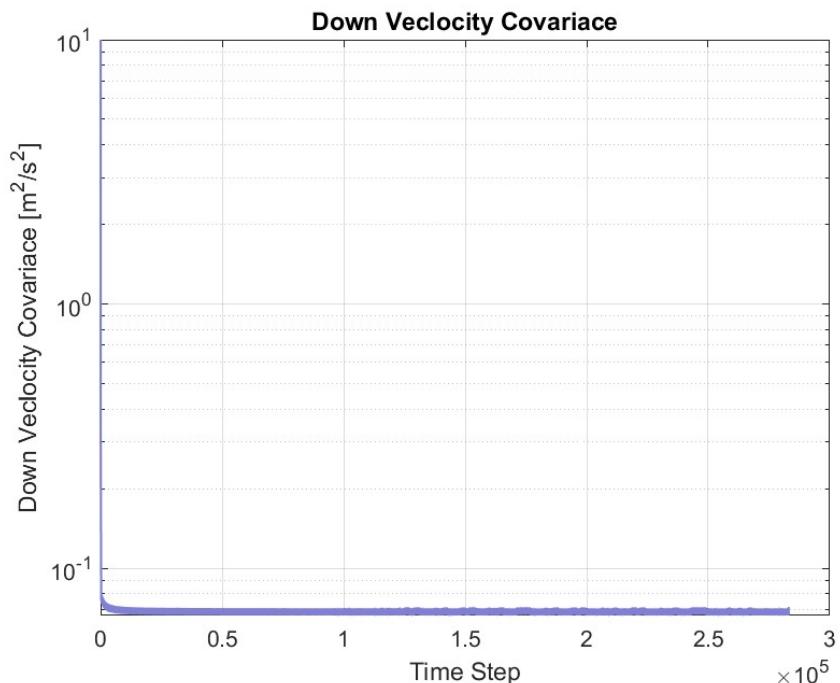
✓ نتایج تخمین سرعت V_d



تصویر 60 مقایسه سرعت V_d

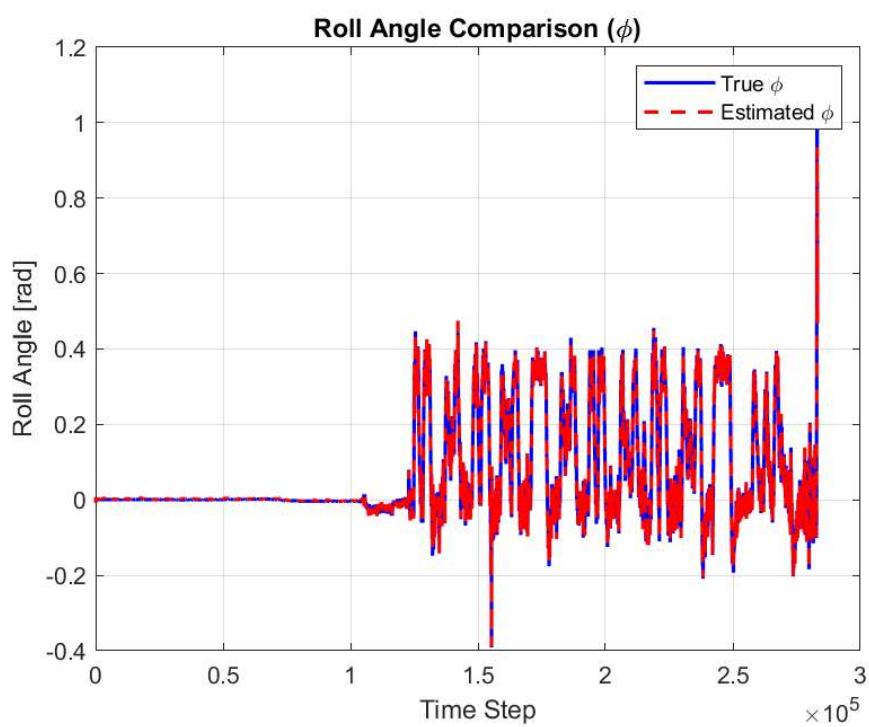


تصویر 61 خطای تخمین سرعت V_d

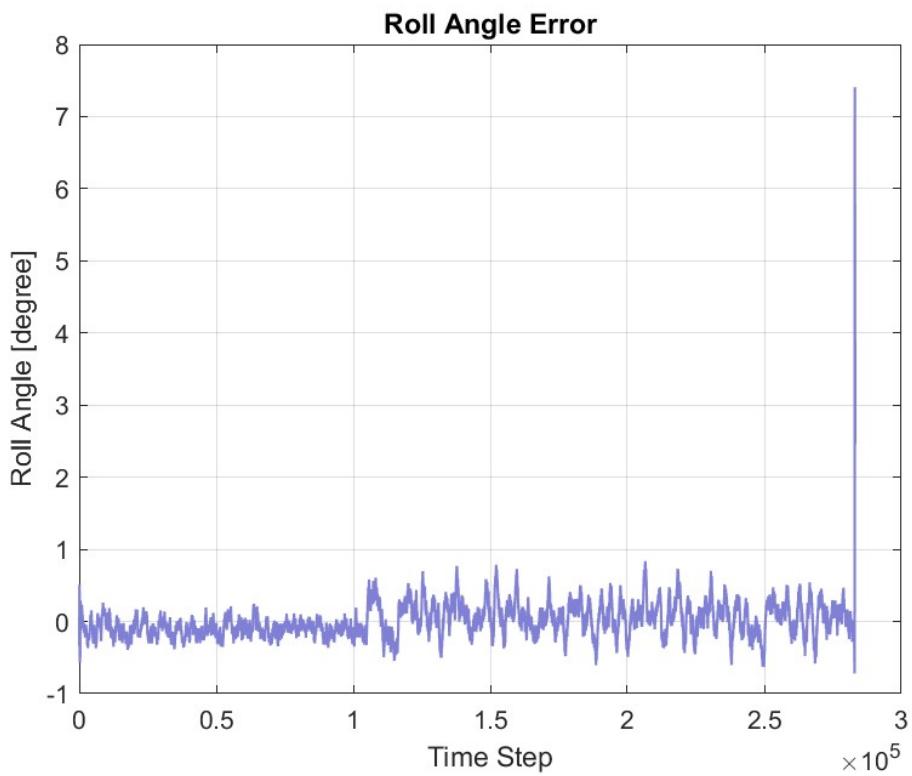


تصویر 62 کوواریانس سرعت V_d

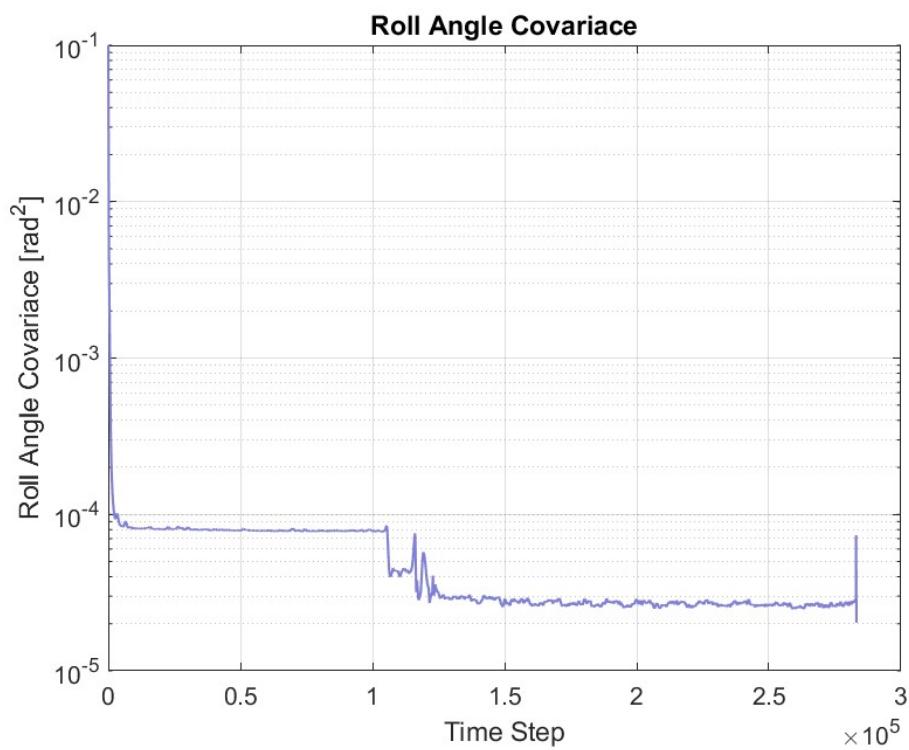
✓ نتایج شبیه سازی زاویه Roll



تصویر 63 مقایسه زوایه Roll

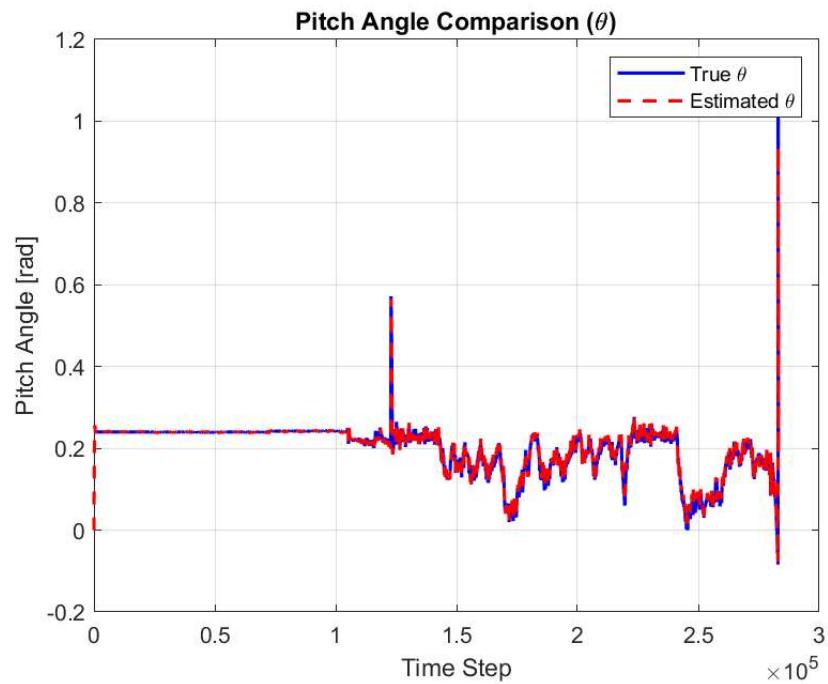


تصویر 64 خطای تخمین زاویه Roll

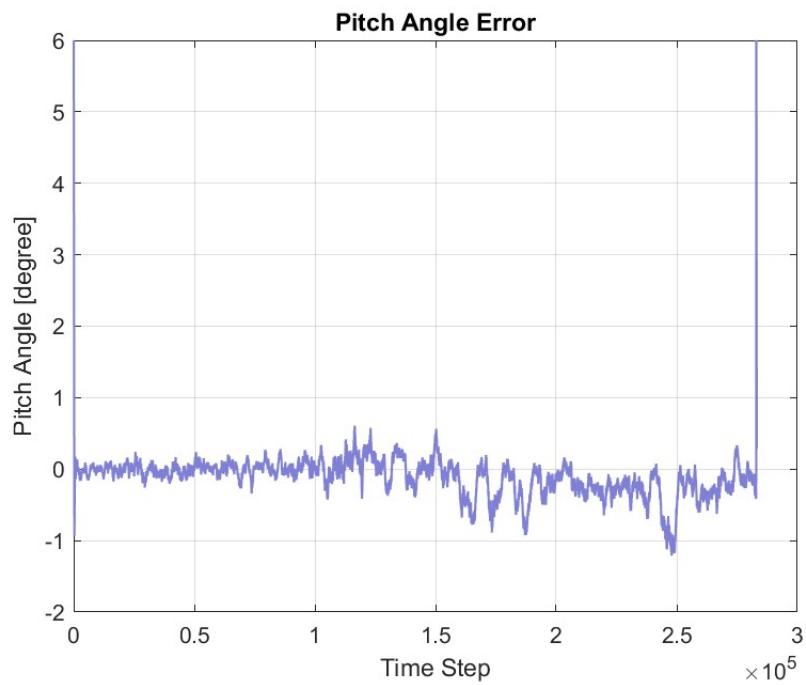


تصویر 65 کوواریانس زاویه Roll

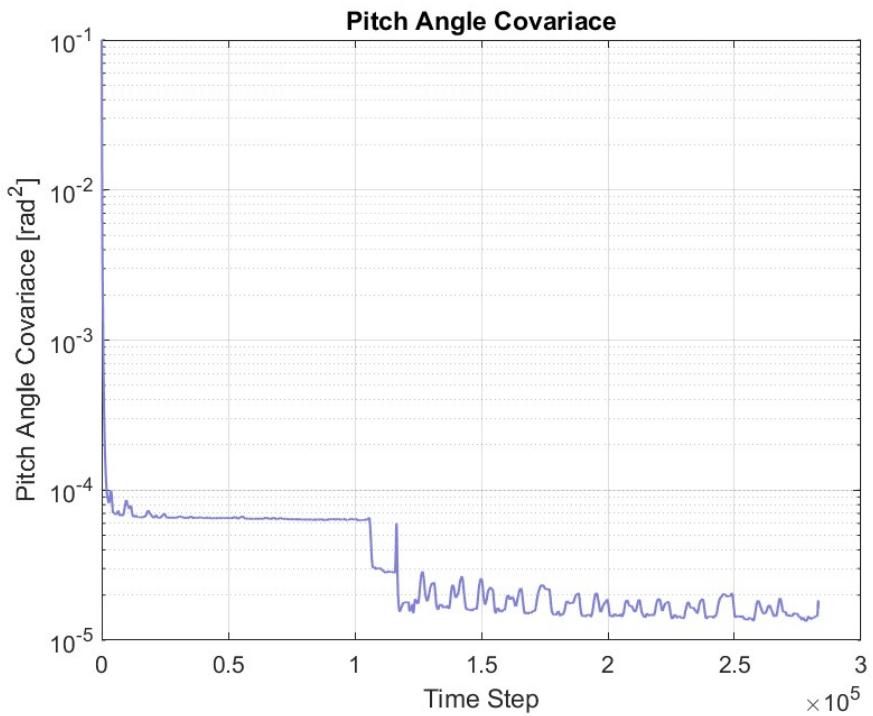
✓ نتایج شبیه سازی زاویه Pitch



تصویر ۶۶ متابعه زاویه Pitch

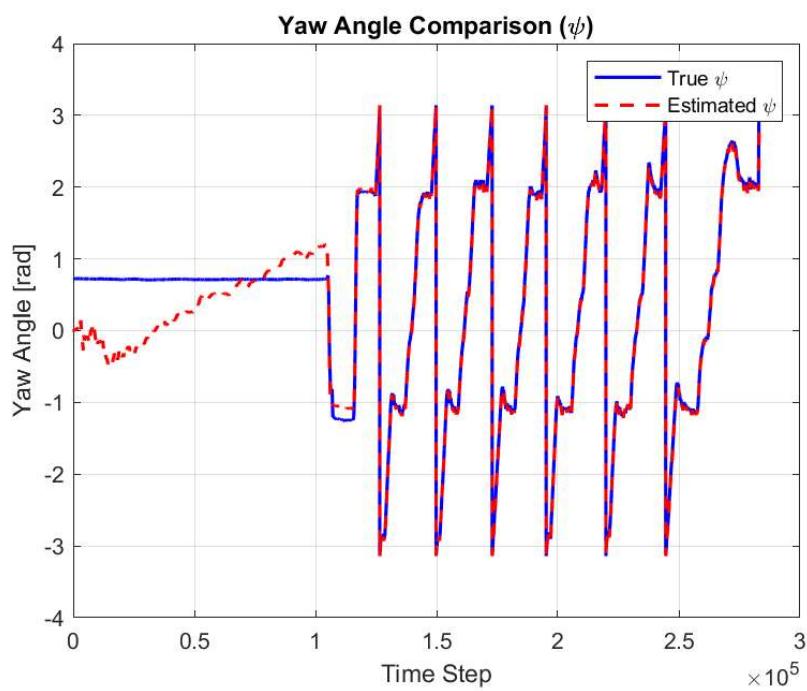


تصویر ۶۷ خطای تخمین زاویه Pitch

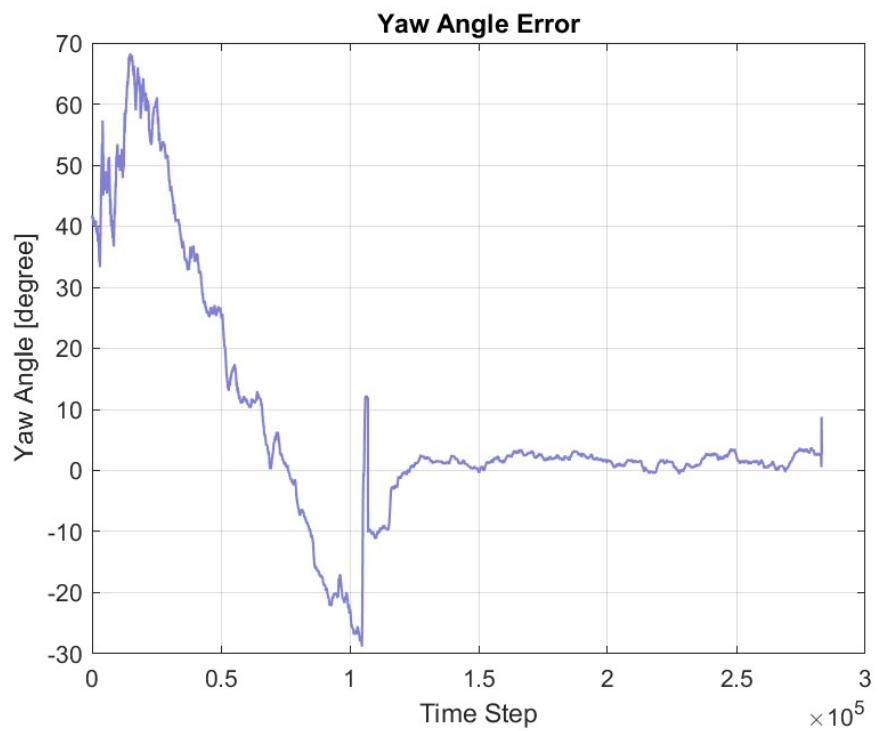


تصویر 68 کوواریانس زاویه Pitch

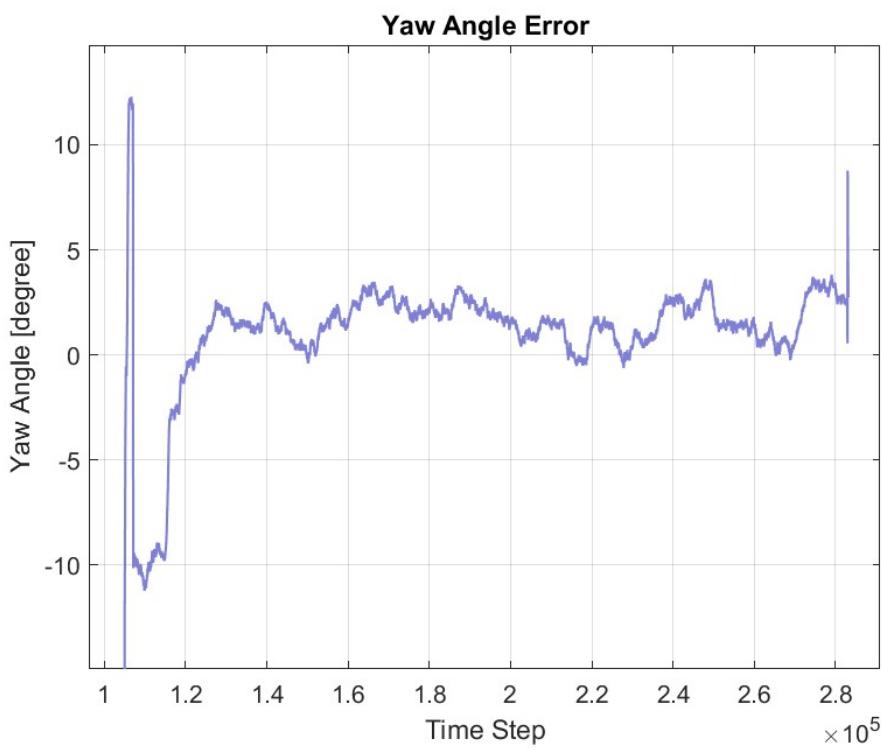
✓ نتایج شبیه سازی زاویه Yaw



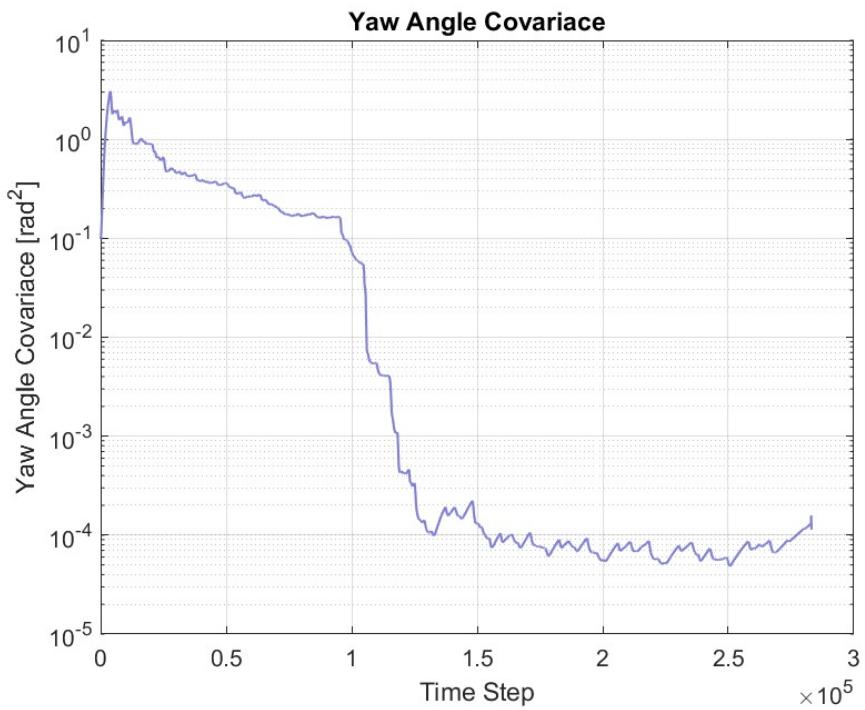
تصویر 69 مقایسه زوایه Yaw



تصویر 70 خطای تخمین زاویه yaw

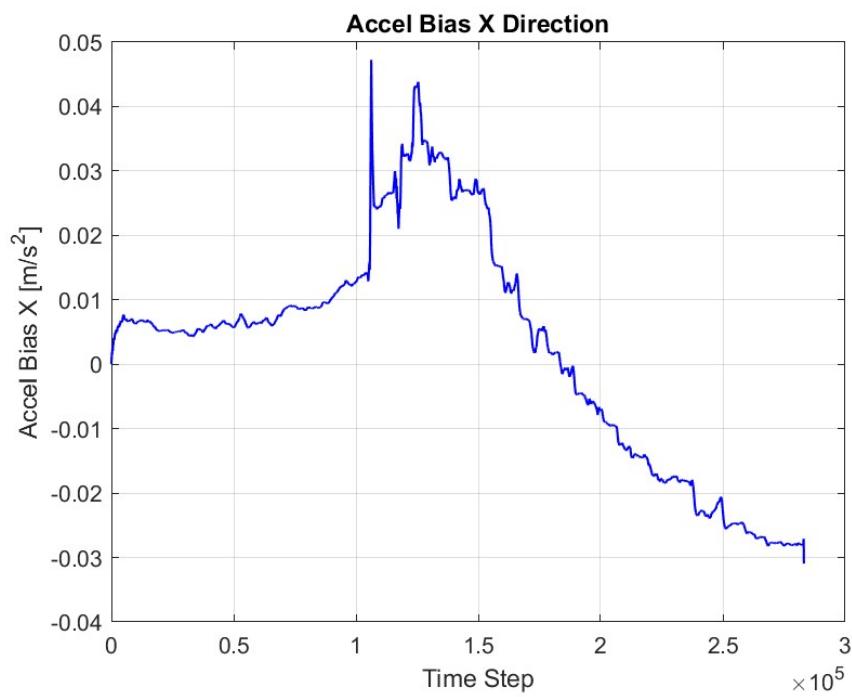


تصویر 71 نمای نزدیک از خطای زاویه yaw پس از همگرایی بایاس های زایرو

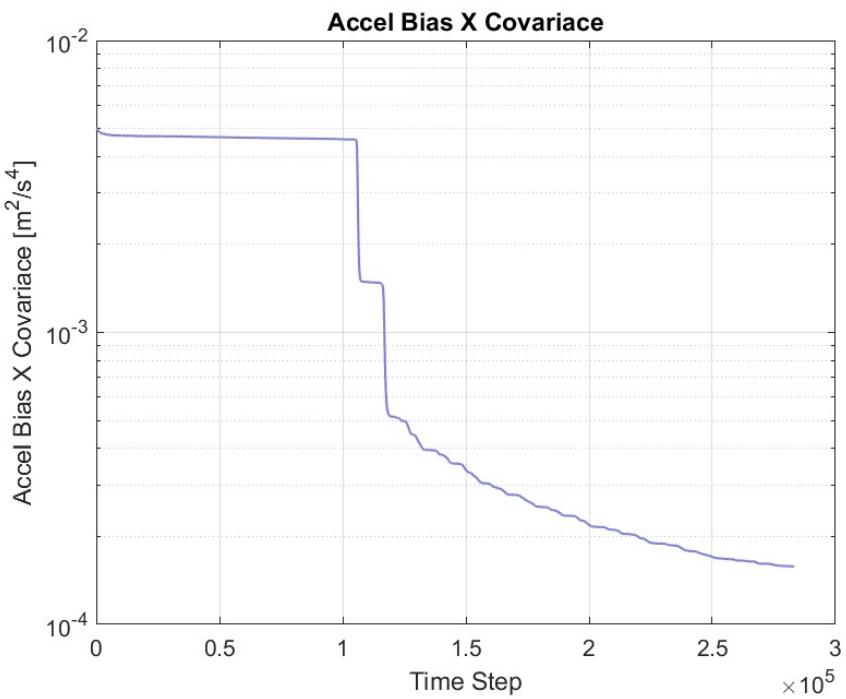


تصویر 72 کوواریانس زاویه Yaw

✓ نتایج شبیه سازی بایاس شتاب سنج محور X

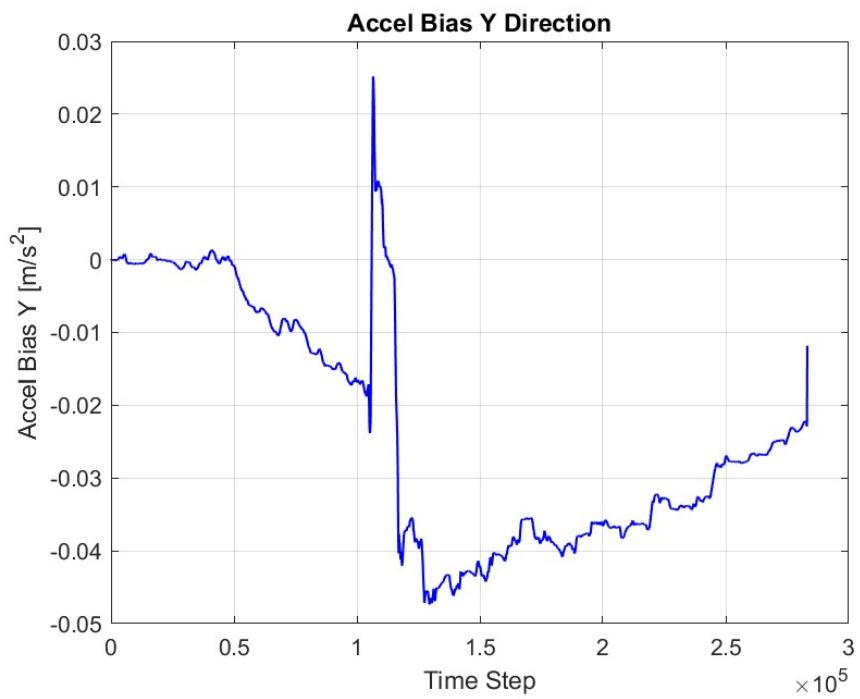


تصویر 73 بایاس شتاب سنج محور X

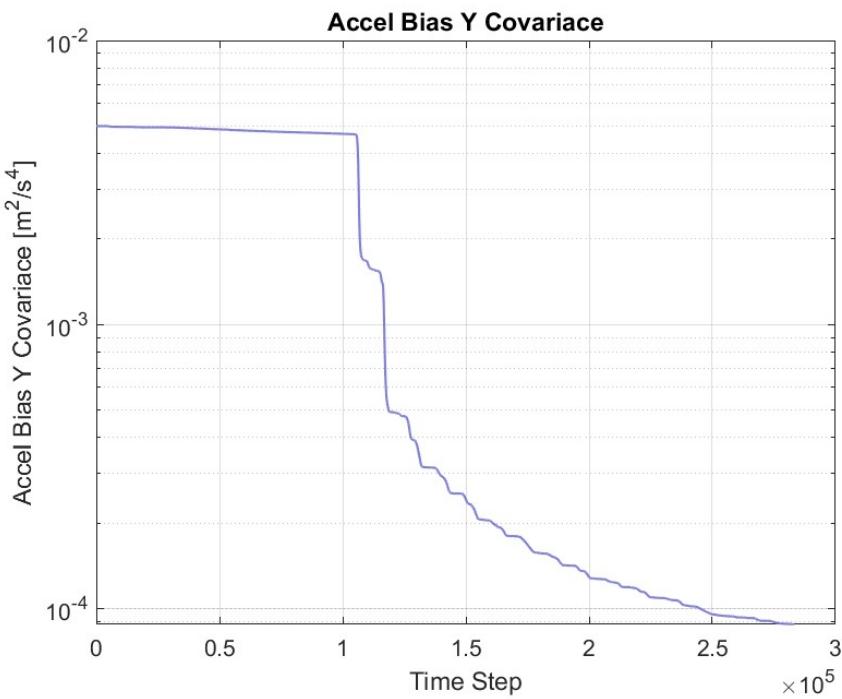


تصویر 74 کوواریانس بایاس شتاب سنج محور X

✓ نتایج شبیه سازی بایاس شتاب سنج محور Y

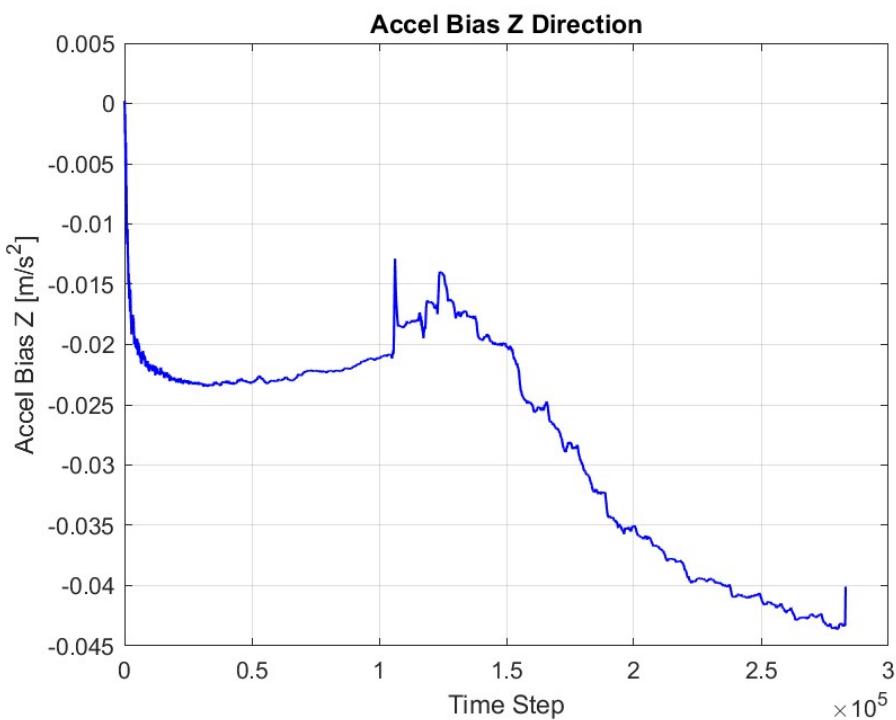


تصویر 75 بایاس شتاب سنج محور Y

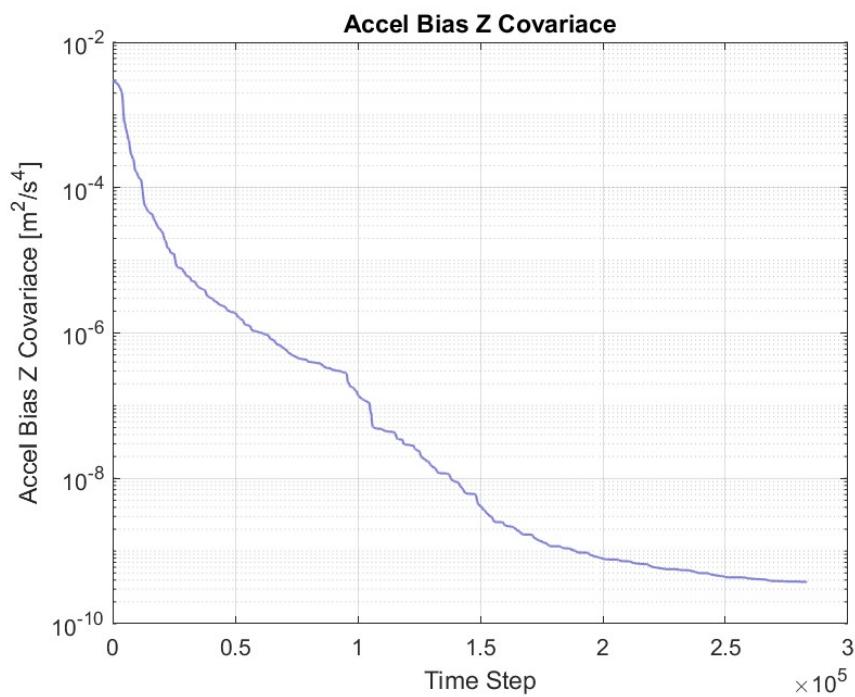


تصویر 76 کوواریانس بایاس شتاب سنج محور Y

✓ نتایج شبیه سازی بایاس شتاب سنج محور Z

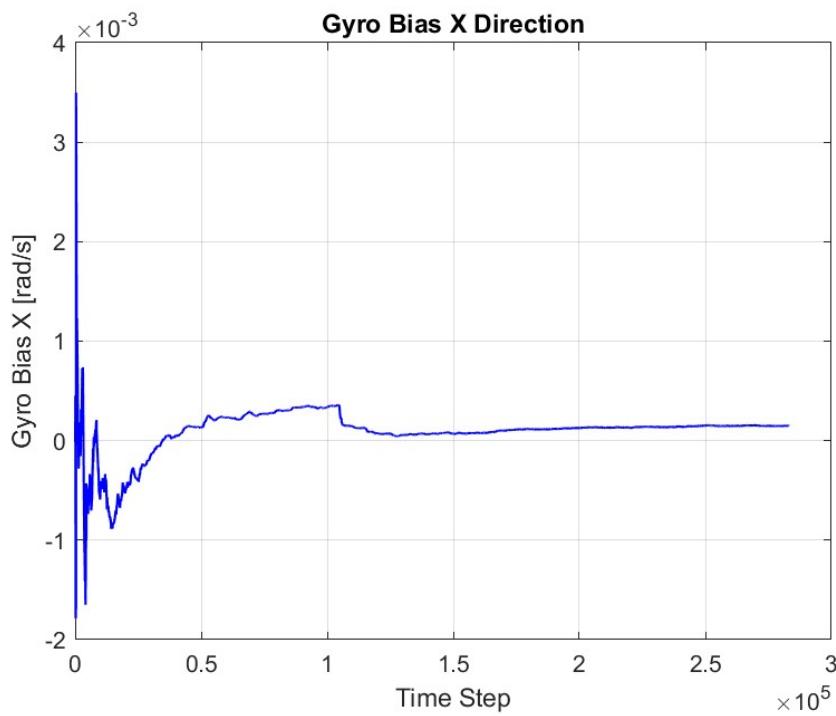


تصویر 77 بایاس شتاب سنج محور Z

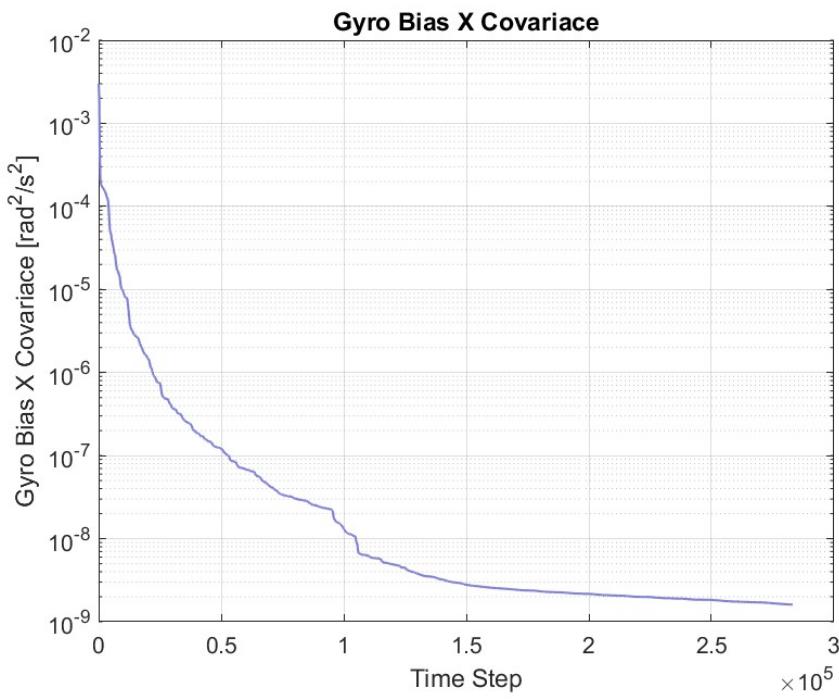


تصویر 78 کواریانس بایاس شتاب سنج محور Z

✓ نتایج شبیه سازی بایاس ژایرو محور X

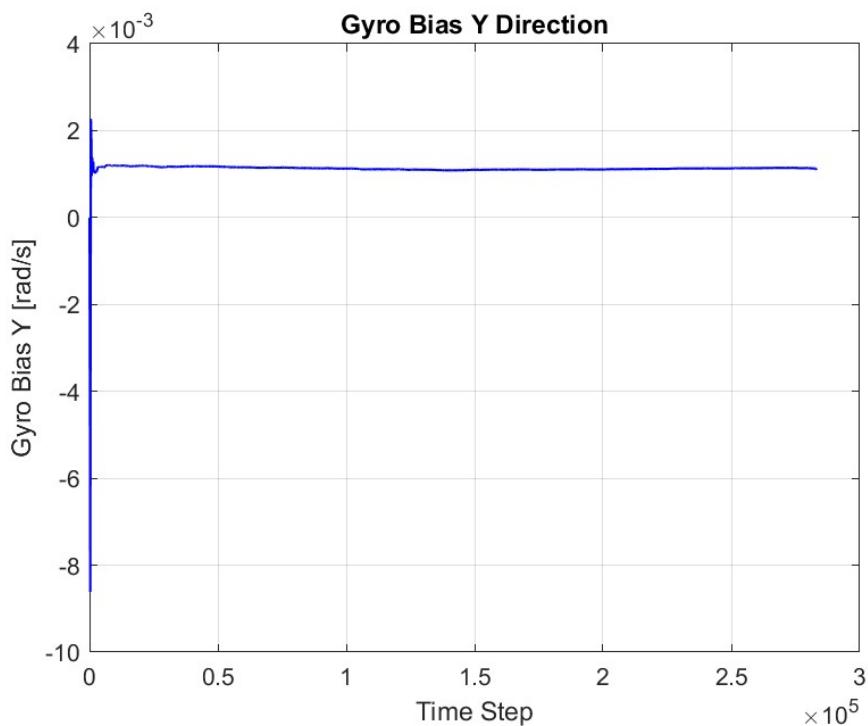


تصویر 79 بایاس ژایرو محور X

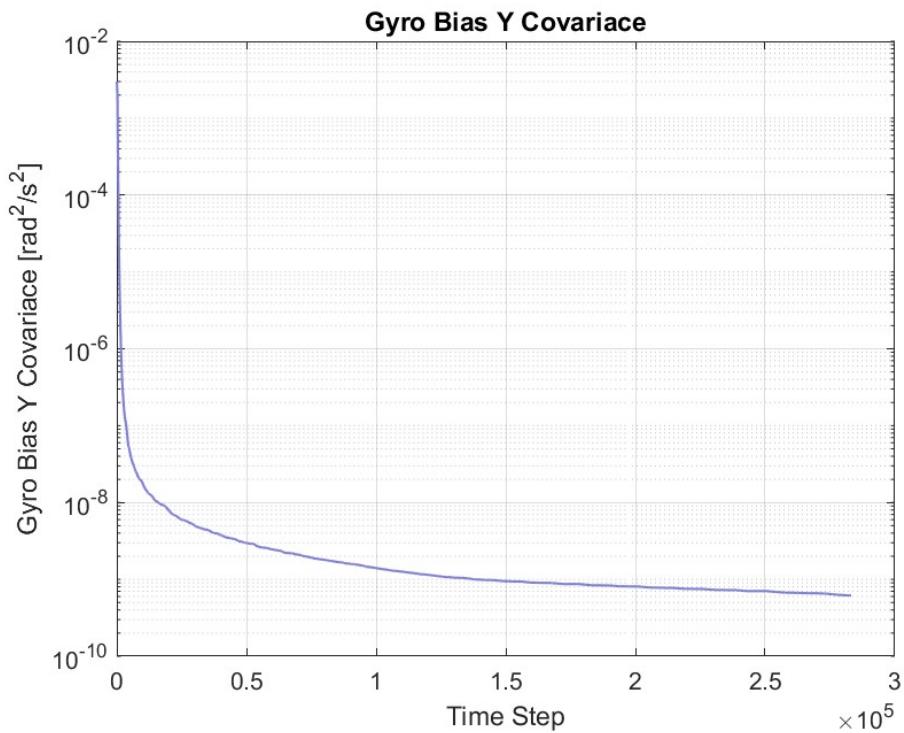


تصویر 80 کواریانس بایاس ژایرو محو را در مسیر

✓ نتایج شبیه سازی بایاس ژایرو محو γ

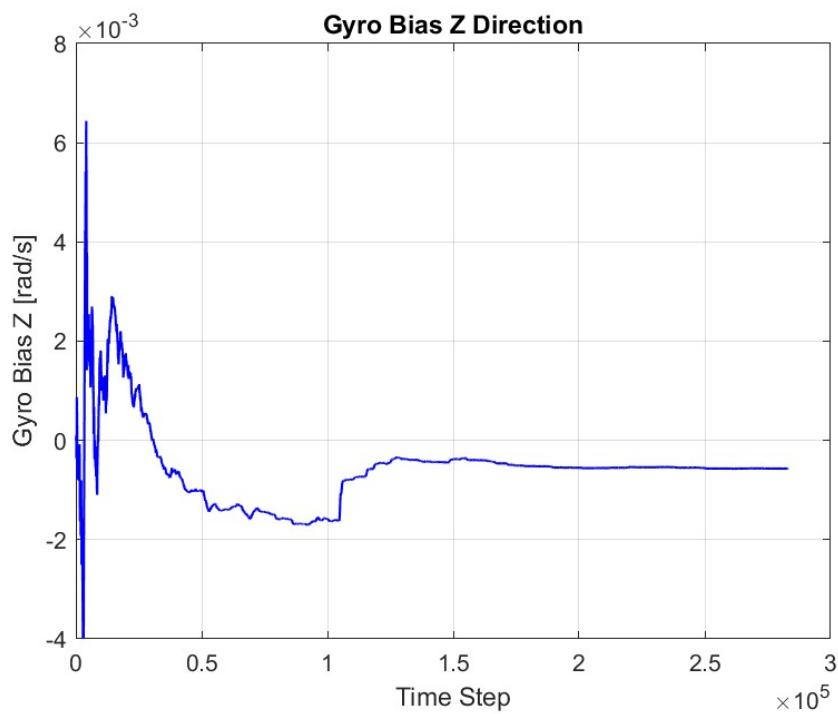


تصویر 81 بایاس ژایرو محو γ

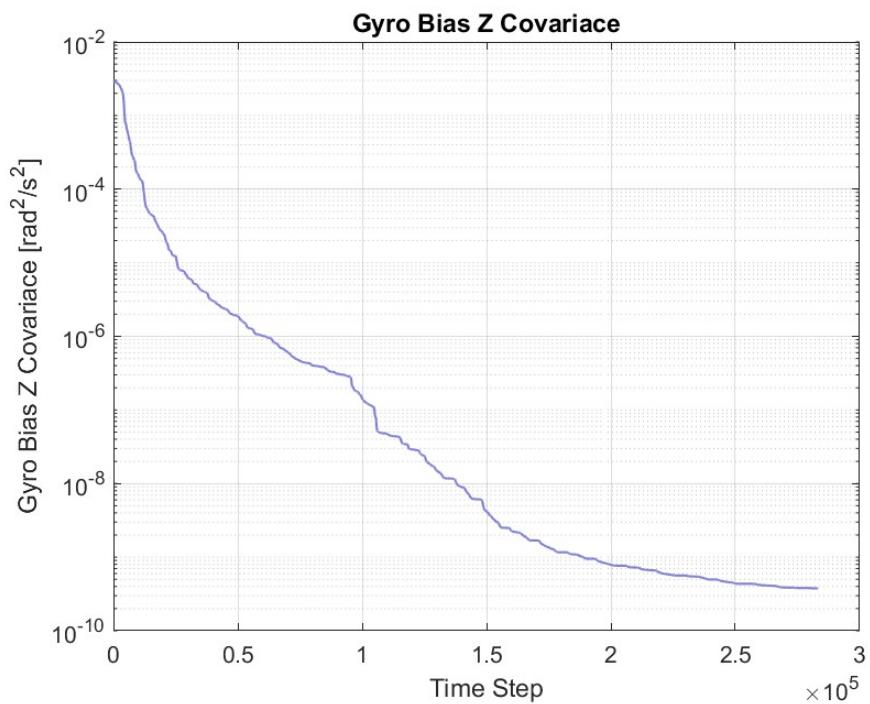


تصویر 82 کواریانس بایاس ژایرو محور Y

✓ نتایج شبیه سازی بایاس ژایرو محور Z



تصویر 83 بایاس ژایرو محور Z

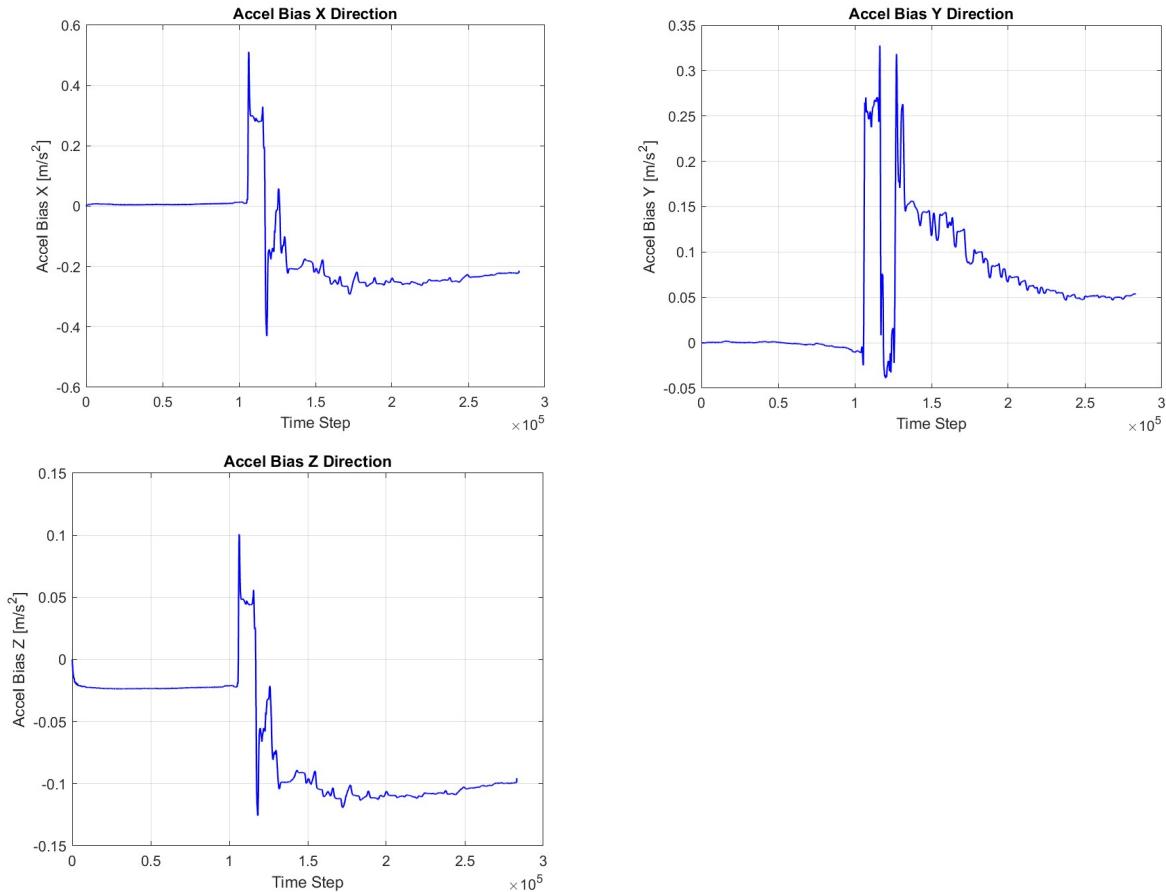


تصویر 84 کوواریانس بایاس زاویه محور Z

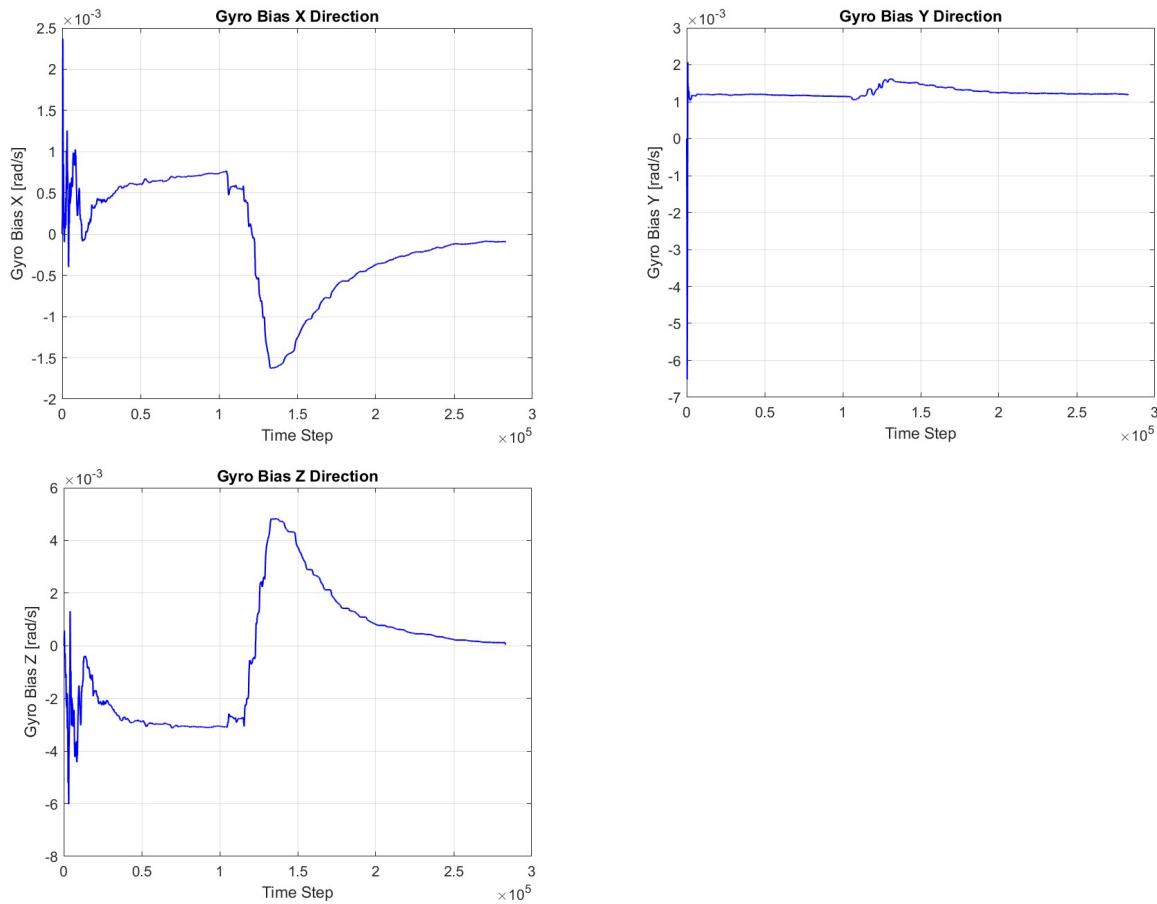
7. بررسی اثر افزایش reseting_rate

در این بخش مقدار reseting_rate را از مقدار 0.1 به 1 ثانیه تغییر میدهیم تا اثر آن در روند تخمین بایاس شتاب سنج و ژایرو را مشخص کنیم.

✓ بایاس شتاب سنج

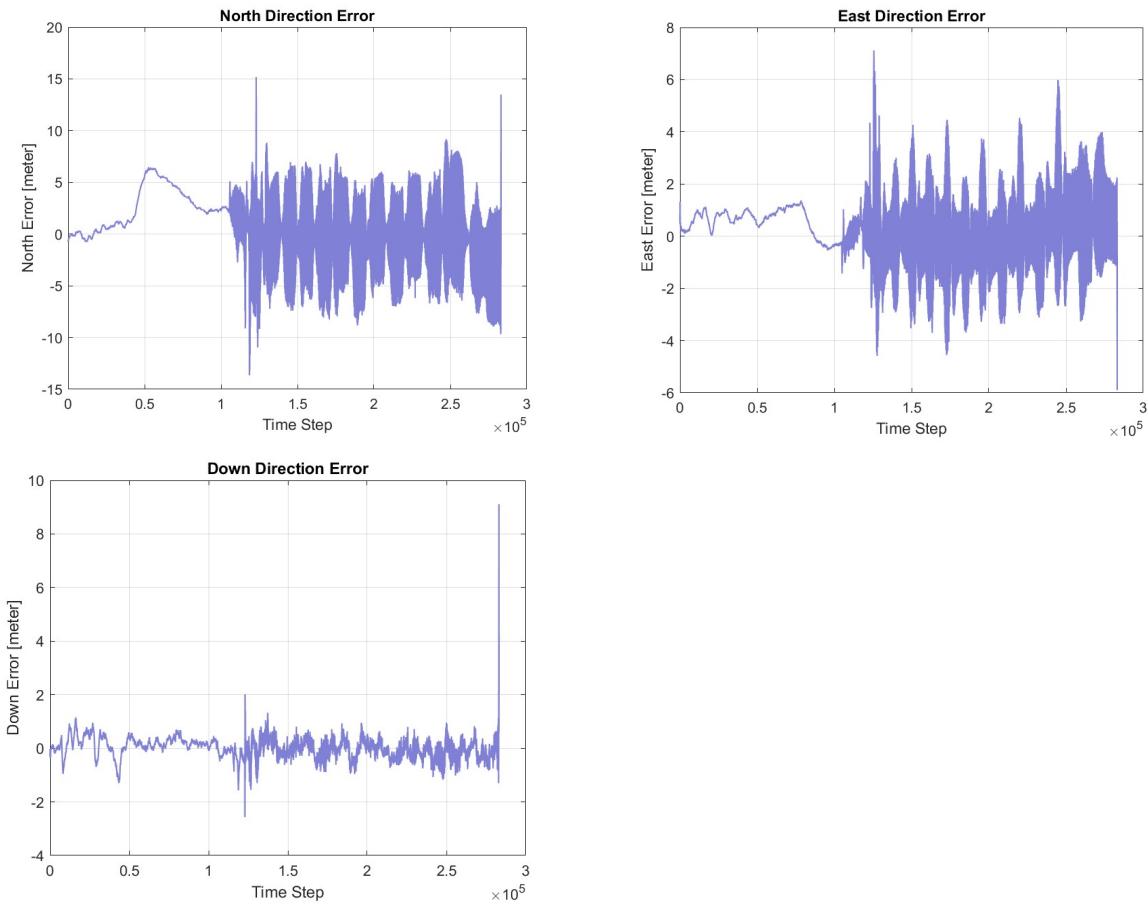


بايس ڙايروسکوپ ✓



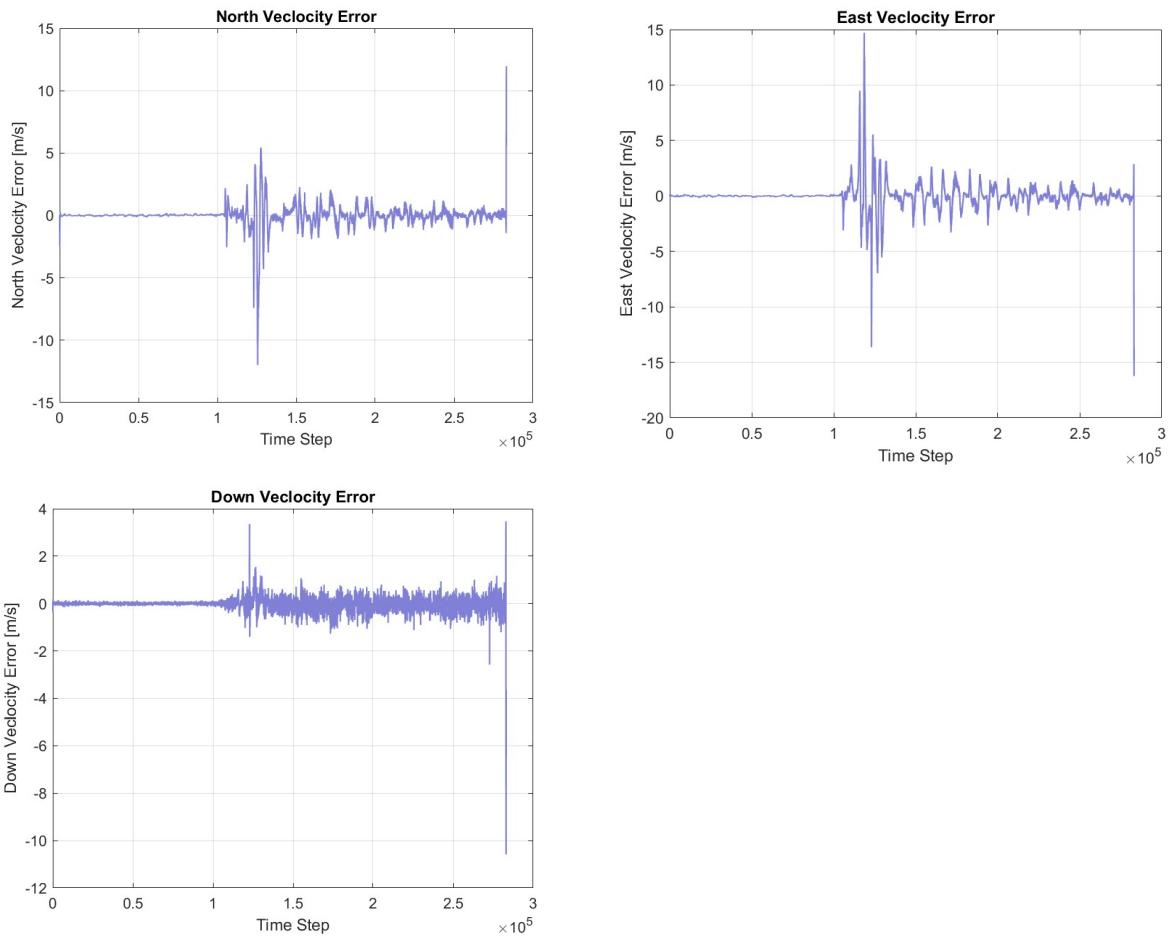
همانطور که ملاحظه میشود روند تخمین بايس شتاب سنج و ڙايرو ها بهبود يافته است ولی افزایش **reseting_rate** به معنی آن است که تخمین حالت ها دیر تر اصلاح میشوند و در نتیجه در جاهایی که مانور سریع در سیستم رخ دهد این باعث میشود که خطای تخمین رشد زیادی در این بین داشته باشد. به طور مثال مقدار خطای حالت ها در این حالت به شکل زیر بوده است.

✓ خطأ مكان در جهت شمال، شرق، پایین



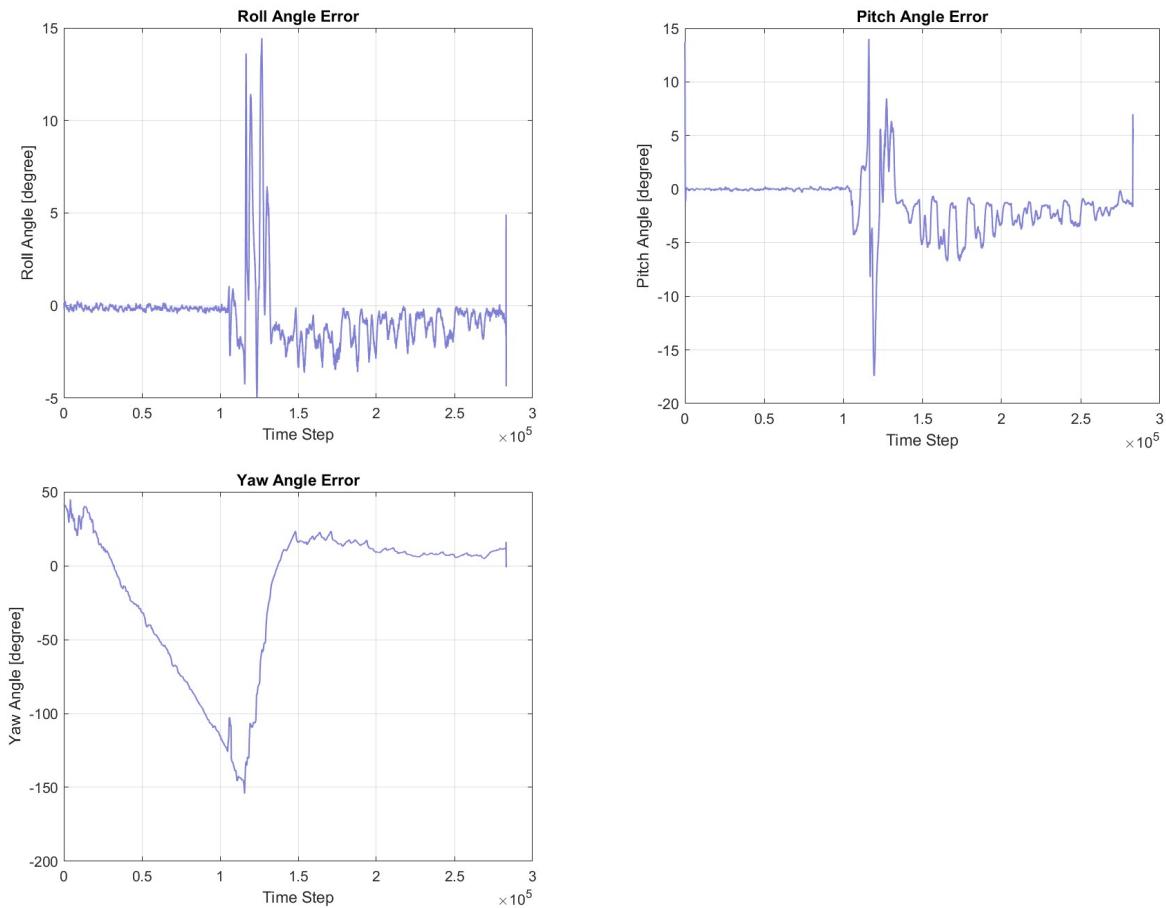
همانطور که ملاحظه میشود جهش های بیشتری در خطأ به وجود آمده ولی با همگرایی سریع تر بایاس شتاب سنج میتوان مشاهده نمود که بهبود نسبی ای در تخمین موقعیت در راستای شمالی ایجاد شده است.

✓ خطای سرعت



همانطور که مشاهده میشود در لحظه ای که سیستم شروع به حرکت میکند و قبل از آن بایاس شتاب سنج ها همگرا نشده است. به دلیل دقت پایین بایاس در ابتدای حرکت مشاهده میکنیم که خطای سیستم زیاد بوده است ولی با همگرایی سریع بایاس ها این خطا ها نیز در محدوده منطقی قرار میگیرند.

✓ خطای وضعیت



در تخمین وضعیت نیر همان روند قبلی توضیح داده شده اتفاق افتاده است.

8. جمع بندی

با توجه به نتایج گزارش شده میتوان مشاهده کرد که عملکرد فیلتر کالمن مستقیم و غیرمستقیم نزدیک به یکدیگر میباشد. در نهایت نیاز است که مشخصه RMS برای خطای تخمین را در دو الگوریتم با یکدیگر مقایسه کنیم تا دید جامعی درباره نحوه عملکرد به ما نشان دهد.

جدول 8-1 مقادیر RMS/رور در روش کالمن فیلتر مستقیم

Latitude Angle Error RMS [rad]	2.6546e-07
Longitude Angle Error RMS [rad]	4.6572e-07
Altitude Error RMS [meter]	0.3291
North Direction Error RMS [meter]	2.9694
East Direction Error RMS [meter]	1.0547
Down Direction Error RMS [meter]	0.3291
North Velocity Error RMS [m/s]	0.1805
East Velocity Error RMS [m/s]	0.3172
Down Velocity Error RMS [m/s]	0.2155
Roll Angle Error RMS [deg]	0.1915
Pitch Angle Error RMS [deg]	0.2632
Yaw Angle Error RMS [deg]	23.3584
Converged Yaw Angle Error RMS [deg]	1.1863

لازم به ذکر است که منظور از Converged Yaw Angle زوایای یاو تخمین زده شده پس از ثانیه 1300 در شبیه سازی است که در نمودار ها مشخص است که تخمین زاویه یاو از آنجا همگرا شده است. دلیل استفاده از 1300 آن است که تا قبل این زمان سیستم حرکتی نداشته و بنابراین نمیتوان از داده های GPS به صورت مناسبی زاویه yaw را تشخیص داد و پس از آنکه سیستم حرکت میکند زاویه yaw نیز با توجه به میزان کوپلینگ بین حالت ها در معادلات به مقدار درست خود میل میکند.

جدول ۸-۲ مقادیر RMS/رور در روش کالمون فیلتر غیر مستقیم با $reseting_rate$ برابر ۰.۱ ثانیه

Latitude Angle Error RMS [rad]	2.7055e-07
Longitude Angle Error RMS [rad]	4.7733e-07
Altitude Error RMS [meter]	0.3325
North Direction Error RMS [meter]	3.0434
East Direction Error RMS [meter]	1.0749
Down Direction Error RMS [meter]	0.3325
North Velocity Error RMS [m/s]	0.1966
East Velocity Error RMS [m/s]	0.3180
Down Velocity Error RMS [m/s]	0.2128
Roll Angle Error RMS [deg]	0.2393
Pitch Angle Error RMS [deg]	0.3262
Yaw Angle Error RMS [deg]	21.0077
Converged Yaw Angle Error RMS [deg]	1.9074

همانطور که از نتایج مشخص است، عملکرد دو الگوریتم بسیار نزدیک به یک دیگر میباشد ولی عملکرد الگوریتم مستقیم مقدار کمی از الگوریتم غیر مستقیم بهتر بوده است اما در نقطه مقابل به نظر میرسد الگوریتم غیر مستقیم عملکرد پایدار تری در تخمین بایاس سنسور ها دارد.