AER1513 Assignment 2 Report

Kevin Hu

October 26, 2023

1 Introduction

For this assignment we use an iterated Extended Kalman Filter (IEKF) to solve a nonlinear estimation problem of a robot's position given odometry measurements and range measurements to known landmark locations.

The assumption of zero-mean Gaussian noise is reasonable for the range and bearing errors as well as the translational and rotational speed errors because the error histograms resemble zero-mean Gaussians reasonably well.

The value of the variance $\mathbf{Q_k}$ that should be used is $\begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix}$ where σ_v^2 is the standard deviation squared or variance of the Gaussian curve fit to the translational speed error and σ_{om}^2 is the standard deviation squared or variance of the Gaussian curve fit to the rotational speed error. In this case $\sigma_v^2 = 0.066485_2 = 0.004420$ and $\sigma_\omega^2 = 0.090477_2 = 0.008186$, so the value of $\mathbf{Q_k}$ that should be used is $\begin{bmatrix} 0.004420 & 0 \\ 0 & 0.008186 \end{bmatrix}$.

The value of the variance $\mathbf{R_k^l}$ that should be used is $\begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_b^2 \end{bmatrix}$ where σ_r^2 is the standard deviation squared or variance of the Gaussian curve fit to the range error of the laser rangefinder and σ_b^2 is the standard deviation squared or variance of the Gaussian curve fit to the bearing error of the laser rangefinder. In this case $\sigma_r^2 = 0.030006_2 = 0.000900$ and $\sigma_{om}^2 = 0.025912_2 = 0.000671$, so the value of $\mathbf{R_k^l}$ that should be used is $\begin{bmatrix} 0.000900 & 0 \\ 0 & 0.000671 \end{bmatrix}$.

The expression for the Jacobian for the motion model $\mathbf{h}(\mathbf{x_{k-1}}, \mathbf{u_k}, \mathbf{w_k})$, $\mathbf{F_{k-1}}$, is a 3 by 3 matrix such that $\mathbf{F_{k-1}} = \begin{bmatrix} \frac{\partial h_1}{\partial x_{1,k-1}} & \frac{\partial h_1}{\partial x_{2,k-1}} & \frac{\partial h_1}{\partial x_{3,k-1}} \\ \frac{\partial h_2}{\partial x_{1,k-1}} & \frac{\partial h_2}{\partial x_{2,k-1}} & \frac{\partial h_2}{\partial x_{3,k-1}} \\ \frac{\partial h_3}{\partial x_{1,k-1}} & \frac{\partial h_3}{\partial x_{2,k-1}} & \frac{\partial h_3}{\partial x_{3,k-1}} \end{bmatrix}$ where h_i is the i-th row or i-th equation of the motion model and $x_{i,k-1}$ is the i-th element of the vector $\mathbf{x_{k-1}}$. In this case, $x_{1,k-1} = x_{k-1}$, $x_{2,k-1} = y_{k-1}$, and $x_{3,k-1} = \theta_{k-1}$. The expression of the Jacobian $\mathbf{F_{k-1}}$ with the partial derivatives filled in is: $\begin{bmatrix} 1 & 0 & -T(v_k + w_{1,k}) sin\theta_{k-1} \\ 0 & 1 & T(v_k + w_{1,k}) cos\theta_{k-1} \\ 0 & 0 & 1 \end{bmatrix}$ where $w_{1,k}$ is the first element of the vector $\mathbf{w_k}$.

The expression for the Jacobian for the observation model $\mathbf{g}(\mathbf{x_k}, \mathbf{n_k^l})$, $\mathbf{G_k}$, is a $2L$ by 3 matrix such that $\mathbf{G_k} = \begin{bmatrix} \mathbf{G_k^1} \\ \mathbf{G_k^2} \\ \vdots \\ \mathbf{G_k^L} \end{bmatrix}$ and $\mathbf{G_k^l} = \begin{bmatrix} \frac{\partial g_1^l}{\partial x_{1,k}} & \frac{\partial g_1^l}{\partial x_{2,k}} & \frac{\partial g_1^l}{\partial x_{3,k}} \\ \frac{\partial g_2^l}{\partial x_{1,k}} & \frac{\partial g_2^l}{\partial x_{2,k}} & \frac{\partial g_2^l}{\partial x_{3,k}} \end{bmatrix}$ where L is the number of landmarks seen by the laser rangefinder and g_n^l is the n-th row or n-th equation of the observation model for landmark l. The expression of the Jacobian $\mathbf{G_k^l}$ with the partial derivatives filled in is:

$$\begin{bmatrix} \frac{x_k - x_l + dcos\theta_k}{\sqrt{(x_l - x_k - dcos\theta_k)^2 + (y_l - y_k - dsin\theta_k)^2}} & \frac{y_k - y_l + dsin\theta_k}{\sqrt{(x_l - x_k - dcos\theta_k)^2 + (y_l - y_k - dsin\theta_k)^2}} & \frac{d(x_l - x_k + dcos\theta_k) sin\theta_k + d(y_l - y_k + sin\theta_k) cos\theta_k}{\sqrt{(x_l - x_k - dcos\theta_k)^2 + (y_l - y_k - dsin\theta_k)^2}} \\ \frac{y_l - y_k - dsin\theta_k}{(x_l - x_k - dcos\theta_k)^2 + (y_l - y_k - dsin\theta_k)^2} & \frac{x_k - x_l + dcos\theta_k}{(x_l - x_k - dcos\theta_k)^2 + (y_l - y_k - dsin\theta_k)^2} & \frac{dsin\theta_k (y_k - y_1 + dsin\theta_k) + dcos\theta_k (x_k - x_l + dcos\theta_k)}{(x_l - x_k - dcos\theta_k)^2 + (y_l - y_k - dsin\theta_k)^2} - 1 \end{bmatrix}$$

Remember that the actual Jacobian matrix for the observation model, $\mathbf{G_k}$, has L of these blocks stacked on top of each other with L being the number of landmarks visible at the current time step k.

The Jacobians for the noise terms $\mathbf{w_k}$ and $\mathbf{n_k}$ when multiplied by their vectors give the terms $\mathbf{w'_k}$ and $\mathbf{n'_k}$. The expression for the Jacobian of the motion model noise term $\frac{\partial \mathbf{h}}{\partial \mathbf{w_k}} = \begin{bmatrix} \frac{\partial h_1}{\partial w_{1,k}} & \frac{\partial h_1}{\partial w_{2,k}} \\ \frac{\partial h_2}{\partial w_{1,k}} & \frac{\partial h_2}{\partial w_{2,k}} \\ \frac{\partial h_3}{\partial w_{1,k}} & \frac{\partial h_3}{\partial w_{2,k}} \end{bmatrix}$ where $w_{i,k}$ is the i-th element of the noise vector $\mathbf{w}$. The expression of the Jacobian with the partial derivatives filled in is: $\begin{bmatrix} Tcos\theta_{k-1} & 0 \\ Tsin\theta_{k-1} & 0 \\ 0 & T \end{bmatrix}$. Remember that this term needs to be multiplied by the vector $\mathbf{w_k}$ to get the term $\mathbf{w'_k}$.

The expression for the Jacobian of the observation model noise term is a $2L$ by $2L$ matrix such that $\frac{\partial \mathbf{g}}{\partial \mathbf{n_k}} = \begin{bmatrix} \frac{\partial \mathbf{g^1}}{\partial \mathbf{n_k^1}} & & & \\ & \frac{\partial \mathbf{g^2}}{\partial \mathbf{n_k^2}} & & \\ & & \ddots & \\ & & & \frac{\partial \mathbf{g^L}}{\partial \mathbf{n_k^L}} \end{bmatrix}$ and $\frac{\partial \mathbf{g^l}}{\partial \mathbf{n_k^l}} = \begin{bmatrix} \frac{\partial g_1^l}{\partial n_{1,k}^l} & \frac{\partial g_1^l}{\partial n_{2,k}^l} \\ \frac{\partial g_2^l}{\partial n_{1,k}^l} & \frac{\partial g_2^l}{\partial n_{2,k}^l} \end{bmatrix}$ where $n_{i,k}^l$ is the i-th component of the vector n_k^l and g_i^l is the i-th equation of the measurement equations for the l-th landmark. The index l indicates the l-th landmark and L is the total number of visible landmarks at the current timestep. After filling in the partial derivatives, the matrix $\frac{\partial \mathbf{g}}{\partial \mathbf{n_k}}$ looks like $\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$ or L 2 by 2 identity matrices in a diagonal.

The process noise Jacobian covariance matrix $\mathbf{Q_k'} = \frac{\partial \mathbf{h}}{\partial \mathbf{w_k}} \mathbf{Q_k} \frac{\partial \mathbf{h}}{\partial \mathbf{w_k}}^\top$ where $\mathbf{Q_k}$ is the process noise covariance matrix. $\mathbf{Q_k'}$ can be written as $\begin{bmatrix} T^2\sigma_v^2 cos^2\theta_k & T^2\sigma_v^2 sin\theta_k cos\theta_k & 0 \\ T^2\sigma_v^2 sin\theta_k cos\theta_k & T^2\sigma_v^2 sin^2\theta_k & 0 \\ 0 & 0 & T^2\sigma_\omega^2 \end{bmatrix}$ where σ_v is the standard deviation of the Gaussian curve fit to the translational speed error and σ_ω is the standard deviation of the Gaussian curve fit to the rotational speed error.

The measurement noise Jacobian covariance matrix $\mathbf{R_k'} = \frac{\partial \mathbf{g}}{\partial \mathbf{n_k}} \mathbf{R_k} \frac{\partial \mathbf{g}}{\partial \mathbf{n_k}}^\top$ where $\mathbf{R_k}$ is the measurement noise covariance matrix. The measurement noise covariance matrix $\mathbf{R_k}$ is in the form $diag(\mathbf{R_k^1}, \mathbf{R_k^2}, \ldots, \mathbf{R_k^L})$ where $\mathbf{R_k^l}$ is the measurement noise covariance of landmark l. In this case, $\mathbf{R_k^l}$ is the same for each landmark so the measurement noise covariance matrix will be a diagonal matrix with repeating entries every two entries. Here the two Jacobians are L by L identity matrices. Thus $\mathbf{R_k'}$ can be written as $diag(\mathbf{R_k^1}, \mathbf{R_k^2}, \ldots, \mathbf{R_k^L})$ where $\mathbf{R_k^l} = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{bmatrix}$, L is the total number of visible landmarks at the current time step, σ_r is the standard deviation of the Gaussian curve fit to the range reading error and σ_ϕ is the standard deviation of the Gaussian curve fit to the range bearing error.

The measurement vector $\mathbf{y_k} = \begin{bmatrix} \mathbf{y_k^1} \\ \mathbf{y_k^2} \\ \vdots \\ \mathbf{y_k^L} \end{bmatrix}$ where $\mathbf{y_k^l}$ is a measurement of a visible landmark. $\mathbf{y_k}$ varies in size depending on the number of landmarks L that are visible. Just a reminder that $\mathbf{y_k^l} = \begin{bmatrix} r_k^l \\ \phi_k^l \end{bmatrix}$. The similarly the function $\mathbf{g}(\mathbf{x_k}, \mathbf{n_k}) = \begin{bmatrix} \mathbf{g_k^1} \\ \mathbf{g_k^2} \\ \vdots \\ \mathbf{g_k^L} \end{bmatrix}$ where $\mathbf{g_k^l} = \begin{bmatrix} \sqrt{(x_l - x_k - dcos\theta_k)^2 + (y_l - y_k - dsin\theta_k)^2} \\ atan2(y_l - y_k - dsin\theta_k, x_l - x_k - dcos\theta_k) - \theta_k \end{bmatrix}$ for landmark l out of the number of landmarks L that are visible at the current time step.

The Extended Kalman Filter (EKF) equations for this problem are written in a somewhat simplified form below:

$$\check{\mathbf{P}}_k = \mathbf{F_{k-1}}\hat{\mathbf{P}}_{\mathbf{k-1}}\mathbf{F}_{\mathbf{k-1}}^{\top} + \mathbf{Q}_{\mathbf{k}}' \tag{1}$$

$$\check{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + T \begin{bmatrix} cos\theta_{k-1} & 0 \\ sin\theta k - 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} v_k \\ \omega_k \end{bmatrix} \tag{2}$$

$$\mathbf{K_k} = \check{\mathbf{P}}_{\mathbf{k}}\mathbf{G}_{\mathbf{k}}^{\top}(\mathbf{G_k}\check{\mathbf{P}}_{\mathbf{k}}\mathbf{G}_{\mathbf{k}}^{\top} + \mathbf{R}_{\mathbf{k}}')^{-\mathbf{1}} \tag{3}$$

$$\hat{\mathbf{P}}_k = (\mathbf{1} - \mathbf{K_k}\mathbf{G_k})\check{\mathbf{P}}_{\mathbf{k}} \tag{4}$$

$$\hat{\mathbf{x}}_k = \check{\mathbf{x}}_k + \mathbf{K_k}(\mathbf{y_k} - \mathbf{g}(\check{\mathbf{x}}_{\mathbf{k}}, \mathbf{0})) \tag{5}$$

Note that the Jacobian $\mathbf{F_{k-1}}$ is evaluated using $\hat{\mathbf{x}}_{k-1}$, $\mathbf{v_k}$ and $\mathbf{w_k} = \mathbf{0}$. The Jacobian $\mathbf{G_k}$ is evaluated using $\check{\mathbf{x}}_{k-1}$ and $\mathbf{n_k} = \mathbf{0}$. The vector $\mathbf{n_k}$ is a $2L$ by 1 vector which is of the form $\begin{bmatrix} \mathbf{n_k^1} \\ \mathbf{n_k^2} \\ \vdots \\ \mathbf{n_k^L} \end{bmatrix}$ where $\mathbf{n_l^1}$ is the measurement noise of landmark l out of L visible landmarks at the current time step. In the equations above, $\mathbf{1}$ is a 3 by 3 identity matrix and $\mathbf{0}$ is a $2L$ by 1 vector of zeros.

4a

$\hat{\mathbf{x}}_0 = \mathbf{x}_0$:

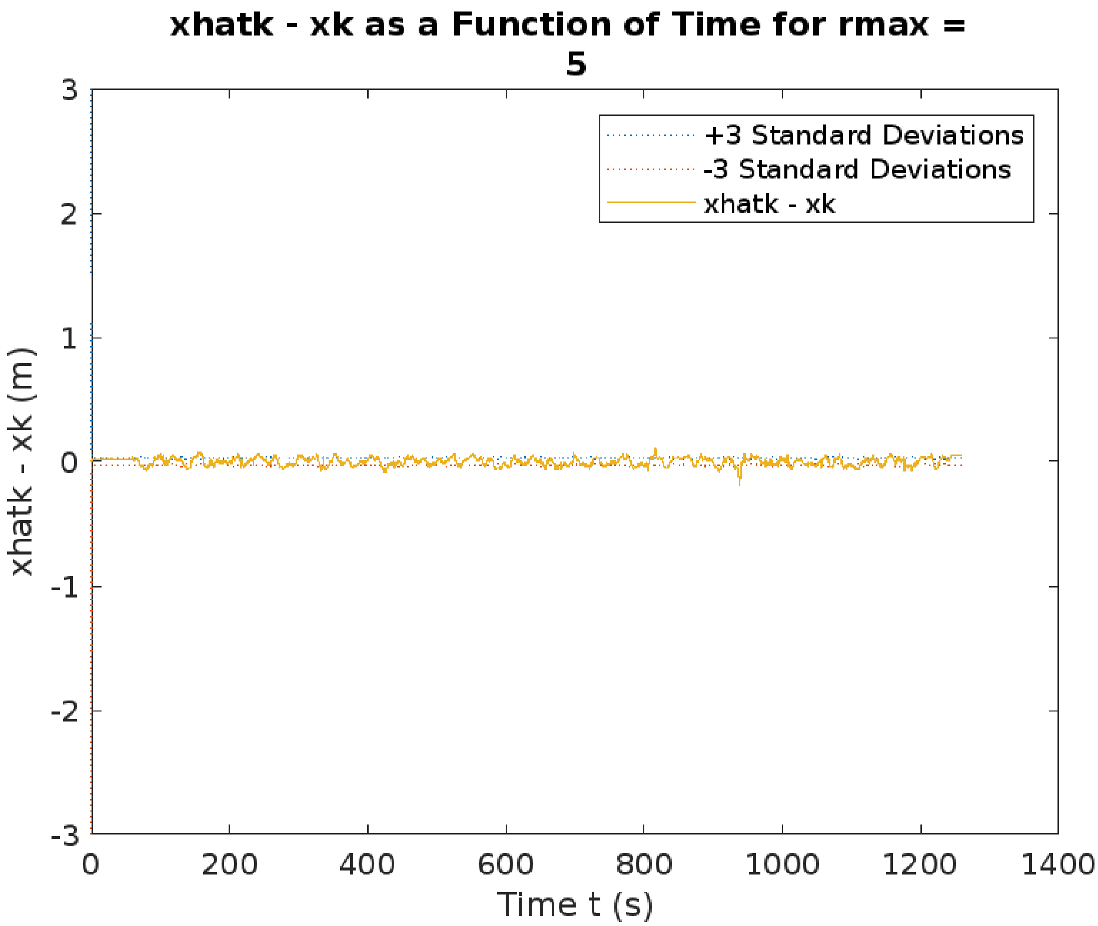

Figure 1: Zoomed Out View

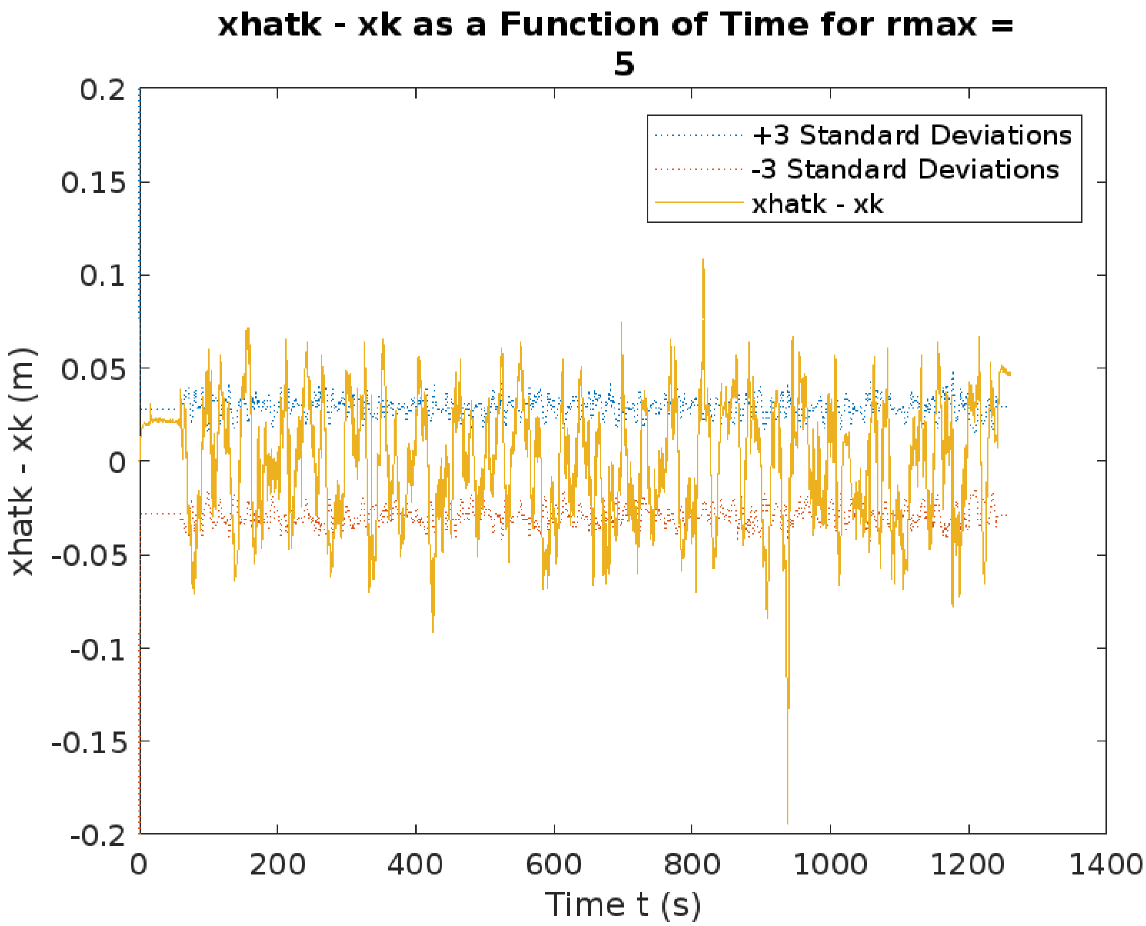

Figure 2: Zoomed In View

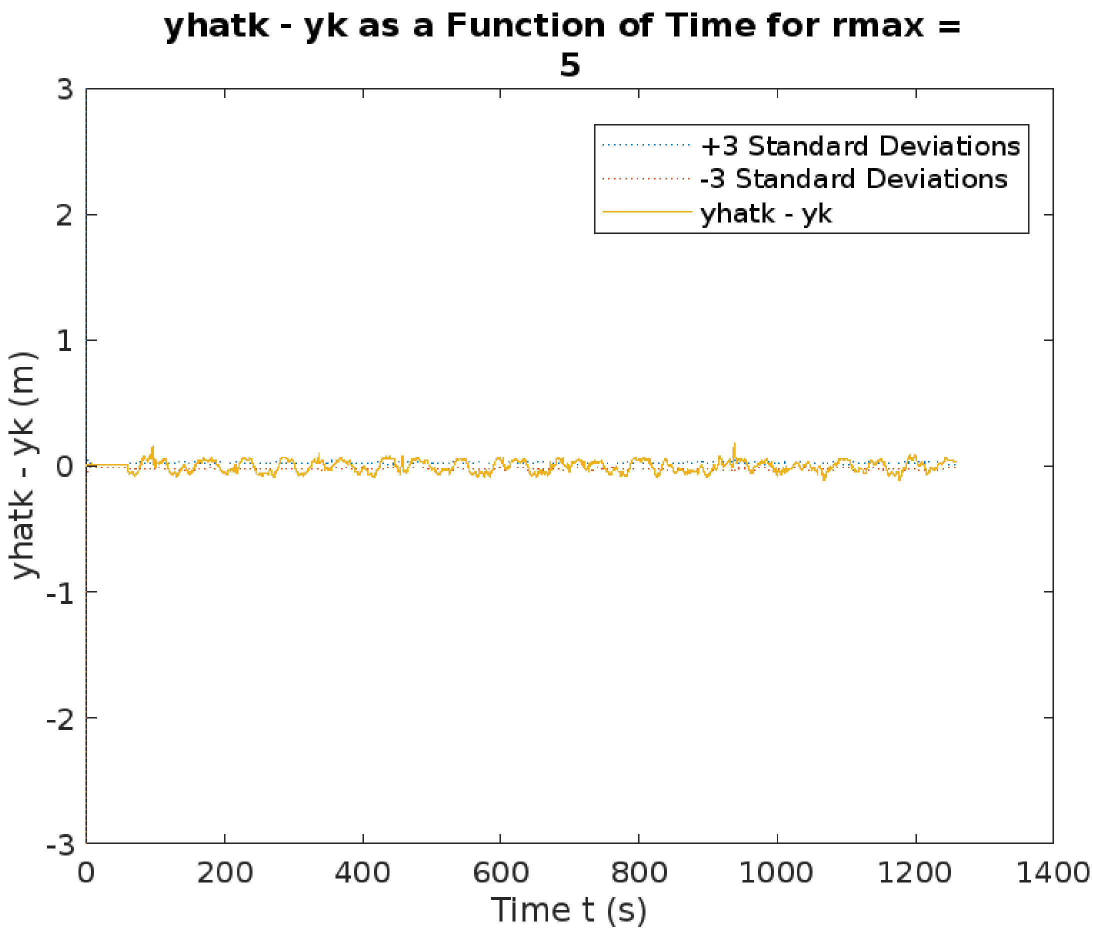

Figure 3: Zoomed Out View

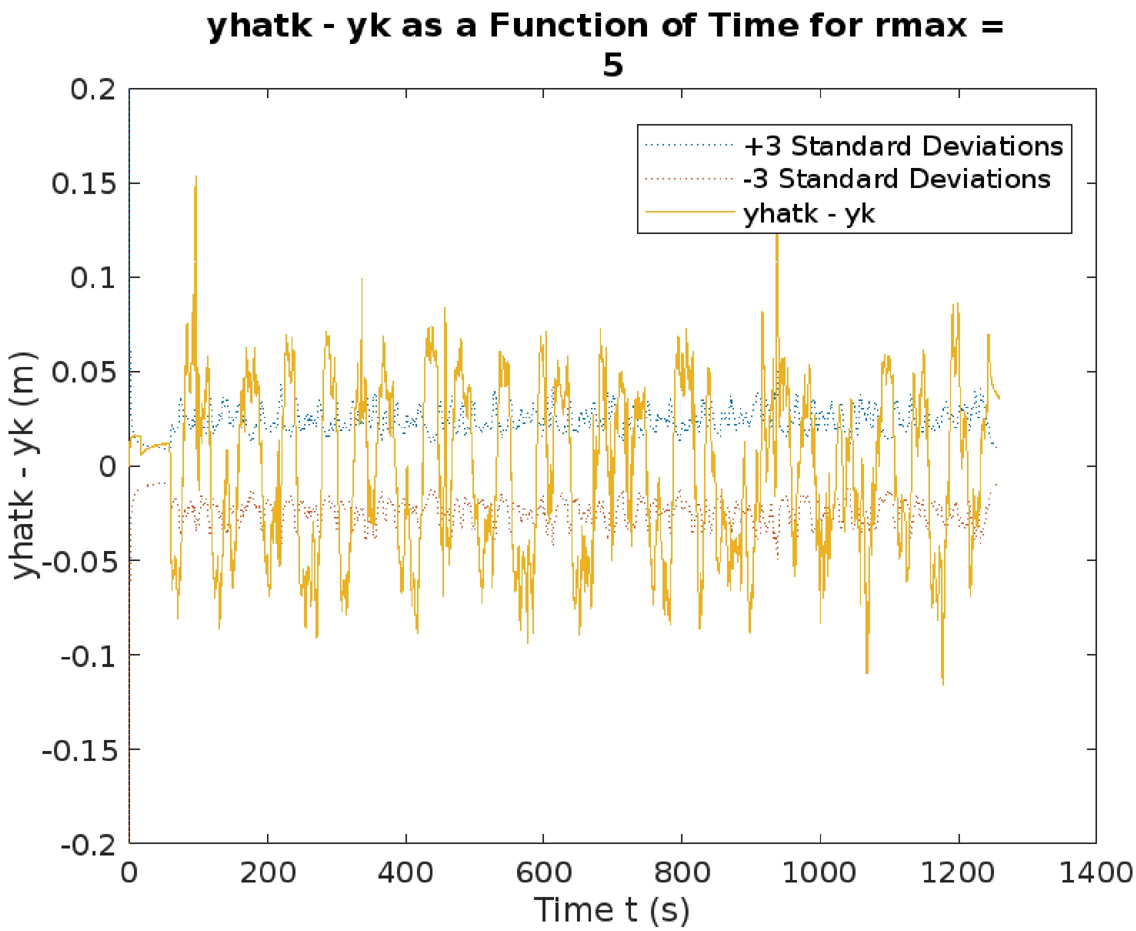

Figure 4: Zoomed In View

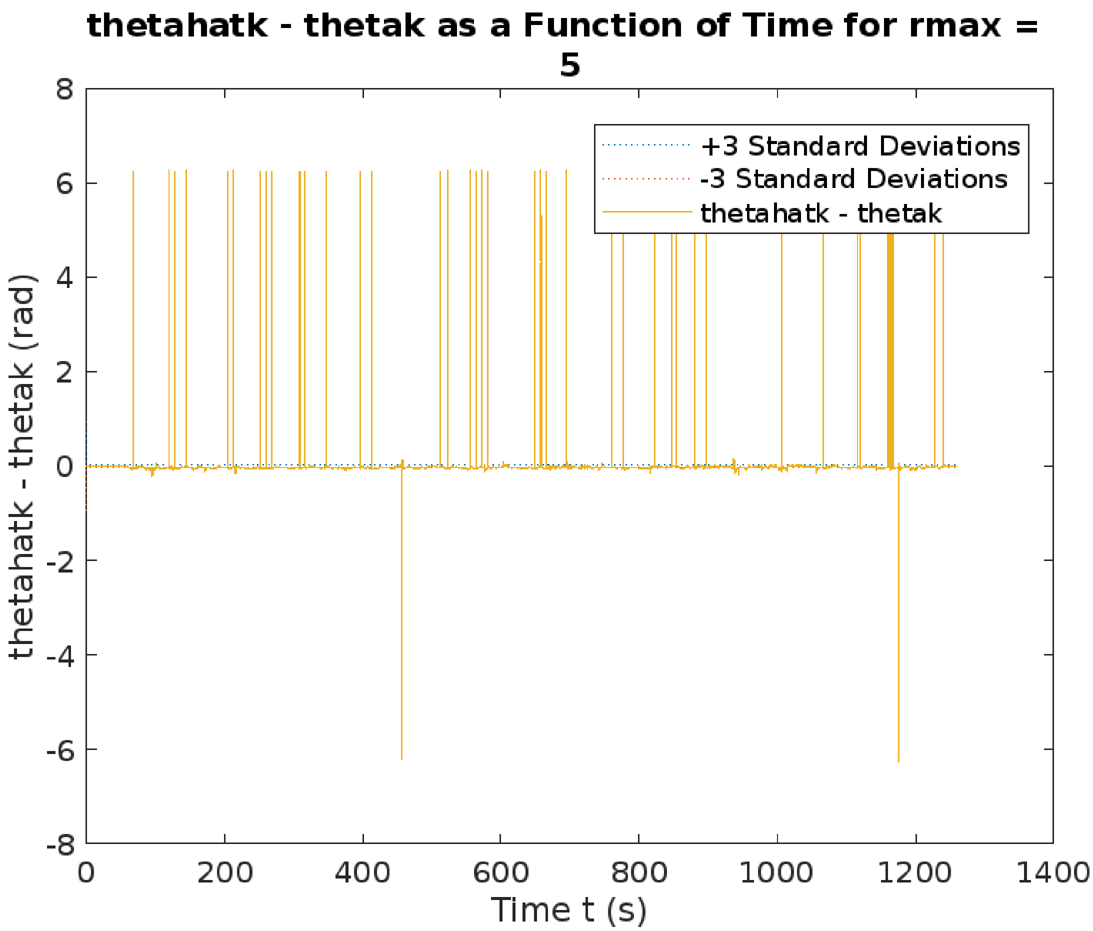

Figure 5: Zoomed Out View

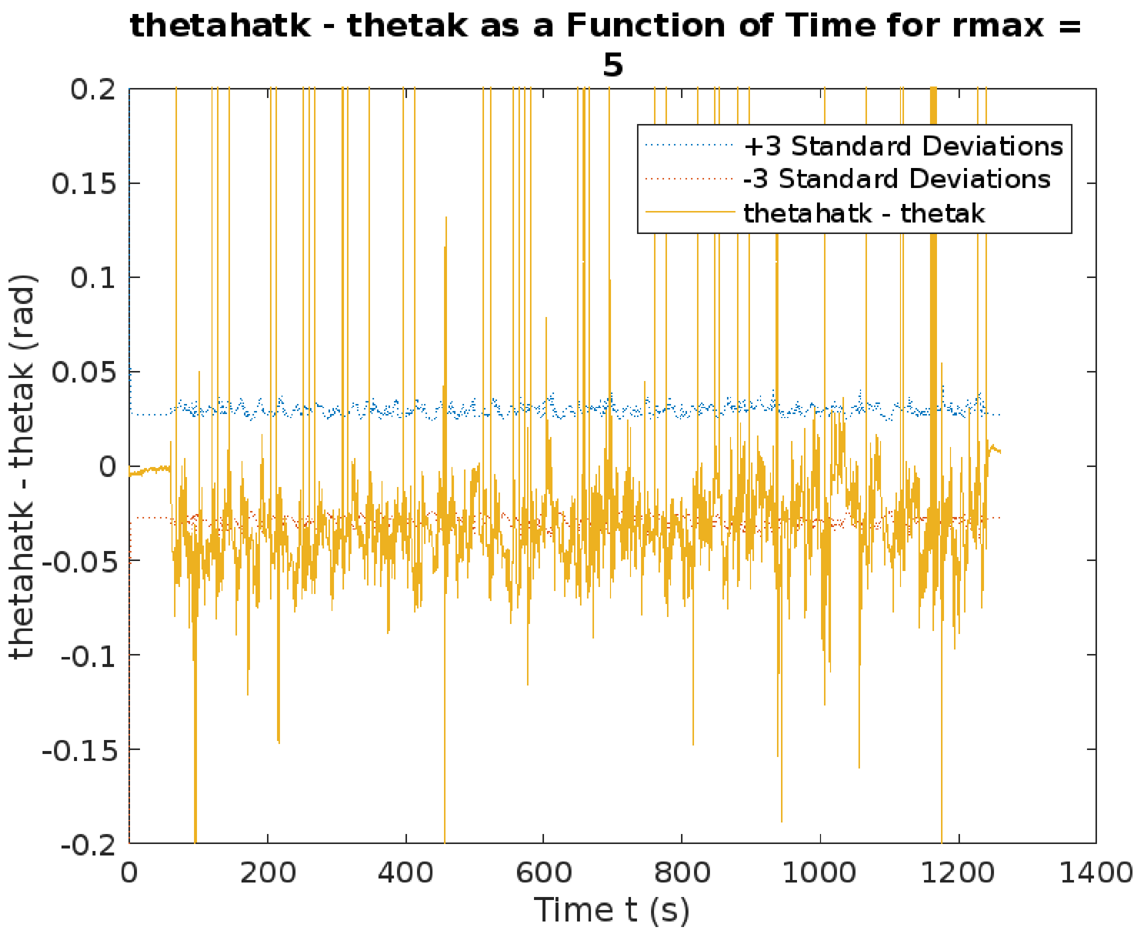

Figure 6: Zoomed In View

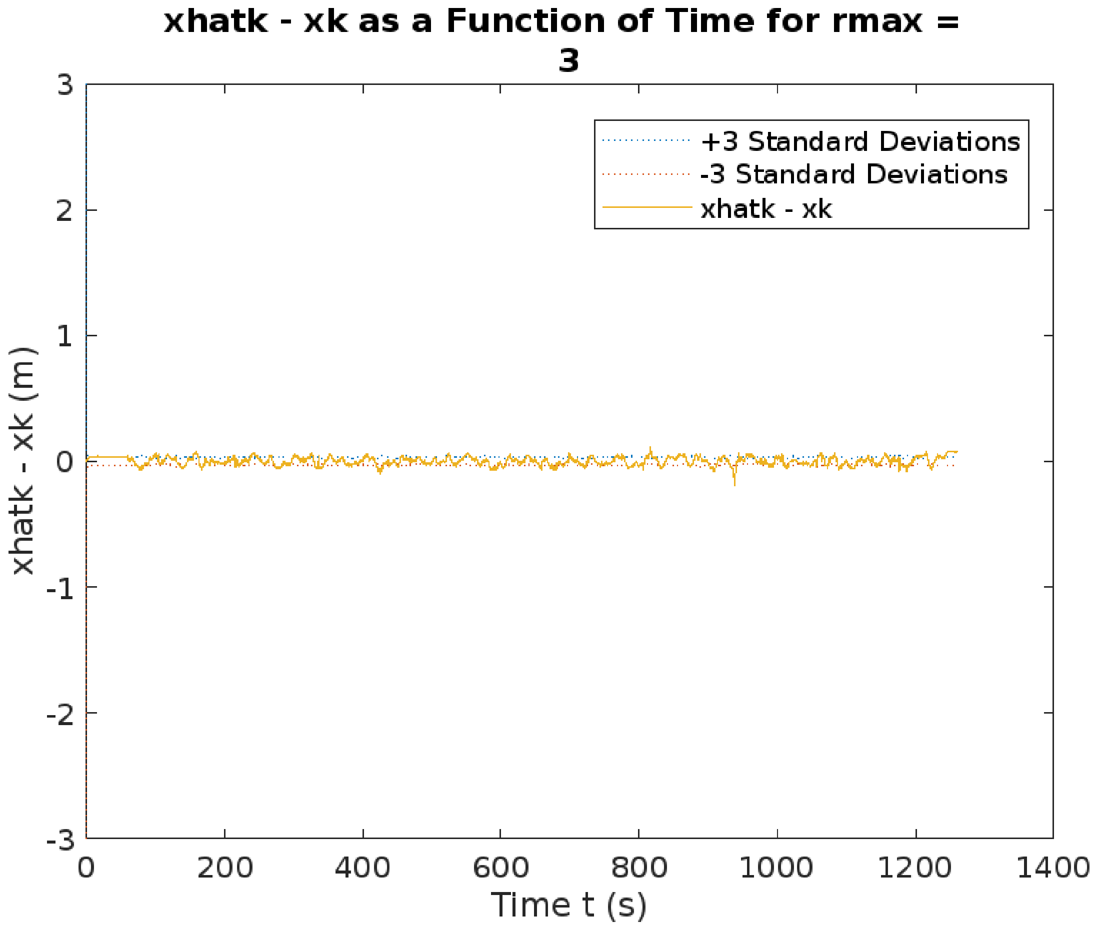

Figure 7: Zoomed Out View

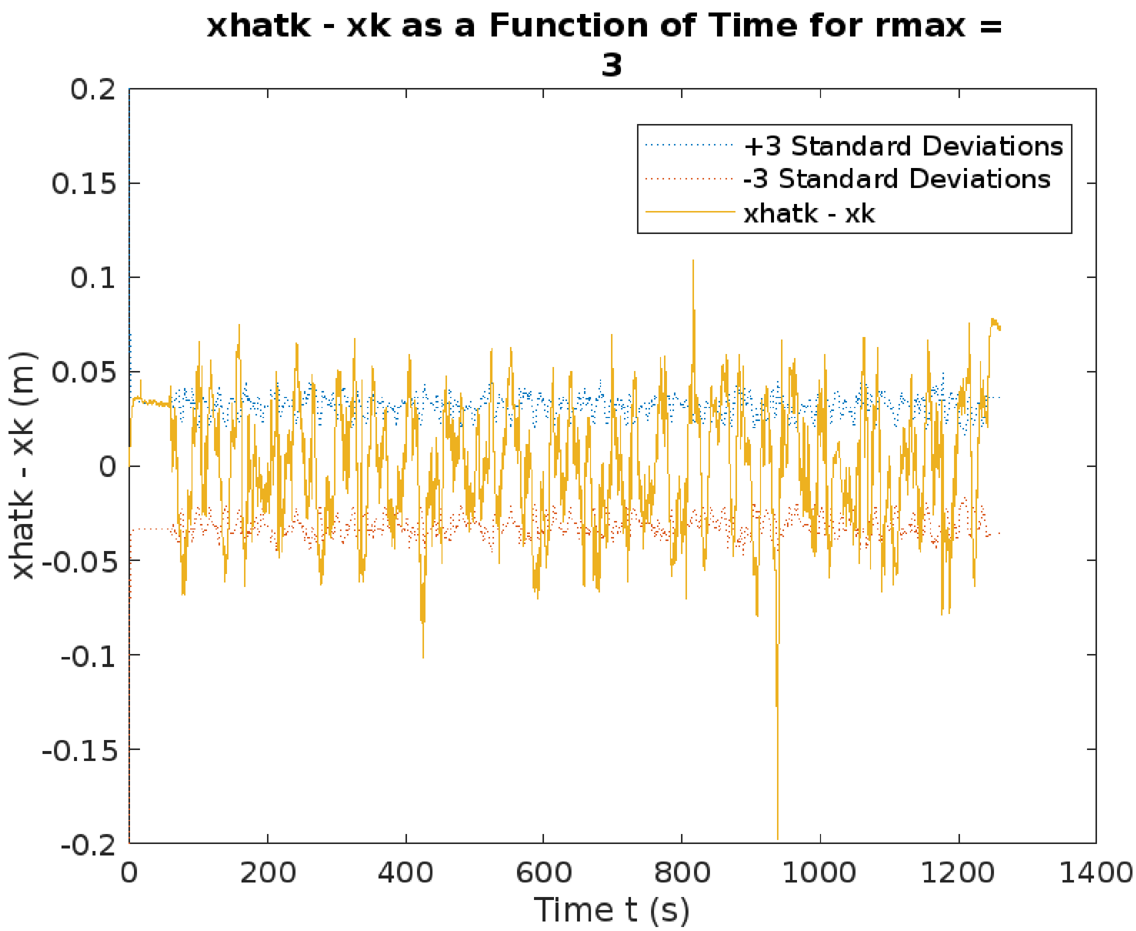

Figure 8: Zoomed In View

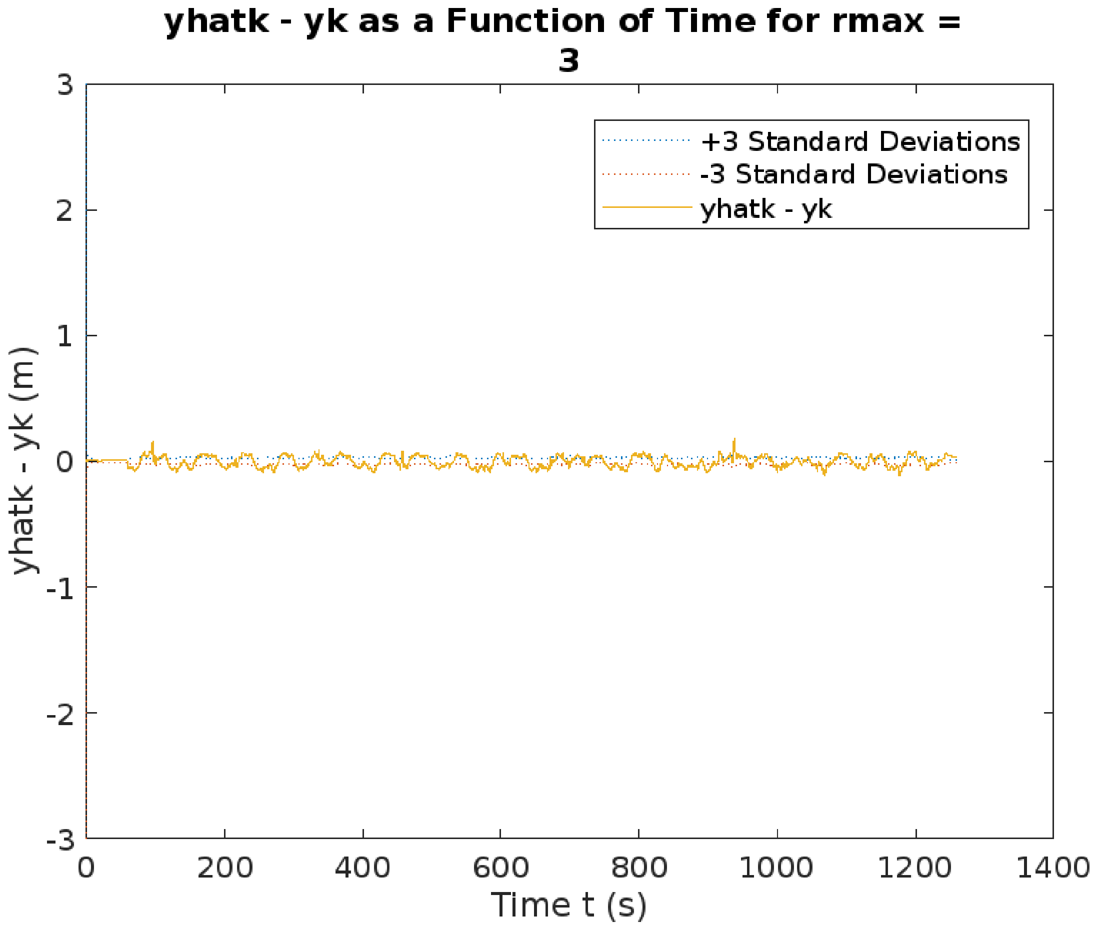

Figure 9: Zoomed Out View

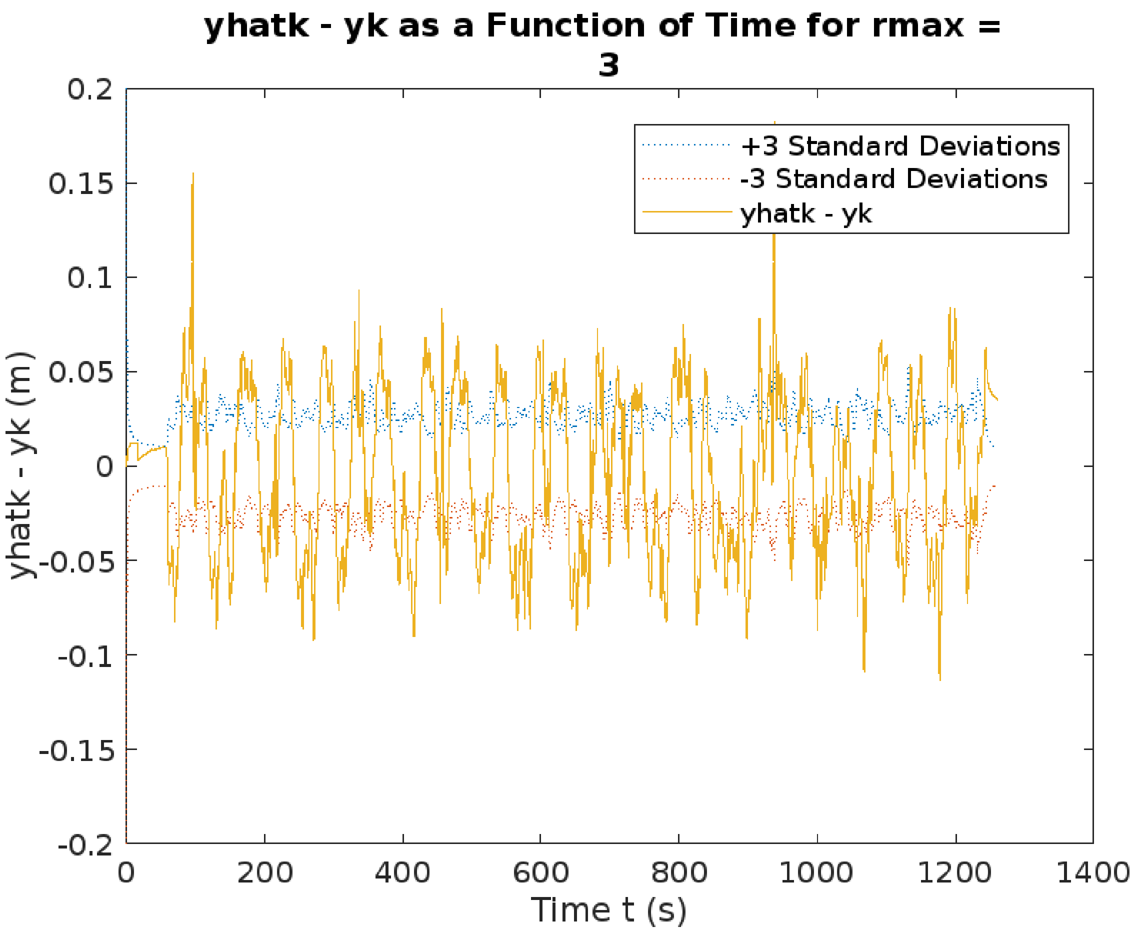

Figure 10: Zoomed In View

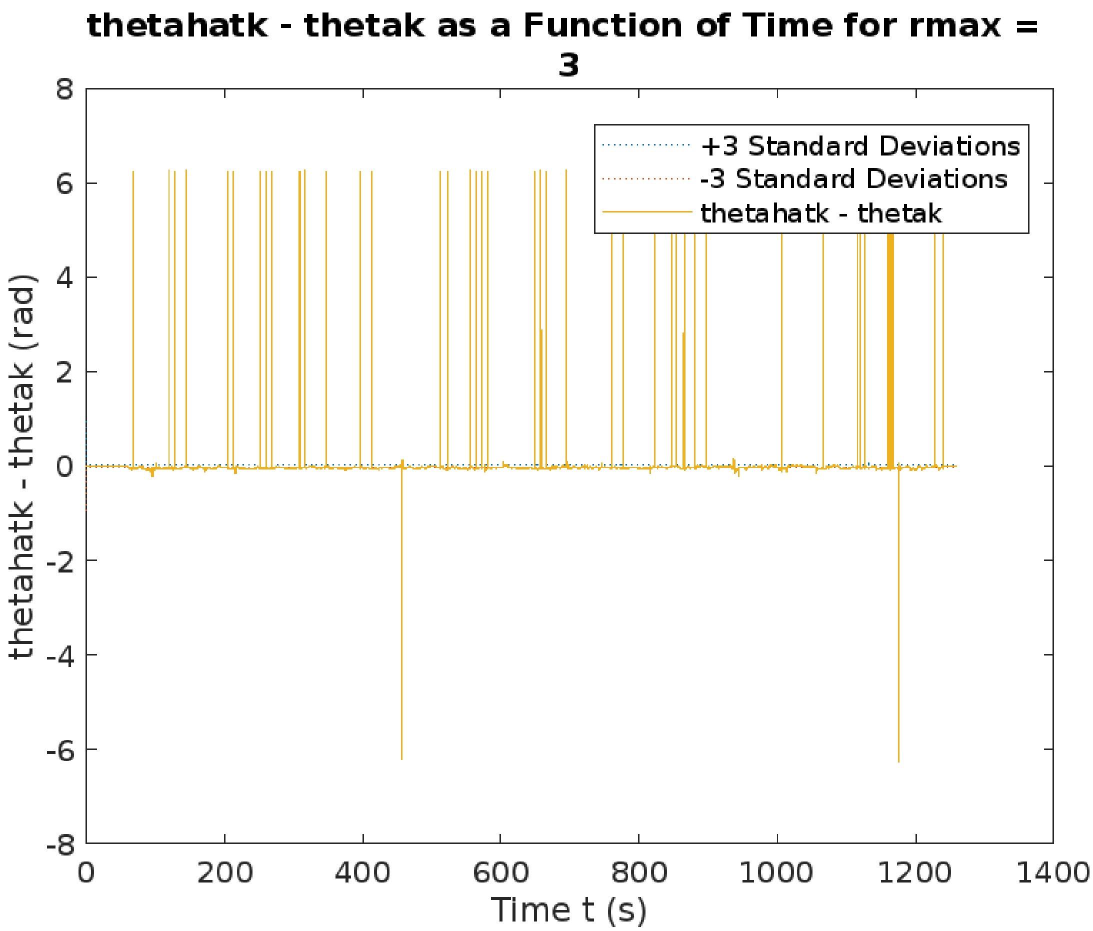

Figure 11: Zoomed Out View

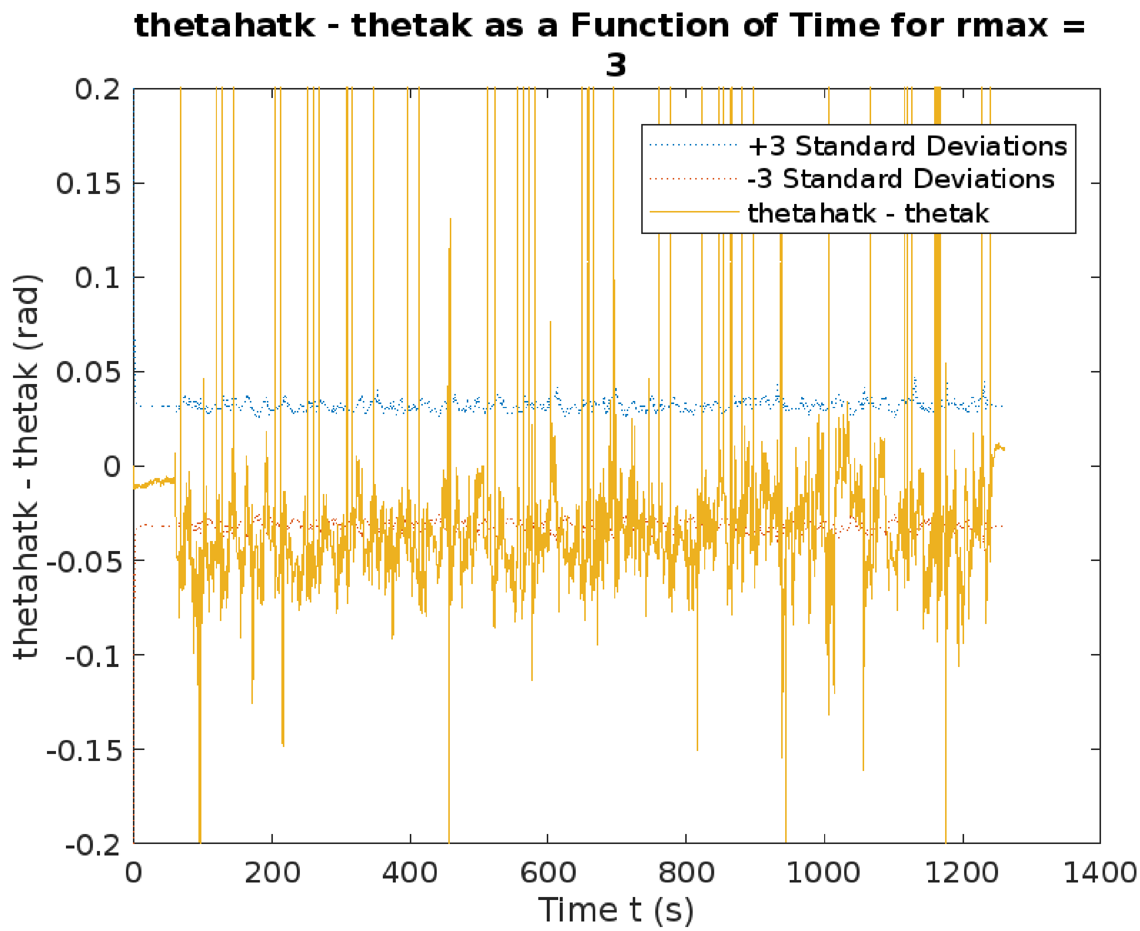

Figure 12: Zoomed In View

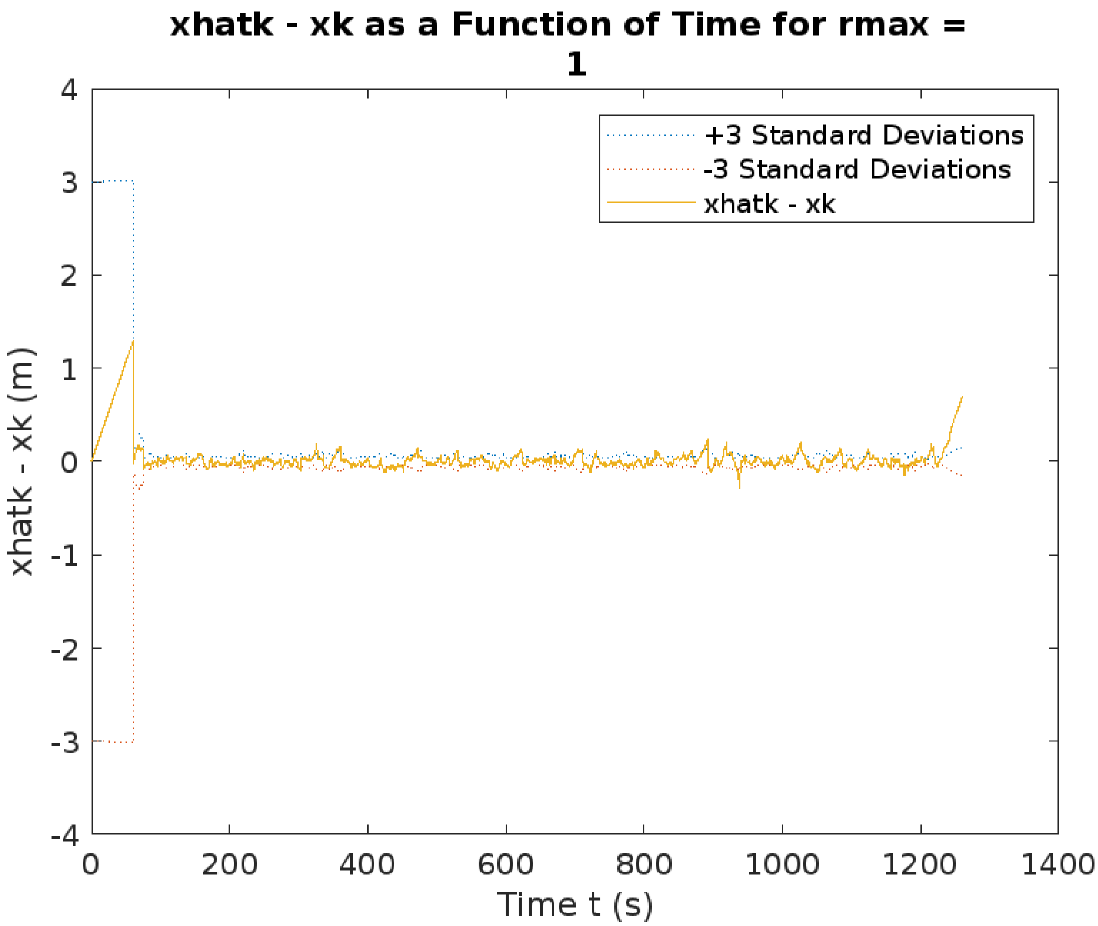

Figure 13: Zoomed Out View

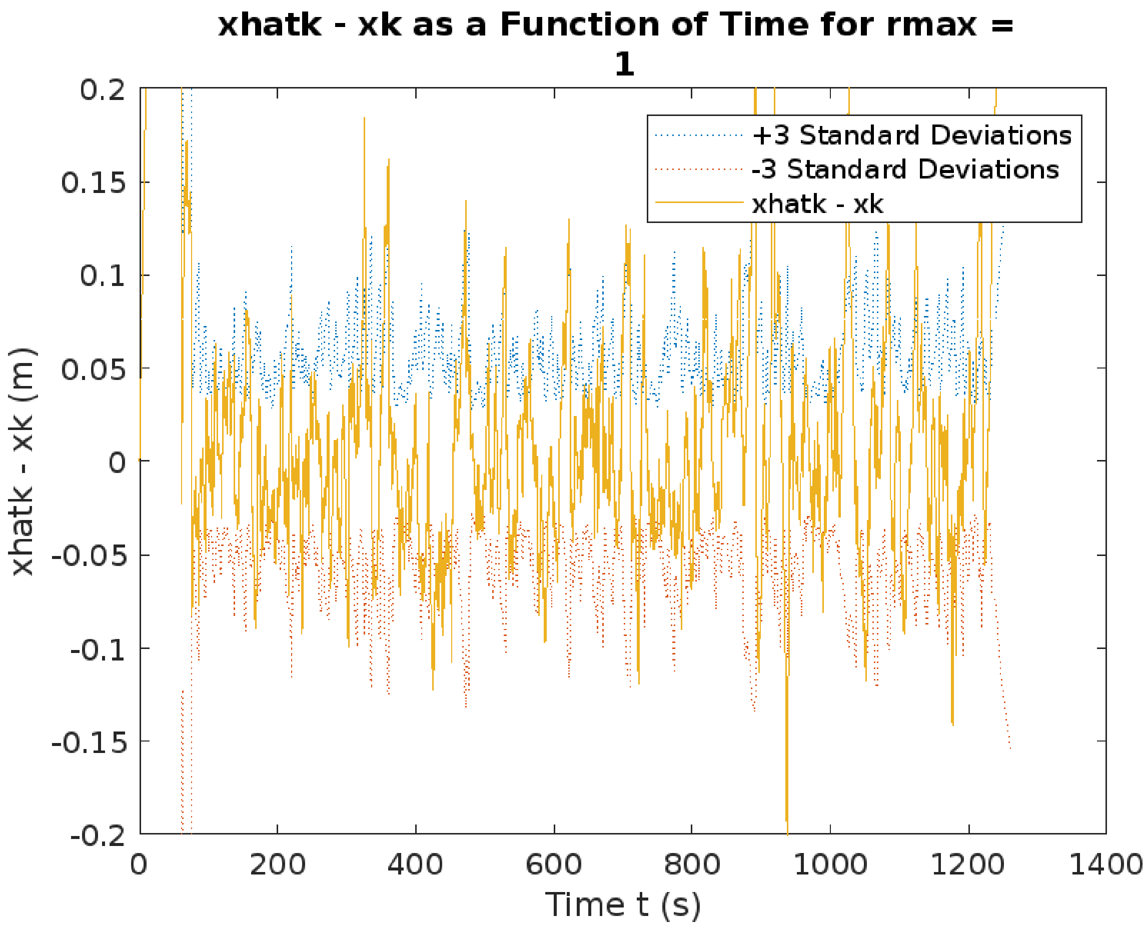

Figure 14: Zoomed In View

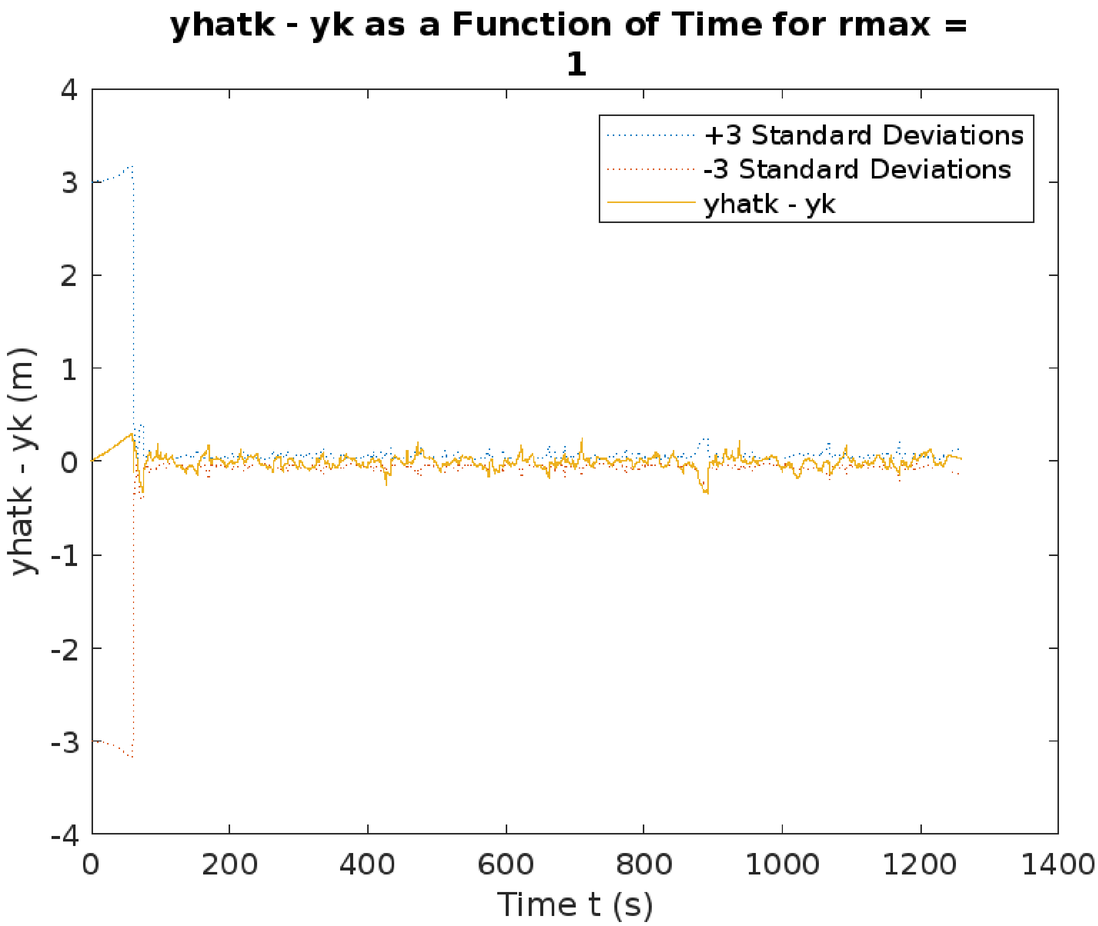

Figure 15: Zoomed Out View

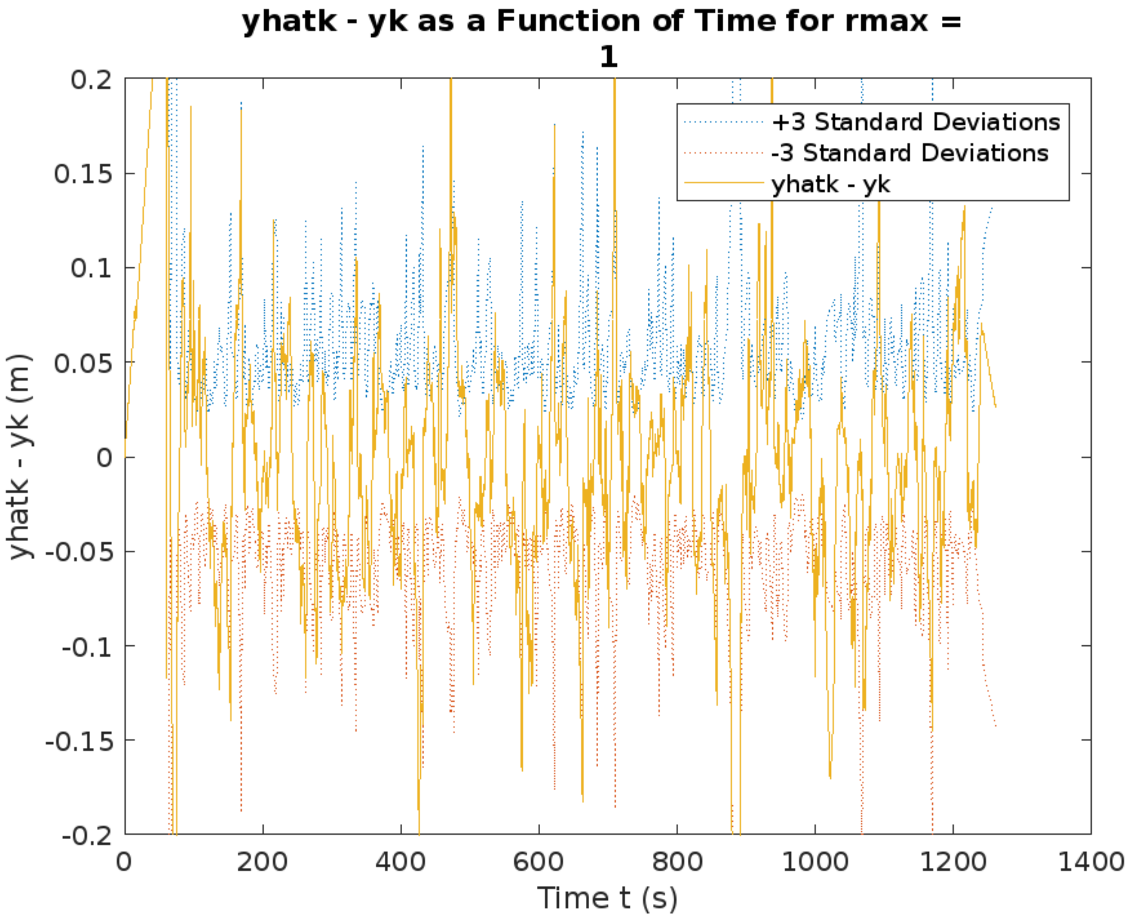

Figure 16: Zoomed In View

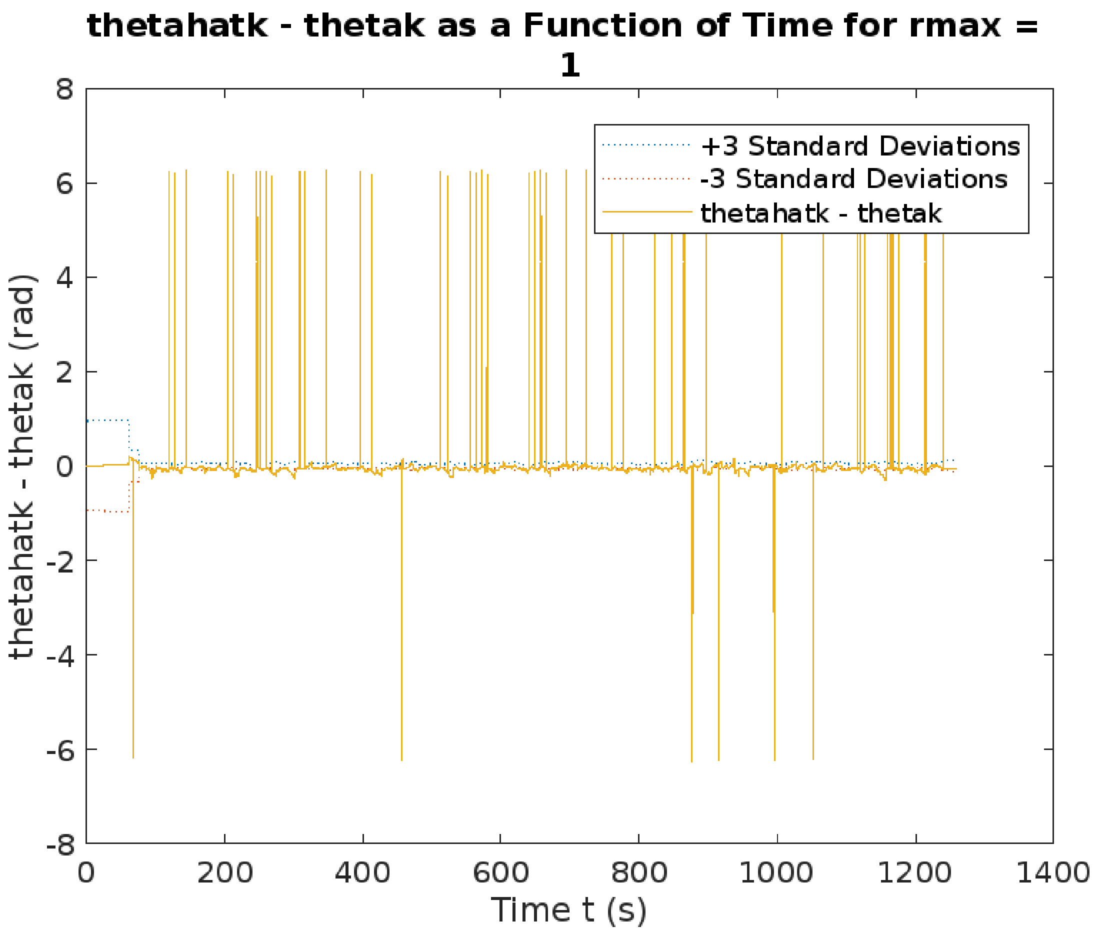

Figure 17: Zoomed Out View

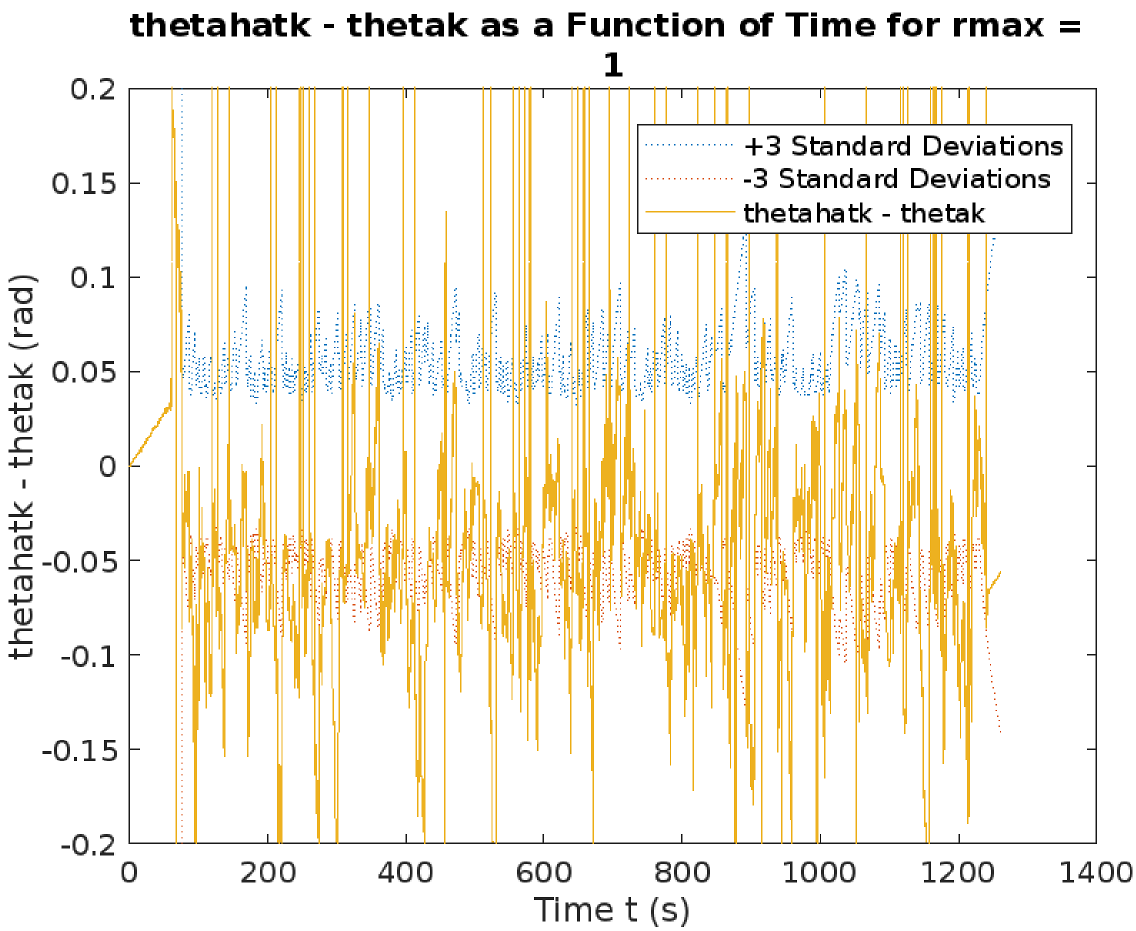

Figure 18: Zoomed In View

4b

$\hat{\mathbf{x}}_0 = (1, 1, 0.1)$:

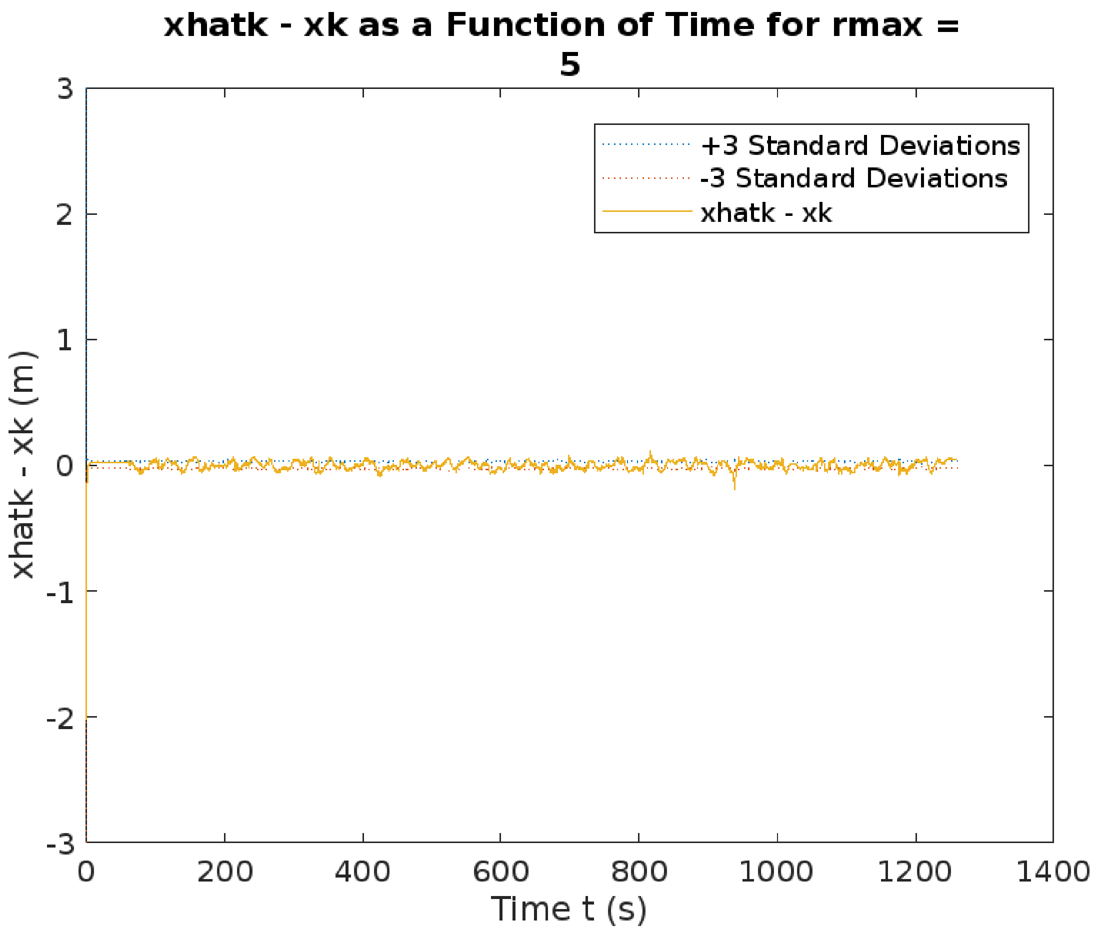

Figure 19: Zoomed Out View

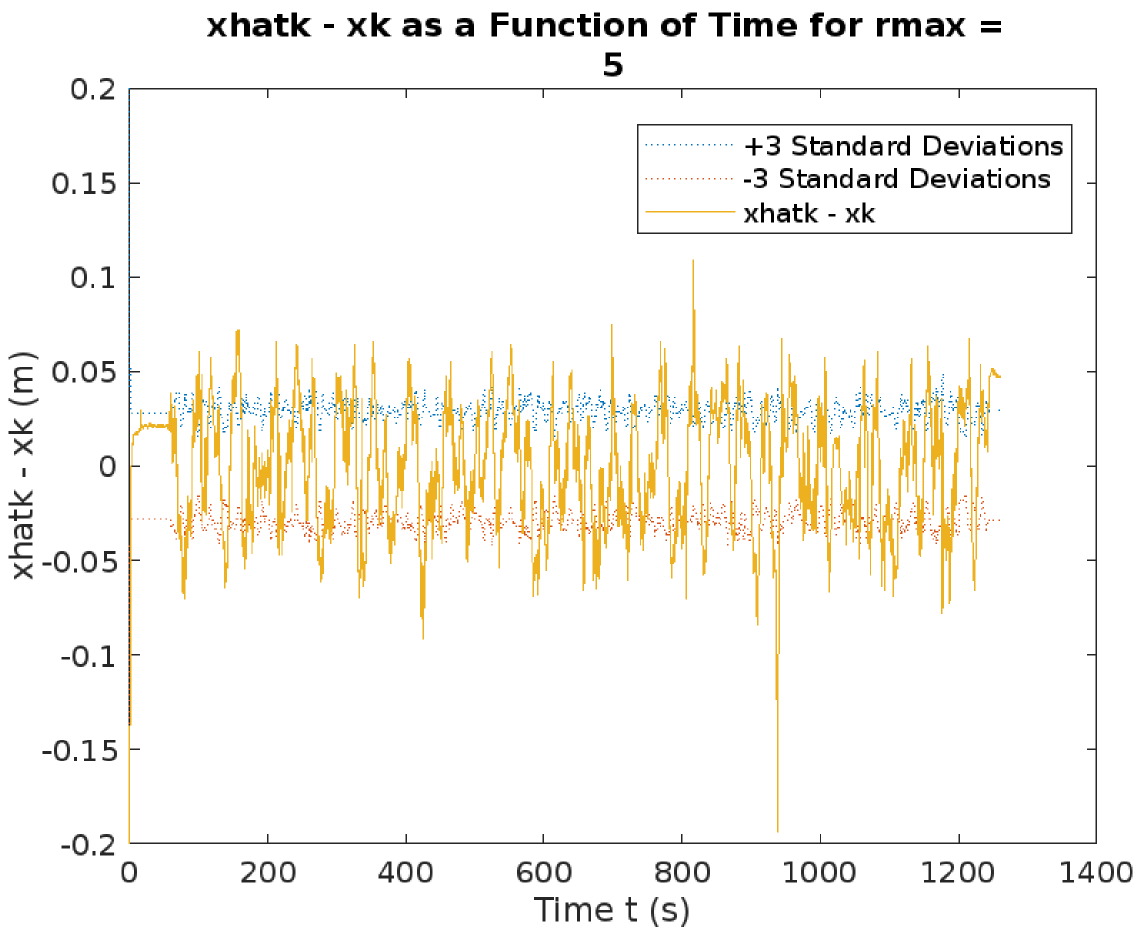

Figure 20: Zoomed In View

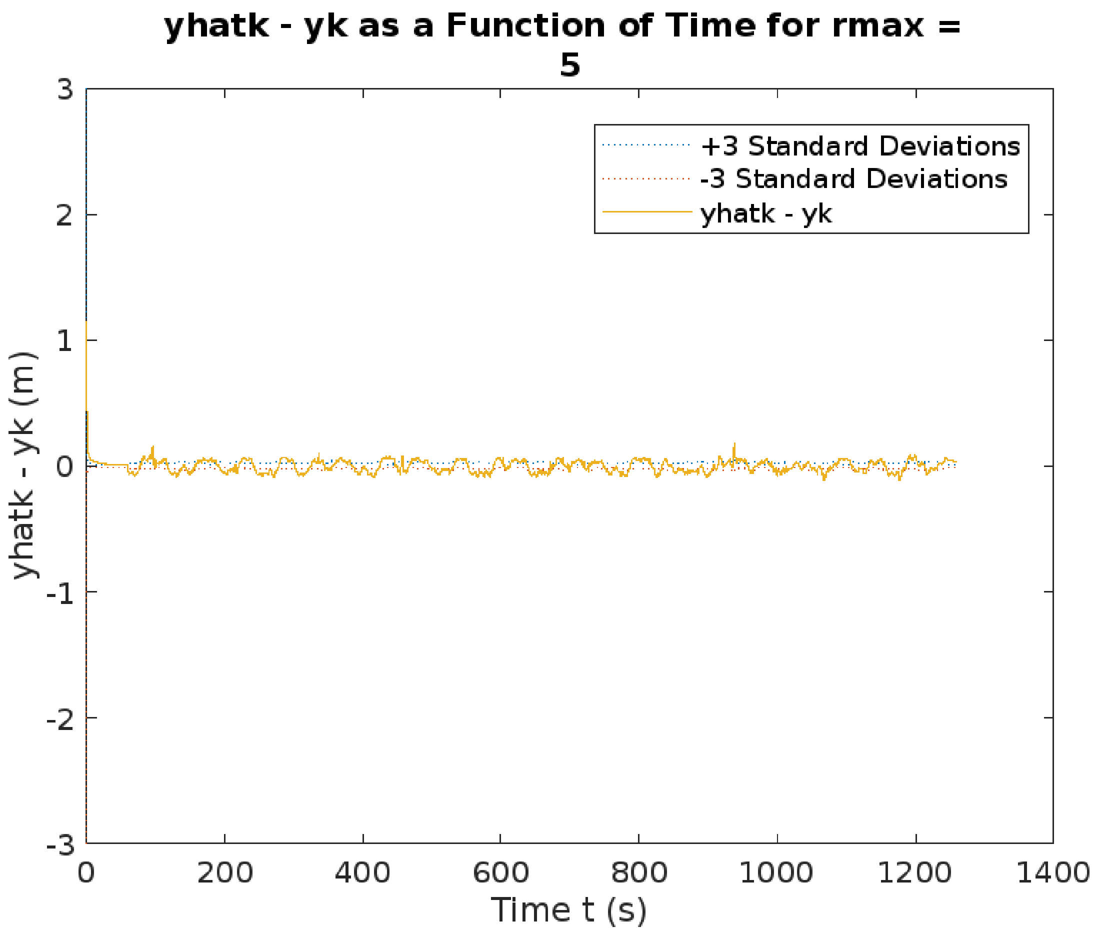

Figure 21: Zoomed Out View

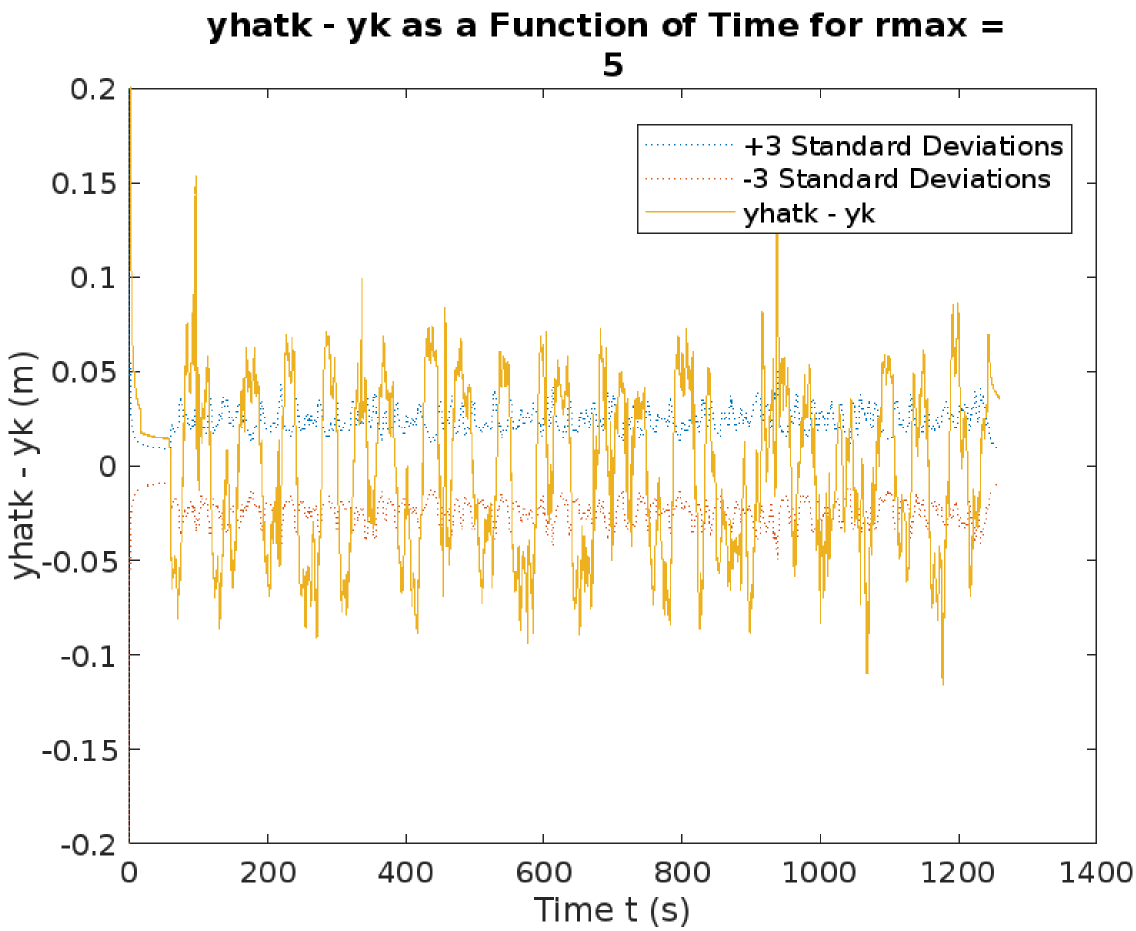

Figure 22: Zoomed In View

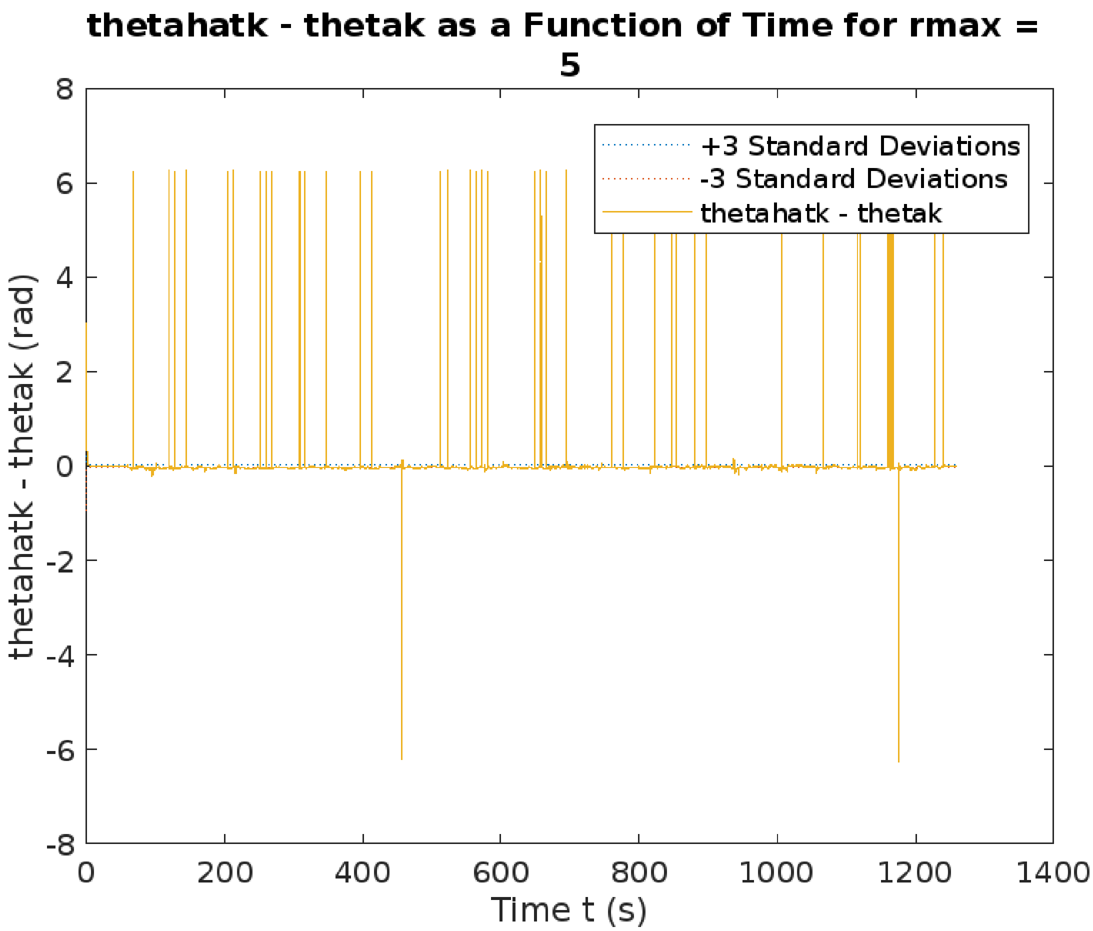

Figure 23: Zoomed Out View

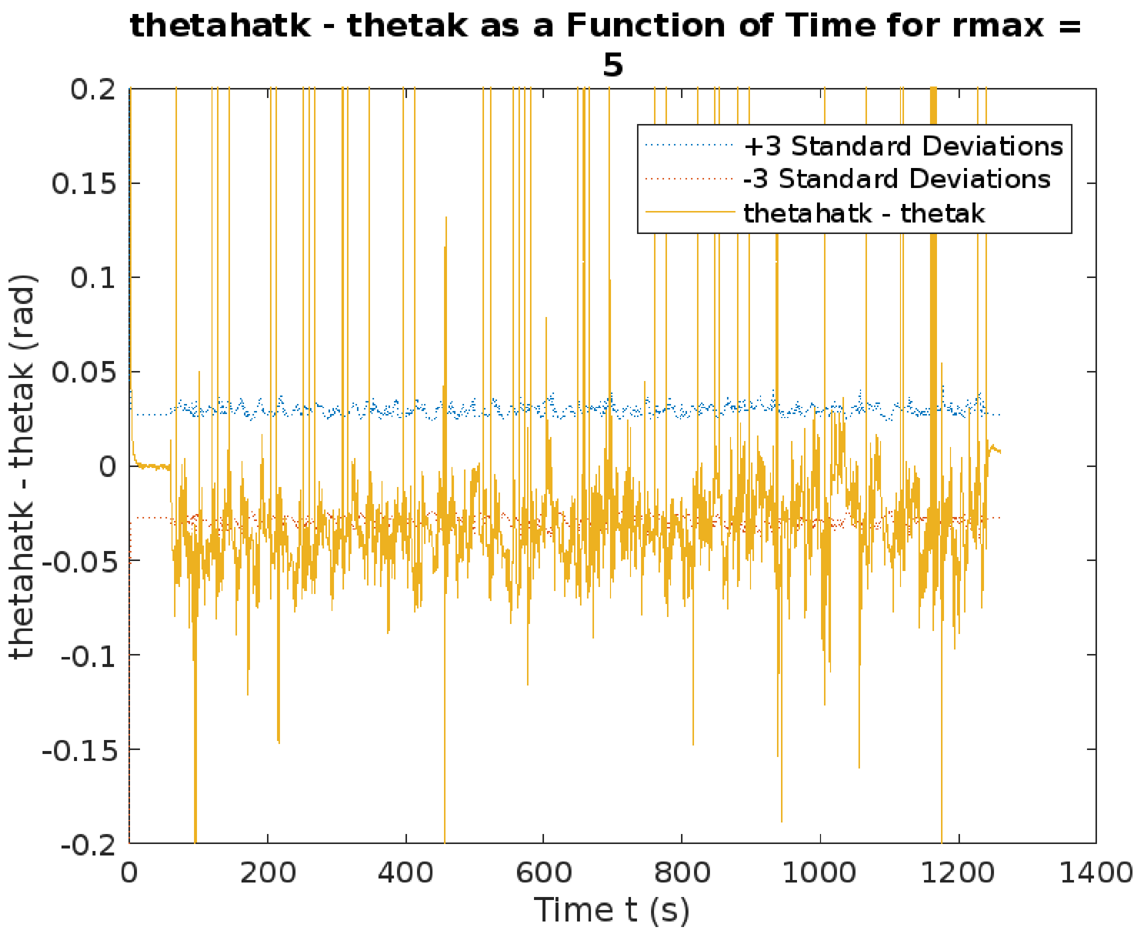

Figure 24: Zoomed In View

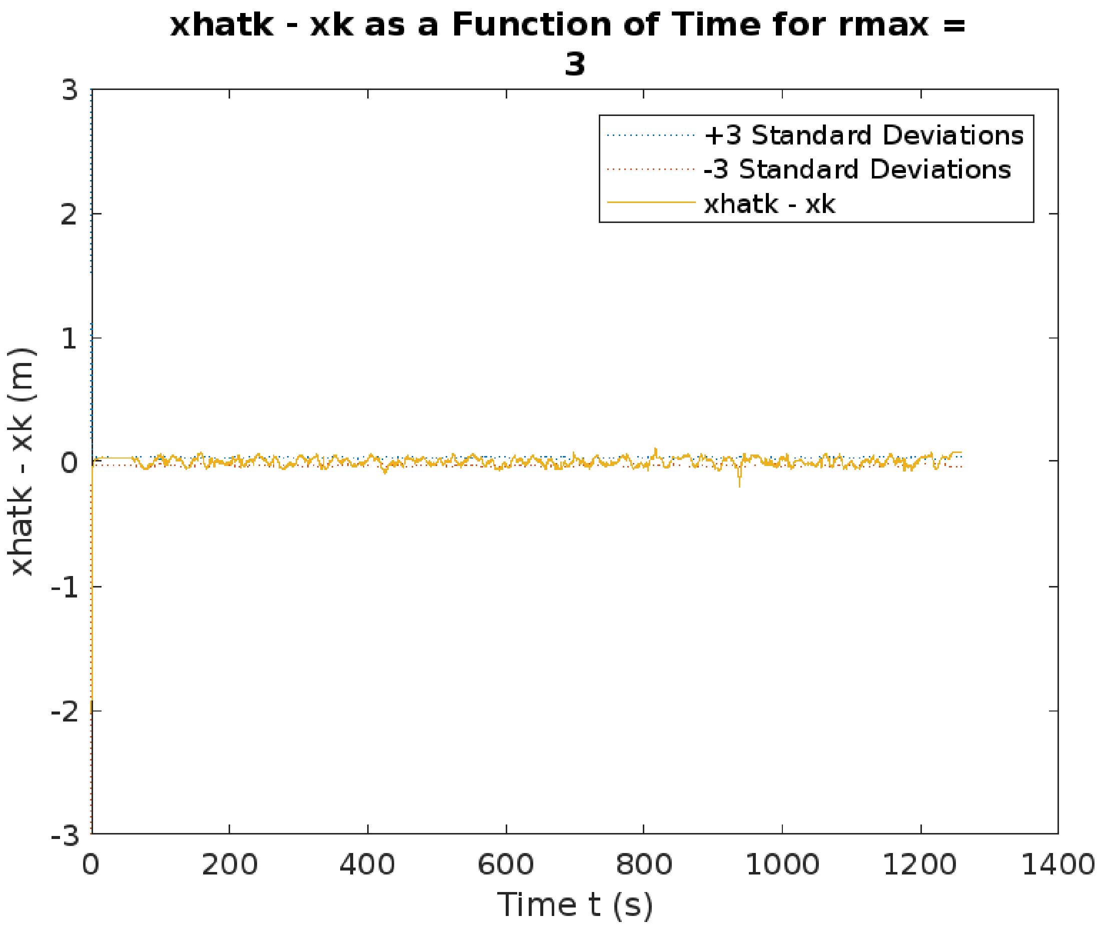

Figure 25: Zoomed Out View

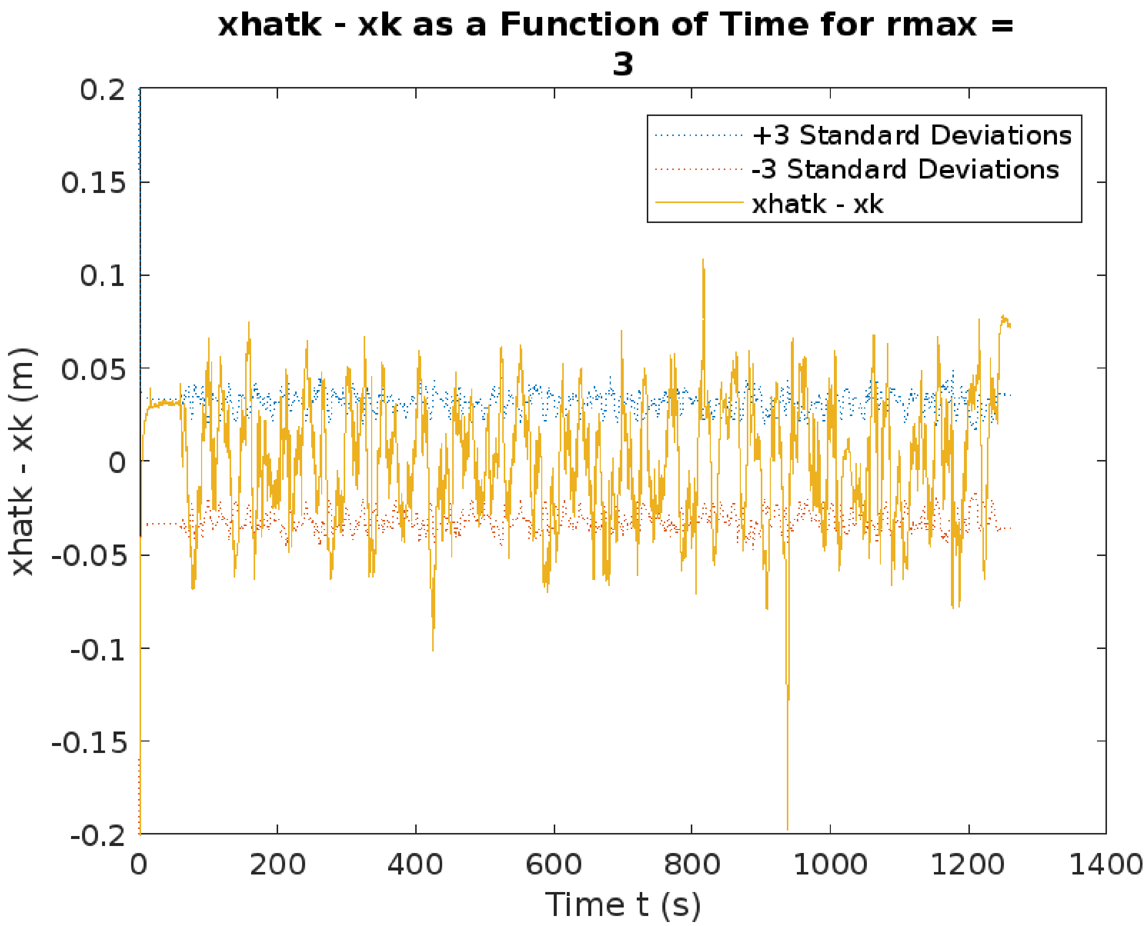

Figure 26: Zoomed In View

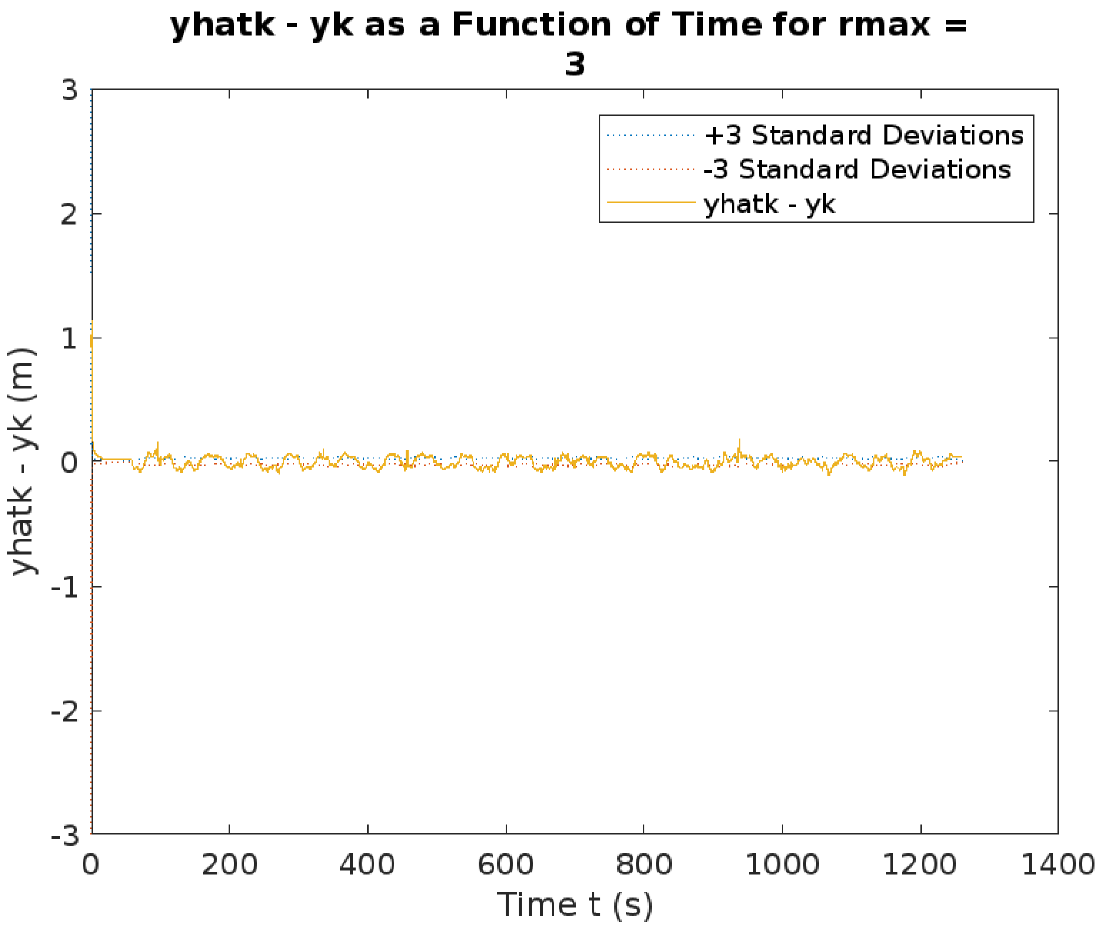

Figure 27: Zoomed Out View

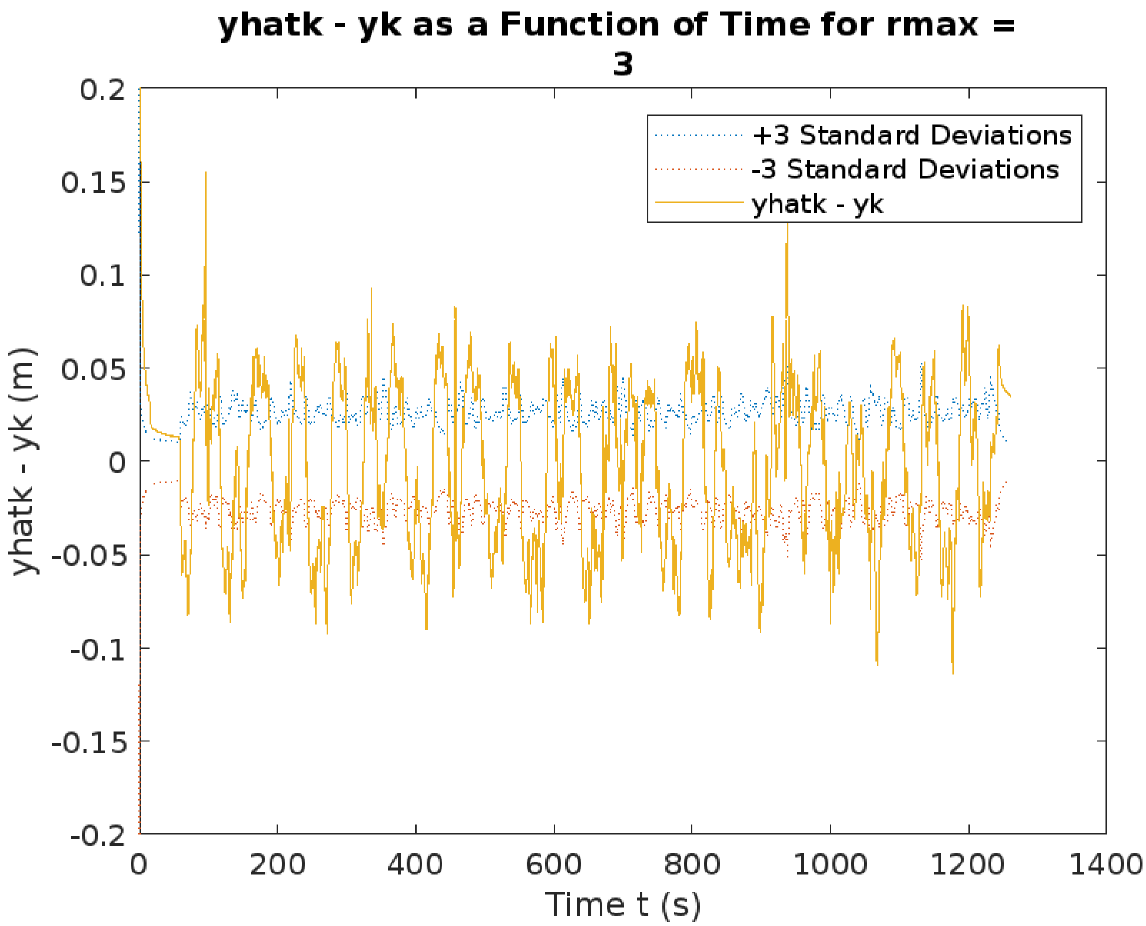

Figure 28: Zoomed In View

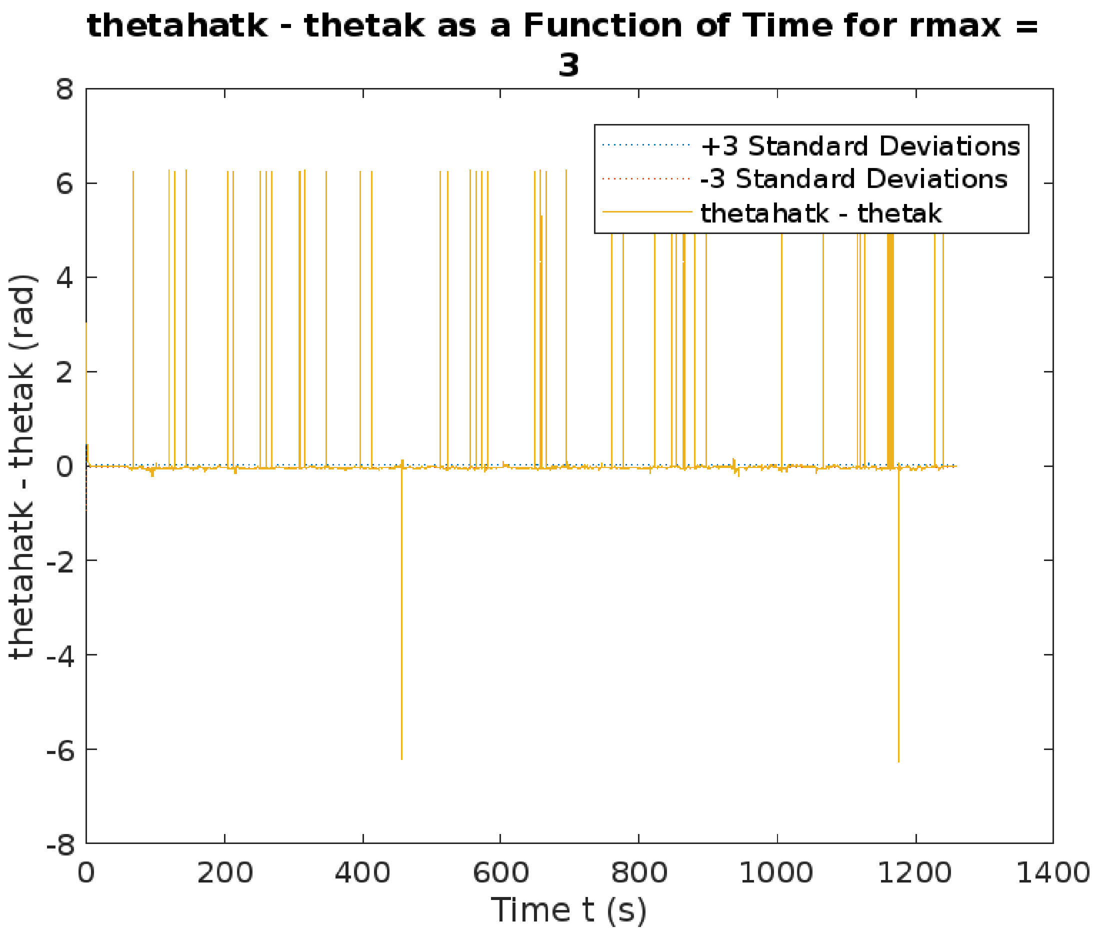

Figure 29: Zoomed Out View

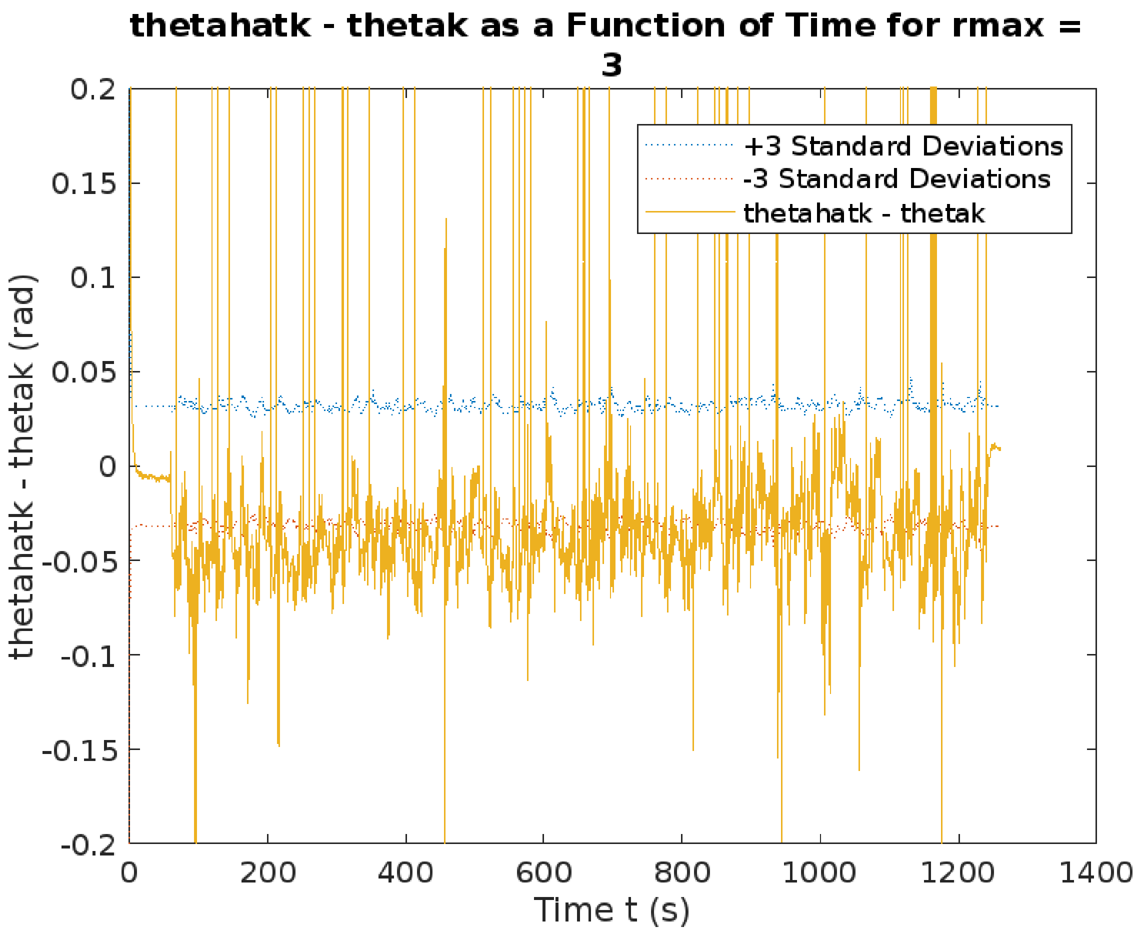

Figure 30: Zoomed In View

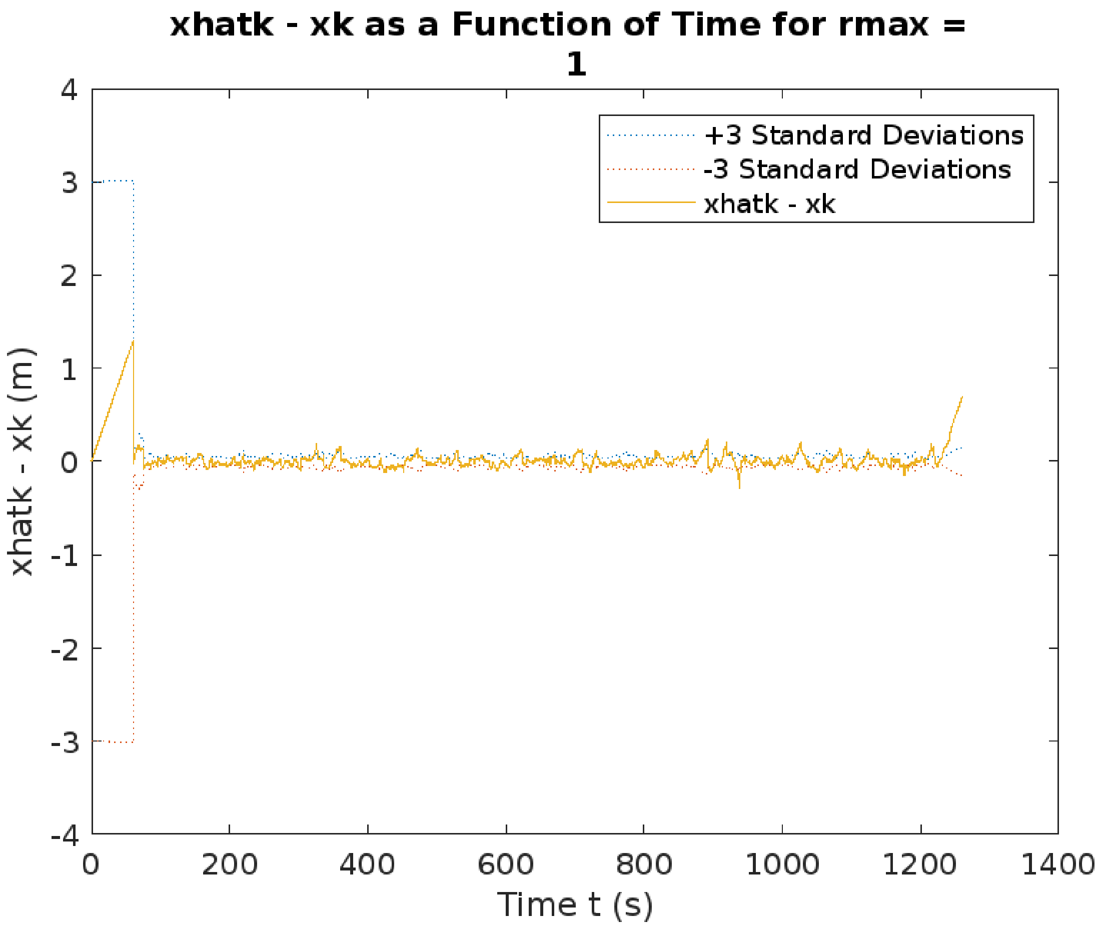

Figure 31: Zoomed Out View

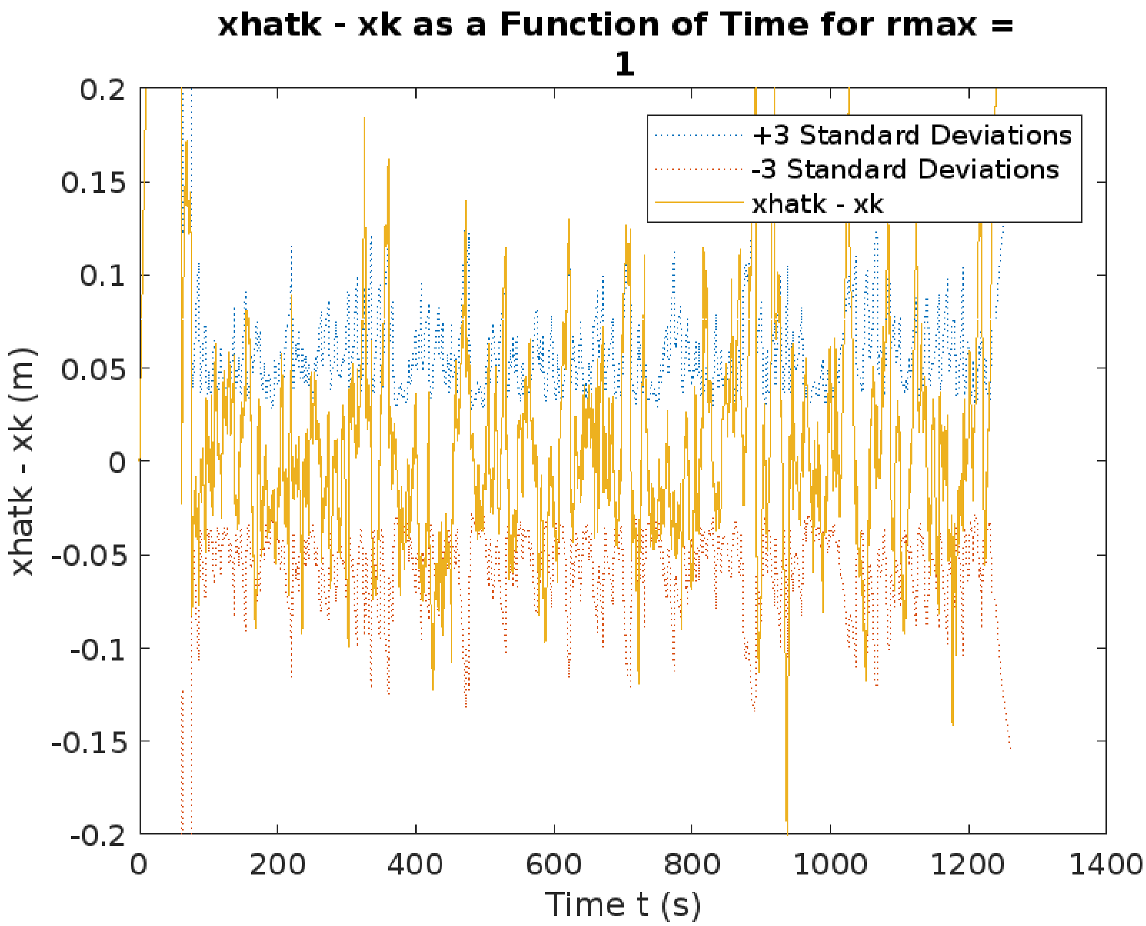

Figure 32: Zoomed In View

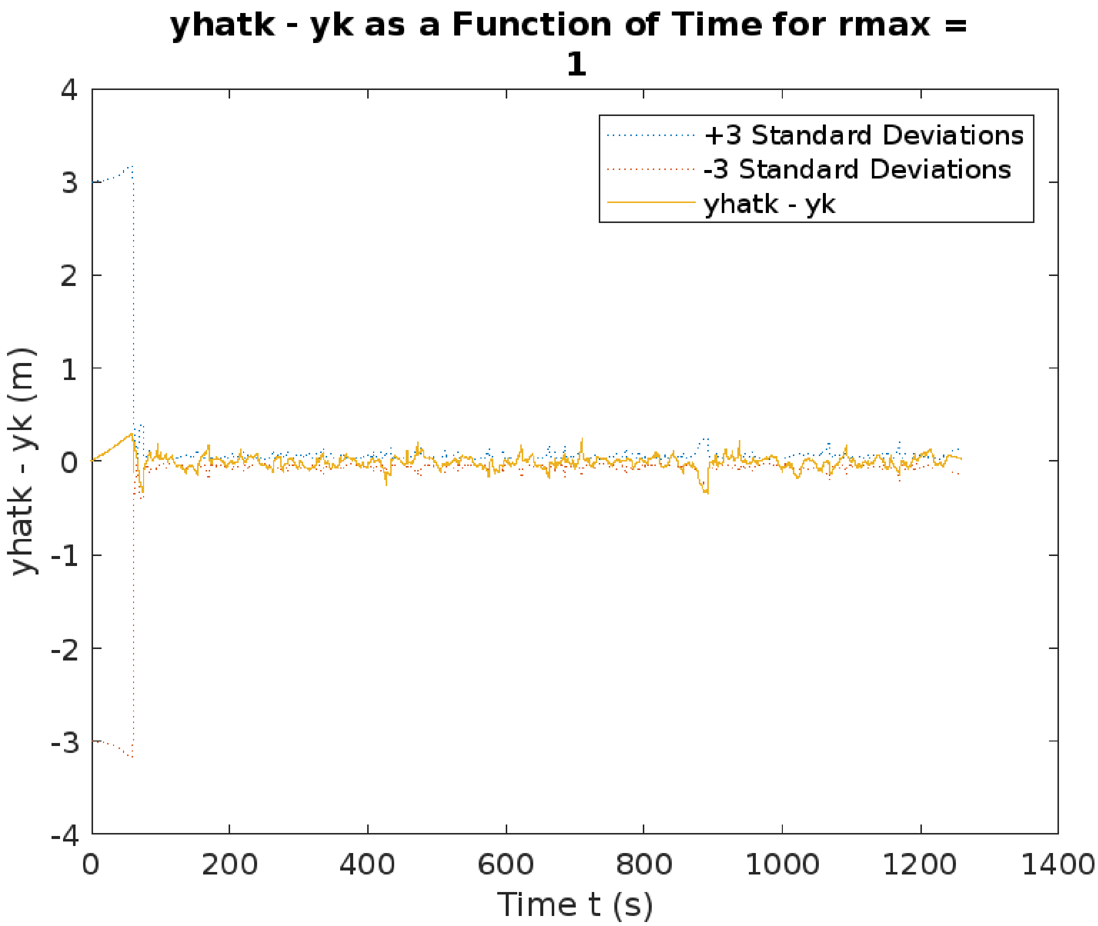

Figure 33: Zoomed Out View

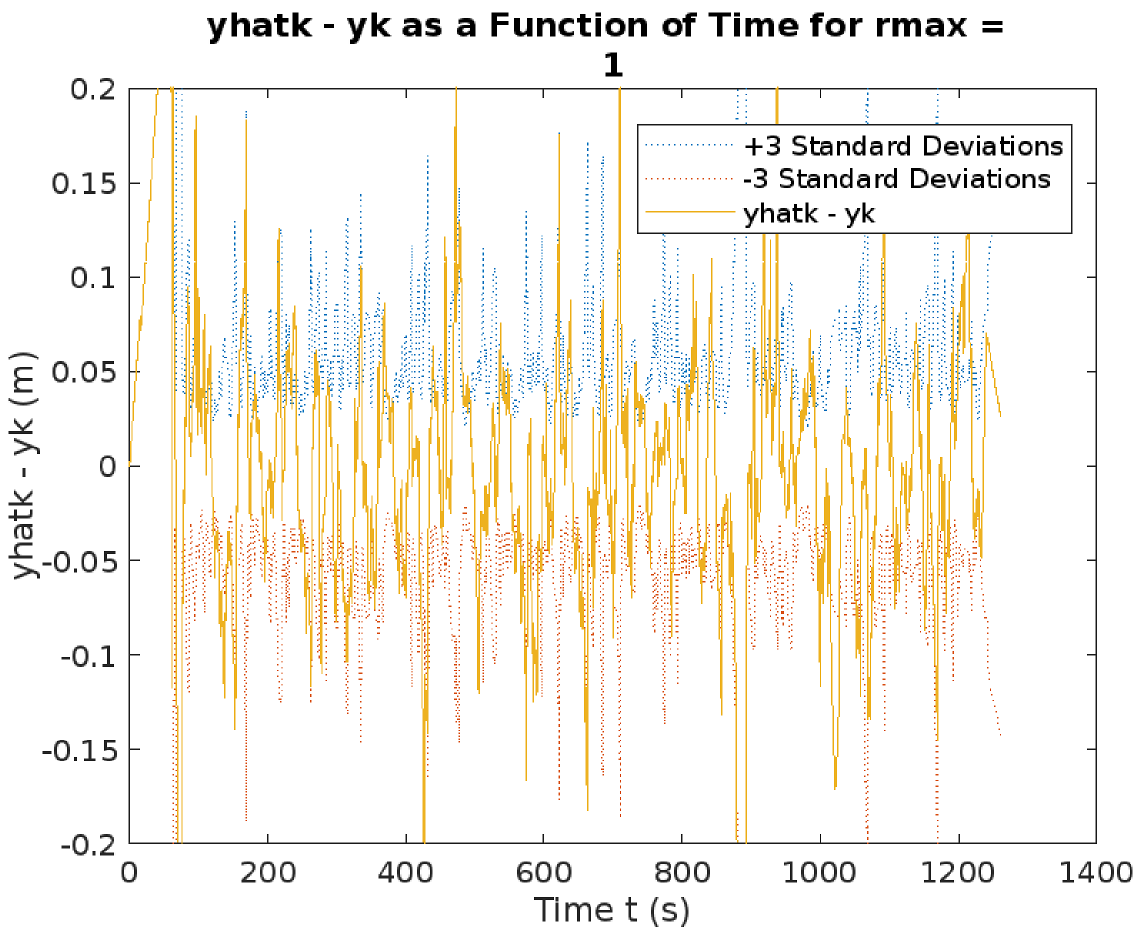

Figure 34: Zoomed In View

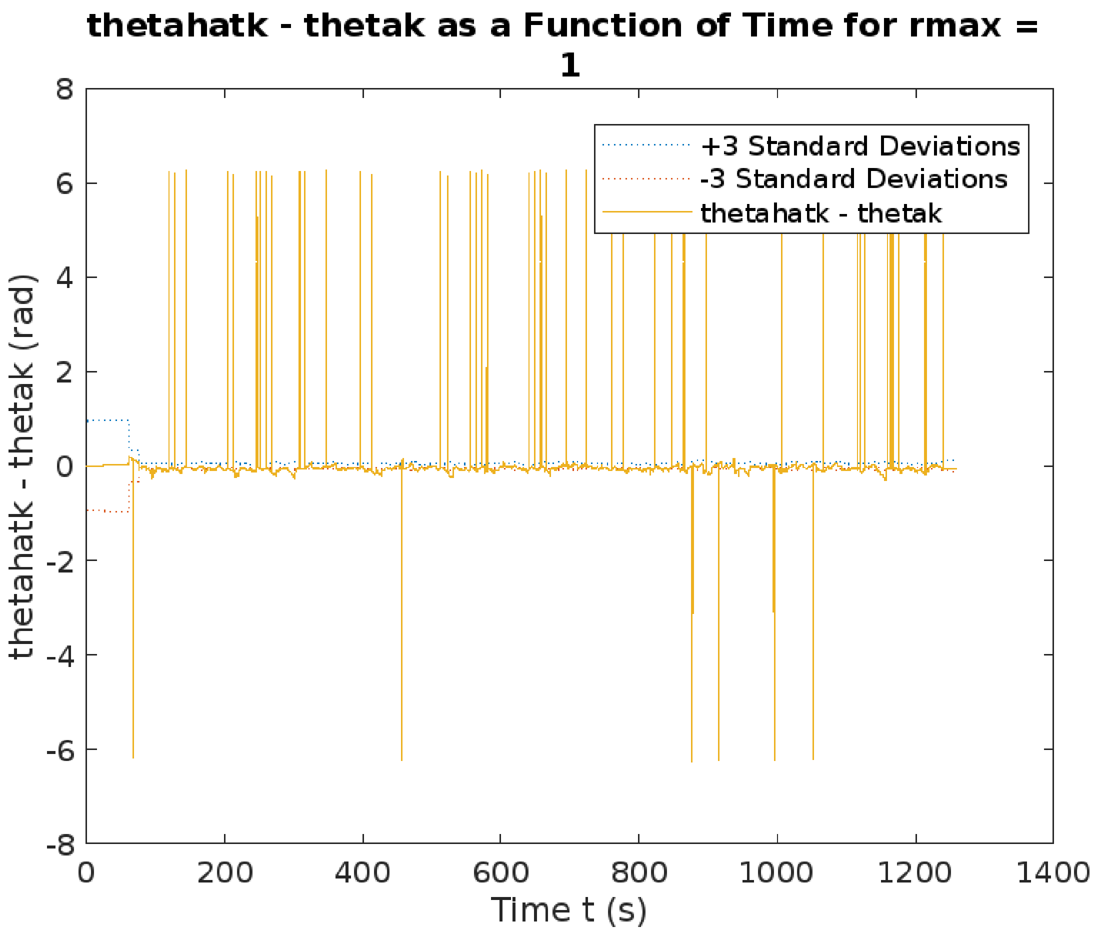

Figure 35: Zoomed Out View

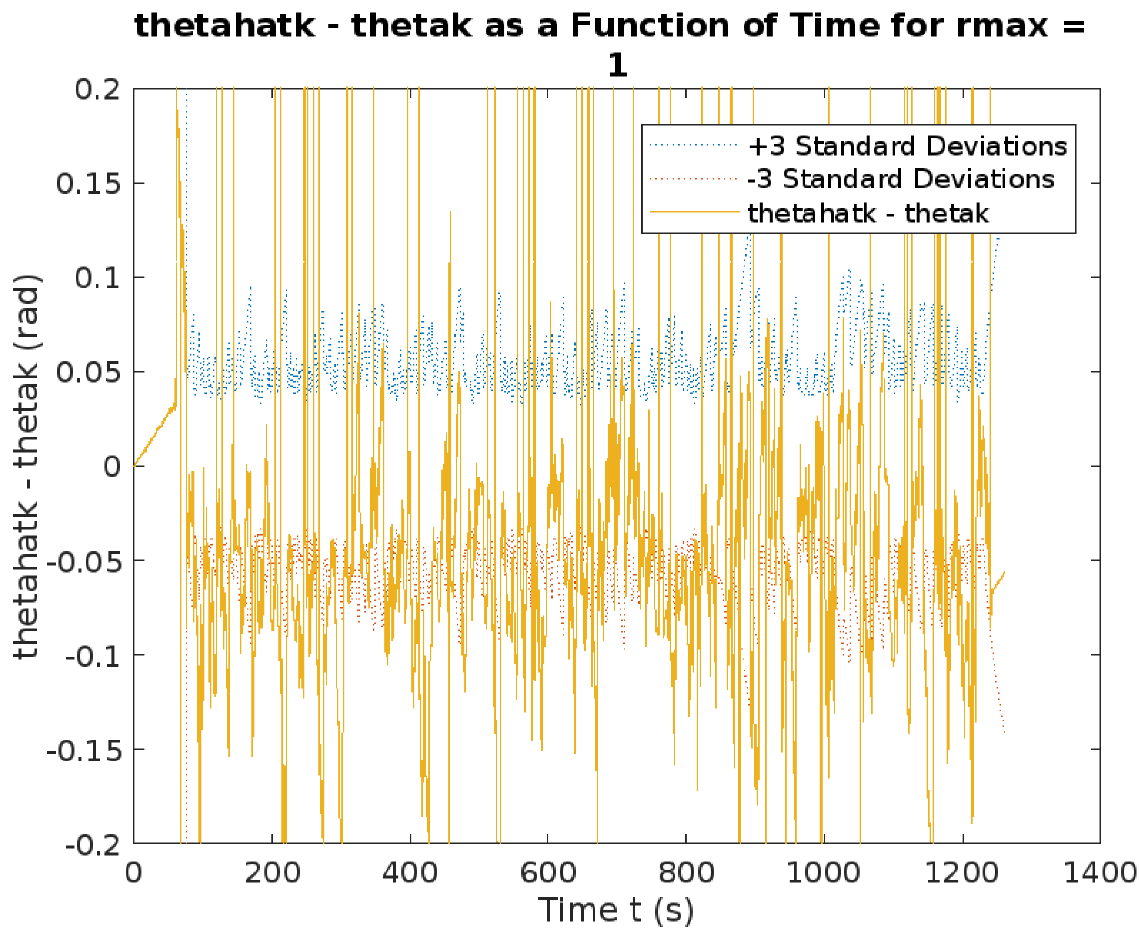

Figure 36: Zoomed In View

The estimator still converges when the initial x and y is -3 but does not converge when the initial x and y is -10.

4c

Jacobians evaluated at $\hat{\mathbf{x}}_k = \mathbf{x}_{\mathbf{true,k}}$:

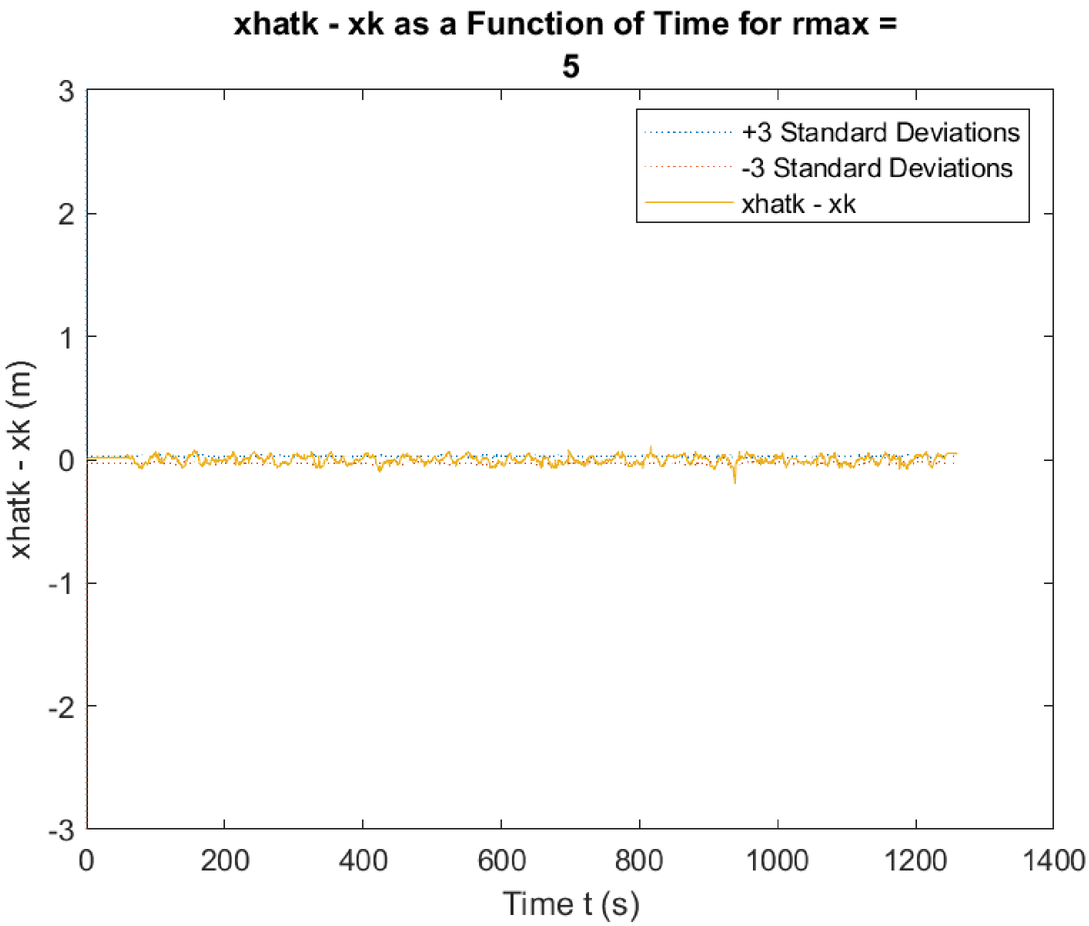

Figure 37: Zoomed Out View

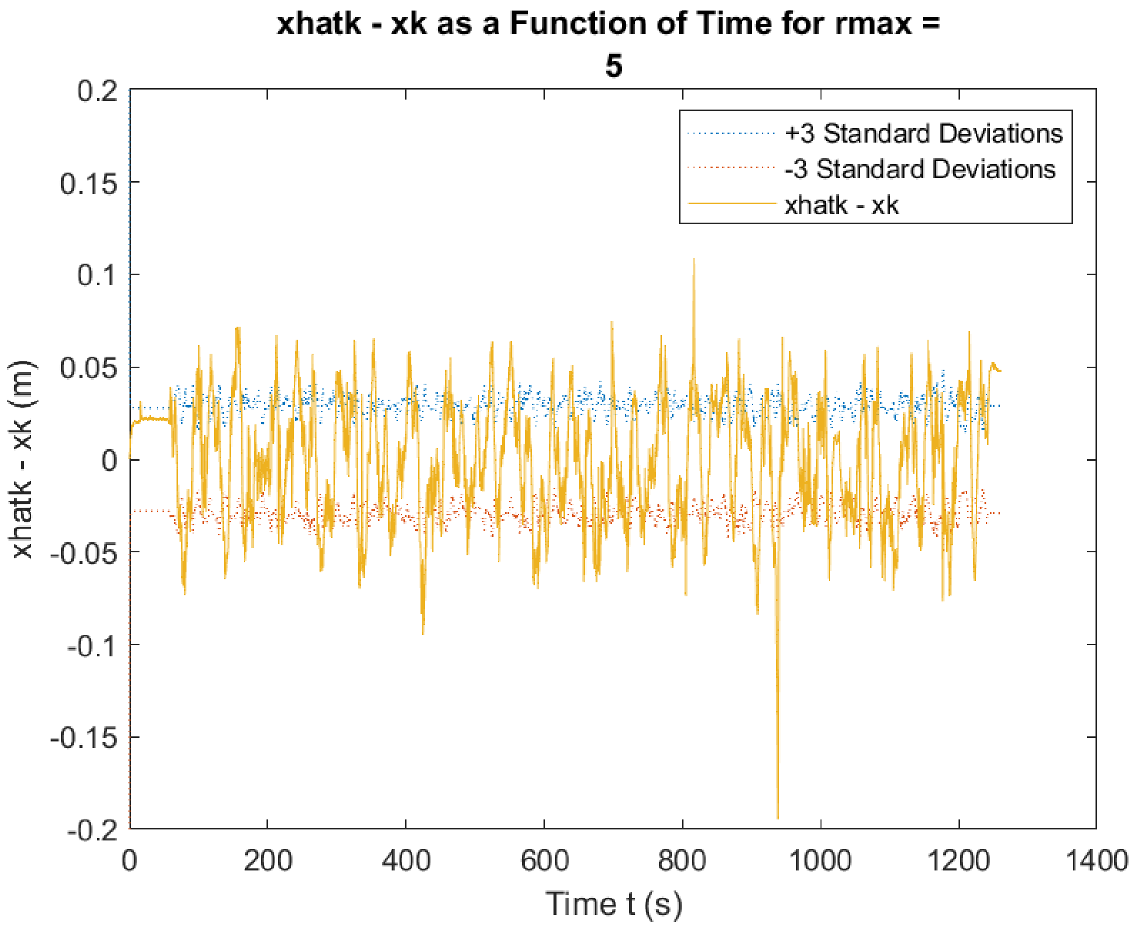

Figure 38: Zoomed In View

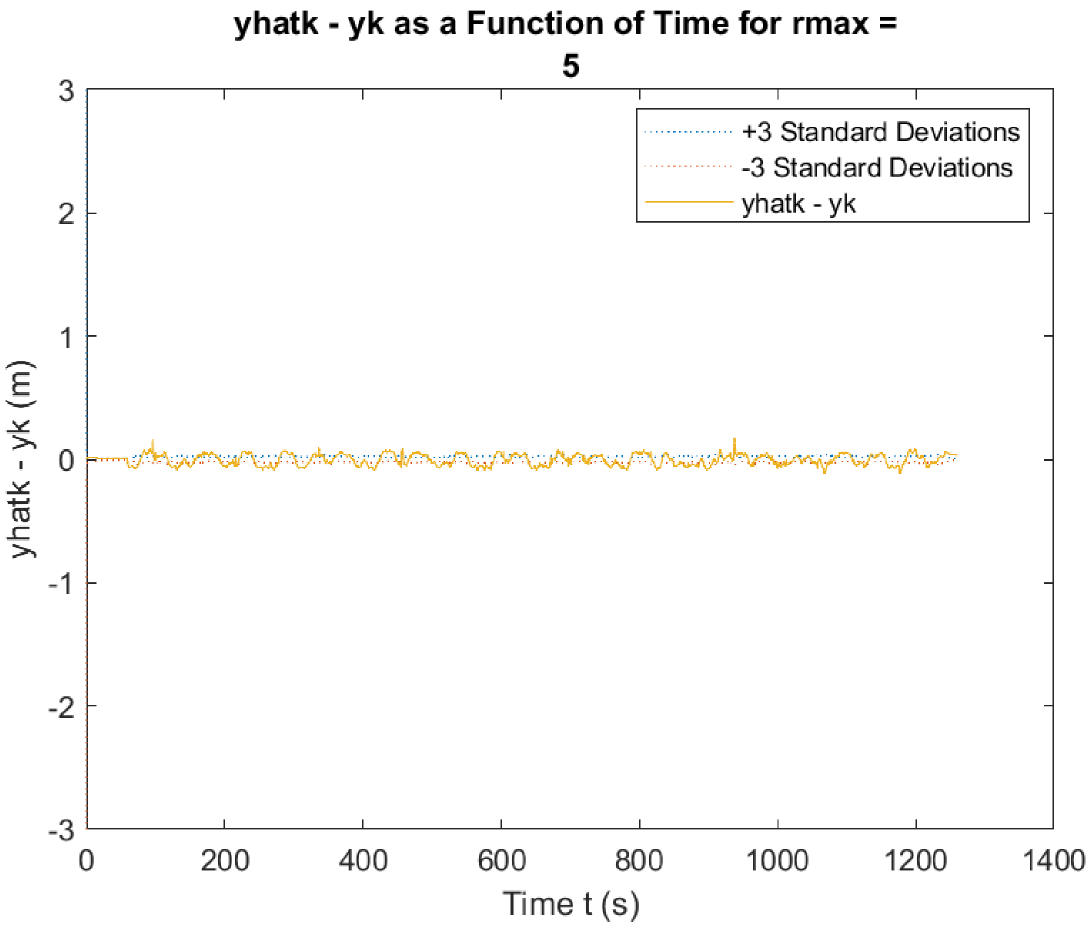

Figure 39: Zoomed Out View

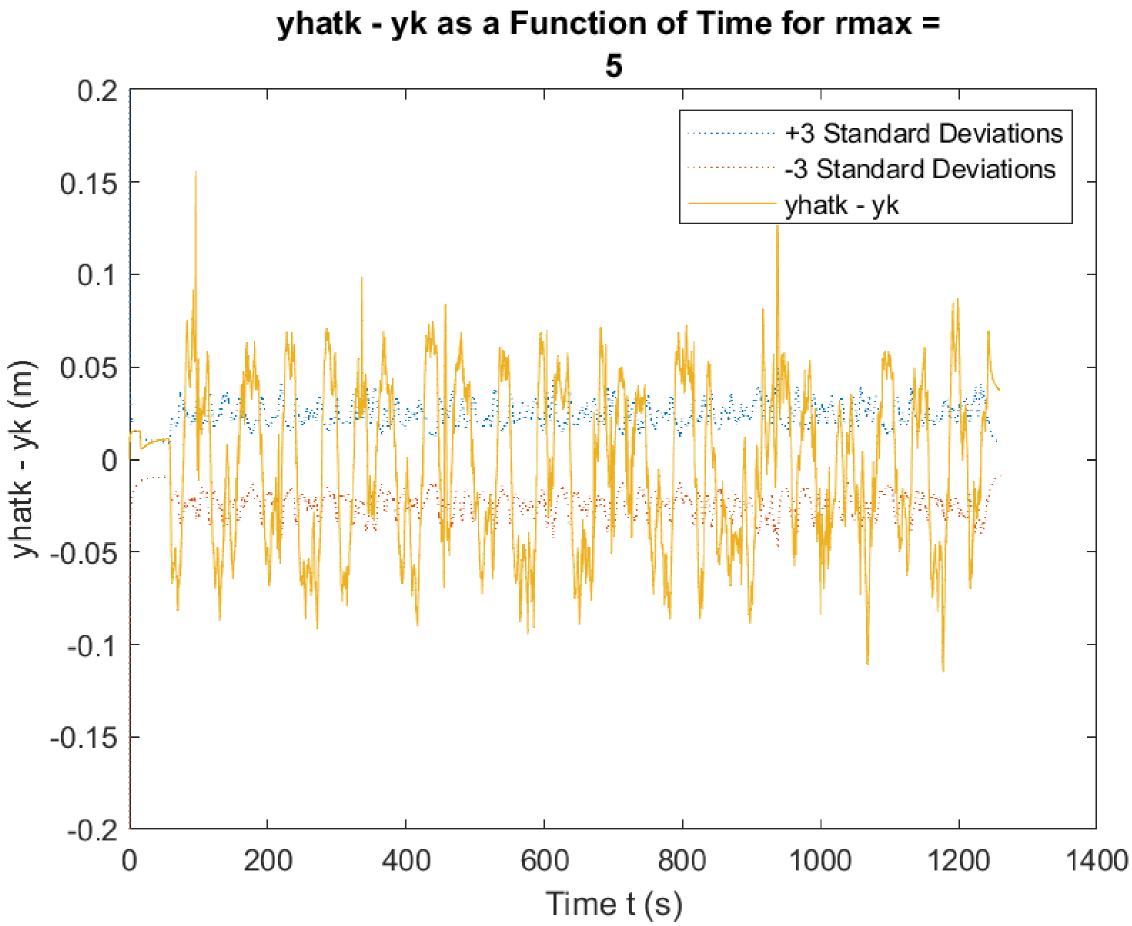

Figure 40: Zoomed In View

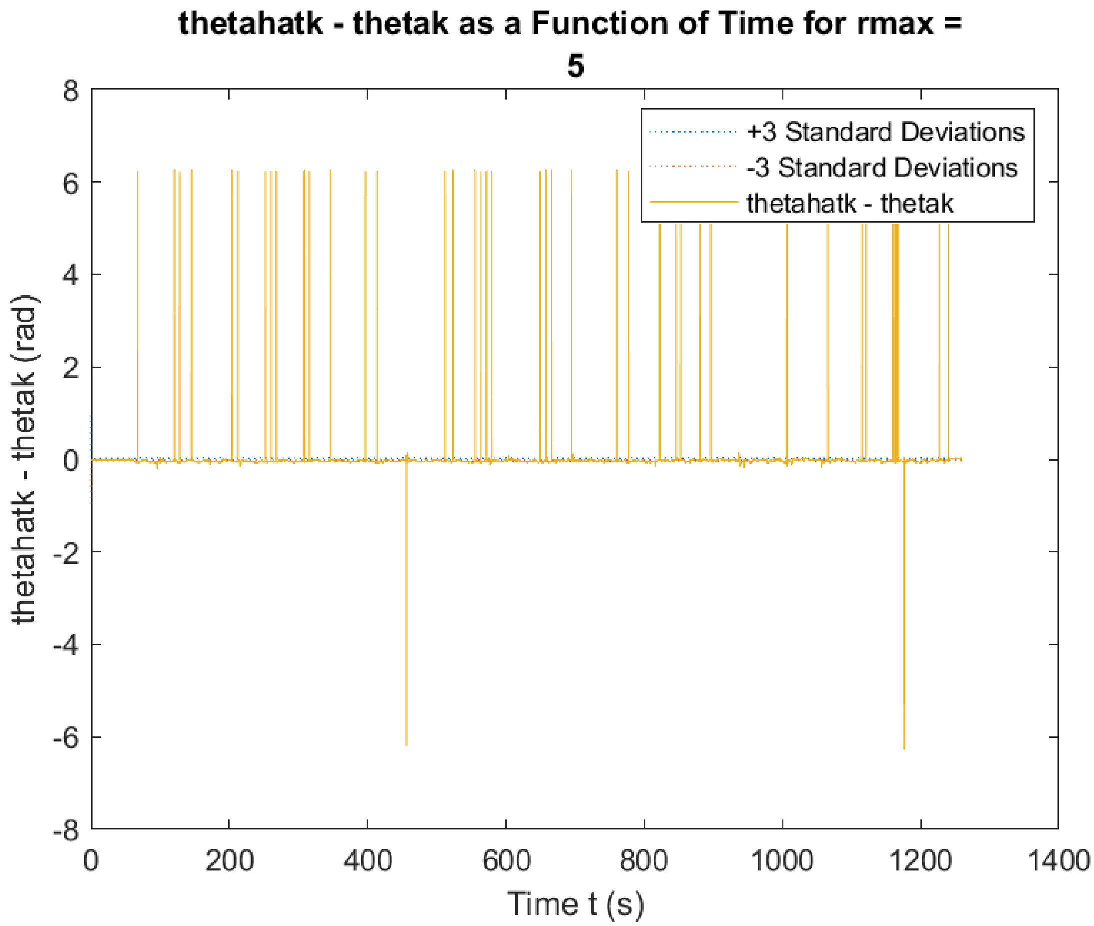

Figure 41: Zoomed Out View

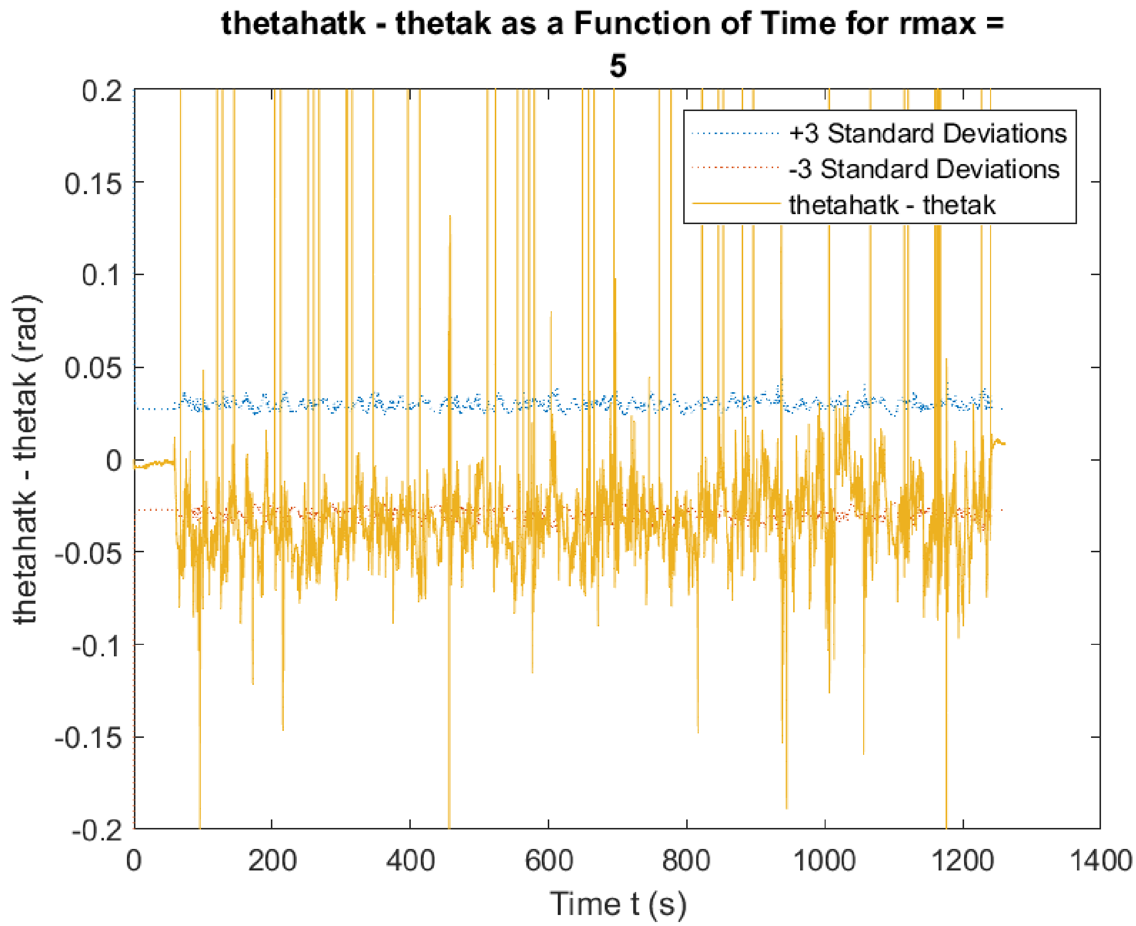

Figure 42: Zoomed In View

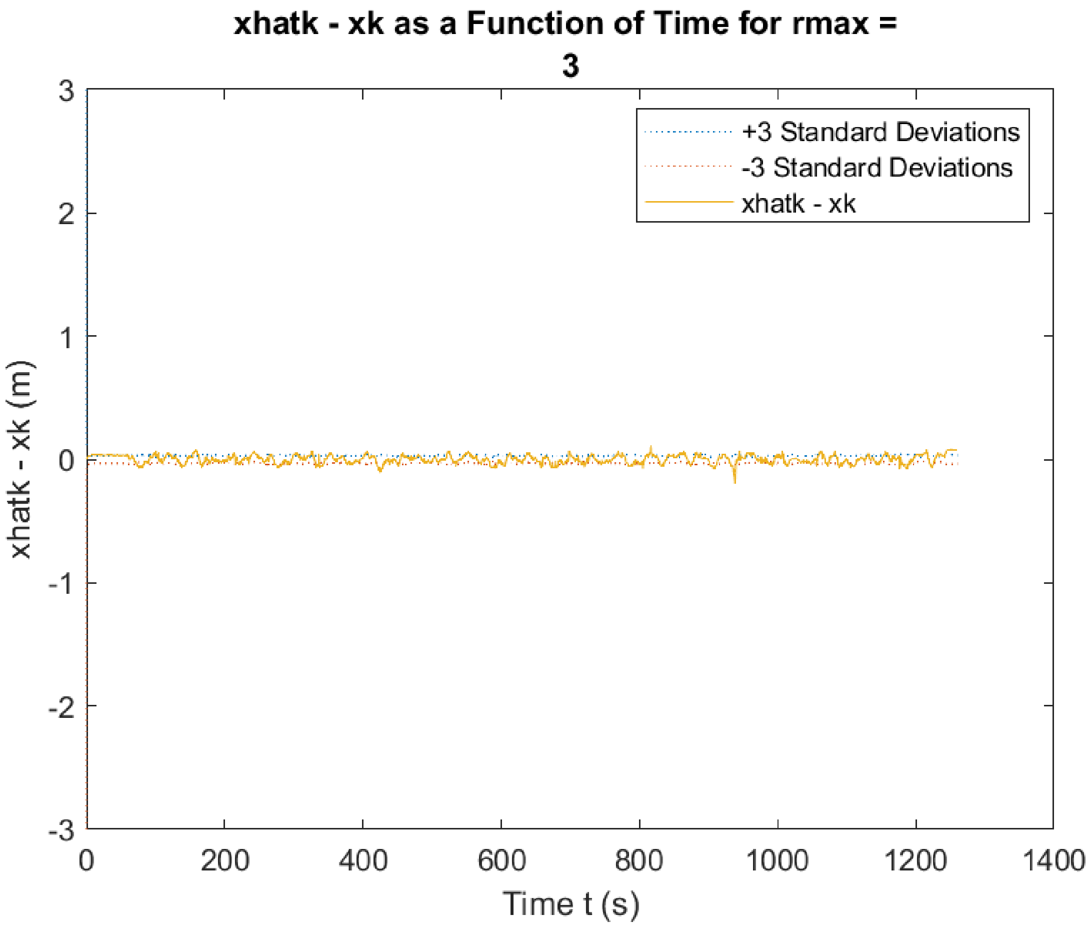

Figure 43: Zoomed Out View

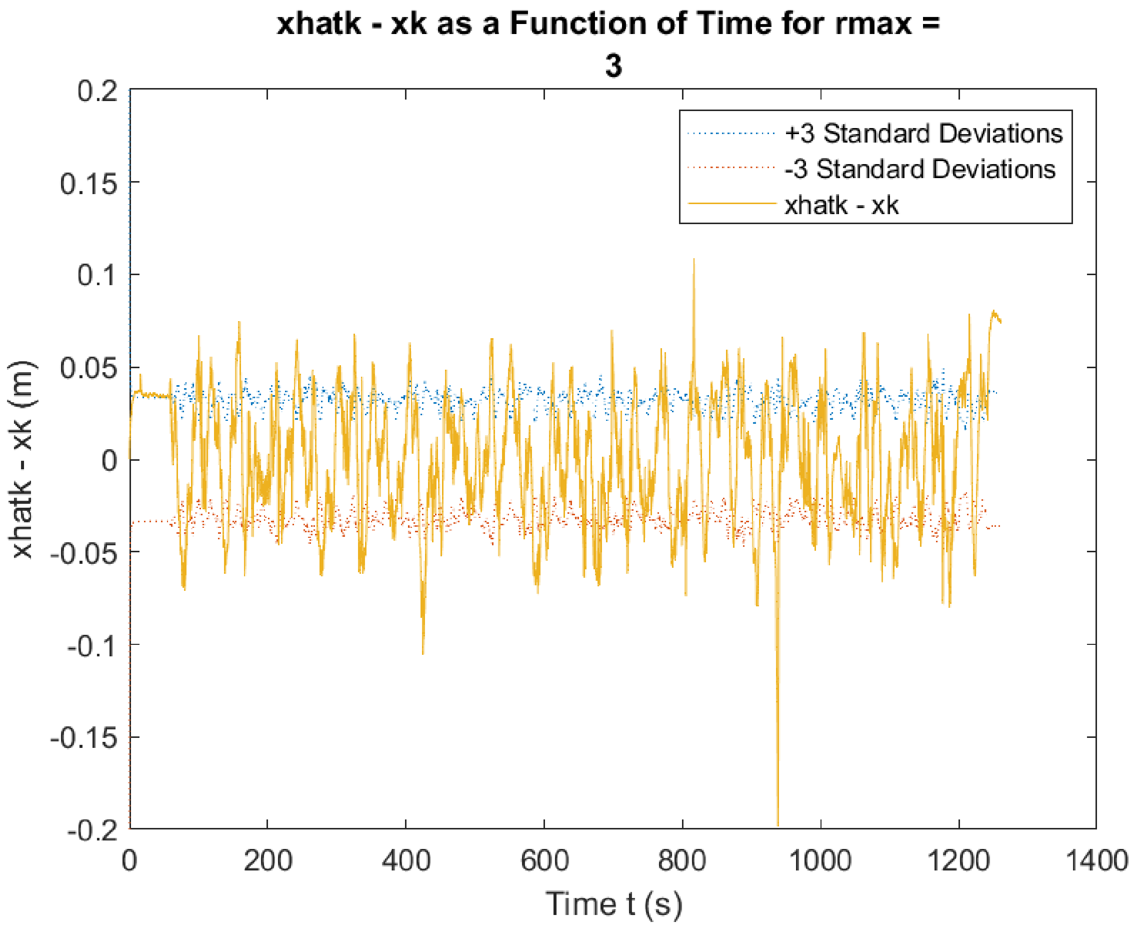

Figure 44: Zoomed In View

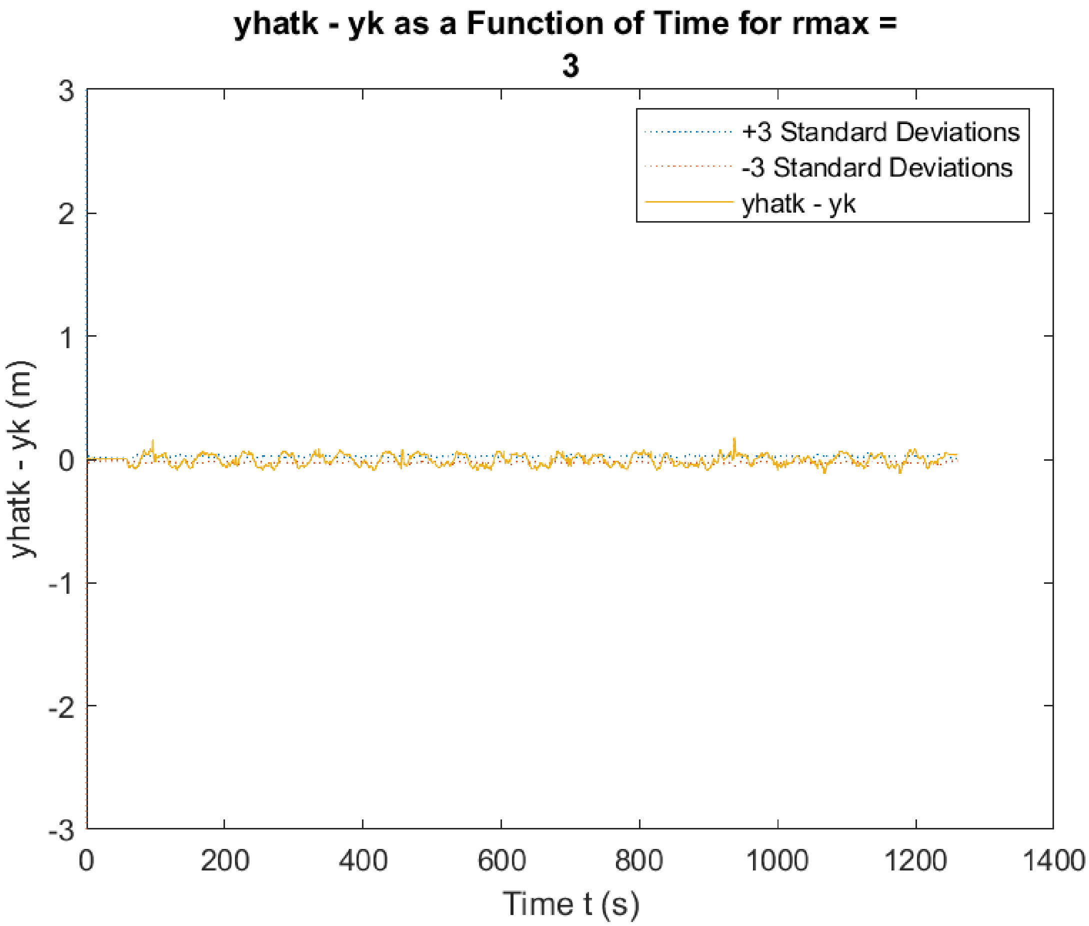

Figure 45: Zoomed Out View

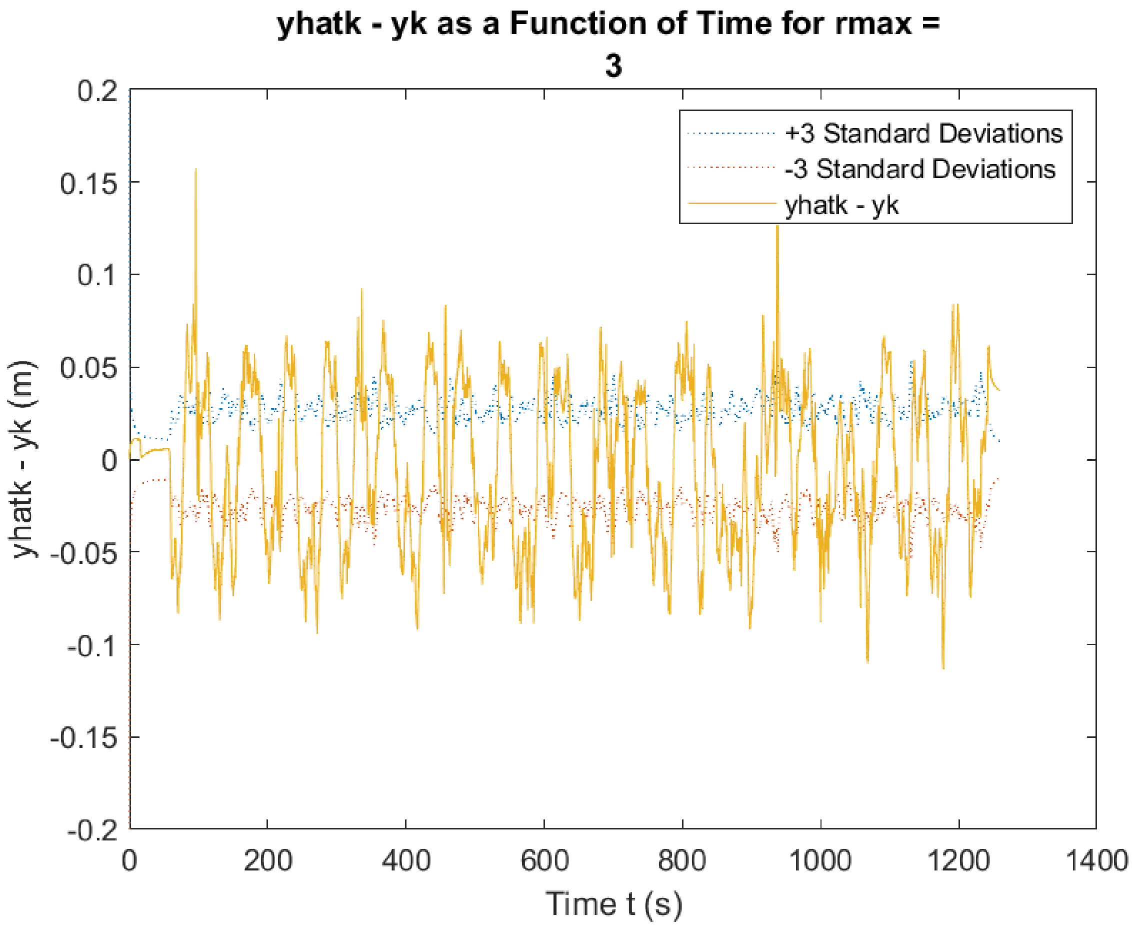

Figure 46: Zoomed In View

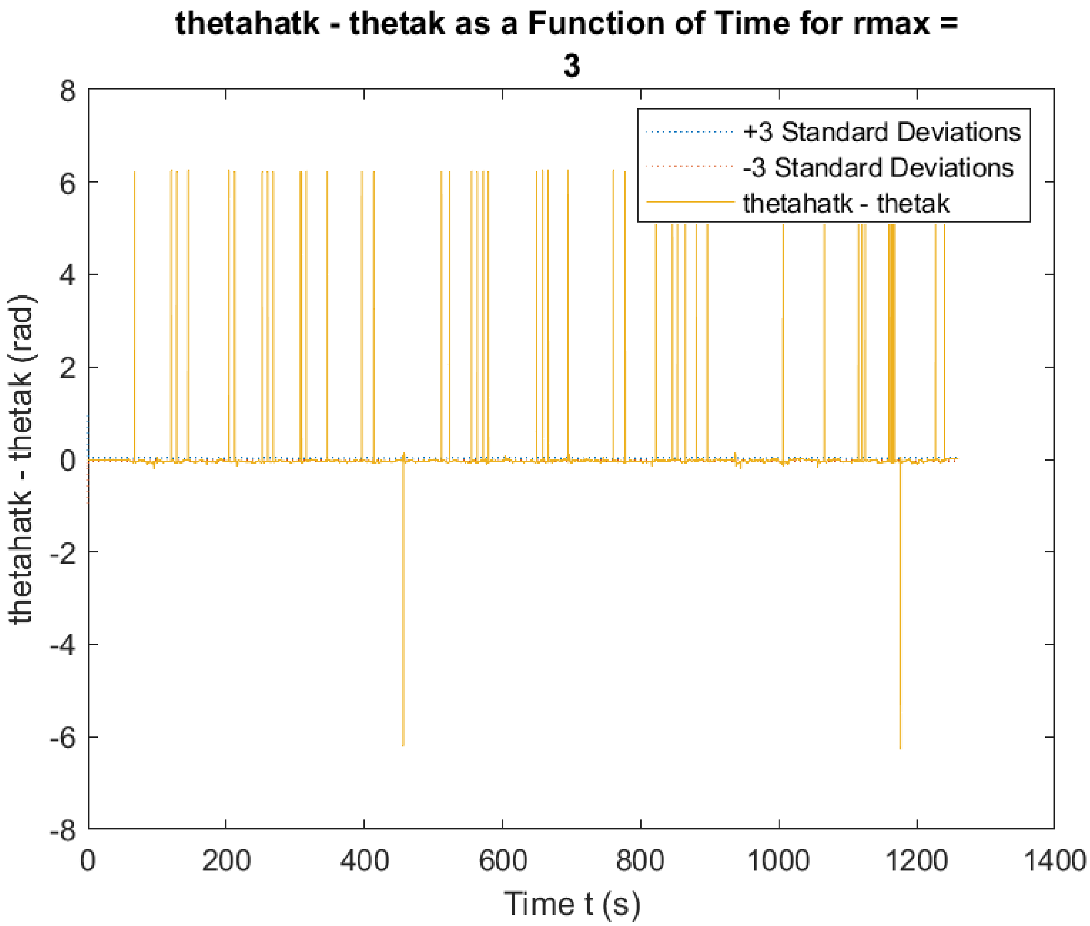

Figure 47: Zoomed Out View

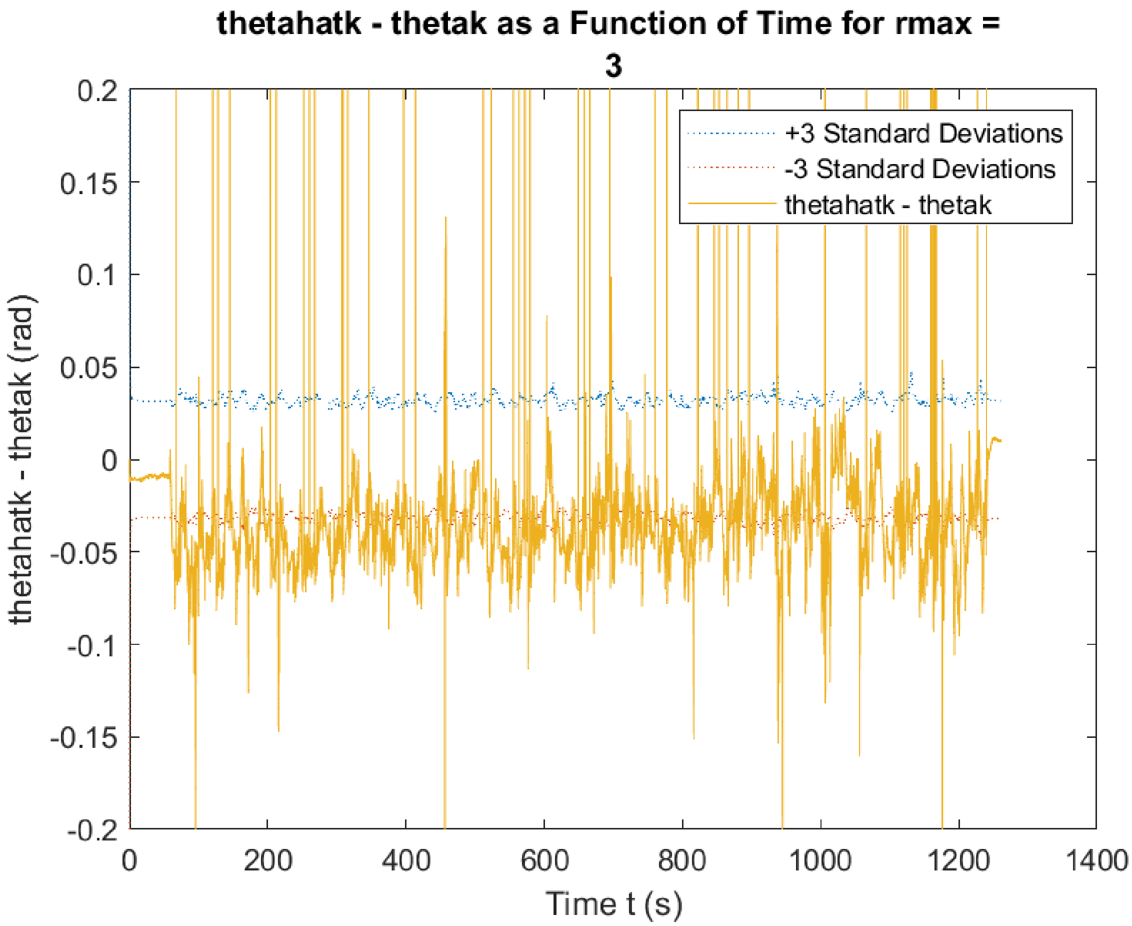

Figure 48: Zoomed In View

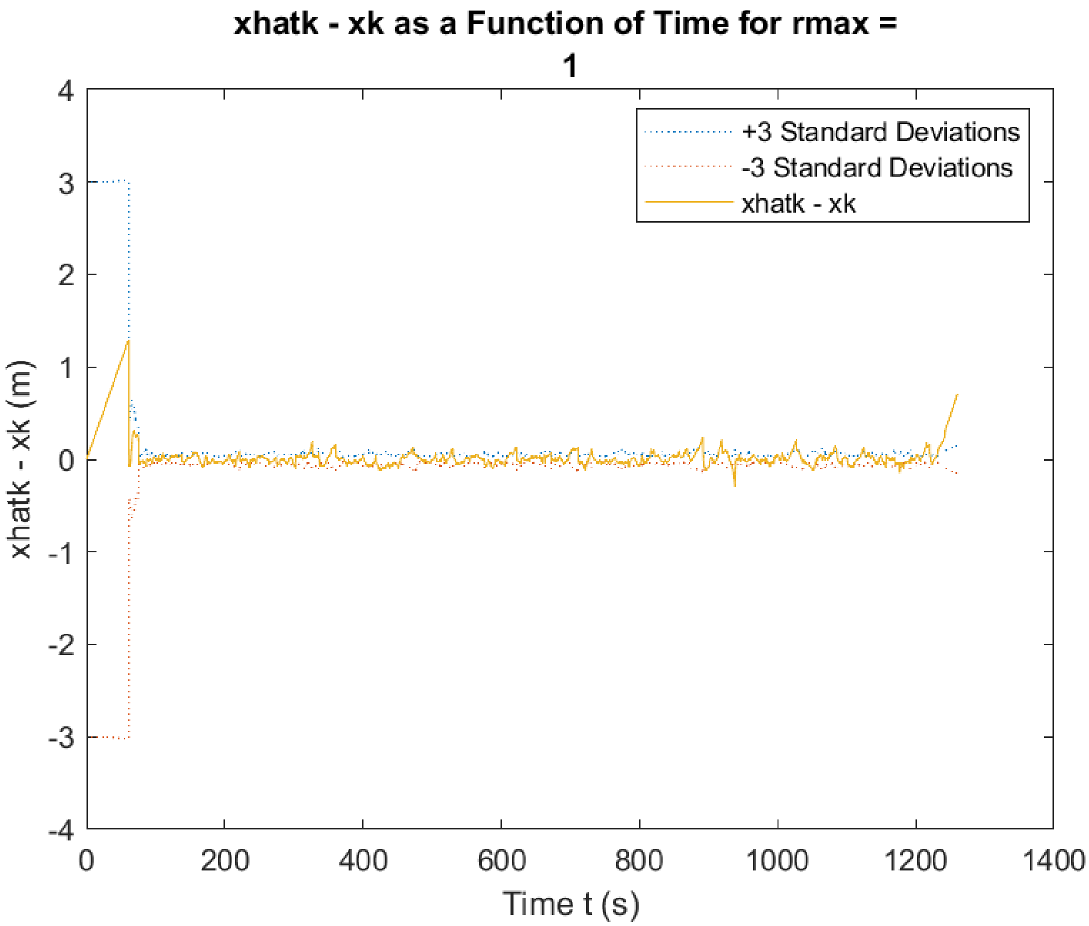

Figure 49: Zoomed Out View

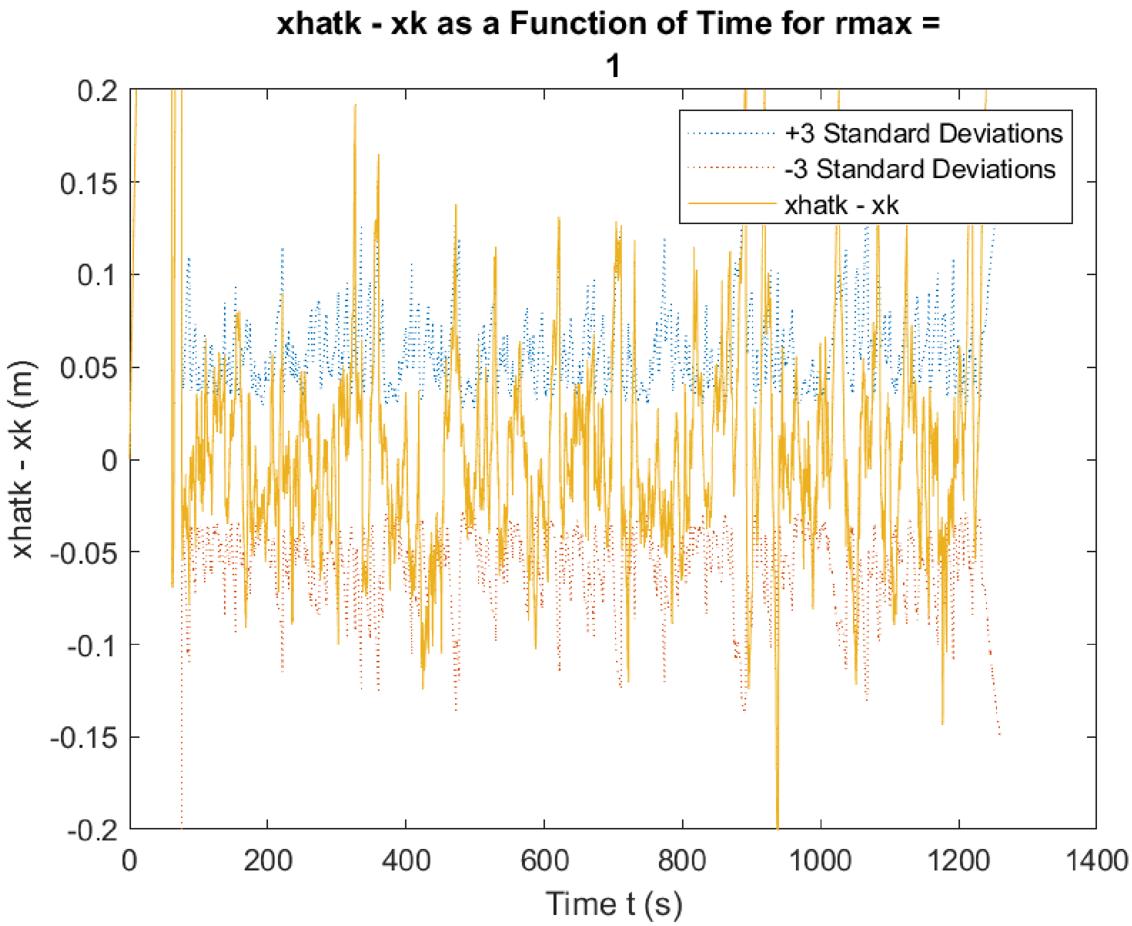

Figure 50: Zoomed In View

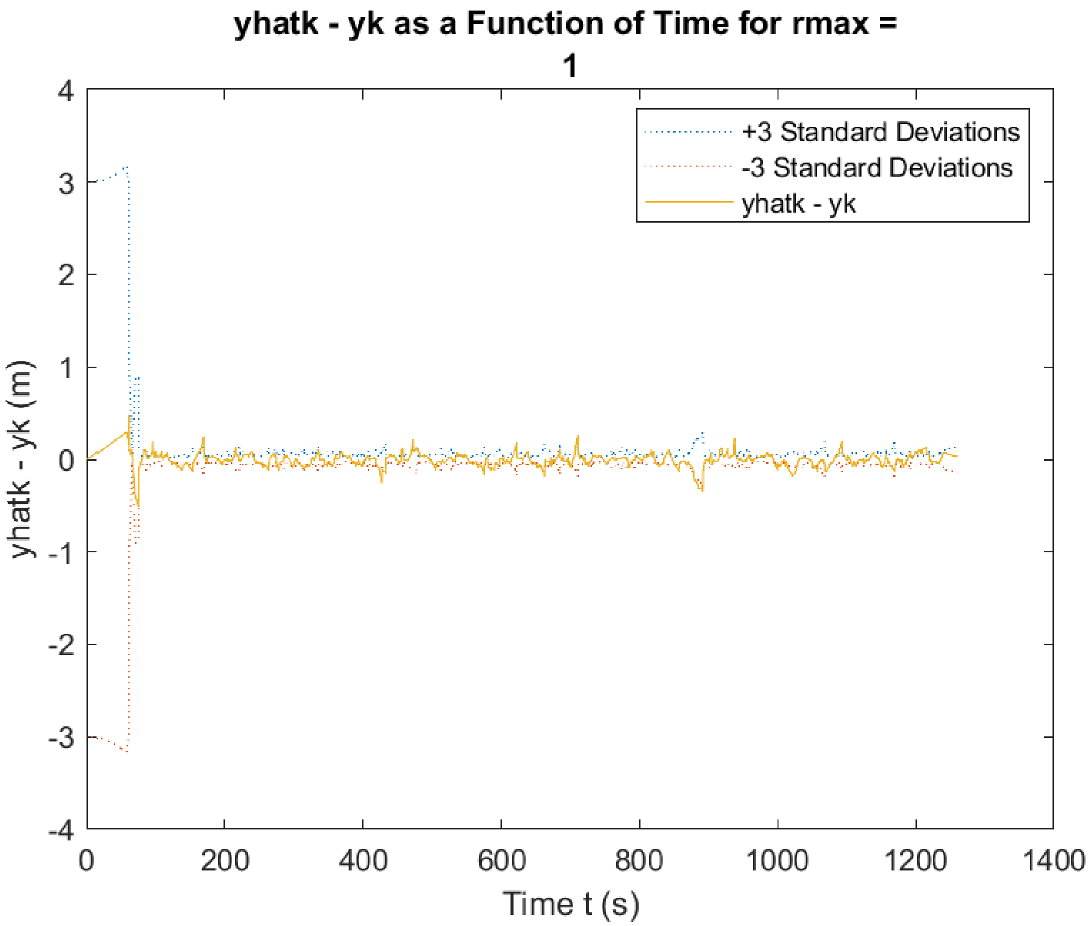

Figure 51: Zoomed Out View

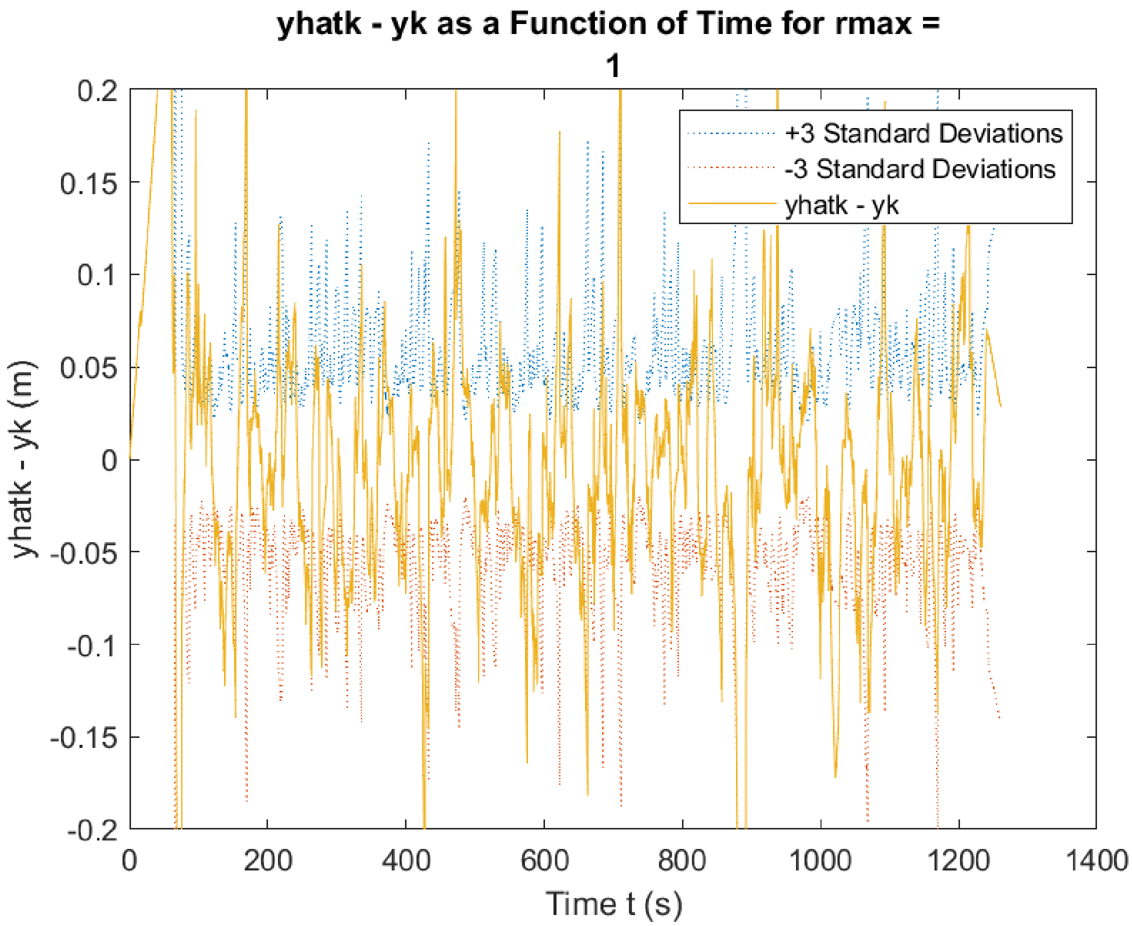

Figure 52: Zoomed In View

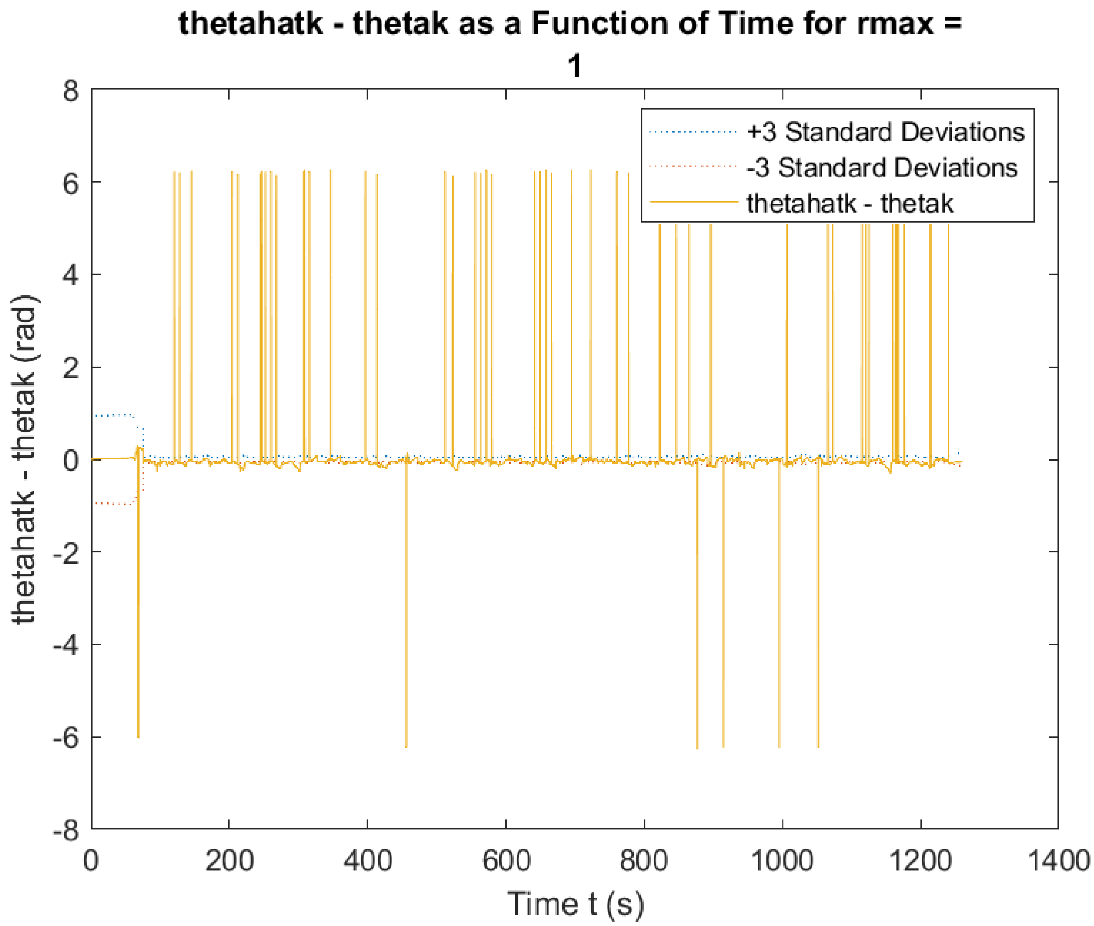

Figure 53: Zoomed Out View

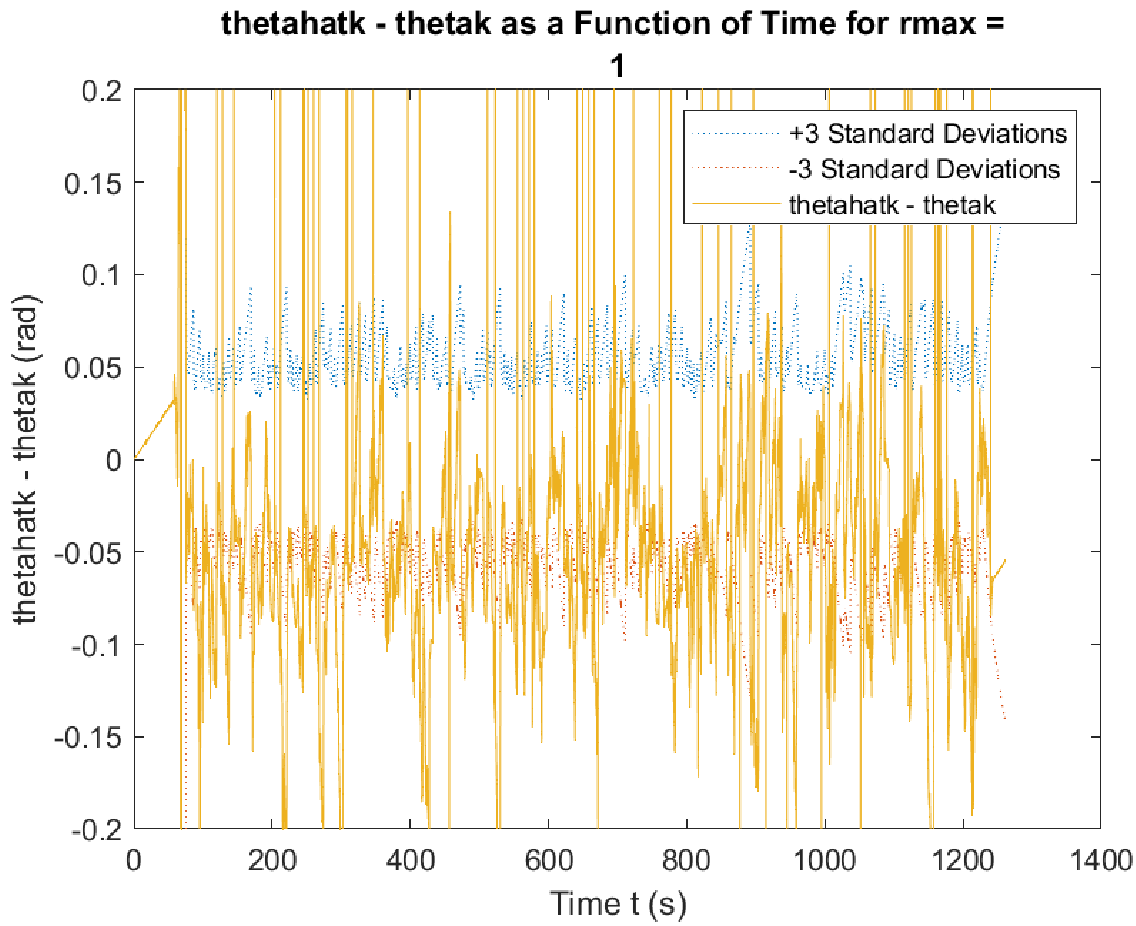

Figure 54: Zoomed In View

The CLRB version of the EKF is very similar to version of the EKF where all Jacobians are evaluated at the estimated state $\hat{\mathbf{x}}_k$. It means that the normal version of the EKF is very close to the CLRB version with the lowest possible uncertainty bound which is good.

Appendix

```
clear
clc
%Load dataset
load('dataset2.mat')

syms x y th T d_ real
syms u w sigq [2 1] real
syms y_ xyl nl sigr [17 2] real

r_maxs = [5,3,1];

T_val = t(2) - t(1);
K = length(t)-1;

x_hat_0 = [-3;-3;-3];%[x_true(1);y_true(1);th_true(1)];
vars_0 = [1,1,0.1];
P_hat_0 = diag(vars_0);

vs = [v.';om.'];

h = [x + T*cos(th)*(u1+w1); y + T*sin(th)*(u1+w1) ; th + T*(u2+w2)];
h_syms = [x,y,th,u1,u2,w1,w2,T];

x_ = [x,y,th];
F = jacobian(h,x_);
F_syms = [th, u1, w1, T];

w_ = [w1,w2];
w_p = jacobian(h,w_);
w_p_syms = [th,T];

Q = diag([sigq1,sigq2]);

for r_max = r_maxs

    x_hat = zeros(3,K+1);
    vars = zeros(3,K+1);
    x_hat(:,1) = x_hat_0;
    vars(:,1) = vars_0;

    x_hat_k = x_hat_0;
    P_hat_k = P_hat_0;
    for k = 1:K
        disp((k/K)*100);
        r_k = r(k,:);
        b_k = b(k,:);
        cond_1 = r_k ~= 0;
        cond_2 = r_k < r_max;
        r_k_valid = cond_1 & cond_2;
        r_k_new = r_k(r_k_valid);
```

```
        b_k_new = b_k(r_k_valid);
        r_k_valid2 = repmat(r_k_valid.',1,2);

        l_vals = l(r_k_valid2);
        y_vals = [r_k_new.',b_k_new.'];
        y_vals_ordered = reshape(y_vals.',[],1);

        y_syms = y_(r_k_valid2);
        l_syms = xyl(r_k_valid2);
        nl_syms = nl(r_k_valid2);
        sigr_syms = sigr(r_k_valid2);
        nl_vals = zeros(length(l_vals),1);

        y_syms_ordered = reshape(reshape(y_syms,[],2).',[],1);
        nl_syms_ordered = reshape(reshape(nl_syms,[],2).',[],1);
        sigr_syms_ordered = reshape(reshape(sigr_syms,[],2).',[],1);

        L = size(l_vals,1)/2;

        g = sym(zeros(2*L,1));
        g_syms = [x_,d_,l_syms.',nl_syms.'];

        for i = 1:L
            gli = [sqrt((l_syms(i) - x - d_*cos(th))^2 + (l_syms(L+i) - y - d_*sin(th))↙
^2) + nl_syms(i) ; atan2((l_syms(L+i) - y - d_*sin(th)), (l_syms(i) - x - d_*cos(th)))↙
- th + nl_syms(L+i)];
            g((2*i-1):2*i) = gli;
        end

        R = diag(sigr_syms_ordered);

        G_ = jacobian(g,x_);
        G__syms = g_syms;

        nl_p = jacobian(g,nl_syms_ordered);

        h_vals = [x_hat_k.',vs(:,k).',0,0,T_val];
        F_vals = [x_hat_k(3),vs(1,k),0,T_val];

        g_vals = [x_hat_k.', d, l_vals.',nl_vals.'];
        G__vals = g_vals;

        Q_p = w_p*Q*w_p.';
        Q_p_syms = [w_p_syms,sigq.'];
        Q_p_vals = [x_hat_k(3),T_val,r_var,b_var];

        sigr_vals = [v_var*ones(L,1),om_var*ones(L,1)];
        sigr_vals_ordered = reshape(sigr_vals.',[],1);

        R_p = nl_p*R*nl_p.';
```

```
        R_p_syms = sigr_syms_ordered;
        R_p_vals = sigr_vals_ordered;


        F_k_1 = double(subs(F,F_syms,F_vals));
        Qp = double(subs(Q_p,Q_p_syms,Q_p_vals));
        h_k = double(subs(h,h_syms,h_vals));
        G_k = double(subs(G_,G__syms,G__vals));
        Rp = double(subs(R_p,R_p_syms,R_p_vals));
        y_k = double(subs(y_syms_ordered,y_syms_ordered.',y_vals_ordered.'));
        g_k = double(subs(g,g_syms,g_vals));

        P_check_k = F_k_1*P_hat_k*F_k_1.' + Qp;
        x_check_k = h_k;
        x_check_k(3) = wrapToPi(x_check_k(3));

        K_k = (P_check_k*G_k.')/(G_k*P_check_k*G_k.' + Rp);
        P_hat_k = (eye(3)-K_k*G_k)*P_check_k;

        innovation = y_k-g_k;
        innovation = wrapinno(innovation);

        x_hat_k = x_check_k + K_k*innovation;
        x_hat_k(3) = wrapToPi(x_hat_k(3));

        x_hat(:,k+1) = x_hat_k;
        vars(:,k+1) = diag(P_hat_k);
    end
    stds = sqrt(vars);

    % Difference between predicted and true state
    x_diff = x_hat(1,:).' - x_true;
    y_diff = x_hat(2,:).' - y_true;
    th_diff = x_hat(3,:).' - th_true;

    % Upper and lower bounds of standard deviation (+/- 3 std deviations)
    x_upper = 3*stds(1,:).';
    x_lower = -3*stds(1,:).';

    y_upper = 3*stds(2,:).';
    y_lower = -3*stds(2,:).';

    th_upper = 3*stds(3,:).';
    th_lower = -3*stds(3,:).';

    % Plot difference between predicted and true state

    figure
    plot(t,x_upper,':',t,x_lower,':',t,x_diff,'-');
    title(["xhatk - xk as a Function of Time for rmax = ", num2str(r_max)]);
```

```
xlabel("Time t (s)");
ylabel("xhatk - xk (m)");
%ylim([-0.2,0.2]);
legend(["+3 Standard Deviations", "-3 Standard Deviations","xhatk - xk"]);
fname = sprintf('Plotxneg3_uz%d.png', r_max);
saveas(gcf,fname)

figure
plot(t,x_upper,':',t,x_lower,':',t,x_diff,'-');
title(["xhatk - xk as a Function of Time for rmax = ", num2str(r_max)]);
xlabel("Time t (s)");
ylabel("xhatk - xk (m)");
ylim([-0.2,0.2]);
legend(["+3 Standard Deviations", "-3 Standard Deviations","xhatk - xk"]);
fname = sprintf('Plotxneg3_z%d.png', r_max);
saveas(gcf,fname)

figure
plot(t,y_upper,':',t,y_lower,':',t,y_diff,'-');
title(["yhatk - yk as a Function of Time for rmax = ", num2str(r_max)]);
xlabel("Time t (s)");
ylabel("yhatk - yk (m)");
%ylim([-0.2,0.2]);
legend(["+3 Standard Deviations", "-3 Standard Deviations","yhatk - yk"]);
fname = sprintf('Plotyneg3_uz%d.png', r_max);
saveas(gcf,fname)

figure
plot(t,y_upper,':',t,y_lower,':',t,y_diff,'-');
title(["yhatk - yk as a Function of Time for rmax = ", num2str(r_max)]);
xlabel("Time t (s)");
ylabel("yhatk - yk (m)");
ylim([-0.2,0.2]);
legend(["+3 Standard Deviations", "-3 Standard Deviations","yhatk - yk"]);
fname = sprintf('Plotyneg3_z%d.png', r_max);
saveas(gcf,fname)

figure
plot(t,th_upper,':',t,th_lower,':',t,th_diff,'-');
title(["thetahatk - thetak as a Function of Time for rmax = ", num2str(r_max)]);
xlabel("Time t (s)");
ylabel("thetahatk - thetak (rad)");
%ylim([-0.2,0.2]);
legend(["+3 Standard Deviations", "-3 Standard Deviations","thetahatk - thetak"]);
fname = sprintf('Plotthetaneg3_uz%d.png', r_max);
saveas(gcf,fname)

figure
plot(t,th_upper,':',t,th_lower,':',t,th_diff,'-');
title(["thetahatk - thetak as a Function of Time for rmax = ", num2str(r_max)]);
```

```
    xlabel("Time t (s)");
    ylabel("thetahatk - thetak (rad)");
    ylim([-0.2,0.2]);
    legend(["+3 Standard Deviations", "-3 Standard Deviations","thetahatk - thetak"]);
    fname = sprintf('Plotthetaneg3_z%d.png', r_max);
    saveas(gcf,fname)

end


fig = figure;
num_frames = K+1;

vid = VideoWriter('animation.avi');
vid.FrameRate = 1/T_val;
open(vid);

for frame = 1:num_frames

    a=3*stds(1,frame); % horizontal radius
    b=3*stds(2,frame); % vertical radius
    x0=x_hat(1,frame); % x0,y0 ellipse centre coordinates
    y0=x_hat(2,frame);
    x0true=x_true(frame);
    y0true=y_true(frame);

    tparam=-pi:0.01:pi;
    x_ellipse=x0+a*cos(tparam);
    y_ellipse=y0+b*sin(tparam);

    scatter(l(:,1).',l(:,2).',100,'k','filled','o');
    hold on
    plot(x_ellipse, y_ellipse, 'r', 'LineWidth', 2);
    plot(x0,y0,'Color', 'r', 'Marker', 'o', 'MarkerFaceColor', 'r', 'LineWidth',1);
    plot(x0true,y0true,'Color', 'b', 'Marker', 'o', 'MarkerFaceColor', 'b',↙
'LineWidth',1);
    hold off

    frame_data = getframe(fig);
    writeVideo(vid, frame_data);
end

close(vid);
```

```
function inno = wrapinno(innovation)
    inno = innovation;
    n = length(inno);
    for i = 1:n
        if mod(i,2) == 0
            inno(i) = wrapToPi(inno(i));
        end
    end
end
```