AER 1513: Assignment 3

Charles Horn - 1006958721

October 2020

1 Questions

1. For the IMU measurements, the histogram plots are quite close to a Gaussian distribution, so this is a fair assumption.

   The histogram plots for the stereo measurements are not as close to a Gaussian distribution. In particular, both the left and right cameras seem to have an error bias in the vertical measurements. This bias tends towards negative values, so we will want to keep this in mind when using pixel measurements to determine location of ground points in relation to the camera reference frame.

   Some reasonable values for Q_k are below.

$$T_k = t(k) - t(k-1) \tag{1}$$

$$\begin{aligned} Q_k &= T_k^2 * \begin{bmatrix} \delta v_x^2 &&&&& \\ & \delta v_y^2 &&&& \\ && \delta v_z^2 &&& \\ &&& \delta\psi_x^2 && \\ &&&& \delta\psi_y^2 & \\ &&&&& \delta\psi_z^2 \end{bmatrix} \\ &= T_k^2 * \begin{bmatrix} 0.0513^2 &&&&& \\ & 0.0456^2 &&&& \\ && 0.0281^2 &&& \\ &&& 0.095^2 && \\ &&&& 0.13^2 & \\ &&&&& 0.42^2 \end{bmatrix} \\ &= T_k^2 * \begin{bmatrix} 0.0026 &&&&& \\ & 0.0021 &&&& \\ && 0.00079 &&& \\ &&& 0.009 && \\ &&&& 0.017 & \\ &&&&& 0.175 \end{bmatrix} \end{aligned}$$

   R_k will be a combination of R_k^j matrices. We will create a block diagonal matrix where each of the diagonal entries will be a diagonal block R_k^j seen below, and there will be a block per visible point at time k.

$$\begin{aligned} R_k^j &= \begin{bmatrix} n_{u_l}^2 &&& \\ & n_{v_l}^2 && \\ && n_{u_r}^2 & \\ &&& n_{v_r}^2 \end{bmatrix} \\ &= \begin{bmatrix} 6.164^2 &&& \\ & 11.395^2 && \\ && 6.478^2 & \\ &&& 11.511^2 \end{bmatrix} \\ &= \begin{bmatrix} 37.98 &&& \\ & 129.84 && \\ && 41.95 & \\ &&& 132.49 \end{bmatrix} \end{aligned}$$

2. We construct an error term for both the motion and observation models. The objective function for the motion model is:

$$\begin{aligned} e_{v,k}(x) &= ln(\check{T}_0 T_0^{-1})^{\vee} \ , \ k = 0 \\ e_{v,k}(x) &= ln(\Xi T_{k-1} T_k^{-1})^{\vee} \ , \ k = 1, ..., K \\ J_{v,k}(x) &= \frac{1}{2} e_{v,0}(x)^T \check{P}_0^{-1} e_{v,0}(x), k = 0 \\ J_{v,k}(x) &= \frac{1}{2} e_{v,k}(x)^T Q_k^{-1} e_{v,k}(x), k = 1, ..., K \\ J_v(\bar{x}_{k_1:k_2}) &= \sum_{k1}^{k2} J_{v,k}(x) \end{aligned}$$

   Where Q_k is defined in question 1. The observation model objective function measured in stereo camera model vectors:

$$e_{y,jk}(x) = y_{jk} - g(T_{cv_k} T_{v_k i} p_j) \tag{2}$$

$$J_{y,jk}(x) = \frac{1}{2} e_{y,jk}(x)^T R_{kj}^{-1} e_{y,jk}(x) \tag{3}$$

$$J_y(\bar{x}_{k_1:k_2}) = \frac{1}{2} \sum_{visible\ points\ j} \sum_{k1}^{k2} J_{y,jk}(x) \tag{4}$$

Where R_k^j and T_k are defined in question 1, and we only sum $J_{y,jk}$ for visible points j at time k. Combined, the loss function becomes:

$$J(\bar{x}_{k_1:k_2}) = J_v(\bar{x}_{k_1:k_2}) + J_y(\bar{x}_{k_1:k_2}) \tag{5}$$

3. To perform Gauss-Newton, we first linearize the error terms. First, for the motion model error:

$$\begin{aligned}
e_{v,k}(x) &= ln(\check{T}_0 T_{op,0}^{-1} exp(-\epsilon_0))^{\vee} \\
&\approx e_{v,k}(x_{op}) - \epsilon_0\ ,\ k = 0 \\
e_{v,k}(x) &= ln(\Xi T_{k-1} * T_k^{-1})^{\vee} \\
&\approx ln(\Xi_k T_{op,k-1} * T_{op,k}^{-1})^{\vee} + Ad(T_{op,k} T_{op,k-1}^{-1})\epsilon_{k-1} - \epsilon_k\ ,\ k = 1, ..., K \\
&= e_{v,k}(x_{op}) + F_{k-1}\epsilon_{k-1} - \epsilon_k
\end{aligned}$$

Then for the observation model:

$$\begin{aligned}
e_{y,jk}(x) &= y_{jk} - g(T_{cv_k} * T_{v_k i} * p_j) \\
&\approx y_{jk} - (1 + \epsilon_k) g(T_{cv_k} * T_{op,v_k i} * p_j) \\
&= y_{jk} - D^T J_g T_{cv} (T_{op,k} p_j)^{\odot} \epsilon_k \\
&= e_{y,jk}(x_{op}) + G_{jk}\epsilon_k
\end{aligned}$$

Where J_g is the Jacobian for the stereo camera model.

$$J_g = M * \frac{1}{z} \begin{bmatrix} 1 & 0 & -x/y & 0 \\ 0 & 1 & -y/z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/z & 1 \end{bmatrix} \tag{6}$$

Where:

$$M = \begin{bmatrix} f_u & 0 & c_u & f_u b/2 \\ 0 & f_v & c_v & 0 \\ f_u & 0 & c_u & -f_v b/2 \\ 0 & f_v & c_v & 0 \end{bmatrix} \tag{7}$$

Again, we stack these values for each *visible* point j:

$$e_{y,k}(x) = \begin{bmatrix} e_{y,1k}(x) \\ \vdots \\ e_{y,20k}(x) \end{bmatrix},\ e_{y,k}(x_{op}) = \begin{bmatrix} e_{y,1k}(x_{op}) \\ \vdots \\ e_{y,20k}(x_{op}) \end{bmatrix},\ G_k = \begin{bmatrix} G_{1k} \\ \vdots \\ G_{20k} \end{bmatrix} \tag{8}$$

Then we must define:

$$\delta x = \begin{bmatrix} \epsilon_{k1} \\ \vdots \\ \epsilon_{k2} \end{bmatrix},\ H = \left[\begin{array}{ccccc} 1 & & & & \\ -F_{k_1} & 1 & & & \\ & -F_{k_1+1} & \ddots & & \\ & & \ddots & 1 & \\ & & & -F_{k_2-1} & 1 \\ \hline G_{k_1} & & & & \\ & G_{k_1+1} & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & G_{k_2} \end{array}\right],$$

$$e(x_{op}) = \left[\begin{array}{c} e_{v,k_1}(x_{op}) \\ \vdots \\ e_{v,k_2}(x_{op}) \\ \hline e_{y,k_1}(x_{op}) \\ \vdots \\ e_{y,k_2}(x_{op}) \end{array}\right],\ W = \begin{bmatrix} \check{P}_0, & & & & & \\ & Q_{k_1} & & & & \\ & & \ddots & & & \\ & & & Q_{k_2} & & \\ & & & & R_0 & \\ & & & & & \ddots & \\ & & & & & & R_{k_2} \end{bmatrix}$$

Redefining our objective function with linearized error terms, we get:

$$J(\bar{x}_{k_1:k_2}) \approx J(\bar{x}_{k_1:k_2,op}) - b^T \delta\bar{x}_{k_1:k_2} + \frac{1}{2}\delta\bar{x}_{k_1:k_2}^T A \delta\bar{x}_{k_1:k_2} \tag{9}$$

$$A = H^T W^{-1} H,\ b = H^T W^{-1} e(x_{op}) \tag{10}$$

Deriving wrt. $\delta\bar{x}_{k_1:k_2}$, and setting that to zero, we get the system of equations:

$$A\delta\bar{x}_{k_1:k_2} = b \tag{11}$$

The solutions to which can be defined as:

$$\delta \bar{x}_{k_1:k_2} = \begin{bmatrix} \epsilon_{k1} \\ \vdots \\ \epsilon_{k2} \end{bmatrix} \tag{12}$$

We then update our current operating point $T_{op,k}$:

$$T_{op,k} \longleftarrow exp(\epsilon_k\hat{}) T_{op,k} \tag{13}$$

Finally, we iterate this process until convergence.

4. Below is the plot of visible points vs time, for the time frame 1215-1714 which will be analyzed in the following section.

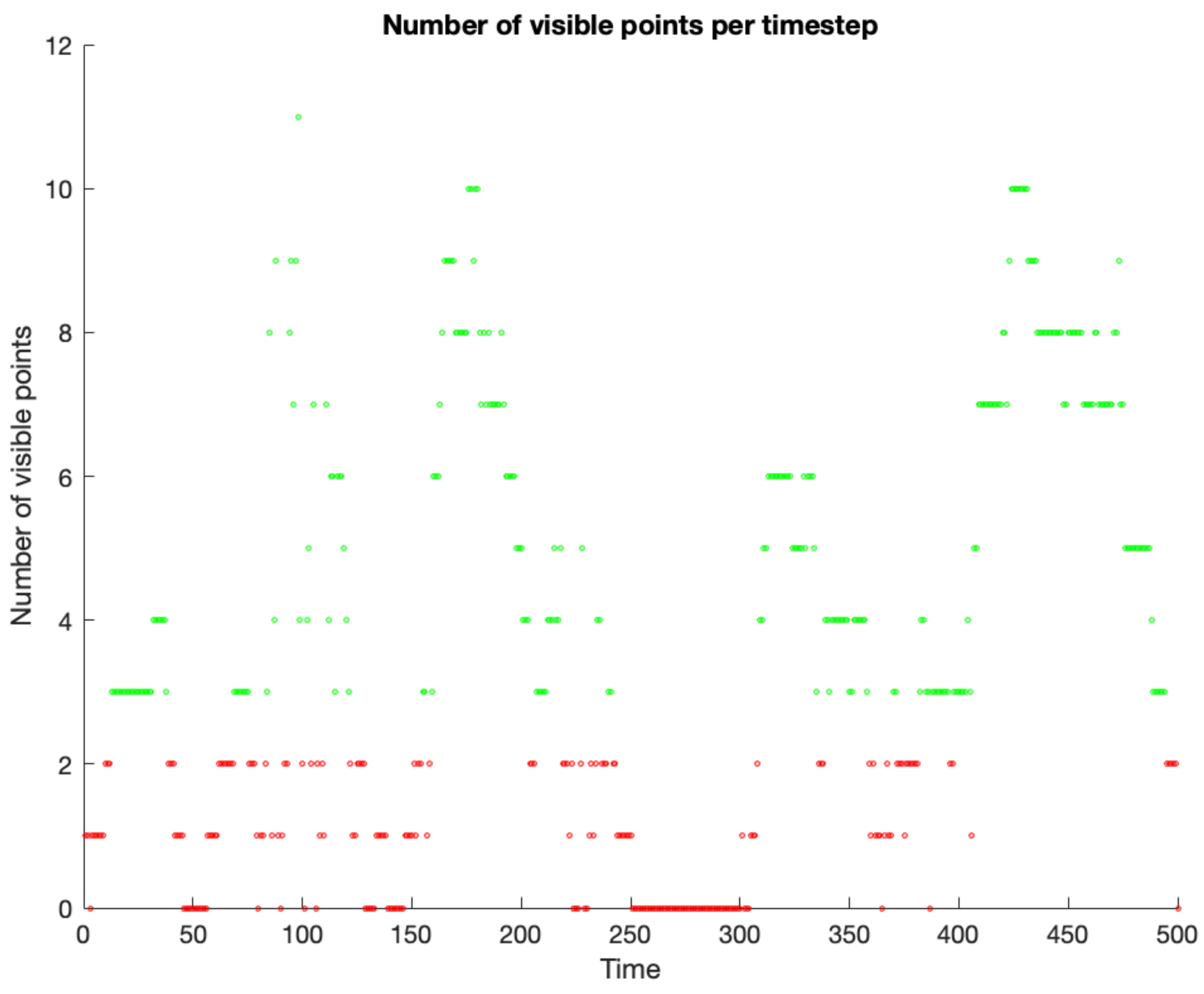

Figure 1: Visible points vs time

5. (a) Below are the plots for the batch GN update process. This was run from timesteps 1215-1714, with 5 GN iterations.

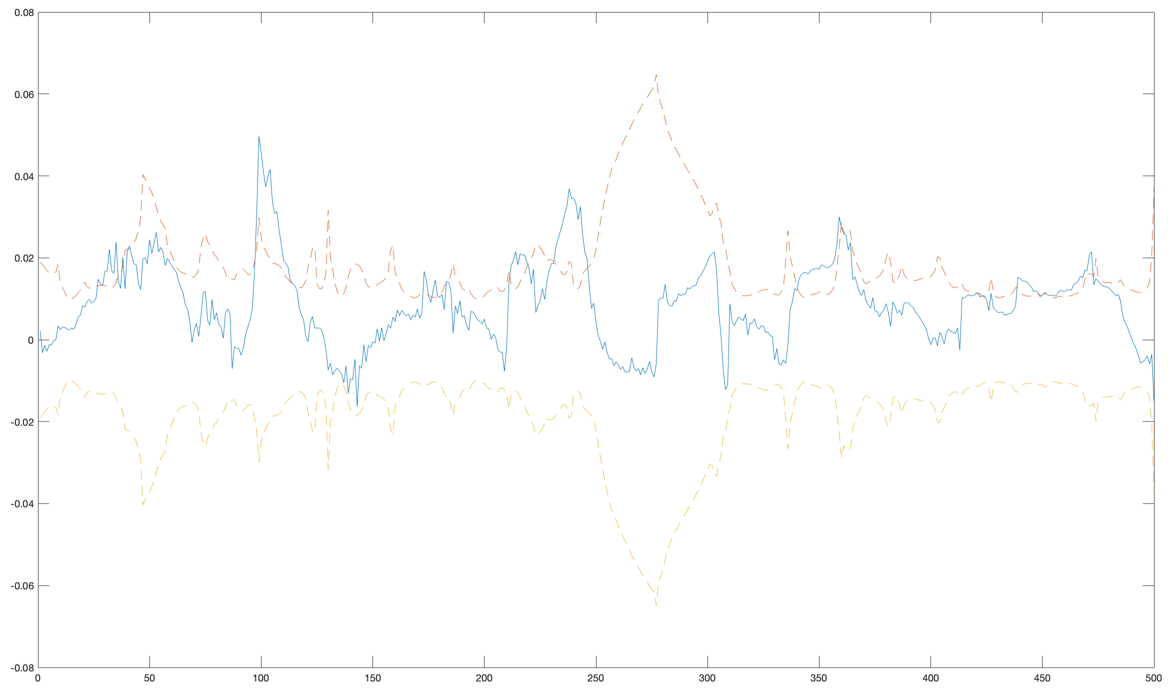

Figure 2: X Error vs Time

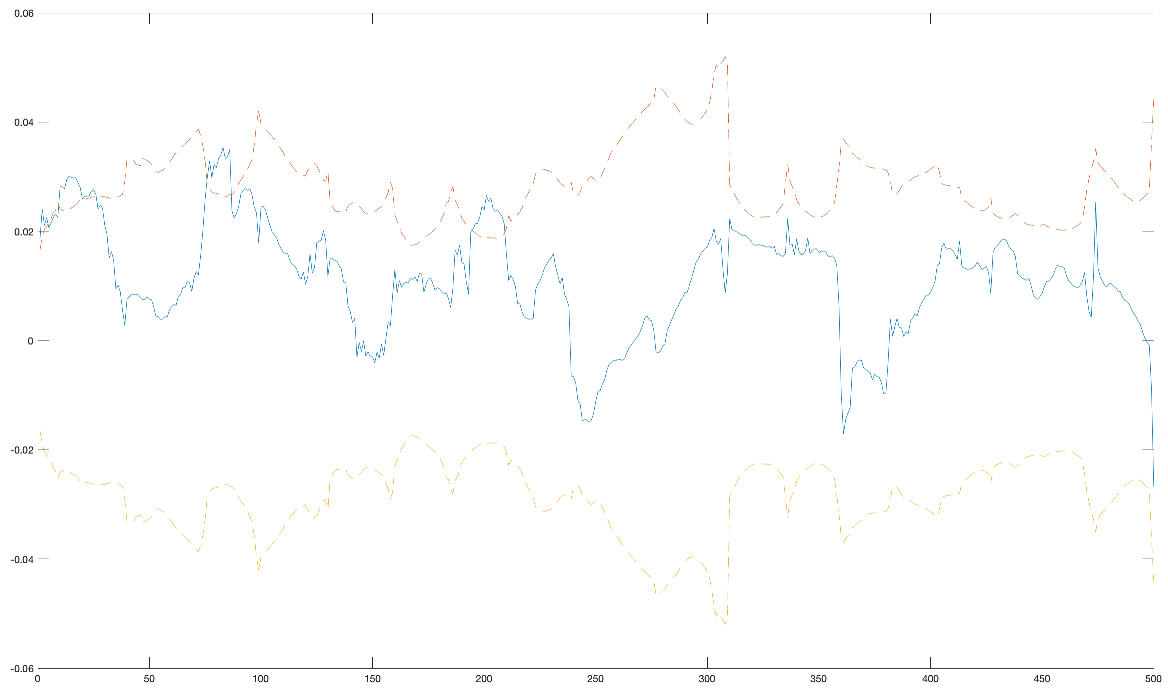

Figure 3: Y Error vs Time

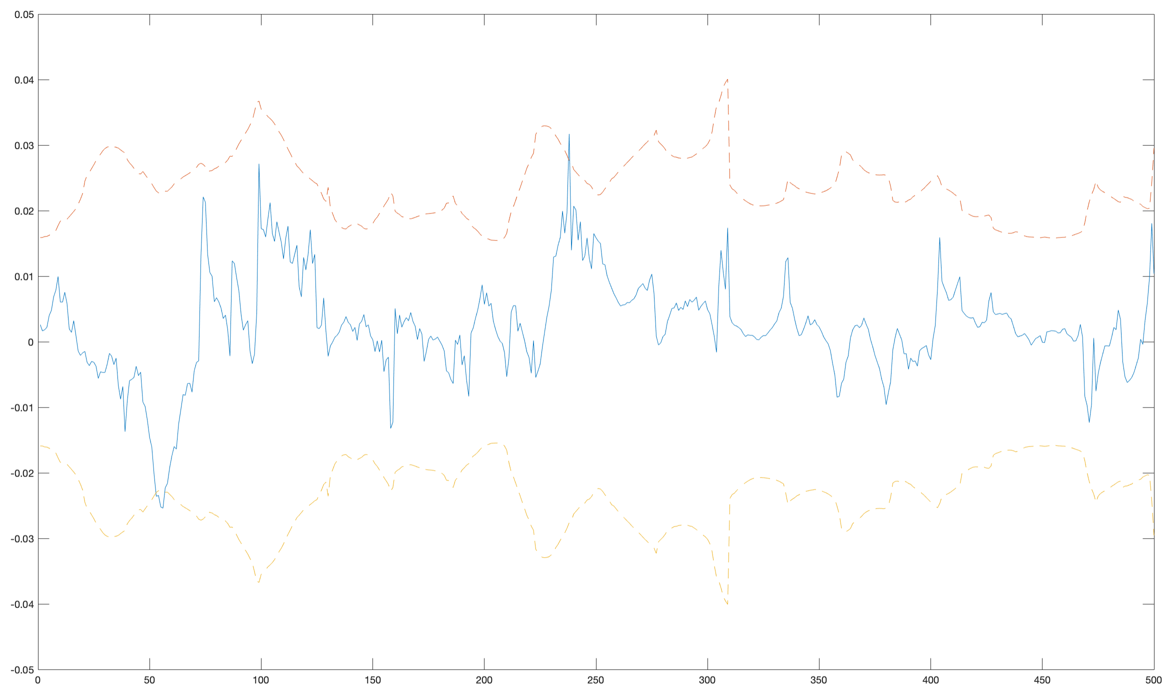

Figure 4: Z Error vs Time

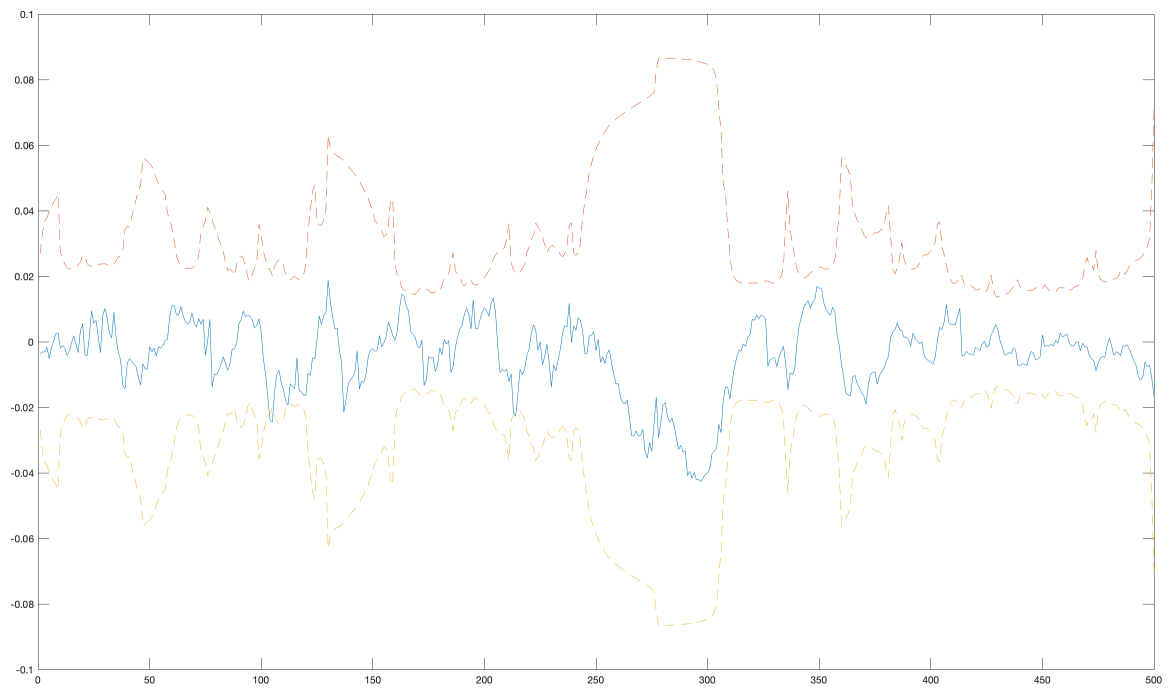

Figure 5: Theta 1 Error vs Time

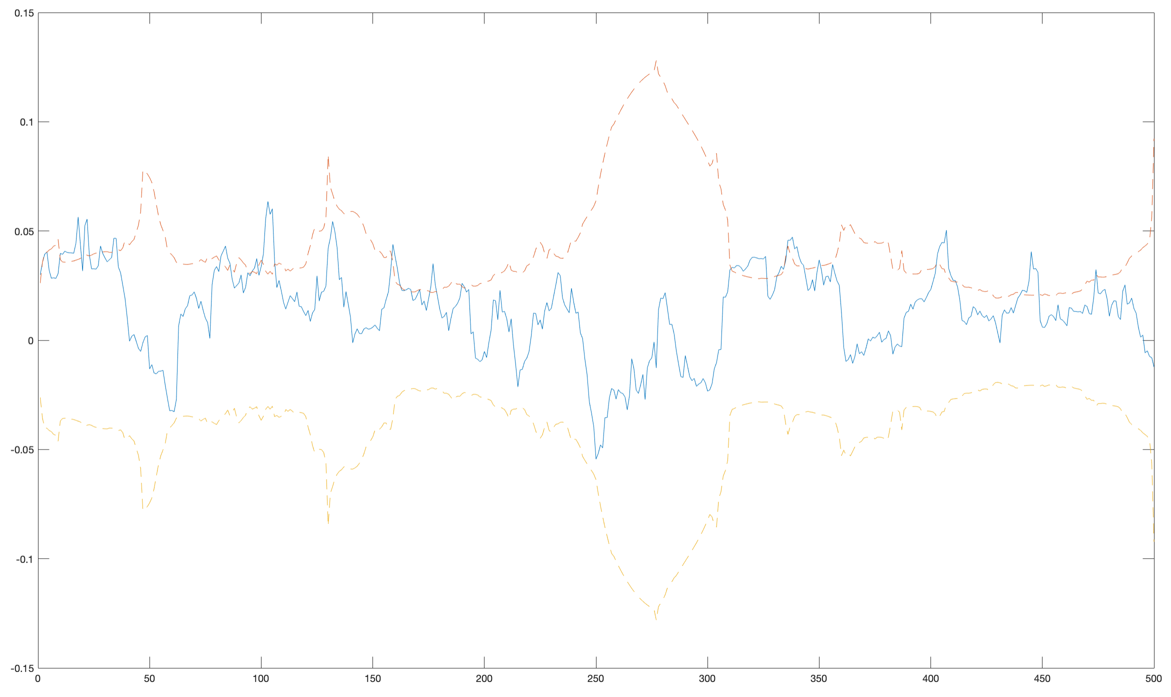

Figure 6: Theta 2 Error vs Time

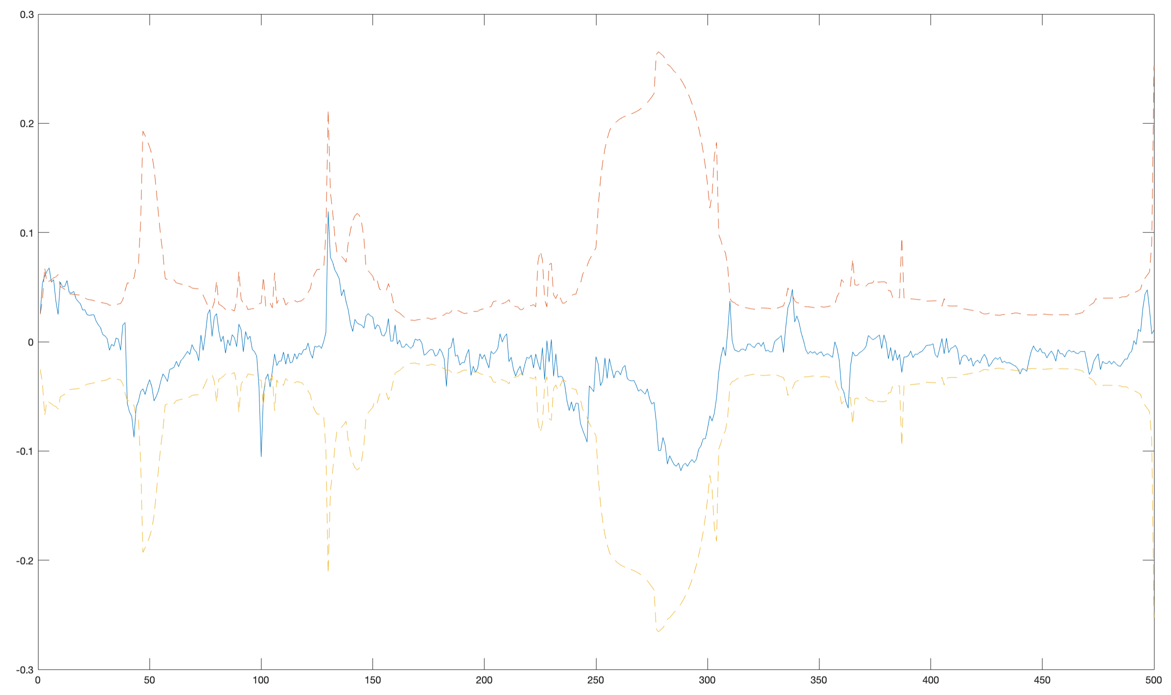

Figure 7: Theta 3 Error vs Time

Figure 8 shows the original path estimate using dead reckoning. Figure 9 shows the corrected estimate using the Batch GN method.

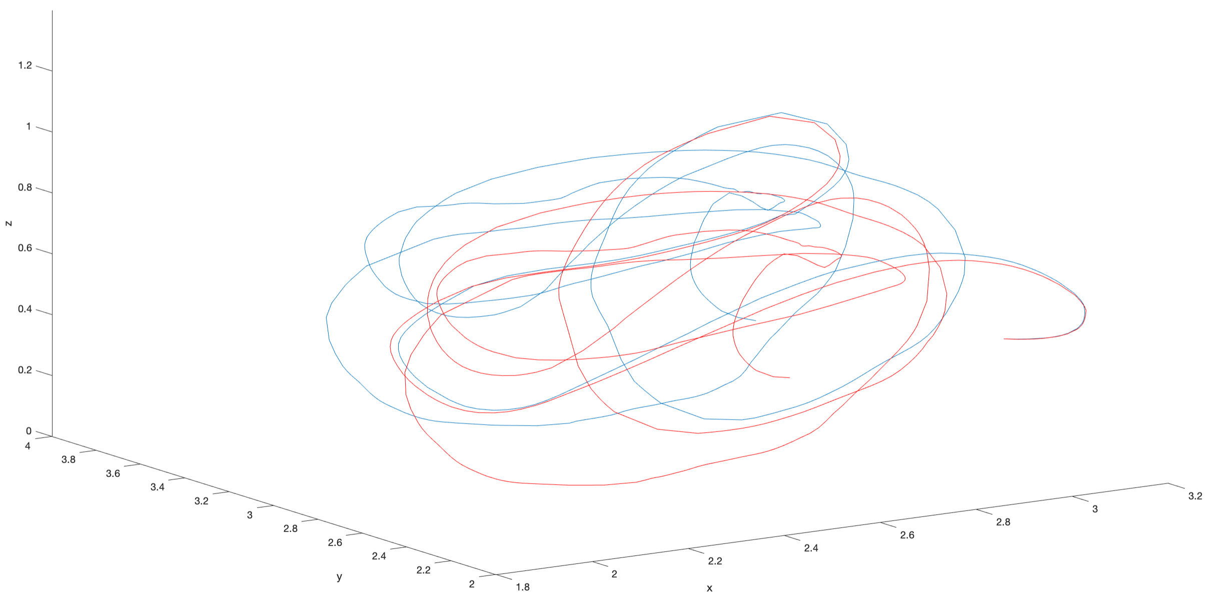

Figure 8: Original estimate using dead reckoning.

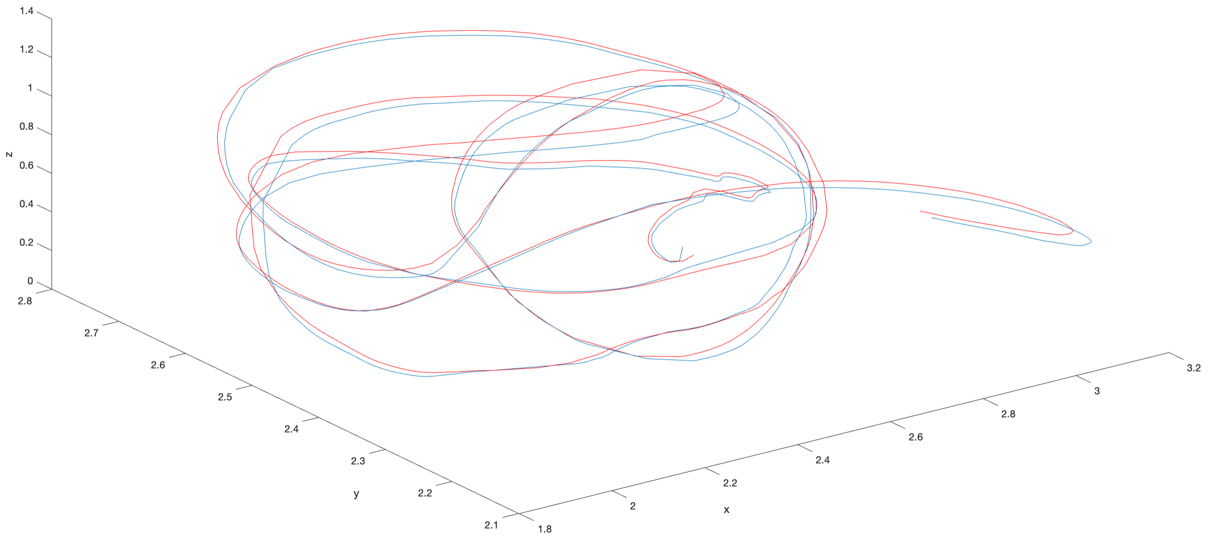

Figure 9: Corrected estimate using Batch GN method

(b) The sliding window plots can be seen below with $\kappa = 50$, running 5 GN iterations per window.

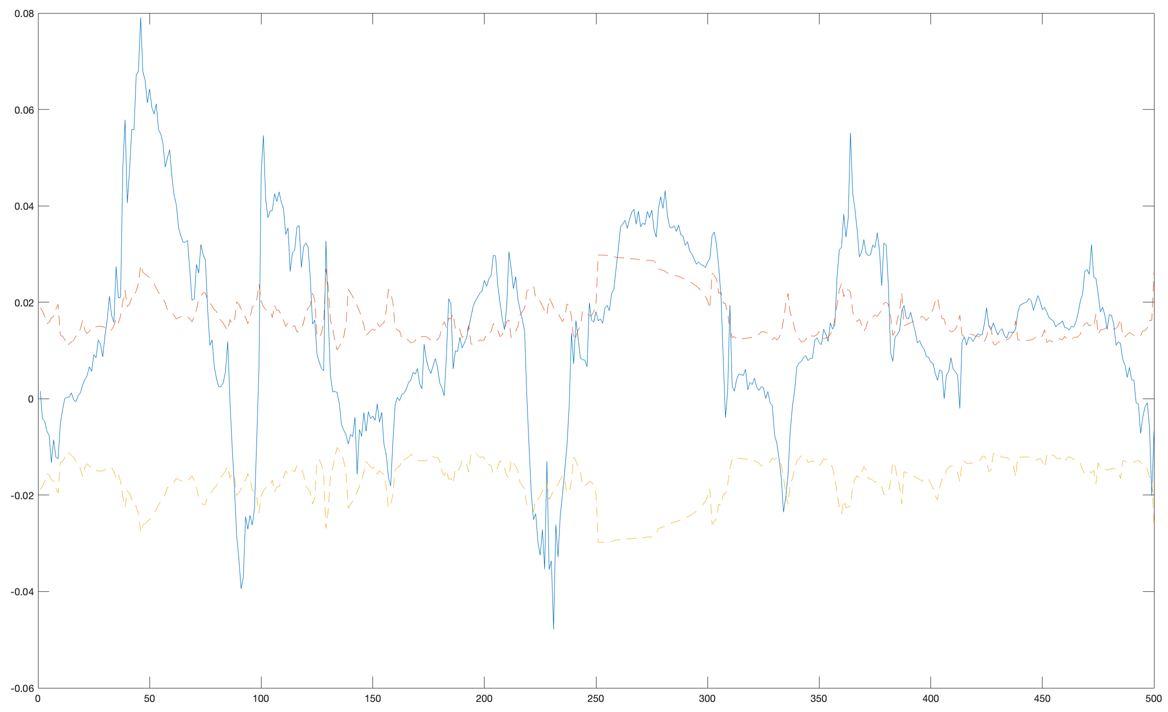

Figure 10: X Error vs Time

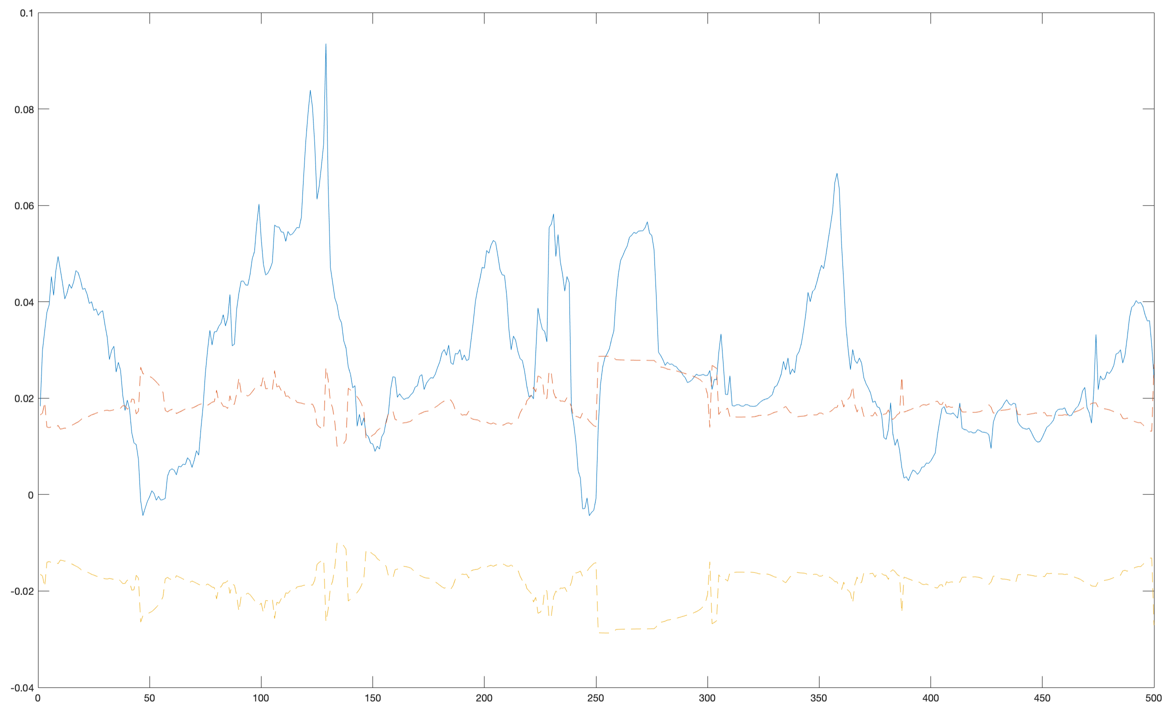

Figure 11: Y Error vs Time

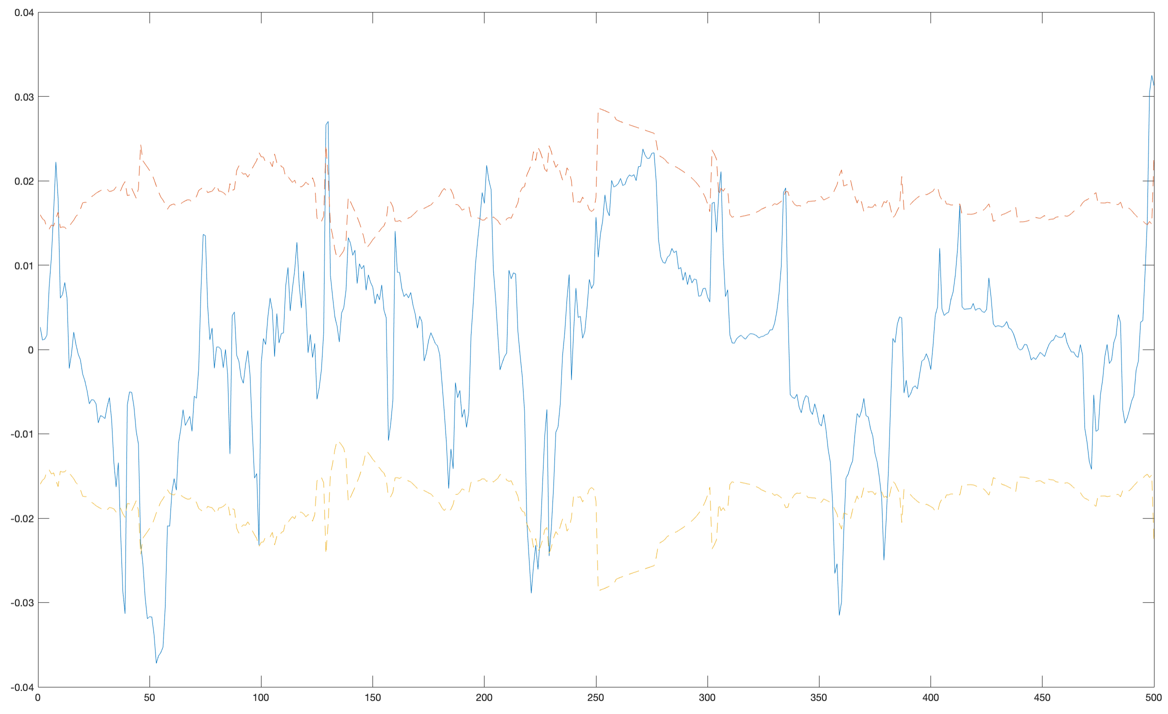

Figure 12: Z Error vs Time

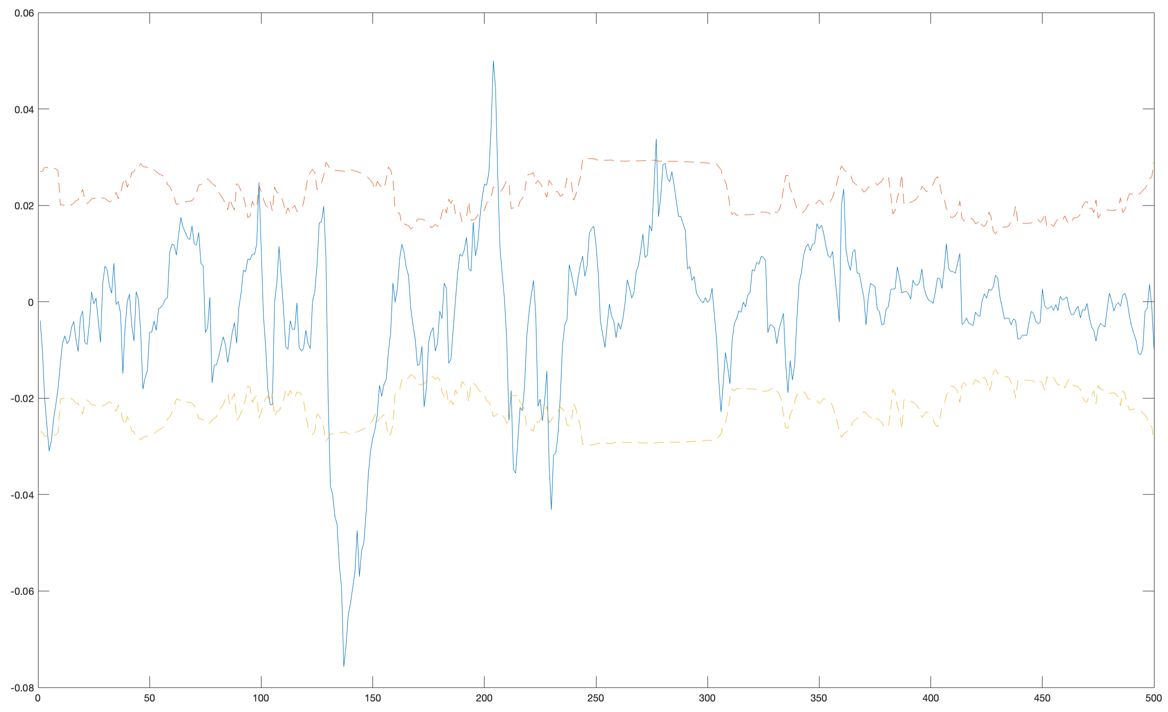

Figure 13: Theta 1 Error vs Time

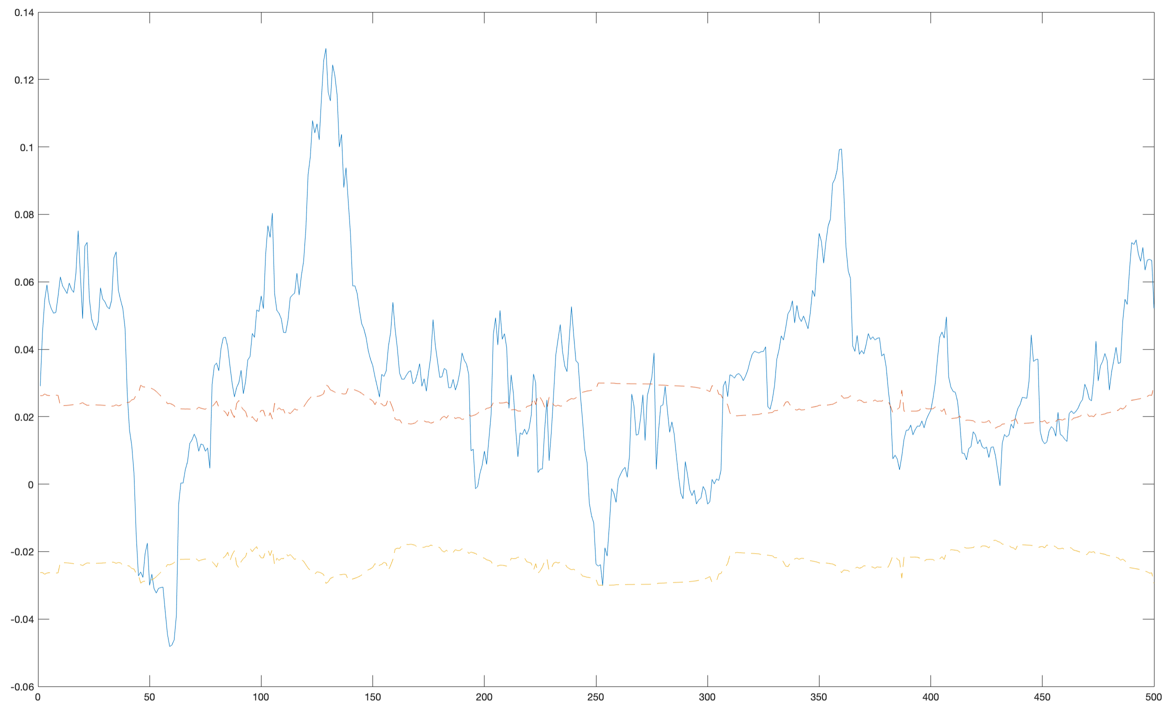

Figure 14: Theta 2 Error vs Time

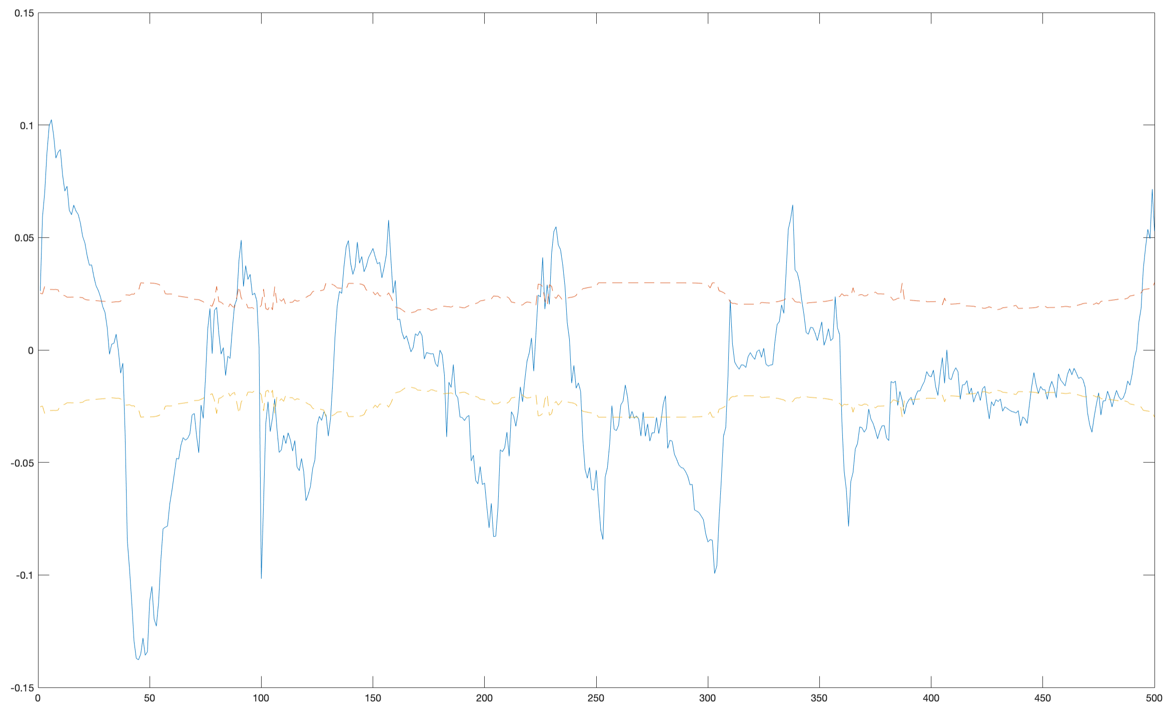

Figure 15: Theta 3 Error vs Time

(c) The sliding window plots can be seen below with $\kappa = 10$, running 5 GN iterations per window.

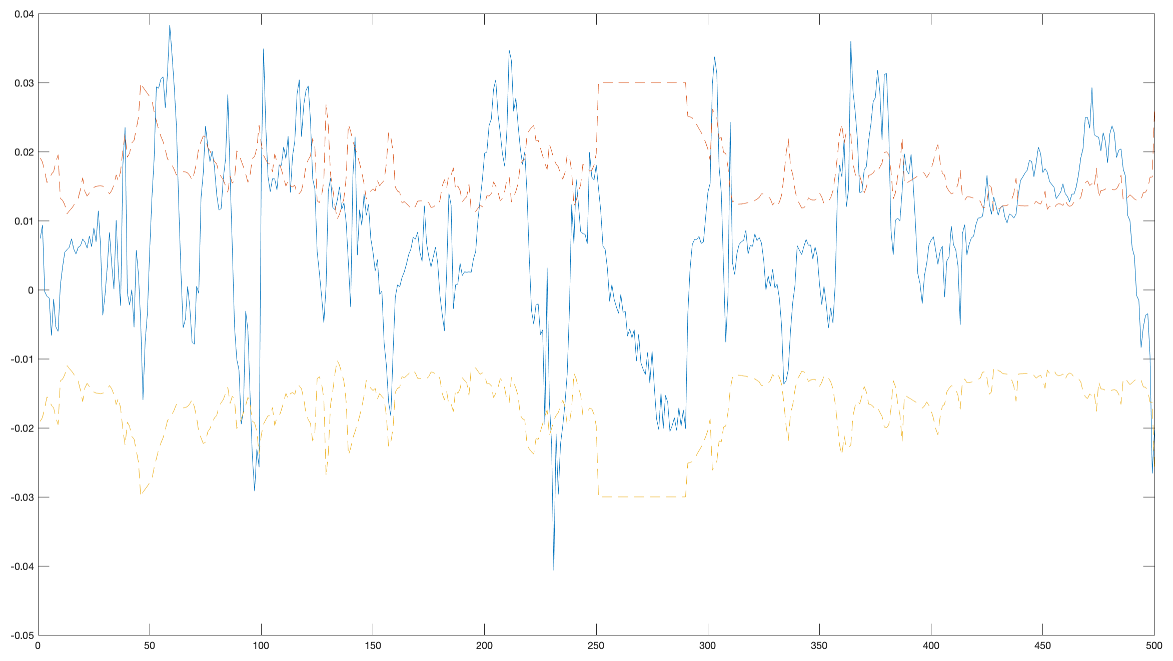

Figure 16: X Error vs Time

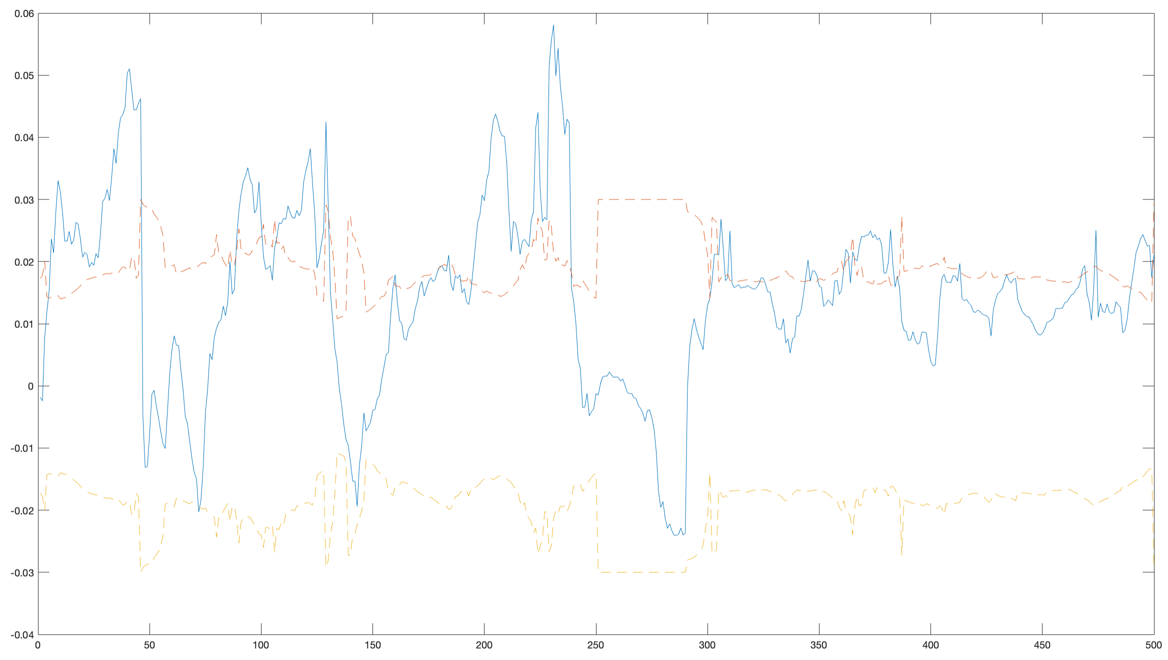

Figure 17: Y Error vs Time

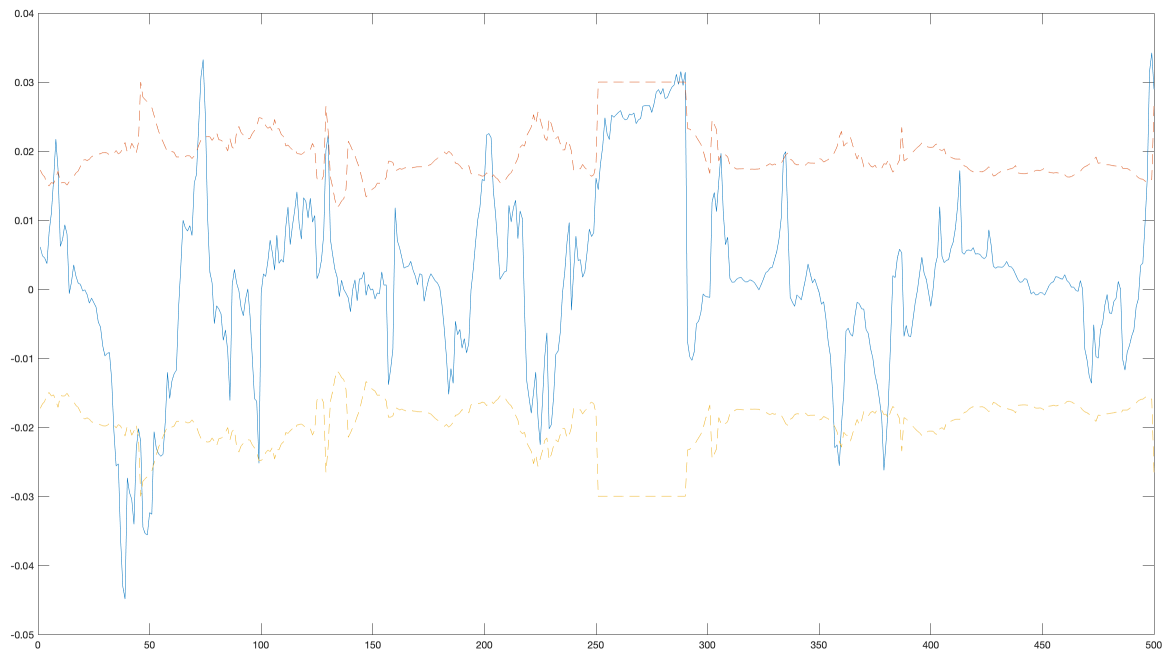

Figure 18: Z Error vs Time

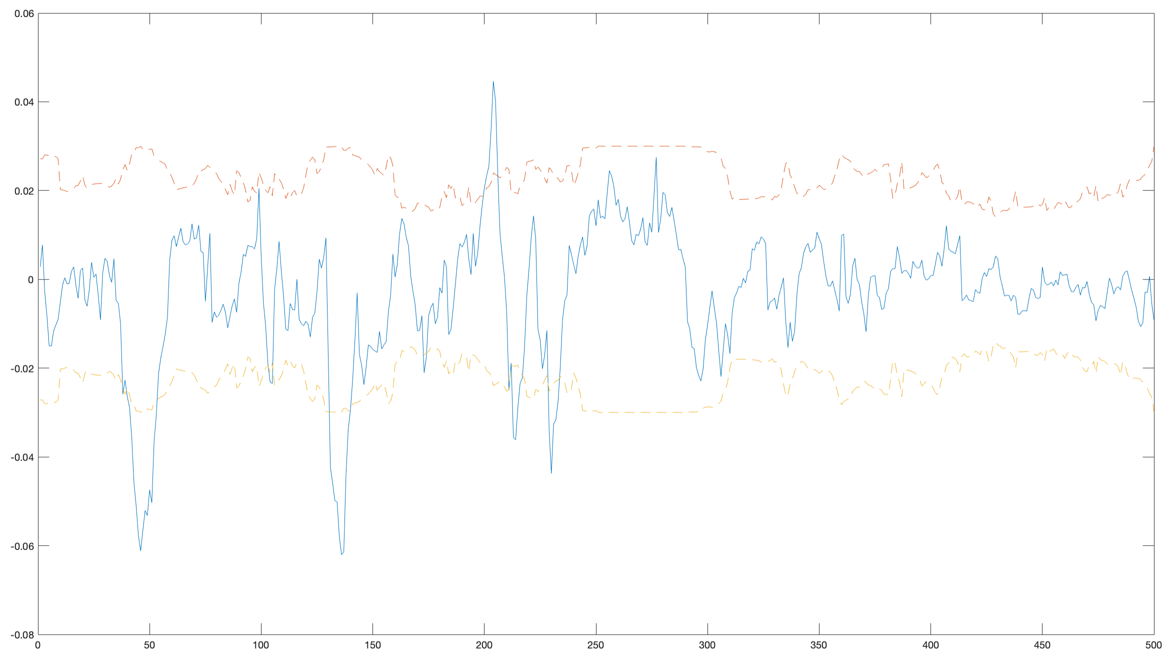

Figure 19: Theta 1 Error vs Time

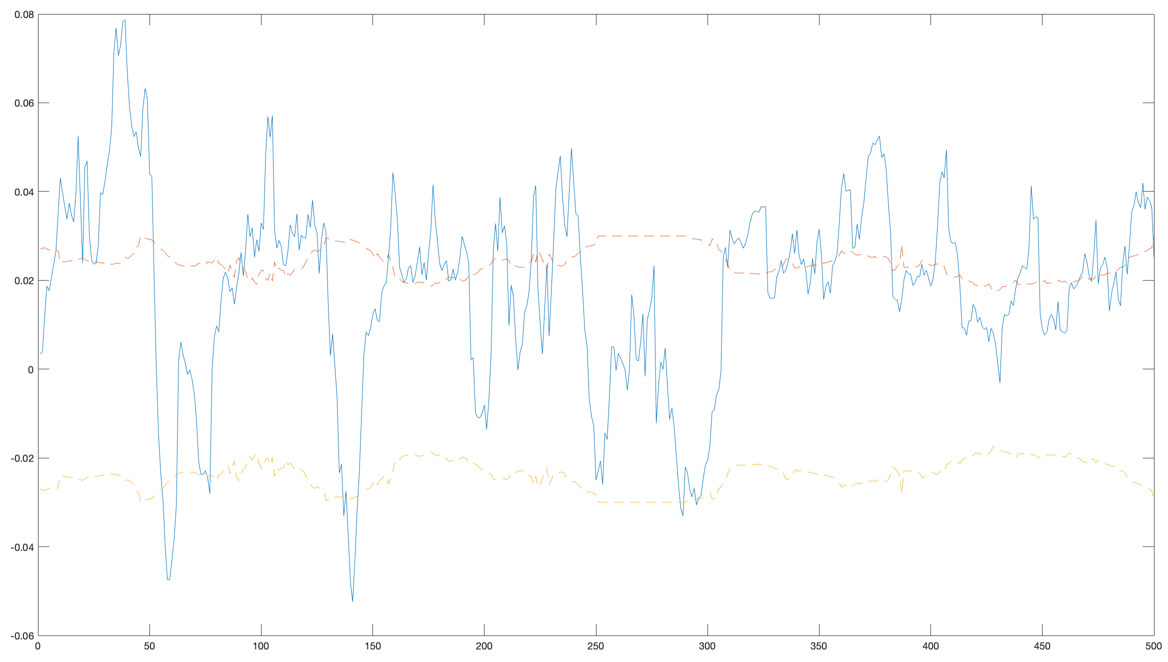

Figure 20: Theta 2 Error vs Time

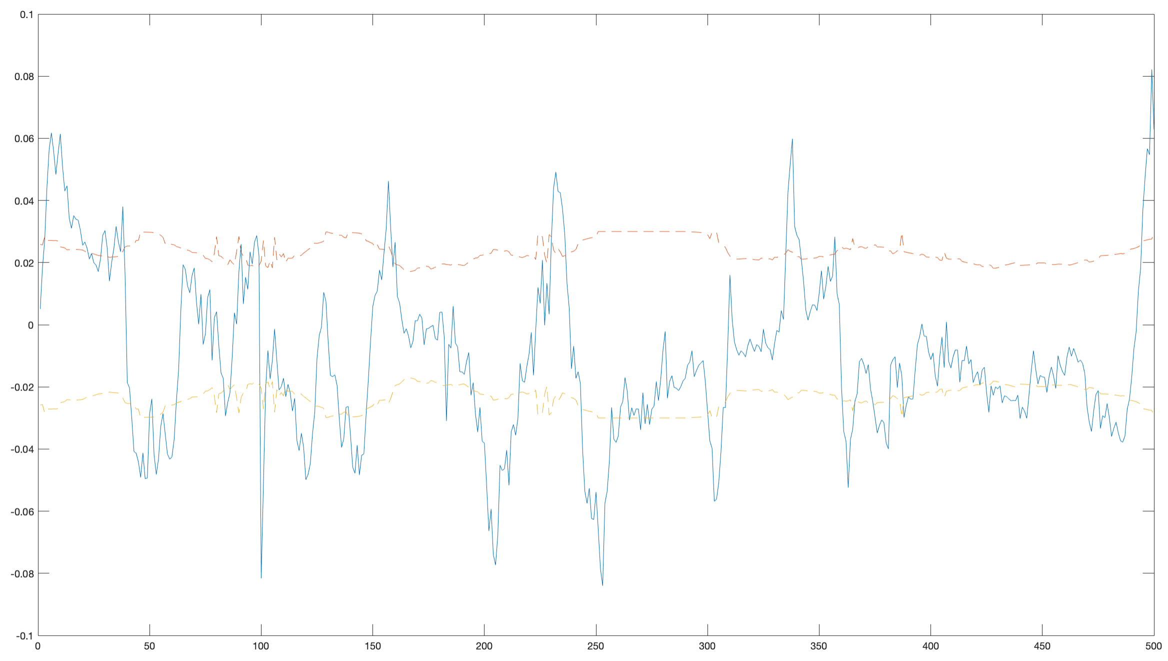

Figure 21: Theta 3 Error vs Time

(d) Compared to the batch method, the sliding window is more computationally expensive and less accurate. The sliding window has the benefit of smaller stacked matrices, which makes for more efficient inversions when solving for the optimal perturbations. However, the cost of repeating the GN iterations for each time step negated that performance gain in my implementation of the algorithm.

The decreased accuracy is not completely surprising with the sliding window method. Since we are only storing the pose at time k, we are not fully taking advantage of what the algorithm was optimizing. The algorithm attempted to optimize the state which included poses from time $k = [k, k + \kappa]$, and we only held onto the first one.

As we can see from Figure 1, the spikes in the uncertainty envelope tend to line up with areas of low visibility. Most notable, from $k = [250, 300]$ where there are no visible points. This intuitively makes sense, since visible points provide more certainty about our state estimate. As a result, our error plots often increase in areas of low visibility as well. Sometimes the estimates are still accurate, but error tends to increase in areas of high uncertainty, as we would expect.

2 Appendix

2.1 Plot Visible Points

```
clear
close all

load dataset3.mat

% Plot visible points over time
num_visible_features = zeros(1,500);
colours = zeros(500,3);
k1 = 1215;
k2 = 1714;
for k = k1:k2
    visible_features = 0;
    for j = 1:20
        if y_k_j(:,k,j) ~= [-1;-1;-1;-1]
            visible_features = visible_features + 1;
        end
    end
    if visible_features >= 3
        colours(k-k1+1,:) = [0 255 0];
    else
        colours(k-k1+1,:) = [255 0 0];
    end
    num_visible_features(k-k1+1) = visible_features;
end

scatter(1:500,num_visible_features, 4, colours)
title("Number of visible points per timestep")
xlabel("Time")
ylabel("Number of visible points")
```

2.2 Batch Method

```
% AER 1513 Assignment 3

clear
close all

load dataset3.mat

% Q. 5 Gauss Newton process
D = [1 0 0;
     0 1 0;
     0 0 1;
     0 0 0];
baseline = b;
M = [fu 0 cu fu*b/2;
    0 fv cv 0;
    fu 0 cu -fu*b/2;
    0 fv cv 0];
Z = zeros(1,3);
Q = diag([0.0026,0.0021,0.00079,0.009,0.017,0.175]);
%y_noise = g_inverse([37.98 129.84 41.95 132.49]',fu,fv,cu,cv,b);
y_noise = camInv(M,[37.98 129.84 41.95 132.49]');
%R = 1000*diag([y_noise(1),y_noise(2),y_noise(3)]);
R = diag(y_var);

% Camera Jacobian
syms x_sym y_sym z_sym;
g = [fu*x_sym/z_sym + cu;
    fv*y_sym/z_sym+cv;
    fu*(x_sym-baseline)/z_sym+cu;
    fv*y_sym/z_sym + cv];

jac = jacobian(g,[x_sym,y_sym,z_sym]);

T_c_v = [C_c_v -C_c_v'*rho_v_c_v;
         0 0 0 1];
T_v_c = inv(T_c_v);
P_check_0 = 0.0001*eye(6);

k1 = 1215;
k2 = 1714;

t_k_avg = (t(k2)-t(k1))/(k2-k1);

% Create initiaal guess with one gt point and dead reckoning
T_v_i = [];
%r_i_vk_i_hat = [];
r_hat = [];
for k = k1:k2
    lead = 4*(k-k1)+1;
    if k == k1
        r_k = r_i_vk_i(1:3,k);
        C_k = vec2rot(-theta_vk_i(1:3,k));
        C_k_1 = C_k;
    else
        t_k = t(k) - t(k-1);
        %C_k_1 = T_v_i(1:3,(lead-4):(lead-2));
        r_k_1 = -C_k_1'*T_v_i(1:3,lead-1);
        %r_k_1 = r_hat(:,k-k1);
        d = t_k*v_vk_vk_i(:,k-1);
        psi_k_bold = w_vk_vk_i(:,k-1)*t_k;
        %psi_k = sum(psi_k_bold);
        %Psi_k = cos(psi_k)*eye(3) + (1-cos(psi_k))*(psi_k_bold/psi_k)*(psi_k_bold/psi_k)'-sin(psi_k)*skew(psi
        Psi_k = vec2rot(-psi_k_bold);
        r_k = r_k_1 + C_k_1'*d;
        C_k = Psi_k*C_k_1;
        C_k_1 = C_k;
        %disp(C_k*C_k')
    end
    r_hat = [r_hat r_k];
    T_v_k_i = [C_k -C_k*r_k;
               Z 1];
    T_v_i = [T_v_i T_v_k_i];
end
init_T = T_v_i;
% Plot dead reckoning vs real data
x_op_len = size(T_v_i);
x_op_len = x_op_len(2);
figure('Name',"Original guess")
%plot3(T_v_i(1,4:4:x_op_len),T_v_i(2,4:4:x_op_len),T_v_i(3,4:4:x_op_len));
plot3(r_hat(1,:),r_hat(2,:),r_hat(3,:));
xlabel('x')
ylabel('y')
zlabel('z')
hold on
plot3(r_i_vk_i(1,k1:k2),r_i_vk_i(2,k1:k2),r_i_vk_i(3,k1:k2),'r');
hold off

% Iterative Gauss-Newton Method

for i = 1:5
    tic;
    % Initialize stacked values
    %W = P_check_0;
    J_v = 0;
    J_y = 0;
    W = [];
    F = [];
    G = [];
    Q_vals = [];
    R_vals = [];
    e_v_x_op = [];
    e_y_x_op = [];
    J_v = 0;
    J_y = 0;
    % Construct stacked values
    for k = k1:k2
        lead = 4*(k-k1)+1;
        % Setup
        t_k = t(k) - t(k-1);
        T_op_k = T_v_i(:,lead:lead+3); % Current guess

        % Motion Model
        if k == k1
            Q_k = P_check_0;
            T_check_0 = T_op_k;
            e_v_k_x_op = unskew_overload(logm(T_check_0/T_op_k));
        else
            Q_k = t_k^2*Q;
            varpi = [v_vk_vk_i(:,k);w_vk_vk_i(:,k)];
            Xi_k = vec2tran(-t_k*varpi);
            T_op_k_1 = T_v_i(:,lead-4:lead-1);
            e_v_k_x_op = unskew_overload(logm(Xi_k*T_op_k_1/T_op_k));
            if imag(e_v_k_x_op) ~= [0;0;0;0;0;0]
                error('Imaginary');
            end
            F_k_1 = tranAd(T_op_k/T_op_k_1);
            F = blkdiag(F,-F_k_1);
        end
        Q_vals = blkdiag(Q_vals,Q_k);
        J_v = J_v + e_v_k_x_op'/Q_k*e_v_k_x_op/2;
        e_v_x_op = [e_v_x_op;
                    e_v_k_x_op];

        % Observation model
        e_y_k_x_op = [];
        J_y_k = 0;
        G_k = zeros(0,6);
        for j = 1:20
            if y_k_j(1,k,j) == -1
                continue
            end
            R_vals = blkdiag(R_vals,R);
            %pixel value of observation
            y_k_j_p = y_k_j(:,k,j);

            % Ground point inertial frame
            p_j_i = rho_i_pj_i(:,j);
            C_v_i = T_op_k(1:3,1:3);
            r_i_v_i = -C_v_i'*T_op_k(1:3,4);

            % Ground point in camera frame
            p_c_p_c = [C_c_v*(C_v_i*(p_j_i-r_i_v_i)-rho_v_c_v);1];

            % Ground point in pixels
            x = p_c_p_c(1);
            y = p_c_p_c(2);
            z = p_c_p_c(3);
            ul = fu*x/z + cu;
            vl = fv*y/z + cv;
            ur = fu*(x-baseline)/z + cu;
            vr = fv*y/z + cv;
            p_p = [ul vl ur vr]';

            e_y_j_k_x_op = y_k_j_p - p_p;

            G_j_k = eval(subs(jac,[x_sym y_sym z_sym],[x y z]))*D'*T_c_v*circle_dot(T_op_k*[p_j_i;1]);

            G_k = [G_k;
                   G_j_k];
            J_y_k = J_y_k + e_y_j_k_x_op'/R*e_y_j_k_x_op/2;
            e_y_k_x_op = [e_y_k_x_op;
                          e_y_j_k_x_op];
        end

        G = blkdiag(G,G_k);
        J_y = J_y + J_y_k;
        e_y_x_op = [e_y_x_op;
                    e_y_k_x_op];

    end
    e_x_op = [e_v_x_op;
              e_y_x_op];
    F_size = size(F);
    F_padded = [zeros(6,F_size(2)+6);
                F zeros(F_size(1),6)];
    H_top = eye(F_size(2)+6)+F_padded;
    H_bottom = G;
    H = [H_top;
         H_bottom];
    W = blkdiag(Q_vals,R_vals);

    A = H'/W*H;
    b = H'/W*e_x_op;

    delta_x_star = A\b;
    for k = k1:k2
        lead = 4*(k-k1)+1;
        T_op_k = T_v_i(:,lead:lead+3);
        eps_star_k = delta_x_star(6*(k-k1)+1:6*(k-k1)+6);
        T_v_i(:,lead:lead+3) = expm(skew_overload(eps_star_k))*T_op_k;
    end
    J = J_v + J_y;
    toc
end

r_check = [];
theta_check = [];
for k = k1:k2
    lead = 4*(k-k1)+1;
    C_k = T_v_i(1:3,lead:lead+2);
    r_k = -C_k'*T_v_i(1:3,lead+3);
    r_check = [r_check r_k];
    theta_check = [theta_check rot2vec(C_k)];
end

x_op_len = size(T_v_i);
x_op_len = x_op_len(2);
figure('Name', "New guess")
plot3(r_check(1,:),r_check(2,:), r_check(3,:));
xlabel('x')
ylabel('y')
zlabel('z')
hold on
plot3(r_i_vk_i(1,k1:k2),r_i_vk_i(2,k1:k2),r_i_vk_i(3,k1:k2),'r');
hold off

delta_r = r_i_vk_i(:,k1:k2) - r_check;
delta_theta = [];
for k = k1:k2
    lead = 4*(k-k1)+1;
    C_vk_i_star = vec2rot(-theta_vk_i(1:3,k));
    C_vk_i = T_v_i(1:3,lead:lead+2);
    delta_theta_k = unskew(eye(3) - C_vk_i_star*C_vk_i');
    delta_theta = [delta_theta delta_theta_k];
end
sigma = diag(inv(A));


figure('Name', "X error vs time")
plot(delta_r(1,:))
hold on
plot(3*sqrt(sigma(1:6:6*(k2-k1+1))),'--')
hold on
plot(-3*sqrt(sigma(1:6:6*(k2-k1+1))),'--')

figure('Name', "Y error vs time")
plot(delta_r(2,:))
hold on
plot(3*sqrt(sigma(2:6:6*(k2-k1+1))),'--')
hold on
plot(-3*sqrt(sigma(2:6:6*(k2-k1+1))),'--')

figure('Name', "Z error vs time")
plot(delta_r(3,:))
hold on
plot(3*sqrt(sigma(3:6:6*(k2-k1+1))),'--')
hold on
plot(-3*sqrt(sigma(3:6:6*(k2-k1+1))),'--')


figure('Name', "Theta X error vs time")
plot(delta_theta(1,:))
hold on
plot(3*sqrt(sigma(4:6:6*(k2-k1+1))),'--')
hold on
plot(-3*sqrt(sigma(4:6:6*(k2-k1+1))),'--')

figure('Name', "Theta Y error vs time")
plot(delta_theta(2,:))
hold on
plot(3*sqrt(sigma(5:6:6*(k2-k1+1))),'--')
hold on
plot(-3*sqrt(sigma(5:6:6*(k2-k1+1))),'--')

figure('Name', "Theta Z error vs time")
plot(delta_theta(3,:))
hold on
plot(3*sqrt(sigma(6:6:6*(k2-k1+1))),'--')
hold on
plot(-3*sqrt(sigma(6:6:6*(k2-k1+1))),'--')
```

2.3 Sliding Window

```
    % AER 1513 Assignment 3

clear
close all

load dataset3.mat

% Q. 5 Gauss Newton process - sliding window
D = [1 0 0;
     0 1 0;
     0 0 1;
     0 0 0];
baseline = b;
M = [fu 0 cu fu*b/2;
    0 fv cv 0;
    fu 0 cu -fu*b/2;
    0 fv cv 0];
Z = zeros(1,3);
Q = diag([0.0026,0.0021,0.00079,0.009,0.017,0.175]);
%y_noise = g_inverse([37.98 129.84 41.95 132.49]',fu,fv,cu,cv,b);
y_noise = camInv(M,[37.98 129.84 41.95 132.49]');
%R = 1000*diag([y_noise(1),y_noise(2),y_noise(3)]);
R = diag(y_var);

% Camera Jacobian
syms x_sym y_sym z_sym;
```

```
g = [fu*x_sym/z_sym + cu;
    fv*y_sym/z_sym+cv;
    fu*(x_sym-baseline)/z_sym+cu;
    fv*y_sym/z_sym + cv];

jac = jacobian(g,[x_sym,y_sym,z_sym]);

T_c_v = [C_c_v -C_c_v'*rho_v_c_v;
         0 0 0 1];
T_v_c = inv(T_c_v);
P_check_0 = 0.0001*eye(6);

% Sliding window values
slide_k1 = 1215;
slide_k2 = 1714;
kappa = 10;
T_op = [];
sigma = [];

for slide_k = slide_k1:slide_k2
    k1 = slide_k;
    k2 = slide_k + kappa;

    % Create initial guess with one gt point and dead reckoning
    T_v_i = [];
    %r_i_vk_i_hat = [];
    r_hat = [];
    for k = k1:k2
        lead = 4*(k-k1)+1;
        slide_lead = 4*(slide_k-slide_k1)+1;
        if k == slide_k1
            r_k = r_i_vk_i(1:3,k);
            C_k = vec2rot(-theta_vk_i(1:3,k));
        elseif k == k1
            %C_k = T_op(1:3,(slide_lead):(slide_lead+2));
            %C_k = C_k_dr;
            %r_k = -C_k'*T_op(1:3,slide_lead+3);
            %r_k = r_k_dr;
            t_k = t(k) - t(k-1);
            C_k_1 = T_op(1:3,(slide_lead-4):(slide_lead-2));
            r_k_1 = -C_k_1'*T_op(1:3,slide_lead-1);
            %r_k_1 = r_hat(:,k-k1);
            d = t_k*v_vk_vk_i(:,k-1);
            psi_k_bold = w_vk_vk_i(:,k-1)*t_k;
            Psi_k = vec2rot(-psi_k_bold);
            r_k = r_k_1 + C_k_1'*d;
            C_k = Psi_k*C_k_1;
            %C_k_1 = C_k;
        else
            t_k = t(k) - t(k-1);
            C_k_1 = T_v_i(1:3,(lead-4):(lead-2));
            r_k_1 = -C_k_1'*T_v_i(1:3,lead-1);
            %r_k_1 = r_hat(:,k-k1);
            d = t_k*v_vk_vk_i(:,k-1);
            psi_k_bold = w_vk_vk_i(:,k-1)*t_k;
            Psi_k = vec2rot(-psi_k_bold);
            r_k = r_k_1 + C_k_1'*d;
            C_k = Psi_k*C_k_1;
            %C_k_1 = C_k;
        end
        r_hat = [r_hat r_k];
        T_v_k_i = [C_k -C_k*r_k;
                   Z 1];
        T_v_i = [T_v_i T_v_k_i];
    end
    %init_T = T_v_i;

    % Plot dead reckoning vs real data

    %figure('Name',"Original guess")
    %plot3(r_hat(1,:),r_hat(2,:),r_hat(3,:));
    %xlabel('x')
    %ylabel('y')
    %zlabel('z')
    %hold on
    %plot3(r_i_vk_i(1,k1:k2),r_i_vk_i(2,k1:k2),r_i_vk_i(3,k1:k2),'r');
    %hold on
```

```
% Iterative Gauss-Newton Method

for i = 1:5
    % Initialize stacked values
    %W = P_check_0;
    J_v = 0;
    J_y = 0;
    W = [];
    F = [];
    G = [];
    Q_vals = [];
    R_vals = [];
    e_v_x_op = [];
    e_y_x_op = [];
    % Construct stacked values
    for k = k1:k2
        lead = 4*(k-k1)+1;
        % Setup
        t_k = t(k) - t(k-1);
        T_op_k = T_v_i(:,lead:lead+3); % Current guess

        % Motion Model
        if k == k1
            Q_k = P_check_0;
            T_check_0 = T_op_k;
            e_v_k_x_op = unskew_overload(logm(T_check_0/T_op_k));
        else
            Q_k = t_k^2*Q;
            varpi = [v_vk_vk_i(:,k);w_vk_vk_i(:,k)];
            Xi_k = vec2tran(-t_k*varpi);
            T_op_k_1 = T_v_i(:,lead-4:lead-1);
            e_v_k_x_op = unskew_overload(logm(Xi_k*T_op_k_1/T_op_k));
            F_k_1 = tranAd(T_op_k/T_op_k_1);
            F = blkdiag(F,-F_k_1);
        end
        Q_vals = blkdiag(Q_vals,Q_k);
        J_v = J_v + e_v_k_x_op'/Q_k*e_v_k_x_op/2;
        e_v_x_op = [e_v_x_op;
                    e_v_k_x_op];
        % Observation model
        e_y_k_x_op = [];
        J_y_k = 0;
        G_k = zeros(0,6);
        for j = 1:20
            if y_k_j(1,k,j) == -1
                continue
            end
            R_vals = blkdiag(R_vals,R);
            %pixel value
            y_k_j_p = y_k_j(:,k,j);
            % Ground point inertial frame
            p_j_i = rho_i_pj_i(:,j);
            C_v_i = T_op_k(1:3,1:3);
            r_i_v_i = -C_v_i'*T_op_k(1:3,4);
            % Ground point in camera frame
            p_c_p_c = [C_c_v*(C_v_i*(p_j_i-r_i_v_i)-rho_v_c_v);1];
            % Ground point in pixels
            x = p_c_p_c(1);
            y = p_c_p_c(2);
            z = p_c_p_c(3);
            ul = fu*x/z + cu;
            vl = fv*y/z + cv;
            ur = fu*(x-baseline)/z + cu;
            vr = fv*y/z + cv;
            p_p = [ul vl ur vr]';
            e_y_j_k_x_op = y_k_j_p - p_p;
            G_j_k = eval(subs(jac,[x_sym y_sym z_sym],[x y z]))*D'*T_c_v*circle_dot(T_op_k*[p_j_i;1]);
            G_k = [G_k;
                   G_j_k];
            J_y_k = J_y_k + e_y_j_k_x_op'/R*e_y_j_k_x_op/2;
            e_y_k_x_op = [e_y_k_x_op;
                          e_y_j_k_x_op];
        end

        G = blkdiag(G,G_k);
        J_y = J_y + J_y_k;
        e_y_x_op = [e_y_x_op;
                    e_y_k_x_op];
        end
        e_x_op = [e_v_x_op;
                  e_y_x_op];
        F_size = size(F);
        F_padded = [zeros(6,F_size(2)+6);
                    F zeros(F_size(1),6)];
        H_top = eye(F_size(2)+6)+F_padded;
        H_bottom = G;
        H = [H_top;
             H_bottom];
        W = blkdiag(Q_vals,R_vals);
        A = H'/W*H;
        b = H'/W*e_x_op;

        delta_x_star = A\b;
        for k = k1:k2
            lead = 4*(k-k1)+1;
            T_op_k = T_v_i(:,lead:lead+3);
            eps_star_k = delta_x_star(6*(k-k1)+1:6*(k-k1)+6);
            T_v_i(:,lead:lead+3) = expm(skew_overload(eps_star_k))*T_op_k;
        end
        J = J_v + J_y;
    end
    T_op = [T_op T_v_i(1:4,1:4)];
    C_k_dr = T_v_i(1:3,5:7);
    r_k_dr = -C_k_dr'*T_v_i(1:3,8);
    sigma_k = diag(inv(A));
    sigma = [sigma sigma_k(1:6)];
end

r_check = [];
%theta_check = [];
for k = slide_k1:slide_k2
    lead = 4*(k-slide_k1)+1;
    C_k = T_op(1:3,lead:lead+2);
    r_k = -C_k'*T_op(1:3,lead+3);
    r_check = [r_check r_k];
    %theta_check = [theta_check rot2vec(C_k)];
end

delta_r = r_i_vk_i(:,slide_k1:slide_k2) - r_check;
delta_theta = [];
for k = slide_k1:slide_k2
    lead = 4*(k-slide_k1)+1;
    C_vk_i_star = vec2rot(-theta_vk_i(1:3,k));
    C_vk_i = T_op(1:3,lead:lead+2);
    delta_theta_k = unskew(eye(3) - C_vk_i_star*C_vk_i');
    delta_theta = [delta_theta delta_theta_k];
end
%sigma = diag(inv(A));


figure('Name', "X error vs time")
plot(delta_r(1,:))
hold on
plot(3*sqrt(sigma(1:6:6*(slide_k2-slide_k1+1))),'--')
hold on
plot(-3*sqrt(sigma(1:6:6*(slide_k2-slide_k1+1))),'--')

figure('Name', "Y error vs time")
plot(delta_r(2,:))
hold on
plot(3*sqrt(sigma(2:6:6*(slide_k2-slide_k1+1))),'--')
hold on
plot(-3*sqrt(sigma(2:6:6*(slide_k2-slide_k1+1))),'--')

figure('Name', "Z error vs time")
plot(delta_r(3,:))
hold on
plot(3*sqrt(sigma(3:6:6*(slide_k2-slide_k1+1))),'--')
hold on
plot(-3*sqrt(sigma(3:6:6*(slide_k2-slide_k1+1))),'--')


figure('Name', "Theta X error vs time")
plot(delta_theta(1,:))
hold on
plot(3*sqrt(sigma(4:6:6*(slide_k2-slide_k1+1))),'--')
```

```
hold on
plot(-3*sqrt(sigma(4:6:6*(slide_k2-slide_k1+1))),'--')

figure('Name', "Theta Y error vs time")
plot(delta_theta(2,:))
hold on
plot(3*sqrt(sigma(5:6:6*(slide_k2-slide_k1+1))),'--')
hold on
plot(-3*sqrt(sigma(5:6:6*(slide_k2-slide_k1+1))),'--')

figure('Name', "Theta Z error vs time")
plot(delta_theta(3,:))
hold on
plot(3*sqrt(sigma(6:6:6*(slide_k2-slide_k1+1))),'--')
hold on
plot(-3*sqrt(sigma(6:6:6*(slide_k2-slide_k1+1))),'--')
```