Aniket Sanjay Gujarathi
1007629979

Assignment 1

AER 1513
October 8, 2021

Problem 1

According to the given data, the histogram of sensor errors suggest a Gaussian distribution with zero mean. As the velocity values from the odometer are taken as inputs and the laser rangefinder values are taken as measurements, they satisfy the zero mean Gaussian assumption and hence it is reasonable to consider similar zero-mean Gaussian distribution for the associated process and measurement noise.

The values for the variances are $\sigma_q^2 = 0.0023m^2/s^2$ obtained from the odometer values and $\sigma_r^2 = 3.6692e^{-04}m^2$ obtained from the laser rangefinder values.

Problem 2

For the model

$$\text{motion: } x_k = x_{k-1} + Tu_k + w_k \tag{1}$$

$$\text{observation: } y_k := x_c - r_k = x_k + n_k \tag{2}$$

with

$$\mathbf{A_{k-1}} = [1], \mathbf{C_k} = [1], \mathbf{v} = \begin{bmatrix} Tv_1 \\ Tv_2 \\ \vdots \\ Tv_k \end{bmatrix}_{k\times 1}, \mathbf{y} = \begin{bmatrix} x_c - r_1 \\ x_c - r_2 \\ \vdots \\ x_c - r_k \end{bmatrix}_{k\times 1} \tag{3}$$

$$\mathbf{Q} = \begin{bmatrix} \sigma_q^2 & & & & \\ & \sigma_q^2 & & & \\ & & \ddots & & \\ & & & \sigma_q^2 & \\ & & & & \sigma_q^2 \end{bmatrix}_{k\times k}, \mathbf{R} = \begin{bmatrix} \sigma_r^2 & & & & \\ & \sigma_r^2 & & & \\ & & \ddots & & \\ & & & \sigma_r^2 & \\ & & & & \sigma_r^2 \end{bmatrix}_{k\times k} \tag{4}$$

we need to find the argument that minimizes the objective function. As we have no information of our initial state, let's assume the initial estimate as the first measurement from the rangefinder. This is possible in the given case as we can reconstruct our state from measurements from the sensors, but would not be the case where we cannot reconstruct all the states using measurement readings.

$$x_1 = y_1 = x_c - r_1, P_1 = R_1 \tag{5}$$

The expression for batch-linear Gaussian objective function to minimize is:

$$J(\mathbf{x_{1:k}}|\mathbf{v_{1:k}}, \mathbf{y_{1:k}}) = \sum_{k=1}^{K} \left(J_{v,k}(\mathbf{x}) + J_{y,k}(\mathbf{x})\right) \tag{6}$$

with

$$\begin{aligned} J_{v,k}(\mathbf{x}) &= \begin{cases} \frac{1}{2}(x_1 - \check{x_1})^\top \mathbf{\check{P}_1^{-1}}(x_1 - \check{x_1}) & k = 1 \\ \frac{1}{2}\left(\mathbf{x}_k - \mathbf{A}_{k-1}\mathbf{x}_{k-1} - \mathbf{v}_k\right)^T \mathbf{Q}_k^{-1}\left(\mathbf{x}_k - \mathbf{A}_{k-1}\mathbf{x}_{k-1} - \mathbf{v}_k\right) & k = 2 \dots K \end{cases} \\ J_{y,k}(\mathbf{x}) &= \frac{1}{2}\left(\mathbf{y}_k - \mathbf{C}_k\mathbf{x}_k\right)^T \mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{C}_k\mathbf{x}_k\right) \qquad k = 2 \dots K \end{aligned} \tag{7}$$

Problem 3

For the purpose of better visibility, let us write some initial equations:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{1}{2}(x_1 - y_1)R_1(x_1 - y_1) \\ \frac{1}{2}(x_2 - A_1x_1 - v_2)Q_2(x_2 - A_1x_1 - v_2) + \frac{1}{2}(y_2 - C_2x_2) \\ \vdots \end{bmatrix} \tag{8}$$

This can be further decoupled and written in the stacked form as:
First Term :

$$\begin{bmatrix} y_1 \\ v_2 \\ v_3 \\ \vdots \\ v_k \\ y_1 \\ y_2 \\ \vdots \\ y_{k-1} \\ y_k \end{bmatrix} - \begin{bmatrix} 1 & & & & & & \\ -A_1 & 1 & & & & & \\ & -A_2 & 1 & & & & \\ & & \ddots & \ddots & & & \\ & & & & -A_{k-1} & 1 & \\ C_1 & & & & & & \\ & C_2 & & & & & \\ & & C_3 & & & & \\ & & & & \ddots & & \\ & & & & & C_{k-1} & \\ & & & & & & C_k \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} \tag{9}$$

Middle Term:

$$\begin{bmatrix} R_1 & & & & & \\ & Q_2 & & & & \\ & & Q_3 & & & \\ & & & \ddots & & \\ & & & & Q_k & \\ R_1 & & & & & \\ & R_2 & & & & \\ & & R_3 & & & \\ & & & \ddots & & \\ & & & & R_k & \end{bmatrix} \tag{10}$$

The cost function can now be written compactly as :

$$\mathbf{J}(\mathbf{x}) = \frac{1}{2}(\mathbf{z} - \mathbf{H}\mathbf{x})^\top \mathbf{W^{-1}} (\mathbf{z} - \mathbf{H}\mathbf{x}) \tag{11}$$

with

$$\mathbf{z} = \begin{bmatrix} \mathbf{v} \\ \mathbf{y} \end{bmatrix}, \mathbf{H} = \begin{bmatrix} A^{-1} \\ C \end{bmatrix}, \mathbf{W} = \begin{bmatrix} \mathbf{Q} & \\ & \mathbf{R} \end{bmatrix} \tag{12}$$

where the lifted matrices are as follows

$$\mathbf{A} = \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ \vdots & & \ddots & \\ 1 & 1 & \dots & 1 \end{bmatrix}_{k \times k}, \mathbf{C} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}_{k \times k} \tag{13}$$

As the cost function is a quadratic, we can find its minimum by setting the partial derivative with respect to x, to zero.

$$\frac{\partial J(x)}{\partial x^\top}\Big|_{\hat{x}} = \mathbf{H}^\top \mathbf{W^{-1}} (\mathbf{z} - \mathbf{H}\hat{\mathbf{x}}) = 0 \tag{14}$$

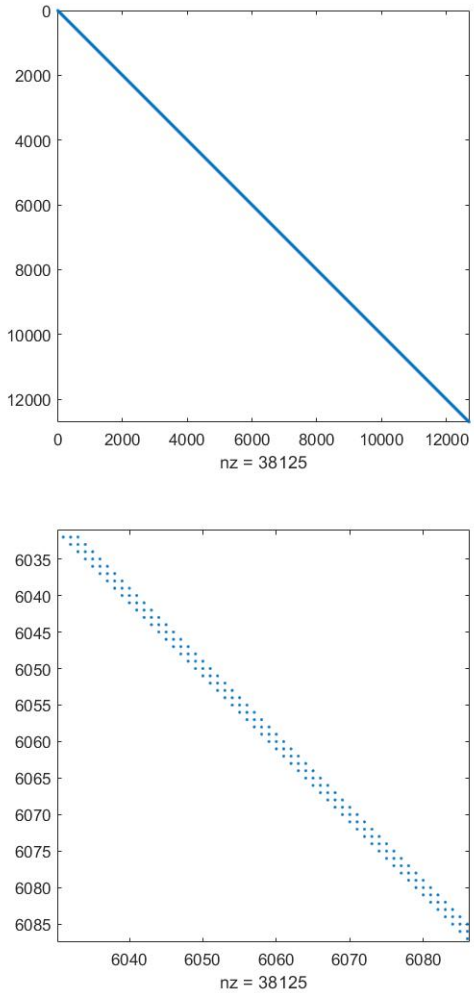
$$(\mathbf{H}^\top \mathbf{W^{-1}} \mathbf{H}) \hat{\mathbf{x}} = \mathbf{H}^\top \mathbf{W^{-1}} \mathbf{z} \tag{15}$$

By solving this system of linear equations, we can get the optimal position estimates $\hat{\mathbf{x}}$.

Problem 4

As K is a large number (12709) it is computationally expensive to calculate the inverse of $\mathbf{H}^\top\mathbf{W}^{-1}\mathbf{H}$. The expensive step in this is due to the calculation of A^{-1}, but as our system obeys the Markov property that the state at the next timestep only depends on the state at the previous timestep, we get a sparse A^{-1} matrix as shown in Equation 9. Due to this, we get a sparse block tri-diagonal matrix of $\mathbf{H}^\top\mathbf{W}^{-1}\mathbf{H}$ which is relatively less expensive to compute.

Figure 1: Sparse Pattern



Due to the sparse nature of $\mathbf{H}^\top\mathbf{W}^{-1}\mathbf{H}$ in Fig 1, we can solve for $\hat{x}$ in $\mathcal{O}(K)$ instead of $\mathcal{O}(K^3)$. Further, it is much easier to factorize $H^\top W^{-1} H$ using known factorization methods like Cholesky factorization or Rach-Tung-Striebel (RTS) to get the desired values of $\hat{x}$ and $\hat{P}$.

For this case, I have chosen the RTS smoother to exploit the sparsity pattern. The RTS smoother works in two steps : forward pass and a backwards pass.
For the forward pass, $k = 2 \dots K$:

$$\check{\mathbf{P}}_{k,f} = \mathbf{A}_{k-1}\hat{\mathbf{P}}_{k-1,f}\mathbf{A}_{k-1}^{T} + \mathbf{Q}_k \tag{16}$$
$$\check{\mathbf{x}}_{k,f} = \mathbf{A}_{k-1}\mathbf{x}_{k-1,f} + \mathbf{v}_k \tag{17}$$
$$\mathbf{K}_k = \mathbf{P}_{k,f}\mathbf{C}_k^{T}\left(\mathbf{C}_k\check{\mathbf{P}}_{k,f}\mathbf{C}_k^{T} + \mathbf{R}_k\right)^{-1} \tag{18}$$
$$\hat{\mathbf{P}}_{k,f} = \left(\mathbf{1} - \mathbf{K}_k\mathbf{C}_k\right)\check{\mathbf{P}}_{k,f} \tag{19}$$
$$\hat{\mathbf{x}}_{k,f} = \check{\mathbf{x}}_{k,f} + \mathbf{K}_k\left(\mathbf{y}_k - \mathbf{C}_k\check{\mathbf{x}}_{k,f}\right) \tag{20}$$

For the backwards pass, $k = K \dots 2$:

$$\hat{\mathbf{x}}_{k-1} = \hat{\mathbf{x}}_{k-1,f} + \hat{\mathbf{P}}_{k-1,f}\mathbf{A}_{k-1}^{T}\check{\mathbf{P}}_{k-1,f}^{-1}\left(\hat{\mathbf{x}}_k - \check{\mathbf{x}}_{k,f}\right) \tag{21}$$
$$\hat{\mathbf{P}}_{\mathbf{k-1}} = \hat{\mathbf{P}}_{\mathbf{k-1,f}} + (\hat{\mathbf{P}}_{\mathbf{k-1,f}}\mathbf{A}_{\mathbf{k-1}}^{\top}\check{\mathbf{P}}_{\mathbf{k,f}}^{\mathbf{-1}})(\hat{\mathbf{P}}_{\mathbf{k-1,f}}\mathbf{A}_{\mathbf{k-1}}^{\top}\check{\mathbf{P}}_{\mathbf{k,f}}^{\mathbf{-1}})^{\top} \tag{22}$$

initialized with

$$\begin{aligned} \check{P}_{0,f} &= \check{P}_0 = R_1 \\ \check{x}_{0,f} &= \check{x}_0 = y_1 \\ \hat{x}_K &= \hat{x}_{K,f} \end{aligned} \tag{23}$$

Problem 5

By solving for the robot positions for all K timesteps, I get the following error plots for different values of delta (1, 10, 100, 1000):

For the case where $\delta = 1$, we correct our predictions on every timestep. We can see that the estimator is consistent, with the mean hovering near 0. Unlike the error plots for odometry errors which drift with time, now we have a correction for every timestep from the rangefinder measurements, and hence get low values of errors constrained within the 3 σ bounds. As seen in Fig. 3, we can see that the mean of errors over all timesteps is 0. Hence, our estimator is unbiased.

In the case of $\delta = 10$, the estimations are corrected by the measurements at every 10th step. As a result, the covariance values increase with respect to the condition where corrections were done at every timestep. As seen in Fig. 4, we can see that the covariance values increase as between the prediction steps, and decrease when the estimate is corrected through a measurement. At the last timestep, as there is no correction step after it, the covariance values seem to increase. However, even in this case, the errors still have approximately zero mean and are constrained within the 3 σ bound.
In the case of $\delta = 100$ in Fig. 6, the error seems to increase as the correction is done only

after every 100^{th} timestep. Hence, between that, the estimator only relies on the prediction from the motion model being propagated forward through the Gaussian process noise due to the odometer input values. Similarly in the case of $\delta = 1000$, the covariance and error values increase due to the lack of correction steps. However, in all the plots, we can see that the error is constrained within the $3\,\sigma$ bounds and hence our estimator is consistent and unbiased.

Figure 2: Error Plot ($\delta = 1$)

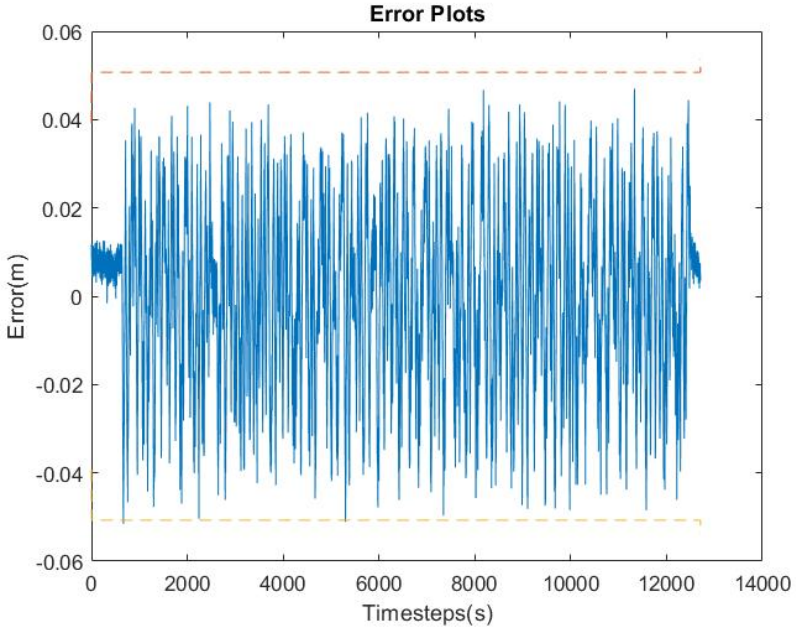


Figure 3: Histogram of errors ($\delta = 1$)

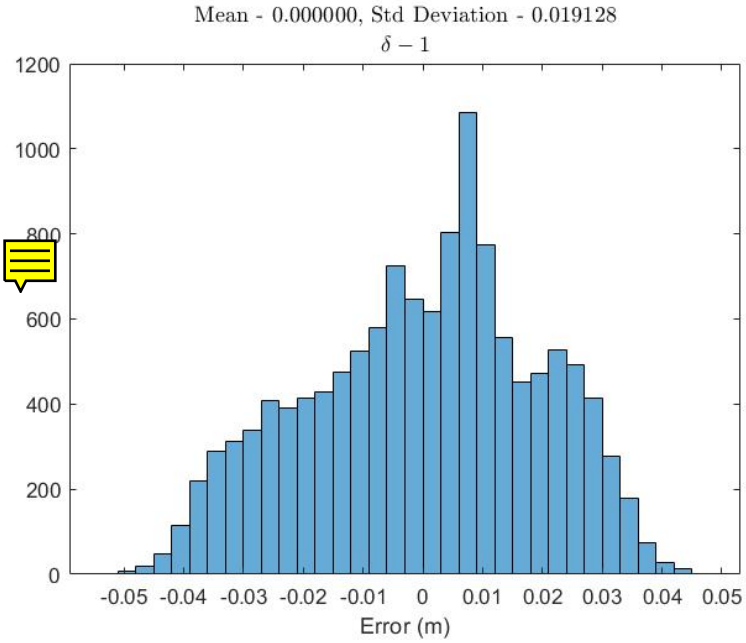


Figure 4: Error Plot ($\delta = 10$)

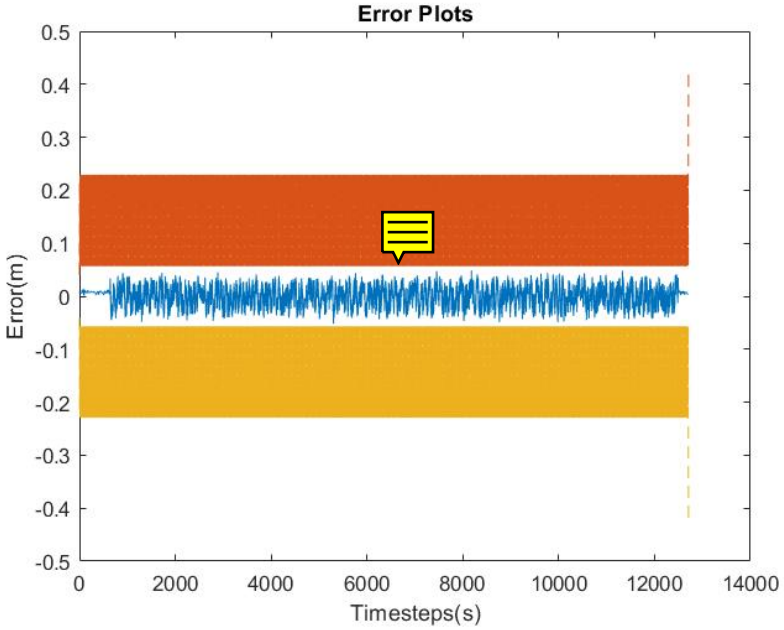


Figure 5: Histogram of errors ($\delta = 10$)

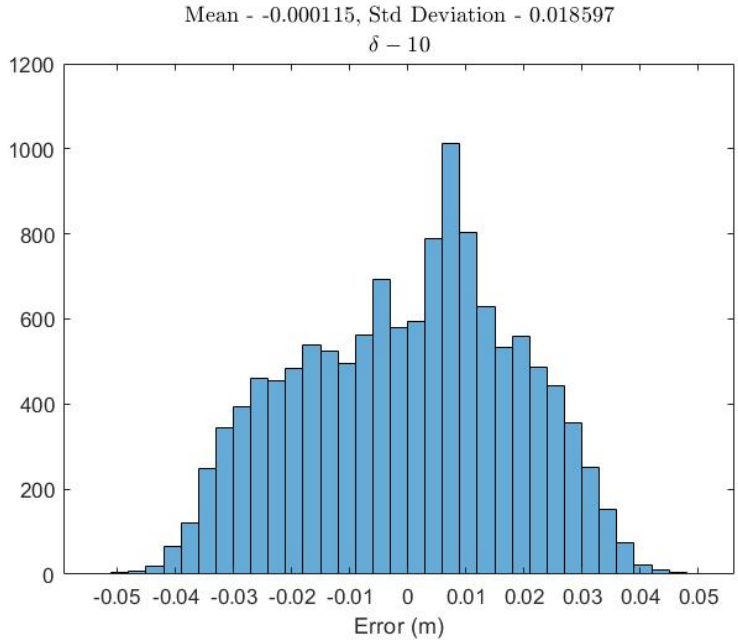


Figure 6: Error Plot ($\delta = 100$)

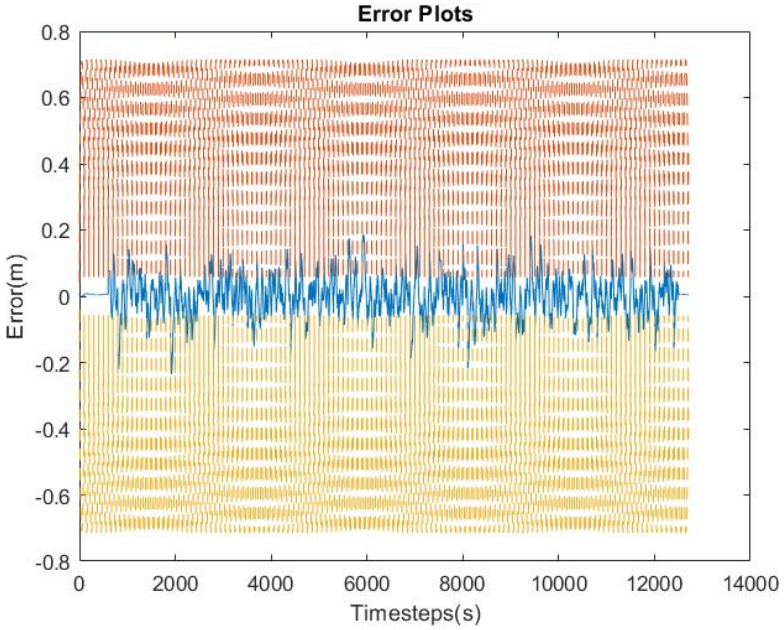


Figure 7: Histogram of errors ($\delta = 100$)

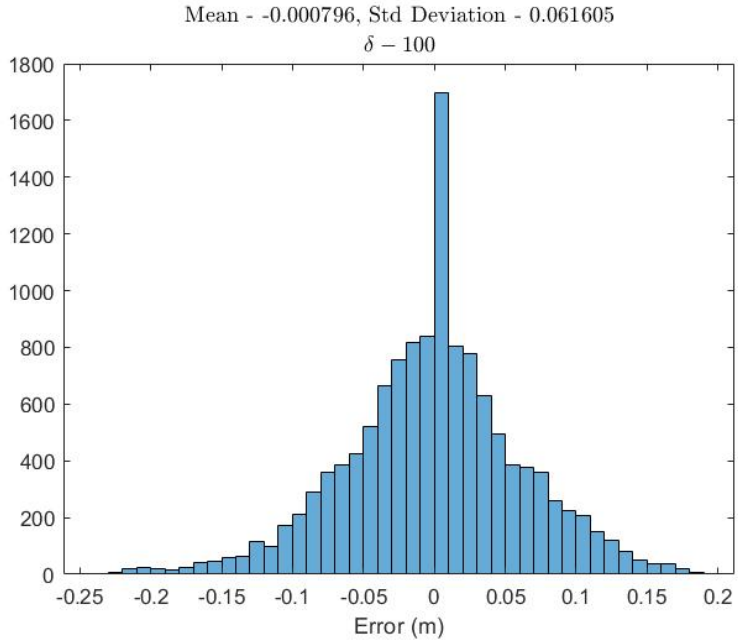


Figure 8: Error Plot ($\delta = 1000$)

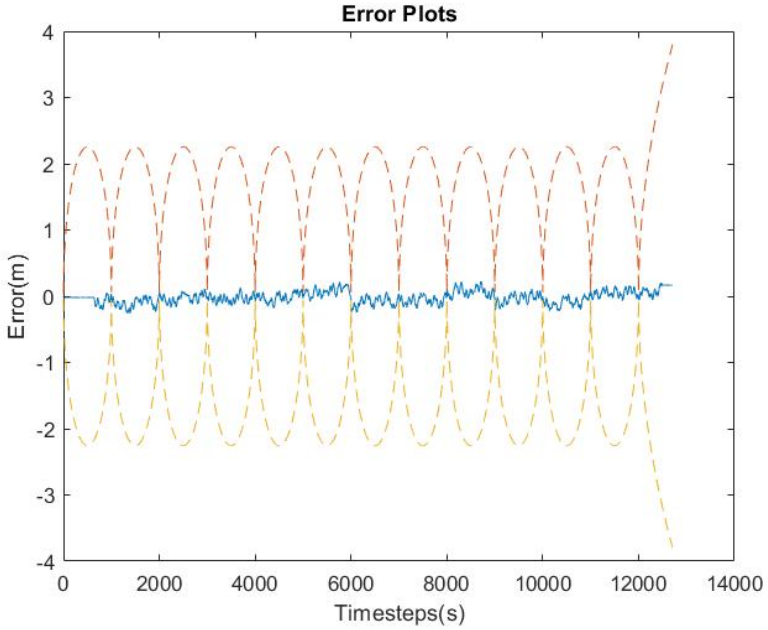
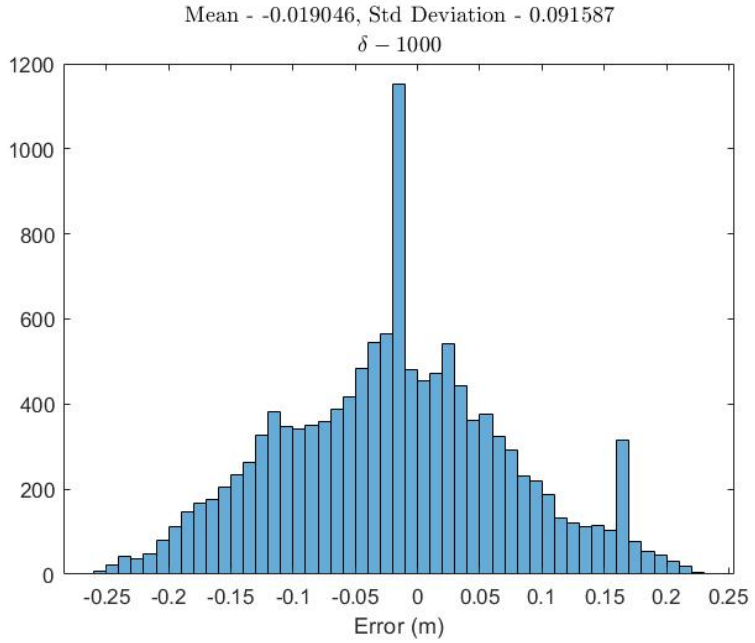


Figure 9: Histogram of errors ($\delta = 1000$)



Appendix

Code for estimating the state and covariance values using batch linear-Gaussian estimator.

```
% estimate x (mean and covariance)

%load dataset
load dataset1.mat;

k = size(v);

% assuming initial measurement as initial state
x0_check = l - r(1);

% the uncertainty of initial state will be the same as uncertainty in
% initial measurement
P0_check = r_var;

% measurements
y = l - r;

% z matrix = [v, y].T
v = v * 0.1;
v(1) = x0_check;
z = [v; y];

% A matrix in lifted form for given motion model
A = ones(k(1), k(1));
A = tril(A);

% A inverse
A_inv = sparse(inv(A));

% C matrix
C = eye(k(1));

% H = [A_inv, C].T
H = [A_inv; C];

% Q process noise
process_noise = v_var*ones(1, k(1));
process_noise(1) = P0_check;
Q = diag(process_noise);

```

```
41 % R sensor noise
42 R = r_var*diag(ones(1, k(1)));
43
44 % W = [Q, 0; 0 R] blockdiagonal matrix
45 W = blkdiag(Q, R);
46 W_inv = sparse(inv(W));
47
48 % H.T*W_inv*H
49 lhs = transpose(H) * W_inv * H;
50 spy(lhs);
51
52 % initalize
53 prompt = 'What is the delta ? ';
54 delta = input(prompt);
55 Phat_f = ones(k(1), 1);
56 xhat_f = ones(k(1), 1);
57 Pcheck_f = ones(k(1), 1);
58 xcheck_f = ones(k(1), 1);
59 xcheck_0f = y(1);
60 Pcheck_0f = Q(1, 1);
61 Pcheck_f(1) = Pcheck_0f;
62 xcheck_f(1) = xcheck_0f;
63
64 % P0_check = Q(1, 1)
65
66 K = Pcheck_0f * transpose(C(1, 1)) * inv((C(1, 1) *
67     Pcheck_0f * transpose(C(1, 1)) + R(1, 1)));
68 Phat_f(1) = (1 - K * C(1, 1)) * Pcheck_0f;
69 xhat_f(1) = xcheck_0f + K * (y(1) - C(1, 1) * xcheck_0f);
70
71
72 for i = 2 : k(1) %matlab indexing starts from 1
73     % forward pass
74     % prediction
75     xcheck_f(i) = A(i-1, i-1) * xhat_f(i - 1) + v(i);
76     Pcheck_f(i) = A(i - 1, i - 1) * Phat_f(i-1) * transpose(A(i - 1, i
           - 1)) + Q(i, i);
77     % correction
78     if mod(i, delta) == 0
79         K = Pcheck_f(i) * transpose(C(i, i)) * inv((C(i, i) * Pcheck_f(
               i) * transpose(C(i, i)) + R(i, i)));
80         Phat_f(i) = (1 - K*C(i, i)) * Pcheck_f(i);
81         xhat_f(i) = xcheck_f(i) + K * (y(i) - C(i, i) * xcheck_f(i));
82     else
```

```
83          Phat_f(i) = Pcheck_f(i);
84          xhat_f(i) = xcheck_f(i);
85      end
86  end
87  % initialize backward pass
88  xhat = ones(k(1), 1);
89  Phat = ones(k(1), 1);
90  Phat(k(1)) = Phat_f(k(1));
91  xhat(k(1)) = xhat_f(k(1));
92  for i = k(1) : -1 : 2
93      % backward pass
94      xhat(i - 1) = xhat_f(i - 1) + Phat_f(i - 1) * transpose(A(i - 1, i
            - 1)) * inv(Pcheck_f(i - 1)) * (xhat(i) - xcheck_f(i));
95      Phat(i - 1) = Phat_f(i - 1) + (Phat_f(i - 1) * transpose(A(i - 1, i
            - 1)) * inv(Pcheck_f(i))) * (Phat(i) - Pcheck_f(i)) * transpose
            (Phat_f(i - 1) * transpose(A(i - 1, i - 1)) * inv(Pcheck_f(i)));
96  end
97
98  % error plots
99
100 error = x_true - xhat;
101 x_axis = 1 : k(1);
102 y_axis = error;
103 figure(1);
104 plot(x_axis, y_axis);
105 hold on;
106 % 3 sigma plots
107 plot(sqrt(Phat) * 3, '--')
108 hold on;
109 plot(- sqrt(Phat) * 3, '--')
110 hold on;
111 title('Error Plots');
112 xlabel('Timesteps(s)');
113 ylabel('Error(m)');
114 % histogram
115 figure(2);
116 histogram(error);
117 title(sprintf('Mean - %f, Std Deviation - %f', mean(error), std(error))
        , '$\delta - 1000$', 'Interpreter', 'latex');
118 xlabel('Error (m)');
```