D&E: Midterm Problem #4

Due on Monday, March 31, 2014

Prof. Erdogmus 8:00am

Sam Nazari

EKF Study: Non-Gaussian Statistics & Nonlinearities

Problem Description

An autonomous vehicle is performing a constant radius turn around a fixed point $C = [c_x\ c_y]^T$. It is of interest to quantify, by means of a simulation, the performance of an EKF for estimating the states of this system. The key features of this problem are:

- The vehicle acceleration term is nonlinear and corrupted by AWGN.
- The vehicle observation/measurement model is nonlinear with a multiplicative non-Gaussian noise term.

The system simulation block diagram is shown in Figure 1 below. This system was implemented in Simulink with MATLAB code for initialization, Monte Carlo Simulation and post processing of the results. Also, the EKF was implemented as a single eML (embedded MATLAB) block. This allows one to insert MATLAB code into the Simulink block diagram.

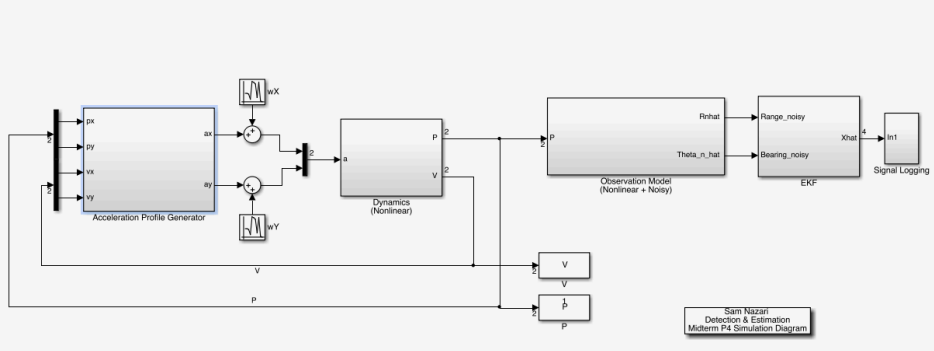

Figure 1: Simulink Block Diagram: Top Level

Discrete Time Dynamics

The vehicle dynamics provide the true state of the vehicle as it is undergoing noisy accelerations. The *discrete time dynamics* are:

$$\mathcal{S}: \begin{cases} P_{n+1} = P_n + V_n T_s + (a_n + w_n)(\frac{T_s^2}{2}) \\ V_{n+1} = V_n + (a_n + w_n)(\frac{T_s^2}{2}) \\ a_n = (C - P_n)\dfrac{\|V_n\|_2^2}{\|C - P_n\|_2^2} \end{cases} \tag{1}$$

Where $C, P_n, V_n \in \mathbb{R}^2$ and $w_n \in \mathbb{R}^2$ is a zm, Q-cov, white iid Gaussian process used to represent unmodeled dynamics such as turbulence or the nonuniformity of the surface the vehicle is traversing. The Gaussian process noise is characterized by the mean and covariance matrices:

$$\begin{bmatrix} w_x \\ w_y \end{bmatrix}_n \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_{wx}^2 & 0 \\ 0 & \sigma_{wy}^2 \end{bmatrix} \right) \tag{2}$$

The Simulink implementation of the vehicle dynamics are found in the simulation diagram inside the subsystems labeled *Dynamics* and *Acceleration Profile Generator*. The dynamics are nonlinear coupled difference

equations. This can be seen if they are written out in a more explicit form that is amenable for Simulink implementation. That is:

$$\begin{bmatrix} p_x \\ p_y \\ \hline v_x \\ v_y \end{bmatrix}_{n+1} = \begin{bmatrix} p_x \\ p_y \\ \hline v_x \\ v_y \end{bmatrix}_n + T_s \begin{bmatrix} v_x \\ v_y \\ \hline a_x + w_x \\ a_y + w_y \end{bmatrix}_n + \frac{T_s^2}{2} \begin{bmatrix} a_x + w_x \\ a_y + w_y \\ \hline 0 \\ 0 \end{bmatrix}_n \tag{3}$$

where

$$a_x = \frac{v_x^2 + v_y^2}{(c_x - p_x)^2 + (c_y - p_y)^2}(c_x - p_x) \tag{4}$$

$$a_y = \frac{v_x^2 + v_y^2}{(c_x - p_x)^2 + (c_y - p_y)^2}(c_y - p_y) \tag{5}$$

In MATLAB and Simulink, eqns (3),(4) and (5) are implemented within specific subsystems shown in Figure 1. The implementation of the difference equations is shown in Figure 2 and the acceleration profile implementation is shown in Figure 3. Note that, as can be seen from Figure 1, the inclusion of the AGWN to the acceleration terms $[a_x \;\; a_y]^T$ is happening between the acceleration profile subsystem and the discrete time dynamics subsystem.

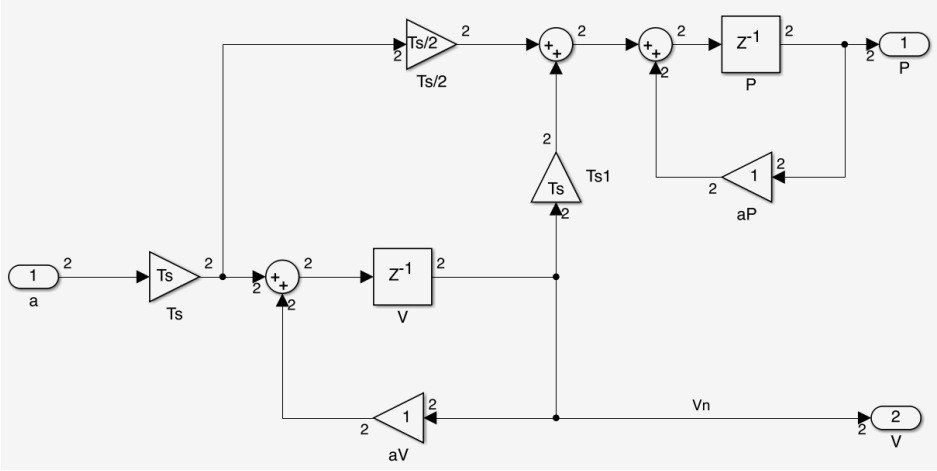

Figure 2: Simulink implementation of Eqn (3) found in the *Dynamics* subsystem

Measurement Model

The vehicle is moving around a fixed point. The distance from the fixed point to the vehicle is constant, C. We observe this motion from another fixed point. We measure the vehicle distance to our fixed point as it moves around the fixed point C and call this measurement the *range*. Similarly, we measure the heading angle of the vehicle. These two measurements are our observations and we shall refer to them as the *range* and *azimuth* observations, respectively. We suppose that our sensors are corrupted by noise and that we cannot make the range and azimuth measurements perfectly as desired. Specifically, we have:

- a range equation that is corrupted by multiplicative noise that is log-normally distributed

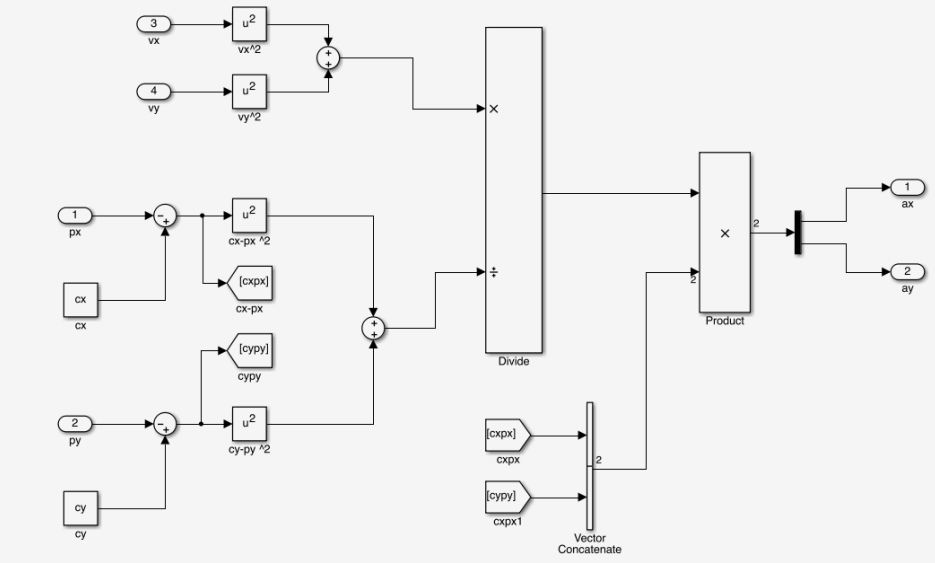

Figure 3: Simulink implementation of Eqn(4) and Eqn (5) found in the *Acceleration Profile Generator* subsystem

$$R_n = \tilde{u}_n^R \sqrt{p_x^2 + p_y^2}$$

where

$$\tilde{u}_n^R \sim \ \ln \mathcal{N}(0, \sigma_R^2)$$

is distributed according to the log-normal distribution with zm and σ_R variance.

- and an azimuth measurement equation that is corrupted by additive but truncated Gaussian noise

$$\theta_n = \arctan\left(\frac{p_y}{p_x}\right) + u_n^\theta$$

where the additive, truncated-Gaussian noise term u_n^θ is given by

$$P_{u_n^\theta} = \begin{cases} Ce^{-\frac{\xi^2}{2\sigma_{u_n^\theta}^2}}, & \text{if } |\xi| < \pi \\ 0, & \text{otherwise} \end{cases}$$

The Simulink implementation for the measurement model is found in the *Observation Model (Nonlinear + Noisy)* subsystem. The contents of that subsystem are shown in Figure 4. The Simulink block diagram contains two *interpreted MATLAB Function* blocks. Each of these blocks is needed to implement the log-normal noise distribution and the truncated Gaussian distribution. After some experimentation, it would found that the MATLAB Statistics Toolbox can be used to generate each of these distributions. The MATLAB Code that can produce these distributions is found in the `MIDTERMP4Driver.m` file and is displayed here for reference:

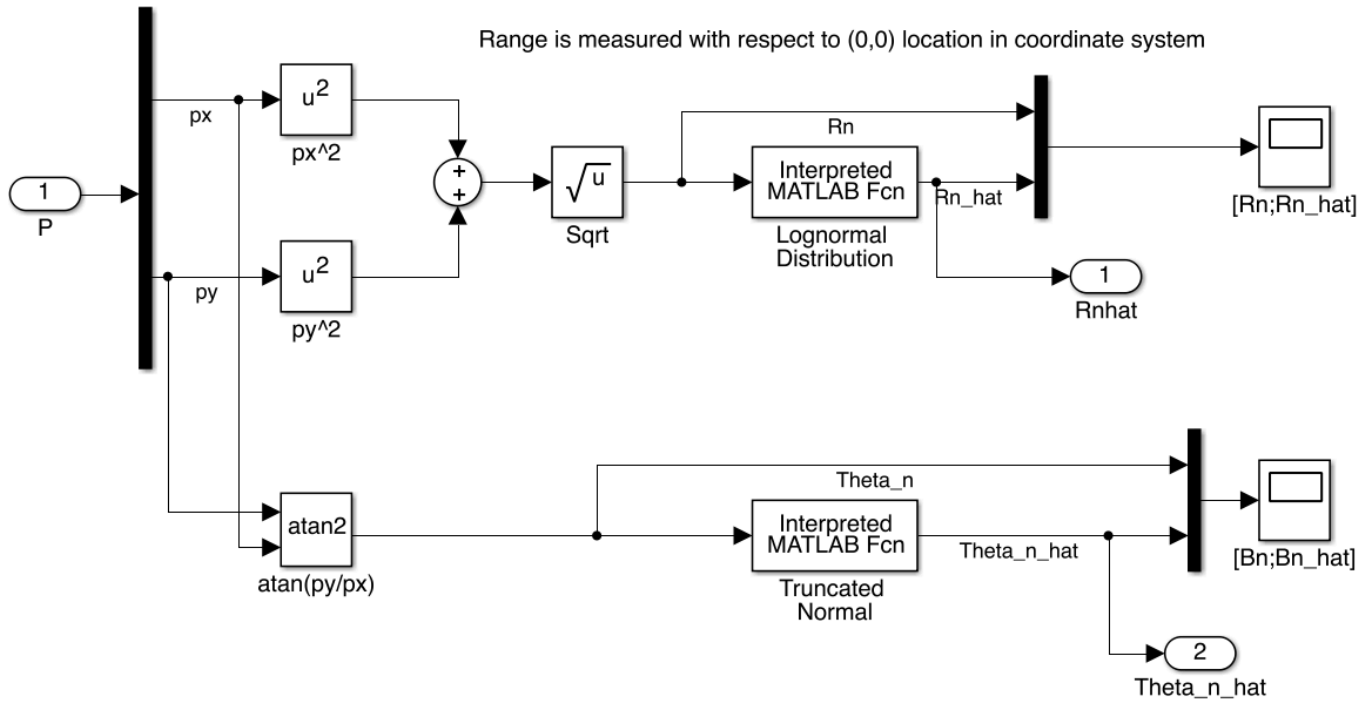

Figure 4: Simulation block diagram of the range and azimuth measurement model.

```
% Observation Parameters
pdR = makedist('Lognormal');
pdR.sigma = sigR;

pdB = makedist('Normal');
pdBT= truncate(pdB,-Z,Z);
pdBT.sigma = sigB;
```

The MATLAB Code to be used in the *interpreted MATLAB Function* blocks is

```
% To be entered at the MATLAB command prompt
u*random(pdR,1,1);
u+random(pdBT,1,1);
```

The observation noise intensities are parameters that can be set within a MATLAB script. This script is called `MIDTERMP4Driver.m` and is used to initialize the Simulink block diagram. The script is listed in this section for convenience.

A plot of the range observations is shown in Figure 5. The true range is shown in blue while the noisy observations are shown in red. Similarly, the true azimuth angle and the corresponding azimuth observations are shown in Figure 6. Figure 7 also shows the vehicle trajectory around the fixed point, C, over the course of one simulation run. The vehicle position is shown in blue in Figure 7. The big dot in the center of Figure 7 represents the fixed point that the vehicle is making a constant turn about.

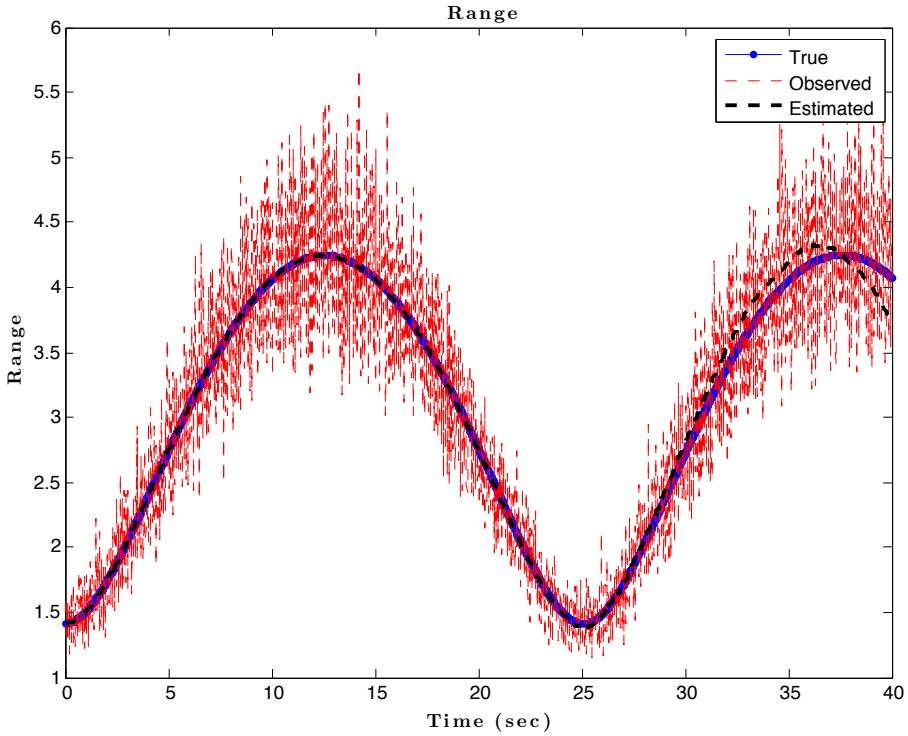

Figure 5: True (blue), observed (red) and estimated (black) ranges with perfect IC knowledge.

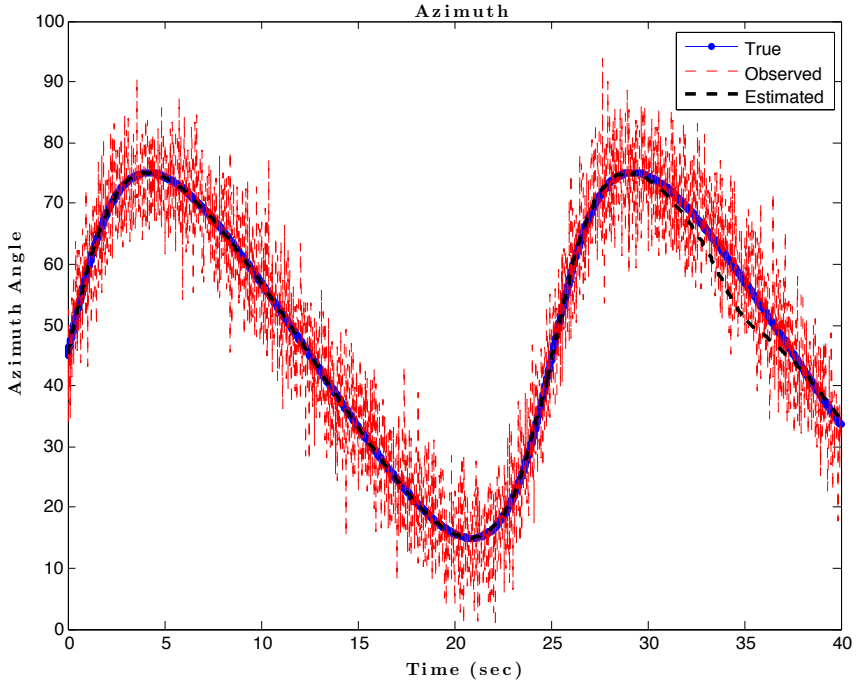

Figure 6: True (blue), observed (red) and estimated (black) azimuth with perfect IC knowledge.

Listing 1: MIDTERMP4Driver.m

```
%% Sam Nazari Midterm Detection & Estimation Theory P4
Ts      = 1/100;
N       = 4000;
TFinal  = Ts*N;
muWx    = 0;
muWy    = 0;
wx = 0; % Normal RNG Seed x
wy = 0; % Normal RNG Seed y
sigWx   = 1e-7;
sigWy   = 1e-7;
sigR    = 1e-1;
sigB    = 1e-1;
cx      = 2;
cy      = 2;
Z = 2;
% Initial Conditions
% V0 = [-0.25;0.25];
V0 = [-0.25;0.25];
P0 = [1;1];
M0 = 0*rand(1,1)*eye(4);
X0 = [P0;V0]+0*rand(4,1);
C0 = [cx;cy];

% Observation Parameters
pdR = makedist('Lognormal');
pdR.sigma = sigR;
pdB = makedist('Normal');
pdBT= truncate(pdB,-Z,Z);
pdBT.sigma = sigB;
%% Run Sim
sim('MIDTERMP4')
% Plot
set(0,'defaulttextinterpreter','latex')
plot(P(:,1),P(:,2),'-'),hold on,plot(pxhat,pyhat,'--k'),
scatter(cx,cy,250,'fill'),
title('\textbf{Trajectory Around C}'),
legend('True','Estimated'),xlabel('\textbf{px}'),ylabel('\textbf{py}')
figure,
plot(Rnhat(:,1),Rnhat(:,2),'.-'),hold on,
plot(Rnhat(:,1),Rnhat(:,3),'--r'),
plot(Rnhat(:,1),Rhat,'--k','linewidth',2),
legend('True','Observed','Estimated'),
title('\textbf{Range}'),xlabel('\textbf{Time (sec)}'),
ylabel('\textbf{Range}')
figure,
plot(Bnhat(:,1),Bnhat(:,2)*(180/pi),'.-'),hold on,
plot(Bnhat(:,1),Bnhat(:,3)*(180/pi),'--r'),
plot(Bnhat(:,1),Azhat*(180/pi),'--k','linewidth',2),
legend('True','Observed','Estimated'),
title('\textbf{Azimuth}'),xlabel('\textbf{Time (sec)}'),
ylabel('\textbf{Azimuth Angle}')
```

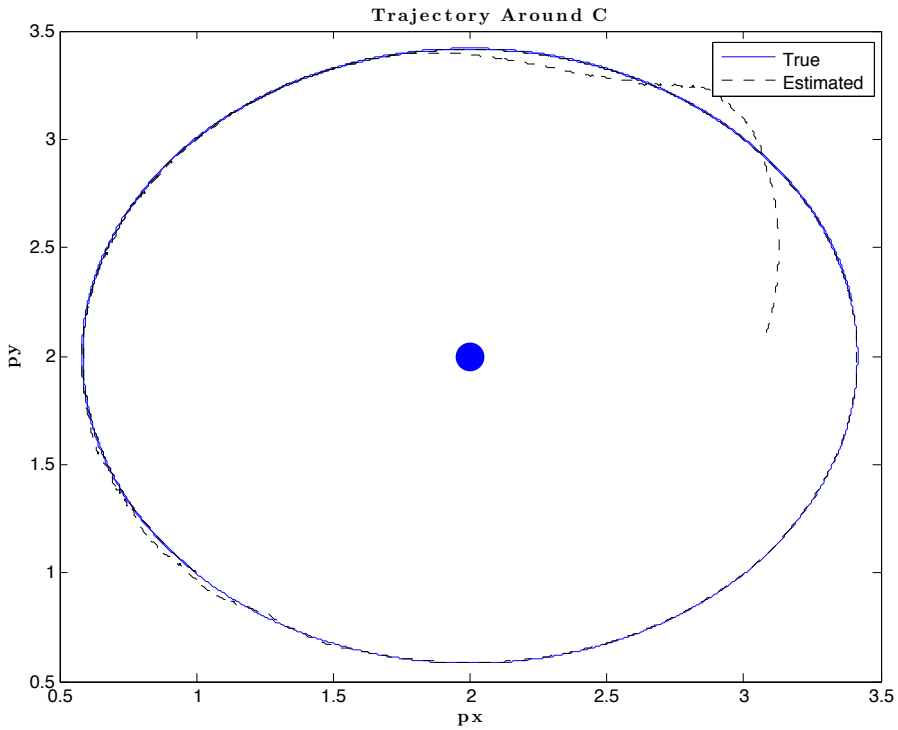

Figure 7: Robot trajectory (blue) and estimate (red) around point $C = [2 \;\; 2]^T$ with perfect IC knowledge.

Extended Kalman Filter

For a discrete time system with dynamics and measurements equations written as:

$$\begin{aligned} x_n &= f_{n-1}(x_{n-1}, u_{n-1}, w_{n-1}) \\ y_n &= h_n(x_n, v_n) \\ w_n &\sim \mathcal{N}(0, Q_n) \\ v_n &\sim \mathcal{N}(0, R_n) \end{aligned}$$

The Extended Kalman Filter Equations are:

Prediction

$$\hat{x}_{n|n-1} = f_{n-1}(\hat{x}_{n-1|n-1}, u_{n-1}, 0) \tag{6}$$
$$P_{n|n-1} = F_{n-1}P_{n-1|n-1}F_{n-1}^T + L_{n-1}Q_nL_{n-1}^T \tag{7}$$

Correction

$$K_n = P_{n|n-1}H_n^T(H_nP_{n|n-1}H_n^T + M_nR_nM_n^T)^{-1} \tag{8}$$
$$P_{n|n} = (I - K_nH_n)P_{n|n-1} \tag{9}$$
$$\hat{x}_{n|n} = \hat{x}_{n|n-1} + K_n[y_n - h_n(\hat{x}_{n|n-1}, 0)] \tag{10}$$

Where

$$F_{n-1} = \frac{\partial f_{n-1}}{\partial x}|_{x_{n-1}=\hat{x}_{n-1|n-1}} \tag{11}$$

$$L_{n-1} = \frac{\partial f_{n-1}}{\partial w}|_{x=\hat{x}_{n-1|n-1}} \tag{12}$$

$$H_n = \frac{\partial h_n}{\partial x}|_{\hat{x}_{n|n-1}} \tag{13}$$

$$M_n = \frac{\partial h_n}{\partial v}|_{\hat{x}_{n|n-1}} \tag{14}$$

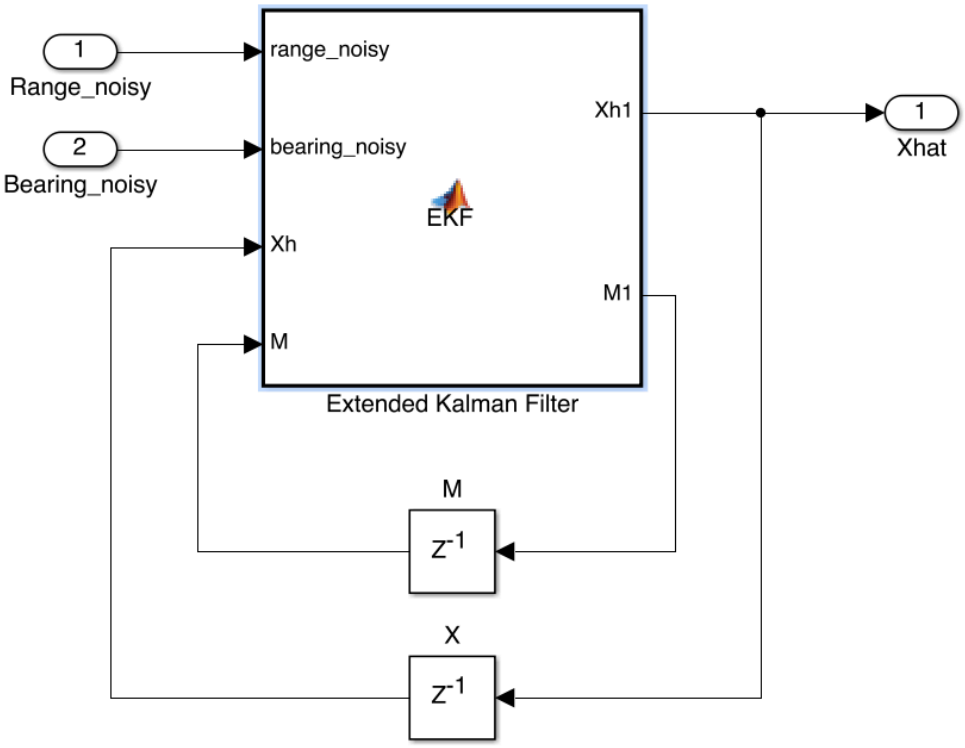

Figure 8: The block diagram for the EKF. The code in the eML block implements the EKF.

For this problem, the messy calculations to find the above Jacobians are implemented in the MATLAB script `P4Lin.m`. The resulting Jacobians are then encoded into the EKF code so that they could be updated recursively as the algorithm is running. These be seen in the `EKF.m` file listing, which also includes the EKF algorithm.

The EKF for this problem relies on linearization in order to propagate the mean and covariance of the state. Propagating the mean and covariance of the state based on Jacobian linearization can be a crude approximation. This is because the Jacobian of the original nonlinear dynamics discards the higher order terms of the nonlinear dynamics.

In addition to this approximation, Kalman Filter theory is abound by Gaussian statistical assumptions. In other words, Kalman Filters of almost all varieties assume that the noise distributions are Gaussi‘therefore they propagate the first and second moments through some version of the dynamics (in this case of the EKF, its the Jacobian of the nonlinear dynamics). This can lead to unpleasant estimation errors if the underlying true statistical distribution has dominant higher order moments (i.e. a great deal of the distribution is characterized by its Skewness or Kurtosis). Since reality is not always Gaussian,this problem is real and must be acknowledged.

For highly nonlinear systems with non-Gaussian noise, more refined techniques can be used to improve the estimator performance. For instance, one may consider a Particle Filter implementation or, if one is familiar with the Unscented Transformation, the UKF may be an option. Note that with the UKF, one is still constrained by the Gaussian assumptions on the noise distributions, but the actual nonlinear dynamics are used to propagate key points through the system dynamics [1].

Despite its shortcomings for this nonlinear/non-Gaussian problem, we have been able to obtain simulation results to discuss. When the initial conditions of the states are known perfectly by the EKF then the covariance propagation matrix initial conditions become zeros(4,4). In this case, depending on the noise covariance intensities, the EKF may track well in this problem for a short period of time. This is shown in the zoomed plots of the Range and the Azimuth (Figure 9 and 10). In those plots, the estimate is shown as a black dotted line and actual true trajectory of the state is shown in blue. The red trace is the noisy observations. On the other hand, if the initial conditions of the underlying state dynamics are not known well then the EKF will not have desirable performance. This situation is shown clearly in Figure 14 and Figure 15, for example.

Listing 2: `EKF.m`

```
function [Xh1,M1] = EKF(range_noisy,bearing_noisy,Xh,M)
Ts = 1/100;
sigWx   = 2e-9;
sigWy   = 1e-9;
sigR    = 1e-2;
sigB    = 1e-2;

% Form Q
Q       = [sigWx 0;0 sigWy];
R       = [sigR 0; 0 sigB];
cx      = 2;
cy      = 2;
px      = Xh(1);
py      = Xh(2);
vx      = Xh(3);
vy      = Xh(4);

%% Compute partials for Update...

%F
a = (cx-px)^2+(cy-py)^2;
b = vx^2+vy^2;
c = b/a;

f11 = 1+c*((cx-px)^2/a -(1/2))*Ts^2;
f12 = c*(1/a)*(cx-px)*(cy-py)*Ts^2;
f13 = (1/a)*vx*(cx-px)*Ts^2;
f14 = (1/a)*vy*(cy-py)*Ts^2;

f21 = c*(1/a)*(cx-px)*(cy-py)*Ts^2;
f22 = 1+(Ts^2*c*((1/a)*(cy-py)^2 - (1/2)));
f23 = (Ts/a)*(vx*(cy-py));
f24 = Ts+((vy/a)*(cy-py))*Ts^2;
```

```
f31 = c*(Ts*(2/a)*(cx-px)^2-Ts);
f32 = (2*Ts*c)*(1/a)*(cy-py)*(cx-px);
f33 = 1+(2*Ts/a)*vx*(cx-px);
f34 = (2*Ts/a)*vy*(cx-px);

f41 = 2*(c/a)*(cx-px)*(cy-py);
f42 = c*((2/a)*(cy-py)^2 - 1)*Ts;
f43 = (2*Ts/a)*vx*(cy-py);
f44 = 1+(2*Ts/a)*(cy-py);

F = [f11 f12 f13 f14;
    f21 f22 f23 f24;
    f31 f32 f33 f34;
    f41 f42 f43 f44];

%L
l11 = (Ts^2)/2;
l12 = 0;
l21 = 0;
l22 = l11;
l31 = Ts;
l32 = 0;
l41 = 0;
l42 = Ts;
L = [l11 l12; l21 l22; l31 l32; l41 l42];

%% Time Update of State & Covariance
P = [px;py];
V = [vx;vy];

X = [P;V];

ax = ((vx^2 + vy^2)*(cx - px))/((cx - px)^2 + (cy - py)^2);
ay = ((vx^2 + vy^2)*(cy - py))/((cx - px)^2 + (cy - py)^2);

alph = [ax;ay];

fup = [P;zeros(2,1)]+[Ts*V;V]+Ts*[zeros(2,1);alph]+((Ts^2)/2)*[alph;zeros(2,1)];
Mup = F*M*F'+L*Q*L';

pxup = fup(1);
pyup = fup(2);
vxup = fup(3);
vyup = fup(4);

%% Compute Partials for Observation
H   = [            0                  0                0 0;
       -pyup/(pxup^2 + pyup^2),pxup/(pxup^2 + pyup^2),0,0];

Mk   = [(pxup^2 + pyup^2)^(1/2), 0;
          0,                    1];
```

```
%% Measurement update
K    = Mup*H'*inv(H*Mup*H'+Mk*R*Mk');
Xh1  = fup+K*([range_noisy;bearing_noisy]-[(pxup^2+pyup^2)^0.5;atan2(pyup,pxup)]);
M1   = (eye(4)-K*H)*Mup;
```

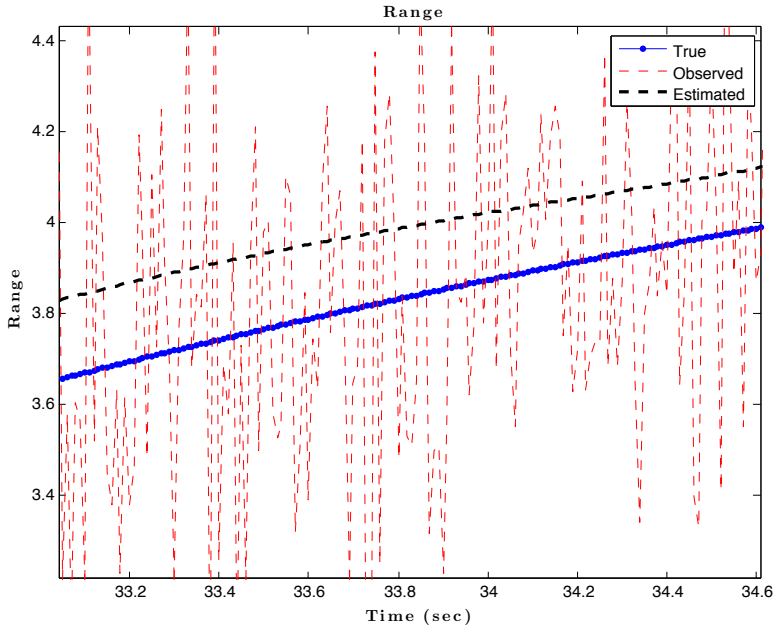

Figure 9: Range Plots with perfect IC knowledge zoomed to show estimates.

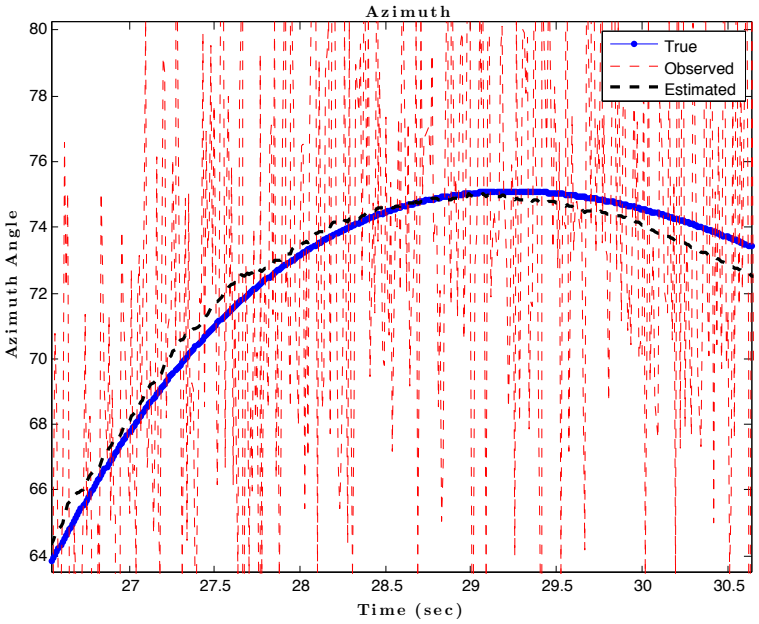

Figure 10: Azimuth estimates zoomed to show EKF accuracy under ideal conditions.

Monte Carlo Simulations and Mean Square Error

Because of the sensitivity of the EKF in this problem to initial conditions and noise intensities, a set of 2500 Monte Carlo simulation runs were performed in order to bound the MSE of the estimator. Usually, the parameters of interest are sampled from a uniform distribution for the Monte Carlo Simulation. For the purpose of this study, however, we will constrain the MC simulations to repeated sampling of the provided distributions and keep the initial conditions fixed for the MC sims.

The MSE of the estimator was obtained and is plotted in Figure 11. Note that there is a bias in the Range and in the Azimuth estimate. This means that the EKF is not an unbiased estimator for this problem. The corresponding histograms for the Range and Azimuth estimates are shown in Figures 12 and 13. The MATLAB code for the MC simulations is given below.

Listing 3: `P4MC.m`

```
%% MC Sims

clear, clc, close all
M = 1000;
eR = zeros(M,1);
eB = zeros(M,1);
mseR = zeros(M,1);
mseB = zeros(M,1);

for k = 1:M+1
    k
    tic
    MIDTERMP4Driver;
    toc
    eR = Rhat-Rnhat(:,2);
    eB = Azhat-Bnhat(:,2);
    mseR(k)  = mean(eR.^2);
    mseB(k)  = mean(eB.^2);
end
```

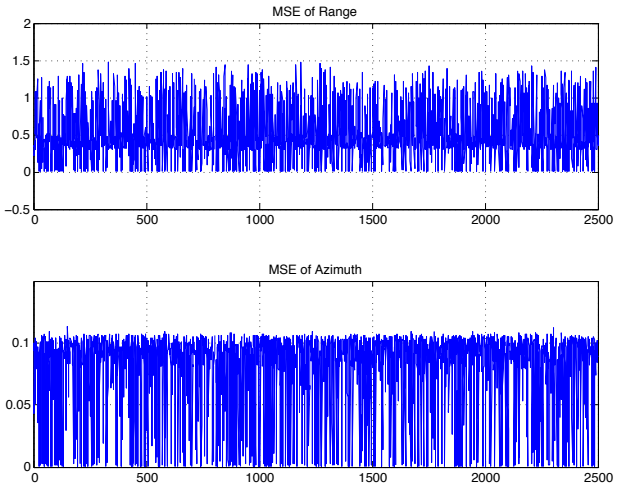

Figure 11: The Mean Square Error of the Range and Azimuth Estimates for the EKF over 2500 Monte Carlo Simulation Runs

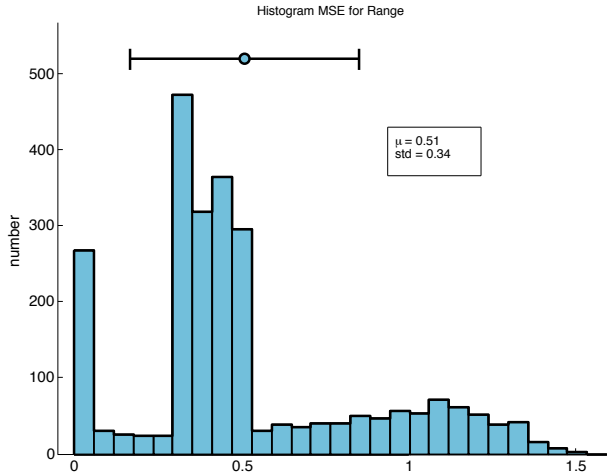

Figure 12: The MSE of the EKF for Range estimates over a set of 2500 Monte Carlo Simulation runs, revealing an estimator bias of 0.51.

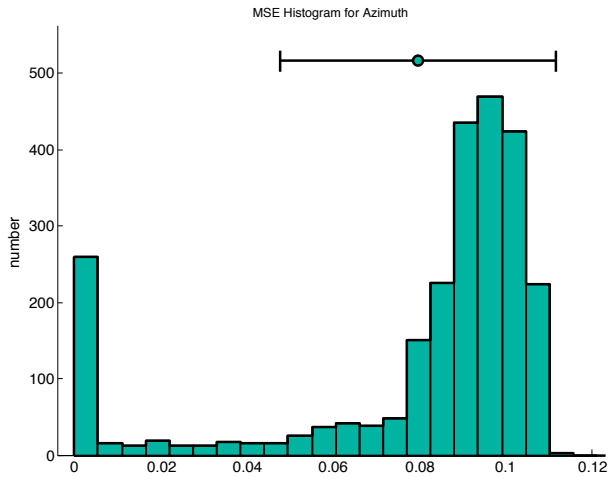

Figure 13: The MSE of the EKF for Azimuth estimates over a set of 2500 Monte Carlo Simulation runs, revealing an estimator bias of 0.08.

References

[1] M. SanjeevArulampalam, Simon Maskell, Neil Gordon, and Tim Clapp *A Tutorial on Particle Filters For Online Nonlinear/Non-Gaussian Bayesian Tracking*. IEEE Transactions on Signal Processing. 2002.

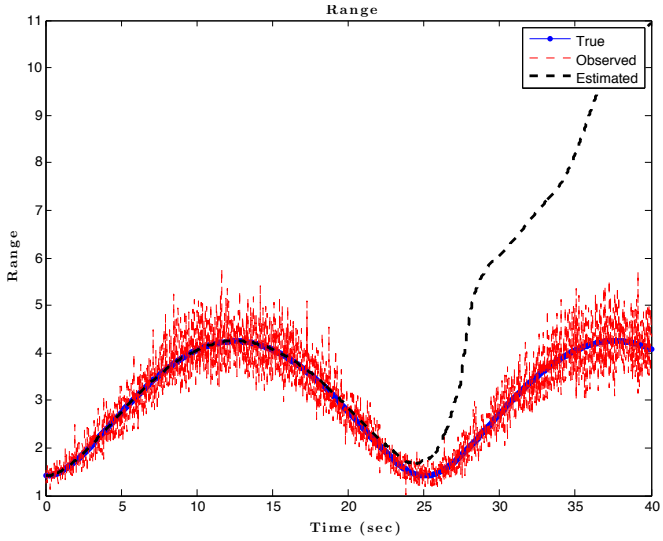

Figure 14: Range estimate with nonzero initial covariance matrix.

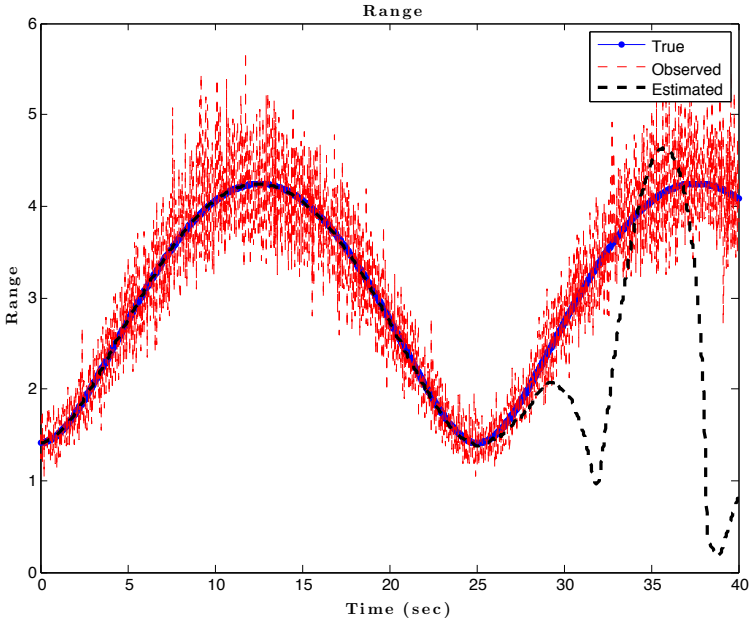

Figure 15: Range estimate with uncertainty in initial conditions.

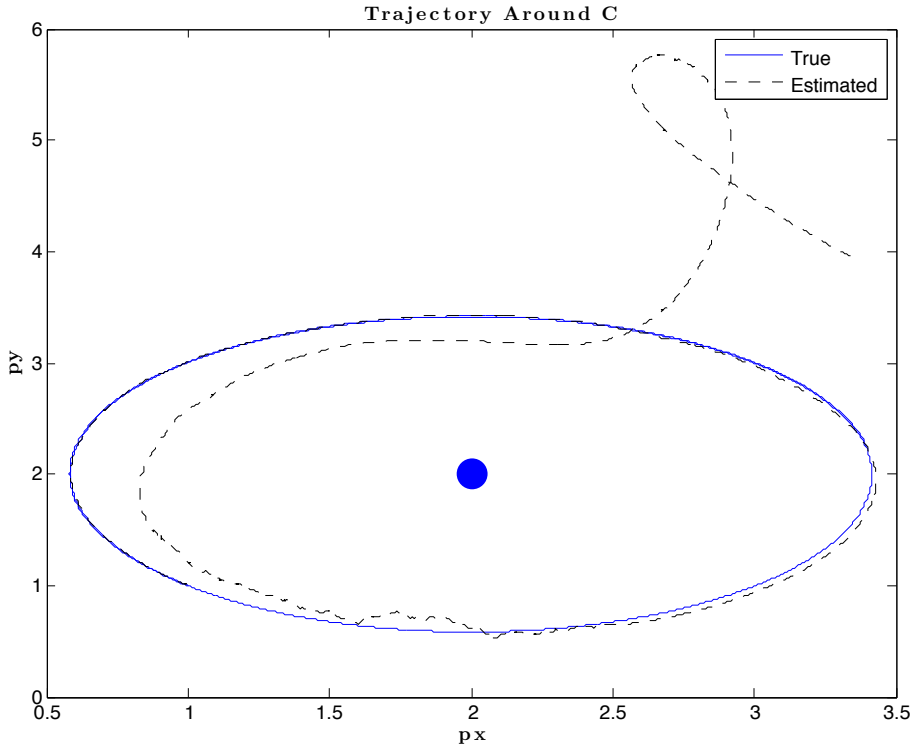

Figure 16: Vehicle Trajectory with uncertainty in initial conditions and nonzero initial covariance matrix.