HW5 Submission

AE-6505 - Kalman Filtering

Kadhir Umasankar

Spring 2022

04-13-22 - HW 5

Wednesday, April 13, 2022 6:15 PM

13.7. We know that when an object reaches terminal velocity, $\dot{x}_2 = 0$

So $\rho_0 e^{(-x_1/k)} \frac{x_2^2}{2x_3} - g = 0 \Rightarrow x_2 = (2g x_3 e^{(x_1/k)}/\rho_0)^{1/2}$

Given altitude = 1mi = 5280 ft = x_1

$k = 22000$

$g = 32.2$

$\rho_0 = 0.0034$

$\dot{x}_3 = \omega_3$ and $E[\omega_3^2] = 0 \therefore x_3 \approx x_{3,0} = 2000$

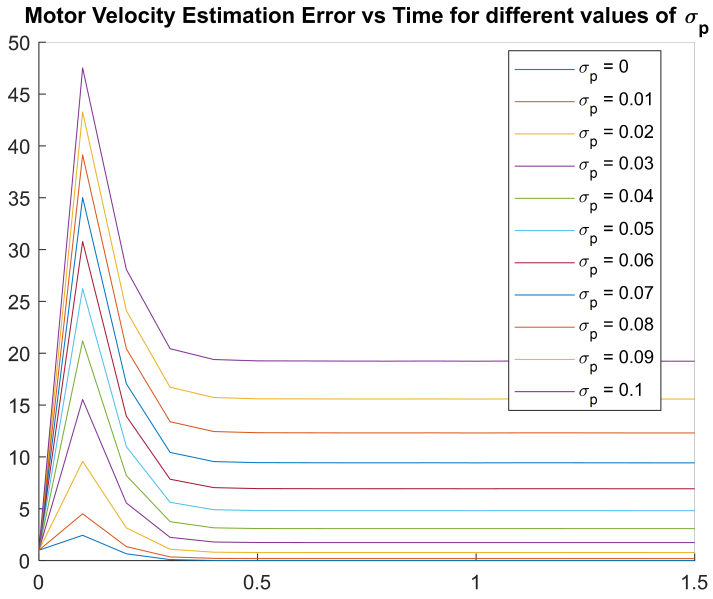
Plugging into previous equation, we get

$x_2 = 6939.590$ ft/s

13.17. Using the constants given in Eg 13.1, a measurement period of 0.1s, measurement noise of 0.1 amps, a hybrid EKF was implemented and and analyzed for control input noise of $\sigma_q = 0 : 0.01 : 0.1$ (code in Appendix)

The following table and plot shows the performance for different values of σ_q

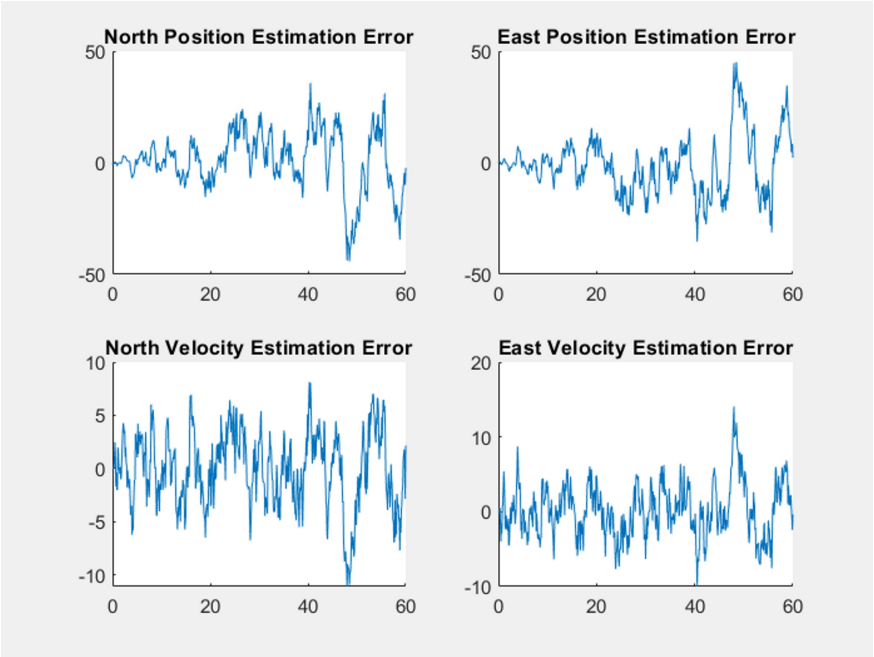
σ_p	Std Dev
0	0.38554
0.01	0.36312
0.02	0.30936
0.03	0.24823
0.04	0.19452
0.05	0.15224
0.06	0.12038
0.07	0.09663
0.08	0.07888
0.09	0.06547
0.1	0.05525



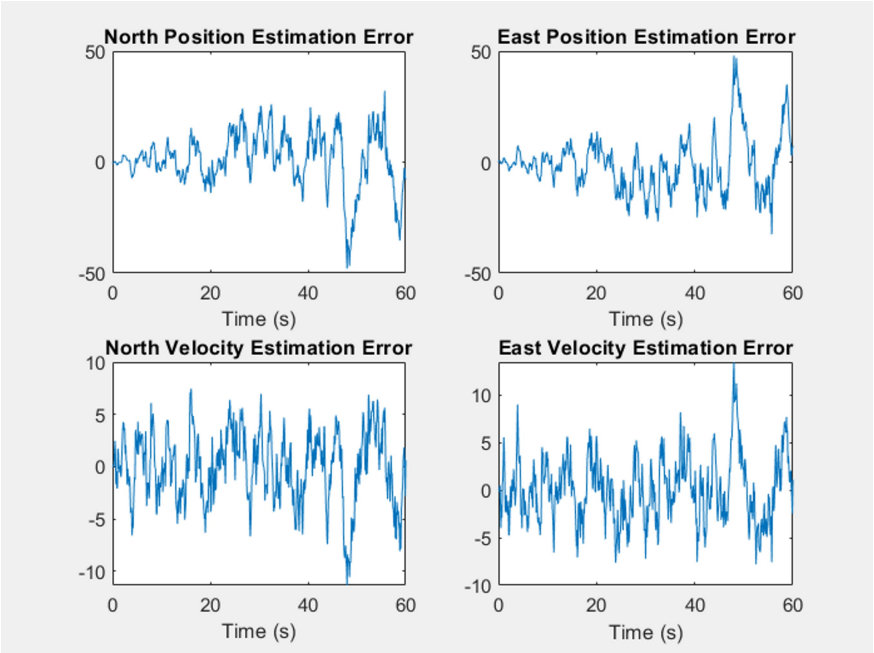
The table shows that the std. dev. of the error decreases as the process noise σ_p increases. The plot shows that the steady state P increases as σ_p increases. Both of these trends are intuitive, since increasing process noise adds a noise floor to P, and this forces the minimum P value to be high, and forces the filter to consider new measurements, which causes the filter to move away from the dynamics and thus increases uncertainty.

14.14. Designed an EKF and UKF (code in Appendix)

EKF results:



UKF results:



To make comparisons more accurate, both filters were initialized with the same seed.

The plots show that the filters do not converge to a steady state value. Position errors are within 50, and velocity errors are within 10. We can also note that there is not a noticeable

difference between the UKF and EKF performance. Since the system is not very nonlinear, the UKF does not give an advantage over EKF.

15.15.a. A particle filter was implemented for the system in Eq 15.1 (code in Appendix) and was analyzed for different values of N. The following was the output from MATLAB:

```
N = 10
Kalman filter RMS error = 17.3186
Particle filter RMS error = 6.6167

N=100
Kalman filter RMS error = 13.1397
Particle filter RMS error = 3.2323

N=1000
Kalman filter RMS error = 13.5754
Particle filter RMS error = 3.0222
```

It can be seen that the particle filter consistently performs better than the EKF. As N increases, the particle filter fits the data better until it converges around 3.

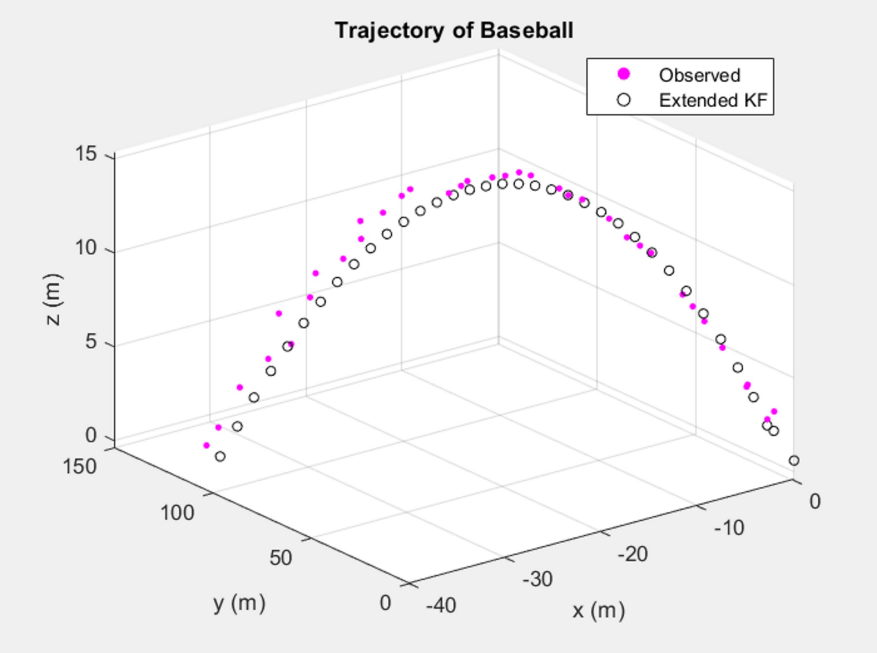
b. The effect of diffrent values of Q was tested:

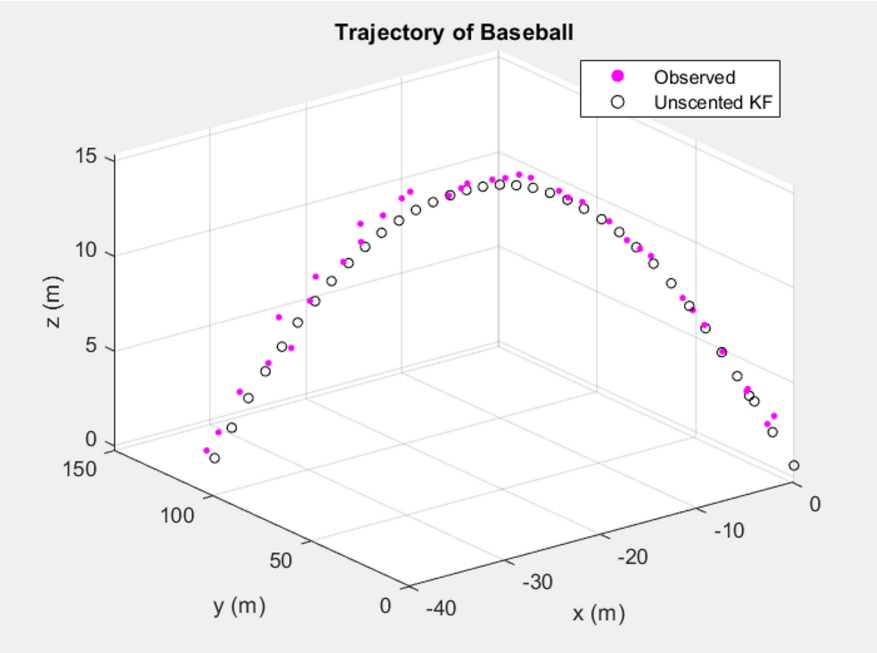
```
Q = 0.1
Kalman filter RMS error = 13.6732
Particle filter RMS error = 1.486

Q = 1
Kalman filter RMS error = 14.6671
Particle filter RMS error = 3.3447

Q = 10
Kalman filter RMS error = 25.1239
Particle filter RMS error = 5.0115
```

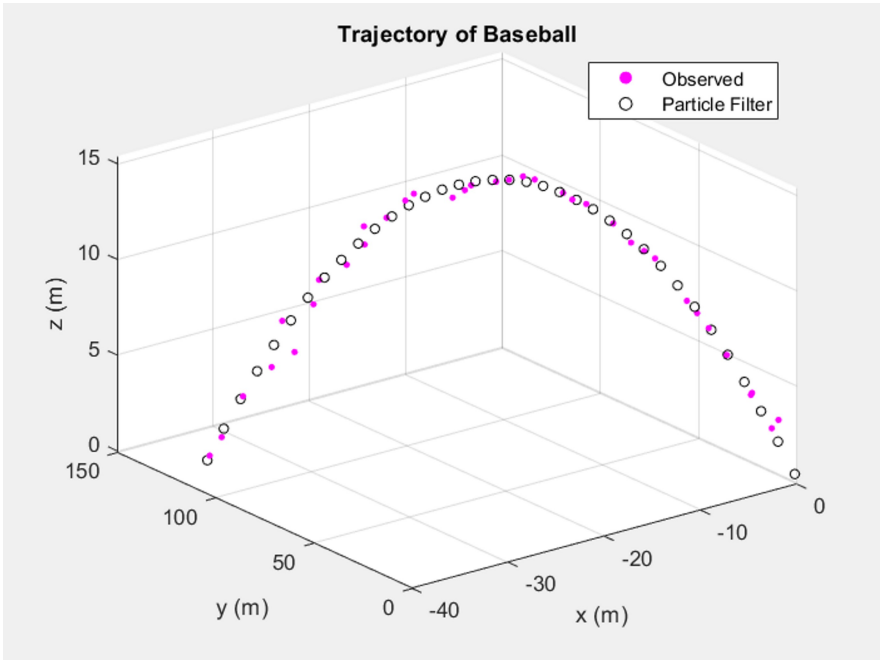
As Q increases, the RMS increases. This is intuitive because

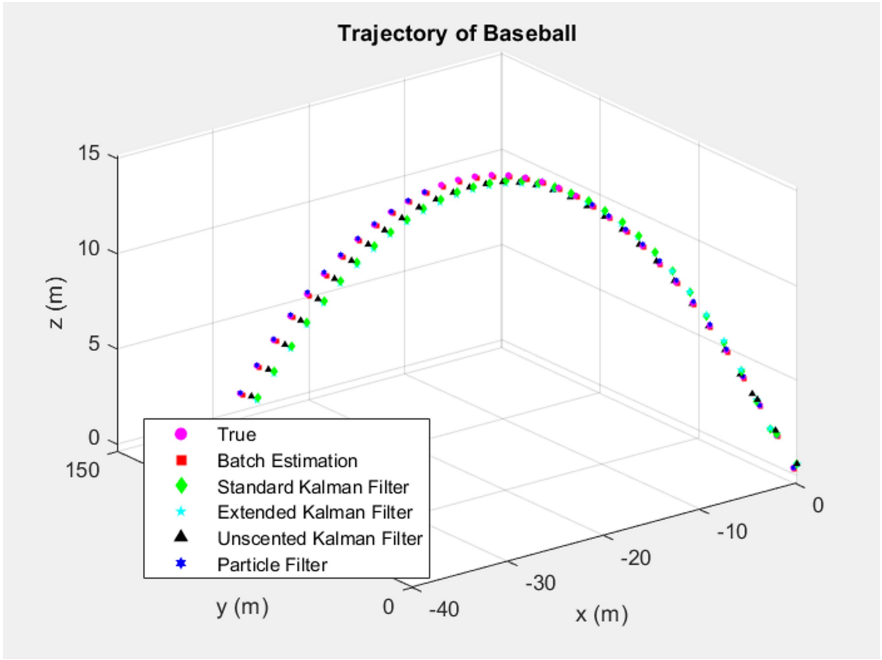
as Q increases, the filter begins to ignore the dynamics more and pays more attention to the measurements. This causes the filter to move away from the true states.

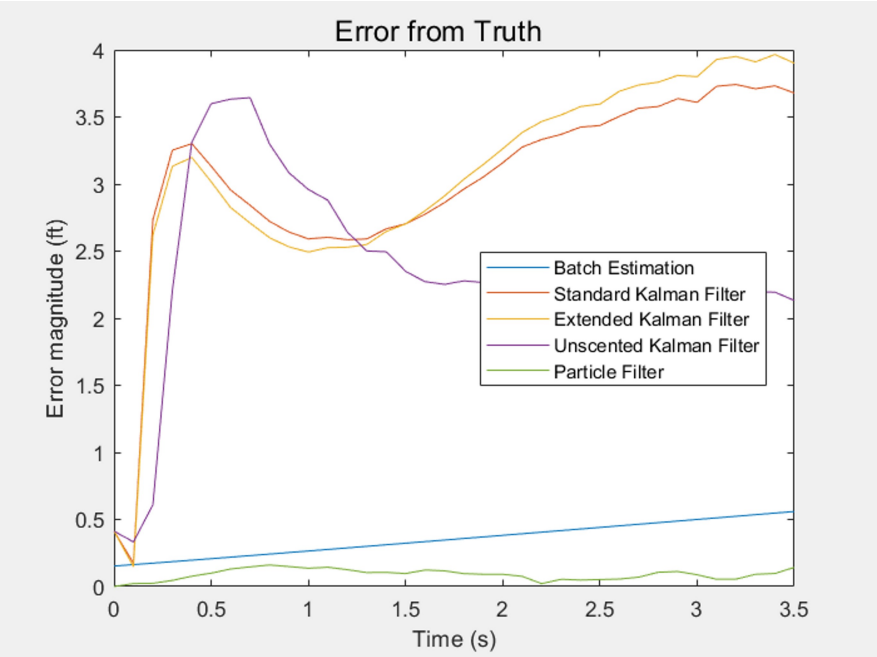
2. Code was written to implement an EKF, a UKF, and a particle filter. All of the filters compared with no added process noise.






Trajectory of Baseball
Observed
Particle Filter
15
10
5
0
z (m)
150
100
50
0
y (m)
-40
-30
-20
-10
0
x (m)


Trajectory of Baseball
True
Batch Estimation
Standard Kalman Filter
Extended Kalman Filter
Unscented Kalman Filter
Particle Filter
15
10
5
0
z (m)
150
y (m)
0
-40
-30
-20
-10
0
x (m)



We can see that the batch filter performs well, as it gets very close to the initial state. However, this approach is not good for real-time systems.

The extended KF does not perform much better than the standard KF, but the UKF performs much better, since it addresses the non-linearity of the system. The particle filter also performs very well.

The std dev of the estimation error of position by the various filters is as follows:

```
Std Dev of Estimation Error of position with batch estimation    0.122937
Std Dev of Estimation Error of position with standard KF         0.77594
Std Dev of Estimation Error of position with EKF                 0.84945
Std Dev of Estimation Error of position with UKF                 0.744295
Std Dev of Estimation Error of position with particle filter     0.040796
```

1 Appendix: MATLAB Code

```
%% q1317.m
clc; clear all; close all;

for q = 0:0.01:0.1
    rng('default')
    rng(12);
    Ra = 1.9; % Winding resistance
    L = 0.003; % Winding inductance
    lambda = 0.1; % Motor constant
    J = 0.00018; % Moment of inertia
    B = 0.001; % Coefficient of viscous friction

    R = diag([0.1^2 0.1^2]);
    ControlNoise = q; % std dev of uncertainty in control inputs
    xdotNoise = [ControlNoise/L ControlNoise/L 0.5 0]; % THIS IS FROM THE EXAMPLE
    Q = diag([xdotNoise(1)^2 xdotNoise(2)^2 xdotNoise(3)^2 xdotNoise(4)^2]);
    P = 1*eye(4); % Initial state estimation covariance
    x = [0; 0; 0; 0]; % initial state
    xhat = [0; 0; 0; 0]; % initial state estimate
    w = 2 * pi; % Control input frequency
    H = [1 0 0 0; 0 1 0 0]; % measurement matrix

    T = 0.1; % measurement time step
    tf = 1.5; % simulation length
    dt = tf / 40000; % time step for integration
    xArray = x;
    xhatArray = xhat;
    Parray = P(3,3);
    for t = T : T : tf
       % Simulate the system.
       for tau = dt : dt : T
          ua0 = sin(w*tau);
          ub0 = cos(w*tau);
          xdot(1,1) = -Ra/L*x(1) + x(3)*lambda/L*sin(x(4)) + ua0/L;
          xdot(2,1) = -Ra/L*x(2) - x(3)*lambda/L*cos(x(4)) + ub0/L;
          xdot(3,1) = -3/2*lambda/J*x(1)*sin(x(4)) + 3/2*lambda/J*x(2)*cos(x(4)) - B/J*x(3);
          xdot(4,1) = x(3);
          x = x + xdot * dt;
       end
       % Simulate the measurement.
       z = H * x + [sqrt(R(1,1)); sqrt(R(2,2))] * randn;
       % Simulate the continuous-time part of the filter.
       for tau = dt : dt : T
          ua0 = sin(w*tau);
          ub0 = cos(w*tau);
          xhatdot(1,1) = -Ra/L*xhat(1) + xhat(3)*lambda/L*sin(xhat(4)) + ua0/L;
          xhatdot(2,1) = -Ra/L*xhat(2) - xhat(3)*lambda/L*cos(xhat(4)) + ub0/L;
          xhatdot(3,1) = -3/2*lambda/J*xhat(1)*sin(xhat(4)) + 3/2*lambda/J*xhat(2)*cos(xhat(4)) - B/J*xhat(3);
          xhatdot(4,1) = xhat(3);
          xhat = xhat + xhatdot * dt;
          F = [-Ra/L 0 lambda/L*sin(xhat(4)) xhat(3)*lambda/L*cos(xhat(4));
               0 -Ra/L -lambda/L*cos(xhat(4)) xhat(3)*lambda/L*sin(xhat(4));
               -3/2*lambda/J*sin(xhat(4)) 3/2*lambda/J*cos(xhat(4)) -B/J
↪  -3/2*lambda/J*(xhat(1)*cos(xhat(4))+xhat(2)*sin(xhat(4)));
               0 0 1 0];
          Pdot = F * P + P * F' + Q;
          P = P + Pdot * dt;
       end
       % Simulate the discrete-time part of the filter.
       K = P * H' * inv(H * P * H' + R);
       xhat = xhat + K * (z - H * xhat);
       P = (eye(4) - K * H) * P * (eye(4) - K * H)' + K * R * K';
```

```
        % Save data for plotting.
        xArray = [xArray x];
        xhatArray = [xhatArray xhat];
        Parray = [Parray P(3,3)];
    end

    % Plot data
%     close all;
    t = 0 : T : tf;

    figure(1)
    hold on
    plot(t, Parray)

    N = size(xArray, 2);
    N2 = round(N / 2);
    N2 = 1;
    xArray = xArray(:,N2:N);
    xhatArray = xhatArray(:,N2:N);
%     wEstErr = sqrt(norm(xArray(3,:)-xhatArray(3,:))^2 / size(xArray,2))
    fprintf("Std Dev of Estimation Error of motor velocity for sigma_p = %G = %G\n", q,
    ↪  std(xArray(3,:)-xhatArray(3,:)))
%     std(xArray(3,:)-xhatArray(3,:))
end
title('Motor Velocity Estimation Error vs Time for different values of \sigma_p')
legend('\sigma_p = 0', '\sigma_p = 0.01', '\sigma_p = 0.02', '\sigma_p = 0.03', '\sigma_p = 0.04', '\sigma_p
    ↪  = 0.05', '\sigma_p = 0.06', '\sigma_p = 0.07', '\sigma_p = 0.08', '\sigma_p = 0.09', '\sigma_p = 0.1',
    ↪  'Location', 'best')

%% q1414_ekf.m
clc; clear all; close all;
rng(12)
store_x = [];
dt = 0.1;
tf = 60;
MeasNoise = 1;
xdotNoise = [0,0,4,4];
Q = diag(xdotNoise);
P = diag([1 1 1 1]); % IF PERFECTLY KNOWN IS IT ZERO COV OR 1 COV
R = diag([1 1]);

x_obs(:, 1) = [0; 0; 50; 50];
N1 = 20;
E1 = 0;
N2 = 0;
E2 = 20;
y_obs(:, 1) = [sqrt((x_obs(1,1)-N1)^2 + (x_obs(2,1)-E1)^2);
               sqrt((x_obs(1,1)-N2)^2 + (x_obs(2,1)-E2)^2)];
A = [1 0 dt 0;
     0 1 0 dt;
     0 0 1 0;
     0 0 0 1];

x_obs = [0; 0; 50; 50]; % initial state
xhat = [0; 0; 50; 50]; % initial state estimate

T = 0.1; % measurement time step
tf = 60; % simulation length
dt = tf / 40000; % time step for integration
xArray = x_obs;
xhatArray = xhat;
for t = T : T : tf
   % Simulate the system.
   A = [1 0 T 0;
        0 1 0 T;
```

```
          0 0 1 0;
          0 0 0 1];
    x_obs = A*x_obs;

    % Simulate the measurement.
    y_obs = [sqrt((x_obs(1)-N1)^2 + (x_obs(2)-E1)^2);
             sqrt((x_obs(1)-N2)^2 + (x_obs(2)-E2)^2)] + sqrt(diag(R)).*randn(2,1);
    % Simulate the continuous-time part of the filter.
    A = [1 0 T 0;
         0 1 0 T;
         0 0 1 0;
         0 0 0 1];
    xhat = A*xhat;
    P = A*P*A' + Q;

    x1 = xhat(1);
    x2 = xhat(2);
    H = [-(N1 - x1)/((E1 - x2)^2 + (N1 - x1)^2)^(1/2) -(E1 - x2)/((E1 - x2)^2 + (N1 - x1)^2)^(1/2) 0 0;
         -(N2 - x1)/((E2 - x2)^2 + (N2 - x1)^2)^(1/2) -(E2 - x2)/((E2 - x2)^2 + (N2 - x1)^2)^(1/2) 0 0];
    y_comp = [sqrt((x1-N1)^2 + (x2-E1)^2);
              sqrt((x1-N2)^2 + (x2-E2)^2)];
    K = P * H' * inv(H * P * H' + R);
    xhat = xhat + K * (y_obs - y_comp);
    P = (eye(4) - K * H) * P;
    % Save data for plotting.
    xArray = [xArray x_obs];
    xhatArray = [xhatArray xhat];
%    Parray = [Parray P(3,3)];
end

% Plot data
%     close all;
t = 0 : T : tf;

err = xArray - xhatArray;

figure(1)
subplot(2,2,1)
hold on
plot(t, err(1,:))
title('North Position Estimation Error')
subplot(2,2,2)
hold on
plot(t, err(2,:))
title('East Position Estimation Error')
subplot(2,2,3)
hold on
plot(t, err(3,:))
title('North Velocity Estimation Error')
subplot(2,2,4)
hold on
plot(t, err(4,:))
title('East Velocity Estimation Error')

figure(2)
hold on
plot(t, xArray(1,:))
plot(t, xhatArray(1,:))
legend('Obs', 'Comp')

%% q1414.m
clc; clear all; close all;
rng(12)
tf = 60; % Simulation length
dt = 0.1; % Simulation step size
```

```
x_obs = [0 0 50 50]'; % Initial state
n = length(x_obs);
xhat = x_obs;
xhat_ukf = xhat;

% Tracking station location
N1 = 20;
E1 = 0;
N2 = 0;
E2 = 20;

P = eye(4);
Pukf = P;
R = diag([1,1]);
Q = diag([0, 0, 4, 4]);
W = ones((2*n),1) / (2*n);

A = [1 0 dt 0;
     0 1 0 dt;
     0 0 1 0;
     0 0 0 1];

x_obs_list = x_obs;
xhatukf_list= xhat_ukf;
Pukf_list = diag(Pukf);

for t = dt : dt : tf
    % Simulate observations
    x_obs = A*x_obs;
    x_obs_list = [x_obs_list x_obs];
    x1 = x_obs(1);
    x2 = x_obs(2);
    H = [-(N1 - x1)/((E1 - x2)^2 + (N1 - x1)^2)^(1/2) -(E1 - x2)/((E1 - x2)^2 + (N1 - x1)^2)^(1/2) 0 0;
         -(N2 - x1)/((E2 - x2)^2 + (N2 - x1)^2)^(1/2) -(E2 - x2)/((E2 - x2)^2 + (N2 - x1)^2)^(1/2) 0 0];
    z_obs = H * x_obs + sqrt(diag(R)).*randn(2,1);

    % Generate the UKF sigma points.
    [root,p] = chol(n*Pukf);
    for i = 1 : n
       xhat_k_1(:,i) = xhat_ukf + root(i,:)';
       xhat_k_1(:,i+n) = xhat_ukf - root(i,:)';
    end
    % Time update
    for i = 1 : (2*n)
          xhat_k_1(:,i) = A*xhat_k_1(:,i);
    end
    xhat_ukf = zeros(n,1);
    for i = 1 : (2*n)
       xhat_ukf = xhat_ukf + W(i) * xhat_k_1(:,i);
    end
    Pukf = zeros(n,n);
    for i = 1 : (2*n)
       Pukf = Pukf + W(i) * (xhat_k_1(:,i) - xhat_ukf) * (xhat_k_1(:,i) - xhat_ukf)';
    end
    Pukf = Pukf + Q;

    % Measurement update
    for i = 1 : (2*n)
      zukf(:,i) = H*xhat_k_1(:,i);
    end
    zhat = 0;
    for i = 1 : (2*n)
      zhat = zhat + W(i) * zukf(:,i);
    end
    Py = 0;
```

```
    Pxy = zeros(n,1);
    for i = 1 : (2*n)
      Py = Py + W(i) * (zukf(:,i) - zhat) * (zukf(:,i) - zhat)';
      Pxy = Pxy + W(i) * (xhat_k_1(:,i) - xhat_ukf) * (zukf(:,i) - zhat)';
    end
    Py = Py + R;
    Kukf = Pxy * inv(Py);
    xhat_ukf = xhat_ukf + Kukf * (z_obs - zhat);
    Pukf = Pukf - Kukf * Py * Kukf';
    xhatukf_list = [xhatukf_list xhat_ukf];
    Pukf_list = [Pukf_list diag(Pukf)];

end

% visualize
x_err = x_obs_list - xhatukf_list;
t = 0 : dt : tf;
figure()
subplot(2,2,1)
plot(t,x_err(1,:))
xlabel('Time (s)');
title('North Position Estimation Error');
subplot(2,2,2)
plot(t,x_err(2,:))
xlabel('Time (s)');
title('East Position Estimation Error');
subplot(2,2,3)
plot(t,x_err(3,:))
xlabel('Time (s)');
title('North Velocity Estimation Error');
subplot(2,2,4)
plot(t,x_err(4,:))
xlabel('Time (s)');
title('East Velocity Estimation Error');

%% q1515.m
clc; clear all; close all;
counter = 0;
ekfRMS = [];
pfRMS = [];
while counter < 100
    % Particle filter example, adapted from Gordon, Salmond, and Smith paper.

    x = 0.1; % initial state
    Q = 10; % process noise covariance
    R = 1; % measurement noise covariance
    tf = 50; % simulation length

    N = 100; % number of particles in the particle filter

    xhat = x;
    P = 2;
    xhatPart = x;

    % Initialize the particle filter.
    for i = 1 : N
        xpart(i) = x + sqrt(P) * randn;
    end

    xArr = [x];
    yArr = [x^2 / 20 + sqrt(R) * randn];
    xhatArr = [x];
    PArr = [P];
    xhatPartArr = [xhatPart];

```

```
    close all;

    for k = 1 : tf
        % System simulation
        x = 0.5 * x + 25 * x / (1 + x^2) + 8 * cos(1.2*(k-1)) + sqrt(Q) * randn;
        y = x^2 / 20 + sqrt(R) * randn;
        % Extended Kalman filter
        F = 0.5 + 25 * (1 - xhat^2) / (1 + xhat^2)^2;
        P = F * P * F' + Q;
        H = xhat / 10;
        K = P * H' * (H * P * H' + R)^(-1);
        xhat = 0.5 * xhat + 25 * xhat / (1 + xhat^2) + 8 * cos(1.2*(k-1));
        xhat = xhat + K * (y - xhat^2 / 20);
        P = (1 - K * H) * P;
        % Particle filter
        for i = 1 : N
            xpartminus(i) = 0.5 * xpart(i) + 25 * xpart(i) / (1 + xpart(i)^2) + 8 * cos(1.2*(k-1)) + sqrt(Q)
↪   * randn;
            ypart = xpartminus(i)^2 / 20;
            vhat = y - ypart;
            q(i) = (1 / sqrt(R) / sqrt(2*pi)) * exp(-vhat^2 / 2 / R);
        end
        % Normalize the likelihood of each a priori estimate.
        qsum = sum(q);
        for i = 1 : N
            q(i) = q(i) / qsum;
        end
        % Resample.
        for i = 1 : N
            u = rand; % uniform random number between 0 and 1
            qtempsum = 0;
            for j = 1 : N
                qtempsum = qtempsum + q(j);
                if qtempsum >= u
                    xpart(i) = xpartminus(j);
                    break;
                end
            end
        end
        % The particle filter estimate is the mean of the particles.
        xhatPart = mean(xpart);
        % Plot the estimated pdf's at a specific time.
%         if k == 20
%             % Particle filter pdf
%             pdf = zeros(81,1);
%             for m = -40 : 40
%                 for i = 1 : N
%                     if (m <= xpart(i)) && (xpart(i) < m+1)
%                         pdf(m+41) = pdf(m+41) + 1;
%                     end
%                 end
%             end
%             figure;
%             m = -40 : 40;
%             plot(m, pdf / N, 'r');
%             hold;
%             title('Estimated pdf at k=20');
%             disp(['min, max xpart(i) at k = 20: ', num2str(min(xpart)), ', ', num2str(max(xpart))]);
%             % Kalman filter pdf
%             pdf = (1 / sqrt(P) / sqrt(2*pi)) .* exp(-(m - xhat).^2 / 2 / P);
%             plot(m, pdf, 'b');
%             legend('Particle filter', 'Kalman filter');
%         end
        % Save data in arrays for later plotting
        xArr = [xArr x];
```

```
        yArr = [yArr y];
        xhatArr = [xhatArr xhat];
        PArr = [PArr P];
        xhatPartArr = [xhatPartArr xhatPart];
    end
    counter = counter + 1
    xhatRMS = sqrt((norm(xArr - xhatArr))^2 / tf);
    xhatPartRMS = sqrt((norm(xArr - xhatPartArr))^2 / tf);
    ekfRMS = [ekfRMS xhatRMS];
    pfRMS = [pfRMS xhatPartRMS];
end

t = 0 : tf;

%figure;
%plot(t, xArr);
%ylabel('true state');

figure;
plot(t, xArr, 'b.', t, xhatArr, 'k-', t, xhatArr-2*sqrt(PArr), 'r:', t, xhatArr+2*sqrt(PArr), 'r:');
axis([0 tf -40 40]);
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('time step'); ylabel('state');
legend('True state', 'EKF estimate', '95% confidence region');

figure;
plot(t, xArr, 'b.', t, xhatPartArr, 'k-');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('time step'); ylabel('state');
legend('True state', 'Particle filter estimate');


disp(['Kalman filter RMS error = ', num2str(mean(ekfRMS))]);
disp(['Particle filter RMS error = ', num2str(mean(pfRMS))]);

%% extended_kalman_filter.m
clear all; close all;

data = importdata('homerun_data_HW4.txt');
% data(1,:) = [];
tspan = data(:, 1);
rho = data(:, 2)/3.281; % in m
alpha = data(:, 3)*pi/180; % in rad
beta = data(:, 4)*pi/180; % in rad
g = 9.81;
dt = 0.1;
store_x = [];

% Visualizing data
figure(1)
[plotx ploty plotz] = sph2cart(alpha, beta, rho);
scatter3(plotx, ploty, plotz, 10, 'm', 'filled')

% Initial Conditions
x_(:,1) = [0.4921 0.4921 2.0013 -26.2467 114.3051 65.9941]'/3.281;
% x_(:,1) = [-0.6562; 0.9843; 1.4764;  -32.8084; 119.6219; 55.7806]/3.281;
P = diag([4 4 4 0.1 0.1 0.1])/3.281^2;
R = [(1.524)^2 0 0;
     0 (0.1*pi/180)^2 0;
     0 0 (0.1*pi/180)^2];      %the error covariance constant to be used
% Q = diag([0.5^2,0.5^2,0.5^2,0.01^2,0.01^2,0.01^2]);
Q = zeros(6);
M = eye(3); % COMBAK: is M eye(3) correct?

for i =2:length(rho)
```

```
    % Observed
    y_obs(:,i) = [rho(i); alpha(i); beta(i)];% + randn*sigmaw;

    % Propagation of state
    xhatdot = [x_(4,i-1) x_(5,i-1) x_(6,i-1)-g*dt 0 0 -g]';
    x_(:,i) = x_(:,i-1) + xhatdot*dt; % COMBAK: is rectangular integration ok?

    % Propagation of state covariance
    A = [0 0 0 1 0 0;
         0 0 0 0 1 0;
         0 0 0 0 0 1;
         0 0 0 0 0 0;
         0 0 0 0 0 0;
         0 0 0 0 0 0];
    Pdot = A*P + P*A' + Q; % + LQL' COMBAK: I didn't use Q here bc we weren't given one. OK?
    P = P + Pdot*dt;

    dt = 0.1;

    % Assembling y_computed
    X_ = x_(1,end);
    Y_ = x_(2,end);
    Z_ = x_(3,end);
    rho_ = sqrt(X_^2 + Y_^2 + Z_^2);
    alpha_ = atan2(Y_, X_);
    beta_ = atan2(Z_, sqrt(X_^2 + Y_^2));

    y_comp(:,i) = [rho_; alpha_; beta_];

    % Computing H using y_comp
    H = [X_/rho_ Y_/rho_ Z_/rho_ 0 0 0;
         (-Y_/X_^2)/(1 + (Y_/X_)^2) (1/X_)/(1+(Y_/X_)^2) 0 0 0 0;
         ((-X_*Z_)/(X_^2 + Y_^2)^(3/2))/(1+(Z_^2)/(X_^2 + Y_^2)) ((-Y_*Z_)/(X_^2 + Y_^2)^(3/2))/(1 +
↪  (Z_^2)/(X_^2 + Y_^2)) ((1)/(X_^2 + Y_^2)^(1/2))/(1 + (Z_^2)/(X_^2 + Y_^2)) 0 0 0];
    % Kalman gain
    K = P*H'*inv(H*P*H'+ R);
    % Measurement update
    x_(:,i) = x_(:,i) + K * (y_obs(:,i) - y_comp(:,i));
    P = (eye(6)-K*H)*P*(eye(6)-K*H)' + K*R*K';

    store_x = [store_x x_(:, i)];

    figure(1)
    hold on
    scatter3(X_, Y_, Z_, 20, 'black')
end

xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')
title('Trajectory of Baseball')
legend('Observed', 'Predicted', 'Location', 'best')
hold off;
writematrix(x_, 'baseball_ekf.csv')

%% unscented_kalman_filter.m
clear all; close all;

data = importdata('homerun_data_HW4.txt');
% data(1,:) = [];
tspan = data(:, 1);
rho = data(:, 2)/3.281; % in m
alpha = data(:, 3)*pi/180; % in rad
beta = data(:, 4)*pi/180; % in rad
g = 9.81;
```

```
dt = 0.1;
store_x = [];

% Visualizing data
figure(1)
[plotx ploty plotz] = sph2cart(alpha, beta, rho);
scatter3(plotx, ploty, plotz, 10, 'm', 'filled')

% Initial Conditions
x_(:,1) = [0.4921 0.4921 2.0013 -26.2467 114.3051 65.9941]'/3.281;
% x_(:,1) = [-0.6562; 0.9843; 1.4764;  -32.8084; 119.6219; 55.7806]/3.281;
n = size(x_, 1);
P = diag([4 4 4 0.1 0.1 0.1])/3.281;
R = [(1.524)^2 0 0;
     0 (0.1*pi/180)^2 0;
     0 0 (0.1*pi/180)^2];      %the error covariance constant to be used
% Q = diag([0.5^2,0.5^2,0.5^2,0.01^2,0.01^2,0.01^2]);
Q = zeros(6);
M = eye(3); % COMBAK: is M eye(3) correct?
W = ones(2*n,1) / (2*n); % UKF weights

for i =2:length(rho)
    % Observed
    y_obs(:,i) = [rho(i); alpha(i); beta(i)];% + randn*sigmaw;

    xhat_k_1 = [];
    % Sigma points
    for j = 1:2*n
        if j<=n
            xtilda = chol(n*P);
            xtilda = xtilda(j,:)';
        else
            xtilda = -chol(n*P);
            xtilda = xtilda(j-n,:)';
        end
        xhat_k_1 = [xhat_k_1 x_(:,i-1)+xtilda];
    end

    % Propagation of state
    xhat_k = [];
    for j = 1:2*n
        current_x = xhat_k_1(:,j);
        current_xhatdot = [current_x(4) current_x(5) current_x(6)-g*dt 0 0 -g]';
        xhat_k = [xhat_k current_x + current_xhatdot*dt]; % COMBAK: is rectangular integration ok?
    end
    xhatk_ = sum(xhat_k, 2)/(2*n);
    temp_Pk_ = 0;
    for j = 1:2*n
        temp_Pk_ = temp_Pk_ + (xhat_k(:,j) - xhatk_)*(xhat_k(:,j) - xhatk_)';
    end
    Pk_ = temp_Pk_/(2*n) + Q;

    xhat_k = [];
    % Sigma points
    for j = 1:2*n
        if j<=n
            xtilda = chol(n*P);
            xtilda = xtilda(j,:)';
        else
            xtilda = -chol(n*P);
            xtilda = xtilda(j-n,:)';
        end
        xhat_k = [xhat_k xhatk_+xtilda];
    end

```

```
    % Assembling y_computed
    y_comp = [];
    for j = 1:2*n
        current_x = xhat_k(:,j);
        X_ = current_x(1,end);
        Y_ = current_x(2,end);
        Z_ = current_x(3,end);
        rho_ = sqrt(X_^2 + Y_^2 + Z_^2);
        alpha_ = atan2(Y_, X_);
        beta_ = atan2(Z_, sqrt(X_^2 + Y_^2));

        y_comp = [y_comp [rho_; alpha_; beta_]];
    end
    y_comp_hat = sum(y_comp, 2)/(2*n);

    temp_Py = 0;
    temp_Pxy = 0;
    for j = 1:2*n
        temp_Py = temp_Py + (y_comp(:,j) - y_comp_hat)*(y_comp(:,j) - y_comp_hat)';
        temp_Pxy = temp_Pxy + (xhat_k(:,j)-xhatk_)*(y_comp(:,j)-y_comp_hat)';
    end
    Py = temp_Py/(2*n) + R; % Add Q for noise floor + Q;
    Pxy = temp_Pxy/(2*n);

    K = Pxy*inv(Py);
    x_(:,i) = xhatk_ + K*(y_obs(:,i) - y_comp_hat);
    P = Pk_ - K*Py*K';

    store_x = [store_x x_(:, i)];

    figure(1)
    hold on
    scatter3(x_(1,i), x_(2,i), x_(3,i), 20, 'black')
end

xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')
title('Trajectory of Baseball')
legend('Observed', 'Predicted', 'Location', 'best')
hold off;

writematrix(x_, 'baseball_ukf.csv')

%% particle_filter.m
clc; clear all; close all;

data = importdata('homerun_data_HW4.txt');
% data(1,:) = [];
tspan = data(:, 1);
rho = data(:, 2)/3.281; % in m
alpha = data(:, 3)*pi/180; % in rad
beta = data(:, 4)*pi/180; % in rad
g = 9.81;
dt = 0.1;
store_x = [];

% Visualizing data
figure(1)
[plotx ploty plotz] = sph2cart(alpha, beta, rho);
scatter3(plotx, ploty, plotz, 10, 'm', 'filled')

% Initial Conditions
% x_(:,1) = [0.4921 0.4921 2.0013 -26.2467 114.3051 65.9941]'/3.281;
x_(:,1) = [-0.6562; 0.9843; 1.4764;  -32.8084; 119.6219; 55.7806]/3.281;
```

```
% x_(:,1) = zeros(6,1);% Given initial conditions
P = diag([4 4 4 0.1 0.1 0.1])/3.281^2;
R = [(1.524)^2 0 0;
     0 (0.1*pi/180)^2 0;
     0 0 (0.1*pi/180)^2];      %the error covariance constant to be used
% Q = diag([0.5^2,0.5^2,0.5^2,0.01^2,0.01^2,0.01^2]);
Q = zeros(6);
% Q = eye(6);

N = 1000;

xpartArray = [];
PpartArray = [];

Phi = [1 0 0 dt 0 0;
       0 1 0 0 dt 0;
       0 0 1 0 0 dt;
       0 0 0 1 0 0;
       0 0 0 0 1 0;
       0 0 0 0 0 1];

for i = 2:length(rho)

    for j = 1 : N
        xpart(:,j) = x_(:,i-1) + sqrt(diag(P)) .* randn(size(diag(P)));
        xpartminus(:,j) = Phi*xpart(:,j) + [0; 0; -dt^2/2; 0; 0; -dt]*g + sqrt(diag(Q)).* randn(6,1);

        X_ = xpartminus(1,j);
        Y_ = xpartminus(2,j);
        Z_ = xpartminus(3,j);
        rho_    = sqrt(X_^2 + Y_^2 + Z_^2);
        alpha_  = atan2(Y_,X_);
        beta_   = atan2( Z_, sqrt(X_^2+Y_^2) );
        y_comp = [rho_; alpha_; beta_];
        y_comp = y_comp + sqrt(diag(R)).*randn(size(diag(R)));

        vhat = [rho(i); alpha(i); beta(i)] - y_comp;
        q1(j) = (1./sqrt(2*pi*R(1,1)))*(exp(-vhat(1)^2/(2*R(1,1))));
        q2(j) = (1./sqrt(2*pi*R(2,2)))*(exp(-vhat(2)^2/(2*R(2,2))));
        q3(j) = (1./sqrt(2*pi*R(3,3)))*(exp(-vhat(3)^2/(2*R(3,3))));
%         q1(j) = (1 / sqrt(R(1,1)) / sqrt(2*pi)) * exp(-vhat(1)^2*R(1,1)^-1/2);
%         q2(j) = (1 / sqrt(R(2,2)) / sqrt(2*pi)) * exp(-vhat(2)^2*R(2,2)^-1/2);
%         q3(j) = (1 / sqrt(R(3,3)) / sqrt(2*pi)) * exp(-vhat(3)^2*R(3,3)^-1/2);
    end

    qsum1 = sum(q1);
    qsum2 = sum(q2);
    qsum3 = sum(q3);
    tolerance = 1e-99;
    if qsum1 > tolerance
        q1 = q1./qsum1;
    end
    if qsum2 > tolerance
        q2 = q2./qsum2;
    end
    if qsum3 > tolerance
        q3 = q3./qsum3;
    end
    q = mean([q1; q2; q3], 1);

    for j = 1 : N
        u = rand;
        qtempsum = 0;
        for k = 1 : N
            qtempsum = qtempsum + q(k);
```

```
            if qtempsum >= u
                resampled_xpart(:,j) = xpartminus(:,j);
                break;
            end
        end
    end

    x_(:,i) = mean(resampled_xpart,2);
    P = diag(var(resampled_xpart,0,2));

    figure(1)
    hold on
    scatter3(x_(1,end), x_(2,end), x_(3,end), 20, 'black')
end

xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')
title('Trajectory of Baseball')
legend('Observed', 'Predicted', 'Location', 'best')
hold off;
writematrix(x_, 'baseball_pf.csv')

%% q2_analysis.m
clc; clear all; close all;

data = importdata('homerun_data_HW4.txt');
% data(1,:) = [];
tspan = data(:, 1);
rho = data(:, 2)/3.281; % in m
alpha = data(:, 3)*pi/180; % in rad
beta = data(:, 4)*pi/180; % in rad
batch_data = importdata('baseball_batch.csv');
kf_data = importdata('baseball_kf.csv');
ekf_data = importdata('baseball_ekf.csv');
ukf_data = importdata('baseball_ukf.csv');
pf_data = importdata('baseball_pf.csv');

true_data = [];
t_all = tspan;
X_0 = [-0.6562; 0.9843; 1.4764;  -32.8084; 119.6219; 55.7806]/3.281;
g = 9.81;
for k=1:length(t_all)
    X0 = X_0(1);
    Y0 = X_0(2);
    Z0 = X_0(3);
    Vx0 = X_0(4);
    Vy0 = X_0(5);
    Vz0 = X_0(6);
    X = X0 + Vx0 * t_all(k);
    Y = Y0 + Vy0 * t_all(k);
    Z = Z0 + Vz0 * t_all(k) - (1/2)*g*t_all(k)^2;
    Vx = Vx0;
    Vy = Vy0;
    Vz = Vz0 - g*t_all(k);
    true_data = [true_data [X; Y; Z; Vx; Vy; Vz]];
end

% Visualizing data
figure(1)
[plotx ploty plotz] = sph2cart(alpha, beta, rho);
scatter3(plotx, ploty, plotz, 10, 'm', 'filled')
hold on
scatter3(ekf_data(1,:),ekf_data(2,:),ekf_data(3,:), 20, 'black')
xlabel('x (m)')
```

```
ylabel('y (m)')
zlabel('z (m)')
title('Trajectory of Baseball')
legend('Observed', 'Extended KF', 'Location', 'best')
hold off;

figure(2)
scatter3(plotx, ploty, plotz, 10, 'm', 'filled')
hold on
scatter3(ukf_data(1,:),ukf_data(2,:),ukf_data(3,:), 20, 'black')
xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')
title('Trajectory of Baseball')
legend('Observed', 'Unscented KF', 'Location', 'best')
hold off;

figure(3)
scatter3(plotx, ploty, plotz, 10, 'm', 'filled')
hold on
scatter3(pf_data(1,:),pf_data(2,:),pf_data(3,:), 20, 'black')
xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')
title('Trajectory of Baseball')
legend('Observed', 'Particle Filter', 'Location', 'best')
hold off;

figure(4)
scatter3(true_data(1,:),true_data(2,:),true_data(3,:), 10, 'm', 'filled')
hold on
scatter3(batch_data(1,:),batch_data(2,:),batch_data(3,:), 10, 's', 'filled', 'red')
scatter3(kf_data(1,:),kf_data(2,:),kf_data(3,:), 10, 'd', 'filled', 'green')
scatter3(ekf_data(1,:),ekf_data(2,:),ekf_data(3,:), 20, 'p', 'filled', 'cyan')
scatter3(ukf_data(1,:),ukf_data(2,:),ukf_data(3,:), 10, '^', 'filled', 'black')
scatter3(pf_data(1,:),pf_data(2,:),pf_data(3,:), 10, 'h', 'filled', 'blue')
xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')
title('Trajectory of Baseball')
legend('True', 'Batch Estimation', 'Standard Kalman Filter', 'Extended Kalman Filter', 'Unscented Kalman
↪  Filter', 'Particle Filter', 'Location', 'best')
hold off;

figure(5)
% subplot(3,1,1)
sgtitle('Error from Truth')
plot(t_all, vecnorm(batch_data(1:3,:)-true_data(1:3,:)))
hold on
plot(t_all, vecnorm(kf_data(1:3,:)-true_data(1:3,:)))
plot(t_all, vecnorm(ekf_data(1:3,:)-true_data(1:3,:)))
plot(t_all, vecnorm(ukf_data(1:3,:)-true_data(1:3,:)))
plot(t_all, vecnorm(pf_data(1:3,:)-true_data(1:3,:)))
ylabel('Error magnitude (ft)')
xlabel('Time (s)')
legend('Batch Estimation', 'Standard Kalman Filter', 'Extended Kalman Filter', 'Unscented Kalman Filter',
↪  'Particle Filter', 'Location', 'best')
hold off

fprintf("Std Dev of Estimation Error of position with batch estimation \t%g\n",
↪  std(vecnorm(batch_data(1:3,:)-true_data(1:3,:))))
fprintf("Std Dev of Estimation Error of position with standard KF \t\t%g\n",
↪  std(vecnorm(kf_data(1:3,:)-true_data(1:3,:))))
fprintf("Std Dev of Estimation Error of position with EKF \t\t\t\t%g\n",
↪  std(vecnorm(ekf_data(1:3,:)-true_data(1:3,:))))
```

```
832 fprintf("Std Dev of Estimation Error of position with UKF \t\t\t\t%g\n",
    ↪ std(vecnorm(ukf_data(1:3,:)-true_data(1:3,:))))
833 fprintf("Std Dev of Estimation Error of position with particle filter \t%g\n",
    ↪ std(vecnorm(pf_data(1:3,:)-true_data(1:3,:))))
```