Tracking a Ballistic Target:

An Application of Extended Kalman Filter

1. Abstract

This project studies the problem of tracking a ballistic object by processing simulated radar measurements. A suitable (nonlinear) model of target motion and the observation error model are developed. The scenario we chose is firing and tracking 800 mm heavy artillery shells. The shell normally weight more than 4000 kg and the muzzle velocity is around 800 m/s. For simplification, we will confine our discussion in 2-D space. As a result, the radar measurements are range and elevation angle. In order to deal with this nonlinear model, an Extended Kalman Filter was designed. In this report, the true trajectory, trajectory with process noise, trajectory with observation noise and the estimated trajectory based on noisy radar signal will be compared.

2. Introduction

The problem of tracking ballistic objects attracted much attention of the researchers for both theoretical and practical reasons. Technically speaking we need to set up a stochastic nonlinear filter due to the nonlinearity of the dynamic state equation of the target; tracking filters have been conceived for this purpose since the early days of the invention of the Kalman filter. More recently, the following papers have been published on this subject. Practical applications are in the fields of surveillance for defense and for safety against the reentry of old satellites. In relation to the second application, it is known that the number of objects orbiting the Earth has been continuously increasing since the launch of the first satellite in 1957. They are old satellites, pieces of satellites due to explosion and erosion, upper stages of missiles, etc. It is relevant to have means to detect, classify, and track these pieces of debris big objects that reenter the atmosphere should be accurately tracked to anticipate their landing points on Earth. The aim of our study is to formulate the nonlinear tracking filtering problem and to discuss the application of several approximate filters. The next section will provide the models of kinematics of the target and of the radar measurements.

3. Description

A Cannon Ball following parabolic path is incorporated with air drag in order to make it realistic. Consider an object which is launched from one point on Earth to another point along a ballistic flight. The kinematics of the ballistic object in the reentry phase is derived under the following hypotheses. The forces acting on the target are gravity, and drag. The effects of centrifugal acceleration, Coriolis acceleration, wind, lift force, and spinning motion are ignored, due to their small effect on the trajectory. Another simplifying assumption is related to flat Earth approximation. Having assumed a flat Earth, an orthogonal coordinate reference system can be used with the following variables (Fig. 1):

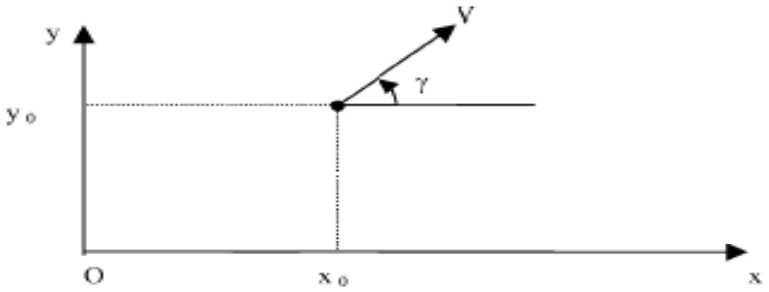


Fig. 1. Coordinate reference system.

x is the abscissa,
y is the ordinate,
x_0 and y_0 are the target coordinates at time t_0,
v is the velocity module,
γ is the angle between the horizontal axis and the direction of motion.

Without an air drag it would be five times longer than the realistic. Here we are not taking into account the parameters like wind, humidity and Coriolis Effect, so we are adding Radar Noise and System Noise to the path.

The Kalman Filtering and Extended Kalman Filtering is used for Non-Linear Equation. The target motion is described by the following discrete-time nonlinear dynamic state equation:

$$\mathbf{s}_{k+1} = \psi_k(\mathbf{s}_k) + \mathbf{G}\begin{bmatrix} 0 \\ -g \end{bmatrix} + \mathbf{w}_k$$

where the state vector is

$$\mathbf{s}_k \stackrel{\Delta}{=} [x_k \quad \dot{x}_k \quad y_k \quad \dot{y}_k]^{\mathrm{T}}$$

and

$$\psi_k(\mathbf{s}_k) = \mathbf{\Phi}\mathbf{s}_k + \mathbf{G}\mathbf{f}_k(\mathbf{s}_k)$$

T is the time interval between the radar measurements. The drag is a force directed in opposition to the target speed and with intensity equal to 12 (g=¯) ½v2 [25]; being: g the

gravity acceleration, ¯1 the ballistic coefficient, ½ the air density (typically it is an exponentially decaying function of height, ½ = c1e^c2y.

The measurements, collected by the radar for target tracking, are the range r and elevation "; the radar is located at xR =0, yR = 0 in all the numerical evaluations here. The error standard deviations of these measurements are denoted as ¾r (for range) and ¾" (for elevation). Radar measurements are transformed to the Cartesian coordinates d = r cos" and h = r sin" (for measured target abscissa and ordinate) so that the measurement equation is linear it is zero-mean white Gaussian with covariance matrix **R**k with elements. Thus, the Observation Error is:

$$\sigma_d^2 = \sigma_r^2 \cos^2(\varepsilon) + r^2 \sigma_\varepsilon^2 \sin^2(\varepsilon)$$
$$\sigma_h^2 = \sigma_r^2 \sin^2(\varepsilon) + r^2 \sigma_\varepsilon^2 \cos^2(\varepsilon)$$
$$\sigma_{dh} = (\sigma_r^2 - r^2 \sigma_\varepsilon^2) \sin(\varepsilon) \cos(\varepsilon).$$

Process noise (System Noise) wk in non linear state equation is modeled as a zero-mean white Gaussian process with nonsingular covariance matrix

$$\mathbf{Q} = q \begin{bmatrix} \theta & \underline{0} \\ \underline{0} & \theta \end{bmatrix}, \qquad \text{with} \quad \theta = \begin{bmatrix} \frac{T^3}{3} & \frac{T^2}{2} \\ \frac{T^2}{2} & T \end{bmatrix}$$

Where q is a parameter related to process noise intensity. The process noise (System Noise) accounts for all forces that have not been considered in the model and possible deviations of the model from the reality.

4. Objective

The objective of this section is to track actual Cannon Ball Trajectory. The optimal estimator for this problem cannot be built and hence we resort to approximate filters. The theoretical lower bound, which defines the best achievable performance, plays an important role in algorithm evaluation and assessment of the level of approximation introduced in the filtering algorithms.

$$\hat{\mathbf{s}}_{k+1|k} = \psi_k(\hat{\mathbf{s}}_{k|k}) + \mathbf{G}\begin{pmatrix} 0 \\ -g \end{pmatrix}$$

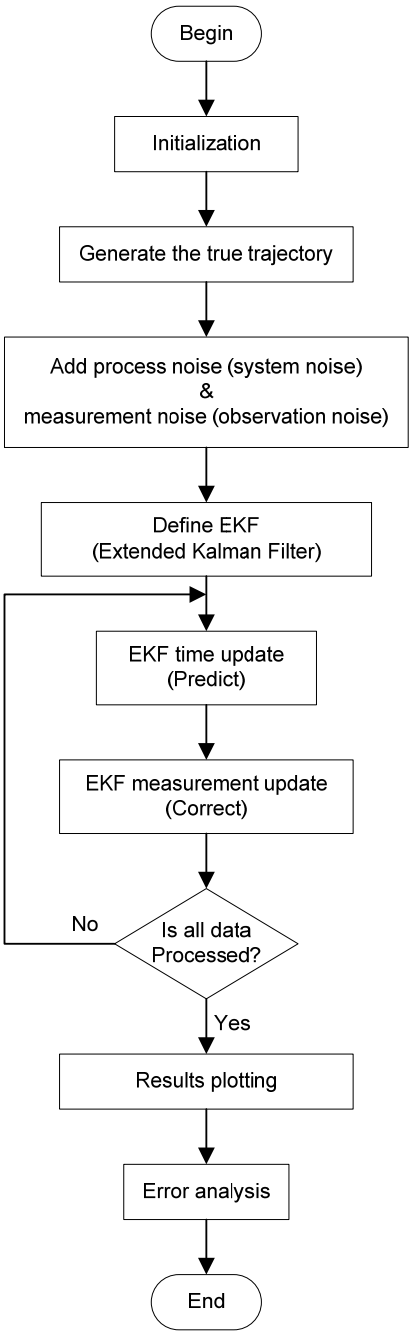
$$\mathbf{P}_{k+1|k} = (\Phi + \mathbf{G}\cdot\mathbf{F}_k)\mathbf{P}_{k|k}(\Phi + \mathbf{G}\cdot\mathbf{F}_k)^T + \mathbf{Q}$$

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1|k}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k+1|k}\mathbf{H}^T + \mathbf{R}_k)^{-1}$$

$$\hat{\mathbf{s}}_{k+1|k+1} = \hat{\mathbf{s}}_{k+1|k} + \mathbf{K}_{k+1}(\mathbf{z}_{k+1} - \mathbf{H}\hat{\mathbf{s}}_{k+1|k})$$

$$\mathbf{P}_{k+1|k+1} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}_{k+1|k}.$$

5. Flowchart


Begin
Initialization
Generate the true trajectory
Add process noise (system noise)
&
measurement noise (observation noise)
Define EKF
(Extended Kalman Filter)
EKF time update
(Predict)
EKF measurement update
(Correct)
No
Is all data
Processed?
Yes
Results plotting
Error analysis
End

6. Error Analysis

Errors are convergent to 0, because of the Non linear Dynamic State Equation. This subsection reports the mean and the standard deviation of the estimation error all performance curves were obtained by averaging over 100 independent Monte Carlo runs. First we present the results obtained using the EKF: the mean of estimation error in target position x and y and velocity vx = _x and vy = _y is shown in Fig. 5. The standard deviation of error is displayed. The Monte Carlo error analysis suggests that the EKF is approximately unbiased with standard deviation just on top of the square root of the CRLB. As usual, the curves are compared with the CRLB. In this case the filter is also approximately unbiased with standard deviation close to the square root of the CRLB.

A technique which has had a great impact in many different fields of computational science is a technique called "Monte Carlo Simulation." This technique derives its name from the casinos in Monte Carlo - a Monte Carlo simulation uses random numbers to model some sort of a process. This technique works particularly well when the process is one where the underlying probabilities are known but the results are more difficult to determine.

7. Simulation Results

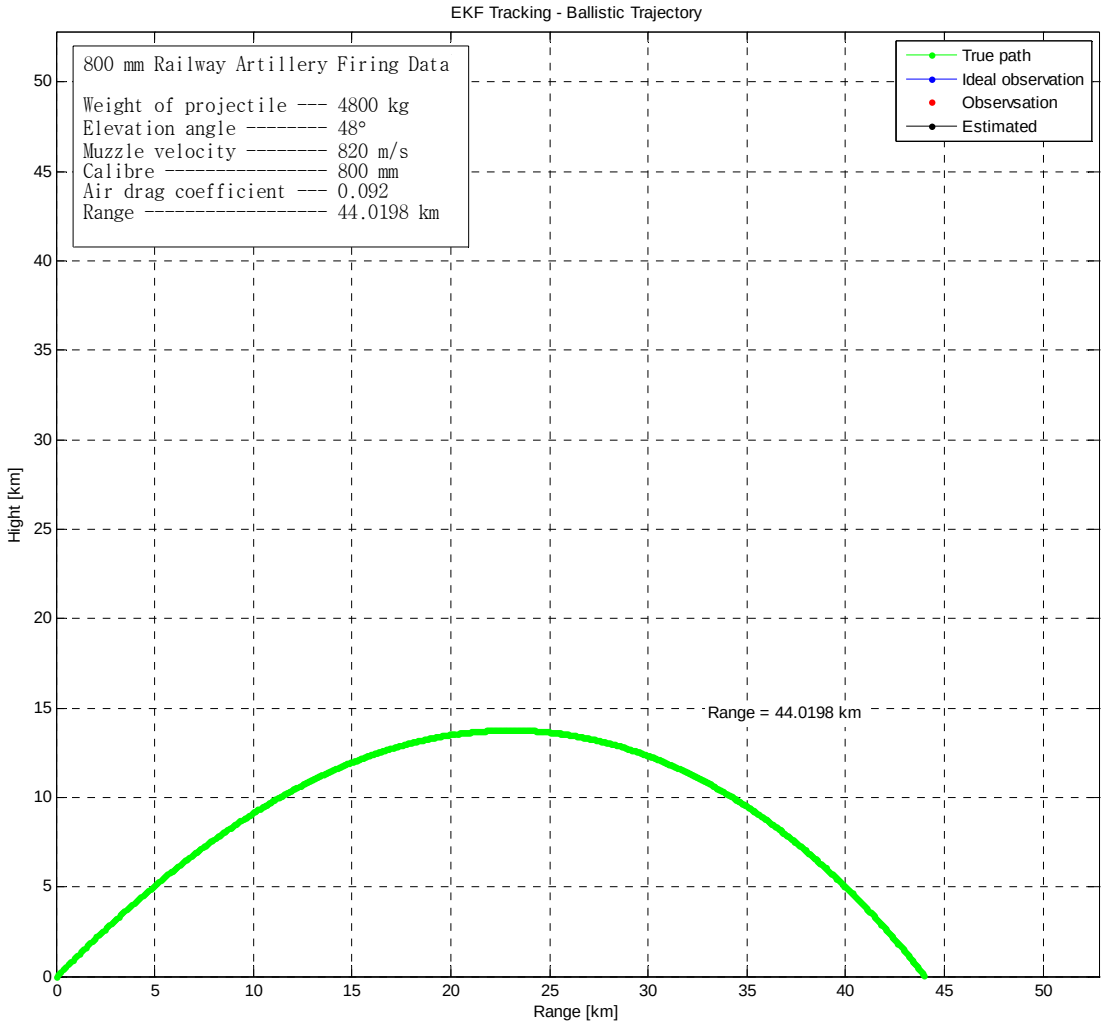


Figure 1. True (ideal) Trajectory – (green curve)

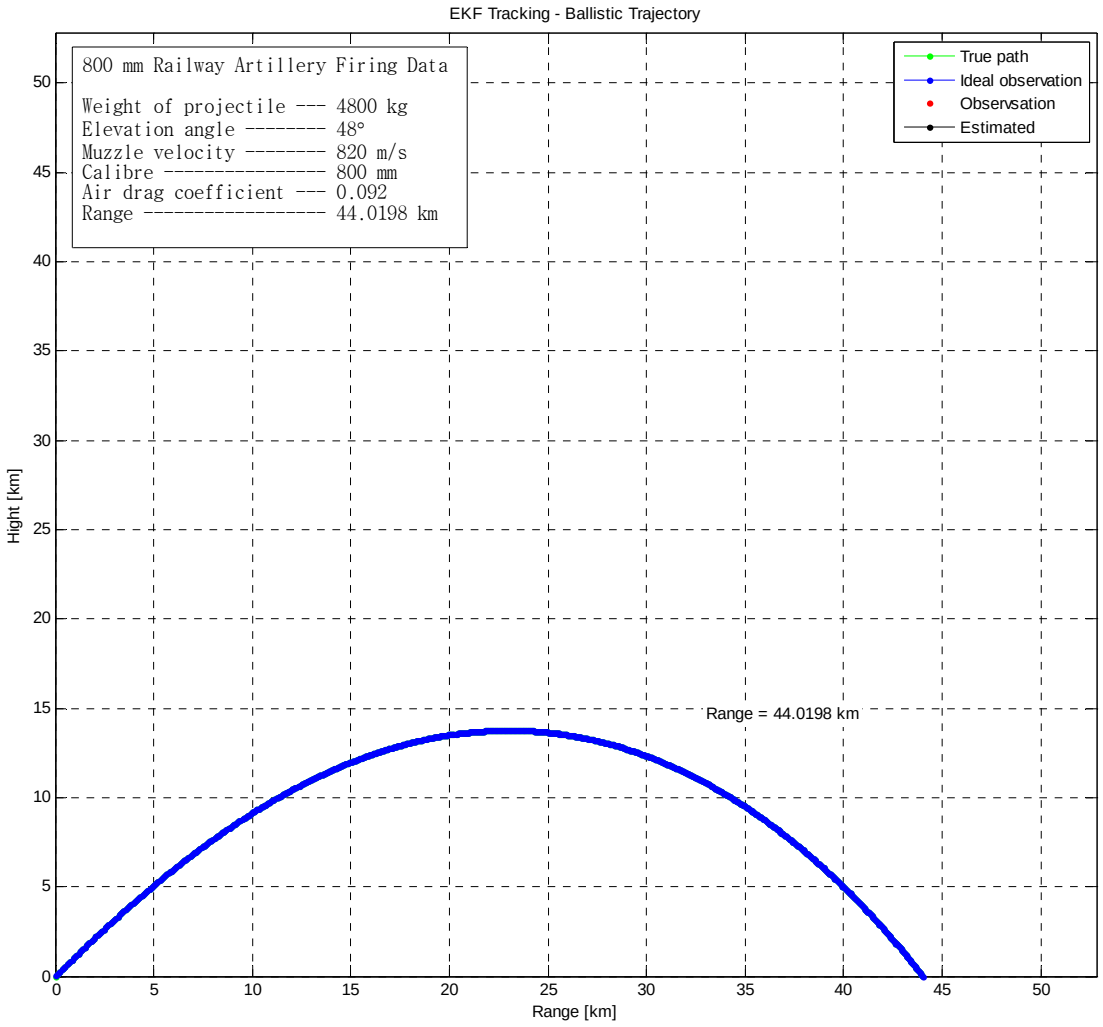


Figure 2. Ideal observation Trajectory (with process noise only) – (blue curve)

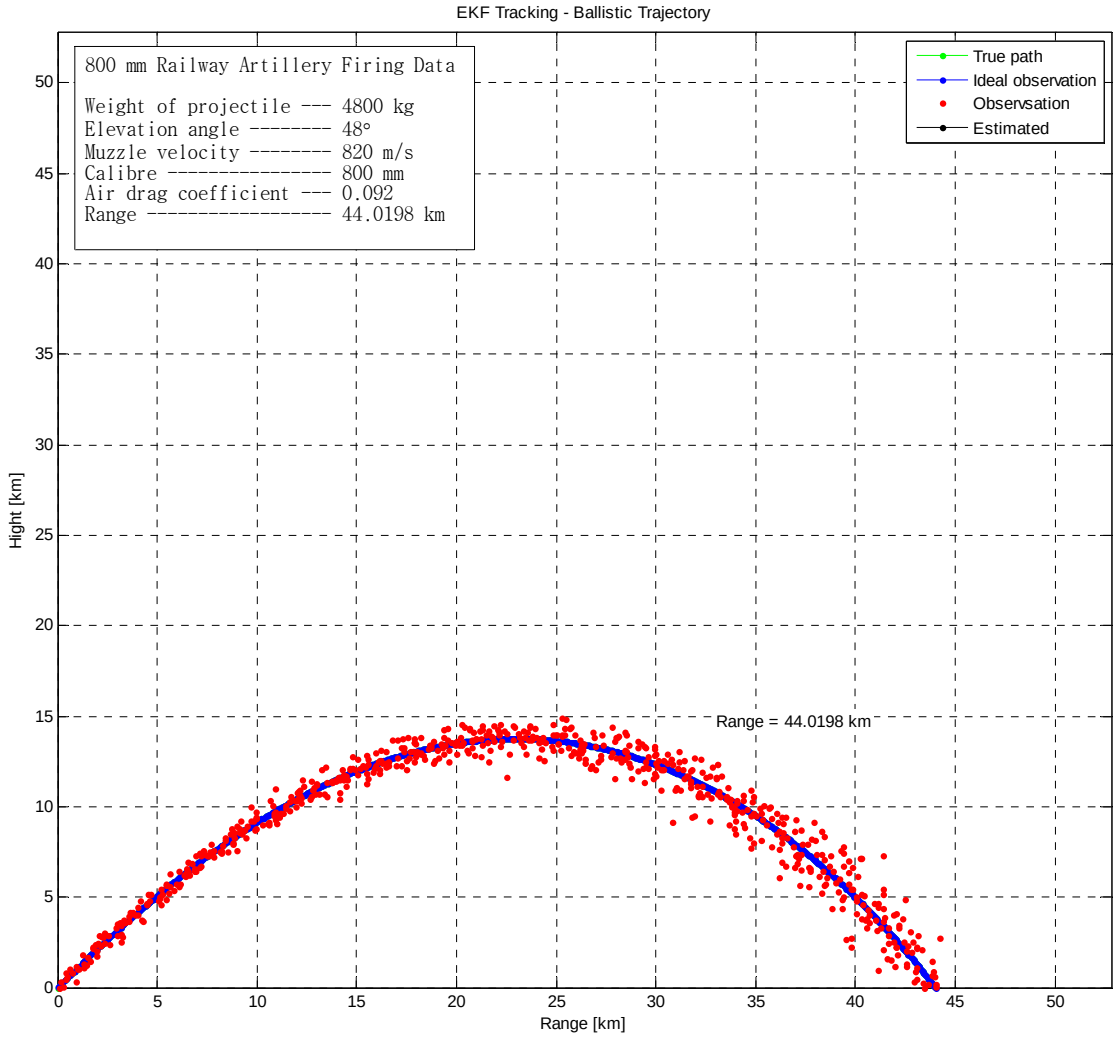


Figure 3. Observation Trajectory (with process noise and observation noise) – (red dots)

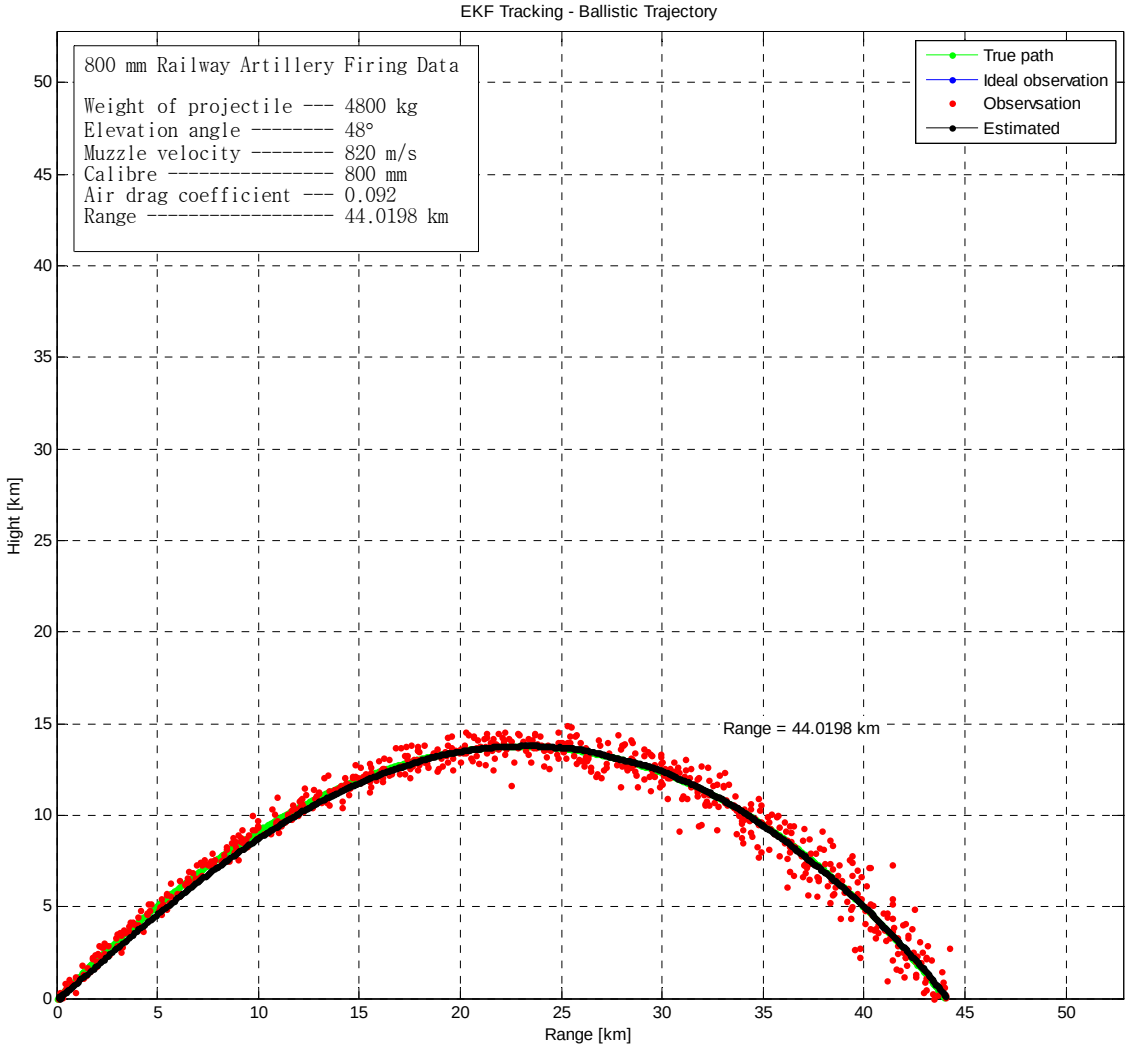


Figure 4. Estimated Trajectory (black curve), Observation Trajectory and True Trajectory

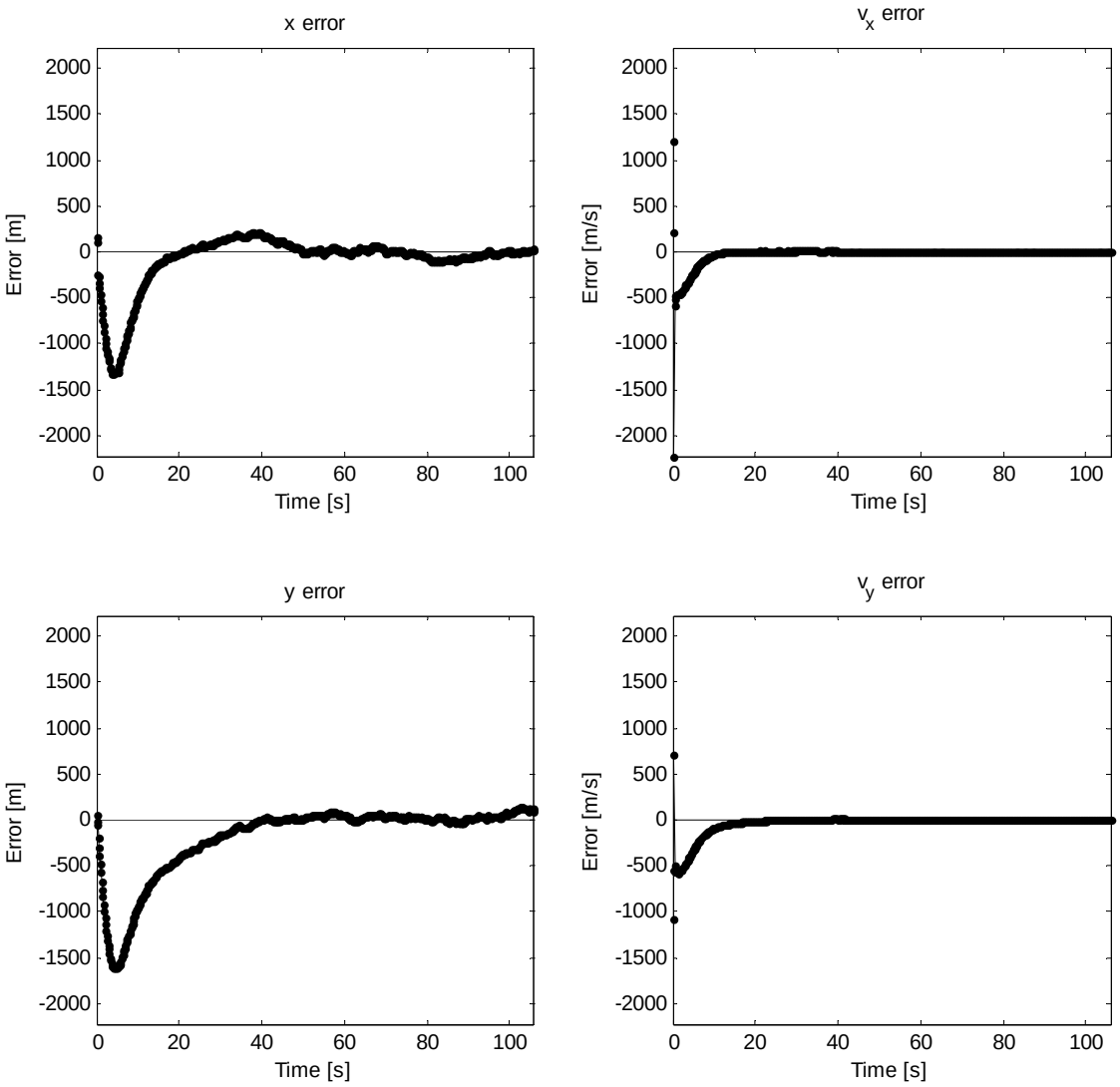


Figure 5. Error Analysis (one time)

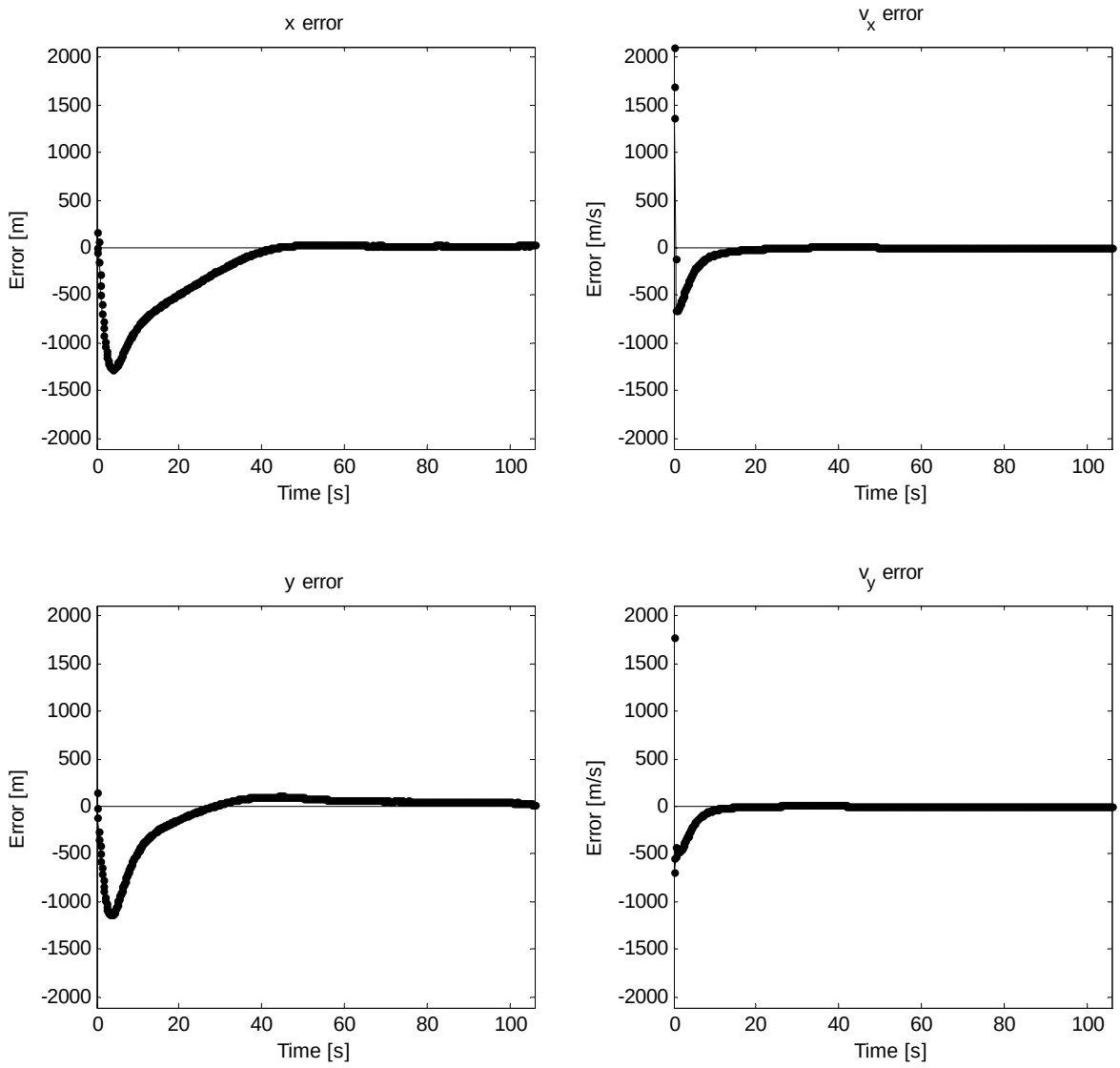


Figure 6. Error Analysis (100 times average)

8. Conclusions

The future work of this project is in three dimensional. Here we have used ranged elevation angle but for extension Actual aero plane path i.e.; not smooth as parabola it can be of any random shape modeled with the Non linear dynamic State Equation.

9. Source Code

```
function [e, t] = EKFTracking(showresults)

    % ------------------------------------------------------------------------
    % Extended Kalman Filter Tracking Problem
    % ------------------------------------------------------------------------
    % clc;
    % clear;
    % ------------------------------------------------------------------------
    % 800 mm cannon firing data
    % ------------------------------------------------------------------------
    m   = 4800;                 % kg
    u   = deg2rad(48);          % degree -> radians
    v   = 820;                  % m/s
    r   = 800;                  % Calibre in mm
    Cr  = 0.092;                % drag coefficient
    g   = 9.81;                 % constant
    q   = 1;                    % m^2 * s^-3

    S = pi * (r/1000)^2;
    % ------------------------------------------------------------------------
    % Data generator
    % ------------------------------------------------------------------------
    tx(:,1) = 0;
    ty(:,1) = 0;
    tvx(:,1) = 0;
    tvy(:,1) = 0;

    tx(1) = 0;
    ty(1) = 0;
    tvx(1) = v * cos(u);
    tvy(1) = v * sin(u);

    n = 1;

    dm = (v^2 * sin(2*u)) / g;
    hm = (dm/2) * tan(u) - (g * (dm/2)^2)/(2*(v*cos(u))^2);

    Fig = figure('Name', 'Ballistic Curve', 'NumberTitle', 'off');
    set(Fig, 'Position', [385 363 781 691]);
    h = plot(tx/1000, ty/1000, 'g.-'); grid on; v
    title('Ballistic Curve');
    xlabel('Range (km)');
    ylabel('Hight (km)');
    axis([0 max([dm/1000 hm/1000]) 0 max([dm/1000 hm/1000])]);

    tf = dm / (v * cos(u));     % estimated flight time
    dT = tf / 800;              % second

    while true
        tx(n+1)  = tx(n) + tvx(n) * dT - 0.5 * m^(-1) * S * Cr * Rou(ty(n)) * sqrt(tvx(n)^2+tvy(n)^2) * tvx(n)
* 0.5 * dT^2;
        tvx(n+1) = tvx(n) - 0.5 * m^(-1) * S * Cr * Rou(ty(n)) * sqrt(tvx(n)^2+tvy(n)^2) * tvx(n) * dT;
        ty(n+1)  = ty(n) + tvy(n) * dT - 0.5 * m^(-1) * S * Cr * Rou(ty(n)) * sqrt(tvx(n)^2+tvy(n)^2) * tvy(n)
* 0.5 * dT^2 - g * 0.5 * dT^2;
        tvy(n+1) = tvy(n) - 0.5 * m^(-1) * S * Cr * Rou(ty(n)) * sqrt(tvx(n)^2+tvy(n)^2) * tvy(n) * dT - g *
dT;
        n = n + 1;
        if tx(n) < 0 || ty(n) < 0
            Range = tx(n);
            break;
        end
```

```
    set(h, 'xdata', tx/1000, 'ydata', ty/1000);
    drawnow;
  end
  text(0.75*Range/1000,  Range/3000, ['Range = ', num2str(Range/1000), ' km'], 'BackgroundColor', [1 1
1]);
  if dm > Range
    axis([0 (6/5)*Range/1000 0 (6/5)*Range/1000]);
  end
  % -------------------------------------------------------------------------
  % Add process noise & measurement noise
  % -------------------------------------------------------------------------
  sigr2  = 280^2;              % sigr = 280 m
  sigel2 = 0.027^2;            % sigel = 0.027 rad (1.547 degree)

  x(:,1)  = 0;
  y(:,1)  = 0;
  vx(:,1)  = 0;
  vy(:,1)  = 0;

  d(:,1) = 0;
  h(:,1) = 0;

  N = numel(tx);

  Q = q * [ (dT^3)/3    (dT^2)/2    0           0          ;
            (dT^2)/2    dT          0           0          ;
            0           0           (dT^3)/3    (dT^2)/2   ;
            0           0           (dT^2)/2    dT         ];

  for n = 1:N
  %    ux = randsign * rand * sqrt(varu);
  %    uy = randsign * rand * sqrt(varu);
    us = sqrtm(Q) * [wgn(1,1,0) wgn(1,1,0) wgn(1,1,0) wgn(1,1,0)]';
  %    ux = wgn(1,1,0) * sqrt(varu);
  %    uy = wgn(1,1,0) * sqrt(varu);

    x(n)  = tx(n)  + us(1);
    vx(n) = tvx(n) + us(2);
    y(n)  = ty(n)  + us(3);
    vy(n) = tvy(n) + us(2);

  %    ed = randsign * rand * sqrt(sigr2);
  %    eh = randsign * rand * sqrt(sigel2);

    R = Rn(x(n), y(n), sigr2, sigel2);
    ed = wgn(1,1,0) * sqrt(R(1,1));
    eh = wgn(1,1,0) * sqrt(R(2,2));

  %    ed = wgn(1,1,0) * sqrt(sigr2);
  %    eh = wgn(1,1,0) * sqrt(sigel2);

    d(n) = x(n) + ed;
    h(n) = y(n) + eh;
  end
  % -------------------------------------------------------------------------
  % Extended Kalman Filter
  % -------------------------------------------------------------------------
  I = [    1   0   0   0   ;
           0   1   0   0   ;
           0   0   1   0   ;
           0   0   0   1   ];

  A = [    1   dT  0   0   ;
           0   1   0   0   ;
```

```
        0   0   1   dT  ;
        0   0   0   1   ];

% Q = q * [ (dT^3)/3    (dT^2)/2    0           0          ;
%          (dT^2)/2    dT          0           0          ;
%          0           0           (dT^3)/3    (dT^2)/2   ;
%          0           0           (dT^2)/2    dT         ];

G = [     (dT^2)/2    0          ;
          dT          0          ;
          0           (dT^2)/2   ;
          0           dT         ];

H = [     1   0   0   0   ;
          0   0   1   0   ];

R1 = Rn(d(1), h(1), sigr2, sigel2);
sigd2 = R1(1,1);
sigh2 = R1(2,2);
sigdh = R1(1,2);

P0n0 = [   sigd2       -sigd2/dT       sigdh       -sigdh/dT       ;
          -sigd2/dT    2*sigd2/dT^2    -sigdh/dT   2*sigdh/dT^2    ;
           sigdh       -sigdh/dT       sigh2       -sigh2/dT       ;
          -sigdh/dT    2*sigdh/dT^2    -sigh2/dT   2*sigh2/dT^2    ];

% s0n0   = [d(2) (d(2)-d(1))/dT h(2) (h(2)-h(1))/dT]';
s0n0   = [0 vx(1) 0 vy(1)]';

s1n0   = A * s0n0 + G * fs(s0n0, m, S, Cr) + G * [0 -g]';
P1n0   = (A + G*F(s0n0, m, S, Cr)) * P0n0 * (A + G*F(s0n0, m, S, Cr))' + Q;
K1n    = P1n0 * H' * inv(H*P1n0*H' + Rn(d(1),h(1),sigr2,sigel2));
shat(:,1) = s1n0 + K1n * ([d(1) h(1)]' - H * s1n0);
P0n0   = (I - K1n * H) * P1n0;

for n = 2:N
   s1n0   = A * shat(:,n-1) + G * fs(shat(:,n-1), m, S, Cr) + G * [0 -g]';
   P1n0   = (A + G*F(shat(:,n-1), m, S, Cr)) * P0n0 * (A + G*F(shat(:,n-1), m, S, Cr))' + Q;
   K1n    = P1n0 * H' * inv(H*P1n0*H' + Rn(d(n),h(n),sigr2,sigel2));
   shat(:,n) = s1n0 + K1n * ([d(n) h(n)]' - H*s1n0);
   P0n0   = (I - K1n * H) * P1n0;

end
% ---------------------------------------------------------------------
% Error analysis
% ---------------------------------------------------------------------
s = [x(1,:); vx(1,:); y(1,:); vy(1,:)];
e(1:4,1) = 0;
t(1,:) = 0;
for i = 1:N
   e(1:4,i) = shat(:,i) - s(:,i);
   t(i+1) = t(i) + dT;
end
t(N+1) = [];
Xmax = max(t);
emin = min([min(e(1,:)) min(e(2,:)) min(e(3,:)) min(e(4,:))]);
emax = max([max(e(1,:)) max(e(2,:)) max(e(3,:)) max(e(4,:))]);
Ymax = max([abs(emin) abs(emax)]);
Ymin = -Ymax;

if showresults == 1
   figure('Name', 'Error Analysis', 'NumberTitle', 'off');
   subplot 221;
   plot(t, zeros(1,N), 'k'); hold on;
```

```
    plot(t, e(1,:), 'k.-');
    title('x error');
    xlabel('Time [s]');
    ylabel('Error [m]');
    axis([0 Xmax Ymin Ymax]);

    subplot 222;
    plot(t, zeros(1,N), 'k'); hold on;
    plot(t, e(2,:), 'k.-');
    title('v_x error');
    xlabel('Time [s]');
    ylabel('Error [m/s]');
    axis([0 Xmax Ymin Ymax]);

    subplot 223;
    plot(t, zeros(1,N), 'k'); hold on;
    plot(t, e(3,:), 'k.-');
    title('y error');
    xlabel('Time [s]');
    ylabel('Error [m]');
    axis([0 Xmax Ymin Ymax]);

    subplot 224;
    plot(t, zeros(1,N), 'k'); hold on;
    plot(t, e(4,:), 'k.-');
    title('v_y error');
    xlabel('Time [s]');
    ylabel('Error [m/s]');
    axis([0 Xmax Ymin Ymax]);
end
% ---------------------------------------------------------------------
% Result (plot & display)
% ---------------------------------------------------------------------
Xmax = (6/5)*Range/1000;
Ymax = (6/5)*Range/1000;
if showresults == 1
    Fig = figure('Name', 'Kalman Tracking', 'NumberTitle', 'off');
    set(Fig, 'Position', [385 363 781 691]);
    plot(d/1000, h/1000, 'r.'); grid on; hold on;
    plot(tx/1000, ty/1000, 'g', 'LineWidth', 3); hold on;
    plot(x/1000, y/1000, '.-'); hold on;
    plot(shat(1,:)/1000, shat(3,:)/1000, 'k.-'); hold on;

    % InterV = 3;
    % for j = 1:N
    %     if mod(j, InterV) == 0
    %         text(shat(1,j)/1000+0.02, shat(3,j)/1000+0.02, num2str(j), 'Color', 'k');
    %     end
    % end
    %
    % for j = 1:N
    %     if mod(j, InterV) == 0
    %         text(x(1,j)/1000+0.02, y(1,j)/1000+0.02, num2str(j), 'Color', 'b');
    %     end
    % end

    legend('Observsation', 'True path', 'Ideal observation', 'Estimated');
    Tit = [num2str(r), ' mm Railway Artillery Firing Data'];
    WoP = ['Weight of projectile --- ', num2str(m), ' kg'];
    ElA = ['Elevation angle -------- ', num2str(rad2deg(u)), '\circ'];
    MuV = ['Muzzle velocity -------- ', num2str(v), ' m/s'];
    Cal = ['Calibre ---------------- ', num2str(r), ' mm'];
    ADC = ['Air drag coefficient --- ', num2str(Cr)];
    Ran = ['Range ------------------ ', num2str(Range/1000), ' km'];
```

```
%       annotation(Fig, 'textbox', [0.1434 0.7329 0.2477 0.1798], ...
%           'String', {   Tit,    ...
%                       '',     ...
%                       WoP,    ...
%                       ElA,    ...
%                       MuV,    ...
%                       Cal,    ...
%                       ADC,    ...
%                       Ran},   ...
%           'FontSize', 12,     ...
%           'FontName', 'BatangChe', ...
%           'FitBoxToText', 'on', ...
%           'BackgroundColor', [1 1 1], ...
%           'VerticalAlignment', 'top');
        title('EKF Tracking - Ballistic Trajectory');
        xlabel('Range [km]');
        ylabel('Hight [km]');
        axis([0 Xmax 0 Ymax]);
    end

    %end
```