Car alignment tracking in a parking space using 2 cameras

MAE 6760: Model Based Estimation

Madhav Gupta

Abstract

The aim of this project is track the forward most point of a car entering a parking spot. This system can be especially applied in formula racing pit box. The motion of the vehicle is non linear, therefore the Extended Kalman Filter has been used to estimate the states of the vehicle. Hypothesis Testing has also been used to determine if the vehicle is on the centre line. Two pinhole pan tilt cameras are placed at either side of the parking space at the same x and y coordinate.

Contents

1 Introduction and Theory

A qualitative method to train Formula1 drivers is proposed. The scenario consists of a F1 car which needs to stop at a specific position. The specific position is given by the four vertical (yellow) markers in the image below. These markers are representative of where the tyres should be for the perfect pit stop. The drivers come into this pit-box at various different speeds. The car consists of an accelerator in the form of an IMU. A camera exists which is tracking the front axle of the vehicle. It is proposed that given a deceleration, the car will stop at the right location. In order to train the drivers, a hypothesis test is proposed to warn if it is certain that they will miss the markers (due to excess speed). We do not estimate the angular position of the vehicle as it is highly unlikely that the IMU of the vehicle and the external cameras will communicate.

Figure 1: F1 car which needs to stop at a specific position

1.1 Extended Kalman Filter

The extended Kalman filter (EKF) is an algorithm used to estimate the state of a nonlinear system in the presence of noisy sensor data. It works by linearizing the system's dynamics around the current estimate of the system state, using the Jacobian matrix. This allows the standard Kalman filter equations to be applied to nonlinear systems. The EKF algorithm works in two steps: prediction and correction. In the prediction step, the algorithm uses the current state estimate and the system dynamics to predict the state of the system at the next time step. In the correction step, the algorithm uses noisy sensor measurements to refine the state estimate. This process is repeated at each time step, with the state estimate becoming more accurate as more sensor measurements are taken. The extended Kalman filter (EKF) algorithm can be expressed using the following equations:

$$\hat{\mathbf{x}}_{k+1|k} = \mathbf{f}(\mathbf{x}_k, \mathbf{w_k})$$
$$P_{k+1|k} = FP_{k|k}F^\top + GQG^\top$$
$$K_{k+1} = P_{k+1|k}H_{k+1}^\top(H_{k+1}P_{k+1|k}H_{k+1}^\top + R)^{-1}$$
$$\hat{\mathbf{x}}_{k+1|k+1} = \hat{\mathbf{x}}_{k+1|k} + K_{k+1}(\mathbf{z}_{k+1} - \mathbf{h}(\hat{\mathbf{x}}_{k+1|k}))$$
$$P_{k+1|k+1} = (I - K_{k+1}H_{k+1})P_{k+1|k}(I - K_{k+1}H_{k+1})^\top K_{k+1}RK_{k+1}^\top$$

In these equations, $\mathbf{x}_k$ is the state estimate at time k, $\mathbf{P}_k$ is the state covariance matrix, $\mathbf{u}_k$ is the control input, $\mathbf{F}_k$ is the state transition matrix, $\mathbf{Q}_k$ is the process noise covariance matrix, $\mathbf{z}_k$ is the sensor measurement, $h(\cdot)$ is the measurement function, $\mathbf{H}_k$ is the measurement matrix, $\mathbf{R}_k$ is the measurement noise covariance matrix, $\mathbf{y}_k$ is the measurement residual, $\mathbf{S}_k$ is the residual covariance matrix, and $\mathbf{K}_k$ is the Kalman gain matrix.

1.2 Dynamic Model

A eight state system has been considered for this system with states:

$$X = \begin{bmatrix} x & \dot{x} & y & \dot{y} & z & \dot{z} & \theta & \dot{\theta} \end{bmatrix}^T \tag{1}$$

The dynamic model of the vehicle is given as follows:

$$\dot{X} = \begin{bmatrix} \dot{x} & a\cos\theta + w_x & 0 & w_y & \dot{z} & a\sin\theta + w_z & \dot{\theta} & \eta + w_\theta \end{bmatrix}^T = f \tag{2}$$

Here, a is the acceleration or deceleration rate of the vehicle. η is the rate of change of angular velocity. They are set to be constants.

The discretized model is given in the euler form of $X_{k+1} = FX_k + Gw_k$ as:

$$F = \frac{\delta f}{\delta X} \tag{3}$$

$$= \begin{bmatrix} 1 & dt & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -a * dt * \sin\theta & 0 \\ 0 & 0 & 1 & dt & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & dt & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & a * dt * \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 \\ dt & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & dt & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & dt \end{bmatrix} \tag{5}$$

1.3 Pinhole camera measurement

There are two pin hole cameras which are placed at the zero line of the parking space. Both of the cameras measure the polar and azimuthal angles as given by β and γ in the figure below.

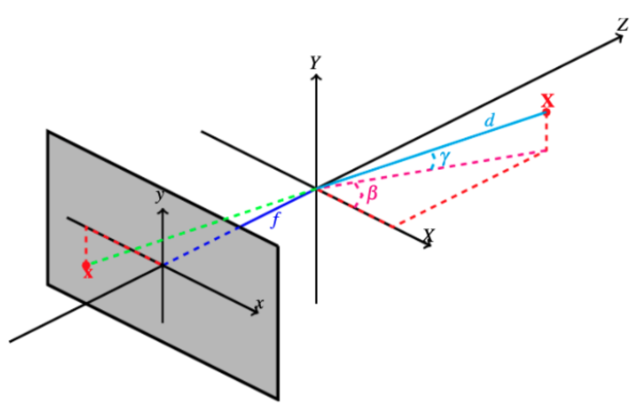

Figure 1. The pinhole camera model. A point **X** is projected on the image plane in **x**. d is the distance from **X** to the camera center. β and γ are the azimuthal and polar angles of the ray from **X** through the optical center, respectively. f is the focal length of the camera.

The measurements are generated using the equations below.

$$Z = \begin{bmatrix} \tan^{-1}\frac{z_k}{x_k} \\ \tan^{-1}\frac{y_k}{\sqrt{x_k^2+z_k^2}} \\ \tan^{-1}\frac{W-z_k}{x_k} \\ \tan^{-1}\frac{y_k}{\sqrt{x_k^2+(W-z_k)^2}} \end{bmatrix} \tag{6}$$

$$H = \frac{\delta z_k}{\delta X} \tag{7}$$

$$= \begin{bmatrix} \frac{-z_k}{x_k^2+z_k^2} & 0 & 0 & 0 & \frac{x_k}{x_k^2+z_k^2} & 0 & 0 & 0 \\ \frac{-x_k y_k}{(x_k^2+y_k^2+z_k^2)\sqrt{x_k^2+z_k^2}} & 0 & \frac{\sqrt{x_k^2+z_k^2}}{(x_k^2+y_k^2+z_k^2)} & 0 & \frac{-z_k y_k}{(x_k^2+y_k^2+z_k^2)\sqrt{x_k^2+z_k^2}} & 0 & 0 & 0 \\ \frac{-(W-z_k)}{x_k^2+(W-z_k)^2} & 0 & 0 & 0 & \frac{-x_k}{x_k^2+(W-z_k)^2} & 0 & 0 & 0 \\ \frac{-x_k y_k}{(x_k^2+y_k^2+(W-z_k)^2)\sqrt{x_k^2+(W-z_k)^2}} & 0 & \frac{\sqrt{x_k^2+(W-z_k)^2}}{(x_k^2+y_k^2+(W-z_k)^2)} & 0 & \frac{(W-z_k)y_k}{(x_k^2+y_k^2+(W-z_k)^2)\sqrt{x_k^2+(W-z_k)^2}} & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

1.4 Hypothesis Testing

Hypothesis testing has been used to test if the vehicle has crossed the marked yellow lines based on our estimates. As there are mechanics on either side in the z direction and the vehicle is not moving in the vertical y direction, the testing has been done only for the x direction. Since all three dimensions are independent, we can write:

$$p(goal|\hat{X_{k|k}}) = p(goal|\hat{x_{k|k}})p(goal|\hat{y_{k|k}})p(goal|\hat{z_{k|k}}) \tag{9}$$

$$= p(goal|\hat{x_{k|k}})\delta(goal|\hat{y_{k|k}})\delta(goal|\hat{z_{k|k}}) \tag{10}$$

Here $\delta(goal|\hat{y_{k|k}})$ and $\delta(goal|\hat{z_{k|k}})$ are the dirac delta functions which tell wether the vehicle is within the box or not. Now, for simplicity, the markers are at $x = -0.1$. Hence,

$$p(goal|\hat{x_{k|k}}) = p(x_k \geq -0.1|x_{k|k}) \tag{11}$$

$$= 1 - p(L|\mathcal{N}(\hat{x_{k|k}}, \sigma_{x_{k|k}})) > 95\% \tag{12}$$

2 Results

As the dynamic model is custom built for this project, two cases have been used to justify the model. The first one is a linear case, where given an initial velocity, the vehicle enters the space. The second case is more realistic, where the vehicle enters the space from a different lane.

2.1 Linear case

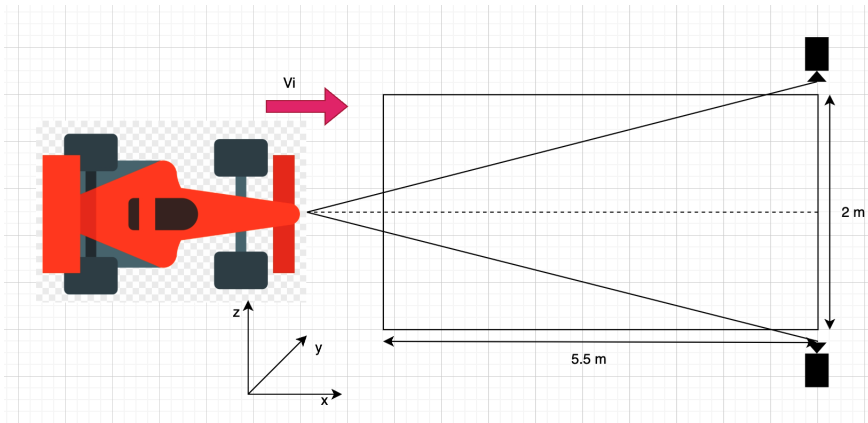

Figure 2: Linear Case

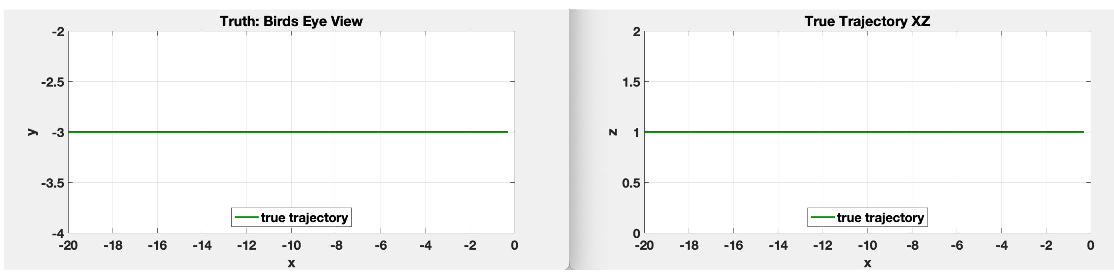

Figure 3: True trajectory

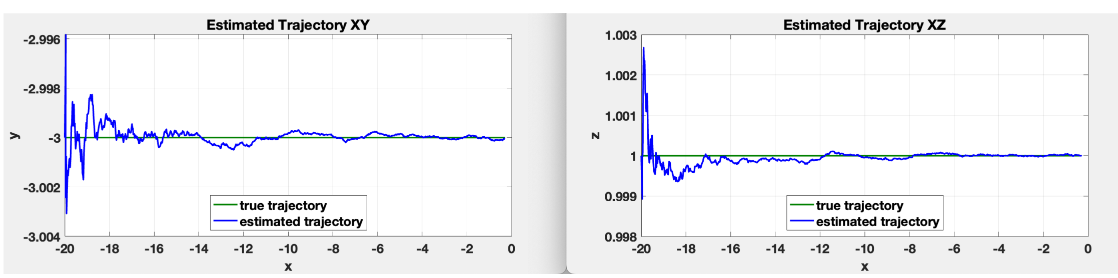

Figure 4: Estimated trajectory

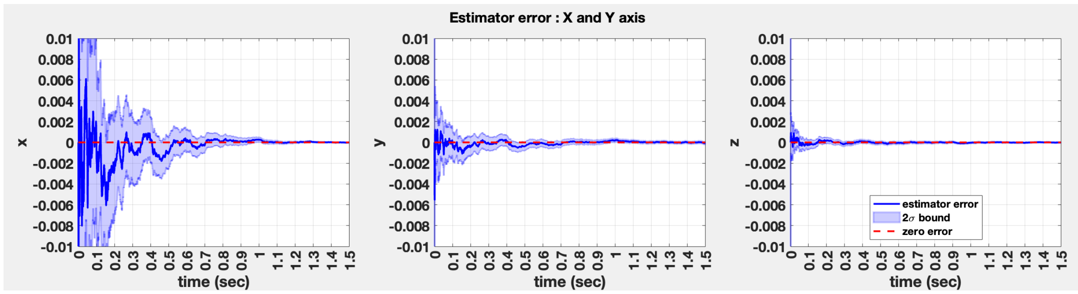

Figure 5: Estimated error

2.2 Non-Linear Case

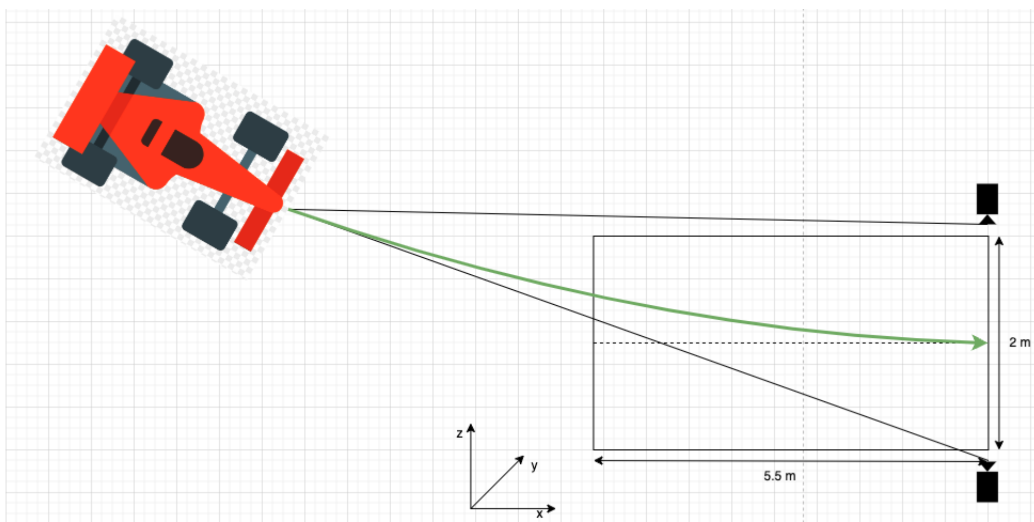

Figure 6: Non-Linear case

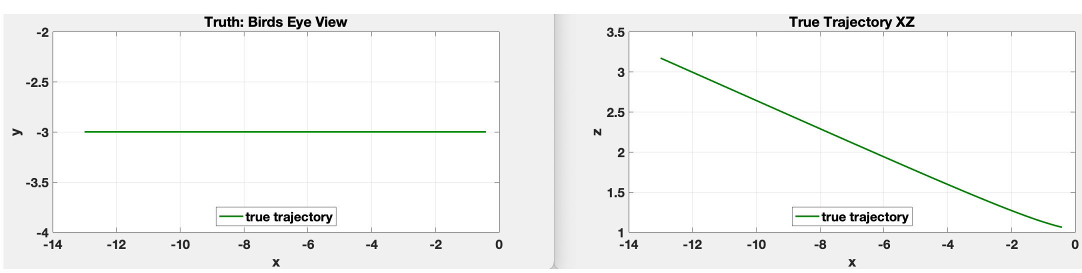

Figure 7: True Trajectory

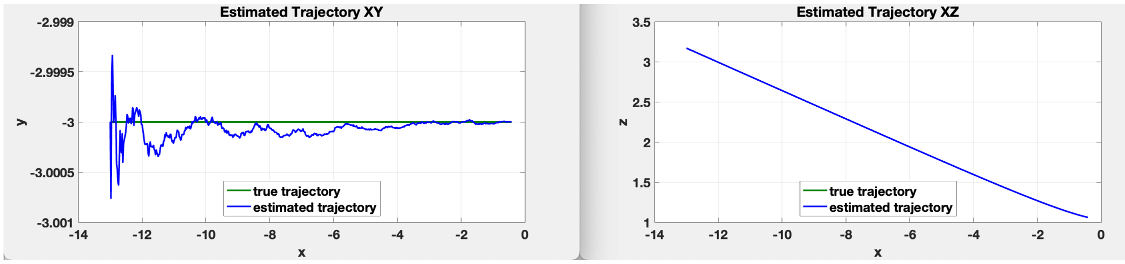

Figure 8: Estimated Trajectory

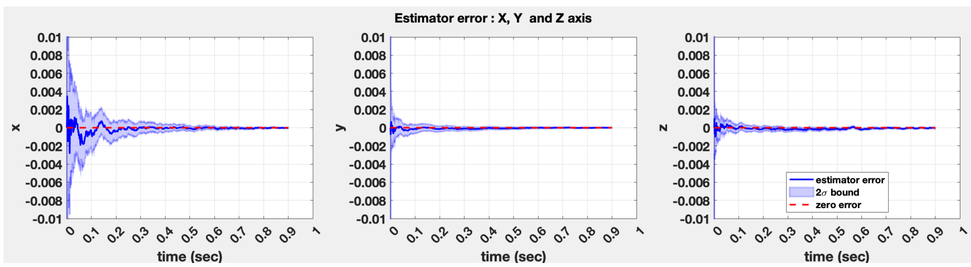

Figure 9: Estimator Error

2.3 Hypothesis Testing

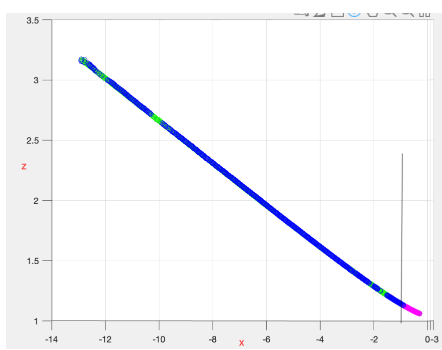

Figure 10: Hypothesis Test

Hypothesis Testing with 95% confidence level has been used to justify that the vehicle is in the marked spot. This is given by the graph above. As soon as the vehicle is at the marked spot, the system should tell the driver to apply brakes.

3 Discussion and conclusion

To make our system visible, we employed two different cameras. We can see that, in the instance of a single camera, the aforementioned data will prevent us from determining the camera's range. We are able to collect two different measurements—one for z and one for W z, where W is the pitch width along the z-axis—by using two separate cameras. Our system is now observable as a result.

Under the provided model and Pan Tilt camera data, our EKF tracking model appears to operate rather well. The non-linearities in our data are quite well captured by the EKF, and ultimately, in both cases, our estimations converge to the true no noise situation. I utilised the two examples to see how our model functions under incorrect beginning circumstances. I also experimented to see how the starting covariances affected the convergence in this situation and discovered that the convergence is considerably smoother and quicker with larger beginning covariance.

As seen in the graphic, our hypothesis testing, which employed 95% confidence intervals on our estimations, provides a reasonably accurate goal line tracking.

4 References

1. T. Yomchinda, "A method of multirate sensor fusion for target tracking and localization using extended Kalman Filter," 2017 Fourth Asian Conference on Defence Technology - Japan (ACDT), 2017, pp. 1-7, doi: 10.1109/ACDTJ.2017.8259590.

2. Allebosch, Gianni Hamme, David Veelaert, Peter Philips, Wilfried. (2019). Robust Pan/Tilt Compensation for Foreground–Background Segmentation. Sensors. 19. 2668. 10.3390/s19122668.

3. VISION-BASED POSITION ESTIMATION UTILIZING AN EXTENDED KALMAN FILTER by Joseph B. Testa III https://apps.dtic.mil/sti/pdfs/AD1031534.pdf

Matlab Code

```
%% MAE 6760 Final Project

clc; clear;
close all;
rng(100);

%% Simulation

n=8; %number of states
box_width = 2;
x0 = -13; y0= -3; z0 = 3.17; T0=deg2rad(-10); %initial position and orientation

dt = 0.001;
t = 0:dt:0.9;
```

```
nt = length(t);

omega = -T0/t(end);
omegadot = -omega/t(end);

a = -18.52; %deceleration m/s2
vi = 22.5; %initial velocity m/s

xdot0 = vi*cos(T0); %initial velocity %m/s
ydot0 = 0; %rate of change of height
zdot0 = vi*sin(T0);
Tdot0 = 0;

Xnonoise = [x0; xdot0; y0; ydot0; z0; zdot0; T0; Tdot0];

for k=1:(nt-1)
    xdot = Xnonoise(2,k);
    ydot = Xnonoise(4,k);
    zdot = Xnonoise(6,k);
    Tdot = Xnonoise(8,k);
    T = Xnonoise(7,k);
    Xnonoise(:, k+1)= Xnonoise(:,k) + dt*...
        [xdot;
        a*cos(T);
        ydot;
        0;
        zdot;
        a*sin(T);
        Tdot;
        omegadot];
end
%
% plot_birdseyeview(Xnonoise,[],[],'Truth: Birds Eye View', [1,3]);
% PrepFigPresentation(gcf)
% plot_birdseyeview(Xnonoise , [], [],"True Trajectory XZ", [1,5])
% PrepFigPresentation(gcf)
% camup([0 1 0]);

%% cameras setup
psi1 = atan (Xnonoise(5 ,:)./Xnonoise(1 ,:));
phi1 = atan (Xnonoise(3 ,:)./sqrt(Xnonoise(1 ,:).^2 + Xnonoise(5,:).^2) );
psi2 = atan ((box_width - Xnonoise (5 ,:))./ Xnonoise (1 ,:));
phi2 = atan (Xnonoise(3, :)./sqrt(Xnonoise(1,:).^2 + (box_width-Xnonoise(5,:)).^2));
nz = 4;
nw = 4;
```

```
R= eye(nz)*0.005^2; v= sqrtm(R)*randn(nz,nt);
Q=eye(nw)*0.5^2 ; w= sqrtm(Q)* randn(nw ,nt);
Z = (180/pi)*[psi1 ; phi1; psi2; phi2] +v;
Zacc = w;
% figure(7)
% plot(t, rad2deg(psi1))
% hold on
% plot(t, rad2deg(phi1))
% plot(t, rad2deg(psi2), "g--")
% plot(t, rad2deg(phi2), "--")
% legend(["Psi1", "Phi1", "Psi2", "Phi2"])
% ylabel("angle (degrees)")
% xlabel("time")
% title("Measurements verses time")


%% EKF - nonlinear

xdot0 = vi*cos(T0); %initial velocity %m/s
ydot0 = 0; %rate of change of height
zdot0 = vi*sin(T0);
Tdot0 = 0;

x0 = [x0; xdot0; y0; ydot0; z0; zdot0; T0; Tdot0];

%P0=eye(n)*0.001^2;
P0=diag([1 1 1 1 1 1 0 0])*1^2;
n= length(x0);
xhatp =x0;Pp(1:n ,1:n ,1)=P0;
xhatu =x0;Pu(1:n ,1:n ,1)=P0;
h = 3; %height of camera

for k=1:(nt-1)
    %predict state
    xhatp(:,k+1) = predict_state(xhatu(:,k), Zacc(:,k), dt, a);
    %predict covariance
    [F,G] = getFG(xhatu(:,k), dt, a);
    Pp(1:n, 1:n, k+1) = F*Pu(1:n, 1:n, k)*F' + G*Q*G';
    %Kalman Gain
    H = getH(xhatp(:, k+1));
    Zp = (180/pi)*...
        [atan(xhatp(5,k+1)/xhatp(1,k+1));
        atan(xhatp(3,k+1)/sqrt(xhatp(1,k+1)^2 + xhatp(5,k+1)^2));
        atan((box_width - xhatp(5,k+1))/xhatp(1,k+1));
        atan(xhatp(3,k+1)/sqrt(xhatp(1,k+1)^2 + (box_width - xhatp
            (5,k+1))^2))];
    inn = Z(:, k+1) - Zp;
    S = H*Pp(1:n, 1:n, k+1)*H' + R;
```

```
    K = Pp(1:n, 1:n, k+1)*H'*inv(S);

    %Update
    xhatu(:, k+1) = xhatp(:, k+1) + K*(inn);
    Pu(1:n, 1:n, k+1) = (eye(n) - K*H)*Pp(1:n, 1:n, k+1)*(eye(n) - K*H)' + K*R*K';

    %test hypothesis
    goal = check(xhatu(:, k+1), Pu(:, :, k+1));
end
%
% plot_birdseyeview(Xnonoise, xhatu, Pu,'Estimated Trajectory XY',[1 ,3])
% PrepFigPresentation(gcf)
% plot_birdseyeview(Xnonoise, xhatu, Pu,'Estimated Trajectory XZ',[1 ,5])
% PrepFigPresentation(gcf)
% ii_plot = [1 3 5];
% plot_estimator_error(t, Xnonoise, xhatu, Pu, ii_plot, "Estimator error : X, Y and Z axis");
% ii_plot = [5 7];
% plot_estimator_error(t, Xnonoise, xhatu, Pu, ii_plot, "Estimator error : and \theta axis");

%% Plotting Trajectory
figure(8)
for k =1:(nt)
    goal = check(xhatu(:,k),Pu(:,:,k));
    scatter3(Xnonoise(1,k),Xnonoise(3,k), Xnonoise(5,k),'green')
    if goal == 1
        scatter3(xhatu(1,k),xhatu(3,k),xhatu(5,k),'magenta')
        hold on;
    else
        scatter3(xhatu(1,k),xhatu(3,k),xhatu(5,k),'blue')
        hold on;
    end
end

%% EKF function calls

function Xkp1 = predict_state(Xk, U, dt, a)
wx = 0*U(1); wy = 0*U(2); wz = 0*U(3); wT = 0*U(4);
T = Xk(7);
Xkp1 = Xk + dt*...
    [Xk(2);
    a*cos(T);
    Xk(4);
    0;
```

```
    Xk(6);
    a*sin(T);
    Xk(8);
    0];
end

function [F,G] = getFG(X,dt,a)

Tk = X(7);

F = [1 dt 0 0 0 0 0 0;
    0 1 0 0 0 0 dt*a*sin(Tk) 0;
    0 0 1 0 0 0 0 0;
    0 0 0 1 0 0 0 0;
    0 0 0 0 1 dt 0 0;
    0 0 0 0 0 1 dt*a*cos(Tk) 0
    0 0 0 0 0 0 1 dt;
    0 0 0 0 0 0 0 1];
G = [0 0 0 0;
    dt 0 0 0;
    0 0 0 0;
    0 dt 0 0;
    0 0 0 0;
    0 0 dt 0;
    0 0 0 0;
    0 0 0 dt];
end

function H = getH(X)
L = 2; %2 m wide pit box
h = 3; %camera is placed 5 m above
x = X(1); y = X(3); z=X(5);T = X(7);

H = (180/pi)*...
    [-z/(x^2+z^2), 0, 0, 0, x/(x^2+z^2), 0, 0, 0;
    -x*y/((x^2+ y^2+ z^2)*sqrt(x^2+ z^2)), 0, sqrt(x^2+z^2)/(x^2+
        y^2+ z^2), 0, -z*y/((x^2+y^2+z^2)*sqrt(x^2+z^2)), 0, 0, 0;
    -(L-z)/(x^2+(L-z)^2), 0, 0, 0, -x/(x^2+(L-z)^2), 0, 0, 0;
    -x*y/((x^2+ y^2+(L-z)^2)*sqrt(x^2+(L-z)^2)), 0, sqrt(x^2+(L-z)
        ^2)/(x^2+ y^2+(L-z)^2), 0, (L-z)*y/((x^2+ y^2+(L-z)^2)*
        sqrt(x^2+(L-z)^2)),0, 0, 0];
end


%% Hypothesis Testing Function

function goal = check(Xk, Pk)
prob = 1 - cdf('Normal', 1, Xk(5), sqrt(Pk(5,5)));
```

```
if prob>0.97
    goal = 1;
else
    goal = 0;
end
end



%% Plotting Functions

function plot_birdseyeview (x1 ,x2 ,P2 , title_name , ii_plot );
% x1 is the true value or reference comparison
% x2 ,P2 is the estimator state and covariance
%
label_name ={ 'x','vx ','y','vy ','z','vz ','theta', 'thetadot'};
ii_x1 =[]; ii_x2 =[]; ii_P2 =[]; % for legend
figure ;
if ~ isempty (x1)
    plot (x1( ii_plot (1) ,:) ,x1( ii_plot (2) ,:) , 'Color' ,[0 0.5 0]) ; ii_x1 =1;
end
hold on;
if ~ isempty (x2),
    plot (x2( ii_plot (1) ,:) ,x2( ii_plot (2) ,:) ,'b-'); ii_x2 =2;
end
xlabel ( label_name ( ii_plot (1) )); ylabel ( label_name ( ii_plot (2) )); grid ;
hold off ;
legend_names ={ 'true trajectory','estimated trajectory','3\ sigma bound '};
legend ( legend_names { ii_x1 }, legend_names { ii_x2 }, legend_names { ii_P2 },'Location','South')
title (title_name);
PrepFigPresentation (gcf);
end

function [Xe,Ye] = calculateEllipseCov(X, P, nsig, steps)
    %# This functions returns points to draw an ellipse
    %#
    %#  @param X     x,y coordinates
    %#  @param P     covariance matrix
    %#

    error(nargchk(2, 4, nargin));
    if nargin<3, nsig = 1; end
    if nargin<4, steps = 36; end
```

```
    [U,S,V]=svd(P);
    s1=sqrt(S(1,1));s2=sqrt(S(2,2));angle=acos(U(1,1))*180/pi;
    x=X(1);
    y=X(2);

    %scale by nsig
    s1=nsig*s1;
    s2=nsig*s2;

    beta = angle * (pi / 180);
    sinbeta = sin(beta);
    cosbeta = cos(beta);

    alpha = linspace(0, 360, steps)' .* (pi / 180);
    sinalpha = sin(alpha);
    cosalpha = cos(alpha);

    Xe = x + (s1 * cosalpha * cosbeta - s2 * sinalpha * sinbeta);
    Ye = y + (s1 * cosalpha * sinbeta + s2 * sinalpha * cosbeta);

end

function plot_estimator(k,x1,x2,z,P2,title_name);
% x1 is the true value or reference comparison
% x2,P2 is the estimator state and covariance
% z is the measurement
%
% DOES NOT CHECK FOR SIZES!!
%
figure;
if ~isempty(z),
    pp=plot(k,x1,'r-',k,x2,'b-.',k,z,'g:');set(pp(3),'color',[0 0.5 0]);
else,
    pp=plot(k,x1,'r-',k,x2,'b-.');
end
hold on;
if ~isempty(P2)
    plot(k,x2-2*sqrt(P2),'b:',k,x2+2*sqrt(P2),'b:');
    axis_name='state estimate';
    legend('true state','estimate','measurement','2\sigma bound','Location','Northeast');
else,
    axis_name='state';
    legend('true state','no noise','measurement','Location','Northeast');
end
```

```matlab
hold off
xlabel('time (sec)');ylabel(axis_name);grid;
title(title_name);
PrepfigPresentation(gcf);
end

function plot_estimator_error(t,x1,x2,P2,ii_plot,title_name);
% x1 is the true value or reference comparison
% x2,P2 is the estimator state and covariance
% ii_plot: 2x1 vector of which states to plot
%
axis_names={'x','xdot','y','ydot','z','zdot', 'T', 'Tdot'};
figure;subplot(122);
%
for i=1:length(ii_plot),
    ii=ii_plot(i);
    subplot(1,3,i);
    err=x2(ii,:)-x1(ii,:);
    plot(t,err,'b-');
    hold on;
    if ~isempty(P2)
        tbound=[t fliplr(t)];
        xbound=[[err+2*sqrt(squeeze(P2(ii,ii,:)))'] fliplr([err-2*sqrt(squeeze(P2(ii,ii,:)))'])];
        patch(tbound,xbound,'b','EdgeColor','b','FaceAlpha',0.2,'EdgeAlpha',0.2);
        plot(t,zeros(length(t),1),'r--');
    end
    hold off
    xlabel('time (sec)');ylabel(axis_names(ii));grid;
    xlim([0 1]);set(gca,'xtick',[0:0.1:1.5]);
    ylim([-0.01 0.01]);set(gca,'ytick',[-0.01:0.002:0.01]);
end
sgtitle(title_name);
%
PrepFigPresentation(gcf);
legend('estimator error','2\sigma bound','zero error','Location','South','fontsize',12);
end


function PrepFigPresentation(fignum);
%
% prepares a figure for presentations
%
% Fontsize: 14
% Fontweight: bold
% LineWidth: 2
```

```
%
figure(fignum);
fig_children=get(fignum,'children'); %find all sub-plots

for i=1:length(fig_children),

    set(fig_children(i),'FontSize',16);
    set(fig_children(i),'FontWeight','bold');

    fig_children_children=get(fig_children(i),'Children');
    set(fig_children_children,'LineWidth',2);
end
end
```