Automated Lane Keeping System for Self-driving Vehicles on Highway

Liangchun Xu, *Tufts University*

1. INTRODUCTION

Automated lane keeping system, which enables the center of mass (CoM) of the vehicle to track the center of the traveling lane (Cerone, 2002), is an essential part for self-driving car. Specifically, this project aims to design lane keeping controller and estimator for self-driving vehicles on highway. With this automated lane keeping system, the vehicle on highway would be capable to track the center line of the lane when initialized or disturbed with small lateral position and yaw angle offsets.

Particularly, four states are chosen to control the vehicles. They are the lateral velocity component v_y, the yaw rate of the vehicle $\dot{\psi}$, the lateral offset between the vehicle and lane centerline y_L, and the angle between the tangent to the lane and the vehicle heading ε_L. The only system control input is the steering angle of the vehicle δ_f. The vehicle motion characteristics can be found in Fig. 1 and Fig. 2 as follows.

The goal of the controller is to keep the vehicle in the center lane, and also the heading of the vehicle parallel to the lane curve. Mathematically it means four states are equal to zero in steady state. This paper assumes the lane width is 3 m on highway, and the maximum vehicle width is 2 m. Initialized with 0.5 m lateral position offset from the center of the traveling lane, the automated lane keeping system should pull the vehicle back to the center within 1 s adjustment. To achieve this control goal, the state estimator must provide lateral position estimate as accurate as $\pm 0.1\ m$, and the standard deviation of the estimate error for the angle between the vehicle heading and the lane curve shouldn't be larger than 1 degrees.

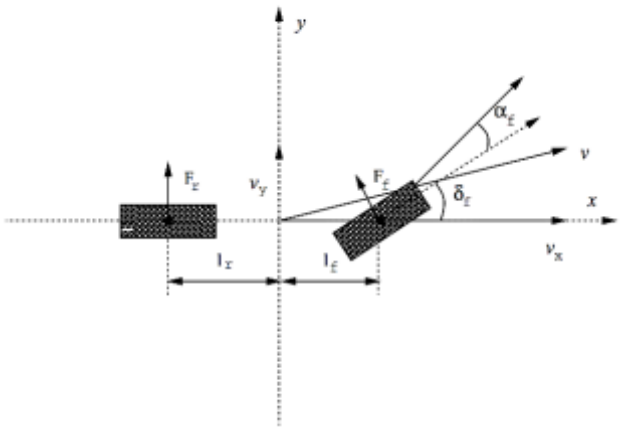

Fig. 1 Dynamic characteristics of vehicle wheels (Kosecka, 1996)

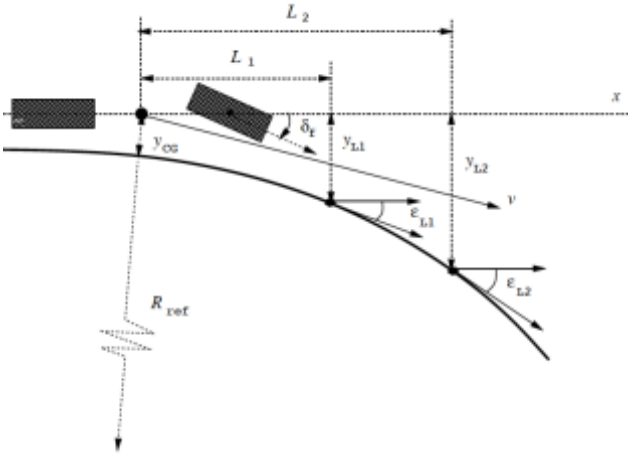

Fig. 2 Kinematic characteristics of vehicle wheels (Kosecka, 1996)

2. ANALYTICAL METHOD

Nonlinear dynamics and observation models are used to build Extended Kalman filter (EKF) and Unscented Kalman filter (UKF) estimators. A linear dynamics model with small angle approximation is used to build a LQR controller for the proposed automated lane keeping system. The nonlinear dynamics with process noise can be represented as Eqs. (1-4) (Kosecka, 1996).

$$\dot{v_y} = -v_x\dot{\psi} + \frac{c_f}{m}\delta_f - \frac{c_f}{m}\arctan\left(\frac{v_y + l_f\dot{\psi}}{v_x}\right) - \frac{c_r}{m}\arctan\left(\frac{v_y - l_r\dot{\psi}}{v_x}\right) \quad (1)$$

$$\ddot{\psi} = \frac{l_f c_f}{I_\psi}\delta_f - \frac{l_f c_f}{I_\psi}\arctan\left(\frac{v_y + l_f\dot{\psi}}{v_x}\right) - \frac{l_f c_r}{I_\psi}\arctan\left(\frac{v_y - l_r\dot{\psi}}{v_x}\right) \quad (2)$$

$$y_L = -v_y - L_a\dot{\psi} - v_x\varepsilon_L \quad (3)$$

$$\dot{\varepsilon_L} = -\dot{\psi} + v_x K_L \quad (4)$$

where v_y is the lateral velocity component, $\dot{\psi}$ is the yaw rate of the vehicle, y_L is the lateral offset between the vehicle and lane centerline, and ε_L is the angle between the tangent to the lane and the vehicle heading. The only process noise ω in system dynamics is the curvature K_L at the looking ahead distance L_a, which is the reciprocal of the radius R_{ref} of the reference lane center line. In this paper, we don't take the lateral slip into account, which means there would be no process noises associated with

other forces and torques imposed on driving vehicles such as the wind drag etc. For highway designing, the curvature is strictly restricted to a small number so that the vehicles won't need to make a dangerous sharp turn. Conversely, the radius is larger than 1000 m, which is also the assumption for setting the covariance matrix for the process noise.

As for the observation equations, Eqs. (5-8) describe four measurements, which are the lateral acceleration a_y, the yaw rate $\dot{\psi}$, the lateral offset between the vehicle and lane centerline y_L, and the angle between the tangent to the lane and the vehicle heading ε_L measured by inertial and vision sensors mounted on the vehicle.

$$a_y = \frac{c_f}{m}\delta_f - \frac{c_f}{m}\arctan\left(\frac{v_y + l_f\dot{\psi}}{v_x}\right) - \frac{c_r}{m}\arctan\left(\frac{v_y - l_r\dot{\psi}}{v_x}\right) + v_1 \quad (5)$$

$$\dot{\psi} = \dot{\psi} + v_2 \quad (6)$$

$$y_L = y_L + v_3 \quad (7)$$

$$\varepsilon_L = \varepsilon_L + v_4 \quad (8)$$

Moreover, the physical meanings and values of other vehicle parameters in the dynamics and observation equations are listed in Table 1.

Table 1 Vehicle parameters

Parameter	Value	Description
$m\ (kg)$	1573	Vehicle mass
$I_\psi\ (kg\ m^2)$	2753	Vehicle inertial
$c_r\ (N/rad)$	2*50000	Cornering stiffness (rear wheel)
$c_f\ (N/rad)$	2*60000	Cornering stiffness (front wheel)
$l_r\ (m)$	1.530	Distance between rear wheel and CoM
$l_f\ (m)$	1.137	Distance between front wheel and CoM
$v_x\ (m/s)$	25	Longitudinal velocity
$L_a\ (m)$	15	Looking ahead distance

Assuming the input noise signals are white Gaussian random variables, the covariance matrices for process and observation noises can be modeled as follows, according to the precision of the off-the-shelf inertial and vision sensors.

$$Q = E(K_L^T * K_L) = \left[\left(\frac{1}{1000}\right)^2\right] \quad (9)$$

$$R = \begin{bmatrix} (1.7*9.8)^2 & 0 & 0 & 0 \\ 0 & \left(\frac{10*\pi}{180}\right)^2 & 0 & 0 \\ 0 & 0 & 0.3^2 & 0 \\ 0 & 0 & 0 & \left(\frac{3*\pi}{180}\right)^2 \end{bmatrix} \quad (10)$$

The observation noise for the lateral acceleration is set to be 1.7 g, while the noise for the yaw rate is set to be 10 deg/s. Both are based on normal performance of off-shore MEMS inertial sensors. The vision sensor can guarantee the measurement noise of the lateral position is less than 0.3 m, and the angular observation error less than 3 degrees. Next, both Kalman filter and Particle filter are designed and simulated under the same initial conditions so that both estimators' performance can be evaluated and compared.

3. STATE ESTIMATOR

Two estimators based on EKF and UKF respectively are built and tested. The initial states are set to be [12, 7*pi/180, 0.5, 3*pi/180], which includes the lateral position offset the automated lane keeping system should be capable to pull back to zero. However, to evaluate the estimators separately, the controller is switched off during the estimator test so that the states are only affected by estimators. The states estimated from EKF and UKF are quite close. Moreover, the Kalman gain matrices for both EKF and UKF estimators are the same for calculation in most time steps. Eqn. (11) is the Kalman gain matrix calculated in one time step.

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0371 & 0.1229 \\ 0 & 0 & 0.0037 & 0.0411 \end{bmatrix} \quad (11)$$

The first two columns of Kalman gain matrix are zeros because in system dynamics equation, the process noise is only imposed on the angle between the tangent to the lane and the vehicle heading ε_L. As a result, the dynamic states v_y, and $\dot{\psi}$ are noisily free in dynamics model, which means the gains assigned to the observation become rather small, even close to zero. The observation estimates from EKF and UKF, the simulating true sensor outputs, and the sensed measurements with noise are compared in Fig. 3. The errors between the EKF and UKF observation estimates, and the simulating true sensor outputs are shown in Fig. 4.

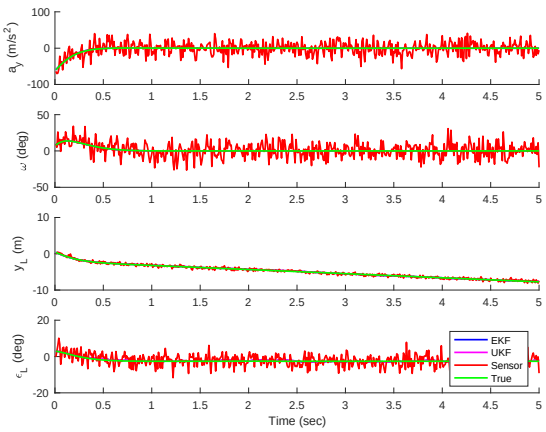

Fig. 3 Estimate, true and sensed observations

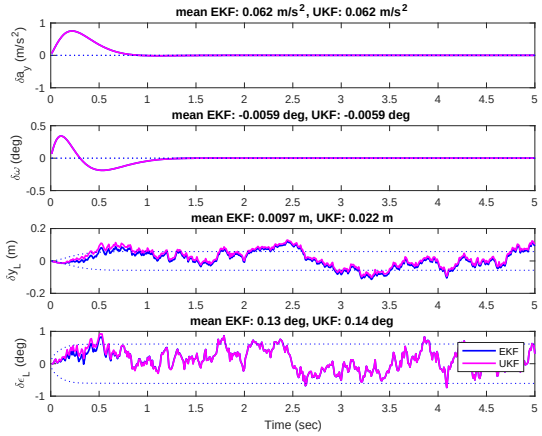

Fig. 4 Error between estimate and true observations

Above estimation error statistics include both the initial transient and steady states. The standard deviation of lateral position estimate error is about 0.06 m. As for the angular offset, the standard deviation of the estimate error is about 0.61 degrees. Considering the error mean of four state estimates are around zero, the lateral position estimate error is bounded in $\pm 0.1\ m$, and the angular offset estimate error is bounded in ± 1 deg. Both estimates satisfy the system requirement.

The lateral position estimate shown in Fig. 3 is drifting away from zero with time increased, which means the vehicle will eventually depart the center of the traveling lane with initial lateral position and angle offset imposed. The introduction of a LQR controller would address this issue after the estimator is built.

4. LQR CONTROLLER

To develop a LQR controller, a linearized dynamics model is needed. With small angle approximation, Eqs. (1-4) can be linearized as Eqs. (12) (Kosecka, 1996).

$$\begin{bmatrix} \dot{v}_y \\ \ddot{\psi} \\ \dot{y}_L \\ \dot{\varepsilon}_L \end{bmatrix} = \begin{bmatrix} -\frac{c_f + c_r}{m v_x} & -v_x + \frac{c_r l_r - c_f l_f}{m v_x} & 0 & 0 \\ \frac{-l_f c_f + l_r c_r}{I_\psi v_x} & -\frac{{l_f}^2 c_f + {l_r}^2 c_r}{I_\psi v_x} & 0 & 0 \\ -1 & -L_a & 0 & v_x \\ 0 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_y \\ \dot{\psi} \\ y_L \\ \varepsilon_L \end{bmatrix} + \begin{bmatrix} \frac{c_f}{m} \\ \frac{l_f c_f}{I_\psi} \\ 0 \\ 0 \end{bmatrix} \delta_f + \begin{bmatrix} 0 \\ 0 \\ 0 \\ v_x \end{bmatrix} K_L \quad (12)$$

Meanwhile the largest tolerable state errors and control input are defined in Eqs (13-14). Based on realistic sinerio, the lateral velocity error should be no larger than 1.5 m/s. Otherwise, the vehicle would enter neighbor lanes causing danger. The largest tolerable errors for other three states are set to be the same values in R matrix in Kalman filter since these states are observed directly from the sensors. The largest control input offset, which is the steering angle error, should be no larger than 5 degrees at an instantaneous operation.

$$q = \begin{bmatrix} (1.5)^{-2} & 0 & 0 & 0 \\ 0 & \left(\frac{10 * \pi}{180}\right)^{-2} & 0 & 0 \\ 0 & 0 & 0.3^{-2} & 0 \\ 0 & 0 & 0 & \left(\frac{3 * \pi}{180}\right)^{-2} \end{bmatrix} \quad (13)$$

$$r = \left[\left(\frac{5 * \pi}{180}\right)^{-2}\right] \quad (14)$$

$$K = [0.0287 \quad 0.5483 \quad -0.2909 \quad -1.3474] \quad (15)$$

The calculated LQR control gain is in Eqn. (15). With the same initialized states, the observation estimates, the true value, and the sensed measurements after applying LQR control are compared in Fig. 5. The errors between the observation estimates and the true values after applying LQR control are shown in Fig. 6.

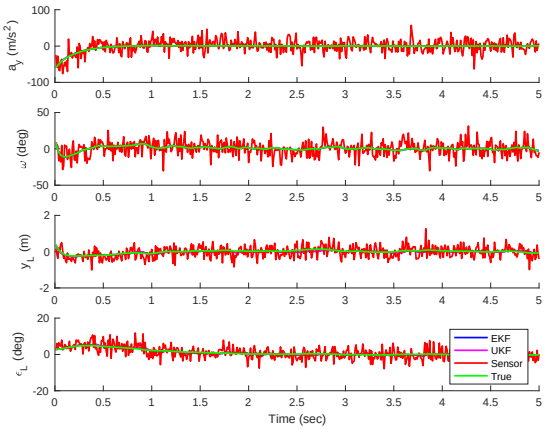

Fig. 5 Estimate, true and sensed observations after applying LQR control

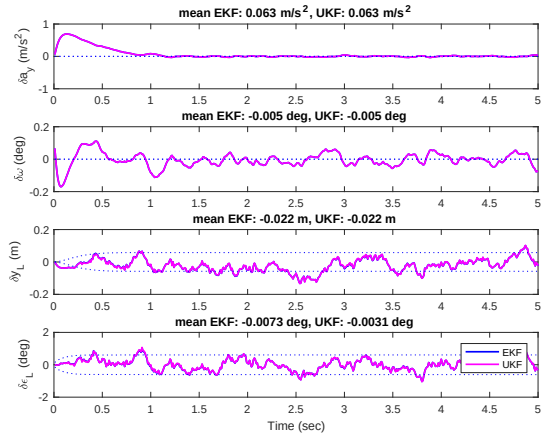

Fig. 6 Error between estimate and true observations after applying LQR control

With LQR controller introduced, the lateral position offset can be pulled back to zero when the vehicle is initialized

with 0.5 m lateral offset. The settling time is far less than 1 second, which satisfy our control design goal. However, it may be more comfortable for the driver if the settling time is larger around 1 second. That would be investigated in the future work.

5. CONCLUSIONS

This project investigates the design of estimators and controller for automated lane keeping system. Under the same initial conditions, the UKF and EKF estimators have similar precision performance regarding our specific requirement about the maximum lateral position and angular offsets. The designed LQR controller can pull the vehicle back when it is imposed with initial lateral position and angle offsets.

REFERENCES

Kosecka, Jana. "Vision-Based Lateral Control of Vehicles: Look-ahead and Delay Issues." (1996).

Kosecka, J., et al. "Vision-based lateral control of vehicles." Intelligent Transportation System, 1997. ITSC'97., IEEE Conference on. IEEE, 1997.

Cerone, Vito, A. Chinu, and D. Regruto. "Experimental results in vision-based lane keeping for highway vehicles." Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301). Vol. 2. IEEE, 2002.

APPENDIX

```
clear all
close all

%Given:
dt = 0.01;          %sec
% Qc = (1/249.55)^2;
Qc = (1/1000)^2;
R = diag([(1.7*9.8)^2 (10*pi/180)^2 0.3^2 (3*pi/180)^2]);

%Initial conditions –  perfect knowledge at time 0; radius extended to 2 m
x = [12 7*pi/180 0.5 3*pi/180]';
%xRef = x;
xHatPe = x;
xHatPl = x;
xHatPu = x;
PPe = zeros(4);
PPl = zeros(4);
PPu = zeros(4);

u = 0;

[K] = LQR;

%Simulate for 10 seconds
steps = 5/dt;
for k=1:steps                % purely discrete sim
   % Control
   t = k*dt;

   % Simulated Truth -- use very small time steps
   % to simulate continuous system
   stepsPerSample = 50;
   dtSub = dt/stepsPerSample;
   for sub_k = 1:stepsPerSample
      % Continuous Noise
      wc = sqrtm(Qc)*randn(1,1);
      % Propagation matrices
      [f,A,B,Btil] = NL_prop(x,u,wc);
      x = x + f*dtSub;
   end
   v = sqrtm(R)*randn(4,1);
   [y,C,D] = NL_sense(x,u,v);

   % EKF step
   [xHatPe,PPe]=EKF_step(y,u,xHatPe,PPe,Qc,R,dt);

   % LKF step
   %
[xHatPl,PPl]=LKF_step(y,u,xHatPl,xRef,PPl,Qc,R,dt);

   % UKF step
   [xHatPu,PPu]=UKF_step(y,u,xHatPu,PPu,Qc,R,dt);

   % For comparison (monitoring) and for LKF - Obtain reference Trajectory
   % [f,A,B,Btil] = NL_prop(xRef,u,0);
   % [yRef,C] = NL_sense(xRef,u,zeros(4,1));
   % xRef = xRef + f*dt;

   % Data Storage
   yStore(k,1:4)=y';
   xStore(k,1:4)=x';
   % xRefStore(k,1:4)=xRef';
   xHatEStore(k,1:4)=xHatPe';
   % xHatLStore(k,1:4)=xHatPl';
   xHatUStore(k,1:4)=xHatPu';
   PcartE = C*PPe*C';                % Polar to Cartesian transform
   % PcartL = C*PPl*C';
   PcartU = C*PPu*C';
   sigEStore(k,1:4)=sqrt(diag(PcartE));      % Store (x,y) standard dev.
   % sigLStore(k,1:2)=sqrt(PcartL([1 4]));
   sigUStore(k,1:4)=real(sqrt(diag(PcartU)));

   u = -K*xHatPe;
end

% Plot position information
multiKF_plot_lk;
```

LKF_step.m

```
function [xHatP,PP]=EKF_step(y,u,xHatP,PP,Qc,R,dt)

   % Discretization (Forward Euler)
   [f,A,B,Btil] = NL_prop(xHatP,u,0);
   [yHat,C] = NL_sense(xHatP,u,zeros(4,1));
   F = eye(size(A))+A*dt;
   H = C;
   Qk = Btil*Qc*Btil'*dt;

   % Extended Kalman Filter
   xHatM = xHatP + f*dt;             % Forward Euler integration (A priori)
   PM = F*PP*F'+Qk;                  % cov. Estimate
         L = PM*H'*inv(H*PM*H'+R);
   LE = L
         xHatP = xHatM + L*(y-yHat);       %a posteriori estimate
         PP = (eye(size(F))-L*H)*PM*(eye(size(F))-L*H)'+L*R*L';

End
```

UKF_step.m

```
function [xHatP,PP]=UKF_step(y,u,xHatP,PP,Qc,R,dt)

   % Parameters
   numState = length(xHatP);
   numSense = length(y);

   % Discretization (Forward Euler)
   [f,A,B,Btil] = NL_prop(xHatP,u,zeros(2,1));
   Qk = Btil*Qc*Btil'*dt;

   % Unscented Kalman Filter
   % Prediction
```

```
   [sigMat,junk] = sqrtm(PP*numState);    % Including
"junk" as an output suppresses messages
   sigMat = real(sigMat);
   runSum = zeros(numState,1);
   runSumSq = zeros(numState,numState);
   for n=1:numState
      % Generate Sigma Points
      xHatF_p =  sigMat(:,n)+xHatP;
      xHatF_m = -sigMat(:,n)+xHatP;

      % Propagate Sigma Points
      f_p = NL_prop(xHatF_p,u,zeros(1,1));
      f_m = NL_prop(xHatF_m,u,zeros(1,1));
      xHatF_p = xHatF_p + f_p*dt;
      xHatF_m = xHatF_m + f_m*dt;

      % Accumulate
      runSum = runSum+xHatF_p+xHatF_m;
      runSumSq                                          =
runSumSq+xHatF_p*xHatF_p'+xHatF_m*xHatF_m';
   end
   xHatM = runSum/2/numState;
   PM = (runSumSq/2/numState-xHatM*xHatM')+Qk;
   %
   % Correction
   [sigMat,junk] = sqrtm(PM*numState);
%     Pn = PM*numState
%     eig(Pn)
   sigMat = real(sigMat);
   runSumY = zeros(numSense,1);
   runSumX = zeros(numState,1);
   runSumSqY = zeros(numSense,numSense);
   runSumXY = zeros(numState,numSense);
   for n=1:numState
      % Generate Sigma Points
      xHatH_p =  sigMat(:,n)+xHatM;
      xHatH_m = -sigMat(:,n)+xHatM;

      % Obtain Sensor Values
      yHatH_p = NL_sense(xHatH_p,u,zeros(4,1));
      yHatH_m = NL_sense(xHatH_m,u,zeros(4,1));

      % Accumulate
      runSumY = runSumY+yHatH_p+yHatH_m;
      runSumX = runSumX+xHatH_p+xHatH_m;
      runSumSqY                                         =
runSumSqY+yHatH_p*yHatH_p'+yHatH_m*yHatH_m';
      runSumXY                                          =
runSumXY+xHatH_p*yHatH_p'+xHatH_m*yHatH_m';
   end
   yHat = runSumY/2/numState;
   xHatH = runSumX/2/numState;
   Pxy = (runSumXY/2/numState-xHatH*yHat');
   Py = (runSumSqY/2/numState-yHat*yHat')+R;
   %
   L = Pxy*inv(Py);
   LU = L
         xHatP = xHatM + L*(y-yHat);       %a posteriori
estimate
         PP = PM - L*Py*L';
End

NL_sense.m
function [h,C,D] = NL_sense(x,u,v);

   cf = 2*60000;
   cr = 2*50000;
   m = 1573;
   lr = 1.530;
   lf = 1.137;
   vx = 25;
   Iphi = 2753;
   La = 15;

   % Unpack states
         vy=x(1); phiDot=x(2); yL=x(3); eL=x(4);

   % Nonlinear model
   h = [    - cf/m*atan((vy+lf*phiDot)/vx) - cr/m*atan((vy-
lr*phiDot)/vx) + cf/m*u + v(1); ...
         phiDot+v(2); ...
         yL+v(3); ...
         eL+v(4)    ];

   % Linear model (time-varying)
         C = [-(cf+cr)/(m*vx),          (cr*lr-cf*lf)/(m*vx),
0,  0;  ...
      0,                1,                0,  0;  ...
      0,                0,                1,  0;  ...
      0,                0,                0,  1];

   D = [cf/m; 0; 0; 0];

end

NL_prop.m
function [f,A,B,Btil] = NL_prop(x,u,w);

%    % Unpack states
%       rDot=x(1); r=x(2); thDot=x(3); th=x(4);
%
%    % Nonlinear model
%    f = [   thDot^2*r+u(1)+w(1);
%          rDot;
%          (-2*thDot*rDot+u(2)+w(2))/r;
%          thDot   ];
%
%    % Linear model (time-varying)
%    r2rdot = (2*thDot*rDot-u(2)-w(2))/r^2;
%       A = [     0                 thDot^2  2*thDot*r
         0;
%                 1                 0                 0
                  0;
%                 -2*thDot/r        r2rdot            -
2*rDot/r 0;
%                 0                 0                 1
                  0];
%       B = [1 0; 0 0; 0 1/r; 0 0];
%    Btil = B;
```

```
    cf = 2*60000;
    cr = 2*50000;
    m = 1573;
    lr = 1.530;
    lf = 1.137;
    vx = 25;
    Iphi = 2753;
    La = 15;

    % Unpack states
            vy=x(1); phiDot=x(2); yL=x(3); eL=x(4);

    % Nonlinear model
    f = [     -vx*phiDot - cf/m*atan((vy+lf*phiDot)/vx) -
cr/m*atan((vy-lr*phiDot)/vx) + cf/m*u; ...
            cf*lf/Iphi*(u-
atan((vy+lf*phiDot)/vx))+cr*lr/Iphi*atan((vy-
lr*phiDot)/vx);  ...
            vx*eL-vy-phiDot*La; ...
            vx*w-phiDot    ];

    % Linear model (time-varying)
             A  =  [-(cf+cr)/(m*vx),                  -vx+(cr*lr-
cf*lf)/(m*vx),     0, 0;  ...
          (cr*lr-cf*lf)/(Iphi*vx),  -(lf^2*cf+lr^2*cr)/(Iphi*vx),
0, 0;  ...
           -1,                -La,                  0, vx; ...
           0,                 -1,                  0, 0];
    B = [cf/m; lf*cf/Iphi; 0; 0];

    Btil = [0;0;0;vx];

end
```

multiKF_plot_lk.m:

```
% Convert all polar coordinates to Cartesian
cf = 2*60000;
cr = 2*50000;
m = 1573;
lr = 1.530;
lf = 1.137;
vx = 25;
Iphi = 2753;
La = 15;

u = 0;

tVec = [1:steps]*dt;
yTrueStore = [ - cf/m*atan((xStore(:,1)+lf*xStore(:,2))/vx)
- cr/m*atan((xStore(:,1)-lr*xStore(:,2))/vx) + cf/m*u ...
            xStore(:,2) ...
            xStore(:,3) ...
            xStore(:,4) ];
%               yRefStore               =               [               -
cf/m*atan((xRefStore(:,1)+lf*xRefStore(:,2))/vx)          -
cr/m*atan((xRefStore(:,1)-lr*xRefStore(:,2))/vx) + cf/m*u
...
%             xRefStore(:,2) ...
%             xRefStore(:,3) ...
%             xRefStore(:,4) ];
yHatEStore                  =                  [                  -
cf/m*atan((xHatEStore(:,1)+lf*xHatEStore(:,2))/vx)       -
cr/m*atan((xHatEStore(:,1)-lr*xHatEStore(:,2))/vx)       +
cf/m*u ...
            xHatEStore(:,2) ...
            xHatEStore(:,3) ...
            xHatEStore(:,4) ];
yHatLStore = yHatEStore;
yHatUStore                  =                  [                  -
cf/m*atan((xHatUStore(:,1)+lf*xHatUStore(:,2))/vx)       -
cr/m*atan((xHatUStore(:,1)-lr*xHatUStore(:,2))/vx)       +
cf/m*u ...
            xHatUStore(:,2) ...
            xHatUStore(:,3) ...
            xHatUStore(:,4) ];

% Plot state information
figure(1); clf;
subplot(4,1,1);
hold on; grid on;
p1=plot(tVec,yHatEStore(:,1),'b',tVec,yHatUStore(:,1),'m'
,tVec,yStore(:,1),'r',tVec,yTrueStore(:,1),'g');
ylabel('{a_y (m/s^2)}');
set(p1,'LineWidth',1.5);
subplot(4,1,2);
hold on; grid on;
p1=plot(tVec,yHatEStore(:,2)/pi*180,'b',tVec,yHatUStore
(:,2)/pi*180,'m',tVec,yStore(:,2)/pi*180,'r',tVec,yTrueStor
e(:,2)/pi*180,'g');
ylabel('{\omega (deg)}');
set(p1,'LineWidth',1.5);
subplot(4,1,3);
hold on; grid on;
p1=plot(tVec,yHatEStore(:,3),'b',tVec,yHatUStore(:,3),'m'
,tVec,yStore(:,3),'r',tVec,yTrueStore(:,3),'g');
ylabel('{y_L (m)}');
set(p1,'LineWidth',1.5);
subplot(4,1,4);
hold on; grid on;
p1=plot(tVec,yHatEStore(:,4)/pi*180,'b',tVec,yHatUStore
(:,4)/pi*180,'m',tVec,yStore(:,4)/pi*180,'r',tVec,yTrueStor
e(:,4)/pi*180,'g');
set(p1,'LineWidth',1.5);
ylabel('{{\epsilon}_L (deg)}');

xlabel('Time (sec)');

legend('EKF','UKF','Sensor','True');

% Plot measurements estimate error information
figure(2); clf;
subplot(4,1,1);
p1=plot(tVec,yHatEStore(:,1)-
yTrueStore(:,1),'b',tVec,yHatUStore(:,1)-
yTrueStore(:,1),'m');
hold on; grid on;
p2=plot(tVec,sigEStore(:,1),'b:');
p3=plot(tVec,-sigEStore(:,1),'b:');
```

```
meanVec = mean([yHatEStore(:,1)-yTrueStore(:,1) yHatUStore(:,1)-yTrueStore(:,1)],1);
title(['mean EKF: ' num2str(meanVec(1),2) ' {m/s^2}, UKF: ' num2str(meanVec(2),2) ' {m/s^2}']);
%stdVec = std([yHatEStore(:,1)-yTrueStore(:,1) yHatLStore(:,1)-yTrueStore(:,1) yHatUStore(:,1)-yTrueStore(:,1)],[],1);
%title(['one-sigma EKF: ' num2str(stdVec(1),2) ' m, LKF: ' num2str(stdVec(2),2) ' m, UKF: ' num2str(stdVec(3),2) ' m']);
ylabel('{\delta}a_y (m/s^2)}');
set(p1,'LineWidth',1.5);
subplot(4,1,2);
p1=plot(tVec,(yHatEStore(:,2)-yTrueStore(:,2))/pi*180,'b',tVec,(yHatUStore(:,2)-yTrueStore(:,2))/pi*180,'m');
hold on; grid on;
p2=plot(tVec,sigEStore(:,2)/pi*180,'b:');
p3=plot(tVec,-sigEStore(:,2)/pi*180,'b:');

meanVec = mean([yHatEStore(:,2)-yTrueStore(:,2) yHatUStore(:,2)-yTrueStore(:,2)],1)/pi*180;
title(['mean EKF: ' num2str(meanVec(1),2) ' deg, UKF: ' num2str(meanVec(2),2) ' deg']);
%stdVec = std([yHatEStore(:,2)-yTrueStore(:,2) yHatLStore(:,2)-yTrueStore(:,2) yHatUStore(:,2)-yTrueStore(:,2)],[],1);
%title(['one-sigma EKF: ' num2str(stdVec(1),2) ' m, LKF: ' num2str(stdVec(2),2) ' m, UKF: ' num2str(stdVec(3),2) ' m']);
ylabel('{\delta}\omega (deg)}');
set(p1,'LineWidth',1.5);

subplot(4,1,3);
p1=plot(tVec,yHatEStore(:,3)-yTrueStore(:,3),'b',tVec,yHatUStore(:,3)-yTrueStore(:,3),'m');
hold on; grid on;
p2=plot(tVec,sigEStore(:,3),'b:');
p3=plot(tVec,-sigEStore(:,3),'b:');
meanVec = mean([yHatEStore(:,3)-yTrueStore(:,3) yHatUStore(:,3)-yTrueStore(:,3)],1);
title(['mean EKF: ' num2str(meanVec(1),2) ' m, UKF: ' num2str(meanVec(2),2) ' m']);
%stdVec = std([yHatEStore(:,1)-yTrueStore(:,1) yHatLStore(:,1)-yTrueStore(:,1) yHatUStore(:,1)-yTrueStore(:,1)],[],1);
%title(['one-sigma EKF: ' num2str(stdVec(1),2) ' m, LKF: ' num2str(stdVec(2),2) ' m, UKF: ' num2str(stdVec(3),2) ' m']);
ylabel('{\delta}y_L (m)}');
set(p1,'LineWidth',1.5);
subplot(4,1,4);
p1=plot(tVec,(yHatEStore(:,4)-yTrueStore(:,4))/pi*180,'b',tVec,(yHatUStore(:,4)-yTrueStore(:,4))/pi*180,'m');
hold on; grid on;
p2=plot(tVec,sigEStore(:,4)/pi*180,'b:');
p3=plot(tVec,-sigEStore(:,4)/pi*180,'b:');
meanVec = mean([yHatEStore(:,4)-yTrueStore(:,4) yHatUStore(:,4)-yTrueStore(:,4)],1)/pi*180;
title(['mean EKF: ' num2str(meanVec(1),2) ' deg, UKF: ' num2str(meanVec(2),2) ' deg']);
%stdVec = std([yHatEStore(:,2)-yTrueStore(:,2) yHatLStore(:,2)-yTrueStore(:,2) yHatUStore(:,2)-yTrueStore(:,2)],[],1);
%title(['one-sigma EKF: ' num2str(stdVec(1),2) ' m, LKF: ' num2str(stdVec(2),2) ' m, UKF: ' num2str(stdVec(3),2) ' m']);
ylabel('{\delta}{\epsilon}_L (deg)}');
xlabel('Time (sec)');
set(p1,'LineWidth',1.5);

legend('EKF','UKF');
```