```
function [x, P] = nonLinKFupdate(x, P, y, h, R, type)
%NONLINKFUPDATE calculates mean and covariance of predicted state
%   density using a non-linear Gaussian model.
%
%Input:
%   x           [n x 1] Prior mean
%   P           [n x n] Prior covariance
%   y           [m x 1] measurement vector
%   h           Measurement model function handle
%               [hx,Hx]=h(x)
%               Takes as input x (state),
%               Returns hx and Hx, measurement model and Jacobian
 evaluated at x
%               Function must include all model parameters for the
 particular model,
%               such as sensor position for some models.
%   R           [m x m] Measurement noise covariance
%   type        String that specifies the type of non-linear filter
%
%Output:
%   x           [n x 1] updated state mean
%   P           [n x n] updated state covariance
%
    [hx,Hx]=h(x);
    n=length(x);
    m=length(y);
    switch type
        case 'EKF'
            S=Hx*P*Hx'+R;
            K=P*Hx'*S^-1;

            x=x+K*(y-hx);
            P=P-K*S*K';
        case 'UKF'
            [SP,W]=sigmaPoints(x,P,'UKF');

            for i=0:2*n
                yp(:,i+1)=h(SP(:,i+1))*W(i+1);
            end
            yp=sum(yp,2);

            for i=0:2*n
                Pp(:,:,i+1)=(SP(:,i+1)-x)*(h(SP(:,i+1))-yp)'*W(i+1);
            end
            Pp=sum(Pp,3);

            for i=0:2*n
                Sp(:,:,i+1)=(h(SP(:,i+1))-yp)*(h(SP(:,i+1))-yp)'*W(i
+1);
            end
            Sp=sum(Sp,3)+R;
```

```
            x=x+Pp*inv(Sp)*(y-yp);
            P=P-Pp*inv(Sp)*Pp';

            % Make sure the covariance matrix is semi-definite
            if min(eig(P))<=0
                [v,e] = eig(P, 'vector');
                e(e<0) = 1e-4;
                P = v*diag(e)/v;
            end

        case 'CKF'
            [SP,W]=sigmaPoints(x,P,'CKF');

            for i=1:2*n
                yp(:,i)=h(SP(:,i))*W(i);
            end
            yp=sum(yp,2);

            for i=1:2*n
                Pp(:,:,i)=(SP(:,i)-x)*(h(SP(:,i))-yp)'*W(i);
            end
            Pp=sum(Pp,3);

            for i=1:2*n
                Sp(:,:,i)=(h(SP(:,i))-yp)*(h(SP(:,i))-yp)'*W(i);
            end
            Sp=sum(Sp,3)+R;

            x=x+Pp*Sp^-1*(y-yp);
            P=P-Pp*Sp^-1*Pp';

        otherwise
            error('Incorrect type of non-linear Kalman filter')
    end

end
```

Published with MATLAB® R2020a