Kalman Filter, Extended Kalman Filter, Unscented Kalman Filter

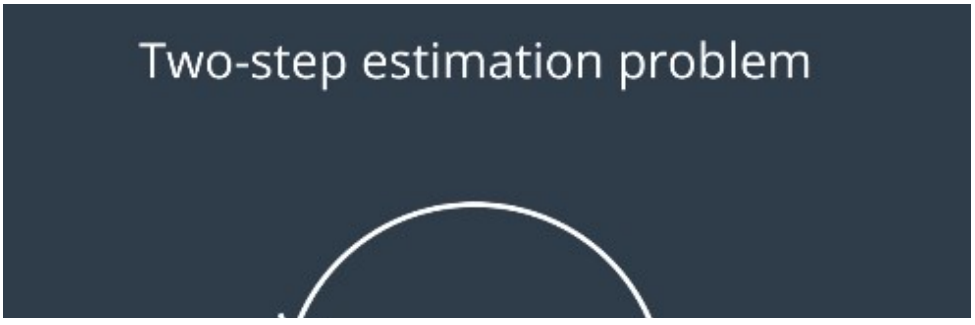
Alena Kastsiukavets Follow
May 25, 2017 · 6 min read

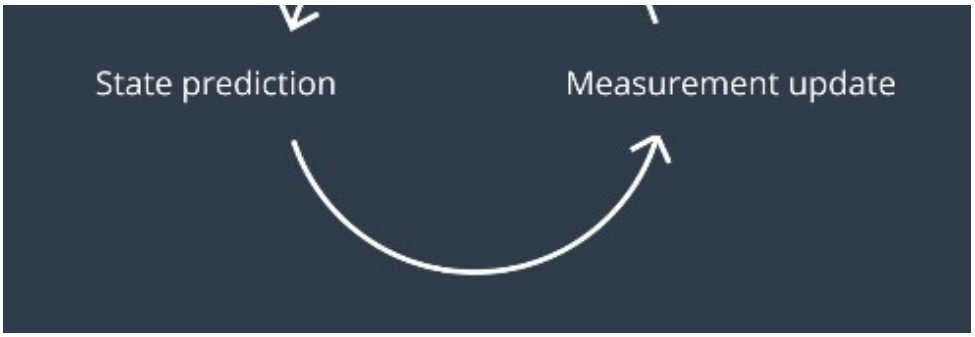
The second term of Self-Driving Car Engineer Nanodegree devotes Robotics. Therefore, the first two projects we spend on learning Kalman filter (KF) and its variations. We implemented three different versions of KF suitable for SDC and I decided to write and overview which describe key differences.

The first question is why we need KF at all. Why can't we rely on measurement we receive from our sensors? The answer is simple. We do not live in a perfect world, and we can not trust our sensors and measurements 100%. And KF gives us a way to combine measurements from different sensors (like LIDAR or RADAR) and mathematical model we built to predict out position. How does it do it? Basically, it finds a weighted sum of our measurements depending on how much we trust a particular sensor or our model.

Standard Kalman filter includes two steps:

1. Predict car's position based on our mathematical model
2. Update position based on data we get from the sensor





We repeat those two steps every time we get new data from the sensors. On each step we update our position **x** (state estimate) and the accuracy of our estimate **P** (error covariance matrix).

Predict step

I provide Wikipedia's equations since they are more intuitive regarding the time interval. For prediction step we need to update our position according to our mathematical model (*priori* state estimate):

$$x(k+1|k)=F(k+1)x(\boldsymbol{k|k})+u(k+1) \quad \textbf{(1)}$$

Here we took the value we got from the previous iteration and update it according to our mathematical model. Then we need to correct our belief about the accuracy of our measurements taking into account process noise Q:

$$P(k+1|k)=F(k+1)P(\boldsymbol{k|k})F`(k+1)+Q(k+1) \quad \textbf{(2)}$$

Update step

For update step, we need to update our estimate according to the measurement we got from the sensor. First of all we need to transform our state vector **x** to the same space as our measurement **z**: H(k+1)*x(k+1|k). Then we find the difference between our estimate and the measurement from the sensor:

$$y(k+1) = z(k+1)\text{-}H(k+1)x(\boldsymbol{k+1|k}) \quad \textbf{(3)}$$

After this we calculate Kalman gain **K** based on covariance matrix from the Prediction step:

$$S(k+1) = H(k+1)P(\boldsymbol{k+1|k})*H`(k+1)+R(k+1)$$ **(4)**

$$K(k+1)=P(\boldsymbol{k+1|k})*H`(k+1)S\hat{}(-1)(k+1)$$ **(5)**

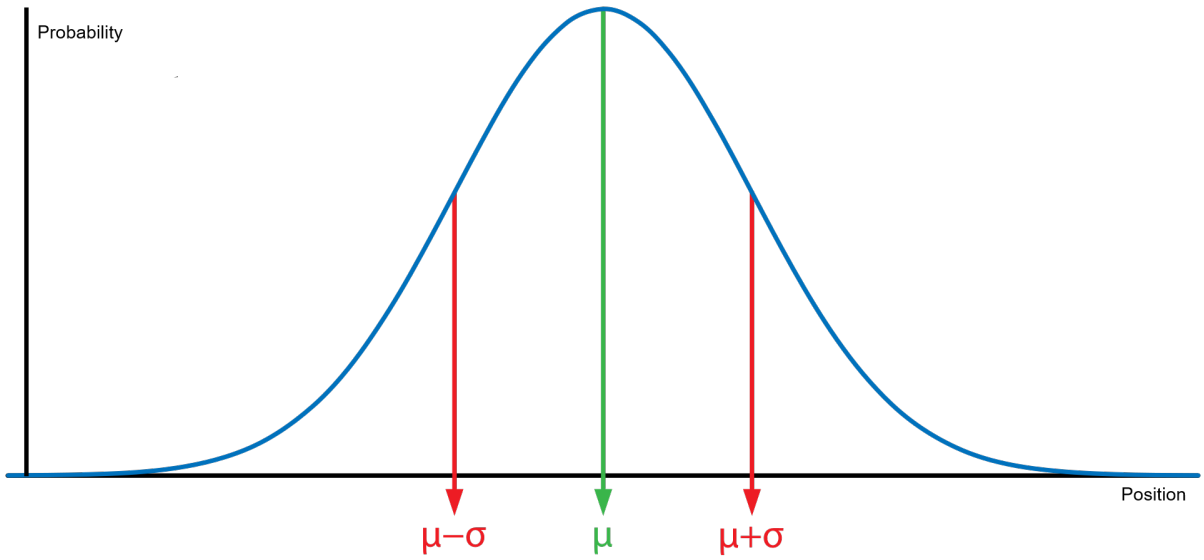
And now the time to update our position (remember, we just find a weighted sum according to our belief what we trust more our math model or sensor measurement) and accuracy according to the measurement :

$$x(k+1|k+1)=x(\boldsymbol{k+1|k})+K(k+1)y(k+1)$$ **(6)**

$$P(k+1|k+1)=(I-K(k+1)H(k+1))P(\boldsymbol{k+1|k})$$ **(7)**

So far, so good. Why do we need Extended Kalman Filter (*EKF*) or Unscented Kalman Filter (*UKF*) then?

The primary assumption we have for KF is that our position has a Gaussian(normal) distribution. It means that if we expect to see our car at position **μ**, it has the highest probability to be at point **μ** and the farther from **μ**, the smaller probability of seeing our car at that point:



Gaussian distribution has one very useful property:

The family of normal distributions is closed under linear transformations: if X *is normally distributed with mean* μ *and standard deviation* σ*, then the variable* Y = aX + b*, for any real numbers* a *and* b*, is also normally distributed, with mean* aμ + b *and standard deviation* |a|σ.

Also if X1 *and* X2 *are two independent normal random variables, with means* μ1*,* μ2 *and standard deviations* σ1*,* σ2*, then their sum* X1 + X2 *will also be normally distributed, with mean* μ1 + μ2 *and variance* $\sigma 1^2 + \sigma 2^2$.

More generally, any linear combination of independent normal deviates is a normal deviate. © Wikipedia

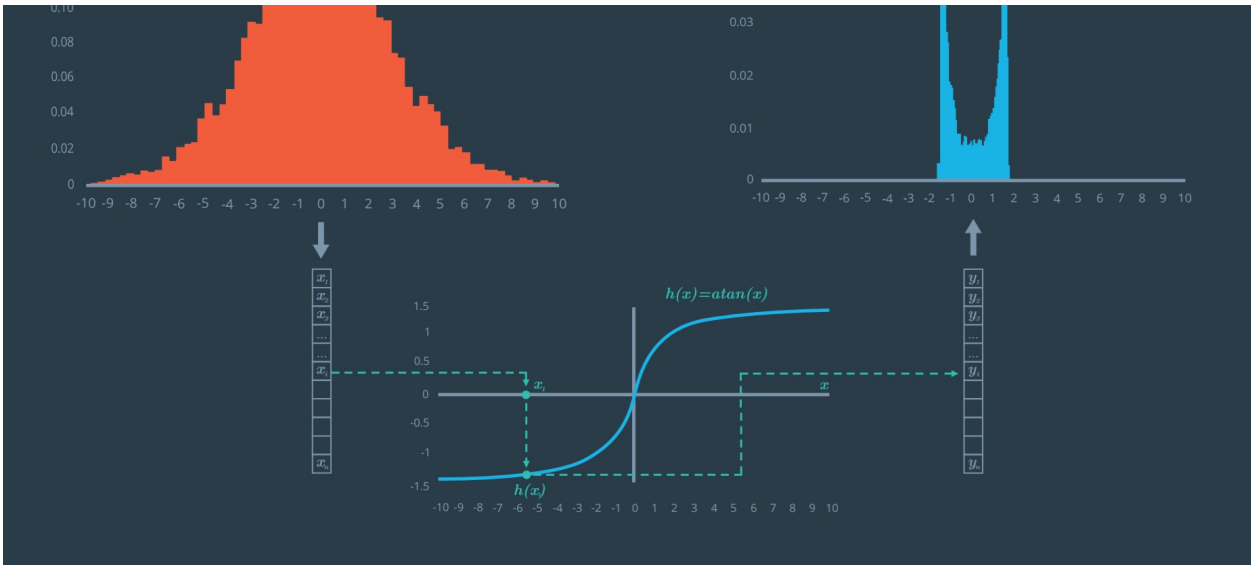
It means that KF equations must be linear. Sometimes it is true (LIDAR measurements), but sometimes it is not (RADAR measurements).

RADAR provides measurements in polar coordinates and our model works with cartesian coordinates. As a result, to calculate **y** we have to convert from cartesian coordinates to polar:

$$h(x') = \begin{pmatrix} \rho \\ \phi \\ \dot{\rho} \end{pmatrix} = \begin{pmatrix} \sqrt{p'^2_x + p'^2_y} \\ \arctan(p'_y / p'_x) \\ \frac{p'_x v'_x + p'_y v'_y}{\sqrt{p'^2_x + p'^2_y}} \end{pmatrix}$$

Hence for RADAR y(k+1) = z(k+1)-H(k+1)x(**k+1|k**) becomes y(k+1) = z(k+1)-h(x(**k+1|k**)).

This transformation is not linear. We loose our Gaussian distribution and can not use KF anymore.

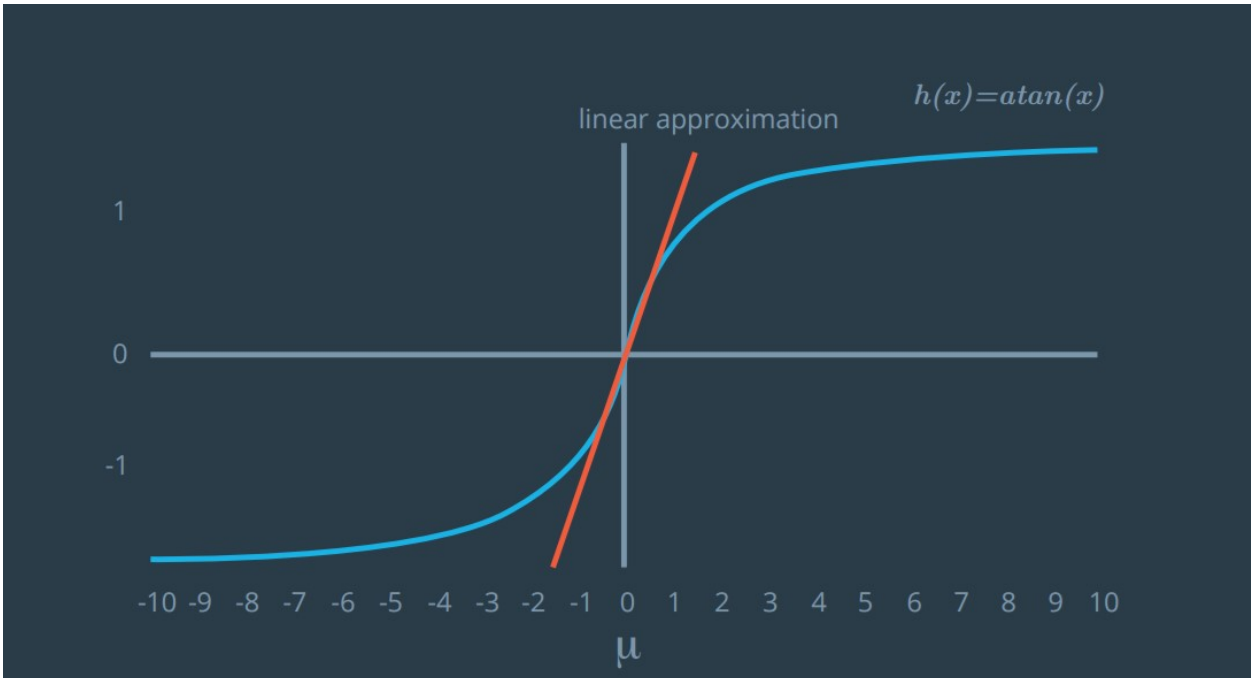


Gaussian distribution after nonlinear transformation

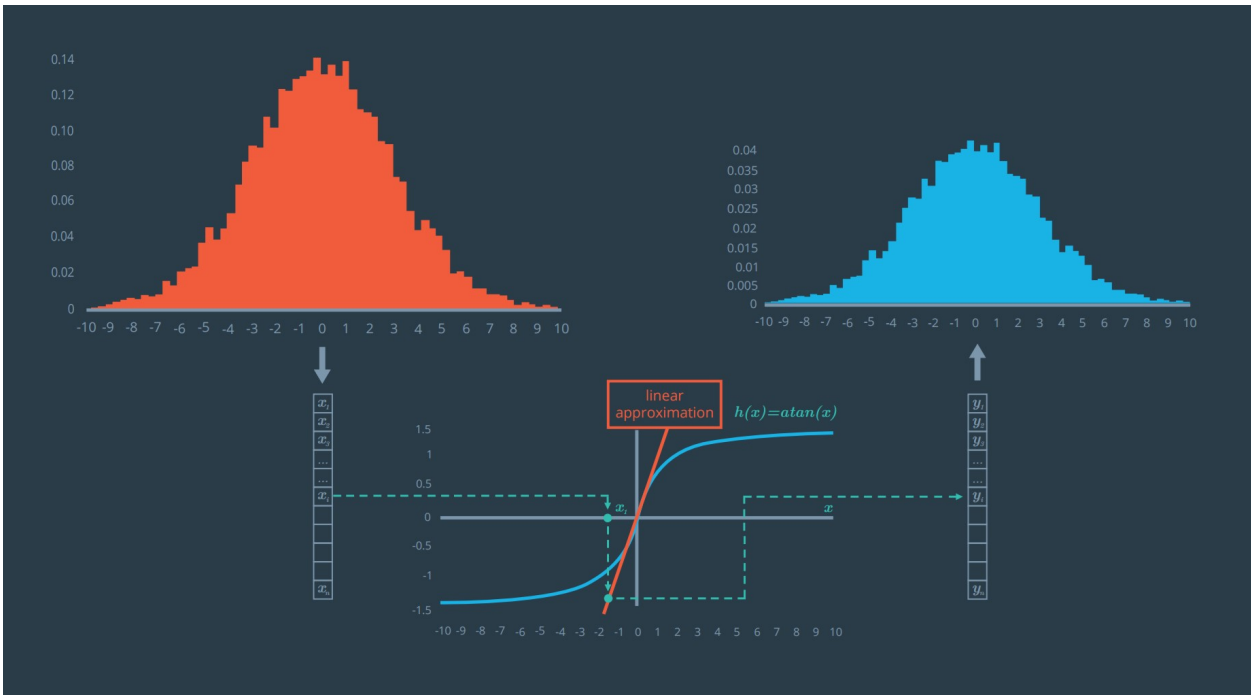
. . .

Extended Kalman Filter (EKF) proposes a solution to this problem. The EKF use Taylor expansion to construct a linear approximation of nonlinear function h(x):

$$h(x) \approx h(\mu) + \frac{\partial h(\mu)}{\partial x}(x - \mu)$$



First order Taylor expansion for **atan** at point **μ**



Gaussian distribution after applying a first order Taylor expansion

For our multi-dimensional case we use Taylor expansion up to Jacobian matrix. Resulting EKF algorithm looks almost identical to standard Kalman filter with two major differences. Firstly, we use a slightly changed equation for **y**:

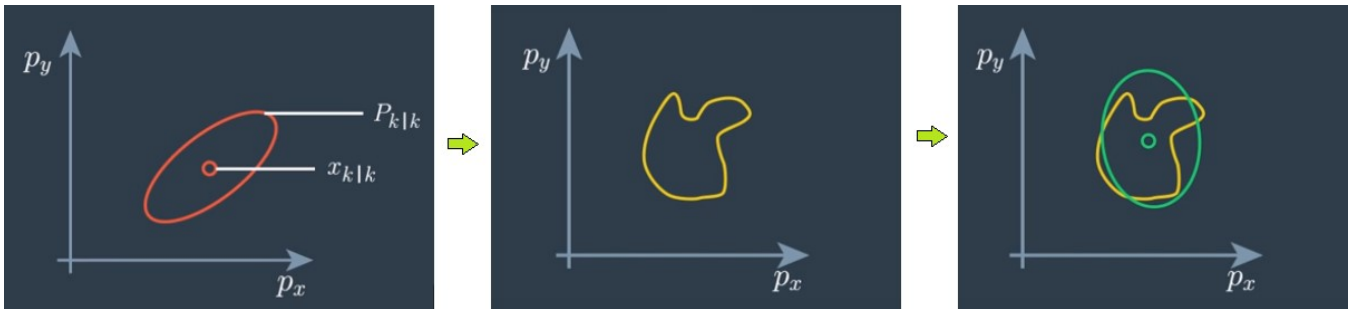
$$y(k) = z(k) - h(x(\mathbf{k+1|k})) \quad \mathbf{(3`)}$$

Secondly, instead of H(k+1) for **(4)**, **(5)** and **(7)** we use a Jacobian matrix Hj calculated at the point x(**k+1|k**), what is our best guess for the Gaussian mean.

. . .

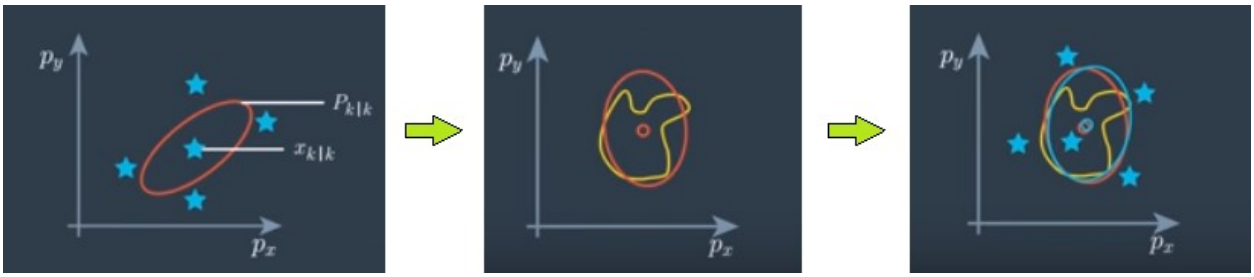
Unscented Kalman Filter (UKF) proposes a different solution. Instead of linearizing our transformation function we make an approximation one step later. Yes, we know that distribution, we get after nonlinear transformation, is not

Gaussian. Nevertheless, we pretend it is Gaussian and try to find the best approximation of real distribution.



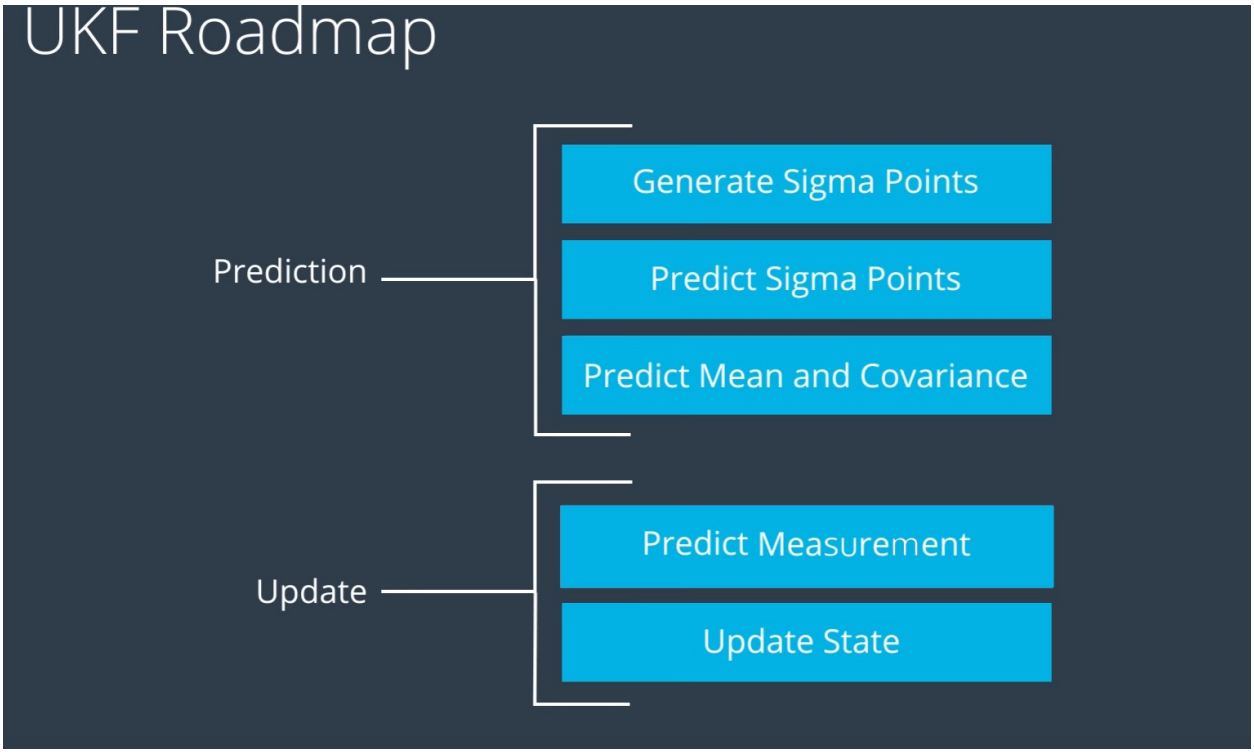
Unscented Kalman Filter approximation

The question is how to find this approximation. In the second project of the Nanodegree we use sigma points method for this purpose. The idea is to take several points around the expected position and apply our nonlinear transformation. As a result, we get points (**x_pred**) which are not normally distributed. Now our goal is to find a Gaussian distribution based on the transformed points. Ideally, we want to find the best approximation of Gaussian distribution. Sigma points method gives us the result pretty close to the expected one:



Sigma point method

For Update step we have to do a similar manipulation to predict $h(x(\mathbf{k+1|k}))$. It is important to note that information about x(k+1|k) is stored inside **x_pred**, so we can calculate $h(x(\mathbf{k+1|k}))$ from (3`) based on **x_pred** in the same way we predict **x**.



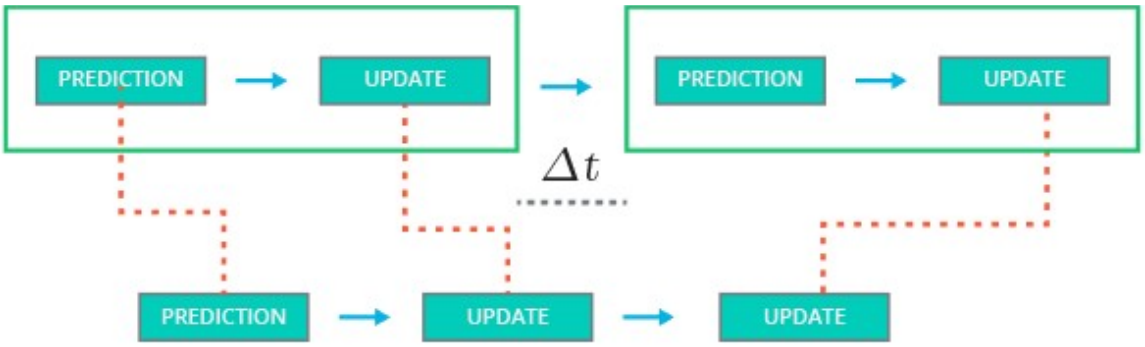
Predict and Update steps for UKF

The main difference between standard KF and UKF is the way we calculate Kalman gain **K**. For UKF we based **K** on cross-correlation between sigma points in state space and measurement space. But after a closer look, it is possible to notice that we try to achieve the same thing as for standard KF.

Although we can use UKF for linear transformation and get the same result, it is preferable to use standard KF, since UKF is more computationally-extensive technic.

. . .

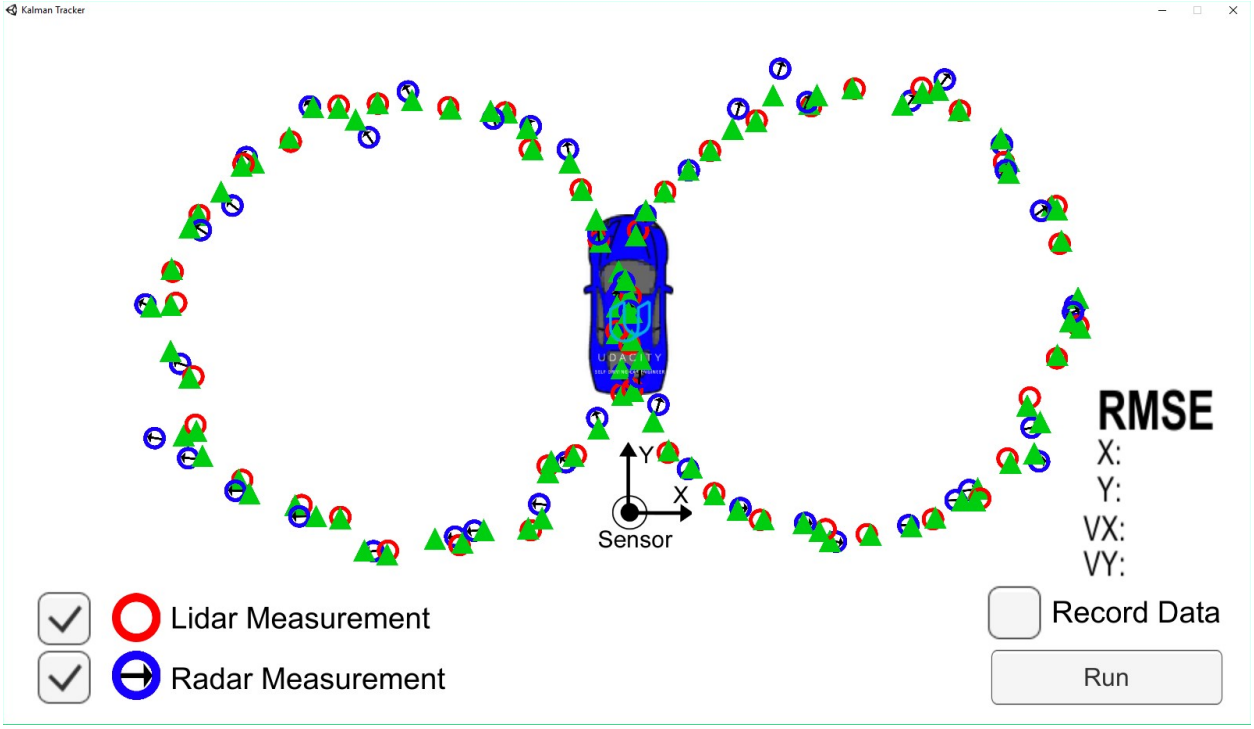
For any variation of KF we can use one simplification. Sometimes you get measurements data from your sensor almost simultaneously. In such situation, you can simply ignore the Prediction step for the second time and call Update step twice instead (the order does not matter).



However, with UKF you have to be careful. If you miss Prediction step, you have to generate sigma points for Update RADAR measurements step since they are outdated!

. . .

KF predicts position extremely well and it is the main reason it is so popular.



In a case of nonlinear transformation EKF gives good results, and for highly nonlinear transformation it is better to use UKF.