Sensor Fusion with Kalman Filter (2/2)

Using an Unscented Kalman Filter to fuse radar and lidar data for object tracking.

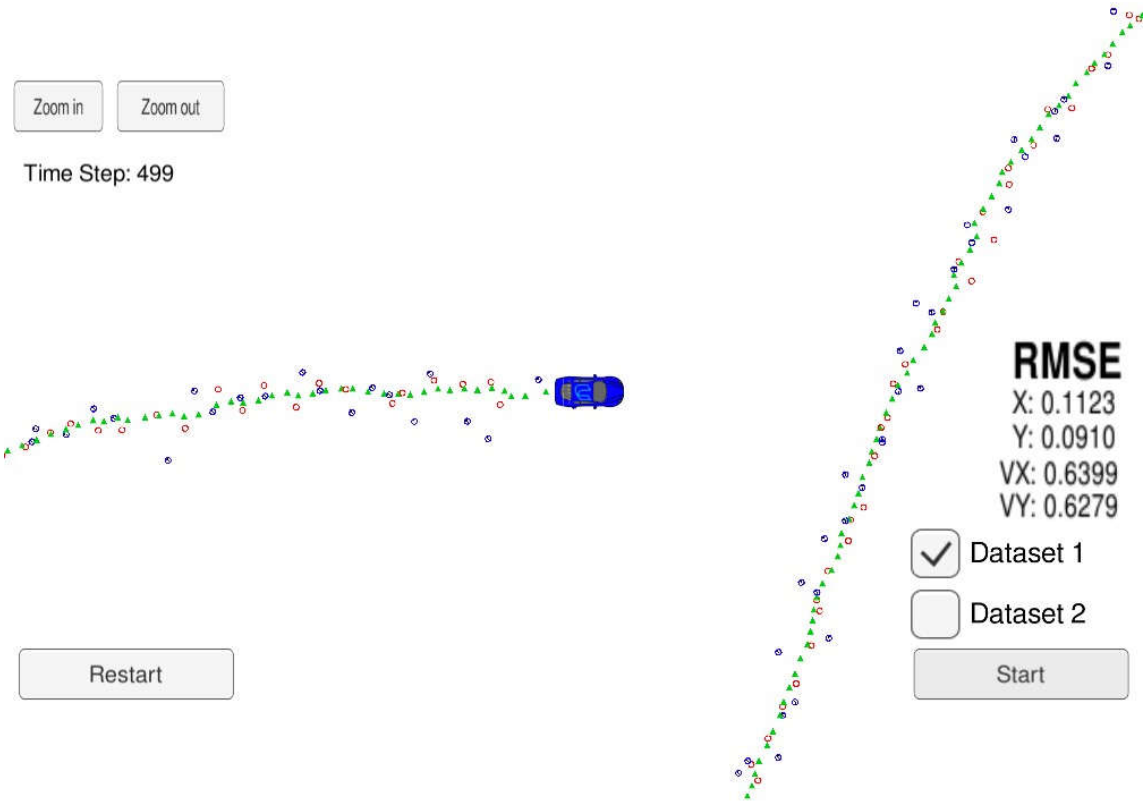
View on Github (https://winfriedauner.de/github_error/)

Task

The object and the setting is the same as in the previous EKF project (to fuse lidar and radar measurements in order to track a bicyclist), but this time a more advanced filter is used. This more advanced filter is called the Unscented Kalman Filter (UKF) (https://www.wikiwand.com/en/Unscented_transform). A good paper on the UKF can be found here (https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdf).

The key differences to the EKF project are:

- a more complex, non-linear motion model,
- the way non-linearities are dealt with.



Problem

What's wrong with the EKF, why the need for another KF? As discussed in the previous post the EKF handles non-linearities by applying a first-order linearization as an approximation. This can be good enough, but it is always an approximation and may fail in certain situations. The first project only needed this linearization for the non-linear mapping from state space to radar measurement space, since the motion model, represented by the dot product `F * x` was still linear. The UKF is able to handle non-linearities more accurate. It captures the posterior mean and covariance to the 3rd order Taylor series expansion for any non-linearity, whereas the EKF only achieves first order accuracy (Wan, Merwe (https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdf)).

State Vector & Motion Model

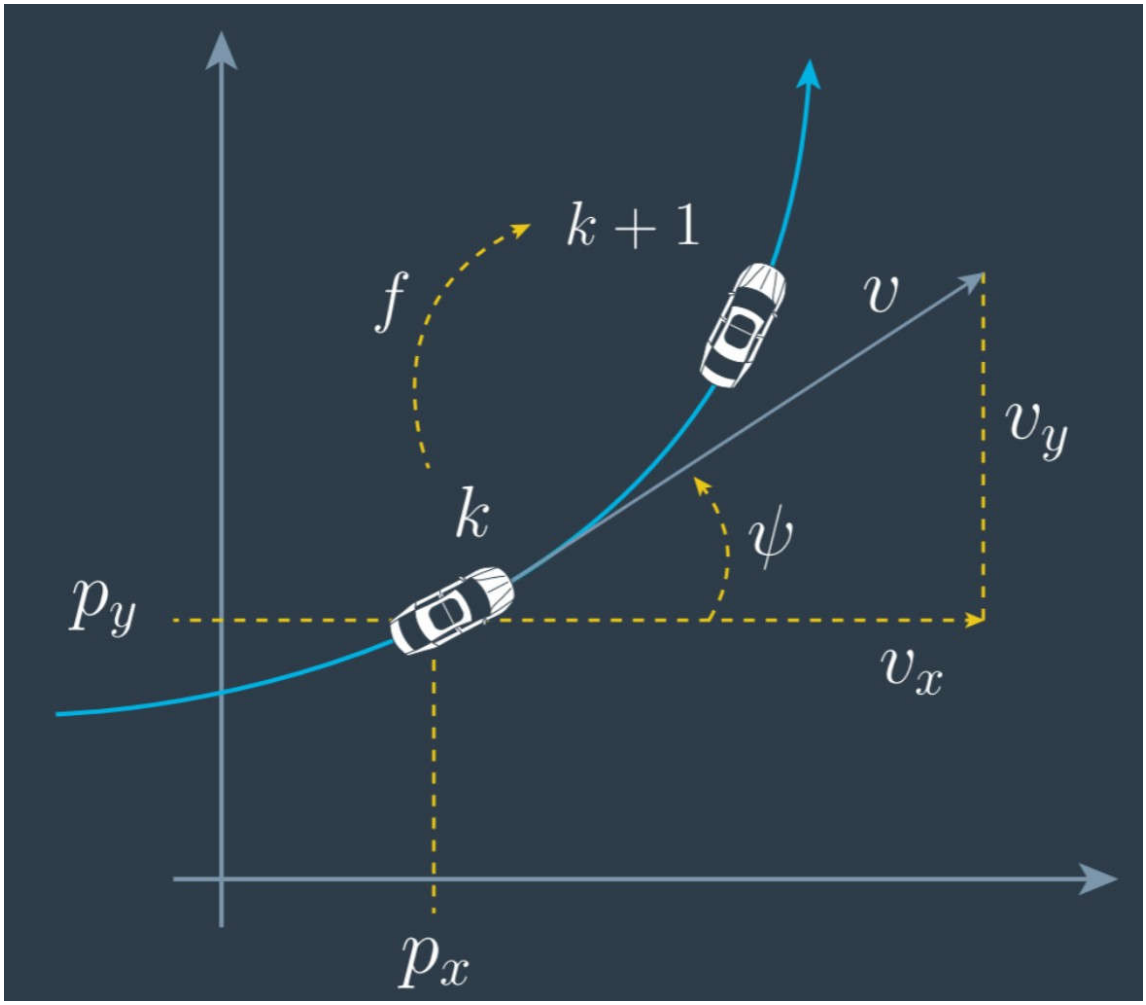
The state vector `x` in the EKF project consists of the four variables `[x, y, v_x, v_y]` and uses the very basic constant velocity (CV) motion model. In this project we'll use the constant turn rate and velocity magnitude model (CTRV). The new state vector contains the `x`, `y` position, the vehicle's velocity `v` as well as the yaw angle `psi` and its rate of change `psi_dot` (derivative).

$$\vec{x}_k = \begin{bmatrix} p_x \\ p_y \\ v \\ \psi \\ \dot{\psi} \end{bmatrix}$$

The state transition from timestep `k` to timestep `k+1` can then be defined as follows:

$$\vec{x}_{k+1} = \vec{x}_k + \int_{t_k}^{t_{k+1}} \begin{bmatrix} \dot{p}_x(t) \\ \dot{p}_y(t) \\ \dot{v}(t) \\ \dot{\psi}(t) \\ \ddot{\psi}(t) \end{bmatrix} dt + \vec{\nu}$$

where `nu` is the noise vector.



Source: Udacity's course lectures

The solution of the differential equation for the case `yaw rate != 0` is shown below:

$$\vec{x}_{k+1} = \vec{x}_k + \begin{bmatrix} \frac{v_k}{\dot{\psi}_k}\left(sin(\psi_k + \dot{\psi}_k \Delta t) - sin(\psi_k)\right) \\ \frac{v_k}{\dot{\psi}_k}\left(-cos(\psi_k + \dot{\psi}_k \Delta t) + cos(\psi_k)\right) \\ 0 \\ \dot{\psi}_k \Delta t \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 cos(\psi_k) \cdot \nu_{a,k} \\ \frac{1}{2}\Delta t^2 sin(\psi_k) \cdot \nu_{a,k} \\ \Delta t \cdot \nu_{a,k} \\ \frac{1}{2}\Delta t^2 \cdot \nu_{\ddot{\psi},k} \\ \Delta t \cdot \nu_{\ddot{\psi},k} \end{bmatrix}$$

To keep this article short, I'll explain the derivation of the solution in an extra article.

View derivation

Algorithm

The key idea of the UKF is to use so-called Sigma points to map distributions through a non-linear function instead of approximating this non-linear function with a linear one. So the UKF creates some Sigma points, maps them through the non-linear transition function and then tries to reconstruct what the original distribution would look like in the new space. The UKF can be structured into the following steps which are then repeated:

Prediction:

- Generate Sigma points
- Predict Sigma points
- Predict state vector & covariance

Update:

`If measurement model is non-linear:`

- Map predicted Sigma points to measurement space
- Update state and covariance

`Else if measurement model is linear:`

- normal KF update

Before we start there is a trick...

UKF Augmentation

The process model is defined as

$$\vec{x}_{k+1} = f(\vec{x}_k, \vec{\nu}_k), \quad \text{with } \vec{\nu}_k = \begin{bmatrix} \nu_{a,k} \\ \nu_{\ddot{\psi},k} \end{bmatrix}$$

which means that it depends on the state vector as well as on the noise vector. But the noise vector also has a non-linear effect
😱

. Luckily the UKF is pretty
😎
about this. We just have to augment our state vector with the noise variables, which are both zero (white noise and stuff...). This also means that the process covariance matrix gets expanded. The new state and covariance matrix looks like:

$$\vec{x}_{a,k} = \begin{bmatrix} p_x \\ p_y \\ v \\ \psi \\ \dot{\psi} \\ \nu_a \\ \nu_{\ddot{\psi}} \end{bmatrix}, \quad P_{a,k|k} = \begin{bmatrix} P_{k|k} & 0 \\ 0 & Q \end{bmatrix}$$

Prediction Steps

1 Generate Sigma points

The Sigma points are sampled from specific locations of the distribution. Together they form a matrix of dimensions `n_a  n_si ma + 1`, where `n_a` is the number of state variables (after augmentation) and `n_si ma` the number of Sigma points. It is defined as follows:

$$X_{\sigma,k|k} = \begin{bmatrix} \vec{x}_{k|k,1} & \cdots & \vec{x}_{k|k,n_a} \\ \vec{x}_{k|k,1} + \sqrt{(\lambda+n_a)P_{k|k,1,1}} & \cdots & \vec{x}_{k|k,n_a} + \sqrt{(\lambda+n_a)P_{k|k,n_a,1}} \\ \vdots & \vdots & \vdots \\ \vec{x}_{k|k,1} + \sqrt{(\lambda+n_a)P_{k|k,1,n_a}} & \cdots & \vec{x}_{k|k,n_a} + \sqrt{(\lambda+n_a)P_{k|k,n_a,n_a}} \\ \vec{x}_{k|k,1} - \sqrt{(\lambda+n_a)P_{k|k,1,1}} & \cdots & \vec{x}_{k|k,n_a} - \sqrt{(\lambda+n_a)P_{k|k,n_a,1}} \\ \vdots & \vdots & \vdots \\ \vec{x}_{k|k,1} - \sqrt{(\lambda+n_a)P_{k|k,1,n_a}} & \cdots & \vec{x}_{k|k,n_a} - \sqrt{(\lambda+n_a)P_{k|k,n_a,n_a}} \end{bmatrix}^T$$

To fit the image I had to use the transpose here. The original equation can be shown with right click on the graphic below.

$$X_{\sigma,k|k} = \begin{bmatrix} \vec{x}_{k|k,1} & \vec{x}_{k|k,1} + \sqrt{(\lambda + n_a)P_{k|k,1,1}} & \dots & \vec{x}_{k|k,1} + \sqrt{(\lambda + n_a)P_{k|k,1,n_a}} & \vec{x}_{k|k,1} - \sqrt{(\lambda + n_a)P_{k|k,1,1}} & \dots & \vec{x}_{k|k,1} - \sqrt{(\lambda + n_a)P_{k|k,1,n_a}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vec{x}_{k|k,n_a} & \vec{x}_{k|k,n_a} + \sqrt{(\lambda + n_a)P_{k|k,n_a,1}} & \dots & \vec{x}_{k|k,n_a} + \sqrt{(\lambda + n_a)P_{k|k,n_a,n_a}} & \vec{x}_{k|k,n_a} - \sqrt{(\lambda + n_a)P_{k|k,n_a,1}} & \dots & \vec{x}_{k|k,n_a} - \sqrt{(\lambda + n_a)P_{k|k,n_a,n_a}} \end{bmatrix}$$

with the scaling parameter `lam da` set to:

$$\lambda = 3 - n_a$$

How many Sigma points to choose? This many:

$$n_\sigma = 2n_a + 1$$

In C++ I use the Eigen Library (https://eigen.tuxfamily.org/dox/group__QuickRefPage.html) which is also able to take the square root of a matrix using the Cholesky decomposition (http://www.wikiwand.com/en/Cholesky_decomposition). Each sigma point vector is the sum of the state vector and only one column of the covariance matrix. So we are basically stacking vectors horizontally and make a matrix sandwich. This may be more clear in code:

```
   rior state and  ovarian e
 e tor d x =  e tor d n_x
x          some values
Matrix d   = Matrix d n_x, n_x
           some values

Matrix d  si  = Matrix d n_x,   * n_x + 1
   al ulate s uare root of
Matrix d   =   llt   matrix

 si   ol 0  = x
for  int i = 0  i   n_x  ++i
   si   ol i + 1  = x + s rt lam da + n_x  *    ol i
   si   ol i + 1 + n_x  = x - s rt lam da + n_x  *    ol i
```

Back to overview

2 Predict Sigma points

Now every Sigma point (which really is a vector with 7 values) is mapped through the process model into the state space, defined by the 5 state variables. So the process model performs the mapping
$f : \mathbb{R}^7 \rightarrow \mathbb{R}^5$
. The process model is not only a space mapping, but also predicts where the points should be at the next timestep, based on the motion model. We apply the process model for every column of the augmented Sigma point matrix to predict a new state vector. The result is a matrix with the dimensions `n_x_ori  n_si ma + 1`. This gives us the a priori estimation for the Sigma points matrix. I'll drop the sigma from here on, too many indices...

$$f : X_{k|k} \rightarrow X_{k+1|k}$$

The process model makes predictions for every Sigma point (=column) in the Sigma point matrix. The result is the predicted Sigma point matrix in state space.

Back to overview

3 Predict state vector and covariance

Based on the predictions for the Sigma points, we are now able to predict what we're really after, namely the *real* state vector and its covariance matrix. The prediction for the state vector is the weighted sum of the predicted Sigma points, where the index `i` of the predicted Sigma point matrix denotes the specific column:

$$\vec{x}_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i X_{k+1|k,i}$$

The weights are defined as

$$w_i = \frac{\lambda}{\lambda + n_a}, \quad i = 1$$

$$w_i = \frac{1}{2(\lambda + n_a)}, \quad i = 2..n_\sigma$$

The predicted covariance matrix is the weighted square difference between each Sigma point and the predicted mean

$$P_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i (X_{k+1|k,i} - \vec{x}_{k+1|k})(X_{k+1|k,i} - \vec{x}_{k+1|k})^T$$

Back to overview

Update Steps

I only describe the case for a non-linear measurement model. This is the case for radar measurements, but not for lidar. The processing chain for lidar measurements from here on is the same as for the vanilla KF.

4 Map predicted Sigma points to measurement space

If the state to measurement space transform is non-linear, we could start the unscented transform again, based on our predicted mean and covariance matrix. But instead of generating new Sigma points, we're going green and recycle our predicted Sigma point matrix. The measurement model is described by

$$\vec{z}_{k+1|k} = h(\vec{x}_{k+1}) + \vec{\omega}_{k+1}$$

We can get around the augmentation step, because the noise vector `ome a` has a purely additive effect and we can set it to 0 if we compensate for that later, when we calculate the covariance in measurement space.

The transformation can be described as

$$Z_\sigma = Z_{k+1|k} = h(X_{k+1|k})$$

The mapping is performed by applying the measurement model on each column of the predicted Sigma point matrix. Since the predicted Sigma points live in the original state space with 5 dimensions and radar measurements contain 3 variables, this gives us
$f : \mathbb{R}^5 \rightarrow \mathbb{R}^3$

. The measurement transform for the whole predicted Sigma point matrix can then be described as
$h : \mathbb{R}^{5 \times n_\sigma} \rightarrow \mathbb{R}^{3 \times n_\sigma}$
.

The radar measurement vector is given by

$$\vec{z}_{k+1} = \begin{bmatrix} \rho \\ \varphi \\ \dot{\rho} \end{bmatrix}$$

and the mapping from state variables to measurement variables is defined with

$$\rho = \sqrt{p_x^2 + p_y^2}$$

$$\varphi = arctan(\frac{p_y}{p_x})$$

$$\dot{\rho} = \frac{v(p_x cos(\psi) + p_y sin(\psi))}{\sqrt{p_x^2 + p_y^2}}$$

With the Sigma point matrix transformed to measurement space, we can use it to calculate the predicted mean in measurement space the same way we did before in state space:

$$\vec{z}_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i Z_{k+1|k,i}$$

and also the covariance in measurement space, but now we compensate for the (linear) noise by also adding the sensor covariance matrix R.

$$S_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i (Z_{k+1|k,i} - \vec{z}_{k+1|k})(Z_{k+1|k,i} - \vec{z}_{k+1|k})^T + R$$

The sensor covariance matrix $\square$ contains the standard deviations for every measurement variable (assuming they are independent from each other) and is given by the sensor manufacturer.

$$R = E(w_k \cdot w_k^T) = \begin{bmatrix} \sigma_\rho^2 & 0 & 0 \\ 0 & \sigma_\varphi^2 & 0 \\ 0 & 0 & \sigma_{\dot{\rho}}^2 \end{bmatrix}$$

Almost done! What's left is the final update step
😅

.

Back to overview

5 Update state and covariance

Now we connect the uncertainty of the state and measurement space by calculating the cross-correlation matrix $\square$, which we need to calculate the Kalman Gain $\square$. Then we can finally perform the posterior update.

$$T_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i (X_{k+1|k,i} - \vec{x}_{k+1|k})(Z_{k+1|k,i} - \vec{z}_{k+1|k})^T$$

$$K_{k+1|k} = T_{k+1|k} S_{k+1|k}^{-1}$$

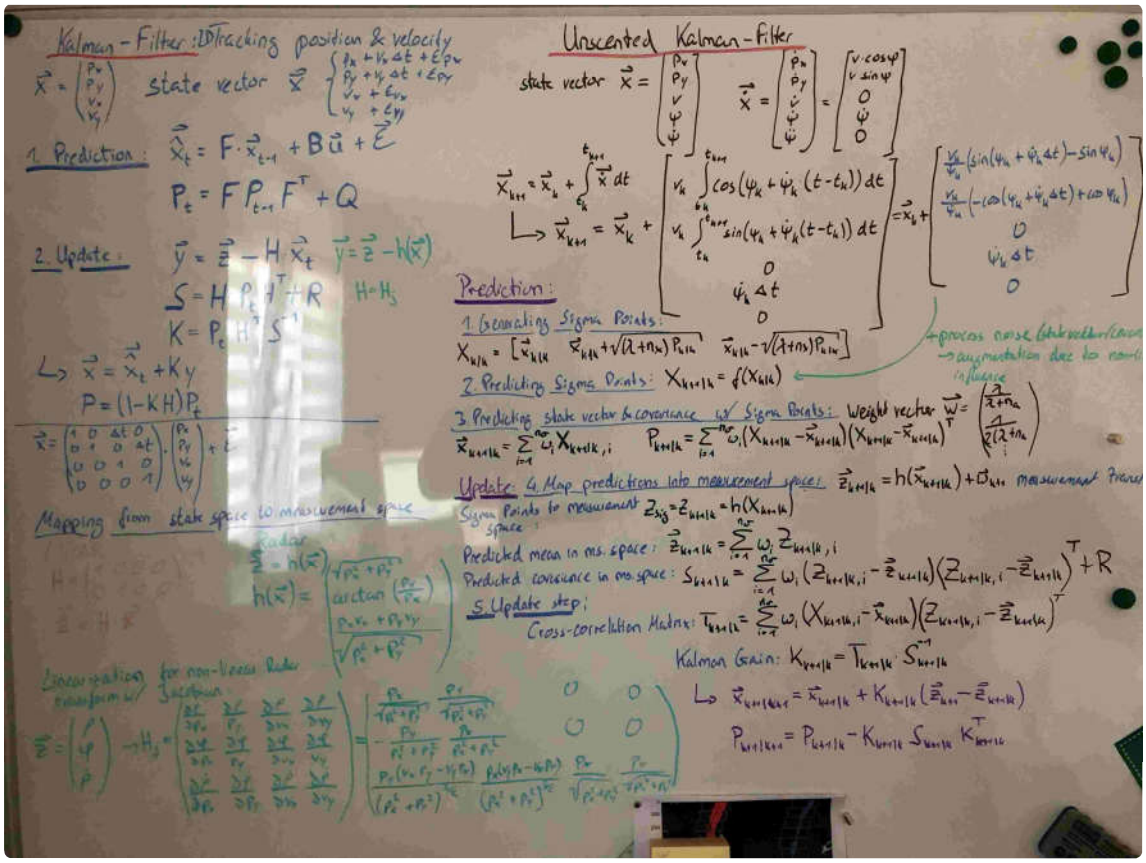
And now the drum roll please … here comes the final update for the state vector and its covariance matrix:

$$\vec{x}_{k+1|k+1} = \vec{x}_{k+1|k} + K_{k+1|k}(\vec{z}_{k+1} - \vec{z}_{k+1|k})$$

$$P_{k+1|k+1} = P_{k+1|k} - K_{k+1|k} S_{k+1|k} K_{k+1|k}^T$$

And this is why we can't have nice things…
😒
Now that I got that out of my head, I can finally ~~make my whiteboard white again~~

erase my whiteboard!



(https://winfriedauner.de/images/ukf/whiteboard_small.jpeg)
This helped me to keep myself sane during the project...

Back to overview

Repository

View on Github

Tags: C++ (https://winfriedauner.de/tags/#c) Sensor Fusion (https://winfriedauner.de/tags/#sensor-fusion)

Categories: Robotics (https://winfriedauner.de/categories/#robotics)