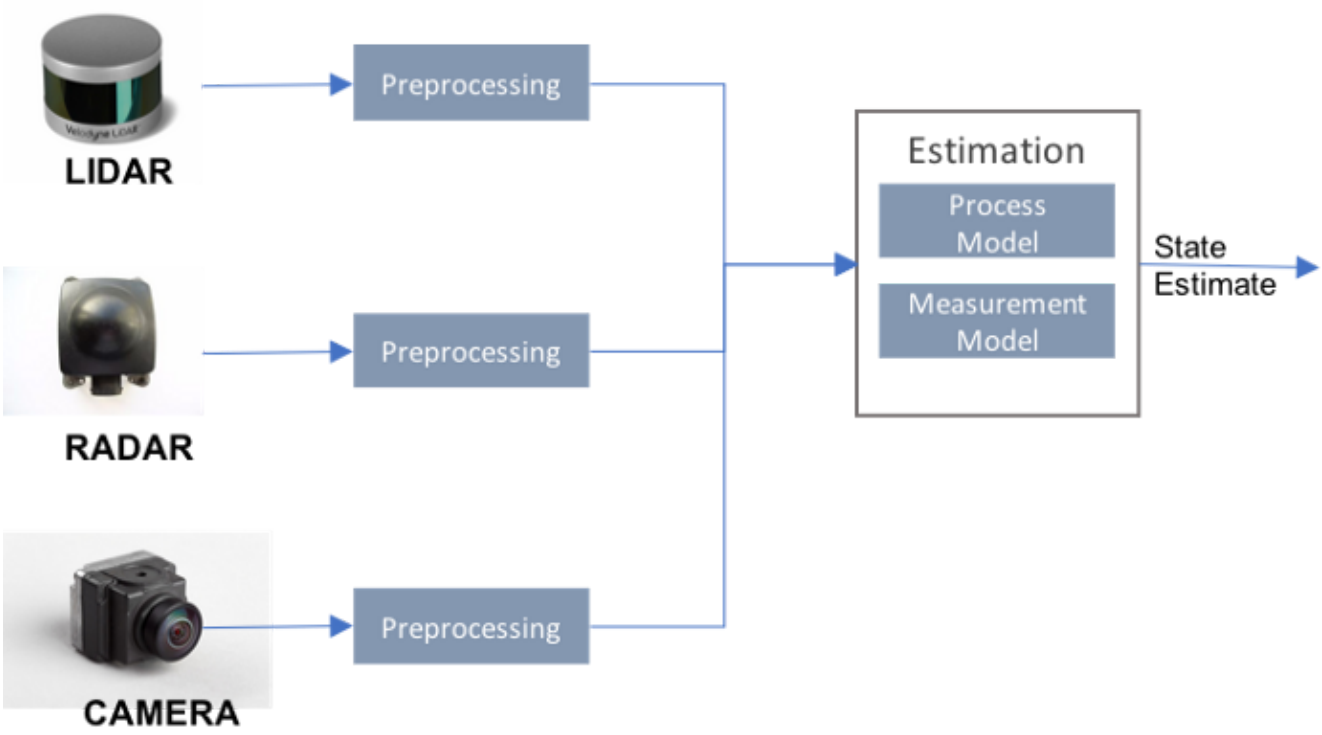
Wilbur de Souza · Follow
Jun 13, 2017 · 6 min read

Sensor Fusion Algorithms For Autonomous Driving: Part 1 — The Kalman filter and Extended Kalman Filter

Introduction

Tracking of stationary and moving objects is a critical function of Autonomous driving technologies. Signals from several sensors, including camera, radar and lidar (Light Detection and Ranging device based on pulsed laser) sensors are combined to estimate the position, veloctiy, trajectory and class of objects i.e. other vehicles and pedestrians. A good introduction to this topic can be found at: http://www.onenewspage.com/video/20161006/5695999/Mercedes-Benz-presents-the-Sensor-Fusion-at-2016.htm

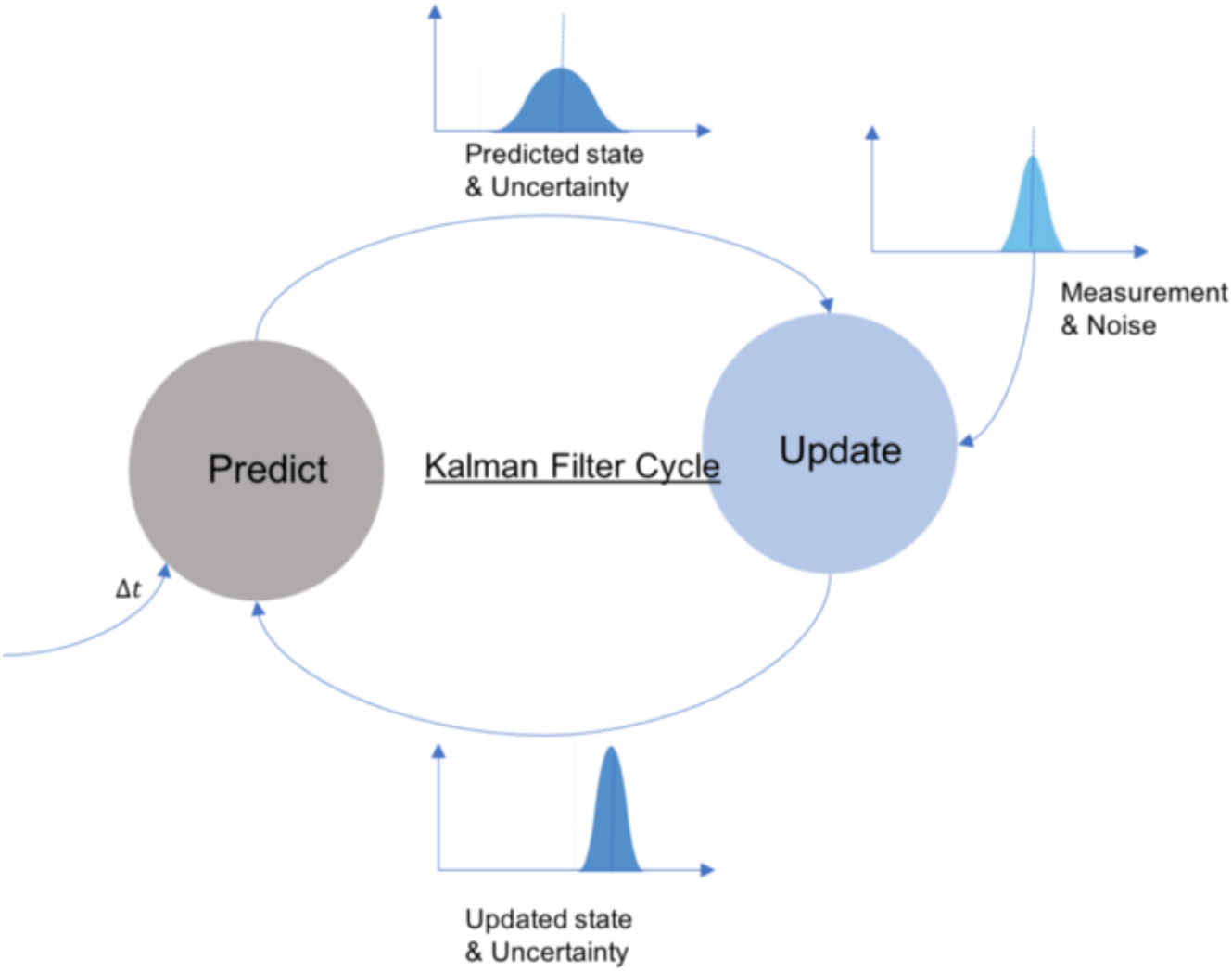
One may question — *why do we need several sensors?* This is because, each sensor provides different types of information about the tracked object position with differing acuracies especially in different weather conditions. For e.g. a lidar based sensor can provide good resolution about the position but can suffer for accuracy in poor weather. On the other hand, the spatial resolution of a radar sensor is relatively poor compared to laser but provides better accuracy in poor weather. Also, unlike a lidar sensor, a radar can provide information about the velocity and bearing of the object. Laser data is also more computationally intensive because a laser sends lots of data about each individual laser point of range data, which you have to make sense of in your algorithm. The techniques used to merge information from different sensor is called senssor fusion. For reasons discussed earlier, algorithms used in sensor fusion have to deal with temporal, noisy input and generate a probabilistically sound estimate of kinematic state. This blog post covers one of the most common algorithm used in position and tracking estimation called the Kalman filter and its variation called ‘Extended Kalman Filter’. In future articles we will cover other techniques such as Unscented Kalman Filters and Particle filters.



1. The Basic Kalman Filter — using Lidar Data

The Kalman filter is over 50 years old, but is still one of the most powerful sensor fusion algorithms for smoothing noisy input data and estimating state. It assumes that location variables are gaussian i.e. can be completely parametrized by the mean and the covariance: $X \sim N(\mu, \sigma^2)$

As information from the sensor flows, the kalman filter uses a series of state prediction and measurement update steps to update its belief about the state of the tracked object. These predict and update steps are described below.



State Prediction:

We will use a simplified linear state space model (see https://uk.mathworks.com/help/ident/ug/what-are-state-space-models.html) to illustrate the workings of the filter. The linear state state of a system at a time t can be estimated from state at time t-1 according to the equation(1):

$x_t = F_t x_{t-1} + B_t u_t + w_t$ -------*equation(1)*

where

- x_t is the state vector (postion and velocity) at time t
- u_t is the motion vector representing stimulus (steering angle, throttle)
- F_t is the state transition matrix
- B_t is the control input
- w_t is the noise term for the state vector with zero mean and covariance Q_t

The state vector x_t holds the position and velocity of the object. Let's represent this state by the vector $x_t = \begin{pmatrix} p \\ v \end{pmatrix}$ where

- p is the position with dimensions p_x and p_y
- v is the velocity with dimensions v_x and v_y

Let us assume that we are tracking a pedestrian moving at constant velocity. The relationship between the predicted state x', the previous state and velocity v is given by the kinematic equation: $x' = x + v\Delta t + 0.5a\Delta t^2 + \nu$

where:

- Δt is the prediction interval
- \nu is the process noise with zero mean and covariance Q. The covariance Q surfaces as acceleration noise in the state uncertainty prediction.
- a is the acceleration. This will be zero for an object moving at constant velocity

In matrix form we can write the above equation as equation(2):

$$x' = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix}$$ ---------*equation(2)*

If we compare the above to equation(1), this gives us the state transition matrix as: $F = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Besides predicting the state, we also need to be able to predict the uncertainty in our belief. This is given by the process covariance matrix P. In the initialisation of the filter, we make assumptions about the how certain we are about our initial state. The variance associated with the prediction is given by: $P_t = FP_{t-1}F^T + Q_t$

where Q_t is the process noise covariance associated with noisy inputs

Measurement Update

The next part of the Kalman filter algorithm is to use real measurements z to update the predicted state x′ by a scaling factor (called the Kalman Gain) proportional to the error between the measurment and the the predicted state.

This error y (also called the residual) is given by: $y = z - Hx'$

where H is a projection of the state vector in to the measurement space.

The Kalman gain is calaculated as :

$S = HPH^T + R$

$K = H^T PS^{-1}$

$x' = x + Ky$ ----------*Update the predicted state*

$P' = (I - KH)P$ ----------*Update the process uncertainty*

where

- S is a projection of the process uncertainty into the measurement space
- R is the measurement noise

You can find the derivation of the measurement update equations at: http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf

An Example

Enough of theory! Let's try some code to illustrate the basic workings of the KF. Here, we simulate an object whose state is modeled as a 4-dimensional vector x=[px py vx vy] In our case, the measurement sensor is laser sensor that returns the position data but no velocity information. In order to observe velocity we need to use a Radar sensor. This will be covered in the next section when we discuss Extended Kalman filters. We will start with a set of assumptions:

Model Assumptions

Time interval between predictions dt = 0.1

The initial state vector $x = \begin{pmatrix} 4 \\ 12 \\ 0 \\ 0 \end{pmatrix}$

Motion vector $u = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Initial state uncertainty: we will assume 10 for positions x and y, 100 for the two velocities $P = \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{pmatrix}$

The State transition function $F = \begin{pmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

H matrix to map predicted state into measurement space. In the case of a lidar sensor, velocity is not in the measurment space.
$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$

Measurement noise $R = \begin{pmatrix} 0.1 & 0 \end{pmatrix}$

Psuedo code

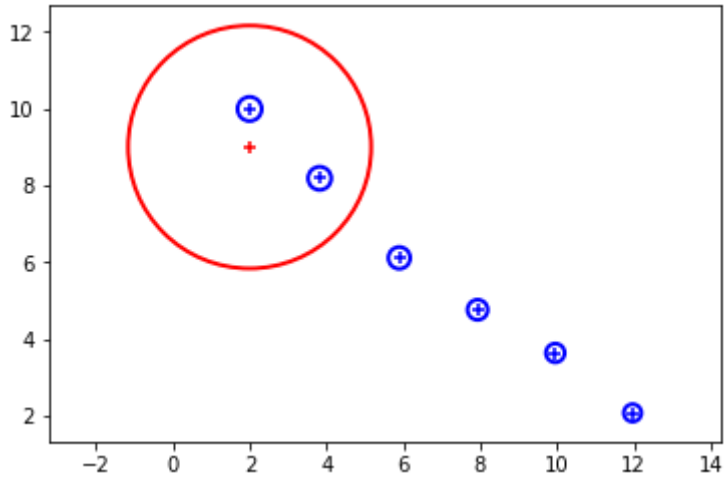
The basic code for the Kalman filter steps is listed below. You can find the code for the basic example at: https://github.com/asterixds/ExtendedKalmanFilter/python

```
"""prediction step"""
def predict(x, P):
    x = (F * x) + u
    P = F * P * F.transpose() #Acceleration noise Q is assumed to be
zero
    return x, P

"""measurement update step"""
def update(x, P,z):
    # measurement update
    Z = matrix([z])
    y = Z.transpose() - (H * x)
    S = H * P * H.transpose() + R
    K = P * H.transpose() * S.inverse()
    x = x + (K * y)
    P = (I - (K * H)) * P
    return x, P
```

The final step iterates through the measurements and applies the prediction and update steps of the filter as listed above.

```
plot_position_variance(x,P,edgecolor='r')  #plot initial position
and covariance in red
for z in measurements:
    x,P = predict(x, P)
    x,P = update(x, P,z)
    plot_position_variance(x,P,edgecolor='b') #plot updates in blue
    print(x)
    print(P)
```



Kalman Filter Iterations: The filter converges to the truth after a few iterations

The above figure illustrates each iteration of the kalman filter for the px and py dimensions of the state vector along with the positional covariance. The red circle is a visualisation of our initial process uncertainty. As we go through the incremental predictions and measurement updates, we begin to develop a better estimate of state with less uncertainty (variance). As you can see, the final state vector x=[11.99, 2.05] is very close to the final measurement value and the positional state variance is also minimal at 0.05

2. The Extended Kalman filter — using Radar Data

Radar data poses a slightly more difficult challenge. Radar data is returned in Polar co-ordinates. Radar data consists of 3 components i.e.
- ρ or Range (distance from the origin)
- ϕ or bearing (the angle between ρ and x), and
- $\rho^{\cdot}$ which is the range rate.

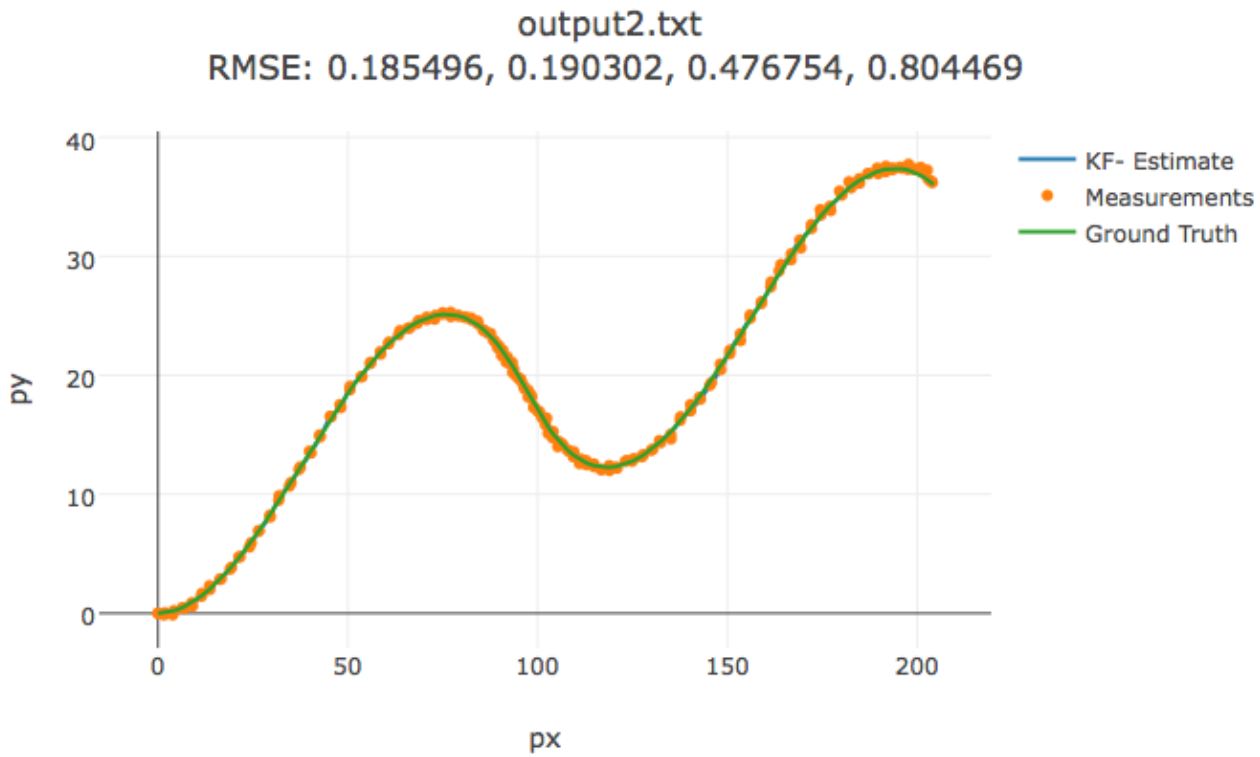
As there is no H matrix that will map the state vector into the radar measurement space, we need a new function h(x) that will map the state space into the measurement space for the measurement update step. This function is derived by mapping the polar cordinates into the cartesian space and is defined as:

$$h(x) = \begin{pmatrix} \sqrt{p_x^2 + p_y^2} \\ \arctan(p_y/p_x) \\ \frac{p_x v_x + p_y v_y}{\sqrt{p_x^2 + p_y^2}} \end{pmatrix}$$

then applies the Kalman filter to this approximation. This local linear approximation is obtained by computing a first order Taylor expansion around the current estimate. The first order approximations are also called the Jacobian Matrix. The derivations of the Jacoboians are a bit involved and we will not be covering these here. However, these are well documented on several internet resources on the topic, but if you want to use these straight off the cuff then you can refer to the implementation code in the github reference below:

Implementation reference:

You can find the code for a C++ impementation of the Kalman filter in the github repository: https://github.com/asterixds/ExtendedKalmanFilter



Conclusion

So far we have covered some of the fundamental algorithms used in sensor fusion for object tracking. In the next part of this blog post we will look at the Unscented Kalman filter which overcomes the need to use an approximation for the projection. We will