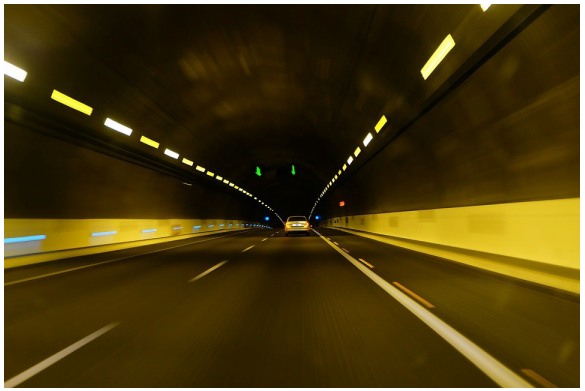
# The Kalman Filter: An algorithm for making sense of fused sensor insight

Sharath Srini Follow
Apr 18, 2018 · 14 min read

You're driving your car through a tunnel. The GPS signal is gone. Nevertheless, you might want to get notified that you should take the exit in the tunnel. How should we navigate on a car inside a tunnel, which should know where it is right now given only the last position?



## A Short Note on Sensors

Global Positioning System receivers calculate their locations by analyzing signals that they receive from satellites. These signals don't pass through solid objects. A GPS in a vehicle may have an external antenna, or it may pick up enough of bounced signal out of the air to operate. If signals in a tunnel are too weak, the GPS may still function, depending on its quality and features.

| GPS | IMU | Wheel speed sensors |
|---|---|---|
| Provides accurate measurement of absolute speed via the Doppler effect | Measures the acceleration and can quickly represent high-dynamic speed profiles via integration | Gives an accurate and high-frequency information about the current peripheral speed of each wheel |
| BUT: It is delayed and has a too low update rate (5 Hz) | BUT: with every integration a drift error occurs, which makes the speed determined by the IMU unusable in the medium term | BUT: due to the nature of the power transmission of the tire and the track, the circumferential speed at higher slip may differ significantly from the speed vector of the wheel hub. In addition, the wheel speed for the traction control serve as a controlled variable. How was that the same with the dog chasing its own tail? |

The table explains the pros and cons of each some of the sensors.

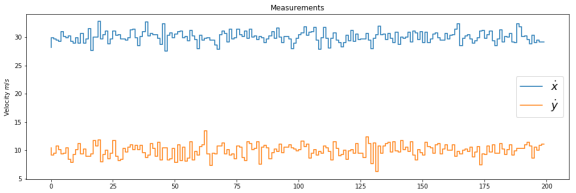
> *An approach that merges the vehicle sensors, can calculate the position .*

## The Measurement from the sensors

Assume that we were at the tunnel entrance and we were driving at 50km / h, then the navigation can indeed be calculated exactly where (x = position) we would be 1 minute (t = time) later.

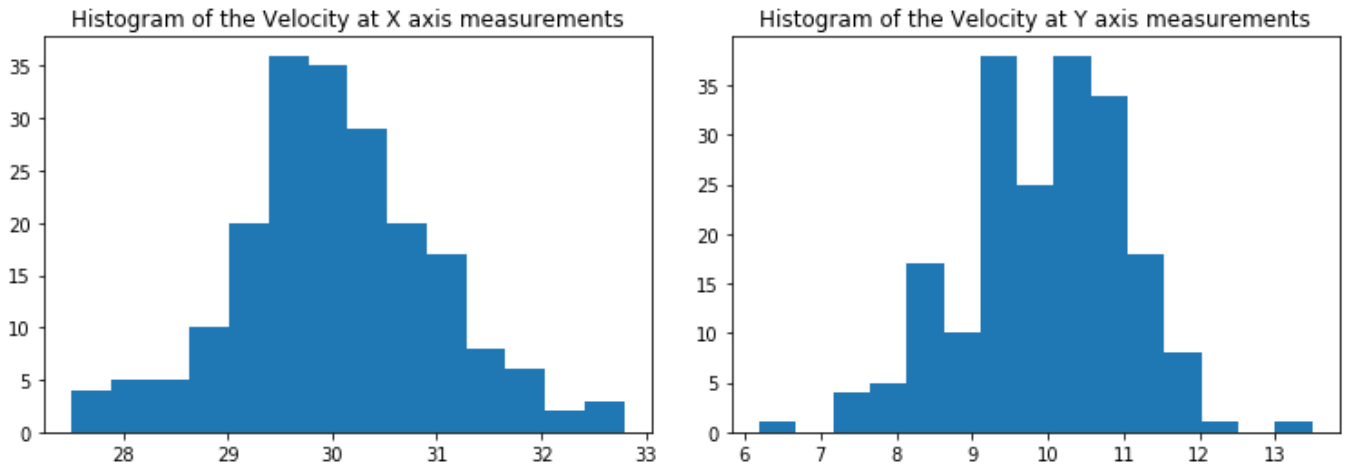
Both the sensors have random errors, the transmission path has interference, and the resolution of CAN bus or analog-to-digital converters can cause many inaccuracies in the simple statement “speed”.

For example, a speed signal looks like this:




The Velocity is considered to be with a variance on both the axes. (image source)

On average, the measured speed has some “noise” added to it which differentiates them from the ground truth. If one calculates a histogram of the determined speeds, one sees that the determined values are approximately subject to a normal distribution.




This is the histogram representation of the velocity measurements. (image-source)

So there is one, and really only one, maximum value (unimodal) and a spread (variance). If this is the case, we can do the calculation very well with a trick nevertheless.

## Idea of the Kalman filter in a single dimension

I would like to first explain the idea of the Kalman filter (according to Rudolf Emil Kalman ) with only one dimension . The following explanation is borrowed from the *Udacity CS373* course by *Prof. Sebastian Thrun*.

### Calculated noise helps

In order to perform the calculation optimally despite measurement noise, the *“how strong”* parameter must be known. This *“how strong”* is expressed with the variance of the normal distribution. This is

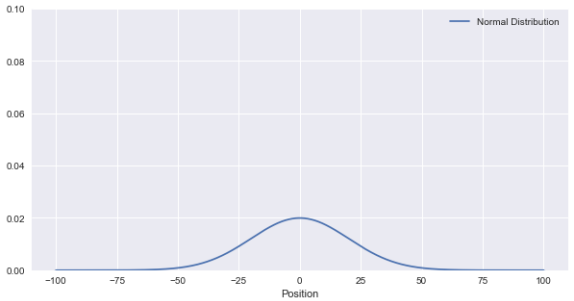
determined once for a sensor that is being used and then uses only this "uncertainty" for the calculation.

In the following, it is no longer calculated with absolute values but with mean values (μ) and variances $\sigma^2$ of the normal distribution. The mean of the normal distribution is the value that we would want to calculate. The variance indicates how confidence level. The narrower the normal distribution (low variance), the confident the sensors are with the measurements.

> *A sensor that measures 100% exactly has a variance of $\sigma^2 = 0$ (it does not exist).*

Let's assume that the GPS signal has just been lost and the navigation system is completely unclear where you are. The variance is high, the curve corresponding is really flat. There is an uncertainty.

Normal distribution with variance = 20 and mean = 0




The Uncertainty is High, as the variance is in a large magnitude.(image-source)

Now comes a speed measurement from the sensor, which is also "inaccurate" with appropriate variance. These two uncertainties must now be linked together. With the help of Bayes rule, the addition of two Gaussian function is performed . Prof. Thrun explains this very clearly in the Udacity CS373 course.

The two pieces of information (one for the current position and one for the measurement uncertainty of the sensor) actually gives a better result!. The narrower the normal distribution, the confident the result. Movement worsens the estimate.

Of course, the vehicle also moves, which adversely affects the accuracy of the position determination. A sensor, for example, can determine the

rotation of the wheel and make the assumption of the radius of the wheel and could have a conclusion on the distance traveled, but this will always remain somewhat inaccurate. This inaccuracy of movement is also described with a normal distribution. Computing with the current estimate runs a little differently this time, because the “movement” can also be called “Prediction”. You can estimate after the calculation, where you will be next (measurement) time.

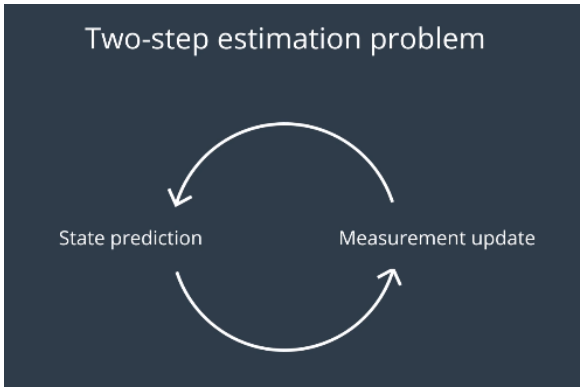
In our example, μ is simply v * dt, which is the distance we had traveled in the calculation time.

A simple implementation of this is:

```
def predict(mean1, var1, mean2, var2):
 new_mean = mean1 +mean2
 new_var = var1 + var2
 return [new_mean, new_var]
```

## Measuring & Updating: The Kalman filter

The Kalman filter simply calculates these two functions over and over again.




The filter loop that goes on and on.

The filter cyclically overrides the mean and the variance of the result. The filter will always be confident on where it is, as long as the readings do not deviate too much from the predicted value.

*A new measurement improves the estimate*

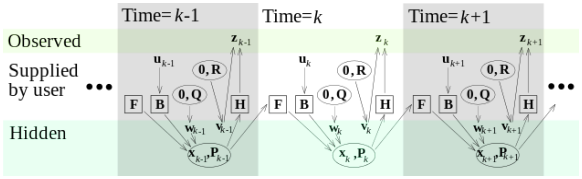
> *Since the measured values (in update) fit relatively well to the predicted ones (by predict), the filter improves step by step to ensure that it is correct (normal distributions become narrower and higher), even though the values are noisy.*

```
def update(mean1, var1, mean2, var2):

 sum = var1+var2
 pr_s = mean1*var2 + mean2*var1
 #print(pr_s)
 new_mean =1/(sum) * pr_s
 product = var1*var2
 new_var = product/sum
 return [new_mean, new_var]
```

*Without matrices, you can only count in one dimension, which is insufficient for...*

## Multidimensional Kalman filter




The Picture Illustrates the Kalman Filter 's Predition step in various time-stages. (Image Source)

I would like to explain the procedure again using the example of a vehicle with navigation device, which enters a tunnel. The last known position is before losing the GPS signal. Afterwards only the speed information of the vehicle (wheel speeds & yaw rate) is available as a normal distributed noisy measured variable. From here, the velocity is calculated.

We now turn to the more complicated part. The procedure mentioned with multiplying or adding the mean values and variances thus only works in the one-dimensional case. In the multi-dimensional problem, we would have the mean and the variance inside a matrix on which all the operations are performed.That is, when the state you want to measure can be fully described with just one variable. The example, which was mentioned at the beginning, to determine the position of a vehicle in the tunnel, can no longer be completely described with a variable. Although only interested in the position, but this is already a two dimensional problem in the plane. In addition, only the velocity can be measured, not the position directly. This results in a Kalman filter with the following state variables.

$$x = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

The state matrix consists of position and velocity in the x and y coordinates.

## Initial conditions / initialization

### System state X

At the beginning we will have to initialize with an initial state. In the one dimensional case the state was a vector.

$$x = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

If nothing is known, you can simply enter zero here. If some boundary conditions are already known, they can be communicated to the filter. The choice of the following covariance matrix controls how fast the filter converges to the correct (measured) values

### Co-variance matrix P

An uncertainty must be given for the initial state . In the one-dimensional case, the variance was a vector, but now is matrix of uncertainty for all states. Here is an example with for all the four states.

$$P = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

The covariance matrix consists of uncertainty in the position and the velocity in the x and y coordinates.

This matrix is most likely to be changed during the filter passes. It is changed in both the predict and correct steps. The Matrices can be initialized on the basis of the sensor accuracy. If the sensor is very accurate, small values should be used here. If the sensor is relatively inaccurate, large values should be used here to allow the filter to converge relatively quickly. If the sensor is very accurate, small values should be used here.

### Dynamics matrix A

The core of the filter, however, is the following definition, which we should set up with great understanding of the physical context. This is not easy for many real problems. For our simple example (in-plane motion), the physics behind it comes from the smooth motion. For the state matrix shown above, the dynamics in matrix notation is as follows:

$$A = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The Dynamic matrix helps us in defining the equations for predicting the Vehicle Motion Model.

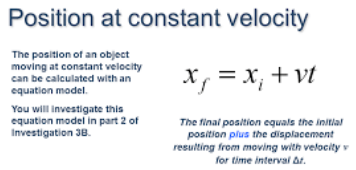
This states “where” the state vector moves from one calculation step to the next within. This dynamic model is in our case is “constant

velocity" model because it assumes that the velocity remains constant during a filter's calculation step($dt$).

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{t+1} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{t}$$

This is the prediction step for the State matrix.

This simply reflects physical relationships for the uniform motion. A higher form would be the Constant Acceleration model, which would be a 6-D filter and still includes the accelerations in the state vector. In principle, other dynamics can be specified here.




(Image Source)

### Process noise co-variance matrix Q

As the movement of the vehicle (in the sense of a superimposed, normally distributed noise) may also be disturbed, this is where the process noise co-variance matrix is introduced. This matrix tells us about the filter , and how the system state can "jump" from one step to the next. Imagine the vehicle that drives autonomously. It can be disturbed by a gust of wind or road bumps, which has a force effect. A speed change by the driver is also an acceleration that acts on the vehicle. If an acceleration now affects the system state , then the physical dependence for it is Q. The matrix is a co-variance matrix containing the following elements:

$$Q = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\dot{x}} & \sigma_{x\dot{y}} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\dot{x}} & \sigma_{y\dot{y}} \\ \sigma_{\dot{x}x} & \sigma_{\dot{x}y} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x}\dot{y}} \\ \sigma_{\dot{y}x} & \sigma_{\dot{y}y} & \sigma_{\dot{y}\dot{x}} & \sigma_{\dot{y}}^2 \end{bmatrix}$$

The Process Noise Co-variance matrix consist of the errors caused in the process.

It is easy to calculate by placing the vector and then multiplying it by the assumed standard deviation for the acceleration.

$$Q = G \cdot G^T \cdot \sigma_a^2$$

$$G = \begin{bmatrix} 0.5\,\Delta t^2 & 0.5\,\Delta t^2 & \Delta t & \Delta t \end{bmatrix}^T$$

The Equations to set the Q matrix appropriately.

### Control Matrix B and Control Input u

An external control variables (eg: steering, braking, acceleration, etc.) is possible via the control matrix . The u matrix will contain the robotic input of the system which could be the instantaneous acceleration or the distance traveled by the system from a IMU or a odometer sensor.

### Measuring matrix H

The filter must also be told what is measured and how it relates to the state vector. In the example of the vehicle, the car enters a tunnel with only position measured at first point, only the speed is measured! The values can be measured directly with the factor 1.0 (ie the velocity is measured directly in the correct unit), which is why in only 1.0 is set to the appropriate position.

$$H = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The H-matrix.

*If the sensors measure in a different unit or the size by detours, the relationships in the measuring matrix must be mapped in a formula.*

### Measurement noise covariance matrix R

As in the one-dimensional case the variance, a measurement uncertainty must also be stated here.

$$R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

This measurement uncertainty indicates how much one trusts the measured values of the sensors. Since we measure the position and the velocity , this is a $2 \times 2$ matrix. If the sensor is very accurate, small values should be used here. If the sensor is relatively inaccurate, large values should be used here.

### Unit matrix I

Last but not least, a unit matrix is necessary, which would be used to simplify the Kalman equations.

## Filtering step Prediction / Predict

This part of the Kalman filter now dares to predict the state of the system in the future. In addition, under certain conditions, a state can be calculated with it which cannot be measured! That's amazing, but in our case exactly what we need. We cannot measure the position of the vehicle because the GPS of the navigation device has no reception in a tunnel. By initializing the state vector with a position and measuring the velocity, however, the dynamics still be used to make an optimal prediction about the position.

$$x_{t+1} = A * x_t$$

The co-variance must also be recalculated. Uncertainty about the state of the system increases in the predict step, as we have seen in the one dimension case. In the multidimensional case, the measurement uncertainty is added, so the uncertainty becomes larger and larger.

$$P = A \cdot P \cdot A' + Q$$

That's it. The Kalman filter has made a prediction statement about the expected system state in the future or in the upcoming time-step. The filter will now be measuring / correcting and checking whether the prediction of the system state fits well with the new measurements.

The co-variance chosen to be smaller by the filter illustrates the certainty, if not, then something is wrong, which makes the filter more uncertain.

## Filter step Measure / Correct

*Note : The following mathematical calculations do not need to be derived.*

From the sensors come current measured values, with which an innovation factor (y) is obtained by using the measurements ,the state vector with the measuring matrix .

$$y = Z - (H \cdot x)$$

Then it is looks at with which variance can be further calculated. For this, the uncertainty and the measurement matrix and the measurement uncertainty required.

$$S = (H \cdot P \cdot H' + R)$$

This determines the so-called Kalman gain. It states whether the readings or system dynamics should be more familiar.

$$K = P \cdot H' S$$

> *The Kalman Gain will decrease if the readings (measurements) match the predicted system state. If the measured values say otherwise, the elements of matrix K become larger.*

This information is now used to update the system state.

$$x = x + (K \cdot y)$$

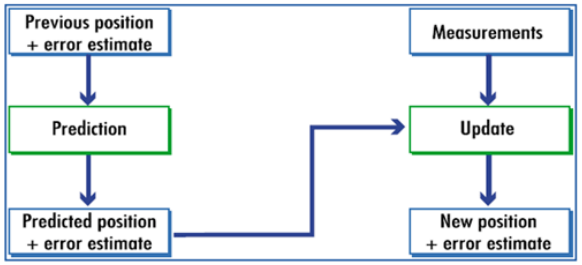
And also determined a new co-variance for the upcoming predict step.

$$P = P - (K \cdot H.P)$$

which is ,

$$P = (I - (K \cdot H)) \cdot P$$

Now it's back to the step prediction. Graphically it looks like this:




The Kalman filter could be understood as a loop (image source)

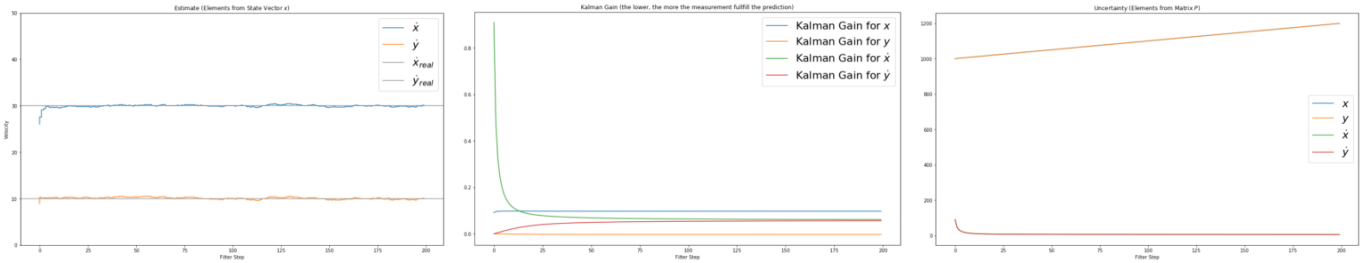
> *This filter runs permanently as long as measured values come in. It can also be open loop, so only the prediction step will be executed if no measurements are available. Then the uncertainty gets bigger and bigger.*

## The filter at work

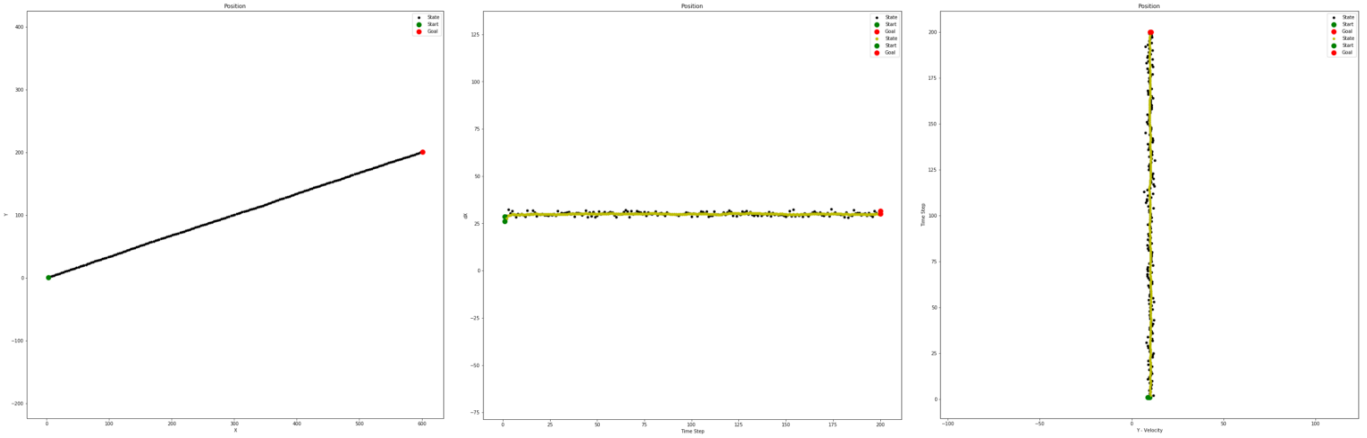
When we drive into a tunnel , the last known position is recorded which is received from the GPS. The Kalman filter can still predict the position of the vehicle, although it is not being measured at all time.

Now assuming the vehicle speed is available about every 20 m/s via the CAN bus, 6 iterations are only 0.1 s. The filter converges relatively quickly, depending on the choice of initial conditions. For example,

after 100 iterations (equivalent to 2s on the vehicle), the variance is already very low, so the filter is confident on its estimated and updates states. My implementation of the linear Kalman filter could be found here . The results are :




The First Image depicts the Estimated Value of each estimated value of the velocity in the X and Y co-ordinates. The Second Image Illustrates the Kalman gain for each of the parameter in the State Matrix. The Uncertainty Matrix containing the variance in each parameter of the State Matrix. (https://github.com/sharathsrini/Self-Driving-Car/blob/master/Kalman_Filter.ipynb)




The First Image Plots the position of the vehicle in both X and Y Coordinate. The Second and Third Image Tells us the relationship between the estimated and the actual velocity in the X and Y Co-ordinates. (https://github.com/sharathsrini/Self-Driving-Car/blob/master/Kalman_Filter.ipynb)

## Python implementation of the Kalman filter

```
def Kalman_Filter() :
  for n in range(measurements):
     x = A*x+B*u[n]
     P = A*P*A.T + Q
```

```
 # Measurement Update (Correction)
 # ===============================
 # Compute the Kalman Gain
    S = H*P*H.T + R
    K = (P*H.T) * np.linalg.pinv(S)

# Update the estimate via z
    Z = mx[n]
    y = Z - (H*x) # Innovation or Residual
    x = x + (K*y)

 # Update the error covariance
    P = (I - (K*H))*P
```

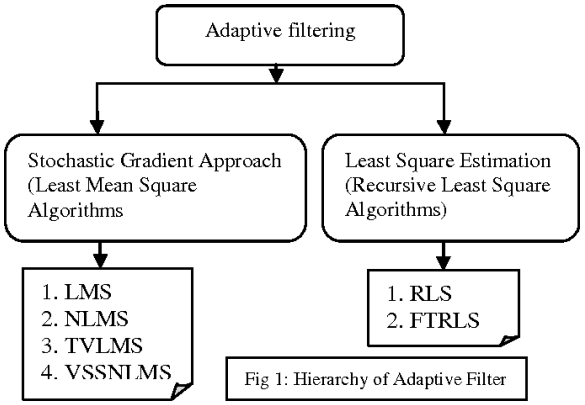
## Filter design: How do I choose Q and R?

Overall, no matter how large the numerical values are, but rather in what proportion they are. If the values chosen are ten times larger, this will hardly affect the filter. **The ratio of values is crucial.** The correct choice would be directly responsible for the filter performance and form the basic question of filter design.

This either / or question can only be decided on an application-specific basis. In some cases:

1. We would just want to filter poorly measuring sensors for a relatively constant process. For example, we can implement kalman filter to optimize temperature controller in a furnace in a rocket or in chemical furnace.
2. We would also want to merge several sensors and the dynamics should be preserved. Accordingly, the matrices should be selected. Alternatively, of course, the filter can be designed to adapt automatically during operation.

### How does a Kalman filter differ from Recursive Least Squares?




flowchart of adaptive filtering techniques (image).

**Kalman Filter** works on prediction-correction model used for linear and time-variant or time-invariant systems. Prediction model involves the actual system and the process noise .The update model involves updating the predicated or the estimated value with the observation noise. Kalman gain is calculated based on RLS algorithm in order to reach the optimal value within less amount of time.

**Recursive Least Squares** is based on weighted least squares in which previous values taken in account for determining the future value. Each weight is exponentially assigned to each previous value of the actual system. The weights are updated recursively based on memory.

**Notable Differences:**

$$O(N^2)$$

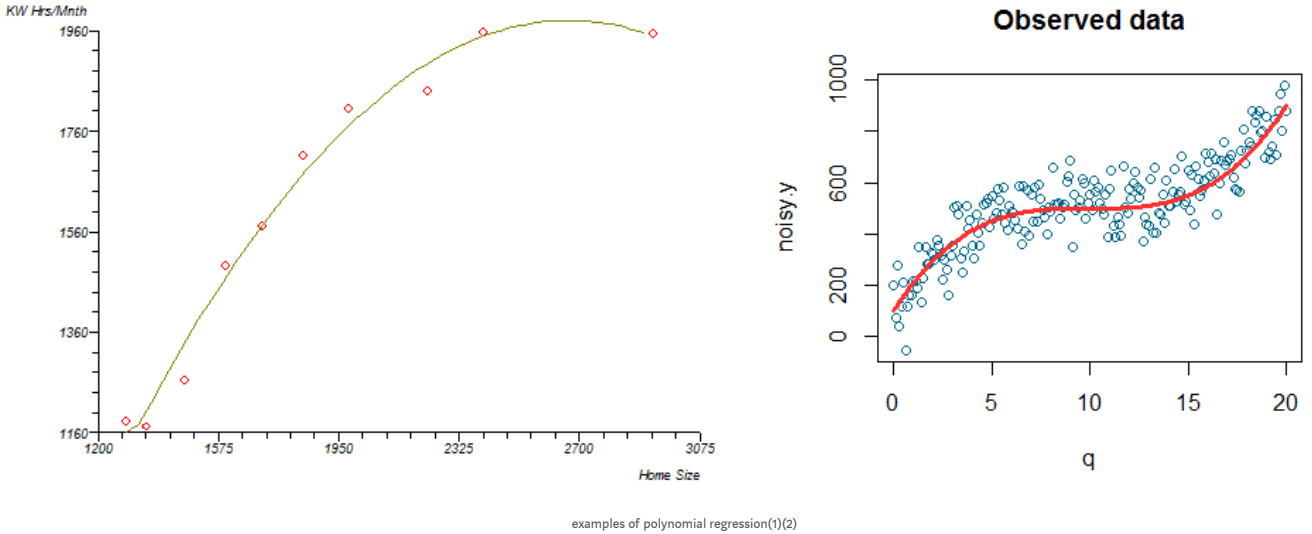
Computational Time complexity of RLS

$$O(N^3)$$

Computational Time complexity of Kalman Filter

1. RLS is faster than Kalman Filter.
2. Accuracy of Kalman Filter is high.
3. Kalman Filter is based on State-Space model where we need to model entire system to achieve optimal value.

**Kalman filter and Polynomial regression**




examples of polynomial regression(1)(2)

Polynomial regression is a method of function approximation. We have a data set and we have to determine the functional relationship, which is often expressed by estimating the probability density $p(z|x)$. Under the assumption that this probability is a Gaussian, we get the least squares solution as a maximum likelihood estimator.

Since Linear dynamic systems are state space models, we assume that the data we observe is generated by the application of a linear transform. The model we come with is the probability of a time series. The process then predicts the next value of a time series.

> *Polynomial regression does function approximation, Kalman filtering does time series prediction.*

The time series prediction is a special case of function approximation. Both the models are modeled from different assumptions on the data they observe.

## Conclusion

The Kalman filter is relatively quick and easy to implement and provides an optimal estimate of the condition for normally distributed noisy sensor values under certain conditions. Mr. Kalman was so convinced of his algorithm that he was able to inspire a friendly engineer at NASA. And so this filter helped for the first time in the Apollo Guidance Computer at the moon landings .

### Useful links:

(Image source)

1. The Jupyter Notebook Collection of Roger Labbe.
2. The amazing video series of Michel van Biezen
3. Kalman Filters in Pictures .
4. The the blog post of Vivek Yadav .

5. The amazing post by David Silver.