无人驾驶汽车系统入门（一）——卡尔曼滤波与目标追踪

无人驾驶汽车系统入门（一）——卡尔曼滤波与目标追踪

说明：

> 介绍无人驾驶汽车系统感知模块的重要技术——卡尔曼滤波

卡尔曼滤波按如下三个章节说明：

> 卡尔曼滤波与行人状态估计
>
> 扩展卡尔曼滤波（EKF）与传感器融合
>
> 处理模型，无损卡尔曼滤波（UKF）与车辆状态轨迹

本节为卡尔曼滤波，主要讲解卡尔曼滤波的具体推导，卡尔曼滤波在行人状态估计中的一个小例子。

为什么要学卡尔曼滤波?

> 卡尔曼滤波以及其扩展算法能够应用于目标状态估计，如果这个目标是行人，那么就是行人状态估计（或者说行人追踪）
>
> 如果这个目标是自身，那么就是车辆自身的追踪（结合一些地图的先验，GPS等数据的话就是自身的定位）。
>
> 在很多的无人驾驶汽车项目中，都能找到卡尔曼滤波的扩展算法的身影（比如说EKF，UKF等等）。
>
> 本节我们从最简单的卡尔曼滤波出发，完整的理解一遍卡尔曼滤波的推导过程，并实现一个简单的状态估计Python程序。

卡尔曼滤波是什么?

> 我们通常要对一些事物的状态去做估计，为什么要做估计呢？因为我们通常无法精确的知道物体当前的状态。
>
> 为了估计一个事物的状态，我们往往会去测量它，但是我们不能完全相信我们的测量，因为我们的测量是不精准的，它往往会存在一定的噪声，这个时候我们就要去估计我们的状态。
>
> 卡尔曼滤波就是一种结合预测（先验分布）和测量更新（似然）的状态估计算法。

概率论的知识基础

> 下面是一些概率论的基础知识，如果之前有这方面的知识储备那当然是最好的，很有利于我们理解整个博客内容
>
> 如果没有这方面的基础而且也看不懂下面的内容也没关系，我会以一个相对直观的方式来展现整个理论部分。
>
> 先验概率 P(X)P(X)：仅仅依赖主观上的经验，事先根据已有的只是的推断
>
> 后验概率 P(X|Z)P(X|Z)：是在相关证据或者背景给定并纳入考虑以后的条件概率
>
> 似然 P(Z|X)P(Z|X)：已知结果区推测固有性质的可能性
>
> 贝叶斯公式：

$$P(A|B) = \frac{P(A) \times P(B|A)}{P(B)}$$

后验分布正比于先验分布乘以似然。

卡尔曼滤波完整推导

1.简单的例子

若干年后，我们的可回收火箭要降落到地球，我们比较关心的状态就是我们的飞行器的高度了，飞行器的高度就是我们想要估计的状态，我们会通过一些传感器去测量我们当前的高度信息，比如说使用气压计。

假如我们每一次测量，当前高度都变成上一次测量的 95%，那么我们就可以得到如下关系：

$$Height^{(t)} = 0.95 \times Height^{(t-1)}$$

我们可以使用递归来表示这样的公式——为了计算我们当前的高度，我们必须知道我们上一个测量的高度，以此递推，我们就会推到我们的飞行器的初始高度。

但是，由于我们的测量往往来自于一些传感器（比如说GPS，气压计），所以测量的结果总是带有噪声的，这个噪声是有传感器本身引起的，那么我们的表达式就变成了：

$$Measurement^{(t)} = Height^{(t)} + Noise^{(t)}$$

这个噪声我们称之为测量噪声 (Measurement Noise)。

通常来说，这种噪声都满足高斯分布。我们用符号描述以上两部分：

$$x_k = ax_{k-1}$$

$$z_k = x_k + v_k$$

第一个式子是我们的 处理模型，是我们的经验（比如说一些运动模型，牛顿力学等等），我们用这种处理模型去 预测 我们考察的事物的状态（在没有任何信息的情况下，或者说在没有任何测量数据的情况下）；

第二个式子是测量的表达式，它大致描述了我们测量的组成成分，我们用这个测量去 更新 我们对状态的估计。

其中 xk 是我们的飞行器当前的状态，xk−1 是我们上一个状态（注意状态和测量的区别）

zk 是我们当前对飞行器的测量，vk 是我们当前的测量噪声， a 是一个常数，在我们这个例子里面就是 0.95

很显然，现实中的运动不会像我们这个简单的处理模型一样高度按比例缩小。

在这里为了简化，我们先假设我们的模型挺好的，能够大致描述出这颗“神奇火箭”的运动规律，只是偶尔会存在一定的偏差（比如所空气湍流的影响）

那么我们在计算xkxk 时在加一个噪声来描述我们的处理模型与实际运动的差异，这个噪声我们称之为处理噪声 (ProcessProcess NoiseNoise)，我们用 wt 来表示这种噪声，那么 xk 的计算公式就变成：

$$x_k = ax_{k-1} + w_k$$

为了简化，后面的分析我们会先忽略处理噪声，但是我们在传感器融合的部分会将处理噪声重新考虑进来。

2.状态估计

因为我们是要估计飞行器的状态（高度），所以我们把测量的公式变换成：

$$x_k = z_k - v_k$$

显然，在这里 v_k 是没办法知道的，卡尔曼滤波通过同时考虑上一状态值和当前的测量值来获得对当前状态值的估计。一般我们用 $\hat{x}$ 来表示对状态 x 的估计。那么 $\hat{x}_k$ 就表示当前状态的估计。下面我们可以用如下公式来描述卡尔曼滤波如何结合上一个估计和和现在的测量来产生对当前的估计：

$$\hat{x}_k = \hat{x}_{k-1} + g_k(z_k - \hat{x}_{k-1})$$

这里的 g_k 叫做 **卡尔曼增益（Kalman Gain）**，它描述的是之前的估计和当前的测量对当前的估计的影响的分配权重。为了理解，我们考虑极端的例子，如果 $g_k = 0$，也就是说增益为 0，那么：

$$\hat{x}_k = \hat{x}_{k-1}$$

也就是说我们非常不信任我们当前的测量，我们直接保留了我们上一次的估计作为我们对当前状态的估计。如果 $g_k = 1$，即增益为 1，则：

$$\hat{x}_k = z_k$$

即我认为当前的测量非常可信，我彻底接受它作为我当前状态的估计。那么当 g_k 介于 $(0, 1)$，则表示对两者的权重分配。

3. 计算卡尔曼增益

那么如何计算卡尔曼增益呢？我们使用一种间接的方法，我们虽然不知道测量噪声 v_k 的值，但是我们知道它的均值，前面我们提到，测量噪声来自传感器本身，并且符合高斯分布，所以我们能够从传感器厂商那里获得测量噪声的均值 r，那么 g_k 可以表示为：

$$g_k = p_{k-1}/(p_{k-1} + r)$$

其中 p_k 叫预测误差，表达式为：

$$p_k = (1 - g_k)p_{k-1}$$

那么怎么理解 g_k 和 p_k 呢？

那么假设前一次的预测误差 $p_{k-1} = 0$， 那么根据公式，当前的增益 $g_k = 0$，一维着舍弃掉当前的测量而完全采用上一个时刻的估计，如果$p_{k-1} = 1$ 那么增益变成 $1/(1 + r)$ 通常 r 是个很小的数值，所以增益为1，所以完全接受这一次的测量作为我们的估计（因为上一次的的预测误差太大了，为1，所以一旦拿到了新的测量，如获至宝，就干脆把不准确的上次的估计舍弃掉了）

对于下面的公式的分析是一样的，我们考虑极端的例子，当增益为 0，$p_k = p_{k-1}$，因为我们彻底舍弃掉了本次的测量，所以本次的预测误差只能接受上一次的。当增益为 1，$p_k = 0$。

4. 预测和更新

那么现在我们有关于当前的火箭高度状态的两个公式了，它们分别是：

$$x_k = ax_{k-1}$$

和

$$\hat{x}_k = \hat{x}_{k-1} + g_k(z_k - \hat{x}_{k-1})$$

那么我们到底要用哪一个呢？答案是我们都用，第一个公式我们称之为预测，是基于一些先验的知识（比如说运动模型，牛顿力学等等）觉得我们的状态应该是这样的，而第二个公式呢，就是我们基于我们"**不完美的**"的传感器的测量数据来更新我们对状态的估计。另为，预测，理论上只考虑了一个固定的处理模型和处理噪声，但是由于我们现在是对机械的状态进行估计，在预测过程中需要对机械本身的控制建模，我们在预测部分再新增一个控制信号，我们用 bu_k 表示。实际的传感器测量除了会有测量噪声 v_k 以外，还会存在一定的关于真实状态的缩放，因此我们使用 x_k 表示测量时通常还会在其前面加一个缩放系数 c。结合这写我们就可以得到卡尔曼滤波预测和更新过程了：

预测

$$\hat{x}_k = a\hat{x}_{k-1} + bu_k$$

$$p_k = ap_{k-1}a$$

卡尔曼滤波**更新**的过程为：

$$g_k = p_k c/(cp_k c + r)$$

$$\hat{x}_k \leftarrow \hat{x}_k + g_k(z_k - c\hat{x}_k)$$

$$p_k \leftarrow (1 - g_k c)p_k$$

---

5. 使用线性代数的方法来表示预测和更新

预测

$$\hat{x}_k = A\hat{x}_{k-1} + Bu_k$$

$$P_k = AP_{k-1}A^T$$

卡尔曼滤波**更新**的过程为：

$$G_k = P_k C^T(CP_k C^T + R)^{-1}$$

$$\hat{x}_k \leftarrow \hat{x}_k + G_k(z_k - C\hat{x}_k)$$

$$P_k \leftarrow (1 - G_k C)P_k$$

至此，卡尔曼滤波的完整推导就结束了。

下面，我们来看看卡尔曼滤波在无人汽车的感知模块的应用

卡尔曼滤波算法为什么会叫滤波算法?

以一维卡尔曼滤波为例，如果我们单纯的相信测量的信号，那么这个信号是包含噪声的，是很毛糙的

但是当我们运行卡尔曼滤波算法去做估计，我们估计的信号会很光滑，看起来似乎滤掉了噪声的影响，所以称之为滤波算法。

实际上，卡尔曼滤波不仅仅过滤掉了测量信号的噪声，它同时也结合了以往的估计，卡尔曼滤波在线性问题中被证明是最优估计。

卡尔曼滤波在无人驾驶汽车感知模块的应用

1. 无人车感知模块的传感器

无人驾驶汽车要安全的在道路上行驶，需要“耳听六路，眼观八方”。

那么无人车的耳朵和眼睛是什么呢？那就是安装在无人车上的各种各样的传感器了。

无人车上的传感器能够多达几十个，而且是不同种类的，比如：

立体摄像机

交通标志摄像机

雷达（RADAR）

激光雷达（LIDAR）

立体摄像机往往用于获取图像和距离信息；交通标志摄像机用于交通标志的识别；

雷达一般安装在车辆的前保险杆里面，用于测量相对于车辆坐标系下的物体，可以用来定位，测距，测速等等，容易受强反射物体的干扰，通常不用于静止物体的检测

激光雷达往往安装在车顶，使用红外激光束来获得物体的距离和位置，但是空间分辨率高，但是笨重，容易被天气影响。

由此可知，各种传感器都有其优点和缺点，在实际的无人驾驶汽车里，我们往往结合多种传感器的数据去感知我们的车辆周边的环境。

这种结合各种传感器的测量数据的过程我们称之为传感器融合（Sensor Fusion）。

本系列博客后面的章节我将详细给大家介绍基于扩展卡尔曼滤波和无损卡尔曼滤波在传感器融合中的应用。

在本节中，我们主要考虑卡尔曼滤波基于单一的传感器数据来估算行人位置的方法

2.基于卡尔曼滤波的行人位置估算

卡尔曼滤波虽然简单，确实无人驾驶汽车的技术树中非常重要的一部分，当然，在真实的无人驾驶汽车项目中使用到的技术是相对更加复杂的，但是其基本原理仍然是本博客介绍的这些内容。

在无人驾驶中，卡尔曼滤波主要可以用于一些状态的估计，主要应用于行人，自行车以及其他汽车的状态估计。下面，我们以行人的状态估计为例展开。

当我们要估计一个行人的状态时，首先就是要建立起腰估计的状态的表示。这里，行人的状态大致可以表示为 $x = (p, v)$，其中 p 为行人的位置，而 v 则是行人此时的速度。我们用一个向量来表示一个状态：

$$x = (p_x, p_y, v_x, v_y)^T$$

在确定了我们要估计的状态以后，我们需要确定一个**处理模型**，如前文所说，处理模型用于预测阶段来产生一个对当前估计的先验。在本文中，我们先以一个最简单的处理模型来描述行人的运动——**恒定速度模型**：

$$x_{k+1} = Ax_k + \nu$$

即：

$$x_{k+1} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix}_k + \nu$$

之所以称之为恒定速度模型，是因为我们将上面这个行列式展开可以得到：

$$p_x^{k+1} = p_x^k + v_x^k \Delta t$$

$$p_y^{k+1} = p_y^k + v_y^k \Delta t$$

$$v_x^{k+1} = v_x^k$$

$$v_y^{k+1} = v_y^k$$

恒定速度处理模型假定预测的目标的运动规律是恒定的速度的，在行人状态预测这个问题中，很显然行人并不一定会以恒定的速度运动，所以我们的处理模型包含了一定的 **处理噪声**，在这个问题中处理噪声也被考虑了进来，其中的 ν 是我们这个处理模型的处理噪声。在行人状态估计中的处理噪声其实就是行人的加减速，那么我们原来的处理过程就变成了：

$$x_{k+1} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix}_k + \begin{bmatrix} \frac{1}{2} a_x \Delta t^2 \\ \frac{1}{2} a_y \Delta t^2 \\ a_x \Delta t \\ a_y \Delta t \end{bmatrix}_k$$

我们预测的第二步就变成了：

$$A' = APA^T + Q$$

这是我们的A的更新过程，它本质上是我们的估计状态概率分布的协方差矩阵。Q 是我们的处理噪声的协方差矩阵，由于处理噪声是随机带入的，所以 ν 本质是一个高斯分布：$\nu \sim N(0, Q)$，Q是处理噪声的协方差，Q的形式为：

$$Q = \begin{bmatrix} \sigma_{p_x}^2 & \sigma_{p_x p_y} & \sigma_{p_x v_x} & \sigma_{p_x v_y} \\ \sigma_{p_y p_x} & \sigma_{p_y}^2 & \sigma_{p_y v_x} & \sigma_{p_y v_y} \\ \sigma_{v_x p_x} & \sigma_{v_x p_y} & \sigma_{v_x}^2 & \sigma_{v_x v_y} \\ \sigma_{v_y p_x} & \sigma_{v_y p_y} & \sigma_{v_y v_x} & \sigma_{v_y}^2 \end{bmatrix}$$

进而表示为：

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

其中$G = [0.5\Delta t^2, 0.5\Delta t^2, \Delta t, \Delta t]^T$，$\sigma_v^2$ 对于行人我们可以设置为$0.5m/s^2$（这一部分大家如果想要详细了解可以阅读这篇文章：http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5940526）。

测量步骤中，我们直接可以测量到速度 v_x 和 v_y，所以我们的测量矩阵 C 可以表示为：

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

测量噪声的协方差矩阵 R 为：

$$R = \begin{bmatrix} \sigma_{v_x}^2 & 0 \\ 0 & \sigma_{v_y}^2 \end{bmatrix}$$

其中的 $\sigma_{v_x}^2$ 和 $sigma_{v_x}^2$ 描述了我们的传感器的测量能够有多“差”，这是传感器固有的性质，所以往往是传感器厂商提供。

最后，在测量的最后一步，我们使用单位矩阵 I 来代替原式中的1:

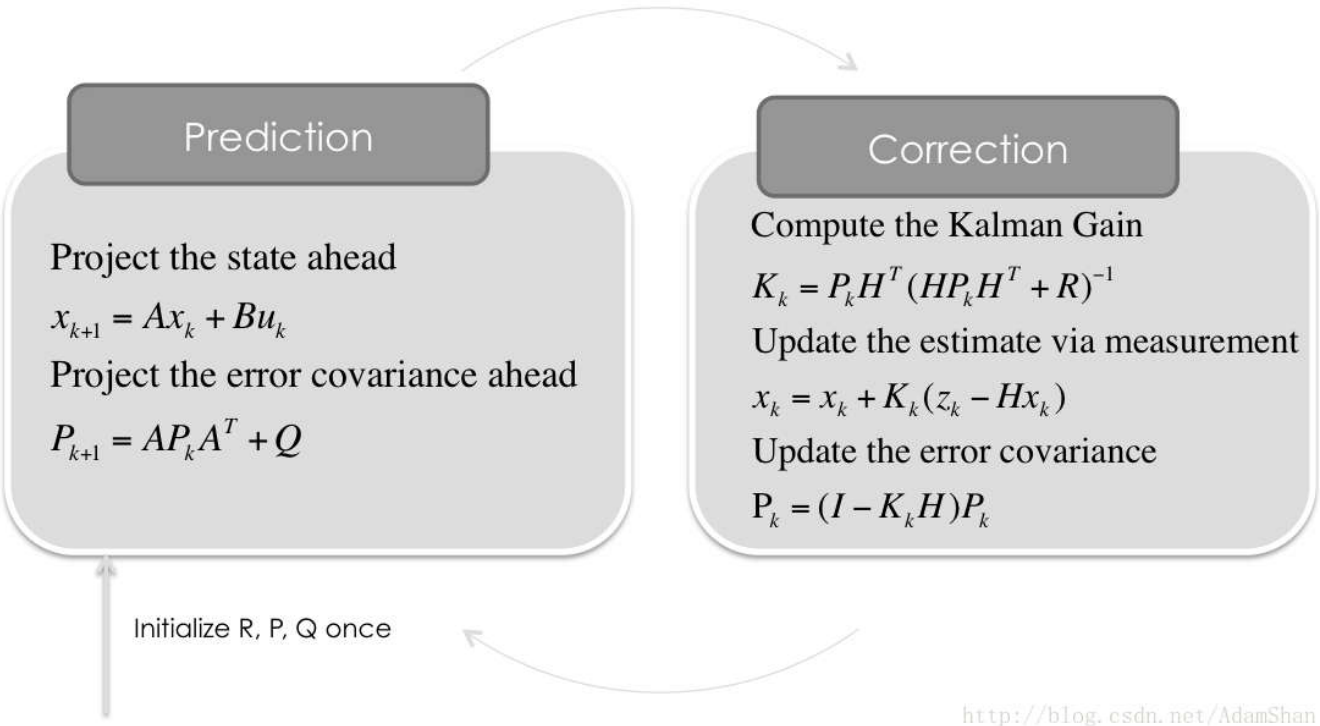
$$P_k \leftarrow (I - G_k C) P_k$$

至此，基于**恒定速度处理模型**卡尔曼滤波的行人状态估计的整个流程我们就讲完了，下面我们使用Python将整个过程实现一下。

3.卡尔曼滤波行人状态估计Python例子

首先我们看一下卡尔曼滤波的整个流程，其实在实际的论文和资料中，预测矩阵通常使用FF来表示（我们前文中一直是使用AA），测量矩阵通常使用 HH 表示（我们前文中使用的是 CC），卡尔曼增益通常使用 KK 来表示（我们前文中使用的是 GG）

下面是文献材料中通常意义的卡尔曼滤波过程：



注意：公式还包含一项：Bμ ,这一项是指我们在追踪一个物体的状态的时候把它内部的控制也考虑进去了，这在行人，自行车，其他汽车的状态估计问题中是无法测量的，所以在这个问题中我们设置 Bμ 为 0.

我们的代码将在IPython中执行，大家可以中jupyter notebook交互式地运行。

首先我们载入必要的库：

```
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.stats import norm
```

接着我们初始化行人状态x, 行人的不确定性（协方差矩阵）P，测量的时间间隔dt，处理矩阵F以及测量矩阵H：

```
x = np.matrix([[0.0, 0.0, 0.0, 0.0]]).T
print(x, x.shape)
P = np.diag([1000.0, 1000.0, 1000.0, 1000.0])
print(P, P.shape)

dt = 0.1 # Time Step between Filter Steps
F = np.matrix([[1.0, 0.0, dt, 0.0],
               [0.0, 1.0, 0.0, dt],
               [0.0, 0.0, 1.0, 0.0],
               [0.0, 0.0, 0.0, 1.0]])
print(F, F.shape)

H = np.matrix([[0.0, 0.0, 1.0, 0.0],
               [0.0, 0.0, 0.0, 1.0]])
print(H, H.shape)

ra = 10.0**2

R = np.matrix([[ra, 0.0],
               [0.0, ra]])
```

计算测量过程的噪声的协方差矩阵R和处理噪声的协方差矩阵Q：

```
ra = 0.09
R = np.matrix([[ra, 0.0],
               [0.0, ra]])
print(R, R.shape)

sv = 0.5
G = np.matrix([[0.5*dt**2],
               [0.5*dt**2],
               [dt],
               [dt]])
Q = G*G.T*sv**2
from sympy import Symbol, Matrix
from sympy.interactive import printing
printing.init_printing()
```

```
dts = Symbol('dt')
Qs = Matrix([[0.5*dts**2],[0.5*dts**2],[dts],[dts]])
Qs*Qs.T
```

定义一个单位矩阵：

```
I = np.eye(4)
print(I, I.shape)
```

我们随机产生一些测量数据：

```
m = 200 # Measurements
vx= 20 # in X
vy= 10 # in Y

mx = np.array(vx+np.random.randn(m))
my = np.array(vy+np.random.randn(m))
measurements = np.vstack((mx,my))

print(measurements.shape)
print('Standard Deviation of Acceleration Measurements=%.2f' % np.std(mx))
print('You assumed %.2f in R.' % R[0,0])

fig = plt.figure(figsize=(16,5))
plt.step(range(m),mx, label='$\dot x$')
plt.step(range(m),my, label='$\dot y$')
plt.ylabel(r'Velocity $m/s$')
plt.title('Measurements')
plt.legend(loc='best',prop={'size':18})
```

一些过程值，用于结果的显示：

```
xt = []
yt = []
dxt= []
dyt= []
Zx = []
Zy = []
Px = []
Py = []
Pdx= []
Pdy= []
Rdx= []
Rdy= []
Kx = []
Ky = []
Kdx= []
Kdy= []

def savestates(x, Z, P, R, K):
    xt.append(float(x[0]))
    yt.append(float(x[1]))
```

卡尔曼滤波：

```
for n in range(len(measurements[0])):

    # Time Update (Prediction)
    # ========================
    # Project the state ahead
    x = F*x

    # Project the error covariance ahead
    P = F*P*F.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 = measurements[:,n].reshape(2,1)
    y = Z - (H*x)                            # Innovation or Residual
    x = x + (K*y)
```

显示一下关于速度的估计结果：

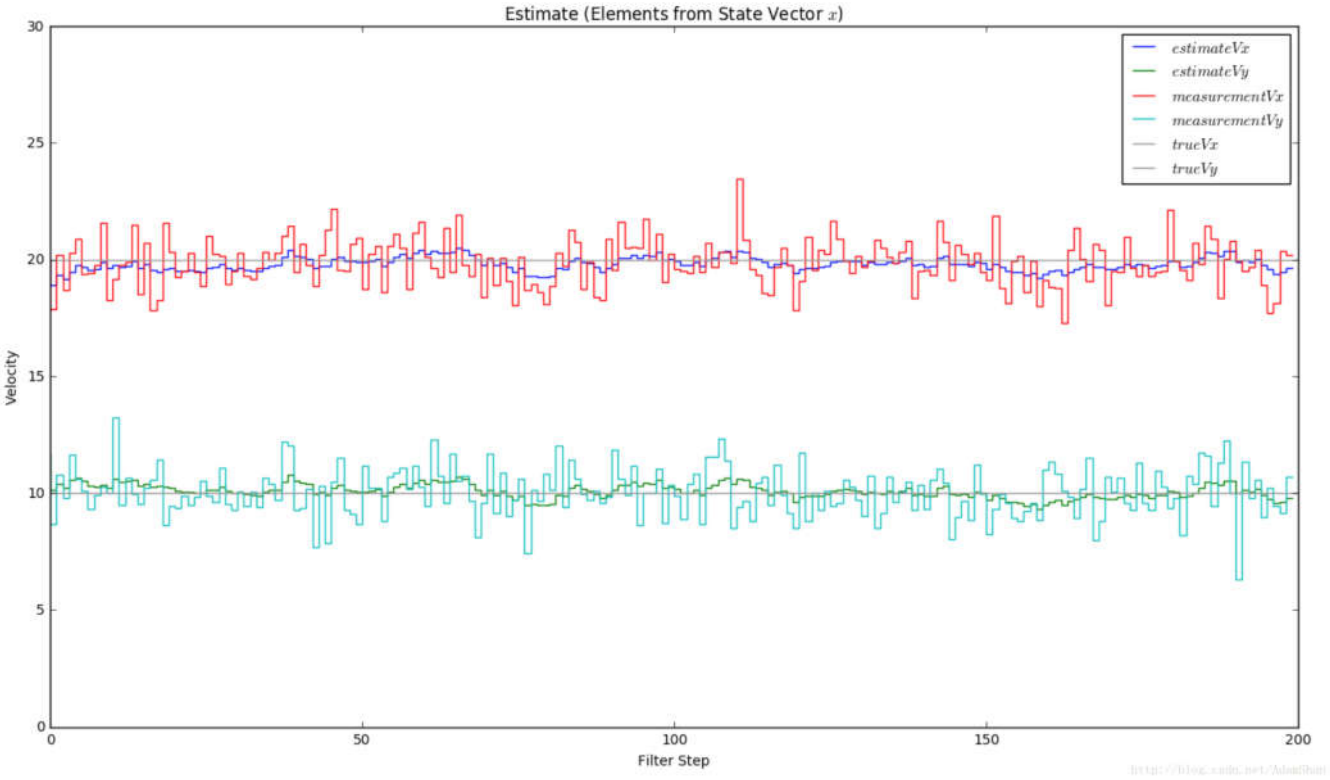
```
def plot_x():
    fig = plt.figure(figsize=(16,9))
    plt.step(range(len(measurements[0])),dxt, label='$estimateVx$')
    plt.step(range(len(measurements[0])),dyt, label='$estimateVy$')

    plt.step(range(len(measurements[0])),measurements[0], label='$measurementVx$')
    plt.step(range(len(measurements[0])),measurements[1], label='$measurementVy$')

    plt.axhline(vx, color='#999999', label='$trueVx$')
    plt.axhline(vy, color='#999999', label='$trueVy$')

    plt.xlabel('Filter Step')
    plt.title('Estimate (Elements from State Vector $x$)')
    plt.legend(loc='best',prop={'size':11})
    plt.ylim([0, 30])
    plt.ylabel('Velocity')
plot_x()
```

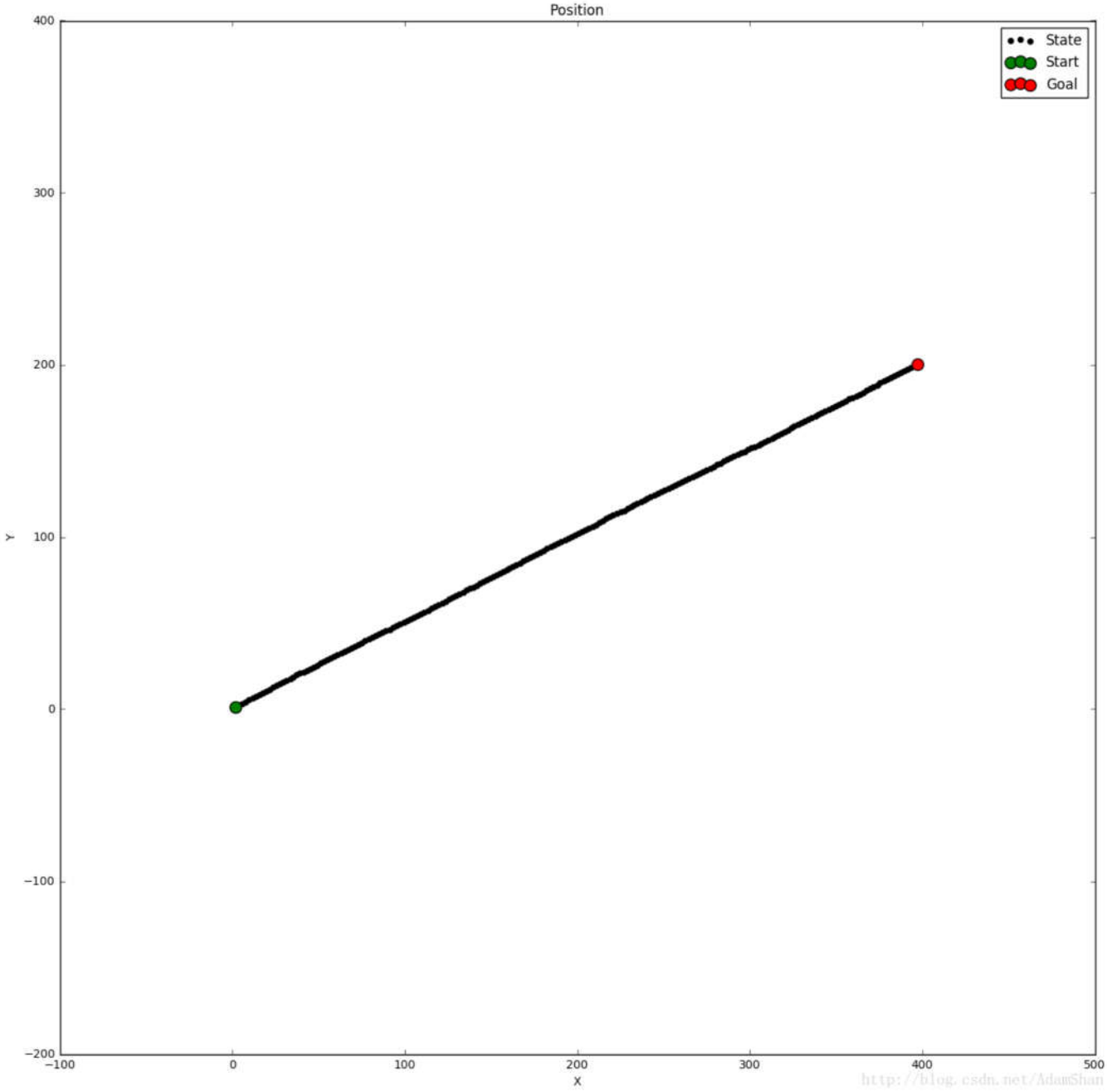
预测的结果如图所示：



位置的估计结果：

```
def plot_xy():
    fig = plt.figure(figsize=(16,16))
    plt.scatter(xt,yt, s=20, label='State', c='k')
    plt.scatter(xt[0],yt[0], s=100, label='Start', c='g')
    plt.scatter(xt[-1],yt[-1], s=100, label='Goal', c='r')

    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Position')
    plt.legend(loc='best')
    plt.axis('equal')
plot_xy()
```



至此，最最简单的卡尔曼滤波以及其在无人驾驶汽车的感知模块中的应用就完成了。

这篇博客仅仅只讲了卡尔曼滤波的基本形式，并不是实际无人车项目中的所使用的技术，但是其基本原理与我们目前在Google无人车感知模块的技术是相通的。

虽然本例中以无人驾驶的行人检测作为落脚点讲解卡尔曼滤波，但是实际上我们的无人车并不会仅仅使用原始的卡尔曼滤波作为行人感知的状态估计，因为卡尔曼滤波存在着一个非常大的局限性——它仅能对线性的处理模型和测量模型进行精确的估计，在非线性的场景中并不能达到最优的估计效果，为了能够设定线性的环境，我们假定了我们的处理模型为恒定速度模型，但是显然在显示的应用中不是这样的，不论是处理模型还是测量模型，都是非线性的

下一篇我将介绍一种能够应用于非线性情况的卡尔曼滤波算法——扩展卡尔曼滤波。

参考：