0.1 前置介绍

卡尔曼滤波用于滤除波形中的噪声，例如传感器传回的数据中可能存在环境噪声，产生不利于分析的抖动，卡尔曼滤波通过对线性系统的预测值来补偿这些误差，使得数据曲线整体更更加平滑。

理想状态是把输入的波形变为低通滤波（信号权值为 1，噪声权值为 0，信号与噪声分别为低频与高频）

卡尔曼滤波的本质是通过调节估计值（算法根据系统状态转移过程计算得到）与观测值（实际接收到的传感器数值）的权重，得到一个准确度高的修正值

0.2 适用情景

卡尔曼滤波适用于线性高斯系统

线性：满足可加性与可乘性/叠加性与齐次性；高斯：噪声的随机分布满足正态分布，表现为高斯白噪声，均值为 0

这是图片：

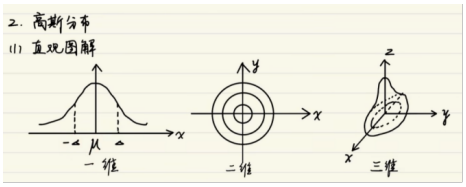

图 1: 高斯分布示意图

0.3 算法过程

卡尔曼滤波算法的实现过程：使用上一时刻的最优结果（修正值）预测当前的值（状态转移矩阵计算得到），同时使用观测结果（传感器的实际输出）修正当前的值，得到最终结果，即当前时刻的修正值

接下来会用到五条公式：

- 状态预测：$\hat{x}_t^- = F\hat{x}_{t-1} + Bu_t$
- 协方差预测：$P_t^- = FP_{t-1}F^T + Q$
- 状态更新：$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-)$
- 卡尔曼增益计算：$K_t = P_t^- H^T(HP_t^- H^T + R)^{-1}$
- 协方差更新：$P_t = (I - K_k H)P_t^-$

具体的公式，推导过程简要了解即可，重点在于理解每个变量的意思，并编程还原这一过程。下面具体介绍这些公式

0.3.1 状态预测：$\hat{x}_t^- = F\hat{x}_{t-1} + Bu_t$

这条公式描述了上一个状态推演得到当前状态的过程，右上角带负号表示先验估计值，即没有观测值修正的纯粹的估计值，F 为状态转移矩阵，B 为控制矩阵，Bu_t 表示控制输入

以匀速直线运动为例，这条公式的具体意义如下：

$$\begin{array}{ccccccc} \begin{bmatrix} P_i \\ v_i \end{bmatrix} & = & \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} P_{i-1} \\ v_{i-1} \end{bmatrix} & + & \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} a_i \\ \downarrow & & \downarrow & \downarrow & & \downarrow & \\ \hat{x}_t^- & = & F & \hat{x}_{t-1} & + & Bu_t & \end{array}$$

0.3.2 协方差预测：$P_t^- = FP_{t-1}F^T + Q$

前置知识：cov 表示协方差（协方差可描述每个量各自的离散程度，以及各个量之间的相关程度），var 表示一维方差（中学讲的那个方差），A 是一个矩阵：

$$cov(x, x) = var(x)$$

$$cov(Ax, Ay) = Acov(x, y)A^T$$

P_t^- 展开来其实是 $cov(\hat{x}_t^-, \hat{x}_t^-)$，也就是 $\hat{x}_t^-$ 的先验估计协方差，把 P_{t-1} 左乘、右乘状态转移矩阵 F，相当于

$$cov(F\hat{x}_{t-1} + Bu_{t-1}, F\hat{x}_{t-1} + Bu_{t-1})$$

也就是把公式一写到了协方差里面，描述了协方差的转移过程。（其实再算一遍协方差也是一样的...）

注意到这里多了一个 Q，表示的是过程噪声方差，显然过程噪声方差会对预测的协方差产生影响，所以公式里带上了 Q。这里的 Q 是和 K_p, K_i, K_d 一样的参数，需要根据实际情况不断调整。这也好理解，因为不同的实际情况对应的噪声大小肯定不同。

0.3.3 状态更新：$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-)$

补充知识：z 表示传感器传回来的数值

H 可以理解为一个“掩膜矩阵”：状态 x 中无法被传感器测量的量在矩阵相乘时变为 0（z 中本来就没有这一项，相减是无意义的），传感器可以测得的量相乘不变

这条公式描述了后验估计值（最终滤波结果/修正值）的生成过程，也就是先验估计值与观测值取加权平均。这里的权值 K($0 \leq K \leq 1$) 称为卡尔曼增益，其计算方法下面会给出。观测值与预测值之差，也就是 $z_t - H\hat{x}_t^-$，乘上比例系数 K，得到的 $\hat{x}_t$ 是一个介于观测值与预测值之间的修正值。

0.3.4 卡尔曼增益计算：$\frac{K_t = P_t^- H^T}{H P_t^- H^T + R}$

这条公式描述了卡尔曼增益的计算过程，其中 R 代表观测噪声方差，和 Q 一样是需要手调的参数

假设 H 为一维向量（即传感器只传回了一个量，状态转移也只考虑一个量），那么可以简化如下（只是为了方便理解）

$$K_t = \frac{P_t^-}{P_t^- + R} = \frac{P_{t-1} + Q}{P_{t-1} + Q + R}$$

可以看到复杂的 H 左乘右乘只是为了处理多维向量的情况，不影响整体的理解。

本质就是计算先验协方差加上观测方差后，先验协方差所占比例，如果占比大，说明观测噪声小，此时修正值更倾向于取观测值（噪声小，更准确），反之取预测值会更准确。

0.3.5 协方差更新：$P_t = (I - K_k H)P_t^-$

这条公式描述了后验协方差的生成过程，P_t 即是修正值（最终结果）的协方差，其实没什么好理解的，单纯是修正值套公式 cov(x_t,x_t) 算出来的，

只是为了方便计算，给它单独写了一个转移公式。重点在于理解为什么要转移：转移后的式子融合了观测值在里面，也就是当前的观测值会对后面的协方差计算产生影响。

0.4 总结

总体顺下来还是很好理解的，x 的转移与 P 的转移双线并行，P 用于计算增益权值，x 用于计算结果，“转移”包括了预测与观测两个步骤，最终二者加权融合得出了当前阶段的结果，留给下一阶段使用。

0.5 使用指南

主要调节两个超参数：Q 和 R（类似于 PID 中的 P,I,D 三个比例项，这种需要根据实际环境调节的参数就被称为超参数）

Q 是过程噪声方差

R 是观测噪声方差

假设一个情境，需要测量平衡车的俯仰角，控制量是该时刻内电机旋转过的角度（直立静止时为 0），观测量是 6 轴传感器的加速度轴与角速度轴互补滤波后输出的值，对其进行卡尔曼滤波算法优化

```
#include <stdint.h>
#include <math.h>

// 卡尔曼滤波状态结构体（单状态量，简化设计）
typedef struct {
    double x;           // 状态估计值：俯仰角 （单位：rad）
    double P;           // 状态协方差（估计误差的方差）
    double Q;           // 过程噪声协方差（需调试，初始建议0
        .001~0.01）
    double R;           // 观测噪声协方差（需调试，初始建议0
        .1~0.5）
    double B;           // 控制输入系数（电机角度→俯仰角的转换系
        数，需标定）
} KalmanFilter;

/**
 * @brief  卡尔曼滤波初始化
```

```
 * @param  kf: 卡尔曼滤波结构体指针
 * @param  init_theta: 初始俯仰角（互补滤波的初始输出，单位: rad）
 * @param  Q: 过程噪声协方差（建议0.001~0.01）
 * @param  R: 观测噪声协方差（建议0.1~0.5）
 * @param  B: 控制系数（电机旋转角度→俯仰角的转换系数，需标定）
 */
void KalmanFilter_Init(KalmanFilter *kf, double init_theta,
    double Q, double R, double B) {
    kf->x = init_theta;  // 初始状态为互补滤波的初始值
    kf->P = 0.1f;        // 初始估计误差协方差（经验值）
    kf->Q = Q;
    kf->R = R;
    kf->B = B;
}

/**
 * @brief  卡尔曼滤波迭代（核心函数）
 * @param  kf: 卡尔曼滤波结构体指针
 * @param  z: 观测值（加速度计+陀螺仪互补滤波结果，单位: rad）
 * @param  u: 控制量（电机旋转角度，单位: rad，需与小车机械结构匹配）
 * @return 修正后的俯仰角（单位: rad）
 */
double KalmanFilter_Update(KalmanFilter *kf, double z, double u
    ) {
    // 1. 预测阶段: 根据控制量预测当前状态
    double x_pred = kf->x + kf->B * u;  // 状态预测（线性控制模型: _new = _old + B*电机角度）
    double P_pred = kf->P + kf->Q;      // 协方差预测（过程噪声叠加）

    // 2. 更新阶段: 根据观测值修正预测值
    double K = P_pred / (P_pred + kf->R);  // 卡尔曼增益（权衡预测与观测的可信度）
    kf->x = x_pred + K * (z - x_pred);     // 状态修正（观测残差加权更新）
    kf->P = (1 - K) * P_pred;              // 协方差修正
```

```
    return kf->x;  // 输出修正后的俯仰角
}
```

以上 AI 辅助生成的参考代码，要应用到其他情境，只需调整状态转移矩阵、Q、R 值即可。下面给出接口函数：

```
    void KalmanFilter_Init(KalmanFilter *kf, double init_theta,
         double Q, double R, double B)
```

这个是卡尔曼滤波参数结构体初始化函数，传入参数：

*kf 卡尔曼滤波参数结构体，每一个需要滤波的数据单独使用一个结构体参数组

init_theta 初始估计值，也就是一开始的俯仰角

Q 过程噪声协方差，参考值 0.001 0.01

R 观测噪声协方差，参考值 0.1 0.5

B 控制量变换向量，控制量–> 与变换向量相乘–> 变换为状态向量的同型矩阵，两者可相加。这里只有一维向量，直接给 1 就可以

```
    double KalmanFilter_Update(KalmanFilter *kf, double z,
        double u)
```

*kf 同上

z 观测值，也就是互补滤波计算的结果

u 控制量，也就是电机转动的角度，电机转动-> 姿态角改变，所以用于预测值估计。