关于梯度下降的部分

已知

$$\omega = Asin(Bt) + C \tag{1}$$

算法按照如下步骤进行:

1. 连续观测4个θ，记为$\theta_1, \theta_2, \theta_3, \theta_4$。可求得$\omega_0 = \frac{\theta_2 - \theta_1}{2}, w_1 = \frac{\theta_4 - \theta_3}{2}$。由于$\theta$的观测存在误差，因此需要检查$|\frac{\omega - C}{A}|$是否在$[0, 1]$范围内。
2. 将ω_1代入式(1)，反解出$t = \frac{1}{B}arcsin(\frac{\omega - C}{A})$，记为$\hat{t}$。由于$arcsin(x)$的值一定位于$[-\frac{\pi}{2}, \frac{\pi}{2}]$之间，所以还要对结果做一些处理才能得到正确的$\hat{t}$（画画图，这个不好说）。
3. $\hat{t}$存在误差，因此引入参数α来修正$\hat{t}$。在计算$\hat{t}$的同时，记录系统时间T_0，以及初始角度$\theta_0 = \theta_4$。初始角度也存在误差，引入参数β来修正初始角度。
4. 对于以后的任意系统时间T，可以通过积分求得预测的θ，具体如下:

$$\begin{aligned}
\omega(\Delta t) &= Asin(B(\hat{t} + \Delta t + \alpha)) + C, \quad \Delta t = T - T_0 \\
\theta(\Delta t) &= \theta_0 + \beta + \int_0^{\Delta t} w(x)dx \\
&= \theta_0 + \beta + \int_0^{\Delta t} (Asin(B(\hat{t} + x + \alpha)) + C)dx \\
&= \theta_0 + \beta + (-\frac{A}{B}cos(B(\hat{t} + x + \alpha)) + Cx)\Big|_0^{\Delta t} \\
&= \theta_0 + \beta + \frac{A}{B}(cos(B(\hat{t} + \alpha)) - cos(B(\hat{t} + \Delta t + \alpha))) + C\Delta t
\end{aligned}$$

5. 设系统时间为T时，观测到的(也就是识别出来的)$\theta = \theta_{observe}$。此时预测的$\theta = \theta(\Delta t) = \theta_{predict}$，记损失为

$$Loss = \frac{1}{2}(\theta(\Delta t) - \theta_{predict})^2$$

   损失是由θ_0与$\hat{t}$的误差带来的，通过调整α与β来降低这个损失。首先求出$Loss$关于α与β的梯度。

$$\begin{aligned}
\frac{\partial Loss}{\partial \alpha} &= \frac{\partial Loss}{\partial(\theta_{predict})} * \frac{\partial(\theta_{predict})}{\partial \alpha} \\
&= (\theta_{predict} - \theta_{observe}) \times (Asin(B(\hat{t} + \Delta t + \alpha)) - Asin(B(\hat{t} + \alpha))) \\
&= (\theta_{predict} - \theta_{observe}) \times A(sin(B(\hat{t} + \Delta t + \alpha)) - sin(B(\hat{t} + \alpha))) \\
\frac{\partial Loss}{\partial \beta} &= \frac{\partial Loss}{\partial \theta_{predict}} \times \frac{\partial \theta_{predict}}{\partial \beta} \\
&= (\theta_{predict} - \theta_{observe}) \times 1 \\
&= \theta_{predict} - \theta_{observe}
\end{aligned}$$

   梯度指向函数值上升最快的方向，负梯度指向函数值下降最快的方向，利用梯度来更新两个参数:

$$\begin{aligned}
new_\alpha &= \alpha + step \times (-\frac{\partial Loss}{\partial \alpha}) \\
new_\beta &= \beta + step \times (-\frac{\partial Loss}{\partial \beta})
\end{aligned}$$

   这样就完成了参数更新。理论上每一帧都可以更新一次，因为事先计算好公式以后，上述操作时间都很短。**梯度下降的前提是：观测到的$\theta_{observe}$是准确的，否则梯度就不准确。**

关于卡尔曼滤波的部分

1. 状态向量为：

$$\overrightarrow{x_k} = [\theta, \Delta t]^T$$

   其中θ是系统时间为T时观测所得到的角度，Δt是下一帧到达所需要的时间。

2. 转移矩阵为：

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

   其中ω_k是用上面那个ω关于Δt的式子估计出来的T时刻角速度。原本转移矩阵A应该是定值，这边做了个尝试，让它是变动的。因为只是改变了矩阵的一个元素，所以基本上不耗时间。

3. 观测向量为：

$$\overrightarrow{z_k} = [\theta_{observe}, \Delta t_{observe}]^T。$$

   其中$\theta_{observe}$是观测到的角度，$\Delta t_{observe}$是根据系统时间计算出的真正的帧间隔时间。

4. 将状态映射为观测的矩阵H为：

$$H = Identity(2, 2)$$

   是一个2×2的单位矩阵。

5. 状态向量的协方差矩阵P为：

$$P = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.1 \end{bmatrix}$$

   0.2跟0.1瞎给的= =。反正P会被修正。

6. 观测值的噪声R为：

$$R = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.01 \end{bmatrix}$$

   也是瞎给的。感觉$\theta_{observe}$的方差会比$\Delta t_{observe}$的方差大一个数量级。毕竟Δt本来就很小。

7. 由于外部控制所带来的噪声Q为：0。因为在状态转移时无外部控制。

最后卡尔曼预测所需的x个方程如下：

$$[\theta_k, \Delta t_k]^T = \begin{bmatrix} 1 & \omega_k \\ 0 & 1 \end{bmatrix} [\theta_{k-1}, \Delta t_{k-1}]^T + B\vec{u}, \vec{u} = 0 \tag{1}$$

$$P_k = \begin{bmatrix} 1 & \omega_k \\ 0 & 1 \end{bmatrix} P_{k-1} \begin{bmatrix} 1 & \omega_k \\ 0 & 1 \end{bmatrix}^T + Q, Q = 0 \tag{2}$$

经过(1),(2)两次计算，可以得到对状态的初步估计。

$$[\theta_{observe}, \Delta t_{observe}]^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} [\theta_{k-1}, \Delta t_{k-1}]^T \tag{3}$$

$$\begin{aligned} K_k &= P_k H^T (H P_k H^T + R)^{-1} \\ &= P_k (P_k + R)^{-1} \end{aligned} \tag{4}$$

$$\begin{aligned} \overrightarrow{x_k}_{best} &= \overrightarrow{x_k} + K_k(\overrightarrow{z_k} - H * \overrightarrow{x_k}) \\ &= \overrightarrow{x_k} + K_k(\overrightarrow{z_k} - \overrightarrow{x_k}) \end{aligned} \tag{5}$$

$$P_k = (I - K_k H)P = (I - K_k)P \tag{6}$$

经过(4),(5),(6)之后，可以对初步估计进行修正。

整个算法的运行流程：

1. 识别出大风车扇叶上的装甲（同时得到角度），开始截取连续4帧角度，初步估算$\hat{t}$。并用最后一帧的相关信息（例如角度，角速度）来初始化卡尔曼滤波器。同时记录系统时间。
2. 进入训练模式。每一帧都获取一次角度观测值。首先用卡尔曼滤波器进行数据融合，然后用融合的结果作为梯度下降中的$\theta_{observe}$，来进行梯度下降，我觉得会准确一些。
3. 其它细节还没想，比如说训练需要的帧数是否还能再降低，误差是否服从正态分布等等，也不急~

代码参数设置：

```
(1) 观测到的theta的噪声方差设置为0.2。
main.cpp 68:  theta[idx] = t2theta(init_theta, t) + get_noise(0.2);
main.cpp 105: double theta_view = theta_real + get_noise(0.2);
(2) 卡尔曼滤波器初始噪声就是上面所提到的。
main.cpp 92:
    k.init_filter(init_theta + beta, omega_1, frame_gap, 0.1, 0.01);
```

代码运行结果：

在使用上述参数以及算法下，经过300帧，平均的绝对值误差大约为在0.05rad左右，也就是3°左右。还是有点高(..... 唉)。但跟不进行修正相比，已经改正很多了。而且在卡尔曼滤波的帮助下梯度下降不怎么受观测值误差的影响了。