KALMAN FILTER EXAMPLE

1. **Find the mathematical model of the system.** The first thing to do is to find out the state vector you want to estimate $\vec{x}$ and the inputs $\vec{u}$ of the system. After this, find the non-linear process function f that describes the evolution of the state vector through time, that is:

$$\vec{x}_k = f(x_{k-1}, u_{k-1}, w_{k-1}),$$

where w is the process noise vector due to uncertainty and process modeling errors. Then, find the non-linear relation between your state vector $\vec{x}$ and the measure vector $\vec{z}$:

$$\vec{z}_k = h(x_{k-1}, u_{k-1}),$$

where v is the measure noise vector.

In this example: a plane flies in a 2D space where the x axis is the distance traveled by the plane and y axis is its altitude. This system can be represented by the following differential equations:

$$\ddot{x} = u/m - b_x/m \cdot \dot{x}^2,$$

$$\ddot{y} = p/m \cdot \dot{x}^2 - g,$$

where m is the plane's weight ($1000\,kg$), b_x is the drag coefficient ($0.35\ N/m^2/s^2$), p is the lift force ($3.92\ N/m^2/s^2$), g is the gravitational acceleration ($9.8\ m/s^2$) and u (the input) is the motor's thrust.

The discrete equation is:

$$\vec{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \\ y_k \\ \dot{y}_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + T\dot{x}_{k-1} + T^2/2 \cdot (u/m - b_x/m \cdot \dot{x}^2{}_{k-1}) \\ \dot{x}_{k-1} + T \cdot (u/m - b_x/m \cdot \dot{x}^2{}_{k-1}) + w_1 \\ y_{k-1} + T\dot{y}_{k-1} + T^2/2 \cdot (p/m \cdot \dot{x}^2{}_{k-1} - g) \\ \dot{y}_{k-1} + T \cdot (p/m \cdot \dot{x}^2{}_{k-1} - g) + w_2 \end{bmatrix},$$

where w_1 and w_2 are the random variables which represent the process noise.

A station on the ground (at the origin) measures the angle between the plane and the ground (x axis) and the distance between the plane and the station. These measures are based on the following equation:

$$\vec{z}_k = \begin{bmatrix} \theta \\ r \end{bmatrix} = \begin{bmatrix} atan(y/x) + v_1 \\ \sqrt{x^2 + y^2} + v_2 \end{bmatrix},$$

where v_1 and v_2 are the random variables which represent the process noise.

2. **Calculate Jacobian matrix.** The second step is the calculation Jacobian matrices A, W, H and V where:

   A is an n by n Jacobian matrix of partial derivatives, defined as follow:

   $$A_{[i,j]} = \partial f_{[i]} / \partial x_{[j]},$$

   W is an n by nw Jacobian matrix of partial derivatives, defined as follow:

   $$W_{[i,j]} = \partial f_{[i]} / \partial w_{[j]},$$

   H is an m by n Jacobian matrix of partial derivatives, defined as follow:

   $$H_{[i,j]} = \partial h_{[i]} / \partial x_{[j]},$$

   V is an m by nv Jacobian matrix of partial derivatives, defined as follow:

   $$V_{[i,j]} = \partial h_{[i]} / \partial v_{[j]},$$

   n is the number of elements in state vector, m is the number of measures, nw is the number of process noise random variables, nv is the number of measure noise random variables.

   In this example:

$$A = \begin{bmatrix} 1 & T - T^2 \cdot b/m \cdot \dot{x}_{k-1} & 0 & 0 \\ 0 & 1 - 2 \cdot T \cdot b/m \cdot \dot{x}_{k-1} & 0 & 0 \\ 0 & T^2 \cdot p/m \cdot \dot{x}_{k-1} & 1 & T \\ 0 & 2 \cdot T \cdot p/m \cdot \dot{x}_{k-1} & 0 & 1 \end{bmatrix},$$

$$W = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix},$$

$$H = \begin{bmatrix} -y_{k-1}/(x_{k-1}^2 + y_{k-1}^2) & 0 & x_{k-1}/(x_{k-1}^2 + y_{k-1}^2) & 0 \\ x_{k-1}/\sqrt{x_{k-1}^2 + y_{k-1}^2} & 0 & y_{k-1}/\sqrt{x_{k-1}^2 + y_{k-1}^2} & 0 \end{bmatrix},$$

$$V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

3. **Initial conditions and covariance matrix.** The next step is to set initial estimation of the state vector. After, set the covariance matrix P which represents the covariance of the error of the state vector estimation. Then, set the covariance Q and R which represent the covariance matrix of process noise and measurement noise, respectively.

   In this example:

$$\vec{x} = \begin{bmatrix} r\cos\theta \\ 60 \\ r\sin\theta \\ 0 \end{bmatrix},$$

$$P = \begin{bmatrix} 100^2 & 0 & 0 & 0 \\ 0 & 10^2 & 0 & 0 \\ 0 & 0 & 25^2 & 0 \\ 0 & 0 & 0 & 10^2 \end{bmatrix},$$

$$Q = \begin{bmatrix} 0.01^2 & 0.01^2/10 \\ 0.01^2/10 & 0.01^2 \end{bmatrix},$$

$$R = \begin{bmatrix} 0.01^2 & 0 \\ 0 & 0.01^2 \end{bmatrix}.$$

4. **Implementation of the Kalman filter.** The last step is to create the implementation of the Kalman filter. So, write functions called `makeProcess`, `makeMeasure`, `makeA`, `makeH`, `makeQ`, `makeR`, `makeV` and `makeW`. These functions will set the value of each matrix.

   The first thing to do is to create Kalman filter class.

```
class PlaneFilter : public Kalman::Extended<double>
{
public:
    PlaneFilter();

protected:
    void makeA();
    void makeH();
    void makeV();
    void makeR();
    void makeW();
    void makeQ();
    void makeProcess();
    void makeMeasure();

protected:
    double Period;
    double Mass;
    double Drag;
    double Lift;
    double Gravity;
};
```

   In this example, our Kalman filter inherits from the Extended Kalman Filter, because it's a non-linear problem. You should declare each functions named previously in this class. You can declare variables too.

   After, write the class constructor. You should call the functions `setSize<*>` here. These functions sets the number of states, the number of inputs, the number of process noise random variables, the number of measures and the number of measurement noise random variables.

```
PlaneFilter::PlaneFilter()
{
    setSizeX(4);
    setSizeU(1);
    setSizeW(2);
    setSizeZ(2);
    setSizeV(2);

    Period = 0.2;
    Mass = 1000;
    Drag = 0.35;
    Lift = 3.92;
    Gravity = 9.8;
}
```

In the function `makeProcess`, you should use a temporary vector to store the new state vector like this:

```
void PlaneFilter::makeProcess()
{
    Kalman::Vector<double> x_(x.Size());
    x_(0) = x(0) + x(1) * Period + (Period * Period) / 2 * (u(0) /
        Mass - Drag / Mass * x(1) * x(1));
    x_(1) = x(1) + (u(0) / Mass - Drag / Mass * x(1) * x(1)) * Period;
    x_(2) = x(2) + x(3) * Period + (Period * Period) / 2 *
        (Lift / Mass * x(1) * x(1) - Gravity);
    x_(3) = x(3) + (Lift / Mass * x(1) * x(1) - Gravity) * Period;
    x.Swap(x_);
}
```

In the function `makeMeasure`, you update directly the measures vector $\vec{z}$. These are the predicted measures.

```
void PlaneFilter::makeMeasure()
{
    z(0) = atan2(x(2), x(0));
    z(1) = sqrt(x(0) * x(0) + x(2) * x(2));
}
```

Then, write all other functions `make<*>` like this:

```
void PlaneFilter::makeA()
{
    A(0,0) = 1.0;
    A(0,1) = Period - Period * Period * Drag / Mass * x(1);
    A(0,2) = 0.0;
    A(0,3) = 0.0;

    A(1,0) = 0.0;
    A(1,1) = 1 - 2 * Period * Drag / Mass * x(1);
    A(1,2) = 0.0;
    A(1,3) = 0.0;

    A(2,0) = 0.0;
    A(2,1) = Period * Period * Lift / Mass * x(1);
    A(2,2) = 1.0;
    A(2,3) = Period;

    A(3,0) = 0.0;
    A(3,1) = 2 * Period * Lift / Mass * x(1);
    A(3,2) = 0.0;
    A(3,3) = 1.0;
}
```

```
void PlaneFilter::makeW()
{
    W(0,0) = 0.0;
    W(0,1) = 0.0;

    W(1,0) = 1.0;
    W(1,1) = 0.0;

    W(2,0) = 0.0;
    W(2,1) = 0.0;

    W(3,0) = 0.0;
    W(3,1) = 1.0;
}
```

```
void PlaneFilter::makeQ()
{
    Q(0,0) = 0.01 * 0.01;
    Q(0,1) = 0.01 * 0.01/10.0;

    Q(1,0) = 0.01 * 0.01 / 10.0;
    Q(1,1) = 0.01 * 0.01;
}
```

```
void PlaneFilter::makeH()
{
    H(0,0) = -x(2) / (x(0) * x(0) + x(2) * x(2));
    H(0,1) = 0.0;
    H(0,2) = x(0) / (x(0) * x(0) + x(2) * x(2));
    H(0,3) = 0.0;

    H(1,0) = x(0) / sqrt(x(0) * x(0) + x(2) * x(2));
    H(1,1) = 0.0;
    H(1,2) = x(2) / sqrt(x(0) * x(0) + x(2) * x(2));
    H(1,3) = 0.0;
}
```

```
void PlaneFilter::makeV()
{
    V(0,0) = 1.0;
    V(0,1) = 0.0;

    V(1,0) = 0.0;
    V(1,1) = 1.0;
}
```

```
void PlaneFilter::makeR()
{
    R(0,0) = 0.01 * 0.01;
    R(0,1) = 0.0;

    R(1,0) = 0.0;
    R(1,1) = 50 * 50;
}
```

Now, filter is ready to be used. In this example, the measures and the inputs are stored in `data.dat` file. The `init` function sets the initial state and the initial covariance matrix. Call the function `step` for each iteration and pass the new inputs and the new measures.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include "plane.h"

using namespace std;
using namespace Kalman;

int main()
{
    const unsigned count = 500;
    const unsigned n = 4; // states
    const unsigned m = 2; // measures

    const double raw[] = { 100.0 * 100.0, 0.0, 0.0, 0.0,
                           0.0, 10.0 * 10.0, 0.0, 0.0,
                           0.0, 0.0, 25.0 * 25.0, 0.0,
                           0.0, 0.0, 0.0, 10.0 * 10.0 };

    ifstream input;
    PlaneFilter filter;

    Vector<double> x(n);
    Vector<double> z(m);
    Matrix<double> p(n, n, raw);

    // read the inputs vector and the measures matrix
    input.open("data.dat", ifstream::in);

    double fs[count] = {0.0};
    double ms[count * 2] = {0.0};

    for (double& fi : fs) input >> fi;
    for (double& mi : ms) input >> mi;

    Vector<double> f(count, fs);
    Matrix<double> measure(m, count, ms);

    // initialization
    x(0) = cos(measure(0,0)) * measure(1,0);
    x(1) = 60;
    x(2) = sin(measure(0,0)) * measure(1,0);
    x(3) = 0;

    filter.init(x, p);

    // process
    for (unsigned i = 1; i < count; ++i)
    {
        for(unsigned j = 0; j < m; ++j)
            z(j) = measure(j,i);

        Vector<double> u(1, f(i));

        filter.step(u, z);

        Vector<double> result = filter.getX();

        cout << result(0) << ' ' << result(1) << ' '
             << result(2) << ' ' << result(3) << endl;
    }

    return 0;
}
```