应用cv::KalmanFilter的一些实例

1、OpenCV自带的示例程序

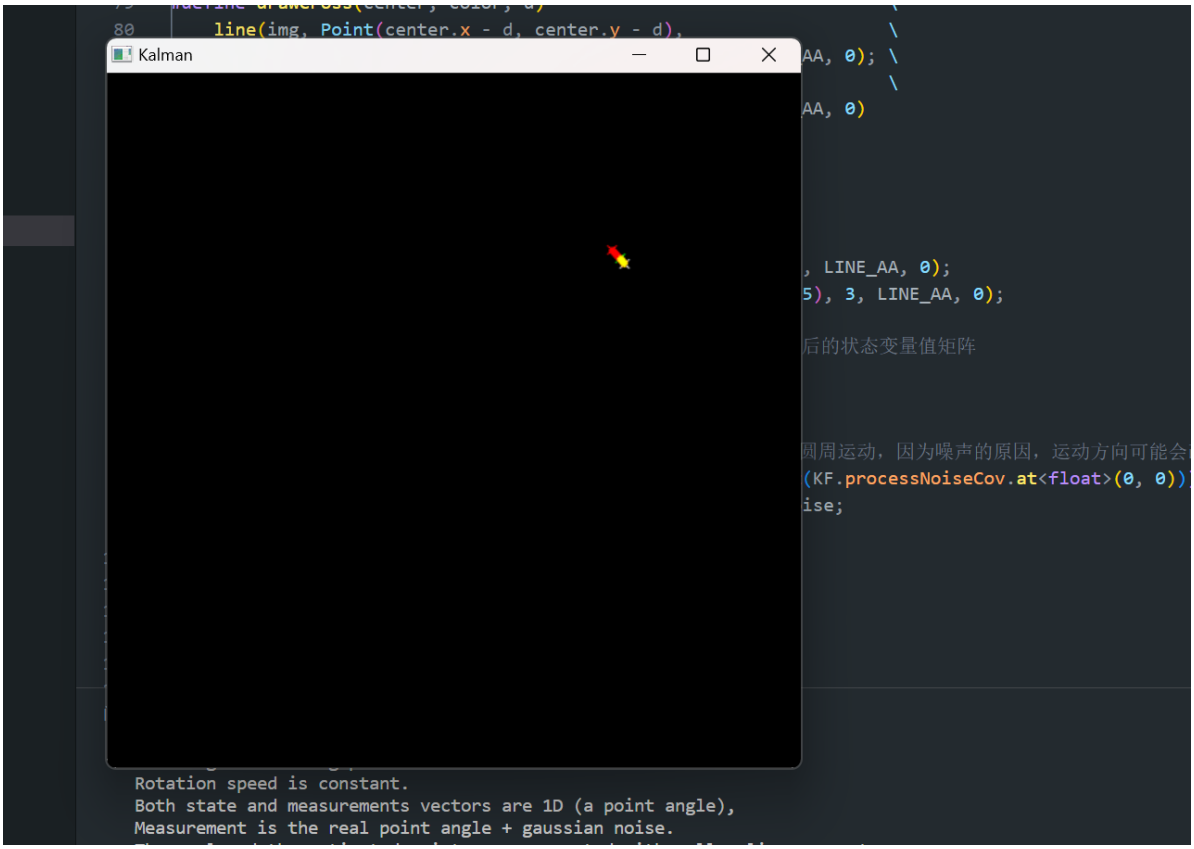
```
#include "opencv2/video/tracking.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
#include <stdio.h>
using namespace std;
using namespace cv;

// 计算相对窗口的坐标值，因为坐标原点在左上角，所以sin前有个负号
static inline Point calcPoint(Point2f center, double R, double angle)
{
    return center + Point2f((float)cos(angle), (float)-sin(angle)) * (float)R;
}

static void help()
{
    printf("\nExamle of c calls to OpenCV's Kalman filter.\n"
           "   Tracking of rotating point.\n"
           "   Rotation speed is constant.\n"
           "   Both state and measurements vectors are 1D (a point angle),\n"
           "   Measurement is the real point angle + gaussian noise.\n"
           "   The real and the estimated points are connected with yellow line
segment,\n"
           "   the real and the measured points are connected with red line
segment.\n"
           "   (if Kalman filter works correctly,\n"
           "    the yellow segment should be shorter than the red one).\n"
           "\n"
           "   Pressing any key (except ESC) will reset the tracking with a
different speed.\n"
           "   Pressing ESC will stop the program.\n");
}

int main(int, char **)
{
    help();
    Mat img(500, 500, CV_8UC3);
    KalmanFilter KF(2, 1, 0); // 创建卡尔曼滤波器对象KF
    Mat state(2, 1, CV_32F);  // state(角度，△角度)
    Mat processNoise(2, 1, CV_32F);
    Mat measurement = Mat::zeros(1, 1, CV_32F); // 定义测量值
    char code = (char)-1;
```

```
    for (;;)
    {
        // 1.初始化
        randn(state, Scalar::all(0), Scalar::all(0.1));          //
        KF.transitionMatrix = (Mat_<float>(2, 2) << 1, 1, 0, 1); // 转移矩阵
A[1,1;0,1]

        // 将下面几个矩阵设置为对角阵
        setIdentity(KF.measurementMatrix);                       // 测量矩阵H
        setIdentity(KF.processNoiseCov, Scalar::all(1e-5));      // 系统噪声方差矩阵
Q
        setIdentity(KF.measurementNoiseCov, Scalar::all(1e-1)); // 测量噪声方差矩阵
R
        setIdentity(KF.errorCovPost, Scalar::all(1));            // 后验错误估计协方
差矩阵P

        randn(KF.statePost, Scalar::all(0), Scalar::all(0.1)); // x(0)初始化

        for (;;)
        {
            Point2f center(img.cols * 0.5f, img.rows * 0.5f); // center图像中心点
            float R = img.cols / 3.f;                         // 半径
            double stateAngle = state.at<float>(0);           // 跟踪点角度
            Point statePt = calcPoint(center, R, stateAngle); // 跟踪点坐标statePt

            // 2. 预测
            Mat prediction = KF.predict();                          // 计算预测值，返
回x'
            double predictAngle = prediction.at<float>(0);          // 预测点的角度
            Point predictPt = calcPoint(center, R, predictAngle); // 预测点坐标
predictPt

            // 3.更新
            // measurement是测量值
            randn(measurement, Scalar::all(0),
Scalar::all(KF.measurementNoiseCov.at<float>(0))); // 给measurement赋值N(0,R)的随机
值

            // generate measurement
            measurement += KF.measurementMatrix * state; // z = z + H*x;

            double measAngle = measurement.at<float>(0);
            Point measPt = calcPoint(center, R, measAngle);

// plot points
// 定义了画十字的方法，值得学习下
#define drawCross(center, color, d)                                 \
    line(img, Point(center.x - d, center.y - d),                    \
         Point(center.x + d, center.y + d), color, 1, LINE_AA, 0); \
    line(img, Point(center.x + d, center.y - d),                    \
         Point(center.x - d, center.y + d), color, 1, LINE_AA, 0)

            img = Scalar::all(0);
            drawCross(statePt, Scalar(255, 255, 255), 3);
```

```
            drawCross(measPt, Scalar(0, 0, 255), 3);
            drawCross(predictPt, Scalar(0, 255, 0), 3);
            line(img, statePt, measPt, Scalar(0, 0, 255), 3, LINE_AA, 0);
            line(img, statePt, predictPt, Scalar(0, 255, 255), 3, LINE_AA, 0);

            // 调用kalman这个类的correct方法得到加入观察值校正后的状态变量值矩阵
            if (theRNG().uniform(0, 4) != 0)
                KF.correct(measurement);

            // 不加噪声的话就是匀速圆周运动，加了点噪声类似匀速圆周运动，因为噪声的原因，运动方向可能会改变
            randn(processNoise, Scalar(0),
Scalar::all(sqrt(KF.processNoiseCov.at<float>(0, 0)))); // vk
            state = KF.transitionMatrix * state + processNoise;

            imshow("Kalman", img);
            code = (char)waitKey(100);

            if (code > 0)
                break;
        }
        if (code == 27 || code == 'q' || code == 'Q')
            break;
    }

    return 0;
}
```

程序结果

上述程序就是cv自带的通过卡尔曼滤波实时跟踪目标点的位置

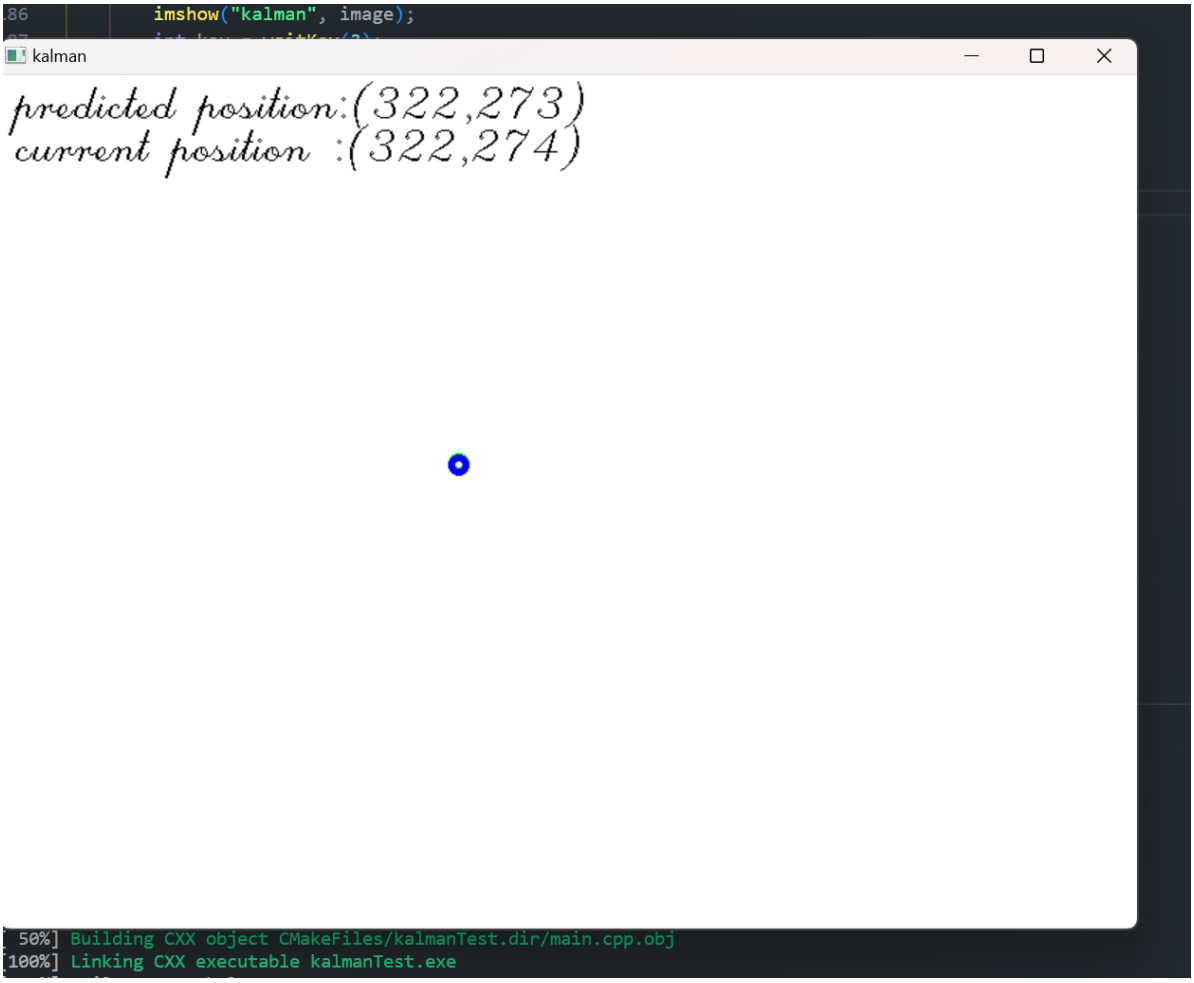


2、跟踪鼠标位置

```
#include "opencv2/video/tracking.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdio.h>
using namespace cv;
using namespace std;

const int winHeight = 600;
const int winWidth = 800;

#include "opencv2/video/tracking.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdio.h>

using namespace cv;
using namespace std;

Point mousePosition = Point(winWidth >> 1, winHeight >> 1);

// mouse event callback
void mouseEvent(int event, int x, int y, int flags, void *param)
{
    if (event == EVENT_MOUSEMOVE)
    {
        mousePosition = Point(x, y);
    }
}

int main(void)
{
    RNG rng;
    // 1.kalman filter setup
    const int stateNum = 4;   // 状态值4x1向量(x,y,△x,△y)
    const int measureNum = 2; // 测量值2x1向量(x,y)
    KalmanFilter KF(stateNum, measureNum, 0);

    KF.transitionMatrix = (Mat_<float>(4, 4) << 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,
0, 0, 0, 0, 1); // 转移矩阵A
    setIdentity(KF.measurementMatrix);
                // 测量矩阵H
    setIdentity(KF.processNoiseCov, Scalar::all(1e-5));
                 // 系统噪声方差矩阵Q
    setIdentity(KF.measurementNoiseCov, Scalar::all(1e-1));
                 // 测量噪声方差矩阵R
    setIdentity(KF.errorCovPost, Scalar::all(1));
                 // 后验错误估计协方差矩阵P
    rng.fill(KF.statePost, RNG::UNIFORM, 0, winHeight > winWidth ? winWidth :
winHeight);        // 初始状态值x(0)
    Mat measurement = Mat::zeros(measureNum, 1, CV_32F);
                // 初始测量值x'(0)，因为后面要更新这个值，所以必须先定义

    namedWindow("kalman");
    setMouseCallback("kalman", mouseEvent);
```

```
    Mat image(winHeight, winWidth, CV_8UC3, Scalar(0));

    while (1)
    {
        // 2.kalman prediction
        Mat prediction = KF.predict();
        Point predict_pt = Point(prediction.at<float>(0), prediction.at<float>
(1)); // 预测值(x',y')

        // 3.update measurement
        measurement.at<float>(0) = (float)mousePosition.x;
        measurement.at<float>(1) = (float)mousePosition.y;

        // 4.update
        KF.correct(measurement);

        // draw
        image.setTo(Scalar(255, 255, 255, 0));
        circle(image, predict_pt, 5, Scalar(0, 255, 0), 3);    // predicted
point with green
        circle(image, mousePosition, 5, Scalar(255, 0, 0), 3); // current
position with red

        char buf[256];
        sprintf_s(buf, 256, "predicted position:(%3d,%3d)", predict_pt.x,
predict_pt.y);
        putText(image, buf, Point(10, 30), FONT_HERSHEY_SCRIPT_COMPLEX, 1,
Scalar(0, 0, 0), 1, 8);
        sprintf_s(buf, 256, "current position :(%3d,%3d)", mousePosition.x,
mousePosition.y);
        putText(image, buf, Point(10, 60), FONT_HERSHEY_SCRIPT_COMPLEX, 1,
Scalar(0, 0, 0), 1, 8);

        imshow("kalman", image);
        int key = waitKey(3);
        if (key == 27)
        { // esc
            break;
        }
    }
}
```

结果



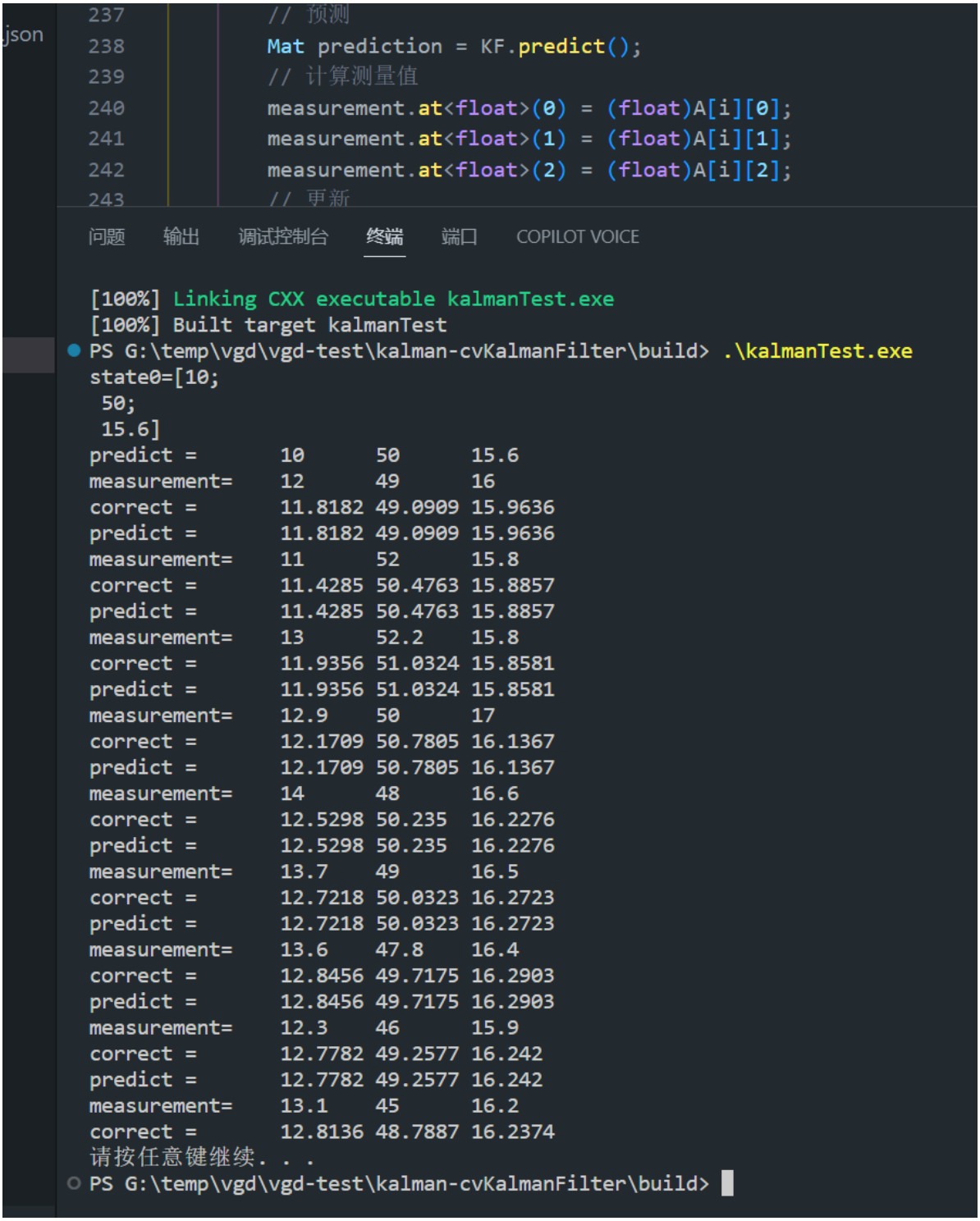
通过卡尔曼滤波实时计算鼠标坐标并实现跟踪

3、卡尔曼滤波过程解析示例

```
#include "opencv2/video/tracking.hpp"
// #include <opencv2/legacy/legacy.hpp> //#include "cvAux.h"
#include "opencv2/video/tracking.hpp"
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/core/core.hpp>
#include <stdio.h>
#include <iostream>
using namespace cv;
using namespace std;

int main()
{
    float A[10][3] =
        {
            10, 50, 15.6,
            12, 49, 16,
            11, 52, 15.8,
            13, 52.2, 15.8,
            12.9, 50, 17,
            14, 48, 16.6,
            13.7, 49, 16.5,
            13.6, 47.8, 16.4,
            12.3, 46, 15.9,
            13.1, 45, 16.2};
```

```
    const int stateNum = 3;
    const int measureNum = 3;
    KalmanFilter KF(stateNum, measureNum, 0);
    KF.transitionMatrix = (Mat_<float>(3, 3) << 1, 0, 0, 0, 1, 0, 0, 0, 1); // 转
移矩阵A
    setIdentity(KF.measurementMatrix);                                      // 测
量矩阵H
    setIdentity(KF.processNoiseCov, Scalar::all(1e-5));                     // 系
统噪声方差矩阵Q
    setIdentity(KF.measurementNoiseCov, Scalar::all(1e-1));                 // 测
量噪声方差矩阵R
    setIdentity(KF.errorCovPost, Scalar::all(1));
    Mat measurement = Mat::zeros(measureNum, 1, CV_32F);

    // 初始状态值
    KF.statePost = (Mat_<float>(3, 1) << A[0][0], A[0][1], A[0][2]);
    cout << "state0=" << KF.statePost << endl;

    for (int i = 1; i <= 9; i++)
    {
        // 预测
        Mat prediction = KF.predict();
        // 计算测量值
        measurement.at<float>(0) = (float)A[i][0];
        measurement.at<float>(1) = (float)A[i][1];
        measurement.at<float>(2) = (float)A[i][2];
        // 更新
        KF.correct(measurement);
        // 输出结果
        cout << "predict ="
             << "\t" << prediction.at<float>(0) << "\t" << prediction.at<float>
(1) << "\t" << prediction.at<float>(2) << endl;
        cout << "measurement="
             << "\t" << measurement.at<float>(0) << "\t" <<
measurement.at<float>(1) << "\t" << measurement.at<float>(2) << endl;
        cout << "correct ="
             << "\t" << KF.statePost.at<float>(0) << "\t" <<
KF.statePost.at<float>(1) << "\t" << KF.statePost.at<float>(2) << endl;
    }
    system("pause");
}
```

结果如下



这里预测值和上一个状态值一样，原因是转移矩阵A是单位阵，如果改成非单位阵，结果就不一样了。

上述可以参考着对kalman做过程调试检测