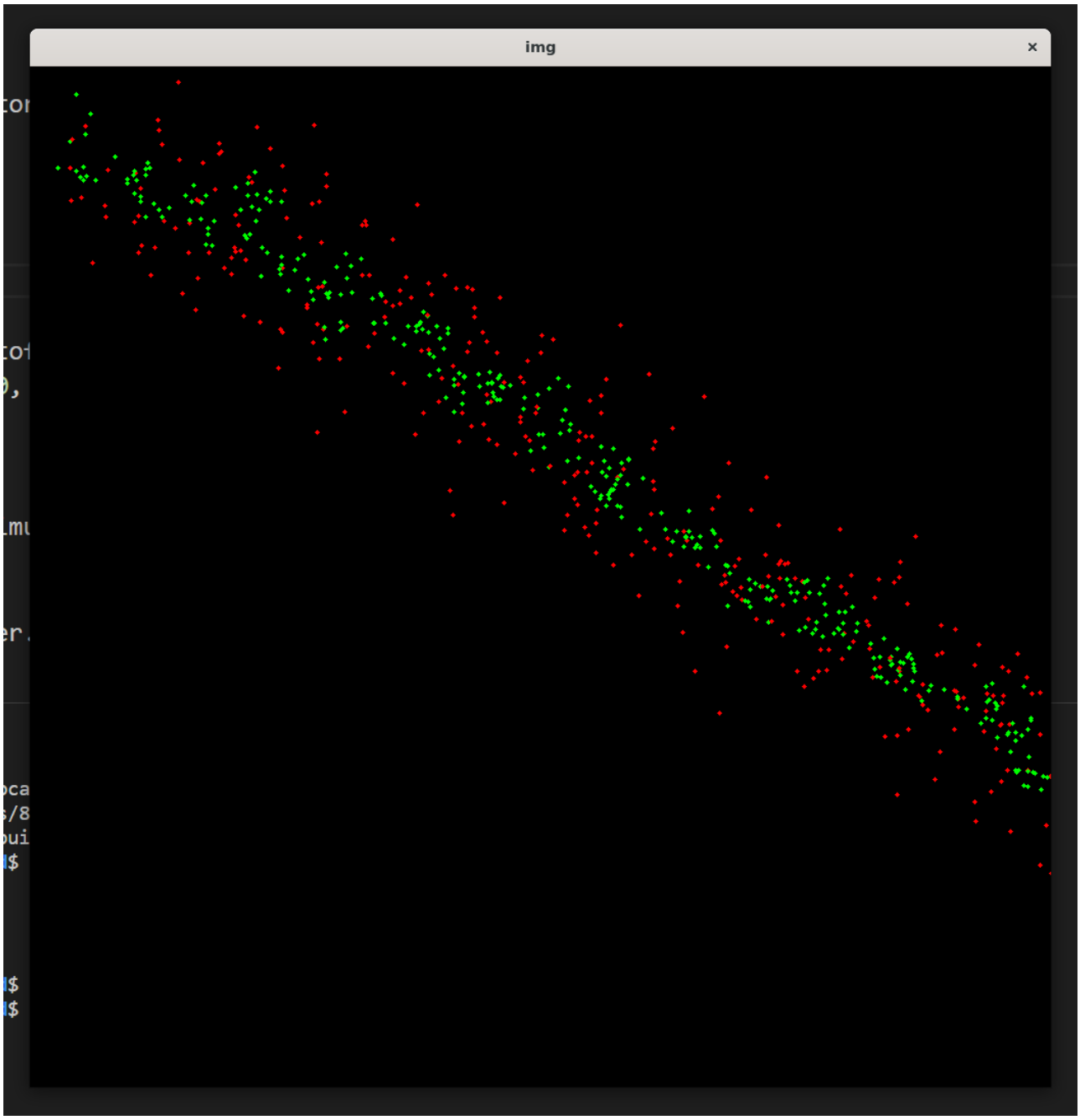
子任务一（低通滤波）

从零实现一个低通滤波器。

- 实现效果



> （对匀速直选运动进行模拟，然后利用低通滤波器进行预测，可以发现低通滤波器可以相对有效的过滤噪声。

- 核心代码

```
#include "Eigen/Dense"

  template<class T, int x> //类模板
  class LowPassFilter
  {
```

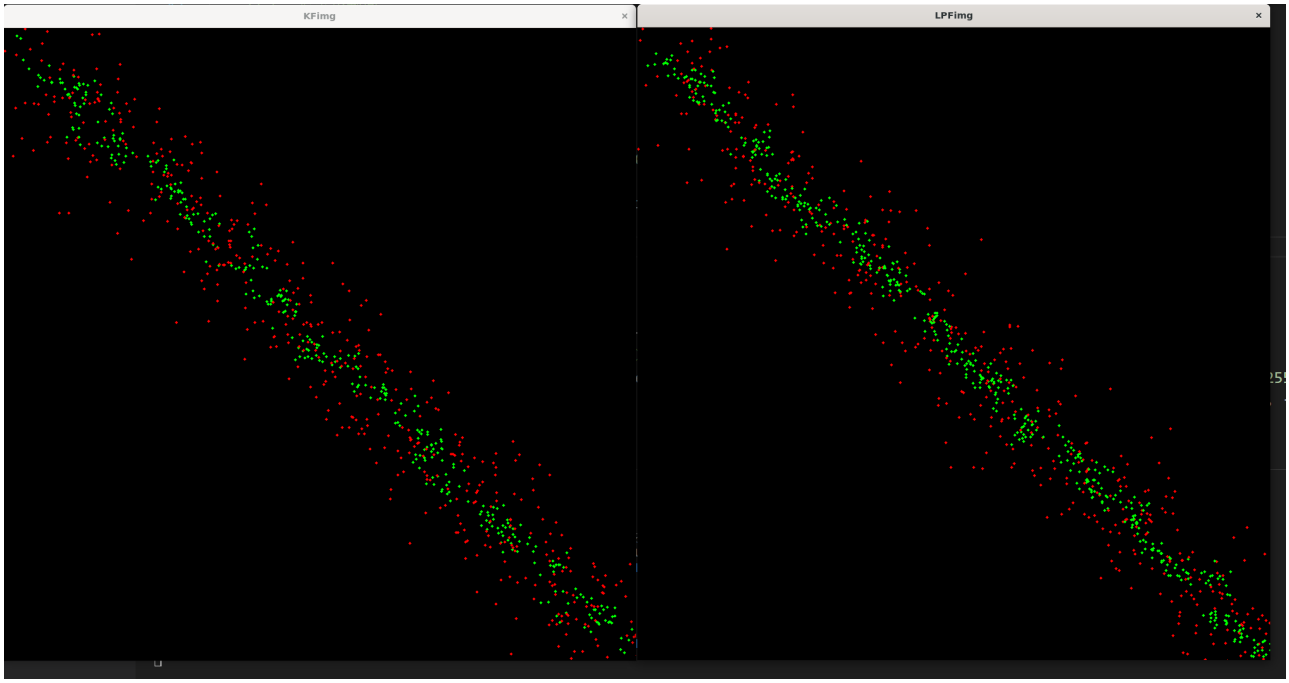
```
        Eigen::Matrix<T, x, 1> prev_output; //之前的输出
        double alpha; //采样率
    public:
        LowPassFilter(double sample_rate, double cutoff_frequency) //构造函数,
以采样率和截至频率作为参数进行构造
        {
            double dt = 1.0 / sample_rate;
            double RC = 1.0 / (cutoff_frequency * 2.0 * M_PI);
            alpha = dt / (dt + RC);
            prev_output = Eigen::Matrix<T, x, 1>::Zero();
        }//低通滤波器原理

        Eigen::Matrix<T, x, 1> update(const Eigen::Matrix<T, x, 1>& input)
        {
            Eigen::Matrix<T, x, 1> output = alpha * input + (1.0 - alpha) *
prev_output; //简单的公式套用得到滤波后输出
            prev_output = output; //更新prev_output
            return output; //返回输出矩阵
        }
        ~LowPassFilter() = default;
    };
```

子任务二（卡尔曼滤波）

实现对二维匀速运动物体的卡尔曼滤波。

- 实现效果



(左侧为卡尔曼滤波，右侧为低通滤波器)

对比分析，两种滤波在自设超参下预测效果接近，但似乎右侧低通滤波器下点更集中一些。这或许证明一些参数的设置会影响滤波器的性能。

- 核心代码解释

```
int main() {
    srand(114514);
    // 以一个静止滤波为例展示使用方法
    // 1. 初始化
    // 滤波器初始化
    KalmanFilter<double, 4, 4> *kf; // 包含位置和速度作为状态
    kf = new KalmanFilter<double, 4, 4>();

    // 仿真器初始化
        Simulator<double, 4> *simulator;
        simulator = new Simulator<double, 4>(Eigen::Matrix<double, 4, 1>{0, 0, 0,
0}, 5, Eigen::Matrix<double, 4, 1>{0.2, 0.2, 0.2, 0.2}); // 输入为起始点、方差和初始
速度

    // 2. 设置状态转移矩阵
    kf->transition_matrix << 1, 0, 1, 0,
                             0, 1, 0, 1,
                             0, 0, 1, 0,
                             0, 0, 0, 1;
    // 3. 设置测量矩阵
    kf->measurement_matrix << 1, 0, 0, 0,
                              0, 1, 0, 0,
                              0, 0, 1, 0,
                              0, 0, 0, 1;
    // 4. 设置过程噪声协方差矩阵
    kf->process_noise_cov << 0.01, 0, 0, 0,
                             0, 0.01, 0, 0,
                             0, 0, 0.01, 0,
                             0, 0, 0, 0.01;
//      // 5. 设置测量噪声协方差矩阵
//      kf->measurement_noise_cov << 5, 0,
//                                   0, 5;
    // 5. 设置测量噪声协方差矩阵
    kf->measurement_noise_cov <<5, 0, 0, 0,
                             0, 5, 0, 0,
                             0, 0, 5, 0,
                             0, 0, 0, 5;
    // 6. 设置控制向量
    kf->control_vector << 0, 0,
                          0, 0;

    // 生成随机点
    Eigen::Matrix<double, 4, 1> measurement;

    cv::Mat img(1000, 1000, CV_8UC3, cv::Scalar(0, 0, 0));
    for(int i = 0; i < 1000; i ++) {
        measurement = simulator->getMeasurement(i);
        // 7. 预测
        kf->predict(measurement);
        // 8. 更新
        kf->update();
        // // // 9. 获取后验估计
```

```
        Eigen::Matrix<double, 4, 1> estimate = kf->posteriori_state_estimate;
        // 10. 绘制出观测点和滤波点（平移到绘图中心），亦可采用其他可视化方法
 （matplotlib、VOFA+、Foxglove均可）
        cv::circle(img, cv::Point((int)(measurement[0] * 10 + 50),
int(measurement[1] * 10 + 50)), 2, cv::Scalar(0, 0, 255), -1);
        cv::circle(img, cv::Point((int)(estimate[0] * 10 + 50), (int)(estimate[1] *
10 + 50)), 2, cv::Scalar(0, 255, 0), -1);

        cv::imshow("KFimg", img);
        cv::waitKey(10);
    }


    /*construct*/
    // 定义低通滤波器的采样率和截止频率
    double sample_rate = 100.0;
    double cutoff_frequency = 5.0;

    // 创建低通滤波器
    LowPassFilter<double, 4> filter(sample_rate, cutoff_frequency);
    cv::Mat LPFimg(1000, 1000, CV_8UC3, cv::Scalar(0, 0, 0));

    for(int i = 0; i < 1000 ; i ++) {
        // 获取测量值
        Eigen::Matrix<double, 4, 1> measurement = simulator->getMeasurement(i);

        // 使用低通滤波器进行滤波
        Eigen::Matrix<double, 4, 1> estimate = filter.update(measurement);

        // 输出结果
        cv::circle(LPFimg, cv::Point((int)(measurement[0] * 10 + 50),
int(measurement[1] * 10 + 50)), 2, cv::Scalar(0, 0, 255), -1);
        cv::circle(LPFimg, cv::Point((int)(estimate[0] * 10 + 50), (int)
(estimate[1] * 10 + 50)), 2, cv::Scalar(0, 255, 0), -1);

        cv::imshow("LPFimg", LPFimg);
        cv::waitKey(10);
    }
    return 0;
}
```