狼牙暑假

task1

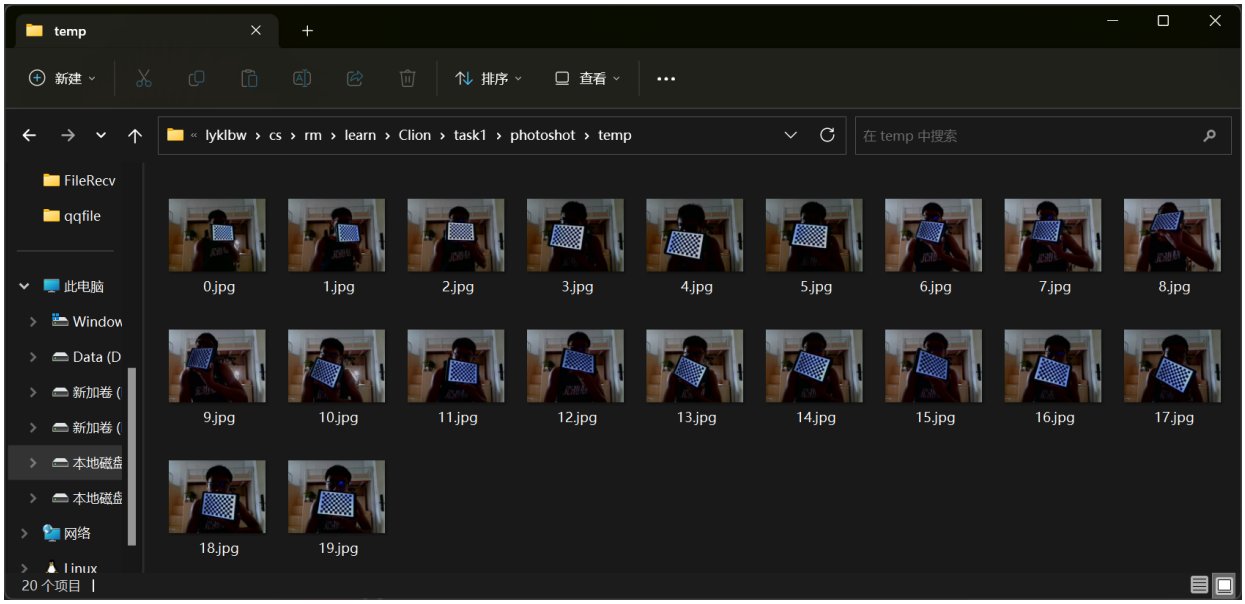
思路：先拍照把自己照片丢到`matlab`里面，得到参数和矩阵

之后先用二维码识别函数`QRcodeDetecter`得到四个点的信息，将其存储

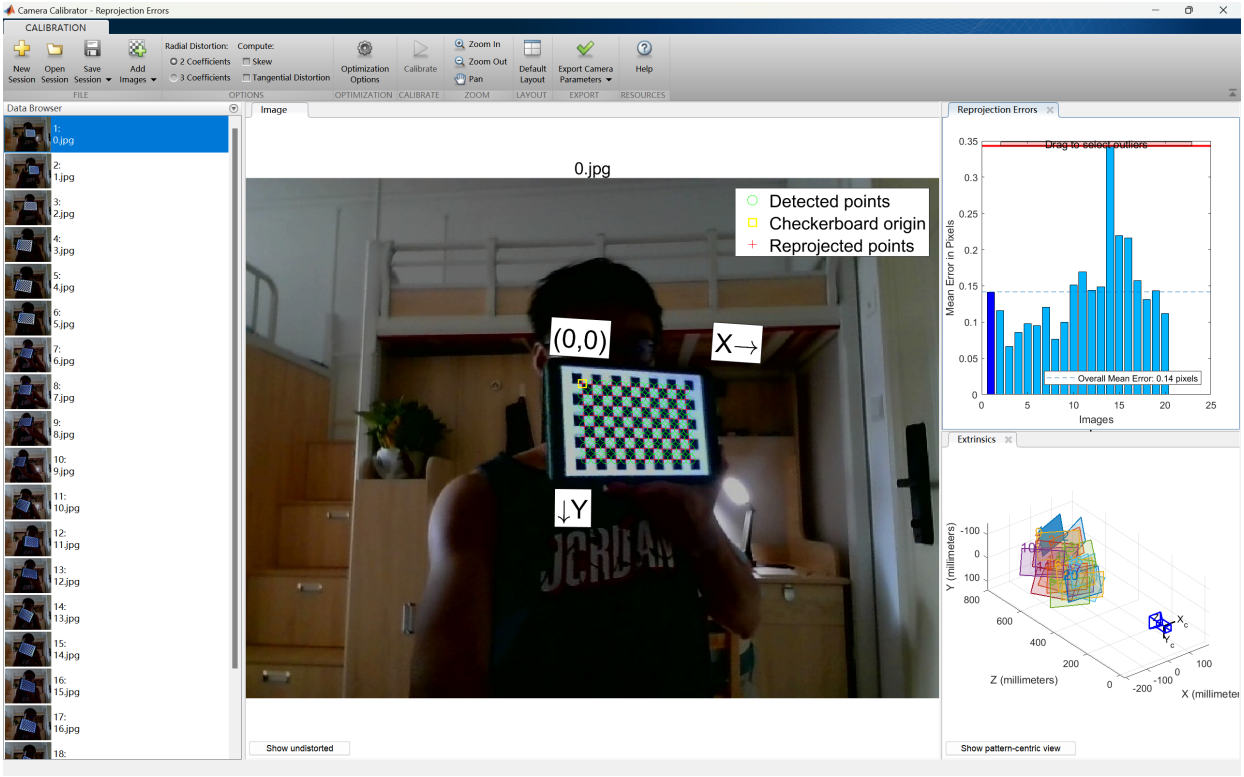
再直接向`pnp`输入参数即可,再将平移矩阵，旋转矩阵输出即可

拍照

```
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace std;
using namespace cv;
string writePath = "../temp/";
int main(int argc, char** argv){
    VideoCapture capture(0);
    string name;
    namedWindow("hello", CV_WINDOW_AUTOSIZE);
    int i=0;
    while (1) {
        Mat frame;
        capture >> frame;
        if (32 == waitKey(20)) {
            name = writePath + to_string(i)+".jpg";
            imwrite(name, frame);
            cout << name << endl;
            i++;
        }
    }
}
```



matlab求参



内参矩阵:

```
[511.246638444332,0,0,0,510.951006801711,0,316.910083508333,238.170857302231,1]
```

畸变参数

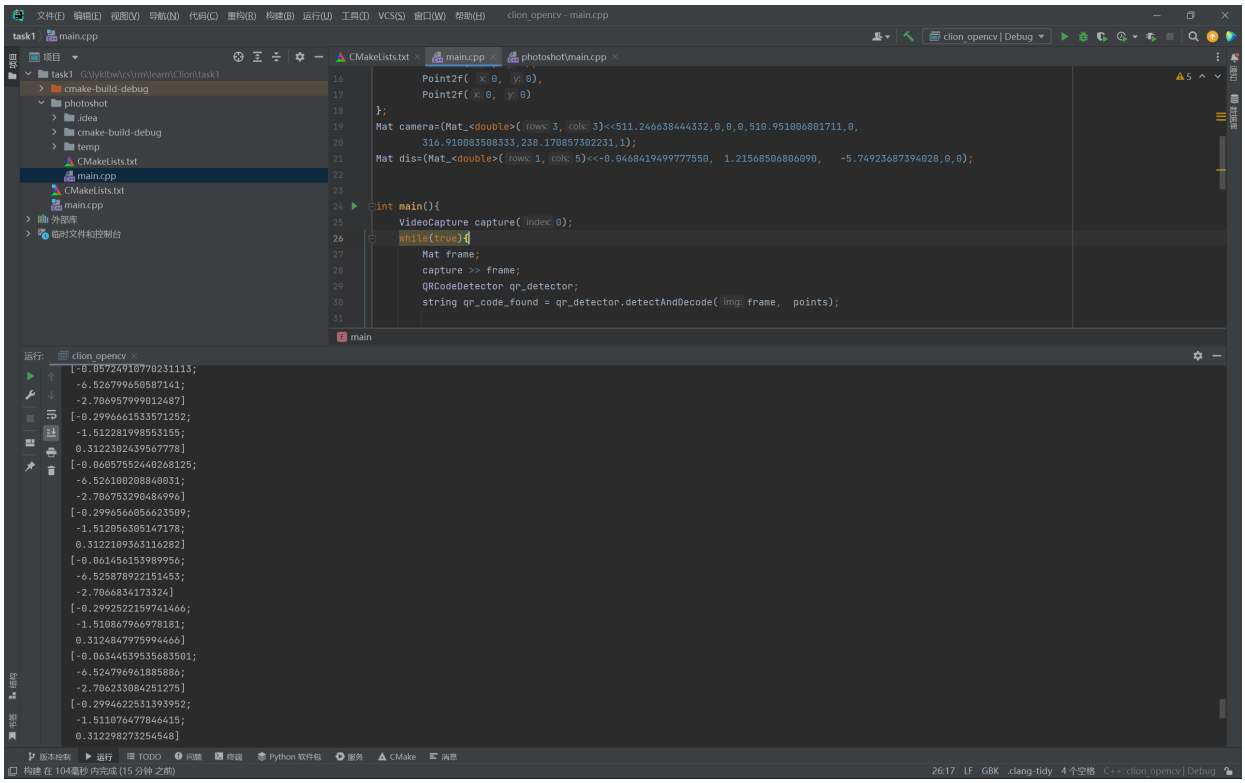
```
-0.0468419499777550,    1.21568506806090,    -5.74923687394028
```

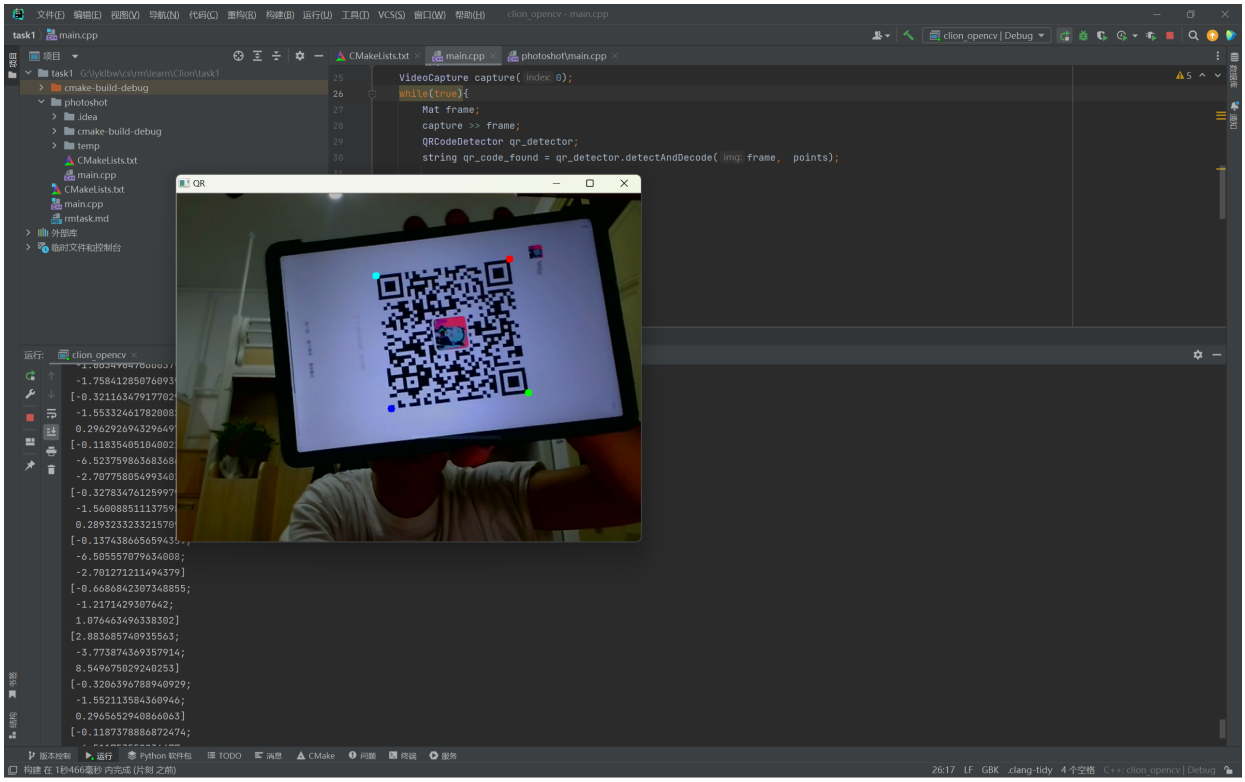
pnp

```
#include<iostream>
#include<opencv2/opencv.hpp>
#include<vector>
using namespace std;
using namespace cv;
int length=10;
vector<Point3f>obj=vector<Point3f>{
        Point3f(0, 0, 0),
        Point3f(0, length, 0),
        Point3f(length, 0, 0),
        Point3f(length, length, 0)
};
```

```
vector<Point2f>points=vector<Point2f>{
        Point2f(0, 0),
        Point2f(0, 0),
        Point2f( 0, 0),
        Point2f(0, 0)
};
Mat camera=(Mat_<double>(3,3)<<511.246638444332,0,0,0,510.951006801711,0,
        316.910083508333,238.170857302231,1);
Mat dis=(Mat_<double>(1,5)<<-0.0468419499777550,    1.21568506806090,
-5.74923687394028,0,0);
int main(){
    VideoCapture capture(0);
    while(true){
        Mat frame;
        capture >> frame;
        QRCodeDetector qr_detector;
        string qr_code_found = qr_detector.detectAndDecode(frame,  points);
        // 检查是否成功检测到二维码
            // 提取四个角点
            if (points.size()==4) {
                cv::Point pt1 = points[0];
                cv::Point pt2 = points[1];
                cv::Point pt3 = points[2];
                cv::Point pt4 = points[3];
                // 在图像上绘制四个角点
                cv::circle(frame, pt1, 5, cv::Scalar(0, 0, 255), -1);
                cv::circle(frame, pt2, 5, cv::Scalar(0, 255, 0), -1);
                cv::circle(frame, pt3, 5, cv::Scalar(255, 0, 0), -1);
                cv::circle(frame, pt4, 5, cv::Scalar(255, 255, 0), -1);

                // 保存结果图像
                cv::imwrite("result_image.jpg", frame);
                Mat rVec=  Mat::zeros(3,1,CV_64FC1);
                Mat tVec=  Mat::zeros(3,1,CV_64FC1);
                solvePnP(obj,points,camera,dis,rVec,tVec,false);
                cout<<rVec<<endl;
                cout<<tVec<<endl;
            }
        imshow("QR",frame);
            waitKey(1);
    }
}
```





具体数据(部分)：

```
[-0.06752523125136817;
 -6.525424088579817;
 -2.706320477236354]
[-0.2988702008847547;
 -1.509139465618202;
 0.312712350924616]
[-0.06747683407974332;
```

```
 -6.525537769920231;
 -2.706350706150762]
[-0.3030731731989073;
 -1.518500464298627;
 0.3095717289791208]
[-0.05826776517577667;
 -6.531752640877979;
 -2.708746516968604]
[-0.3003671055949202;
 -1.512551747355342;
 0.3116024767180746]
[-0.06353918705238425;
 -6.527750817668084;
 -2.707100668033683]
```

task3

大一上册看的类，这里记不太清了，临时学一下

特点：封装，继承，多态

主要形式：

```
class name{
    public:
    //外部接口
    类的公有成员在程序的任何地方都可以直接访问，没有任何限制
    protected:
    类的私有成员只能被本类的成员函数访问(从本类的内部访问)，来自类外部的任何访问都是非法的
    private:
    类的保护成员的性质和私有成员的性质相似，其差别在于继承过程中对产生的新类影响不同(将在后面学习继承的时候详细介绍）
};
//类成员的访问控制是实现封装的重要机制
```

类是一种抽象机制，它描述了一类事物的共同属性和行为。在C++中，类的对象就是该类的某一特定**实体**(也称实例)。

函数成员必须在体内声明之后才能在体外定义

```
inline void cTime::show()...
```

笔记有点懒得记，差不多想起来了

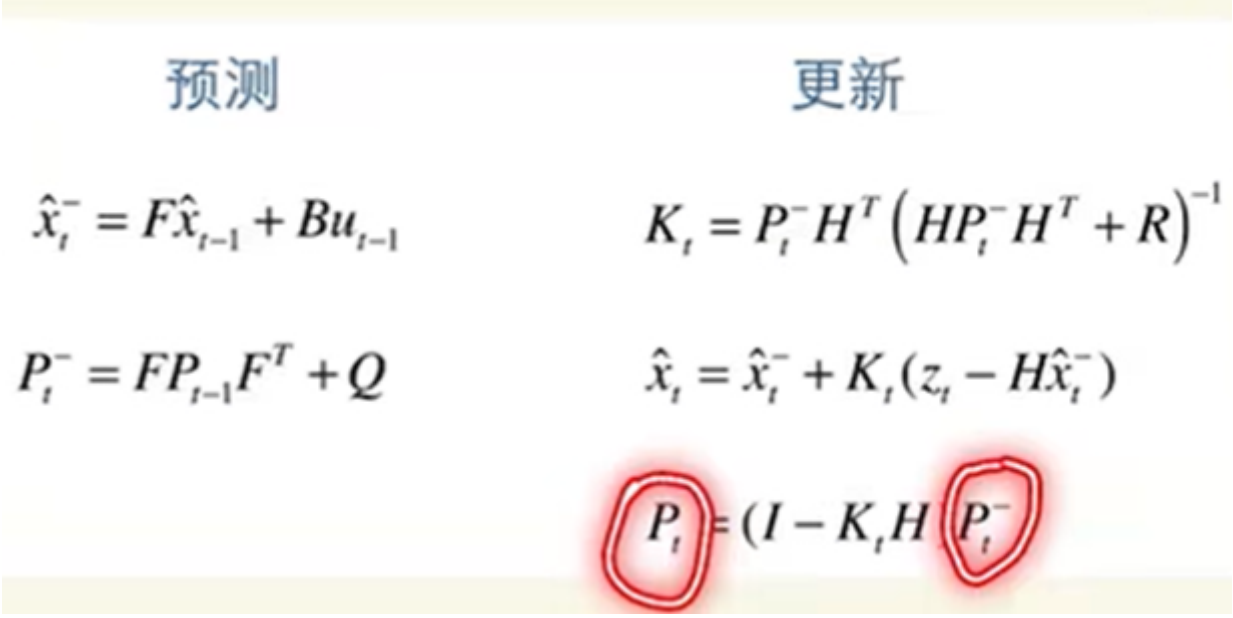
思路：

先构造类及其方法和成员模拟运动状态——C++

复习卡尔曼滤波知识——b站华南虎

构造函数实现卡尔曼滤波，传递数据——即可实现

学习



代码

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <cmath>

class KalmanPredictor {
public:
    KalmanPredictor(double initialPosition, double initialVelocity, double
noiseLevel)
            : x(initialPosition), v(initialVelocity), Q(noiseLevel), R(0.1) {}

    void predict(double dt, double acceleration) {
        x = x + v * dt + 0.5 * acceleration * dt * dt;
        v = v + acceleration * dt;
    }

    void update(double measurement) {
        double K = v / (v + R);
        x = x + K * (measurement - x);
        v = v + K * (measurement - x);
    }

    double getPredictedState() {
        return x;
```

```
    }

private:
    double x; // Predicted position
    double v; // Predicted velocity
    double Q; // Process noise
    double R; // Measurement noise
};
class AcceleratedMotionSimulator {
private:
    double initialPosition;
    double initialVelocity;
    double acceleration;
    double timeStep;
    double noiseLevel;

public:
    AcceleratedMotionSimulator(double initialPos, double initialVel, double accel,
double step, double noise)
            : initialPosition(initialPos),
              initialVelocity(initialVel),
              acceleration(accel),
              timeStep(step),
              noiseLevel(noise) {
        // Seed the random number generator
        std::srand(static_cast<unsigned int>(std::time(nullptr)));
    }

    double generateNoise() {
        // Generate random noise within the specified noise level
        return (2.0 * noiseLevel * (std::rand() / static_cast<double>(RAND_MAX)) -
noiseLevel);
    }

    double simulatePosition(double time) {
        // Simulate position at a given time with noise
        double truePosition = initialPosition + initialVelocity * time + 0.5 *
acceleration * time * time;
        double noisyPosition = truePosition + generateNoise();
        return noisyPosition;
    }

    double runSimulation(double endTime,KalmanPredictor kalmanPredictor) {
        for (double t = 0; t <= endTime; t += timeStep) {
            double simulatedPosition = simulatePosition(t);
            kalmanPredictor.predict(0.1, acceleration);
            kalmanPredictor.update(simulatedPosition);
            std::cout << "True Measurement: " << simulatedPosition << " | Predicted
Position: " << kalmanPredictor.getPredictedState() << std::endl;
        }
    }
};
```

```
int main() {
    double initialPosition = 0.0;
    double initialVelocity = 10.0;
    double acceleration = 2.0;
    double timeStep = 0.1;
    double noiseLevel = 1.0;

    AcceleratedMotionSimulator simulator(initialPosition, initialVelocity,
acceleration, timeStep, noiseLevel);
    KalmanPredictor kalmanPredictor(initialPosition, initialVelocity,noiseLevel);

    double simulationTime = 5.0; // Total simulation time
    simulator.runSimulation(simulationTime,kalmanPredictor);

    return 0;
}
```

结果

```
True Measurement: 0.321777 | Predicted Position: 0.332729
True Measurement: 1.47568 | Predicted Position: 1.4748
True Measurement: 2.57573 | Predicted Position: 2.57543
True Measurement: 3.20608 | Predicted Position: 3.21021
True Measurement: 4.55671 | Predicted Position: 4.5546
True Measurement: 6.84015 | Predicted Position: 6.83007
True Measurement: 7.24295 | Predicted Position: 7.24925
True Measurement: 9.21775 | Predicted Position: 9.21102
True Measurement: 9.9163 | Predicted Position: 9.9203
True Measurement: 11.7143 | Predicted Position: 11.7095
True Measurement: 12.913 | Predicted Position: 12.9132
True Measurement: 13.5582 | Predicted Position: 13.563
True Measurement: 14.9604 | Predicted Position: 14.9594
True Measurement: 15.9547 | Predicted Position: 15.9569
True Measurement: 16.4075 | Predicted Position: 16.4139
True Measurement: 17.5875 | Predicted Position: 17.5887
True Measurement: 19.133 | Predicted Position: 19.1316
True Measurement: 21.6554 | Predicted Position: 21.6471
True Measurement: 23.1642 | Predicted Position: 23.1633
True Measurement: 24.8336 | Predicted Position: 24.8317
True Measurement: 25.3289 | Predicted Position: 25.3353
True Measurement: 26.7007 | Predicted Position: 26.7013
True Measurement: 28.8279 | Predicted Position: 28.8235
True Measurement: 29.4683 | Predicted Position: 29.4739
True Measurement: 30.7117 | Predicted Position: 30.7135
True Measurement: 33.5568 | Predicted Position: 33.5483
True Measurement: 35.1265 | Predicted Position: 35.1264
True Measurement: 35.93 | Predicted Position: 35.9348
True Measurement: 38.1014 | Predicted Position: 38.0978
True Measurement: 38.3631 | Predicted Position: 38.3714
True Measurement: 40.8452 | Predicted Position: 40.8401
```

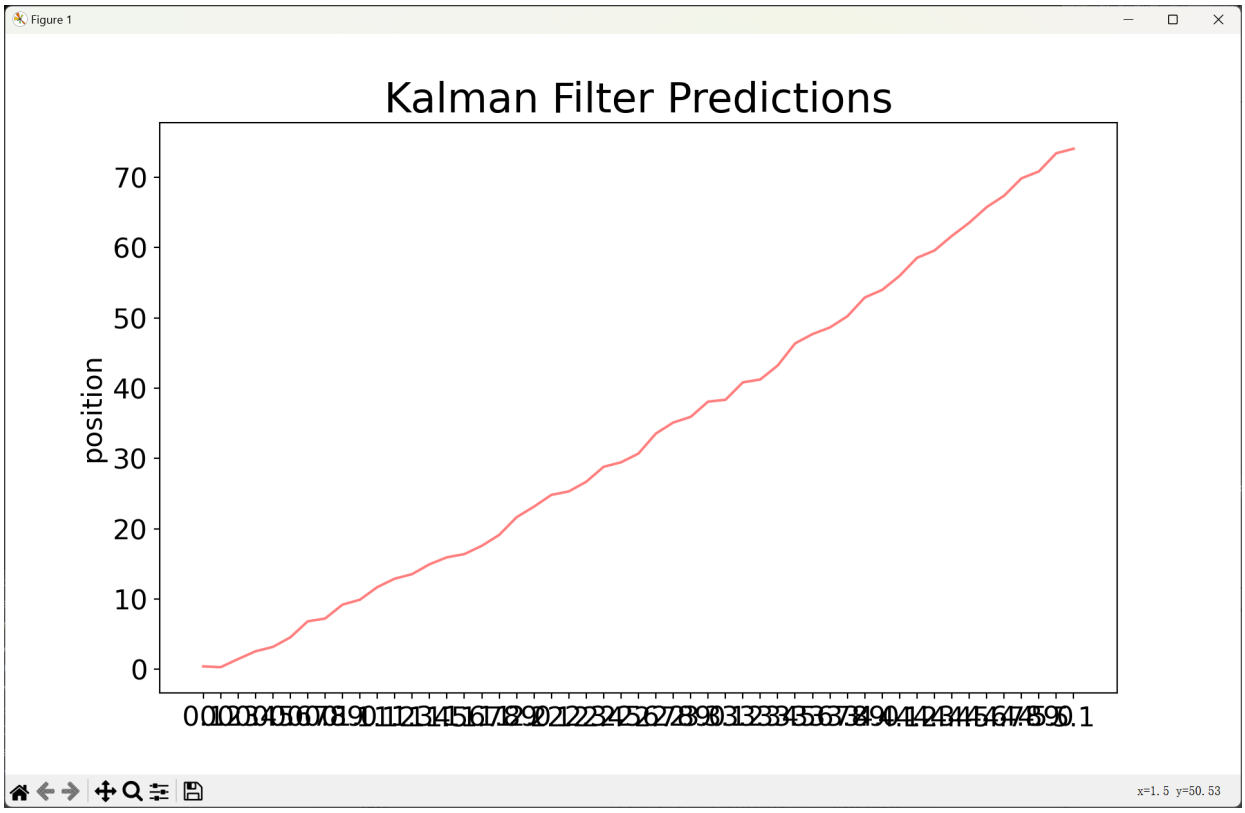
```
True Measurement: 41.2507 | Predicted Position: 41.2581
True Measurement: 43.2379 | Predicted Position: 43.2361
True Measurement: 46.3885 | Predicted Position: 46.3799
True Measurement: 47.7104 | Predicted Position: 47.7126
True Measurement: 48.6549 | Predicted Position: 48.6594
True Measurement: 50.2385 | Predicted Position: 50.2394
True Measurement: 52.9114 | Predicted Position: 52.9064
True Measurement: 53.9962 | Predicted Position: 54
True Measurement: 55.9889 | Predicted Position: 55.9879
True Measurement: 58.5571 | Predicted Position: 58.5531
True Measurement: 59.5902 | Predicted Position: 59.5945
True Measurement: 61.6665 | Predicted Position: 61.6654
True Measurement: 63.5455 | Predicted Position: 63.5456
True Measurement: 65.7616 | Predicted Position: 65.76
True Measurement: 67.3826 | Predicted Position: 67.3842
True Measurement: 69.8645 | Predicted Position: 69.8618
True Measurement: 70.8456 | Predicted Position: 70.8505
True Measurement: 73.4389 | Predicted Position: 73.4359
True Measurement: 74.0797 | Predicted Position: 74.0864
```

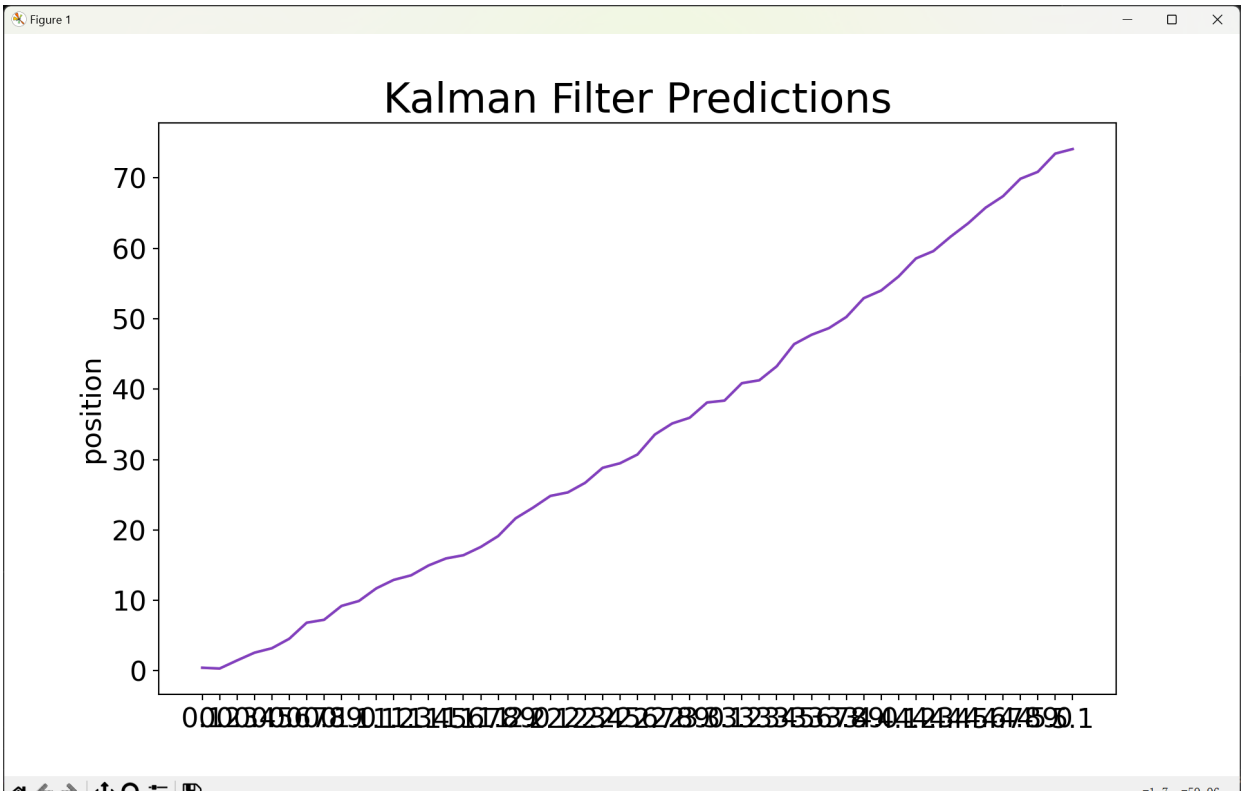
将两组数据导入wps（doge

	A	B	C	D
10	9.9163	9.9203		
11	11.7143	11.7095		
12	12.913	12.9132		
13	13.5582	13.563		
14	14.9604	14.9594		
15	15.9547	15.9569		
16	16.4075	16.4139		
17	17.5875	17.5887		
18	19.133	19.1316		
19	21.6554	21.6471		
20	23.1642	23.1633		
21	24.8336	24.8317		
22	25.3289	25.3353		
23	26.7007	26.7013		
24	28.8279	28.8235		
25	29.4683	29.4739		
26	30.7117	30.7135		
27	33.5568	33.5483		
28	35.1265	35.1264		
29	35.93	35.9348		
30	38.1014	38.0978		
31	38.3631	38.3714		
32	40.8452	40.8401		
33	41.2507	41.2581		
34	43.2379	43.2361		
35	46.3885	46.3799		
36	47.7104	47.7126		
37	48.6549	48.6594		
38	50.2385	50.2394		
39	52.9114	52.9064		
40	53.9962	54		
41	55.9889	55.9879		
42	58.5571	58.5531		
43	59.5902	59.5945		

44	61.6665	61.6654		
45	63.5455	63.5456		
46	65.7616	65.76		
47	67.3826	67.3842		
48	69.8645	69.8618		
49	70.8456	70.8505		
50	73.4389	73.4359		
51	74.0797	74.0864		
52				

使用matplotlib画图



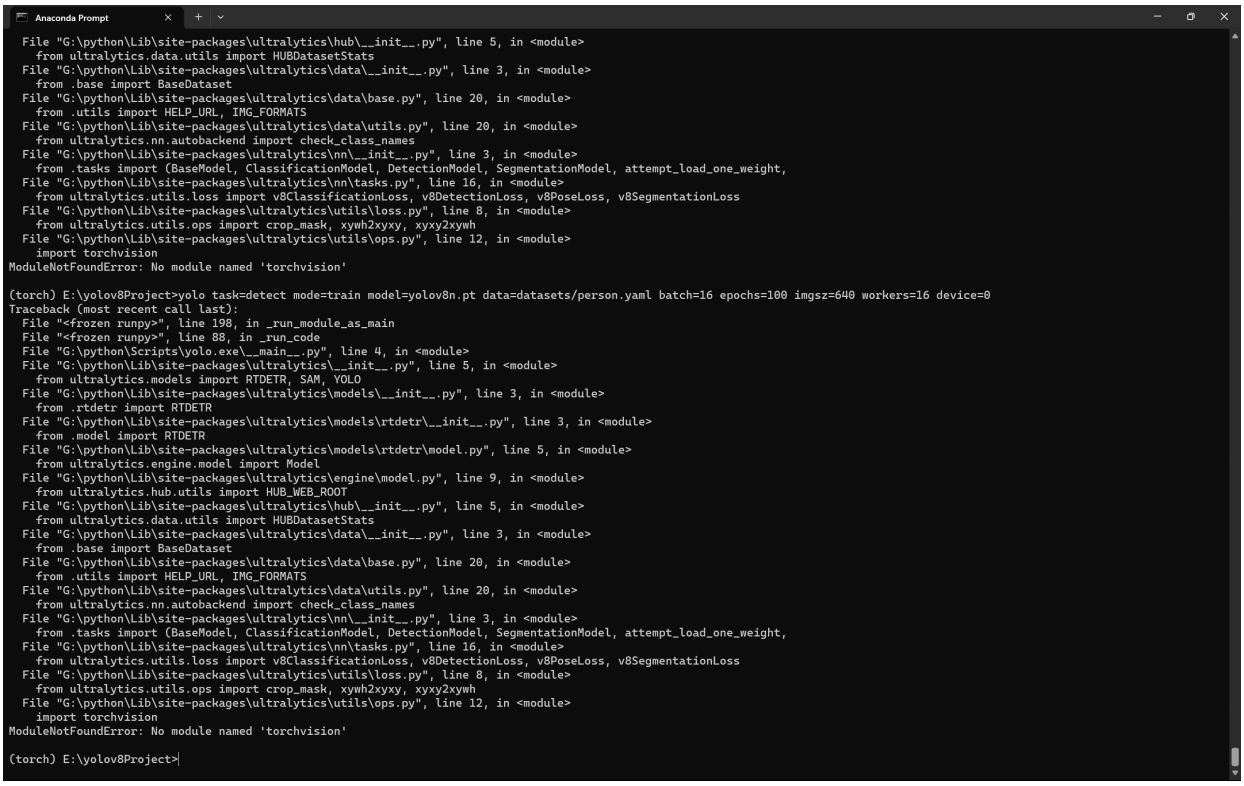


task2

跟着教程

配置完了cuda和cudnn,安装了pytorch

前前后后绕了不少弯路，但还是到了最后几步（只差训练的一行代码跑不通了呜呜）——可是奈何对版本的理解还不够深刻或是什么，在配置环境的问题上始终无法突破



个人猜测是下载的torchvison的版本不够高，但是更高版本不管爬不爬梯子都下不下来

task2浪费的时间过长故而放弃