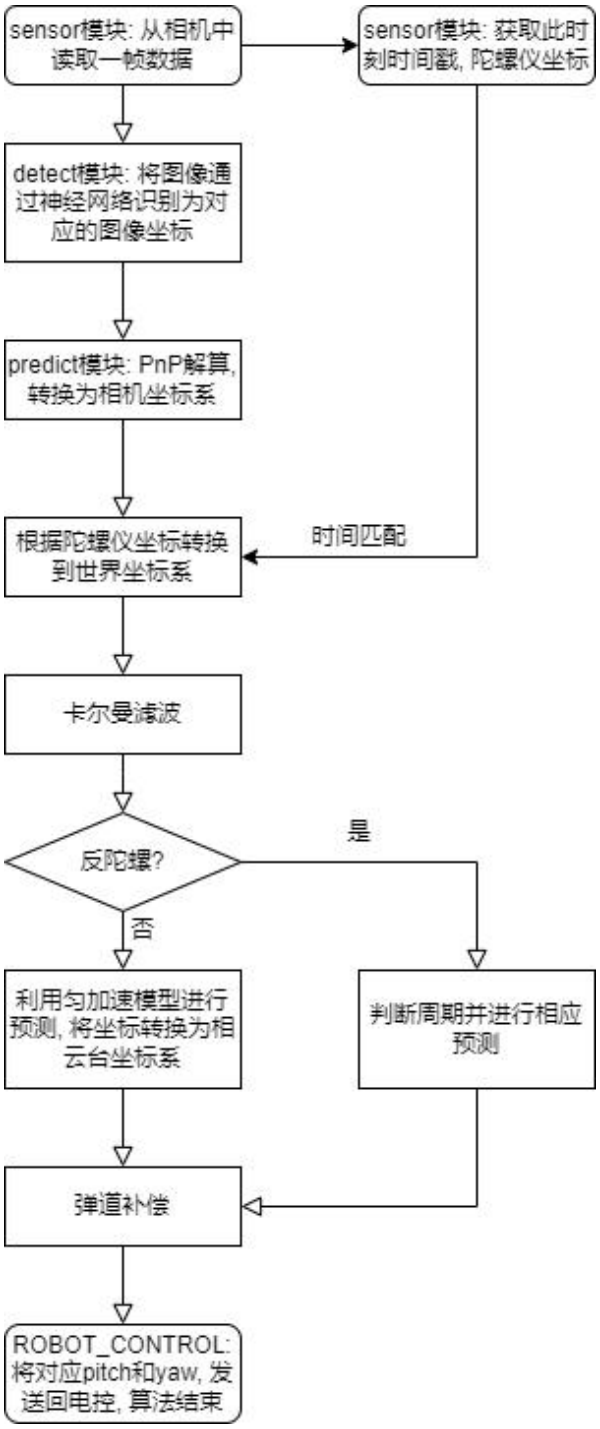
算法设计

1.1.1.1 Pipeline 与功能简介

整个算法流程主要分为三个部分，sensor, detect 和 predict

- Sensor 主要处理传感器的数据
- Detect 负责识别装甲板
- Predict 进行解算， 预测以及弹道计算等后处理步骤



1.1.1.2 特色功能

一，多线程并发库

简要介绍: 采用ROS中消息模型的思想，实现了Publisher/Subscriber信息模型，用于各个模块之间的管道通信，并且经过包装，使得调用十分简便，且线程锁的加入也使整个并发过程具有良好的线程安全性，可以从容处理各个模块之间来回传输的数据

原理解释:

摘要: 采用了 STL 中的队列 std::queue<T>实现了核心的信息存储，以字符串作为每个收发管道的唯一标识符(以哈希表保存)，从 Publisher 中通过 push(const T&)方法传入数据，然后在需要数据的地方声明具有相同标识符的 Subscriber 并调用 pop()方法获取数据

详细描述:

创建管道

当使用构造函数 Publisher<T>(const string&)或 Subscriber<T>(const string&)时，查找静态的无序表(unordered_map<string, weak_ptr>)，若没有结果，则创建强指针然后返回(指针指向的类型是一个辅助类 ObjManager)

绑定到管道

与上一条情况相同，但是如果此时在表中查找到已经存在的管道，则直接将弱指针转换为强指针，然后获得该管道

发布信息

Publisher::push 方法

如果管道不为空，使用 unique_lock 锁住，然后将信息写入管道，完成之后唤醒其他的锁(在 Subscriber 里面的 condition_variable)

获取信息

Subscriber::pop 方法

使用 condition_variable 等待队列不为空，收到唤醒之后使用 std::queue::pop()获取队列的首元素

析构

使用了智能指针中强指针和弱指针并用的方式，在记录进哈希表中时保存弱指针，而建立起与管道连接时使用的是强指针，可以既保留哈希表中对应的地址，又可以保证析构时所有的队列都能够成功释放

清空时会自动销毁队列中的元素

使用示例:

```
// in the first part...
Publisher<std::string> pub("sensor_data");
pub.push("hello");

//in the other part...
Subscriber<std::string> sub("sensor_data");
assert("hello", sub.pop());
```

优点:

使用简单，无需初始化即可使用，语法也易于理解，并且具有可靠的并发能力，可以在多个线程中互相通讯，而且也可以实现多个发布多个订阅

缺点:

Publisher 必须在另一边 Subscriber 开始进行 pop()之前声明，对于采用主函数中开启多线程的操作会导致捕捉管道失败，所以需要提前在开启其他线程之前开启发布器的线程

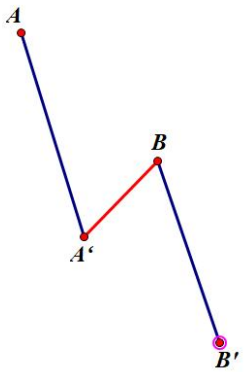
此外，由于从内部取数据是 move 操作，所以数据只能在进程间拿取一次，对于需要多方接收统一数据的场景(比如赛场内录)就会需要再次发送，较为麻烦。

二，预测装甲板

简要介绍: 这一部分完成了从带有时间戳的装甲板坐标到发射方向的计算，包括了坐标系转换，编号修正，卡尔曼滤波，自瞄优先级，发射角度与时机的计算，这一部分采用了面向过程的设计，封装为一个函数，只要开启一个线程执行此函数，就可以进入预测状态，这一部分的预测并非深度学习的"预测(inference)"

分模块详细介绍:

- 坐标系转换
- 通过已知的装甲板大小和神经网络给出的坐标，定义世界坐标系原点为装甲板中心点，，通过 PnP 解算，解算结果中的平移矩阵就是装甲板中心点在相机系下的坐标。
- 通过手眼标定 (见手眼标定部分)得到旋转矩阵和平移矩阵，可以完成到 base 系和云台系的转换
-
- 编号修正
- 编号可能存在误识别问题，为了处理这种情况，采用两部分进行补偿，一部分是基于贪心的帧间匹配，另一部分是基于摩尔投票算法的编号修正。
- 该算法本质上是利用识别信息在空间上较强的连续性，来过滤编号识别中的闪烁情况。
-
- 1. 帧间匹配
- 在帧间匹配之前，首先需要剔除掉过去装甲板信息中时间戳与当前时间戳差距过大的信息。
- 然后对任意一个当前装甲板，都需要枚举过去处理好的装甲板，判断二者的中心点距离是否再预设范围内以及突变幅度是否在某范围内，如果符合条件，则在二者之间连一条边，边长为二者中心点距离。
- 将所有的边按照边长从小到大排序，顺次枚举每一条边，如果边上两点均未匹配，则将他们匹配。
- 这样的贪心匹配其实是不优秀的，比如如下情况（A 本应对应 A'，B 本应对应 B'）



-
- 上述贪心算法会将 B 与 A'相匹配
- 但是由于帧数较高（达到了每秒 100 帧），实际情况中 AA'与 BB'的长度都

很短

- 即便是出现这种误匹配情况也可以通过编号修正来改正
-
- 2. 编号修正
- 通过帧间匹配我们可以得到空间中的多条路径。
- 利用摩尔投票算法统计每条路径中出现次数超过一半的编号的数目，并将此编号作为编号过滤的结果。。
- 关于误匹配，我们可以设置摩尔投票法中的权重最大值 Inertia。
- 这样的话，如果连续出现 Inertia 帧误识别（在此之前未出现误识别），最终的编号过滤结果才会改变。如果出现误匹配，在经过 Inertia 帧之后也会修正编号。
- 由于误识别的存在，可能会出现某一编号的装甲板多次出现的情况，我们只需找到同一编号中距离最近的两块装甲板（最可能是同一机器人上的两块装甲板）作为处理结果传输给 Part3 即可。

- 弹道补偿
- 经过推导得出发射时 pitch 应该为
- $\text{pitch} = arctan\left(\frac{v_y}{v_x}\right) = \arctan\left(\frac{y+\frac{1}{2}gt^2}{x}\right)$
- yaw 可以直接通过坐标来计算, roll 不做考虑
- 当远距离吊射时，需要考虑空气阻力，此时分段调参，当距离大于阈值时，将 x 与 y 乘上系数后再进行计算，考虑到大弹与小弹的空气阻力系数不同，因此选用了不同的参数进行调参
-

优化方案:

在编号过滤中，不同的编号信息

三，手眼标定

算法介绍：

算法意在建立完善的机器人坐标系统，打通电控与视觉的联系，并标定相机光心的实际坐标，使得预测器 EKF 有坚实的算法和数据基础。具体表现为，能稳定解算目标在世界坐标系下的位置，机器人移动云台，相机从不同角度拍摄到目标，解算得到的目标在世界坐标系下的位置变换在可接受的范围内。

算法原理：

1、 坐标系定义：

Base 系（世界坐标系）：原点在云台的旋转中心（yaw 转轴与 pitch 转轴的近似交点）

云台系（gripper 系）：原点与 base 系重合，到 base 系的旋转矩阵由陀螺仪 imu 解算获得。

相机系（cam 系）：“固定”在云台系上，与云台系的旋转和平移都是定值，是待标定的内容。

2、 手眼标定：

手眼标定实际上是求解矩阵方程 AX=XB，其中 A 是相机（cam 系）前后两次空间变换的其次矩阵，B 是机械臂末端（gripper）系前后两次变换的其次矩阵，通过多次求解该方程即可解出 X。

在实现上采用 opencv4 的 calibrateHandEye()函数进行求解。

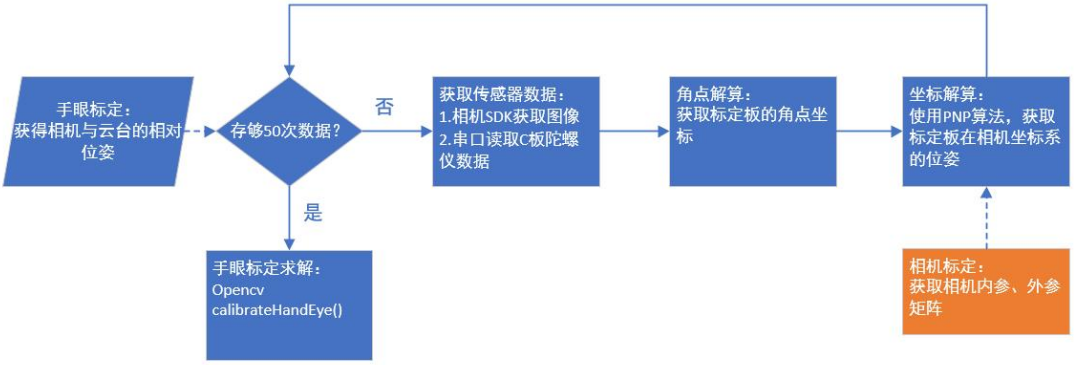
§ calibrateHandEye()

```
void cv::calibrateHandEye ( InputArrayOfArrays       R_gripper2base,
                            InputArrayOfArrays       t_gripper2base,
                            InputArrayOfArrays       R_target2cam,
                            InputArrayOfArrays       t_target2cam,
                            OutputArray              R_cam2gripper,
                            OutputArray              t_cam2gripper,
                            HandEyeCalibrationMethod method = CALIB_HAND_EYE_TSAI
                          )
```

其中:

R_gripper2base，t_gripper2base 是云台系（gripper）相对于机器人基坐标系（base）的旋转矩阵与平移向量，平移向量为 0，旋转矩阵由陀螺仪数据计算得到。R_target2cam ， t_target2cam 是标定板相对于相机的旋转矩阵和平移向量，由 SolvePNP()运行 PNP 算法结合标定相机标定数据得到。

流程图：



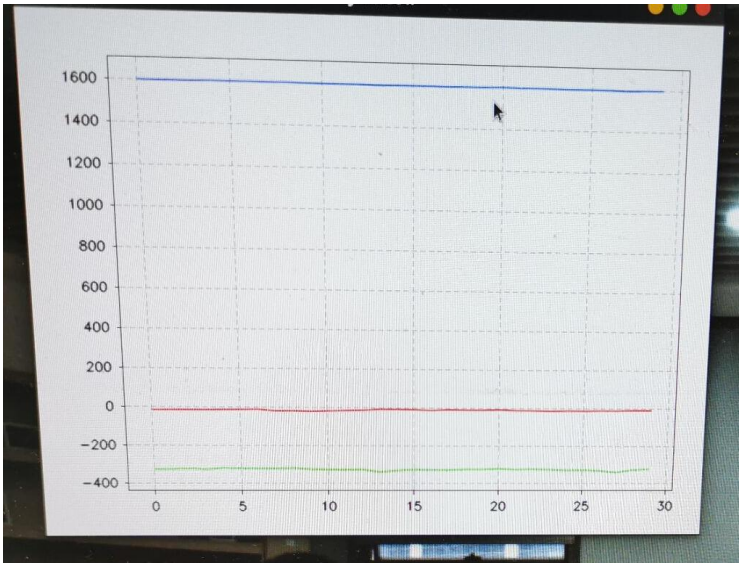
3、 坐标转换：

通过 PNP 算法获得装甲板在相机坐标系（Cam）下的位姿（平移和旋转），

通过旋转和平移变换解算到云台系（利用 R_cam2gripper 和 t_cam2griper），再通过旋转变换解算到 base 系（通过 R_gripper2bass 和 t_gripper2base）。这样，将在 base 系完成对目标的 EKF 跟踪，更具有鲁棒性和可解释性，同时使得目标的位置不收到云台自身运动状态的影响。

算法验证

为了验证算法的效果，我们将机器人底盘固定，并且在其正前方约 1.6m 处固定放置了一块标定板，在持续晃动云台（改变 pitch 和 yaw）的过程中，观察解算的标定板坐标值。其结果如下：



蓝色代表目标解算位置的距离（z 方向位移），红色和绿色分别代表目标解算位置的 x 方向和 y 方向位移。（单位 mm）

可以得出结论，我们的坐标系统和标定方法具有可行性和可以接受的准确性。

四，大能量机关预测

1. 功能简介:

预测器功能主要分为三部分：坐标解算、参数估计、滤波器预测

1.1 坐标解算

1.2 坐标解算部分在开始的时候通过对神经网络得到的特征点做 pnp 解算得到目标的相机坐标，结合从电控读取的 imu 数据转换到云台系，再通过预先标定好的相机系-世界系转换矩阵得到待击打目标在世界系下的坐标。计算旋转中心->装甲板中心的向量，并降维、归一化转化到单位圆

上计算旋转角度，与上一次记录的数据比较判断是否切换目标，如果有记录偏置。

1.3 经过滤波器之后将预测出的角度量加上偏置，结合弹道模型计算待击打点。

1.4

1.5 参数估计

1.6 考虑到滤波器收敛时间问题，先对运动参数进行估计，使用估计值初始化滤波器。通过对前 1.5s 采样得到的数据做非线性方程最小二乘拟合，计算旋转运动方程 spd=a*sin(wt)+(2.090-a) 中的参数 a，w 和当前时间 t（实际上相当于三角函数的相位\phi =wt）。

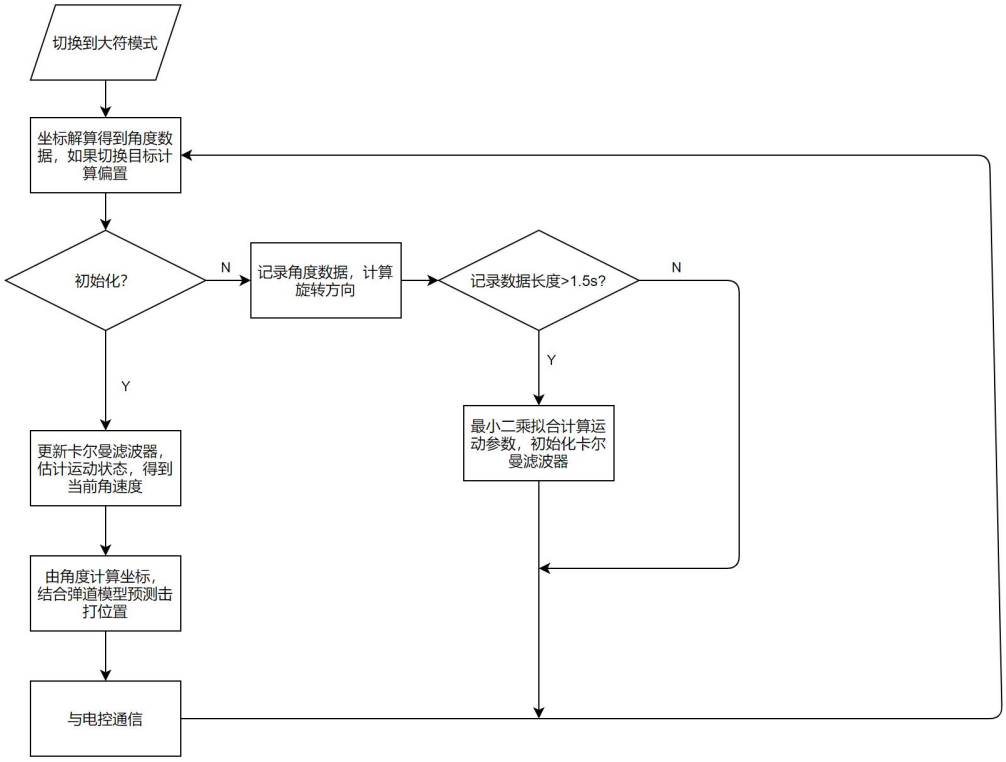
1.7

1.8 滤波器预测

1.9 对于非线性的运动方程，使用扩展卡尔曼滤波（EKF）算法跟踪和预测击打点。

1.10

1.11 预测器流程图



1.12

1.13

2. **重要算法原理阐述**
3. **2.1 非线性方程最小二乘法拟合—LM 算法**
4. 对于非线性的运动方程，使用扩展卡尔曼滤波（EKF）算法跟踪和预测击打点。
5. LM 算法本质上就是在 Gauss-Newton 法的基础上添加了一个惩罚项，用信赖域法求解惩罚因子。该算法的主要思想是通过惩罚因子来调节步长。

6. 迭代公式如下：
7. $\boldsymbol{x_{k+1} = x_k + (H + \alpha I)^{-1} G}$
8. 其中$\boldsymbol{H}$为多维向量的 Hessian 矩阵，近似计算公式为$\boldsymbol{H = J^T J}$ ，$\boldsymbol{J}$ 为雅可比矩阵，可以通过解析法或数值法求取；$\boldsymbol{G}$为多维向量的一阶梯度。
9. 当下降太快时使用较小的 $\boldsymbol{\alpha}$ ，使整个公式接近高斯牛顿法；当下降太慢时使用较大的 $\boldsymbol{\alpha}$，使整个公式接近梯度法。
10. 算法伪代码如下：

Input: A vector function $f : \mathcal{R}^m \rightarrow \mathcal{R}^n$ with $n \geq m$, a measurement vector $\mathbf{x} \in \mathcal{R}^n$ and an initial parameters estimate $\mathbf{p}_0 \in \mathcal{R}^m$.
Output: A vector $\mathbf{p}^+ \in \mathcal{R}^m$ minimizing $||\mathbf{x} - f(\mathbf{p})||^2$.
Algorithm:
$k := 0$; $\nu := 2$; $\mathbf{p} := \mathbf{p}_0$;
$\mathbf{A} := \mathbf{J}^T\mathbf{J}$; $\epsilon_\mathbf{p} := \mathbf{x} - f(\mathbf{p})$; $\mathbf{g} := \mathbf{J}^T\epsilon_\mathbf{p}$;
stop:=($||\mathbf{g}||_\infty \leq \varepsilon_1$); $\mu := \tau * \max_{i=1,\ldots,m}(A_{ii})$;
while (not stop) and ($k < k_{max}$)
  $k := k + 1$;
  repeat
    Solve $(\mathbf{A} + \mu\mathbf{I})\delta_\mathbf{p} = \mathbf{g}$;
    if ($||\delta_\mathbf{p}|| \leq \varepsilon_2||\mathbf{p}||$)
      stop:=true;
    else
      $\mathbf{p}_{new} := \mathbf{p} + \delta_\mathbf{p}$;
      $\rho := (||\epsilon_\mathbf{p}||^2 - ||\mathbf{x} - f(\mathbf{p}_{new})||^2)/(\delta_\mathbf{p}^T(\mu\delta_\mathbf{p} + \mathbf{g}))$;
      if $\rho > 0$
        $\mathbf{p} = \mathbf{p}_{new}$;
        $\mathbf{A} := \mathbf{J}^T\mathbf{J}$; $\epsilon_\mathbf{p} := \mathbf{x} - f(\mathbf{p})$; $\mathbf{g} := \mathbf{J}^T\epsilon_\mathbf{p}$;
        stop:=($||\mathbf{g}||_\infty \leq \varepsilon_1$) or ($||\epsilon_\mathbf{p}||^2 \leq \varepsilon_3$);
        $\mu := \mu * \max(\frac{1}{3}, 1 - (2\rho - 1)^3)$; $\nu := 2$;
      else
        $\mu := \mu * \nu$; $\nu := 2 * \nu$;
      endif
    endif
  until ($\rho > 0$) or (stop)
endwhile
$\mathbf{p}^+ := \mathbf{p}$;

11.
12. 在实现上，使用 C++的矩阵库 Eigen3 完成基本的矩阵运算，使用数值法求解雅可比矩阵。
13.
14. **2.2 拓展卡尔曼滤波**
15.
16. 2.2.1 算法介绍
17. 卡尔曼滤波是一种已知系统数学模型，通过系统实际输入输出数据，将预测值和实际观测值进行数据融合，对系统状态做出最优估计的算法。
18. 扩展卡尔曼滤波（Extended Kalman Filter，EKF）是标准卡尔曼滤波在非线性

情形下的一种扩展形式，它是一种高效率的递归滤波器（自回归滤波器）。EKF 的基本思想是利用泰勒级数展开将非线性系统线性化，然后采用卡尔曼滤波框架对信号进行滤波，因此它只具有一阶精度，是一种次优滤波。

2.2.2 算法流程

状态转移方程：

$x_k = f(x_{k-1}) + u$

观测方程：

$y_k = h(x_k) + v$

u,v为过程噪声和观测噪声

将方程线性化

$x_k = F_{k-1}x_{k-1} + u$

$y_k = H_k x_k + v$

F，H为f和h的雅可比矩阵

由此得到 EKF 算法预测和更新两个步骤：

预测：

$x_p = Fx_e$

$P = FPF^T + Q$

更新：

$y_p = Hx_p$

$K = PH^T(HPH^T + R)^{-1}$

$x_e = x_p + K(y - y_p)$

$P = (I - KH)P$

式中Q为状态转移误差协方差矩阵，R 为观测误差协方差矩阵，无下标量为观测量，下标$\boldsymbol{p}$表示预测量（predict），下标$\boldsymbol{e}$表示滤波后的估计量（estimate）。

在实现上，使用 C++的矩阵库 Eigen3 完成基本的矩阵运算，使用解析法求解雅可比矩阵。

2.2.3 预测器使用的数学模型

状态转移方程：

$a_{k+1} = a_k$

$w_{k+1} = w_k$

$spd_{k+1} = a_k * sin(w_k * t_k) + (2.090 - a_k)$

$t_{k+1} = t_k + \triangle t$

$re_{k+1} = re_k * cos(spd_k * \triangle t) - im_k * sin(spd_k * \triangle t)$

$im_{k+1} = re_k * sin(spd_k * \triangle t) - im_k * cos(spd_k * \triangle t)$

观测方程:

$re = re$

51. $im = im$
52. re，im 分别为装甲板在单位圆上的实部和虚部
53. 值得注意的是，由于 EKF 只具有一阶精度，所以在最后两个方程中角度变化用 $spd * \Delta t$ 来近似表示而没有采用积分的方式。经测试，所用状态转移方程与积分表示的方程滤波效果无明显差异，且雅可比求解更加简单

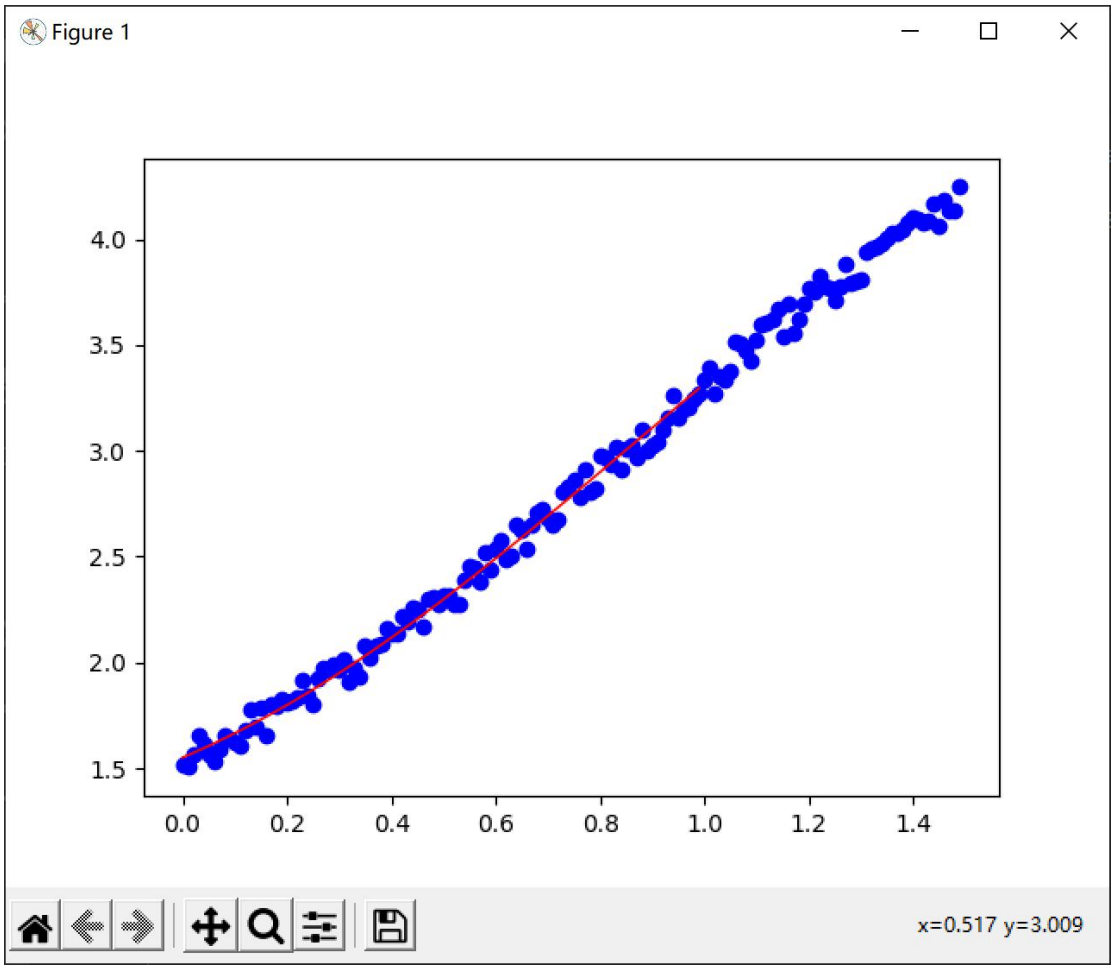
54. 算法性能与结果展示

54.1 运动方程参数估计

54.2 蓝色为采集到的角度数据，红色为最小二乘估计得到的函数曲线。

54.3 计算时间在 0.01s 以内；参数准确度：幅值误差±0.05，频率误差±0.01；使用拟合的参数初始化滤波器，收敛时间在 0.1s 以内。

54.4



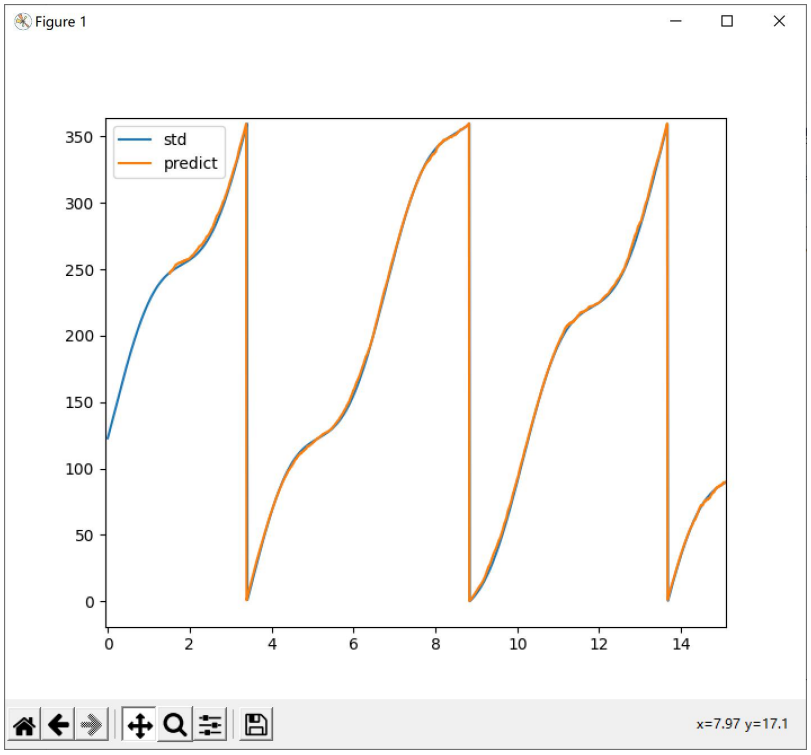

```
cal_time: 0.008974313735961914s
real_a=1.045,real_w=1.884
a_est=1.0590674613144233,w_est=1.8864665069574191
```

54.5

54.6 卡尔曼滤波跟踪和预测性能

蓝色为实际角度数据，黄色为使用滤波得到的状态量估计值预测 1s 以后

的角度 ， 误差在 ±2°以内，结果如下



可以得出结论，算法具有较好的预测性能。

四，反陀螺算法

算法简介：反陀螺已经是一种常见的技术，而且并没有过深的技术细节，因此不占用过多篇幅. 算法的主要思想在于通过观察装甲板数量改变的周期推测下一次击打的时机，然后经过延迟之后发射

算法原理：

- **反陀螺的判定：**
- 对于每一个机器人都开一个循环队列，储存装甲板数目 2->1 时刻的时间戳与装甲板中心点坐标，同时增加一个最低时间限制来防止云台抖动
- 若一定的时间范围内变换的次数超过了阈值，则将其判定为陀螺模式，也可以手动进入反陀螺模式
-
- **陀螺周期的测定**
- 进入反陀螺模式之后先舍去判定范围之外的时间戳，取临近的多个时间戳进行周期计算，并且取周期的 1/4(可以有效应对变速陀螺)
-
- **反陀螺情况下的预测算法**

- 转动预测:
- 枚举 i 的值，计算发射延迟:
- 发射延迟 $=$ 时间差 $+\frac{\text{周期}\times i}{4}-$ 飞行时间(i)
- 其中飞行时间是针对不同时间对应预测位置所需的飞行时间，是 i 的函数
- 当发射延迟>0，为合法解，返回
-
- 平移预测:
- 考虑在击打陀螺的时候，己方或敌方都可能发生移动，所以此时还需要新建一个滤波器，将循环队列中的装甲板坐标依次放入滤波器进行预测
-
- 综合上面两部分的因素之后才共同给出反陀螺模式的打击方向，经过实际测试，反陀螺算法具有较不错的可用性(算法效果见视频)