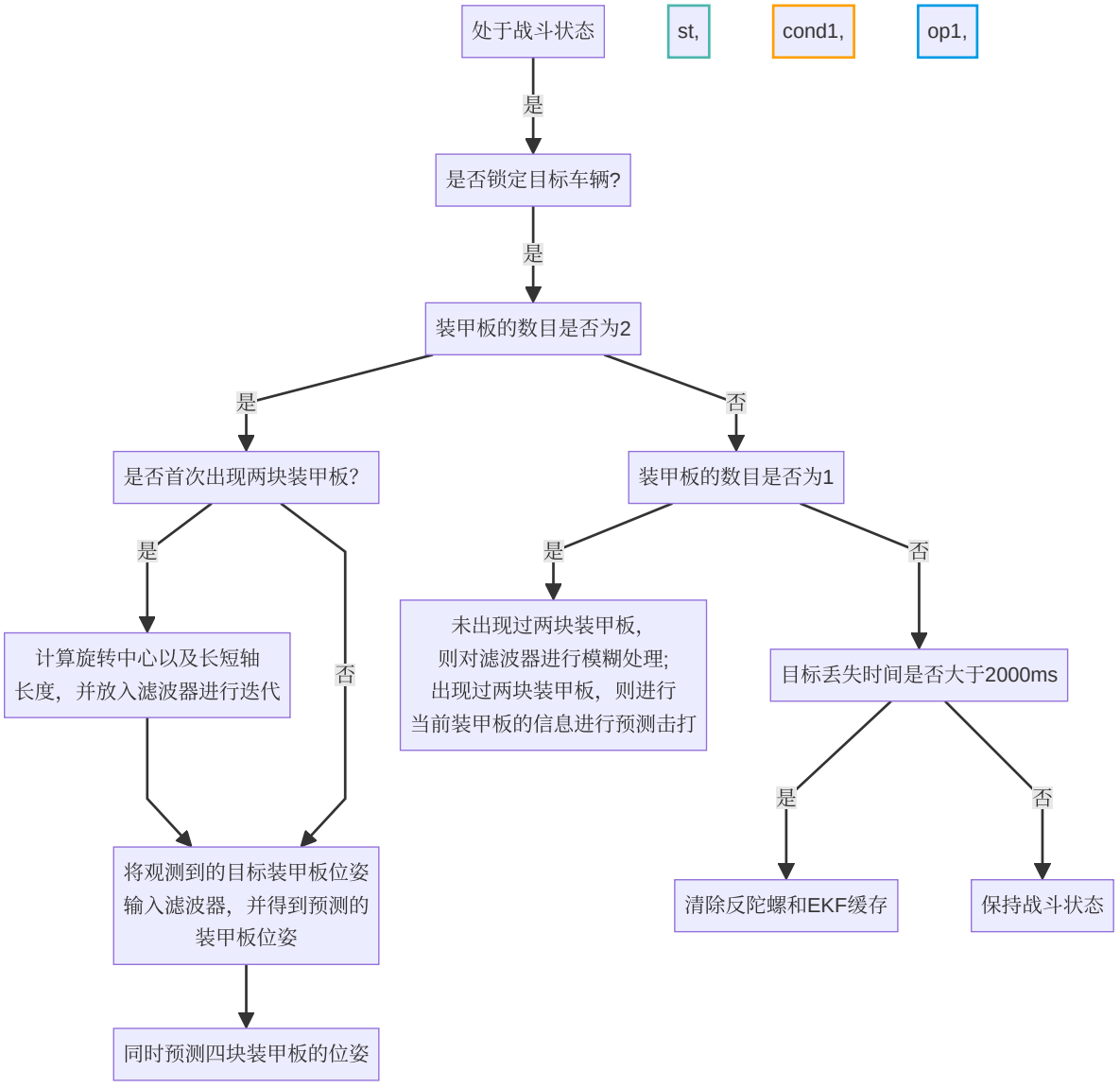
CUBOT自瞄文档

1 自瞄介绍

自瞄代码是一套适用于步兵、哨兵的通用代码，为CUBOT视觉代码的一部分，主要在`robot_brain_core.cpp`、solver文件夹中的`predictor`部分。此部分代码主要为对目标装甲板的识别、跟踪和预测，从而简化操作手的击打流程，~~并提高命中率(提高不了一点)~~。

2 自瞄思路



3 自瞄代码

> 自瞄主要由三大部分组成：瞄准、预测和反陀螺

3.1 瞄准

我们识别到装甲板后，要如何击打呢？那就需要我们能够给下位机提供枪口抬升的pitch角和旋转的yaw角。

我们在识别到装甲板后，通过pnp得到装甲板中心点的三维坐标，然后我们就可以将枪口对齐目标点了。但这又会出现新的问题，我们并没有考虑弹丸的重力！重力会使得弹丸在空中的飞行轨迹并不会是一条直线，所以显然，我们不能能够直接将枪口对其装甲板，那我们要如何才能准确的瞄准装甲板呢？

那就得用到**四阶龙格库塔法**去拟合弹道曲线，我们的代码参考了沈阳航空航天大学2022技术报告

四阶龙格库塔运用在弹丸上的原理为：

弹丸在飞行时主要受到重力和空气阻力两个力：

$$F_g = mg$$

$$F_f = \frac{1}{2}C\rho Sv^2$$

> C为空气阻力系数
>
> ρ为空气密度
>
> S为弹丸所受阻力的参考面积

其中重力竖直向下，阻力的方向和弹丸速度方向相反，所以我们可以算出两个力所提供的加速度：

$$a_g = g$$

$$a_f = \frac{\frac{1}{2}C\rho Sv^2}{m}$$

阻力加速度中除去v的其他值均为常数，故我们可以令$k = \frac{\frac{1}{2}C\rho S}{m}$,则$a_f = kv^2$

由上面的式子，我们可以推出：

$$\frac{du}{dt} = -kv^2 * cos\theta = -kvu$$

$$\frac{dw}{dt} = -kv^2 * sin\theta - g = -kwv - g$$

$$\frac{dx}{dt} = u$$

$$\frac{dy}{dt} = w$$

> 前两项为x方向上加速度和y方向上的加速度

上述方程中自变量均为时间t，这样很难算出解析解，因为我们基本无法得到各个阶段的时间，所以我们把x当作自变量，那么公式更新为：

$$\frac{dy}{dx} = p$$

$$\frac{du}{dx} = -kv$$

$$\frac{dp}{dx} = -\frac{g}{u^2}$$

所以pitch角和x的微分关系就显而易见了，CUBOT此部分的代码在**/src/solver/src/solver.cpp**中的**getPitchandYaw**函数中，该函数可以直接得到云台需要旋转的yaw和pitch角，其中龙格库塔部分的代码为：

```
//开始使用龙格库塔法求解弹道补偿
    for (int i = 0; i < max_iter; i++)
    {
        //初始化名
        float x = 0.0;
        float y = 0.0;
        float p =static_cast<float>(tan(pitch_new / 180 * CV_PI));
        float v = bullet_speed;
        float u = v / sqrt(1 + pow(p,2));
        float delta_x = dist_horizonal / R_K_iter;
        for (int j = 0; j < R_K_iter; j++)
        {
            float k1_u = -k * u * sqrt(1 + pow(p, 2));
            float k1_p = -g / pow(u, 2);
            float k1_u_sum = u + k1_u * (delta_x / 2);
            float k1_p_sum = p + k1_p * (delta_x / 2);

            float k2_u = -k * k1_u_sum * sqrt(1 + pow(k1_p_sum, 2));
            float k2_p = -g / pow(k1_u_sum, 2);
            float k2_u_sum = u + k2_u * (delta_x / 2);
            float k2_p_sum = p + k2_p * (delta_x / 2);
            float k3_u = -k * k2_u_sum * sqrt(1 + pow(k2_p_sum, 2));
            float k3_p = -g / pow(k2_u_sum, 2);
            float k3_u_sum = u + k3_u * delta_x;
            float k3_p_sum = p + k3_p * delta_x;
            float k4_u = -k * k3_u_sum * sqrt(1 + pow(k3_p_sum, 2));
            float k4_p = -g / pow(k3_u_sum, 2);
            u += (delta_x / 6) * (k1_u + 2 * k2_u + 2 * k3_u + k4_u);
            p += (delta_x / 6) * (k1_p + 2 * k2_p + 2 * k3_p + k4_p);
            x += delta_x;
            y += p * delta_x;
        }
        ......
    }
```

3.2 预测

在3.1中，我们已经可以通过四阶龙格库塔法击打目标，但在目前的比赛中，几乎所有的车可以陀螺，如果我们直接击打我们所识别到的装甲板，那么大概率击打不中，为了打中正在陀螺的目标车辆，我们需要预测装甲板的位姿。

CUBOT中使用的是拓展卡尔曼滤波（EKF）去预测目标

EKF的基本公式为：

先验预测：$x_{k|k-1} = f(x_{k-1|k-1})$

$$P_{k|k-1} = F * P_{k-1|k-1} * F^{-1} + Q$$

更新：$K = P_{k|k-1} * H^T * (H * P_{k|k-1} * H^{-1} + R)^{-1}$

$$x_{k|k} = x_{k|k-1} * K * (z_k - h(x_{k|k-1}))$$

$$P_{k|k} = (1 - K * H) * P_{k|k-1}$$

我们的预测值x为九维参数，其值为$[x, v_x, y, v_y, z, v_z, \theta, w, r]$, 观测值z为四维参数，其值为$[x, y, z, \theta]$，其中$\theta$为装甲板的yaw角

Q为观测协方差矩阵，R为预测协方差矩阵.CUBOT中的Q、R矩阵最开始均为对角阵，但效果不佳，后参考西北工业大学开源的技术文档中，将Q矩阵修改为

$$\begin{pmatrix} \frac{t^3}{3}\sigma_0 & \frac{t^2}{2}\sigma_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{t^2}{2}\sigma_0 & t\sigma_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{t^3}{3}\sigma_0 & \frac{t^2}{2}\sigma_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{t^2}{2}\sigma_0 & t\sigma_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{t^3}{3}\sigma_2 & \frac{t^2}{2}\sigma_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{t^2}{2}\sigma_2 & t\sigma_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_3 \end{pmatrix}$$

为什么Q矩阵要变成这样呢?

因为Q本质上是一个噪声矩阵，如果单把x和v_x拿出来他们的噪声有两种情况:

1. 噪声离散

此种情况下我们可以令$D(\ddot{x}) = \sigma$,则有

$$\begin{aligned} D(x) &= D(\frac{1}{2}\ddot{x}\Delta t^2) = \frac{1}{4}\Delta t^4 D(\ddot{x}) = \frac{1}{4}\Delta t^4 \sigma \\ D(\dot{x}) &= D(\ddot{x}\Delta t) = \Delta t^2 D(\ddot{x}) = \Delta t^2 \sigma \\ \operatorname{cov}(x, \dot{x}) &= \sqrt{D(x)D(\dot{x})} = \frac{1}{2}\Delta t^3 \sigma \end{aligned}$$

再根据Q的定义:

$$Q = \begin{pmatrix} D(x) & \operatorname{cov}(x, \dot{x}) \\ \operatorname{cov}(x, \dot{x}) & D(\dot{x}) \end{pmatrix}$$

即可得到x, v_x的噪声协方差。同理，可以由此推广到其他数据组上

2. 噪声连续

噪声连续的情况可以参考这个，这是github上介绍EKF比较详细的文章。

其公式为:

$$
\begin{aligned}
Q &= \int_0^{\Delta t} e^{A\tau} Q(\sigma) e^{A^T\tau}\, d\tau \\
&= \int_0^{\Delta t} \begin{pmatrix} 1 & \tau \\ 0 & 1c \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & \sigma \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \tau & 1 \end{pmatrix} d\tau \\
&= \begin{pmatrix} \frac{1}{3}\Delta t^3 & \frac{1}{2}\Delta t^2 \\ \frac{1}{2}\Delta t^2 & \Delta t \end{pmatrix} \sigma
\end{aligned}
$$

其原理不再赘述

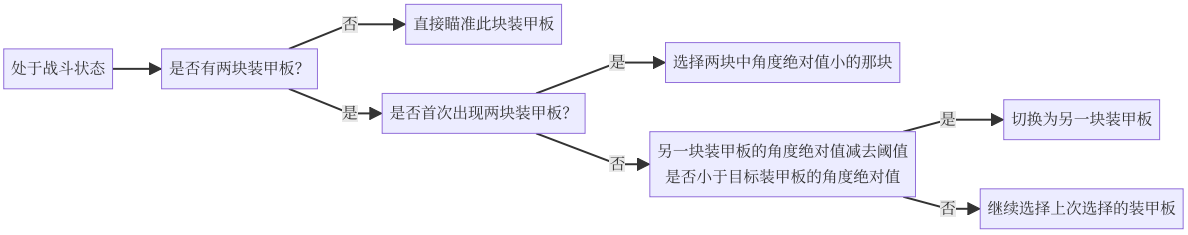
至此EKF的所有参数均能给出，但仅仅将参数给入，其给出的预测值是不准确的，其原因在于目标车辆在陀螺状态时，我们瞄准的目标装甲板是一直在变化的，我们输入的观测值会突变，而EKF的预测值无法突变，就会导致EKF的预测状态量发生无规则的跳变。为了使得预测量准确，我们必须使得观测状态量不发生跳变，我们暂未找到合适且有效的方法来实现这一想法。所以我们的做法为在目标装甲板切换时，更换滤波器的状态量，由于车的装甲板是成对的，并且就是相对的装甲板半径相等，所以我们可以将四块装甲板分成2组，每组的装甲板半径，z轴坐标都是相等的，所以在装甲板切换时，我们将滤波器中的状态值替换为另一组装甲板的状态值，那么这样，即使输入值不连续，我们的预测值也能比较准确的输出。

此部分代码在**/src/solver/src/predictor_EKF.cpp**文件中，此处不再粘贴。

3.3 反陀螺

由3.1和3.2中的内容，我们已经可以实现击打移动的目标，但问题又双叒叕出现了，在目标陀螺时，会出现两块装甲板的情况，我们该击打哪一块呢？如果击打的目标随机，那么我们的云台会出现乱抖的情况，如果我们一直选择两块中的左边或右边那块，那么我们的目标装甲板出现的范围就一直在很小的范围，就会导致我们击打效率非常低。所以我们装甲板的选定显得尤为重要。

那我们如何选定装甲板呢？我们的思路为



在此种方式下，我们能够很规律的选取陀螺状态下的装甲板。此部分代码在**/src/solver/predictor_AntiTop.cpp**文件中。

3.4 其他

上述方法基本可以稳定击打陀螺，但在此笔者还想拓展一些东西~~（夹藏私货）~~,笔者在24赛季部分时间在解决角度识别问题，上交在23赛季的青工会上提出了一种比较好的角度计算方法，我在24赛季之初就在尝试此种方法。

上交提供的思路为我们在得出装甲板中心点的世界坐标后，可以根据中心点以及给出的yaw角将装甲板的四个角点重投影到相机坐标系上，那这样，我们只要给出一个yaw角，就可以得到四个重投影角点。

这样，我们就有两组装甲板的角点，一个是识别到的装甲板角点，另一个是重投影装甲板角点。我们将两组交点作比较，其比较值可以反映出两组点的重合度。那我们就可以以yaw角为自变量，两组装甲版点的比较值为因变量，理论上就可以得到一条平滑的单极值曲线。我们在尝试此种方法的时候，上交并未提供二者比较的方法，我们在尝试多种方法后，发现在我们的框架下，两组点的iou作为因变量的效果最佳。其代码在**predictor_EKF.cpp**文件中的**drawAndCompareIOU**函数中。

在我们实际测试中，发现此曲线并非单极值曲线，而是双极值曲线，我们分析此种现象是由于互为相反数的yaw角iou非常接近，所以导致曲线关于0度近似轴对称。我们的措施为通过计算识别装甲板的左上角点和右下角点处的两个角来判断装甲板是左偏还是右偏。从而剔除一半的曲线，使得曲线强行成为单极值曲线。

我们得到了一条单极值曲线，我们只需要得到曲线在极值时的yaw角就能得到装甲板的yaw角，我们的想法是用梯度下降法去求得极值，但是我们又发现曲线抖动比较大，梯度下降法并不适用，最后我们使用的是0.618黄金分割优选法，此种方法能较好的得到yaw值。

4 推荐学习路线

自瞄对视觉新人并不友好，笔者 ~~苦不堪言~~ 深知这一点,所以结合本赛季的学习经验，给后人推荐一条自瞄的学习路线。

众所周知，自瞄最重要的就是 ~~调参~~ 学好EKF,所以我主要推荐的是学习EKF:

在学习之初，我推荐你去看b站dr_can的视频，一共6集，前5集讲的KF，非常详细的推理了KF的公式，建议跟着手推一下；最后一集是EKF的推理，也讲述了EKF的原理，但是并不详细，简单看看懂原理就行。

之后，我建议看看开源代码，比如上交21年开源代码，湖大开源代码等等。

> Robomaster论坛上可以搜索到

最后，可以看看陈君的论文，西工大的技术文档、以及上面Q矩阵噪声连续处给出的链接中的文档，这些都学习完后，相信你会对EKF有不一样的理解。

- Anthor：刘洋
- 联系方式：

  > QQ :1249968144
  >
  > weixin :19516133992