4T 四足机器人固件库
使用指南

文档版本：V1.0

四梯科技 | making growth more efficient.

一、逆运动学解析和站立姿态控制算法固件库

1. 固件概述

Leg_ik.c 与 Leg_ik.h 文件包含逆运动学求解、关节角度到舵机控制信号的转换以及姿态控制算法。该系统通过接收机体姿态指令（RPY 角）和位置偏移参数，计算出四条腿的足端坐标，进而转换为 12 路舵机的控制信号，实现机器人的稳定运动控制。

系统采用模块化设计，主要包含以下核心功能：

- 基于 RPY 角的足端位置计算
- 单腿三自由度逆运动学求解
- 关节角度到舵机 PWM 信号的转换
- 舵机控制信号的帧封装与发送

2. 核心功能模块详解

(1) 主控制函数 Dog_Control

该函数是整个腿部控制系统的入口，实现从足端坐标到舵机控制信号的完整转换流程。

```
void Dog_Control(void)
{
    float* pos[4] = { Dog.rf, Dog.lf, Dog.rb, Dog.lb };
    for (int i = 0; i < 4; ++i)
    {
        single_ik(pos[i][0], pos[i][1], pos[i][2],&hip[i], &thigh[i],  &knee[i]);
    }
    Servo_Angle_Conversion();
    Send_32Servo_HEX();
}
```

工作流程：

- 将四条腿的足端坐标（存储在 Dog 结构体中）分别传入逆运动学求解函数
- 调用 single_ik 函数计算每条腿的髋关节、大腿关节和膝关节角度
- 通过 Servo_Angle_Conversion 函数将关节角度转换为舵机 PWM 值
- 调用 Send_32Servo_HEX 函数将 PWM 值封装成通信帧并发送

(2) 逆运动学求解 single_ik

该函数实现单腿的逆运动学计算，将足端坐标 (x,y,z) 转换为三个关节的角度值。

逆运动学求解采用几何分解法，将三维空间问题分解为平面问题处理：

1．首先在 YZ 平面内计算髋关节旋转角度

2．然后在经过坐标变换的 XZ 平面内计算大腿和膝关节角度

```
void single_ik(float x, float y, float z,int16_t *h, int16_t *t, int16_t *k)
{
    // YZ 平面内计算髋关节角度
    float dyz = sqrtf(y*y + z*z);
    float lyz = sqrtf(dyz*dyz - H*H);
    float gamma_yz = -atan2f(y, z);
    float gamma_off = -atan2f(H, lyz);
    float theta0 = gamma_yz - gamma_off;
    // 计算大腿和膝关节角度
    float lxzp = sqrtf(lyz*lyz + x*x);
    float n = (lxzp*lxzp - L1*L1 - L2*L2) / (2.0f*L1);
    float theta2 = -acosf(n / L2);
    float alfa_xzp = -atan2f(x, lyz);
    float alfa_off = acosf((L1 + n) / lxzp);
    float theta1 = alfa_xzp + alfa_off;
    // 角度转换与范围限制
    *h = (int16_t)(theta0 * 180.0f/M_PI + 0.5f);
    *t = (int16_t)(theta1 * 180.0f/M_PI + 0.5f);
    *k = (int16_t)(theta2 * 180.0f/M_PI + 0.5f);
    *h = (*h < -75) ? -75 : ((*h > 75) ? 75 : *h);
    *t = (*t < -80) ? -80 : ((*t > 80) ? 80 : *t);
}
```

几何解释：

- theta0：髋关节旋转角度，使大腿杆对准足端在 YZ 平面的投影
- theta1：大腿关节角度，实现前后摆动
- theta2：膝关节角度，控制小腿伸缩

安全机制：代码对计算出的髋关节和大腿关节角度进行了范围限制，防

止机械结构超限损坏。

(3) 膝关节舵机角度转换 Knee_ServoAngle

由于膝关节采用连杆机构间接驱动，需要特殊的角度转换计算。

```
uint16_t Knee_ServoAngle(int16_t pulse_angle)
{
    // 输入角度范围限制
    int16_t constrained_angle = constrain(pulse_angle, 35, 135);
    // 角度转弧度
    float theta = constrained_angle * M_PI / 180.0f;
    float Gamma_Wucha = Gamma_deg * M_PI / 180.0f;
    // 几何计算
    float h1 = (LR * cosf(theta)) / cosf(Gamma_Wucha);
    float h2 = HH - h1;
    // 安全处理
    if (h2 < 1e-6f) return 0;
    float Lc = sqrtf(h2 * h2 + I*I);
    // 余弦定理计算
    float numerator = Lc * Lc + Rs * Rs - L * L;
    float denominator = 2.0f * h2 * Rs;
    float cos_phi = constrain(numerator / denominator, -1.0f, 1.0f);
    // 角度计算与转换
    float knee_rad = acosf(cos_phi) + acosf(h2/Lc);
    float knee_deg = knee_rad * 180.0f / M_PI;
    // 输出角度范围限制
    return (uint16_t)constrain(roundf(knee_deg), 35, 140);
}
```

转换原理：该函数考虑了膝关节的连杆机构特性，通过余弦定理和几何关系，将期望的膝关节角度转换为实际舵机需要转动的角度，同时进行了必要的误差补偿（Gamma_Wucha）。

(4) 舵机角度到 PWM 转换 Servo_Angle_Conversion

该函数将计算得到的关节角度转换为舵机所需的 PWM 占空比。

```
void Servo_Angle_Conversion(void)
{
```

```
    // 左前腿
    Servo_Pwm[3] = (uint16_t)(Hip_Left_Front_Diff - (hip[1] /     Coe_ServoAngle));
    Servo_Pwm[4] = (uint16_t)(Thigh_Left_Front_Diff + (thigh[1]/
    Coe_ServoAngle));
    Servo_Pwm[5] = (uint16_t)(Knee_Left_Front_Diff -   (Knee_ServoAngle(-knee[1])
/ Coe_ServoAngle ));
    // 右前腿
    Servo_Pwm[0] = (uint16_t)(Hip_Right_Front_Diff + (hip[0] /   Coe_ServoAngle));
    Servo_Pwm[1] = (uint16_t)(Thigh_Right_Front_Diff -(thigh[0] /
    Coe_ServoAngle));
    Servo_Pwm[2] = (uint16_t)(Knee_ServoAngle(-knee[0]) / Coe_ServoAngle +
Knee_Right_Front_Diff);
    // 左后腿
     Servo_Pwm[9] = (uint16_t)(Hip_Left_Back_Diff + (hip[3] /    Coe_ServoAngle ));
     Servo_Pwm[10] = (uint16_t)(Thigh_Left_Back_Diff + (thigh[3] /
    Coe_ServoAngle ));
    Servo_Pwm[11] = (uint16_t)(Knee_Left_Back_Diff -
    (Knee_ServoAngle(-knee[3]) / Coe_ServoAngle ));
    // 右后腿
    Servo_Pwm[6] = (uint16_t)(Hip_Right_Back_Diff - (hip[2] /   Coe_ServoAngle ));
    Servo_Pwm[7] = (uint16_t)(Thigh_Right_Back_Diff - (thigh[2] /
    Coe_ServoAngle ));
    Servo_Pwm[8] = (uint16_t)(Knee_ServoAngle(-knee[2]) /       Coe_ServoAngle+
Knee_Right_Back_Diff);
}
```

转换规则：

- PWM 值范围为 25-125，对应舵机角度 0-180°
- 左右腿采用对称转换公式，适应机械结构的对称性
- 膝关节需要先经过 Knee_ServoAngle 函数转换
- 不同关节使用不同的偏移参数（如 Hip_Left_Front_Diff）进行校准

(5) 数据发送函数 Send_32Servo_HEX

该函数将舵机 PWM 值封装成 65 字节的通信帧并发送给舵机驱动板。

```
void Send_32Servo_HEX(void)

{

    uint8_t Servo_32frame[65] = {0};

    // 填充左前腿数据

    Servo_32frame[Hip_Left_Front*2-2] = (Servo_Pwm[3] >> 8) & 0xFF;

    Servo_32frame[Hip_Left_Front*2-1] = Servo_Pwm[3] & 0xFF;

    // ... 填充其他腿数据 ...

    // 计算校验和

    uint16_t sum = 0;

    for (int i = 0; i < 64; ++i) sum += Servo_32frame[i];

    Servo_32frame[64] = (uint8_t)(sum & 0xFF);


    Send_Frame_UART2(Servo_32frame);

}
```

帧结构：

- 总长度 65 字节，前 64 字节为有效数据，最后 1 字节为校验和
- 每个舵机的 PWM 值（16 位）按高字节在前、低字节在后的方式存储
- 校验和为前 64 字节的累加和的低 8 位

(6) 姿态计算函数 calc_leg_pose

该函数根据机体姿态角（RPY）和位置偏移计算四条腿的足端坐标。

```
void calc_leg_pose(float Rol, float Pitch, float Yaw, float pos_x_mm, float pos_y_mm, float pos_z_mm)

{

    // 角度转弧度

    float r = Rol * M_PI / 180.0f;

    float p = Pitch * M_PI / 180.0f;
```

```
    float y = Yaw * M_PI / 180.0f;
    // 构造旋转矩阵 R = Rz*Ry*Rx
    float R[3][3] = {
    { cp*cy, cp*sy, -sp },
    { sr*sp*cy - cr*sy, sr*sp*sy + cr*cy, sr*cp },
    { cr*sp*cy + sr*sy, cr*sp*sy - sr*cy, cr*cp }
    };
    // 计算各腿足端坐标
    for(uint8_t i = 0; i < 4; ++i)
    {
    matrix3x3_mul_vec(R, hip[i], tmp);
    tmp[0] += pos[0] - foot[i][0];
    tmp[1] += pos[1] - foot[i][1];
    tmp[2] += pos[2] - foot[i][2];
    // 填充结果，注意右腿 Y 坐标取反
    if(i==0){ Dog.rf[0]=tmp[0]; Dog.rf[1]=-tmp[1]; Dog.rf[2]=tmp[2]; }
    // ... 其他腿坐标赋值 ...}
}
```

算法原理：

- 首先将 RPY 角（滚转角、俯仰角、偏航角）转换为旋转矩阵
- 旋转矩阵按照 RzRyRx 的顺序构造，对应绕固定坐标系的旋转
- 通过旋转矩阵和位置偏移计算各髋关节的空间位置
- 最终计算得到各足端的目标坐标，并考虑了左右腿的 Y 坐标对称性处理

3. 关键参数说明

(1) 机体几何参数

参数	数值(mm)	说明
B	104.0	两髋关节左右间距
W	188.0	左腿和右腿之间横向间距

参数	数值(mm)	说明
Len	243.0	机身前后长度

(2) 单腿机械尺寸参数

参数	数值(mm)	说明
H	42.0	髋关节到大腿电机距离
L1	110.0	大腿长度
L2	110.0	小腿长度
Rs	24.0	舵机臂长度
L	70.0	连杆长度
LR	27.0	小腿旋转轴到连杆轴的水平距离
HH	71.6	舵机轴到小腿旋转轴的垂直高度
I	13.0	连杆到小腿传动轴的垂直误差值
Gamma_deg	2.8624	误差角

(3) 舵机控制参数

参数	范围	说明
Servo_Pwm	25-125	舵机 PWM 占空比，对应舵机角度 0-180 °
髋关节角度限制	-65 ° ~65 °	髋关节机械结构限制（25-155 °）
大腿关节角度限制	-80 ° ~80 °	大腿关节机械结构限制（10-170 °）
膝关节角度限制	35 ° ~135 °	机械关节结构限制

4. 使用方法

(1) 系统初始化

- 确保所有机械参数（如 L1, L2, H 等）与实际机器人尺寸一致
- 根据舵机校准结果，调整各关节的偏移参数（如 Hip_Left_Front_Diff）
- 初始化 UART 通信接口，确保与舵机驱动板的通信正常

(2) 基本控制流程

控制示例代码：

```
// 1. 设置期望姿态和位置
float roll = 0.0f; // 滚转角
float pitch = 0.0f; // 俯仰角
float yaw = 0.0f; // 偏航角
float x = 0.0f; // X 轴偏移
float y = 0.0f; // Y 轴偏移
float z = 150.0f; // Z 轴偏移（正值即为机身高度）
// 2. 计算足端坐标
calc_leg_pose(roll, pitch, yaw, x, y, z);
// 3. 执行腿部控制
Dog_Control();
```

(3) 参数调整建议

- 机身高度调整：通过修改 pos_z_mm 参数,z 建议范围 50~180mm
- 姿态调整：RPY 角建议控制在 ±20° 范围内，避免过度倾斜
- 参数校准：更换舵机或机械结构后，需要重新校准各关节的偏移参数

5. 注意事项

1. **参数修改警告**：修改机械参数（如 L1, L2, H 等）会直接影响逆运动学计算结果，建议修改后进行实物测试前先进行仿真验证

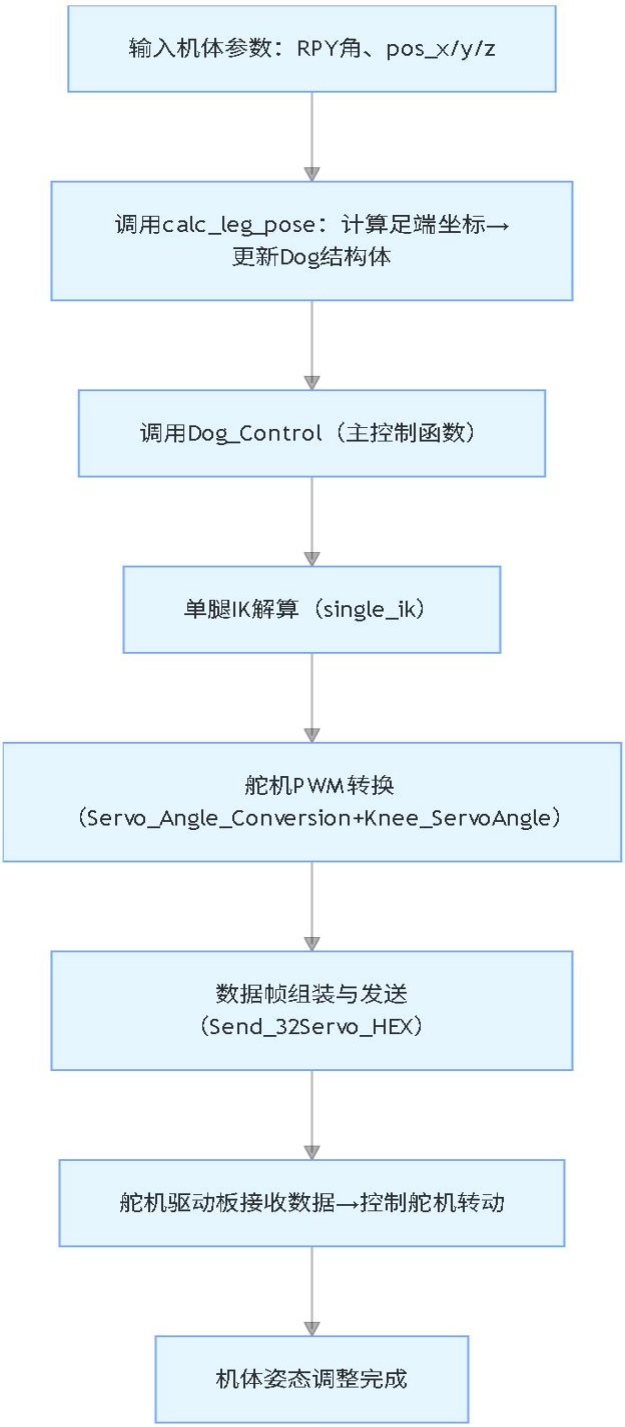
2. **角度范围限制**：代码中已实现基本的角度范围限制，但在实际应用中仍需避免发送可能导致机械结构超限的姿态指令

3. **膝关节特殊性**：膝关节由于采用连杆机构，其角度转换存在非线性关系，更换膝关节机械结构后必须重新校准 Knee_ServoAngle 函数中的参数

4. **通信可靠性**：舵机控制帧包含校验和机制，若出现通信异常，可检查校验和计算是否正确

5. 计算稳定性：逆运动学计算中使用了平方根和反三角函数，需注意避免输入参数导致数学计算异常（如 acos 输入超出 [-1,1] 范围）

6. 四足机器人姿态控制流程图：



二、运动控制算法及辅助功能固件库

1. 固件概述

Dog_Sport Dog_Sport 固件库由 Dog_Sport.c 与 Dog_Sport.h 文件组成，是四足机器人运动控制的核心上层模块。该库依赖 Leg_ik 固件库（逆运动学与舵机控

制），通过封装步态生成、姿态补偿、状态监测等功能，实现机器人“站立初始化”到“动态运动”的全流程控制，并提供电池电压、温度等关键状态的采集与反馈。

系统采用 “模块化 + 分层控制” 设计，核心功能覆盖：

- 对角 Trot 小跑步态生成（支持前进 / 后退、转向、横向移动）
- 平滑站立控制（从低高度到目标高度的逐步抬升）
- 基于 PID 的姿态补偿（通过 LSM6DSOW 传感器数据校正横滚角、俯仰角）
- 六轴参数插补平滑（横滚 / 俯仰 / 偏航角、X/Y/Z 轴位置，避免运动突变）
- 硬件状态监测（电池电压采集、温度采集）
- 直行行驶偏航矫正功能

2. 核心功能模块详解

(1) 模式控制函数 Mode_Control

该函数是 Dog_Sport 的入口，根据 DogCtrl 结构体的 Gait_Mode 参数，切换机器人的控制模式（站立/运动），是上层控制与下层步态 / 姿态算法的桥梁。

void Mode_Control(void)

工作流程：

启动状态判断：仅当 Start_Flag=1（Stand_Up_Smoothly 执行完成，机器人已站立）时，才执行后续控制逻辑；Start_Flag=0 时（站立未完成），不响应任何控制指令，避免运动异常。

模式切换：

- 若 Gait_Mode=0（运动模式）：调用 Cal_trot()函数，执行 Trot 步态计算，实现前进、转向等动态运动。
- 若 Gait_Mode=1（平衡站立模式）：调用 PID_Control()函数，基于传感器反馈进行姿态补偿，维持稳定站立。

算法原理：

通过 Start_Flag 实现 “站立 - 运动” 的安全切换，避免未站立时发送运动指令导

致机器人倾倒；以 Gait_Mode 为核心切换条件，屏蔽下层算法细节，为上层控制提供统一的调用接口，降低使用复杂度。

(2) Trot 步态计算函数 Cal_trot

该函数是运动模式的核心，实现四足机器人对角 Trot 步态（右前 - 左后、左前 - 右后两组对角腿交替运动）的轨迹生成，支持前进 / 后退、转向、侧向移动的组合控制。

void Cal_trot(void);

工作流程：

1. 腿部原点更新：调用 update_leg_origins()，基于 DogCtrl 参数和插值平滑结果，更新四条腿的初始位置（rf0/lf0/rb0/lb0），确保姿态与位置过渡平滑。
2. 航向角校正：调用 Yaw_Correct(DogCtrl.Motion_Speed)，基于 LSM6DSOW 偏航角数据，计算航向补偿偏移量 Offset_Trun，避免直线运动时偏离航向。
3. 模式与速度判断：若 Gait_Mode=1（平衡模式）或 Motion_Speed=0（停止状态），直接返回，不执行步态计算。
4. 步态周期参数初始化：
   - t：当前步态时刻（初始为 0.0f，范围 0~Ts），随 Motion_Speed 递增（t += DogCtrl.Motion_Speed / 100）。
   - Ts=1.0f：步态周期（固定值，确保步态节奏稳定）。
   - faai=0.5f：相位系数，将步态周期分为前、后两半（各占 50%），实现对角腿交替运动。
5. 运动参数计算：
   - 提取 Stride_Length（步长）、Turn_Speed（转向速度）、Side_Length（侧向距离）等参数。
   - 计算左右腿速度：vL = fwd + turn + Offset_Trun（右前 / 左后腿速度）、vR = fwd - turn - Offset_Trun（左前 / 右后腿速度），调用 clamp()限制在 ±70mm 范围。
6. 步长起始 / 结束值计算：调用 step_len_X()和 step_len_Y()，根据运动方向（前进 / 后退、左移 / 右移），计算 X/Y 方向的步长起始（xs）和结束（xf）值。
7. 分周期生成腿部轨迹：
   - 前半周期（t ≤ Ts*faai）：右前 + 左后腿抬起，左前 + 右后腿落地。通过正

弦曲线计算抬升量 zep = h*(1 - cosf(sigma))*0.5（sigma 为相位角，0~2π），实现平滑抬腿；同时计算 X/Y 方向位移，更新足端坐标（Dog.rf/Dog.lb）。

- 后半周期（t > Ts*faai）：左前 + 右后腿抬起，右前 + 左后腿落地。重复正弦抬升逻辑，更新 Dog.lf/Dog.rb 坐标。

8. 周期重置：当 t ≥ Ts 时，将 t 置为 0.0f，开始下一个步态周期。

算法原理：

1. 对角腿交替运动：通过 “前半周期右前 - 左后抬升、后半周期左前 - 右后抬升”的逻辑，模拟动物小跑姿态，提升运动稳定性。
2. 正弦曲线抬腿：相比线性抬腿，正弦曲线的抬升量从 0 平滑过渡到 h（抬腿高度），再平滑回落，避免腿部与地面的冲击。
3. 多运动组合：通过 vL 与 vR 的差值实现转向，结合 Side_Length 实现侧向移动，支持 “前进 + 转向” “侧向 + 前进” 等复合运动。

(3) 平滑站立函数 Stand_Up_Smoothly

该函数实现机器人从初始低位（Z=50mm、X=30mm）到目标站立姿态（Z=140mm、X=0）的平滑过渡，避免起身时的机械冲击。

void Stand_Up_Smoothly(void);

工作流程：

1. 启动标志置零：设置 Start_Flag=0，禁止 Mode_Control()执行任何控制逻辑，防止站立过程中被干扰。
2. 线性过渡循环：
   - 初始化循环变量：z（Z 轴高度）从 50.0f 开始，x（X 轴位移）从 30.0f 开始。
   - 逐步更新参数：z 按 1.0f / 步递增至 140.0f，x 按 1.0f/3 步递减至 0（确保 X 位移与高度同步过渡）。
   - 刷新腿部位置：每次更新 DogCtrl.Z_data 和 DogCtrl.X_data 后，调用 update_leg_origins()，基于插值平滑更新腿部坐标，确保舵机逐步响应。
   - 延迟等待：每步调用 Delay_MS(10)（10ms 延迟），给舵机足够时间转动到目标位置，避免运动卡顿。
3. 站立完成通知：循环结束后，通过 send_str()和 Send_BLE()发送 “STAND_UP_DONE”消息，设置 Start_Flag=1，允许后续模式切换。

算法原理：

通过 “线性参数更新 + 固定延迟” 的组合，将 “快速起身” 拆解为多个小步，每个小步的参数变化量（1mm 高度、~0.33mm X 位移）在舵机响应范围内，避免因参数突变导致的机身晃动或舵机堵转。

(4) PID 控制函数：pid_update 与 PID_Control

① PID 控制器更新函数 pid_update

该函数实现位置式 PID 算法，计算姿态补偿值，是 PID_Control 的核心计算单元。

float pid_update(PID_t *pid, float target, float actual);

输入参数：

- pid：PID 结构体指针（含 kp/ki/kd/out_lim 等参数）。
- target：姿态目标值（如 DogCtrl.Roll_data）。
- actual：姿态实际值（如-LSM6DSOW_Roll，传感器反馈）。

工作流程：

1. 误差计算：err = target - actual，获取目标值与实际值的偏差。
2. 积分项处理：
   - 积分项限幅：若 pid->integral > 50.0f 或< -50.0f，强制置为 50.0f 或 -50.0f，防止积分饱和。
   - 积分项累加：若积分项在范围内，pid->integral += err，累积历史误差以消除静差。
3. 微分项计算：derivative = err - pid->last_err，计算误差变化率，提升动态响应；更新 pid->last_err = err，为下一次计算准备。
4. 小误差清零：若 fabsf(err) < 0.5f（误差小于 0.5 度），将 pid->integral = 0.0f，避免微小误差累积导致姿态震荡。
5. 输出计算与限幅：out = pid->kp*err + pid->ki*pid->integral + pid->kd*derivative，计算 PID 输出；若 out 超出 pid->out_lim（如滚转 ±25 度），强制置为限幅值，返回最终输出。

算法原理：

采用位置式 PID 结构，通过比例项（kp*err）快速响应偏差、积分项（ki*integral）

消除静差、微分项（kd*derivative）抑制震荡，结合积分限幅与输出限幅，确保补偿值在安全范围内，避免姿态失控。

② PID 姿态控制函数 PID_Control

该函数实现平衡站立模式下的姿态稳定控制，基于 pid_update 的补偿值，通过插值平滑后调整腿部姿态。

void PID_Control(void);

工作流程：

1. 补偿值计算：
   - 滚转补偿：roll_comp = pid_update(&roll_pid, DogCtrl.Roll_data, -LSM6DSOW_Roll)，基于滚转角目标值与传感器实际值的误差，计算补偿量。
   - 俯仰补偿：pitch_comp = pid_update(&pitch_pid, DogCtrl.Pitch_data, LSM6DSOW_Pitch)，同理计算俯仰角补偿量。
2. 插值平滑设置：将 interp_roll_pid.target 设为 roll_comp，interp_pitch_pid.target 设为 pitch_comp，为补偿值平滑过渡准备。
3. 补偿值平滑：调用 Interp_Update(&interp_roll_pid)和 Interp_Update(&interp_pitch_pid)，使补偿值从当前值逐步逼近目标值（步长 0.35 度 / 帧），避免姿态突变。
4. 姿态应用：调用 calc_leg_pose()（来自 Leg_ik 库），将平滑后的补偿值（interp_roll_pid.current/interp_pitch_pid.current）代入，计算腿部足端坐标，实现姿态校正。

算法原理

通过 PID 补偿消除 “目标姿态 - 实际姿态” 的偏差，结合插值平滑避免补偿值突变导致的机身抖动，最终通过逆运动学计算将姿态补偿转化为腿部位置调整，实现平衡站立（如机器人倾斜时，自动调整对应腿部高度，恢复水平姿态）。

(5) 航向角校正函数 Yaw_Correct

该函数基于 LSM6DSOW 偏航角（Yaw）数据，计算航向补偿偏移量 Offset_Trun，确保机器人直线运动时不偏离航向。

void Yaw_Correct(uint8_t motion_speed);

输入参数：

motion_speed：当前运动速度（DogCtrl.Motion_Speed），用于判断机器人是否处于

运动状态。

工作流程：

1. 主动转向判断：若 DogCtrl.Turn_Speed != 0（存在主动转向指令），清零 Offset_Trun 并返回，避免校正与主动转向冲突。
2. 参考航向角记录：
   - 定义静态变量 yaw_ref（参考航向角）和 moving_last（上一帧运动状态）。
   - 若 moving_last=0 且 motion_speed>0（运动刚启动），记录当前偏航角为 yaw_ref：yaw_ref = -LSM6DSOW_Yaw，并处理 360 度循环（yaw_ref <0 则加 360，⩾360 则减 360）。
3. 运动中误差计算：
   - 若 motion_speed>0（运动中），获取当前偏航角 yaw_now = -LSM6DSOW_Yaw，同样处理 360 度循环。
   - 计算角度误差 err = yaw_now - yaw_ref，处理跨 0 度的误差（如 err>180 则减 360，err<-180 则加 360），确保误差在 ±180 度范围内。
4. 补偿值计算与限幅：Offset_Trun = clamp(err * k_yaw, -30.0f, 30.0f)（k_yaw=1.5f 为校正系数），限制补偿量在 ±30 度，避免过度校正。
5. 调试信息发送：格式化 Offset_Trun 为字符串，通过串口和 BLE 发送，用于调试时验证航向校正效果。

算法原理：

通过记录运动起始时的“参考航向角”，实时对比当前航向角与参考值的偏差，将偏差转化为腿部速度补偿（Offset_Trun），在 Cal_trot()中调整 vL 与 vR 的差值，使机器人向偏差反方向微调，维持直线运动轨迹（如向右偏离时，减小右前腿速度、增大左前腿速度，向左修正）。

(6) 腿部原点更新函数 update_leg_origins

该函数是姿态与位置平滑的核心中间层，基于 DogCtrl 参数和插值结果，更新腿部初始位置（rf0/lf0/rb0/lb0），为步态计算或姿态控制提供基础数据。

static void update_leg_origins(void);

工作流程：

1. 参数范围限制：调用 DogCtrl_Clamp(), 确保 DogCtrl 的所有参数（如 Roll_data/Z_data）

在安全范围内，避免异常参数导致后续计算错误。

2. 插值目标设置：将各插值变量的 target 设为 DogCtrl 的对应参数：
   - interp_roll.target = DogCtrl.Roll_data
   - interp_x.target = DogCtrl.X_data
   - 同理设置 interp_pitch/interp_yaw/interp_y/interp_z 的 target。
3. 插值更新：调用 Interp_Update()更新所有插值变量的 current 值，使参数从当前值逐步逼近 target（如 interp_z.current 从 100mm 逐步过渡到 140mm）。
4. 姿态计算：调用 calc_leg_pose()，传入插值后的 current 值（interp_roll.current/interp_z.current 等），计算四条腿的实时足端坐标（存储在 Dog 结构体中）。
5. 原点保存：将 Dog 结构体中的足端坐标赋值到 rf0/lf0/rb0/lb0 数组，作为后续步态计算（Cal_trot）的“初始位置”。

算法原理：

通过“参数限幅→插值平滑→姿态计算→原点保存”的流程，屏蔽参数突变和异常值的影响，为下层算法提供“平滑、安全”的腿部初始位置，是运动平稳性的关键保障（如 Z_data 从 100mm 改为 140mm 时，interp_z 会使高度逐步抬升，而非瞬间跳变）。

(7) 参数范围限制函数 DogCtrl_Clamp

该函数是运动安全的核心保障，将 DogCtrl 结构体的所有参数约束在预设安全范围内，防止参数超限导致机械损坏或运动失控。

void DogCtrl_Clamp(void);

工作流程：

调用静态 clamp()函数（float clamp(float v, float min, float max)），对 DogCtrl 的每个参数进行范围限制：

- Gait_Mode：clamp(DogCtrl.Gait_Mode, 0, 1)（仅 0/1 两种模式）。
- Stride_Length：clamp(DogCtrl.Stride_Length, -70.0f, 70.0f)（步长 ±70mm）。
- Motion_Speed：clamp(DogCtrl.Motion_Speed, 0, 5)（速度等级 0~5）。
- Turn_Speed：clamp(DogCtrl.Turn_Speed, -70.0f, 70.0f)（转向速度 ±70mm/s）。
- Raise_Leg_Height：clamp(DogCtrl.Raise_Leg_Height, 10.0f, 50.0f)（抬腿高度

10~50mm）。

- 其余参数（Side_Length/Roll_data/Z_data 等）均按 DogCtrl_t 结构体定义的范围限幅，如 Z_data 限制在 40~190mm。

算法原理：

采用线性限幅逻辑：若参数 v < min，返回 min；若 v > max，返回 max；否则返回 v。通过强制约束参数范围，避免外部指令（如误输入 Stride_Length=100mm）或计算异常导致的腿部运动超限（如小腿与地面碰撞）、舵机堵转（超出机械角度范围）等问题。

(8) 硬件状态监测函数：Read_Battery_Volt 与 Read_Temp

① 电池电压读取函数 Read_Battery_Volt

该函数通过 ADC 采集电池分压信号，计算实际电池电压，并通过串口 / BLE 发送数据，用于低电量报警或电量监测。

void Read_Battery_Volt(void);

工作流程：

1. ADC 值读取：读取寄存器 REG_ADDR[43]的 ADC 采样值(对应电池分压后的电压)。
2. 电压计算：按公式 Battery_Volt = ((REG_ADDR[43]*3.3f)/1024)*17/2 + 0.1 计算实际电压：
   - REG_ADDR[43]*3.3f/1024：将 12 位 ADC 值（0~1023）转换为分压后的电压（0~3.3V）。
   - *17/2：根据硬件分压电路（如 2kΩ 与 15kΩ 电阻分压），还原电池端电压（分压比 2:(2+15)=2:17）。
   - +0.1：零点补偿，修正 ADC 采样误差。
3. 数据格式化与发送：通过 sprintf()将电压格式化为 “Battery_Volt:XX.XX V”（保留 2 位小数），调用 send_str()（串口）和 Send_BLE()（无线）发送数据。

算法原理：

基于 ADC 采样与分压电路原理，通过硬件电阻分压将高电压（如电池 7.4V）降低到 ADC 量程（3.3V）内，再通过软件计算还原实际电压，实现电池状态的实时监测。

② 温度读取函数 Read_Temp

该函数通过 ADC 采集 NTC 热敏电阻的电压信号，计算 NTC 阻值，再匹配预设电阻 - 温度表，得到当前温度。

void Read_Temp(void);

工作流程：

1. NTC 电压计算：读取 REG_ADDR[44] 的 ADC 值，按 NCP_Volt = (REG_ADDR[44]*3.3f)/1024 转换为 NTC 两端的电压。
2. NTC 电阻计算：按公式 NCP_Ohm = (20*NCP_Volt)/(3.3f - NCP_Volt)计算 NTC 阻值（20 为串联的固定电阻值，单位 kΩ）。
3. 温度匹配：遍历 NCP_List (预设的电阻 - 温度对应表，共 24 个元素，索引 0=0℃，索引 23=115℃），找到第一个小于 NCP_Ohm 的电阻值，记录其索引 NCP_Num。
4. 温度计算：Temp = NCP_Num * 5（每级间隔 5℃，如索引 3 对应 15℃）。
5. 数据发送：格式化温度为 “Temp:XX ℃”，通过串口和 BLE 发送。

算法原理：

利用 NTC 热敏电阻 “阻值随温度升高而减小” 的特性，通过 ADC 采集电压计算阻值，再与预设的 NCP_List 匹配，将阻值转换为温度等级，实现环境或硬件温度的低成本监测。

3. 关键参数说明

(1) 运动控制核心参数

参数	范围	单位	说明	默认值
Gait_Mode	0~1	无	控制模式：0 = 运动模式，1 = 平衡站立模式	0
Stride_Length	−70~70	mm	单步长度：正值前进，负值后退	50.0f
Motion_Speed	0~5	无	速度等级：0 = 停止，等级越高速度越快	0.0f
Turn_Speed	−70~70	mm/s	转向速度：正值右转向，负值左转向	0.0f
Raise_Leg_Height	10~50	mm	运动模式下抬腿高度	20.0f
Side_Length	−60~60	mm	侧向移动距离：正值右移，负值左移	0.0f
Roll_data	−25~25	度	滚转角目标值（左右倾斜）	0.0f
Pitch_data	−20~20	度	俯仰角目标值（前后倾斜）	0.0f
Yaw_data	−20~20	度	偏航角目标值（转向，舵机狗效果差）	0.0f
X_data	−50~50	mm	机体 X 轴位移（前后）	0.0f
Y_data	−50~50	mm	机体 Y 轴位移（左右）	0.0f
Z_data	40~190	mm	机体高度（仅正值有效）	140.0f

(2) PID 控制参数

参数	数值	说明	对应控制量
roll_pid.kp	0.6f	滚转 PID 比例系数，快速响应偏差	滚转角姿态补偿
roll_pid.ki	0.03f	滚转 PID 积分系数，消除静差	滚转角姿态补偿
roll_pid.kd	0.01f	滚转 PID 微分系数，抑制震荡	滚转角姿态补偿
roll_pid.out_lim	25.0f	滚转 PID 输出限制（度），防止补偿超限	滚转角姿态补偿
pitch_pid.kp	0.6f	俯仰 PID 比例系数，快速响应偏差	俯仰角姿态补偿
pitch_pid.ki	0.03f	俯仰 PID 积分系数，消除静差	俯仰角姿态补偿
pitch_pid.kd	0.01f	俯仰 PID 微分系数，抑制震荡	俯仰角姿态补偿
pitch_pid.out_lim	20.0f	俯仰 PID 输出限制（度），防止补偿超限	俯仰角姿态补偿
PID 积分项限幅	±50.0f	PID 积分项累加范围，防止积分饱和	所有 PID 补偿
PID 小误差阈值	<0.5f	积分项清零条件（误差绝对值），避免静差累积	所有 PID 补偿

(3) 插值平滑参数

插值变量	step 值	单位	说明	对应参数
interp_roll	1.0f	度 / 帧	滚转角每帧最大变化量	Roll_data

插值变量	step 值	单位	说明	对应参数
interp_pitch	1.0f	度 / 帧	俯仰角每帧最大变化量	Pitch_data
interp_yaw	1.0f	度 / 帧	偏航角每帧最大变化量	Yaw_data
interp_x	3.5f	mm / 帧	X 轴位移每帧最大变化量	X_data
interp_y	3.5f	mm / 帧	Y 轴位移每帧最大变化量	Y_data
interp_z	3.5f	mm / 帧	Z 轴高度每帧最大变化量	Z_data
interp_roll_pid	0.35f	度 / 帧	滚转 PID 补偿每帧最大变化量	roll_comp
interp_pitch_pid	0.35f	度 / 帧	俯仰 PID 补偿每帧最大变化量	pitch_comp

4. 使用方法

(1) 系统初始化

在 mian()函数的 while (1)循环之前调用 Stand_Up_Smoothly()函数使机器人站立。

(2) 基本控制流程

- 在 20ms 定时器 1 中调用：Dog_Control()函数通过足端坐标实时控制舵机。
- 在另一个 20ms 定时器中调用 Mode_Control()函数
- 通过串口修改运动控制核心参数，即可控制机器狗的姿态和运动

5. 注意事项

① 启动标志（Start_Flag）控制：

- Start_Flag=0 时（站立未完成），Mode_Control()不执行任何逻辑，需等待 Stand_Up_Smoothly()完成后再操作，避免机器人倾倒。
- 强制中断站立过程需手动置 Start_Flag=0 并重置 DogCtrl，防止残留参数导致异常。

② 参数范围限制（DogCtrl_Clamp）：所有 DogCtrl 参数修改后必须调用 DogCtrl_Clamp()，否则参数可能超限（如 Stride_Length=-80mm），导致腿部运动超限或舵机堵转。

③ 航向角校正生效条件：仅当 Turn_Speed=0（无主动转向）时 Yaw_Correct()生效，主动转向时 Offset_Trun 清零；运动暂停后恢复，会重新记录参考航向角，

需注意直线运动连续性。

④ 插值平滑依赖：姿态、位置、PID 补偿参数均依赖 Interp_Update 实现平滑，若直接修改 current 值会导致运动突变；新增插值变量需确保 step 与控制周期匹配（如 10ms 周期下，step=3.5f 表示每秒变化 350mm）。

⑤ PID 稳定性保障：

- 积分项限幅（±50.0f）和小误差阈值（0.5f）建议保持默认，修改可能导致积分饱和或静差累积。
- 平衡模式下需确保传感器数据稳定（无剧烈噪声），否则 PID 会基于错误值计算补偿，导致姿态震荡，可添加数据滤波逻辑。

⑥ 硬件监测可靠性：

- 电池电压计算公式依赖硬件分压电路，若电阻值修改，需同步更新公式（*R_total/R_divider）。
- 更换 NTC 型号后需重新校准 NCP_List，否则温度误差会超过 ±5℃，影响过热保护。