2025 年全国大学生电子设计竞赛

简易自行瞄准装置（E 题）

2025 年 8 月 3 日

摘　　要

（小四、宋体，300 字以内）

关键词：脉宽；脉冲；数显；电容（小四、宋体）

简易自行瞄准装置（E 题）

【本科组】

一、系统方案

本系统主要由单片机控制模块、自动循迹模块、自动瞄准模块、电源模块组成，下面分别论证这几个模块的选择。

1、主控制器件的论证与选择

1.1.1 控制器选用

单片机比较

方案一：采用传统的 TIMO 系列 8 位单片机

- 优点：价格低廉，资料丰富，开发门槛低，外围电路简单。
- 缺点：主频低（一般 ≤ 80 MHz），Flash/ RAM 资源有限，片上外设较少，功耗控制与扩展性均一般，难以满足本项目对高速主频、多通道 PWM 及实时通信处理数据包的需求。

方案二：采用 STM32G 系列 32 位单片机

- 优点：基于 Cortex-M4 内核，主频最高可达 170 MHz；Flash 容量 128 KB－512 KB，SRAM 36 KB－128 KB，资源丰富；集成 12-bit 2.5 MSPS ADC、DAC、USB FS、CAN-FD、多种定时器及加密引擎，可满足高速采样、复杂算法和多种通信协议；
- 缺点：单价略高于 8 位机，BGA/QFN 封装对工艺要求稍高。

通过比较，本项目对瞄准模块运算性能、外设资源及后期可扩展性均有较高要求，但对循迹小车要求不高，且有利于低功耗运行，符合节能环保，所以瞄准模块选用了 STM32G474QET6 单片机，而循迹小车选用了 TIMOG3507 芯片驱动。

1.1.2 控制系统及电源方案选择

方案一：采用在面包板上搭建简易单片机系统

在面包板上搭建单片机系统可以方便的对硬件做随时修改，也易于搭建，但是系统连线较多，不仅相互干扰，使电路杂乱无章，而且系统可靠性低，不适合本系统使用。

方案二：模块化 PCB 设计

将系统拆分为三块独立 PCB：

核心板——集成 STM32G 系列 MCU、电源管理、时钟、调试接口与最小系统；
驱动板——承载栅极驱动、隔离电源、信号调理及保护电路，可适配不同功率器件；
功率板——布置大电流走线、功率器件、散热器及传感器接口，便于独立散热与维护。
优点：
各板功能清晰，便于并行开发、测试与维护；
单块 PCB 面积小、成本低，打样周期短，风险可控；
若后期升级，只需更换其中一块板即可，扩展性强。
通过比较，本项目选择“模块化 PCB 设计”方案，即采用核心板＋驱动板＋功率板的三板分离架构。

2、核心驱动模块及方案设计

1.2.1 直流电动机驱动模块

方案一：基于 L298N 的驱动模块。 L298N 是一款接受高电压的电机驱动器，直流电机和步进电机都可以驱动。一片驱动芯片可同时控制两个直流减速电机做不同动作，在 6V 到 46V 的电压范围内，提供 2 安培的电流，并且具有过热自断和反馈检测功能，但是电流过大可能引起 L298N 的烧毁。

方案二：

1.2.2 巡线模块

方案一：基于光电对管原理的巡线模块。基于 TCRT5000 红外光电传感器设计的一款红外反射式光电开关。传感器采用高发射功率红外光电二极管和高灵敏度光电晶体管组成，输出的信号经施密特电路整形。

方案二：摄像头的巡线模块

1.2.3 数字图像信息识别模块

方案一：基于 K210 的数字图像信息识别模块，其具有双核 64 位处理器，并自带独立 FPU；有一块 KPU 用于神经网络加速单元；还有一块 APU 用于语音数据处理。。

方案二：基于 Maxicam Plus 的数字图像信息识别模块

二、电路与程序设计

1、电路的设计

2.1.1 系统总体框图

系统总体框图如图 X 所示，XXXXXX

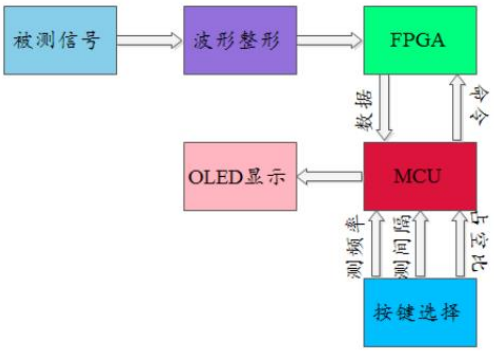

图 X　　系统总体框图

2.1.2 核心版框图及原理图

1、XXXX 子系统框图

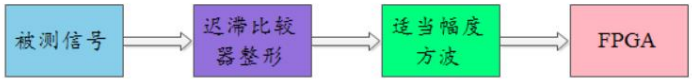

图 X　　XXXX 子系统框图

2、XXXXX 子系统电路

图 X　　XXXX 子系统电路

2.1.3 驱动板框图及原理图

1、XXXX 子系统框图

图 X　　XXXX 子系统框图

2、XXXXX 子系统电路

图 X　　XXXX 子系统电路

2.1.4 电源（功率板）

三、系统软件系统设计分析

1、根据题目要求，设计了如下工作总流程图：

图 2.1.1 系统总体工作流程图

2、程序的设计

3.2.1 程序功能描述与设计思路

1、主控模块

主控分为瞄准模块和自动循迹小车模块，对于瞄准模块采用 CMSIS-DSP 增量式二阶后项查分 PID 算法，来控制舵机转向角度；对于自动循迹小车，只需要采用经典位置式 PID 算法，来控制小车方向。

```
// Incremental PID calculation using DSP library
float PID_Calculate_DsP(PIDController *pid, float current_value, float target)
{
    // Create a PID instance
    arm_pid_instance_f32 pid_instance;
    arm_pid_init_f32(&pid_instance, 1);  // Initialize the PID instance
    // Set PID parameters
    pid_instance.Kp = pid->Kp;
    pid_instance.Ki = pid->Ki;
    pid_instance.Kd = pid->Kd;
    // Compute PID output
    float output = arm_pid_f32(&pid_instance, target - current_value);
    // Apply feed-forward gain if required
    output += pid->kf * (target - pid->pre_target);
    // Clamp output to maximum limits
    if (output > pid->out_max)
        output = pid->out_max;
    else if (output < -pid->out_max)
        output = -pid->out_max;
    // Update previous target
    pid->pre_target = target;
```

```
    return output;
}
```

图 3.2.1.1 瞄准模块二阶后项差分 PID 代码

```
float pid_calculate(PIDcontroller *pid, float current_value, float target)
{
    float speed_ratio;
    // Calculate error
    float error = target - current_value;

    // Low-pass filter, no filter when a == 0
    error = (1 - pid->a) * error + pid->a * pid->pre_error;
    float abs_error = myabs(error);

    // Dead zone judgment
    if (abs_error < pid->Dead_Zone)
    {
        error = 0.0f;                  // Set error to 0 within dead zone
        abs_error = 0.0f;              // Reset absolute error
        // pid->target = current_value; // Optionally update target
    }

    float P_out = pid->Kp * error;

    float I_out = pid->Ki * pid->Integral;
    pid->Integral += error * speed_ratio * pid->D_T;

    // Integral windup limit
    pid->Integral = limit(pid->Integral,
                          -pid->I_out_max / pid->Ki,
                          pid->I_out_max / pid->Ki);
    I_out = pid->Ki * pid->Integral;

    float D_out = pid->Kd * (error - pid->pre_error) / pid->D_T;

    // Feed-forward term
    float F_out = pid->Kf * (target - pid->pre_target);

    // Calculate total output
    float output = P_out + I_out + D_out + F_out;

```

```
    // Clamp output to maximum limits
    output = limit(output, -pid->Out_max, pid->Out_max);

    // Update previous values
    pid->pre_error  = error;
    pid->pre_target = target;

    return output;
}
```

图 3.2.1.2 自动循迹位置式 PID 代码

3、图像模块

四、测试方案与测试结果

1、测试条件与仪器

4.2.1 测试条件：

keil uvsion5: 单片机开发平台

Altium Designer: PCB 电路板设计

4.2.2 测试仪器：高精度的数字毫伏表，模拟示波器，数字示波器，数字万用表，指针式万用表。

2、测试结果及分析

（1）测试结果(数据)

循迹小车移速测试结果：　　（单位/s）

单圈时长	0.2050	0.2100	0.2045	0.4026	1.007	1.542	1.669	1.999
显示	0.2051	0.2100	0.2044	0.4026	1.006	1.542	1.669	1.999

瞄准模块打靶精度测试结果：　　（单位/mm）

单圈时长	0.2050	0.2100	0.2045	0.4026	1.007	1.542	1.669	1.999
显示	0.2051	0.2100	0.2044	0.4026	1.006	1.542	1.669	1.999

（2）测试分析与结论

根据上述测试数据，XXXXXXXXXXXXXXXXXXXXXXXXXXXXX，由此可以得出以下结论：

1、

2、

3、

综上所述，本设计达到设计要求。

五、参考文献

5.参考文献

[1] 全国大学生电子设计竞赛组委会.2011 年全国大学生电子设计竞赛获奖作品选编[M].北京：北京理工大学出版社，2012

[2] 沈建华，杨艳琴，MSP430 超低功耗单片机原理与应用[M].北京：清华大学出版社，2013

[3] 胡寿松.自动控制原理[M].6 版.北京：科学出版社，2013

[4] 李志明.STM32 嵌入式系统开发实战指南[M].北京：机械工业出版社，2013

[5] 童诗白，华成英.模拟电子技术基础[M].4 版.北京：高等教育出版社，2009

[6] 张友德，赵志英，涂时亮.单片机微型机原理，应用与实践[M].5 版.上海:复旦大学出版社，2009

附录 1：电路原理图

附录 2：源程序

格式说明：《设计报告》为 A4 纸张 8 页以内，首页为 300 字内中文摘要，正文小四号宋体，行距固定为 22 磅，不得加页眉页脚。每页右下端注明页码。单页打印。

```
1. while (1) {
2.     // 控制走多少圈（条件未完整）
3.     if (follow_control.rote[1] >= follow_control.rote[0]) {
4.         Load(moto1: 0, moto2: 0);  // 停止电机
5.     }
6.
7.     // 巡线模式
8.     Follow_direct();
9.
10.    // 直角转弯处理
11.    if (follow_control.state == 2) {
12.        DL_TimerA_stopCounter(Pid_INST);  // 停止 PID 定时器
13.        Load(moto1: 1500, moto2: -1500);  // 差速转弯
14.
15.        // 等待巡线传感器检测到特定条件（如 3 个传感器触发）
16.        while (Follow_num >= 3) {
17.            Follow_direct();  // 继续巡线调整
18.        }
19.
20.        // 恢复巡线模式
21.        DL_TimerA_startCounter(Motor_INST);
22.        follow_control.state = 1;
23.    }
24.}
25.PID 中断服务函数
26.c
27.复制
28.void Pid_INST_IRQHandler(void) {
29.    // 位置环 PID 计算（目标值和当前值均为 0，可能为占位符）
30.    volatile float Followpid_output = pid_calculate(
31.        pid: &Follow,
32.        current_value: 0,
33.        target: 0
34.    );
35.
36.    // 根据 PID 输出调整电机速度
37.    Load(
```

```
        moto1: (int)(follow_control.base_speed + Followpid_output),
        moto2: (int)(follow_control.base_speed - Followpid_output)
    );
}
```

```
void task_point_follow(void) {
    // 定义增量式 PID 控制器
    volatile int16_t X_angle_error = (int16_t)(uart_rx_values[0]);
    // 获取 X 轴角度误差
    volatile int16_t Y_angle_error = (int16_t)(uart_rx_values[1]);
    // 获取 Y 轴角度误差

    // 计算 PID 输出值
    volatile float X_angle_error_output = PID_Calculate_DSP_WithError(&point, X_angle_error);  // X 轴
    volatile float Y_angle_error_output = PID_Calculate_DSP_WithError(&point, Y_angle_error);  // Y 轴

    // 调整舵机/步进电机角度
    x_angle += X_angle_error_output;  // 累加 X 轴角度
    y_angle += Y_angle_error_output;  // 累加 Y 轴角度

    // 限制角度范围（30° ~ 150°）
    x_angle = limit(x_angle, 30, 150);
    y_angle = limit(y_angle, 30, 150);

    // 设置步进电机角度
    StepMotor_SetAngle(&stepMotorA, x_angle);  // 电机 A 控制 X 轴
    StepMotor_SetAngle(&stepMotorB, y_angle);  // 电机 B 控制 Y 轴
}
```