基于超声波定位的自动跟随小车设计

指导老师：张士文老师　作者：张宇涵、于沛龙、落学富

2025 年 12 月 27 日

目录

1 实验目的

本实验基于数字电子技术、模拟电子技术与嵌入式系统基础知识，设计并实现基于超声波信号的自动跟随小车系统。通过自行搭建超声波发送与接收电路，并结合单片机进行信号处理与控制，实现小车对超声波声源的自动定位与跟随，若无超声波信号，则小车自动进入旋转搜索模式，等待捕获信号源。

通过本实验，可以加深对超声波传感技术、信号调理电路设计以及嵌入式控制系统开发的理解与应用能力。

2 系统总体方案设计

2.1 系统结构

系统结构主要分为三部分：超声波发送端、超声波接收端、电机驱动部分。

发送端通过 6.5V 电池组模块供电，由单片机信号驱动，产生频率 $40kHz$ 的超声波信号

接收端含电机驱动模块、接收器滤波整形电路以及 MCU 。接收器安装于小车前方两侧接收声波信号，通过滤波整形电路将正弦信号转换为 40kHz 的脉冲信号，输入到 MCU 的 ADC 引脚进行信号处理。

2.2 工作原理

由于发送端与接收端分离，系统无法直接获取发送时刻，因此采用双接收器时间差定位方法。通过比较左右接收器接收到超声波信号的时间差，判断声源相对于小车的方向，结合 PID 控制算法控制电机驱动模块，从而控制小车进行转向与跟随。

当系统检测不到有效超声信号时，小车进入搜索模式，原地旋转以重新寻找信号源。

3 硬件系统设计

3.1 超声波发送电路

超声波发射端用于产生稳定的高频超声信号，为接收端提供连续的声源参考。本系统采用独立供电的手持式超声波发送模块，与小车接收端相互分离，符合实验任务书对系统结构的要求。

实验中采用 MAX232 芯片构成的电平转换电路，将控制器输出的 PWM 信号转换为幅值更高的差分激励信号，从而有效驱动超声波发射器，提高超声波信号的发射强度和传播距离。

在工作过程中，控制器持续输出若干周期的 $40kHz$ 脉冲信号，超声波发射器在电激励作用下产生连续的超声波信号并向空间辐射。接收端通过检测该连续超声波信号的到达时间差，实现对声源方向的判断与跟随控制。

3.2 超声波接收滤波与整形电路

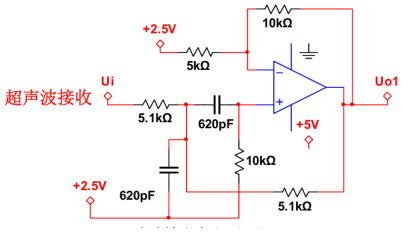

图 1: 滤波电路

该电路为单电源（+5 V）条件下的超声波接收前端放大电路，其主要功能包括：

- 对超声波接收信号进行交流耦合
- 抑制直流与低频噪声
- 在单电源供电下实现交流信号放大
- 通过 2.5 V 偏置实现虚拟地

需要注意的是，实际运用中该电路会产生自激振荡，故需确保超声波接受器在无信号时 Ui 引脚与地之间保持短接状态。

输出信号的频率等于超声波接收器所接收到的信号频率（通常为 40 kHz）。输入端由电阻 R 与电容 C 构成一阶 RC 高通滤波器，其截止频率为：

$$f_c = \frac{1}{2\pi RC} \tag{1}$$

代入参数：

$$R = 5.1\,\mathrm{k\Omega}, \quad C = 620\,\mathrm{pF}$$

$$f_c = \frac{1}{2\pi \times 5100 \times 620 \times 10^{-12}} \approx 5.0 \times 10^4\,\mathrm{Hz} \tag{2}$$

因此，该电路允许约 50 kHz 以上的信号通过，对低频信号具有明显抑制作用，适合用于 $40 \sim 60\,\mathrm{kHz}$ 的超声波接收应用。

由于小车为单电源供电，故采用如图 2 所示的 2.5V 稳压管提供 2.5V 的偏置电压，该电压作为运算放大器的参考电位，并通过 RC 网络进行滤波以提高稳定性。由于运算放大器采用单电源供电（$0 \sim 5\,\mathrm{V}$），无法直接处理负电压信号，因此引入 2.5 V 作为交流意义上的参考地（虚拟地）。设原始超声波信号为：

$$v_{\mathrm{ac}}(t) = \pm V_m$$

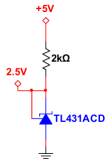

图 2: 2.5v 偏置电路

加入偏置后，运放输入信号变为：

$$v_{\text{in}}(t) = 2.5\,\text{V} \pm V_m \tag{3}$$

从而保证信号始终处于运放允许的输入范围内，避免削波失真。偏置后的输出信号形式为：

$$v_{\text{out}}(t) = 2.5\,\text{V} \pm A_v V_m \tag{4}$$

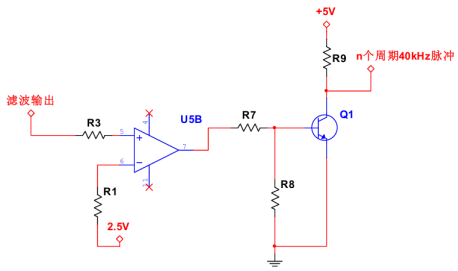

图 3: 放大电路

实际运用中发现 V_m 可达 1V 以上，利用图 3 中的单限比较器可实现方波信号的产生。由于滤波电路输出的偏置高于 2.5V, 并且由于 LM413 运放性能受限，上升沿到达最高前便下降，故滤波输出接入运放 N 端，确保晶体管在无信号时保持高电平，接收到信号时产生频率为 40Hz 的脉冲信号。

3.3 单片机与电机驱动电路

本系统选用 ESP32 单片机作为核心控制器，负责超声波信号的采集、时间差计算、控制算法运算以及电机驱动信号的输出。

在超声波信号采集方面，左右两个超声波接收电路输出经滤波与整形后的 $40kHz$ 数字脉冲信号，分别接入 ESP32 的外部中断引脚。单片机在检测到信号上升沿时记录当前时间，通过比较左右信号的到达时间差判断声源相对于小车的方向。

小车的运动控制采用两轮差速驱动方式。电机驱动电路中，每个电机由单路 PWM 信号控制，其方向端在硬件上固定为正转状态。单片机通过改变 PWM 信号的占空比调节电机转速，从而实现小车的速度控制与转向控制。

在具体控制策略上，系统将小车的运动分解为基础前进速度与转向修正量两部分。基础前进速度用于保证小车持续向前运动，转向修正量由控制算法计算得到，并以速度差分量的形式叠加至左右电机的驱动信号中。通过增大一侧电机转速、减小另一侧电机转速的方式实现转向控制。

当系统未检测到有效的超声波信号时，单片机控制左右电机以不同转速运行，使小车围绕单侧车轮缓慢旋转，从而进入搜索状态以重新捕获超声波信号。

4 软件系统设计

为提高系统的可读性、可维护性与调试效率，本系统的软件部分采用模块化与面向对象相结合的设计方法。

软件系统主要由以下功能模块构成，各模块之间通过清晰的接口进行通信与协作：

- 主控制模块：`main_controller`，负责系统状态管理与各功能模块的协调调度；
- 超声波检测模块：`ultrasonic_controller`，用于采集左右超声波信号并计算时间差；
- PID 控制模块：`pid_controller`，根据时间差计算转向控制量；
- 电机控制模块：`motor_controller`，根据控制指令输出电机驱动信号。

4.1 软件系统总体结构

系统程序以 Arduino 框架为基础，主函数仅负责系统初始化与循环调度。各功能模块在初始化阶段完成硬件配置，在主循环中由主控制模块统一调度执行。

4.2 主控制模块设计

主控制模块为系统的软件核心，负责协调各功能模块的运行，其主要功能包括：超声波信号状态判断、时间差数据获取、控制算法调用以及系统状态切换。系统采用简单的状态机结构，定义跟随状态 FOLLOWING 与搜索状态 SEARCHING 两种工作模式。

在主循环中，控制器首先判断超声波检测模块是否获得了一次完整的左右信号检测。当检测到有效信号时，系统进入跟随状态；若在一定时间内未检测到有效信号，则系统自动切换至搜索状态。

4.3 超声波时间差检测模块

超声波检测模块基于外部中断方式实现。左右两个超声波接收通道分别接入单片机的外部中断引脚，当检测到经整形后的超声波脉冲信号上升沿时，通过微秒级定时函数

记录当前时间。为避免多次触发干扰，每个接收通道在完成一次检测后暂时锁定，直至左右信号均被接收。

当左右接收器均完成一次有效触发后，系统计算左右信号的到达时间差，并将其作为后续方向控制的输入量。为增强系统的稳定性，对时间差数据设置限幅，避免异常信号对控制过程产生过大影响。

4.4 PID 方向控制算法

系统采用经典 PID 控制算法对小车的转向进行控制。控制目标为使左右超声波接收信号的到达时间差趋近于零，即小车正对超声波声源方向。PID 控制器以时间差作为反馈量，输出为转向控制量。

在实现中，PID 控制器采用位置式算法结构，并引入积分限幅与输出限幅机制，以防止积分饱和和控制输出过大。PID 参数通过实验整定获得，在实验运行速度范围内能够保证系统稳定运行。

4.5 电机控制与运动策略

电机控制模块采用两轮差速控制策略。系统将小车的运动分解为基础前进速度与转向修正量两部分，其中基础速度用于保证小车持续向前运动，转向修正量由 PID 控制器计算得到，并以速度差分的形式叠加至左右电机。

在驱动方式上，电机仅支持单方向转动，转速由单路 PWM 信号控制。通过提高一侧电机转速并降低另一侧电机转速，实现小车的转向控制。

为提高系统稳定性，电机 PWM 信号的更新频率受到时间间隔限制，避免因控制量快速变化导致电机频繁调整，从而提升整体运行平滑性。

4.6 搜索模式与状态切换机制

当系统在设定时间内未检测到有效超声波信号时，主控制模块判定信号丢失，系统自动进入搜索状态。在搜索状态下，小车通过单侧驱动方式缓慢转向，以扩大超声波信号的搜索范围。当重新检测到有效信号后，系统立即切换回跟随状态，恢复正常跟随控制。

5 实验结果与分析

实验结果表明，小车能够在一定距离范围内稳定跟随超声波发送端运动。当声源移出接收范围时，小车可自动进入搜索模式并重新捕获信号。

6 总结

本文设计并实现了一种基于超声波定位的自动跟随小车系统。实验结果表明，该系统能够较好地完成自动跟随与搜索任务，达到了实验预期目标。

致谢

在此向张老师和实验室的所有助教老师谨致以诚挚的感谢，也感谢实验室提供的良好实验条件和相关设备支持。

特别感谢张老师在实验方案设计、整体思路把握以及电气系统调试方面给予的指导与建议，使本实验在系统结构和实现路径上更加清晰合理。也感谢助教老师在电路搭建、调试方法以及电机驱动模块的使用等方面提供的细致而及时的帮助，有效解决了实验过程中遇到的多项实际问题，为实验的顺利完成提供了重要保障。

此外，感谢实验室提供的良好实验条件和相关设备支持。本次实验不仅加深了我们对超声波应用、电机控制及嵌入式系统设计的理解，也提升了综合分析与工程实践能力。在此一并表示衷心感谢。

A 小车程序源代码

A.1 main.cpp

Listing 1: 主程序 main.cpp

```
/*====================
* main.cpp
======================*/

#include <Arduino.h>
#include "main_controller.h"

main_controller mc;

void setup()
{
    Serial.begin(SERIAL_BAUDRATE);
    mc.init();
}

void loop()
{
    mc.update();
}
```

A.2 main_controller.h

Listing 2: 主控制器头文件 main_controller.h

```
/*====================
* main_controller.h
======================*/

#ifndef MAIN_CONTROLLER_H
#define MAIN_CONTROLLER_H

#include <Arduino.h>
#include "pid_controller.h"
#include "motor_controller.h"
#include "ultrasonic_controller.h"
```

```
12
13 class main_controller {
14 public:
15     main_controller();
16     bool init();
17     bool update();
18
19 private:
20     pid_controller* pid;
21     motor_controller* motor;
22     ultrasonic_controller* ultrasonic;
23
24     float dir_d;
25     float dir_t;
26     float dir_output;
27
28     unsigned long lastSignalTime;
29
30     enum State {
31         SEARCHING,
32         FOLLOWING
33     } state;
34 };
35
36 #endif
```

A.3 main_controller.cpp

Listing 3: 主控制器代码实现 main_controller.cpp

```
1  /*====================
2  * main_controller.cpp
3  ======================*/
4
5  #include "main_controller.h"
6
7  main_controller::main_controller()
8  {
9      pid = new pid_controller();
10     motor = new motor_controller();
```

```
    ultrasonic = new ultrasonic_controller();

    dir_d = PID_DESIRED_DATA;
    dir_t = 0.0f;
    dir_output = 0.0f;

    lastSignalTime = 0;
    state = SEARCHING;
}

bool main_controller::init()
{
    motor->init();
    ultrasonic->init();

    pid->init(&dir_d, &dir_output);
    pid->SetK(PID_KP, PID_KI, PID_KD);
    pid->SetL(PID_MAX_INTEGRAL, PID_MIN_OUTPUT, PID_MAX_OUTPUT);

    return true;
}

bool main_controller::update()
{
    unsigned long now = millis();

    ultrasonic->checkSilence();

    if (ultrasonic->available())
    {
        int16_t timeDiff = ultrasonic->getTimeDiff();

        double dirNorm = (double)timeDiff / 32767.0;
        dirNorm = constrain(dirNorm, -1.0, 1.0);

        if (fabs(dirNorm) < PID_DEAD_ZONE) dirNorm = 0.0;

        // 低通滤波
        dir_t = LPF_ALPHA * dirNorm + (1.0f - LPF_ALPHA) * dir_t;

```

```
        pid->Calc(dir_t);

        int turn_output = (int)(dir_output * MOTOR_MAX_TURN_SPEED);

        motor->setTurn(turn_output);

        lastSignalTime = now;
        state = FOLLOWING;
    }
    else
    {
        if (state == FOLLOWING && (now - lastSignalTime >
           SIGNAL_TIMEOUT_MS))
        {
            state = SEARCHING;
        }

        if (state == SEARCHING)
        {
            motor->search(SEARCH_TURN_SPEED);
        }

    }

    return true;
}
```

A.4 motor_controller.h

Listing 4: 电机控制模块头文件 motor_controller.h

```
/*====================
* motor_controller.h
======================*/

#ifndef MOTOR_CONTROLLER_H
#define MOTOR_CONTROLLER_H

#include <Arduino.h>
#include "config.h"
```

```
10
11 class motor_controller {
12 public:
13     motor_controller();
14     void init();
15
16     // 设置转向分量（PID 输出）
17     void setTurn(int turn);
18
19     // 搜索模式：原地旋转
20     void search(int speed);
21
22     void stop();
23
24 private:
25     uint8_t leftPin;
26     uint8_t rightPin;
27
28     int baseSpeed;
29     int turnSpeed;
30
31     unsigned long lastPwmUpdate;
32
33     void applyOutput();
34     void driveMotor(int pin, int speed);
35 };
36
37 #endif
```

A.5 motor_controller.cpp

Listing 5: 电机控制模块代码实现 motor_controller.cpp

```
1 /*====================
2 * motor_controller.cpp
3 ======================*/
4
5 #include "motor_controller.h"
6
7 motor_controller::motor_controller()
```

```
    : leftPin(MOTOR_LEFT_PIN),
      rightPin(MOTOR_RIGHT_PIN),
      baseSpeed(MOTOR_BASE_SPEED),
      turnSpeed(0),
      lastPwmUpdate(0)
{}

void motor_controller::init()
{
    pinMode(leftPin, OUTPUT);
    pinMode(rightPin, OUTPUT);

    // B 端在硬件上已拉高
    stop();
}

void motor_controller::setTurn(int turn)
{
    // turn 是“速度差分量”，不做反相
    turnSpeed = constrain(turn, -MOTOR_MAX_SPEED, MOTOR_MAX_SPEED);
    applyOutput();
}

void motor_controller::applyOutput()
{
    unsigned long now = millis();
    if (now - lastPwmUpdate < PWM_UPDATE_INTERVAL_MS) return;
    lastPwmUpdate = now;

    int leftSpeed  = baseSpeed + turnSpeed;
    int rightSpeed = baseSpeed - turnSpeed;

    leftSpeed  = constrain(leftSpeed,  0, MOTOR_MAX_SPEED);
    rightSpeed = constrain(rightSpeed, 0, MOTOR_MAX_SPEED);

    driveMotor(leftPin, leftSpeed);
    driveMotor(rightPin, rightSpeed);
}

void motor_controller::search(int speed)
```

```
{
    unsigned long now = millis();
    if (now - lastPwmUpdate < PWM_UPDATE_INTERVAL_MS) return;
    lastPwmUpdate = now;

    speed = constrain(speed, 0, MOTOR_MAX_SPEED);

    // 原地旋转：一侧快，一侧慢（而不是反向）
    driveMotor(leftPin,  speed);
    driveMotor(rightPin, 0);
}

void motor_controller::driveMotor(int pin, int speed)
{
    // speed: 0 ~ MAX_SPEED
    speed = constrain(speed, 0, MOTOR_MAX_SPEED);

    // 反相 PWM: speed 越大，LOW 占空比越大
    int pwmValue = MOTOR_MAX_SPEED - speed;

    analogWrite(pin, pwmValue);
}

void motor_controller::stop()
{
    // speed = 0 → pwm = MAX_SPEED → A 恒 HIGH → 停
    analogWrite(leftPin,  MOTOR_MAX_SPEED);
    analogWrite(rightPin, MOTOR_MAX_SPEED);
}
```

A.6 ultrasonic_controller.h

Listing 6: 超声波检测模块头文件 ultrasonic_controller.h

```
/*====================
* ultrasonic_controller.h
======================*/

#ifndef ULTRASONIC_CONTROLLER_H
#define ULTRASONIC_CONTROLLER_H
```

```
7
8  #include <Arduino.h>
9  #include "config.h"
10
11 class ultrasonic_controller
12 {
13 public:
14     ultrasonic_controller();
15     void init();
16
17     // 是否已经捕获到“一次完整且未消费的 timeDiff”
18     bool available();
19
20     // 读取本次事件的时间差（只会成功一次）
21     int16_t getTimeDiff();
22
23     void checkSilence();
24
25 private:
26     uint8_t leftPin;
27     uint8_t rightPin;
28
29     // 首次下降沿时间戳
30     volatile unsigned long leftFirstTime;
31     volatile unsigned long rightFirstTime;
32
33     // 是否已捕获首沿
34     volatile bool leftCaptured;
35     volatile bool rightCaptured;
36
37     // 本次超声事件是否已产出结果
38     volatile bool diffReady;
39
40     // 用于防止同一事件内重复触发
41     volatile bool eventLocked;
42
43     volatile unsigned long eventEndTime;   // 记录静默期结束的时间
44     volatile bool eventSilence;             // 控制静默期
45
46     static ultrasonic_controller* instance;
```

```
47
48      static void IRAM_ATTR leftISR();
49      static void IRAM_ATTR rightISR();
50  };
51
52  #endif
```

A.7 ultrasonic_controller.cpp

Listing 7: 超声波检测模块代码实现 ultrasonic_controller.cpp

```
1   /*====================
2   * ultrasonic_controller.cpp
3   ======================*/
4
5   #include "ultrasonic_controller.h"
6
7   ultrasonic_controller* ultrasonic_controller::instance = nullptr;
8
9   ultrasonic_controller::ultrasonic_controller()
10  {
11      leftPin  = ULTRASONIC_RX_LEFT_PIN;
12      rightPin = ULTRASONIC_RX_RIGHT_PIN;
13
14      leftCaptured  = false;
15      rightCaptured = false;
16      diffReady     = false;
17      eventLocked   = false;
18      eventSilence  = false;
19      eventEndTime  = 0;
20
21      instance = this;
22  }
23
24  void ultrasonic_controller::init()
25  {
26      pinMode(leftPin, INPUT);
27      pinMode(rightPin, INPUT);
28
```

```
29      attachInterrupt(digitalPinToInterrupt(leftPin),  leftISR,
            FALLING);
30      attachInterrupt(digitalPinToInterrupt(rightPin), rightISR,
            FALLING);
31  }
32
33  /*
34   * ISR 设计原则:
35   * - 一次超声事件内, 只允许记录“第一次下降沿”
36   * - 一旦 diffReady = true, 本事件后续所有下降沿全部忽略
37   */
38
39  void IRAM_ATTR ultrasonic_controller::leftISR()
40  {
41      if (!instance) return;
42      if (instance->eventLocked) return;
43
44      if (!instance->leftCaptured)
45      {
46          instance->leftFirstTime = micros();
47          instance->leftCaptured = true;
48
49          if (instance->rightCaptured)
50          {
51              instance->diffReady   = true;
52              instance->eventLocked = true;
53          }
54      }
55  }
56
57  void IRAM_ATTR ultrasonic_controller::rightISR()
58  {
59      if (!instance) return;
60      if (instance->eventLocked) return;
61
62      if (!instance->rightCaptured)
63      {
64          instance->rightFirstTime = micros();
65          instance->rightCaptured = true;
66
```

```
67          if (instance->leftCaptured)
68          {
69              instance->diffReady   = true;
70              instance->eventLocked = true;
71          }
72      }
73  }
74
75  bool ultrasonic_controller::available()
76  {
77      return diffReady && !eventSilence;
78  }
79
80  int16_t ultrasonic_controller::getTimeDiff()
81  {
82      if (!diffReady) return 0;
83
84      noInterrupts();
85      long diff = (long)leftFirstTime - (long)rightFirstTime;
86      leftCaptured  = false;
87      rightCaptured = false;
88      diffReady     = false;
89      eventLocked   = false;
90      interrupts();
91
92      // 限幅（安全）
93      if (diff > ULTRASONIC_MAX_TIME_DIFFERENCE) diff =
            ULTRASONIC_MAX_TIME_DIFFERENCE;
94      else if (diff < -ULTRASONIC_MAX_TIME_DIFFERENCE) diff = -
            ULTRASONIC_MAX_TIME_DIFFERENCE;
95
96      // 死区
97      if (abs(diff) <= ULTRASONIC_TIMEDIFF_DEADZONE) return 0;
98
99      eventSilence = true;
100     eventEndTime = millis() + ULTRASONIC_SILENCE_TIME;
101
102     // 将时间差映射到 Q15 整数范围
103     int32_t q15 = (diff * 32767L) / ULTRASONIC_MAX_TIME_DIFFERENCE;
104
```

```
105     return (int16_t)q15;
106 }
107
108 void ultrasonic_controller::checkSilence()
109 {
110     // 检查静默期是否结束
111     if (eventSilence && millis() >= eventEndTime)
112     {
113         eventSilence = false;  // 结束静默期，允许下一次事件
114     }
115 }
```

A.8 pid_controller.h

Listing 8: PID 控制器头文件 pid_controller.h

```
1  /*====================
2  * pid_controller.h
3  ======================*/
4
5  #ifndef PID_CONTROLLER_H
6  #define PID_CONTROLLER_H
7
8  typedef struct pid_data_ pid_data;
9  struct pid_data_
10 {
11     float e[3]; // 误差: 比例误差、积分误差、微分误差
12     float k[3]; // 系数: 比例系数、积分系数、微分系数
13     float l[3]; // 限幅: 积分限幅、输出最小、输出最大
14 };
15
16 class pid_controller
17 {
18 private:
19     pid_data* obs; // PID数据集成
20     float error;   // 误差值
21     float data_;   // 原始PID输出
22     float* data_o; // 输出数据
23     float* data_d; // 预期数据
24
```

```
25 public:
26     /// @brief 构造函数
27     pid_controller();
28 
29     /// @brief 初始化PID控制器
30     /// @param data_d 期望的数据（比如小车的位置）
31     /// @param data_o 实际输出的数据指令（比如小车电机的扭矩）
32     bool init(float* data_d, float* data_o);
33 
34     /// @brief PID计算函数，通过当前获取的数据计算应该输出的数据，并
           存储在指针中
35     /// @param data_t 当前传感器获取的数据，对应data_d，PID需要让
           data_t不断接近data_d
36     bool Calc(float data_t);
37 
38     /// @brief 设置PID控制的系数
39     /// @param kp 比例系数
40     /// @param ki 积分系数
41     /// @param kd 微分系数
42     bool SetK(float kp, float ki, float kd);
43 
44     /// @brief 设置限幅
45     /// @param l_int 积分限幅
46     /// @param l_min 输出最小限幅
47     /// @param l_max 输出最大限幅
48     bool SetL(float l_int, float l_min, float l_max);
49 };
50 
51 #endif
```

A.9 pid_controller.cpp

Listing 9: PID 控制器代码实现 pid_controller.cpp

```
1 /*====================
2 * pid_controller.cpp
3 ======================*/
4 
5 #include "pid_controller.h"
6 
```

```
7  pid_controller::pid_controller()
8  {
9      this->obs = new pid_data();
10 }
11
12 bool pid_controller::init(float* data_d, float* data_o)
13 {
14     this->data_d = data_d;
15     this->data_o = data_o;
16
17     // 初始化误差和系数
18     obs->e[0] = 0.0f;
19     obs->e[1] = 0.0f;
20     obs->e[2] = 0.0f;
21
22     obs->k[0] = 0.0f; // kp
23     obs->k[1] = 0.0f; // ki
24     obs->k[2] = 0.0f; // kd
25
26     obs->l[0] = 0.0f; // 积分限幅
27     obs->l[1] = 0.0f; // 输出最小
28     obs->l[2] = 0.0f; // 输出最大
29
30     return true;
31 }
32
33 bool pid_controller::Calc(float data_t)
34 {
35     error = *data_d - data_t;       // 计算误差
36     obs->e[2] = error - obs->e[0]; // 离散微分
37     obs->e[1] += error;             // 积分
38     obs->e[0] = error;              // 更新误差
39
40     // 积分限幅
41     if (obs->e[1] > obs->l[0])
42         obs->e[1] = obs->l[0];
43     else if (obs->e[1] < -obs->l[0])
44         obs->e[1] = -obs->l[0];
45
46     // 临时数据容器，计算PID原始值
```

```
    data_ = obs->k[0] * obs->e[0] +
            obs->k[1] * obs->e[1] +
            obs->k[2] * obs->e[2];

    // 输出限幅
    if (data_ > obs->l[2])
        *data_o = obs->l[2];
    else if (data_ < obs->l[1])
        *data_o = obs->l[1];
    else
        *data_o = data_;

    return true;
}

bool pid_controller::SetK(float kp, float ki, float kd)
{
    this->obs->k[0] = kp;
    this->obs->k[1] = ki;
    this->obs->k[2] = kd;

    return true;
}

bool pid_controller::SetL(float l_int, float l_min, float l_max)
{
    this->obs->l[0] = l_int;
    this->obs->l[1] = l_min;
    this->obs->l[2] = l_max;

    return true;
}
```

A.10 config.h

Listing 10: 系统参数配置文件 config.h

```
/*====================
* config.h
======================*/
```

```
4
5  #ifndef CONFIG_H
6  #define CONFIG_H
7
8  /*************************************************************
9   * 1. 调试配置
10  *************************************************************/
11 #define SERIAL_BAUDRATE         115200
12
13 /*************************************************************
14  * 2. 超声波控制器参数
15  *************************************************************/
16 #define ULTRASONIC_RX_LEFT_PIN   34
17 #define ULTRASONIC_RX_RIGHT_PIN  35
18
19 #define ULTRASONIC_MAX_TIME_DIFFERENCE     500  // us, 异常时间差限幅
20 #define ULTRASONIC_TIMEDIFF_DEADZONE       10   // us, 时间差死区
21 #define ULTRASONIC_SILENCE_TIME            18   // us, 静默时间
22
23 /*************************************************************
24  * 3. 电机控制参数
25  *************************************************************/
26 #define MOTOR_LEFT_PIN          4
27 #define MOTOR_RIGHT_PIN         17
28
29 #define MOTOR_MAX_SPEED         255     // PWM 最大值
30 #define MOTOR_BASE_SPEED        127     // 直行基础速度
31 #define MOTOR_MAX_TURN_SPEED    127     // 转弯速度
32
33 #define PWM_UPDATE_INTERVAL_MS  10      // 电机 PWM 更新间隔
34
35 /*************************************************************
36  * 4. 行为状态机参数
37  *************************************************************/
38 #define SIGNAL_TIMEOUT_MS       100     // 判定信号丢失时间
39 #define SEARCH_TURN_SPEED       80      // 搜索时原地旋转速度
40
41 /*************************************************************
42  * 5. PID 控制参数（方向）
43  *************************************************************/
```

```
#define PID_KP                  2.0
#define PID_KI                  0.01
#define PID_KD                  0.05

#define PID_MAX_INTEGRAL        1.0
#define PID_MIN_OUTPUT         -1.0
#define PID_MAX_OUTPUT          1.0

#define PID_DEAD_ZONE           0.002    // 输出死区

#define PID_DESIRED_DATA        0.0    // 目标时间差（正对声源）

/*************************************************************
 * 6. 低通滤波参数
 *************************************************************/
#define LPF_ALPHA               0.6    // 低通滤波系数（0.0 - 1.0）

#endif // CONFIG_H
```

B 超声波发生器程序源代码

Listing 11: 主程序 main.cpp

```
#include <Arduino.h>
#include "driver/mcpwm.h"

#define ULTRASONIC_PWM_GPIO     4
#define ULTRASONIC_FREQ_HZ      40000

#define BURST_ON_MS             10
#define BURST_OFF_MS            20

static bool burstOn = false;
static unsigned long lastSwitchTime = 0;

void setup()
{
    Serial.begin(115200);

```

```
    // 1. 绑定 GPIO 到 MCPWM
    mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, ULTRASONIC_PWM_GPIO);

    // 2. 配置 PWM
    mcpwm_config_t pwm_config = {
        .frequency = ULTRASONIC_FREQ_HZ,
        .cmpr_a = 0.0,              // 初始不发射
        .cmpr_b = 0.0,
        .duty_mode = MCPWM_DUTY_MODE_0,
        .counter_mode = MCPWM_UP_COUNTER,
    };

    mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);
    mcpwm_set_duty_type(
        MCPWM_UNIT_0,
        MCPWM_TIMER_0,
        MCPWM_OPR_A,
        MCPWM_DUTY_MODE_0
    );

    lastSwitchTime = millis();
}

void loop()
{
    unsigned long now = millis();

    if (burstOn)
    {
        // 正在发射
        if (now - lastSwitchTime >= BURST_ON_MS)
        {
            // 关闭 PWM（静默）
            mcpwm_set_duty(
                MCPWM_UNIT_0,
                MCPWM_TIMER_0,
                MCPWM_OPR_A,
                0.0
            );

```

```
57                 burstOn = false;
58                 lastSwitchTime = now;
59             }
60         }
61         else
62         {
63             // 静默状态
64             if (now - lastSwitchTime >= BURST_OFF_MS)
65             {
66                 // 开启 40kHz PWM
67                 mcpwm_set_duty(
68                     MCPWM_UNIT_0,
69                     MCPWM_TIMER_0,
70                     MCPWM_OPR_A,
71                     50.0
72                 );
73 
74                 burstOn = true;
75                 lastSwitchTime = now;
76             }
77         }
78 }
```