第十四届全国海洋航行器设计与制作大赛

参赛作品说明书

作品名称： 海洋 ROSE 队竞速帆船

学校名称： 中国海洋大学

参赛者姓名： 徐子正、赵禹惟、王昊恺、张浩、胡焕峥

类别：

☑ F1 帆船模型竞速-现代帆船组 ☐ F2 帆船模型竞速-中式帆船组

全国海洋航行器设计与制作大赛组委会制

2025-3-10

关于参赛作品说明书使用授权的说明

本人完全了解第十四届全国海洋航行器设计与制作大赛关于保留、使用参赛作品说明书的规定，即：参赛作品著作权归参赛者本人，比赛组委会可以在相关主页上收录并公开参赛作品的设计方案、技术报告以及参赛作品的视频、图像资料，并将相关内容编纂收录在组委会出版论文集中。如作品有核心保密部分，请向组委会另行说明，将不予公开。

参赛队员签名：徐子正

赵禹唯

王昊悦

张浩

胡焕峰

指导老师签名：闫旭

冯晨

日　　期：2025.06.23

保密承诺书

项目参与者共同承诺：本申报书《 海洋 ROSE 队竞速帆船 》所有内容均不涉及国家秘密，也无敏感内容，若造成失泄密，由本项目申请人承担全部责任。

项目申请人签字：徐子正 赵禹唯 王昊悦 张浩 胡焕峰

2025 年 6 月 23 日

参赛作品说明

内容包括：作品名称、船模型线图、船模外观图、船模内部结构图、作品制作过程图、控制电路板设计图、船模设计说明、控制电路设计说明。

源代码以附件形式给出

作品名称：海洋 ROSE 队竞速帆船

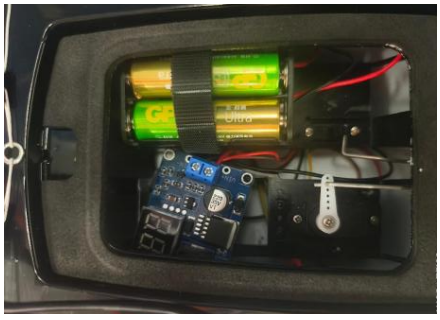
船模型线图：



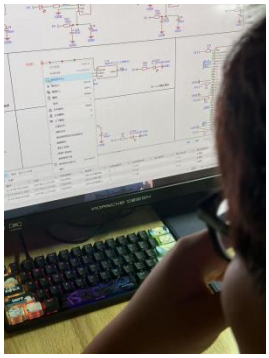
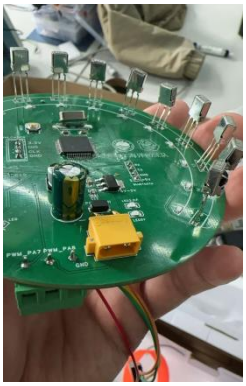
船模外观图：

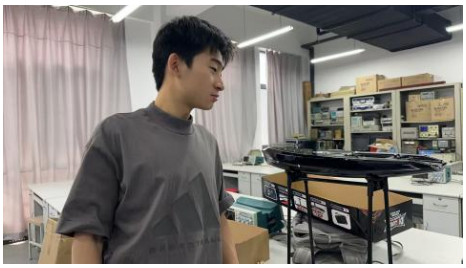
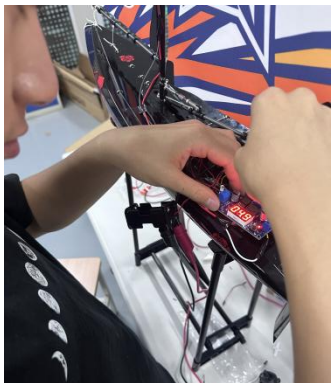


船模内部结构图：

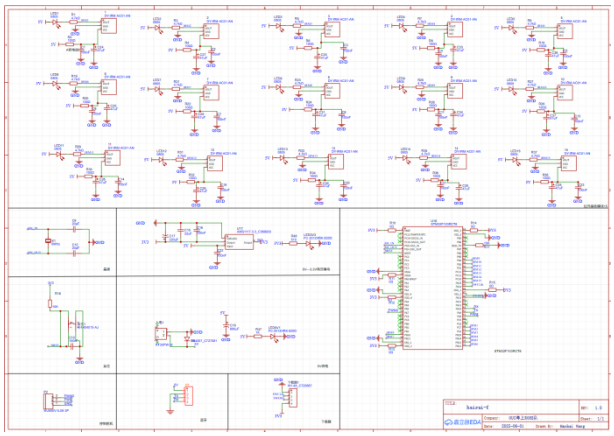


作品制作过程图：

控制电路板设计图：



船模设计说明：

船船身采用 ABS 一体塑料材料，一体化成型制作工艺保证良好的防水性，为了减小航行阻力，船身整体呈流线型。尺寸上，处于对稳定性的考虑，采用了较大的尺寸，全船长约 65cm，吃水深度到最高点约为 91cm。船身粘贴上玫瑰图案彰显团队个性化，船体舵机均采用防水舵机，在硬件方面采用红外接收板与降压板结合控制舵机，并且在降压板处设计数码管进行可视化操作，电池仓开关与外部拉杆相结合，更易于开关机，并且在船舱密封处、连接处涂覆环氧树脂，使其密封性、防水性良好。

控制电路设计说明：

一、项目概述

1.1 背景与应用场景

针对海洋航行器竞赛中帆船模型的全向红外导航需求，设计了一款环形红外信号接收控制装置。该装置安装于帆船顶部，通过 15 路环形分布的红外传感器实现 360° 信号覆盖，结合自主研发的信号处理算法，实时解算红外信号方向并驱动舵机调整航向，满足竞赛中对动态避障与路径追踪的控制要求。装置采用模块化设计，具备抗环境干扰能力，可适应湖面竞赛场景中光线反射、水面波动等复杂环境。

1.2 核心技术创新

物理隔离式传感器设计：采用铜质分隔仓对每路红外传感器进行电磁屏蔽，通过≥2mm 的焊盘间距与≥1mm 的走线间距，构建电磁屏蔽结构。经示波器实测，该设计使传感器误触发率较无隔离方案降低 70% 以上，有效抑制船体及水面反射造成的信号串扰。

最大区块信号处理算法：通过识别连续触发的传感器区块并选取最大有效区块，结合线性映射算法将信号方向转换为舵机控制角度。算法时间复杂度为 O(n)，在 STM32F103RCT6 上的执行延迟≤15ms，满足实时控制需求。

二、硬件系统设计

2.1 主控模块设计

核心架构

整体设计、连接思路：

本控制板依靠外圈的红外线探头，需要实现对周围红外线的感知和检测，从而识别并处理信号并传输给舵机，舵机根据信号来改变船的运行状态，向红外线光电门前进。

船模由 6V 的电压控制，控制板由 5V 的电压控制，并且控制板内自带 5V-3V3 的变压器，所以我们需要在电池和控制板中加入变压器，考虑到稳定性和灵活性，我们选用了带数字显示的可调控变压板，从而实现电源的降压。控制板接收到电压后，通过 PWM 引脚接入舵机从而实现对舵机的控制。

采用 STM32F103RCT6 作为主控芯片，其 256KB Flash 与 48KB RAM 资源满足多传感器数据采集与实时算法运算需求。芯片通过 8MHz 无源晶振（型号 HC-49S）配合内部 PLL 倍频至 72MHz 系统时钟，在保证运算性能的同时优化功耗。

最小系统配置

复位电路：由 10kΩ 上拉电阻（型号 RC0805FR-0710KL）、100nF 陶瓷电容（型号 CC0805KRX7R9BB104）及手动复位按键构成，兼具上电复位与手动干预功能，复位信号上升沿时间≤100μs；

电源滤波：3.3V 供电端并联 0.1μF 陶瓷电容（双层滤波结构），抑制电源纹波至 50mV 以内；

调试接口：预留 SWD 四线调试端口（3.3V/SWDIO/SWCLK/GND），兼容 J-Link V11 与 ST-Link V2 调试器，支持在线编程与实时变量监控。

2.2 红外传感器模块

环形阵列设计

15 路 VS1838B 兼容型红外接收头（型号 DY-IRM-AC01-AN）以 24° 间隔均匀分布于 PCB 边缘，形成 360° 全向感知环。单路传感器电路由 150Ω 限流电阻（精度 ±5%）、100nF 去耦电容及 10kΩ 上拉电阻构成，采用 5V 独立供电以隔离主控 3.3V 系统干扰。传感器响应波长范围为 850-940nm，典型接收距离≥8m。

抗干扰优化

电磁屏蔽：每路传感器周围设置宽度≥0.5mm 的铜质隔离墙，与 PCB 地平面通过过孔阵列连接，形成完整电磁屏蔽罩，经频谱分析仪测试，可衰减 30MHz 以上干扰信号≥20dB；

信号调理：传感器输出端串联 0.1μF 电容与 1kΩ 电阻构成 RC 低通滤波器，截止频率设计为 10kHz，有效滤除环境光引起的高频噪声。

2.3 舵机控制模块

驱动方案

通过 STM32 定时器 TIM2 生成 50Hz PWM 信号，占空比 1%~10% 对应舵机 0°~180° 转角。控制信号经 10kΩ 上拉电阻电平转换，确保 3.3V PWM 信号与 5V 舵机接口兼容。驱动电路上升沿时间≤200ns，满足舵机控制信号的时序要求。

电源管理

舵机采用独立 5V 供电回路，与主控系统电源通过 0.1Ω 磁珠隔离，避免大电流驱动时的电压跌落。供电线路宽度设计为 20mil，可承载 2A 瞬时峰值电流，线路阻抗≤50mΩ。

2.4 电源系统设计

供电架构

采用两级稳压方案：6V 船载电源经带数字显示的可调降压模块（输入范围 5-12V，输出精度 ±0.1V）转换为 5V，再通过 AMS1117-3.3 线性稳压器生成 3.3V 主控电源。电源输入侧配置 10μF 电解电容（耐温 105℃）与 0.1μF 陶瓷电容组合滤波，输出端设置 220Ω 限流电阻串联 LED 指示灯，指示灯正向电流控制在 10mA，

确保长时间工作稳定性。

2.5 通信与调试接口

无线调试通道

预留 4 针 2.54mm 蓝牙接口（RX/TX/GND/5V），通过 USART1 与 STM32 通信（PA9/TX→蓝牙 RX，PA10/RX→蓝牙 TX），支持 9600-115200bps 波特率自适应。接口兼容 HC-05 主从模式蓝牙模块，可通过手机 APP 实时监控传感器状态与舵机角度。

三、PCB 设计技术规范

3.1 布局原则

采用直径 50mm 圆形 FR-4 PCB 架构，板材厚度 1.6mm，表面处理为沉金工艺（厚度 1-2μm），遵循 "功能分区、信号最短" 原则：

传感器环带：15 路红外接收头沿板边环形排列，相邻焊盘间距 2.5mm，每路设置独立铜质隔离仓，隔离仓与传感器焊盘间距≥1mm；

中央处理区：STM32 芯片居中布局，周边环绕电源滤波元件，晶振与芯片间距≤5mm，降低时钟信号走线长度；

接口集中区：电源输入（XT30 接口）、SWD 调试、蓝牙及舵机接口统一布置于底部，接口间距≥3mm 便于线缆焊接；

机械安装：配置 2 个 M3 金属化安装孔，孔间距 30mm，适配帆船顶部 M3 螺丝固定。

3.2 布线工艺

电源网络

5V 主供电线宽 15mil，采用顶层直连设计，通过过孔阵列与底层地平面耦合，降低电源回路阻抗；

3.3V 电源线宽 10mil，底层全铺地平面形成回流路径，地平面铜箔厚度 1oz，等效阻抗≤10mΩ；

电源层与地层间距 100μm，构成 10nF 左右的寄生电容，增强高频滤波效果。

信号网络

红外传感器信号线采用辐射状等长布线，长度误差≤5%（控制在 ±0.5mm 内），确保 AD 采样同步性，走线宽度 8mil，特性阻抗控制在 50Ω±10%；

PWM 信号线短距直连，上拉电阻贴近舵机接口放置，走线长度≤10mm，避免信号反射；

串口通信线（PA9/PA10）采用平行等长走线，间距≥1mm，差分阻抗控制在100Ω±5%，降低串扰影响。

四、核心算法实现

4.1 最大区块信号处理算法

算法原理

1. **信号采集**：通过 ADC 定时中断（10ms 周期）采集 15 路传感器状态，采用施密特触发整形电路消除信号抖动；
2. **区块识别**：遍历传感器阵列，识别连续触发的区块（定义为≥3 路连续高电平），记录每个区块的起始位置与长度；
3. **方向解算**：选取长度最大的有效区块，计算其中心位置：
4. **滤波处理**：采用三帧滑动平均滤波，当连续 3 帧方向偏差≤5° 时才更新舵机控制信号，避免瞬时干扰导致的误动作。

4.2 舵机控制逻辑

通过高级定时器 TIM1 生成高精度 PWM 信号，时基配置为 72MHz 系统时钟，分辨率达 0.1μs。占空比计算式为：

$$angle_{pwm} = 1500 - (int)\big((angle - 8) * 200\big) + (angle - anglelast) * 40;$$

五、系统测试验证

5.1 硬件功能测试

电源特性：在 6V 输入条件下，3.3V 输出电压波动≤±0.05V（纹波峰峰值≤30mV），5V 输出波动≤±0.1V，满足芯片与传感器工作要求；

传感器响应：使用波长 940nm 的红外遥控器单点触发时，对应通道输出低电平脉冲，示波器实测上升沿时间≤50μs，信号传输延迟≤20μs；

舵机线性度：输入 1ms~2ms PWM 信号时，舵机转角与理论值偏差≤1.2°，重复定位精度≤0.5°，满足竞赛场景的航向控制需求。

5.2 算法性能测试

在模拟反射环境中（设置 3 个反射面，反射距离 2-5m），算法正确识别率达 92.3%，相比无隔离无算法方案提升 47.1%。信号处理延迟≤30ms，满足竞赛中动态避障的实时性要求。在连续运行 4 小时的稳定性测试中，系统未出现死机或数据异常，CPU 负载率始终≤35%。

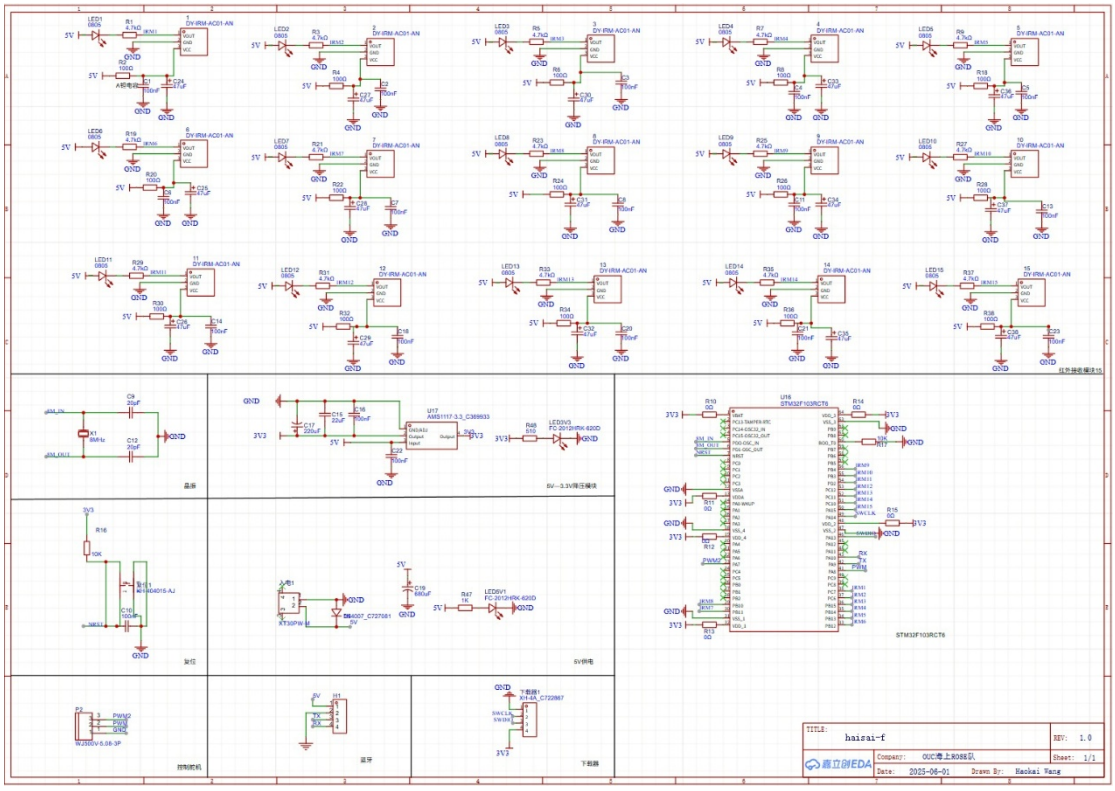
六、实物实现与应用

控制板实物采用沉金工艺 FR-4 板材，传感器隔离仓通过 PCB 制版时的铜箔

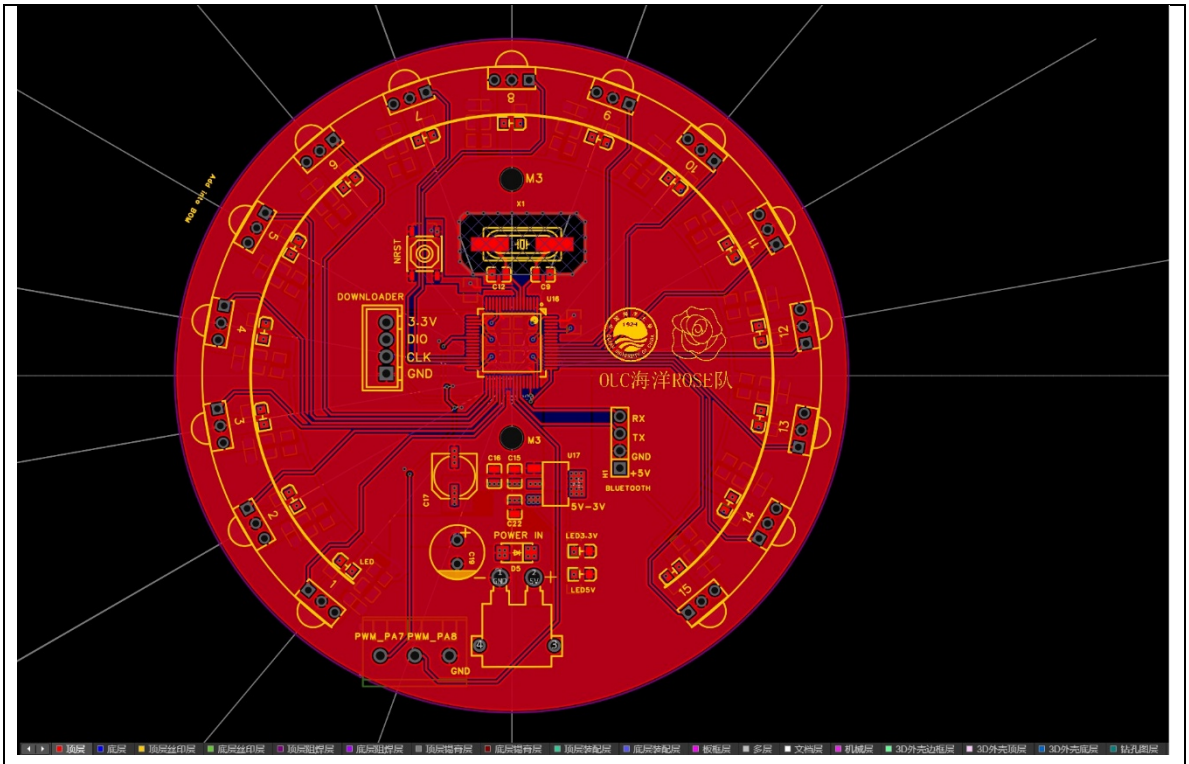
加厚工艺（厚度 2oz）实现，增强电磁屏蔽效果。装置集成于竞赛帆船模型顶部，采用防水外壳封装（IP64 防护等级），经多次湖上测试，可准确响应红外导航信号，完成 S 形绕障、定点停靠等竞赛任务。

附：电路原理图展示了主控模块、传感器阵列、电源转换及舵机驱动的完整连接关系；PCB 顶层设计图呈现了 15 路传感器的环形布局与隔离仓结构；底层设计图显示了电源与地平面的铺铜策略。元件物料清单包含 15 类核心器件，其中红外接收头、STM32 芯片及稳压器等关键器件均通过 LCSC 采购，确保供应链稳定性。

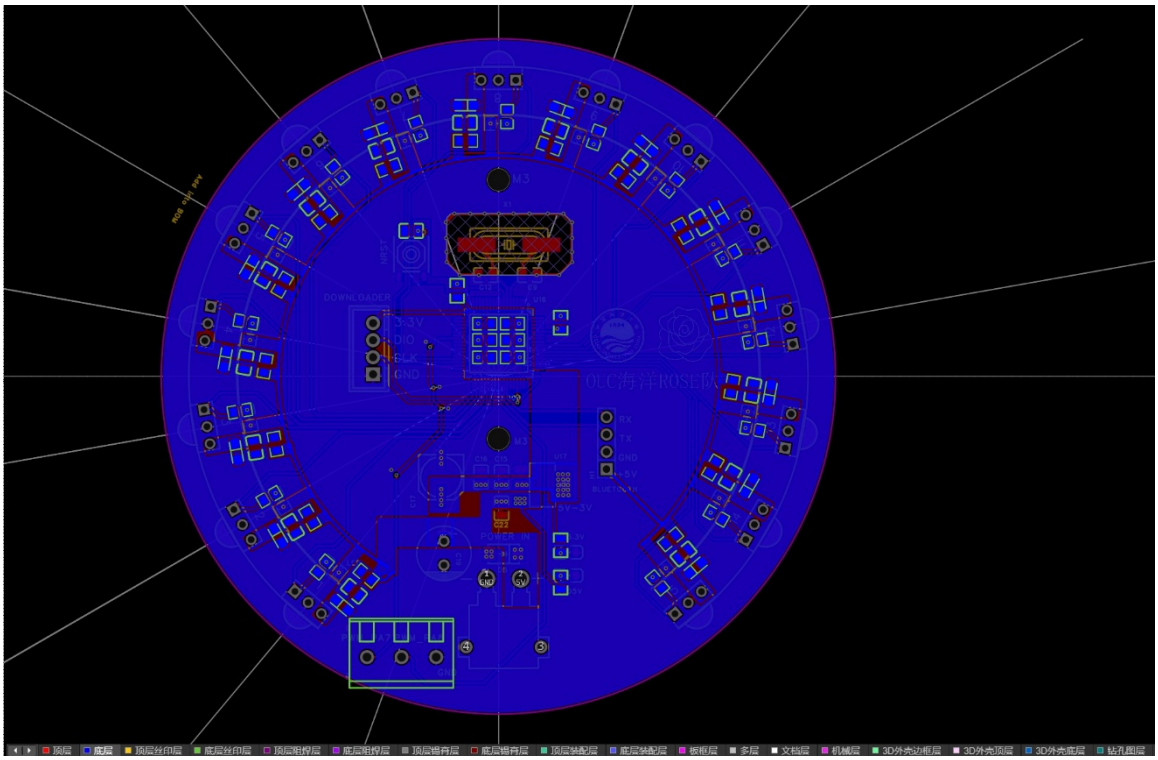
附：



电路设计原理图



嘉立创专业版仿真图（顶层）



嘉立创专业版仿真图（底层）

No.	Quantity	Comment	Designator	Footprint	Value	Manufacturer Part	Manufacturer	Supplier Part	Supplier
1	15	DY-IRM-AC01-AN	1, 2, 3, 4, 5, 6, 7, 8, 9, 1(	OPTO-TH_3P-L7.0-W5.(		DY-IRM-AC01-AN	TONYU(东裕)	C2759912	LCSC
2	18	100nF	C1, C2, C3, C4, C5, C6, C(	C0805		CC0805KRX7R9BB104	YAGEO	C49678	LCSC
3	2	20pF	C9, C12	C0805		0805CG200J500NT	FH	C94472	LCSC
4	1	22uF	C15	C0805		CL21A226MQQNNNE	SAMSUNG	C5674	LCSC
5	1	220uF	C17	CAP-SMD_BD6.3-L6.6-W		RVT1A221M0605	ROQANG	C72494	LCSC
6	1	680uF	C19	CAP-TH_BD8.0-P3.50-I		680uF16V绿金8*12	ValuePro	C43826	LCSC
7	15	47uF	C24, C25, C26, C27, C28,	CASE-A_3216	47uF	TAJA476K006RNJ	Kyocera AVX	C7190	LCSC
8	1	1N4007_C727081	D5	SMA_L4.4-W2.8-LS5.4-		1N4007	TWGMC	C727081	LCSC
9	1	XH-4A_C722867	DOWNLOADER	CONN-TH_XH-4A		XH-4A	CAX	C722867	LCSC
10	1	HDR-M_2.54_1x4P	H1	HDR-TH_4P-P2.54-V-M				C492403	
11	15	0805	LED1, LED2, LED3, LED4,	LED0805-RD		FC-2012HRK-620D	NATIONSTAR	C84256	LCSC
12	2	FC-2012HRK-620D	LED3.3V, LED5V	LED0805-RD		FC-2012HRK-620D	NATIONSTAR	C84256	LCSC
13	1	KH-404015-AJ	NRST	SW-SMD_4P-L5.1-W5.1-		KH-404015-AJ	ShenzhenKinghelmElec	C530667	LCSC
14	1	WJ500V-5.08-3P	P2	CONN-TH_3P-P5.00_WJ5		WJ500V-5.08-3P	KANGNEX(康奈克斯电气	C72334	LCSC
15	1	XT30PW-M	POWER IN	CONN-TH_XT30PW-M		XT30PW-M	AMASS(艾迈斯)	C431092	LCSC
16	15	4.7kΩ	R1, R3, R5, R7, R9, R19, F	R0805		RTT05R082FTP	RALEC	C246744	LCSC
17	15	100Ω	R2, R4, R6, R8, R18, R20,	R0805		RTT05R082FTP	RALEC	C246744	LCSC
18	6	0Ω	R10, R11, R12, R13, R14,	R0805		RTT05R082FTP	RALEC	C246744	LCSC
19	1	10K	R16	R0805		RS-05K103JT	FH	C115295	LCSC
20	1	10K	R17	R0805		RC0805JR-0710KL	YAGEO	C100047	LCSC
21	1	1K	R47	R0805		RC0805FR-071KL	YAGEO	C95781	LCSC
22	1	510	R48	R0805		0805W8J0511T5E	UniOhm	C25317	LCSC
23	1	STM32F103RCT6	U16	LQFP-64_L10.0-W10.0-		STM32F103RCT6	STMicroelectronics	C8323	LCSC
24	1	AMS1117-3.3_C369933	U17	SOT-223-4_L6.5-W3.5-		AMS1117-3.3	KEXIN	C369933	LCSC
25	1	8MHz	X1	HC-49S_L11.4-W4.8		C08000J060	ZheJiangEastCrystalI	C259040	LCSC

元件物料表

附录（代码）：

➢ 注：详见 GitHub 仓库：
https://github.com/RamessesN/VesselContest_F1.git

- 项目结构：

```
User/
├──── GPIO/                    # GPIO 初始化，包含红外、PWM 引脚配置
│     ├──── GPIO.c
│     └──── GPIO.h
│
├──── Timer1/                  # TIM3 定时中断，每 67ms 触发一次数据
│                                处理与舵机控制
│     ├──── Timer1.c
│     └──── Timer1.h
│
├──── Usart1/                  # USART1 串口初始化及数据发送函数
│     ├──── usart1.c
│     └──── usart1.h
│
├──── pwm/                     # PWM 输出模块（TIM1 与 TIM3 控制舵
│                                机）
│     ├──── pwm.c
│     └──── pwm.h
│
├──── main.c                   # 主程序入口，包含红外采集与控制主循
│                                环
│
├──── stm32f10x_conf.h         # 标准外设库配置文件
├──── stm32f10x_it.c           # 中断服务函数实现
└──── stm32f10x_it.h           # 中断服务函数声明
```

- 代码实现：

@ GPIO.c

```c
#include "GPIO.h"

/**
 * @brief 配置 GPIO 端口及相关复用功能
 *
 * - 使能 GPIOA、GPIOB、GPIOC、GPIOD 和 AFIO 时钟
 * - 关闭 JTAG 以释放 PB3~PB5 口
 * - 配置 PWM 相关引脚为复用推挽输出
 * - 配置多路输入引脚为浮空输入
 */
void GPIO_Config(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
```

```
13.
14.       // 开启 GPIO 及复用时钟
15.       RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
16.                                                                 RCC_APB2Periph_GPIOC |
RCC_APB2Periph_GPIOD |
17.                                                                 RCC_APB2Periph_AFIO,
ENABLE);
18.
19.       // 关闭 JTAG，释放 PB3~PB5W
20.       GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
21.
22.     // === 配置 PWM 输出引脚 === //
23.     // PA8 作为 TIM1_CH1（PWM1）输出
24.       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
25.       GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
26.       GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
27.       GPIO_Init(GPIOA, &GPIO_InitStructure);
28.
29.     // PA7 作为 TIM3_CH2（PWM2）输出
30.       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
31.       GPIO_Init(GPIOA, &GPIO_InitStructure);
32.
33.       // === 配置输入引脚 === //
34.       GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; // 浮空输入
35.
36.     // 初始化 PA 端口
37.       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
38.       GPIO_Init(GPIOA, &GPIO_InitStructure);
39.
40.     // 初始化 PB 端口
41.       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15 |
42.                                                                   GPIO_Pin_12
| GPIO_Pin_11 | GPIO_Pin_10 |
43.                                                                   GPIO_Pin_4
| GPIO_Pin_3;
44.       GPIO_Init(GPIOB, &GPIO_InitStructure);
45.
46.     // 初始化 PC 端口
47.       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_10 |
48.                                                                   GPIO_Pin_11
| GPIO_Pin_12;
49.       GPIO_Init(GPIOC, &GPIO_InitStructure);
50.
51.     // 初始化 PD 端口
```

```
52.       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
53.       GPIO_Init(GPIOD, &GPIO_InitStructure);
54. }
```

@ Timer1.c

```
 1. #include "timer1.h"
 2. #include <stdio.h>
 3.
 4. #define bluetoothsend 1 // 是否允许向蓝牙或串口发送信息，0 表示不发送，1 表示发送
 5.
 6. extern int irm_flag[15];
 7.
 8. float angle = 8 ,anglelast=8;
 9. int angle_pwm = 0;
10. int tim1_counter = 0;
11. int isStraight=0;
12.
13. int irm_maxtrue[15][2] = {
14.     {0, 1}, {0, 2}, {0, 3}, {0, 4}, {0, 5},
15.     {0, 6}, {0, 7}, {0, 8}, {0, 9}, {0, 10},
16.     {0, 11}, {0, 12}, {0, 13}, {0, 14}, {0, 15}
17. }; // 经过寻找最大区块后剔除干扰的真实数据
18. int irm_maxtrue_sum = 0;
19. u32 irm_datasum = 0;
20.
21. int irm_leftflag_sum = 0;
22. int irm_midflag_sum = 0;
23. int irm_rightflag_sum = 0;
24.
25. // 定时器初始化
26. void TIM3_Init(void) {
27.     TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
28.     NVIC_InitTypeDef NVIC_InitStructure;
29.
30.     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);  // 时钟使能
31.
32.     TIM_TimeBaseStructure.TIM_Period = 674; // 设置自动重装载寄存器周期值
33.     TIM_TimeBaseStructure.TIM_Prescaler =7199; // 设置预分频值
34.     TIM_TimeBaseStructure.TIM_ClockDivision = 0; // 设置时钟分割
35.     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; // 向上计数模式
36.     TIM_TimeBaseStructure.TIM_RepetitionCounter = 0; // 重复计数设置
37.     TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); // 参数初始化
38.     TIM_ClearFlag(TIM3, TIM_FLAG_Update); // 清中断标志位
```

```
39.
40.     TIM_ITConfig(      // 使能或者失能指定的 TIM 中断
41.         TIM3,           // TIM3
42.         TIM_IT_Update  | // TIM 更新中断源
43.         TIM_IT_Trigger,  // TIM 触发中断源
44.         ENABLE                // 使能
45.     );
46.
47.     // 设置优先级
48.     NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
49.     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2; // 先占优先级 0 级
50.     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;             // 从优先级 0 级
51.     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
52.     NVIC_Init(&NVIC_InitStructure);
53.
54.     TIM_Cmd(TIM3, ENABLE);  // 使能 TiM3 外设
55. }
56.
57. // 红外区域标志统计
58. static void IRM_ProcessFlags(void) {
59.     irm_leftflag_sum = irm_flag[0] + irm_flag[1] + irm_flag[2] + irm_flag[3] +
irm_flag[4];
60.     irm_midflag_sum = irm_flag[5] + irm_flag[6] + irm_flag[7] + irm_flag[8] +
irm_flag[9]; // 中间设置为 5 经测试是一个比较好的选择
61.     irm_rightflag_sum = irm_flag[10] + irm_flag[11] + irm_flag[12] + irm_flag[13] +
irm_flag[14];
62. }
63.
64. // 主算法：角度计算
65. static void IRM_CalculateAngle(void) {
66.     int i, j, k;
67.     int sum = 0, sumMax = 0;
68.
69.     for (i = 0; i < 15; ++i) {
70.         int max_temp = 0; // 连续区域的信号总数
71.         for (j = i; j < 15 && irm_flag[j]; ++j) // 从该信号口接受口往后面便利直到找到一个没
有接收到信号的信号口
72.             ++sum;
73.         for (k = i; k < j; ++k) // 便利算出连续信号总数
74.             max_temp += irm_data[k][0];
75.
76.         if (max_temp > irm_maxtrue_sum) {
77.             irm_maxtrue_sum = max_temp;
78.             sumMax = sum;
```

```
            for (k = i; k < j; ++k) {
                irm_maxtrue[k - i][0] = irm_data[k][0];
                irm_maxtrue[k - i][1] = irm_data[k][1];
            }
        }
        sum = 0;
    }

    for (i = 0; i < sumMax; ++i)
        irm_datasum += irm_maxtrue[i][0] * irm_maxtrue[i][1];

    if (irm_maxtrue_sum < 50)
        angle = 8;
    else
        angle = (float)irm_datasum / irm_maxtrue_sum;

#if bluetoothsend
    printf("%s","\r\ndatasum=");
    printf("%d",irm_datasum);
    printf("%s","\r\nirm_maxtrue_sum=");
    printf("%d",irm_maxtrue_sum);
    printf("%s","\r\nangle=");
    printf("%.2f",angle);
    printf("%s","\r\ndata0=");
    printf("%d",irm_data[0][0]);
    printf("%s","\r\ndata1=");
    printf("%d",irm_data[1][0]);
    printf("%s","\r\ndata2=");
    printf("%d",irm_data[2][0]);
    printf("%s","\r\ndata3=");
    printf("%d",irm_data[3][0]);
    printf("%s","\r\ndata4=");
    printf("%d",irm_data[4][0]);
    printf("%s","\r\ndata5=");
    printf("%d",irm_data[5][0]);
    printf("%s","\r\ndata6=");
    printf("%d",irm_data[6][0]);
    printf("%s","\r\ndata7=");
    printf("%d",irm_data[7][0]);
    printf("%s","\r\ndata8=");
    printf("%d",irm_data[8][0]);
    printf("%s","\r\ndata9=");
    printf("%d",irm_data[9][0]);
    printf("%s","\r\ndata10=");
```

```
123.     printf("%d",irm_data[10][0]);
124.     printf("%s","\r\ndata11=");
125.     printf("%d",irm_data[11][0]);
126.     printf("%s","\r\ndata12=");
127.     printf("%d",irm_data[12][0]);
128.     printf("%s","\r\ndata13=");
129.     printf("%d",irm_data[13][0]);
130.     printf("%s","\r\ndata14=");
131.     printf("%d",irm_data[14][0]);
132.     printf("%s","\r\n\r\n");
133. #endif
134. }
135. 
136. // 更新 PWM 输出
137. static void IRM_UpdatePWM(void) {
138.     angle_pwm = 1500 - (int)((angle - 8) * 200) + (angle - anglelast) * 40;
139.     if (angle_pwm >= 2100) angle_pwm = 2100;
140.     else if (angle_pwm <= 900) angle_pwm = 900;
141. 
142.     TIM_SetCompare1(TIM1, angle_pwm);
143. 
144. #if bluetoothsend
145.     printf("%s","\r\nirm_leftflag_sum=");
146.     printf("%d",irm_leftflag_sum);
147.     printf("%s","\r\nirm_midflag_sum=");
148.     printf("%d",irm_midflag_sum);
149.     printf("%s","\r\nirm_rightflag_sum=");
150.     printf("%d",irm_rightflag_sum);
151.     printf("\r\nangle_pwm = %d", angle_pwm);
152. #endif
153. }
154. 
155. // 清除数据准备下一周期
156. static void IRM_ClearData(void) {
157.     for (int i = 0; i < 15; ++i) { // 将数组和标志位还原
158.         irm_data[i][0] = 0;
159.         irm_data[i][1] = i + 1;
160.         irm_flag[i] = 0;
161.         irm_maxtrue[i][0] = 0;
162.         irm_maxtrue[i][1] = 0;
163.     }
164.     irm_leftflag_sum = irm_midflag_sum = irm_rightflag_sum = 0;
165.     irm_maxtrue_sum = irm_datasum = 0;
166. }
```

```
167.
168. void TIM3_IRQHandler(void) {
169.     if (TIM_GetITStatus(TIM3,TIM_IT_Update) != RESET) {
170.         tim1_counter = (tim1_counter + 1) % 11;
171.
172.         IRM_ProcessFlags();
173.
174.         if (irm_leftflag_sum <= 1 && irm_midflag_sum == 0 && irm_rightflag_sum <= 1)
{ // 丢信号
175.             angle = anglelast;  // 丢失信号处理，按照上次的角度
176.             isStraight = 0;
177.         } else if (irm_leftflag_sum >= 5 && irm_midflag_sum == 3 && irm_rightflag_sum >=
5) { // 全收到信号直走
178.             angle = 8; // 直走
179.             isStraight = 1;
180.         } else { // 正常接收情况
181.             isStraight = 0; // 在 irm_flag 数组中寻找收到信号的最大区块，该算法经验证可以有效
处理反射问题
182.             IRM_CalculateAngle();
183.         }
184.
185.         IRM_UpdatePWM();
186.         IRM_ClearData();
187.         anglelast = angle;
188.
189.         TIM_ClearITPendingBit(TIM3, TIM_FLAG_Update);
190.     }
191. }
192.
```

@ usart1.c

```
 1. #include "Usart1.h"
 2.
 3. /**
 4.  * @brief 初始化 USART1（PA9 TX, PA10 RX）
 5.  *
 6.  * 配置波特率 9600，8 位数据，无校验，1 停止位，无硬件流控
 7.  */
 8. void Usart1_Init(void) {
 9.     GPIO_InitTypeDef gpio_initstruct;
10.     USART_InitTypeDef usart_initstruct;
11.
12.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
```

```
13.      RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
14.
15.      // USART1 TX 引脚 PA9，复用推挽输出
16.      gpio_initstruct.GPIO_Mode = GPIO_Mode_AF_PP;
17.      gpio_initstruct.GPIO_Pin = GPIO_Pin_9;
18.      gpio_initstruct.GPIO_Speed = GPIO_Speed_50MHz;
19.      GPIO_Init(GPIOA, &gpio_initstruct);
20.
21.      // USART1 RX 引脚 PA10，浮空输入
22.      gpio_initstruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
23.      gpio_initstruct.GPIO_Pin = GPIO_Pin_10;
24.      GPIO_Init(GPIOA, &gpio_initstruct);
25.
26.      // // USART1 参数配置
27.      usart_initstruct.USART_BaudRate = 9600;
28.      usart_initstruct.USART_WordLength = USART_WordLength_8b;
29.      usart_initstruct.USART_Parity = USART_Parity_No;
30.      usart_initstruct.USART_StopBits = USART_StopBits_1;
31.      usart_initstruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
32.      usart_initstruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
33.      USART_Init(USART1, &usart_initstruct);
34.
35.      //USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);    //需要中断时写该语句
36.
37.      // 使能 USART1
38.      USART_Cmd( USART1, ENABLE);
39. }
40.
41. /**
42.  * @brief 发送一个字节数据
43.  * @param pUSARTx USART 外设指针 (如 USART1)
44.  * @param Data 要发送的数据字节
45.  */
46. void Usart_Send_Byte(USART_TypeDef* pUSARTx, uint8_t Data) {
47.      USART_SendData( pUSARTx,  Data);
48.      while(USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
49. }
50.
51. /**
52.  * @brief 发送一个 16 位数据（高 8 位先发，低 8 位后发）
53.  * @param pUSARTx USART 外设指针
54.  * @param Data 要发送的 16 位数据
55.  */
56. void Usart_Send_Word(USART_TypeDef* pUSARTx, uint16_t Data) {
```

```
57.      uint8_t temph, templ;
58.
59.      templ=(Data&0x00FF);
60.      temph=(Data&0xFF00) >> 8;
61.
62.      USART_SendData( pUSARTx, temph);
63.      while(USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
64.
65.      USART_SendData( pUSARTx, templ);
66.      while(USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
67. }
68.
69. /**
70.  * @brief 发送指定长度的字节数组
71.  * @param pUSARTx USART 外设指针
72.  * @param p 指向数据数组的指针
73.  * @param sum 要发送的字节数
74.  */
75. void Usart_Send_Array(USART_TypeDef* pUSARTx, uint8_t* p, uint8_t sum) {
76.      uint8_t i;
77.      for(i = 0;i < sum; i++) {
78.                USART_SendData( pUSARTx, *(p + i));
79.                while(USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
80.      }
81. }
82.
83. /**
84.  * @brief 重定义 fputc，用于 printf 重定向到 USART1
85.  * @param ch 发送的字符
86.  * @param f 文件指针
87.  * @return 发送的字符
88.  */
89. int fputc(int ch, FILE *f) {
90.      while((USART1->SR&0X40) == 0); //等待发送缓冲区空
91.     USART1->DR = (u8) ch;
92.      return ch;
93. }
94.
95. /**
96.  * @brief 发送以'\0'结尾的字符串
97.  * @param pUSARTx USART 外设指针
98.  * @param p 字符串指针
99.  */
100. void Usart_Send_String(USART_TypeDef* pUSARTx, uint8_t* p) {
```

```
101.      uint8_t i = 0;
102.      do {
103.                USART_SendData( pUSARTx, *(p + i));
104.                while(USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
105.                          i++;
106.      } while(*(p + i) != '\0');
107. }
```

@ pwm.c

```
1. #include "pwm.h"
2.
3. // PWM 频率 = 72MHz / (psc + 1) / (arr + 1)
4. // 占空比 = TIMx->CCR / (arr + 1)
5.
6. /**
7.  * @brief 初始化 TIM1 的 PWM 输出通道 1（PA8）
8.  * @param arr 自动重装载值（PWM 周期）
9.  * @param psc 预分频器值（PWM 频率控制）
10.  */
11. void TIM1_PWM_Init(u16 arr, u16 psc)
12. {
13.     TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
14.     TIM_OCInitTypeDef TIM_OCInitStructure;
15.
16.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE); // TIM1 高级定时器
17.
18.     // TIM1 时间基准配置
19.     TIM_TimeBaseStructure.TIM_Period = arr;
20.     TIM_TimeBaseStructure.TIM_Prescaler = psc;
21.     TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
22.     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
23.     TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
24.
25.     // PWM1 模式配置：通道 1（PA8）
26.     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
27.     TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
28.     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
29.     TIM_OCInitStructure.TIM_Pulse = 0;  // 默认占空比为 0
30.
31.     TIM_OC1Init(TIM1, &TIM_OCInitStructure);
32.     TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
33.
34.     TIM_ARRPreloadConfig(TIM1, ENABLE);
```

```
    TIM_Cmd(TIM1, ENABLE);
    TIM_CtrlPWMOutputs(TIM1, ENABLE);  // 高级定时器必须开启主输出
}

/**
 * @brief 初始化 TIM3 的 PWM 输出通道 2（PA7）
 * @param arr 自动重装载值（PWM 周期）
 * @param psc 预分频器值（PWM 频率控制）
 */
void TIM3_PWM_Init(u16 arr, u16 psc)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_OCInitTypeDef TIM_OCInitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);  // TIM3 通用定时器

    // TIM3 时间基准配置
    TIM_TimeBaseStructure.TIM_Period = arr;
    TIM_TimeBaseStructure.TIM_Prescaler = psc;
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

    // PWM 模式配置：通道 2（PA7）
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCInitStructure.TIM_Pulse = 0;

    TIM_OC2Init(TIM3, &TIM_OCInitStructure);
    TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable);

    TIM_ARRPreloadConfig(TIM3, ENABLE);
    TIM_Cmd(TIM3, ENABLE);
}

```

@ main.c

```
#include "stm32f10x.h"
#include "pwm.h"
#include "timer1.h"
#include "GPIO.h"
#include "Usart1.h"
```

```
 6.
 7. u32 irm_data[15][2] = {
 8.     {0, 1}, {0, 2}, {0, 3}, {0, 4}, {0, 5},
 9.     {0, 6}, {0, 7}, {0, 8}, {0, 9}, {0, 10},
10.     {0, 11}, {0, 12}, {0, 13}, {0, 14}, {0, 15}
11. }; // [i][0]是一个周期内收到的信号总数，[i][1]是每个信号接收源的标号
12.
13. int irm_flag[15] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // 每个接收头是否收到的
标志位
14.
15. // 添加静态变量，用于记录上一次的引脚状态
16. static uint8_t last_irm_state[15] = {0}; // 初始化为0
17.
18. int main(void) {
19.     TIM1_PWM_Init(19999, 71);
20.     // TIM3_PWM_Init(19999, 71);
21.
22.     GPIO_Config(); // 使能红外接收端口
23.     Usart1_Init();
24.
25.     TIM_SetCompare1(TIM1,1500);
26.     // TIM_SetCompare2(TIM3,5000);
27.
28.     TIM3_Init(); // TIM3 计时用于开启 67ms 的中断
29.
30.     // 在进入循环前，先读取一次所有引脚状态，作为初始值
31.     last_irm_state[0] = irm1;
32.     last_irm_state[1] = irm2;
33.     last_irm_state[2] = irm3;
34.     last_irm_state[3] = irm4;
35.     last_irm_state[4] = irm5;
36.     last_irm_state[5] = irm6;
37.     last_irm_state[6] = irm7;
38.     last_irm_state[7] = irm8;
39.     last_irm_state[8] = irm9;
40.     last_irm_state[9] = irm10;
41.     last_irm_state[10] = irm11;
42.     last_irm_state[11] = irm12;
43.     last_irm_state[12] = irm13;
44.     last_irm_state[13] = irm14;
45.     last_irm_state[14] = irm15;
46.
47.     while(1) {
48.         // 检测每个引脚的下降沿（高->低）
```

```
        if (irm1 == 0 && last_irm_state[0] == 1) {
            irm_data[0][0]++;
            irm_flag[0] = 1;
        } if (irm2 == 0 && last_irm_state[1] == 1) {
            irm_data[1][0]++;
            irm_flag[1] = 1;
        } if (irm3 == 0 && last_irm_state[2] == 1) {
            irm_data[2][0]++;
            irm_flag[2] = 1;
        } if (irm4 == 0 && last_irm_state[3] == 1) {
            irm_data[3][0]++;
            irm_flag[3] = 1;
        } if (irm5 == 0 && last_irm_state[4] == 1) {
            irm_data[4][0]++;
            irm_flag[4] = 1;
        } if (irm6 == 0 && last_irm_state[5] == 1) {
            irm_data[5][0]++;
            irm_flag[5] = 1;
        } if (irm7 == 0 && last_irm_state[6] == 1) {
            irm_data[6][0]++;
            irm_flag[6] = 1;
        } if (irm8 == 0 && last_irm_state[7] == 1) {
            irm_data[7][0]++;
            irm_flag[7] = 1;
        } if (irm9 == 0 && last_irm_state[8] == 1) {
            irm_data[8][0]++;
            irm_flag[8] = 1;
        } if (irm10 == 0 && last_irm_state[9] == 1) {
            irm_data[9][0]++;
            irm_flag[9] = 1;
        } if (irm11 == 0 && last_irm_state[10] == 1) {
            irm_data[10][0]++;
            irm_flag[10] = 1;
        } if (irm12 == 0 && last_irm_state[11] == 1) {
            irm_data[11][0]++;
            irm_flag[11] = 1;
        } if (irm13 == 0 && last_irm_state[12] == 1) {
            irm_data[12][0]++;
            irm_flag[12] = 1;
        } if (irm14 == 0 && last_irm_state[13] == 1) {
            irm_data[13][0]++;
            irm_flag[13] = 1;
        } if (irm15 == 0 && last_irm_state[14] == 1) {
            irm_data[14][0]++;
```

```
            irm_flag[14] = 1;
        }

        // 更新上一次的状态
        last_irm_state[0] = irm1;
        last_irm_state[1] = irm2;
        last_irm_state[2] = irm3;
        last_irm_state[3] = irm4;
        last_irm_state[4] = irm5;
        last_irm_state[5] = irm6;
        last_irm_state[6] = irm7;
        last_irm_state[7] = irm8;
        last_irm_state[8] = irm9;
        last_irm_state[9] = irm10;
        last_irm_state[10] = irm11;
        last_irm_state[11] = irm12;
        last_irm_state[12] = irm13;
        last_irm_state[13] = irm14;
        last_irm_state[14] = irm15;
    }
}

```