/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2025 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

#include "spi.h"

#include "tim.h"

#include "gpio.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

// 调试宏定义 - 如果不需要调试输出，设置为0

#define DEBUG\_ENABLE 1

#if DEBUG\_ENABLE

#define DEBUG\_PRINT(...) printf(\_\_VA\_ARGS\_\_)

#else

#define DEBUG\_PRINT(...)

#endif

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

/\* USER CODE BEGIN PV \*/

// PS2手柄数据结构

typedef struct {

int8\_t lx, ly, rx, ry; // 摇杆数据 (-128 ~ 127)

bool up, down, left, right; // 方向键

bool l1, l2, r1, r2; // 肩键

bool triangle, circle, cross, square; // 形状键

} PS2Data\_t;

// 全局变量

PS2Data\_t ps2\_data;

uint32\_t last\_ps2\_read\_time = 0;

// 电机状态

typedef enum {

MOTOR\_STOP = 0,

MOTOR\_FORWARD,

MOTOR\_BACKWARD

} MotorState\_t;

MotorState\_t left\_motor\_state = MOTOR\_STOP;

MotorState\_t right\_motor\_state = MOTOR\_STOP;

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

/\* USER CODE BEGIN PFP \*/

// 自定义函数声明 - 所有函数都在本文件中实现，不需要BSP！

void Motors\_SetSpeed(float left\_speed, float right\_speed);

void Servo\_SetAngle(float angle);

void BSP\_Motor\_Init(void);

void PS2\_ReadData(PS2Data\_t\* data);

void PS2\_Init(void);

void System\_Init(void);

void Control\_Task(void);

void Debug\_PrintPS2Data(void);

void Hardware\_SelfTest(void);

void PS2\_TestConnection(void);

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

// 重定向printf到串口（如果需要调试）

#ifdef \_\_GNUC\_\_

#define PUTCHAR\_PROTOTYPE int \_\_io\_putchar(int ch)

#else

#define PUTCHAR\_PROTOTYPE int fputc(int ch, FILE \*f)

#endif

PUTCHAR\_PROTOTYPE

{

// 如果需要串口调试，取消注释下面这行，并在CubeMX中配置USART1

// HAL\_UART\_Transmit(&huart1, (uint8\_t \*)&ch, 1, HAL\_MAX\_DELAY);

return ch;

}

// 电机控制函数 - 支持正反转

void Motors\_SetSpeed(float left\_speed, float right\_speed)

{

// 限制速度范围

if (left\_speed < -1.0f) left\_speed = -1.0f;

if (left\_speed > 1.0f) left\_speed = 1.0f;

if (right\_speed < -1.0f) right\_speed = -1.0f;

if (right\_speed > 1.0f) right\_speed = 1.0f;

// 更新电机状态

if (left\_speed > 0.1f) left\_motor\_state = MOTOR\_FORWARD;

else if (left\_speed < -0.1f) left\_motor\_state = MOTOR\_BACKWARD;

else left\_motor\_state = MOTOR\_STOP;

if (right\_speed > 0.1f) right\_motor\_state = MOTOR\_FORWARD;

else if (right\_speed < -0.1f) right\_motor\_state = MOTOR\_BACKWARD;

else right\_motor\_state = MOTOR\_STOP;

// === 左侧电机控制 (PA0-正转, PA2-反转) ===

if (left\_speed > 0) {

// 左电机正转

uint32\_t pulse = (uint32\_t)(19999 \* left\_speed);

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_1, pulse); // PA0 - 正转

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_3, 0); // PA2 - 关闭反转

} else if (left\_speed < 0) {

// 左电机反转

uint32\_t pulse = (uint32\_t)(19999 \* (-left\_speed));

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_3, pulse); // PA2 - 反转

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_1, 0); // PA0 - 关闭正转

} else {

// 左电机停止

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_1, 0);

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_3, 0);

}

// === 右侧电机控制 (PB0-正转, PB1-反转) ===

if (right\_speed > 0) {

// 右电机正转

uint32\_t pulse = (uint32\_t)(19999 \* right\_speed);

\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_3, pulse); // PB0 - 正转

\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_4, 0); // PB1 - 关闭反转

} else if (right\_speed < 0) {

// 右电机反转

uint32\_t pulse = (uint32\_t)(19999 \* (-right\_speed));

\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_4, pulse); // PB1 - 反转

\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_3, 0); // PB0 - 关闭正转

} else {

// 右电机停止

\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_3, 0);

\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_4, 0);

}

DEBUG\_PRINT("电机: 左%.0f%%, 右%.0f%%\r\n", left\_speed \* 100, right\_speed \* 100);

}

// 舵机控制函数

void Servo\_SetAngle(float angle)

{

if (angle < -90.0f) angle = -90.0f;

if (angle > 90.0f) angle = 90.0f;

// 角度映射到PWM脉冲: -90°→500, 0°→1500, 90°→2500

uint32\_t pulse = 1500 + (uint32\_t)(angle / 180.0f \* 1000.0f);

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_2, pulse);

DEBUG\_PRINT("舵机: %.0f度\r\n", angle);

}

// 电机初始化函数

void BSP\_Motor\_Init(void)

{

// 启动所有PWM通道

// TIM2 - 左电机和舵机

HAL\_TIM\_PWM\_Start(&htim2, TIM\_CHANNEL\_1); // PA0 - 左电机正转

HAL\_TIM\_PWM\_Start(&htim2, TIM\_CHANNEL\_2); // PA1 - 舵机

HAL\_TIM\_PWM\_Start(&htim2, TIM\_CHANNEL\_3); // PA2 - 左电机反转

// TIM3 - 右电机

HAL\_TIM\_PWM\_Start(&htim3, TIM\_CHANNEL\_3); // PB0 - 右电机正转

HAL\_TIM\_PWM\_Start(&htim3, TIM\_CHANNEL\_4); // PB1 - 右电机反转

// 设置初始状态

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_1, 0); // 左电机正转停止

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_2, 1500); // 舵机中位

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_3, 0); // 左电机反转停止

\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_3, 0); // 右电机正转停止

\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_4, 0); // 右电机反转停止

DEBUG\_PRINT("电机系统初始化完成\r\n");

}

// PS2手柄通信函数

// PS2片选控制函数

static void PS2\_CS\_Low(void) {

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_4, GPIO\_PIN\_RESET);

}

static void PS2\_CS\_High(void) {

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_4, GPIO\_PIN\_SET);

}

// SPI数据传输函数

static uint8\_t PS2\_TransferByte(uint8\_t data)

{

uint8\_t rx;

HAL\_SPI\_TransmitReceive(&hspi1, &data, &rx, 1, 100);

return rx;

}

// 读取PS2数据函数

void PS2\_ReadData(PS2Data\_t\* data)

{

uint8\_t tx\_buf[9] = {0x01, 0x42, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

uint8\_t rx\_buf[9] = {0};

PS2\_CS\_Low();

HAL\_Delay(1);

for (int i = 0; i < 9; i++) {

rx\_buf[i] = PS2\_TransferByte(tx\_buf[i]);

HAL\_Delay(1);

}

PS2\_CS\_High();

// 解析按键数据

uint16\_t key = (uint16\_t)rx\_buf[3] | ((uint16\_t)rx\_buf[4] << 8);

key = ~key;

data->up = (key & (1 << 4)) != 0;

data->right = (key & (1 << 5)) != 0;

data->down = (key & (1 << 6)) != 0;

data->left = (key & (1 << 7)) != 0;

data->l2 = (key & (1 << 8)) != 0;

data->r2 = (key & (1 << 9)) != 0;

data->l1 = (key & (1 << 10)) != 0;

data->r1 = (key & (1 << 11)) != 0;

data->triangle = (key & (1 << 12)) != 0;

data->circle = (key & (1 << 13)) != 0;

data->cross = (key & (1 << 14)) != 0;

data->square = (key & (1 << 15)) != 0;

// 摇杆数据 (-128 ~ 127)

data->lx = rx\_buf[7] - 128;

data->ly = rx\_buf[8] - 128;

data->rx = rx\_buf[5] - 128;

data->ry = rx\_buf[6] - 128;

}

// PS2初始化

void PS2\_Init(void)

{

DEBUG\_PRINT("初始化PS2手柄...\r\n");

// 确保CS引脚初始为高电平

PS2\_CS\_High();

HAL\_Delay(100);

// 尝试读取数据

PS2\_ReadData(&ps2\_data);

DEBUG\_PRINT("PS2初始化完成 - 绿灯模式可用\r\n");

}

// PS2连接测试

void PS2\_TestConnection(void)

{

DEBUG\_PRINT("PS2连接测试...\r\n");

for(int i = 0; i < 3; i++) {

PS2\_ReadData(&ps2\_data);

DEBUG\_PRINT("测试%d: ", i+1);

DEBUG\_PRINT("LX=%4d LY=%4d RX=%4d RY=%4d ",

ps2\_data.lx, ps2\_data.ly, ps2\_data.rx, ps2\_data.ry);

// 显示按键状态

if(ps2\_data.up) DEBUG\_PRINT("UP ");

if(ps2\_data.down) DEBUG\_PRINT("DOWN ");

if(ps2\_data.left) DEBUG\_PRINT("LEFT ");

if(ps2\_data.right) DEBUG\_PRINT("RIGHT ");

DEBUG\_PRINT("\r\n");

// 如果有有效数据，说明通信正常

if(ps2\_data.lx != 0 || ps2\_data.ly != 0 || ps2\_data.rx != 0 || ps2\_data.ry != 0 ||

ps2\_data.up || ps2\_data.down || ps2\_data.left || ps2\_data.right) {

DEBUG\_PRINT("✅ PS2通信正常\r\n");

return;

}

HAL\_Delay(200);

}

DEBUG\_PRINT("❌ PS2通信失败\r\n");

}

// 系统功能函数

// 系统初始化

void System\_Init(void)

{

DEBUG\_PRINT("=== 双电机小车控制系统 ===\r\n");

DEBUG\_PRINT("所有功能集成在main.c中，无需BSP文件！\r\n");

BSP\_Motor\_Init();

PS2\_Init();

DEBUG\_PRINT("系统初始化完成\r\n");

}

// 调试打印PS2数据

void Debug\_PrintPS2Data(void)

{

static uint32\_t last\_debug\_time = 0;

if (HAL\_GetTick() - last\_debug\_time > 2000) { // 每2秒打印一次

last\_debug\_time = HAL\_GetTick();

DEBUG\_PRINT("PS2状态: ");

DEBUG\_PRINT("LX=%4d LY=%4d RX=%4d RY=%4d | ",

ps2\_data.lx, ps2\_data.ly, ps2\_data.rx, ps2\_data.ry);

if (ps2\_data.up) DEBUG\_PRINT("UP ");

if (ps2\_data.down) DEBUG\_PRINT("DOWN ");

if (ps2\_data.left) DEBUG\_PRINT("LEFT ");

if (ps2\_data.right) DEBUG\_PRINT("RIGHT ");

if (ps2\_data.l1) DEBUG\_PRINT("L1 ");

if (ps2\_data.r1) DEBUG\_PRINT("R1 ");

DEBUG\_PRINT("| 电机: ");

if (left\_motor\_state == MOTOR\_FORWARD) DEBUG\_PRINT("左前 ");

else if (left\_motor\_state == MOTOR\_BACKWARD) DEBUG\_PRINT("左后 ");

else DEBUG\_PRINT("左停 ");

if (right\_motor\_state == MOTOR\_FORWARD) DEBUG\_PRINT("右前 ");

else if (right\_motor\_state == MOTOR\_BACKWARD) DEBUG\_PRINT("右后 ");

else DEBUG\_PRINT("右停 ");

DEBUG\_PRINT("\r\n");

}

}

// 主控制任务 - 适配绿灯模式

void Control\_Task(void)

{

// 读取PS2手柄数据（每100ms读取一次）

if (HAL\_GetTick() - last\_ps2\_read\_time > 100) {

last\_ps2\_read\_time = HAL\_GetTick();

PS2\_ReadData(&ps2\_data);

// 调试输出

Debug\_PrintPS2Data();

// === 控制逻辑 - 适配绿灯模式 ===

float left\_speed = 0.0f;

float right\_speed = 0.0f;

float steer\_angle = 0.0f;

// 1. 方向键控制基本运动

if (ps2\_data.up) {

// 前进

left\_speed = 0.6f;

right\_speed = 0.6f;

} else if (ps2\_data.down) {

// 后退

left\_speed = -0.4f;

right\_speed = -0.4f;

} else if (ps2\_data.left) {

// 原地左转

left\_speed = -0.4f;

right\_speed = 0.4f;

steer\_angle = -30.0f;

} else if (ps2\_data.right) {

// 原地右转

left\_speed = 0.4f;

right\_speed = -0.4f;

steer\_angle = 30.0f;

}

// 2. 肩键控制

if (ps2\_data.l1) {

// L1急停

left\_speed = 0;

right\_speed = 0;

}

if (ps2\_data.r1) {

// R1舵机回中

steer\_angle = 0.0f;

}

// 3. 形状键组合控制

if (ps2\_data.triangle) {

// △ + 方向键：弧线运动

if (ps2\_data.up) {

left\_speed = 0.4f;

right\_speed = 0.6f; // 右弧线

steer\_angle = 20.0f;

} else if (ps2\_data.right) {

left\_speed = 0.5f;

right\_speed = 0.3f; // 左弧线

steer\_angle = -20.0f;

}

}

if (ps2\_data.cross) {

// ×键：停止所有

left\_speed = 0;

right\_speed = 0;

steer\_angle = 0.0f;

}

// 4. 设置执行器

Servo\_SetAngle(steer\_angle);

Motors\_SetSpeed(left\_speed, right\_speed);

}

}

// 硬件自检函数

void Hardware\_SelfTest(void)

{

DEBUG\_PRINT("开始硬件自检...\r\n");

// 舵机测试

DEBUG\_PRINT("舵机测试...\r\n");

Servo\_SetAngle(-45.0f);

HAL\_Delay(1000);

Servo\_SetAngle(45.0f);

HAL\_Delay(1000);

Servo\_SetAngle(0.0f);

HAL\_Delay(1000);

// 电机测试

DEBUG\_PRINT("左电机测试...\r\n");

Motors\_SetSpeed(0.4f, 0.0f);

HAL\_Delay(2000);

Motors\_SetSpeed(0.0f, 0.0f);

HAL\_Delay(500);

DEBUG\_PRINT("右电机测试...\r\n");

Motors\_SetSpeed(0.0f, 0.4f);

HAL\_Delay(2000);

Motors\_SetSpeed(0.0f, 0.0f);

HAL\_Delay(500);

DEBUG\_PRINT("双电机测试...\r\n");

Motors\_SetSpeed(0.3f, 0.3f);

HAL\_Delay(2000);

Motors\_SetSpeed(-0.3f, -0.3f);

HAL\_Delay(2000);

Motors\_SetSpeed(0.0f, 0.0f);

DEBUG\_PRINT("硬件自检完成！\r\n");

}

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

int main(void)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

SystemClock\_Config();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_SPI1\_Init();

MX\_TIM2\_Init();

MX\_TIM3\_Init();

/\* USER CODE BEGIN 2 \*/

// 系统初始化

System\_Init();

// PS2连接测试

PS2\_TestConnection();

// 硬件自检（可选，注释掉可跳过）

Hardware\_SelfTest();

DEBUG\_PRINT("系统就绪！控制方式：\r\n");

DEBUG\_PRINT("↑:前进 ↓:后退 ←:左转 →:右转\r\n");

DEBUG\_PRINT("L1:急停 R1:舵机回中 ×:停止\r\n");

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

// 主控制任务

Control\_Task();

// 短延迟

HAL\_Delay(10);

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.HSEPredivValue = RCC\_HSE\_PREDIV\_DIV1;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL9;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

Error\_Handler();

}

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/