STM32 ????????? (Basic Throttle Test)

? ????

驗證底層硬體與通訊是否正常工作:

1. ADC 油門訊號讀取是否平順。
2. CAN Bus 傳送是否正常，沒有卡死。
3. 4 顆 VESC 是否都能正確接收指令並轉動。
4. 確認 **4** 顆馬達的正反轉方向是否一致（若方向反了，請直接在 VESC Tool 裡將該馬達反轉，不要在程式裡改，保持程式單純）。

? ??????

1. 將車輛四輪確實架空。
2. 燒錄這份程式碼。
3. 輕踩油門，觀察四個輪子是否「同時轉動」且「方向一致往前」。
4. 放開油門，觀察輪子是否能正常停止。
5. 如果發現有輪子往後轉，請將筆電接上該輪的 VESC，打開 VESC Tool，將 `Motor Configuration` -> `General` -> `Invert Motor Direction` 打勾（設為 True），寫入後再試一次。

? ??????? (`main.c`)

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : 最基礎的油門直通測試 (四輪同等輸出)
  ******************************************************************************
  */
/* USER CODE END Header */
```

```
#include "main.h"

/* USER CODE BEGIN PD */
// --- 測試用安全參數 ---
// 單顆馬達最大電流限制 (測試階段建議設小一點，例如 5A = 5000mA)
#define TEST_MAX_CURRENT_MA   5000

// 油門死區 (0~4095 之間，小於此值馬達不輸出，防止雜訊抖動)
#define THROTTLE_DEADZONE     150

// VESC CAN ID 定義
#define VESC_ID_FRONT_L       0xB1
#define VESC_ID_FRONT_R       0xB2
#define VESC_ID_REAR_L        0xA3
#define VESC_ID_REAR_R        0xA4

#define CAN_PACKET_SET_CURRENT 1
/* USER CODE END PD */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
CAN_HandleTypeDef hcan1;

/* USER CODE BEGIN PV */
uint16_t throttle_adc = 0;
int32_t  target_current = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_CAN1_Init(void);

/* USER CODE BEGIN PFP */
void VESC_Send_Current(uint8_t controller_id, int32_t current_ma);
void Basic_Drive_Loop(void);
/* USER CODE END PFP */

int main(void)
{
```

```
  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_CAN1_Init();

  /* USER CODE BEGIN 2 */
  // 啟動 CAN 總線
  HAL_CAN_Start(&hcan1);

  // 啟動 ADC 連續轉換
  HAL_ADC_Start(&hadc1);
  /* USER CODE END 2 */

  /* Infinite loop */
  while (1)
  {
      Basic_Drive_Loop();
  }
}

/* USER CODE BEGIN 4 */

/**
 * @brief 最基礎的直通驅動迴圈 (100Hz)
 */
void Basic_Drive_Loop(void)
{
    static uint32_t last_tick = 0;
    uint32_t current_tick = HAL_GetTick();

    // 限制執行頻率為 100Hz (每 10ms 執行一次即可，測試階段不需太快)
    if ((current_tick - last_tick) < 10) {
        return;
    }
    last_tick = current_tick;

    // 1. 讀取油門 ADC
    throttle_adc = HAL_ADC_GetValue(&hadc1);
```

```
    // 2. 加入死區與計算目標電流
    if (throttle_adc < THROTTLE_DEADZONE) {
        target_current = 0;
    } else {
        // 將 ADC 值 (扣除死區後) 映射到目標電流
        // 公式: (當前ADC - 死區) * 最大電流 / (4095 - 死區)
        uint32_t active_adc = throttle_adc - THROTTLE_DEADZONE;
        target_current = (active_adc * TEST_MAX_CURRENT_MA) / (4095 - THROTTLE_DEADZONE);
    }

    // 3. 將相同的電流指令，依序發送給 4 顆馬達
    // 這裡加上微小的延遲(1ms)避免 CAN Bus 瞬間塞滿 4 個封包
    VESC_Send_Current(VESC_ID_FRONT_L, target_current);
    HAL_Delay(1);

    VESC_Send_Current(VESC_ID_FRONT_R, target_current);
    HAL_Delay(1);

    VESC_Send_Current(VESC_ID_REAR_L,  target_current);
    HAL_Delay(1);

    VESC_Send_Current(VESC_ID_REAR_R,  target_current);
}

/**
 * @brief 發送電流命令給 VESC (含防卡死保護)
 */
void VESC_Send_Current(uint8_t controller_id, int32_t current_ma)
{
    CAN_TxHeaderTypeDef TxHeader;
    uint32_t TxMailbox;
    uint8_t TxData[4];

    TxHeader.ExtId = (CAN_PACKET_SET_CURRENT << 8) | controller_id;
    TxHeader.IDE = CAN_ID_EXT;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.DLC = 4;

    TxData[0] = (uint8_t)(current_ma >> 24);
    TxData[1] = (uint8_t)(current_ma >> 16);
    TxData[2] = (uint8_t)(current_ma >> 8);
```

```
    TxData[3] = (uint8_t)(current_ma);

    uint32_t timeout = 0;
    // 等待 Mailbox 有空位
    while (HAL_CAN_GetTxMailboxesFreeLevel(&hcan1) == 0)
    {
        timeout++;
        if (timeout > 50000) return; // 超時放棄, 防止當機
    }
    HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
}

/* USER CODE END 4 */
```

Revision #2
Created 2026-04-01 02:06:09 UTC by TaipeiTechRacing
Updated 2026-04-06 06:22:52 UTC