TTR9_WSM??????? (Python ??)

STM32 CAN ??????? (Python ??)

本教學介紹如何使用 Python 建立一個專業的 CAN 監控工具。該工具支援阻塞式 **(Blocking)** 數據接收、實時數字顯示、動態圖表縮放以及 **CSV** 自動存檔功能。

1. ????

在執行 Python 程式之前，請確保已安裝必要的函式庫。打開終端機 (Terminal/CMD) 執行：

```
# 安裝 CAN 通訊庫
pip install python-can

# 安裝圖表繪製與數值處理庫
pip install matplotlib numpy

# Tkinter 通常內建於 Python 中，若 Linux 環境缺少可執行：
# sudo apt-get install python3-tk
```

2. ?????? (STM32 ?)

本程式預設接收來自 STM32 的 FDCAN/CAN 數據，格式如下：

- **ID:** `0x123` (可於程式內修改 `TARGET_ID`)
- 數據長度: 8 Bytes
- 編碼方式: Little-Endian (小端序)

Byte 偏移	數據內容	類型	說明

`Data[0:1]`	Wheel RPM	int16	原始數值
`Data[2:3]`	Current Temp	int16	原始值需 `/ 100.0` 還原為攝氏度
`Data[4:7]`	預留	-	目前填 `0`

3. ?????? (?? GUI ???)

您可以將以下程式碼儲存為 `can_monitor_pro.py` 並執行。

```
import can
import struct
import threading
import collections
import csv
import time
import tkinter as tk
from tkinter import ttk, messagebox
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.animation import FuncAnimation

# ================ 配置區 =================
CAN_INTERFACE = 'pcan'         # PCAN 使用 'pcan', CANable 使用 'slcan'
CAN_CHANNEL = 'PCAN_USBBUS1'     # 依硬體調整 (如 'COM3')
CAN_BITRATE = 500000
TARGET_ID = 0x123              # 須與 STM32 CAN_ID 一致
# ========================================

class ProfessionalCANMonitor:
    def __init__(self, root):
        self.root = root
        self.root.title("CAN Pro Monitor - RPM & Temp Dashboard")
        self.root.geometry("1000x750")

        # 資料控制緩衝區
        self.max_points = 100
        self.rpm_history = collections.deque([0]*self.max_points, maxlen=self.max_points)
        self.temp_history = collections.deque([0]*self.max_points, maxlen=self.max_points)
```

```
        self.data_lock = threading.Lock()
        self.is_running = True
        self.is_recording = tk.BooleanVar(value=False)

        self.setup_ui()

        # 啟動後台阻塞式接收執行緒
        self.thread = threading.Thread(target=self.can_reader, daemon=True)
        self.thread.start()

        # 啟動圖表動畫更新 (每 50 毫秒更新一次畫面)
        self.ani = FuncAnimation(self.fig, self.update_ui, interval=50, blit=False)

    def setup_ui(self):
        """ 建立 UI 佈局 """
        # --- 1. 左側控制面板 ---
        control_frame = ttk.LabelFrame(self.root, text=" 系統設定 ", padding="15")
        control_frame.pack(side=tk.LEFT, fill=tk.Y, padx=10, pady=10)

        # Y 軸範圍設定 (讓使用者可動態調整圖表高度)
        ttk.Label(control_frame, text="RPM 圖表上限:", font=('Arial', 10)).pack(anchor=tk.W)
        self.rpm_limit_entry = ttk.Entry(control_frame, width=15)
        self.rpm_limit_entry.insert(0, "5000")
        self.rpm_limit_entry.pack(pady=5)

        ttk.Label(control_frame, text="溫度上限 (°C):", font=('Arial', 10)).pack(anchor=tk.W)
        self.temp_limit_entry = ttk.Entry(control_frame, width=15)
        self.temp_limit_entry.insert(0, "100")
        self.temp_limit_entry.pack(pady=5)

        ttk.Button(control_frame, text="更新顯示範圍", command=self.update_limits).pack(fill=tk.X,
pady=15)

        # 功能開關
        ttk.Separator(control_frame, orient='horizontal').pack(fill=tk.X, pady=10)
        ttk.Checkbutton(control_frame, text="開啟 CSV 存檔",
variable=self.is_recording).pack(anchor=tk.W, pady=5)

        ttk.Label(control_frame, text="存檔路徑:\ncan_log.csv",
```

```
foreground="gray").pack(anchor=tk.W)

        # --- 2. 右側主顯示區 ---
        right_frame = ttk.Frame(self.root)
        right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        # 即時數據面板 (大字體)
        display_frame = ttk.Frame(right_frame, padding="10", relief="groove")
        display_frame.pack(side=tk.TOP, fill=tk.X, padx=10, pady=10)

        self.rpm_val = ttk.Label(display_frame, text="RPM: 0", font=('Arial', 28, 'bold'),
foreground="#1f77b4")
        self.rpm_val.pack(side=tk.LEFT, padx=40)

        self.temp_val = ttk.Label(display_frame, text="0.00 °C", font=('Arial', 28, 'bold'),
foreground="#d62728")
        self.temp_val.pack(side=tk.LEFT, padx=40)

        # 下方 Matplotlib 圖表嵌入
        self.fig, (self.ax1, self.ax2) = plt.subplots(2, 1, figsize=(7, 7))
        self.line_rpm, = self.ax1.plot(range(self.max_points), self.rpm_history,
color='#1f77b4', lw=2, label='RPM')
        self.line_temp, = self.ax2.plot(range(self.max_points), self.temp_history,
color='#d62728', lw=2, label='Temp')

        self.ax1.set_ylim(0, 5000)
        self.ax2.set_ylim(0, 100)
        self.ax1.set_title("Wheel Speed Trend")
        self.ax2.set_title("Temperature Trend")
        self.ax1.grid(True, linestyle='--', alpha=0.6)
        self.ax2.grid(True, linestyle='--', alpha=0.6)

        self.canvas = FigureCanvasTkAgg(self.fig, master=right_frame)
        self.canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

    def update_limits(self):
        """ 動態調整圖表 Y 軸範圍 """
        try:
            new_rpm = float(self.rpm_limit_entry.get())
```

```
            new_temp = float(self.temp_limit_entry.get())
            self.ax1.set_ylim(0, new_rpm)
            self.ax2.set_ylim(0, new_temp)
            self.canvas.draw()
        except ValueError:
            messagebox.showwarning("格式錯誤", "請輸入有效的數值")

    def can_reader(self):
        """ 駐塞式讀取執行緒: 確保數據抓取不漏接 """
        try:
            bus = can.Bus(interface=CAN_INTERFACE, channel=CAN_CHANNEL, bitrate=CAN_BITRATE)
            bus.set_filters([{"can_id": TARGET_ID, "can_mask": 0xFFF, "extended": False}])

            with open('can_log.csv', mode='a', newline='') as f:
                writer = csv.writer(f)
                # 若檔案是空的則寫入表頭
                if f.tell() == 0:
                    writer.writerow(['Time', 'RPM', 'Temp_C'])

                while self.is_running:
                    # 阻塞式接收, 直到有封包進來或逾時 0.5 秒 (逾時是為了檢查終止訊號)
                    msg = bus.recv(timeout=0.5)
                    if msg:
                        # 解析 STM32 發送的 Little-Endian 數據 (<hh 代表兩個 int16)
                        rpm_raw, temp_raw = struct.unpack('<hh', msg.data[0:4])
                        actual_temp = temp_raw / 100.0

                        # 進入 Critical Section 更新緩衝區
                        with self.data_lock:
                            self.rpm_history.append(rpm_raw)
                            self.temp_history.append(actual_temp)

                        # CSV 存檔功能
                        if self.is_recording.get():
                            writer.writerow([time.strftime("%H:%M:%S"), rpm_raw, actual_temp])

        except Exception as e:
            print(f"CAN 錯誤: {e}")
```

```
    def update_ui(self, frame):
        """ 更新 UI 上的數字與圖表曲線 """
        with self.data_lock:
            curr_rpm = self.rpm_history[-1]
            curr_temp = self.temp_history[-1]

            self.rpm_val.config(text=f"RPM: {curr_rpm}")
            self.temp_val.config(text=f"{curr_temp:.2f} °C")

            self.line_rpm.set_ydata(list(self.rpm_history))
            self.line_temp.set_ydata(list(self.temp_history))

        return self.line_rpm, self.line_temp

    def on_closing(self):
        """ 安全關閉程式 """
        self.is_running = False
        self.root.destroy()

if __name__ == "__main__":
    root = tk.Tk()
    app = ProfessionalCANMonitor(root)
    root.protocol("WM_DELETE_WINDOW", app.on_closing)
    root.mainloop()
```

4. ??????

- **阻塞式讀取 (`bus.recv`):** 與輪詢方式不同，阻塞式讀取會在有 CAN 封包進入時立即觸發，這對於高頻率的輪速與溫度數據（如 10ms 一筆）能保證數據的完整性。
- **`struct.unpack('<hh', ...)`:** 這是處理 STM32 小端序數據最安全的方法。它會自動處理負數符號位，並將兩個 Byte 直接合併為整數。
- **多執行緒架構:**
  - **主執行緒:** 負責 Tkinter 視窗更新與 Matplotlib 繪圖（約 20FPS）。
  - **背景執行緒:** 負責 `can.Bus` 數據接收，兩者透過 `threading.Lock` 確保數據同步時不會發生衝突。
- **CSV 存檔:** 勾選 UI 上的開關後，數據會以 `Time`, `RPM`, `Temp_C` 格式記錄，方便後續導入 Excel 分析。

> 後續建議：
>
> - 如果您發現數據更新過快導致圖表閃爍，可以適度調大 `FuncAnimation` 中的 `interval`。
> - 若要增加多個 ID 的過濾，只需在 `bus.set_filters` 列表中增加項目即可。

Revision #2
Created 2026-04-01 02:06:33 UTC by TaipeiTechRacing
Updated 2026-04-11 14:36:21 UTC by AI Agent