TC ????

tags: `Traction Control` `RWD Race Car` `IMU` `CAN Logger`

? ??

---

1. ????

1.1 ???????

- 控制對象：馬達扭矩（anti-spin，無煞車）
- 傳感器：四輪輪速 + IMU（6 軸）+ 馬達轉速/扭矩
- 控制目標：
  - 讓後驅輪 slip ratio 收斂在 **0.10~0.20**（直線）/ **0.08~0.12**（彎道）
  - 減少低 μ 路面打滑，提升 **0-60 km/h** 加速性能
  - 增加出彎穩定性（防過度 yaw）

1.2 ????

- 賽車應用：降低加速時間 20~30%（低 μ）
- **IMU** 自適應：彎道自動降低 slip 目標，平衡速度與穩定
- 即時反饋：10 Hz 控制頻率，<100 ms 響應延遲

---

2. ????????

2.1 ?????CAN ?????

訊號	符號	來源	單位	採樣率
前左輪速	(\omega_{FL})	CAN/WSM	rad/s	100 Hz
前右輪速	(\omega_{FR})	CAN/WSM	rad/s	100 Hz
後左輪速	(\omega_{RL})	CAN/WSM	rad/s	100 Hz
後右輪速	(\omega_{RR})	CAN/WSM	rad/s	100 Hz
縱向加速度	(a_x)	IMU	m/s²	100 Hz
橫向加速度	(a_y)	IMU	m/s²	100 Hz
偏航角速度	(\dot{\psi})	IMU	rad/s	100 Hz
馬達轉速	(\omega_m)	CAN/inverter	rpm	100 Hz
馬達命令扭矩	(T_{driver})	CAN/VCU	N·m	333Hz
馬達實際扭矩	(T_{actual})	CAN/inverter	N·m	100 Hz

2.2 ????????????

2.2.1 ???? (v) ?Low-pass + IMU Fusion?

前輪平均速度（原始）：

$$v_{wheel} = (\omega_{FL} + \omega_{FR}) / 2 \times R_{front}$$

IMU 縱向積分（漂移補正）：

$$v_{imu} = \int a_x \, dt \text{ (with high-pass filter)}$$

融合（Complementary Filter）：

$$v = \alpha \times v_{wheel} + (1 - \alpha) \times v_{imu}$$

其中 $\alpha = 0.8$（高速時多信任前輪，低速多信任 IMU）

防止低速除零：

$$v_{safe} = \max(v, 0.1 \text{ m/s})$$

Python 實作：

```
def estimate_vehicle_speed(omega_fl, omega_fr, a_x, R_front=0.32, alpha=0.8):
    """
    輸入: omega_fl, omega_fr (rad/s), a_x (m/s²)
    輸出: v_safe (m/s)
    """
    v_wheel = (omega_fl + omega_fr) / 2 * R_front
    # IMU 積分部分（假設已有高通濾波）
    v_imu = alpha_imu * v_imu_prev + (1 - alpha_imu) * a_x * dt
    v = alpha * v_wheel + (1 - alpha) * v_imu
    return max(v, 0.1)
```

2.2.2 ?? Slip Ratio (\lambda_L, \lambda_R)

```
定義（ISO 6954 標準）：
    λ_L = (ω_RL × R_rear - v) / max(v, 0.1)
    λ_R = (ω_RR × R_rear - v) / max(v, 0.1)

平均 Slip:
    λ¯ = (λ_L + λ_R) / 2

物理意義：
    λ = 0       : 完全滾動，無打滑
    0 < λ < 0.3 : 加速打滑（TC 目標區間）
    λ > 0.5     : 嚴重打滑（失控）
    λ_L ≠ λ_R   : 單輪打滑（可診斷直線穩定性）
```

Python 實作：

```
def compute_slip_ratio(omega_rl, omega_rr, v_safe, R_rear=0.32):
    """計算後輪 slip ratio"""
    slip_L = (omega_rl * R_rear - v_safe) / max(v_safe, 0.1)
    slip_R = (omega_rr * R_rear - v_safe) / max(v_safe, 0.1)
    slip_avg = (slip_L + slip_R) / 2
    return slip_L, slip_R, slip_avg
```

2.2.3 ?????IMU Yaw Rate?

```
直線 / 彎道判定閾值（可調，初始值推薦）：
    θ_straight = 5°/s  （直線）
    θ_light    = 15°/s （輕彎）
```

```
    θ_heavy    = 25°/s (重彎)


實現:
    if |ψ̇| < θ_straight:
        state = "STRAIGHT"
    elif θ_straight ≤ |ψ̇| < θ_light:
        state = "LIGHT_TURN"
    elif θ_light ≤ |ψ̇| < θ_heavy:
        state = "HEAVY_TURN"
    else:
        state = "SPIN_RISK"
```

Python 實作：

```
def detect_turn_state(psi_dot, thresholds=(5, 15, 25)):
    """
    輸入: psi_dot (rad/s, IMU yaw rate)
    輸出: state (str)
    """
    psi_dot_deg = abs(psi_dot) * 180 / 3.14159
    if psi_dot_deg < thresholds[0]:
        return "STRAIGHT"
    elif psi_dot_deg < thresholds[1]:
        return "LIGHT_TURN"
    elif psi_dot_deg < thresholds[2]:
        return "HEAVY_TURN"
    else:
        return "SPIN_RISK"
```

---

3. ????????

3.1 TC ?????

```
[油門輸入 T_driver]
        ↓
[後輪 Slip 計算] → λ¯ = (λ_L + λ_R) / 2
        ↓
```

```
[IMU Yaw Rate] → |ψ˙| → 選定 Target λ*
        ↓
[Slip Error] → e = λ* - λ¯
        ↓
[PI Controller] → ΔT = K_p × e + K_i × ∫e dt
        ↓
[扭矩限縮] → T_cmd = T_driver × (1 + ΔT)
        ↓
[馬達執行 & 保護邏輯]
        ↓
[實際扭矩送出]
```

3.2 PI ?????

3.2.1 Slip Error ??

```
e(t) = λ*(t) - λ¯(t)

e > 0  : slip 低於目標 → 要放寬扭矩（加油）
e < 0  : slip 高於目標 → 要限縮扭矩（減油）
```

3.2.2 ??? (Proportional)

```
ΔT_p = K_p × e

物理意義:
    - K_p 越大，反應越快（但容易超調）
    - K_p 越小，反應越慢（但更穩定）

推薦初值（賽車用）:
    直線: K_p = 3.0 ~ 5.0  (aggressive，想要快速加速)
    彎道: K_p = 2.0 ~ 3.0  (保守，防 spin)
    低μ:  K_p = 2.0 ~ 2.5  (中等)
```

物理直覺：如果打滑很嚴重（e = -0.3），用 K_p=5，會一次砍 150% 的扭矩（實際限縮到 0.5 倍），很快就能回到目標。

3.2.3 ??? (Integral)

```
ΔT_i = K_i × ∫e dt
```

物理意義：

- 消除穩態誤差（e 小但持久時逐漸累積）
- 低速起步用（此時 P 反應慢）

推薦初值：

K_i = 0.3 ~ 1.0

積分限制（Anti-windup）：

∫e dt 最大值 = 1.0 → 防止 I 項無限累積

3.2.4 ?? PI ???

```
ΔT(t) = K_p × e(t) + K_i × ∫₀ᵗ e(τ) dτ

T_cmd = T_driver × (1 + ΔT)  ; 原始扭矩限縮
T_cmd = max(0.3 × T_driver, min(T_driver, T_cmd))  ; 限縮範圍 [30%~100%]
```

Python 實作：

```
class PIController:
    def __init__(self, Kp=3.0, Ki=0.5, dt=0.01, anti_windup_max=1.0):
        self.Kp = Kp
        self.Ki = Ki
        self.dt = dt
        self.integral = 0.0
        self.anti_windup_max = anti_windup_max

    def update(self, error):
        """計算 PI 輸出"""
        # P 項
        p_term = self.Kp * error

        # I 項（含 anti-windup）
        self.integral += error * self.dt
        self.integral = max(-self.anti_windup_max,
                            min(self.anti_windup_max, self.integral))
        i_term = self.Ki * self.integral
```

```
        # 總輸出
        delta_T = p_term + i_term
        return delta_T

    def reset(self):
        """低速時重置積分"""
        self.integral = 0.0
```

3.3 ???????????

```
Step 1: 計算 Slip Error
    e = λ* - λ¯

Step 2: PI 控制器
    ΔT = Kp × e + Ki × ∫e dt

Step 3: 扭矩限縮計算
    T_cmd_raw = T_driver × (1 + ΔT)

Step 4: 飽和限縮 (Saturation)
    T_cmd_saturated = clamp(T_cmd_raw, 0.3 × T_driver, 1.0 × T_driver)

Step 5: 安全邏輯 (Safety Logic)

    if v < 3 km/h:  // 低速起步
        T_cmd = T_driver  // TC OFF, 用前輪參考加速

    elif λ_L > 0.4 or λ_R > 0.4:  // 單輪過度打滑
        T_cmd = 0.5 × T_driver  // 強制砍 50%

    elif state == "SPIN_RISK":  // IMU 偵測過度 yaw
        T_cmd = 0.3 × T_driver  // 進入失控防護

    else:
        T_cmd = T_cmd_saturated  // 正常 PI 控制
```

Python 實作：

```
def compute_torque_command(T_driver, slip_avg, slip_L, slip_R,
                           v, psi_dot, pi_controller,
                           turn_state, thresholds=(5, 15, 25)):
    """完整扭矩命令生成"""

    # Safety: 低速關閉 TC
    if v < 3 / 3.6:  # 3 km/h
        return T_driver, "LOW_SPEED"

    # Safety: 單輪過度打滑
    if abs(slip_L) > 0.4 or abs(slip_R) > 0.4:
        return 0.5 * T_driver, "SINGLE_WHEEL_SLIP"

    # Safety: 過度 yaw (失控風險)
    psi_dot_deg = abs(psi_dot) * 180 / 3.14159
    if psi_dot_deg > thresholds[2]:
        return 0.3 * T_driver, "SPIN_RISK"

    # 正常 TC 控制
    delta_T = pi_controller.update(error=slip_avg)
    T_cmd_raw = T_driver * (1 + delta_T)
    T_cmd = max(0.3 * T_driver, min(T_driver, T_cmd_raw))

    return T_cmd, turn_state
```

---

4. ??? Target Slip ??

4.1 Target Slip ????

根據 IMU yaw rate 和檔位即時調整 slip 目標:

```
┌──────────────────────────────────────────────────────┐
│          IMU Yaw Rate 與 Target Slip 自適應表          │
├──────────────────┬──────────────┬──────────┬─────────┤
│   駕駛狀態        │ yaw rate範圍 │ λ*       │ 控制策略  │
├──────────────────┼──────────────┼──────────┼─────────┤
│ 直線全油門加速    │ |ψ̇| < 5°/s  │ 0.18~0.20│ 最大縱向力 │
```

```
|                    |                | (Gear 1) | 0-60 最快      |
├────────────────────┼────────────────┼──────────┼────────────────┤
| 輕微轉向加速       | 5°/s ≤|ψ̇|   | 0.14~0.16| 平衡加速       |
|                    |      < 15°/s   | (Gear 2) | 與側向穩定     |
├────────────────────┼────────────────┼──────────┼────────────────┤
| 中等轉向 (彎道)    |15°/s ≤|ψ̇|   | 0.10~0.12| 優先穩定,      |
|                    |      < 25°/s   | (Gear 3) | 防 oversteer   |
├────────────────────┼────────────────┼──────────┼────────────────┤
| 急彎 / 失控預防    | |ψ̇| ≥ 25°/s | 0.08~0.10| 全力穩定,      |
|                    |                | (Gear 4) | 強制限縮       |
└────────────────────┴────────────────┴──────────┴────────────────┘
```

4.2 ????????

```
def compute_target_slip(psi_dot, v):
    """
    根據 yaw rate 和速度自適應計算 target slip

    輸入:
        psi_dot: IMU 偏航角速度 (rad/s)
        v: 車速 (m/s)

    輸出:
        lambda_target: 目標 slip ratio
        gear: 檔位標籤 (用於調試)
    """
    psi_dot_deg = abs(psi_dot) * 180 / 3.14159

    # 檔位切換 & Target Slip
    if psi_dot_deg < 5:
        lambda_target = 0.18 + 0.02 * (v / 20)  # 速度越高, target 稍高
        gear = "GEAR_1_MAX_ACCEL"

    elif psi_dot_deg < 15:
        # 線性插值: 5°/s 時 0.15, 15°/s 時 0.12
        lambda_target = 0.15 - 0.03 * (psi_dot_deg - 5) / 10
        gear = "GEAR_2_LIGHT_TURN"

    elif psi_dot_deg < 25:
```

```
        lambda_target = 0.10 + 0.02 * (25 - psi_dot_deg) / 10
        gear = "GEAR_3_HEAVY_TURN"

    else:  # Spin risk
        lambda_target = 0.08
        gear = "GEAR_4_SPIN_RISK"

    return lambda_target, gear
```

4.3 ? ? ???????

可選擴展：如果有簡易的摩擦力估測（例如從加速度飽和判斷），進一步調整：

```
def estimate_mu_level(a_x_max_recent, slip_avg_recent):
    """
    簡化的摩擦力等級估測


    邏輯：
    - 高 μ (乾柏油): a_x_max > 8 m/s², slip < 0.3
    - 中 μ (濕地):   a_x_max ≈ 4~6 m/s², slip 0.3~0.4
    - 低 μ (冰/雪):  a_x_max < 3 m/s², slip > 0.4
    """
    if a_x_max_recent > 8:
        return "HIGH_MU"
    elif a_x_max_recent > 5:
        return "MEDIUM_MU"
    else:
        return "LOW_MU"

def adjust_target_slip_for_mu(lambda_target, mu_level):
    """在摩擦力低時，進一步降低 target slip（保守）"""
    if mu_level == "LOW_MU":
        return lambda_target * 0.85  # 降低 15%
    elif mu_level == "MEDIUM_MU":
        return lambda_target * 0.92  # 降低 8%
    else:
        return lambda_target
```

---

5. ?????????

5.1 ?????????

地面類型	路面特徵	**μ** 等級	場地要求
高 **μ**	乾燥柏油、新鋪	μ ≈ 0.85~1.0	平坦，無雨
中 **μ**	濕柏油、一般水泥	μ ≈ 0.6~0.75	濕潤但不積水
低 **μ**	碎石、濕草地、冰	μ ≈ 0.3~0.5	或人為灑水

5.2 ???? SOP?Standard Operating Procedure?

5.2.1 ??????15 ???

□ 輪胎檢查 □ 胎壓一致（左右前後差異 < 0.1 bar） □ 胎溫 20~30°C（冷胎，確保可重複性） □ 胎面無異物

□ CAN Logger 配置 □ 記錄頻率設定： 100 Hz (wheel speed, IMU) □ CAN ID 表確認無誤 □ 存儲空間檢查（預期每次 5~10 MB）

□ 馬達 / 油門校準 □ 油門全開 -> T_driver max 確認 □ 油門全關 -> T_driver = 0 確認

□ 天氣記錄 □ 溫度、溼度、風速 □ 路面溫度（紅外溫度計）

5.2.2 Baseline ?? - TC OFF?3 ??

```
場景： 直線加速，全油門 0~60 km/h


每趟流程：
  1. 車停在起點，冷靜等待 (IMU 零點漂移穩定)
  2. 油門緩慢踩到 50% (預熱)
  3. 在 v = 0 瞬間，全油門 (t=0 記點)
  4. 保持全油門直到 60 km/h，然後收油
  5. 停車 5 分鐘，冷卻胎溫


數據記錄：
  ├─ 起點位置 (GPS)
  ├─ 環境溫度、胎溫
  ├─ CAN 原始數據 (導出為 CSV)
  └─ 駕駛備註 (路面狀況、車感描述)
```

```
觀察重點:
  ├─ 最大 slip ratio: λ_max ?
  ├─ 0~60 km/h 時間: t_60 ?
  ├─ 打滑持續時間 (λ > 0.3): t_slip ?
  └─ 前後輪 slip 差異: |λ_L - λ_R| ?
```

5.2.3 TC ON - ?????3 ??

```
場景: 同上, 但啟動 TC (Target λ* = 0.18)

參數設定:
  ├─ Kp = 3.5 (aggressive)
  ├─ Ki = 0.5
  └─ 彎道檢測閾值: θ_straight = 5°/s

觀察重點 (對比 Baseline):
  ├─ 最大 slip ratio 是否 < 0.25?
  ├─ 0~60 km/h 時間是否縮短 (預期 -15~25%)?
  ├─ 縱向加速度是否更平順 (std 是否降低)?
  └─ 打滑時間是否 < 1 s?

數據分析:
  - 畫圖: slip(t) 對比 (OFF 紅、ON 藍)
  - 計算: 平均縱向 g (m/s²)
```

5.2.4 TC ON - ?????3 ???????

```
場景: 轉向加速 (方向盤 ≈ 30°) + 全油門, 通過彎道

參數設定:
  ├─ Kp = 2.5 (moderate)
  ├─ Ki = 0.5
  └─ 自適應 target slip (根據 |ψ̇|)

觀察重點:
  ├─ 出彎 yaw rate 峰值: max(|ψ̇|)?
  ├─ Oversteer 徵兆 (yaw rate 尖峰)
  ├─ 加速度 (a_x, a_y) 是否均衡
  └─ slip 是否穩定在 0.10~0.12?
```

5.2.5 TC ON - ? ? ?????5 ??

```
場景: 低 μ 地面 (濕碎石), 全油門起步


關鍵觀察:
  ├─ 起步時打滑時間 (λ > 0.3 持續多久)?
  ├─ 穩定後的 slip 值是否在目標範圍?
  ├─ 連續 5 次起步中胎溫上升對 slip 的影響
  └─ 預期: 打滑時間逐次縮短 (胎溫升高 μ 增加)
```

5.3 ?????????????

優先級	地面	TC 狀態	彎度	趟數	預期時間	備註
1	高 μ	OFF	直線	3	10 min	Baseline
2	高 μ	ON	直線	3	10 min	主要對比
3	中 μ	OFF	直線	3	10 min	
4	中 μ	ON	直線	3	10 min	
5	低 μ	OFF	直線	3	10 min	最難
6	低 μ	ON	直線	5	15 min	重點
7	高 μ	ON	彎 30°	3	15 min	穩定性
8	低 μ	ON	彎 30°	3	15 min	進階
完	-	-	-	-	**100 min**	≈ 半天

---

6. ?????

6.1 ???????????

在每趟實驗後, 用此表快速檢查數據品質:

```
### 6.1.1 CAN 數據完整性


測試編號: _______  日期: _______  地面條件: _______


☐ 訊號檢查
  ☐ 前左輪速: < 2 s 無跳動? Y / N
```

- ☐ 前右輪速: < 2 s 無跳動? Y / N
- ☐ 後左輪速: < 2 s 無跳動? Y / N
- ☐ 後右輪速: < 2 s 無跳動? Y / N
- ☐ IMU 加速度: 在合理範圍 (< 2g, < 1g)? Y / N
- ☐ IMU 角速度: max(|ψ̇|) < 100°/s? Y / N
- ☐ 馬達扭矩: 在 [0, T_max] 範圍? Y / N

☐ 時間同步

- ☐ 所有訊號長度一致（同一時間範圍）? Y / N
- ☐ 無掉幀或時間戳異常? Y / N

☐ 統計檢查

- ☐ 車速: 0 ~ 60 km/h 正常遞增? Y / N
- ☐ Slip ratio: 0 ~ 0.5 正常變化? Y / N
- ☐ 加速度: 0~1 g, 無 spike > 3g? Y / N

6.2 ????????Python ??

6.2.1 ?????

```
def verify_data_quality(df_log):
    """
    輸入: DataFrame with 'time', 'v', 'slip_L', 'slip_R', 'a_x', 'psi_dot', 'T_cmd'
    輸出: 品質報告
    """
    print("=" * 60)
    print("DATA QUALITY REPORT")
    print("=" * 60)

    # 訊號範圍檢查
    checks = {
        "v (m/s)": (df_log['v'].min(), df_log['v'].max(), "0 ~ 25"),
        "slip_L": (df_log['slip_L'].min(), df_log['slip_L'].max(), "-0.2 ~ 0.6"),
        "slip_R": (df_log['slip_R'].min(), df_log['slip_R'].max(), "-0.2 ~ 0.6"),
        "a_x (m/s²)": (df_log['a_x'].min(), df_log['a_x'].max(), "-1 ~ 2"),
        "psi_dot (°/s)": (df_log['psi_dot'].min() * 180/3.14,
                          df_log['psi_dot'].max() * 180/3.14, "-50 ~ 50"),
        "T_cmd (N·m)": (df_log['T_cmd'].min(), df_log['T_cmd'].max(), "0 ~ T_max"),
    }
```

```
    for signal, (v_min, v_max, range_str) in checks.items():
        print(f"{signal:20} | Min: {v_min:8.3f} | Max: {v_max:8.3f} | Range: {range_str}")

    # 異常檢測
    print("\n" + "=" * 60)
    print("ANOMALY DETECTION")
    print("=" * 60)

    # 檢查加速度 spike
    a_x_spike = (df_log['a_x'].diff().abs() > 0.5).sum()  # 0.5 m/s² 變化
    print(f"Acceleration spikes (Δa > 0.5 m/s²): {a_x_spike} frames")

    # 檢查 slip 不連續
    slip_jump = ((df_log['slip_L'].diff().abs() > 0.1).sum() +
                 (df_log['slip_R'].diff().abs() > 0.1).sum())
    print(f"Slip discontinuities (Δλ > 0.1): {slip_jump} frames")

    # 檢查時間戳
    dt = df_log['time'].diff().dropna()
    dt_expected = 0.01  # 100 Hz
    dt_anomaly = (dt[dt > 1.5 * dt_expected]).count()
    print(f"Timestamp gaps (Δt > 15ms): {dt_anomaly} frames")
```

6.2.2 ??????

```
def compute_performance_metrics(df_log, tc_status="ON"):
    """計算實驗的性能指標"""
    print(f"\n{'='*60}")
    print(f"PERFORMANCE METRICS (TC {tc_status})")
    print(f"{'='*60}\n")

    # 1. 加速性能
    v_0_to_60_mask = (df_log['v'] >= 0) & (df_log['v'] <= 60/3.6)
    if v_0_to_60_mask.sum() > 0:
        t_0_60 = df_log[v_0_to_60_mask]['time'].max() - df_log[v_0_to_60_mask]['time'].min()
        print(f"[1] Acceleration 0~60 km/h: {t_0_60:.2f} s")

    # 2. 打滑統計
    slip_avg = (df_log['slip_L'] + df_log['slip_R']) / 2
```

```
    slip_high_mask = slip_avg > 0.25
    t_slip_high = (slip_high_mask.sum()) * 0.01  # 100 Hz
    print(f"[2] High slip time (λ > 0.25): {t_slip_high:.2f} s")
    print(f"    Max slip: {slip_avg.max():.3f}")
    print(f"    Mean slip (steady): {slip_avg[v_0_to_60_mask].mean():.3f}")

    # 3. □□□□□
    a_x_mean = df_log['a_x'].mean()
    a_x_std = df_log['a_x'].std()
    print(f"[3] Longitudinal acceleration: {a_x_mean:.2f} ± {a_x_std:.2f} m/s²")

    # 4. □□□□□□□□□□□□
    if df_log['psi_dot'].abs().max() > 5 * 3.14159 / 180:  # > 5°/s
        psi_dot_peak = df_log['psi_dot'].abs().max() * 180 / 3.14159
        print(f"[4] Peak yaw rate: {psi_dot_peak:.1f} °/s")

    # 5. □□□□
    print(f"[5] Torque limiting:")
    print(f"    Max commanded: {df_log['T_cmd'].max():.0f} N·m")
    print(f"    Avg limited: {(df_log['T_cmd'] < df_log['T_driver'] * 0.95).mean() *
100:.1f}%")

    return {
        't_0_60': t_0_60 if v_0_to_60_mask.sum() > 0 else None,
        'slip_max': slip_avg.max(),
        'slip_mean': slip_avg.mean(),
        'a_x_mean': a_x_mean,
        'a_x_std': a_x_std,
    }
```

6.2.3 ?????

```
def compare_tc_on_off(df_log_off, df_log_on):
    """□□ TC OFF vs ON □□□"""
    print("\n" + "="*80)
    print("COMPARISON: TC OFF vs ON")
    print("="*80 + "\n")

    m_off = compute_performance_metrics(df_log_off, "OFF")
    m_on = compute_performance_metrics(df_log_on, "ON")
```

```
# □□
import pandas as pd
comparison = pd.DataFrame({
    'Metric': [
        '0~60 km/h Time (s)',
        'Max Slip Ratio',
        'Mean Slip Ratio',
        'Longitudinal g (mean)',
        'Acceleration Smoothness (std)',
    ],
    'TC OFF': [
        f"{m_off['t_0_60']:.2f}" if m_off['t_0_60'] else "—",
        f"{m_off['slip_max']:.3f}",
        f"{m_off['slip_mean']:.3f}",
        f"{m_off['a_x_mean']:.2f}",
        f"{m_off['a_x_std']:.2f}",
    ],
    'TC ON': [
        f"{m_on['t_0_60']:.2f}" if m_on['t_0_60'] else "—",
        f"{m_on['slip_max']:.3f}",
        f"{m_on['slip_mean']:.3f}",
        f"{m_on['a_x_mean']:.2f}",
        f"{m_on['a_x_std']:.2f}",
    ],
})

# □□□□□□□
if m_off['t_0_60'] and m_on['t_0_60']:
    t_improve = (m_off['t_0_60'] - m_on['t_0_60']) / m_off['t_0_60'] * 100
    comparison.loc[0, 'Improvement %'] = f"{t_improve:+.1f}%"

slip_improve = (m_off['slip_max'] - m_on['slip_max']) / m_off['slip_max'] * 100
comparison.loc[1, 'Improvement %'] = f"{slip_improve:+.1f}%"

print(comparison.to_string(index=False))
print("\n")
```

6.3 ???????? Script

```
import pandas as pd
import matplotlib.pyplot as plt

def plot_tc_analysis(df_log, save_path="tc_analysis.png"):
    """□□□□□□□□□□"""
    fig, axes = plt.subplots(2, 2, figsize=(14, 8))

    # Subplot 1: □□ & Slip
    ax1 = axes[0, 0]
    ax1.plot(df_log['time'], df_log['v'] * 3.6, 'b-', label='Vehicle Speed', linewidth=2)
    ax1_slip = ax1.twinx()
    ax1_slip.plot(df_log['time'], (df_log['slip_L'] + df_log['slip_R'])/2, 'r--',
                  label='Avg Slip Ratio', linewidth=2)
    ax1.set_xlabel('Time (s)')
    ax1.set_ylabel('Speed (km/h)', color='b')
    ax1_slip.set_ylabel('Slip Ratio', color='r')
    ax1.grid(True, alpha=0.3)
    ax1.set_title('Vehicle Speed & Slip Ratio vs Time')

    # Subplot 2: □□□ Slip □□
    ax2 = axes[0, 1]
    ax2.plot(df_log['time'], df_log['slip_L'], 'g-', label='Left Wheel', linewidth=2)
    ax2.plot(df_log['time'], df_log['slip_R'], 'orange', label='Right Wheel', linewidth=2)
    ax2.axhline(y=0.1, color='r', linestyle='--', alpha=0.5, label='Target (0.1)')
    ax2.axhline(y=0.2, color='r', linestyle='--', alpha=0.5, label='Target (0.2)')
    ax2.set_xlabel('Time (s)')
    ax2.set_ylabel('Slip Ratio')
    ax2.legend()
    ax2.grid(True, alpha=0.3)
    ax2.set_title('Left vs Right Wheel Slip')

    # Subplot 3: □□□□□
    ax3 = axes[1, 0]
    ax3.plot(df_log['time'], df_log['a_x'], 'purple', linewidth=2)
    ax3.set_xlabel('Time (s)')
    ax3.set_ylabel('Acceleration (m/s²)')
    ax3.grid(True, alpha=0.3)
    ax3.set_title(f'Longitudinal Acceleration (Mean: {df_log["a_x"].mean():.2f}, Std:
{df_log["a_x"].std():.2f})')
```

```
    # Subplot 4: 扭矩命令
    ax4 = axes[1, 1]
    ax4.plot(df_log['time'], df_log['T_driver'], 'b-', label='T_driver', linewidth=2)
    ax4.plot(df_log['time'], df_log['T_cmd'], 'r-', label='T_cmd (limited)', linewidth=2)
    ax4.set_xlabel('Time (s)')
    ax4.set_ylabel('Torque (N·m)')
    ax4.legend()
    ax4.grid(True, alpha=0.3)
    ax4.set_title('Motor Torque: Driver Input vs TC Command')

    plt.tight_layout()
    plt.savefig(save_path, dpi=150)
    print(f"Figure saved to {save_path}")
    return fig
```

7. ?????

7.1 ??? Python ?????

```
"""
TC 開發主程序框架 (Python 3.8+)
用途: 離線分析 CAN 數據，測試控制邏輯
"""

import pandas as pd
import numpy as np
from pathlib import Path
import matplotlib.pyplot as plt

class TCController:
    def __init__(self, Kp=3.0, Ki=0.5, R_rear=0.32, R_front=0.32):
        self.Kp = Kp
        self.Ki = Ki
        self.R_rear = R_rear
        self.R_front = R_front
        self.integral = 0.0
        self.dt = 0.01  # 100 Hz
```

```
def estimate_vehicle_speed(self, omega_fl, omega_fr, a_x, alpha=0.8):
    """車速估測 (Complementary Filter)"""
    v_wheel = (omega_fl + omega_fr) / 2 * self.R_front
    return max(v_wheel, 0.1)

def compute_slip_ratio(self, omega_rl, omega_rr, v_safe):
    """計算後輪 slip ratio"""
    slip_L = (omega_rl * self.R_rear - v_safe) / max(v_safe, 0.1)
    slip_R = (omega_rr * self.R_rear - v_safe) / max(v_safe, 0.1)
    return slip_L, slip_R, (slip_L + slip_R) / 2

def compute_target_slip(self, psi_dot):
    """自適應 target slip"""
    psi_dot_deg = abs(psi_dot) * 180 / 3.14159
    if psi_dot_deg < 5:
        return 0.18
    elif psi_dot_deg < 15:
        return 0.15 - 0.03 * (psi_dot_deg - 5) / 10
    else:
        return 0.10

def pi_control(self, slip_error):
    """PI 控制器"""
    p_term = self.Kp * slip_error
    self.integral = max(-1.0, min(1.0, self.integral + slip_error * self.dt))
    i_term = self.Ki * self.integral
    return p_term + i_term

def compute_torque_command(self, T_driver, slip_avg, slip_L, slip_R,
                           v, psi_dot):
    """完整扭矩命令生成"""
    # 低速關閉 TC
    if v < 3 / 3.6:
        return T_driver, "LOW_SPEED"

    # 單輪過度打滑
    if abs(slip_L) > 0.4 or abs(slip_R) > 0.4:
        return 0.5 * T_driver, "SINGLE_WHEEL_SLIP"
```

```
        # 過度 yaw
        psi_dot_deg = abs(psi_dot) * 180 / 3.14159
        if psi_dot_deg > 25:
            return 0.3 * T_driver, "SPIN_RISK"

        # 正常 TC
        lambda_target = self.compute_target_slip(psi_dot)
        error = lambda_target - slip_avg
        delta_T = self.pi_control(error)
        T_cmd = T_driver * (1 + delta_T)
        T_cmd = max(0.3 * T_driver, min(T_driver, T_cmd))

        return T_cmd, f"NORMAL (λ*={lambda_target:.2f})"

def main():
    # 1. 讀取 CAN 數據
    log_file = Path("tc_test_20260313.csv")  # 修改為你的文件
    df_raw = pd.read_csv(log_file)

    # 2. 初始化控制器
    tc = TCController(Kp=3.5, Ki=0.5)

    # 3. 處理數據
    df_processed = df_raw.copy()
    df_processed['v'] = df_raw.apply(
        lambda row: tc.estimate_vehicle_speed(row['omega_fl'], row['omega_fr'], row['a_x']),
        axis=1
    )

    df_processed[['slip_L', 'slip_R', 'slip_avg']] = df_raw.apply(
        lambda row: pd.Series(tc.compute_slip_ratio(row['omega_rl'], row['omega_rr'],
                                                    df_processed.loc[row.name, 'v'])),
        axis=1
    )

    # 4. 計算 TC 命令
    df_processed[['T_cmd', 'tc_status']] = df_raw.apply(
        lambda row: pd.Series(tc.compute_torque_command(
```

```
            row['T_driver'], df_processed.loc[row.name, 'slip_avg'],
            df_processed.loc[row.name, 'slip_L'], df_processed.loc[row.name, 'slip_R'],
            df_processed.loc[row.name, 'v'], row['psi_dot']
        )),
        axis=1
    )

    # 5. 性能分析
    print("Performance Analysis Complete!")

    # 6. 生成報告
    plot_tc_analysis(df_processed, save_path="tc_analysis_result.png")
    df_processed.to_csv("tc_processed_data.csv", index=False)

if __name__ == "__main__":
    main()
```

7.2 STM32 CubeIDE C ?????

```
/*
 * TC_Controller.h
 * STM32 CAN 實時 TC 控制模組
 */

#ifndef TC_CONTROLLER_H
#define TC_CONTROLLER_H

#include "stdint.h"
#include "stdbool.h"

// 結構體定義
typedef struct {
    float omega_fl, omega_fr, omega_rl, omega_rr;  // rad/s
    float a_x, a_y, psi_dot;  // IMU
    float T_driver, T_cmd;     // N·m
} TCInput_t;

typedef struct {
    float v;            // m/s
```

```
    float slip_L, slip_R, slip_avg;
    float lambda_target;
    float T_cmd_final;
    uint8_t tc_status;  // 0=OFF, 1=NORMAL, 2=SAFETY
} TCOutput_t;

typedef struct {
    float Kp, Ki;
    float integral;
    float R_rear, R_front;
    float dt;
} TCController_t;

// □□□□
void TC_Init(TCController_t *ctrl, float Kp, float Ki);
void TC_Update(TCController_t *ctrl, TCInput_t *input, TCOutput_t *output);

#endif

/**
 * TC_Controller.c - □□
 */

void TC_Init(TCController_t *ctrl, float Kp, float Ki) {
    ctrl->Kp = Kp;
    ctrl->Ki = Ki;
    ctrl->integral = 0.0f;
    ctrl->R_rear = 0.32f;
    ctrl->R_front = 0.32f;
    ctrl->dt = 0.01f;  // 100 Hz
}

static float estimate_vehicle_speed(float omega_fl, float omega_fr, float R_front) {
    float v = ((omega_fl + omega_fr) / 2.0f) * R_front;
    return (v > 0.1f) ? v : 0.1f;
}

static void compute_slip_ratio(float omega_rl, float omega_rr, float v, float R_rear,
                               float *slip_L, float *slip_R, float *slip_avg) {
    float v_safe = (v > 0.1f) ? v : 0.1f;
```

```
    *slip_L = ((omega_rl * R_rear) - v_safe) / v_safe;
    *slip_R = ((omega_rr * R_rear) - v_safe) / v_safe;
    *slip_avg = (*slip_L + *slip_R) / 2.0f;
}

static float compute_target_slip(float psi_dot_deg) {
    if (psi_dot_deg < 5.0f) {
        return 0.18f;
    } else if (psi_dot_deg < 15.0f) {
        return 0.15f - 0.03f * (psi_dot_deg - 5.0f) / 10.0f;
    } else {
        return 0.10f;
    }
}

void TC_Update(TCController_t *ctrl, TCInput_t *input, TCOutput_t *output) {
    // 1. □□□□
    output->v = estimate_vehicle_speed(input->omega_fl, input->omega_fr, ctrl->R_front);

    // 2. Slip ratio □□
    compute_slip_ratio(input->omega_rl, input->omega_rr, output->v, ctrl->R_rear,
                       &output->slip_L, &output->slip_R, &output->slip_avg);

    // 3. Target slip □□□
    float psi_dot_deg = fabsf(input->psi_dot) * 180.0f / 3.14159f;
    output->lambda_target = compute_target_slip(psi_dot_deg);

    // 4. □□□□ (Safety First)
    output->tc_status = 1;  // NORMAL

    if (output->v < 0.83f) {  // < 3 km/h
        output->T_cmd_final = input->T_driver;
        output->tc_status = 0;  // OFF
    }
    else if (fabsf(output->slip_L) > 0.4f || fabsf(output->slip_R) > 0.4f) {
        output->T_cmd_final = input->T_driver * 0.5f;
        output->tc_status = 2;  // SAFETY
    }
    else if (psi_dot_deg > 25.0f) {
        output->T_cmd_final = input->T_driver * 0.3f;
```

```
        output->tc_status = 2;  // SAFETY
    }
    else {
        // PI 控制
        float error = output->lambda_target - output->slip_avg;
        float p_term = ctrl->Kp * error;

        ctrl->integral += error * ctrl->dt;
        if (ctrl->integral > 1.0f) ctrl->integral = 1.0f;
        if (ctrl->integral < -1.0f) ctrl->integral = -1.0f;
        float i_term = ctrl->Ki * ctrl->integral;

        float delta_T = p_term + i_term;
        float T_raw = input->T_driver * (1.0f + delta_T);

        // 飽和限縮
        float T_min = input->T_driver * 0.3f;
        float T_max = input->T_driver * 1.0f;
        output->T_cmd_final = (T_raw > T_max) ? T_max : ((T_raw < T_min) ? T_min : T_raw);
    }

    output->T_cmd = output->T_cmd_final;
}
```

?? A: ???????

Q1: Slip ratio ?????> 0.5???????

原因可能　　：

- 輪胎半徑　　R 估測錯誤
- 前輪打滑（後驅時不應該發生，除非彎度很大）
- 輪速感測器信號雜訊大

排查：

1. 用非驅動輪直線加速，手工驗證　　R 值
2. 比較　FL/FR 輪速，差異　　> 5% 表示有問題
3. 加低通濾波到輪速訊號

Q2: TC ??????????

原因可能：

- Target slip 設定太低
- Kp/Ki 過大，過度限縮

解決：

1. 逐次提高 target slip (0.15 → 0.18 → 0.20)，找最快點
2. 降低 Kp (5.0 → 3.5 → 2.5)，觀察 slip 波動

Q3: ??????? yaw?????

原因：

- Target slip 在彎道時未降低
- IMU yaw rate 閾值設定不當

解決：

1. 確認自適應邏輯正確執行（加 log 輸出）
2. 提高彎道 target slip 降幅（0.15 → 0.08）

?? B: ??????

1. 路面摩擦力估測：從加速度飽和點推估 μ，動態調整 target slip
2. 單輪扭矩分配：左右驅動輪獨立控制，改善彎道穩定性
3. **Neural Network** 增益：結合你之前研究的 observer，用 NN 預測最佳 Kp/Ki
4. 實時濾波優化：應用 Kalman filter 融合前輪和 IMU 速度估測

> 最後建議：先用 Python 在桌上測試演算法邏輯（週 1-2），確認無誤後移植到 STM32（週 3-4），場測時邊跑邊調參數，逐次優化。祝你實驗順利！□□□□

Revision #2
Created 2026-04-01 02:06:27 UTC by TaipeiTechRacing
Updated 2026-04-06 16:39:15 UTC