?????

- TC □□□□ ___
- TC STM32F446 - Code
- PID □□□□□□□□□ _____ MATLAB □□ __
- MATLAB □□□□□□□ _____ (Basic Syntax Cheat Sheet)
- MATLAB App Designer (□□□□□ ____) □□□□□ ____
- MATLAB Plot □□□□□ ____ (□□ __ Simulink □□□□ ___)
- MATLAB-Simulink □□□□□□□□□□ _____
- □□□□ ___ TC □□□□□ ____ STM32F446 □□□□ ___
- □□□□ ___ Firmware
- RWD TC □□□□□□□□□□□□□□□ _____

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] → |ψ̇| → 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 ? ? ???????

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
def estimate_mu_level(a_x_max_recent, slip_avg_recent):
    """
    □□□□□□□□□□

    □□□
    - □ μ (□□□): a_x_max > 8 m/s², slip < 0.3
    - □ μ (□□):   a_x_max ≈ 4~6 m/s², slip 0.3~0.4
    - □ μ (□/□):  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):
    """□□□□□□□□□□□□ target slip□□□□"""
    if mu_level == "LOW_MU":
        return lambda_target * 0.85  # □□ 15%
    elif mu_level == "MEDIUM_MU":
        return lambda_target * 0.92  # □□ 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！ □□□

TC STM32F446 - Code

tags: `Traction Control` `RWD` `STM32F446` `CAN` `Steering Angle`

---

0. ???????

0.1 ??

- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□ STM32F446□ 180 MHz, FPU□□
- □□□
  - □□□□ + IMU + □□□□□ + □□□□ → □□□□□□ slip□
  - □□ □□□□□□□ □ Target Slip□□□ IMU □□□□
  - PI □□□□□□□□□□□□□□□□□□
  - □□□□ □□□ / □ 8 / □□□□□□□□□□□□□□□
  - □□□□□□□ → TC □□□□□ Fail-safe□□
  - □□□□□□□ Anti-windup□□
  - □□□□ μ □□ → □□□ Kp□

0.2 ??????????

- `Core/Src/main.c`□□□□□□□ TIM/CAN □□□
- `Core/Inc/tc_types.h`□□□□□□□□ enum□
- `Core/Inc/tc_controller.h` / `Core/Src/tc_controller.c`□ TC □□□□□□
- `Core/Inc/tc_fault.h` / `Core/Src/tc_fault.c`□□□□□□□□ & □□□□□
- `Core/Inc/tc_mode.h` / `Core/Src/tc_mode.c`□□□□□□□□ /□ 8/□□□□
- `Core/Inc/tc_io.h` / `Core/Src/tc_io.c`□ CAN □□ / □□ + □□□□□
- `Core/Inc/tc_params.h`□□□□□□ Kp, Ki, slip □□□□□

---

1. ??????????tc_types.h?

```
#ifndef TC_TYPES_H
#define TC_TYPES_H


#include "stdint.h"
```

```
#include "stdbool.h"

/* ---- 模式 ---- */
typedef enum {
    TC_MODE_STRAIGHT = 0,  // 直線加速
    TC_MODE_FIGURE8  = 1,  // 繞 8 / skidpad
    TC_MODE_TRACK    = 2   // 賽道
} TC_Mode_t;

/* ---- TC 狀態 ---- */
typedef enum {
    TC_STATUS_OFF      = 0,  // 關閉
    TC_STATUS_NORMAL   = 1,  // 正常控制
    TC_STATUS_SAFETY   = 2,  // 安全限扭
    TC_STATUS_FAULT    = 3   // 錯誤關閉
} TC_Status_t;

/* ---- 錯誤旗標 ---- */
typedef enum {
    FAULT_NONE         = 0,
    FAULT_WHEEL        = 1 << 0,
    FAULT_IMU          = 1 << 1,
    FAULT_STEER        = 1 << 2,
    FAULT_CAN_TIMEOUT  = 1 << 3,
    FAULT_TORQUE       = 1 << 4
} TC_FaultFlags_t;

/* ---- 單次迴圈輸入（10 ms） ---- */
typedef struct {
    /* 原始感測器 + 指令（從 CAN 或 ADC 來） */
    float omega_fl, omega_fr;     // 前輪輪速 rad/s
    float omega_rl, omega_rr;     // 後輪輪速 rad/s
    float ax_imu;                 // 縱向加速度 m/s2
    float ay_imu;                 // 橫向加速度（目前可不必用）
    float psi_dot;                // yaw rate rad/s
    float steer_deg_raw;          // 方向盤轉角 deg（未濾波）
    float T_driver;               // 駕駛要求扭矩 Nm

    /* 系統時間（ms） */
    uint32_t timestamp_ms;
```

```
} TC_Input_t;

/* ---- 單次迴圈輸出 ---- */
typedef struct {
    float v_mps;                  // 車速 m/s
    float slip_L, slip_R, slip_avg;
    float steer_deg_f;            // 濾波後方向盤 deg
    float lambda_target;          // 目標 slip
    float delta_T;                // PI 輸出比例（無單位）
    float T_cmd;                  // 實際下發扭矩 Nm
    TC_Status_t tc_status;
    uint32_t fault_flags;
} TC_Output_t;

#endif
```

2. ??????????tc_params.h?

```
#ifndef TC_PARAMS_H
#define TC_PARAMS_H

/* 輪胎半徑 */
#define R_FRONT        0.32f
#define R_REAR         0.32f

/* TC 主迴圈週期 */
#define TC_DT_SEC      0.01f  // 10 ms, 100 Hz

/* 方向盤濾波係數（低通） */
#define STEER_ALPHA    0.92f  // 越接近 1 越慢，約 3 Hz

/* Slip 安全閾值 */
#define SLIP_SINGLE_MAX 0.40f   // 單輪 > 0.4 視為嚴重打滑

/* 低速關閉 TC 門檻 */
#define V_TC_OFF_KMH   3.0f

/* 積分上限（Anti-windup） */
```

```
#define INTEGRAL_MAX   1.0f

/* ---- 三種模式基線參數 ---- */

/* 直線模式 */
#define KP_STRAIGHT    4.2f
#define KI_STRAIGHT    0.35f
#define LAMBDA_BASE_STRAIGHT  0.20f
#define STEER_SENS_STRAIGHT   0.4f   // 方向盤敏感度較低
#define MIN_TQ_RATIO_STRAIGHT 0.55f  // 最小 55% 扭矩

/* 繞 8 模式 */
#define KP_FIGURE8     3.1f
#define KI_FIGURE8     0.42f
#define LAMBDA_BASE_F8 0.16f
#define STEER_SENS_F8  1.0f
#define MIN_TQ_RATIO_F8 0.40f

/* 賽道模式 */
#define KP_TRACK       2.8f
#define KI_TRACK       0.50f
#define LAMBDA_BASE_TRACK 0.14f
#define STEER_SENS_TRACK  1.3f
#define MIN_TQ_RATIO_TRACK 0.30f

#endif
```

---

3. ?????tc_mode.h / tc_mode.c?

3.1 ????tc_mode.h?

```
#ifndef TC_MODE_H
#define TC_MODE_H

#include "tc_types.h"
#include "tc_params.h"
```

```
typedef struct {
    TC_Mode_t current_mode;   // 目前實際模式
    TC_Mode_t target_mode;    // 車手選擇的模式
    bool      change_pending; // 是否等待靜止後切換

    float Kp, Ki;
    float lambda_base;
    float steer_sens;
    float min_tq_ratio;       // 最低扭矩比例
} TC_ModeState_t;

void TC_Mode_Init(TC_ModeState_t *m);
void TC_Mode_RequestChange(TC_ModeState_t *m, TC_Mode_t new_mode, float v_kmh);
void TC_Mode_ApplyIfStopped(TC_ModeState_t *m, float v_kmh);

#endif
```

3.2 ???tc_mode.c?

```
#include "tc_mode.h"

static void TC_Mode_ApplyParams(TC_ModeState_t *m, TC_Mode_t mode) {
    m->current_mode = mode;

    switch (mode) {
    case TC_MODE_STRAIGHT:
        m->Kp          = KP_STRAIGHT;
        m->Ki          = KI_STRAIGHT;
        m->lambda_base = LAMBDA_BASE_STRAIGHT;
        m->steer_sens  = STEER_SENS_STRAIGHT;
        m->min_tq_ratio= MIN_TQ_RATIO_STRAIGHT;
        break;

    case TC_MODE_FIGURE8:
        m->Kp          = KP_FIGURE8;
        m->Ki          = KI_FIGURE8;
        m->lambda_base = LAMBDA_BASE_F8;
        m->steer_sens  = STEER_SENS_F8;
        m->min_tq_ratio= MIN_TQ_RATIO_F8;
```

```
        break;

    case TC_MODE_TRACK:
    default:
        m->Kp          = KP_TRACK;
        m->Ki          = KI_TRACK;
        m->lambda_base = LAMBDA_BASE_TRACK;
        m->steer_sens  = STEER_SENS_TRACK;
        m->min_tq_ratio= MIN_TQ_RATIO_TRACK;
        break;
    }
}

void TC_Mode_Init(TC_ModeState_t *m) {
    m->current_mode   = TC_MODE_FIGURE8;   // 建議預設繞 8 模式
    m->target_mode    = m->current_mode;
    m->change_pending = false;
    TC_Mode_ApplyParams(m, m->current_mode);
}

/* 車手按按鈕後呼叫: 若靜止就立即切, 若在跑就設成 pending */
void TC_Mode_RequestChange(TC_ModeState_t *m, TC_Mode_t new_mode, float v_kmh) {
    m->target_mode = new_mode;

    if (v_kmh < 1.0f) { // < 1 km/h 視為靜止, 直接切
        TC_Mode_ApplyParams(m, new_mode);
        m->change_pending = false;
    } else {
        m->change_pending = true;
    }
}

/* 每次迴圈呼叫: 若 pending 且已靜止 → 套用新模式 */
void TC_Mode_ApplyIfStopped(TC_ModeState_t *m, float v_kmh) {
    if (m->change_pending && v_kmh < 1.0f) {
        TC_Mode_ApplyParams(m, m->target_mode);
        m->change_pending = false;
    }
}
```

4. ??????????tc_fault.h / tc_fault.c?

4.1 ????tc_fault.h?

```
#ifndef TC_FAULT_H
#define TC_FAULT_H

#include "tc_types.h"

typedef struct {
    uint32_t flags;
    uint32_t wheel_err_count;
    uint32_t imu_err_count;
    uint32_t steer_err_count;
    uint32_t torque_err_count;
    uint32_t can_timeout_count;

    float    omega_prev[4];
    float    steer_prev;
} TC_FaultState_t;

void TC_Fault_Init(TC_FaultState_t *f);
void TC_Fault_Update(const TC_Input_t *in, TC_FaultState_t *f);
bool TC_Fault_IsSafeToRun(const TC_FaultState_t *f);

#endif
```

4.2 ???tc_fault.c?

```
#include "tc_fault.h"
#include "math.h"

#define OMEGA_MIN      -1.0f
#define OMEGA_MAX      200.0f
#define OMEGA_JUMP_MAX  50.0f
```

```
#define AX_MAX         20.0f
#define STEER_JUMP_MAX 15.0f   // deg/10ms

#define FAULT_COUNT_MAX 3

void TC_Fault_Init(TC_FaultState_t *f) {
    f->flags = FAULT_NONE;
    f->wheel_err_count = 0;
    f->imu_err_count   = 0;
    f->steer_err_count = 0;
    f->torque_err_count= 0;
    f->can_timeout_count = 0;
    for (int i = 0; i < 4; i++) f->omega_prev[i] = 0.0f;
    f->steer_prev = 0.0f;
}

void TC_Fault_Update(const TC_Input_t *in, TC_FaultState_t *f) {
    /* 1. □□□□ + □□ */
    float omegas[4] = { in->omega_fl, in->omega_fr, in->omega_rl, in->omega_rr };
    bool wheel_fault = false;

    for (int i = 0; i < 4; i++) {
        if (omegas[i] < OMEGA_MIN || omegas[i] > OMEGA_MAX) {
            wheel_fault = true;
        }
        float diff = fabsf(omegas[i] - f->omega_prev[i]);
        if (diff > OMEGA_JUMP_MAX) {
            wheel_fault = true;
        }
        f->omega_prev[i] = omegas[i];
    }

    if (wheel_fault) {
        if (++f->wheel_err_count >= FAULT_COUNT_MAX) {
            f->flags |= FAULT_WHEEL;
        }
    } else {
        f->wheel_err_count = 0;
    }
```

```
    /* 2. IMU 範圍 */
    if (fabsf(in->ax_imu) > AX_MAX) {
        if (++f->imu_err_count >= FAULT_COUNT_MAX) {
            f->flags |= FAULT_IMU;
        }
    } else {
        f->imu_err_count = 0;
    }

    /* 3. 方向盤跳動 */
    float steer_diff = fabsf(in->steer_deg_raw - f->steer_prev);
    if (steer_diff > STEER_JUMP_MAX) {
        if (++f->steer_err_count >= FAULT_COUNT_MAX) {
            f->flags |= FAULT_STEER;
        }
    } else {
        f->steer_err_count = 0;
    }
    f->steer_prev = in->steer_deg_raw;

    /* 4. 扭矩範圍（依車輛實際調整上限） */
    if (in->T_driver < 0.0f || in->T_driver > 1000.0f) {
        f->flags |= FAULT_TORQUE;
    }
}

bool TC_Fault_IsSafeToRun(const TC_FaultState_t *f) {
    /* 有關鍵錯誤就不允許跑 TC */
    if (f->flags & (FAULT_WHEEL | FAULT_TORQUE | FAULT_CAN_TIMEOUT)) {
        return false;
    }
    return true;
}
```

5. TC ?????tc_controller.h / tc_controller.c?

5.1 ????tc_controller.h?

```
#ifndef TC_CONTROLLER_H
#define TC_CONTROLLER_H

#include "tc_types.h"
#include "tc_mode.h"

typedef struct {
    /* 內部狀態 */
    float integral;          // PI 積分
    float v_imu_est;         // 從 IMU 積分的速度估計
    float steer_deg_f;       // 濾波後方向盤 deg
    float ax_peak;           // 用於 μ 估測的加速度峰值

    uint32_t integral_reset_count;
} TC_Internal_t;

void TC_Controller_Init(TC_Internal_t *s);
void TC_Controller_Update(
    const TC_Input_t  *in,
    const TC_ModeState_t *mode,
    TC_Internal_t     *state,
    TC_Output_t       *out
);

#endif
```

5.2 ??????

1. 由前輪輪速估算車速 (v)。
2. 後輪 slip ratio: (\lambda_L = (\omega_{RL} R_{rear} - v) / v)，(\lambda_R) 類似。
3. 方向盤轉角低通濾波。
4. 根據模式參數 + 濾波後轉角 + 車速，算出 Target Slip (\lambda^*)。
5. 計算誤差 (e = \lambda^* - \bar{\lambda})。
6. PI 控制（含積分 Anti-windup + 重置邏輯）。
7. 安全邏輯（低速、嚴重 slip、錯誤）覆蓋一切。
8. 最後輸出 T_cmd。

5.3 ???tc_controller.c?

```
#include "tc_controller.h"
#include "tc_params.h"
#include "math.h"

void TC_Controller_Init(TC_Internal_t *s) {
    s->integral = 0.0f;
    s->v_imu_est = 0.0f;
    s->steer_deg_f = 0.0f;
    s->ax_peak = 0.0f;
    s->integral_reset_count = 0;
}

/* 車速估計（簡化版: 先只用前輪） */
static float TC_EstimateSpeed(const TC_Input_t *in) {
    float v = (in->omega_fl + in->omega_fr) * 0.5f * R_FRONT;
    if (v < 0.1f) v = 0.1f;
    return v;
}

/* Slip Ratio */
static void TC_ComputeSlip(const TC_Input_t *in, float v,
                           float *slip_L, float *slip_R, float *slip_avg) {
    float v_safe = (v > 0.1f) ? v : 0.1f;
    *slip_L = (in->omega_rl * R_REAR - v_safe) / v_safe;
    *slip_R = (in->omega_rr * R_REAR - v_safe) / v_safe;
    *slip_avg = (*slip_L + *slip_R) * 0.5f;
}

/* 方向盤低通濾波 */
static float TC_FilterSteer(float raw_deg, TC_Internal_t *s) {
    s->steer_deg_f = STEER_ALPHA * s->steer_deg_f +
                     (1.0f - STEER_ALPHA) * raw_deg;
    return s->steer_deg_f;
}

/* 簡易 μ 估測，用 ax 峰值分級 → 回傳倍數 0.5~1.0 */
static float TC_EstimateMu(TC_Internal_t *s, const TC_Input_t *in) {
    float ax_abs = fabsf(in->ax_imu);
    /* 指數平滑追蹤峰值 */
    s->ax_peak = fmaxf(s->ax_peak * 0.95f + ax_abs * 0.05f, ax_abs);
```

```
    if (s->ax_peak > 8.5f)   return 1.0f;  // 高 μ
    if (s->ax_peak > 5.5f)   return 0.8f;  // 中 μ
    if (s->ax_peak > 3.5f)   return 0.65f; // 低 μ
    return 0.5f;
}

/* 方向盤為主的 Target Slip（模式 + steer_sens） */
static float TC_ComputeTargetSlip(
    float steer_deg_f,
    float v,
    const TC_ModeState_t *mode)
{
    float d = fabsf(steer_deg_f);
    float k_steer;

    /* 分段線性: 可把斜率乘上 mode->steer_sens 調整靈敏度 */
    if (d < 4.0f) {
        k_steer = 1.0f;
    } else if (d < 20.0f) {
        float slope = 0.018f * mode->steer_sens;
        k_steer = 1.0f - slope * (d - 4.0f);    // 直到中彎
    } else {
        float slope2 = 0.01f * mode->steer_sens;
        k_steer = 0.65f - slope2 * (d - 20.0f);
    }

    if (k_steer < 0.4f) k_steer = 0.4f;

    /* 速度補償（高速略放寬） */
    float speed_factor = fminf(v * 3.6f / 40.0f, 1.0f) * 0.03f + 0.97f;

    float lambda = mode->lambda_base * k_steer * speed_factor;
    if (lambda < 0.08f) lambda = 0.08f;
    return lambda;
}

/* 積分重置邏輯（你之前討論的「防飽和」） */
static void TC_IntegralResetLogic(const TC_Input_t *in,
                                  const TC_ModeState_t *mode,
```

```
                                    TC_Internal_t *s,
                                    float v_mps)
{
    float v_kmh = v_mps * 3.6f;

    /* 1) 低速: 直接清零, 起步最安全 */
    if (v_kmh < 2.0f) {
        if (s->integral != 0.0f) {
            s->integral = 0.0f;
            s->integral_reset_count++;
        }
        return;
    }

    /* 2) 駕駛幾乎沒踩油門: 快速衰減 */
    if (fabsf(in->T_driver) < 2.0f) {
        s->integral *= 0.1f;
        return;
    }

    /* 3) 模式剛剛切換（因為你只在靜止時切換, 這裡可省略或保留） */
    (void)mode; // 目前不需要特別判斷, 保留介面
}

/* 主控制: 不含 Safety（Safety 放在 main.c） */
void TC_Controller_Update(
    const TC_Input_t  *in,
    const TC_ModeState_t *mode,
    TC_Internal_t     *state,
    TC_Output_t       *out)
{
    /* 1. 車速 & slip */
    out->v_mps = TC_EstimateSpeed(in);
    TC_ComputeSlip(in, out->v_mps, &out->slip_L, &out->slip_R, &out->slip_avg);

    /* 2. 方向盤濾波 */
    out->steer_deg_f = TC_FilterSteer(in->steer_deg_raw, state);

    /* 3. 目標 slip（模式 + 方向盤） */
    out->lambda_target = TC_ComputeTargetSlip(out->steer_deg_f, out->v_mps, mode);
```

```
    /* 4. 積分重置邏輯 */
    TC_IntegralResetLogic(in, mode, state, out->v_mps);

    /* 5. 自適應 Kp by μ */
    float mu_scale = TC_EstimateMu(state, in);
    float Kp_eff = mode->Kp * mu_scale;
    float Ki_eff = mode->Ki;  // 也可以乘 mu_scale, 看你喜好

    /* 6. PI 計算 */
    float error = out->lambda_target - out->slip_avg;

    float p_term = Kp_eff * error;

    state->integral += error * TC_DT_SEC;
    if (state->integral > INTEGRAL_MAX)   state->integral = INTEGRAL_MAX;
    if (state->integral < -INTEGRAL_MAX)  state->integral = -INTEGRAL_MAX;
    float i_term = Ki_eff * state->integral;

    out->delta_T = p_term + i_term;
}
```

6. Main Loop + Safety + CAN?main.c ???

6.1 ?????TIM2 10 ms ???

```
#include "main.h"
#include "tc_types.h"
#include "tc_mode.h"
#include "tc_fault.h"
#include "tc_controller.h"
#include "tc_io.h"   // 你自己實作: CAN 收發, 按鈕處理

TC_ModeState_t   g_mode;
TC_FaultState_t  g_fault;
```

```
TC_Internal_t    g_state;
TC_Input_t       g_in;
TC_Output_t      g_out;

/* 初始化 */
void TC_SystemInit(void) {
    TC_Mode_Init(&g_mode);
    TC_Fault_Init(&g_fault);
    TC_Controller_Init(&g_state);
}

/* 10 ms 定時中斷 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM2) {

        /* 1. 讀取 CAN / ADC → 填 g_in */
        TC_IO_ReadInputs(&g_in);     // 需要你自己實作

        /* 2. 錯誤偵測 */
        TC_Fault_Update(&g_in, &g_fault);

        /* 3. 模式在靜止時套用 */
        float v_kmh_fake = 0.0f;    // 這裡可以直接用上次 g_out.v_mps 轉 km/h
        v_kmh_fake = g_out.v_mps * 3.6f;
        TC_Mode_ApplyIfStopped(&g_mode, v_kmh_fake);

        /* 4. 判斷是否允許跑 TC */
        bool safe = TC_Fault_IsSafeToRun(&g_fault);

        if (!safe) {
            /* 4-1. 發生嚴重錯誤 → 關閉 TC, 扭矩交回駕駛或安全模式 */
            g_out.tc_status = TC_STATUS_FAULT;
            g_out.T_cmd = 0.0f;  // 或者 0.3 * g_in.T_driver, 看你的需求
        } else {
            /* 4-2. 正常執行控制（含 PI, 但不含最外層 Safety） */
            TC_Controller_Update(&g_in, &g_mode, &g_state, &g_out);

            float v_kmh = g_out.v_mps * 3.6f;

            /* 5. 最外層 Safety 覆蓋 */
```

```
            if (v_kmh < V_TC_OFF_KMH) {
                g_out.T_cmd = g_in.T_driver;
                g_out.tc_status = TC_STATUS_OFF;
            }
            else if (fabsf(g_out.slip_L) > SLIP_SINGLE_MAX ||
                     fabsf(g_out.slip_R) > SLIP_SINGLE_MAX) {
                g_out.T_cmd = 0.5f * g_in.T_driver;
                g_out.tc_status = TC_STATUS_SAFETY;
            }
            else if (fabsf(g_out.steer_deg_f) > 30.0f) {
                g_out.T_cmd = g_in.T_driver * g_mode.min_tq_ratio;
                g_out.tc_status = TC_STATUS_SAFETY;
            }
            else {
                /* □□ PI □□ + □□ */
                float T_raw = g_in.T_driver * (1.0f + g_out.delta_T);
                float T_min = g_in.T_driver * g_mode.min_tq_ratio;
                float T_max = g_in.T_driver;

                if (T_raw < T_min) T_raw = T_min;
                if (T_raw > T_max) T_raw = T_max;

                g_out.T_cmd = T_raw;
                g_out.tc_status = TC_STATUS_NORMAL;
            }
        }

        g_out.fault_flags = g_fault.flags;

        /* 6. □□ CAN □□□T_cmd, □□, □□, fault */
        TC_IO_SendOutputs(&g_in, &g_out, &g_mode, &g_fault);
    }
}
```

7. ???????????tc_io.h / tc_io.c ???

□□□□

- 用 GPIO 按鈕改 `g_mode.target_mode`:
  - 短按 MODE: `STRAIGHT → FIGURE8 → TRACK → STRAIGHT...`
  - 呼叫 `TC_Mode_RequestChange(&g_mode, new_mode, v_kmh)`。
- 由 CAN 顯示:
  - 目前模式 `g_mode.current_mode`。
  - `g_out.tc_status`（OFF / NORMAL / SAFETY / FAULT）。
  - 扭矩限縮比例 `g_out.T_cmd / g_in.T_driver`。
  - μ 等級（可以從 `g_state.ax_peak` 推估）。

你可以自行定義 `0x200 ~ 0x20F` 做儀表板訊息。

---

8. ??????????

1. 先鎖在「繞 **8** 模式」，在乾地繞 8 練習，調整:
   - `LAMBDA_BASE_F8` 先從 0.16 → 看實際 slip 波形。
   - `KP_FIGURE8` 3.1 若太慢可到 3.5; 若抖就往下。
2. 直線模式：在直線起步測 `LAMBDA_BASE_STRAIGHT` 0.20 是否太滑 / 太鈍。
3. 賽道模式：上賽道後，再微調 `LAMBDA_BASE_TRACK` 和 `STEER_SENS_TRACK`，讓大轉角出彎更穩。

---

9. ??????

- 模式：決定「基準 λ*、Kp/Ki、最低扭矩比例」。
- 方向盤：決定「當下允許的縱向抓地比例 (k_{steer})」。
- **μ** 估測：決定「 Kp 要多兇」。
- 積分重置：保證起步、收油、模式切換時不會被舊積分害死。
- **Safety**: 永遠在最外層，隨時可以一刀切斷 PI 的輸出。

照這份 `.md` 建專案與檔案，功能會跟我們前面討論的一樣齊全，你可以先把這整段貼進 HackMD，之後要補圖（流程圖、波形）可以在各章下面加上自己的實測截圖。

PID ????????? MATLAB ??

PID ????????? MATLAB ??

PID □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□

□□□□□□ $u(t)$ □□□□□□□□ $$ u(t) = K_p e(t) + K_i \int_{0}^{t} e(\tau) d\tau + K_d \frac{de(t)}{dt} $$

□□□□ $e(t)$ □□ □□ **(Error)**□□□□□□□ - □□□□□ K_p□ K_i□ K_d □□□□□□□□□□□□□□

□□□□□□□□□□□ PID □□□ **□□□□□□□□□□□□□□□□□**□

---

1. ???? P?I?D ????

? P (Proportional???) —— ?????

- □□□ □□□□□□□□□/□□□□□□□□
- □□□□ □□□□□ 100 □□□□□□□□□□□□□□□□□□□□□□□ 10 □□□□□□□□□□□□□□□□□□□
- □□□ □□□□□□□□□
- □□□ □□□ **□□□□□ (Steady-state error)□ ** □□□□□□□□□□□□□□□□□□□□ 1 □□□□ P □□□□□□□□□□□□□□□□□□□□□□□□□□□□ 0 □□□□□

? I (Integral???) —— ?????

- □□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□ □□□□□□□ 1 □□□□□□□□□□□□□□ I □□□□□□□□□□□ 1 □□□□ 2 □□□□ 3 □ ...□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 1 □□□
- □□□ □□□□□□□□□□□□□□□□□□□□□□
- □□□ □□□□□□□□□□□□□□□□□ **□□□ (Overshoot)□ ** □□□□□□□□□□□□□□□□□□□□□

? D (Derivative???) —— ?????

- 原理: 計算誤差的「變化率」。它感覺到誤差正在快速縮小，就會提早踩煞車。
- 開車情境: 雖然離停止線還有 20 公尺，但你看時速表發現車速高達 100 km/h，你知道如果再不踩煞車絕對會衝過頭！所以你提早收油門踩煞車。這就是 D 的作用: 阻尼（煞車）效果。
- 優點: 抑制 P 和 I 造成的超調 (衝過頭) 現象，讓系統更快穩定下來。
- 缺點: 對「雜訊 (Noise)」極度敏感。如果感測器數值跳動很大，D 會以為誤差劇烈變化，導致系統跟著發瘋狂抖。

---

2. ????? PID ???? (Tuning Guide)

工程師在現場調 PID 參數時，絕對不是亂猜，通常會遵循這個順序:

1. 先調 **K_p (P 參數)**: 把 K_i 和 K_d 都設為 0。慢慢把 K_p 調大，直到系統反應夠快，但開始出現一點點震盪為止。
2. 再加 **K_i (I 參數)**: 把 K_i 慢慢調大，目的是消除最後那一點點停不下來的穩態誤差。這時系統可能會衝過頭 (Overshoot)。
3. 最後加 **K_d (D 參數)**: 把 K_d 稍微調大，用來壓制剛剛 K_i 造成的衝過頭現象，讓曲線變得平滑完美。

---

3. MATLAB ?? PID ????

在 MATLAB 中，你完全不需要自己刻那個複雜的微積分方程式，官方已經幫你寫好超級方便的 `pid` 函數了!

```
% 1. 設定你的 PID 參數
Kp = 1.5;
Ki = 0.5;
Kd = 0.1;

% 2. 建立一個 PID 控制器物件
C = pid(Kp, Ki, Kd);

% 3. 假設你有一個馬達的數學模型 (轉移函數 Transfer Function)
% 這裡假設一個簡單的系統 G(s) = 1 / (s^2 + 2s + 1)
numerator = 1;
denominator = [1, 2, 1];
```

```
G = tf(numerator, denominator);

% 4. 將 PID 控制器與馬達系統組合成「閉迴路 (Closed-loop)」
sys_cl = feedback(C * G, 1);

% 5. 畫出系統的「步階響應 (Step Response)」看看控制效果!
figure;
step(sys_cl);
title('PID 控制系統響應');
grid on;
```

4. ???? PID ????? (Step Response)

在畫出來的曲線上，橫軸是「時間 (秒)」，縱軸是「系統的當前數值」。完美的情況是線條瞬間從 0 垂直升到 1 然後死死貼在 1 上面，但現實世界的物理系統做不到，一定會有一個過渡期。

我們最需要盯著看的，就是以下兩個指標:

? 1. ??? (Overshoot, %OS) —— ????????

- 圖上怎麼看: 觀察曲線 第一次超過目標值 **(1)** 的最高峰 **(Peak)**。
- 定義: 那個最高峰比起目標值多出了百分之多少。
- 實例: 目標是 1，但系統一開始衝太快，最高點來到 1.25 才掉下來，那你的 Overshoot 就是 **25%**。
- 工程意義: * 如果 Overshoot 太大，代表系統太「激動」了。
  - 在某些場合這是致命的! 例如: 電梯如果 Overshoot 很大，乘客會撞到天花板; 機械手臂如果 Overshoot 很大，會把原本要夾的零件撞飛。
- 怎麼解決: 通常是 P 或 I 給太大了，可以嘗試 增加 **D (微分)** 參數來增加阻尼（煞車），把那個山峰壓下來。

?? 2. ???? (Settling Time, T_s) —— ??????????

- 圖上怎麼看: 觀察曲線上下震盪到最後，什麼時候開始「永遠」停留在目標值附近不再亂跑。
- 定義: 系統進入並一直保持在「目標值的 $\pm 2%$ (或 $\pm 5%$) 誤差範圍內」所需的時間。
- 實例: 系統雖然很快就衝到 1，但在 0.9 到 1.1 之間上下震盪了很久，直到第 5 秒鐘，它才穩定在 0.98 到 1.02 這個超窄的管子裡不出來，那 Settling Time 就是 **5** 秒。
- 工程意義: 決定了你的機器「反應有多乾脆」。Settling Time 越短，代表機器越快準備好進行下一個動作。

---

5. MATLAB ????????????`stepinfo`

[illegible] MATLAB [illegible] `stepinfo`
[illegible]

```
% [illegible] sys_cl
% sys_cl = feedback(C * G, 1);

% 1. [illegible] stepinfo [illegible]
info = stepinfo(sys_cl);

% 2. [illegible]
disp(['[illegible] (Overshoot): ', num2str(info.Overshoot), ' %']);
disp(['[illegible] (Settling Time): ', num2str(info.SettlingTime), ' [illegible]']);
```

? ??? UI ??????????

[illegible] `step(sys_cl)` [illegible]

1. [illegible] [illegible] [illegible]
2. [illegible] **Characteristics ([illegible])**[illegible]
3. [illegible] **Peak Response ([illegible] / [illegible] Overshoot)** [illegible] **Settling Time ([illegible])**[illegible]
4. [illegible]

MATLAB ??????? (Basic Syntax Cheat Sheet)

這份筆記整理了 MATLAB 最核心、最常用的基本語法。寫 MATLAB 腳本 (`.m` 檔) 時，隨時可以拿出來參考。

---

1. ????? (????)

每次執行新腳本前，最好先清空舊的資料和視窗，避免被上一次的變數干擾。這三行通常會放在程式的最開頭：

```
clc;          % 清空指令視窗 (Command Window) 的文字
clear;        % 清空工作區 (Workspace) 的所有變數
close all;    % 關閉所有之前畫的圖片視窗 (Figures)
```

---

2. ????? (Operators)

MATLAB 的運算子主要分為三大類：算術、關係（比較）與邏輯運算子。

(1) ????? (Arithmetic Operators)

前面提過矩陣運算的「點運算」，這裡做個總整理：

- `+`：加法
- `-`：減法
- `*`：矩陣乘法 (Matrix multiplication)
- `/`：矩陣除法 (Matrix right division)
- `^`：矩陣次方 (Matrix power)
- `.*`：元素對應相乘 (Element-wise multiplication) ⚠ 最常用
- `./`：元素對應相除 (Element-wise division)
- `.^`：元素對應次方 (Element-wise power)

(2) ??/????? (Relational Operators)

用來比較數值大小，回傳結果為邏輯值 `1` (True/真) 或 `0` (False/假)。 ⚠ 特別注意: **MATLAB** 的「不等於」是波浪號 `~=`，不是 `!=` 喔!

- `==` ：等於 (注意是兩個等號，一個等號 `=` 是指派變數)
- `~=` ：不等於 (Not equal)
- `>` ：大於
- `<` ：小於
- `>=` ：大於或等於
- `<=` ：小於或等於

```
% 關係運算子範例
a = 5;
b = 5;
c = 10;

result1 = (a == b);   % 結果為 1 (True)
result2 = (a ~= c);   % 結果為 1 (True)
```

(3) ????? (Logical Operators)

用來組合多個條件。 MATLAB 的邏輯運算子有分「純量 (單一數字)專用」和「陣列專用」兩種。

A. 雙符號（用於純量 /單一變數的 `if` / `while` 條件判斷，具有短路特性，執行較快）：

- `&&` ：邏輯 AND (兩邊條件都要成立)
- `||` ：邏輯 OR (只要有一邊成立即可)

B. 單符號（用於陣列 /矩陣的元素對元素邏輯判斷）：

- `&` ：陣列 AND
- `|` ：陣列 OR
- `~` ：邏輯 NOT (反相，把 True 變 False，False 變 True)

```
% 邏輯運算子範例
age = 25;
has_license = 1; % 1 代表 True

% 使用 && 判斷兩個條件是否同時成立
if (age >= 18) && (has_license == 1)
    disp('可以合法開車!');
end
```

```
% 邏輯 NOT 範例
is_raining = 0; % 0 代表 False
if ~is_raining
    disp('沒下雨，可以出門！');
end
```

3. ?????

```
% 這是一行註解 (MATLAB 的註解是用百分比符號)

a = 10;      % 宣告變數 (句尾加上分號 `;` 代表不要在指令視窗印出結果)
b = 5        % 句尾「沒有」分號，執行時會在螢幕上印出 b = 5

% 字串 (字元陣列)
myText = 'Hello World';
```

4. ?? (Vector) ??? (Matrix) ??

這是在 MATLAB 中最常做的事情: 建立一維或二維的數據。

```
% 1. 手動輸入建立
V_row = [1, 2, 3, 4];        % 橫列向量 (Row Vector)，用逗號或空格隔開
V_col = [1; 2; 3; 4];        % 直行向量 (Column Vector)，用分號換行
M = [1, 2; 3, 4];            % 2x2 矩陣 (第一列1,2; 第二列3,4)

% 2. 自動生成等差數列 (格式: 起點 : 步長 : 終點)
t = 0 : 0.1 : 10;            % 建立一個從 0 開始，每次加 0.1，到 10 結束的陣列 (常用於時間軸)
nums = 1 : 5;                % 步長若省略，預設為 1 (即 1, 2, 3, 4, 5)

% 3. 常用內建矩陣
Z = zeros(3, 3);             % 建立 3x3 的全零矩陣
O = ones(2, 4);              % 建立 2x4 的全一矩陣
```

5. ???? (??????)

MATLAB 的索引是 從 **1** 開始 的（這跟 C, Python 從 0 開始不一樣喔！）。

```
A = [10, 20, 30, 40, 50];

x = A(1);        % 抓取第 1 個元素（結果為 10）
y = A(3);        % 抓取第 3 個元素（結果為 30）
z = A(2:4);      % 抓取第 2 到第 4 個元素（結果為 [20, 30, 40]）
last = A(end);   % 抓取最後一個元素（結果為 50，end 是超級好用的關鍵字）
```

---

6. ?????????? vs. ????? (???? ??)

這是初學者最常報錯的地方！在做兩個陣列相乘除時，一定要分清楚你是要做「線性代數的矩陣相乘」還是「對應位置的數字各自相乘」。

```
A = [1, 2, 3];
B = [4, 5, 6];

% 【矩陣乘法】（符合線性代數規則）
% C = A * B;     % 這樣寫會報錯！因為 1x3 不能乘 1x3
C = A * B';      % 正確：1x3 乘 3x1（B' 代表轉置矩陣）

% 【元素對元素運算】（點運算 Element-wise）
% 在乘號 `*`、除號 `/`、次方 `^` 前面加上一個「點 `.`」
D = A .* B;      % 結果為 [1*4, 2*5, 3*6] = [4, 10, 18]
E = A .^ 2;      % 把 A 裡面的每個數字各自平方 = [1, 4, 9]
```

---

7. ???? (if / else)

```
score = 85;

if score >= 90
    disp('等級：A');        % disp() 用來在螢幕上印出文字
elseif score >= 80
    disp('等級：B');
else
    disp('等級：C');
```

```
end
```

8. ?? (for / while)

```
% 【for 迴圈】（已知要跑幾次時使用）
for i = 1:5
    % 將會執行 5 次，i 會依序變成 1, 2, 3, 4, 5
    disp(['現在跑到第 ', num2str(i), ' 圈']);
end

% 【while 迴圈】（達到某條件才停止時使用）
count = 1;
while count < 4
    disp(count);
    count = count + 1;
end
```

9. ????????

```
length(A)    % 查詢一維陣列的長度（裡面有幾個數字）
size(M)      % 查詢矩陣的維度（回傳 幾列 x 幾欄）
max(A)       % 找出陣列中的最大值
min(A)       % 找出陣列中的最小值
mean(A)      % 計算平均值
```

10. ???? (Custom Functions)

自訂函數就像是你自己發明了一台小機器：你丟東西進去（輸入參數 Input），它幫你算完後把結果吐出來（輸出結果 Output）。

?? ??????????????

在 MATLAB 中建立函數時，通常會額外開一個新的 `.m` 檔案。

這個檔案的存檔名稱，必須跟你的函數名稱一模一樣！

(1) ?????????

□□

```
% 1. □□□□□□□□□□□□□□my_math_calc.m

function [out_add, out_sub] = my_math_calc(in1, in2)
    % □□□□□□□□□□□□□□□□□□□□ help my_math_calc □□□□□□□□□□□□

    % --- □□□□□□□□□□□ ---
    out_add = in1 + in2;   % □□□□□□□□□□□□□□□□
    out_sub = in1 - in2;   % □□□□□□□□□□□□□□□□

end % □□□□□□ (□□□□□□□□□□□□□□□□□□□□)
```

(2) ???? (??) ????

□□□□□□□□□□□□□□□□ `main_script.m` □□□□□□□□□□□□□□□□□□□□□
□□□□□□ □□□□□□□□□□□□□□□□□□□□

```
% --- □ main_script.m □□□□□□□ ---

a = 10;
b = 3;

% □□□□□□□□□□□□□□□□□□ [ ] □□□□□□□□
[result_plus, result_minus] = my_math_calc(a, b);

disp(['□□□□□□□', num2str(result_plus)]);
disp(['□□□□□□□', num2str(result_minus)]);
```

? ????? (????/??????)

- □□□□□□□ □□□□□□□□□□□□□
  ```
  % □□□□□ function y = calc_square(x)
  y = calc_square(5); % □□□□
  ```
- **□□□□ (□□□□□□□□)□**
  □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
% 宣告函數: function draw_beautiful_plot(t, y)
draw_beautiful_plot(time_data, speed_data); % 直接呼叫，不需要等號接收
```

MATLAB App Designer (?????) ?????

MATLAB App Designer (?????) ?????

App Designer 是 MATLAB 現代化設計 GUI (圖形化使用者介面) 的專屬工具，用來取代舊版的 GUIDE 。它把畫面設計（ Design View）跟程式邏輯（ Code View）完美結合在一起。

1. ???????

在 MATLAB 指令視窗 (Command Window) 輸入 `appdesigner` 並按下 Enter，就會跳出專屬的設計視窗。

裡面主要分為兩個世界，你可以透過畫面右上角的按鈕隨時切換：

- **Design View (**設計視角 **)**：左邊有元件庫 (Component Library)，你可以直接把「按鈕 (Button)」、「滑桿 (Slider)」、「畫布 (Axes)」拖拉到中間的空白視窗上排版。
- **Code View (**程式視角 **)**：這裡會自動生成所有 UI 元件的底層程式碼（背景是灰色的，你不能改）。你只能在「白色的區域」寫你自己的邏輯。

---

2. ????????? (Callback)

App 怎麼知道你按下了按鈕？這全靠 **Callback (**回呼函數 **)**。白話文：Callback 就是「當某個事件發生時（例如按鈕被點擊、滑桿被拉動），App 應該要去執行的那段程式碼」。

? ??????? Callback?

1. 在 Design View 中，對著你拉出來的按鈕按 滑鼠右鍵 。
2. 選擇 **Callbacks -> Add ButtonPushedFcn callback**。
3. 畫面會瞬間切換到 Code View，並且幫你開好一個白色的區塊，把游標停在那裡等你寫程式！

---

3. ??????????????

假設我們在畫面上拉了三個東西:

1. 一個畫布 (預設名稱會是 `app.UIAxes`)
2. 一個滑桿 (預設名稱會是 `app.Slider`)
3. 一個按鈕 (預設名稱會是 `app.Button`)

我們現在要在剛剛自動生成的按鈕 Callback 裡面寫程式。目標是:
按下按鈕時，讀取滑桿的數字，並在畫布上畫出對應頻率的波浪!

```
% 這是 App Designer 自動幫你生成的 Callback 結構
function ButtonPushed(app, event)

    % 1. 【讀取】滑桿現在的值 (Value)
    % 語法: app.元件名稱.屬性
    freq = app.Slider.Value;

    % 2. 建立時間軸與計算訊號 (這跟我們平常寫的 MATLAB 一模一樣)
    t = 0 : 0.01 : 1;
    y = sin(2 * pi * freq * t);

    % 3. 【輸出】把圖畫在 App 專屬的畫布上 (⚠◌ 初學者最常死在這裡! )
    % 以前我們直接寫 plot(t, y)，但在 App 裡「絕對不行」!
    % 你必須明確告訴 MATLAB 要畫在哪個元件上，否則它會另外彈出一個醜醜的 Figure 視窗。
    plot(app.UIAxes, t, y);

    % 4. 美化 App 裡的畫布
    title(app.UIAxes, ['目前頻率: ', num2str(freq), ' Hz']);
    grid(app.UIAxes, 'on');

end
```

---

4. ?? App Designer ?????? (????)

1. 所有的變數前面都要加 `app.` 在一般的腳本裡，變數叫 `x` 就是 `x`。但在 App 裡面，如果你希望這個變數可以在不同的按鈕、不同的 Callback 之間共用 (例如按鈕 A 算出來的結果，按鈕 B 要拿去存檔)，你必須把它宣告為 屬性 **(Properties)**。
   - 操作方式: 在 Code View 點擊上方的 `Property` 按鈕新增。

   - 使用方式：呼叫時必須寫成 `app.my_variable = 10;`。
2. 所有改變 **UI** 畫面的指令，第一個參數都是 `app.元件名稱`
   - 畫圖：`plot(app.UIAxes, x, y)`
   - 改文字：`app.Label.Text = '計算完成！'`
   - 清除圖形：`cla(app.UIAxes)`
3. 千萬不要手動去改灰色區域的程式碼那是系統自動生成的 UI 排版碼，一動很容易整個 App 崩潰。請乖乖在白色的 Callback 區域寫你的邏輯就好。

MATLAB Plot ????? (?? Simulink ????)

這是一份專為 Simulink 數據繪圖整理的 MATLAB 語法大全。假設 Simulink 回傳的物件為 `out`，且你在 `To Workspace` 設定的變數名稱為 `sim_result`。

---

1. ??????????????

如果你的 Simulink `To Workspace` 儲存格式 (Save format) 是設定為 **Array**:

```
% 1. 提取資料
t = out.tout;                % 抓取時間軸
y = out.sim_result;          % 抓取模擬數據

% 2. 建立畫布與繪圖
figure;                      % 開啟一個新的空白視窗 (避免覆蓋舊圖)
plot(t, y);                  % 最基本的畫圖指令
```

如果儲存格式是設定為 **Timeseries** (官方推薦):

```
% 最快捷徑：連時間都不用抓，直接丟進去畫！
figure;
plot(out.sim_result, 'LineWidth', 2);
```

---

2. ?????????

在 `plot()` 的第三個參數，可以用字串來快速設定「顏色」、「線型」與「標記」。格式為: `plot(x, y, '顏色+線型+標記')`

- 顏色 **(Colors)**: `r` (紅), `b` (藍), `g` (綠), `k` (黑), `m` (洋紅), `c` (青)
- 線型 **(Line Styles)**: `-` (實線), `--` (虛線), `:` (點線), `-.` (點劃線)
- 標記 **(Markers)**: `o` (圓圈), `*` (星號), `x` (叉叉), `s` (正方形)

```
% 範例: 畫一條紅色、虛線、帶有圓圈標記的線，線寬為 2
plot(t, y, 'r--o', 'LineWidth', 2);
```

---

3. ??????????

畫完線之後，務必加上標籤讓圖表具有可讀性:

```
plot(t, y, 'b-', 'LineWidth', 1.5);


% 美化語法
title('系統響應結果');          % 圖表大標題
xlabel('時間 (s)');             % X 軸標籤
ylabel('振幅 / 轉速');          % Y 軸標籤
grid on;                        % 打開背景網格 (看數據更清楚)
xlim([0 10]);                   % 限制 X 軸的顯示範圍 (例如只顯示 0 到 10 秒)
ylim([-1 5]);                   % 限制 Y 軸的顯示範圍
```

---

4. ????????? (??????)

使用 `hold on` 可以讓 MATLAB 不要清掉舊的線，繼續在同一張圖上畫新的線。

```
figure;
% 畫第一條線
plot(t1, y1, 'b-', 'LineWidth', 2);
hold on;  % 鎖定畫布! 接下來畫的線都會疊加上去

% 畫第二條線
plot(t2, y2, 'r--', 'LineWidth', 2);

% 加上圖例 (順序必須跟畫線的順序一致)
legend('情境 A (Kp=1.5)', '情境 B (Kp=5.0)');

title('不同參數之結果比較');
xlabel('時間 (s)');
ylabel('數值');
grid on;
```

```
hold off; % 解除鎖定 (好習慣: 畫完後解除)
```

---

5. ?????????? (Subplot)

如果你有三個不同的訊號（例如: 位置、速度、加速度），想放在同一個視窗但「上下分開」顯示，請使用 `subplot(m, n, p)`。

- `m`: 總共有幾列 (Row)
- `n`: 總共有幾欄 (Column)
- `p`: 現在要畫在第幾個位置

```
figure;

% 第一張小圖 (佔用上面)
subplot(2, 1, 1);            % 2列, 1欄, 畫在第1個位置
plot(t, position, 'b-');
title('位置響應');
ylabel('位置 (m)');
grid on;

% 第二張小圖 (佔用下面)
subplot(2, 1, 2);            % 2列, 1欄, 畫在第2個位置
plot(t, velocity, 'r-');
title('速度響應');
xlabel('時間 (s)');
ylabel('速度 (m/s)');
grid on;
```

6. ???? (????? PNG ? PDF)

畫完圖之後，可以使用 MATLAB 的存檔指令將當前的圖片（ `gcf`，代表 Get Current Figure ）自動存到你的資料夾中。

???????? (`saveas`)

適合快速存檔，語法最簡單。

```
% 將當前視窗存成 PNG 圖檔
saveas(gcf, 'Simulink_Result.png');

% 將當前視窗存成 MATLAB 原始圖檔格式 (.fig)，方便日後重新打開修改
saveas(gcf, 'Simulink_Result.fig');
```

??????????? (`exportgraphics` - ???? ?)

這是 MATLAB 較新的指令（ R2020a 之後支援），它會自動裁切掉圖片旁邊多餘的白邊，而且可以設定解析度（ DPI），非常適合用來放在 Word 報告或論文裡!

```
% 1. 存成高解析度 PNG (設定解析度為 300 DPI)
exportgraphics(gcf, 'High_Quality_Result.png', 'Resolution', 300);

% 2. 存成向量圖 PDF (放大絕對不會失真，論文最愛)
exportgraphics(gcf, 'Vector_Result.pdf', 'ContentType', 'vector');
```

MATLAB-Simulink ??????????

MATLAB-Simulink ??????????

當你需要自動化修改 Simulink 裡的模塊參數（例如 Gain 的數值、Transfer Function 的係數），並且讓它自動執行時，請使用這套「程式化模擬 (Programmatic Simulation)」語法。

---

1. ??????????

在用程式控制之前，通常需要先將模型載入到電腦記憶體中。

```
model_name = 'my_model';  % 你的 Simulink 檔名 (不要加 .slx)

% 1. 載入模型 (只放在記憶體，畫面不會跳出來，跑迴圈時最快!)
load_system(model_name);

% 2. 開啟模型 (如果需要讓畫面跳出來給你看)
open_system(model_name);

% ... 中間執行修改參數與模擬 ...

% 3. 關閉模型
close_system(model_name, 0); % 第二個參數 0 代表「不儲存變更直接關閉」，1 代表「儲存變更」
```

---

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

這是大家最常卡關的地方! 要修改 Simulink 裡的模塊，你必須知道它的 **「路徑」**。路徑格式為: `模型名稱/模塊名稱` (例如: `my_model/Controller/Kp_Gain`)。

⚠ 核心陷阱: 在 **Simulink** 裡用程式設定參數時，數值通常必須是「字串 **(String/Char)**」，不能直接丟數字!

???`get_param` (????) ? `set_param` (????)

```
block_path = 'my_model/Gain_Block'; % 假設模型裡有一個 Gain 模塊叫做 Gain_Block

% 【讀取】目前的 Gain 數值
current_gain = get_param(block_path, 'Gain');
disp(['目前的 Gain 是: ', current_gain]);

% 【修改】將 Gain 的數值改成 10.5
% 注意! 不能寫 10.5, 必須用 num2str(10.5) 或直接寫 '10.5' 轉成字串!
set_param(block_path, 'Gain', '10.5');

% 【修改求解器步長】(這就是為什麼前面教的 MaxStep 可以這樣設)
set_param(model_name, 'MaxStep', '0.01');
```

---

3. ?????????

`Simulink.SimulationInput` (R2019a ????)

過去大家都在迴圈裡狂用 `set_param`，但這有個致命缺點: 它會真的改動你的模型檔案，如果你忘記改回來，下次打開模型參數就全亂了。

現代 MATLAB 提供了一個超強物件 `SimulationInput`，它可以 **「只在這次模擬中暫時替換參數，模擬結束後模型毫髮無傷」** **。強烈推薦用這個寫法!

```
% 1. 建立一個針對該模型的 SimulationInput 物件
simIn = Simulink.SimulationInput('my_model');

% 2. 暫時修改模塊參數 (不會存入 .slx 檔案)
simIn = simIn.setBlockParameter('my_model/Gain_Block', 'Gain', '15.2');
simIn = simIn.setBlockParameter('my_model/Step_Input', 'Time', '2');

% 3. 暫時修改模型整體設定 (例如 StopTime 或步長)
simIn = simIn.setModelParameter('StopTime', '20');
simIn = simIn.setModelParameter('MaxStep', '0.01');

% 4. 執行模擬 (把剛剛包裝好的 simIn 丟進 sim 函數)
out = sim(simIn);
```

```
% 5. 抓取資料來畫圖
plot(out.tout, out.sim_result);
```

---

4. ?????????????? Simulink (Base Workspace ??)

有時候你要改的參數太多，用 `setBlockParameter` 一個一個設太慢了。最專業的做法是：在 Simulink 的模塊裡，不要填數字，直接填入變數名稱（例如在 Gain 模塊裡面填 `my_Kp`）。

接著，透過 MATLAB 腳本設定 `my_Kp` 的值，並使用 `setVariable` 把它塞給 Simulink 模擬：

```
% 假設 Simulink 模型中的 Gain 模塊設定為 my_Kp，積分模塊設定為 my_Ki

% 1. 建立 SimulationInput 物件
simIn = Simulink.SimulationInput('my_model');

% 2. 將 MATLAB 工作區的變數值，指派給 Simulink 內部使用
simIn = simIn.setVariable('my_Kp', 2.5);  % 這裡可以直接丟數字！
simIn = simIn.setVariable('my_Ki', 0.1);

% 3. 執行模擬
out = sim(simIn);
```

這招在配合 `for` 迴圈做參數掃描（Parameter Sweep）或是做 PID 參數自動調校時，是最乾淨、最不容易出錯的做法！

???? TC ?????STM32F446 ????

???? TC ?????STM32F446 ????

tags: `Traction Control` `RWD` `STM32F446` `CAN` `Steering Angle`

---

0. ???????

0.1 ??

- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□ STM32F446□ 180 MHz, FPU□□
- □□□
  - □□□□ + IMU + □□□□□ + □□□□ → □□□□□□ slip□
  - □□ □□□□□□□ □ Target Slip□□□ IMU □□□□
  - PI □□□□□□□□□□□□□□□□□□
  - □□□□ □□□ / □ 8 / □□□□□□□□□□□□□□□
  - □□□□□□□ → TC □□□□□ Fail-safe□□
  - □□□□□□□ Anti-windup□□
  - □□□□ μ □□ → □□□ Kp□

0.2 ?????????

- `Core/Src/main.c`□□□□□□□ TIM/CAN □□□
- `Core/Inc/tc_types.h`□□□□□□□□ enum□
- `Core/Inc/tc_controller.h` / `Core/Src/tc_controller.c`□ TC □□□□□□
- `Core/Inc/tc_fault.h` / `Core/Src/tc_fault.c`□□□□□□□□ & □□□□□
- `Core/Inc/tc_mode.h` / `Core/Src/tc_mode.c`□□□□□□□□ /□ 8/□□□□
- `Core/Inc/tc_io.h` / `Core/Src/tc_io.c`□ CAN □□ / □□ + □□□□□
- `Core/Inc/tc_params.h`□□□□□□ Kp, Ki, slip □□□□□

---

1. ??????????tc_types.h?

```
#ifndef TC_TYPES_H
#define TC_TYPES_H

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

/* ---- 模式 ---- */
typedef enum {
    TC_MODE_STRAIGHT = 0,  // 直線加速
    TC_MODE_FIGURE8  = 1,  // 繞 8 / skidpad
    TC_MODE_TRACK    = 2   // 賽道
} TC_Mode_t;

/* ---- TC 狀態 ---- */
typedef enum {
    TC_STATUS_OFF      = 0,  // 關閉
    TC_STATUS_NORMAL   = 1,  // 正常控制
    TC_STATUS_SAFETY   = 2,  // 安全限扭
    TC_STATUS_FAULT    = 3   // 錯誤關閉
} TC_Status_t;

/* ---- 錯誤旗標 ---- */
typedef enum {
    FAULT_NONE         = 0,
    FAULT_WHEEL        = 1 << 0,
    FAULT_IMU          = 1 << 1,
    FAULT_STEER        = 1 << 2,
    FAULT_CAN_TIMEOUT  = 1 << 3,
    FAULT_TORQUE       = 1 << 4
} TC_FaultFlags_t;

/* ---- 單次迴圈輸入（10 ms） ---- */
typedef struct {
    /* 原始感測器 + 指令（從 CAN 或 ADC 來） */
    float omega_fl, omega_fr;     // 前輪輪速 rad/s
    float omega_rl, omega_rr;     // 後輪輪速 rad/s
    float ax_imu;                 // 縱向加速度 m/s2
```

```
    float ay_imu;                   // 橫向加速度（目前可不必用）
    float psi_dot;                  // yaw rate rad/s
    float steer_deg_raw;            // 方向盤轉角 deg（未濾波）
    float T_driver;                 // 駕駛要求扭矩 Nm

    /* 系統時間（ms） */
    uint32_t timestamp_ms;
} TC_Input_t;

/* ---- 單次迴圈輸出 ---- */
typedef struct {
    float v_mps;                    // 車速 m/s
    float slip_L, slip_R, slip_avg;
    float steer_deg_f;              // 濾波後方向盤 deg
    float lambda_target;            // 目標 slip
    float delta_T;                  // PI 輸出比例（無單位）
    float T_cmd;                    // 實際下發扭矩 Nm
    TC_Status_t tc_status;
    uint32_t fault_flags;
} TC_Output_t;

#endif
```

2. ??????????tc_params.h?

```
#ifndef TC_PARAMS_H
#define TC_PARAMS_H

/* 輪胎半徑 */
#define R_FRONT        0.32f
#define R_REAR         0.32f

/* TC 主迴圈週期 */
#define TC_DT_SEC      0.01f  // 10 ms, 100 Hz

/* 方向盤濾波係數（低通） */
#define STEER_ALPHA    0.92f  // 越接近 1 越慢，約 3 Hz
```

```
/* Slip 安全閾值 */
#define SLIP_SINGLE_MAX 0.40f   // 單輪 > 0.4 視為嚴重打滑

/* 低速關閉 TC 門檻 */
#define V_TC_OFF_KMH   3.0f

/* 積分上限（Anti-windup） */
#define INTEGRAL_MAX   1.0f

/* ---- 三種模式基線參數 ---- */

/* 直線模式 */
#define KP_STRAIGHT    4.2f
#define KI_STRAIGHT    0.35f
#define LAMBDA_BASE_STRAIGHT  0.20f
#define STEER_SENS_STRAIGHT   0.4f   // 方向盤敏感度較低
#define MIN_TQ_RATIO_STRAIGHT 0.55f  // 最小 55% 扭矩

/* 繞 8 模式 */
#define KP_FIGURE8     3.1f
#define KI_FIGURE8     0.42f
#define LAMBDA_BASE_F8 0.16f
#define STEER_SENS_F8  1.0f
#define MIN_TQ_RATIO_F8 0.40f

/* 賽道模式 */
#define KP_TRACK       2.8f
#define KI_TRACK       0.50f
#define LAMBDA_BASE_TRACK 0.14f
#define STEER_SENS_TRACK  1.3f
#define MIN_TQ_RATIO_TRACK 0.30f

#endif
```

---

3. ?????tc_mode.h / tc_mode.c?

3.1 ????tc_mode.h?

```
#ifndef TC_MODE_H
#define TC_MODE_H

#include "tc_types.h"
#include "tc_params.h"

typedef struct {
    TC_Mode_t current_mode;   // 目前實際模式
    TC_Mode_t target_mode;    // 車手選擇的模式
    bool      change_pending; // 是否等待靜止後切換

    float Kp, Ki;
    float lambda_base;
    float steer_sens;
    float min_tq_ratio;       // 最低扭矩比例
} TC_ModeState_t;

void TC_Mode_Init(TC_ModeState_t *m);
void TC_Mode_RequestChange(TC_ModeState_t *m, TC_Mode_t new_mode, float v_kmh);
void TC_Mode_ApplyIfStopped(TC_ModeState_t *m, float v_kmh);

#endif
```

3.2 ???tc_mode.c?

```
#include "tc_mode.h"

static void TC_Mode_ApplyParams(TC_ModeState_t *m, TC_Mode_t mode) {
    m->current_mode = mode;

    switch (mode) {
    case TC_MODE_STRAIGHT:
        m->Kp          = KP_STRAIGHT;
        m->Ki          = KI_STRAIGHT;
        m->lambda_base = LAMBDA_BASE_STRAIGHT;
        m->steer_sens  = STEER_SENS_STRAIGHT;
        m->min_tq_ratio= MIN_TQ_RATIO_STRAIGHT;
        break;
```

```
    case TC_MODE_FIGURE8:
        m->Kp          = KP_FIGURE8;
        m->Ki          = KI_FIGURE8;
        m->lambda_base = LAMBDA_BASE_F8;
        m->steer_sens  = STEER_SENS_F8;
        m->min_tq_ratio= MIN_TQ_RATIO_F8;
        break;

    case TC_MODE_TRACK:
    default:
        m->Kp          = KP_TRACK;
        m->Ki          = KI_TRACK;
        m->lambda_base = LAMBDA_BASE_TRACK;
        m->steer_sens  = STEER_SENS_TRACK;
        m->min_tq_ratio= MIN_TQ_RATIO_TRACK;
        break;
    }
}

void TC_Mode_Init(TC_ModeState_t *m) {
    m->current_mode   = TC_MODE_FIGURE8;   // 建議預設繞 8 模式
    m->target_mode    = m->current_mode;
    m->change_pending = false;
    TC_Mode_ApplyParams(m, m->current_mode);
}

/* 車手按按鈕後呼叫: 若靜止就立即切, 若在跑就設成 pending */
void TC_Mode_RequestChange(TC_ModeState_t *m, TC_Mode_t new_mode, float v_kmh) {
    m->target_mode = new_mode;

    if (v_kmh < 1.0f) { // < 1 km/h 視為靜止, 直接切
        TC_Mode_ApplyParams(m, new_mode);
        m->change_pending = false;
    } else {
        m->change_pending = true;
    }
}

/* 每次迴圈呼叫: 若 pending 且已靜止 → 套用新模式 */
void TC_Mode_ApplyIfStopped(TC_ModeState_t *m, float v_kmh) {
```

```
    if (m->change_pending && v_kmh < 1.0f) {
        TC_Mode_ApplyParams(m, m->target_mode);
        m->change_pending = false;
    }
}
```

---

4. ??????????tc_fault.h / tc_fault.c?

4.1 ????tc_fault.h?

```
#ifndef TC_FAULT_H
#define TC_FAULT_H

#include "tc_types.h"

typedef struct {
    uint32_t flags;
    uint32_t wheel_err_count;
    uint32_t imu_err_count;
    uint32_t steer_err_count;
    uint32_t torque_err_count;
    uint32_t can_timeout_count;

    float    omega_prev[4];
    float    steer_prev;
} TC_FaultState_t;

void TC_Fault_Init(TC_FaultState_t *f);
void TC_Fault_Update(const TC_Input_t *in, TC_FaultState_t *f);
bool TC_Fault_IsSafeToRun(const TC_FaultState_t *f);

#endif
```

4.2 ???tc_fault.c?

```
#include "tc_fault.h"
#include "math.h"

#define OMEGA_MIN      -1.0f
#define OMEGA_MAX      200.0f
#define OMEGA_JUMP_MAX  50.0f

#define AX_MAX         20.0f
#define STEER_JUMP_MAX 15.0f   // deg/10ms

#define FAULT_COUNT_MAX 3

void TC_Fault_Init(TC_FaultState_t *f) {
    f->flags = FAULT_NONE;
    f->wheel_err_count = 0;
    f->imu_err_count   = 0;
    f->steer_err_count = 0;
    f->torque_err_count= 0;
    f->can_timeout_count = 0;
    for (int i = 0; i < 4; i++) f->omega_prev[i] = 0.0f;
    f->steer_prev = 0.0f;
}

void TC_Fault_Update(const TC_Input_t *in, TC_FaultState_t *f) {
    /* 1. □□□□ + □□ */
    float omegas[4] = { in->omega_fl, in->omega_fr, in->omega_rl, in->omega_rr };
    bool wheel_fault = false;

    for (int i = 0; i < 4; i++) {
        if (omegas[i] < OMEGA_MIN || omegas[i] > OMEGA_MAX) {
            wheel_fault = true;
        }
        float diff = fabsf(omegas[i] - f->omega_prev[i]);
        if (diff > OMEGA_JUMP_MAX) {
            wheel_fault = true;
        }
        f->omega_prev[i] = omegas[i];
    }

    if (wheel_fault) {
```

```
        if (++f->wheel_err_count >= FAULT_COUNT_MAX) {
            f->flags |= FAULT_WHEEL;
        }
    } else {
        f->wheel_err_count = 0;
    }

    /* 2. IMU 範圍 */
    if (fabsf(in->ax_imu) > AX_MAX) {
        if (++f->imu_err_count >= FAULT_COUNT_MAX) {
            f->flags |= FAULT_IMU;
        }
    } else {
        f->imu_err_count = 0;
    }

    /* 3. 方向盤跳動 */
    float steer_diff = fabsf(in->steer_deg_raw - f->steer_prev);
    if (steer_diff > STEER_JUMP_MAX) {
        if (++f->steer_err_count >= FAULT_COUNT_MAX) {
            f->flags |= FAULT_STEER;
        }
    } else {
        f->steer_err_count = 0;
    }
    f->steer_prev = in->steer_deg_raw;

    /* 4. 扭矩範圍（依車輛實際調整上限） */
    if (in->T_driver < 0.0f || in->T_driver > 1000.0f) {
        f->flags |= FAULT_TORQUE;
    }
}

bool TC_Fault_IsSafeToRun(const TC_FaultState_t *f) {
    /* 有關鍵錯誤就不允許跑 TC */
    if (f->flags & (FAULT_WHEEL | FAULT_TORQUE | FAULT_CAN_TIMEOUT)) {
        return false;
    }
    return true;
}
```

5. TC ?????tc_controller.h / tc_controller.c?

5.1 ????tc_controller.h?

```
#ifndef TC_CONTROLLER_H
#define TC_CONTROLLER_H

#include "tc_types.h"
#include "tc_mode.h"

typedef struct {
    /* 內部狀態 */
    float integral;          // PI 積分
    float v_imu_est;         // 從 IMU 積分的速度估計
    float steer_deg_f;       // 濾波後方向盤 deg
    float ax_peak;           // 用於 μ 估測的加速度峰值

    uint32_t integral_reset_count;
} TC_Internal_t;

void TC_Controller_Init(TC_Internal_t *s);
void TC_Controller_Update(
    const TC_Input_t  *in,
    const TC_ModeState_t *mode,
    TC_Internal_t     *state,
    TC_Output_t       *out
);

#endif
```

5.2 ??????

1. 由前輪輪速估算車速 (v)。
2. 後輪 slip ratio: (\lambda_L = (\omega_{RL} R_{rear} - v) / v)，(\lambda_R) 類似。
3. 方向盤轉角低通濾波。

4. 根據模式參數 + 濾波後轉角 + 車速，算出 Target Slip (\lambda^*)。
5. 計算誤差 (e = \lambda^* - \bar{\lambda})。
6. PI 控制（含積分 Anti-windup + 重置邏輯）。
7. 安全邏輯（低速、嚴重 slip、錯誤）覆蓋一切。
8. 最後輸出 T_cmd。

5.3 ???tc_controller.c?

```
#include "tc_controller.h"
#include "tc_params.h"
#include "math.h"

void TC_Controller_Init(TC_Internal_t *s) {
    s->integral = 0.0f;
    s->v_imu_est = 0.0f;
    s->steer_deg_f = 0.0f;
    s->ax_peak = 0.0f;
    s->integral_reset_count = 0;
}

/* 車速估計（簡化版: 先只用前輪） */
static float TC_EstimateSpeed(const TC_Input_t *in) {
    float v = (in->omega_fl + in->omega_fr) * 0.5f * R_FRONT;
    if (v < 0.1f) v = 0.1f;
    return v;
}

/* Slip Ratio */
static void TC_ComputeSlip(const TC_Input_t *in, float v,
                           float *slip_L, float *slip_R, float *slip_avg) {
    float v_safe = (v > 0.1f) ? v : 0.1f;
    *slip_L = (in->omega_rl * R_REAR - v_safe) / v_safe;
    *slip_R = (in->omega_rr * R_REAR - v_safe) / v_safe;
    *slip_avg = (*slip_L + *slip_R) * 0.5f;
}

/* 方向盤低通濾波 */
static float TC_FilterSteer(float raw_deg, TC_Internal_t *s) {
    s->steer_deg_f = STEER_ALPHA * s->steer_deg_f +
                     (1.0f - STEER_ALPHA) * raw_deg;
```

```
    return s->steer_deg_f;
}

/* 簡易 μ 估測，用 ax 峰值分級 → 回傳倍數 0.5~1.0 */
static float TC_EstimateMu(TC_Internal_t *s, const TC_Input_t *in) {
    float ax_abs = fabsf(in->ax_imu);
    /* 指數平滑追蹤峰值 */
    s->ax_peak = fmaxf(s->ax_peak * 0.95f + ax_abs * 0.05f, ax_abs);

    if (s->ax_peak > 8.5f)   return 1.0f;  // 高 μ
    if (s->ax_peak > 5.5f)   return 0.8f;  // 中 μ
    if (s->ax_peak > 3.5f)   return 0.65f; // 低 μ
    return 0.5f;
}

/* 方向盤為主的 Target Slip（模式 + steer_sens） */
static float TC_ComputeTargetSlip(
    float steer_deg_f,
    float v,
    const TC_ModeState_t *mode)
{
    float d = fabsf(steer_deg_f);
    float k_steer;

    /* 分段線性：可把斜率乘上 mode->steer_sens 調整靈敏度 */
    if (d < 4.0f) {
        k_steer = 1.0f;
    } else if (d < 20.0f) {
        float slope = 0.018f * mode->steer_sens;
        k_steer = 1.0f - slope * (d - 4.0f);    // 直到中彎
    } else {
        float slope2 = 0.01f * mode->steer_sens;
        k_steer = 0.65f - slope2 * (d - 20.0f);
    }

    if (k_steer < 0.4f) k_steer = 0.4f;

    /* 速度補償（高速略放寬） */
    float speed_factor = fminf(v * 3.6f / 40.0f, 1.0f) * 0.03f + 0.97f;
```

```
    float lambda = mode->lambda_base * k_steer * speed_factor;
    if (lambda < 0.08f) lambda = 0.08f;
    return lambda;
}

/* 積分重置邏輯（你之前討論的「防飽和」） */
static void TC_IntegralResetLogic(const TC_Input_t *in,
                                  const TC_ModeState_t *mode,
                                  TC_Internal_t *s,
                                  float v_mps)
{
    float v_kmh = v_mps * 3.6f;

    /* 1) 低速: 直接清零, 起步最安全 */
    if (v_kmh < 2.0f) {
        if (s->integral != 0.0f) {
            s->integral = 0.0f;
            s->integral_reset_count++;
        }
        return;
    }

    /* 2) 駕駛幾乎沒踩油門: 快速衰減 */
    if (fabsf(in->T_driver) < 2.0f) {
        s->integral *= 0.1f;
        return;
    }

    /* 3) 模式剛剛切換（因為你只在靜止時切換, 這裡可省略或保留） */
    (void)mode; // 目前不需要特別判斷, 保留介面
}

/* 主控制: 不含 Safety（Safety 放在 main.c） */
void TC_Controller_Update(
    const TC_Input_t  *in,
    const TC_ModeState_t *mode,
    TC_Internal_t     *state,
    TC_Output_t       *out)
{
    /* 1. 車速 & slip */
```

```
    out->v_mps = TC_EstimateSpeed(in);
    TC_ComputeSlip(in, out->v_mps, &out->slip_L, &out->slip_R, &out->slip_avg);

    /* 2. 方向盤濾波 */
    out->steer_deg_f = TC_FilterSteer(in->steer_deg_raw, state);

    /* 3. 目標 slip（模式 + 方向盤） */
    out->lambda_target = TC_ComputeTargetSlip(out->steer_deg_f, out->v_mps, mode);

    /* 4. 積分重置邏輯 */
    TC_IntegralResetLogic(in, mode, state, out->v_mps);

    /* 5. 自適應 Kp by μ */
    float mu_scale = TC_EstimateMu(state, in);
    float Kp_eff = mode->Kp * mu_scale;
    float Ki_eff = mode->Ki;  // 也可以乘 mu_scale，看你喜好

    /* 6. PI 計算 */
    float error = out->lambda_target - out->slip_avg;

    float p_term = Kp_eff * error;

    state->integral += error * TC_DT_SEC;
    if (state->integral > INTEGRAL_MAX)   state->integral = INTEGRAL_MAX;
    if (state->integral < -INTEGRAL_MAX)  state->integral = -INTEGRAL_MAX;
    float i_term = Ki_eff * state->integral;

    out->delta_T = p_term + i_term;
}
```

6. Main Loop + Safety + CAN?main.c ???

6.1 ?????TIM2 10 ms ???

```
#include "main.h"
#include "tc_types.h"
#include "tc_mode.h"
#include "tc_fault.h"
#include "tc_controller.h"
#include "tc_io.h"   // 你自己實作: CAN 收發, 按鈕處理

TC_ModeState_t   g_mode;
TC_FaultState_t  g_fault;
TC_Internal_t    g_state;
TC_Input_t       g_in;
TC_Output_t      g_out;

/* 初始化 */
void TC_SystemInit(void) {
    TC_Mode_Init(&g_mode);
    TC_Fault_Init(&g_fault);
    TC_Controller_Init(&g_state);
}

/* 10 ms 定時中斷 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM2) {

        /* 1. 讀取 CAN / ADC → 填 g_in */
        TC_IO_ReadInputs(&g_in);     // 需要你自己實作

        /* 2. 錯誤偵測 */
        TC_Fault_Update(&g_in, &g_fault);

        /* 3. 模式在靜止時套用 */
        float v_kmh_fake = 0.0f;    // 這裡可以直接用上次 g_out.v_mps 轉 km/h
        v_kmh_fake = g_out.v_mps * 3.6f;
        TC_Mode_ApplyIfStopped(&g_mode, v_kmh_fake);

        /* 4. 判斷是否允許跑 TC */
        bool safe = TC_Fault_IsSafeToRun(&g_fault);

        if (!safe) {
            /* 4-1. 發生嚴重錯誤 → 關閉 TC, 扭矩交回駕駛或安全模式 */
```

```
    g_out.tc_status = TC_STATUS_FAULT;
    g_out.T_cmd = 0.0f;  // 或者 0.3 * g_in.T_driver, 看你的需求
} else {
    /* 4-2. 正常執行控制（含 PI, 但不含最外層 Safety） */
    TC_Controller_Update(&g_in, &g_mode, &g_state, &g_out);

    float v_kmh = g_out.v_mps * 3.6f;

    /* 5. 最外層 Safety 覆蓋 */
    if (v_kmh < V_TC_OFF_KMH) {
        g_out.T_cmd = g_in.T_driver;
        g_out.tc_status = TC_STATUS_OFF;
    }
    else if (fabsf(g_out.slip_L) > SLIP_SINGLE_MAX ||
             fabsf(g_out.slip_R) > SLIP_SINGLE_MAX) {
        g_out.T_cmd = 0.5f * g_in.T_driver;
        g_out.tc_status = TC_STATUS_SAFETY;
    }
    else if (fabsf(g_out.steer_deg_f) > 30.0f) {
        g_out.T_cmd = g_in.T_driver * g_mode.min_tq_ratio;
        g_out.tc_status = TC_STATUS_SAFETY;
    }
    else {
        /* 正常 PI 輸出 + 飽和 */
        float T_raw = g_in.T_driver * (1.0f + g_out.delta_T);
        float T_min = g_in.T_driver * g_mode.min_tq_ratio;
        float T_max = g_in.T_driver;

        if (T_raw < T_min) T_raw = T_min;
        if (T_raw > T_max) T_raw = T_max;

        g_out.T_cmd = T_raw;
        g_out.tc_status = TC_STATUS_NORMAL;
    }
}

g_out.fault_flags = g_fault.flags;

/* 6. 透過 CAN 發送: T_cmd, 狀態, 模式, fault */
TC_IO_SendOutputs(&g_in, &g_out, &g_mode, &g_fault);
```

```
    }
}
```

---

7. ??????????tc_io.h / tc_io.c ???

你可以:

- 用 GPIO 按鈕改 `g_mode.target_mode`:
  - 短按 MODE: `STRAIGHT → FIGURE8 → TRACK → STRAIGHT...`
  - 呼叫 `TC_Mode_RequestChange(&g_mode, new_mode, v_kmh)`。
- 由 CAN 顯示:
  - 目前模式 `g_mode.current_mode`。
  - `g_out.tc_status`（OFF / NORMAL / SAFETY / FAULT）。
  - 扭矩限縮比例 `g_out.T_cmd / g_in.T_driver`。
  - μ 等級（可以從 `g_state.ax_peak` 推估）。

你可以自行定義 `0x200 ~ 0x20F` 做儀表板訊息。

---

8. ??????????

1. 先鎖在「繞 **8** 模式」，在乾地繞 8 練習，調整:
   - `LAMBDA_BASE_F8` 先從 0.16 → 看實際 slip 波形。
   - `KP_FIGURE8` 3.1 若太慢可到 3.5; 若抖就往下。
2. 直線模式：在直線起步測 `LAMBDA_BASE_STRAIGHT` 0.20 是否太滑 / 太鈍。
3. 賽道模式：上賽道後，再微調 `LAMBDA_BASE_TRACK` 和 `STEER_SENS_TRACK`，讓大轉角出彎更穩。

---

9. ??????

- 模式：決定「基準 λ*、Kp/Ki、最低扭矩比例」。
- 方向盤：決定「當下允許的縱向抓地比例 (k_{steer})」。
- **μ** 估測：決定「 Kp 要多兇」。
- 積分重置：保證起步、收油、模式切換時不會被舊積分害死。
- **Safety**: 永遠在最外層，隨時可以一刀切斷 PI 的輸出。

照這份 `.md` 建專案與檔案，功能會跟我們前面討論的一樣齊全，你可以先把這整段貼進 HackMD，之後要補圖（流程圖、波形）可以在各章下面加上自己的實測截圖。

????Firmware

image

根據 Setting up the Emrax Motor 得知 EEPROm要設定 128 image

然後 Software Users Manual 說大於 60就要選 GROUP2 image

RWD TC ??????????????

tags: `Traction Control` `Control Logic` `Safety` `STM32F446`

---

1. ??????

1.1 ????

- 依照路況與駕駛意圖（油門、方向盤）動態調整後輪 slip ratio ，維持在一個「有抓地、但允許適度打滑」的區間。
- 直線追求最大加速（較高 slip）。
- 繞 8 / 賽道過彎時，優先穩定性（較低 slip，降低過度 oversteer 風險）。
- 在任何感測器異常或條件不明時， 優先安全 ：關閉或大幅限制 TC 介入。

1.2 ?????????10 ms, 100 Hz?

邏輯順序：

1. 讀取感測器與指令：四輪輪速、 IMU、方向盤、 T_driver。
2. 更新錯誤偵測（ Fault 檢查）。
3. 若允許 TC：
   - 估測車速 v。
   - 計算後輪 slip ratio（左右與平均）。
   - 濾波方向盤轉角。
   - 根據模式 + 方向盤 + 車速，計算目標 slip λ^*。
   - 依據誤差 $e = \lambda^* - \bar{\lambda}$，執行 PI 控制，得到 ΔT。
4. 外層安全邏輯覆蓋：低速、嚴重打滑、大轉角、 Fault → 改寫 T_cmd。
5. 將最終 T_cmd 與狀態資訊送出給馬達與儀表。

---

2. ???????? Slip ??

2.1 ????

- 輸入：前輪輪速 $(\omega_{FL}, \omega_{FR})$。
- 輪胎半徑： (R_{front})。

運算:

$$ v = \max\left(\frac{\omega_{FL} + \omega_{FR}}{2} R_{front},\ 0.1 \right) $$

實作重點:

- 只用非驅動輪（前輪）當作車速來源，避免驅動輪打滑時速度估測錯誤。
- 下限 0.1 m/s 避免後面除以 0。

2.2 ?? Slip Ratio

- 輸入: 後輪輪速 (ω_{RL}, ω_{RR})、車速 (v)、後輪半徑 (R_{rear})。

$$ \lambda_L = \frac{\omega_{RL} R_{rear} - v}{\max(v, 0.1)}, \quad \lambda_R = \frac{\omega_{RR} R_{rear} - v}{\max(v, 0.1)} $$

平均 slip:

$$ \bar{\lambda} = \frac{\lambda_L + \lambda_R}{2} $$

物理意義:

- ($\lambda = 0$): 完全滾動，無打滑。
- ($0 < \lambda < 0.3$): 可接受的加速打滑區間。
- ($\lambda > 0.5$): 嚴重打滑，需要強制限扭或收油。

2.3 ???????

方向盤訊號可能有抖動，使用一階低通:

$$ \delta_{f}[k] = \alpha \delta_{f}[k-1] + (1-\alpha) \delta_{raw}[k] $$

- ($\alpha = 0.92$) 左右，對應約 3 Hz 截止頻率，能濾掉高頻雜訊又不延遲太多。

2.4 ???? ? ??

使用縱向加速度最大值作為路面摩擦指標:

- 記錄一個指數平滑峰值:

$$ a_{x,peak}[k] = \max(0.95, a_{x,peak}[k-1] + 0.05, |a_x[k]|, \ |a_x[k]|) $$

- 依 ($a_{x,peak}$) 粗略分級:
  - ($> 8.5,\text{m/s}^2$): 高 μ（乾柏油）。
  - ($5.5 \sim 8.5,\text{m/s}^2$): 中 μ（略濕或一般柏油）。

- (3.5 \sim 5.5,\text{m/s}^2): 低 μ (濕滑地面)。
- (< 3.5,\text{m/s}^2): 很滑 (碎石、草、冰)。

對應一個增益倍數 μ_scale: 0.5–1.0, 用來縮放 Kp。

---

3. ?? Slip ?* ????

3.1 ???????

每個模式有一組基準參數:

- 直線:
  - (K_p) 大、 (\lambda_{base} \approx 0.20)、方向盤敏感度低。
- 繞 8:
  - (K_p) 中、 (\lambda_{base} \approx 0.16)、方向盤敏感度中等。
- 賽道:
  - (K_p) 較小、 (\lambda_{base} \approx 0.14)、方向盤敏感度高 (轉角一大就明顯降 slip)。

這些參數封裝在 `TC_ModeState_t` 中:

- `Kp, Ki`
- `lambda_base`
- `steer_sens`
- `min_tq_ratio`

3.2 ???? ?* ???

令 (d = |\delta_f|) (濾波後方向盤轉角, 單位 deg)。

使用分段線性 + 模式敏感度來估計剩餘縱向抓地比例 (k_{steer}):

1. 小轉角: (d < 4,\deg)
   - 幾乎直線, (k_{steer} = 1.0)。
2. 輕~中彎: (4 \le d < 20,\deg)
   - 以線性方式遞減: [k_{steer} = 1.0 - s_1 (d - 4)]
   - 斜率 (s_1) 由 `steer_sens` 決定 (模式不同斜率不同)。
3. 中~重彎: (d \ge 20,\deg)
   - 再以較小斜率往下掉到下限 (k_{steer,min} \approx 0.4)。

3.3 ????

高速時可接受稍高目標 slip，低速時保守：

- $v_{kmh} = v \times 3.6$。
- 定義

$$\text{speed\_factor} = 0.97 + 0.03 \cdot \min\left(\frac{v_{kmh}}{40}, 1\right)$$

約略在 0.97–1.0 之間微調。

3.4 ?* ????

綜合模式基準、方向盤與速度：

$$\lambda^* = \max\left(\lambda_{base} \cdot k_{steer} \cdot \text{speed\_factor},\ 0.08 \right)$$

- 下限 0.08 避免過度保守導致動力太小。
- 模式越偏「穩定」的（賽道），λ_{base} 越小，且 `steer_sens` 越大（大轉角時 λ* 明顯下降）。

---

4. PI ???????

4.1 ????

$$e = \lambda^* - \bar{\lambda}$$

- $e > 0$：實際 slip 比目標低 → 可以多給一點扭矩。
- $e < 0$：實際 slip 過大 → 需要限縮扭矩。

4.2 ??? Kp

根據 μ 估測結果縮放 Kp：

$$K_{p,eff} = K_p \cdot \text{mu\_scale}$$

- 高 μ：mu_scale ≈ 1.0 → 使用原始 Kp，反應積極。
- 低 μ：mu_scale ≈ 0.5–0.65 → Kp 變小，避免在低抓地路面上震盪。

4.3 PI ???

$$\Delta T = K_{p,eff} e + K_i \int e, dt$$

實作時:

1. 先依當前 Kp_eff 和 Ki_eff 計算 p_term 與 i_term。
2. 積分部分有限制:

[\int e,dt \in [-I_{max}, I_{max}], \quad I_{max} = 1.0]

4.4 ???????Anti-windup?

避免在以下情況發生「積分殘留」:

- 低速起步（ v 很小）。
- 駕駛收油（ T_driver 幾乎為 0）。

設計:

1. 若 (v < 2,\text{km/h}):
   - 直接令 integral = 0。
2. 若 (|T_{driver}| < 2,\text{Nm}):
   - 積分快速衰減，例如 integral *= 0.1。

這樣可避免:

- 上一輪在打滑時累積的積分影響下一次起步。
- 收油後， TC 仍因舊積分而錯誤限扭。

---

5. ???????????

5.1 ??????

根據 PI 結果先算一個「候選」扭矩:

[T_{raw} = T_{driver} (1 + \Delta T)]

再以模式決定的最小扭矩比例做內部飽和:

- (T_{min} = T_{driver} \cdot \text{min_tq_ratio})
- (T_{max} = T_{driver})

[T_{pi} = \min(\max(T_{raw}, T_{min}), T_{max})]

5.2 ?????????? TC

如果速度很低（例如 $v < 3$ km/h）：

- 直接讓馬達扭矩 = 駕駛要求：

$$T_{cmd} = T_{driver},\quad \text{TC 狀態 = OFF}$$

理由：

- 起步時 slip 訊號還不穩定，容易誤判。
- 低速起步可先依靠駕駛腳感。

5.3 ??????????

若任一後輪 slip 超過安全上限（例如 $|\lambda_L|$ or $|\lambda_R| > 0.4$）：

- 強制大幅限扭，保護輪胎與穩定：

$$T_{cmd} = 0.5\, T_{driver}\quad \text{或更低}$$

- TC 狀態標記為 SAFETY。

5.4 ??????????

若濾波後方向盤轉角非常大（例如 $|\delta_f| > 30°$）：

- 視為重彎或近似 U-turn，必須高度保守：

$$T_{cmd} = T_{driver} \cdot \text{min\_tq\_ratio}$$

- 對應模式：
  - 直線模式：min_tq_ratio ≈ 0.55。
  - 繞 8 模式：≈ 0.40。
  - 賽道模式：≈ 0.30。

5.5 Fault ??????

若 `TC_Fault_IsSafeToRun()` 回傳 false：

- 代表感測器或扭矩命令有重大異常。
- 建議作法：
  - 將 `tc_status` 設為 FAULT。
  - 視車輛需求：
    - 直接設 `T_cmd = 0`（完全不輸出，交回原車 ECU 控制）。
    - 或設成一個安全的固定比例（例如 0.3 * T_driver，當作 limp-home）。
- 同時透過 CAN 發出錯誤碼與故障旗標，讓儀表或 log 紀錄。

6. ?????????? / ? 8 / ???

6.1 ????

- 只在靜止時切換模式　　：避免行駛中改變控制器行為，造成車手感覺突變。
- 模式改變時，重新套用一組參數：　　(K_p, K_i, \lambda_{base}, \text{steer_sens}, \text{min_tq_ratio})。
- 模式切換當下會重置部分內部狀態（例如積分、方向盤濾波器）。

6.2 ??????

1. 車手按　　MODE 按鈕，或經由　　CAN 設定　　`target_mode`。
2. 若當下車速　　(< 1,\text{km/h}):
   - 立即套用新模式參數到　　`current_mode`。
   - 將　integral = 0、steer_deg_f = 0。
3. 若當下車速　　(\ge 1,\text{km/h}):
   - 只更新　　`target_mode`，並設　　`change_pending = true`，但不改變實際控制。
   - 等車輛之後停下來時，在控制迴圈中偵測　　v 變小，才套用新模式。

6.3 ??????????

- 儀表顯示目前　　`current_mode`，以及若有　　`change_pending`
  則以閃爍或圖示表示「等待靜止後切換」。
- 這樣車手可以預先選好下一個模式，到起點或　　PIT 停下來時會自動接上新參數。

---

7. ???????

設計這套　　TC 時，可以用「由內而外」的分層方式來理解：

1. 最內層：　　**PI** 控制
   - 只管：誤差　　e → ΔT。
   - 內含自適應　　Kp（看　　μ）與積分　　Anti-windup。
2. 中間層：目標　　**slip** 生成
   - 只管：模式　　＋ 方向盤　　＋ 車速　　→ λ*。
   - 反映駕駛意圖（直線　　/ 繞　8 / 賽道）與當下彎道狀態。
3. 外層：安全機制
   - 只管：是否在危險狀態（低速、嚴重打滑、大轉角、　　Fault）。
   - 一旦觸發，會直接覆蓋內層計算結果，強制進入保守或關閉狀態。

依照這個分層結構，你可以在不破壞外層 Safety 的前提下，調整與實驗內層的 Kp/Ki、λ^* 表格與 μ 估測演算法，使車輛在不同場景（直線、繞 8、賽道）都有一致且可預期的手感與安全邊界。