EMP-S

運動控制函式庫

使用手冊

版本 ：V.1.01

日期 ：2021.07

https://www.epcio.com.tw

目 錄

1. 運動控制函式庫簡介

EtherCAT 全數位運動控制軟體平台(EtherCAT Motion Control Platform Soft-Motion, **EMP-S**)所提供的**運動控制函式庫**(Motion Control Command Library, **MCCL**)，以 EtherCAT(Ethernet for Control Automation Technology；乙太網路控制自動化技術)通訊作為基礎，使用者即可透過乙太網路傳送資料封包與 EtherCAT 從站(伺服驅動器、周邊 I/O 裝置等)進行資料交換與控制。

EMP-S 加入即時性子系統，將 Windows 作業系統從通用型作業系統(General Purpose Operating System, GPOS)轉變為即時作業系統(Real-time Operating System, RTOS)，確保執行多軸運動控制的即時性與可靠性，使控制週期時間(Cycle time)最小為 250 μs。EMP-S 具備 EtherCAT 通訊協議的分散式時鐘(Distributed Clock, DC)機制，保證各從站間同步誤差小於 1 μs，對伺服驅動器及周邊 I/O 下達相容於 CANOpen Over EtherCAT(CoE)通訊協定的運動命令、讀取目前命令位置、編碼器回授位置、EtherCAT I/O 狀態和 ADC 輸入電壓與原點復歸等。EMP-S 可使用在 WINDOWS 7 與 WINDOWS 10 作業平台，並支援 Visual C++與.Net Framework 開發環境。

MCCL 提供 3D 空間中點對點、直線、圓弧、圓、螺線等運動的軌跡規劃函式；除此之外，MCCL 並提供了運動空跑、運動延遲、微動/吋動/連續吋動、運動暫停、繼續、棄置等操作函式。

在軌跡規劃功能方面可設定不同的加/減速時間、加/減速曲線型式、進給速度、最大進給速度與最大加速度；MCCL 也包含軟、硬體過行程保護、平滑運動、動態調整進給速度及錯誤訊息處理等功能。在定位控制方面，MCCL 也提供齒輪齒隙與間隙補償等功能。

在 EtherCAT 系統功能裡，針對 EtherCAT 驅動器方面，使用者可以利用 MCCL 讀取 Home 接點與 Limit Switch 接點的訊號，也可輸出 Servo-On/Off 訊號；MCCL 也提供取得驅動器錯誤碼與層級，以及重置驅動器錯誤(Fault Reset)的功能；使用者可以即時讀取編碼器的計

數值。MCCL 也提供軟體 Watch Dog 功能。串接 EtherCAT 從站的專用輸出入控制硬體介面卡時，MCCL 提供 I/O 接點訊號控制功能。串接 EtherCAT 從站的專用類比電壓輸入控制卡時，MCCL 提供讀取輸入的電壓值(-10V ~ 10V)。

EMP-S 的架構不僅硬體配線維護簡單、抗雜訊干擾能力高、安全可靠性高、節省成本，使用者即可快速開發、整合系統。

2. MCCL 功能

2.1 軟體規格

■ 作業系統環境

✓ WINDOWS 7 (32-bit / 64-bit)

✓ WINDOWS 10 (64-bit)

■ 開發環境

✓ Visual C++

✓ Visual C# .Net

■ 使用 MCCL 時，開發專案時所須附加的檔案

	檔案名稱
Visual C++	MCCL.h MCCL_RTX.h MCCL_Client.lib MCCL_Client.dll
Visual C# .Net	MCCL.cs MCCL_Client.dll

■ 執行環境

✓ RTX Runtime

■ 執行時，須使用的 RTX 檔案

	檔案名稱
32-bit	EcServer.rtss emllRTL8169.rtdll emllI8254x.rtdll

64-bit	EcServer.rtss EcMaster.rtdll emllRTL8169.rtdll emllI8254x.rtdll

2.2 運動軸定義

MCCL 的設計目的是針對三軸直角正交(X-Y-Z)，外加五軸輔助軸(U、V、W、A、B)的運動平台，提供運動控制的功能。如 Figure 2.2.1 所示，U、V、W、A、B 為五個輔助軸，代表五個獨立軸向。

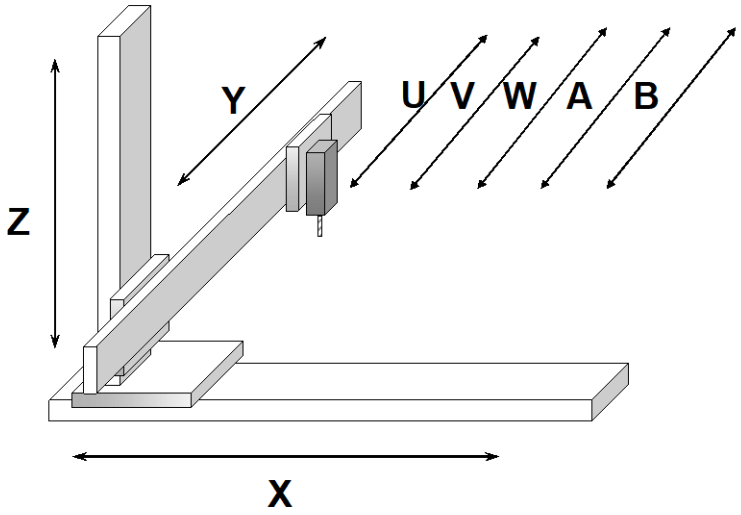


Figure 2.2.1 三軸直角正交(X-Y-Z)，外加五個輔助軸(U、V、W、A、B)

MCCL 提供最大的同動控制軸數為 8 軸，使用者可以使用 EMP-S 執行 1 至 8 軸同動或不同動控制。使用者給定的運動命令可選用絕對、增量座標值，不論使用者選用何種座標值，本函式庫皆會記錄運動位置的絕對座標值(相對於原點)。

2.3 函式庫操作特性

MCCL 運動命令的執行過程是運動函式先將運動命令(OP Code)存放(Put)在 Group 各自的運動命令佇列(Motion Command Queue)中，而***非立即執行***(有關 Group 請參考"**2.5.1 啟動運動控制函式庫**"此章節的說明)；然後 MCCL 會使用先入先出(First In First Out, FIFO)的方式，從佇列中擷取(Get)運動命令進行解譯(Interpolate)及粗插值運算(參考 Figure 2.3.1)。但存放與取出行為並非同步(Asynchronous)動作，並不需要等到運動命令執行完成，即可將新的運動命令送到運動命令佇列中。

由於每個 Group 擁有各自的運動命令佇列，因此可同時執行屬於不同 Group 的運動命令。Figure 2.3.1 為 Group 0 運動命令佇列的操作過程，屬於同一個 Group 的運動命令將被依序執行。

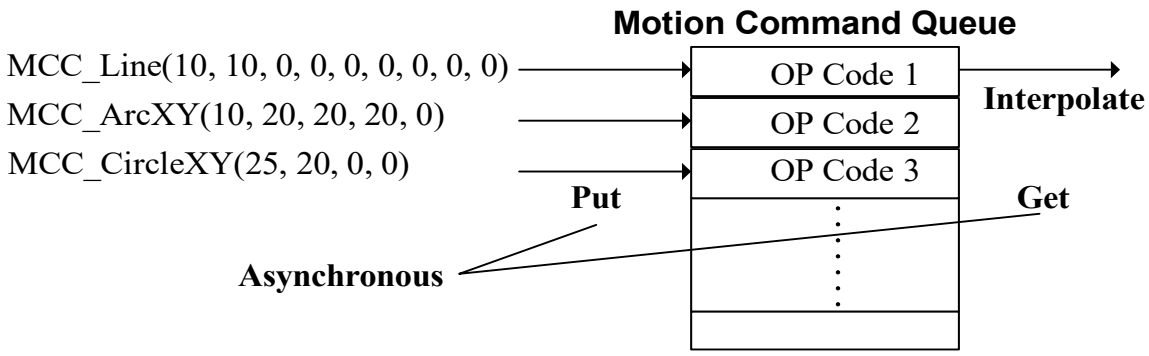


Figure 2.3.1 運動命令佇列

使用 MCC_CreateGroup()建立一 Group(運動群組)時，運動命令佇列預設為可儲存 10000 筆命令。使用 MCC_CreateGroupEx()則可設定運動命令佇列，MCC_CreateGroupEx()的函式原型如下：

```
MCC_CreateGroupEx(  int xMapToCh,
                    int yMapToCh,
                    int zMapToCh,
```

int *uMapToCh*,
int *vMapToCh*,
int *wMapToCh*,
int *aMapToCh*,
int *bMapToCh*,
int *xMapToCh*,
int nM*otionQueueSize*);

其中

nM*otionQueueSize*：自訂運動命令佇列的大小

使用 MCC_GetCmdQueueSize()可讀取運動命令佇列的大小。

MCCL 會將下列命令先存放在運動命令佇列中，在適當條件取出佇列中第一筆命令，並執行對應運動：

A. 直線運動函式

MCC_Line()

B. 圓弧運動函式

MCC_ArcXYZ()　　MCC_ArcXYZ_Aux()
MCC_ArcXY()　　MCC_ArcXY_Aux()
MCC_ArcYZ()　　MCC_ArcYZ_Aux()
MCC_ArcZX()　　MCC_ArcZX_Aux()
MCC_ArcThetaXY()　　MCC_ArcThetaYZ()
MCC_ArcThetaZX()

C. 圓運動函式

MCC_CircleXY()　　MCC_CircleYZ()
MCC_CircleZX()

MCC_CircleXY_Aux()　　MCC_CircleYZ_Aux()
MCC_CircleZX_Aux()

D. 螺線運動函式

MCC_HelicalXY_Z()　　MCC_HelicalYZ_X()
MCC_HelicalZX_Y()
MCC_HelicalXY_Z_Aux()　　MCC_HelicalYZ_X_Aux()
MCC_HelicalZX_Y_Aux()

E. 點對點運動函式

MCC_PtP()　　MCC_PtPX()　　MCC_PtPY()
MCC_PtPZ()　　MCC_PtPU()　　MCC_PtPV()
MCC_PtPW()　　MCC_PtPA()　　MCC_PtPB()

F. 吋動與連續吋動函式

MCC_JogSpace()　　MCC_JogConti()
MCC_JogPulse()

G. 平滑運動函式

MCC_EnableBlend()　　MCC_DisableBlend()

H. 運動延遲函式

MCC_DelayMotion()

若運動命令佇列已滿，上述函式的傳回值為 *COMMAND_BUFFER_FULL_ERR*(-2)，表示此筆運動命令不被接受。

注意：

以下列出使用者在運動命令常見的錯誤用法，使用情境：使 Group 0 之 X 軸移動到座標 10 的位置後，輸出一 Servo-On(IO)訊號，再將此軸移動到座標 20 的位置。程式可能的寫法如下：

```
MCC_Line(10, 0, 0, 0, 0, 0, 0, 0, 0);
MCC_SetServoOn(0);
MCC_Line(20, 0, 0, 0, 0, 0, 0, 0, 0);
```

表示在 MCC_Line()存放於運動命令佇列後(尚未真正執行)，將立即執行 MCC_SetServoOn()，因 MCC_SetServoOn()並不存放於運動命令佇列中而是直接執行，因此在實際位置到達座標 10 之前，Servo-On 訊號將早已送出，此項操作特性須特別注意。

如要求 X 軸移動到座標 10 的位置後才執行 Servo-On 訊號輸出，則使用者須進行額外的判斷，也就是須自行檢查系統的運動狀態或目前座標，來控制訊號的輸出動作，下面為一個簡單的使用範例：

```
//第 0 個 Group 中的 X 軸移動到座標 10 的位置

MCC_Line(10, 0, 0, 0, 0, 0, 0, 0, 0);

//若 MCC_GetMotionStatus()的函式傳回值等於 GMS_STOP，
//表示目前全部的運動命令皆已執行完成
while( MCC_GetMotionStatus(0) != GMS_STOP )
{
 //因使用“while”判斷條件，為避免系統鎖死，影響系統的操作
 //需呼叫 MCC_TimeDelay()釋放 CPU 的使用權
 //在 UI 元件上，請使用 Thread 或 Timer 的方式取得運動狀態與
 //後續判斷動作
 MCC_TimeDelay(10); //   Sleep 10 ms
}

//輸出 Servo-On 訊號
MCC_SetServoOn(0);
```

```
//第 0 個 Group 中的 X 軸移動到座標 20 的位置
MCC_Line(20, 0, 0, 0, 0, 0, 0, 0, 0);
```

2.4 機構參數與Group參數設定

2.4.1 機構參數

MCCL 利用機構參數來定義使用者的機構平台(Table)特性與驅動器使用型式；並利用機構參數規劃相對於邏輯原點的座標系統、座標系統邊界值及各軸的最大安全進給速度。

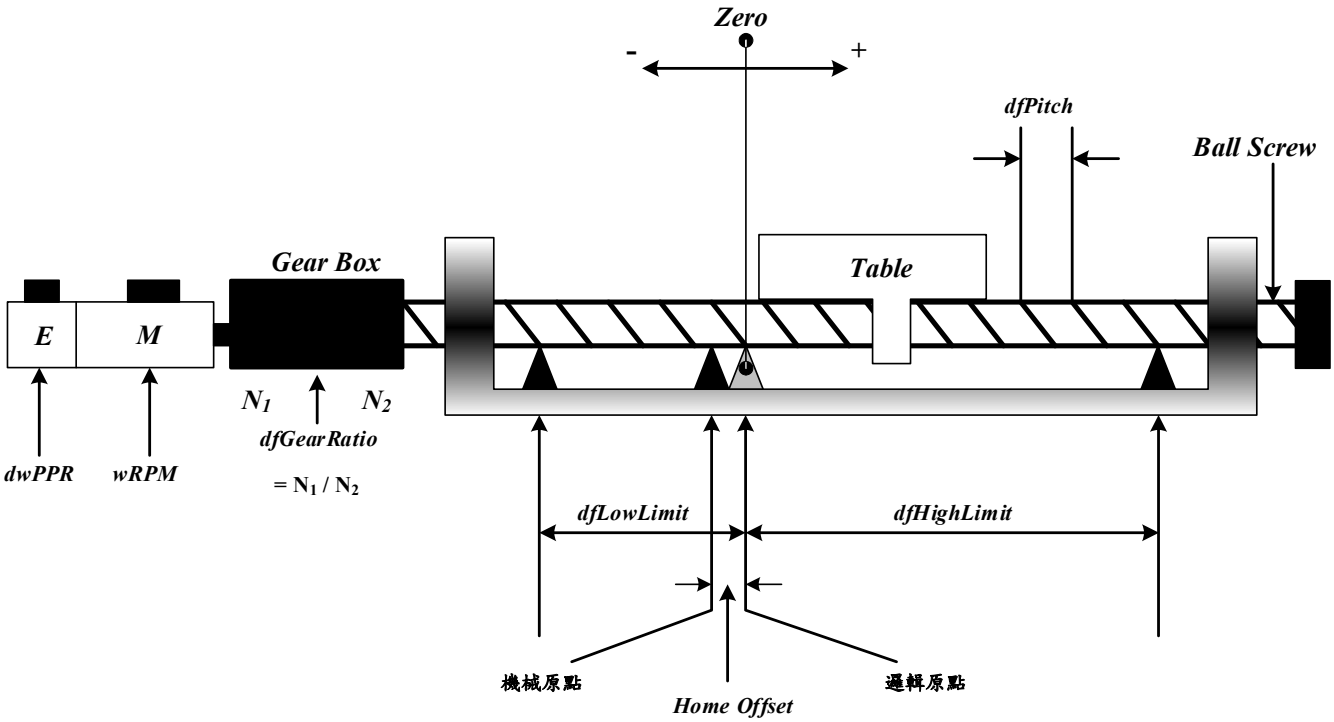


Figure 2.4.1 機構平台特性

以下為機構參數的內容與詳細說明：

```
typedef struct _SYS_MAC_PARAM
{
    WORD        wPosToEncoderDir;
    WORD        wRPM;
    DWORD       dwPPR;
    double      dfPitch;
```

```
    double      dfGearRatio;
    double      dfHighLimit;
    double      dfLowLimit;
    double      dfHighLimitOffset;
    double      dfLowLimitOffset;
    WORD        wPulseMode;
    WORD        wPulseWidth;
    WORD        wCommandMode;
    WORD        wPaddle;
    WORD        wOverTravelUpSensorMode;
    WORD        wOverTravelDownSensorMode;
} SYS_MAC_PARAM;
```

wPosToEncoderDir：方向調整參數

0　　輸出命令不反向

1　　輸出命令反向

此項參數用來修正當運動命令方向與期望的機構運動方向不同時的現象，若送出正向運動命令，例如使用 MCC_JogSpace(10, 10, 0, 0)，但因馬達配線的因素，使機構實際上往使用者定義的負方向移動，此時可設定此項參數為"1"，使運動命令方向與期望的機構運動方向保持一致(不須更改馬達配線)。

wRPM：馬達最大安全轉速

各軸馬達最大安全轉速。**各軸進行點對點快速移動時**，由所設定的速度換算而得的各軸馬達轉速將不會超過 *wRPM* 設定值。

➔ *See Also*　　MCC_SetPtPSpeed()

dwPPR：馬達軸心每旋轉一圈，編碼器所增加的計數值，或每旋轉一圈所須的 pulse 數(設定值須與選用的 EtherCAT 伺服驅動器參數一致)。

使用線性馬達時 *dfPitch* 與 *dfGearRatio* 皆應設定為 1。另外，線性馬達並無 *dwPPR* 相關的定義，且要求移動的距離通常是以 pulse 為單位，此時可將 *dwPPR* 設定為 1，如此可更改 MCCL 所使用的單位為 pulse。例如，當要求 X 軸移動 1000 pulses 時，可呼叫 MCC_Line(1000, 0,0,0,0,0,0,0,0) ， X 軸將輸出 1000 pulses ；當使用 MCC_SetFeedSpeed(500)時，表示要求線性馬達的速度為每秒移動 500 pulses。

dfPitch：導螺桿間隙值

導螺桿每旋轉一圈，機構平台所移動的距離，**單位為 UU(User Unit, 為使用者自訂之單位)。**如無配置導螺桿則此值應設定為 1。

dfGearRatio：齒輪箱減速比

連接馬達軸心與導螺桿之齒輪箱之雙向齒輪比，該值可用齒輪之齒數計算得知，或是簡單定義為「導螺桿每轉一圈，馬達所轉動的圈數」。如無配置齒輪箱則此值應設定為 1。

dfHighLimit：正方向過行程軟體邊界(或稱正方向邊界)

該值為正方向相對於邏輯原點所允許的最大位移量，單位為 UU。

➔ *See Also* MCC_SetOverTravelCheck()

dfLowLimit：負方向過行程軟體邊界(或稱負方向邊界)

該值為負方向相對於邏輯原點所允許的最大位移量，通常給定負值，單位為 UU。

dfHighLimitOffset：保留欄位，使用者須設定為 0

dfLowLimitOffset：保留欄位，使用者須設定為 0

wPulseMode：脈衝輸出格式(EMP 無作用)

wPulseWidth：脈衝輸出寬度(EMP 無作用)

wCommandMode：運動命令輸出型式(EMP 無作用)

wPaddle：保留欄位，使用者須設定為 0

wOverTravelUpSensorMode：正極限開關(Limit Switch +)之配線方式，請參考選用的 EtherCAT 伺服驅動器的說明文件。

SL_NORMAL_OPEN	Active High
SL_NORMAL_CLOSE	Active Low
SL_UNUSED	不檢查是否已碰觸極限開關；指定軸如未安裝極限開關可使用此選項

wOverTravelDownSensorMode：負極限開關(Limit Switch -)之配線方式，請參考選用的 EtherCAT 伺服驅動器的說明文件。

SL_NORMAL_OPEN	Active High
SL_NORMAL_CLOSE	Active Low
SL_UNUSED	不檢查是否已碰觸極限開關；指定軸如未安裝極限開關可使用此選項

使用極限開關功能須依據極限開關配線方式正確設定 *wOverTravelUpSensorMode* 與 *wOverTravelDownSensorMode*。可使用 MCC_GetLimitSwitchStatus()檢查配線方式的設定值是否正確。在未碰觸極限開關時，如果使用 MCC_GetLimitSwitchStatus()所獲得的極限開關為 Active 狀態，則表示配線方式設定值錯誤，應更改 *wOverTravelUpSensorMode* 或 *wOverTravelDownSensorMode* 的設定值。

但要使極限開關正常運作，除了必須正確設定極限開關的配線方

式外，尚必須呼叫 MCC_EnableLimitSwitchCheck()，如此 *wOverTravelUpSensorMode* 與 *wOverTravelDownSensorMode* 的設定才能生效。

但 *wOverTravelUpSensorMode* 與 *wOverTravelDownSensorMode* 如設定為 SL_UNUSED，則呼叫 MCC_EnableLimitSwitchCheck()並無任何意義。

當極限開關保護功能開啟時，在碰觸到該軸運動方向的極限開關時(例如往正方向移動且碰觸到正向極限開關，或往負方向移動且碰觸到負向極限開關)，將會停止輸出 Group 的運動命令並產生錯誤記錄(有關 MCC_EnableLimitSwitchCheck()的參數內容請參考"***EMP-S 運動控制函式庫參考手冊* C. 過行程保護**")。

MCC_EnableLimitSwitchCheck()通常會與 MCC_GetErrorCode()搭配使用，利用不斷呼叫 MCC_GetErrorCode ()可獲知系統是否因碰觸到極限開關而產生錯誤記錄(錯誤代碼 0xF701 ~ 0xF708 分別代表 X ~ B 軸碰觸極限開關)；當發現碰觸極限開關之錯誤時，一般作法可能是：在螢幕上顯示訊息告知操作員，然後在程式中呼叫 MCC_ClearError()清除錯誤，此時系統可再往反方向退開極限開關。

在確定機構參數中各項欄位的內容後，可以使用 MCC_SetMacParam()設定機構參數，下面為使用範例：

```
SYS_MAC_PARAM     stAxisParam;

memset(&stAxisParam, 0, sizeof(SYS_MAC_PARAM));

stAxisParam.wPosToEncoderDir          = 0;
stAxisParam.dwPPR                     = 10000;
stAxisParam.wRPM                      = 3000;
stAxisParam.dfPitch                   = 1.0;
stAxisParam.dfGearRatio               = 1.0;
stAxisParam.dfHighLimit               = 50000.0;
```

```
stAxisParam.dfLowLimit                    = -50000.0;
stAxisParam.wPulseMode                    = 0;
stAxisParam.wPulseWidth                   = 0;
stAxisParam.wCommandMode                  = 0;
stAxisParam.wOverTravelUpSensorMode   =SL_UNUSED; //not check
stAxisParam.wOverTravelDownSensorMode=SL_UNUSED;//not check
MCC_SetMacParam(&stAxisParam, 0); //設定第 0 軸的機構參數
```

一般須在使用 MCC_InitSystem()前將機構參數設定完成，各軸的機構參數須分開設定。

➔ *See Also* MCC_GetMacParam()

注意：

若在呼叫過 MCC_InitSystem()後再去改變機構參數，則須另外呼叫 MCC_UpdateParam()，系統才能反應新的設定值。但須注意：*使用 **MCC_UpdateParam()** 的結果與使用 **MCC_ResetMotion()** 的結果相似，系統將回復到呼叫 **MCC_InitSystem()** 後的初始狀態。*

2.4.2 設定 Group(運動群組)參數

在使用 MCCL 之前，必須先建立所需要的 Group(運動群組)。Group 可視為一獨立之運動系統，其內部各運動軸通常存在相依的關係，明顯的例子為 X-Y-Z 平台。

MCCL 使用 Group 的操作概念，所提供的運動控制函式大部分是以 Group 為單位來操作。每個 Group 皆包含了 X、Y、Z、U、V、W、A、B 等 8 個運動軸，各 Group 間相互獨立，並不影響彼此間的運作；MCCL 最多提供 32 軸運動控制，可支援 1 ~ 32 個 Groups，每個 Group

最多可支援 8 軸。

以 Figure 2.4.4 為例，目前使用了 2 個 Groups 與 6 個 Channels (EtherCAT 伺服驅動器的從站)，其中 Group 0 的 X、Y、Z 軸的軌跡規劃結果將分別從 Channel 0、Channel 1、Channel 2 輸出，並忽略 U、V、W、A、B 軸的軌跡規劃結果；而 Group 1 的 X、Y、Z 軸的軌跡規劃結果將分別從 Channel 3、Channel 4、Channel 5 輸出，並忽略 U、V、W、A、B 軸的軌跡規劃結果。

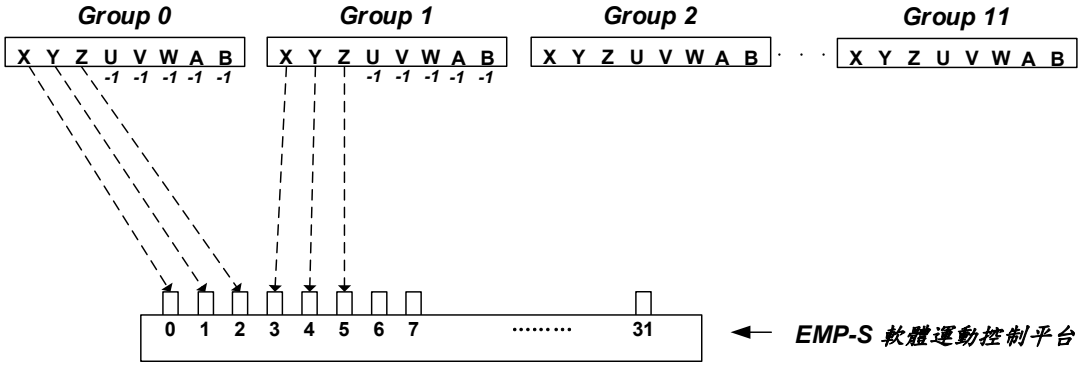


Figure 2.4.4 Group 參數設定

此時程式的寫法可能如下：

```
int    nGroup0, nGroup1;

MCC_CloseAllGroups();                    //關閉全部 Group

nGroup0 = MCC_CreateGroup(    0, //X 軸對應到 Channel 0
                              1, //Y 軸對應到 Channel 1
                              2, //Z 軸對應到 Channel 2
                              -1, //U 軸不作對應
                              -1, //V 軸不作對應
                              -1, // W 軸不作對應
                              -1, //A 軸不作對應
```

```
                                    -1 //B 軸不作對應
                                    );

nGroup1 = MCC_CreateGroup(      3, //X 軸對應到 Channel 3
                                    4, //Y 軸對應到 Channel 4
                                    5, //Z 軸對應到 Channel 5
                                    -1,//U 軸不作對應
                                    -1,//V 軸不作對應
                                    -1,//W 軸不作對應
                                    -1,//A 軸不作對應
                                    -1 //B 軸不作對應
                                    );
```

MCC_CreateGroup()之傳回值代表了新建立之 Group 的編號，此編號將會在往後呼叫運動函式時用到。例：若要使 Group 1 的 X、Y、Z 軸移動到座標位置 10，在程式中就須寫成 MCC_Line(10, 10, 10, 0, 0, 0, 0, 0, ***nGroup1***)；此時 Channel 3、Channel 4、Channel 5 將負責輸出此 Group 1 的 X、Y、Z 軸插值的結果。

```
MCC_Line(10, 10, 10, 0, 0, 0, 0, 0, nGroup0);    //Command 0
MCC_Line(20, 20, 20, 0, 0, 0, 0, 0, nGroup0);    //Command 1
MCC_Line(10, 10, 10, 0, 0, 0, 0, 0, nGroup1);    //Command 2
MCC_Line(20, 20, 20, 0, 0, 0, 0, 0, nGroup1);    //Command 3
```

使用上述的 Group 設定內容，Group 0 將執行 Command 0，並從第個 Channel 0、Channel 1、Channel 2 輸出 X、Y、Z 軸的軌跡規劃結果。Group 0 在完成 Command 0 後，才會再執行屬於相同 Group 之 Command 1。

因各 Group 間獨自運作，故 Group 1 不須等待 Group 0 完成 Command 0，將直接執行 Command 2，並從第 Channel 3、Channel 4、

Channel 5 輸出 X、Y、Z 軸的軌跡規劃結果。而 Group 1 完成 Command 2 後，會再執行屬於相同 Group 之 Command 3。

在啟動 MCCL 前如未建立任何 Group，則 MCCL 會使用預設值運作，預設為僅開啟 Index 為 0 之 Group，且其 X、Y、Z、U、V、W、A、B 軸分別對應到 Channel 0 ~ 7 輸出。

重點資訊：

1. Group 與 Group 間互相不影響。
2. Group 皆包含了 X、Y、Z、U、V、W、A、B 等八個運動軸，各運動軸可規劃是否對應至 Channel 輸出，但 **Group 中須至少有一運動軸作實際對應；且不允許有兩個以上的運動軸對應至同一 Channel**。
3. 為降低 MCCL 對 CPU 的使用率，**所使用的 Group 個數應愈少愈好**。

2.5 啟動與結束EcServer、運動控制函式庫

2.5.1 啟動 EcServer

在開始進行使用運動控制函式庫之前，必須先使用MCC_StartEcServer()，啟動 EcServer 應用程式(檔名：EcServer.rtss)，此檔案存放位置須與執行檔同一目錄(若為 64-bit 系統，動態函式庫：EcMaster.rtdll、emllRTL8169.rtdll 與 emllI8254x.rtdll 亦放置在此目錄)。

再使用 MCC_RtxInit()建立與 EcServer 通訊初始化的功能。

2.5.2 啟動運動控制函式庫

在開始使用 MCCL 時須先設定下面幾項參數的內容，包括：

a. 設定機構參數
   使用 MCC_SetMacParam()
b. 設定運動群組(Group)參數
   使用 MCC_CreateGroup()

未完成這幾項參數的內容設定，或進行這幾項步驟時產生錯誤，將無法使用 MCCL 中的其它函式。機構參數與 **Group**(運動群組)參數的設定請參考”**2.4 機構參數與 Group 參數設定**”與”***EMP-S 運動控制函式庫範例手冊***”的範例說明。

使用 MCC_InitSystem()啟動 MCCL，MCC_InitSystem()的函式宣告如下：

MCC_InitSystem(int *nInterpolateTime*);

參數 *nInterpolateTime* 為插值時間(請參考”**2.6.4 插值時間**”此章

節的說明)，單位為 ms，設定值為 1。

2.5.3 結束運動控制函式庫

如要結束 MCCL 只須使用 MCC_CloseSystem()即可。

2.5.4 結束 EcServer

使用 MCC_RtxClose()關閉且停止與 EcServer 之通訊。

2.6 運動控制

2.6.1 座標系統

座標系統包括下列功能：

I. 選擇絕對或增量座標系統

➔ *See Also* MCC_SetAbsolute()
MCC_SetIncrease()
MCC_GetCoordType()

II. 讀取目前位置座標值

➔ *See Also* MCC_GetCurPos() MCC_GetCurRefPos()
MCC_GetPulsePos()

III. 開啟/關閉軟體過行程檢查功能

使用 MCC_SetOverTravelCheck()開啟此項功能後，MCCL 在計算完每一個插值點時，會檢查此插值點是否會超出各軸的有效工作區間；若判斷已超出工作區間時，則不再對運動控制平台送出命令。使用者可使用 MCC_GetErrorCode()查詢訊息代碼(請參考"***EMP-S 運動控制函式庫參考手冊*** **III. 錯誤訊息代碼**" 此章節的列表)，獲知是否已超出各軸的有效工作區間。

➔ *See Also* MCC_GetOverTravelCheck()
MCC_GetErrorCode()

IV. 開啟/關閉硬體極限開關檢查功能

此項功能請參考"**2.4.1 機構參數**"此章節的說明。

➔ *See Also* MCC_EnableLimitSwitchCheck()
MCC_DisableLimitSwitchCheck()
MCC_GetLimitSwitchStatus()

2.6.2 基本軌跡規劃

MCCL 提供直線、圓弧、圓、螺線等運動(統稱為一般運動)與點對點運動軌跡規劃的功能，在使用這些功能前應先針對機構特性及特殊需求設定加減速型式(S 型或梯型)、加減速時間與進給速度。

I. 一般運動 (直線、圓弧、圓、螺線運動)

一般運動包括直線、圓弧、圓、螺線等多軸同動運動，在使用一般運動的函式時，通常會檢查這些函數的傳回值，傳回值如果小於 0 表示運動命令不被接受，不被接受的原因與傳回值的定義請參考”***EMP-S 運動控制函式庫參考手冊*** **II. MCCL 函式庫**”；傳回值如果大於或等於 0，表示 MCCL 對此運動命令的命令編碼，使用者可從這些運動命令編碼追蹤命令的執行情況。可以使用 MCC_ResetCommandIndex()重置此運動命令編碼值，重新從 0 開始計數。

A. 直線運動

使用直線運動函式時，只須給定各軸的目的位置或位移量，此時將依照給定的進給速度運動，預設的加減速時間為 300ms。

➔ *See Also* MCC_Line()

B. 圓弧運動

呼叫圓弧運動函式時，只須給定參考點與目的點的座標值，此時將依照給定的進給速度運動，預設的加減速時間為 300ms。**MCCL 也提供 3-D 圓弧運動函式**。

➔ *See Also*

MCC_ArcXYZ()	MCC_ArcXYZ_Aux()
MCC_ArcXY()	MCC_ArcXY_Aux()
MCC_ArcYZ()	MCC_ArcYZ_Aux()

MCC_ArcZX()　　MCC_ArcZX_Aux()
MCC_ArcThetaXY()　　MCC_ArcThetaYZ()
MCC_ArcThetaZX()

C. 圓運動

呼叫圓運動函式時，只須給定圓心的座標值，並指定運動方向為順時針或逆時針，此時將依照給定的進給速度運動，預設的加減速時間為 300ms。

➔ *See Also*　　MCC_CircleXY()　　MCC_CircleXY_Aux()
MCC_CircleYZ()　　MCC_CircleYZ_Aux()
MCC_CircleZX()　　MCC_CircleZX_Aux()

D. 螺線運動

呼叫螺線運動函式時，須給定圓周運動的圓心座標值、螺線運動所移動的距離值(垂直於圓心座標軸)與移動的角度值，此時將依照給定的進給速度運動，預設的加減速時間為 300ms。

➔ *See Also*　　MCC_HelicalXY_Z()
MCC_HelicalYZ_X()
MCC_HelicalZX_Y()
MCC_HelicalXY_Z_Aux()
MCC_HelicalYZ_X_Aux()
MCC_HelicalZX_Y_Aux()

E. 一般運動的加減速時間及進給速度設定

欲設定一般運動之加減速時間可使用 MCC_SetAccTime()及 MCC_SetDecTime()；欲設定進給速度時使用 MCC_SetFeedSpeed()，單位為 UU/sec。MCCL 計算一般運動之進給速度時只考慮 X/Y/Z 三軸，而 U/V/W/A/B 軸僅是配合前三軸運動同時開始及結束(作直線運

動)；但若此筆運動命令之 X/Y/Z 三軸無位移，則所設定之進給速度即改為指定 U/V/W/A/B 軸行程最長之速度，其餘四軸則配合同時開始及結束。

使用 MCC_SetSysMaxSpeed()設定系統最高的進給速度，進給速度的設定值不能超過此設定，若超過將以 MCC_SetSysMaxSpeed()的設定值為進給速度。使用 MCC_SetSysMaxSpeed() 必須在 MCC_InitSystem()之前呼叫。

➔ *See Also* MCC_GetFeedSpeed()
MCC_GetCurFeedSpeed()
MCC_GetSpeed()

II. 點對點運動的加減速時間及速度比例設定

欲設定點對點運動之加減速時間可使用 MCC_SetPtPAccTime()及 MCC_SetPtPDecTime()，各軸使用各自獨立的加減速時間。點對點運動採用最大安全速度***比例***的方式設定，對應函式為 MCC_SetPtPSpeed()，計算方式為：

各軸點對點運動的進給速度 =
各軸最大安全速度 × (進給速度比例 / 100)

其中

各軸的最大安全速度 = $(wRPM / 60) \times dfPitch / dfGearRatio$

➔ *See Also* MCC_PtP()
MCC_GetPtPSpeed()

III. 微動、吋動、連續 JOG

A. 微動 JOG：MCC_JogPulse()

要求特定軸移動指定的 pulse 量(最大的位移量為 2048 個 pulses)。在使用此函式時，運動狀態應先處於靜止情況(MCC_GetMotionStatus()的函式傳回值應為 GMS_STOP)。下面為使用範例：

```
MCC_JogPulse(    10,                0,                0);
                 //位移量(pulse)      指定軸            Group 編號
```

B. 吋動 JOG：MCC_JogSpace()

要求特定軸依指定的進給速度比例(可參考點對點運動的說明)移動指定的位移量(單位：UU)。在使用此函式時，運動狀態應先處於靜止情況(使用 MCC_GetMotionStatus()的函式傳回值應為 GMS_STOP)。可以使用 MCC_AbortMotionEx()停止此項運動。下面為使用範例：

```
MCC_JogSpace(    1,        20,               0,          0);
                 //位移量    進給速度比例      指定軸      Group 編號
```

C. 連續 JOG 運動：MCC_JogConti()

要求特定軸依指定的進給速度比例(可參考點對點運動的說明)與方向，移動到使用者設定的有效工作區間邊界才停止(有效工作區間定義在機構參數中)；在使用此函式時，運動狀態應先處於靜止情況(使用 MCC_GetMotionStatus()的函式傳回值應為 GMS_STOP)，可以使用 MCC_AbortMotionEx()停止此運動。下面為使用範例：

```
MCC_JogConti(    1,          20      ,     0,          0);
                 //位移方向    進給速度比例   指定軸      Group 編號
                (1：正向，-1：反向)
```

IV. 運動暫停、持續、棄置

可使用 MCC_AbortMotionEx()捨棄目前正在執行中與運動命令佇列的所有運動命令。也可以使用 MCC_HoldMotion()暫停執行中的運動命令(此時將以等減速的方式停止運動)，待使用 MCC_ContiMotion()後，才繼續執行該筆命令尚未完成的部分；但此時也可以使用 MCC_AbortMotionEx()，捨棄尚未完成的部分。

MCC_AbortMotionEx()可使用指定的減速時間來停止運動，若目前的運動狀態已在 GMS_HOLD 狀態則減速時間參數將被忽略。

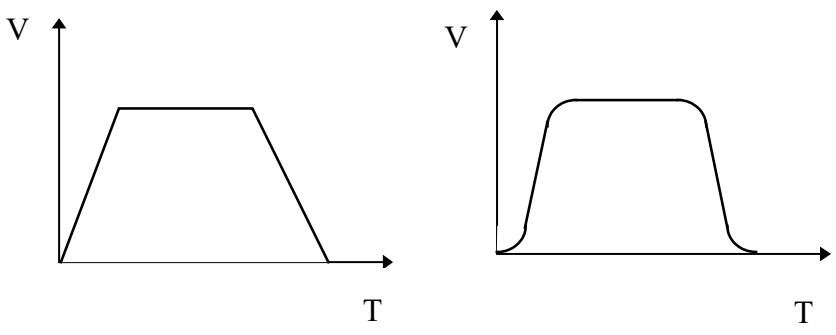
➔ *See Also* MCC_GetMotionStatus()

2.6.3 進階軌跡規劃

為達到更有彈性、更具效率的定位控制，MCCL 提供了幾種進階軌跡規劃功能，例如當不同運動命令間不須精確定位，且須快速到達指定位置時，可使用平滑運動(Motion Blending)功能；在控制系統中常見的追蹤問題，也提供速度強制(Override Speed)功能，允許動態調整進給速度。下面分別說明這些功能：

I. 加減速型式設定

加減速型式可設定為梯型曲線或 S 型曲線(參考 Figure 2.6.1)。



梯型曲線　　S 型曲線

Figure 2.6.1 加減速型式

直線、圓弧、圓、螺線運動之一般運動各軸的加減速型式均以相同加減速型式設定。點對點運動各軸使用各自獨立的加減速型式。

➔ *See Also*

MCC_SetAccType()	MCC_GetAccType()
MCC_SetDecType()	MCC_GetDecType()
MCC_SetPtPAccType()	MCC_GetPtPAccType()
MCC_SetPtPDecType()	MCC_GetPtPDecType()

II. 開啟/關閉平滑運動(Motion Blending)

可以使用 MCC_EnableBlend()開啟平滑運動功能，此項功能可滿足不同運動命令間的等速段達到速度平滑連續的要求(也就是在完成前一段運動命令時速度不須減速到停，可直接加速或減速到下一段運動命令要求的速度)。平滑運動功能包括直線-直線、直線-圓弧、圓弧-圓弧間的平滑運動。

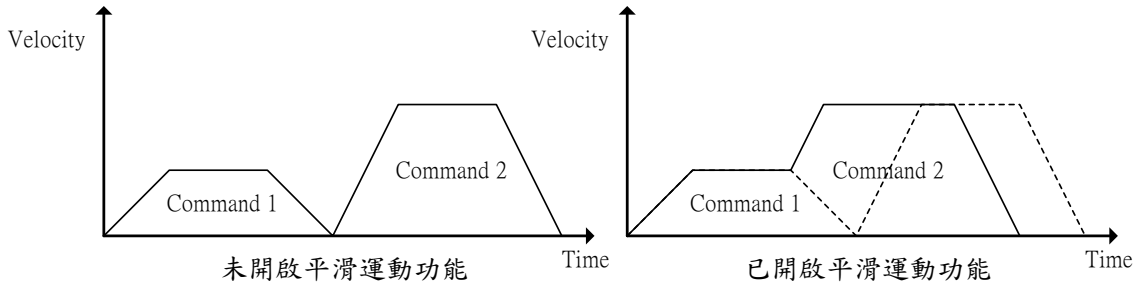
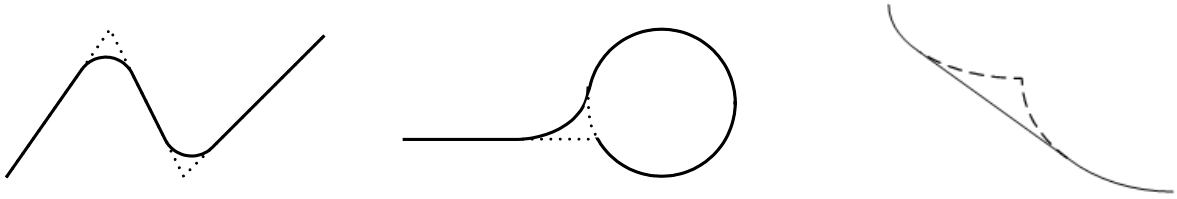


Figure 2.6.2 平滑運動時的速度

由 Figure 2.6.2 可以看出開啟平滑運動功能後的運動情形，第一筆運動命令(Command 1)在達到等速段後不經減速段，而直接加速至第二筆運動命令(Command 2)的等速段(如 Figure 2.6.2 右圖之實線所示)，如此命令的執行時間較快，但各筆命令的連接處會有軌跡失真的狀況存在。Figure 2.6.3 顯示在開啟平滑運動功能後的運動軌跡(虛線代表原先規劃的軌跡曲線)。



直線-直線　　直線-圓弧　　圓弧-圓弧

Figure 2.6.3 直線-直線、直線-圓弧、圓弧-圓弧平滑運動

➔ *See Also*　　MCC_DisableBlend()
MCC_CheckBlend()

III. 速度強制

如須在運動中動態變更進給速度時，可使用速度強制功能，此功能可將執行中運動命令的速度 V_1 加速到要求的速度值 V_2(當 $V_1 < V_2$)，或由目前的速度 V_3 減速到要求的速度值 V_4 (當 $V_3 > V_4$)。

如 Figure 2.6.4，$V_2 = V_1 \times (175 / 100)$【因使用 MCC_OverrideSpeed(175)】；同理，$V_4 = V_3 \times (50 / 100)$【因使用 MCC_OverrideSpeed(50)】。

使用 MCC_OverrideSpeed()指定速度比例，即時強制變更切線速度。速度比例的定義為：

速度比例 = (要求的進給速度 / 原進給速度) × 100

原進給速度是指使用 MCC_SetFeedSpeed()或 MCC_SetPtPSpeed()所設定的速度。

注意：使用 MCC_OverrideSpeed()後，將影響往後全部運動的速度，而不只是影響執行中的運動。

➔ *See Also*　　MCC_GetOverrideRate()

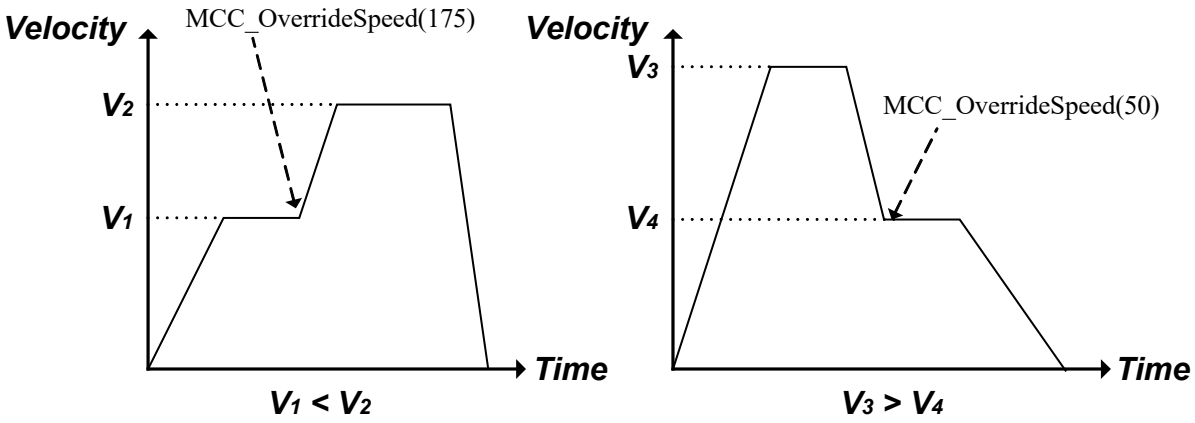


Figure 2.6.4 速度強制

IV. 運動空跑

使用 MCC_EnableDryRun()可開啟運動空跑功能，此時軌跡規劃的結果並不由運動控制平台送出，但使用者仍可利用 MCC_GetCurPos()與 MCC_GetPulsePos()讀取軌跡規劃的內容，除了可事先獲得運動路徑外，使用者並可利用這些資訊在螢幕上模擬運動軌跡。

➔ *See Also* MCC_DisableDryRun()
MCC_CheckDryRun()

V. 運動延遲

可以使用 MCC_DelayMotion()強迫延遲執行下一個運動命令，延遲的時間以毫秒(ms)為單位，下面為使用範例：

```
MCC_Line(10, 10, 10, 0, 0, 0, 0, 0);   -------- 運動命令 A
MCC_DelayMotion(200);
MCC_Line(15, 15, 15, 0, 0, 0, 0, 0);   -------- 運動命令 B
```

則在執行完運動命令 A 後將延遲 200 ms，再繼續執行運動命令 B。

➔ *See Also* MCC_GetMotionStatus()

VI. 錯誤訊息

當發生運動過行程(運動超出軟體邊界)、進給速度超出最大設定值、加/減速度超出最大設定值、圓弧命令參數錯誤、圓弧命令執行錯誤等情況時，可利用 MCC_GetErrorCode()讀取錯誤碼獲知錯誤的內容請參考"***EMP-S 運動控制函式庫參考手冊*** **III. 錯誤訊息代碼**"此章節的列表。

當 Group 發生錯誤時，此 Group 將不再執行運動命令。此時使用者須自行使用 MCC_GetErrorCode()判斷錯誤原因並排除，爾後使用 MCC_ClearError()清除錯誤紀錄，使 Group 恢復至正常狀態。

2.6.4 插值時間與加減速時間

I. 設定插值時間

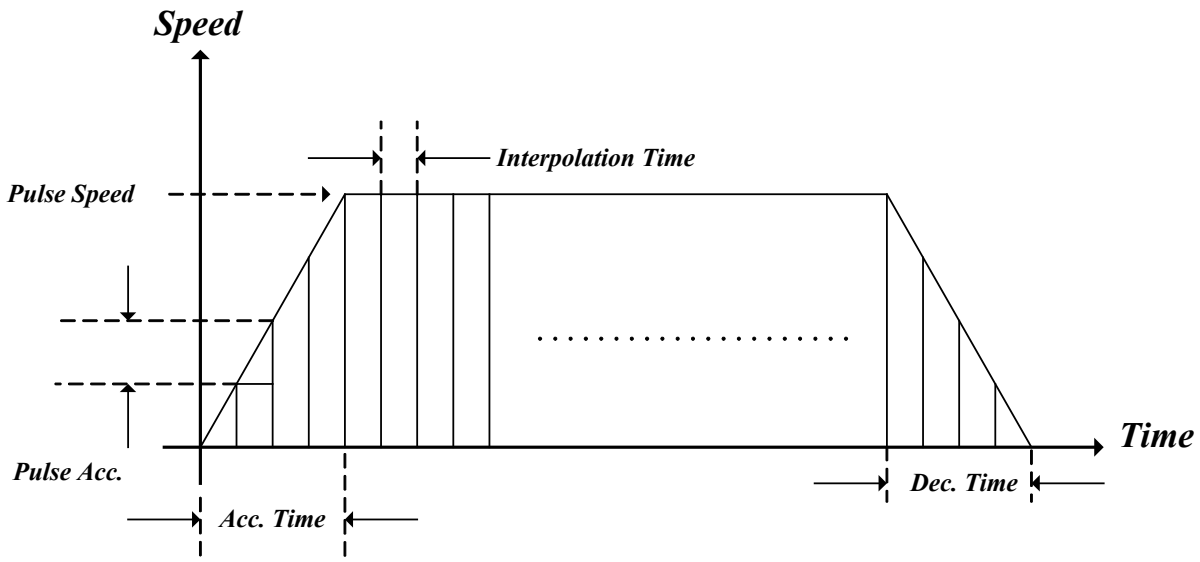


Figure 2.6.5 軌跡規劃設定參數

插值時間(Interpolation Time)是指距離下一個插值點的時間距離

(參考 Figure 2.6.5)，系統設定值為 1 ms。

II. 最大 pulse 速度設定

最大 pulse 速度用來限制各軸在每一個插值時間內所能送出的最大 pulse 量，也就是會限制各軸最大的進給速度。可以使用 MCC_SetMaxPulseSpeed()來設定最大 pulse 速度，可設定的範圍為 1 ~ 1000000，系統預設值為 1000000 pulses。

➔ *See Also* MCC_GetMaxPulseSpeed()

III. 最大 pulse 加、減速度設定

最大 pulse 加、減速度用來限制相鄰插值時間之間所送出 pulse 的最大差量。在運動過程中若加、減速時間不足，有可能加、減速度會超過機構的容許值，如此可能因為運動慣量太大，而造成機構的損傷；此項設定值可用來限制所送出 pulse 的差量在機構的容許範圍內。使用者可以利用 MCC_GetErrorCode()判斷在運動過程中加、減速度是否超出設定範圍。可以使用 MCC_SetMaxPulseAcc()來設定最大 pulse 加、減速度，可設定的範圍為 1 ~ 1000000，預設值為 1000000 pulses。

➔ *See Also* MCC_GetMaxPulseAcc()

IV. 加、減速段所須時間

可以設定一般運動與點對點運動加速到穩定速度所須的時間，也可以設定由穩定速度減速到停止運動所須的時間。使用 MCC_SetAccTime()與 MCC_SetDecTime()設定直線、圓弧、圓、螺線運動所須的加、減速時間；使用 MCC_SetPtPAccTime() 與 MCC_SetPtPDecTime()設定點對點運動所須的加、減速時間。通常在給定較快的進給速度時會要求較長的加速度時間，因此

MCC_SetAccTime() 與 MCC_SetDecTime() 通常會與 MCC_SetFeedSpeed()搭配使用；同樣的，MCC_SetPtPAccTime()與 MCC_SetPtPDecTime()通常會與 MCC_SetPtPSpeed()搭配使用。

下面的範例說明在不同的進給速度時，要求不同的加、減速時間。通常使用者須依照機構特性自訂 SetSpeed()的內容，在要求變更進給速度時須使用 SetSpeed()，而非直接呼叫 MCC_SetFeedSpeed()，尤其是在使用步進馬達時，以免造成失步的情況。

```
void SetSpeed(double dfSpeed)
{
    double dfAcc, dfTime;

    dfAcc = 0.04; //  設定加速度為  0.04 (UU/sec^2)

   if (dfSpeed > 0)
   {
        dfTime = dfSpeed / dfAcc;

        MCC_SetAccTime(dfTime);
        MCC_SetDecTime(dfTime);

        MCC_SetFeedSpeed(dfSpeed);
   }
}
```

2.6.5 系統狀態檢視

MCCL 所提供的函式能檢視目前的實際位置、編碼器的計數值、規劃速度與實際速度、運動狀態、運動命令庫存量、硬體 FIFO 中的細運動命令(Fine Movement Command, FMC)庫存量、執行中運動命令

的內容。

使用 MCC_GetCurPos()可以獲得目前的命令位置，單位為 UU。

MCC_GetPulsePos()可獲得目前已從控制平台輸出的 pulse，MCC_GetCurPos()所讀到的數值為控制平台輸出的 pulse 再經過機構參數轉換的結果。

假使系統有安裝編碼器，則可以使用 MCC_GetENCValue()讀回目前的實際位置(讀取值為編碼器計數值)。

使用 MCC_GetPtPSpeed()可以獲得點對點運動規劃的進給速度比例，而利用 MCC_GetFeedSpeed()可以獲得一般運動規劃的進給速度；而對於一般運動，尚可以使用 MCC_GetCurFeedSpeed()獲得目前實際的切線速度，使用 MCC_GetSpeed()則可以獲得目前各軸實際的進給速度。

利用呼叫 MCC_GetMotionStatus()所獲得的傳回值可以判斷目前的運動狀態；傳回值若為 GMS_RUNNING，表示系統正處於運動狀態；傳回值若為 GMS_STOP，表示系統處於停止狀態，已無任何未執行的庫存命令；傳回值若為 GMS_HOLD，表示系統因使用 MCC_HoldMotion()暫停中；傳回值若為 GMS_DELAYING，表示系統因使用 MCC_DelayMotion()目前正在延遲中。

使用 MCC_GetCurCommand()可以獲得目前正在執行的運動命令相關的資訊，MCC_GetCurCommand()的函式宣告如下：

```
MCC_GetCurCommand( COMMAND_INFO     *pstCurCommand,
                   WORD             wGroupIndex );
```

COMMAND_INFO 儲存目前執行中的運動命令內容，它被定義為：

```
typedef struct _COMMAND_INFO
{
    int             nType;
```

```
    int         nCommandIndex;
    double      dfFeedSpeed;
    double      dfPos[MAX_AXIS_NUM];
} COMMAND_INFO;
```

其中

nType：運動命令類型

0	點對點運動
1	直線運動
2	順時針圓弧、圓運動
3	逆時針圓弧、圓運動
4	順時針螺線運動
5	逆時針螺線運動
6	運動延遲
7	開啟平滑運動
8	關閉平滑運動

nCommandIndex：此運動命令的編碼

dfFeedSpeed：

一般運動	進給速度
點對點運動	進給速度比例
運動延遲	目前剩餘的延遲時間(單位：ms)

dfPos[]：要求的目的點位置

使用 MCC_GetCommandCount()可以獲得目前尚未被執行的運動命令之庫存量，此庫存量不包括正在執行的運動命令。

使用 MCC_GetCurPulseStockCount()可以讀取細運動命令(FMC)

的庫存筆數。在持續運動過程中，FMC 的庫存筆數預設值為 60，使用者可自行依需求設定其 FMC 庫存筆數，確保穩定的運動性能。若存在 FMC 庫存筆數等於 0 的現象，必須延長插值時間(請參考 **2.6.4 插值時間**)。另外，若人機操作畫面的顯示出現遲滯的現象，也可考慮延長插值時間。

2.7 定位控制

2.7.1 齒輪齒隙、背隙補償

機台做定位控制時，可針對因齒輪齒隙或螺桿間隙等因機構設計所產生的位置誤差進行補償(參考 Figure 2.7.1)。

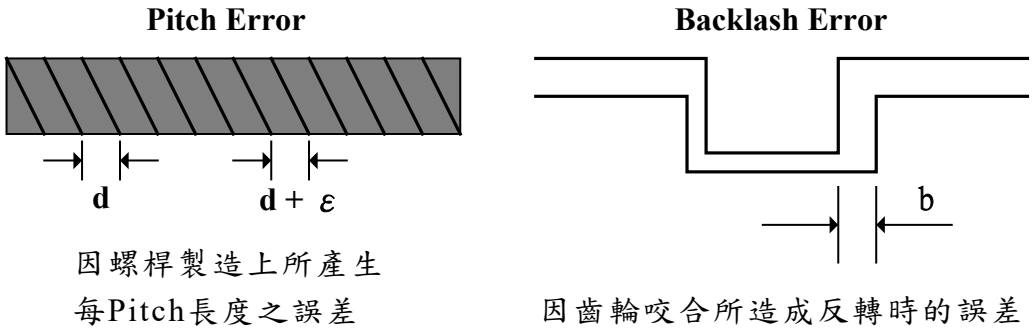


Figure 2.7.1 齒輪齒隙、間隙誤差

使用者可將機台分成多個小區段，使用雷射量測儀，在正負向來回掃瞄一次，將區段點的誤差量記錄下來，並建立正、負向補償表。正、負向補償量表為一個二維陣列，存放各軸所有補償點的補償量，所有補償點皆使用一個量測點為基準，如 Figure 2.7.1 所示為 X 軸進行 6 個補償區段的示意圖。使用者須設定補償參數(*dwInterval*、*wHome_No*、*nForwardTable* 與 *nBackwardTable*)後並呼叫補償設定函式 MCC_SetCompParam()與 MCC_UpdateCompParam()，即可進行補償功能。MCCL 提供各軸 256 個補償點，最多可將機台的各軸分為 255 個補償區段，在各區段間會使用線性補償的方法。

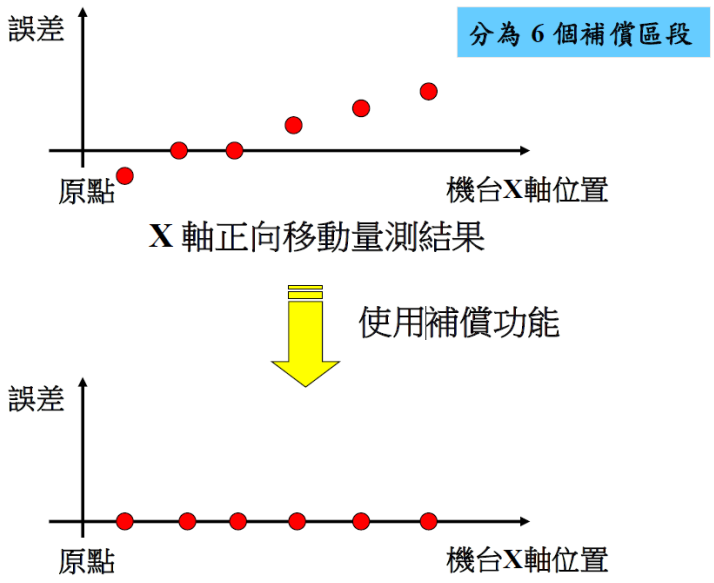


Figure 2.7.2 X 軸進行 6 個補償區段

使用補償功能時，補償參數的內容必須涵蓋機台全部的工作行程，以避免產生不正常的動作，因此應在尚未完成原點復歸動作前啟動補償功能，可以搭配使用 MCC_EcatGetGoHomeStatus()檢查原點復歸動作是否已完成(函式傳回值為 1，表示原點復歸動作已完成)。

在使用補償功能前須先設定補償參數，各軸的補償參數要分開設定，補償參數的定義如下：

```
typedef struct _SYS_COMP_PARAM
{
    DWORD       dwInterval;
    WORD        wHome_No;
    WORD        wPaddle;
    int         nForwardTable[256];
    int         nBackwardTable[256];
} SYS_COMP_PARAM;
```

dwInterval：補償區段的間距，以 pulse 量為單位，此值若小於 0 或

等於 0，表示不進行補償功能

wHome_No：為各軸原點所在位置之補償點編號

wPaddle：保留欄位，使用者須設定為 0

nForwardTable：指向正向補償量表的指標變數

nBackwardTable：指向負向補償量表的指標變數

假設要設定正向補償表，原點所在位置為 0，每段補償區段的間距為 1000 pulses，一共有 5 個補償區段，如 Figure 2.7.3 與 Figure 2.7.4 所示為實際正向與負向的補償量測的果。

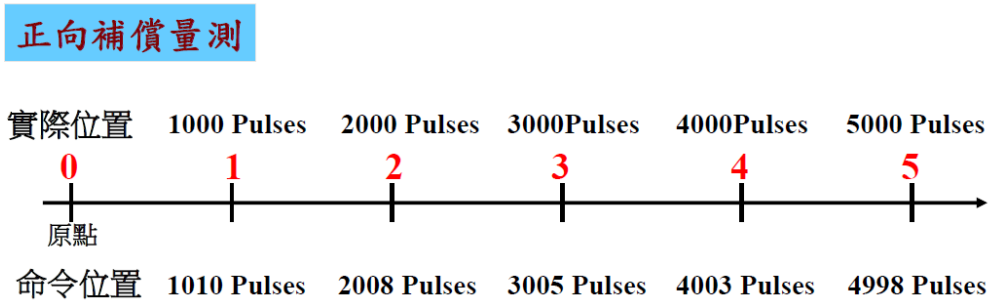


Figure 2.7.3 正向補償量測

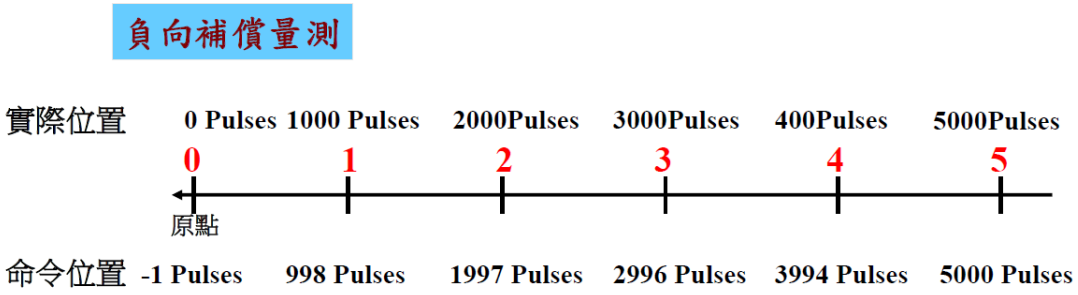


Figure 2.7.4 負向補償量測

下面為設定補償表的程式範例：

```
SYS_COMP_PARAM      stUserCompParam;
stUserCompParam[0].dwInterval = 1000; //補償區段的間距
stUserCompParam[0].wHome_No = 0; //原點所在位置

//5 個正向補償區段；命令位置-實際位置
stUserCompParam[0].nForwardTable[0] = 0;
stUserCompParam[0].nForwardTable[1] = 10;
stUserCompParam[0].nForwardTable[2] = 8;
stUserCompParam[0].nForwardTable[3] = 5;
stUserCompParam[0].nForwardTable[4] = 3;
stUserCompParam[0].nForwardTable[5] = -2;

//5 個負向各補償區段；命令位置-實際位置
stUserCompParam[0].nBackwardTable[0] = -1;
stUserCompParam[0].nBackwardTable[1] = -2;
stUserCompParam[0].nBackwardTable [2] = -3;
stUserCompParam[0].nBackwardTable [3] = -4;
stUserCompParam[0].nBackwardTable[4] = -6;
stUserCompParam[0].nBackwardTable[5] = 0;

//設定與更新齒輪齒隙、間隙補償參數
MCC_SetCompParam(&stUserCompParam, 0);
MCC_UpdateCompParam();
```

停止補償功能的方法是將補償參數的 *dwInterval* 設定為 0，例如停止 Channel 0 的補償功能的程式碼：

```
SYS_COMP_PARAM      stUserCompParam;
```

```
stUserCompParam.dwInterval = 0;

MCC_SetCompParam(&stUserCompParam, 0);
MCC_UpdateCompParam();
```

2.8 EtherCAT原點復歸

使用者須依機構設計與所選用的 EtherCAT 伺服驅動器，進行驅動器上參數的設定，如：原點復歸的模式、原點復歸速度與加減速度以及原點開關(Home Sensor)的配線方式；其中，選擇 EtherCAT 原點復歸的模式須視所使用的 EtherCAT 伺服驅動器、實際機台上檢測 Home Sensor 或極限開關訊號等實際硬體配置而定。

原點開關(Home Sensor)的配線方式，請參考使用的 EtherCAT 伺服驅動器的相關手冊進行硬體配線與驅動器上的參數設定。可使用 MCC_GetHomeSensorStatus()檢查配線方式的設定值是否正確。在未碰觸原點開關時，如果使用 MCC_GetHomeSensorStatus()所獲得的原點開關為 Active 狀態，則表示配線方式設定值錯誤或者配線錯誤。

”啟動 EtherCAT 原點復歸”的步驟如下：

***Step 1*：設定原點復歸的運動軸**

使用 MCC_EcatSetHomeAxis()設定原點復歸的運動軸，函式原型如下：

```
MCC_EcatSetHomeAxis(
    BYTE  byAxisX, BYTE byAxisY, BYTE byAxisZ, BYTE byAxisU,
    BYTE byAxisV, BYTE byAxisW, BYTE  byAxisA, BYTE byAxisB,
    BYTE byAxisX1, BYTE byAxisY1, BYTE byAxisZ1, BYTE byAxisU1,
    BYTE byAxisV1, BYTE byAxisW1, BYTE byAxisA1, BYTE byAxisB1
);
```

參數 *byAxisX* ~ *byAxisB1* 是用來設定對應各運動軸執行原點復歸的功能，設定值為 1 表示該運動軸執行原點復歸動作，設定值為 0 表示不對該運動軸執行原點復歸的動作。

***Step 2*：設定原點復歸模式**

使用 MCC_EcatSetHomeMode()設定原點復歸模式，請依照 Home Sensor、實際配線與選用的 EtherCAT 伺服驅動器而定(請參考 OD 碼 0x6098h)。

***Step 3*：設定原點復歸速度**

使用 MCC_EcatSetHomeZeroSpeed()設定原點復歸尋找零點(Zero)的速度，使用 MCC_EcatSetHomeSwitchSpeed()設定原點復歸時尋找原點開關(Switch)的速度。

***Step 4*：執行原點復歸運動**

使用 MCC_EcatHome()執行原點復歸運動，當完成原點復歸的運動後各軸的直角座標值將被設定為零。

可以使用 MCC_GetHomeSensorStatus()讀取 Home Sensor 的狀態。在原點復歸過程中可以使用 MCC_EcatAbortHome()停止復歸動作；也可以利用 MCC_EcatGetGoHomeStatus()的函式傳回值獲知原點復歸的動作是否已經完成，若傳回值為 1 表示原點復歸動作已經完成，若為 0 表示原點復歸的動作尚在進行中。

2.9 EtherCAT控制功能

以下將說明與 EtherCAT 控制功能的相關函式，包含透過標準的 CoE (CAN application protocol over EtherCAT)通訊指令，對指定的 EtherCAT 從站下達 CANopen 協議的 SDO(Service Data Objects)讀取與傳送指令、取得 EtherCAT 伺服驅動器錯誤碼(Error Code)與重置 EtherCAT 伺服驅動器的錯誤(Fault Reset)功能，相關的函式說明與範例請參考"***EMP-S 運動控制函式庫參考手冊* J. EtherCAT 系統功能**"與"***EMP-S 運動控制函式庫範例手冊***" 。

1、 使用 MCC_EcatCoeSdoUpload()對指定的 EtherCAT 從站下達 SDO 讀取指令，MCC_EcatCoeSdoUpload()的函式原型如下：

```
MCC_EcatCoeSdoUpload( DWORD    dwSlaveId,
                      WORD     wObIndex,
                      BYTE     byObSubIndex,
                      BYTE*    pbyData,
                      DWORD    dwDataLen,
                      DWORD*   pdwOutDataLen );
```

其中

dwSlaveId：EtherCAT 從站編號
wObIndex：物件字典(Object Dictionary, OD)的索引值(Index)
byObSubIndex：物件字典的子索引值(Sub Index)
pbyData：存放欲讀取訊息的資料指標
dwDataLen：欲讀取訊息的資料量大小，單位為 byte
pdwOutDataLen：存放實際讀取訊息的資料長度的指標

如果使用

MCC_EcatCoeSdoUpload(0, 0x607C, 0, (BYTE*)&nData, 4, &len);

表示將對編號為 0 的 EtherCAT 從站下達 SDO 讀取指令，其 OD 碼為 0x607Ch(Home Offset)，參數 *nData* 為欲讀取的資料，欲讀取的資料長度為 4 bytes，參數 *len* 為實際取回的資料長度。

2、 使用 MCC_EcatCoeSdoDownload()對 EtherCAT 從站下達 SDO 傳送指令，MCC_EcatCoeSdoDownload()的函式原型如下：

```
MCC_EcatCoeSdoDownload( DWORD     dwSlaveId,
                        WORD      wObIndex,
                        BYTE      byObSubIndex,
                        BYTE*     pbyData,
                        DWORD     dwDataLen );
```

其中

dwSlaveId：EtherCAT 從站編號
wObIndex：物件字典(Object Dictionary, OD)的索引值(Index)
byObSubIndex：物件字典的子索引值(Sub Index)
pbyData：存放欲傳送訊息的資料指標
dwDataLen：欲傳送訊息的資料量大小，單位為 byte

如果使用

MCC_EcatCoeSdoDownload(0, 0x607C, 0, (BYTE*)&nValue, 4);

表示將對編號為 0 的 EtherCAT 從站下達 SDO 傳送指令，其 OD 碼為 0x607Ch(Home Offset)，參數 *nValue* 為欲傳送的資料，且該傳送的資料長度為 4 bytes。

3、 使用 MCC_EcatGetSlaveErrorCode()取得指定 EtherCAT 伺服驅動器的錯誤碼(Error Code)，MCC_EcatGetSlaveErrorCode()的函式原型如下：

```
MCC_EcatGetSlaveErrorCode( WORD*    pwType,
                           WORD*    pwCode,
                           DWORD    dwSlaveId );
```

其中

pwType：存放錯誤碼層級的資料指標

0 表示無錯誤碼

1 表示為 Fault 層級(OD 碼 0x6041h 的 bit 3)

2 表示為 Warning 層級(OD 碼 0x6041h 的 bit 7)

pwCode：存放錯誤碼的資料指標(OD 碼 0x603Fh)

dwSlaveId：EtherCAT 從站編號

如果使用

```
MCC_EcatGetErrorCode(&pwType, &pwCode, 0);
```

表示欲取得編號為 0 的 EtherCAT 從站的錯誤碼(Error Code)，參數 *pwType* 為錯誤碼的層級，參數 *pwCode* 為錯誤碼，有關錯誤碼的定義與排除方法請參考該從站的相關手冊。

4、 使用 MCC_EcatResetSlaveFault()將重置指定 EtherCAT 從站的錯誤(Fault Reset)，MCC_EcatResetSlaveFault()的函式原型如下：

```
MCC_EcatResetSlaveFault( DWORD  dwSlaveId );
```

如果使用

MCC_EcatResetSlaveFault(0);

表示將重置編號為 0 的 EtherCAT 從站的錯誤(Fault Reset，OD 碼 0x6040h 的 bit 7)。

5、 使用 MCC_EcatGetSlavePresent()取得指定 EtherCAT 從站的連線狀態，MCC_EcatGetSlavePresent()的函式原型如下：

```
MCC_EcatGetSlavePresent( DWORD      dwSlaveId,
                         BOOL*      pbStatus );
```

其中

dwSlaveId：EtherCAT 從站編號

pbStatus：存放 EtherCAT 從站的連線狀態的資料指標；1 表示該從站處於連線狀態，0 則否

2.10網路拓樸ENI檔案

在EMP-S系統必須具備ENI檔案(EtherCAT Network Information File, ENI File)主要用途為描述一個特定的EtherCAT網路；產生ENI檔案的方式有三種，描述如下：

1、 使用Preset Tool應用程式產生ENI檔案(ENIFile.xml)。

2、 全自動模式，必須先將網路硬體裝置設備的描述資訊檔(EtherCAT Slave Information File, ESI File)，放置在指定目錄C:\MccMotion\Esi下，並將該ESI檔案名稱依序更改為ESI1.xml、ESI2.xml、ESI3.xml…依序命名；可以使用MCC_EcatAutoGenerateENI()自動產生ENI檔案，需注意此函式必須在MCC_InitSystem()之前先呼叫。

3、 全手動模式，由使用者自行指定目錄、檔案之操作模式，包含EtherCAT組態設定檔(EC_Config.xml)、ESI與ENI檔案，下面為一個簡單的使用範例：

```
SlaveInfoPre slave;

//載入EC_Config.xml路徑
MCC_EcatLoadCFGFilePath("C:\\MccMotion\\EC_Config.xml");

//載入ESI的存放目錄路徑
MCC_EcatSetESIPath("C:\\MccMotion\\ESI");

//載入ESI的檔案名稱
MCC_EcatLoadESIFileName("Panasonic_MINAS-A6BE_V1_3.xml");
```

```
//載入產生 ENI 檔案的目錄路徑
MCC_EcatSetENIPath("C:\\MccMotion");

//載入產生 ENI 檔案的檔案名稱
MCC_EcatSetENIFileName("ENIFile");

//掃描 EtherCAT 網路拓樸的狀態
MCC_EcatScanBus();

//取得 slave 的資訊
MCC_EcatGetSlaveInfoByID(&slave, 0);
```

2.11 EtherCAT輸入接點與輸出接點控制

ECATS-IO32 為 EtherCAT 從站的專用輸出入控制硬體介面卡。每張 ECATS-IO32 卡可以提供 16 點輸入與 16 點輸出功能。須另外選購、使用者可依實際需求進行串接，有關 ECATS-IO32 控制卡的說明，請參考“***ECATS-IO32 使用手冊***” 。

系統配置的 EtherCAT 從站包含了伺服驅動器與 IO、ADC 模組時，須先串接伺服驅動器後再串接 IO、ADC 模組，如 Figure 2.10.1 為串接 EtherCAT 伺服驅動器、ECATS-IO32 與 ECATS-ADC 的配置示意圖。若僅有串接 IO 模組時則無此限制。

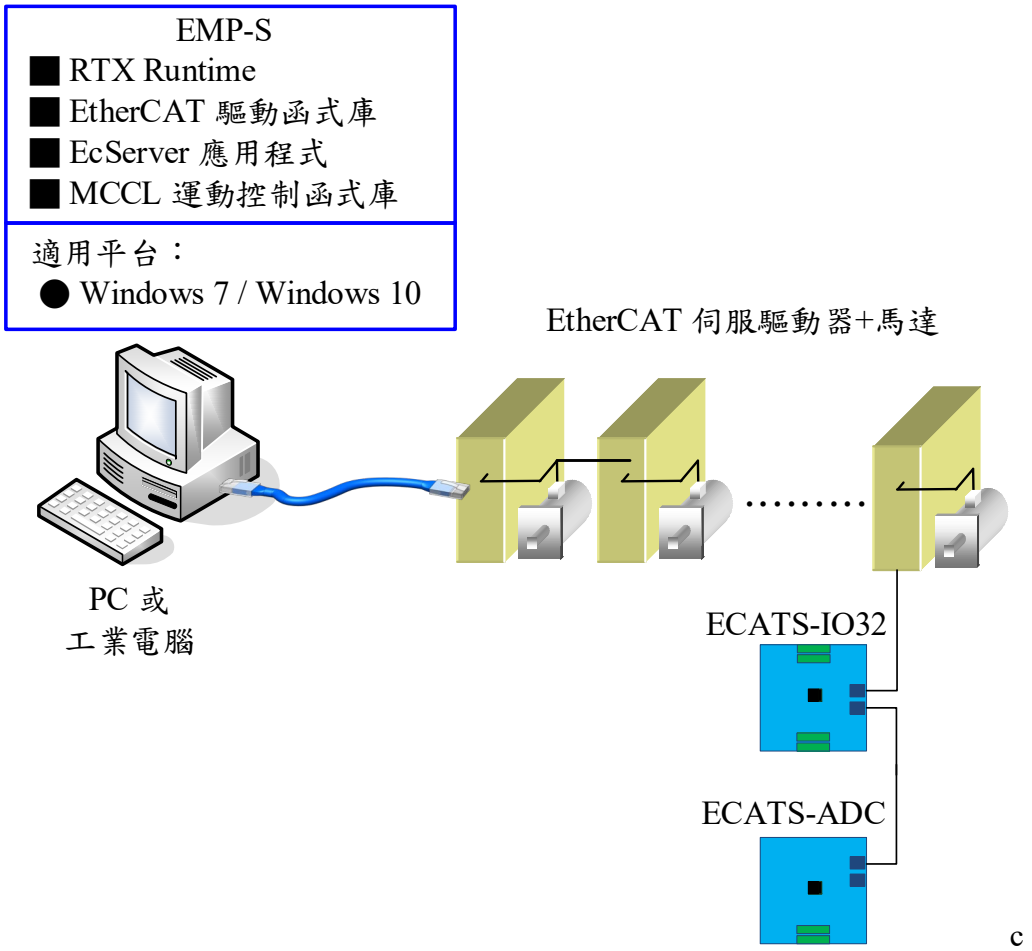


Figure 2.10.1 EtherCAT 從站的配置示意圖

2.10.1 讀取輸入點狀態

可以使用 MCC_EcatGetInput()讀取 ECATS-IO32 的 16 點輸入訊號值，MCC_EcatGetInput()的函式原型如下：

```
MCC_EcatGetInput( DWORD     dwSlaveId,
                  DWORD     *pdwInData );
```

其中

dwSlaveId：EtherCAT 從站編號

pdwInData：存放 16 點的輸入訊號狀態，bit 0 ~ bit 15 分別代表第 0 點到第 15 點輸入點的狀態

下面為一個簡單的使用範例：

```
DWORD nData;
MCC_EcatGetInput(3, &nData);
```

表示指定 EtherCAT 從站編號為 3 的裝置 (ECATS-IO32)，參數 *nData* 表示目前輸入點的狀態。

2.10.2 設定輸出點狀態

可以使用 MCC_EcatSetOutput()設定 ECATS-IO32 的 16 點輸出接點的狀態。MCC_EcatSetOutput()的函式原型如下：

```
MCC_EcatSetOutput( DWORD    dwSlaveId,
                   DWORD    dwOutData );
```

其中

dwSlaveId：EtherCAT 從站編號

pdwOutData：存放 16 點的輸出接點狀態，bit 0 ~ bit 15 分別代表第 0 點到第 15 點輸出點的狀態

下面為一個簡單的使用範例：

```
DWORD nValue;
MCC_EcatSetOutput(3, nValue);
```

表示指定 EtherCAT 從站編號為 3 的裝置 (ECATS-IO32)，參數 *nValue* 表示欲設定的輸出點狀態。

2.10.3 輸出入函式

呼叫 MCCL 中的輸出入控制函式 MCC_EcatSetOutput()後，會***立即執行***輸出入命令，進行解譯及相對應輸出入動作。

呼叫 MCCL 中的輸出入控制函式 MCC_EcatSetOutputEnqueue()後，會將該筆輸出入命令先存放在 Group 各自的運動命令佇列，而***非立即執行***；然後 MCCL 會使用先入先出的方式，從佇列中依序擷取運動命令或輸出入命令，進行解譯及執行相對應動作。

2.12 EtherCAT類比電壓輸入控制

ECATS-ADC 為 EtherCAT 從站的專用類比電壓輸入控制卡。每張 ECATS-ADC 卡可以提供 8 組 A/D Channel 輸入類比電壓控制，輸入源為差動訊號(Differential)，可讀取的電壓範圍為-10V ~ 10V。須另外選購、使用者可依實際需求進行串接。有關 ECATS-ADC 控制卡的說明，請參考“***ECATS-ADC 使用手冊***”。

2.12.1 讀取輸入電壓值

可以使用 MCC_EcatGetADCInput()讀取 ECATS-ADC 的讀取輸入電壓值，MCC_EcatGetADCInput()的函式原型如下：

```
MCC_EcatGetADCInput( DWORD    dwSlaveId,
                     WORD     wChannel,
                     float    *pfInput );
```

其中

dwSlaveId：EtherCAT 從站編號
wChannel：ADC Channel 的編號
pfInput：存放輸入電壓值(-10V ~ 10V)

下面為一個簡單的使用範例：

```
float fInput;
MCC_EcatGetADCInput(4, 0, &fInput);
```

表示指定編號為 4 的 EtherCAT 從站(ECATS-ADC)、第 0 個 ADC Channel，參數 fInput 為目前輸入電壓值。

2.13 軟體Watch Dog控制

使用 MCC_EcatEnableWatchDog()開啟軟體 Watch Dog 功能，必須在 MCC_RTXInit()之後呼叫。

如果使用

MCC_EcatEnableWatchDog(5000);

表示軟體 Watch Dog 的倒數計時時間為 5 秒，設定的計時單位為秒，EcServer 會自動判斷人機介面程式在該時間內無任何回應後，自動終止 EcServer 通訊。

可以使用 MCC_EcatDisableWatchDog()關閉軟體 Watch Dog 功能，必須在 MCC_EcatEnableWatchDog()之後呼叫。

3. 編譯環境

3.1 使用 Visual C++

以下為使用 Visual C++進行開發前須引用的檔案：

- **點擊功能選單：【Project】→【Add Existing Item...】將下列檔案加入至專案**

1. **選取標頭檔：MCCL.h 與 MCCL_Rtx.h 加入至專案**

2. **選取匯入函式庫(import library)：MCCL_Client.lib 加入至專案**

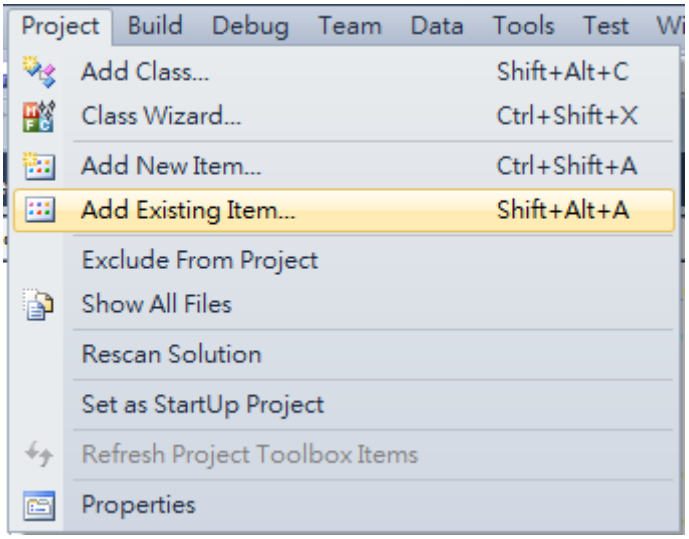


Figure 3.1.1 專案功能選單

3. **複製動態函式庫與 EcServer 應用程式：**
   - **若為 32-bit 系統，將 MCCL_Client.dll、EcServer.rtss 複製至專案執行檔的資料夾。**
   - **若為 64-bit 系統，須再將動態函式庫：EcMaster.rtdll、emllRTL8169.rtdll、emlII8254x.rtdll 一併放入。**

3.2 使用 Visual C# .Net

以下為使用 Visual C# .Net 進行開發前須引用的檔案：

1. 點擊功能選單:【Project】→【Add Existing Item...】將 MCCL.cs 檔案加入至專案

- **選取：MCCL.cs 加入至專案**

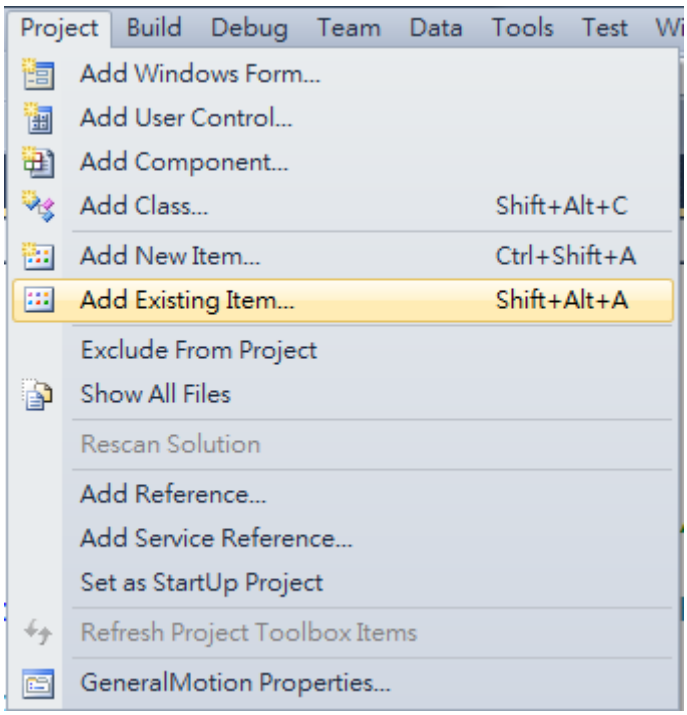


Figure 3.2.1 專案功能選單

2. 複製動態函式庫與 EcServer 應用程式：

- **32-bit 系統，將 MCCL_Client.dll、EcServer.rtss 複製至專案執行檔的資料夾。**
- **64-bit 系統，須再將動態函式庫：EcMaster.rtdll、emllRTL8169.rtdll、emllI8254x.rtdll 一併放入專案執行檔的資料夾。**