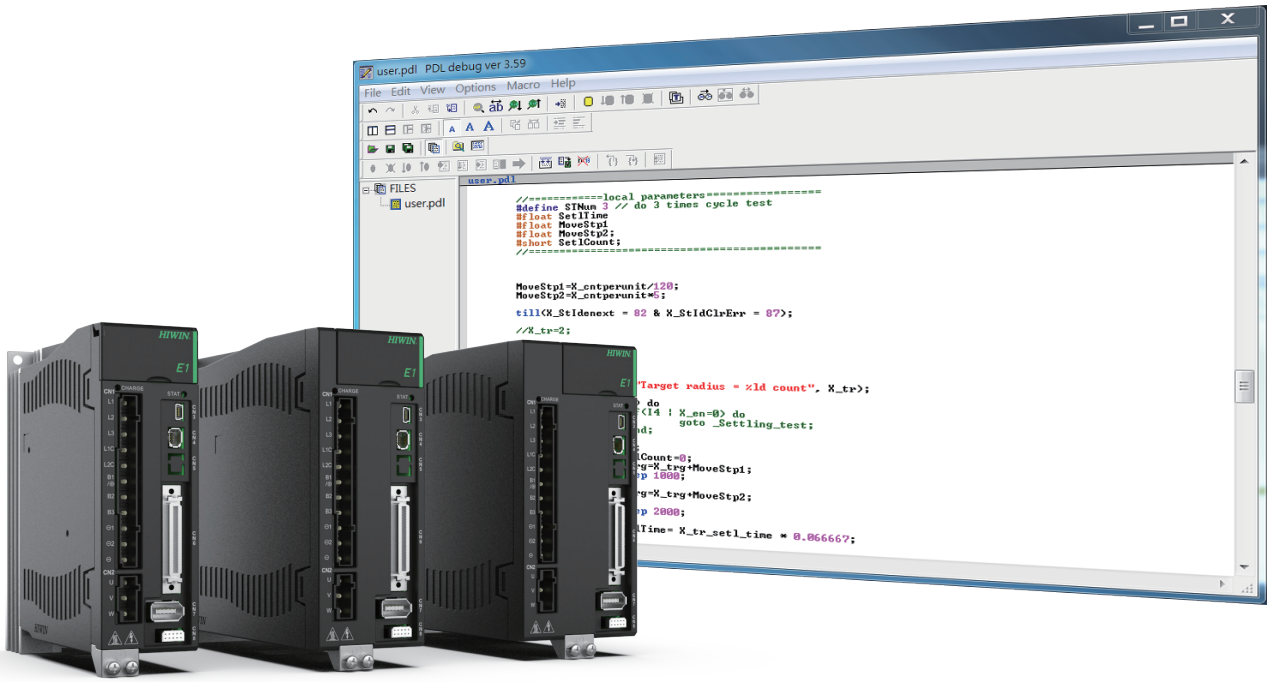
HIWIN®



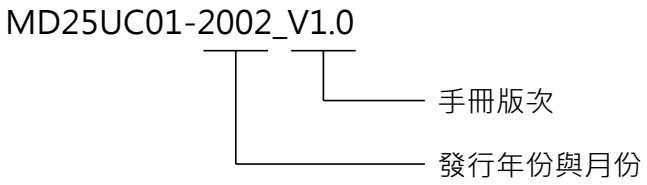
E1系列驅動器

PDL範例程式

www.hiwinmikro.tw
MD25UC01-2002_V1.0

修訂紀錄

手冊版次資訊亦標記於手冊封面右下角。



發行日期	版次	適用產品	更新內容
2020/02/20	1.0	E1 系列驅動器	初版發行。

目錄

目錄

1. 直接搜尋 Index 訊號歸原點

1.1 基本介紹

程式編號	1
檔名	1.pdl
適用驅動器	E1 系列驅動器
適用馬達	AC 伺服馬達、線性馬達、直驅馬達、力矩馬達
驅動器韌體版次需求	Thunder 1.2.14　MDP 2.2.8（含）以上
功能	觸發 I2 數位輸入端，執行驅動器內建歸原點功能。且在歸原點成功後，由 O2 數位輸出端輸出一個脈波給上位控制器。
使用輸入	I2
使用輸出	O2

註：請先至 Thunder 主畫面選擇書籤列中的**工具**、點選 **IO 設定 / 監控**，將 I2 與 O2 設為 **Not configure**，以避免功能相衝突。

1.2 程式內容

```
// ============================ 巨　集 ============================
proc pdl_en() do                                    //  激磁馬達
    X_en = 1;
    till (X_en_fl = 3);
    till (X_dc_bl = 2);
end;
proc pdl_dis() do                                   //  解激磁馬達
    X_en = 0;
    till (X_en_fl = 1);
end;
proc home_para() do                                 //  歸原點參數
    Pt700 = 34;                                     //  歸原點方法
    Pt701 = 30;                                     //  快速歸原點速度
    Pt702 = 10;                                     //  慢速歸原點速度
    Pt703 = 30;                                     //  歸原點逾時 (單位：秒)
end;
proc homing() do
    home_para();
    pdl_en();
```

```
    till (X_servo_ready);
    app.home_cmd = 1;                          //  呼叫驅動器內建歸原點命令
    sleep 100;                                 //  等待 100 毫秒
    till (app.I_flag = 3);                     //  確認歸原點程序 (正在執行中)
    till (app.I_flag = 2 | app.I_flag=-1);     //  確認歸原點程序 (成功或失敗)
    if (app.I_flag<>2) do                      //  確認歸原點狀態是否成功
        printl/101("Homing process failed.");
        pdl_dis();
    end;
    sleep 1000;
end;
// ============================ 主程序 ============================
#task/1;                                       //  建立一編號為 1 的 task
till (~I2);                                    //  等待 I2 關閉
setoff O2;                                     //  關閉 O2
// ============================ 歸原點程序 ============================
go_home:
    till (I2);                                 //  等待 I2 觸發
    homing();                                  //  呼叫歸原點程式
//  歸原點成功後，輸出脈波給上位控制器。
if (app.I_flag=2) do                           //  判斷歸原點狀態是否成功
    seton O2;                                  //  觸發 O2
    sleep 2000;                                //  等待 2 秒
    setoff O2;                                 //  關閉 O2
    sleep 1000;                                //  等待 1 秒
    pdl_dis();
end;
ret;
```

1.3 使用說明

本範例程式藉由觸發 I2 數位輸入端，執行驅動器內建的歸原點功能。且在歸原點程序完成後，由 O2 數位輸出端輸出一個脈波給上位控制器。程式流程圖如圖 1.3.1。

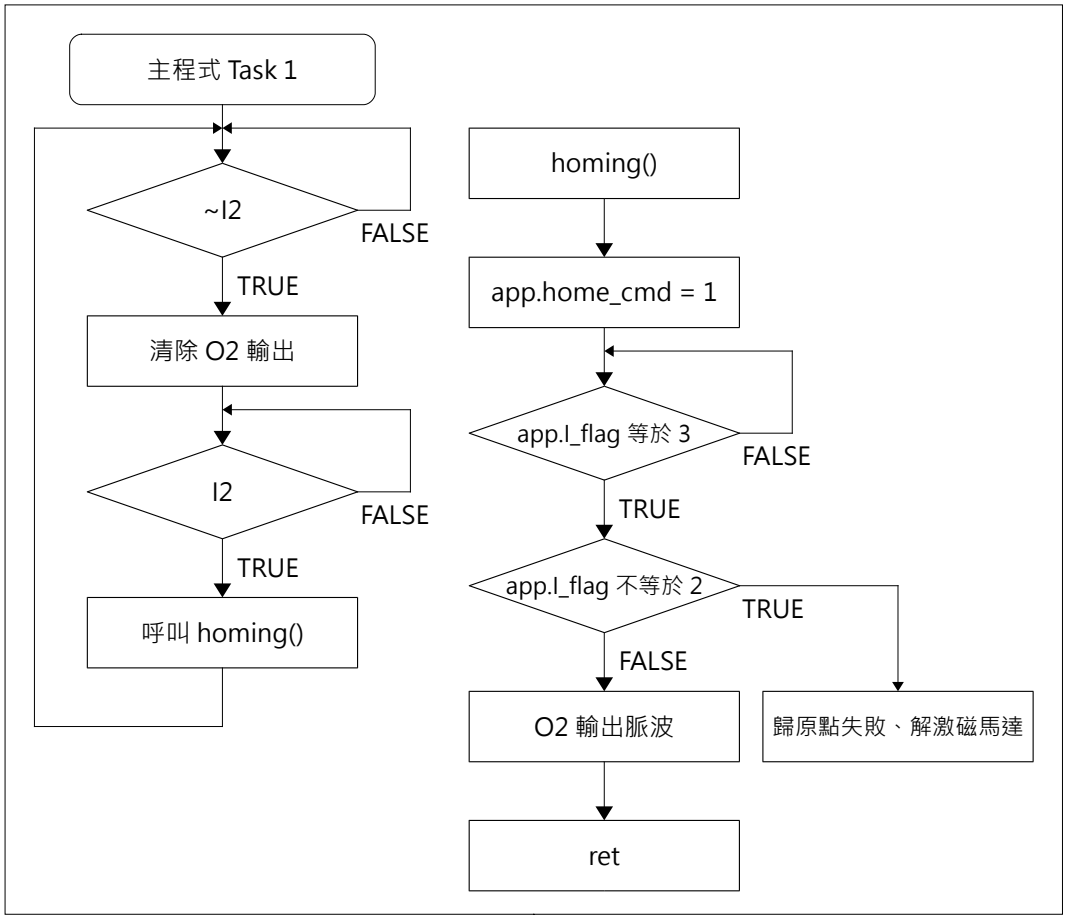


圖 1.3.1 程式流程圖

1.4 應用說明

進入程式主程序前，宣告一 task：#task/1。主要用途為指定使用者所編輯的 PDL 應用程式，在驅動器多工（Multitasking）功能程序中，所佔用的任務編碼。在 E1 系列驅動器中，PDL 最多可同時執行 4 個 task（編碼 0~3)，其中 task 編碼 0 為驅動器作業系統（sys.pdl）所佔用，故使用者可自由編輯其他 task（編碼 1~3）功能。

在驅動器作業系統中，含有一功能命令 app.home_cmd，此命令為驅動器內建歸原點功能。在程式中撰寫 app.home_cmd = 1 時，會開始執行歸原點動作，其歸原點方式為執行方法 34『直接搜尋 Index 訊號歸原點』。相關參數設定位於 Thunder 人機介面的**歸原點**視窗，如圖 1.4.1 所示。

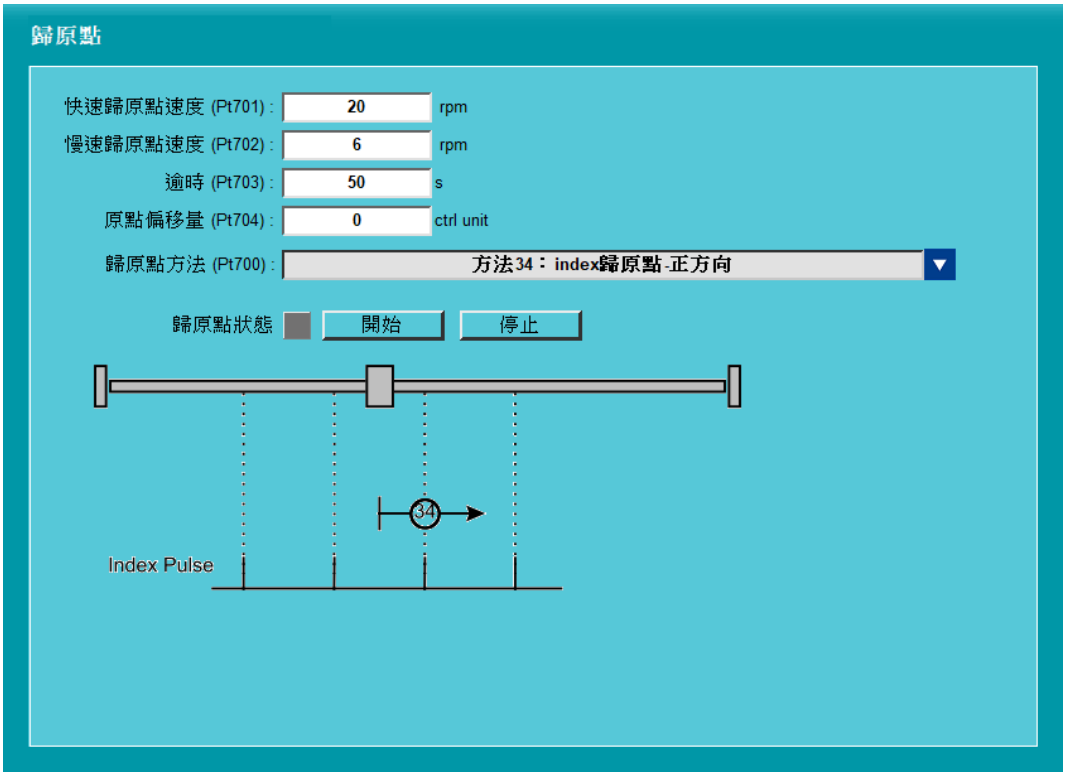


圖 1.4.1 歸原點功能相關參數設定

在本範例程式中，歸原點程序亦使用到一變數 app.I_flag，此變數表示目前歸原點狀態，如表 1.4.1 所示。

表 1.4.1 app.I_flag 變數與歸原點狀態對照表

app.I_flag	歸原點狀態
-1	歸原點失敗
0	尚未執行歸原點
3	正在執行歸原點
2	歸原點成功

(此頁有意留白。)

2. 碰撞 Hard Stop 歸原點

2.1 基本介紹

程式編號	2
檔名	2.pdl
適用驅動器	E1 系列驅動器
適用馬達	AC 伺服馬達（搭配螺桿）、線性馬達
驅動器韌體版次需求	Thunder 1.2.14　MDP 2.2.8（含）以上
功能	在內部位置模式下的歸原點程序中，馬達先往負方向移動，碰撞 Hard Stop 且輸出電流達到設定門檻值時，馬達會開始往正方向搜尋 index，完成歸原點動作。
使用輸入	I2
使用輸出	無

註：請先至 Thunder 主畫面選擇書籤列中的**工具**、點選 **IO 設定 / 監控**，將 I2 設為 **Not configure**，以避免功能相衝突。

2.2 程式內容

```
// ============================ 巨　集 ============================
#define BackDistance   10000                    //  設定遠離 Hard Stop 的距離（單位：ctrl unit)
#define JogVel   Pt383 或 P304                  //  設定搜尋 Hard Stop 的速度
                                                //  線性馬達：用 Pt383；旋轉馬達：用 Pt304。

#float CurrThreshold;
CurrThreshold = X_curr_mot_cont*1.5;            //  設定電流門檻值（單位：A-amp)
proc pdl_en() do                                //  激磁馬達
    X_en = 1;
    till (X_en_fl = 3);
    till (X_dc_bl = 2);
end;
proc pdl_dis() do                               //  解激磁馬達
    X_en = 0;
    till (X_en_fl = 1);
end;
proc home_para() do                             //  歸原點參數
    Pt700 = 34;                                 //  歸原點方法
    Pt701 = 30;                                 //  快速歸原點速度
    Pt702 = 10;                                 //  慢速歸原點速度
```

```
    Pt703 = 30;                                  //  歸原點逾時 (單位：秒)
end;
proc homing() do
    home_para();
    pdl_en();
    till (X_servo_ready);
    app.home_cmd = 1;                            //  呼叫驅動器內建歸原點命令
    sleep 100;                                   //  等待 100 毫秒
    till (app.I_flag = 3);                       //  確認歸原點程序 (正在執行中)
    till (app.I_flag = 2 | app.I_flag=-1);       //  確認歸原點程序 (成功或失敗)
    if (app.I_flag<>2) do                        //  確認歸原點狀態是否成功
        printl/101("Homing process failed.");
        pdl_dis();
    end;
    sleep 1000;
end;
// ============================ 主程序 ============================
#task/1;                                         //  建立一編號為 1 的 task
#long OriginalHomeVel;
#long HomeVel;
Pt305=500;                                       //  軟啟動加速時間
Pt306=500;                                       //  軟啟動減速時間
till (~I2);                                      //  等待 I2 關閉
go_home:
    till (I2);                                   //  等待 I2 觸發
    pdl_en();
    till (X_servo_ready);
    OriginalHomeVel = Pt701;                     //  將原始歸原點速度設為 Pt701 的速度
    printl/101("Start to search negative hard stop.");
    X_jvl = -JogVel;                             //  反轉
    till (X_curr_abs > CurrThreshold);           //  等待碰撞後電流超越預設
    X_stop_m = 1;                                //  運動停止
    if (X_en=0) do
        printl/101("Search negative hard stop failed!");
        pdl_dis();
    end;
```

```
    till (~X_run);
    printl/101("Negative hard stop found, going back.");
    X_trg = X_ref_pos + BackDistance;                    //   離開 Hard Stop
    sleep 100;
    till (~X_run);
    printl/101("Start to search index.");
    homing();
    pdl_dis();
ret;
```

2.3 使用說明

在歸原點程序中，為達到每次搜尋 index 訊號起始基準點與方向的一致性，使用者可選擇以「觸發 Limit Switch」或「碰撞 Hard Stop」的方式定義起始基準點。在沒有 Limit Switch 裝置的情況下，使用者可選擇以「碰撞 Hard Stop」的方式定義起始基準點。

碰撞 Hard Stop 的主要作動方式為碰撞固定方向的 Hard Stop，再往反方向搜尋 index 訊號，以獲得良好的原點重現性。其歸原點程序流程圖如圖 2.3.1。

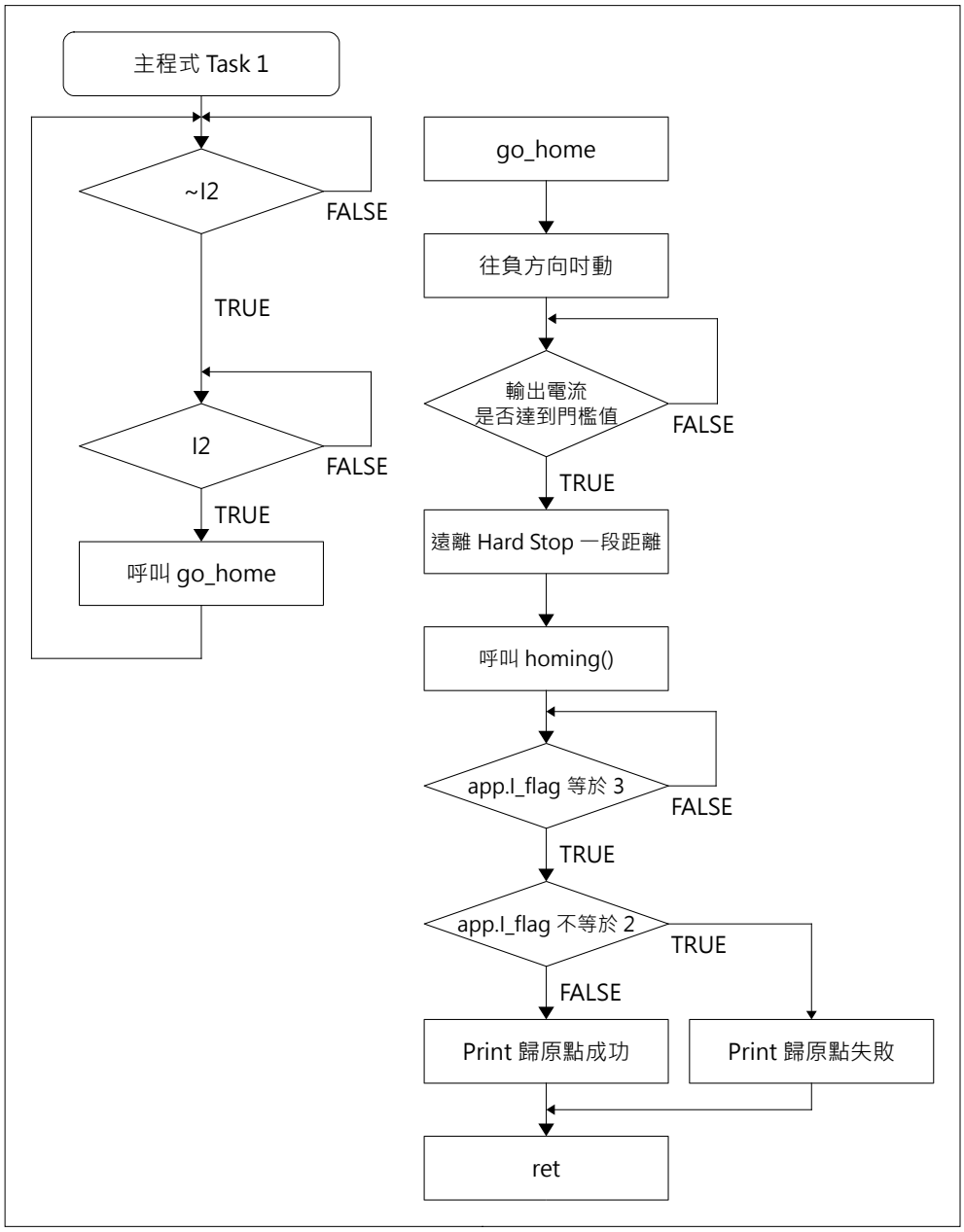


圖 2.3.1 程序流程圖

在程式的最初，使用者須設定電流門檻值 (CurrThreshold)。設定流程如下：

Step 1. 開啟 Thunder 人機介面的**設定精靈**視窗。

Step 2. 於**馬達設定**頁籤查看『連續電流』的值。

Step 3. 設定電流門檻值，稍微大於『連續電流』即可。預設為『連續電流』的 1.5 倍。
註：電流門檻值不可設定太大，否則在碰撞 Hard Stop 時，容易產生『過電流檢出』的警報。

2.4 應用說明

若使用者想使用「觸發 Limit Switch，再執行搜尋 Index」的方式，可直接至 Thunder 人機介面的**歸原點**視窗選擇歸原點方法 1。

3. I/O 觸發馬達移動固定距離

3.1 基本介紹

程式編號	3
檔名	3.pdl
適用驅動器	E1 系列驅動器
適用馬達	AC 伺服馬達、線性馬達、直驅馬達、力矩馬達
驅動器韌體版次需求	Thunder 1.2.14　MDP 2.2.8（含）以上
功能	在內部位置模式下觸發 I2 與 I3 數位輸入端使馬達移動一固定距離；其中 I2 為正轉，I3 為反轉。
使用輸入	I2、I3
使用輸出	無

註：

(1) 若使用線性馬達，須注意行程極限。

(2) 請先至 Thunder 主畫面選擇書籤列中的**工具**、點選 **IO 設定 / 監控**，將 I2 與 I3 設為 **Not configure**，以避免功能相衝突。

3.2 程式內容

```
// ============================ 巨　集 ============================
#define distance   8388608                    //  設定欲移動的距離 (單位：ctrl unit)
proc pdl_en() do                              //  激磁馬達
    X_en = 1;
    till (X_en_fl = 3);
    till (X_dc_bl = 2);
end;
proc pdl_dis() do                             //  解激磁馬達
    X_en = 0;
    till (X_en_fl = 1);
end;
// ============================ 主程序 ============================
#long io_switch, io;
#task/1;                                      //  建立一編號為 1 的 task
io_loop:
if (I2& ~I3) do                               //  判斷 I2 為 ON 且 I3 為 OFF
    io=1;
```

```
    io_switch=1;
    call _trigger_loop;
end;
if (I3 & ~I2) do                                    //  判斷 I2 為 OFF 且 I3 為 ON
    io=2;
    io_switch=1;
    call _trigger_loop;
end;
goto io_loop;

_trigger_loop:
if (io_switch=1) do
    if (io=1) do                                    //  判斷 I2 為 ON 且 I3 為 OFF，正轉
        pdl_en();
        X_trg=X_enc_pos+distance;                   //  正轉並移動設定好的距離
        sleep 50;
        till (~X_run);                              //  等待馬達停止運動
        printl/101("Positive motion ended.");
        sleep 1000;
    end;
    if (io=2) do                                    //  判斷 I2 為 OFF 且 I3 為 ON，反轉
        pdl_en();
        X_trg=X_enc_pos-distance;                   //  反轉並移動設定好的距離
        sleep 50;
        till (~X_run);                              //  等待馬達停止運動
        printl/101("Negative motion ended.");
        sleep 1000;
    end;
end;
io_switch=0;
io=0;
pdl_dis();
goto io_loop;
```

3.3 使用說明

使用者可先設定馬達移動的距離（distance），再觸發 I2 或 I3 數位輸入端，使馬達往正或負方向移動。本範例程式中，重新執行_trigger_loop 前，程式都會先判斷 I2 與 I3 的狀態。其輸入端狀態與移動方向之對照如表 3.3.1 所示。

表 3.3.1 輸入端狀態與移動方向對照表

I2	I3	移動方向
ON	OFF	正
OFF	ON	負

3.4 應用說明

PDL 程式中，X_trg 為執行絕對座標運動的命令。當 X_trg 與當下編碼器的回授值不同時，馬達會移動到 X_trg 的位置。此命令亦可接受運動過程中變更數值，也就是說，最終馬達會移動到最後一次修改 X_trg 數值的位置上。

欲使用 X_trg 實現點對點相對座標運動，須設定一個參考位置，以下舉兩個例子說明。當參考值使用 Feedback position（回授位置），寫法如下：

```
X_trg = X_enc_pos + distance;
```

此做法會造成位置誤差的累積。因時序的緣故，以該值為參考值來下達運動命令，可能會因馬達過衝或尚未完全到位而使用到不恰當的參考值。在多次運動後，會有明顯的位置累積誤差。

因此，使用 PDL 撰寫點對點相對座標運動時，參考位置務必使用 Reference position（位置命令），即可避免上述問題。寫法如下：

```
X_trg = X_ref_pos + distance;
```

另外，可藉 printl/retprintl 將訊息顯示於 Message+command prompt 視窗中。

4. 單位轉換

4.1 基本介紹

程式編號	4
檔名	4.pdl
適用驅動器	E1 系列驅動器
適用馬達	AC 伺服馬達、線性馬達、直驅馬達、力矩馬達
驅動器韌體版次需求	Thunder 1.2.14　MDP 2.2.8（含）以上
功能	將單位由 mm 或 deg 轉換成 ctrl unit，使後接之程式碼，不須由軟體開發工程師自行計算單位轉換。
使用輸入	無
使用輸出	無

4.2 程式內容

```
// ============================ 巨　集 ============================
#define distance   45                                   //  設定移動的距離（mm 或 deg）
// ============================ 主程序 ============================
// =   敘述：使用線性馬達時由 mm 轉換成 ctrl unit
// =           使用 AC 伺服馬達搭配滾珠螺桿時由 mm 轉換成 ctrl unit
// =           使用直驅馬達 / 力矩馬達則由 deg 轉換成 ctrl unit
// =   引數：無
// ================================================================
#task/1;                                                //  建立一編號為 1 的 task
#float LMCntPerMM;
#float disTemp;
#long microm2mm                                         //  Pt20A 進給長度
microm2mm = Pt20A/1000;                                 //  micro meter -> mini meter

if (X_rotaryType=2) do                                  //  確認馬達為 AC 伺服馬達
    disTemp = distance;
    disTemp /= microm2mm;                               //  mm -> rev
    disTemp *= X_cntperunit;                            //  rev -> ctrl unit
end;
```

```
if (X_rotaryType=1) do                    //  確認馬達為直驅馬達 / 力矩馬達
    disTemp = distance;
    disTemp /= 360;                       //  deg -> rev
    disTemp *= X_cntperunit;              //  rev -> ctrl unit
end;

if (X_rotaryType=0) do                    //  確認馬達為線性馬達
    LMCntPerMM = X_cntperunit / 100;      //  count / 100mm -> ctrl unit / mm
    disTemp = distance;
    disTemp *= LMCntPerMM;                //  mm -> ctrl unit
end;

ret;
```

4.3 使用說明

在 PDL 程式中，所有運動參數（如速度 Pt533、加速度 Pt534、減速度 Pt537...等）與位置參數都以 ctrl unit 來計算。本範例程式可解決使用者繁雜的單位換算問題，在執行其他程式前，先行換算須單位轉換的變數。

以下舉例說明。若線性馬達想移動相對於目前位置 45mm 的距離（如本範例中#define distance 45），因程式會將計算結果放入 disTemp，所以可下達下列運動命令來移動 45mm。

```
X_trg = X_ref_pos + disTemp;
```

4.4 應用說明

在本範例程式中，有一驅動器內建變數 Pt20A，此變數表示 AC 伺服馬達所搭配之螺桿導程。另外，X_rotaryType 則表示目前選用的馬達類型，如表 4.4.1 所示。

表 4.4.1 X_rotaryType 變數值與馬達類型對照表

X_rotaryType	馬達類型
0	線性馬達
1	直驅馬達 / 力矩馬達
2	AC 伺服馬達

在本範例程式中，使用了驅動器內建變數 X_cntperunit，該變數為編碼器解析度，此變數之定義會依馬達類型不同而有所差異，如表 4.4.2 所示。

表 4.4.2 馬達類型與 X_cntperunit 定義對照表

馬達類型	X_cntperunit 定義
線性馬達	count / 100mm
直驅馬達 / 力矩馬達	count / rev
AC 伺服馬達	count / rev

5. 教導功能

5.1 基本介紹

程式編號	5
檔名	5.pdl
適用驅動器	E1 系列驅動器
適用馬達	AC 伺服馬達、線性馬達、直驅馬達、力矩馬達
驅動器韌體版次需求	Thunder 1.2.14　MDP 2.2.8（含）以上
功能	在內部位置模式下，使用者以數位輸入觸發教導數個位置後，再藉由數位輸入選擇欲移動的教導位置，以達到可定位至使用者教導的任意點之功能。
使用輸入	I2、I3、I4、I6
使用輸出	無

註：請先至 Thunder 主畫面選擇書籤列中的**工具**、點選 **IO 設定 / 監控**，將 I2、I3、I4 與 I6 設為 **Not configure**，以避免功能相衝突。

5.2 程式內容

```
// ============================ 巨　集 ============================
#define BufferSize   4                          //  定義教導點位置暫存器大小
proc pdl_en() do                                //  激磁馬達
    X_en = 1;
    till (X_en_fl = 3);
    till (X_dc_bl = 2);
end;
proc pdl_dis() do                               //  解激磁馬達
    X_en = 0;
    till (X_en_fl = 1);
end;
// ========================== 全域變數 ==========================
#short BufferNum;                               //  位置暫存器索引值
#long PosBuff[BufferSize];                      //  位置暫存器，用來儲存各個教導點的位置。
// ============================ 主程序 ============================
#task/1;                                        //  建立一編號為 1 的 task
BufferNum=0;                                    //  位置暫存器索引值初始

_trigger_loop:
```

```
if (I2) do                                          //  若 I2 為 ON 狀態，則呼叫教導位置程序。
    call _PosMem;                                   //  呼叫教導位置程序
    till (~I2);
end;
if (I6) do                                          //  若 I6 為 ON 狀態，則呼叫移動到教導位置程序。
    pdl_en();
    till (X_servo_ready);
    if (X_en=0) do                                  //  若無激磁
        printl/101("Motor is not enabled!");        //  列印無激磁警告訊息
        till (~I6);
        goto _trigger_loop;                         //  等待 I6=0 時再回到迴圈開頭
    end;
    call _Move2Tar;                                 //  呼叫移動到教導位置程序
    till (~I6);
end;
goto _trigger_loop;
ret;

// ====================  教導位置程序 PosMem ====================
// =   敘述：依位置暫存器編號 BufferNum 將目前位置儲存到位置暫存器 PosBuff
// =   引數：無
// ============================================================
_PosMem:
#long posTemp;
posTemp=X_enc_pos;
PosBuff[BufferNum]= posTemp;                        //  回授位置儲存到位置暫存器
printl/103("Teaching point %ld, Position=%ld", BufferNum, posTemp);
//  列印儲存的教導點編號與位置
BufferNum+=1;                                       //  位置暫存器移到下一個
    if (BufferNum=BufferSize) do                    //  索引值超出位置暫存器容量
        BufferNum=0;                                //  回到第一個暫存器
    end;
ret;
// ====================  移動到教導位置 Move2Tar ====================
// =   敘述：依一般輸入的 I4、I3 來選擇移動目標
// =   引數：無
```

```
// ============================================================
_Move2Tar:
#short index;
if (~I4 & ~I3) do                          //  (I4, I3) = (0, 0)
    index =0;
end;
if (~I4 & I3) do                           //  (I4, I3) = (0, 1)
    index=1;
end;
if (I4 & ~I3) do                           //  (I4, I3) = (1, 0)
    index=2;
end;
if (I4 & I3) do                            //  (I4, I3) = (1, 1)
    index=3;
end;
X_trg=PosBuff[index];                      //  移動到目標位置
till (~X_run);                             //  等待馬達停止運動
printl/103("Move to point %ld ended.", index);   //  列印到位訊息
printl/103("Position=%ld", PosBuff[index]);      //  列印座標
sleep 100;
ret;
```

5.3 使用說明

本程式利用一般數位輸入執行教導的功能，數位輸入與功能之對照如表 5.3.1。

表 5.3.1 數位輸入與功能對照表

數位輸入	功能
I2	教導位置，記錄目前座標到教導點。
I3	選擇要移動到的教導點。
I4	
I6	正緣觸發移動到選擇的教導點。

I2 正緣觸發一次即教導一次，I2 觸發次數與教導位置關係如表 5.3.2。此程式可教導四個位置，若 I2 觸發超過 4 次，將從第 1 點依序覆蓋先前的教導點不斷循環。教導位置時若想重新開始，可直接重置驅動器或重開 24V 電源。然而，重置後會清除先前記錄的教導點，教導點位置初始皆為零。

表 5.3.2 觸發次數與教導位置對照表

索引值	教導位置
0	把所在位置座標記錄到教導點 0。
1	把所在位置座標記錄到教導點 1。
2	把所在位置座標記錄到教導點 2。
3	把所在位置座標記錄到教導點 3。

若要執行移動到教導點，請先到**參數設定**設定好速度、加速度等運動參數，再利用 I4、I3 選擇要移動到的教導點，利用 I6 正緣觸發開始運動。I4、I3 輸入狀態與相對應程序如表 5.3.3。

表 5.3.3 I3、I4 輸入狀態與對應程序對照表

輸入狀態			動作
I4	I3	I6	
OFF	OFF	正緣觸發	移動到教導點 0
OFF	ON		移動到教導點 1
ON	OFF		移動到教導點 2
ON	ON		移動到教導點 3

(此頁有意留白。)

6. 重新激磁延續未完成的絕對座標運動

6.1 基本介紹

程式編號	6
檔名	6.pdl
適用驅動器	E1 系列驅動器
適用馬達	AC 伺服馬達、線性馬達、直驅馬達、力矩馬達
驅動器韌體版次需求	Thunder 1.2.14　MDP 2.2.8（含）以上
功能	馬達在內部位置模式下執行絕對座標運動時，中途若產生錯誤導致馬達解激磁，排除錯誤後再重新激磁時，馬達會移動到解激磁前的目標位置。
使用輸入	I2、I3
使用輸出	無

註：請先至 Thunder 主畫面選擇書籤列中的**工具**、點選 **IO 設定 / 監控**，將 I2 與 I3 設為 **Not configure**，以避免功能相衝突。

6.2 程式內容

```
// ============================ 巨　集 ============================
#define TargetPos1   Pt531                    //  目標位置 1 (單位：ctrl unit)
#define TargetPos2   Pt532                    //  目標位置 2 (單位：ctrl unit)
proc pdl_en() do                              //  激磁馬達
    X_en = 1;
    till (X_en_fl = 3);
    till (X_dc_bl = 2);
end;
proc pdl_dis() do                             //  解激磁馬達
    X_en = 0;
    till (X_en_fl = 1);
end;
// ============================ 主程序 ============================
#task/1;                                      //  建立一編號為 1 的 task
#long targetpos;
#long MotionCompleteFg;                       //  辨識馬達是否確實到達目標位置的旗標
MotionCompleteFg = 1;                         //  MotionCompleteFlag = 1 表示已到位
_TargetMove:
till (~I2 & ~I3);
```

```
till (I2 | I3);
if (I2 & ~I3) do
    targetpos = TargetPos1;
    MotionCompleteFg = 0;                    //  MotionCompleteFlag = 0 未到位
end;
if (~I2 & I3) do
    targetpos = TargetPos2;
    MotionCompleteFg = 0;
end;
while (MotionCompleteFg = 0) do
    pdl_en();
    till (X_servo_ready);                    //  若因錯誤而跳停，在此等候重新激磁。
    X_trg = targetpos;                       //  開始移動到目標位置
    sleep 100;
    till (~X_run);                           //  等待運動正常結束或因錯誤而跳停
    if (X_servo_ready) do
        MotionCompleteFg = 1;                //  若正常完成運動，則準備離開 while 迴圈。
    end;
end;
goto _TargetMove;
ret;
```

6.3 使用說明

使用者設定絕對座標運動的目標位置(TargetPos1, TargetPos2)後，利用觸發 I2 與 I3 來選取目標位置（請參閱表 6.3.1），馬達將朝目標位置移動。若中途馬達因產生錯誤而停止運動與解激磁，排除錯誤後重新激磁，程式會判斷馬達是否確實到達目標位置。若未到達目標位置，程式會重複執行 X_trg 命令使馬達往目標位置移動，直到到達目標位置後才停止運動。

表 6.3.1 輸入腳位狀態與目標位置對照表

I2	I3	變數名	目標位置
ON	OFF	TargetPos1	Pt531
OFF	ON	TargetPos2	Pt532

6.4 應用說明

以先前範例與本範例程式所使用的 X_trg 命令執行絕對座標移動，作動方式為當 X_trg 座標值不等於 X_enc_pos 值時，馬達便會往 X_trg 座標值移動，直到 X_enc_pos 等於 X_trg，馬達才結束運動。

若因馬達解激磁而結束運動，X_trg 會自動設定為當下 X_ref_pos 值。此時，馬達任意移動只會改變 X_enc_pos 回傳值，X_ref_pos 與 X_trg 值並不會隨著改變。直到接受到 Enable 命令，X_ref_pos 與 X_trg 才會設定為當下 X_enc_pos 回傳值。因此，正常情況下，馬達在運動過程中發生錯誤解激磁，再重新激磁時，馬達不會繼續移動。

因此，本範例程式設計一 while loop，宣告變數 MotionCompleteFg 判斷馬達是否確實到達目標位置。當馬達結束運動後，再判斷馬達是否為激磁狀態。若為激磁狀態，馬達確實到達目標位置；若非激磁狀態，則表示馬達未到達目標位置。

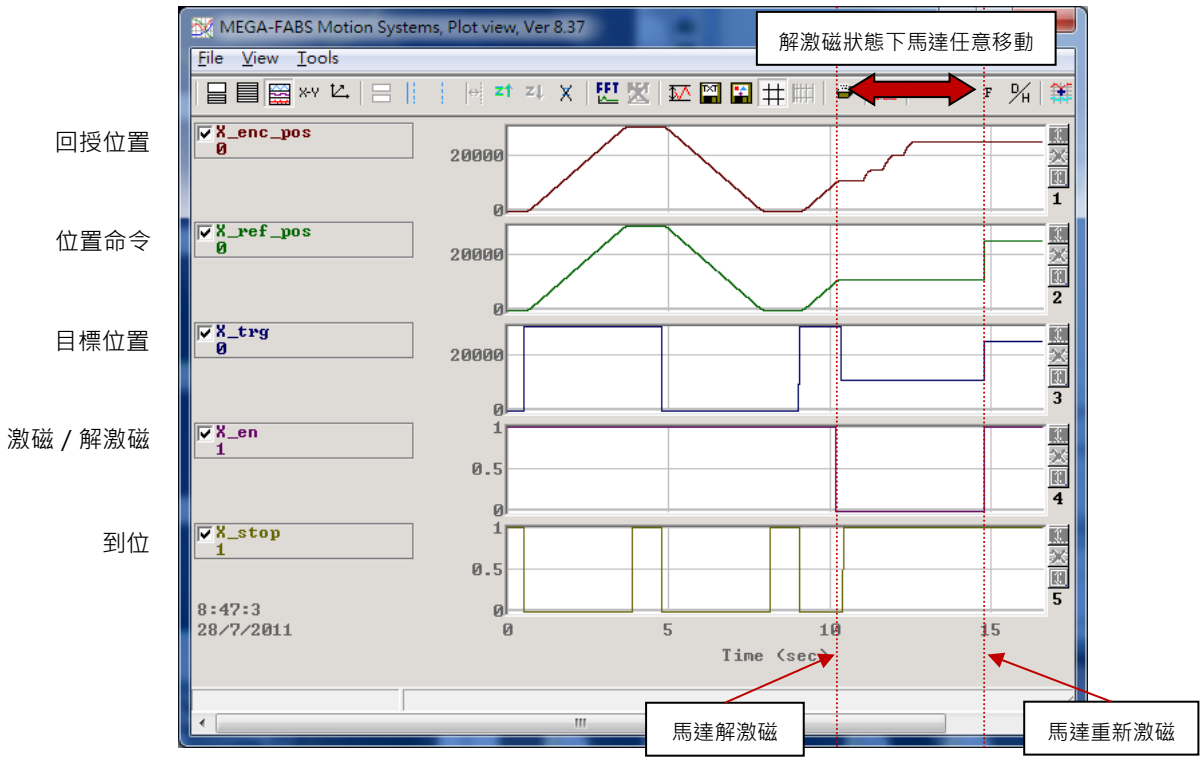


圖 6.4.1 運動過程與激磁 / 解激磁狀態目標位置之變化

7. 吋動

7.1 基本介紹

程式編號	7
檔名	7.pdl
適用驅動器	E1 系列驅動器
適用馬達	AC 伺服馬達、線性馬達、直驅馬達、力矩馬達
驅動器韌體版次需求	Thunder 1.2.14　MDP 2.2.8（含）以上
功能	在位置模式下透過驅動器的數位輸入觸發吋動功能。
使用輸入	I2、I3
使用輸出	無

註：請先至 Thunder 主畫面選擇書籤列中的**工具**、點選 **IO 設定 / 監控**，將 I2 與 I3 設為 **Not configure**，以避免功能相衝突。

7.2 程式內容

```
// ============================ 巨　集 ============================
#define CW_JOG   I2                          //  定義 I2 為正轉吋動按鈕
#define CCW_JOG   I3                         //  定義 I3 為反轉吋動按鈕
proc pdl_en() do                             //  激磁馬達
    X_en = 1;
    till (X_en_fl = 3);
    till (X_dc_bl = 2);
end;
proc pdl_dis() do                            //  解激磁馬達
    X_en = 0;
    till (X_en_fl = 1);
end;
// ============================ 主程序 ============================
#task/1;                                     //  建立一編號為 1 的 task
#long jog_vel;
jog_vel=Pt533 或 Pt585;                      //  設定程式 JOG 移動速度
                                             //  旋轉馬達：用 Pt533；線性馬達：用 Pt585。
_loop:
    pdl_en();
    if (X_en=1) do                           //  若激磁時
```

```
        if (CW_JOG) do                          //  按正轉吋動按鈕時
            X_jvl=X_jog_vel;                    //  以 Pt533 或 Pt585 的速度值執行正轉
            till (~CW_JOG);                     //  直到釋放正轉吋動按鈕
            X_stop_m=1;                         //  運動停止
            till (X_stop_m=0);
        end;
        if (CCW_JOG) do                         //  反轉吋動
            X_jvl=-X_jog_vel;                   //  以 Pt533 或 Pt585 的速度值執行反轉
            till (~CCW_JOG);                    //  直到釋放反轉吋動按鈕
            X_stop_m=1;
            till (X_stop_m=0);
        end;
    else do
        printl/101("Motor is not enabled!");        //  未激磁警告訊息
        printl/103("Please release jog button before enabling motor.");     //  告知釋放吋動按鈕
        till (X_en=1 & ~CW_JOG & ~CCW_JOG); //  直到馬達激磁且已釋放吋動按鈕
    end;
goto _loop;
ret;
```

7.3 使用說明

本程式利用一般數位輸入執行吋動功能，數位輸入與功能對照如表 7.3.1。

表 7.3.1 數位輸入與功能對照表

數位輸入	功能
I2	正轉吋動
I3	反轉吋動

此程式的運動參數是以 Thunder 人機上**試運轉**視窗中的參數來實施的，如表 7.3.2。

表 7.3.2 運動參數與**試運轉**視窗參數對照表

運動參數	對應的**試運轉**視窗參數
速度	Speed
加速度	Acc.
減速度	Dec. Kill

註：欲執行吋動程式，須先設定好參數。

7.4 應用說明

本範例程式開頭使用到#define CW_JOG I2 這個巨集，使程式中的 CW_JOG 直接對應 I2。此為較具彈性的寫法，以本範例正轉吋動(CW_JOG)為例，若使用者想改變輸入為 I12，只要將其修改為#define CW_JOG I12 即可。

本範例程式使用到兩個驅動器內建變數，如表 7.4.1 所示。

表 7.4.1 驅動器內建變數說明

驅動器內建變數	說明
X_jvl	用來設定馬達吋動時的速度 (單位：count/s)
X_stop_m	設為 1 時停止馬達運動

第 3.4 節中提到的 X_trg，在下列兩種狀況中，不可與本範例中的 X_jvl 命令同時下達。

(1) 當 jvl 設定為非零值讓馬達移動，trg 可讓使用者改變，但驅動器不會理會此命令，馬達會持續以 jvl 的速度進行吋動。若要使馬達停止運動，則須將 stop_m 設定為 1。jvl 會自動被歸零，馬達立刻停止，並將 trg 的值設定為停止的座標 (X_enc_pos)。

(2) 當 trg 設定一個值讓馬達移動，馬達移動中嘗試將 jvl 設定為非零值，jvl 無法讓使用者修改，值會永遠維持 0。

8. 類比輸入控制时動

8.1 基本介紹

程式編號	8
檔名	8.pdl
適用驅動器	E1 系列驅動器
適用馬達	AC 伺服馬達、線性馬達、直驅馬達、力矩馬達
驅動器韌體版次需求	Thunder 1.2.14　MDP 2.2.8（含）以上
功能	在位置模式下以類比電壓輸入（0V~+10V）控制馬達速度，並利用數位輸入控制正反轉。
使用輸入	I2、I3、Ref+、Ref-
使用輸出	無

註：請先至 Thunder 主畫面選擇書籤列中的**工具**、點選 **IO 設定 / 監控**，將 I2 與 I3 設為 **Not configure**，以避免功能相衝突。

8.2 程式內容

```
// ============================ 巨　集 ============================
#define JOG_VEL_SCALE    Pt300              //  Pt300 預設為 6V/額定速度。
                                            //  類比電壓輸入為 1V 時，馬達速度為額定速度的 1/6。
proc pdl_en() do                            //  激磁馬達
    X_en = 1;
    till (X_en_fl = 3);
    till (X_dc_bl = 2);
end;
proc pdl_dis() do                           //  解激磁馬達
    X_en = 0;
    till (X_en_fl = 1);
end;
// ========================= 單位轉換 proc =========================
proc Unit_Transform_Vel(float *velTemp) do
    *velTemp /= 60;
    *velTemp *= X_cntperunit;
end;
// ============================ 主程序 ============================
#task/1;
```

```
#float JOG_VEL_VOL, JogVel_Scale;

X_vgfl_all = 1;                          //  修正類比偏壓
till (X_vgfl_all = 0);
JogVel_Scale = JOG_VEL_SCALE;
Unit_Transform_Vel(&JogVel_Scale);       //  執行單位換算

_loop:
pdl_en();
till (X_servo_ready);                    //  等待伺服就緒
JOG_VEL_VOL = Vcmd*JogVel_Scale;         //  由 Ref+/-讀入電壓，並換算為命令速度。
if ( I2 ) do
    if ( I3 ) do                         //  (I2, I3) = (1, 1)
        X_jvl = 0;                       //  停止運動
    else do                              //  (I2, I3) = (1, 0)
        X_jvl = JOG_VEL_VOL;             //  執行正轉吋動
    end;
else do
    if ( I3 ) do                         //  (I2, I3) = (0, 1)
        X_jvl = -JOG_VEL_VOL;            //  執行反轉吋動
    else do                              //  (I2, I3) = (0, 0)
        X_jvl = 0;                       //  停止運動
    end;
end;
goto _loop;
_task_end:
ret;
```

8.3 使用說明

本程式利用類比電壓輸入的大小來控制運動速度，以數位輸入（I2、I3）控制正轉、反轉或停止運動。數位輸入與功能對照如表 8.3.1。

表 8.3.1 數位輸入與功能對照表

I2	I3	動作
0	0	停止運動
0	1	反轉
1	0	正轉
1	1	停止運動

程式內預設每 6V 對應額定速度。欲修改該比例，可在 Thunder 人機介面中修改 Pt300 的值。

註：至 Thunder 人機介面的**設定精靈**視窗，點選**馬達設定**頁籤，即可查看『額定速度』的值。

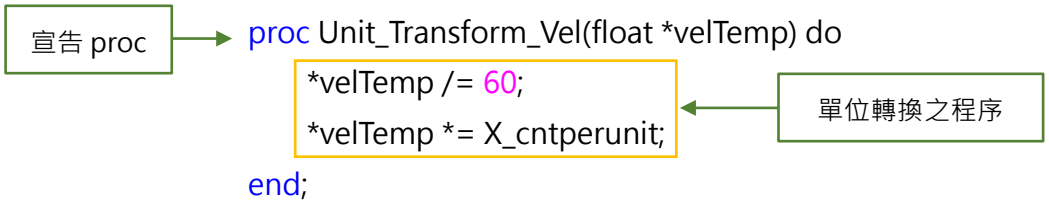
8.4 應用說明

本範例程式利用驅動器內建變數 Vcmd，讀取由驅動器外部（Ref+、Ref-）所輸入之類比電壓，再乘上比例，以得到欲輸出之速度。因線性馬達預設的額定速度為 1500mm/s，以本範例程式的比例 6V=1500mm/s 為例，若使用者輸入類比電壓為 0~10V，則可輸出 0~2500mm/s 的速度。

本範例程式因欲輸入之類比電壓僅能為 0V~+10V，故使用數位輸入（I2、I3）來控制正反轉。欲輸入之類比電壓為-10V~+10V 時，請使用驅動器的速度模式。

本範例使用了 PDL 的 proc 機制來執行單位轉換（請參閱第 4 章）。在執行主程序前先宣告一個 proc，並在 proc 內設定好執行單位轉換的程序，使用者即可在執行主程序時，將 proc 呼叫出來執行單位轉換。可重覆呼叫執行單位轉換，以減少主程序的複雜度。

Step1. 以本範例為例，在主程序之前先宣告 proc，並將單位轉換的程序建立在 proc 之內。



Step2. 在主程序內若須使用此單位轉換之程序，呼叫此 proc 即可。