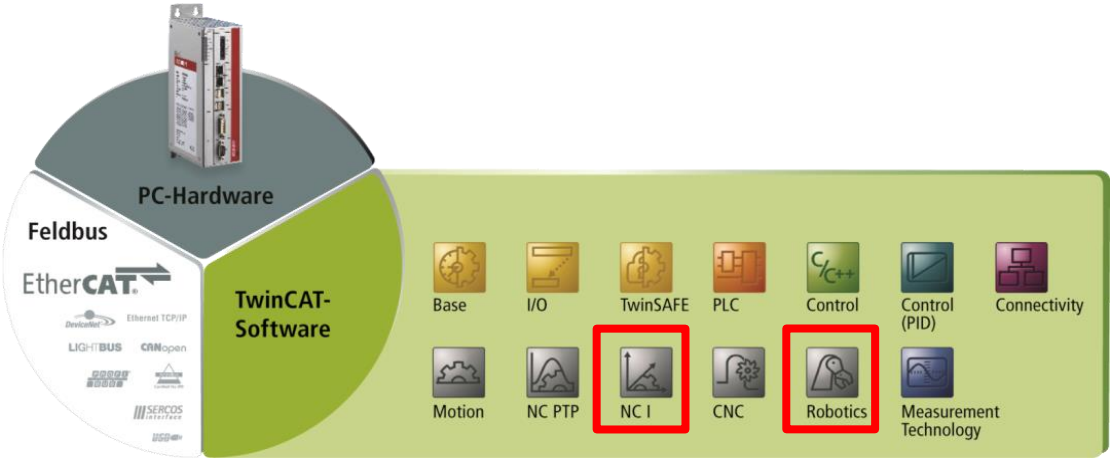


TwinCAT NCI 插补和机器人技术

V1.1

王进 张寅 著

2021.2.4

本书内容纯属个人技术总结，如有纰漏还请谅解和指正，如要深入应用请参考官网帮助文档。

序言

NCI 是倍福自主研发的 CNC 系统，具有对控制器性能要求低、廉价、与 NCPTP 轴无缝融合、无需专业 HMI 软件的优势。广泛应用于装配设备、加工中心、木工门窗、涂胶设备、多关节机械手轨迹规划和上下料系统。TwinCAT NCI 可以实现 3 轴插补以及五个辅助轴的运动，实现机构在空间中以任意的坐标轨迹运动。TwinCAT NCI（TF5100）支持 G 代码指令，同时 NCI 在 G 代码的基础上又发展出 GST 语言编程，可以更好地将 G 代码与 ST 语言相结合。NCI 系统包含多种运动学坐标系变换模型，包括 DELTA 模型、SCARA 模型、六轴机器人模型等。另外 NCI 在原有 G 代码的基础上又针对机械手的应用开发了 PickAndPlace 抓放功能库，为机械手的空间插补轨迹规划提供有力支持。

本书第一章从 NCI 的基础配置讲起，带大家快速配置一个 NCI 通道，同时介绍了常用的 M 函数和 R 变量以及 NCI 的 PLC 功能块和常用 G 代码。学习完第一章的前五节，可以算是完成了 NCI 的入门，能够开发一些简单的 NCI 项目。本章 6 到 12 节详细介绍了 NCI 通道的参数含义和设置方法、零点偏移、NCI 的平滑过渡曲线类型等，便于对 NCI 有一定掌握的人员系统学习 NCI 的机制和参数含义。这样本书既适用于初学 NCI 的技术人员快速入门，又适合对 NCI 有一定使用的人员掌握常用的配置参数的含义和设置方法以及系统学习曲线过渡算法等。

本书第二章主要介绍运动学坐标系的变换，运动学坐标系的变换是我们常说的机器人关节机械臂模型，是 NCI 技术的拓展。本章第一节以 Delta 机器人为例，详细介绍了机器人模型的配置流程和参数含义。随后分别介绍了 SCARA 机器人、2D 并联、2D 串联、6 轴串联、6 轴并联等机器人模型参数含义和坐标系定义。第 5 节和第 6 节介绍了运动学静态模型转换和机器人力矩前馈添加方法。本章第 7 节介绍了 TwinCAT3 的新功能怎么利用 TcCom 功能采用 C++编程方式快速开发一个机器人运动学变换模型。

本书第三章主要介绍了 PickAndPlace 抓放功能库的配置和使用步骤。PickAndPlace 是 NCI 功能的拓展，主要用于机器人轨迹规划，以及与相机和传送带配合的抓放运动。它直接在 PLC 中编辑路径不需要采用 G 代码，可以更方便地与视觉和传送带配合。

本书第四章主要介绍了 GST 语言的语法和功能。GST 语言把高级语言完美地集成在 CNC 编程中，它将描述轮廓的 G 代码(DIN66025)与 PLC 编程用的结构文本 ST 语言结合在一起，使机器制造商能以一个简单的方式实现复杂的编程。GST 使 NCI 可以方便地实现迭代，变量声明，功能与功能块等。

本书前三章由王进编写，第四章 GST 语言由张寅编写，所有内容是作者对 NCI 系统的个人技术总结，由于作者能力有限如有纰漏或者不足之处还请谅解和指正。本书编写过程中得到其他技术人员的支持，如第一章第一节的 NCI 的配置参考了技术支持部的文档，技术支持部工程师张立文也对本书进行了校对，并根据 400 技术热线收到的 NCI 热门问题，提出了增删部分内容的宝贵建议，在此向他们表示感谢。

目录

第一章 NCI 插补技术

NCI 是 Numerical Control Interpolation 的简称，是倍福自主研发的 CNC 系统。TwinCAT NCI 可以实现 3 轴插补，实现机构在空间中任意的坐标轨迹运动。在三维空间里，最常见的是二维平面上的线段，比如 XY 平面上的直线和圆弧以及由直线和圆弧合成的其他图形，都可以由 NCI 实现轨迹运动。另外也可以在 XY 轴做圆弧插补的同时，Z 轴上下移动 ，就会在空间上形成一个螺旋轨迹，从而构成三轴插补。也可以实现三维空间中给定大量点的样条曲线插补，NCI 系统会自动在点与点之间插值，并按设定参数合并或修改小线段以及过小的过渡角，从而保证最快的速度和最小的形变。

TwinCAT NCI（TF5100）支持 G 代码插补指令，G 代码文件是若干行 G 代码的指令 ，有一套标准规范，通常采用 DIN66025。如直线插补指令 G01，圆弧插补指令 G02/G03，以及在 G 代码指令执行过程中发出的外部开关状态指令 M 函数。TwinCAT NCI 包含了 G 代码编译和预读功能，在执行 G 代码文件的时候，NCI 会预读 G 代码行，结合插补通道内每个轴的当前位置，计算出每个轴接下来在每个控制周期的给定位置。

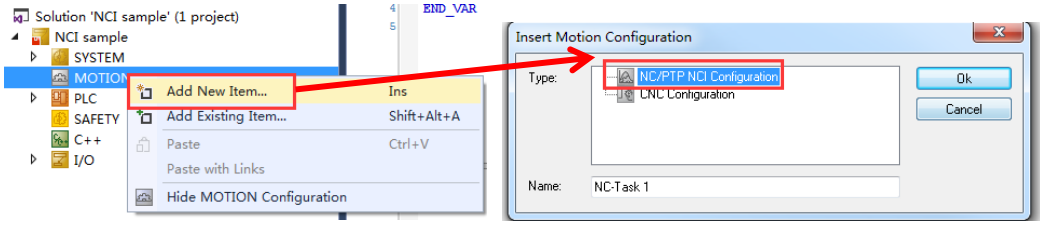
TwinCAT NCI 在做插补运动时，所有轴的物理层都是在 NCPTP 轴中与电机驱动器进行关联的。TwinCAT NCPTP 将一个电机的运动控制划分成：PLC 轴、NC PTP 轴和物理轴。而 TwinCAT NCI 把一个联动机构的控制分成三层：PLC 插补通道、NCI 插补通道、NCPTP 轴。所以使用 NCI 系统的控制器具有更大的灵活性，既可以把 PTP 轴配置到 NCI 插补通道，也可以随时分离出来当普通 PTP 轴使用。

每个控制器最多可以配置 31 个 NCI 通道，每个通道最多包含 3 个插补轴和 5 个辅助轴。3 个插补轴的运动方向在空间上存在正交关系，通常我们会命名为 X Y Z 轴，NCI 的速度就是指空间坐标系三个插补轴的合成速度。5 个辅助轴与插补轴之间没有严格的空间关系，但是可以与插补轴同时启动，同时到达预定位置。

1.1 NCI 的配置和调试

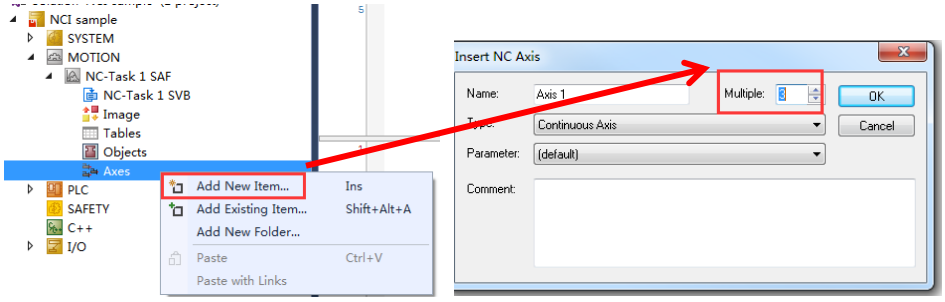
NCI 的配置方式和 NC PTP 有比较大的区别，NC PTP 可以直接通过 axis 的 online 窗口对物理轴进行调试，而 NCI 则必须通过调用 G 代码才可以让电机正反转，下面以虚轴为例，介绍如何在软件中对 NCI 功能进行调试。

1.1.1 新建一个 NCI Group

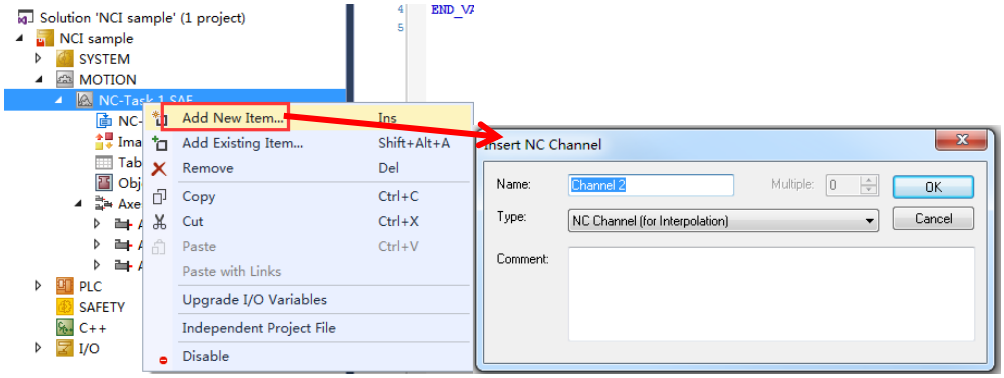


由于本次采用的是虚轴模拟的，所以需要先建三根虚轴：

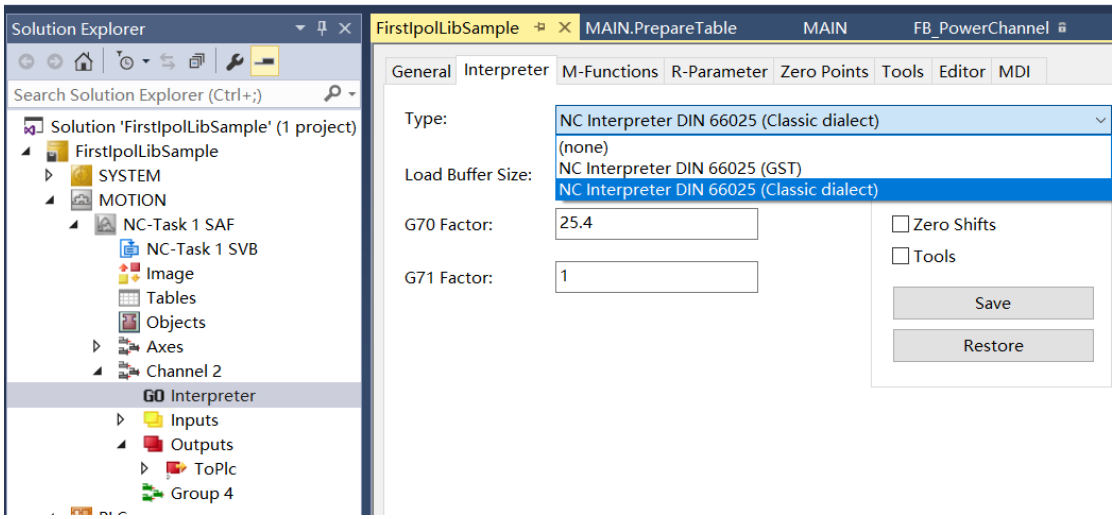
1）添加三个 NC 轴，直接在 Multiple 里面设置 3,可以一下子添加 3 根轴。



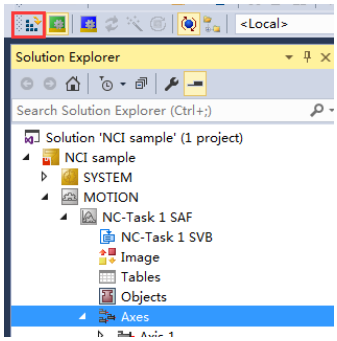
2）右键 NC_Task1 SAF，手动添加一个 NCI 的通道。



3）本教材采用 Simens 编程 G 代码指令，所以设置插补通道支持 G 代码指令格式为 NC Interpreter DIN66025（Classic dialect），老版本以及 TwinCAT2 是 Simens dialect。该设置在 TC3 4022.2 版本中需要手动修改，其他版本默认就是该类型。注意不要选择 GST，关于 GST 的编程后文有专门的章节讲解。

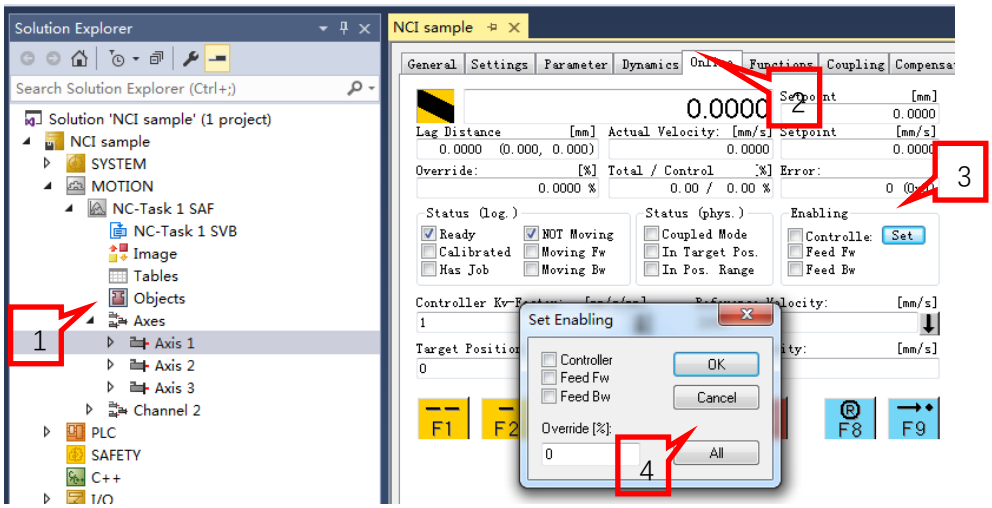


4）激活配置使前面添加的虚轴以及 NC 插补通道生效，输入验证码并切换 TwinCAT 至运行模式。

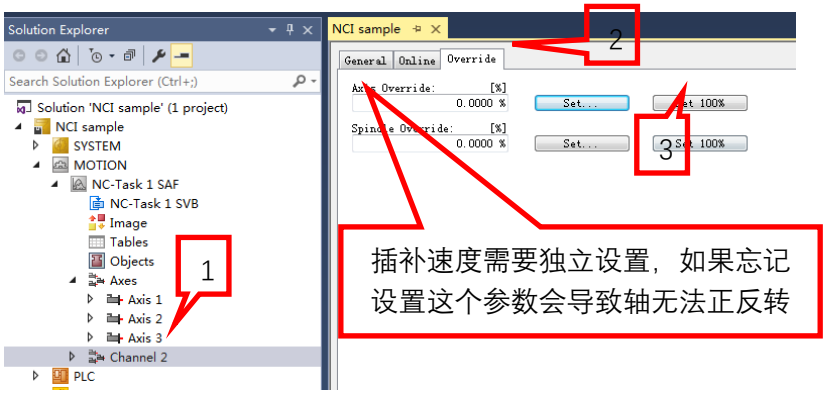


1.1.2 NCI 的调试

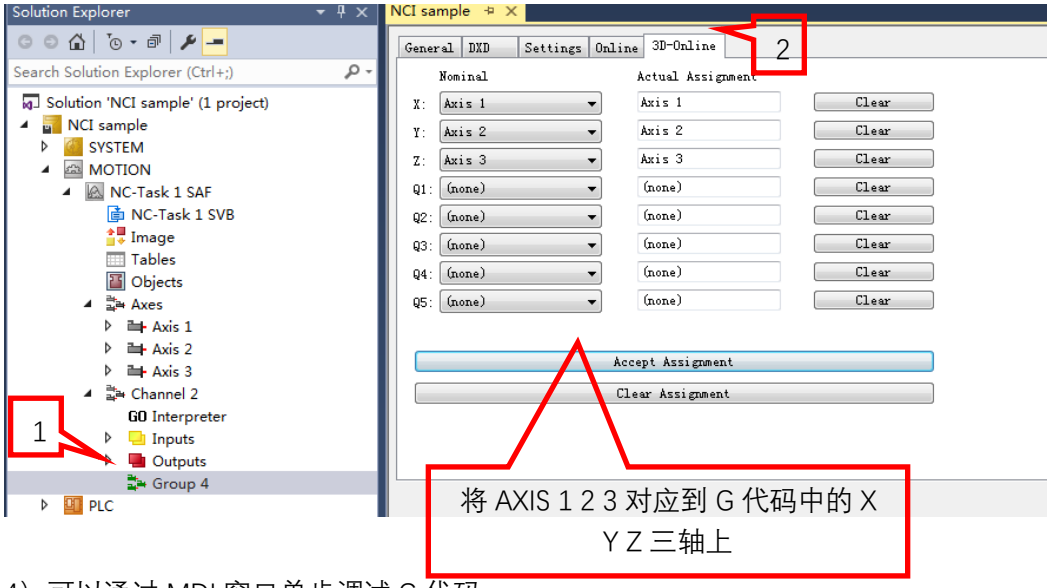
1）将三个虚轴使能，先选中 Axis 1，点击 Online 选项卡，点击 Set，点击 All 对轴进行使能，依次对 Axis2, Axis3 都使能上。



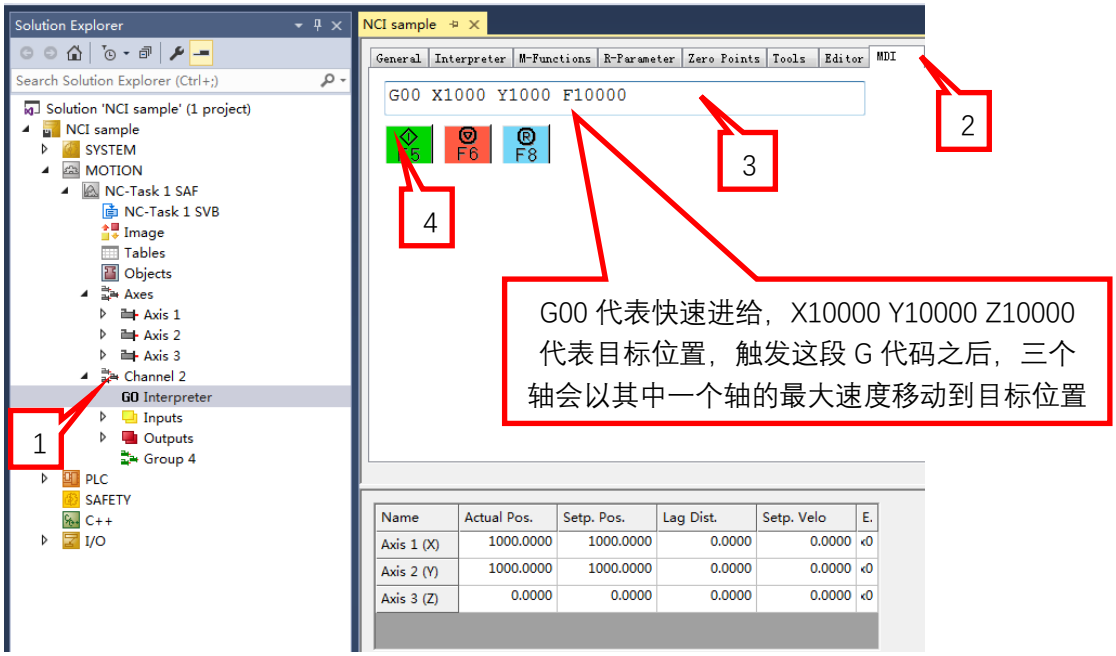
2）设置 NCI 的插补的速度比为 100%。



3）建立 G 代码中 XYZ 轴和 NC 轴的对应关系。



4）可以通过 MDI 窗口单步调试 G 代码。

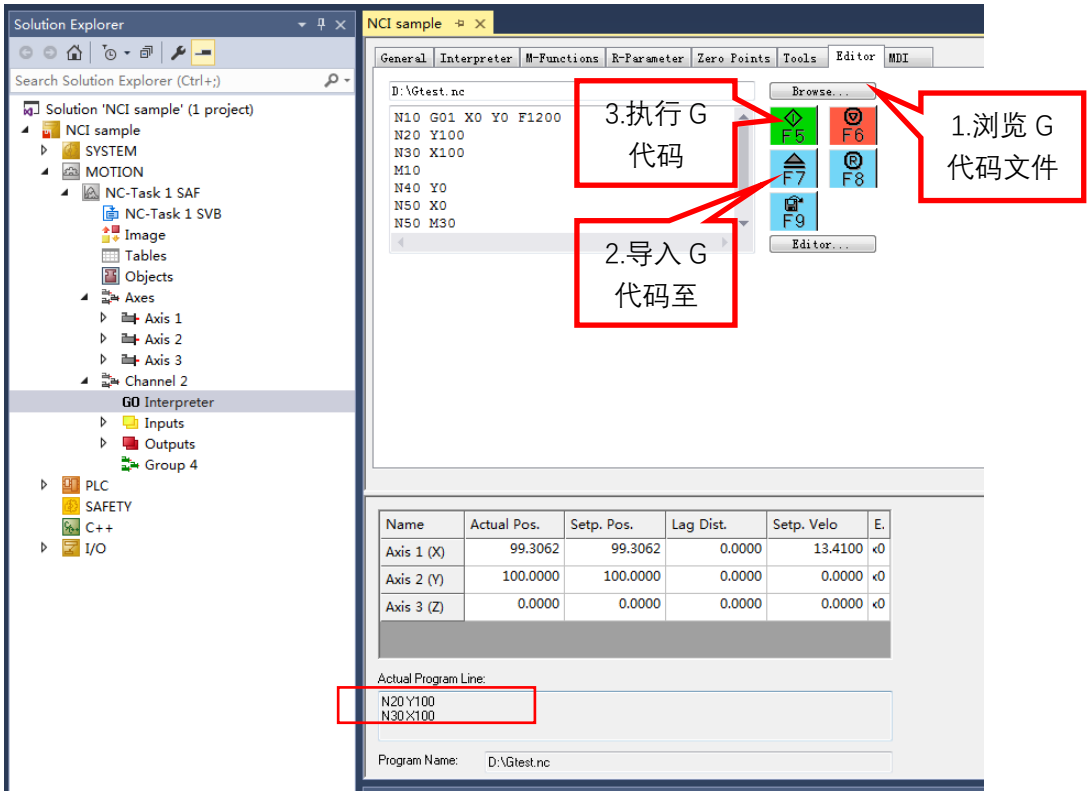


5）Editor 窗口中可以对完整的 G 代码文件进行调试。

调试步骤如下：

① 点击 Browse 选择 G 代码文件，按 F7 将 G 代码文件导入到 NCI 中准备执行，可以在 Program Name 中看到当前已经导入的 G 代码文件；

按 F5 执行 G 代码，此时可以看到 Actual Program line 中当前正在执行的 G 代码行，X Y Z 轴会根据 G 代码的内容执行相应的动作。



1.2 M 函数的使用

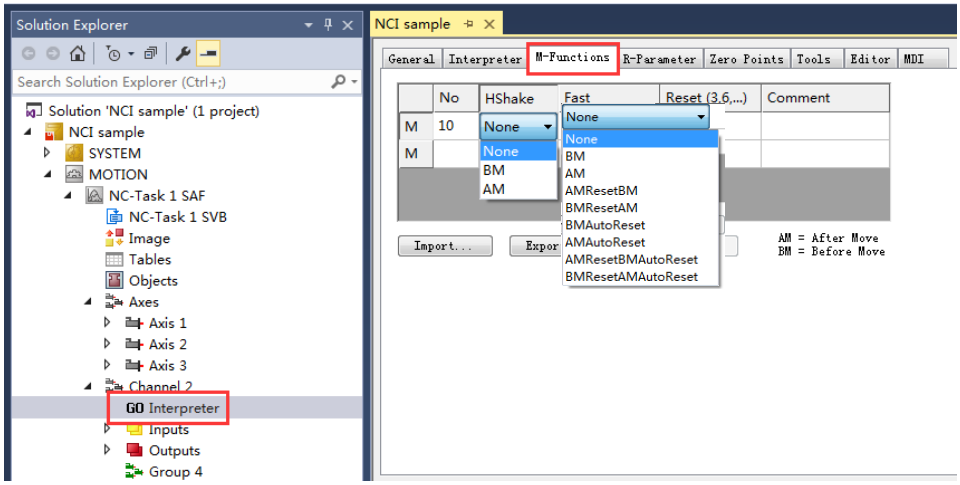
在 G 代码程序中，还会用到 M 函数。M 函数起到 G 代码与 PLC 程序交互的作用，比如 G 代码执行过程中，需要 PLC 里面做一些处理，比如换刀，吹气等操作，可以通过 M 代码来完成，除了部分根据国际标准已经指定好固定用途的 M 函数，根据 M 函数是否打断 G 代码预读，可以被分为握手型 M 函数 Hshake 和快速 M 函数 Fast。握手型 M 函数需要 NCI 与 PLC 握手，M 函数在 NCI 通道中被触发，在 PLC 程序确认这个 M 函数之后，才能继续执行后面的 G 指令。而快速 M 函数不需要 PLC 程序去确认，只是起到通知 PLC 的作用。

1.2.1 M 函数的定义

M 函数	含义
0..159	可以任意定义的 M 功能（除了 2，17，30）
2	程序结束
17	子程序结束
30	程序结束并复位所有快速 M 函数

1.2.2 M 函数的配置

打开 Go Interpreter 的“M-Function”的选项卡，可以对 M 函数进行配置。



其中：

No M 函数的编号，最多可设置 159 个，M2、M17、M30 有特殊应用，不能用于程序中自定义。

G 代码程序：N10 X100 Y100 M10

HShake 握手型 M 函数的配置。

Fast 快速 M 函数的配置。

Reset 触发该 M 函数后对其他 M 函数进行复位

Commet 备注

AM 表示 After Motion，即当 G 代码行有 M 函数时，M 函数会在 G 代码行的运动程序运行完成后被激活；

BM 表示 Before Motion，即当 G 代码行有 M 函数时，M 函数会在 G 代码行的运动程序运行前被激活；

None 指不需要 PLC 握手确认。

1.2.1 握手型 M 函数

Hshake：握手型 M 函数

使用握手型 M 函数，输出后如果没有复位，程序会在当前程序行暂停，直到 M 函数复位后才会向下运行，这是与快速 M 函数的区别。所以同时只会触发一个 M 函数，可以通过 ItpIsHskMFunc 获取是否有 M 函数触发；ItpGetHskMFunc 函数获取 M 函数编号；复位采用 ItpConfirmHsk 函数实现。

样例程序如下：

```
(*hsk M Function31-95,Fast M 96-159*)
nActMFunNr:=ItpGetHskMFunc(sNciToPlc:=stNciToPlc );
bMFunEnable:=ItpIsHskMFunc(sNciToPlc:=stNciToPlc);
IF bMFunEnable THEN
    bMFunNr^[nActMFunNr]:=TRUE;
ELSE
    nActMFunNr := -1;
    bConfirm := FALSE;
        FOR i := nStartIndexMFunction TO 95 DO
        bMFunNr^[i] := FALSE;
        bQuitMFunNr^[i] := FALSE;
    END_FOR
END_IF

(* confirm m function, if bit in array bQuitMFunc is set *)
    IF (nActMFunNr <> -1) THEN
        IF bQuitMFunNr^[nActMFunNr] THEN
            bConfirm := TRUE;
        END_IF
        bQuitMFunNr^[nActMFunNr] := FALSE;
    END_IF

fbConfirm(
        bExecute := bConfirm,
        sNciToPlc:= stNciToPlc,
        sPlcToNci:= stPlcToNci
            );
```

1.2.2 快速 M 函数

Fast：快速 M 函数。

如果直接使用 AM 或者 BM，M 函数触发之后会一直保持为 True，复位需要使用 PLC 中的功能块 ItpResetMFuncEx，但程序不会因为运行到 M 函数而停止。类型选择中有 AutoReset 的表示自动复位该 M 函数。

获取快速 M 函数的状态通过 ItpIsFastMFunc 函数，复位该 M 函数需要通过 ItpResetFastMFuncEx 功能块实现。

样例程序如下，本例中把 M96-M159 作为快速 M 函数：

```
(*fast M function*)(*hsk M Function31-95,Fast M 96-159*)
FOR i:=96 TO 159 DO
bMFunNr^[i]:=ItpIsFastMFunc(UINT_TO_INT(i),stNciToPlc);
fbResetFastMFunc(
    bExecute:= bQuitMFunNr^[i],
    nMFuncNo:= i,
    tTimeOut:= t#100ms,
    sNciToPlc:= stNciToPlc,
    bBusy=> ,
    bErr=> ,
    nErrId=> );
IF NOT bMFunNr^[i] THEN
bQuitMFunNr^[i]:=FALSE;
END_IF
END_FOR
```

1.3 R 参数的使用

G 代码中可以使用固定值编程运动曲线，例如：X100 Y100 F1000，也可以使用 Lreal 类型的变量替代常量增加灵活性，NCI 中称为 R 参数。

如 G01 X100 Y200

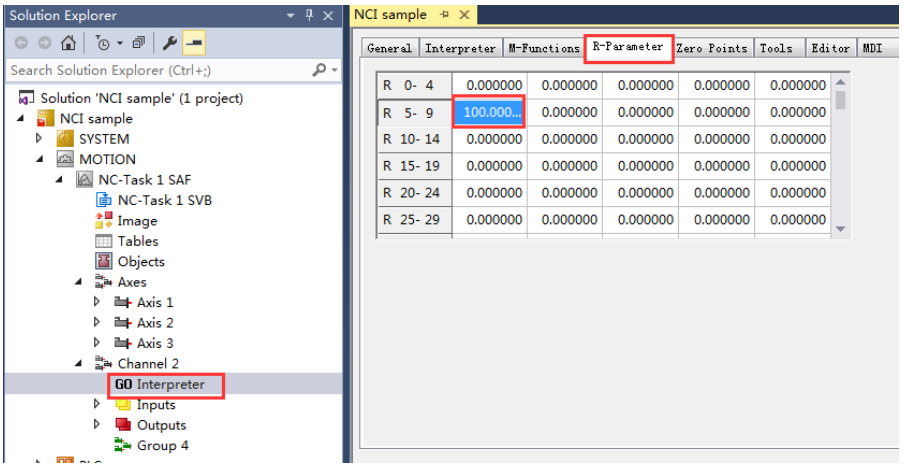
可改成：R5=100

G01 X=R5 Y=2*R5

其中每个 NCI 通道可设置 1000 个 R 参数，从 R0 到 R999，类型均为 Lreal 型，要注意的是 NCI 会对 G 代码进行预读，因此 R 参数需要提前修改。如果在运行中需要修改 R 参数，需要用到终止预读指令@714，通常终止预读用在握手型 M 函数后面。

R 参数有三种访问方式：

1.3.1 “R-Parameter”选项卡

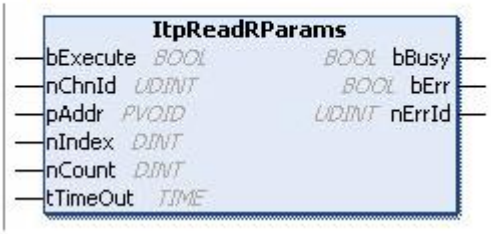


可以在 R 参数选项卡读写 R 参数。

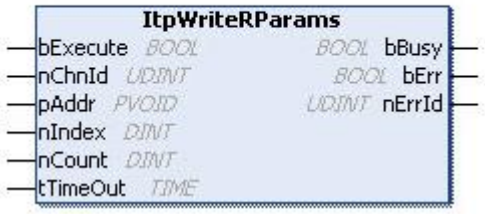
1.3.2 PLC 程序读写

可以通过调用 Tc2_NCI 库中的 ItpReadRParams（读），ItpWriteRParams（写）功能块实现 R 参数读写。

ItpReadRParams



ItpWriteRParams



其中

nChnId 是绑定需要修改 R 参数的 NCI 通道；

pAddr 是 R 变量的地址，如果是数组则为首位的地址；

nIndex 是数组的首位；

nCount 是数组的长度；

比如要写 R0-R200 的变量程序如下

```
fbWriteRParams(
          bExecute:= TRUE,
          sNciToPlc:= stNciToPlc,
          pAddr:= ADR(nRParamWrite^[0]),
          nIndex:= 0,
          nCount:= (200 - 0) + 1,
          tTimeOut:= t#2s,
          bBusy=> ,
          bErr=> ,
          nErrId=> );
```

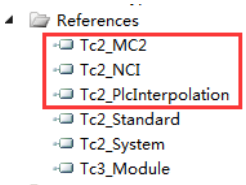
1.3.3 G 代码读写

赋值　N10 R5=100
调用　G01 X=R5 Y=2*R5

1.4 NCI 的 PLC 编程配置

NCI 支持 3 轴插补以及 5 个辅助轴运动，本章结合实验程序采用三个插补轴和一个辅助轴。首先需要在 NC 配置四个轴，分别是插补轴 X Y Z 和辅助轴 C，以及一个 NCI 插补通道，方法见本章第一节。

其次通过程序做 NCI 控制，需要添加三个库文件：①Tc2_MC2 ②Tc_NCI.lib ③Tc2_PlcInterpolation

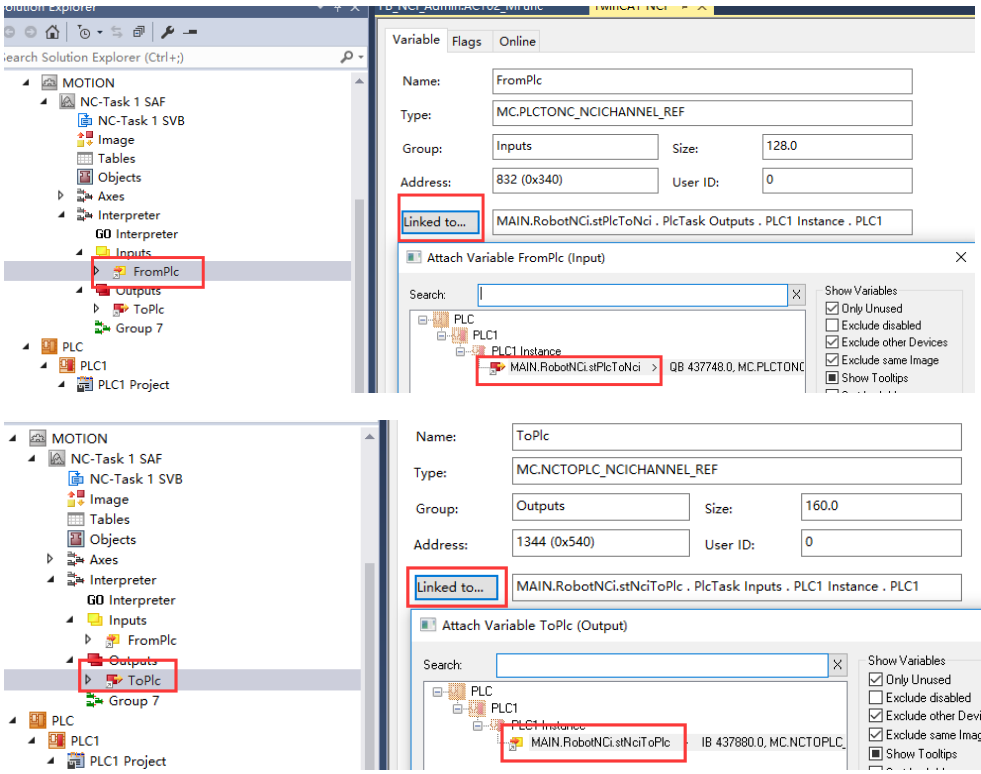


另外还需要定义轴类型结构体，关联到 NC 轴通道，并使用 MC_Power 功能对这些轴进行使能。

```
PROGRAM MAIN
VAR
    bUserEnableAxes:        BOOL    := TRUE;        (*触发使能功能块，由于是模拟程序，所以直接对轴组进行使能*)
    fUserOverride:          LREAL   := 100.0;       (*设置通道速比为100%*)

    io_X:                   AXIS_REF;               (*定义各NC轴*)
    io_Y:                   AXIS_REF;
    io_Z:                   AXIS_REF;
    io_Q1:                  AXIS_REF;
    io_Q2:                  AXIS_REF;
    io_Q3:                  AXIS_REF;

    in_stItpToPlc   AT %I*: NcToPlc_NciChannel_Ref; (*定义NCI通道反馈给PLC的一系列状态字等*)
    out_stPlcToItp  AT %Q*: PLCTONC_NCICHANNEL_REF; (*定义PLC发送给NCI通道的一系列状控制字等*)
```

1.4.1 NCI 的结构体

声明 NCI 结构体 NcToPLc_NCIChannel_Ref 和 PLcToNc_NCIChannel_Ref,如上图并把变量链接到 NC channel 中去，如下图。



声明本例中所用到的其他功能块，如下图:

```
VAR
    sPlcToNci AT %Q*:  PLCTONC_NCICHANNEL_REF;
    sNciToPlc AT %I*:  NcToPlc_NciChannel_Ref;
(*********************************************)
    fbBuildGroup: CfgBuildExt3DGroup;   (*配置NciGroup*)
    fbLoadProg: ItpLoadProgEx;  (*下载Nci程序*)
    fbStart: ItpStartStopEx;(*启动和停止*)
    fbItpEStopEx: ItpEStopEx;(*暂停*)
    fbItpStepOnAfterEStopEx: ItpStepOnAfterEStopEx;(*暂停后的启动*)
    fbClearGrp: CfgReconfigGroup;(*清除NciGroup*)

    fbResetEx2: ItpResetEx2 ;(*复位*)
(*********************************************)
    fbResetFastMFunc:ItpResetFastMFuncEx;(*复位快速M函数*)
    fbConfirm: ItpConfirmHsk;(*复位握手型M函数*)
(*********************************************)
    fbWriteRParams:          ItpWriteRParamsEx;(*写R变量*)
    fbReadRParams:           ItpReadRParamsEx;(*读R变量*)
(*********************************************)
```

1.4.2 创建 Group 功能块

使用 CfgBulidExt3DGroup 功能块创建 NCI 并通过 nGroupId 对应到 NCI 的 Channel，可以理解为把轴通道配置到 NCI。如果有不用的轴则对 AxisID 赋值 0。其中 nGroupId 可以通过 ItpGetGroupID 函数获取。

```
20:

(* build interpolation group *)

        fbBuildGroup(
        bExecute    := TRUE ,
        nGroupId    := nGroupID,
        nXAxisId    :=  pAxis_X^.NcToPlc.AxisId,
        nYAxisId    :=  pAxis_Y^.NcToPlc.AxisId,
        nZAxisId    :=  pAxis_Z^.NcToPlc.AxisId,
        nQ1AxisId   :=  pAxis_C^.NcToPlc.AxisId,
        nQ2AxisId   := 0,
        nQ3AxisId   := 0,
        nQ4AxisId   := 0,
        nQ5AxisId   := 0,
        tTimeOut    := T#200MS);


IF NOT fbBuildGroup.bBusy THEN
    fbBuildGroup( bExecute := FALSE );
    IF NOT fbBuildGroup.bErr THEN
        nState  := 30;
    ELSE
```

1.4.3 下载 G 代码功能块

需要编辑好 G 代码命名并放到对应的文件下。
使用 ItpLoadProg 功能块把 G 代码文件下载到 NCI 的 Channel 中
其中 SPrg 是 G 代码的名称，string 字符串变量，nLength 是 G 代码名称的长度
例如：如果是 Win7、Win10、XP 系统

ProgramName:='C:\TwinCAT\Mc\NCI\Test.nc';

如果是 CE 系统

ProgramName:='\Hard Disk\TwinCAT\CNC\Test.nc';

```
        10:

    (* load part program *)

    fbLoadProg(
        bExecute    :=  TRUE,
        sPrg        := ProgramName,
        nLength         := INT_TO_UDINT (LEN(ProgramName)),
        tTimeOut    := T#2000MS,
        sNciToPlc:= stNciToPlc );

    IF NOT fbLoadProg.bBusy THEN
        fbLoadProg( bExecute:=FALSE, sNciToPlc:=stNciToPlc);
        IF NOT fbLoadProg.bErr THEN
            nMotionState    := 20;
        ELSE
            (* an error occured *)
            bErr        := TRUE;
            nErrId  := fbLoadProg.nErrId;
            nLastState  := nState;
            nMotionState    := 9999;
        END_IF
    END IF
```

1.4.4 读取 NCI 通道的状态函数

可以使用函数 ItpGetStateInterpreter 读取 NCI 的状态。

```
(*chanel state*)
uiItpState  :=  ItpGetStateInterpreter( stNciToPlc );
```

返回值类型：UDINT

着重介绍重要的几个状态

uiItpState 返回值	状态	解释
1	ITP_STATE_IDLE	该状态未加载任何 G 代码程序或是刚执行完复位功能，插补通道处于空闲状态。若当前执行 G 代码停止也会进入该状态。
2	ITP_STATE_READY	成功导入 G 代码程序，插补通道进入准备状态。当 G 代码程序被完整执行完，插补通道也会进入该状态
12	ITP_STATE_ABORTED	如果在 G 代码运行过程中出现报错，插补通道立即进入该模式，当前报错代码也会出现在通道状态中
10	ITP_STATE_SINGLESTOP	此状态仅在 Single Block Mode.中出现。一旦进入 Single Block Mode.，插补通道切入该状态（结合 1.4.10 学习）

其他模式请查看：

https://infosys.beckhoff.com/content/1033/tcnci/9007201622683275.html?id=5407071450484099651

1.4.5 G 代码的运行功能块

通过 ItpStartStopEx 功能块执行 NCI 代码，如下图：

ItpStartStopEx 功能块的 bStart 和 bStop 输入都是上升沿触发，同时 bStop 的优先级高于 bStart，所以如果同时触发信号，bStop 将优先执行。该停止指令将删除 NCI 通道中的数据，不支持从停止前的状态的启动。程序如下

```
30:
    (* start part program *)
    fbStart(
        bStart  := TRUE,
        bStop       := FALSE,
        tTimeOut    := t#200ms,
        sNciToPlc:= stNciToPlc);

    IF NOT fbStart.bBusy THEN
        fbStart( bStart:=FALSE, sNciToPlc:=stNciToPlc );
        IF NOT fbStart.bErr THEN
            nMotionState    := 50;
        ELSE
            (* an error occured *)
            bErr        := TRUE;
            nErrId  := fbStart.nErrId;
            nLastState  := nState;
            nMotionState    := 9999;
        END_IF
    END_IF
```

1.4.6 清除 Group 功能块

代码执行完毕，通过 CfgReconfigGroup 清除 NCI 通道，与创建 Group 功能块 CfgBulidExt3DGroup 相对应。

```
10:

    (* clear interpolation group *)
        fbClearGrp(
        bExecute:= TRUE,
        nGroupId:= ItpGetGroupId(sNciToPlc:=stNciToPlc),
        tTimeOut:= T#200ms);

    IF NOT fbClearGrp.bBusy THEN
        fbClearGrp(bExecute:= FALSE);
        IF NOT fbClearGrp.bErr THEN
            nState  := 20;
        ELSE
            (* an error occured *)
            bErr        := TRUE;
            nErrId  := fbClearGrp.nErrId;
            nLastState  := nState;
            nState  := 9999;
        END_IF
    END_IF
```

1.4.7 NCI 的复位功能块

这里推荐采用新版本的 ItpResetEx2 功能块，可以实现在 NCI 通道报错时的复位。参考程序如下：

```
fbResetEx2(
    bExecute:=bReset ,
    tTimeOut:=t#2s ,
    sNciToPlc:= stNciToPlc,
    bBusy=> ,
    bErr=> ,
    nErrId=> );
```

1.4.8 停止和停止后的启动功能块

功能块 ItpEStopEx 触发 NCI 的急停并以设定的减速度 fDec 和加加速度 fJerk 作为限制参数进行停止，当这两个参数比当前系统被激活的动力学参数更大时有效,以使系统在更短的时间内停下来。

功能块 ItpStepOnAfterEStopE 应用在执行过 ItpEStopEx 后，如果要继续运行 G 代码，可以触发该功能块。

1.4.9 修改 NCI 的 Override 功能

在运行过程中有时候如果需要对 NCI 的速度进行实时调节，可以利用 ItpSetOverridePercent 功能实现。该参数单位为百分比，范围为 0-100。

1.4.10 ItpSingleBlock 单步运行

bExecuteModeChange：上升沿触发单步模式启动

nMode：模式选择一共有三个模式

1）ItpSingleBlockOff

不启用单步模式

2）ItpSingleBlockNck

使用 NC 核单步模式时，G 代码程序在解释器中被预读（利用 M 函数和中止预读指令实现中止预读的除外）。也就是说不会提前预读当前程序行后面的程序。R 变量不会在后面的程序行中刷新。

这种模式相比较编译器单步模式有个优势是可以在 G 代码运行的过程中启动。

3）ItpSingleBlockIntp

使用编译器单步模式时，G 代码程序在编译器中停止。也就是说不会提前预读当前程序行后面的程序。R 变量会在后面的程序行中刷新。

这种模式不能在程序运行过程中启动，如果一定要在 G 代码运行过程中使用，需要利用 M 函数和中止预读指令实现中止预读后启用。

bTriggerNext：触发单步运行

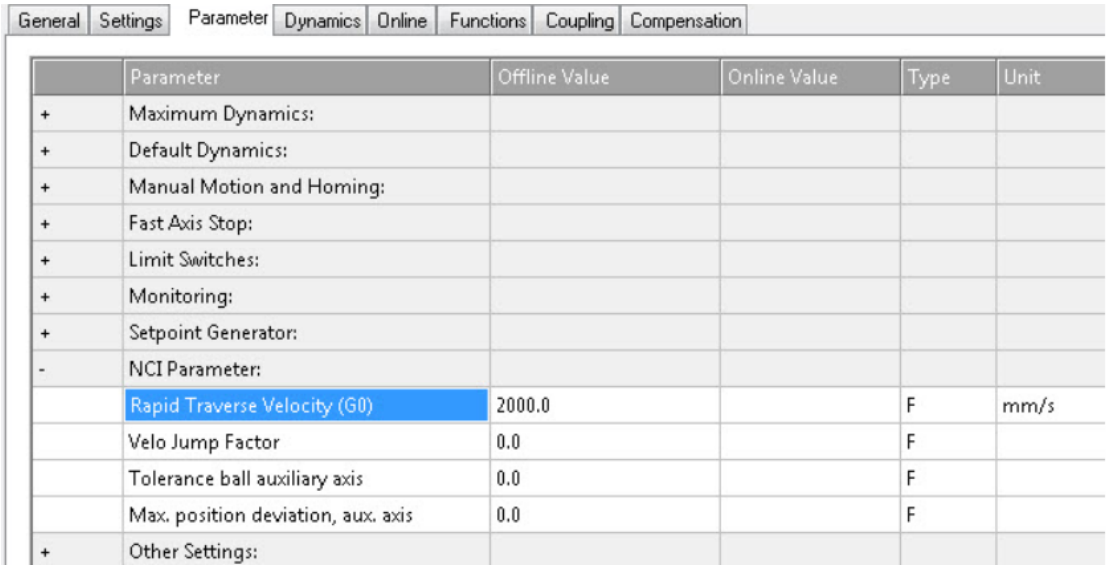
tTimeOut： ADS 超时设置

1.5 简单的 G 代码

1.5.1 G00 快速移动

各轴分别单独移动到设定位置，无插补配合。如果采用 G00 快速移动，轴的速度由 NC 轴参数中的 Rapid Traverse Velocity （单位 mm/s）决定。

例 G00 X100 Y150



	Parameter	Offline Value	Online Value	Type	Unit
+	Maximum Dynamics:				
+	Default Dynamics:				
+	Manual Motion and Homing:				
+	Fast Axis Stop:				
+	Limit Switches:				
+	Monitoring:				
+	Setpoint Generator:				
-	NCI Parameter:				
	Rapid Traverse Velocity (G0)	2000.0		F	mm/s
	Velo Jump Factor	0.0		F	
	Tolerance ball auxiliary axis	0.0		F	
	Max. position deviation, aux. axis	0.0		F	
+	Other Settings:				

1.5.2 G01 直线插补

XYZ 轴以插补方式运行到目标位置，合成速度由指定的速度决定，F6000 是指定速度，单位 mm/min。

例 N10 G90

N20 G01 X100.1 Y200 F6000

辅助轴的运动指令如下：

(start position X=Y=Z=Q1=0)

N10 G01 X100 **Q1=47.11** F6000

其中辅助轴的运行时间和插补轴同时启动同时结束。

1.5.3 G02，G03 圆弧插补

G02 是顺时针运动；G03 是逆时针运动；

圆弧插补需要首先指定圆弧平面

默认 XY 平面（G17）

ZX 平面（G18）

YZ 平面（G19）

1） 半径编程

这种方式通常用于实现一个非完整圆的圆弧曲线，通过指定起始点与结束点和圆弧半径（例子中 B200 指半径 200mm）实现圆弧插补。如何要实现一个大于 180 度的圆弧，需要把半径设置为负。

例：N10 G01 G17 X100 Y100 F6000
N20 G02 X200 B200

2） 中心点编程

中心点编程以相对 I（X 部分）、J（Y 部分）、K（Z 部分）参数为圆心，以上一条指令的结束点为起点进行圆弧插补;

例：N10 G01 G17 X100 Y100 F6000
N20 G02 I50 J0 (J is optional) X200
N30 M30 (program end)

本例中，以（150,100）为圆心，从起始点（100,100）顺时针运行到（200,100），半圆。

例：N10 G01 G18 X100 Y100 Z100 F6000
N20 G02 I0 K50 X150 Z150 (quarter circle in ZX plane)
N30 M30

本例中，G18 指定插补平面 ZX，以（100,150）为圆心，从起始点（100,100）运行到（150,150）四分之三圆。

以上 I、J，K 都是表示以起始点为基准的相对值，如果要使用绝对值，需要通过@402 修改标志位实现，如下例子。

例：N10 G01 G17 X100 Y100 F6000
N20 @402 K5003 K5 K1 (centre point programming on)
N30 G02 I150 J100 X200
N40 @402 K5003 K5 K0 (centre point programming off)
N50 M30

其中@402 K5003 K5 K1 是把机器数据 5003 的第五位置为 1，以开启绝对值模式。
@402 K5003 K5 K0 是把机器数据 5003 的第五位置为 0，以关闭绝对值模式。

3）CIP 空间圆弧

上面两种圆弧指令都需要指定插补平面 XY 或者 ZY、ZX，CIP 指令可以实现空间圆弧。

例：N10 G01 X100 Y100 F6000
N20 CIP X200 Y200 I50 J50 K50

本例中，以（150,150,50）为圆心，从起始点（100,100,0）运行到（200,200,0）。

1.5.4 G04 停顿时间

例：N10 G01 X100 F6000
N20 G04 X0.5 (pause in sec)
N30 G02 X300

本例中 G04 X0.5 是在完成前一指令后停顿 0.5s。

1.5.6 跳转

1）无条件跳转 @100 K or R

例：N10 ..

...

N120 @100 K-10

本例中从 N120 向上跳转到第 N10，如果是向后跳转，K 参数为正，反之为负。

2）条件跳转 @121 R<n> K or R<m> K

例：N10 ..

...

R1=14

N120 @121 R1 K9 K-10

N130 ...

本例中，如果 R1 不等于 9 ，则跳转到 N10，否则顺序执行。

1.5.7 子程序调用

有些加工过程中，基础代码不变，而工件的 G 代码会经常变换，这时候可以使用子程序。

例：(file L2000.NC)

L2000

N100...

N110...

...

N5000 M17 (结束子程序)

主程序的调用：N100 L2000 (call)

1.5.8 修改加速度参数

#set paramPathDynamics(~~<grp>,~~<acc>,<dec>,<jerk>)#

例：N10 G01 X100 Y200 F6000

N15 R4=3000

N20 #set paramPathDynamics(700; 700; R4)#

N30 G01 X500

1.5.9 中止预读

G 代码程序会一次性预读到 NCI 中，但是有些路径参数可能在程序执行的过程中才能计算出，比如机械手配合视觉相机实现飞抓功能，抓取位置需要运行完一段程序后才能给出。这时候可以通过中止预读来实现。编译器一旦运行到中止预读指令行，编译器会一直等待，直到某个外部事件发生。在此事件发生之前，NC 程序不会继续执行。

（1）@714

中止预读后继续预读执行有两种方式：

1） 握手型 M 函数中止预读

例：N10...

N20 M43 (M-function with handshake)

N30 @714 (decoder stop)

N40 ...

本例中可以在 PLC 中复位 M43 函数后，继续预读和执行程序

2） SAF 任务是空的

编译器预读的停止不一定要与 M 函数一起编程。如果 SAF 任务执行完了所有运动命令，则向编译器发送一个事件，这导致编译和预读的重新执行。

（2）@716 中止预读后重新扫描轴位置

使用@716 在编译器预读的停止处将再次读取插补通道的轴位置。

例如，如果在工具切换期间通过 PTP 移动了轴，并且该轴随后没有返回到原来的位置，可以使用@716 从新扫描轴位置。另一个可能的应用是通过 M 函数(握手型 M 函数)暂停后更改 axis 配置，如 offset。

注意：当工具补偿或圆平滑被激活时，中止预读失效。

（3）@717 外部事件触发中止预读

在实际使用中有时候要根据 PLC 的运行情况选择是否把运行的曲线连起来，这时候可以用到@717 配合 PLC 中的外部触发解除中止预读功能块完成。

如下面案例程序，触发快速 M 函数 M70 后，NC 继续运行 N50 行运动指令，同时 PLC 运行事件 A, A 事件处理完成后发送 GoAhead 命令。如果 GoAhead 信号足够早地到达 PLC，则将 N50 和 N70 行的运动指令连接起来，并且不降低路径速度。如果在 N50 减速阶段到达，则再次提高速度。否则，机器将停止并等待来自 PLC 的信号 GoAhead。

例：

N10 ...

N20 G0 X0 Y0 Z0

N30 G01 X500 F6000

N40 M70 (flying M-function that triggers process A)

N50 G01 X700

N60 @717 (decoder stop with external trigger event)

N70 G01 X1000

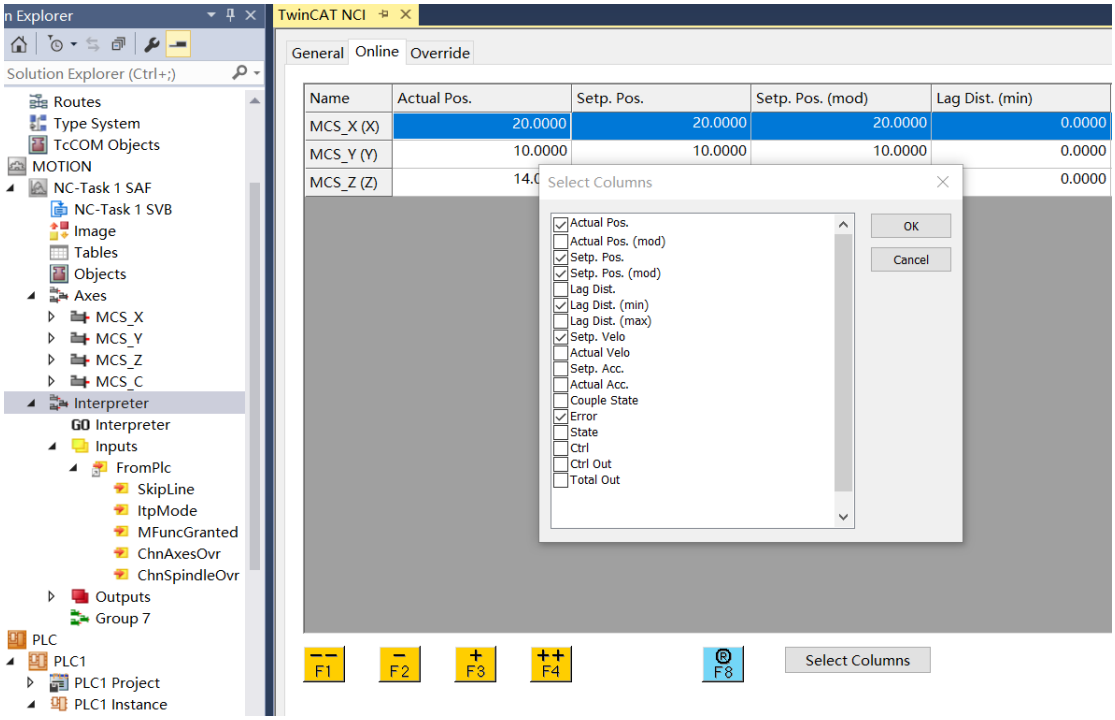
N80 ...

该指令需要和 ItpGoAheadEx 功能块配合使用，ItpGoAheadEx 功能块如下。

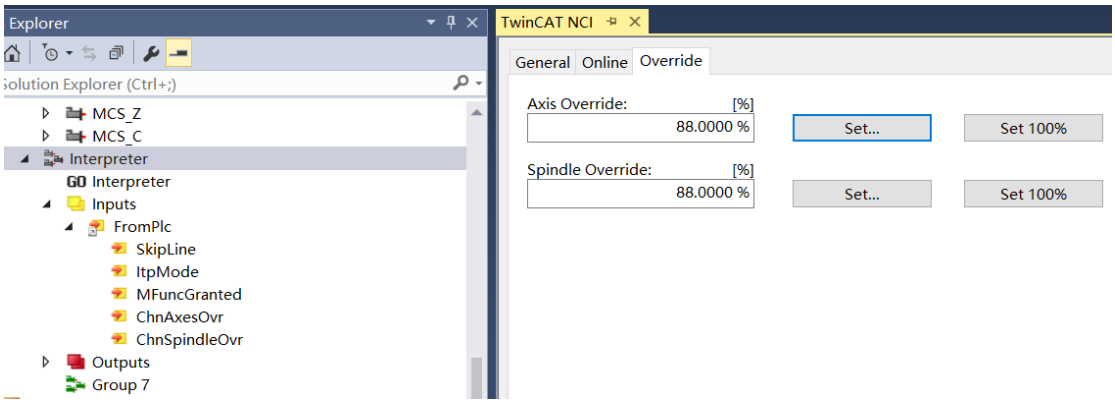
1.6 Interpreter 编译器参数

1.6.1 “Online”选项卡

如果要添加或者删除新的参数可以在 Online 界面添加。如下图 Interpreter-Online 选项卡既可以做单轴的点动操作，也可以添加轴的变量进行观测。添加的轴变量将显示在轴状态栏。



1.6.2 “Override”设置

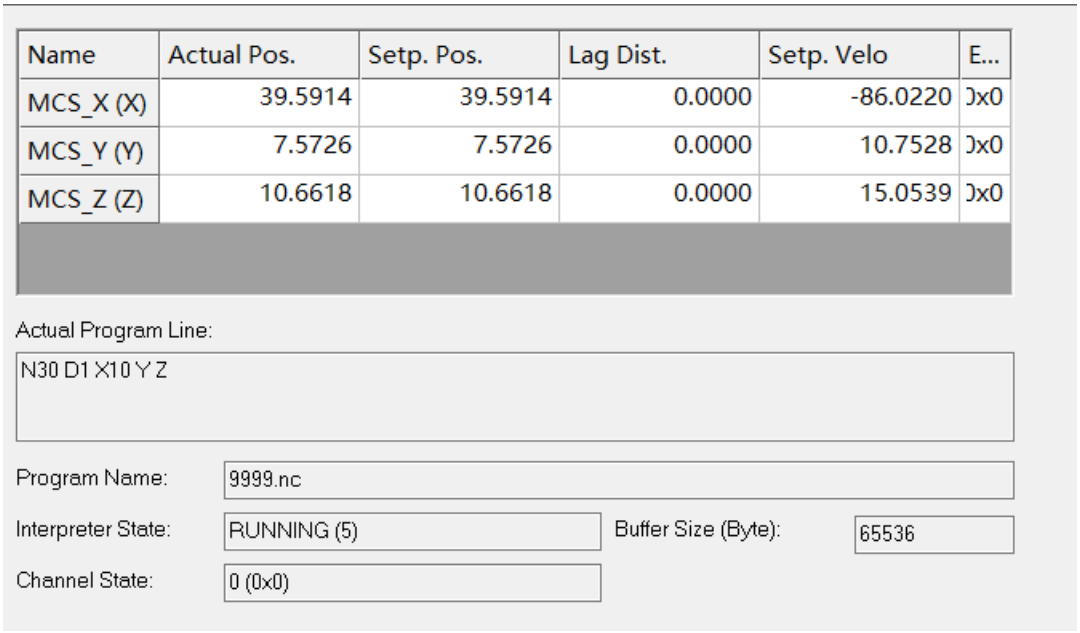


Axis Ovrreide 用来在调试时设置 NCI 的速度倍率，如果在 PLC 中修改需要通过 ItpSetOverridePercent 功能块，详见 1.4.9 节

Spindle Override 是主轴的速度倍率。与变量 ChnSplindleOvr 对应。

1.6.3 状态栏

可以在轴转态显示框观察各轴的位置、速度等参数。



1）监测变量的添加

见 1.6.1 节讲解

2）Actual Program Line

指 G 代码的当前执行指令行。

3）interpreter State

插补通道状态：参考 1.4.4

4）Channal State

指示 Channal 的报错代码，当 G 代码在下载或者运行过程中报错时，报错代码传给 Chananl 状态。在不报错的时候 Channal State 显示 0。

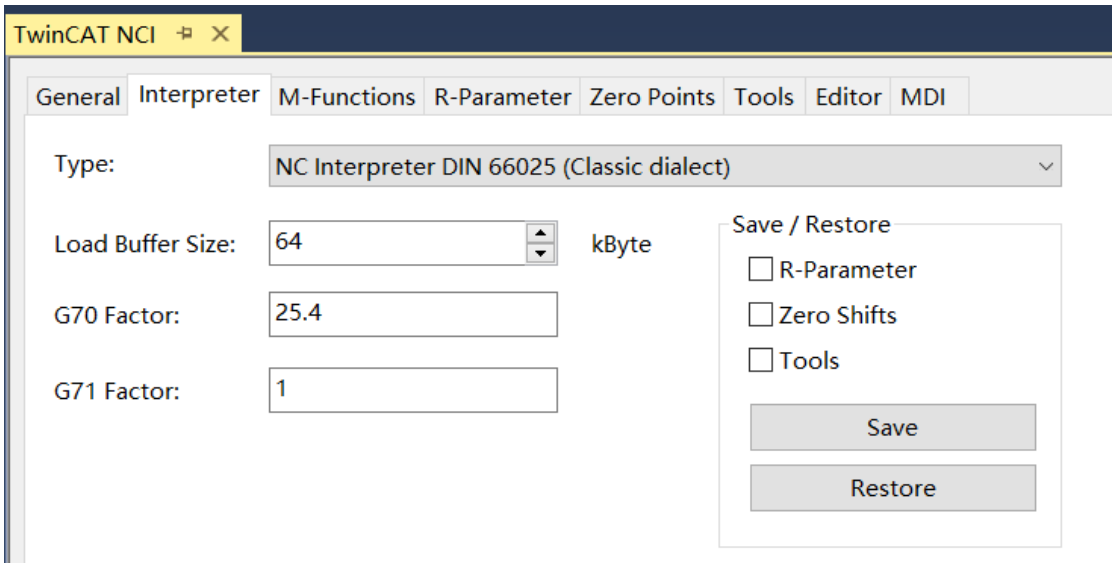
查找报错原因，可使用以下链接：

https://infosys.beckhoff.com/content/1033/tf5100_tc3_nc_i/135107990342671755.html?id=4931302329909152942

5）Loading buffer

下载缓存大小，在“Interpreter”选项卡可以进行调整。

1.6.4 “Interpreter”选项卡



1）Type 类型

NC Interpreter DIN 66025（GST）采用 GST 语言编程，GST 语言是融合了 G 代码（DIN66025）和 ST 语言（结构化文本，PLC 语言）的混合编程，详见本书第四章。

NC Interpreter DIN 66025 （Classic dialect）是采用 DIN66025 规范的 G 代码编程 ，在旧版本中也称为（Simense dialect）。

None 如果选择 PlcInterpolation 库，在 PLC 实现插补，这里选择“None”。

2）Loading Buffer Size

编译器的下载代码容量可以在这里设置。这里设置的存储容量需要大于 G 代码文件的大小。最大所允许的存储容下 4MB。如果存储容量被修改，需要重启 TwinCAT。

3) G70/G71 因数

G 代码程序中默认单位是毫米（G71），G70 用于切换到其他单位，通常为英寸。

如果 G70 用于英寸，由于 1mm=1/25.4 in，所以这里分别填入 25.4 和 1。所以这里用于设置 G70 和 G71 的换算关系。

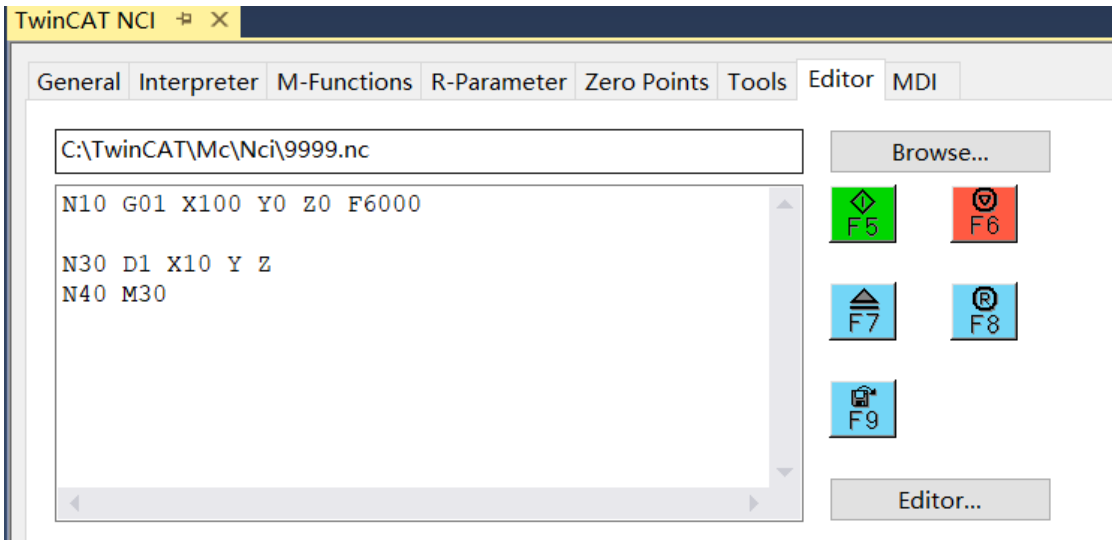
4） Save/Restore

可以选择 R 参数 零点偏移值和工具补偿值进行存储。

点击“save”按钮可以在程序运行过程中把打勾的参数的当前值存储到“TwinCAT\CNC”。

点击“Restore”按钮下载存储的数据。这个功能只用于调试的目的。

1.6.5 “Edit”选项卡



1）Browse 选择 G 代码文件，通常调试采用默认路径下 C:\TwinCAT\Mc\Nci\9999.nc 文件

2）在路径下的空白文本框内编辑 G 代码。

3）使用右侧的按钮运行和复位，按钮含义如下

F5　　执行 G 代码

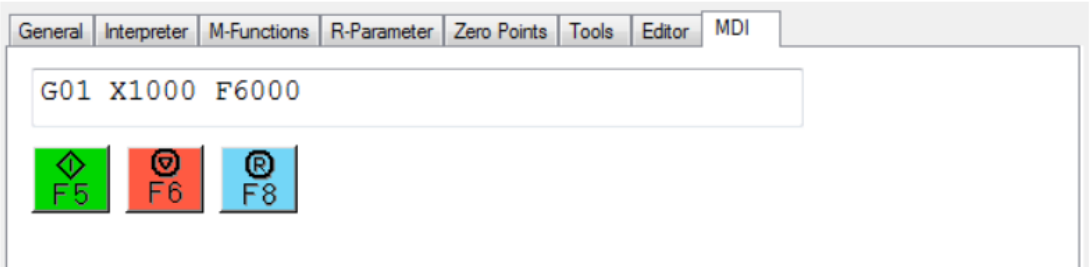
F6　　停止 G 代码

F7　　加载 G 代码

F8　　复位错误

F9　　保存 G 代码至文件

1.6.6 “MDI”选项卡

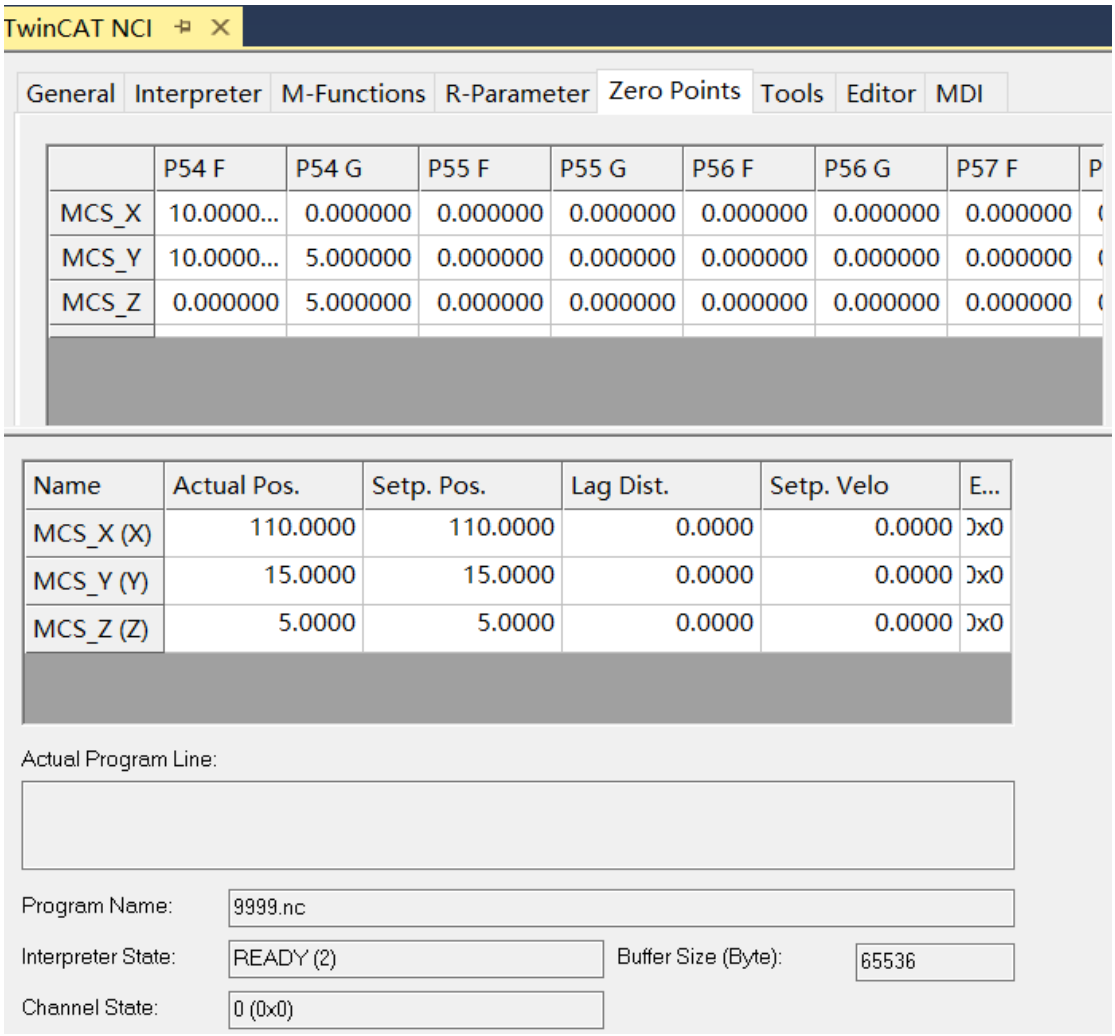


MDI 是“Manual Data Interface”的缩写，可以用于单行 G 代码的调试，既插补的手动操作。

1.7 零点偏移

在加工工件时有时候需要以初始下刀点作为坐标原点，这时候需要用到零点偏移。NCI 指令 G54、G55、G56、G57、G58、G59 都可以作为坐标偏移切换。G53 或者其他零点偏移编号取消当前零点偏移。

1.7.1 “Zero Points”预先设置偏移值



例：N10 G01 X100 Y0 Z0 F6000
N20 G54 (activates adjustable zero offset shift)
N30 G01 X Y Z
N40 M30

本例中通过 Zero Point 指定零点偏移值，在 G 代码程序中通过 G54 激活。
其中 X 运行到 200+X 方向偏移值、Y 运行到 Y 方向偏移值、Z 轴没有运行。最终显示 X 轴当前位置为 110、Y 轴位置为 15、Z 轴位置为 5。其中偏移量是 P54G+P54F。

1.7.2 通过功能块设置偏移值

ItpWriteZeroShiftEx

STRUCT ZeroShiftDesc

Name	Type
fShiftX	LREAL
fShiftY	LREAL
fShiftZ	LREAL

这里 nZsNo 是零点偏移的通道号，G54~G57 有效；

例：N10 G01 X100 Y0 Z0 F6000

N20 G54 (activates adjustable zero offset shift)

N30 G01 X Y Z

N40 M30

本例中通过功能块指定零点偏移值，在 G 代码程序中通过 G54 激活。最终 XYZ 轴的位置显示值和偏移前没有发生变化，但电机轴发生了运行。

1.7.3 在 G 代码中参数化零点偏移

#set paramZeroShift(G<n>; <value x>; <value y>; <value z>)#

例：N10 G01 X100 Y0 Z0 F6000

N20 R12=200

N30 #set paramZeroShift(G54; 100.0; R12; -20)#

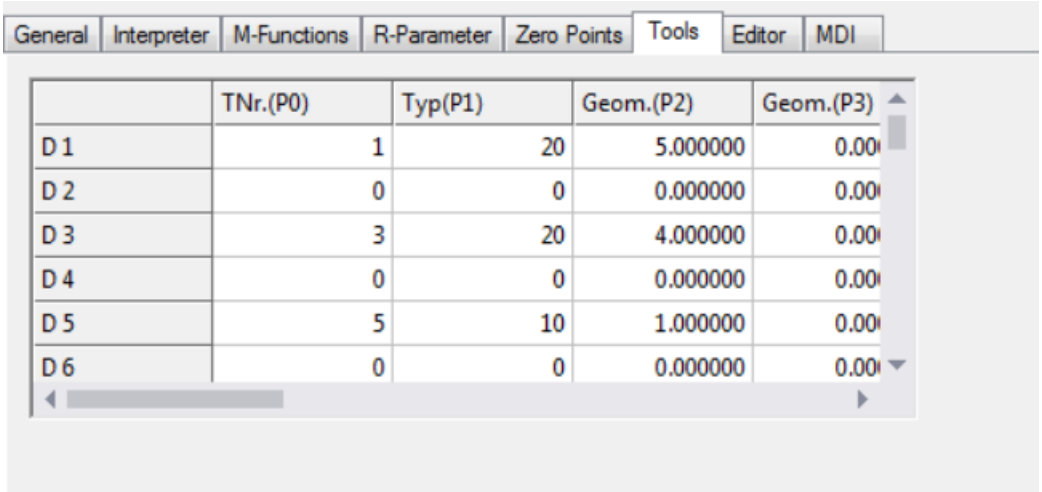
N40 G54 (activates zero offest shift)

N50 G01 X200 Y Z

本例中在 G 代码程序中指定偏移值，通过 G54 激活。

1.8 刀具补偿

1.8.1 刀具数据



	TNr.(P0)	Typ(P1)	Geom.(P2)	Geom.(P3)
D 1	1	20	5.000000	0.00
D 2	0	0	0.000000	0.00
D 3	3	20	4.000000	0.00
D 4	0	0	0.000000	0.00
D 5	5	10	1.000000	0.00
D 6	0	0	0.000000	0.00

每个 NCI Group 可以存储 255 个刀具（D1…D255），这些参数可以直接在“Tool”选项卡中设置，以 ASCII 码的方式保存在 TwinCAT\CNC 目录下。当 TwincAT 启动时，这些参数自动下载到 NCI Group。

目前 TwinCAT NCI 支持两种刀具类型 1.Drill 钻头 2：Shaft Cutters 铣刀。

参数	**钻头参数**	**铣刀参数**
TNr.(P0)	工具编号	工具编号
Tpy(P1)	工具类型，钻头是 10。	工具类型，铣刀是 20。
Geom.(P2)	几何含义：长度 描述钻头的长度	几何含义：长度 描述铣刀的长度
Geom.(P4)		几何含义：半径
Verschl.(P5)	磨损：长度，描述钻头的磨损长度	磨损：长度 描述铣刀的磨损长度
Verschl.(P7)		磨损：半径 描述铣刀的磨损半径
P8	X 方向偏移量	X 方向偏移量
P9	Y 方向偏移量	Y 方向偏移量
P10	Z 方向偏移量	Z 方向偏移量

1.8.2 读写刀具数据

1）从“Tool”选项卡中设置

如上文所说，可以直接在“Tool”选项卡中写入刀具数据。

2）PLC 功能块写入刀具数据

nDNo 是刀具编号，（0…255）.

sToolDesc 是 ToolDesc 结构体,其定义如下:

```
TYPE ToolDesc:
STRUCT
   nToolNumber     : UDINT; (*valid range from 0 .. 65535*)
   nToolType       : UDINT;
   fParam          : ARRAY [2..15] OF LREAL;
END_STRUCT
END_TYPE
```

3）G 代码写入刀具数据

语法规则是：#set ToolParam(<line>; <column>;<value>)#

<line> 描述刀具编号（1…255）

<column> 刀具的参数(P0…P15)

<value> 参数值，可以用 R 变量表示。

如下面的 G 代码例子:

N10 G0 X0 Y0 Z0

N20 G01 X100 F60000

N30 R1=10 R2=4 R3=20.3

N40 #set ToolParam(10; 0; 5)# #set ToolParam(10;1;20)#

N50 #set ToolParam(R1; R2; R3)#

N60 G41 X200 Y D10

…

本例中设置刀具 10 的工具编号为 5、类型为铣刀、半径为 R3。

4）G 代码读取刀具数据

语法规则是：#get ToolParam(<line>; <column>;<R-Param>)#

例子:

N10 G0 X0 Y0 Z0

N20 G01 X100 F60000

N30 R1=10 R2=4

N40 #get ToolParam(10; 0; R5)# #getToolParam(10;1;R20)#

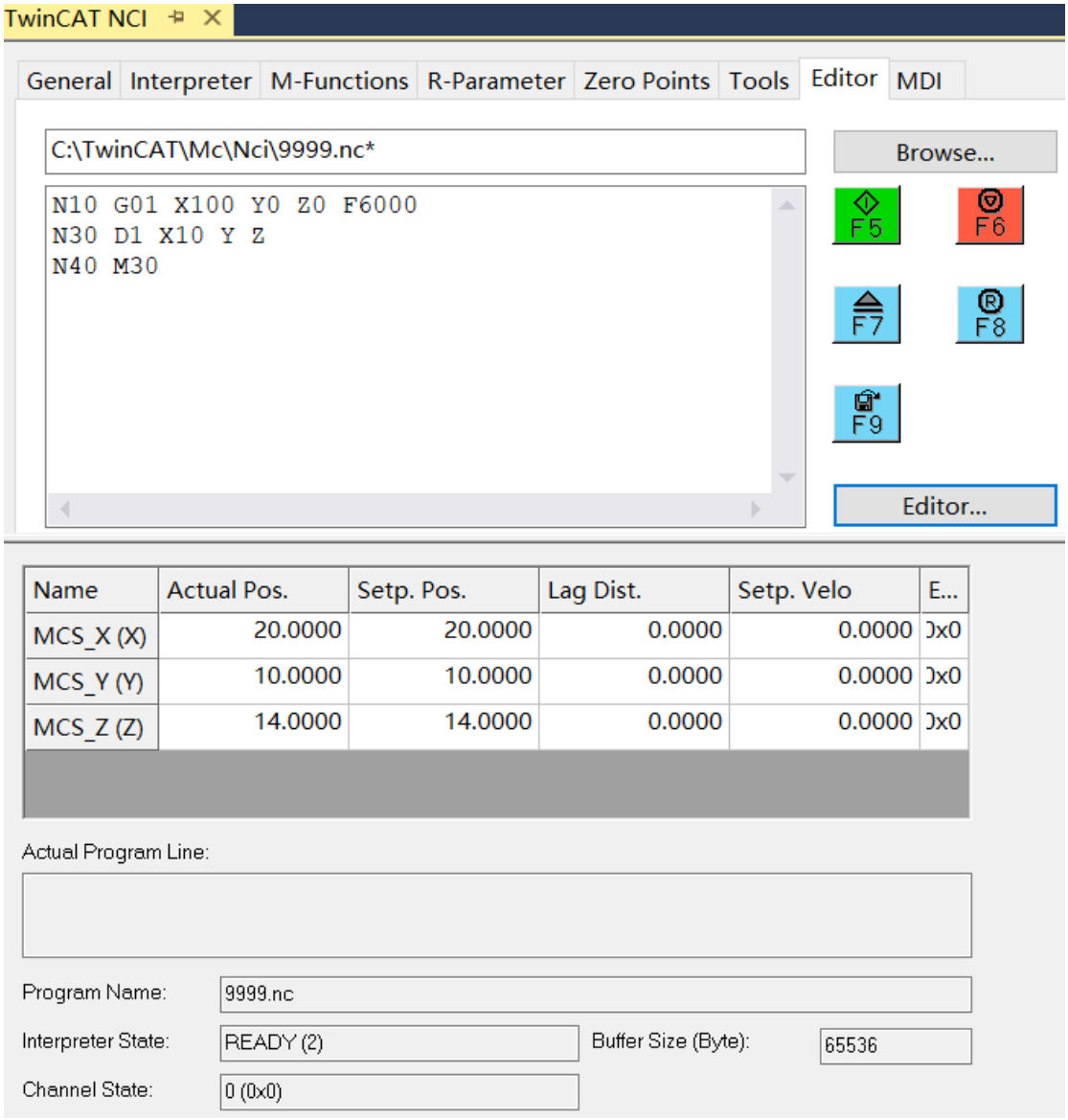
N50 #get ToolParam(R1; R2; R3)#

N60 G41 X200 Y D10

…

本例中读取刀具 10 的工具编号、类型和半径到 R5、R20、R3。

1.8.3 刀具的调用



例：

① 首先在“Tool”中配置 D1 ， P0=1（编号）；P1=10（类型钻头）；P2=9（钻头长度）；P5=-6（钻头磨损）；P8=P9=P10=10（XYZ 方向偏移）；

② 然后在“Edit”中运行以下 G 代码，如上图。

N10 G01 X100 Y0 Z0 F6000

N30 D1 X10 Y Z

N40 M30

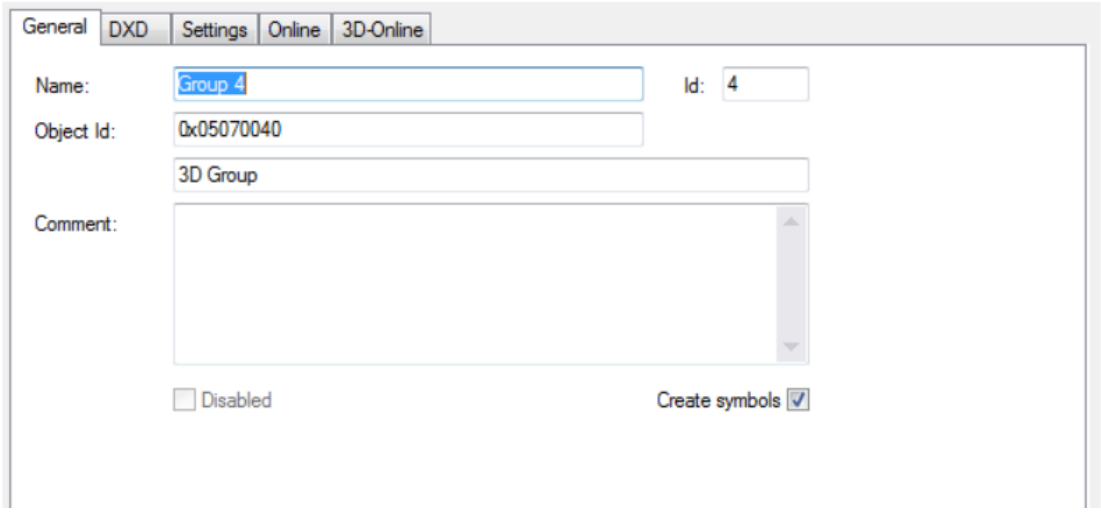
可以看到：X 目标位置为 20，（10+10=20）

Y 目标位置为 10，（10=10）

Z 目标位置为 14，（14=10-6+10）

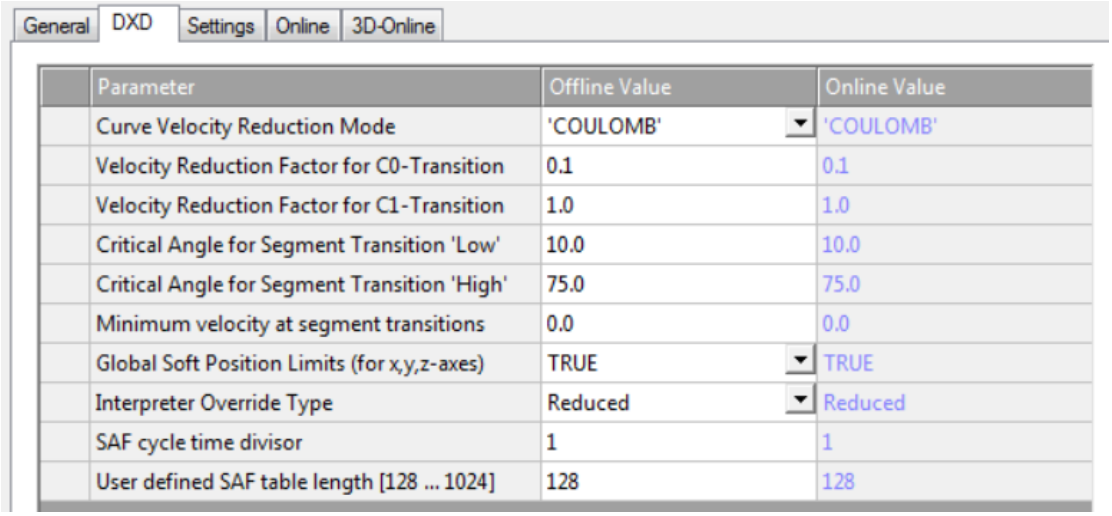
1.9 Group 通道参数

1.9.1 General 选项卡



Id ： 这里指 GroupID 如上图 Id 为 4。
Objec id：通常用于 ADS 通讯。
Create symbols 用于激活 ADS 通讯时对变量的访问。

1.9.2 DXD 选项卡



在介绍 NCI 的通道参数之前，我们先来介绍一些名词解释。
曲线段转换的分类：**C0、C1、C2 段曲线**
一般来说，从一段曲线到下一段的过渡不是无限平滑的。因此，为了避免过渡颠簸，有必要减小过渡点处的速度。为此，对过渡段进行几何分类，分三种确定有效过渡速度 V_link。

曲线从一个线段 S_in 到另一个线段 S_out 的过渡按几何术语分类称为类型 Ck，其中 k 是一个自然数(包括 0)。

定义：如果每个曲线段有 k 个连续的弧长微分，并且在过渡点处的 k 阶导数相同，则称该过渡为 Ck 过渡（k 大于等于 0)。

C0，直线转直线　　C1，直线转圆弧　　C2，加速度平滑过渡

C0 过渡：在过渡点上有一个角度。比如直线转直线。

C0 过渡的类型和方法后文会进一步深入讲解。

C1 过渡：看起来很平滑，但在动态方面并不平滑。一个例子是在体育场的直线-半圆过渡:在过渡点有一个台阶变化的加速度。过渡方法:

首先，将 V_link 设置为两个段目标速度中较低的一个:V_link = min(V_in,V_out)。

在速度 V_link 条件下，根据几何类型 G_in 和 G_out，以及要连接的线段的平面选择 G_in 和 G_out，计算出线段过渡中几何引起的加速度跃变的绝对步长变化。

如果这大于 C1 乘以几何图形和平面允许的路径加速度/(绝对)减速度 AccPathReduced，则速度 V_link 减小，直到产生的加速度步长变化等于 AccPathReduced。

如果这个值小于 V_min，那么 V_min 优先。

注意，当改变动态参数时，几何图形和平面的允许路径加速度以及由此产生的还原反应会自动改变。

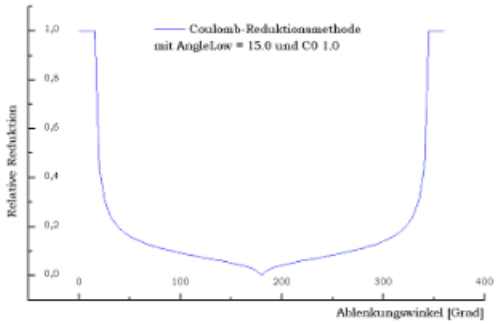
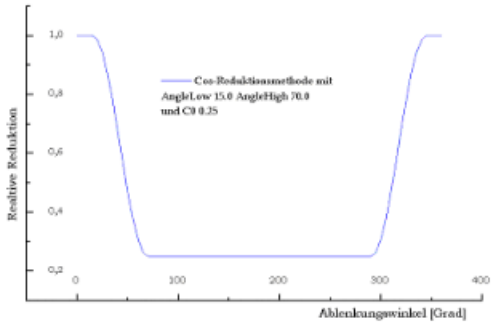
C2 过渡：(包括 Ck， k > 2)是动态平滑的（加加速度限制)。

过渡方法: 在 C2 过渡中, V_link 被设置为两个段速度集的最小值:V_link = min(V_in,V_out)。没有进一步的降低速度。

1） Curve Velocity Reduction Mode

曲线速度下降方式对 C0 过渡有效

方式	描述
Coulomb 库伦	库仑下降方式是一个类似库仑散射的动态过程。 过渡点的偏转角度φ是 S1 段的末尾的切线和 S2 段的开始的切线的夹角。 在库仑散射中，速度设置为在无穷远处的速度， $Vk \propto \tan(0.5(\pi-\varphi))^{\frac{1}{2}}$ ， 然后通过 C0 因子降低. Vk ← C0 Vk. 在一个反向运动中(φ= 180)，速度下降到 Vk = C0。在小偏差角度时发生剧烈下降的情况下,有一个完全下降角 φlow[0，180]生效。为了避免速度下降，设置 φlow = 180。完全下降时(降低到 φ= 0)，设置 C0 = 0.0 和 φlow = 0。

	
Cosine 余弦	余弦下降方式是一个纯粹的几何学过程 涉及到： C0 因数∈ [0,1], 角度 φlow ∈ [0,180], 角度 φlow ∈ [0,180],以及 φlow < φhigh 下降方案： ● φ < φlow: 不减速: Vk←Vk, ● φhigh < φ: 通过 C0 因数下降: Vk← C0 Vk ● φlow < φ <φhigh:在第 1 和第 2 种情况下部分下降持续插值，与余弦函数（0，π/2）的值成比例。 对于完全下降（下降到 φ = 0），设置 C0=0.0 和 φlow = 0 以及 φhigh（非常小，但不等于 0） 
VeloJump 速度跳变	VeloJump 方式是每个轴按允许的速度以周期步长进行降低速度。 这是一个确定过渡段速度在 C0 过渡时的几何过程。该过程根据需要降低的路径速度，使速度的步长变化不超过指定的极限值。计算公式如下：VJump（i）=速度跳变系数*周期时间*min(加速度，减速度)，其中 i 是轴号。（速度跳变系数由于 PTP 轴 parameter-NCI-Velo Jump Factor 决定，参考 1.10.2） 这种过渡将确保速度降低到 V_Link（V_link = min(V_in，V_out),），并由此确定每个轴的最大速度下降值 VJump（i），如果 V_Link<V_min，那么 V_Link=V_min； 注意：当改变动态参数时，轴速度所允许的最大阶跃变换因数会同时自动改变。 通常用于两个不同的机械轴之间的插补过渡。
DEVIATIONANGLE 转折角方式	转折角方式的减少速度取决于偏转角度 φ(输入曲线 S_in 在转折角的切线 T_in 与输出曲线 S_out 在转折角的切线 T_out 的夹角。

	注意：当改变动态参数时，下降因数不会同时自动改变。 这种下降方式**暂未发布**。 通常用于两个关联轴之间的过渡，比如：X 轴耦合到 Y 轴。

2）Velocity reduction factor C0 transition

C0 过渡的速度下降因数

C0 过渡的下降因数，效果取决于下降方式，见上表。

C0 ∈ [0.0, 1]

3）Velocity reduction factor C1 transition

C1 过渡的速度下降因数

首先，将 V_link 设为两段目标速度中较低的一段。V_link = min (V_in ， V_out)。在速度 V_link 条件下，根据几何类型 G_in 和 G_out，以及要连接的线段的平面选择 G_in 和 G_out，计算出线段过渡中几何形状引起的加速度跃变的绝对位置变化。如果这大于 C1 乘以几何图形和平面允许的路径加速度/(绝对)减速度(AccPathReduced)，则速度 V_link 减小，直到产生的加速度变化等于 AccPathReduced。如果这个值小于 V_min，那么采用 V_min。注意，当改变加速度参数时，几何图形和平面的允许路径加速度反过来影响下降变换。

C1 转换的系数:C1≥0.0

4） Critical angle, segment transition 'low'

临界角度，曲线过渡“Low”

参考曲线速度下降方式的 Coulomb 方式和 Cosine 方式。

5） Critical angle, segment transition 'High'

临界角度，曲线过渡“high”

参考曲线速度下降方式的 Coulomb 方式和 Cosine 方式。

6） Minimum velocity at segment transitions

曲线过渡时的最小速度

每个 NCI 组的最小路径速度 V_min≥0.0。通常实际速度应该总是超过这个值，但如果程序指定也会小于这个设置值，比如：程序在线段转换时的停止、路径结束或者 Override 减小导致速度低于最小值。还有一个系统性例外是运动反转。最小速度可以设置为一个新的值 V_min≥0.0 在程序运行的任何时候。单位是毫米/秒。

7） Global soft position limits (for x,y,z-axes)

全局软限位

当 NC 中轴参数开启软限位时，如果 NCI 的轴实际位置超出轴参数的软限位值，会导致速度立刻降低到零。为了避免轴的速度立刻降低到零，这里设置了全局软限位监视功能。

True：使能全局软限位开启使能全局软限位功能时，必须同时开启轴参数中的 software limit 功能。此功能可以监视在标准的几何曲线轨迹运动并列运行，如直线、圆弧、螺旋等。若轴停在限位之外的区域，可用直线命令移回。

False：不开启全局软限位

如果此处为 false，且开启了轴参数中的 software limit 功能，一旦超出限位，会导致插补通道报错。

8） Interpreter override type

编译器倍率类型

（1）Reduced

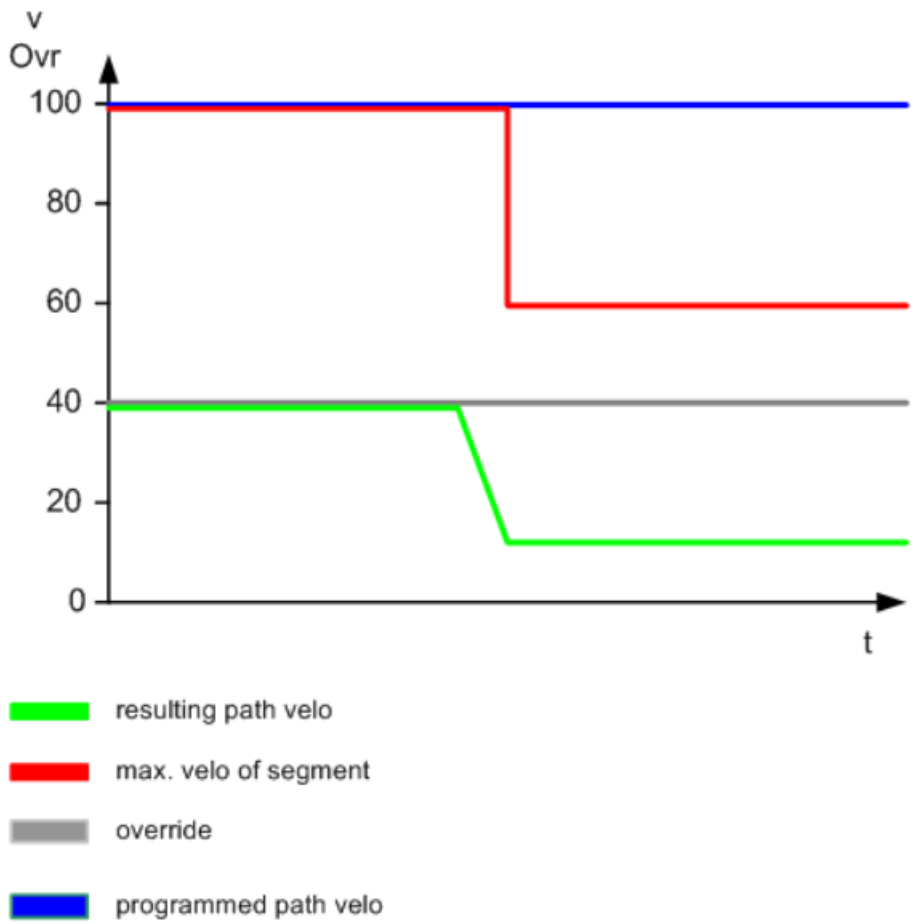
选项“减少”——基于减速过程的(默认使用)

由于相关动态参数(如：制动距离、加速度等)，程序段中的速度(蓝线)不可能每段都能完全达到 。基于这个因素，为每个几何段的运动部分计算降低速度（红线）。在标准情况下，

Override 是参照这段速度。

这种速比类型的优势是，如果速比值很小，设备运作的速度呈线性降低，同时绝大部分应用都是这样设置的。

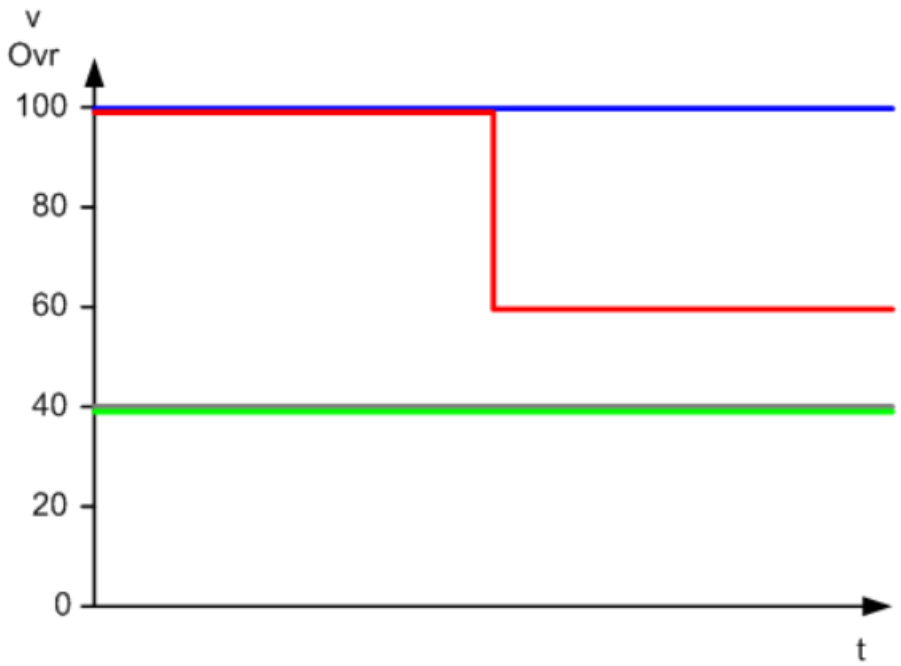
v_res = v_max * Override



（2）Original

选项“原始”-基于程序的路径速度

此时速比值基于用户编程设定的。



（3）Reduced【0···>100%】

基于内部降低速度可以选择指定数值且大于 100%，这种速比类型与“Reduce”模式相似。这种速比类型比在 G 代码程序中编程实现要更快，比如说也不限于 120%。路径最大速度被 Group 里的 G0 最大速度和动态特性限制。在此模式下，如果需要限制特殊值为 120%，可以在 PLC 中进行设置。

10） SAF cycle time divisor

SAF 周期细分

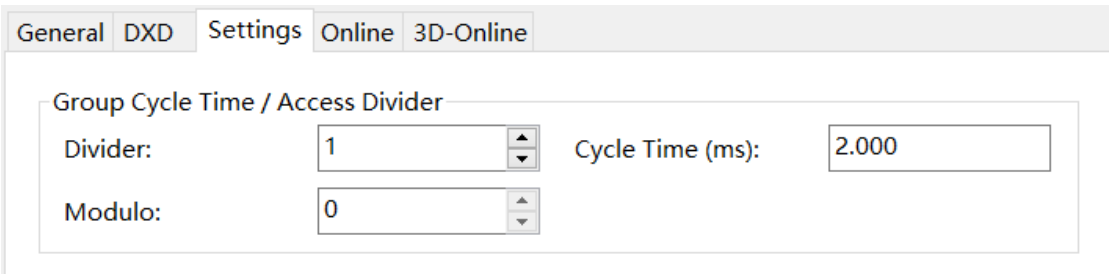
循环时间确保 SAF 中的设定值不是用 SAF 循环时间计算的，而是用时间除以这里指定的值来计算的。对于动态运动来说，需要将参数设置为大于 1 的值，以便于可以最小化离散化误差。增加 SAF 周期细分数可以频繁调用 set 值发生器。

11） User-defined SAF table length

用户定义 SAF 表格长度

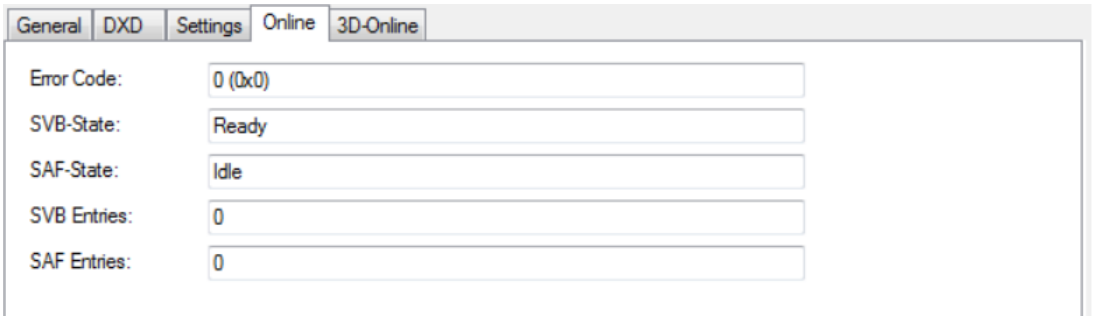
该参数定义 SAF 表的大小，从而定义缓存的 SAF 条目的最大数量。如果 G 代码程序涉及到多个线段运动，增大 SAF table 长度可以有助于避免在过渡区域无法预测的速度降低问题。

1.9.3 Setting 选项卡



在“Setting”标签下，可以设置 Group 中的轴的周期时间。比如说 NC 轴的周期时间是 2ms,但是如果 Group 中轴的周期是 4ms，这时候可以设置 Divider 为 2，则 Group 中的轴的周期时间变为 4ms。这里设置的循环时间是 SAF 任务的循环时间的倍数。

1.9.4 Online 选项卡



1）Error Code

这里显示 NCI 通道的错误代码。这里的错误代码和 NCI 编译器 “Editor”选项卡的通道状态相同。

2）SVB state

显示当前 SVB 状态

3）SAF state

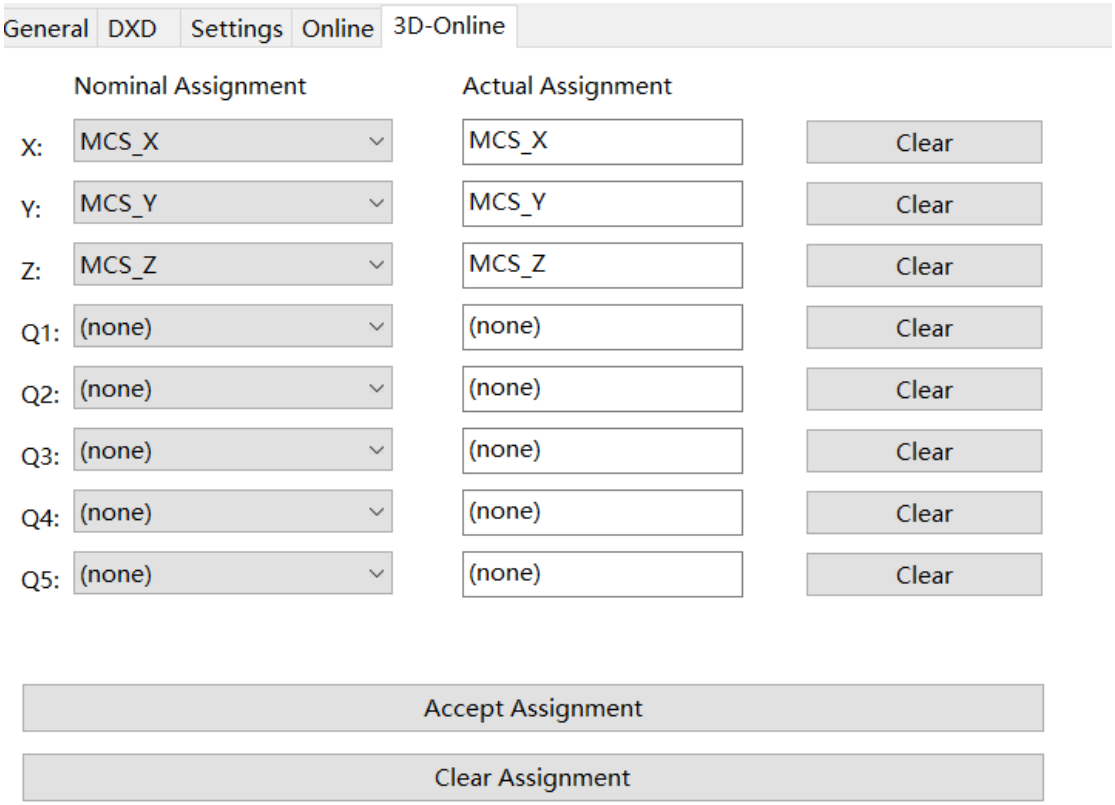
显示当前 SAF 状态

1.9.5 “3D-Online”选项卡

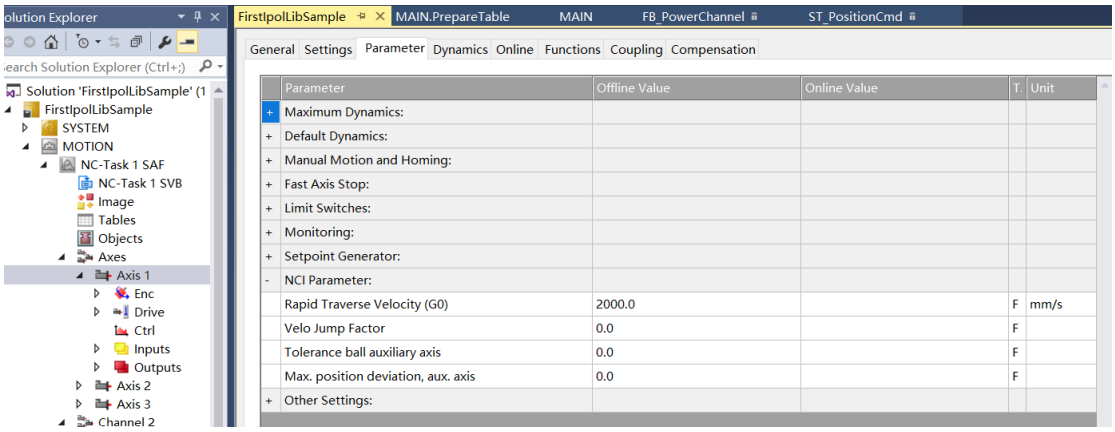
3D-Online 页面用来在调试时把 NCPTP 的轴添加到 NCI 插补通道中。与 1.4.2 节中介绍的 CfgBulidExt3DGroup 功能块相对应。

Accept Assignment 接受配置

Clear Assignment 清除配置



1.10 NC 轴中的 NCI 参数



1.10.1 Rapid Traverse Velocity（G0）

当使用 G0 指令以 PTP 模式运行时，既不考虑插补配合时的轴的速度。通常用于快速定位。

1.10.2 Velo Jump Factor

Velo Jump Factor 速度跳变因数，是轴在 VeloJump 过渡方式时的 C0 过渡的系数。关于 C0 过渡详见 1.9.2。

VeloJump 过渡方式的计算方式如下：

通常，v_link = min(v_in, v_out)。

对于轴[i]，速度允许的绝对阶跃变化为 v_jump[i] = C0[i] * min(A+[i]， -A-[i]) * T，其中 C0[i]为减速因子，A+[i]， A-[i]为轴[i]的加减速极限，T 为周期时间。

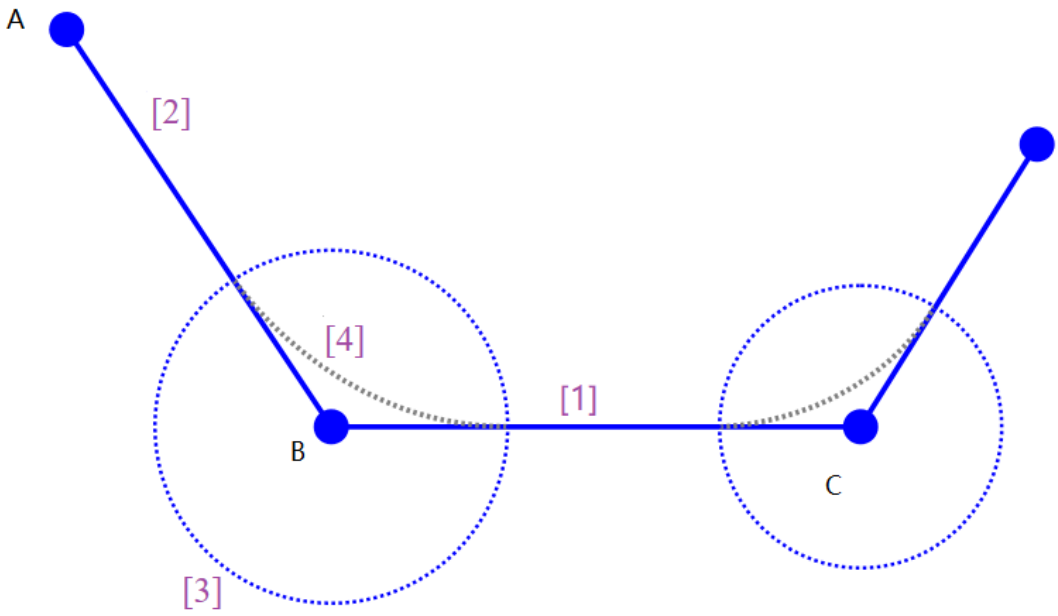
VeloJump 跳变方式保证了路径过渡段 v_link 处的路径速度减小，其中轴[i]的轴设定值速度的绝对步长变化最多为 v_jump[i]。

另外，如果 v_link 小于 v_min，则将 v_link 设置为 v_min。

1.10.3 Tolerance ball auxiliary axis

容差球(Tolerance ball)放置在每个线段过渡的转折点的周围，容差球用于消除曲线过渡段的动态性能不稳定，既加速度或者加加速度突变。为了实现平滑过渡，在容差球内轨迹可能偏离其预先设定的几何形状，但不会超出容差球。容差球的半径(参数化)是由用户预先设定的，并应用于所有的线段过渡，因此在线段过渡中没有精确的定位或停止。容差球的半径可以自动自适应复位，防止多个小线段情况下的容差球重叠。在容差球内 Override 无效，且加速度为 0。

如下图，从 A 点经 B 点到 C 点的曲线过渡过程中，线段 2 经曲线 4 过渡到曲线 1，其发生的形变在容差球 3 的范围内。



Tolerance ball auxiliary axis 是为每个 NCI 的辅助轴设定的一个容差球。当进入该范围时，辅助轴的速度被不断修正，在范围出口达到设定的速度以避免速度的阶跃变化。

1.10.4 Max position deviation aux axis

“辅助轴的最大允许位置误差”只有在容差球内需要改变路径几何形状时才有效。只要产生的位置误差不超过设定值，就可以在较小的偏差范围内保持较高的路径速度。这可以保持辅助轴的速度恒定，并计算位置偏差。如果偏差小于最大位置偏差，则在这一段过渡中保持速度恒定，并在下一段过渡中补偿由此产生的位置偏差。当位置偏差超过最大偏差时，将采取措施减小偏差。

1.11 辅助轴的编程和速度

1.11.1 辅助轴的编程

辅助轴也可以通过 CfgBulidExt3DGroup 功能块进行配置。在 G 代码程序中使用 Q1…Q5 编程。

例 1：(start position X=Y=Z=Q1=Q2=0）

N10 G01 X100 Q1=100 F6000

当辅助轴的运动指令伴随有插补轴的运动时，如例 1。辅助轴与插补轴同时启动、同时停止，辅助轴的速度由插补轴的运动距离和 F 速度参数来计算。

例 2：（start position X=Y=Z=Q1=Q2=0）

N10 G01 X100 F6000

N20 Q1=100 Q2=200 F3000

…

当辅助轴的运动没有伴随插补轴的运动时，如例 2，由于插补轴路径长度为零，所以不存在与路径的配合，这时辅助轴的速度是根据行程距离最大的辅助轴以及 F 参数来计算的。而且所有的辅助轴都是同时开始，同时到达终点。

1.11.2 辅助轴的速度

辅助轴的速度限制因数主要在 NC 轴中的 NCI 参数中设置，详见 1.10 节。

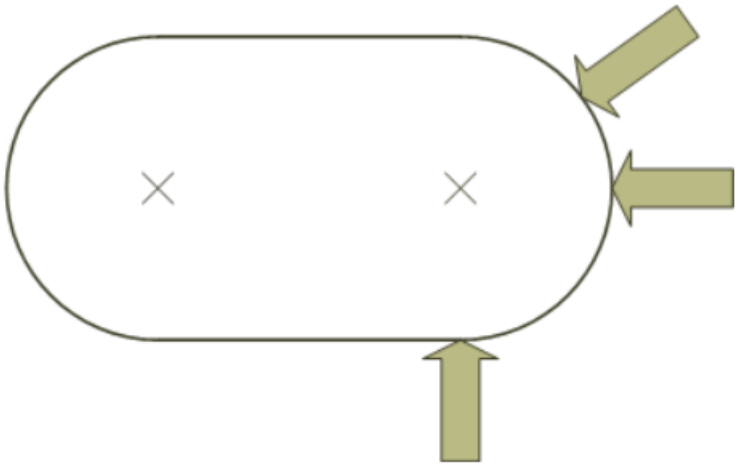
辅助轴对路径速度的间接影响：

虽然辅助轴的运行速度由辅助轴的运行距离和 XYZ 插补轴的运行时间决定，但是如果这个速度大于这个辅助轴允许的最大速度，则插补轴的路径速度减小，直到达到辅助轴最高速度限制。换句话说，超过辅助轴的速度极限也会对路径速度产生间接的影响。

曲线过渡时的路径速度：

下面通过一个例子来解释路径速度的降低过程。我们以体育场的轮廓为例来做解释。假设工艺要求是使刀具的方向始终垂直于路径切线。如果站在球场轮廓上看，刀具的方向保持不变，即到具没有转动始终与球场轮廓的切线垂直。然而如果从世界坐标系的方向看，刀具方向则必须在圆内不断地改变。假设在直线和圆之间的过渡的路径速度没有降为零，那么对于旋转轴(不是路径轴)速度会产生阶跃变化。

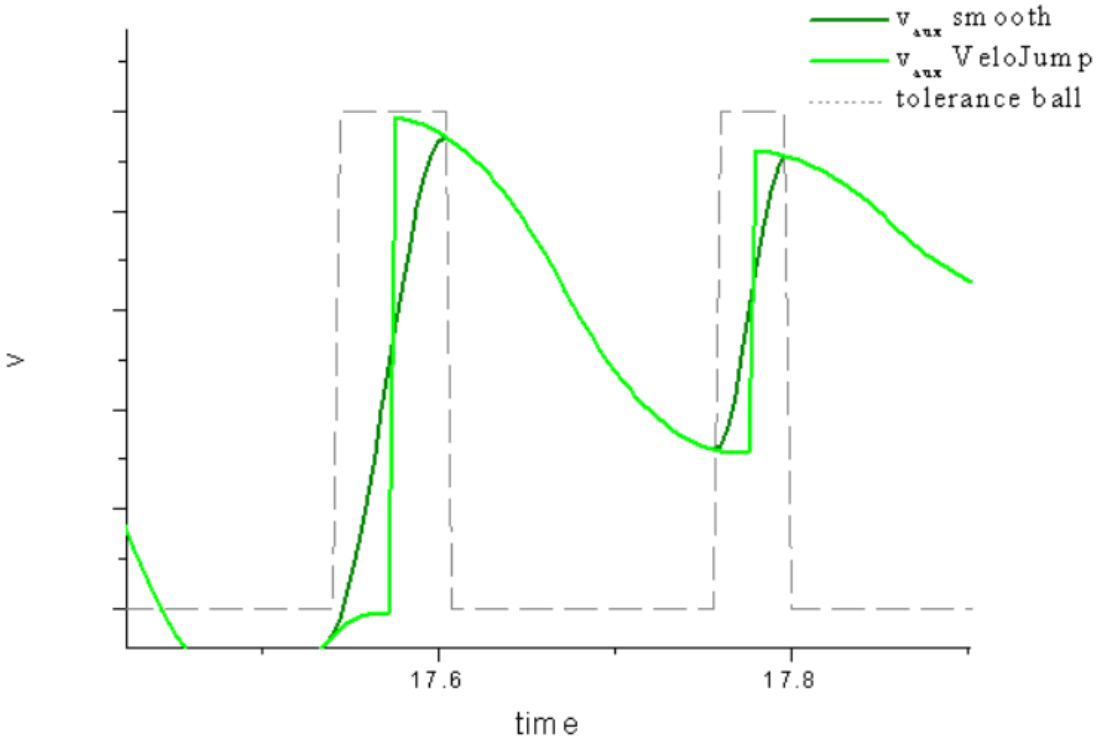
每个轴的 NC 参数中都可以设置全局轴参数“VeloJumpFactor”。在本书上一章节 NC 轴中 NCI 参数中有更深入的讲解。



曲线过渡时的速度平滑：

如上所述，在曲线过渡过程中，会发生速度的阶跃变化，这些变换的大小会受到 VeloJump 参数的影响。还可以为每个辅助轴指定一个容差球（tolerance ball）。当进入该容差球时，辅助轴的速度被不断修正，在范围出口达到设定的速度。避免了速度的阶跃变化。

如果指定的容差球半径大于路径的 1/3，则其半径自动限制为该值。



减少容差球对速度跳跃参数（VeloJumpFactor）的影响

如果根据给定的几何形状来减小容差球的尺寸，则自动调整 VeloJumpFactor 参数以适应曲线过渡，降低 VeloJumpFactor 会使过渡过程中的路径速度降低得更多。

减少容差球对辅助轴的最大位置偏差（Max position deviation aux axis）的影响

辅助轴的最大位置偏差只有在因位置偏差产生的几何形状变化达到容差球限制时才有效。这样做的目的是，只要产生的位置偏差不超过设定值，就可以在较小的偏差范围内保持较高的路径速度。为此，保持辅助轴的速度恒定，并计算位置误差。如果偏差小于最大位置偏差，

则在这一段过渡中保持恒定速度，并在下一段过渡中补偿由此产生的位置误差。当位置偏差超过最大偏差时，采用 VeloJumpFactor 减小路径速度 以减少位置偏差。

1.12 平滑过渡曲线

通常，使用 G01 指令可以实现曲线段过渡时。在过渡时，如果路径速度没有降低到零值，折线轨迹的位置相对于它们的空间坐标不是稳定可微分的，因此导致了速度的不稳定。为了在实际应用中避免将路径速度降低到零值，可以通过对多边形函数的过渡进行平滑处理。

名称	执行	支持的线段转换	轴的加速度	路径最大偏离值	自适应偏离半径	G 代码名称
圆弧平滑 Circular smoothing	编译器	直线/直线	加速度阶跃变化(通过 C1 参数)	1/2 输入输出段	无	paramCircularSmoothing(…)
抛物线平滑 Parabolic smoothing 类型：2	NC 内核	直线/直线	加速度阶跃变化到恒定值(通过 C1 参数)	1/3 输入输出段	可选	paramVertexSmoothing(…)
四次曲线平滑 Biquadratic smoothing 类型：3	NC 内核	直线/直线	恒定加速度-在起始点和结束点的加速度为 0 -不需要中间点	1/3 输入输出段	可选	paramVertexSmoothing(…)
三阶贝塞尔曲线平滑 Bezier Curve of the 3rd Order 类型：4	NC 内核	所有	加速度阶跃变化到直线(通过 C1 参数)	1/3 输入输出段	可选，但对于直线转换有影响	paramVertexSmoothing(…)
五阶贝塞尔曲线平滑 Bezier Curve of the 5rd Order 类型：5	NC 内核	所有	恒定加速度-在起始点和结束点的加速度为 0 -不需要中间点	1/3 输入输出段	可选，但对于直线转换有影响	paramVertexSmoothing(…)
旧贝塞尔平滑过渡 Old Bezier Blending 类型：1	NC 内核	所有	恒定加速度-在起始点、结束点和对称中间点处的加速度为 0	1/4 输入输出段	无	paramSplineSmoothing(…) paramVertexSmoothing(…)

过渡的原则:

过渡影响转折开始前后的两个线段。最大偏离值的半径可以在 G 代码程序中随时改变。并可以通过设置半径为 0 再次关闭平滑多渡，如果不为 0 平滑过渡将一直保持激活状态，直到编译器重置或 TwinCAT 重新启动。

容差球（Tolerance Spheres）：

在每个过渡线段周围都放置一个容差球，在此过渡段内，为了平滑，路径可能偏离其预先设定的几何形状。容差球的半径(参数 Radius)是由用户在平滑过渡 G 代码中设置，意味着在段过渡中没有精确的定位或停止。容差球的半径自动自适应复位，防止小段情况下容差球重叠。

动态参数（Dynamic Parameters）

平滑过渡使系统获得更快的动态性能。用户可以通过 C2 velocity reduction 参数修改过渡段速度，过渡段速度=C2 x VeloLink，其中 VeloLink 是最大段过渡速度。

平滑过渡转换的通用特性

当进入过渡段容差球部分时，路径加速度为 0，在这一区域内的速度是恒定的。Override 倍率在容差范围内是不起作用的，即由 Override 引起的速度变化在容差球内无效，并在路径离开容差球后继续有效。

1.12.1 圆弧平滑过渡

借助于圆形平滑过渡，可以在两条直线之间自动插入一条圆弧。只需要对弧的半径进行编程即可

语法：#set paramCircularSmoothing(<radius>)#

例：N10 R57=4.5
#set paramCircularSmoothing(R57)#
...
#set paramCircularSmoothing(0)#
N1000 M02

1.12.2 抛物线平滑过渡

采用抛物线平滑过渡，在过渡段几何上插入抛物线。这确保了在允许的半径内稳定的速度转变。抛物线仅用于直线/直线过渡。

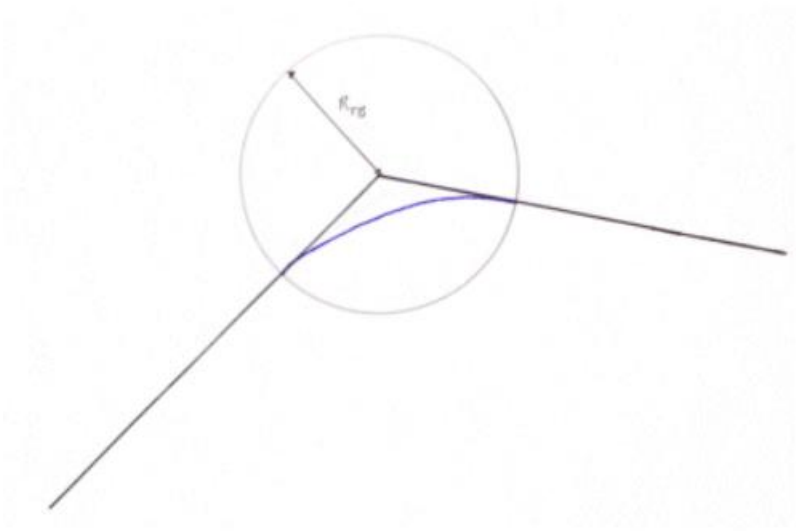
指令	# set paramVertexSmoothing(<type>;<subtype>; radius>)#
参数（type）	抛物线平滑：2
参数（subtype）	1: 常数路径最大偏离半径 Constant tolerance radius
	2: 交点到顶点的距离 Distance between intersection and vertex
	3: 自适应最大偏离半径 Adaptive tolerance radius
参数（radius）	最大偏离半径

例: # set paramVertexSmoothing(2;1; 30)#

1.12.3 平滑过渡的子类型 subtype

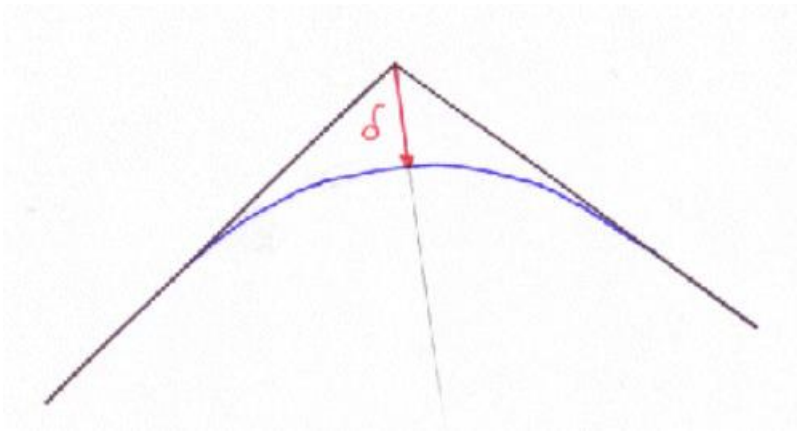
1）常数路径最大偏离半径 Constant tolerance radius

如果选择子类型 1，过渡曲线的最大偏离半径（RTB）为设置的恒定值，只有当起始段或者结束段线段小于三倍的该恒定值（RTB）时，才会减小 RTB 值。



1) 交点到顶点的距离 Distance between intersection and vertex

类型 2，过渡曲线到抛物线顶点的距离由参数指定。最大偏离半径(RTB)由此产生，如果一个线段太短，那么距离就会缩短，从而使偏离半径达到最大值的 1/3。



3）自适应最大偏离半径

在允许的半径内(包括恒定的允许半径)，系统保证不超过最大允许加速度。根据偏转角度和速度的不同，平滑段内的最大轴加速度可能不同。自适应最大半径的目标是实现平滑过程中的最大加速度。为了达到这一目的，根据程序设定的速度和加速度减小了平滑半径。换句话说，如果程序速度改变，最大半径也会改变。但是改变 Override 对半径没有影响。

1.12.4 四次曲线平滑过渡

指令	# set paramVertexSmoothing(<type>;<subtype>; radius>)#
参数（type）	抛物线平滑：3
参数（subtype）	1: 常数路径最大偏离半径 Constant tolerance radius
	2: 交点到顶点的距离 Distance between intersection and vertex
	3: 自适应最大偏离半径 Adaptive tolerance radius
参数（radius）	最大偏离半径

在四次曲线平滑过渡中，轴分量的加速度没有阶跃变化。因此，在半径相同的情况下，

可能需要比抛物线平滑更小的输入速度。子类型的工作原理与抛物型的工作原理相同。

1.12.5 三阶贝塞尔曲线平滑过渡

指令	# set paramVertexSmoothing(<type>;<subtype>; radius>)#
参数（type）	抛物线平滑：4
参数（subtype）	1: 常数路径最大偏离半径 Constant tolerance radius
	2: 交点到顶点的距离 Distance between intersection and vertex
	3: 自适应最大偏离半径 Adaptive tolerance radius
参数（radius）	最大偏离半径

对于三阶贝塞尔曲线，当进入最大偏差半径时，轴向分量的加速度会出现阶跃变化。最大值受到轴的最大加速度和 C1 因子的限制。这种过渡可以用于所有的线段过渡，但是子类型 2 和 3 只适用于直线/直线过渡。

过渡段的锐角情况：

贝塞尔曲线是默认生成的，即使是非常小的锐角。为了避免超过动态值，在这种情况下需要降低相当多的速度。然而，由于动力学参数在样条中保持不变，在样条中的运动可能非常缓慢。在这种情况下，它通常是实际的开始段过渡与精确的停止。

可以使用 AutoAccurateStop 命令来避免手动计算角度。

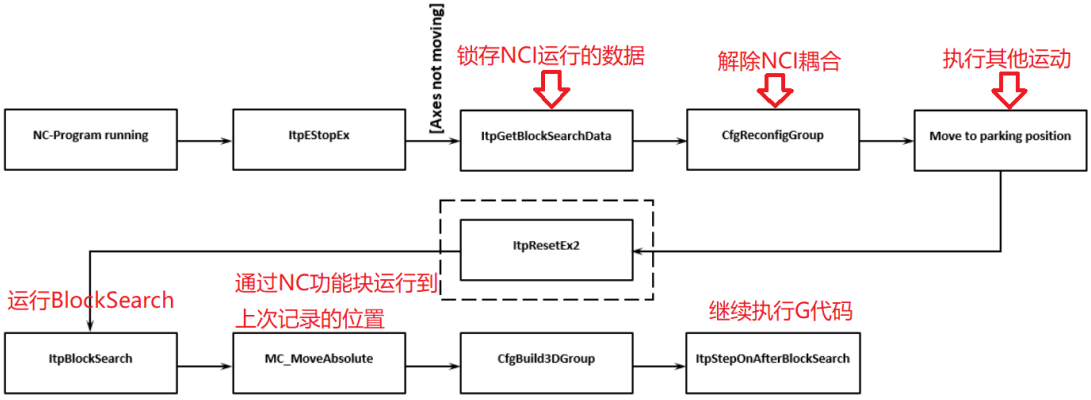
1.12.6 五阶贝塞尔曲线平滑过渡

指令	# set paramVertexSmoothing(<type>;<subtype>; radius>)#
参数（type）	抛物线平滑：5
参数（subtype）	1: 常数路径最大偏离半径 Constant tolerance radius
	2: 交点到顶点的距离 Distance between intersection and vertex
	3: 自适应最大偏离半径 Adaptive tolerance radius
参数（radius）	最大偏离半径

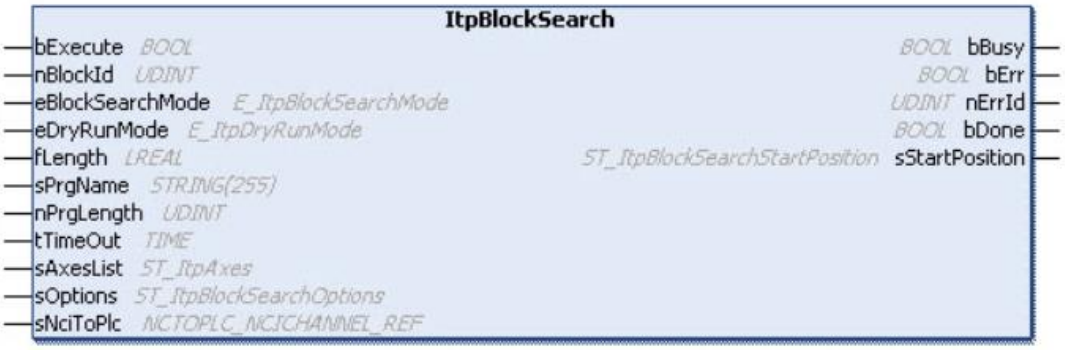
在使用 5 阶贝塞尔曲线平滑过渡时，轨迹在进入最大偏差半径时的加速度没有阶跃变化。换句话说，如果选择了这种过渡类型，路径轴的加速度总是恒定的。这种过渡类型可以用于所有的段转换但子类型 2 和 3 只适用于直线/直线过渡。

1.13 断点搜索 Block Search

经典 NCI 支持断点搜索功能。断点搜索用于运行过程中换刀或者其他需要运动中止的场合，通过功能块记录 NCI 当前执行信息，进入中断后可以通过 NC 功能运行轴，在中断完成后，程序可以在原来的位置和 G 代码行继续执行。其运行流程图如下：



1.13.1 ItpBlockSearch 功能块



功能块 ItpBlockSearch 将编译器设置为在输入接口中定义的点。

ItpBlockSearch 只能在不包含运动的 G 代码段执行，如果第一段 G 代码包含运动指令，该功能块将会报错。也就是说 BlockSearch 只能在断点后的第二段 G 代码开始有效。

这个功能块的输入值可以从函数块 ItpGetBlocksearchData 中获取，也可以手动设置。一旦编译器已经用 ItpBlocksearch 设置到定义的位置，运动就可以在输出变量 sStartPosition 指示的位置继续使用 ItpStepOnAfterBlocksearch 功能块。

主要输入变量含义：

- **nBlockId** BlockSearch 的起始点行号，需要结合 eBlockSearchMode 的模式使用，eg: N4711
- **eBlockSearchMode** 定义 BlockSearch 的搜索模式，

```
TYPE E_ItpBlockSearchMode :
(   ItpBlockSearchMode_Disable          := 0,
    ItpBlockSearchMode_BlockNo          := 1,
    ItpBlockSearchMode_EntryCounter     := 2
    );
END_TYPE
```

ItpBlockSearchMode_Disable: 断点搜索无效 (initial value).

ItpBlockSearchMode_BlockNo: 断点搜索执行通过 G 代码中的程序行号 (e.g. N4711) 这需要 G 代码编辑人员所编写的 G 代码行号不能重复。

ItpBlockSearchMode_EntryCounter: 断点搜索执行通过一个唯一的入口计数 EntryCounter。这个入口计数器是隐藏的并且是唯一的，在 G 代码程序中不可见。

■ E_ItpDryRunMode

枚举变量 **E_ItpDryRunMode** 列举了以什么样的方式进行断点搜索 。

```
TYPE E_ItpDryRunMode :
(
   ItpDryRunMode_Disable            := 0,
   ItpDryRunMode_SkipAll            := 1,
   ItpDryRunMode_SkipMotionOnly     := 2,
   ItpDryRunMode_SkipDwellAndMotion  := 3
);
END_TYPE
```

ItpDryRunMode_Disable: 无效 (初始值).

ItpDryRunMode_SkipAll: 断点以前的 G 代码跳过，但是 R 变量写入.

ItpDryRunMode_SkipMotionOnly: 只有含有运动指令的 G 代码跳过，R 变量写入，停顿时间和 M 函数有效。

ItpDryRunMode_SkipDwellAndMotion: 含有运动指令的 G 代码和停顿时间跳过，R 变量写入， M 函数有效。

■ stOptions

该功能包含了一些额外的断点搜索选项。

```
TYPE ST_ItpBlockSearchOptions :
STRUCT
   bIsRetrace        : BOOL:= FALSE;
   bRetraceBackward  : BOOL:= FALSE;
   bScanStartPos     : BOOL:= FALSE;
END_STRUCT
END_TYPE
```

bIsRetrace:指示回退功能是否激活.

bRetraceBackward: 指示路径中是否有反向运动.

bScanStartPos: 指是否在程序的起始处读取当前轴位置。当与 ST_ItpAxesList 结合时，需要设置为 true，只是在旧版本才需要设置为 false（兼容问题）。

1.13.2 ItpGetBlockSearchData 功能块

功能块 ItpGetBlocksearchData 读取路径上的当前位置。通常这个命令是在暂停状态下调用的。随后可以使用 ItpBlockSearch 将编译器设置为从存储在 sBlockSearchData 中的位置运行。

- **sBlockSearchData:** 包含当前路径上的位置信息

```
TYPE ST_ItpBlockSearchData :
STRUCT
   fLength          : LREAL;(* 当前运行功能块剩余距离的百分比*)
   nBlockNo         : UDINT;(* 当前 G 代码的行号*)
   nBlockCounter    : UDINT;(* 当前 G 代码的入口计数值 *)
   bIsRetrace       : BOOL;(*指示回退功能是否激活*)
   bRetraceBackward : BOOL;(* 指示反向运动是否发生在路径上*)
END_STRUCT
END_TYPE
```

1.13.3 ItpStepOnAfterBlockSearch 功能块

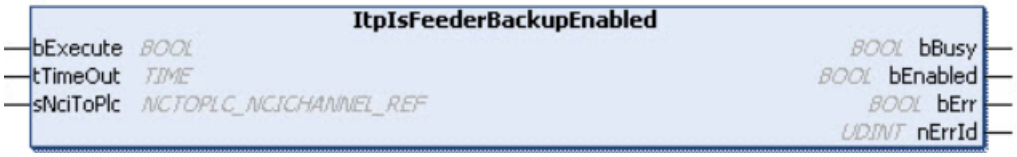
该功能在执行断点搜索后重启 G 代码运行。在使用该功能前需要首先将各轴移动到 BlaockSearch 记录的位置。

1.14 回退 Retrace

1.14.1 ItpEnableFeederBackup

功能块 ItpEnableFeederBackup 用于存储进给路径。它必须在 NC 程序(G 代码)启动之前被激活。如果使用了 Blocksearch 功能，ItpEnableFeederBackup 必须在调用 ItpBlocksearch 之前被激活。只要在 bExecute 上升沿时触发该功能块，并且该功能块触发激活后直到 TwinCAT 重启或 bEnable = FALSE。如果进给备份没有启动，则回退不起作用。
这个功能可以通过 ItpIsFeederBackupEnabled 功能块进行验证。

1.14.2 ItpIsFeederBackupEnabled



功能块 ItpIsFeederBackupEnabled 用于指示是否开启了进给备份功能。
在使用回退之前，必须启用进给备份。

1.14.3 ItpIsFeedFromBackupList

函数 ItpIsFeedFromBackupList 用于当 NCI 的进给入口条目(SAF 和 SVB)从备份列表发送时，函数 ItpIsFeedFromBackupList 变为 TRUE。在向后移动过程中，所有条目都从备份列表中发送。如果程序以正向模式执行，第一个条目通常也来源于备份列表。这取决于回退条目的数量，以及调用跟踪时 SVB 和 SAF 表中的条目数量。所有进一步的命令都来自原始的代码。
当 NCI 处理备份列表时，并不是所有的函数都是可用的或有意义的。比如：

- @714 这样的编译器中止不会被执行
- 只要运动发生在备份路径上(向前或向后)，r 参数的修改就不会生效。当路径数据不再来自备份列表时，r 参数修改将再次生效。

1.14.4 ItpIsFristSegmentReached

ItpIsFirstSegmentReached 是一个函数，它指示在回退过程中是否到达了程序的起始位置。

1.14.5 ItpIsMovingBackwards

ItpIsMovingBackwards 是一个函数, 用于指示当前 G-Code 程序的路径上是否发生向后移动。

1.14.6 ItpRetraceMoveBackward

功能块 ItpRetraceMoveBackward 处理程序(G-Code)开始处实际位置的几何入口。
处理流程:
1. 激活进给备份列表(参见 ItpEnableFeederBackup), 用 ItpEStopEx 停止 NC 程序
2. 等待并确保组中的所有轴都处于静止状态
3. 调用 ItpRetraceMoveBackward
4. 用 ItpEStop 停止向后移动, 否则程序返回起点
5. 再次调用 ItpRetraceMoveBackward 来继续前进
6. 如果需要, 调用 ItpEStopEx 和 ItpRetraceMoveBackward 等。
注意不要与顶点过渡一起使用。并且 m 函数在向后运动动时被抑制无效。

1.14.7 ItpRetraceMoveForward

函数块 ItpRetraceMoveForward 将当前块(例如位置)的所有条目向前移动到 NC 内核。
调用它是为了在调用 ItpRetraceMoveBackward 之后反转方向。

1.13 代码检查

经典的 G 代码编译器支持 BlockSerach 功能。我们可以借用这个小技巧。可以在使用无效的块 id 启动 g-code 文件之前调用 BlockSearch。在这种情况下, 编译器会评估所有代码, 在开始运行前检查是否存在错误。示例如下:

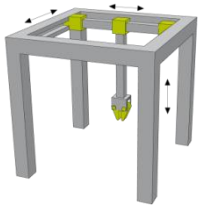
```
bBlockSearch(
        bExecute:=bExecBlockSearch ,
        nBlockId:= 999999,
        eBlockSearchMode:= ItpBlockSearchMode_BlockNo,
        eDryRunMode:= ItpDryRunMode_SkipAll,
        fLength:= ,
        sPrgName:= 'Test2.nc',
        nPrgLength:=LEN('Test2.nc') ,
```

```
tTimeOut:= ,
sAxesList:= ,
sOptions:= ,
bBusy=> ,
bErr=> ,
nErrId=> ,
bDone=> ,
sStartPosition=> ,
sNciToPlc:= in_stItpToPlc);
```

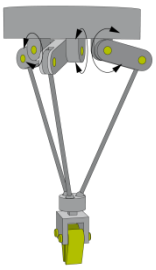
第二章 运动学坐标系变换

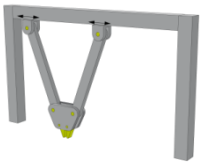
运动学坐标系变换可以把机器人关节坐标系变换集成到一个复杂的机器生产线中，机器人单元不再是一个黑匣子针对所有机器部件的编程工具，不需要特殊的机器人编程语言，一个 CPU 能够控制整条生产线。在 TwinCAT 中集成机器人控制节省了额外的 CPU，一个机器人系统中的配置、参数化和诊断都可以利用一些现有的工具，由于不需要在不同 CPU 和 PLC 之间进行数据交换所以提高了效率，由于不需要复杂通信和接口机器人所以能实现高性能和高精度。

TwinCAT 运动学坐标系变换支持的关节如下，而且所有关节的轨迹规划都需要用到上一章节所讲的 NCI 编程。

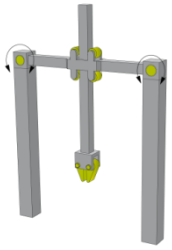


Cartesian Portal



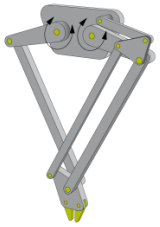


Shear kinematic

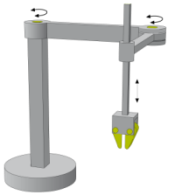


Roll kinematic (H-Bot)

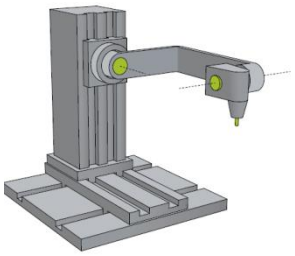
3D Delta



2D Parallel kinematic



SCARA



Crane kinematic

Hexapod

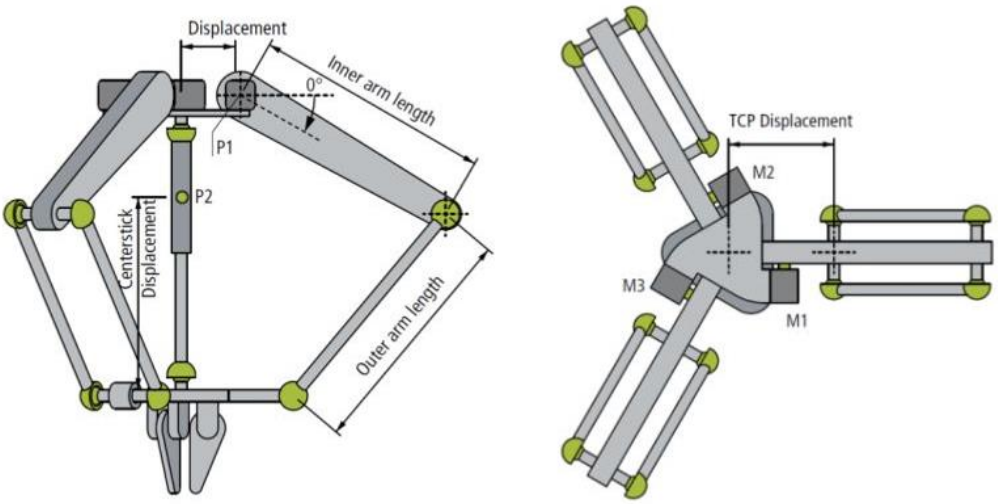
(restricted)

5D Kinematics (XYZAB)

6 axes seriell kinematic

(restricted)

2.1 运动学坐标系变换配置流程（Delta 为例）



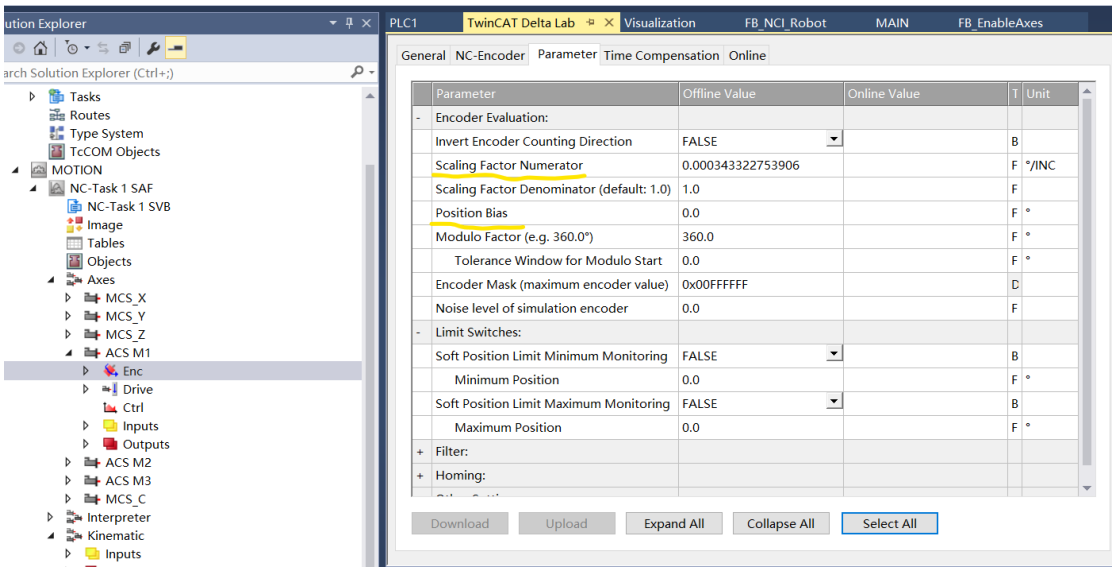
Delta 机器人参数和标定示意图

2.1.1 安装软件开发包

TF511x-Kinematic-Transformation.exe（TwinCAT3 平台）或者 TS511x-Kinematic-Trafo.exe（TwinCAT2 平台）。Delta 机器人需要 TF5112 级别的授权。

2.1.2 添加 NC 轴和设置参数

扫描机器人电机，并链接到 NC 轴 AcsM1、 AcsM2、AcsM3 和 McsC（旋转轴）。另外添加世界坐标系轴 McsX、McsY、McsZ，如下图所示。



先对四个关节轴设置 NC 参数。

1）设置 ScalingFactor

ScalingFactor =360/减速比/编码器旋转一周的脉冲数

2）根据电机性能设置 Acs 轴的速度

ReferenceVelocity= 110%*RateVelocity

MaximumVelocity= 100%*RateVelocity

ManualVelocity(Fast) =30%*RateVelocity

ManualVelocity(slow) =5%*RateVelocity

3）设置跟随误差限制和软限位

根据机器人的机械限位，设置软限位,以及跟随误差限制。

4）调节电机参数

让电机做循环运行，调整 PID 使电机平稳运行，详细步骤见伺服电机的速度环和位置环整定文 档。

5）调整电机的方向

Delta 机器人的 Acs1、Acs2 和 Acs3 的机械臂以角度为单位，以向下为正。其中 M1 轴为平行于传送带方向的轴。旋转轴 McsC 也是以角度为单位，绕 Z 轴方向旋转。参见 Delta 机器人的标定示意图。

如果方向有误，需要调整 InvertEncoder CountingDirection 和 InvertEncoder Counting Direction 两个参数。注意，务必切换到配置模式去掉使能后同时修改再激活和使能，务必确保两个参数同时为 true 或者同时为 false，否则会造成飞车等严重后果。

6）设置世界坐标系轴的加速度等动力学参数和软限位

2.1.3 原点标定

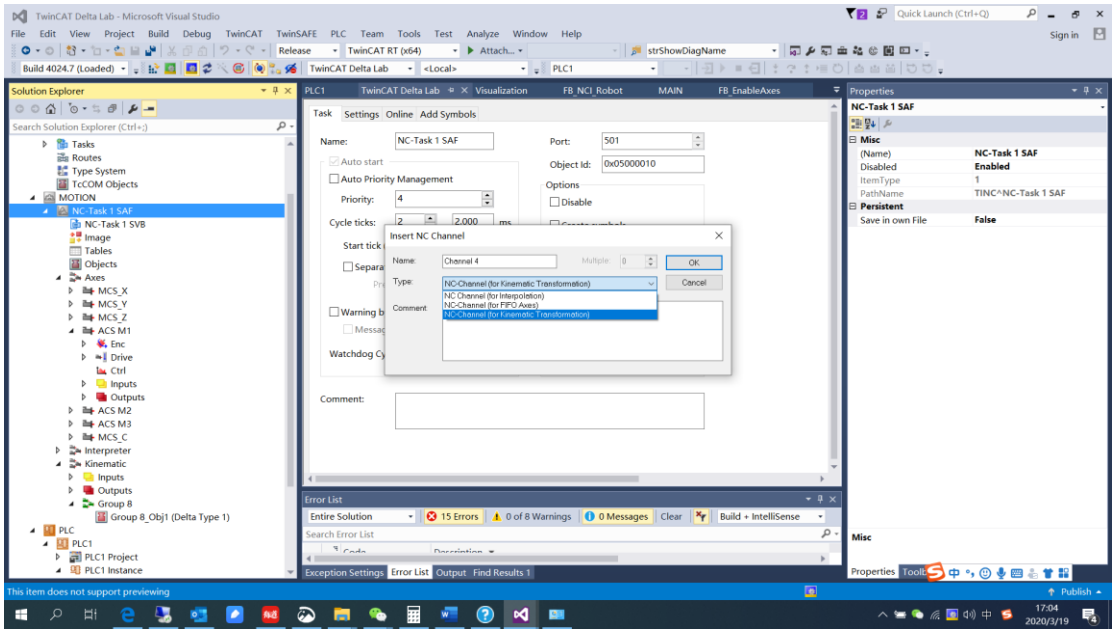
Delta 机器人上铰链中心点与关节中心点水平时为机械臂的零点，需要通过工具进行精确标定。

如果 online 界面的当前位置值不为 0，需要修改 PositionBias（见上图）将当前位置置为 0，修改步骤是： a，将 PositionBias 设置为 0。 b，查看 online 中当前位置值。 c，将当前位置值取负数填入 PositionBias，保存并 SaveNow。 可以看到当机械臂都处于水平位置时，online 界面当前位置为 0。旋转轴电机的标定可以根据应用进行调整。

2.1.4 添加 Group

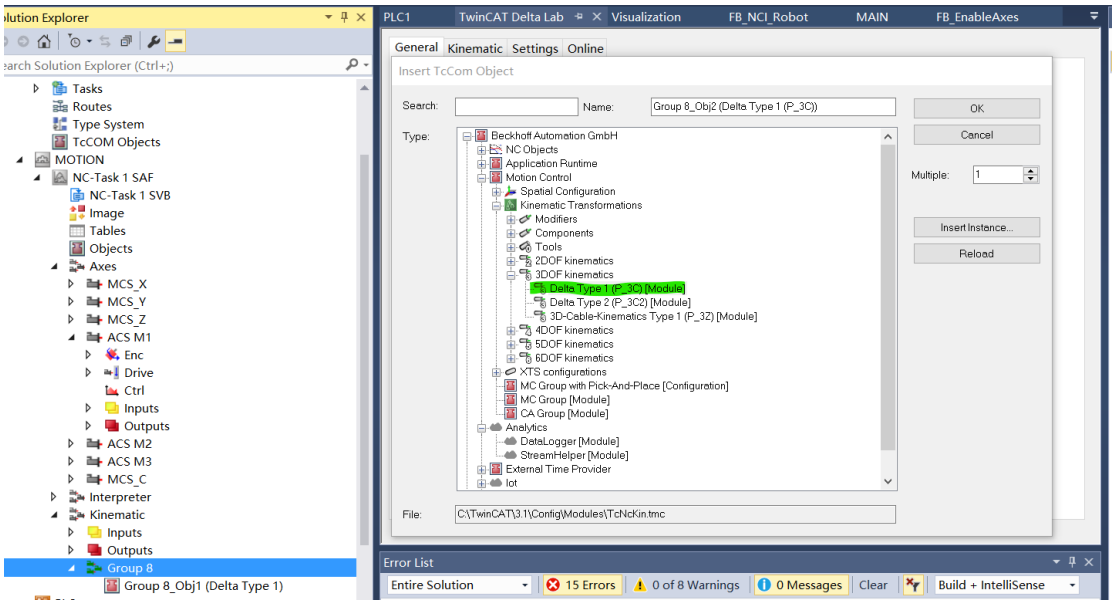
1）添加运动学坐标转换系统

在 NC-Task SAF 上点右键添加 NC—Channel Kinematic，如下图。并更改名称为 kinematic



2）添加 Group

在新出现 kinematic 坐标系统中，鼠标放在 Group8 上，点右键 Append Object（s）出现如下图所示，如果已经安装机器人软件，则会找到 MotionControl/Kinematic Transforms 点击打开进行添加。 旋转 Delta Type1。其他机器人系统，可以参照此步骤。



2.1.5 参数设置

完成上述步骤后，在新出现的 Group8——Obj1（Delta type1）右边的 Parameter 中设置机器人臂的长度，如下图所示，机器人的参数定义见 Delta 机器人参数和标定示意图

x-shift　是 x 方向的偏置

y-shift　是 y 方向的偏置

z-shift　是 z 方向的偏置

Inner Arm Length 是上臂的长度;
Outer Arm Length 是下臂的长度;
Displacement 是上铰链中心点到机器人中心点的距离;
TCP Displacement 是两相邻下铰链中心点的中心点到机器人中心点的距离;
Inner Arm Mass 上臂的重量;
Inner Arm moment of Inertia 上臂的惯量
Outer Arm Mass 下臂的重量;
Tcp Mass 下平台（动平台）的重量

Object | Context | Parameter (Init) | Interfaces

	Name	Value	CS	Unit	Type	PT...	Comment
-	Configuration						
-	MCS offset	...	☐			0x0...	Static offs...
	x-shift	0.0		mm	LREAL		Static x-off...
	y-shift	0.0		mm	LREAL		Static y-off...
	z-shift	0.0		mm	LREAL		Static z-off...
+	Spatial reference definition	...	☐			0x0...	Specificati...
-	Kinematic						
	Inner arm length	400.0	☐	mm	LREAL	0x0...	Length of t...
	Outer arm length	850.0	☐	mm	LREAL	0x0...	Length of t...
	Displacement	150.0	☐	mm	LREAL	0x0...	Offset dist...
	TCP displacement	50.0	☐	mm	LREAL	0x0...	Radius of t...
-	Dynamic						
	Inner arm mass	1.0	☐	kg	LREAL	0x0...	Mass of th...
	Inner arm moment of inertia	20000.0	☐	kg mm²	LREAL	0x0...	Moment o...
	Outer arm mass	0.5	☐	kg	LREAL	0x0...	Mass of th...
	Link mass	0.0	☐	kg	LREAL	0x0...	Mass of th...
	TCP mass	1.0	☐	kg	LREAL	0x0...	Mass of th...
	Center stick mass	0.0	☐	kg	LREAL	0x0...	Mass of th...
	Center stick: moment of inertia	0.0	☐	kg mm²	LREAL	0x0...	Moment o...
	Center stick: center of mass displacement	0.0	☐	mm	LREAL	0x0...	Displacem...
	Drive Torques						

2.1.6 机器人结构体和配置功能块

1）在 PLC 程序中添加机器人坐标系轴和机器人结构体。

机器人的轴包含三个关节坐标系轴和四个世界坐标系轴，本例中在上一章节 NCI 程序的集成上添加 A1/A2/A3，三个关节坐标系轴。

```
PROGRAM MAIN
VAR

    io_X                    : AXIS_REF;
    io_Y                    : AXIS_REF;
    io_Z                    : AXIS_REF;
    io_C                    : AXIS_REF;
    io_A1                   : AXIS_REF;
    io_A2                   : AXIS_REF;
    io_A3                   : AXIS_REF;

    xUserEnableAxes         : BOOL;
    fbEnableAxes            : FB_EnableAxes;
    bAllAxesReady           : BOOL;


```

2）机器人配置功能块

```
(************Robot Config*******************************************)

in_stKinToPlc    AT %I*       : NcToPlc_NciChannel_Ref;
out_stPlcToKin   AT %Q*       : PLCTONC_NciChannel_Ref;
stAxesConfig                  : ST_KinAxes;
fbConfigKinGroup              : FB_KinConfigGroup;
fbResetKinGroup               : FB_KinResetGroup;
fbCheckActualKinStatus        : FB_KinCheckActualStatus;
xUserConfigRobotGroup           : BOOL := FALSE;
xUserCartesianMode            : BOOL := TRUE;
xUserResetKinGroup            : BOOL := FALSE;
xUserkinGroupReadStatus       : BOOL;
eKinStatus                    :E_KINSTATUS;(*机器人组状态*)
```

- 机器人输入输出结构体变量

  定义机器人输入输出结构体变量，如上图所示,并把该变量与机器人组的输入输出变量进行链接。

  in_stKinToPlc　　AT %I*　　: NcToPlc_NCIChannel_Ref;

  out_stPlcToKin　　AT %Q*　　: PLCTONC_NCIChannel_Ref;

- ST_KinAxes

  添加机器人轴结构体 ST_KinAxes，Delta 机器人有三个关节坐标系轴和三个世界坐标系轴参与运算，把这些轴的 AxisId 赋值给 ST_KinAxes。

  stAxesConfig.nAxisIdsAcs[1]　　:= io_A1.NcToPlc.AxisId;

  stAxesConfig.nAxisIdsAcs[2]　　:= io_A2.NcToPlc.AxisId;

  stAxesConfig.nAxisIdsAcs[3]　　:= io_A3.NcToPlc.AxisId;

  stAxesConfig.nAxisIdsMcs[1]　　:= io_X.NcToPlc.AxisId;

  stAxesConfig.nAxisIdsMcs[2]　　:= io_Y.NcToPlc.AxisId;

  stAxesConfig.nAxisIdsMcs[3]　　:= io_Z.NcToPlc.AxisId;

- FB_KinConfigGroup

  功能块 FB_KinConfigGroup 是对机器人进行世界坐标系模式和关节坐标系模式进行切换。当

  bCartesianMode 为 True 时，bExecute 收到上升沿信号把机器人切换到世界坐标系轴模式，当 bCartesianMode 为 false 时，bExecute 收到上升沿信号把机器人切换到关节坐标系轴模

```
fbConfigKinGroup(
    bExecute                  := xUserConfigRobotGroup,
    bCartesianMode            := xUserCartesianMode, (*坐标系切换*)
    stAxesList                := stAxesConfig,
    stKinRefIn                := in_stKinToPlc);
```

- FB_KinResetGroup

  功能块 FB_KinResetGroup 是当机器人报错时对机器人 Group 进行复位。

```
fbResetKinGroup(
    bExecute                    := xUserResetKinGroup,
    nItpChannelId                := ,
    stKinRefIn                  := in_stKinToPlc,
    stAxesList                  := stAxesConfig);
```

- FB_KinCheckActualStatus

  功能块 FB_KinCheckActualStatus 检测当前机器人的状态，上升沿触发。

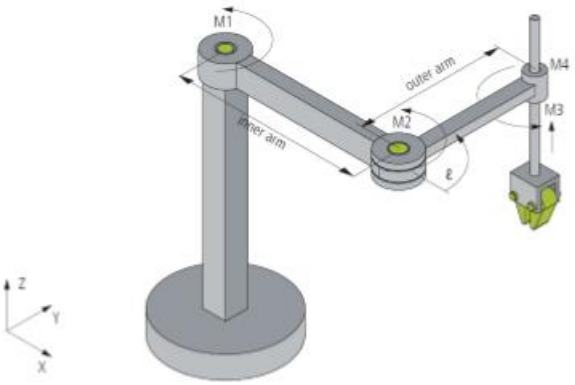
```
fbCheckActualKinStatus(
    bExecute:=xUserkinGroupReadStatus ,
    stAxesListReference:=stAxesConfig ,
    stKinRefIn:=in_stKinToPlc ,
    eKinStatus=>eKinStatus,
    bDone=> ,
    bError=> ,
    nErrorId=> );
```

  其中 eKinStatus 是枚举类型，共有 6 种状态

Name	类型	含义
KinStatus_Error	INT	错误
KinStatus_Empty	Int	机器人未激活或者切换到关节坐标系模式时的状态
KinStatus_Unknown	Int	未知状态
KinStatus_StartPending	Int	挂起，比如电机位使能激活机器人，可能出现这个状态
KinStatus_Ready	Int	机器人准备，切换到世界坐标系模式的状态
KinStatus_InvalidItVersion	int	无效的版本，需要安装对应模型的机器人安装包

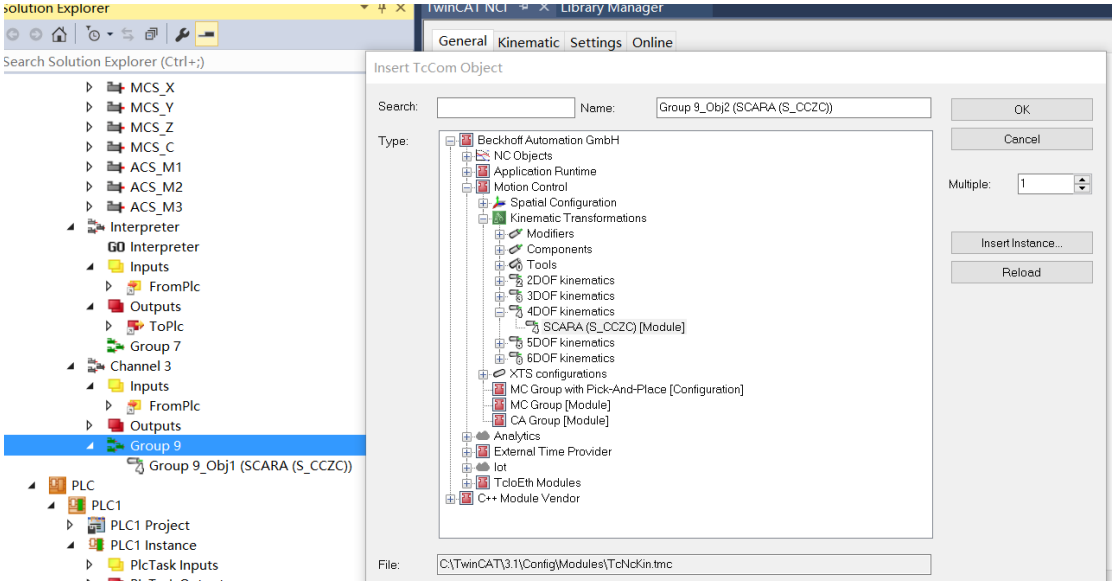
2.2 SCARA 类型配置

SCARA 机器人有 3 个旋转关节，其轴线相互平行，在平面内进行定位和定向。另一个关节是移动关节，用于完成末端件在垂直于平面的运动。手腕参考点的位置是由两旋转关节的角位移 φ1 和 φ2，及移动关节的位移 z 决定的，即 p=f(φ1,φ2,z)，如图所示。这类机器人的结构轻便、响 应快，例如 Adept1 型 SCARA 机器人运动速度可达 10m/s，比一般关节式机器人快数倍。它最适用于平面定位，垂直方向进行装配的作业。其坐标系和关节定义如下图所示。

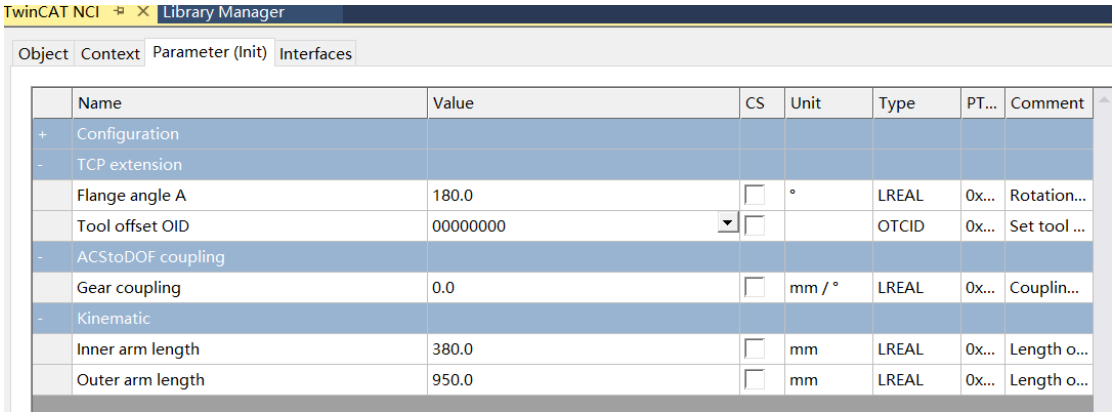


2.2.1 Group 添加

SCARA 机器人的坐标系配置流程可以参考上一章的 Delta 机器人的配置流程，其中在选择 Group 类型时，选择 SCARA 类型，如下图所示。



2.2.2 参数设置



Flange Angle A 是末端夹具的旋转角度限制。

Tool Offset OID 可以添加一个工具偏移坐标系

Gear coupling 当旋转轴 C 和上下轴 Z 存在机械上的耦合关系时，这里需要填入机械齿轮比。

Inner arm Length 内臂长度，见示意图。

Outer arm Length 外臂长度，见示意图。

2.2.3 PLC 程序

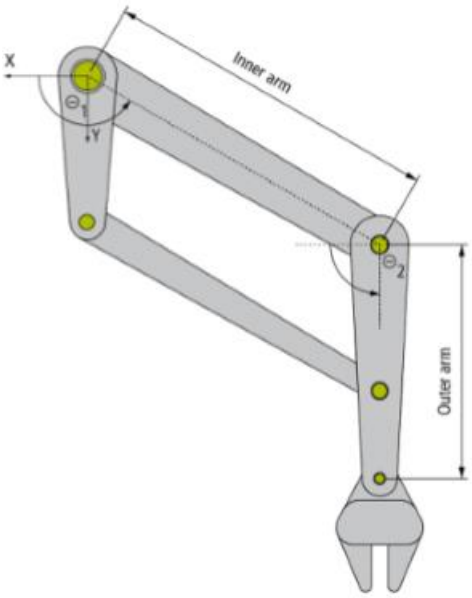
SCARA 有四个关节电机轴参与机器人的运算，所以需要四个关节坐标系轴，四个世界坐标

系轴。其他配置与 Delta 机器人相同。

MAIN.ACT03_Robot | MAIN.ACT01_RPara | FB_NCI_Admin.ACT09_Reset | FB_

```
stAxesConfig.nAxisIdsAcs[1]      := Acs[1].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsAcs[2]      := Acs[2].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsAcs[3]      := Acs[3].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsAcs[4]      := Acs[4].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[1]      := Mcs[1].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[2]      := Mcs[2].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[4]      := Mcs[4].AxisRef.NcToPlc.AxisId;

fbConfigKinGroup(
    bExecute                     := xUserConfigRobotGroup,
    bCartesianMode               := xUserCartesianMode, (*坐标系切换*)
    stAxesList                   := stAxesConfig,
    stKinRefIn                   := in_stKinToPlc);

fbResetKinGroup(
    bExecute                     := xUserResetKinGroup,
    nItpChannelId                    := ,
    stKinRefIn                   := in_stKinToPlc,
    stAxesList                   := stAxesConfig);

(*读取机器人状态*)
IF NOT  fbCheckActualKinStatus.bBusy THEN
    xUserkinGroupReadStatus:=NOT xUserkinGroupReadStatus;
END_IF
fbCheckActualKinStatus(
    bExecute:=xUserkinGroupReadStatus ,
    stAxesListReference:=stAxesConfig ,
    stKinRefIn:=in_stKinToPlc ,
    eKinStatus=>eKinStatus,
    bDone=> ,
    bError=> ,
    nErrorId=> );
```

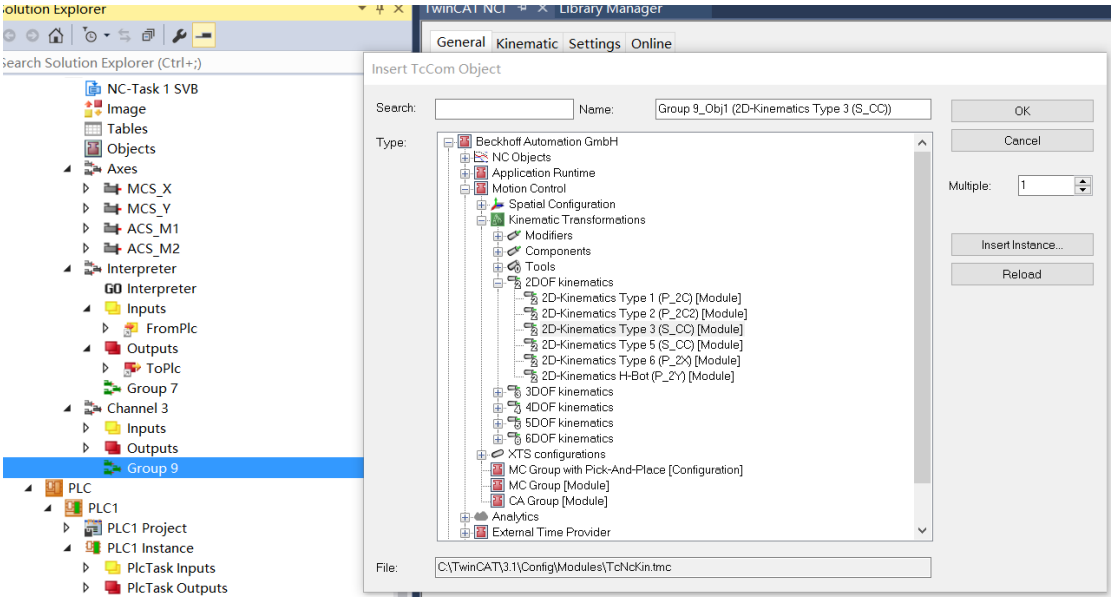
2.3 2D 串联



2D 串联机器人其中一个电机与坐标原点水平，带动 Inner Arm 臂运动，另一个电机带动 Outer Arm 臂，注意不是安装在关节上，是以水平方向为移动角度的初始值

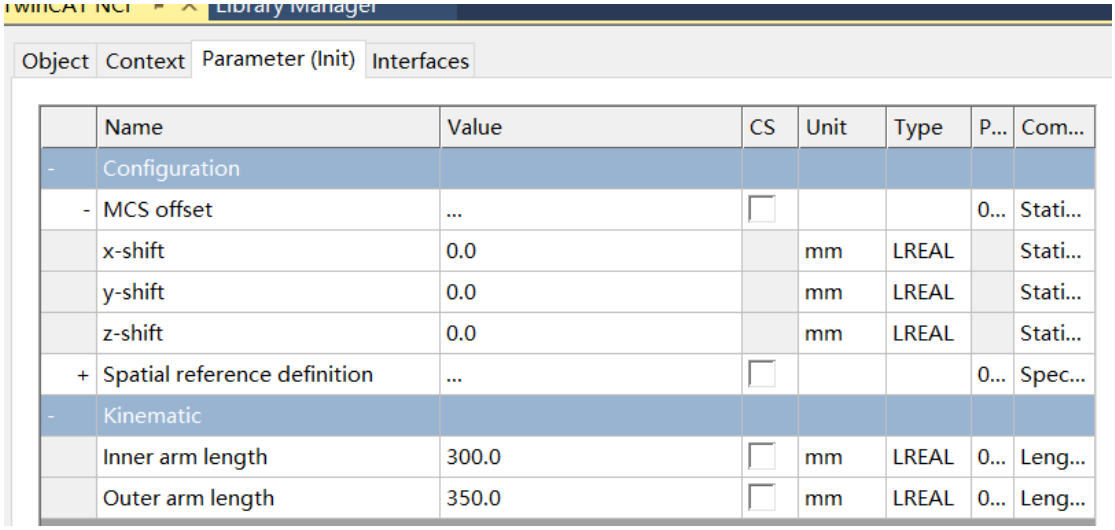
2.3.1 Group 添加

2D 串联机器人的 Group 添加 2D-Kinematics Type 3 类型。



2.3.2 参数设置

2D 串联机器人的参数只要输入 Inner Arm 长度和 Outer Arm 长度即可

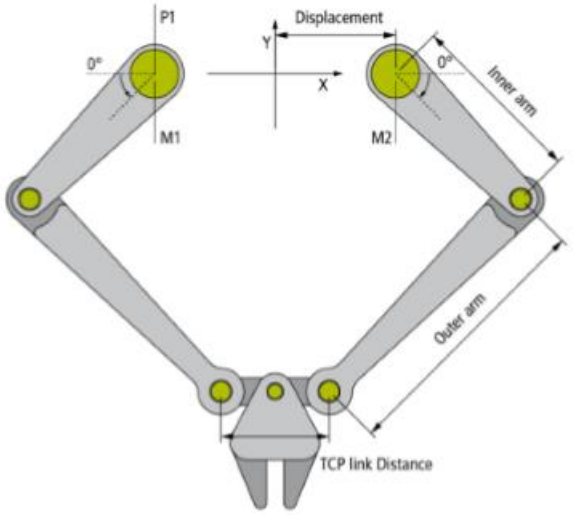


2.3.3 PLC 程序

2D 串联机器人有两个关节电机轴参与机器人的运算，所以需要两个关节坐标系轴，两个世界坐标系轴。其他配置与 Delta 机器人相同。

```
stAxesConfig.nAxisIdsAcs[1]     := Acs[1].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsAcs[2]     := Acs[2].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[1]     := Mcs[1].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[2]     := Mcs[2].AxisRef.NcToPlc.AxisId;
```

2.4 2D 并联



2.4.1 Group 添加

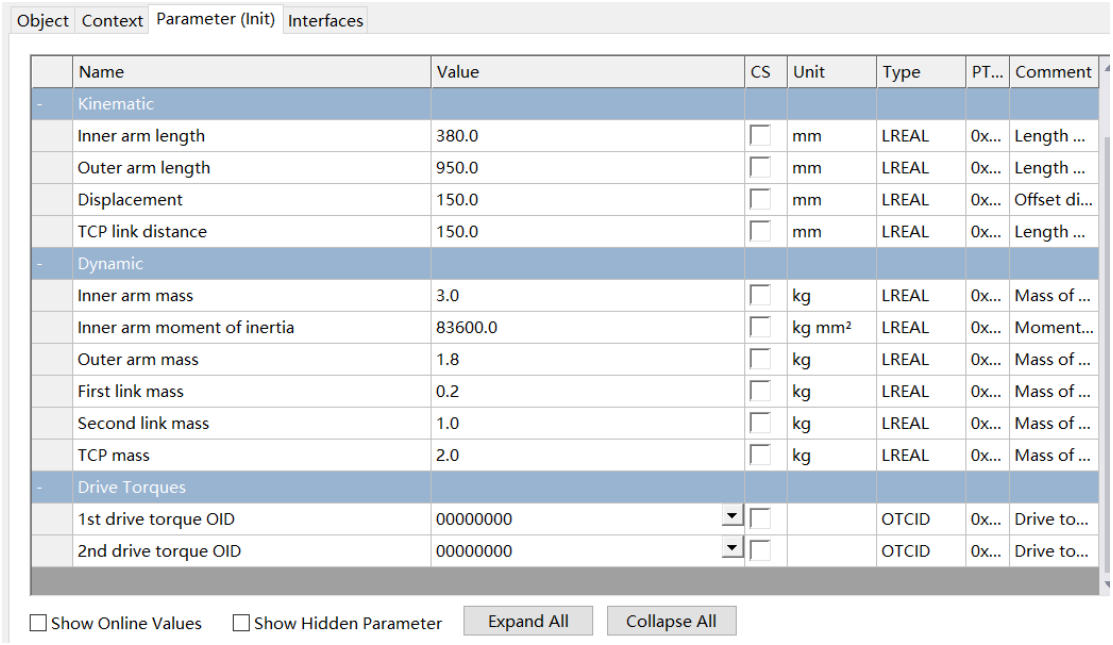
2D 并联机器人的 Group 添加 2D-Kinematics Type 2 类型。如果 TCP link distance=0 则选择 2D-Kinematics Type 1。

2.4.2 参数设置

2D 并联机器人的参数除了输入 Inner Arm 长度和 Outer Arm 长度外，还有

Displacement　两个上臂中心点的距离的一半
TCP link Displacement　下臂末端中心点距离的一半

Inner Arm mass 上臂的重量;
Inner Arm moment of Inertia 上臂的惯量
Outer Arm Mass 下臂的重量;
Tcp Mass　工具夹具的重量

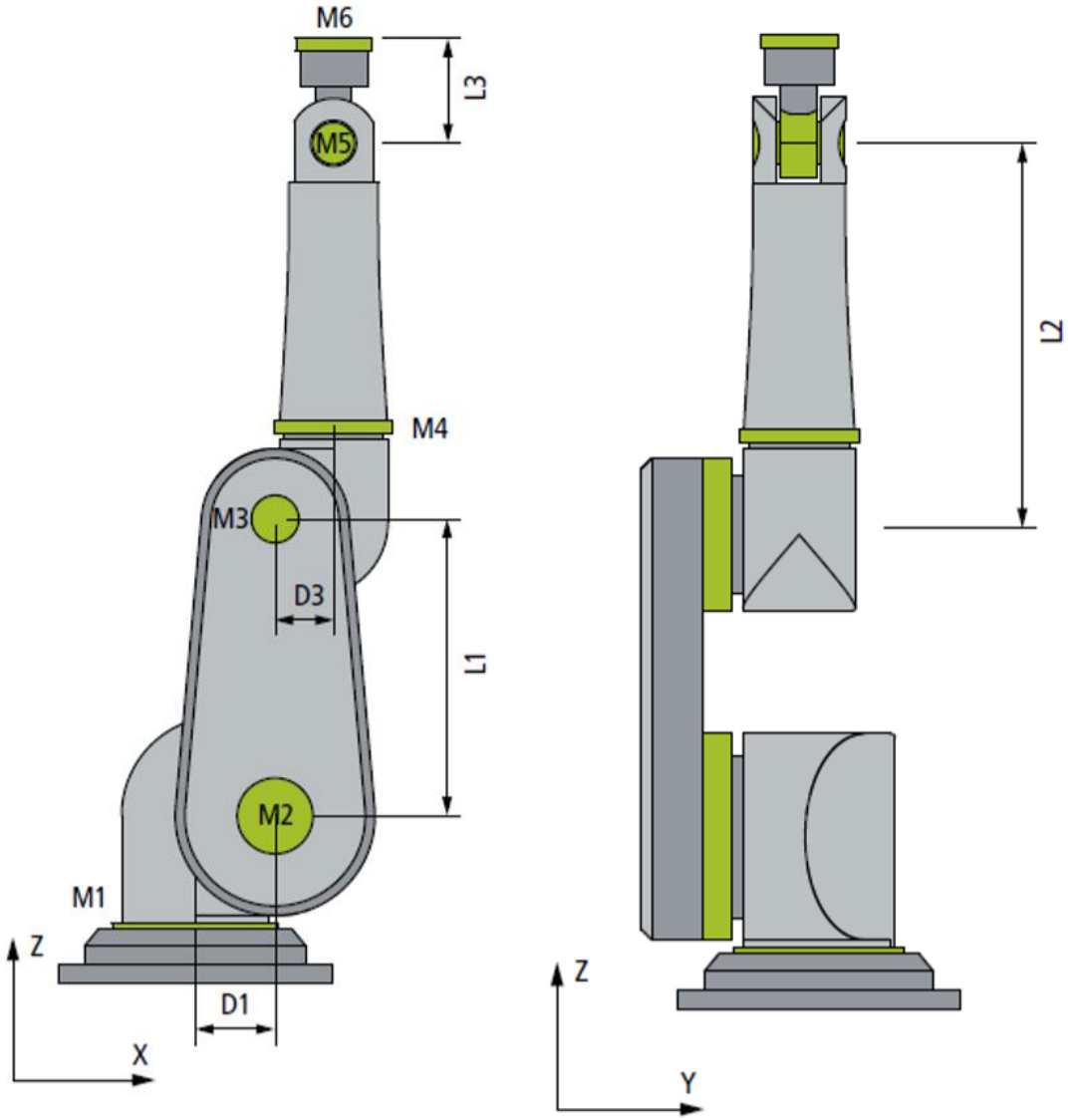


2.4.3 PLC 程序

2D 串联机器人有两个关节电机轴参与机器人的运算，所以需要两个关节坐标系轴，两个世界坐标系轴。其他配置与 Delta 机器人相同。

```
stAxesConfig.nAxisIdsAcs[1]     := Acs[1].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsAcs[2]     := Acs[2].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[1]     := Mcs[1].AxisRef.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[2]     := Mcs[2].AxisRef.NcToPlc.AxisId;
```

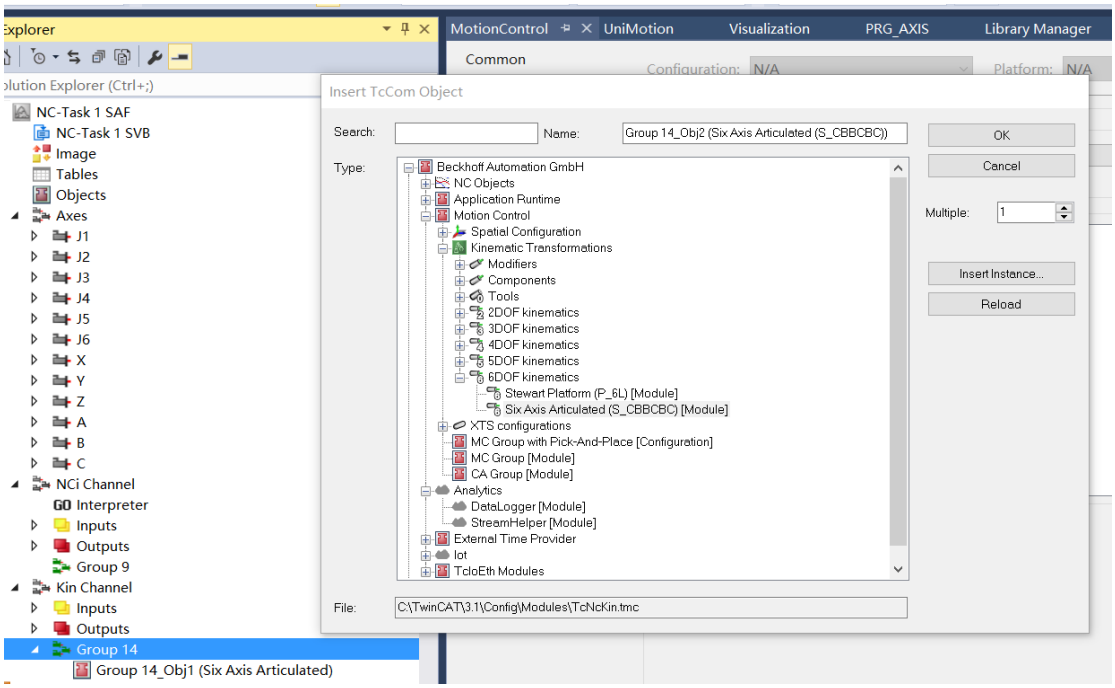
2.5 六轴串联



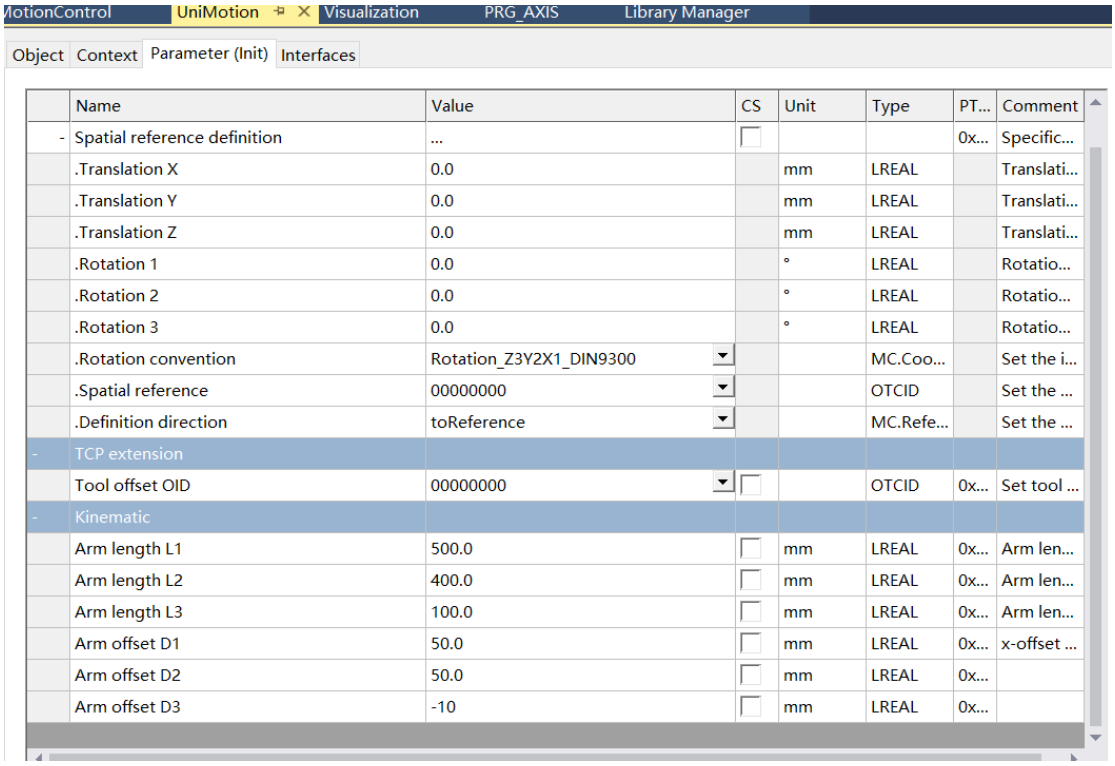
倍福 TwinCAT 软件支持 6 轴串联标准机器人模型，但是受欧盟出口管制限制，购买六轴串联软件包需要在德国商务部备案，具体流程请咨询相关销售。另外 6 轴软件安装包也在购买后才能获得和使用，不能直接在网站上下载。

2.5.1 Group 添加

在安装 6 轴机器人软件开发包后，如下图所示，在 6DOF kinematic 中选择 Six Axis Architectural



2.5.2 参数设置



L1~L3 是 6 轴机器人的臂长，D1~D3 是两臂之间的偏移值，具体参考示意图。

RotationConvention 是机器人矩阵运算规则，如果要和其他品牌机器人的位置计算一致，这里也要选择对应的旋转运算方式。

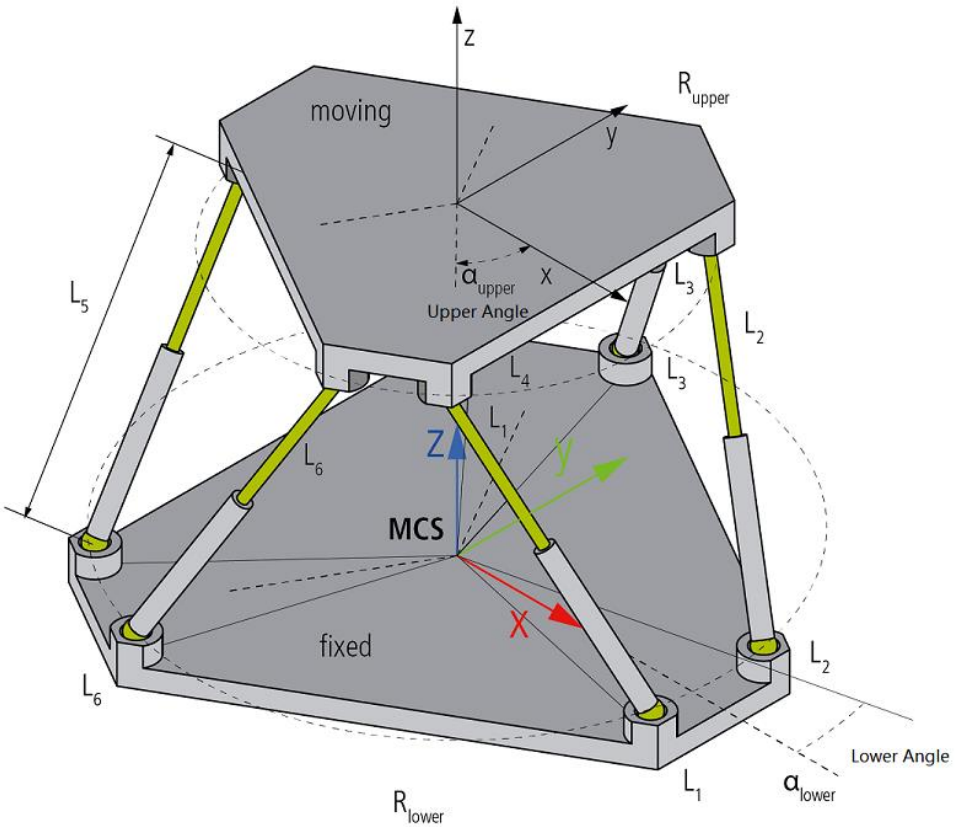
2.5.3 PLC 程序

六轴串联机器人有六个关节坐标系和六个世界坐标系，所以配置六个关节坐标系轴，六个世界坐标系轴。其他配置与 Delta 机器人相同。

MAIN.ACT03_Robot* | MAIN | TwinCAT NCI | Library Manager

```
1  stAxesConfig.nAxisIdsAcs[1]     := Acs[1].AxisRef.NcToPlc.AxisId;
2  stAxesConfig.nAxisIdsAcs[2]     := Acs[2].AxisRef.NcToPlc.AxisId;
3  stAxesConfig.nAxisIdsAcs[3]     := Acs[3].AxisRef.NcToPlc.AxisId;
4  stAxesConfig.nAxisIdsAcs[4]     := Acs[4].AxisRef.NcToPlc.AxisId;
5  stAxesConfig.nAxisIdsAcs[5]     := Acs[5].AxisRef.NcToPlc.AxisId;
6  stAxesConfig.nAxisIdsAcs[6]     := Acs[6].AxisRef.NcToPlc.AxisId;
7
8  stAxesConfig.nAxisIdsMcs[1]     := Mcs[1].AxisRef.NcToPlc.AxisId;
9  stAxesConfig.nAxisIdsMcs[2]     := Mcs[2].AxisRef.NcToPlc.AxisId;
10 stAxesConfig.nAxisIdsMcs[3]     := Mcs[3].AxisRef.NcToPlc.AxisId;
11 stAxesConfig.nAxisIdsMcs[4]     := Mcs[4].AxisRef.NcToPlc.AxisId;
12 stAxesConfig.nAxisIdsMcs[5]     := Mcs[5].AxisRef.NcToPlc.AxisId;
13 stAxesConfig.nAxisIdsMcs[6]     := Mcs[6].AxisRef.NcToPlc.AxisId;
14
```

2.6 Stewart 并联机构

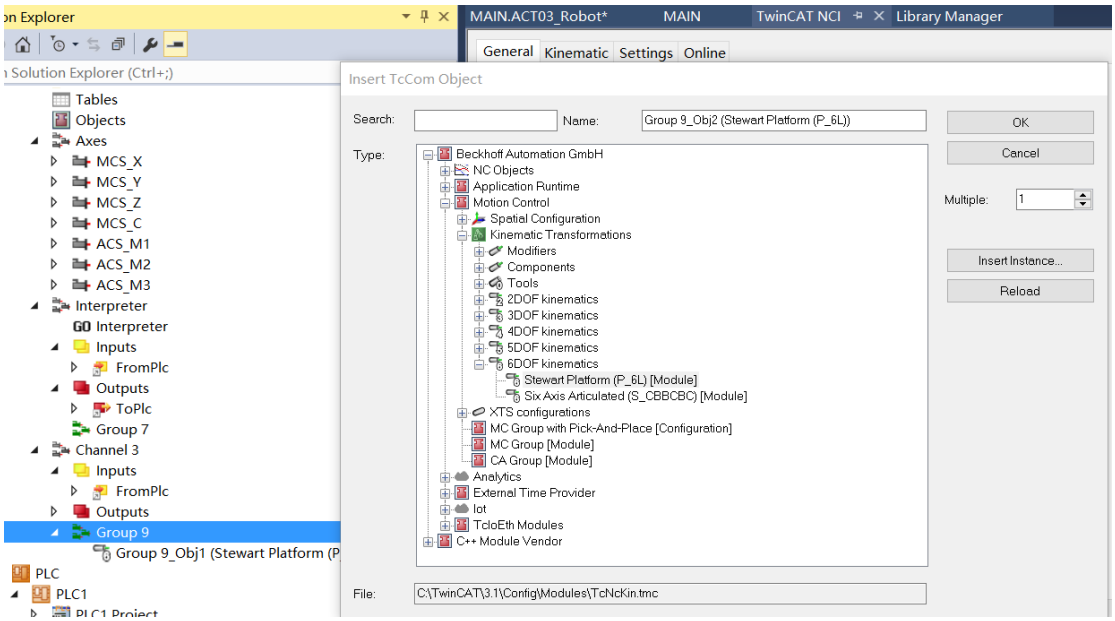


Stewart 并联机构可以用于飞机、汽车等模拟训练，也可以用于 6D 影院等设备，倍福 TwinCAT 软件支持 6 轴并联模型，但是受欧盟出口管制限制，购买六轴并联模型软件包需要

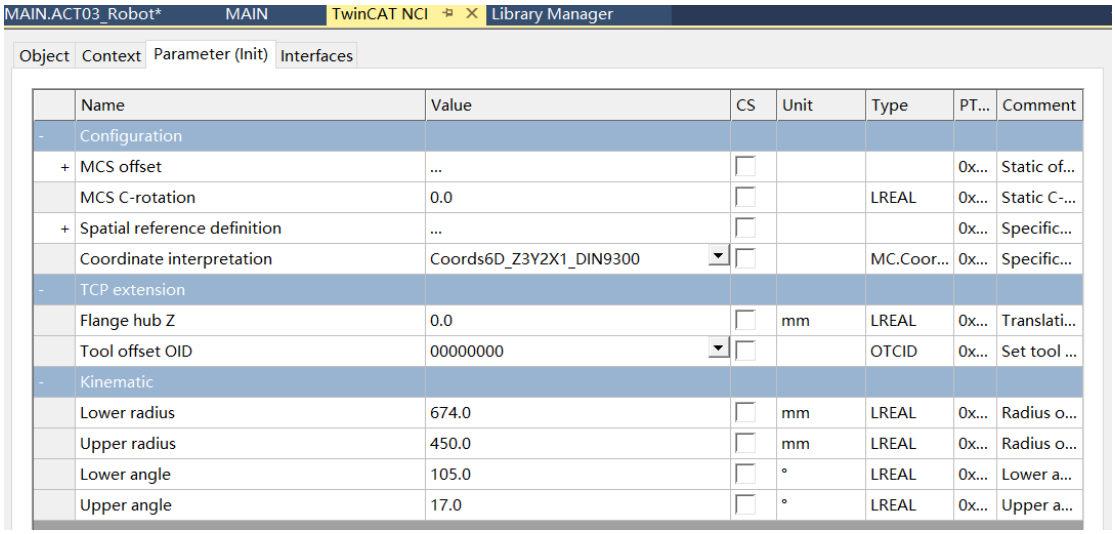
在德国商务部备案，具体流程请咨询相关销售。另外 6 轴软件安装包也在购买后才能获得和使用，不能直接在网站上下载。

2.6.1 Group 添加

在安装 6 轴机器人软件开发包后，如下图所示，在 6DOF kinematic 中选择 Stewart Platform



2.6.2 参数设置



	Name	Value	CS	Unit	Type	PT...	Comment
-	Configuration						
+	MCS offset	...				0x...	Static of...
	MCS C-rotation	0.0			LREAL	0x...	Static C-...
+	Spatial reference definition	...				0x...	Specific...
	Coordinate interpretation	Coords6D_Z3Y2X1_DIN9300			MC.Coor...	0x...	Specific...
-	TCP extension						
	Flange hub Z	0.0		mm	LREAL	0x...	Translati...
	Tool offset OID	00000000			OTCID	0x...	Set tool ...
-	Kinematic						
	Lower radius	674.0		mm	LREAL	0x...	Radius o...
	Upper radius	450.0		mm	LREAL	0x...	Radius o...
	Lower angle	105.0		°	LREAL	0x...	Lower a...
	Upper angle	17.0		°	LREAL	0x...	Upper a...

参数含义见上图所示，

Lower radius 下平台的半径
Upper radius 上平台的半径
Lower angle 下平台两个距离最近的球铰的夹角的一半

Upper angle 上平台两个对应球铰的夹角的一半

2.6.3 PLC 程序

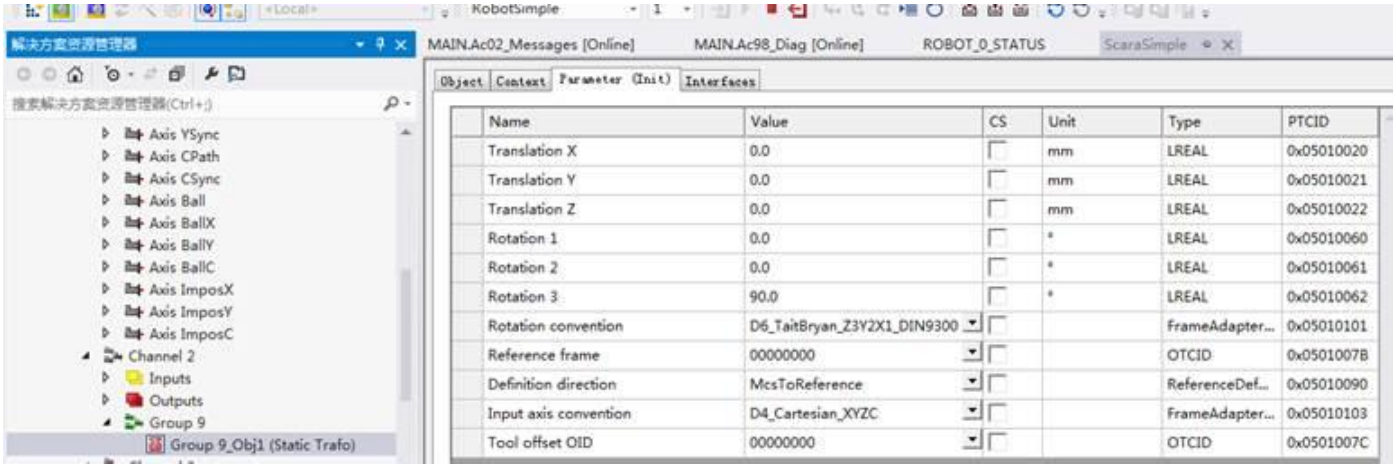
MAIN.ACT03_Robot* | MAIN | TwinCAT NCI | Library Manager

```
1  stAxesConfig.nAxisIdsAcs[1]     := Acs[1].AxisRef.NcToPlc.AxisId;
2  stAxesConfig.nAxisIdsAcs[2]     := Acs[2].AxisRef.NcToPlc.AxisId;
3  stAxesConfig.nAxisIdsAcs[3]     := Acs[3].AxisRef.NcToPlc.AxisId;
4  stAxesConfig.nAxisIdsAcs[4]     := Acs[4].AxisRef.NcToPlc.AxisId;
5  stAxesConfig.nAxisIdsAcs[5]     := Acs[5].AxisRef.NcToPlc.AxisId;
6  stAxesConfig.nAxisIdsAcs[6]     := Acs[6].AxisRef.NcToPlc.AxisId;
7
8  stAxesConfig.nAxisIdsMcs[1]     := Mcs[1].AxisRef.NcToPlc.AxisId;
9  stAxesConfig.nAxisIdsMcs[2]     := Mcs[2].AxisRef.NcToPlc.AxisId;
10 stAxesConfig.nAxisIdsMcs[3]     := Mcs[3].AxisRef.NcToPlc.AxisId;
11 stAxesConfig.nAxisIdsMcs[4]     := Mcs[4].AxisRef.NcToPlc.AxisId;
12 stAxesConfig.nAxisIdsMcs[5]     := Mcs[5].AxisRef.NcToPlc.AxisId;
13 stAxesConfig.nAxisIdsMcs[6]     := Mcs[6].AxisRef.NcToPlc.AxisId;
14
```

2.7 坐标系的静态变换

2.7.1 用于直角坐标模型

对于直角坐标机器人，虽然说他的世界坐标系轴 XYZ 和关节坐标系轴一一对应，没有变化关系，可以直接对关节坐标系轴操作就行了。但是有些场合会将坐标系原点进行偏移或者对坐标系进行旋转，这时我们需要用到 TF5110 的 Static Trafo 坐标系转化功能。
使用方法是如下：
1）在电脑上安装 TF5110, 目前测试只有 TwinCAT3 可以用, (TF5112 包含 TF5111 和 TF5110)。
2）在 nc 中添加 3 个实轴对应到驱动器中，三个虚拟轴对应笛卡尔坐标系。
3）在 nc 中添加机器人通道选择 stacic trofo，如果只有三个轴，input Axis convenient 选择 XYZ。如果四个轴，input Axis convenient 选择 XYZC



4）在 PLC 中添加 如下程序 ， 可参考前面章节的 TF5110-TF5113 文档和例子

```
VAR
    io_X                    : AXIS_REF;
    io_Y                    : AXIS_REF;
    io_Z                    : AXIS_REF;
    io_M1                   : AXIS_REF;
    io_M2                   : AXIS_REF;
```



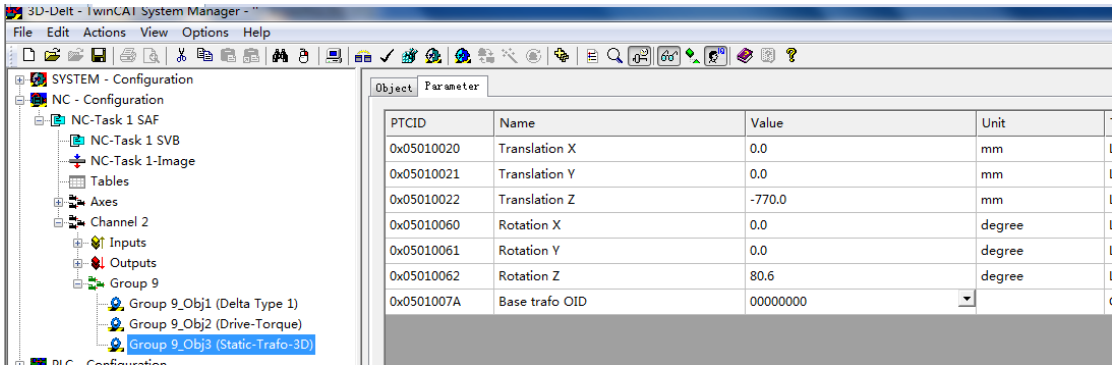
```
    io_M3                   : AXIS_REF;
    out_stPlcToKin AT %Q*   : NCTOPLC_NCICHANNEL_REF;
    out_stPlcToItp AT %Q*   : NCTOPLC_NCICHANNEL_REF;
    fbConfigKinGroup        : FB_KinConfigGroup;
    stAxesConfig            : ST_KinAxes;
    bAllAxesReady           : BOOL;
    bExecuteConfigKinGroup: BOOL;
    bUserConfigKinGroup     : BOOL;
    bUserCartesianMode      : BOOL; := TRUE;
    (*true: cartesian mode - false: direct mode (without transformation) *)
END_VAR
(* read the IDs from the cyclic axis interface so the axes can mapped later to the kinematic group
*)
stAxesConfig.nAxisIdsAcs[1] := io_M1.NcToPlc.AxisId;
stAxesConfig.nAxisIdsAcs[2] := io_M2.NcToPlc.AxisId;
stAxesConfig.nAxisIdsAcs[3] := io_M3.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[1] := io_X.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[2] := io_Y.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[3] := io_Z.NcToPlc.AxisId;

IF bAllAxesReady AND bUserConfigKinGroup THEN
    bExecuteConfigKinGroup := TRUE;
ELSE
    bExecuteConfigKinGroup := FALSE;
END_IF

fbConfigKinGroup(
    bExecute        := bExecuteConfigKinGroup ,
    bCartesianMode  := bUserCartesianMode ,
    stAxesList      := stAxesConfig,
    stKinRefIn      := in_stKinToPlc );
```

5）配置完成后可以直接对机器人的 XYZ 进行操作，相关路径规划可以采用 NCI 或者用 NC 现有功能块实现

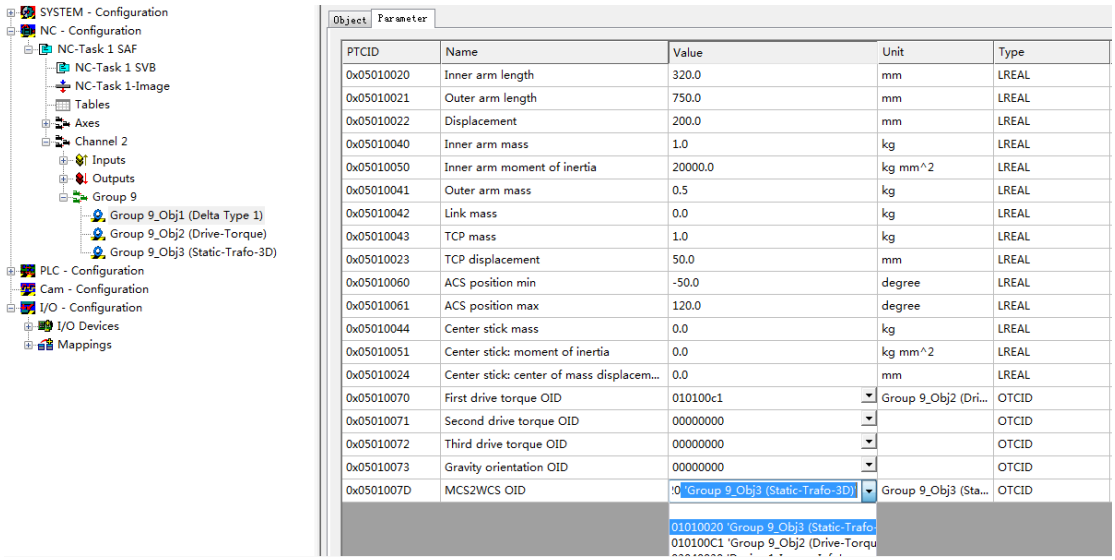
2.7.2 用于机器人坐标系的偏移和旋转

另外静态转化功能也可以在 TwinCAT2 中用于其他类型机器人的坐标系偏移和旋转，在 TwinCAT3 中该功能由 Coordinate Frame 功能取代。具体做法是。

1）在 group 中添加 Static Trafo，输入偏移和旋转值。见下图

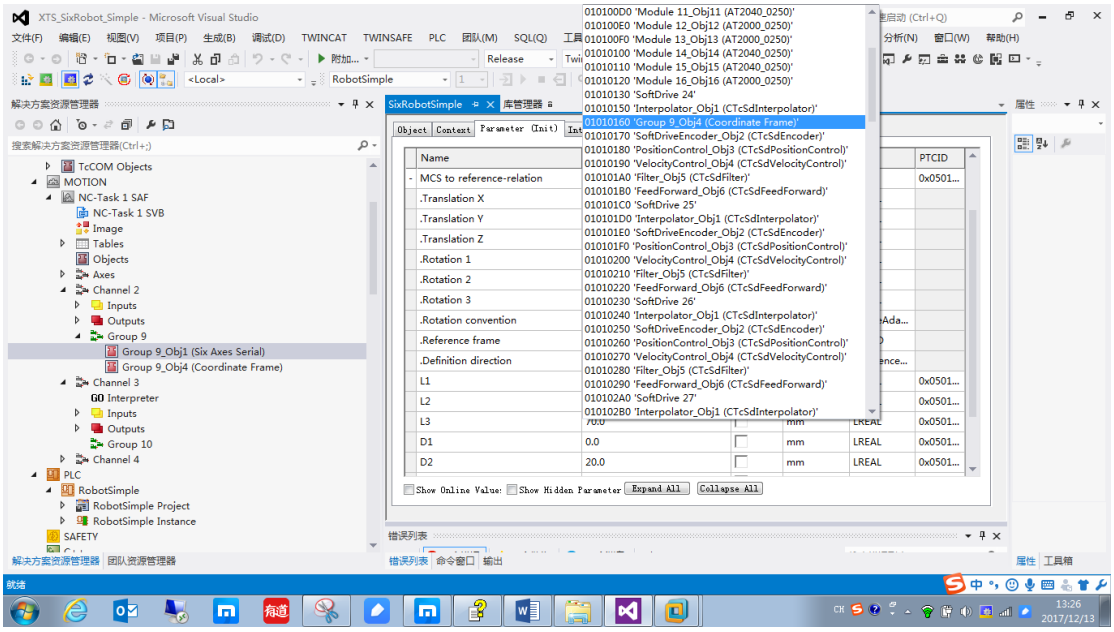


2）在机器人类型 group 参数中在 MCS2WCS OID 中选择所添加的 Static-Trafo



3）重新激活并运行，机器人坐标系会按照设置进行旋转和偏移。

4）如果是 TwinCAT3 系统，则先创建 Coordinate Frame，再在机器人类型 Group 参数 Reference Frame 中选择对应的 Coordinate Frame

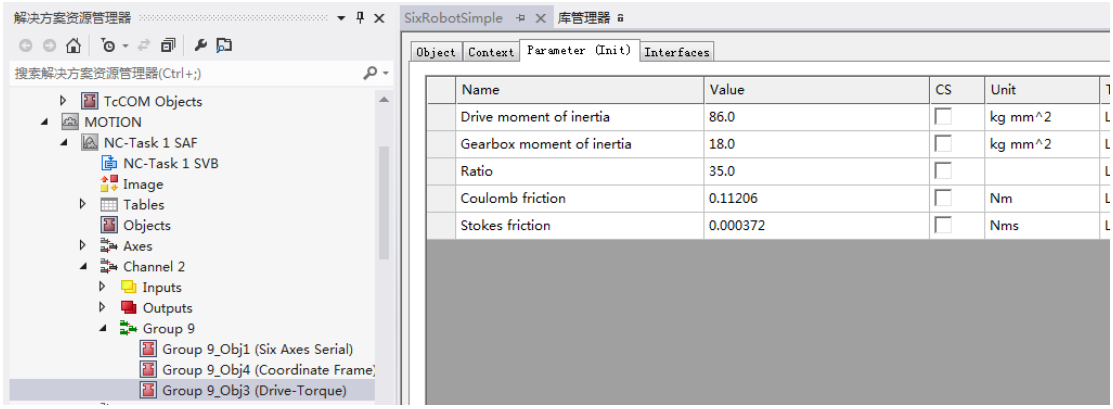


2.8 力矩前馈配置

机器人力矩前馈功能把机器人的惯量和传递效率以及摩擦力等因素用于机器人动力学模型的精确计算，得出每个对应电机的精确力矩前馈值，实现机器人的精确柔和控制。目前倍福机器人力矩前馈功能只能用于 AX5000 以及其他有力矩前馈功能的伺服驱动器。

主要配置步骤如下：

1）在机器人 Group 中添加 driver-Torque，如下图



Driver moment of inertia　　电机的转动惯量

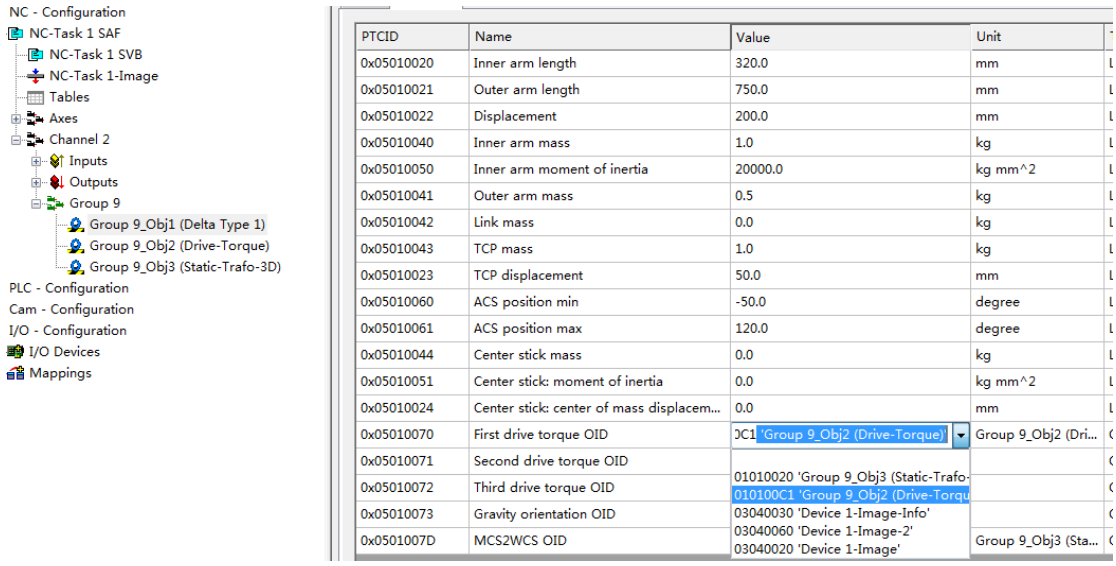
GearBox moment of inertia　　减速机的转动惯量

Ratio　　减速机减速比

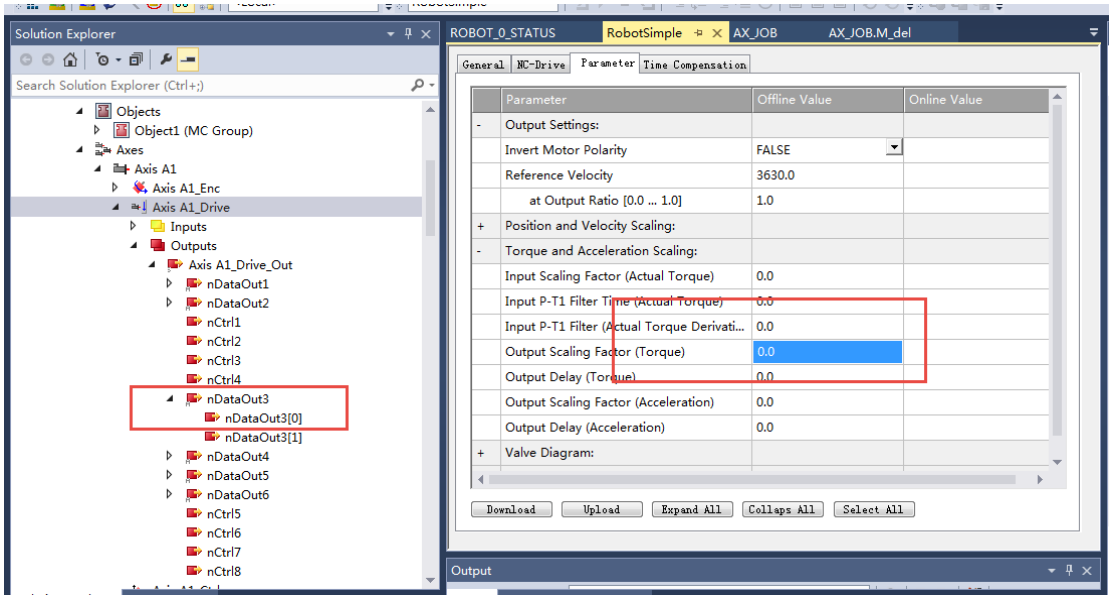
Coulomb friction　库伦摩擦力系数，静态摩擦力与压力的关系系数。可使用默认值

Stokes friction　　斯托克斯摩擦力系数，表示摩擦力系数与速度的关系。可使用默认值。

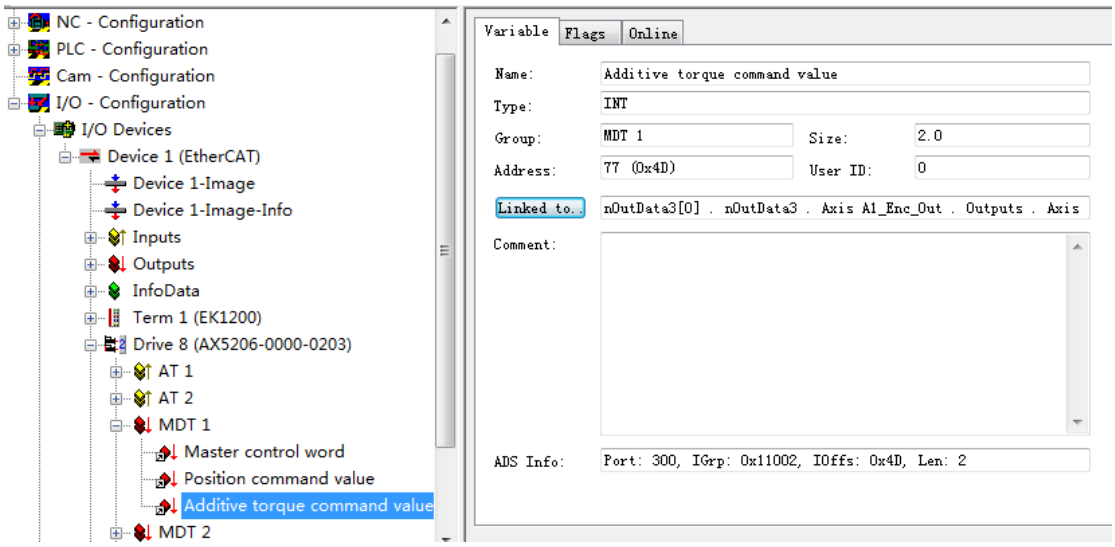
2）在机器人模型 Group 参数 Frist driver torque OID 中添加刚刚建立的 Driver Torque。同理在 Second driver torque OID 和 Third driver torque OID 也进行添加。

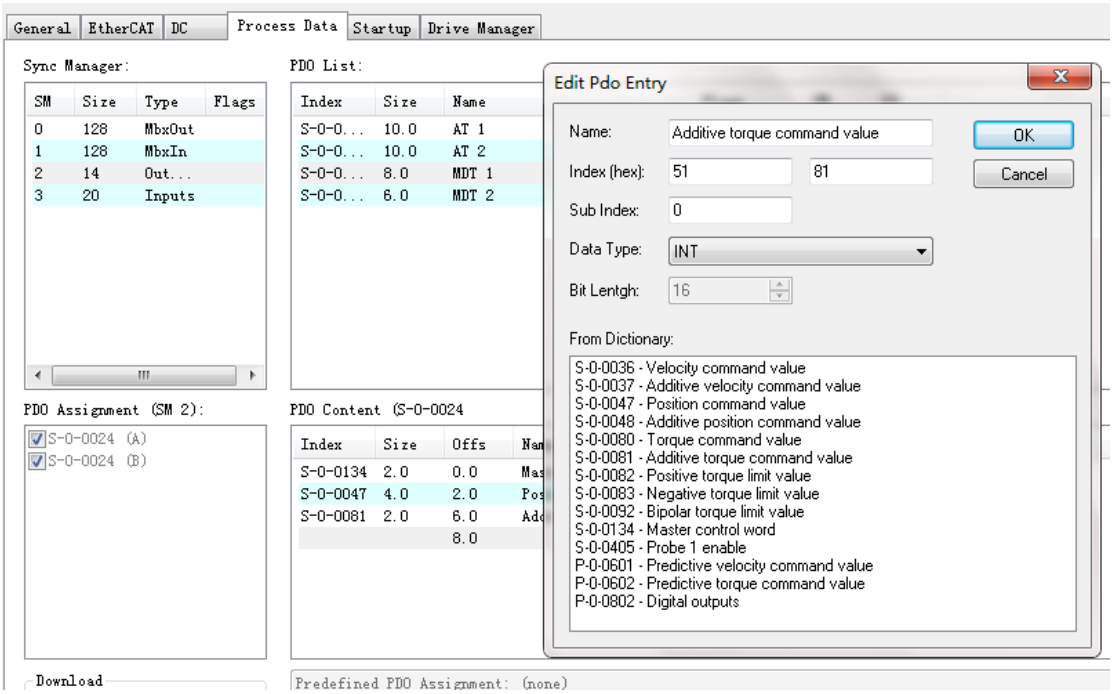


3） 在 NC 关节坐标系轴 Driver 参数中找到 Output ScalingFactor(Torque)，设置为 1。该值可设置 0 到 1 之间，根据具体情况设置。



4）把 NC 关节坐标系轴 Driver/OutPuts/nDataOut3 链接到驱动器对应 Additive torque command value（S-0-181）。如果没有这个变量需要在 Process Data MDT 中手动添加见下图。





完成以上步骤，就会发现机器人运行过程中，相应的力矩前馈值也会自动计算出来并丢给伺服驱动器。

2.9 C++编写机器人模型算法（TcCom）

我们经常会碰到机械手模型的开发，有些我们的机器人库模型里面没有标准模型，这就需要自己写机器人模型。自己写机器人模型如果只搞清楚了正逆解算法还是远远不够的，还需要一定的工程实践能力，比如说机器人初始化的标定、世界坐标系和关节坐标系单位的换算、利用外部给定功能将关节角度实时发送给电机，稍微不注意很容易出现报错、噪声过大、甚至飞车等。

结合我们 TwinCAT3 软件兼容 C++的特点，TwinCAT3 软件推出了自定义机器人模型开发功能，只需要将正逆解算法填入 C++中，生成 Object，在机器人的 Group 中调用生成的机器人 Object，填入参数，就可以在 PLC 程序中使用我们原有的机器人配置功能块，实现类似于我们标准库机器人模型一样的 PLC 编程，让编程人员只需要研究机器人的运动学和动力学算法，对于底层的机器人初始化、坐标系转换和外部给定与伺服电机的关联，底层统统自动完成，节省了开发时间，提高了效率。

本文以 SCARA 机器人为例，深入浅出介绍了在 TwinCAT3 中开发一个机器人模型过程，

没有使用我们标准机器人模型，可以作为培训文档供自定义机器人模型开发时使用。

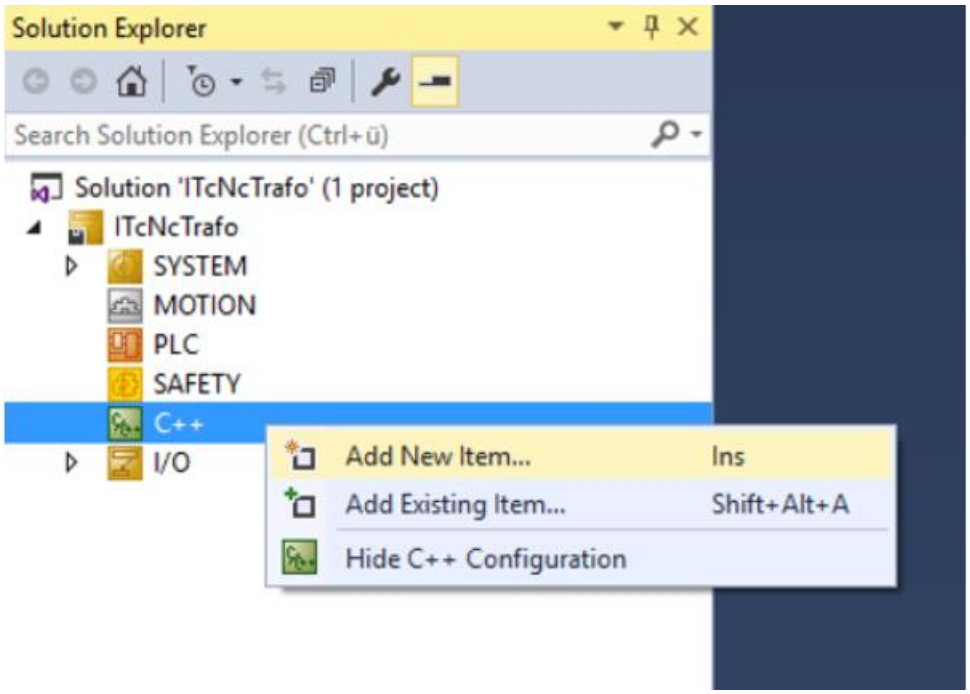
2.9.1 软件平台

需要 TwinCAT3 4024.7 以上版本。如果计算机是 64 位系统需要安装电子签名，或者使用 TC0008 工具，电子签名安装方法见《TwinCAT3_C++_Simulink 教程 杨煜敏》。

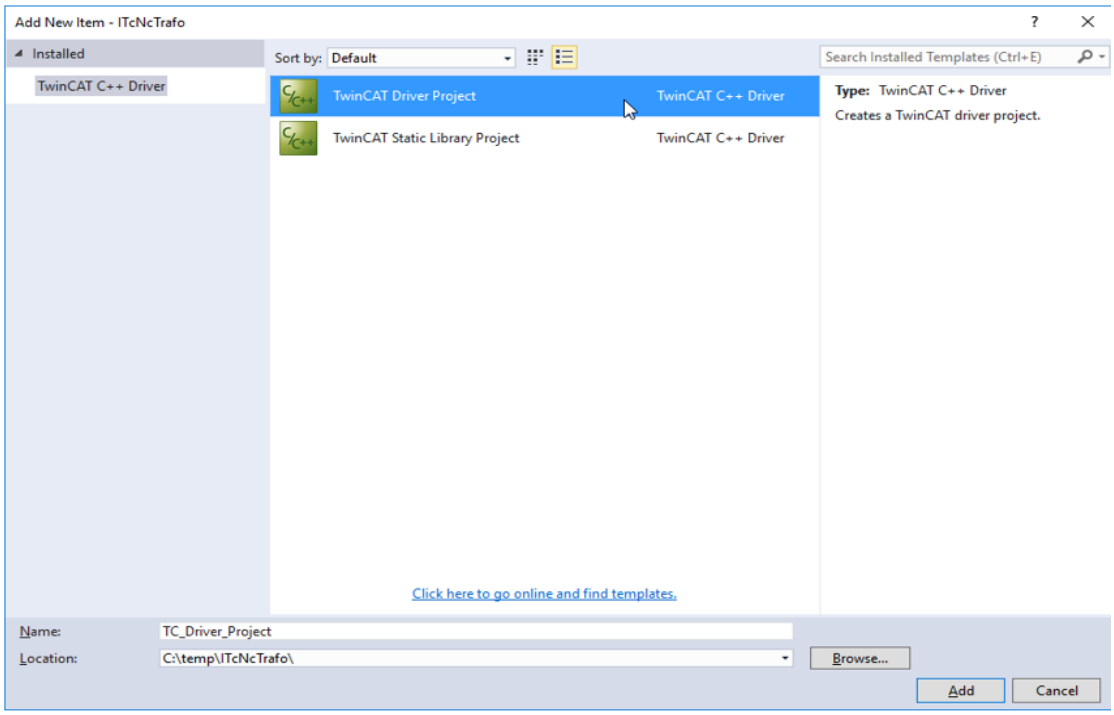
另外需要安装 TF5112 机器人软件包，可以在官网下载安装。

2.9.2 C++的配置和编程

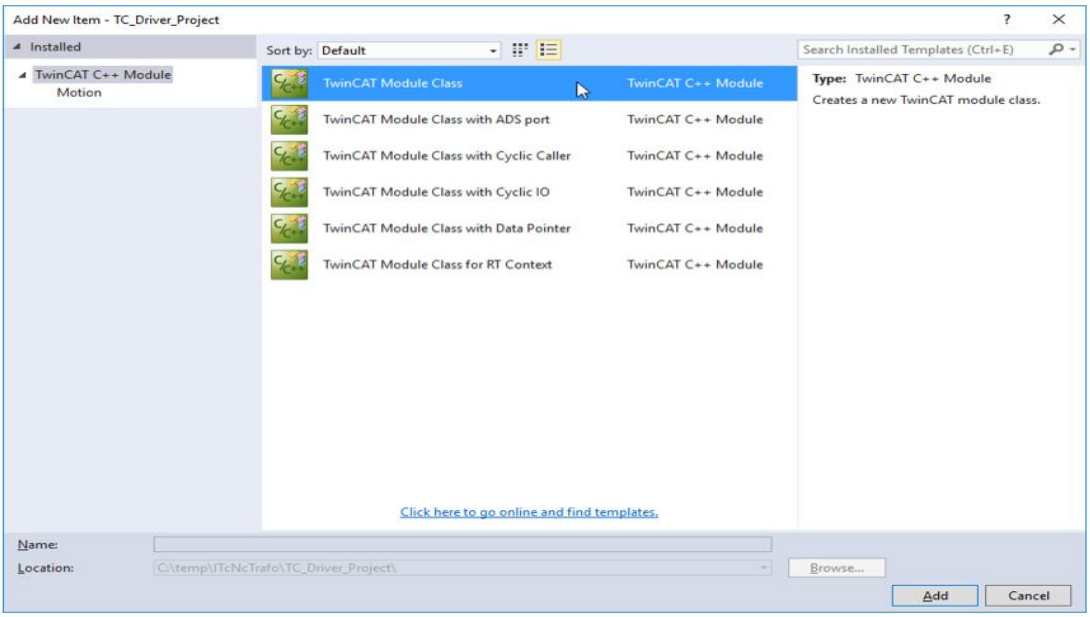
1）新建 TwinCAT 项目并添加 C++项目，如下图：



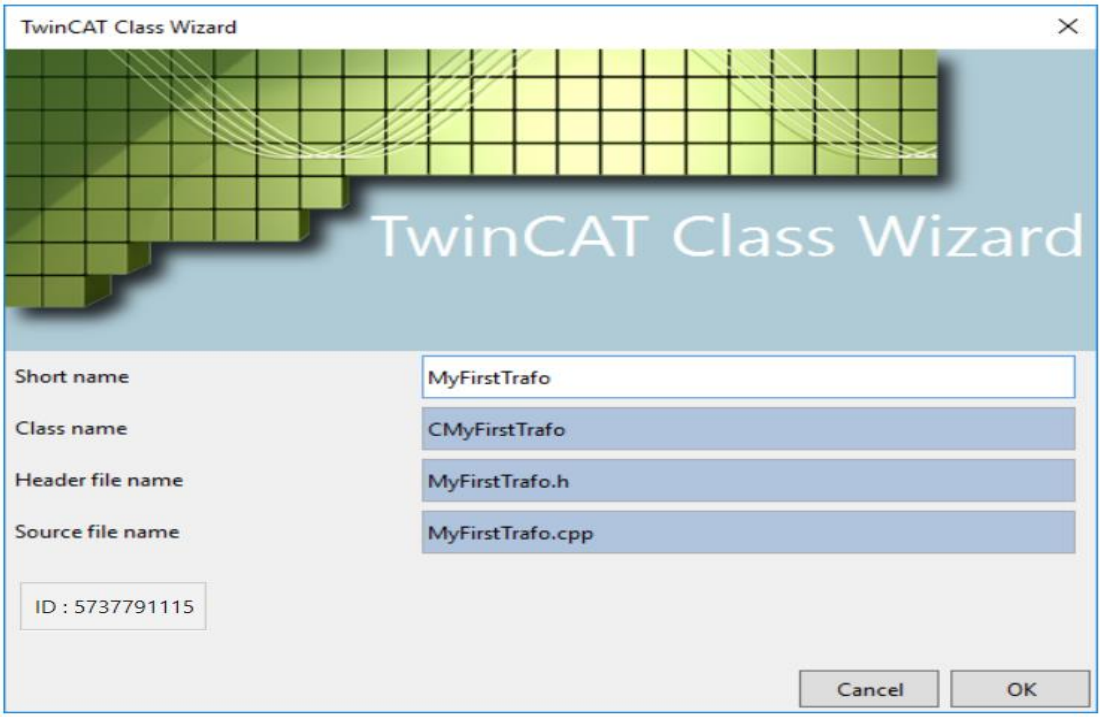
2）选择 TwinCAT3 Driver Project，如下图：



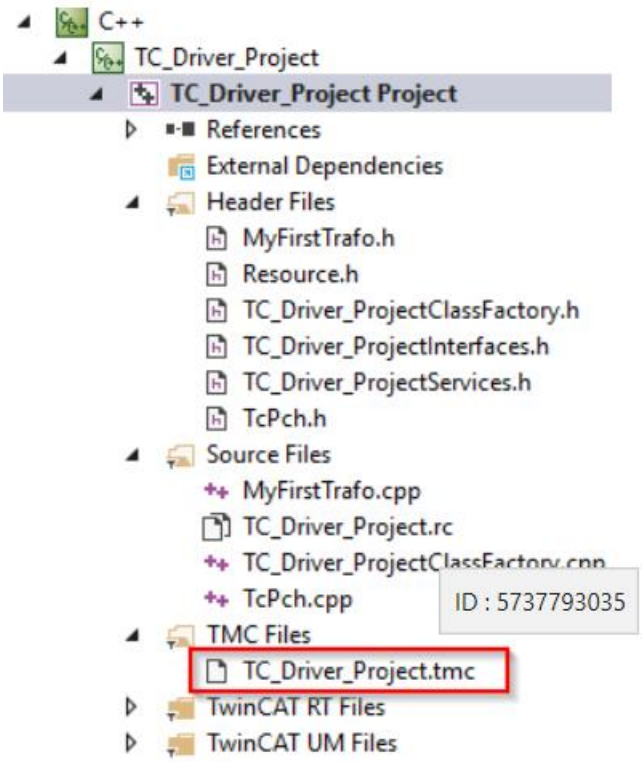
3）选择 TwinCAT Module Class



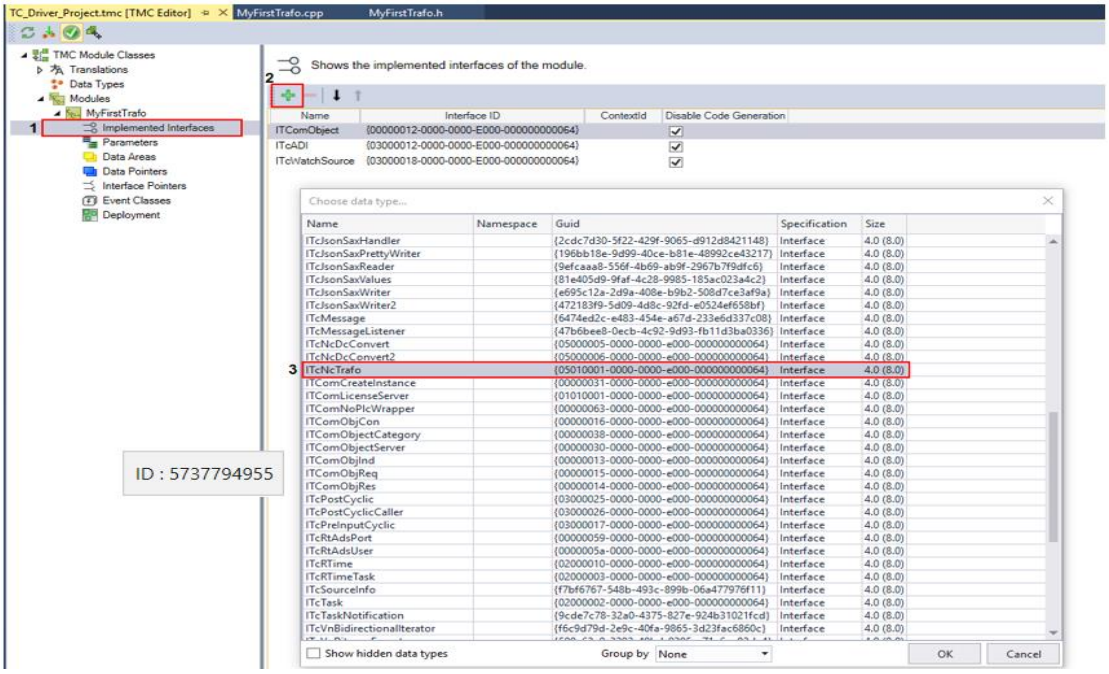
4）修改名称，如下图



5）选择 TMC Files

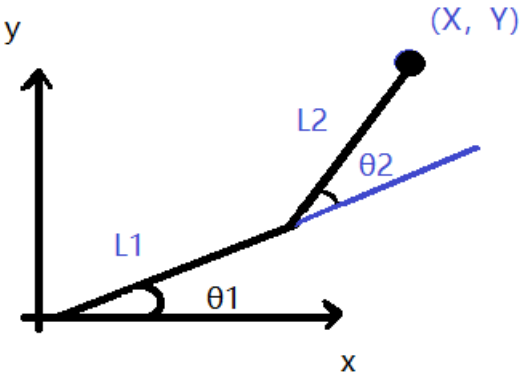


6）添加 ITcNcTrafo ， 步骤如下：



7）添加机器人模型的参数，如臂长，底端偏移等。

今天的例子我们以 SCARA 机械手为例，其 XY 平面模型如下图，暂不考虑多解情况以及 C 轴，



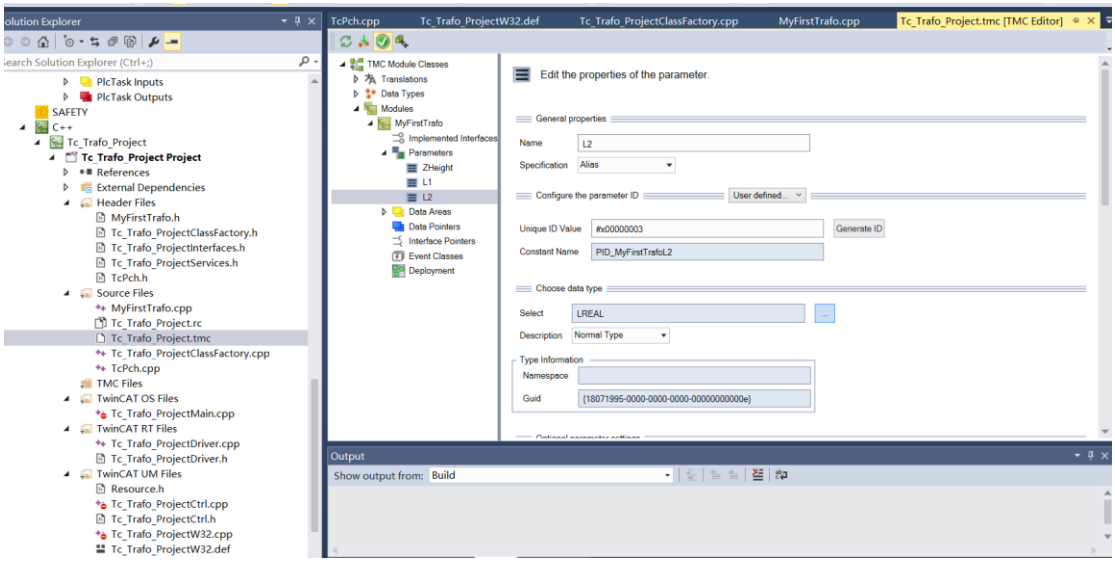
串联机器人正解很容易得到：

$$\begin{cases} X=L1\cos\theta1+L2\cos(\theta1+\theta2) \\ Y=L1\sin\theta1+L2\sin(\theta1+\theta2) \\ Z=Z'+Height \end{cases}$$

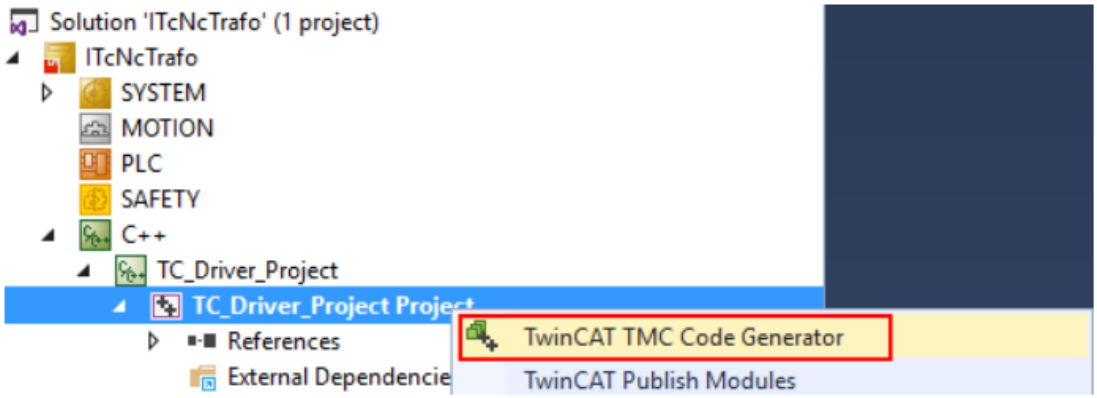
根据余弦定理也能得到他的逆解，这里我们只考虑一组解，不考虑另一组情况。

$$\begin{cases} \theta 1 = \text{arctg}\left(\dfrac{Y}{X}\right) - \arccos\left(\dfrac{x^2 + y^2 + L1^2 - L2^2}{2L1\sqrt{x^2 + y^2}}\right) \\ \theta 2 = \arccos\left(\dfrac{x^2 + y^2 - L1^2 - L2^2}{2L1L2}\right) \\ Z' = Z - Height \end{cases}$$

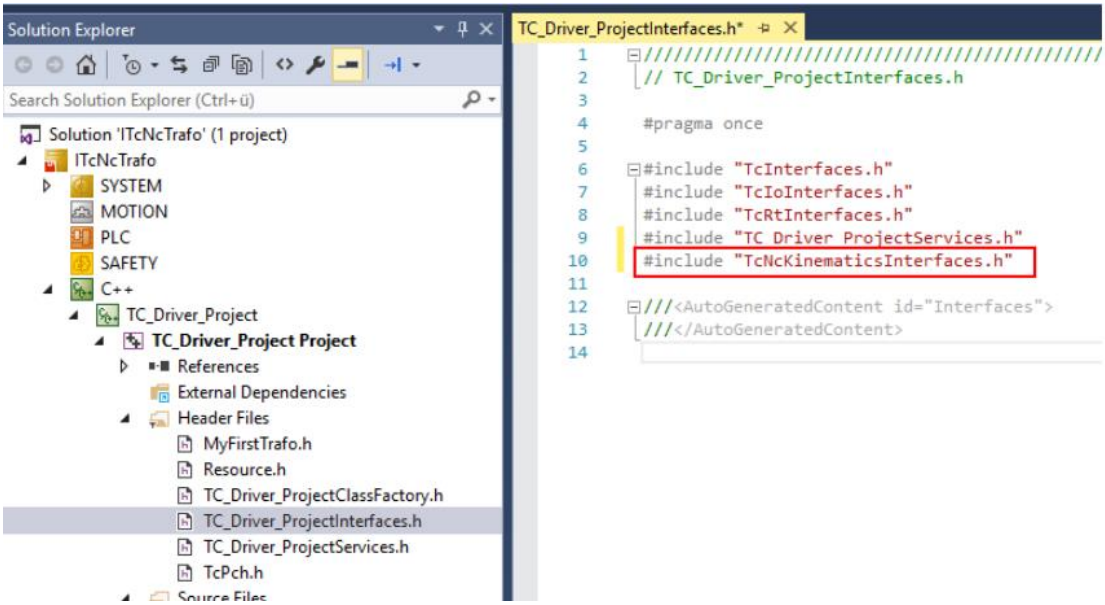
8）在 Parameter 中添加 Zheight 和 L1、L2 三个变量都是 Lreal 类型



9）运动 TwinCAT TMC Code Generator。

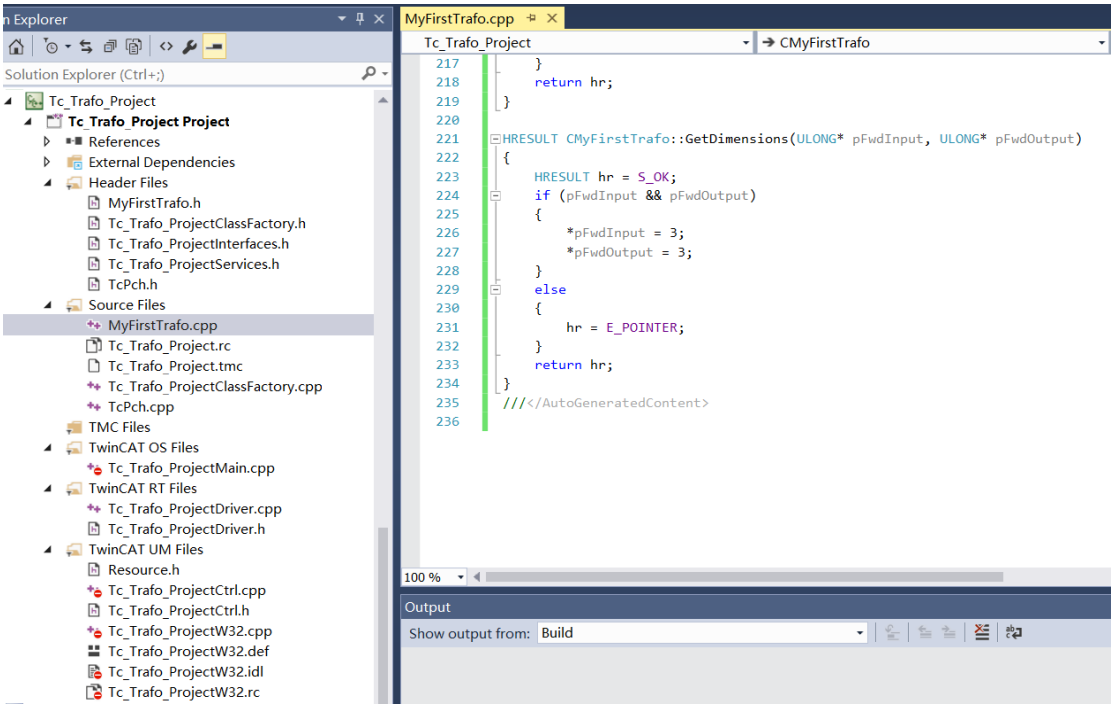


10）添加头文件 #include "TcNcKinematicsInterfaces.h"



11）在 C++程序中定义关节轴数量 pFwdInPut 和笛卡尔坐标系轴数量 pFwdOutPut

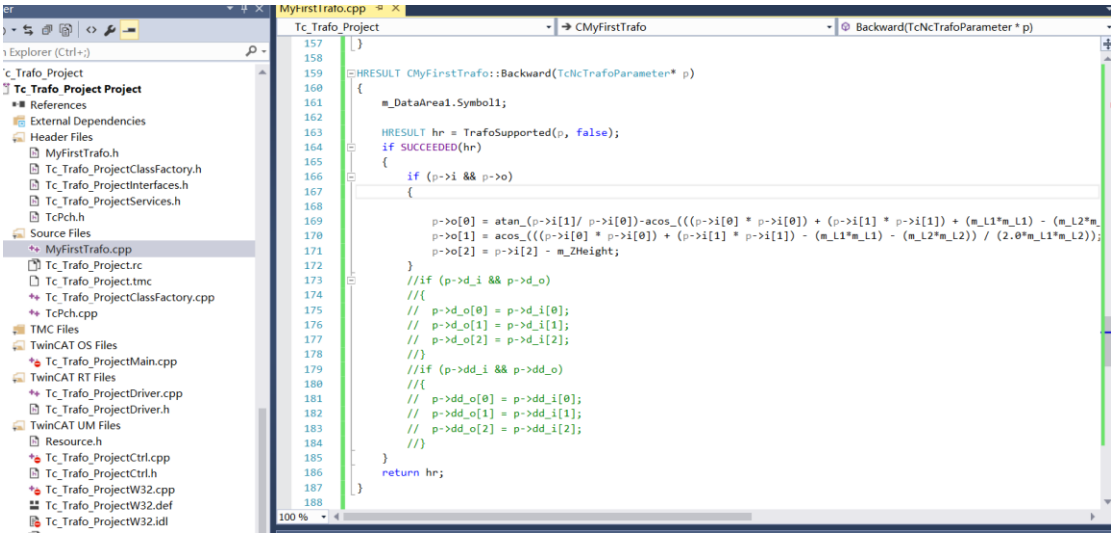
本例中 SCARA 机器人各有三个轴。

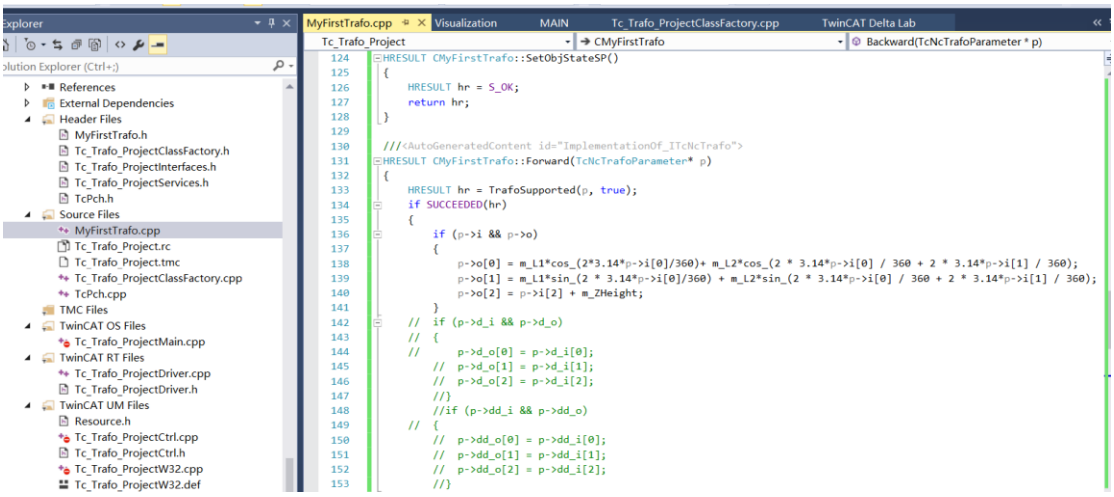


```
HRESULT CMyFirstTrafo::TrafoSupported(TcNcTrafoParameter* p, bool fwd)
{
    HRESULT hr = S_OK;
    if (p)
    {
        if (fwd)
        {
            if (p->dim_i != 3 || p->dim_o != 3)

            {
                hr = MAKE_ADS_HRESULT(NCERR_KINTRAFO_INVALIDDIM);
            }
        }
        else
        {
            if (p->dim_i != 3 || p->dim_o != 3)

            {
                hr = MAKE_ADS_HRESULT(NCERR_KINTRAFO_INVALIDDIM);
            }



        }
    }
```

12）根据公式编写正逆解程序，本例程序未对机器人进行速度和加速度分析，所以只填入位置的对应公式，屏蔽了速度和加速度的对应公式。在实际应用中需要结合实际情况选择合适的控制方式。

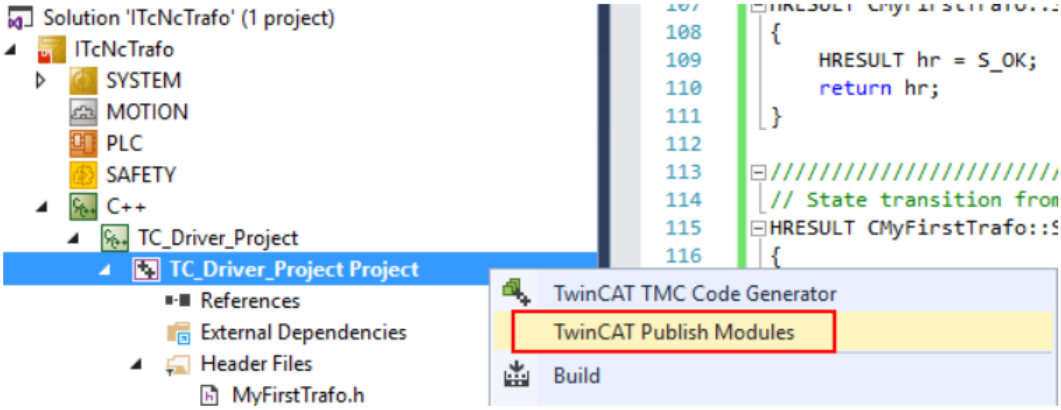
注意 C++中三角函数是以弧度 rad 为单位，如果关节电机是角度需要换算。





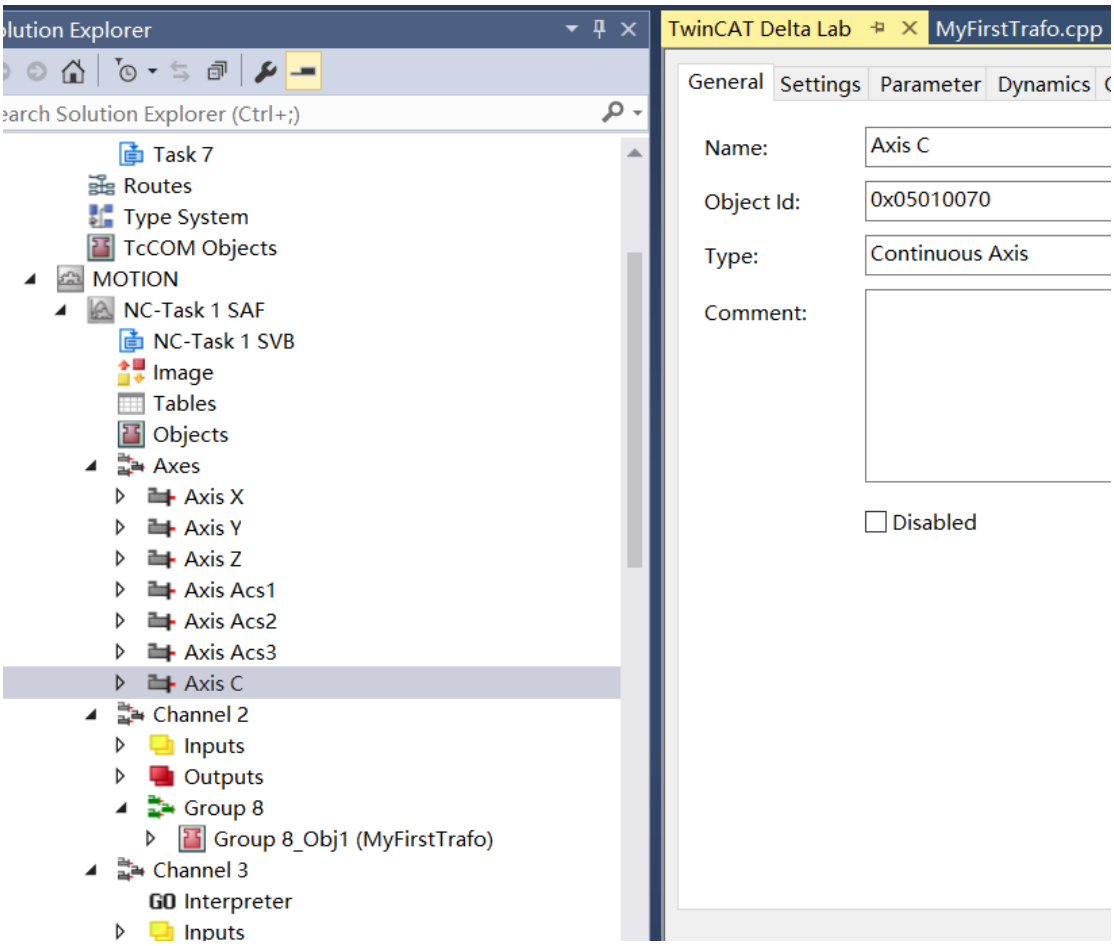
13）运行 TwinCAT publish Modules 生成 object 以供调用，至此已完成 C++的配置

- Build and publish the modules.

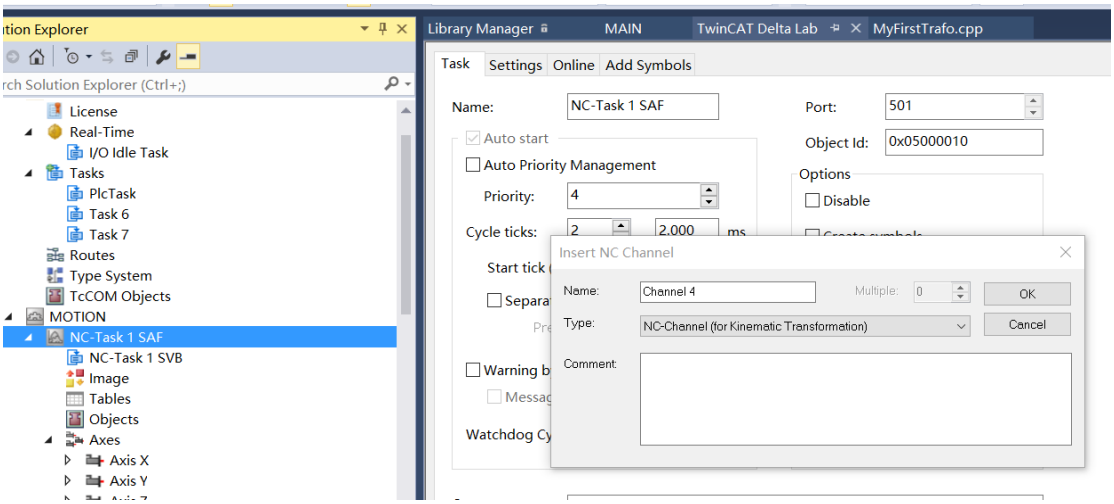


2.9.3 PLC 程序的配置和编程

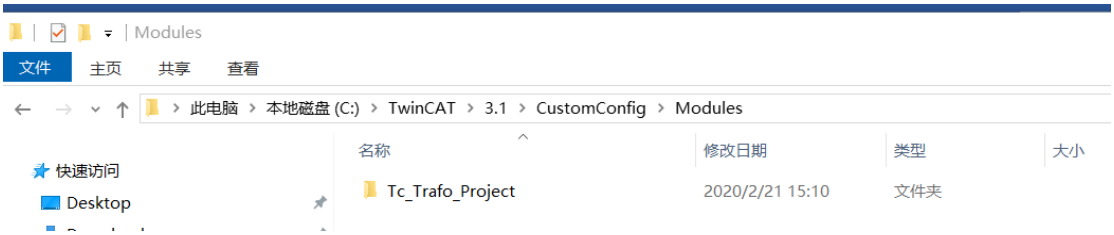
1）首先在 NC 中添加 6 个轴，包含三个笛卡尔坐标系轴 XYZ 和三个关节坐标系轴，

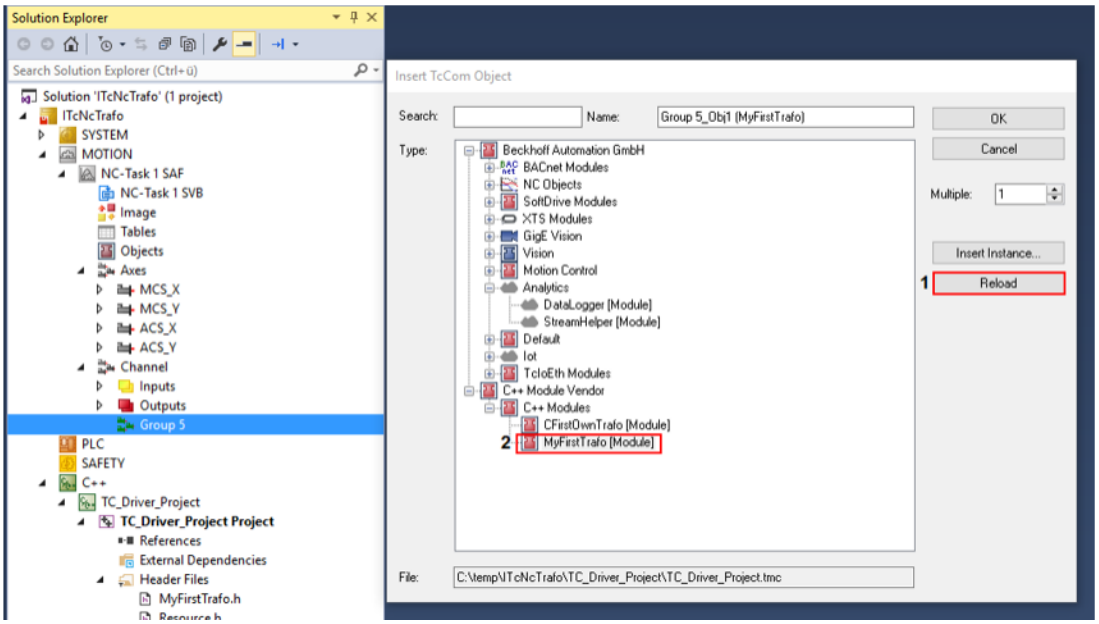


2) 在 Motion 中选择插入 NC 通道，类型选择 Kinematic Transformation

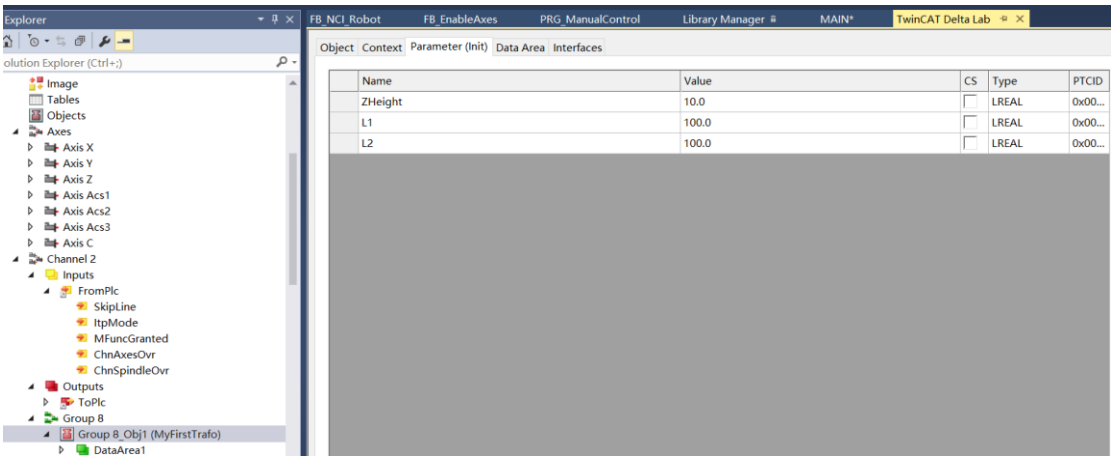


3) 在新添加的 Channel 的 Group 中选择我们刚审查 C++Moudle，如下图。
其中 C++Moudle 默认保存放在下图文件夹。

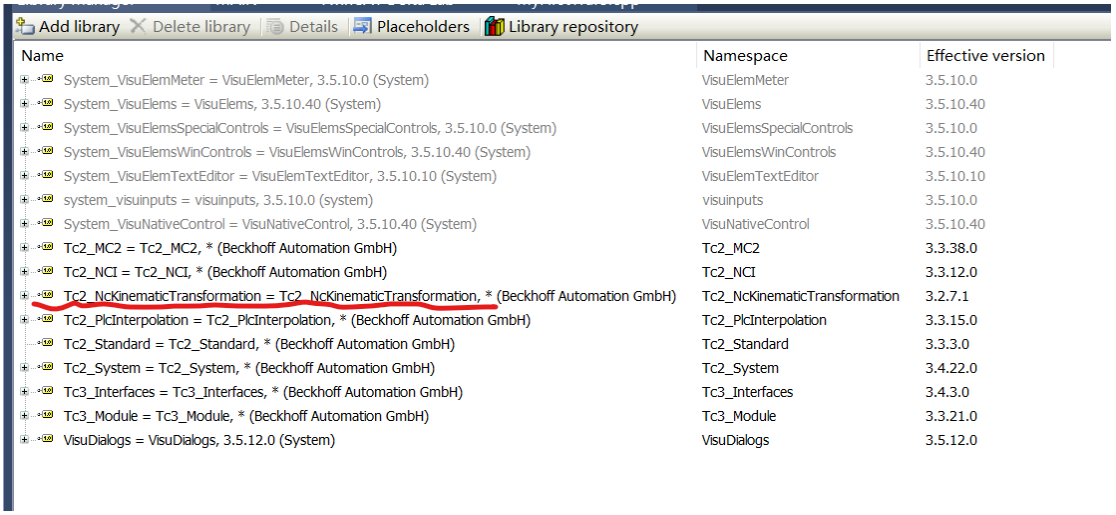




4) 在机器人的 Group 的参数中，输入对应的机器人本体参数



5) 添加机器人库 TC2_NCKinematicTransformation,(可到官网下载 TF5110 安装包，安装后直接生成)



6) PLC 程序中首先对 NC 的六个轴进行声明和使能，另外添加机器人结构体类型

NCI_Robot | FB_EnableAxes | PRG_ManualControl | Library Manager | MAIN*

```
PROGRAM MAIN
VAR
    io_X                        : AXIS_REF;
    io_Y                        : AXIS_REF;
    io_Z                        : AXIS_REF;
    io_M1                       : AXIS_REF;
    io_M2                       : AXIS_REF;
    io_M3                       : AXIS_REF;
    io_C                        : AXIS_REF;

    in_stKinToPlc   AT %I*      : NcToPlc_NciChannel_Ref;
    out_stPlcToKin  AT %Q*      : PLCTONC_NCICHANNEL_REF;
    nChnId                      : UDINT := 0;

    stAxesConfig                : ST_KinAxes;
    fbConfigKinGroup            : FB_KinConfigGroup;
    fbResetKinGroup             : FB_KinResetGroup;
```

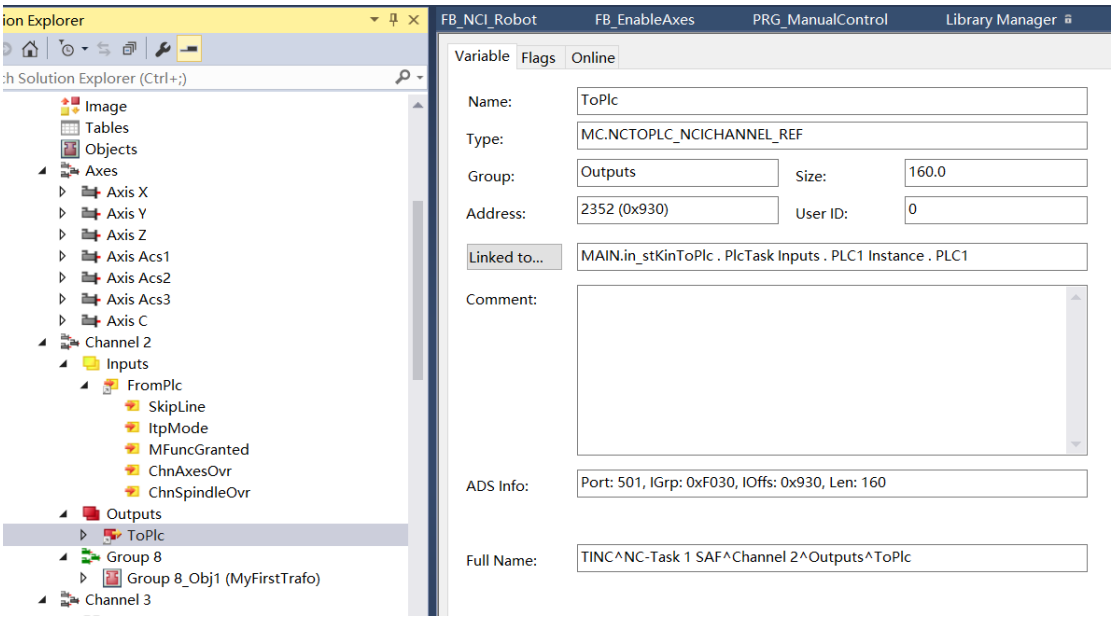
7) 把关节坐标系轴和机器人坐标系轴的 ID 分配给 ST_KinAxes

```
// Before building the kin group, take care that M1..M3 are moved to a possible position.
// A position of 0 would represent a cable length of 0mm & is not possible. 600 could be a good start

// Read axes id's from cylic interface
// so they can be mapped later to the kinematic group
stAxesConfig.nAxisIdsAcs[1]     := io_M1.NcToPlc.AxisId;
stAxesConfig.nAxisIdsAcs[2]     := io_M2.NcToPlc.AxisId;
stAxesConfig.nAxisIdsAcs[3]     := io_M3.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[1]     := io_X.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[2]     := io_Y.NcToPlc.AxisId;
stAxesConfig.nAxisIdsMcs[3]     := io_Z.NcToPlc.AxisId;
```

8) 通过 FB_KinConfigGroup 控制机器人在笛卡尔坐标系和关节坐标系的切换，
当 bCartesianMode 为 True 时，触发 FB_KinConfigGroup 的 bExecute 信号，机器人切换到笛卡尔坐标系；当 bCartesianMode 为 False 时，触发 FB_KinConfigGroup 的 bExecute 信号，机器人切换到关节坐标系。当机器人报错时可通过 FB _KinResetGroup 对机器人进行复位

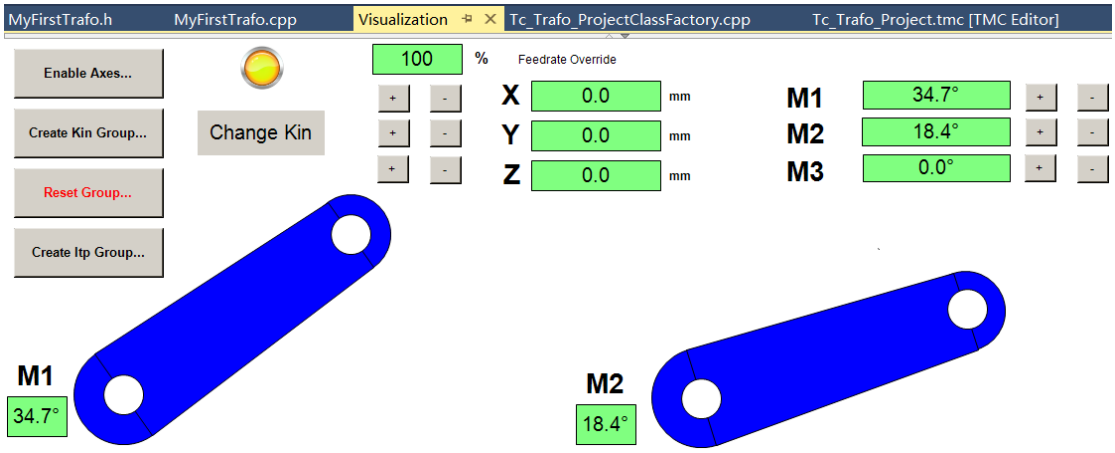
```
fbConfigKinGroup(
    bExecute                    := xUserConfigGroup,
    bCartesianMode                      := xUserCartesianMode,
    stAxesList                  := stAxesConfig,
    stKinRefIn                  := in_stKinToPlc);

fbResetKinGroup(
    bExecute                    := xUserResetKinGroup,
    nItpChannelId                   := nChnId,
    stKinRefIn                  := in_stKinToPlc,
    stAxesList                  := stAxesConfig);
```

9) 完成 PLC 程序的编写后，将各轴变量以及机器人变量与 NC 中配置进行链接

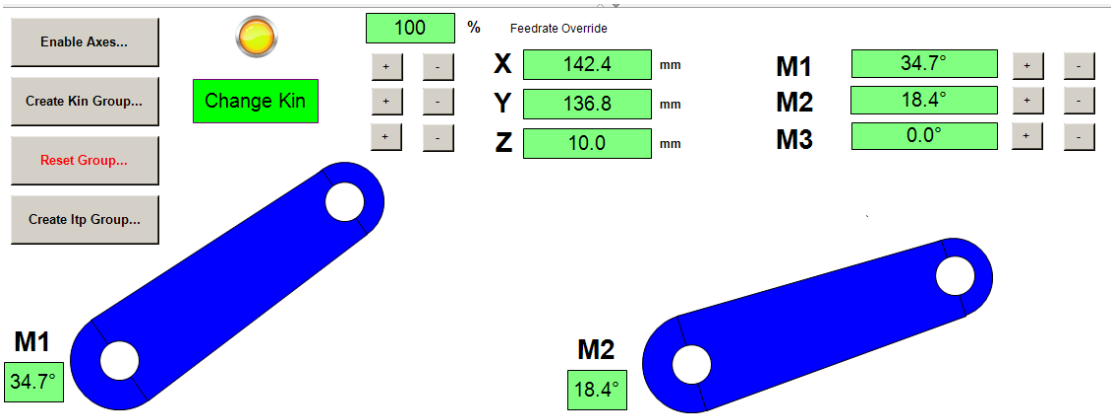


2.9.4 运行和测试

1) 运行 PLC 程序，当各轴使能完成后，可以将关节坐标系轴适当点动一个位置，如 M1=15.7；M2=16.3；避免初始奇异点状态。



2) 这时可以将机器人切换到笛卡尔坐标系模式，可以调用 TC_MC2 中的功能块直接运行笛卡尔坐标系轴。



3).如果要实现 XYZ 轴的插补可以，可以把笛卡尔坐标系轴配置到 NCI 插补通道中实现。方法参考第一章。

第三章 PickAndPlace 抓放功能库

TF5420 TC3 Motion Pick-And-Place 可以执行多维的空间插补运动。其功能与 NCI 是类似的，但是其编程放在 TwinCAT3 的 PLC 中进行。相比较与 NCI，PickAndPlace 专门用来做机械手的抓放运动，在一些高速、高精度场合，优先选择 PickAndPlace 功能。

3.1 软件安装和购买

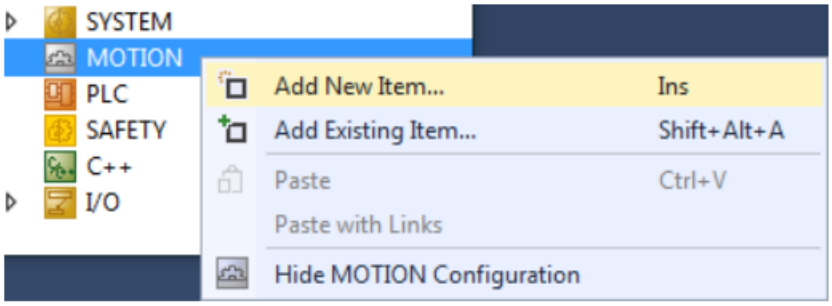
PickAndPlace 抓放功能库需要 NCI 以上级别控制器，且只能用在 TC3 中。在购买授权时首先添加 NCI 授权（TF5100），再添加 TF5420。购买了 NCI 之后 TF5420 是免费使用的。同时需要注意目前 PickAndPlace 抓放功能库不支持 CE 系统。

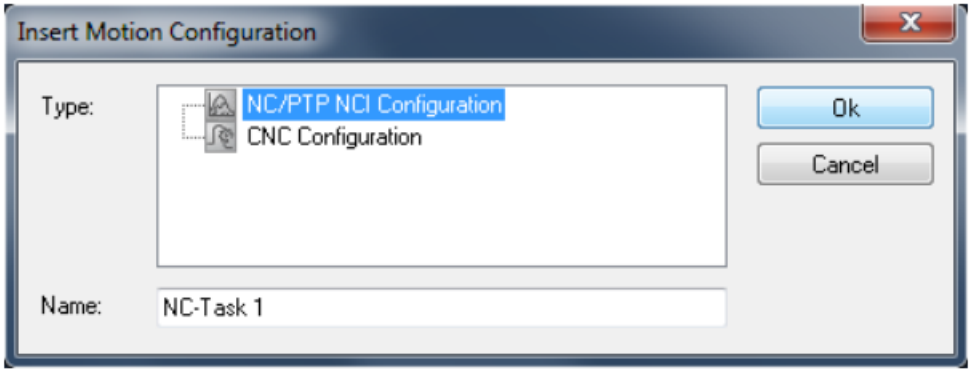
在本例中实验可以在官网下载 TF5400 安装包，安装并激活授权供使用。

3.2 软件配置

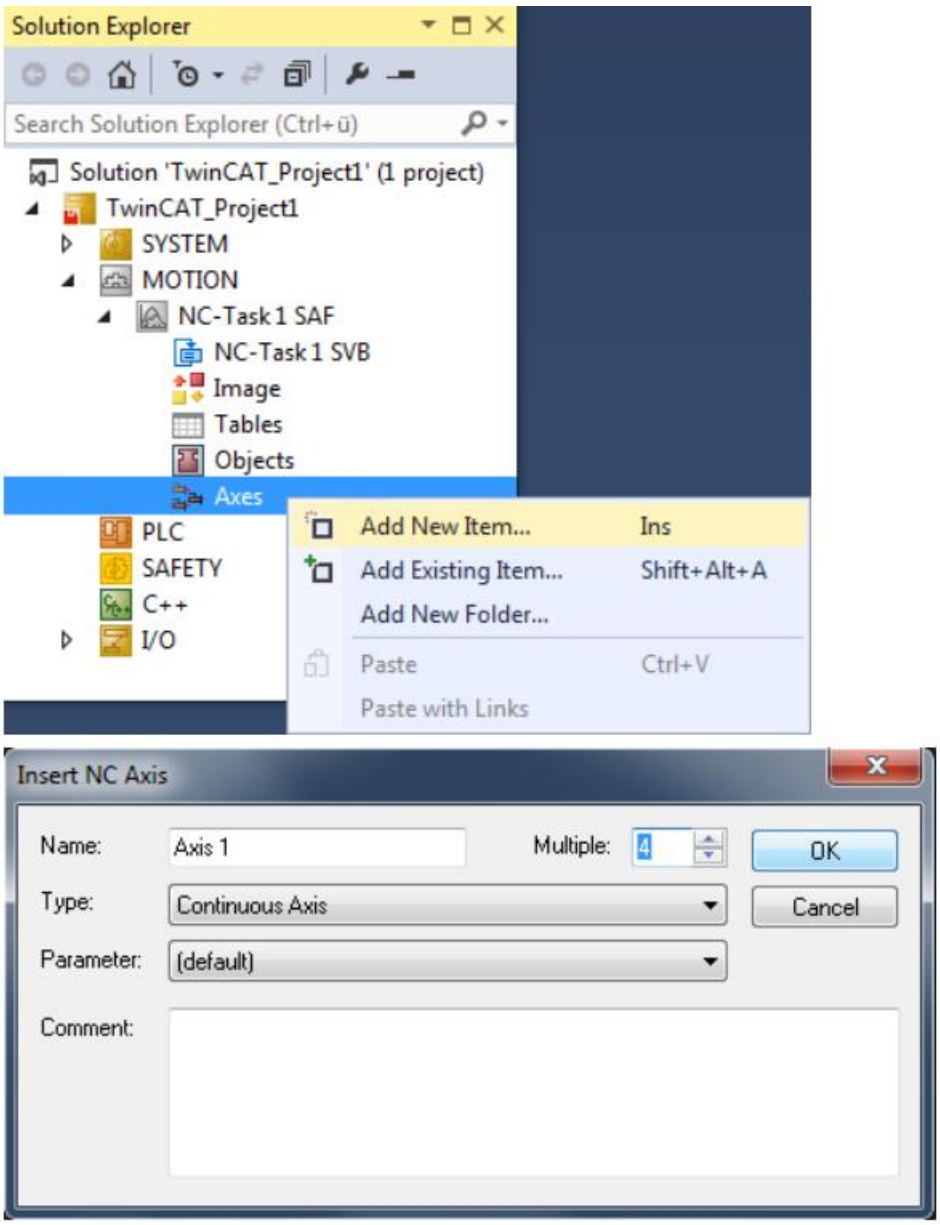
3.2.1 配置 MC_Group

1）在 Motion 中添加“NC/PTP NCI Congfiguration”

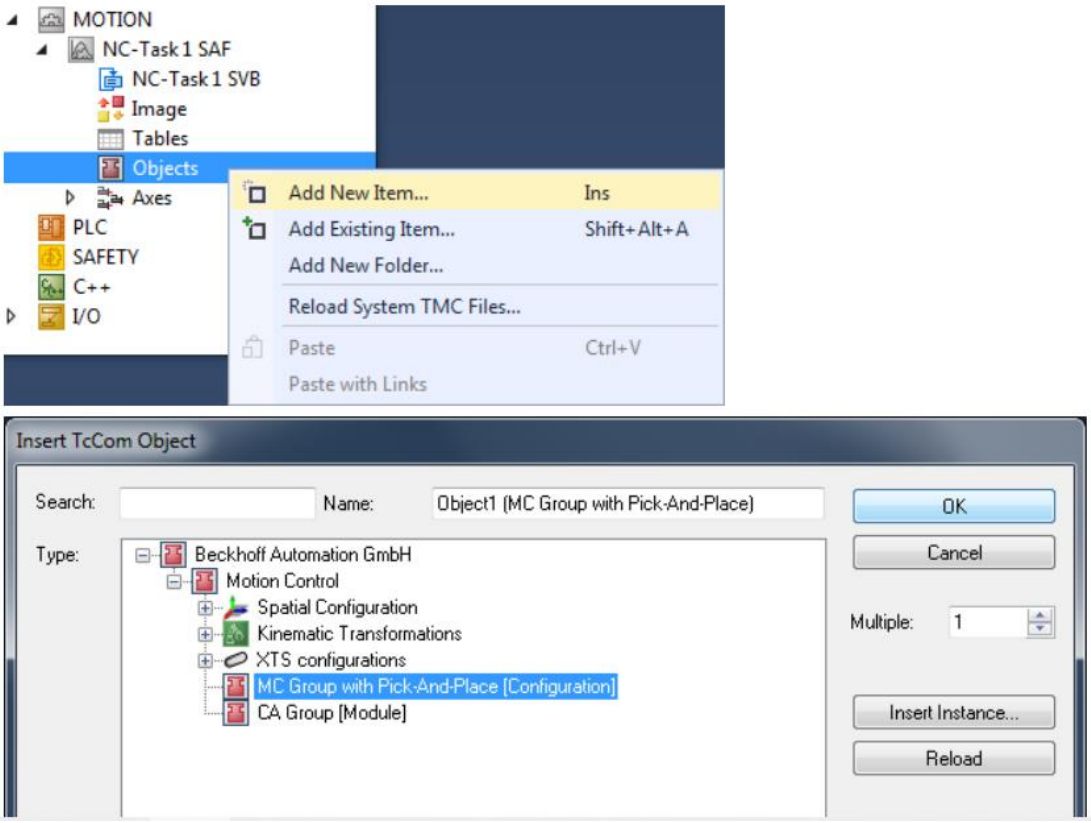




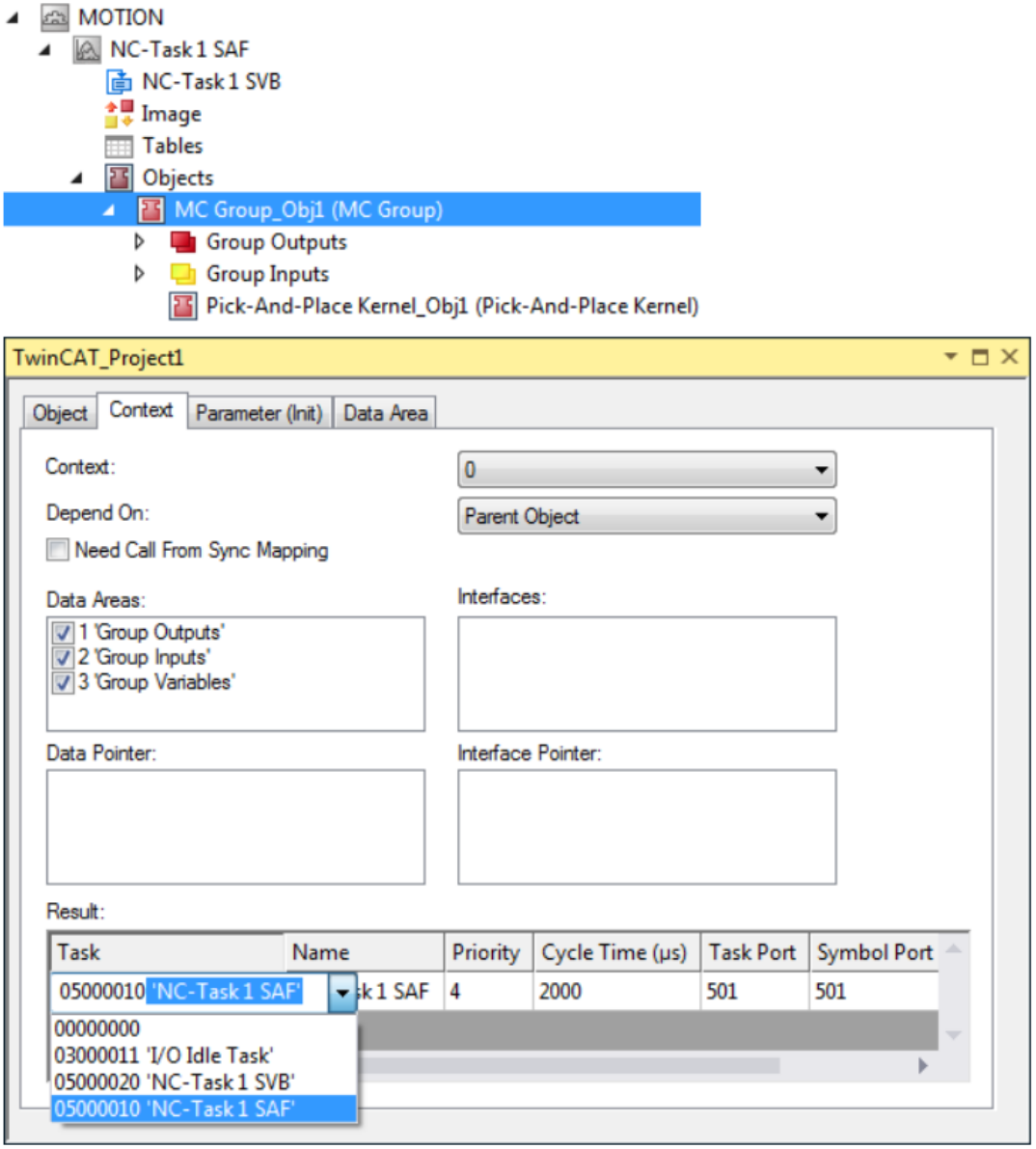
2）在 NC 配置中添加 4 个轴(本例以 XYZC 为例)



3）添加 PickAndPlace Group



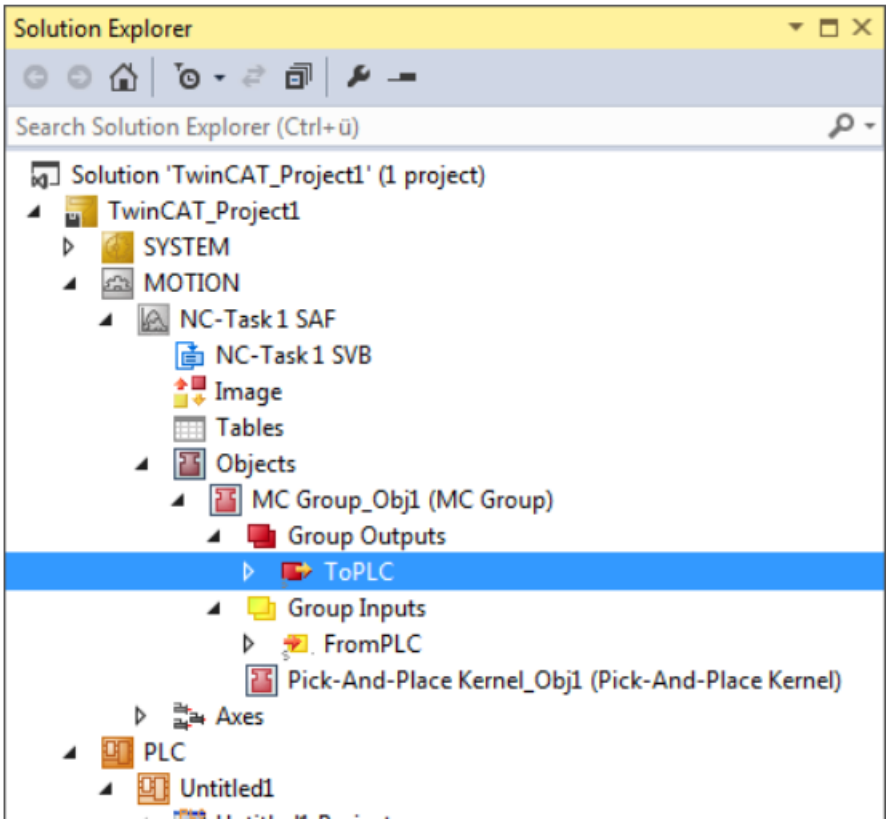
4）检查 Group 中的执行任务，在“NC-Task 1 SAF”



5）与 PLC 进行变量链接

需要在 PLC 中定义 AXES_GROUP_REF 结构体类型

```
VAR
    stGroupRef : AXES_GROUP_REF;
END_VAR
```

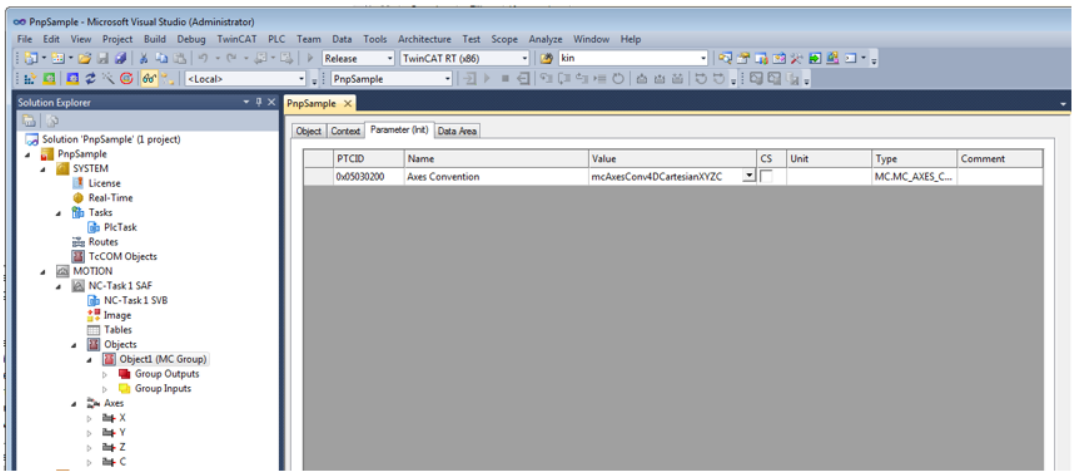


3.2.2 Group 参数设置

1）Axis Conventions

Axis Conventions 轴约定

根据应用需求配置 Group 参数



有三种类型的约定可供设置，分别对应 2D、3D 和 4D，根据用到的插补轴数量进行选择。

2） NET Cycle Time Divisor

提高由于离散化的时间导致的精度。

3）Time Override Ramp Time

Override 突变后的加减速时间

3.3 PLC 编程和功能块

1）Group 结构体类型

stGroupRef : AXES_GROUP_REF;

AXES_GROUP_REF 包含了 NcToPlc 和 PlcToNc 两个类型，需要与上一节添加的 Group 做变量链接。

2）MC_AddAxisToGroup

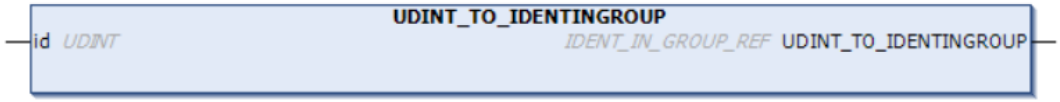
功能块 MC_AddAxisToGroup 是把轴添加到 Group 中。

AxesGroup 对应上文提到的定义的 AXES_GROUP_REF 类型;

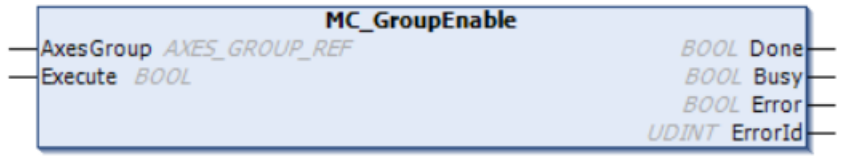
Axis 是要添加轴变量

Execute 上升沿触发

IdentInGroup 是为该轴指定一个身份标识，可以不填。如果要使用可以通过 UDINT_TO_IDENTINGROUP 函数，如下图。



3）MC_GroupEnable

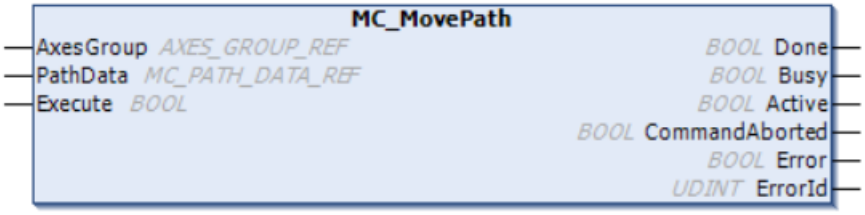


MC_GroupEnable 是对 Group 进行使能，上升沿触发;

AxesGroup 对应上文提到的定义的 AXES_GROUP_REF 类型;

Execute 上升沿触发;

4）MC_MovePath



MC_MovePath 用于触发执行一个曲线运动，曲线运动的特征点数据放在 PathData 中。

AxesGroup 对应上文提到的定义的 AXES_GROUP_REF 类型;

PathData 路径特征点数组，可以做以下定义:

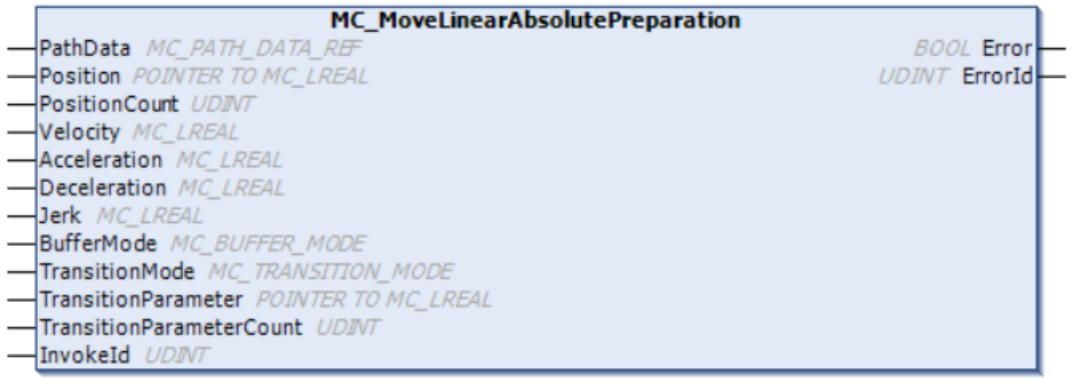
buffer : ARRAY[1..4096] OF BYTE;

path : MC_PATH_DATA_REF(ADR(buffer), SIZEOF(buffer));

注意：如果要删除数组中的所有点可以使用 Path.ClearPath()。不要在 MC_PATH_DATA_REF 数据类型上使用 MEMCMP、MEMCPY、MEMSET 或 MEMMOVE 等内存函数。

Execute 上升沿触发

5）MC_MoveLinearAbsolutePreparation



MC_MoveLinearAbsolutePreparation 功能块将绝对直线运动的特征点添加到结构 PathData 中的段表中。创建表之后，可以通过 MC_MovePath 执行它。每个循环周期可以多次调用该功能块。

PathData： 路径特征点数组，见上文。

Positon：位置数组的指针地址。如有四个轴配置到 Group，则可设置一个 4 维的位置数组，把这个数组的地址赋值给 Position，如下文声明和程序。

```
aTargetPos : ARRAY[1..4] OF MC_LREAL;
fbMoveLinPrep.Position := ADR(aTargetPos); //pointer to position array
```

PositionCount：配置到 Group 中的轴的数量。

Velocity：曲线的运行的最大速度。需要设置>0。

Acceleration：最大加速度。

Deceleration：最大减速度。

Jerk：加加速度。

BufferMode：放弃当前运动或者重置目标位置时的曲线转换方式，可参考 NCPTP 的 BufferMode。

TransitionMode：曲线的过渡转换方式

TransitionMode : MC_TRANSITION_MODE := mcTransModeNone;

曲线的转换方式有以下六种类型，但主要使用 mcTransModeNone 和 mcTransModeCornerDistanceAdvanced 。

```
TYPE MC_TRANSITION_MODE :
(
   mcTransModeNone                  := 16#1000,
   mcTransModeStartVelocity         := 16#1001,
   mcTransModeConstantVelocity      := 16#1002,
   mcTransModeCornerDistance        := 16#1003,
   mcTransModeMaxCornerDeviation    := 16#1004,
   mcTransModeCornerDistanceAdvanced := 16#100A
);
```

END_TYPE

mcTransModeNone　　不执行混合过渡，在过渡段停止。

mcTransModeCornerDistanceAdvanced

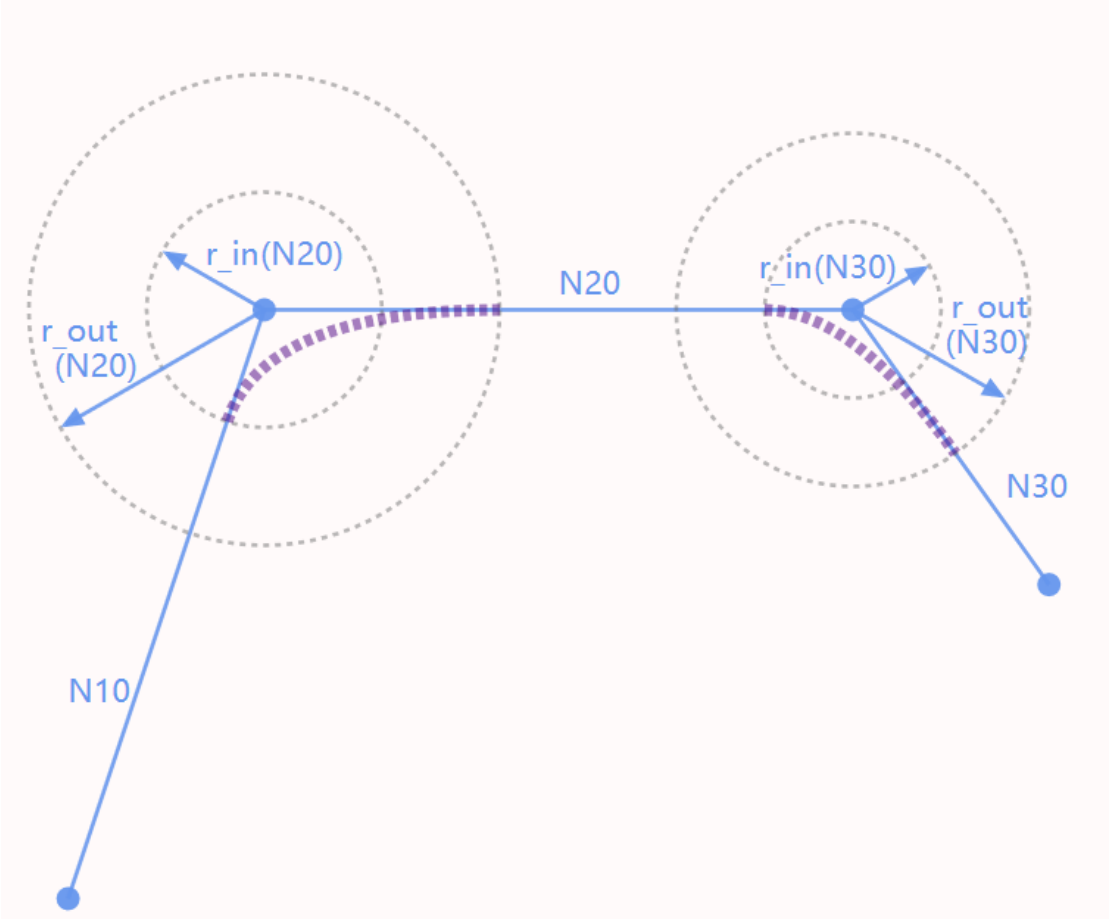
在两段直线之间执行混合运动。过渡参数采用不遵循程序路径的容差球。

第一个参数描述了开始混合的前一段的半径(r_in)。

第二个参数描述了下一段(r_out)上的半径，它定义了一个位置，确保完成了混合。

参数 r_out 是一个最大值。

混合可以在到达 r_out 之前结束。



TransitionParameter ： 曲线的过渡转换参数的指针

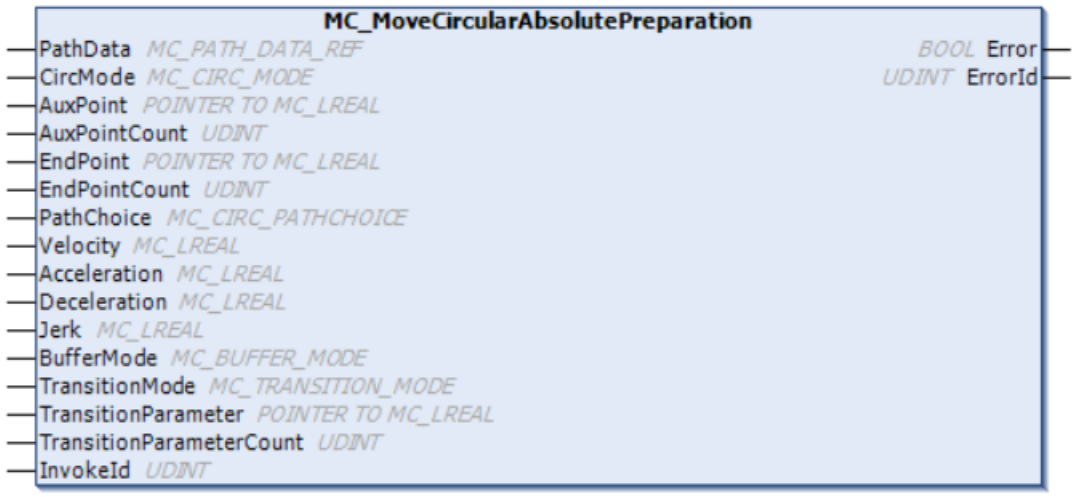
如一个曲线在直线转直选过渡时需要定义一个有两个元素的数组，对应上文提到的 r_in 和 r_out，并把该数组的指针给此输入变量。如下文：

```
aTransitionParam  : ARRAY[1..2] OF MC_LREAL;
fbMoveLinPrep.TransitionParameter  := ADR (aTransitionParam); //pointer to transition
aTransitionParam[1]    := 50; //blending distance on previous segment, no influence here
aTransitionParam[2]    := 100; //blending distance on this segment, no influence here
```

TransitionParameterCount　曲线的过渡转换参数的数量。

InvokedID 过渡点的 ID，用于程序调试分析

6)　MC_MoveCircularAbsolutePreparation

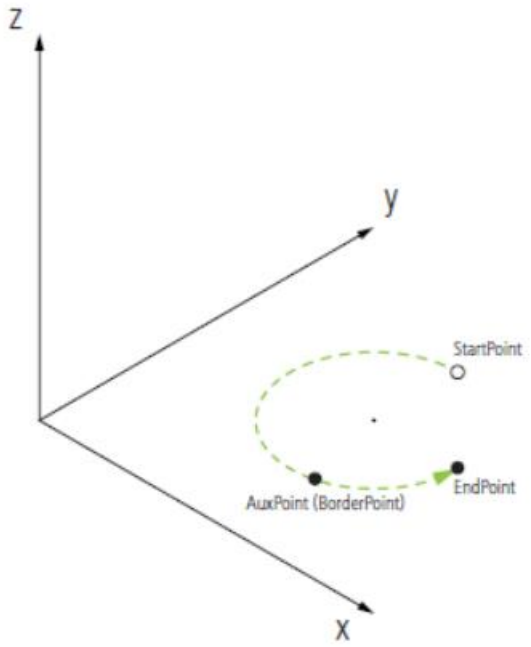


CircMode: 指定圆弧运动的类型

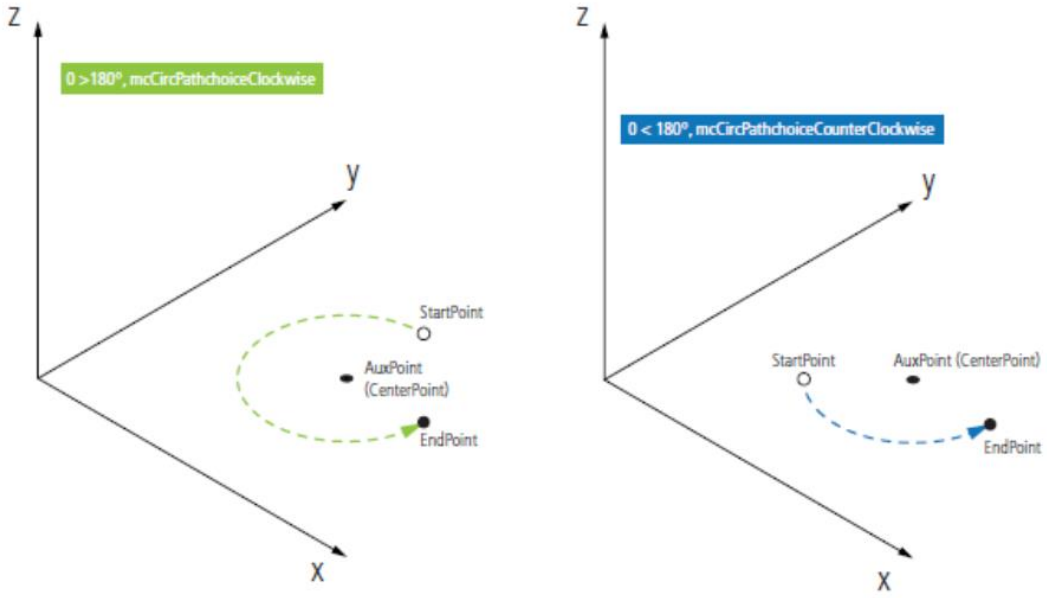
主要有以下三种类型:

mcCircModeInvalid 参数无效，返回一个错误。

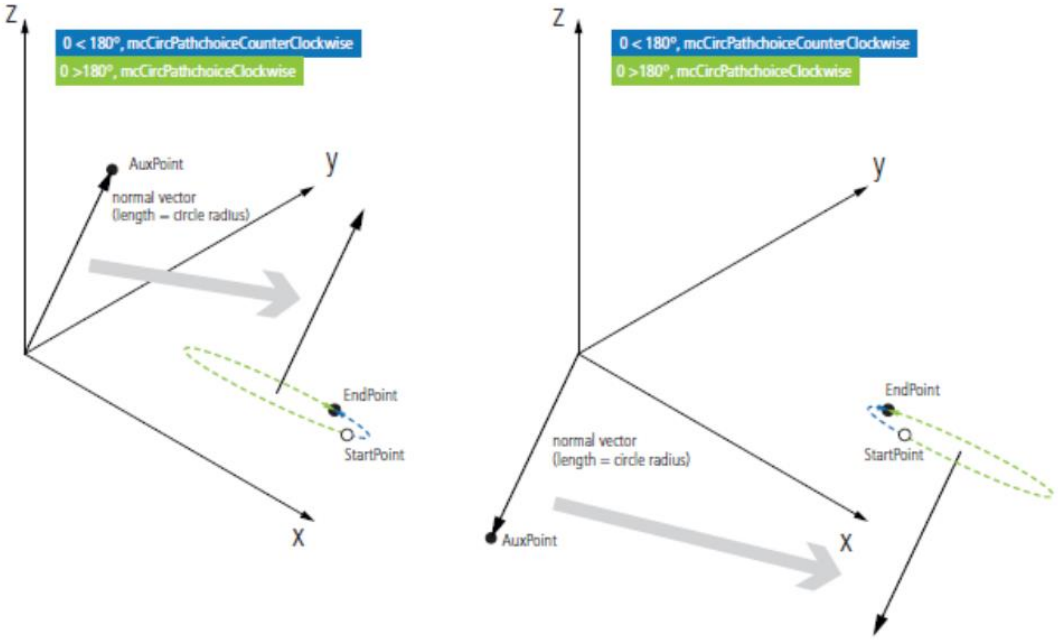
mcCircModeBorder 指定三点生成一个圆弧。其中辅助点在圆上，可以通过 PathChoice 选择顺时针或者逆时针。



mcCircModeCenter 指定三点生成一个圆弧。其中辅助点是圆心，可以通过 PathChoice 选择顺时针或者逆时针。



mcCircModeRadius 辅助点作为矢量，作为圆弧面的法线方向。



AuxPoint:辅助点的坐标数组指针，同常是(x, y, z)。可以把该数组的地址赋值给**AuxPoint。**

AuxPointCount: 坐标数组的元素数量。

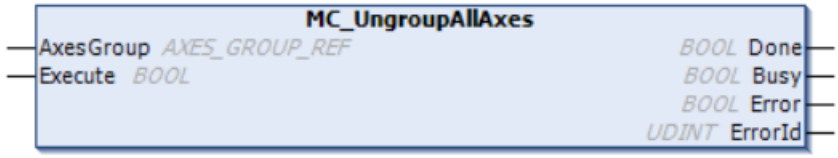
EndPoint: 圆弧的结束点。

EndPointCount: 结束点坐标数组的元素数量。

PathChoice: 选择圆弧以顺时针运行或者逆时针运行。

其他参数和含义与 MC_MoveLinearAbsolutePreparation 功能块的参数相同，请参数上一节。

7）MC_UngroupAllAxes

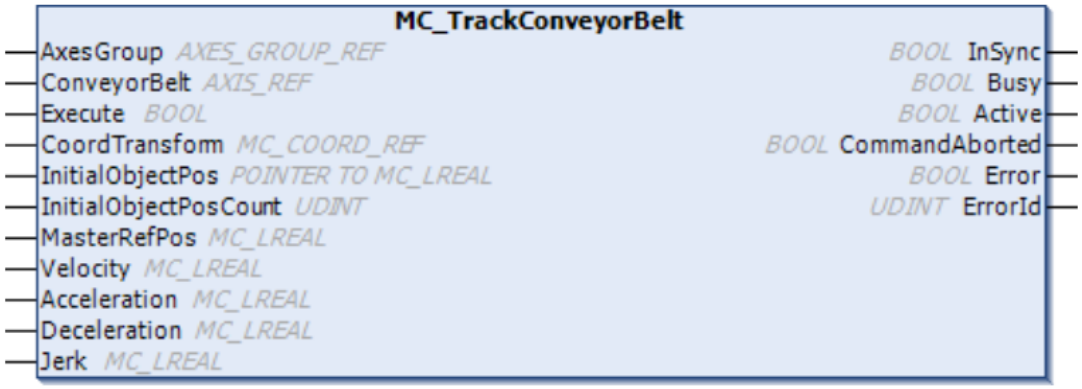


MC_UngroupAllAxes 对应 MC_GroupEnable 功能块。在需要解除 Group 耦合时使用；
AxesGroup 对应上文提到的定义的 AXES_GROUP_REF 类型；
Execute 上升沿触发；

8） MC_GroupStop

MC_GroupStop 是在运行过程中以指定减速度进行停止。
AxesGroup 对应上文提到的定义的 AXES_GROUP_REF 类型；
Execute 上升沿触发；
Deceleration：最大减速度。
Jerk：加加速度。

3.4 传送带跟随



TwinCAT 的 PickAndPlace 功能可以直接通过 MC_TrackConveyorBelt 功能块实现与视觉的配合完成传动带上的抓放功能。与 FlysingSaw 相比不需要使用额外的虚拟轴，更加简单方便。该功能块仍然需要与 MC_MovePath 配合使用。当 InSync 信号与 MC_MovePath 的 Done 都出来时，完成追踪抓取。

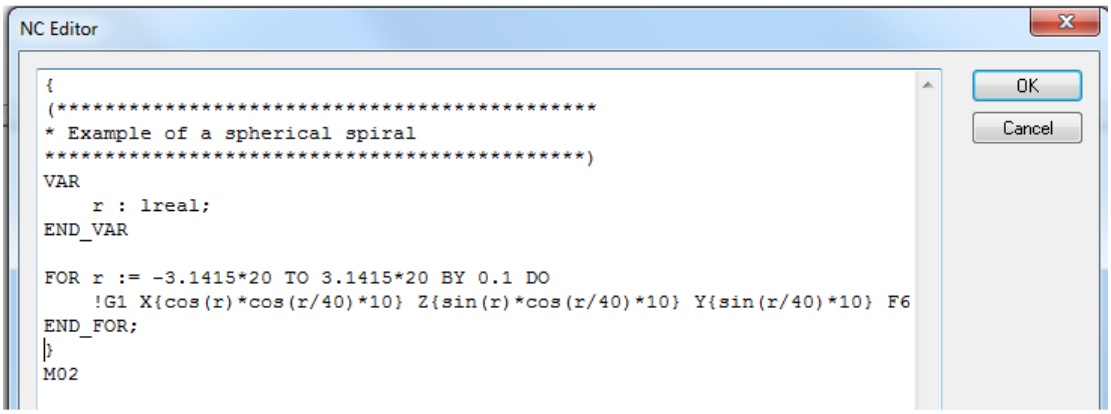
```
IF  MC_TrackConveyorBelt.InSync  AND  MC_MovePath.Done  THEN
//position synchronisation is reached
END_IF
```

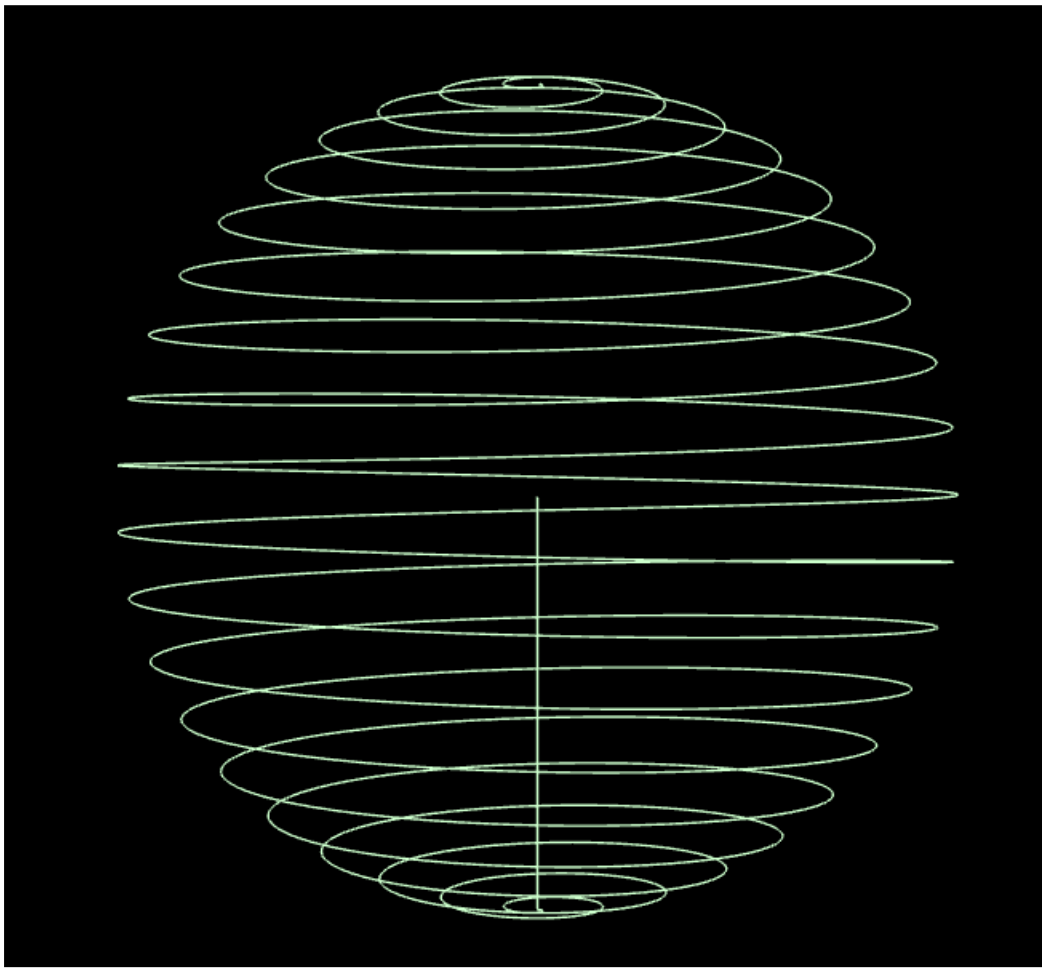
AxesGroup 对应上文提到的定义的 AXES_GROUP_REF 类型。
ConveryorBelt 传送带轴。
Execute 上升沿触发。
CoordTransform 机械手所在的坐标系，参考第二章。
InitalObjectPos 位置数组的指针地址。该位置是指由视觉传递过来的数据。
InitalObjectposCount 位置数组的元素数量。
MasterRefPos 在触发相机拍照瞬间的皮带位置值。
Acceleration：最大加速度。
Deceleration：最大减速度。
Jerk：加加速度。

第四章 NCI 的 GST 语言编程

GST 语言把高级语言完美地集成在 CNC 编程中，它将描述轮廓的 G 代码(DIN66025)与 PLC 编程用的结构文本 ST 语言结合在一起，使机器制造商能以一个简单的方式实现复杂的编程。GST 使 NCI 可以方便地实现迭代，变量声明，功能与功能块等。GST 功能已经在 TC 3.1.4022.0 及以上版本发布。

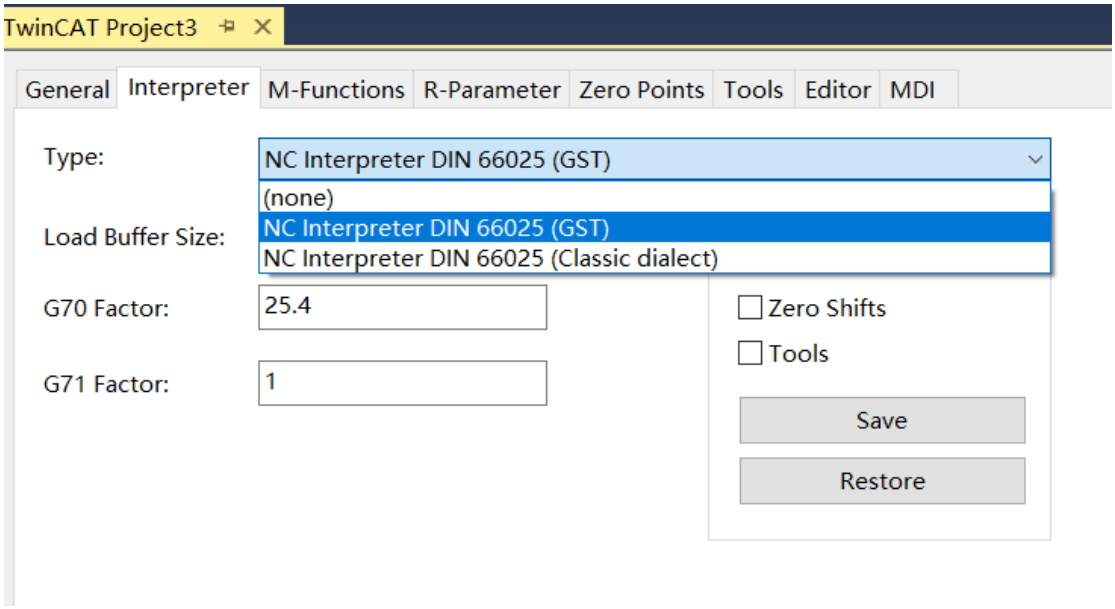
如下图的例子是用 GST 语言描述一个圆球形螺旋曲线逐渐上升的过程。可以看出 G 代码做运动指令配合 ST 语言的 FOR 循环指令描述这个曲线具有代码简洁，执行效率高的优势。





4.1 GST 的配置

TwinCAT V3.1.4024 中采用了新的解释器命名，将'Siemens dialect'改为'Classic Interpreter'，如果要使用 GST，这里选择 NC Interpreter DIN 6025。



4.2 预处理指令

#include “<path>”
#include <<path>>
#include 指令用于插入另一个文件的内容。插入的文件以 path 来引用。它一般用于导入常用代码（库），用法类似于 C 语言的预处理符。

例：下面例子里，文件 a.gst 包含文件 b.gst。执行 a.gst 时以 b.gst 的字符取代 include 行。

因此执行 a.gst 程序与 c.gst 程序具有相同的效果。

文件 a.gst:

G1 X0 Y0 F2000

#include “b.gst”
G1 X0 Y100

文件 b.gst:

G1 Z-2

G1 X100
G1 Z2

文件 c.gst

```
G1 X0 Y0 F2000
G1 Z-2
G1 X100
G1 Z2
G1 X0 Y100
```

4.3 G 代码和 ST 的结合

GST 程序

```
<g-code>
<g-code>
! <st-code>
<g-code>
<g-code>
{
<st-code>
<st-code>
! <g-code>
<st-code>
<st-code>
}
<g-code>
<g-code>
```

GST 文件由 G 代码和 ST 代码组成，如上面所示交错而成。每个程序都以 G 代码模式开始，某一行可用感叹号('!')切换到 ST 模式。ST 模式会在行尾自动结束。此外，可以用大括号('{'…'}')来定义 ST 代码。在 GST 程序中用它来定义一长串 ST 代码。在 ST 程序块中可用感叹号输入一行 G 代码。G 代码模式在行尾自动结束。

4.4 ST 语言

GST 语言中的 ST 语言的语法与 PLC 编程的 ST 语法相同，比如注释、变量和常量命名规则、逻辑语法以及一些标准函数等。

4.5 NCI 功能

4.5.1 字符串和消息

打印输出给定的文字，仅当解释器在 console 上运行时。

print

print(<arg0>,…,<argN>)
例：浮点数以 6 位小数的精度来显示。位字符串类型 LWORD 以十六进制表示法显示。时间类型以 1ns 精度显示。其中用 ST-standard function: currentLdt()来获取当前时间。

```
{
VAR
value1: REAL;
value2: LWORD;
END_VAR
value1:=2010.1980;
value2:=3538837009;
print('Hello World$n');
print('Value1:',value1,'$nValue2:',value2,'$nCurrent Date and Time:',currentLdt(),'$n');
}
```

Output:

```
Hello World
Value 1:2010.1980;
Value 2:d2ee5e11
Current Date and Time: 2015-03-18-13:03:33.000000000
```

toString

toString(<arg0>,…<argN>): STRING

将给定参数转化和连接成一个字符串。字符串上限为 255 个字符（缺省字符串长度）。

msg

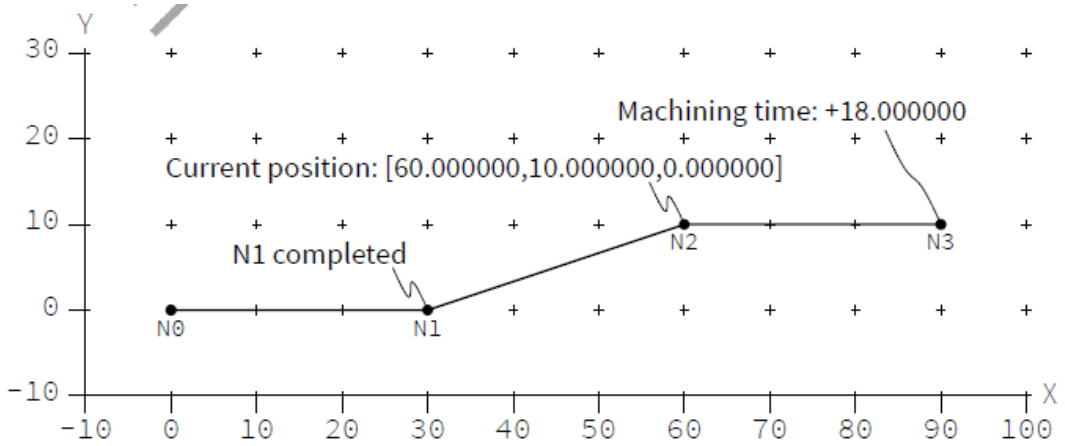
msg(str: String[81])

把指定信息送到 TwinCAT 信息列表。当前面所有 NC 指令执行完毕，它出现在用户界面上。它可以和 toString 函数结合使用。信息限制为 81 个字符，超过的文字将被删节。其中 frameGet()用于保存 PCS 当前坐标值，具体参考 4.5.9；sync()类似于@714 功能，具体参考 4.5.6 部分介绍。

例：

```
{
```

```
VAR
x,y,z: LREAL;
start: LDT;
END_VAR

!G0 X0 Y0 F300
start:=currentLdt();
!N1 G1 X30
msg('N1 completed');
!N2 X60 Y10
frameGet(x=>x,y=>y,z=>z);
msg(toString('Current position: ['',x,'','',y,'','',z,'']'));
!N3 X90
sync();
msg(toString('Machining time:',currentLdt()-start));
}
```



4.5.2 运动

moveCircle3d

moveCircle3d(cx: LReal, cy: LReal, cz: LReal, nx: LReal, ny: LReal, nz: LReal, angle: LReal, height: LReal)

围绕中心(cx,cy,cz)和法线向量(nx,ny,nz)旋转指定角度。如果 height 不为零，描述的是一个螺旋线。如果 angle 大于一个整圆，描述的是一个多圈的圆/螺旋线。旋转方向基于右手定则。一个负角度或翻转法线将颠倒旋转方向。

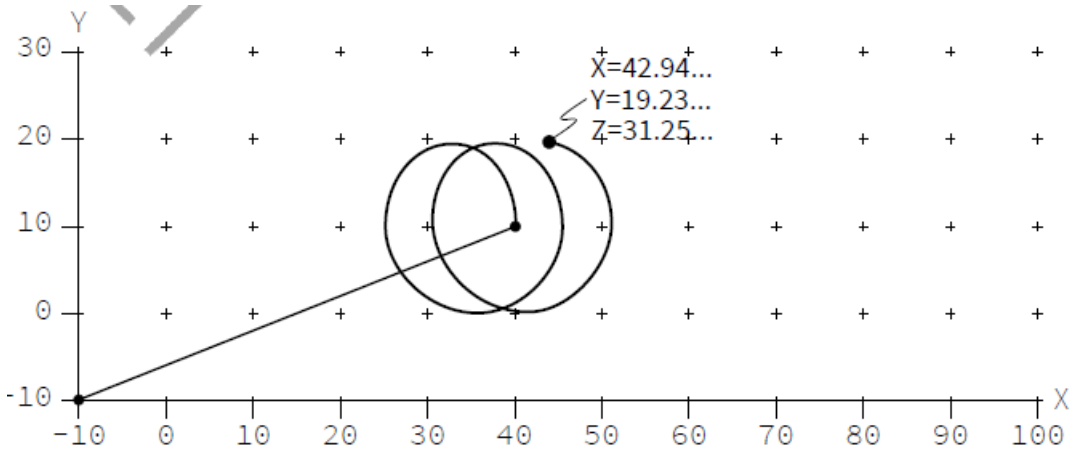
angle 值使用当前的角度单位。参数 x,y,z,cx,cy,cz 使用当前的长度单位。半径不能为零。

例：例中描述了一个螺旋线运动。从当前点[40,10,0]开始，螺旋线的中心轴以[30,10,0]和方向[gSin(22.5),0,gCos(22.5)]来定义。相比于工作平面的法向[0,0,1]轴在 X 方向倾斜 22.5 度。720+90 度角描述了一个多圈螺旋线。相对中心轴有个 30 的高度。

螺旋线的终点为避免冗余没有明确编程。如果用户需要这些坐标，它们可以用 frameGet

（…）函数检索。坐标近似值如图所示。

```
{
VAR
x,y,z: LReal;
END_VAR

!N1 G01 X40 Y10 F200
moveCircle3d(cx:=30, cy:=10, cz:=0, nx:=gSin(22.5), ny:=0, nz:=gCos(22.5), angle:=720+90,
height:=30);
frameGet(x=>x,y=>y,z=>z);
}
```



4.5.3 中心点修正

centerpointCorrectionSet

centerpointCorrectionSet(on:bool)

G2/G3 用中心点编程时，由于 CAD 程序的圆整误差等，相对于中心点的起点半径与终点半径会有不同。如果中心点修正被激活，中心将会移动，使起点半径与终点半径等于它们之前的平均值。

centerpointCorrectionLimitSet 可以配置中心点修正的限制值，如果超出，将报告一个 runtime error。

centerpointCorrectionLimitSet

centerpointCorrectionLimitSet(limit: LReal)

设置圆中心点的精度限制，默认限制值：0.1mm。如果超出，将报告一个 runtime error。

4.5.4 刀具

toolParamSet

toolParamSet(tidx:USint, col: USint, val: LReal)
把 val 赋给刀具 tidx(1..255)的参数。参数用 col(1..15)来确定。
COL DESCRIPTION
0 tool number
1 tool type(10: drill, 20: miller)
2 length
3
4 radius
5 length (磨损长度，一般负值，与 2 叠加使用)
6
7 radius (磨损半径)
8 x-shift
9 y-shift
10 z-shift

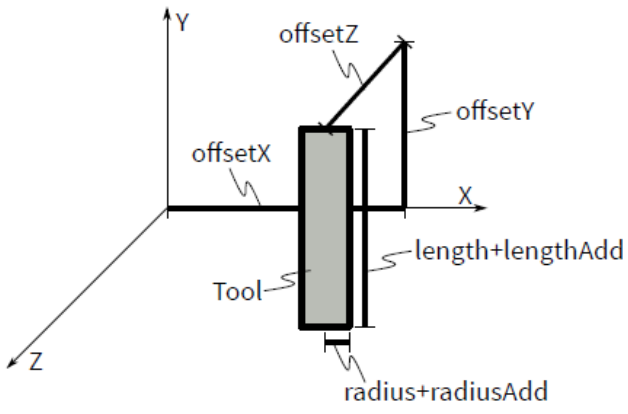
toolParam

toolParam(tidx: USint, col: USint): LReal
输出指定的刀具参数

toolSet

toolSet(index: USint, nr: Int, tooltype: ToolType, length: LReal, radius: LReal, lengthAdd: LReal, radiusAdd: LReal, offsetX: LReal, offsetY: LReal, offsetZ: LReal)

设置刀具的所有参数。index 被用在 D-Words 里描述刀具。它的范围是 1 到 255。参数 nr 只有信息的目的，通常是一个公司内部的数字来标志一个特定刀具。Tooltype 确定刀具的种类，如钻头。剩下的参数是尺寸，在下图中显示。如果刀具方向改为负(见 P-Word)，值‘length+lengthAdd’被隐式地取消。参数 length，radius，lengthAdd，radiusAdd，offsetX，offsetY，offsetZ 用当前的长度单位表示。



例：定义刀具 1 为总长 48.5 的钻头，刀具 2 为长度 30 和直径 5 的铣刀。
!toolSet(index:=1, nr:=4711, tooltype:=tooltypeDrill, length:=50, lengthAdd:=-1.5);
!toolSet(index:=2, nr:=10783, tooltype:=tooltypeMill, length:=30, radius:=2.5);

ToolType

ToolType

刀具种类的枚举
tooltypeDrill
tooltypeMill

4.5.5 同步

sync

sync()

sync 指令阻断代码执行直到所有 NC 指令完成，即直到 NC 通道任务队列为空。这个指令取代了之前的@714 指令。

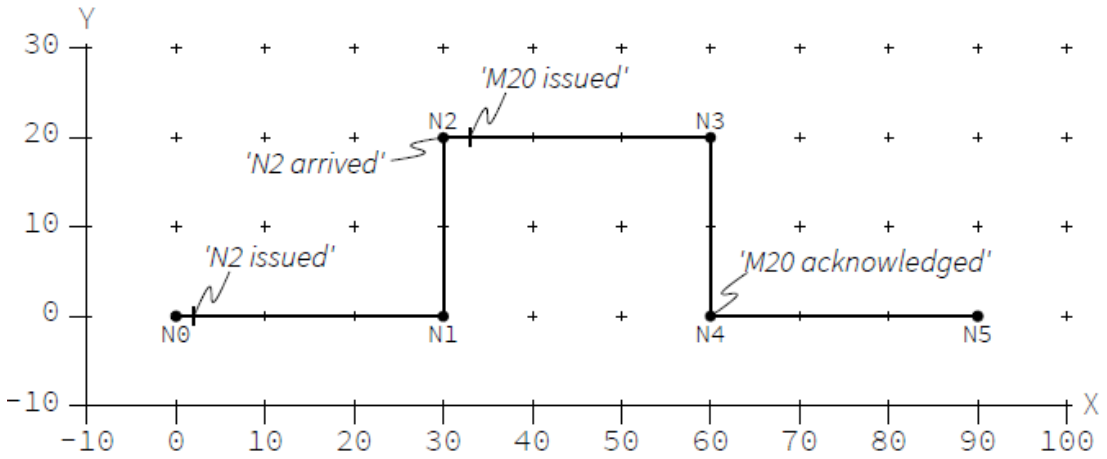
sync()指令常常与其前面的握手类型 M 函数组合使用。sync 指令将阻断代码执行直到 M 函数被 PLC 确认。

例：例中生成轨迹如下图。解释器直接打印出那些注释信息。例中 M20 是“handshake after”类型的 M 函数。

N1/N2 发送给 NC 通道后打印信息“N2 issued”。下面 sync()阻止代码执行直到 NC 通道为空，即 N1/N2 执行结束，刀具定位在 N2。然后显示信息“N2 arrived”。而后在任务队列插入 N3/N4 段，紧跟 M20。M 函数保留在队列中直到 PLC 确认。

如前，“M20 issued”立即显示。第二次调用 sync()阻止代码执行直到 N2/N3, N3/N4 处理完且 M20 被确认。然后在 N4 打印信息“M20 acknowledged”。

```
N0
N1 G1 X30 F200
N2 G1 Y20
!print('N2 issued');
!sync();
!print('N2 arrived');
N3 G1 X60
N4 G1 Y0 M20
!print('M20 issued');
!sync();
!print('M20 acknowledged');
N5 G1 X90
```



wait

wait()

等待 PLC 的 GoAhead 信号。wait()阻止指令的执行直到收到信号。它取代之前的@717 指令。

比较 M 函数与 sync()的组合，这种同步不会导致一个空的任务队列。一个空的任务队列会导致机器暂停。

注：GoAhead 信号可能在相关 wait()功能调用前由 PLC 发出。这时 wait()功能不会阻止代码执行。

例：例中的轨迹如下图所示。解释器在 t=0s 时开始。为简单起见，假定轨迹以 10mm/s 的恒定速度执行(启停时无限大加速度)。

GoAhead 信号(通过功能块 ItpGoAheadEx)发出在

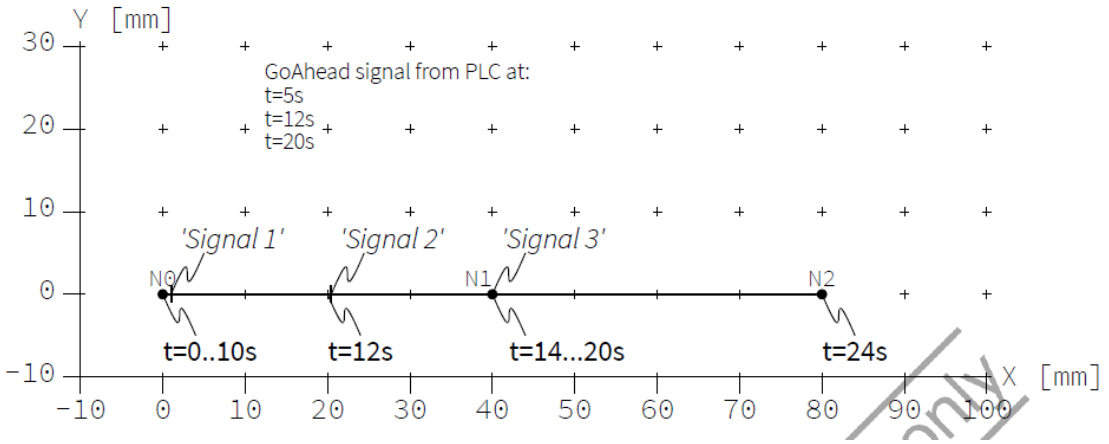
t=5s

t=12s

t=20s

G4 与 sync()组合使解释器停止 10s。此时收到第一个 GoAhead 信号，因此第一次调用 wait()不会阻止代码执行。在 N0/N1 段开始打印信息“Signal 1”。第二次调用 wait 阻止代码直到 t=12s。立即打印信息“Signal 2”。最后一次调用 wait 阻止代码直到 t=20s。由于 N0/N1 段在 t=14s 时完成。机器在 N1 停止 6s。

```
N0 G4 F10
!sync();
N1 G1 X40 F600
!wait();
!print('Signal 1');
!wait();
!print('Signal 2');
!wait();
!print('Signal 3');
N2 G1 X80
```



4.5.6 轴系查询

queryAxes()

将物理轴实际坐标设为解释器的 MCS 坐标。MCS 坐标自动转换为 PCS 坐标。它们可以用 frameGet(…)检索。

sync()和 queryAxes()取代之前的@716 指令。

注：queryAxes()前需先用 sync()以避免异常。

例：例中 M20 是“handshake after”类型的 M 函数。

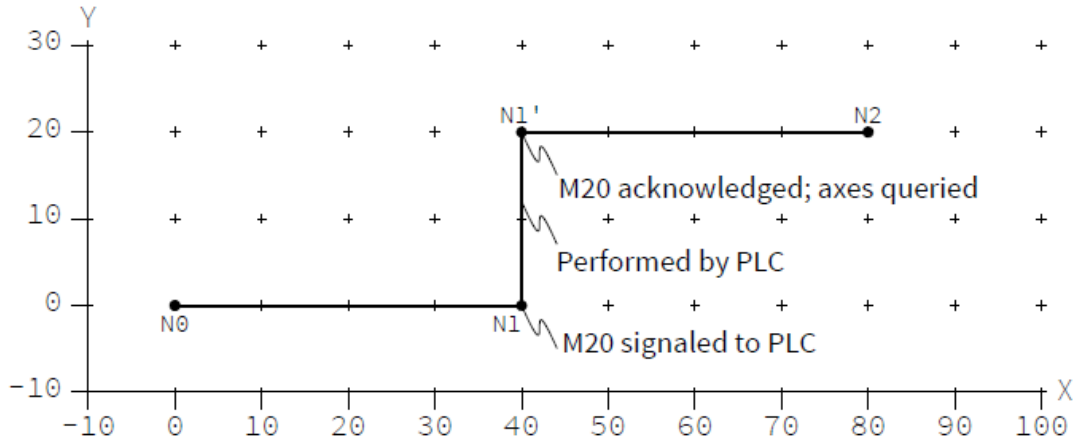
PLC 等待 M20

Y 轴移动到 20，运动完成

M20 确认

调用 sync()阻止代码的运行。N0/N1 直线段完成后 M 函数执行。PLC 移动刀具从 N1 到 N1‘并确认 M20。解释器再调用 queryAxes()将内部当前点设为 N1’。最后将直线段 N1’/N2 发给 NC 通道。

```
N0
N1 G1 X40 F200 M20
!sync();
!queryAxes();
N2 G1 X80
```



4.5.7 工作平面

workingplaneSet(x:LReal, y:LReal, z:LReal)

将法线向量[x,y,z]定义的平面设为工作平面。

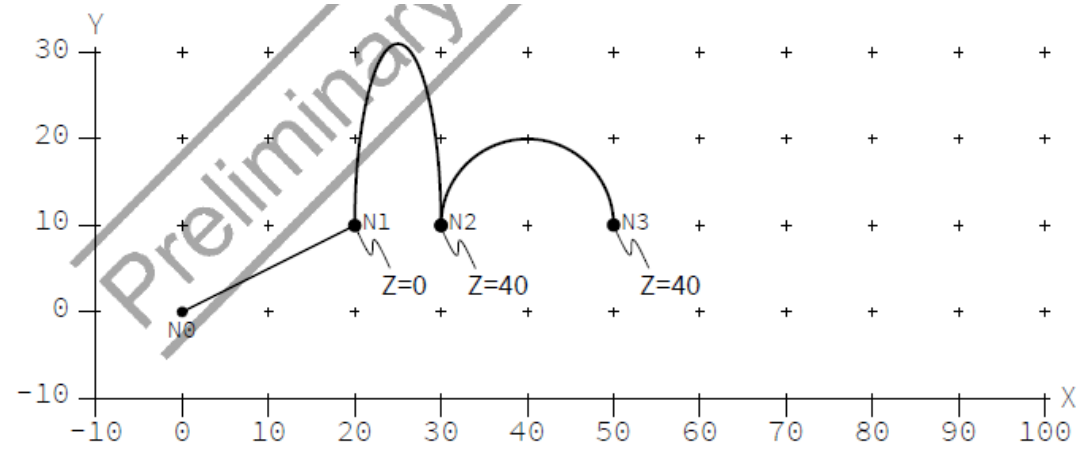
G17 对应的是 workingplaneSet(0，0，1)。

工作平面法向用于 G2 和 G3 的旋转轴。工作平面法向用 M41,M42 激活后用于刀具半径补偿。

例：工作平面法向先设为[-2,0,0.5]。这个新的工作平面影响 N1/N2 圆弧插补。因为是 2D 投影，尽管它实际是个圆，看上去却呈椭圆的形状。N2 的高度为 40。

工作平面复位为[0,0,1]后，下段圆弧 N2/N3 位于 XY 平面，也是缺省平面。

```
!workingplaneSet(x:=-2, y:=0, z:=0.5);
G1 X20 Y10 Z0 F200
G2 X30 Z40 I5 K20
!workingplaneSet(x:=0, y:=0, z:=1);
G2 X50 I10
```



4.5.8 当前点

frameGet(x:LReal, y:LReal, z:LReal, a:LReal, b:LReal, c:LReal,)
保存当前 PCS 的坐标。
例：G 代码先直线移动到 PCS 点[10,20,30]，通过 frameGet(…)把这些坐标存为 curX, curY, curZ。转换[1,2,3]压入运动转换堆栈，MCS 点[10,20,30]保持不变。接着调用 frameGet(…)检索得到 PCS 点[9,18,27]。

```
{
VAR
curX, curY, curZ: LREAL;
END_VAR

!G1 X10 Y20 Z30 F200

frameGet(x=>curX,y=>curY,z=>curZ);
print(curX, ‘ ‘, curY, ‘ ‘, curZ, ‘ $n‘);
transTranslate(1,2,3);
frameGet(x=>curX,y=>curY,z=>curZ);
print(curX, ‘ ‘, curY, ‘ ‘, curZ, ‘ $n‘);
}

Output
10.000000    20.000000    30.000000
9.000000    18.000000    27.000000
```

4.5.9 刀具半径补偿

trcSet

trcSet(nx:LReal, ny:LReal, nz:LReal, offset: LReal, approachRadius:LReal, approachAngle:LReal, departRadius:LReal, departAngle:LReal, limit:Int)

按法向向量[nx,ny,nz]激活刀具半径补偿。法向向量长度定义了刀具半径。offset 定义了闭合接缝的分段延伸量。approachRadius、approachAngle、departRadius 和 departAngle 定义了接近与离开的行为。limit 定义了防撞的预读量。负值将产生无限制的预读量，缺省-1。

trcApproachSet

trcApproachSet(radius: LREAL, angle: LREAL)
用给定半径和角度的圆弧去配置接近和离开的行为。这个配置用于 G41/G42。

trcOffsetSet

trcOffsetSet(offset: LREAL)

配置闭合接缝的分段延伸量。这个配置用于 G41/G42。

trcLimitSet

trcLimitSet(limit: LREAL)
配置防撞的预读量。这个配置用于 G41/G42。

collisionElimination

collisionElimination(nx: LREAL, ny: LREAL, nz: LREAL)
按法向 nx.ny,nz 的平面激活防撞。平面内轨迹的干涉将被删除。

4.5.10 G 代码的抑制

disableMask

disableMask(): LWord
得到 disable mask 的当前值。

disableMaskSet

disableMaskSet(mask: LWord)

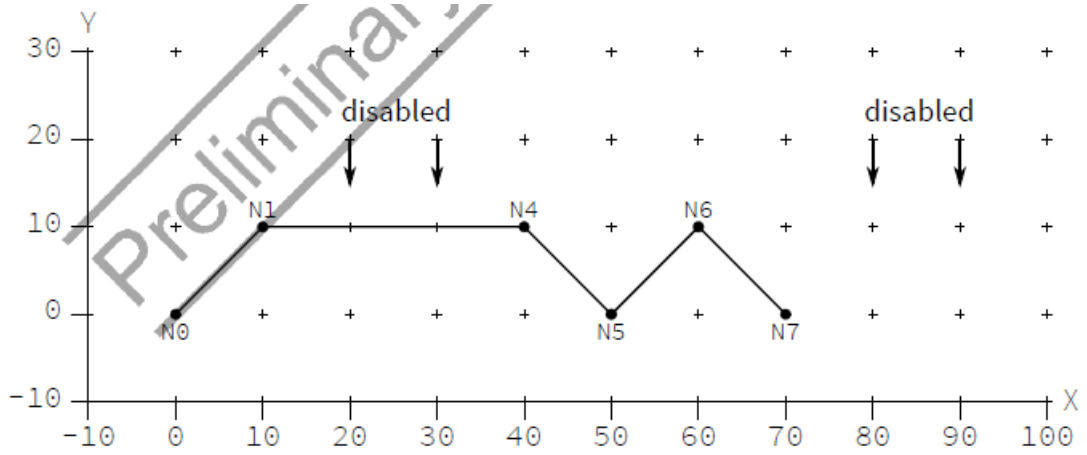
给内部 disable mask 赋值。mask 用于抑制 G 代码块的执行。disable mask 的缺省值为零即没有抑制。mask 由 64 个位构成。

如 2#1101 的二进制表示法，位从右往左数，从 0 开始。所以前面的表示法中位 0，2 和 3 置位。

例：disable-mask 初始值为二进制 1101。第一行 G 代码 N1 通常都是执行的，不管当前的 disable-mask。只有第 0 位为 0，N2 才执行。例中 N2 被抑制。N3 相同，因为'/'是'/0'的简写。由于第 1 位未置位，所以 N4 未被抑制。disableMaskSet(0)的 G 代码 N5,N6 执行，因为 disable-mask 中没有位被置位。相反 disableMaskSet(-1)将 mask 中的所有位都置位，后面所有前缀'/'的 G 代码都被抑制执行。

```
!disableMaskSet(2#1101);
N1 G1 X10 Y10 F200
/0 N2 G1 X20 Y0
/N3 G1 X30 Y0
/1 N4 G1 X40 Y10
!disableMaskSet(0);
/N5 G1 X50 Y0
/1 N6 G1 X60 Y10
!disableMaskSet(-1);
N7 G1 X70 Y0
/1 N8 G1 X80 Y10
```

/2 N9 G1 X90 Y20

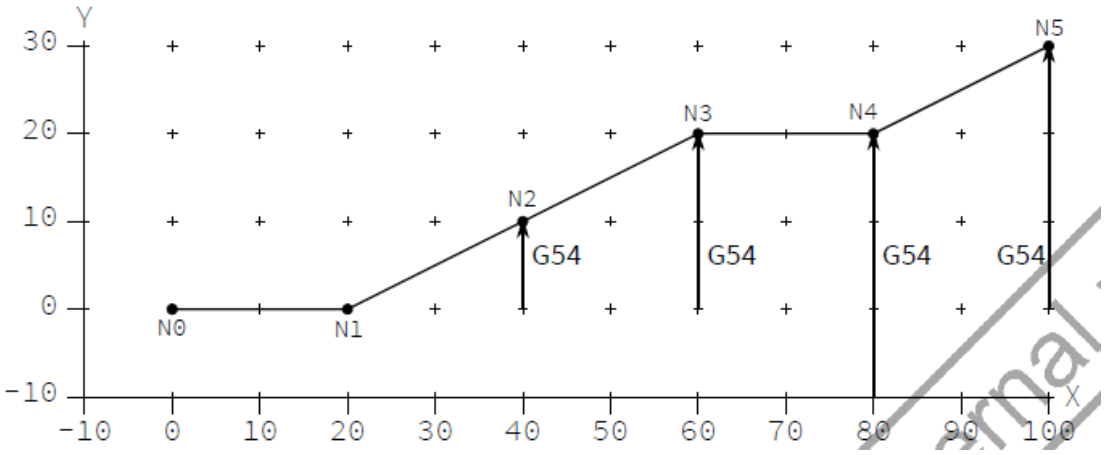


4.5.11 零点偏移

zeroOffsetShiftSet

zeroOffsetShiftSet(g: USINT, x: LREAL, y: LREAL, z: LREAL)
设置 G 代码坐标系转换，g 必须是 54，55，56，57 之一。
例：G54 的坐标系偏置首先设为[0,10,0]。第二次调用 zeroOffsetShiftSet 立即生效，N3 使用了这个偏置。最后一次调用同理。然而，N4 没有编写 Y 坐标，因此没有显示这个变化。

```
! zeroOffsetShiftSet(g:=54, x:=0, y:=10, z:=0);
G1 N1 X20 Y0 F200
G1 N2 X40 Y0 G54
! zeroOffsetShiftSet(g:=54, x:=0, y:=20, z:=0);
G1 N3 X60 Y0
! zeroOffsetShiftSet(g:=54, x:=0, y:=30, z:=0);
G1 N4 X80
G1 N5 X100 Y0
```



4.5.12 单位

unitAngleSet

unitAngleSet(unitangle: UnitAngle)

将角度单位设为 unitangle。缺省值是 unitAngleDegree。角度单位适用所有像 transRotX 的 NC 函数。不适用像 sin 的 ST 标准函数。因此，解释器提供一套如 gsin 的 NC 特定函数，使用这些角度单位。

UnitAngle

UnitAngle
具有以下枚举值:
unitAngleRadian: 0…2π
unitAngleDegree: 0…360
unitAngleGon: 0…400
unitAngleTurn: 0…1

unitLengthSet

unitLengthSet(unitlength: UnitLength)

将长度单位设为 unitlength。缺省值是 unitLengthMillimeter。角度单位适用所有像 G01 或 zeroOffsetShiftSet(…)的 NC 函数。

UnitLength

UnitLength
具有以下枚举值:
unitLengthMeter
unitLengthCentmeter
unitLengthMillimeter
unitLengthMicrometer
unitLengthNanometer
unitLengthInch
unitLengthFoot

unitTimeSet

unitTimeSet(unittime: UnitTime)

将时间单位设为 unittime。缺省值是 unitTimeSecond。时间单位适用所有像 G04 的 NC 函数。不适用像 currentLdt()的 ST 标准函数。

UnitTime

UnitTime
具有以下枚举值:
unitTimeSecond
unitTimeMilliSecond
unitTimeMicroSecond
unitTimeMinute
unitTimeHour

unitVelocitySet

unitVelocitySet(unitlength: UnitLength, unittime: UnitTime)

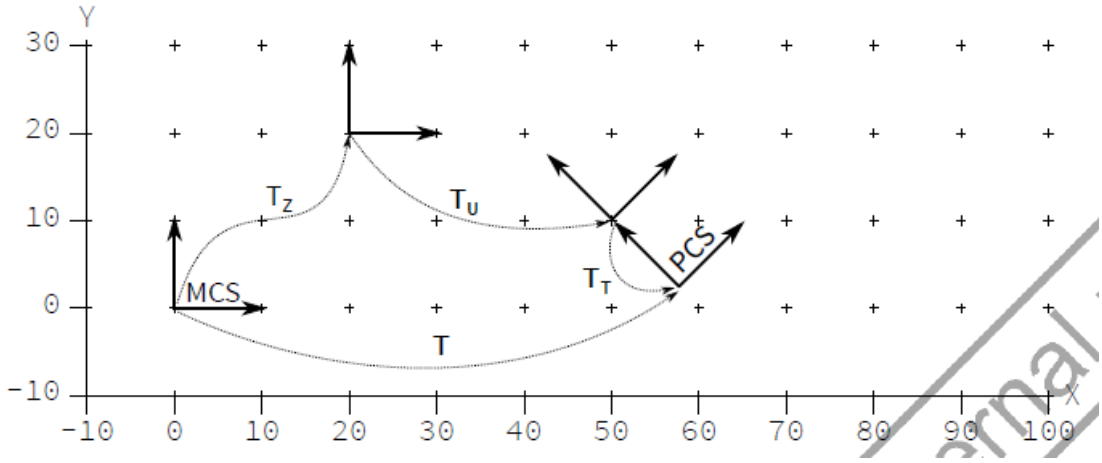
将速度单位设为 unitlength/unittime。缺省值是 unitLengthMillimeter/unitTimeMinute。这个速度单位适用于所有 NC 相关函数。用于 F 参数。

4.5.13 三角函数

gSin(angle: LReal) 返回给定角度 angle 的 sine 值， 返回类型是 LREAL。
gCos(angle: LReal) 返回给定角度 angle 的 cosine 值， 返回类型是 LREAL。
gTan(angle: LReal) 返回给定角度 angle 的 tangent 值， 返回类型是 LREAL。
gASin(val: LReal) 返回给定值 val 的 arcsine 值，返回类型是 LREAL。结果在[-c/4, c/4]，c 是当前角度单位表示的一个整圆的角度。val 必须在[-1,1]内。
gACos(val: LReal) 返回给定值 val 的 arccosine 值， 返回类型是 LREAL。结果在[0, c/2]，c 是当前角度单位表示的一个整圆的角度。val 必须在[-1,1]内。
gATan(val: LReal) 返回给定值 val 的 arctangent 值， 返回类型是 LREAL。结果在[-c/4, c/4]，c 是当前角度单位表示的一个整圆的角度。
gATan2(y: LReal, x: LReal) 返回给定角度 y/x 的 arctangent 值， 返回类型是 LREAL。结果在[-c/2, c/2]，c 是当前角度单位表示的一个整圆的角度。

4.6 变换

T 是 MCS(机器坐标系)和 PCS(编程坐标系)间的关系。T 是转换 Tz，Tu 和 TT 的串联。Tz 代表零点偏置变换，Tu 代表用户自定义变换，Tт代表工具变换。
下图中 Tz=[20,20,0]。Tu 是变换[30,-10,0]后再绕 Z 轴旋转 45 度。Tт=[0,-10,0]。



4.6.1 变换的语法

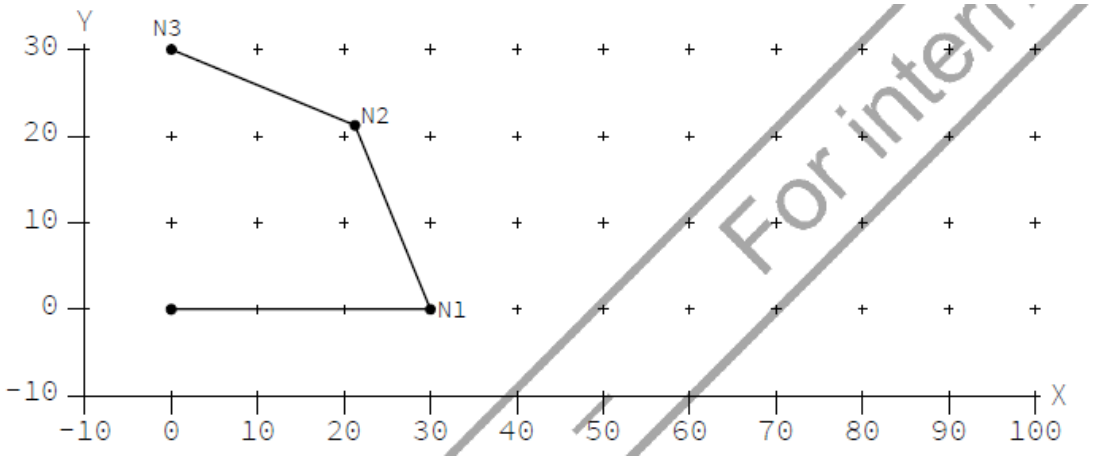
transRotX/Y/Z

transRotX(angle: LReal)
transRotY(angle: LReal)
transRotZ(angle: LReal)

把指定的轴旋转给定的角度，使用当前的角度单位。旋转被压入转换堆栈。

例：N1 的 PCS 与 MCS 相同。N2 是以[0,0,0]绕 Z 轴逆时针旋转 45 度后的坐标系编程，转换被压入堆栈。接着另一个 45 度逆时针旋转被压入堆栈。总的旋转角度达到 90 度，N3 的 MCS 坐标为[0,30,0]。

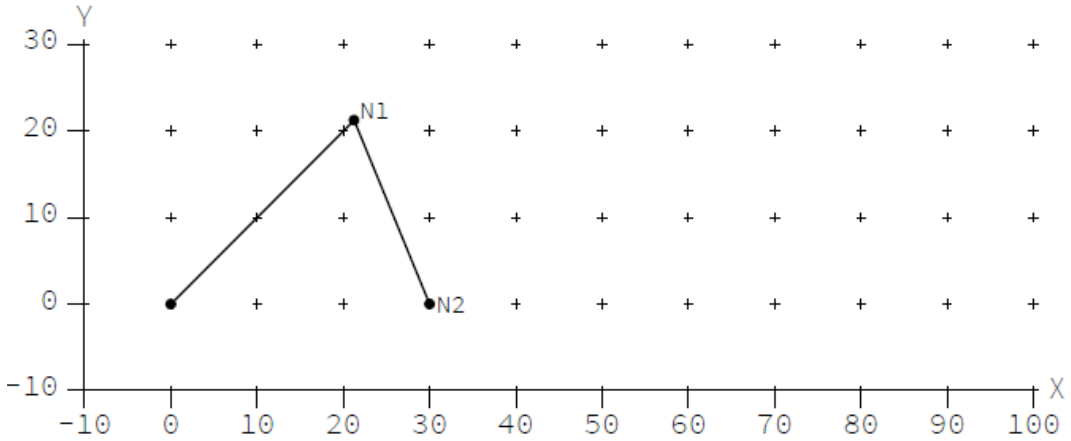
```
N1 G01 X30 Y0 F200
!transRotZ(45);
N2 G01 X30 Y0
!transRotZ(45);
N3 G01 X30 Y0
!transPop();
!transPop();
```



transRotA

transRotA(x:LReal, y:LReal, z:LReal, angle:LReal)
绕[x,y,z]旋转指定的角度，使用当前的角度单位。旋转被压入转换堆栈。
例：第一次调用 transRotA 将 PCS 围绕 Z 轴正向（右手定则）旋转 45 度。第二次调用绕 Z 轴负向旋转相同角度，即反方向。

```
!transRotA(0,0,1,45);
N1 G01 X30 Y0 F200
!transRotA(0,0,-1,45);
N2 G01 X30 Y0
!transPop();
!transPop();
```

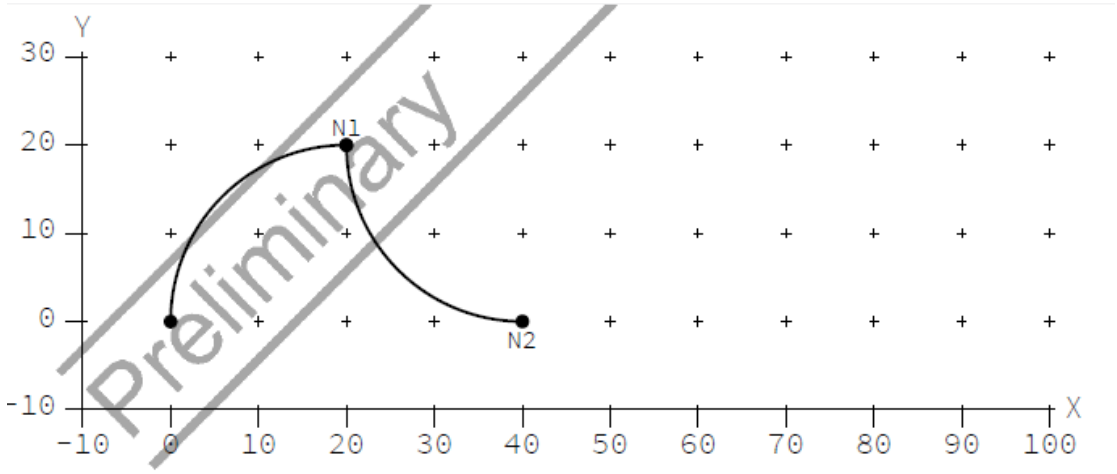


transMirrorX/Y/Z

transMirrorX()
transMirrorY()
transMirrorZ()
相对于当前 PCS 原点将 X,Y 或 Z 轴方向进行的镜像，转换被压入转换堆栈。
例：

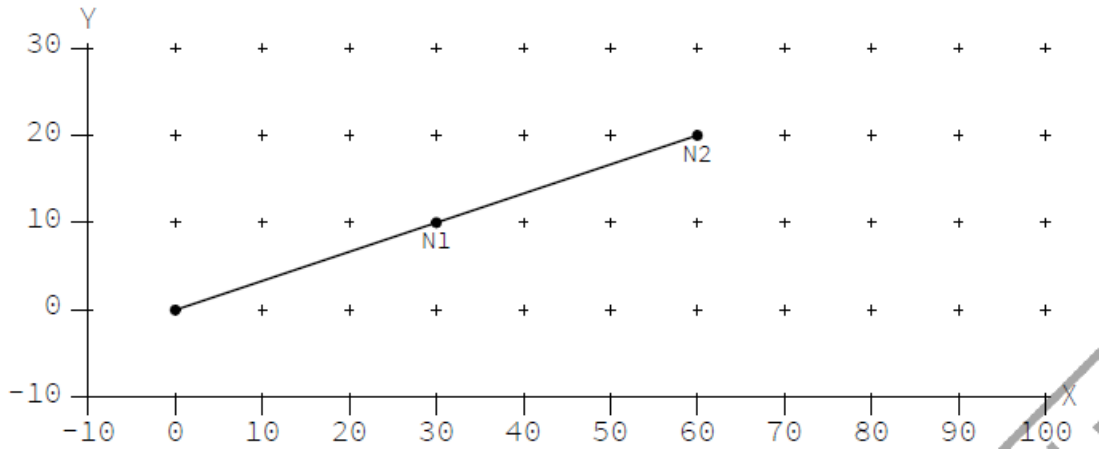
```
N1 G02 X20 Y20 U20 F200
!transMirrorX();
```

N2 G02 X-40 Y0 U20
!transPop();



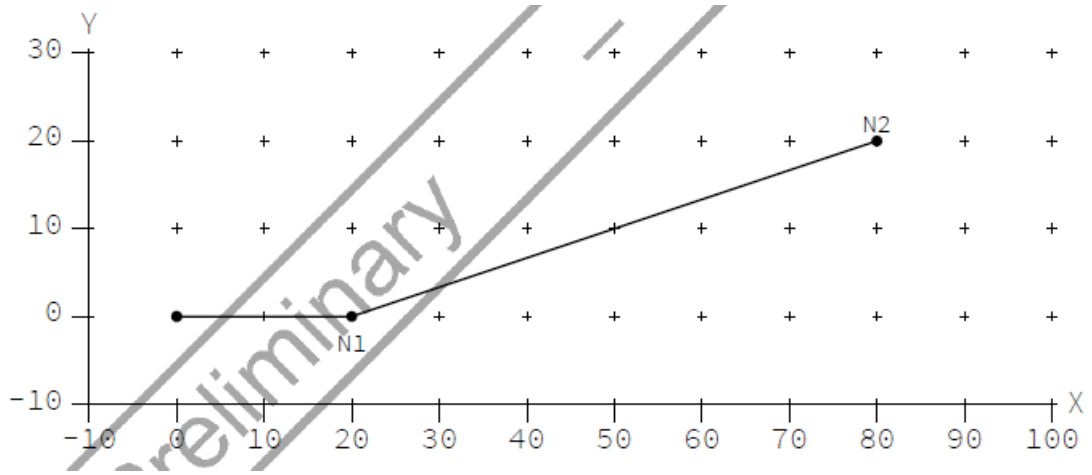
transScale

transScale(factor: LREAL)
伸缩系数 factor 为非零值，转换被压入转换堆栈。
例:
N1 G01 X30 Y10 U20 F200
!transScale(2);
N1 G01 X30 Y10 U20
!transPop();



transTranslate(x:LReal, y:LReal, z:LReal)

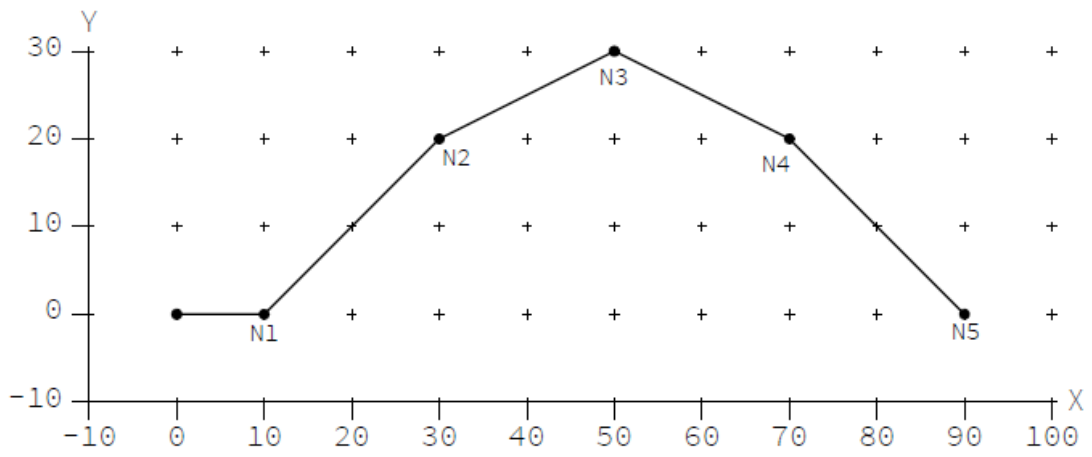
以向量[x,y,z]偏置，转换被压入转换堆栈。
例：
在偏置[40,20,0]后 N2 映射到[80,20,0]。
N1 G01 X20 Y0 F200
!transTranslate(40,20,0);
N2 G01 X40 Y0
!transPop();



transPop()

transPop()
从转换堆栈中释放一个转换。
例：首先把[0,20,0]压入堆栈，再把[0,10,0]压入堆栈，所以 N30 的有效转换是[0,30,0]。第一次调用 transPop 把[0,10,0]释放堆栈，因此 N4 点按[0,20,0]转换。把最后一个转换移出堆栈后，N5 没有做转换。

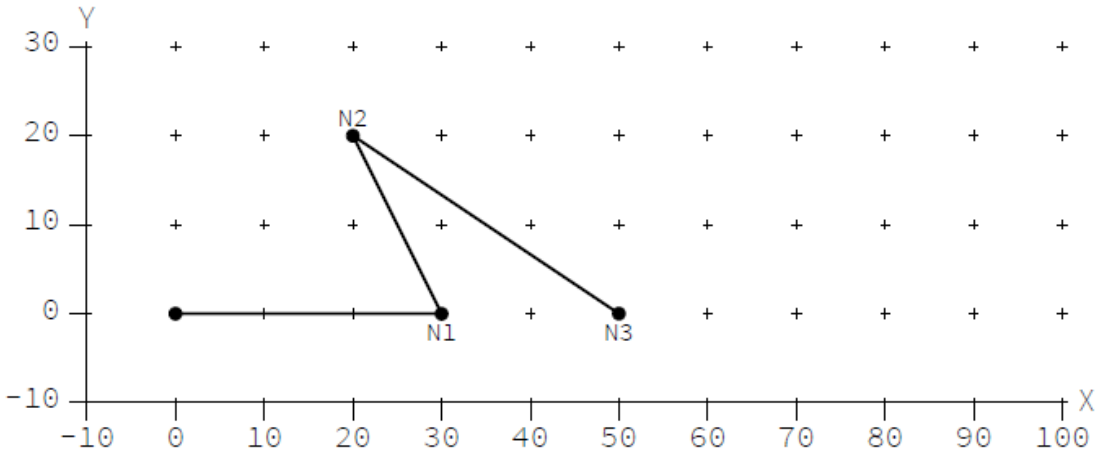
```
N1 G01 X10 Y0 F200
!transTranslate(0,20,0);
N2 G01 X30 Y0
!transTranslate(0,10,0);
N3 G01 X50 Y0
!transPop();
N4 G01 X70 Y0
!transPop();
N5 G01 X90 Y0
```



例 2：下例中变换对 N1,N2 和 N3 有效。旋转仅对 N2 有效，由于它被 transPop()弹出。下图显示了 MCS 的路径结果。注：MCS 的旋转中心是[20,0,0]，它是前面转换后的 PCS 原点。

```
!transTranslate(20,0,0);
```

```
N1 X10 Y0 F200
!transRotZ(90);
N2 X20 Y0
!transPop();
N3 X30 Y0
!transPop();
```



transDepth

transDepth(): UInt

产生转换堆栈的深度即有效转换的数量。该参数用于计算执行到该行时，已进行有效转换的次数。

transRestore

transRestore(depth: UInt)

将转换堆栈减少到指定深度。通常与 transDepth()一起用于恢复堆栈的一个早期状态。当前堆栈深度不小于指定深度。用于恢复到堆栈中指定转换次数的深度处，如 depth 为 2，那么保留前 2 次转换坐标结果，其他转换移出堆栈。

4.6.2 有效变换 T 和效果

大多数 G 代码仅定义运动的目标位置。位置点可以用 MCS 和 PCS 坐标表示，$CurrentPoint_{MCS}=T* CurrentPoint_{PCS}$。

例：下面 G 代码执行 N1 后，因为没有有效变换，当前 PCS 和 MCS 的坐标是[20,10,80]。变换把当前 PCS 改为[28,7,84]。MCS 仍为[20,10,80]。因此变换没有效果。N2 编程了一个运动到 PCS 点[25,7,10]，映射为 MCS 坐标[17,10,6]。调用 transPop()后，将转换移除，此时 MCS 和 PCS 又是同一个点了。

```
N1 G1 X20 Y10 Z80
!transTranslate(-8,3,-4);
N2 G1 X25 Z10
```

```
!transPop();
```

例：如果用户期望 PCS 点保持不变，他需要像下面代码来检索和编程。然而期望不变的 PCS 点通常表现出不好的编程风格。故实际上不需要下面的代码。

```
{
VAR
pcsX, pcsY, pcsZ: LREAL;
END_VAR

//…G-Code…

frameGet(x=>pcsX, y=>pcsY, z=>pcsZ);
//…modify transformation…
!G01 x=pcsX, y=pcsY, z=pcsZ
}
```

4.6.3 有效变换 T

零点偏移：T_Z

TZ 变换受 G 代码影响。如果 G53 激活，变换失效。否则 Tz 是三个变换 TZ58,TZ59 和{TZ54,…TZ57}之一的组合。前面两个通过 G58 和 G59 设置。最后的通过 G54 到 G57 选择。可以通过 PLC 或 ST 功能 zeroOffsetShiftSet 设置。

刀具变换：T_T

如选择 Tool 0 (D0)该变换无效。否则用转换[offsetX, offsetY, offsetZ] + (length+lengthAdd)*D, D 是当前工作平面的法向。

自定义变换：T_U

TU 以变换堆栈定义。堆栈包含 TU1，TU2，…, TU<N.>，TU<N.>位于顶部。堆栈最初是空的，自定义变换是这些基本变换的串联 TU= TU1* TU2*…* TU<N.>。它们的顺序很重要，如果堆栈为空，TU 是同一变换，没有影响。

4.6.4 变换撤消

transPop()从堆栈中弹出最顶端的变换。这个功能通常用于撤消一个临时变换。详见 4.6.1。

4.6.5 堆栈复原

transDepth()产生堆栈的当前深度。transRestore(depth)从堆栈中弹出变换直到给定堆栈深度。
这两个功能通常组合使用以保存和复原变换堆栈的状态。详见 4.6.1。

4.6.6 变换的应用

例 1：堆栈深度开始存储在 depth 中。function 的最后用 transRestore 复原最初的状态。堆栈深度也可以用 transPop 复原。然而如果变换是有条件地压入，使压入和弹出变换保持同步比较麻烦。

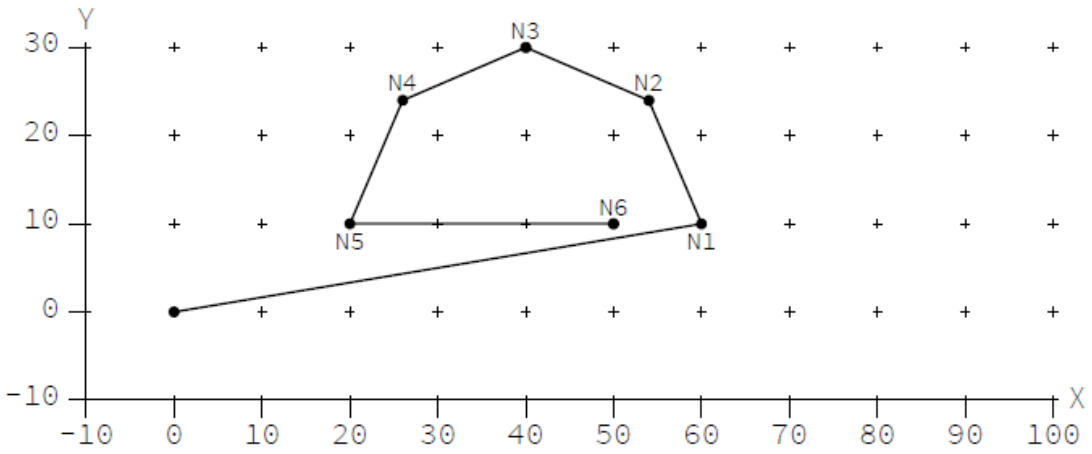
```
{
FUNCTION draw
VAR
depth: UINT;
END_VAR
depth:= transDepth();
transTranslate(10,0,0);
//…G-Code…
transRotZ(45);
//…G-Code…
transMirrorX();
//…G-Code…
transRestore(depth);
END_FUNCTION
}
```

例 2：一开始[40,10,0]被压入转换堆栈。堆栈深度存储于变量 savedDepth 中。后面的代码重复直线移动到 X20 Y0 和 45 度旋转。轨迹是八角形的一半，从 N1 到 N5。当 N5 处理完后，转换堆栈包括最初的转换和 4 个 45 度旋转。调用 transRestore(savedDepth)恢复堆栈深度到 1，移出所有的旋转转换。因此 N6 只用到这个转换。

```
!VAR savedDepth: UINT; END_VAR
!transTranslate(40,10,0);
!savedDepth:=transDepth();

N1 G01 X20 Y0 F200
!transRotZ(45);
N2 G01 X20 Y0
!transRotZ(45);
N3 G01 X20 Y0
!transRotZ(45);
N4 G01 X20 Y0
!transRotZ(45);
N5 G01 X20 Y0
```

!transRestore(savedDepth);

N6 G01 X10 Y0



4.7 刀具半径补偿

4.7.1 用户提供的参数

刀具半径补偿(TRC)可用以下参数配置：

- 参考平面的法向[nx,ny,nz]
- 接近/离开段的半径与角度
- 扩展相邻段以闭合接缝的偏移量
- 避免轮廓碰撞的预读上限

4.7.2 碰撞消除

刀具半径补偿(TRC)用于补偿柱形刀具的半径，如铣刀。TRC 激活时，编程路径是需要加工的轮廓，并不是刀具的实际路径。下面，轮廓称为编程路径而刀具路径称为补偿路径。补偿路径基于以下规则由编程路径生成：

1. 沿补偿路径加工不会损坏轮廓。
2. 当第一条未违反时沿补偿路径加工正好生成轮廓。

例：下例是 TRC 激活时编程的燕尾。下图中编程路径 (虚线)，补偿路径(实线)，加工零件(灰色区域)。N3 和 N4 两个角由于刀具有大小为 5 的半径而无法完全加工。故编程路径和实际路径有所差异。当刀具半径增加到 10 时，差异更加明显，如图 2。当半径慢慢增加到 11，刀具不再穿过缝隙，如图 3。刀具尽可能地进入燕尾而不损坏 N2 和 N5。

```
!trcSet(nz:=-5);
N1 G1 X10 Y10 F200
N2 X20
N3 X0 Y30
N4 X60
N5 X40 Y10
N6 X50
!trcSet();
N7 X60 Y0
```

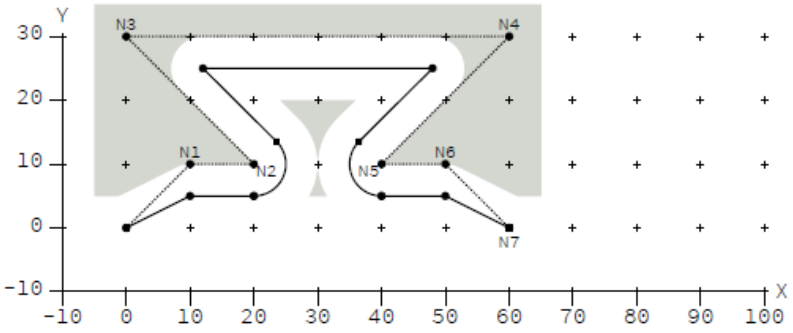


Fig. 1:

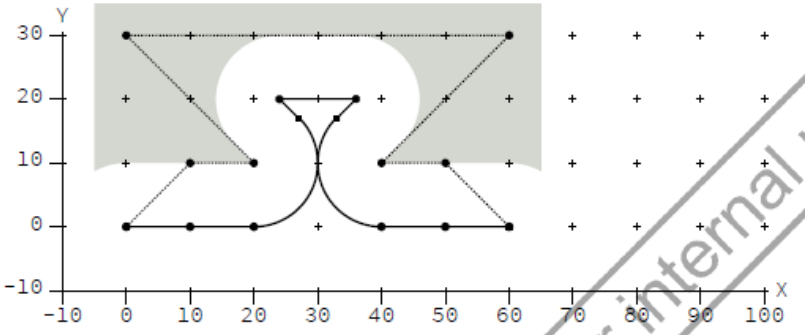
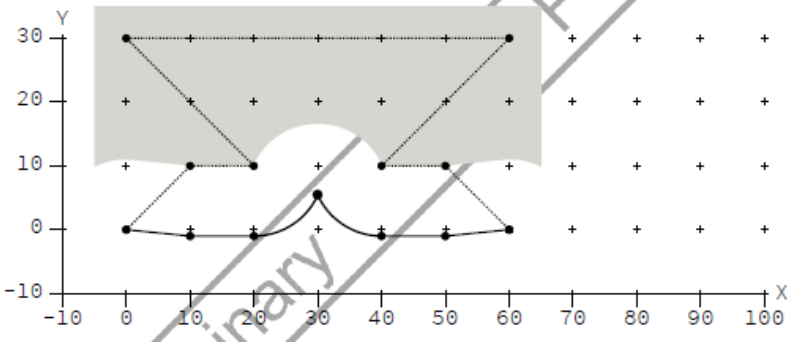


Fig. 2:



4.7.3 路径重叠

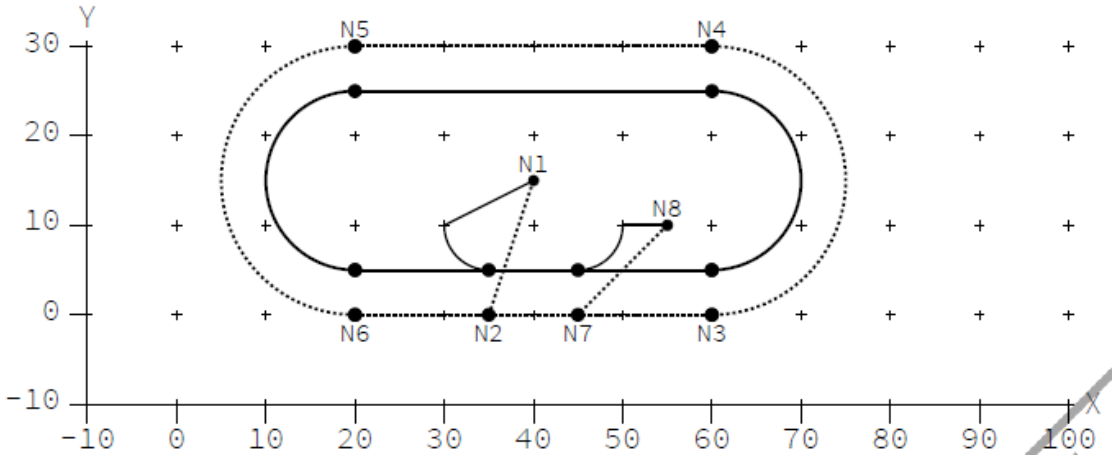
加工封闭图形时，通常如下图使第一段和最后一段重叠。
理论上两个重叠路径完美结合。由于浮点误差(如旋转变换)结合处会微微交叉。
TRC 包含对这种情况的特殊处理，允许一个很小的轮廓损坏。

例：

N2/N3 段与 N6/N7 段重叠。如果 N2 和 N7 吻合，轮廓无缝接合。

```
G0 X40 Y15
!trcSet(nz:=5, approachRadius:=5, approachAngle:=90, departRadius:=5, departAngle:=90);
N2 X35 Y0 Z-2
N3 X60
N4 G3 Y30 U15 F200
N5 G1 X20
```

N6 G3 Y0 U15
N7 G1 X45
!trcSet();
G1 X55 Y10 Z0



4.7.4 用于消除碰撞的预读

trcLimitSet(limit:= LReal)，TRC 算法生成补偿路径使编程路径不会损坏。然而用户期望使消除碰撞限制到编程路径上一个小范围内。这个限制可以用 limit 参数来配置，其缺省配置为“unlimited”。

如果 limit 设置为 1，TRC 仅消除段 i 和段 i+1 间的碰撞。如果 limit 设置为 2，所有段 i 和段 i+1 及段 i+2 间的碰撞消除。总之所有段 i 和段 i+1 到段 i+limit 间的碰撞消除。

然而正交段被 TRC 有意忽略，因为他不会导致轮廓破坏。因而之前谈到的规则仅考虑非正交段。

4.7.5 刀具半径补偿的再激活

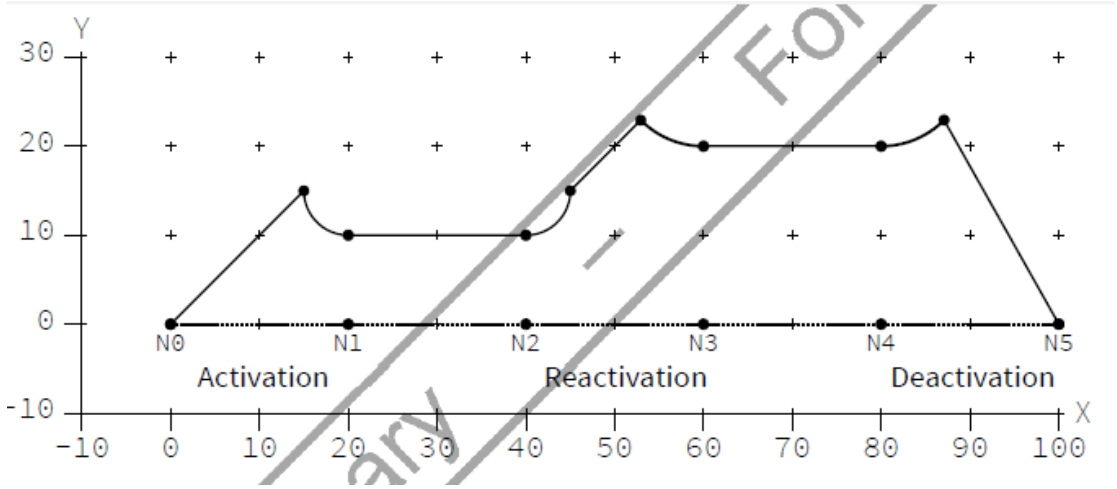
当 TRC 激活时，它可用新的补偿参数再激活。这些参数包括半径，接缝闭合与进出行为。再激活可用来实现以下任务:

- TRC 激活期间的换刀
- 在从左到右和从右到左间切换 TRC 方向
- 无轮廓损坏的通过编程路径

例：图中编程路径(虚线)和补偿路径(实线)。TRC 在 N1 前激活，N2 和 N3 间再次激活。再次激活配置不同的刀具半径和不同的进出行为。N2 和 N3 间补偿路径包含第一次激活的离开段和再次激活的进入段。两端间直线相连。

!trcSet(nz:=10, approachRadius:=5, approachAngle:=90, departRadius:=5, departAngle:=90);
N1 G1 X20 F200
N2 X40
!trcSet(nz:=20, approachRadius:=10, approachAngle:=45, departRadius:=10, departAngle:=45);
N3 X60 Y0

N4 X80
!trcSet();
N5 X100



4.7.6 接缝闭合

编程路径上的凸角会导致补偿路径上的缝隙。例 1 显示了这样一个编程路径(虚线)和补偿路径(实线)。直线段 N0/N1 和 N1/N2 映射为补偿直线 N0'/N1'和 N1''/N2''。N1'与 N1''间的缝隙用延伸和圆弧组合来闭合。相邻段(N0'/N1'和 N1''/N2'')用一个长度 offset 的直线延伸。余下的由一段圆弧来闭合。

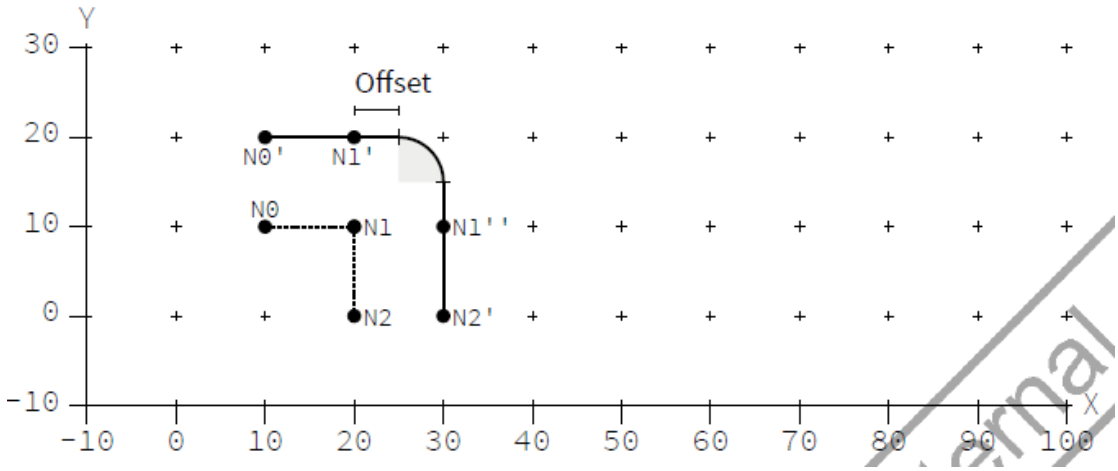
如果延伸重叠，它们将在交叉点被截断，不使用圆弧(例 3)。

如果 offset 为零，没有延伸，所有缝隙用圆弧闭合(例 2)。这是不损坏轮廓来闭合缝隙的最短路径。然而像铣刀这种刀具沿补偿路径移动时会持续接触这个角，这会因为摩擦和不精确的运动引起过热而损伤轮廓。一个合理的 offset 取决于加工材料，刀具种类和机器公差。另一方面如果 offset 设为一个很大的值，所有接缝(除很尖的角外)将用延伸来闭合，如例 3。然而大的 offset 值会增加加工时间，也会增加轮廓损坏的风险。

例 1：

下例中缝隙通过一个延伸与圆弧来闭合。

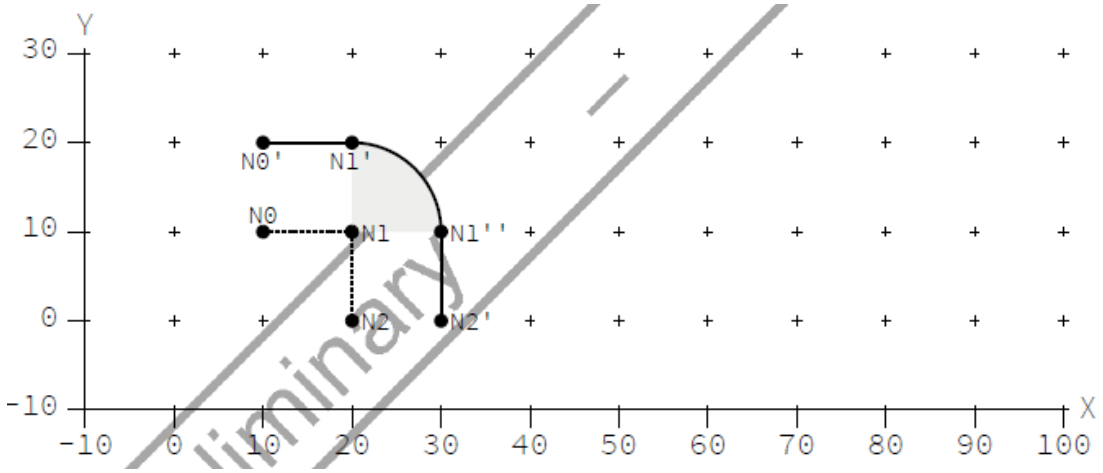
!trcSet(nz:=10,offset:=5);
N0 G0 X10 Y10
N1 G1 X20 F200
N2 G1 Y0
!trcSet();



例 2：

下例中缝隙由零偏置闭合，没有延伸。

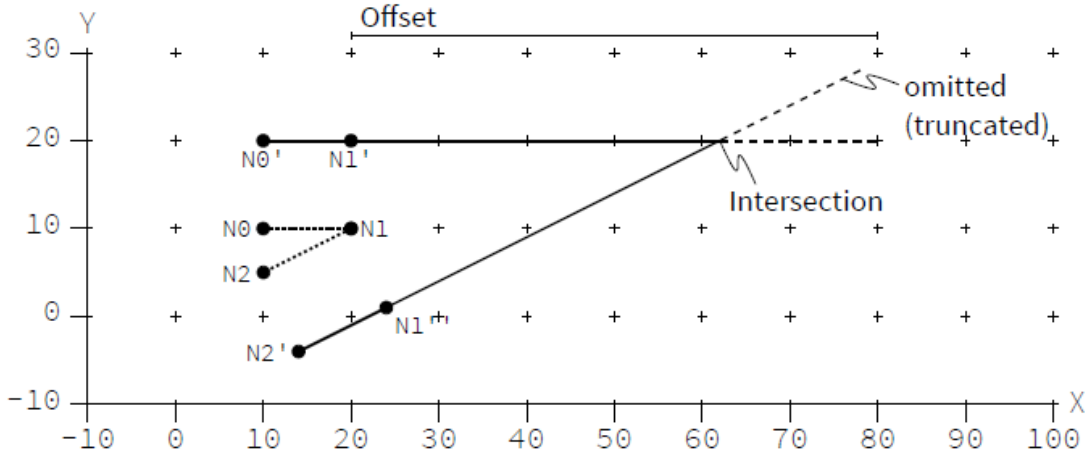
```
!trcSet(nz:=10,offset:=0);
N0 G0 X10 Y10
N1 G1 X20 F200
N2 G1 Y0
!trcSet();
```



例 3：

下例通过一个很大的偏置来闭合缝隙，圆弧被去除。

```
!trcSet(nz:=10,offset:=60);
N0 G0 X10 Y10
N1 G1 X20 F200
N2 G1 X10 Y5
!trcSet();
```

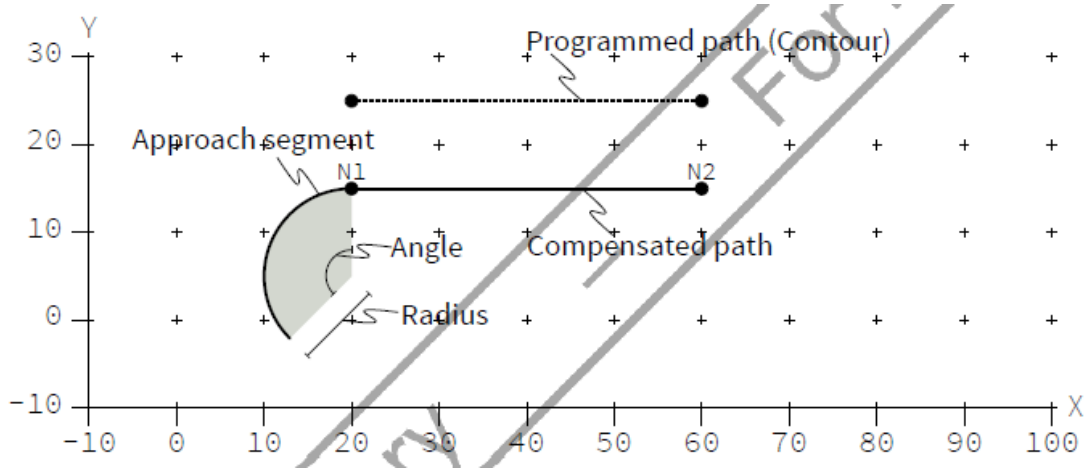


4.7.7 接近/离开行为

接近/离开行为描述激活 TRC 时接近与禁止 TRC 时离开的补偿路径。它避免由于低速导致轮廓上的燃烧印记。下面仅考虑接近行为，离开行为是相似的。

接近行为用参数 radius 和 angle 配置。这些参数定义一小段圆弧前置于补偿路径，如下图。圆弧的正切与 N1 的下一段相等。圆弧位于编程路径外。位于 N1 和 TRC 法向向量定义的平面内。

半径必须是非负值，角度可以是任意非负值，可以大于一个整圆。



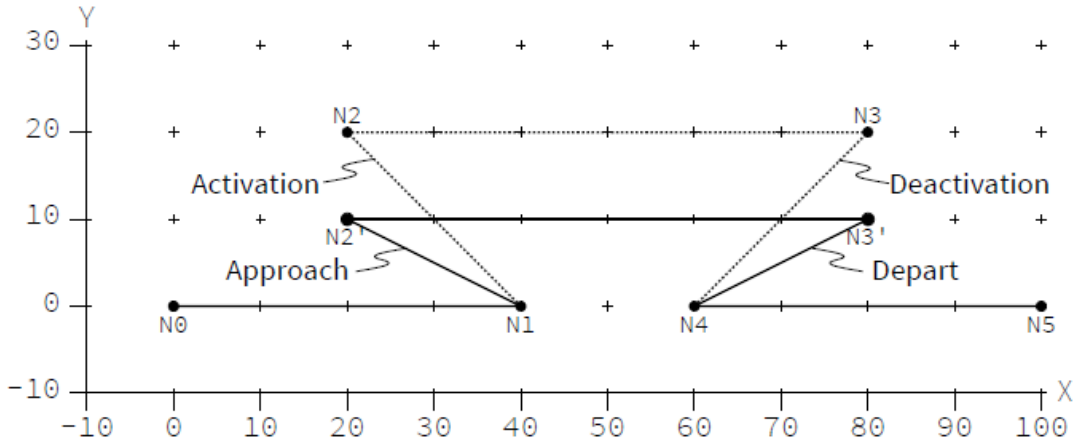
如果直径或角度为零，圆弧将被舍弃。补偿路径是直接进入，如下例所示。

例 1：

编程路径(虚线)和补偿路径(实线)如下例图中所示。TRC 在 N1 和 N2 间激活，N3 和 N4 间关闭。补偿路径以直线段 N1/N2'进入，直线段 N3'/N4 离开。尖角 N2 和 N3 会导致低速。例如加工木头时，会导致明显的烧伤痕迹。

```
N1 G01 X40 F200
!trcSet(nz:=-10);
N2 X20 Y20
N3 X80
!trcSet();
```

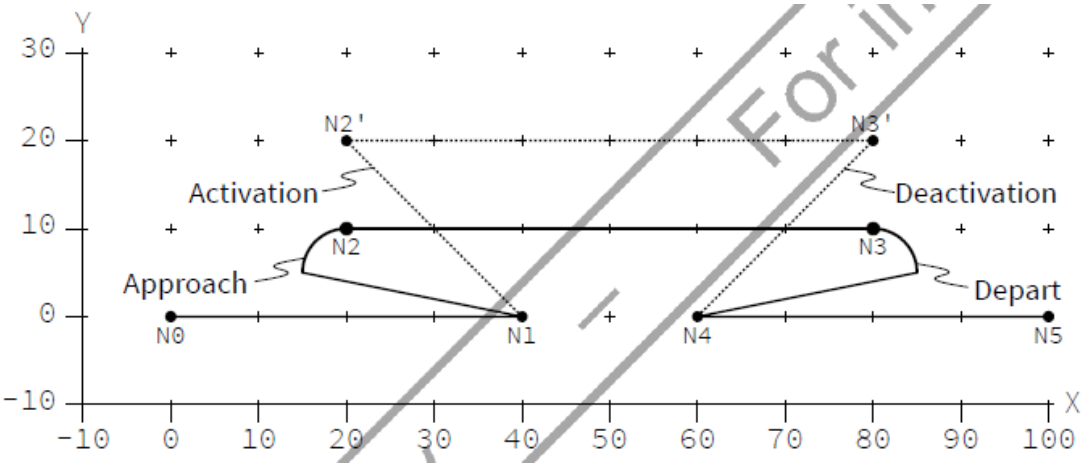
N4 X60 Y0
N5 X100



例 2：

编程路径(虚线)和补偿路径(实线)如下例图中所示。接近段定义为半径 5 和角度 90。刀具经过 N2 和 N3 时不会完全停下。以加速和减速移入进入和退出段。

N1 G01 X40 F200
!trcSet(nz:=-10, approachRadius:=5, approachAngle:=90, departRadius:=5, departAngle:=90);
N2 X20 Y20
N3 X80
!trcSet();
N4 X60 Y0
N5 X100



合适的半径与角度值取决于机器的动态性能和几何限制。一方面进入段需要足够长以在 N2 达到达到足够的速度。另一方面，它又要足够小以避免轮廓变形。
禁用 TRC 后，当前点设为退出段的终点。用户可通过 frameGet(…)检索。

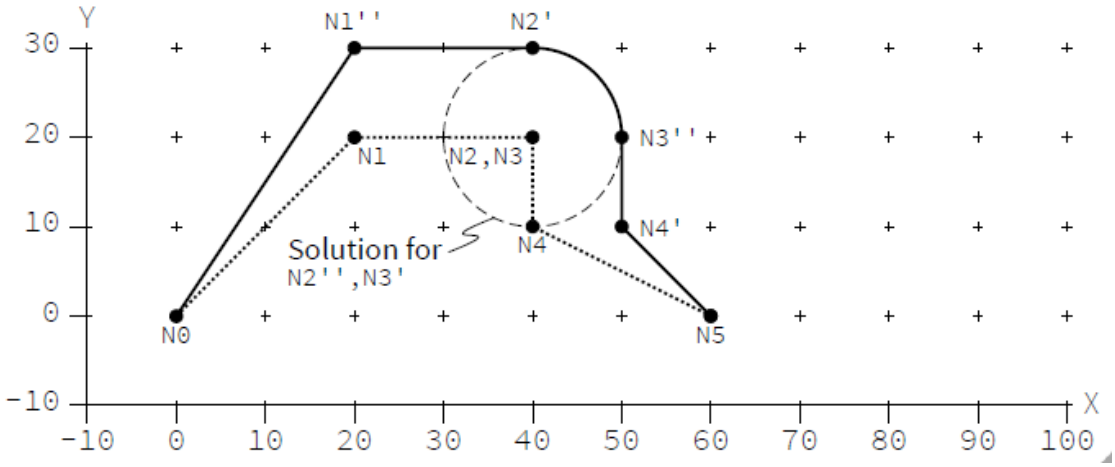
4.7.8 正交运动

TRC 激活时正交运动不是参考平面内的运动，因此不提供补偿的方向。

例:

编程路径(虚线)和补偿路径(实线)如下图。直线段 N2/N3 不是参考平面 XY 内的运动。映射段 N2''/N3'并不能从 N2/N3 唯一地生成。虚线圆中的任何位置都是可行的。TRC 把 N2'' 设为 N2‘，解决了这个不唯一的问题。从而正交运动绑定到之前段的终点。这个法则也适用于之后正交运动序列的编程。

```
!trcSet(nz:=10);
N1 G1 X20 Y20 F200
N2 X40
N3 Z10
N4 Y10
!trcSet();
N5 X60 Y0
```

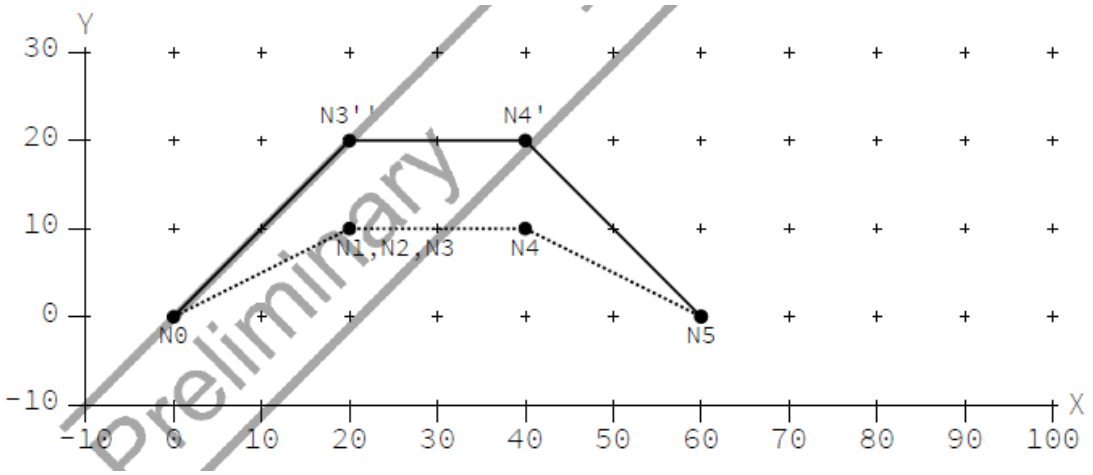


如 TRC 从一个正交运动开始，没有前面的补偿点。这种情况下使用后面第一个补偿点。需要的预读没有限制即可以以一个任意数量的正交运动开始。

例 1：

编程路径(虚线)和补偿路径(实线)如下图。编程路径 N1 到 N4 中 TRC 生效。N1/N2 与 N2/N3 正交于参考平面。N3/N4 段是第一个非正交段。将 N3'设为 N3''和 N2‘设为 N2''来解决正交运动。补偿段 N1''/N2'和 N2''/N3'绑定到 N3''。

```
!trcSet(nz:=10);
N1 G1 X20 Y10 F200
N2 Z10
N3 Z20
N4 X40
!trcSet();
N5 X60 Y0
```



TRC 下的路径可能包含任意位置任意数量的正交段。然而它不是唯一地包含正交运动。这种情况下补偿是不受限定的且会产生一个 runtime 的错误。

例 2：

下例中 TRC 有效路径下不包含非正交运动。它产生 runtime 错误。

```
!trcSet(nz:=10);
N1 G1 X20 Y10 F200
N2 Z10
N3 Z20
N4 Z30
!trcSet();
N5 X60 Y0
```

4.8 B 样条曲线

transBSpline 用于由一系列如 CADCAM 或小线段的分段折线生成一个连续的曲线。这个曲线以输入的多段线为边界，在起点和终点间插补，内部点是曲线的控制点(Deboor 点)，至少需要三个点。

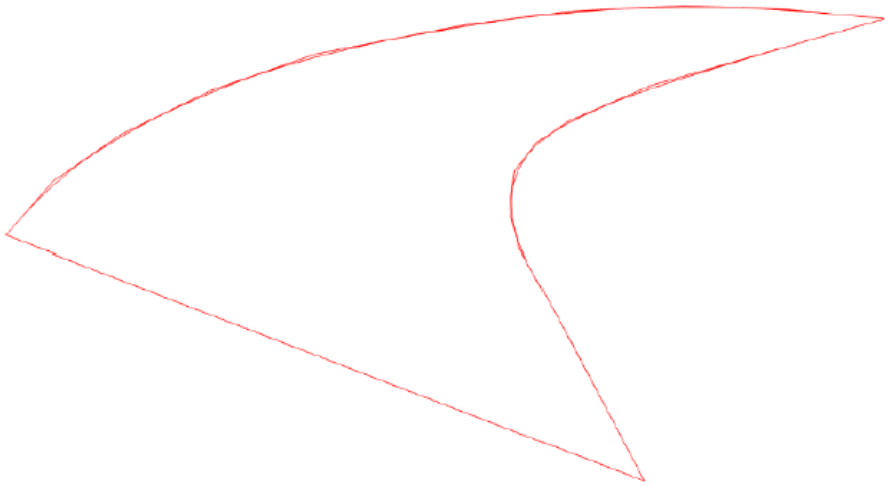
4.8.1 语法

transBSpline(BreakAngle, BreakLength, MergeDiff, LineBreakAngle, LineBreakLength, LineMergeDiff);

Enable: transBSpline(BreakAngle:=70, BreakLength:=1000);

Disable: transBSpline();

例：



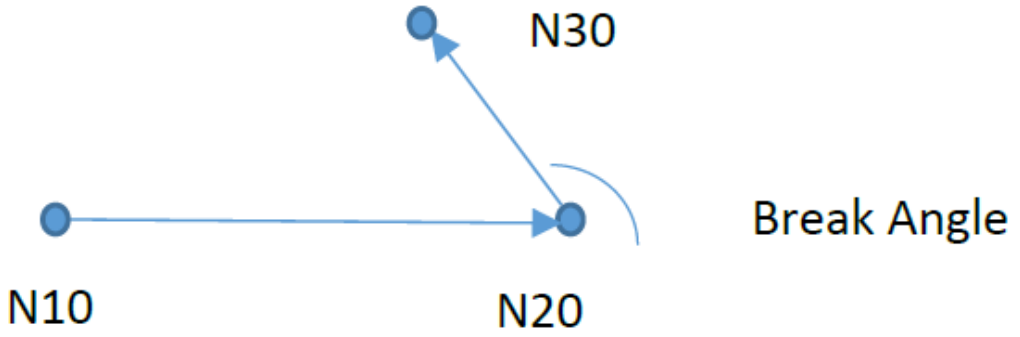
```
//Control Polygon
N10 G00 X18.498 Y0

N20 G01 X18.498 Y0 Z0 F6000
N30 X16.572 Y6.543 Z1
N40 X15.616 Y9.715 Z2
N50 X15.121 Y11.275 Z3
N60 X14.838 Y13.196 Z4
N70 X14.982 Y15.085 Z5
N80 X15.595 Y16.485 Z6
N90 X16.396 Y17.490 Z7
N100 X18.653 Y19.243 Z8
N110 X25.07 Y22.526 Z9
N120 X22.228 Y22.997 Z8
N130 X19.569 Y23.174 Z7
N140 X16.488 Y22.884 Z6
N150 X13.634 Y22.228 Z5
N160 X9.533 Y20.793 Z4
N170 X6.668 Y19.009 Z3
N180 X4.224 Y16.877 Z2
N190 X2.376 Y14.61 Z1
N200 X1.068 Y11.959 Z0
```

```
//BSpline
N10 G00 X18.498 Y0
!transBSpline(BreakAngle:=70.0, BreakLength:=1000.0);
N20 G01 X18.498 Y0 Z0 F6000
N30 X16.572 Y6.543 Z1
N40 X15.616 Y9.715 Z2
N50 X15.121 Y11.275 Z3
N60 X14.838 Y13.196 Z4
N70 X14.982 Y15.085 Z5
N80 X15.595 Y16.485 Z6
N90 X16.396 Y17.490 Z7
N100 X18.653 Y19.243 Z8
N110 X25.07 Y22.526 Z9
N120 X22.228 Y22.997 Z8
N130 X19.569 Y23.174 Z7
N140 X16.488 Y22.884 Z6
N150 X13.634 Y22.228 Z5
N160 X9.533 Y20.793 Z4
N170 X6.668 Y19.009 Z3
N180 X4.224 Y16.877 Z2
N190 X2.376 Y14.61 Z1
N200 X1.068 Y11.959 Z0
! transBSpline();
```

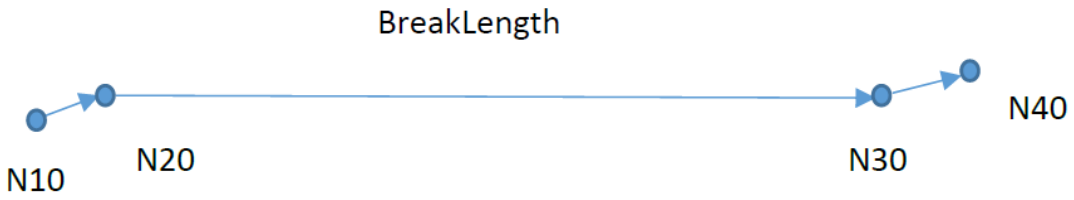
参数中如果 BreakAngle 或 BreakLength 是零，则无处理。另外 LineBreakAngle，LineBreakLength 和 LineMergeDiff 首先得到处理用于简化控制点折线。BreakAngle，BreakLength 和 MergeDiff 最后处理用于生成 B 样条曲线。具体参数含义见下节。

4.8.2 参数

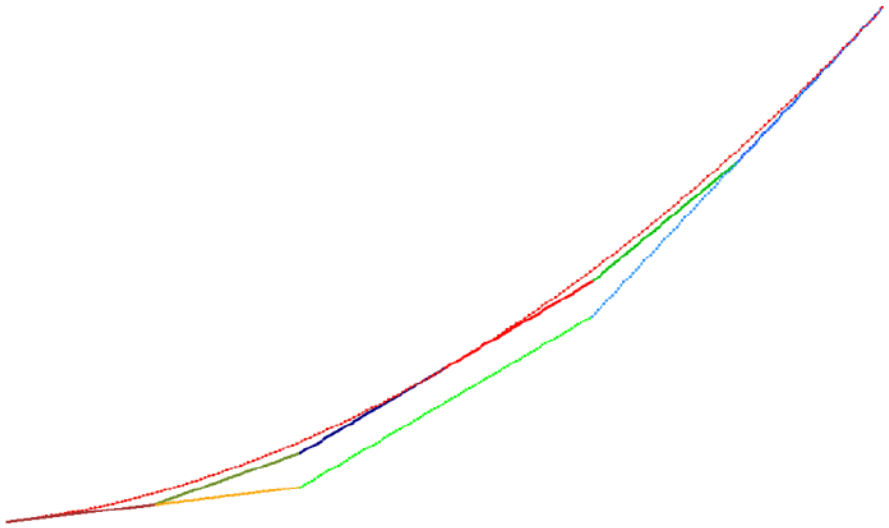
BreakAngle: 允许保留路径中的锐角特征。当路径偏差超过 BreakAngle 时样条曲线将断开。样条曲线将终止并插值该点。



BreakLength: 允许保留路径中的长度特征。当段长度超过 BreakLength 时样条曲线将断开。样条曲线将终止并插值长段的起点和终点。



MergeDiff: BSpline 由贝塞尔段构成。为了提高处理速度，样条曲线可以通过合并来压缩。当控制点间的差距小于 MergeDiff 时相邻段将被合并在一起。下面的相邻段被合并成一条。

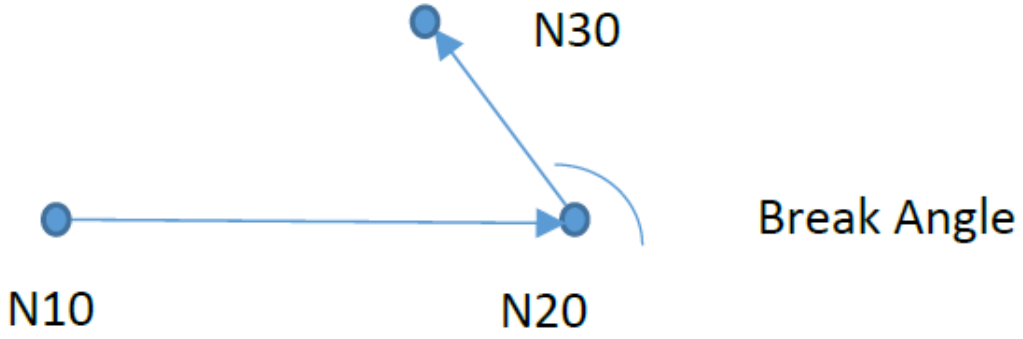


注意：

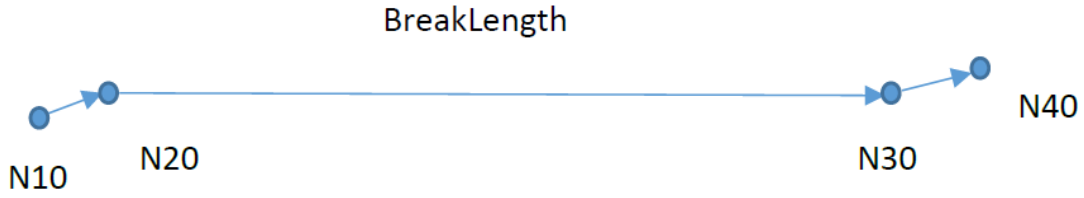
- 过于激进的合并会导致过度的扭曲。
- 曲率过大的段将被以 runtime error 被拒绝。
- 可接受的曲率由轨迹速度和加速度生成。

BSpline 由如 CADCAM 产生的 G01 线段形成的控制点多边形构成。为了提高处理速度，控制点多边形可以靠合并相邻段来压缩和简化。

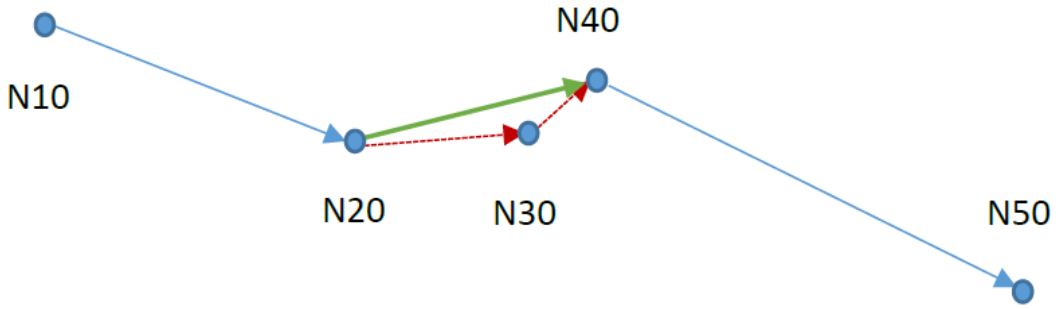
LineBreakAngle: 如果角度偏差超过 LineBreakAngle 将打断合并相邻控制点。



LineBreakLength: 如果长度超过 LineBreakLength 合并相邻控制点将被打断。



LineMergeDiff: 如果变化 (垂直距离)小于 LineMergeDiff 相邻控制点将被合并。例中 N30 可被删掉以简化控制多边形。



警告：

- 过于激进的合并会导致过度的扭曲。
- 过度弯曲的段将因 runtime error 被拒绝。
- 以轨迹速度和加速度产生可接受的曲率。

中止预读和握手型 M 函数：

B 样条用!transBSpline()中止，优先于中止预读既 decoder stop 或握手型 M 函数。

```
//BSpline
N10 G00 X18.498 Y0
!transBSpline(BreakAngle:=70.0, BreakLength:=1000.0);
N20 G01 X18.498 Y0 Z0 F6000
N30 X16.572 Y6.543 Z1
N40 X15.616 Y9.715 Z2
N50 X15.121 Y11.275 Z3
N60 X14.838 Y13.196 Z4
N70 X14.982 Y15.085 Z5
N80 X15.595 Y16.485 Z6
N90 X16.396 Y17.490 Z7
N100 X18.653 Y19.243 Z8
N110 X25.07 Y22.526 Z9
!transBSpline();
!sync();
!transBSpline(BreakAngle:=70.0, BreakLength:=1000.0);
N120 X22.228 Y22.997 Z8
N130 X19.569 Y23.174 Z7
N140 X16.488 Y22.884 Z6
N150 X13.634 Y22.228 Z5
N160 X9.533 Y20.793 Z4
N170 X6.668 Y19.009 Z3
N180 X4.224 Y16.877 Z2
N190 X2.376 Y14.61 Z1
N200 X1.068 Y11.959 Z0
! transBSpline();
```

4.8.3 B 样条支持的 G 代码和函数

B 样条支持除 G01 以外的 G 代码和函数。如下:
G00 G60
G02, G03 圆和螺旋线。B 样条将先终止后继续。
G04
G09
G54 和变换
disableMask()
runFile(path:=)
smoothingSet(mainType:=smoothingTwinBezier, subType:=smoothingRadius, value:=)
不支持刀具半径补偿。