算法设计

[1 文档概述 4](#_Toc451342763)

[1.1 版本更新内容 4](#_Toc451342764)

[1.2 文档内容介绍 4](#_Toc451342765)

[2 坐标变换 4](#_Toc451342766)

[2.1 比例缩放 4](#_Toc451342767)

[2.2 镜像 5](#_Toc451342768)

[2.3 极坐标 6](#_Toc451342769)

[2.3.1 极坐标增量指令 6](#_Toc451342770)

[2.3.1.1 增量模式下指定极径指令 7](#_Toc451342771)

[2.3.1.2 增量模式下指定极角指令 7](#_Toc451342772)

[2.3.2 极坐标绝对指令 8](#_Toc451342773)

[3 刀具补偿功能 9](#_Toc451342774)

[3.1 刀具长度补偿 9](#_Toc451342775)

[3.2 刀具半径补偿 9](#_Toc451342776)

[3.2.1 C类刀具半径补偿 9](#_Toc451342777)

[3.2.2 刀具半径补偿的建立和取消 10](#_Toc451342778)

[3.3 刀具补偿流程设计 12](#_Toc451342779)

[3.3.1 刀具补偿输入输出接口 13](#_Toc451342780)

[3.3.1.1 输入接口 13](#_Toc451342781)

[3.3.1.2 输出接口 13](#_Toc451342782)

[3.3.2 刀补模块类图 15](#_Toc451342783)

[3.4 刀具补偿算法设计 15](#_Toc451342784)

[3.4.1 直线与直线刀补后交点计算 15](#_Toc451342785)

[3.4.2 直线与圆弧刀补后交点计算 17](#_Toc451342786)

[3.4.3 圆弧和圆弧刀补后交点计算 17](#_Toc451342787)

[4 圆弧过渡 18](#_Toc451342788)

[4.1 线段-线段过渡 20](#_Toc451342789)

[4.2 线段-圆弧过渡： 20](#_Toc451342790)

[4.3 圆弧-线段过渡 23](#_Toc451342791)

[4.4 圆弧-圆弧过渡 23](#_Toc451342792)

[4.4.1 外接 23](#_Toc451342793)

[4.4.2 内接 24](#_Toc451342794)

[4.4.3 混合（内外）连接 25](#_Toc451342795)

[4.4.4 无解情况 26](#_Toc451342796)

[5 前瞻 26](#_Toc451342797)

[5.1 前瞻目的 26](#_Toc451342798)

[5.2 需要考虑的主要因素 27](#_Toc451342799)

[5.3 前瞻算法描述 27](#_Toc451342800)

[5.3.1 预处理 27](#_Toc451342801)

[5.3.2 速度校验 29](#_Toc451342802)

[5.4 前瞻算法公式 29](#_Toc451342803)

[5.4.1 最大加速度和停止加速度条件 29](#_Toc451342804)

[5.4.2 相邻段夹角条件 30](#_Toc451342805)

[5.4.3 弓高误差条件 30](#_Toc451342806)

[5.4.4 向心加速度条件 30](#_Toc451342807)

[5.5 前瞻处理 30](#_Toc451342808)

[5.5.1 前瞻处理分类 30](#_Toc451342809)

[5.5.1.1 预处理 30](#_Toc451342810)

[5.5.1.2 综合处理 30](#_Toc451342811)

[5.6 流程图 31](#_Toc451342812)

[5.7 上下文接口 32](#_Toc451342813)

[5.8 前瞻函数及其说明 33](#_Toc451342814)

[5.8.1 Process函数 33](#_Toc451342815)

[5.8.2 LookAheadHandle函数 33](#_Toc451342816)

[5.8.3 PrepareProcess函数 33](#_Toc451342817)

[5.8.4 EndProcess函数 34](#_Toc451342818)

[5.8.5 GetLookAheadData函数 34](#_Toc451342819)

[5.8.6 ForceOneProcess函数 34](#_Toc451342820)

[5.8.7 NormalProcess函数 34](#_Toc451342821)

[5.8.8 CalcSpeed函数 34](#_Toc451342822)

[5.8.9 AngleSpeed函数 35](#_Toc451342823)

[5.8.10 Get3DArcVector函数 35](#_Toc451342824)

[5.8.11 Distant函数 35](#_Toc451342825)

[5.8.12 DistantCircle函数 35](#_Toc451342826)

[6 插补 36](#_Toc451342827)

[6.1 插补模块结构和流程 36](#_Toc451342828)

[6.1.1 插补模块结构图 36](#_Toc451342829)

[6.1.2 插补模块流程图 36](#_Toc451342830)

[6.2 速度规划 39](#_Toc451342831)

[6.2.1 S型曲线速度规划 40](#_Toc451342832)

[6.2.1.1 S型曲线参数 40](#_Toc451342833)

[6.2.1.2 速度规划类型 40](#_Toc451342834)

[6.2.1.3 S型曲线规划公式表达 41](#_Toc451342835)

[6.2.2 T型曲线速度规划 44](#_Toc451342836)

[6.3 核心算法说明 44](#_Toc451342837)

[6.3.1 快速移动 44](#_Toc451342838)

[6.3.2 直线插补 45](#_Toc451342839)

[7 补偿 45](#_Toc451342840)

[7.1 补偿实体类 46](#_Toc451342841)

[7.2 补偿装饰类基类 48](#_Toc451342842)

[7.3 反向间隙补偿 49](#_Toc451342843)

[7.3.1 反向间隙参数 49](#_Toc451342844)

[7.3.2 反向间隙补偿装饰类 49](#_Toc451342845)

[7.3.3 反向间隙补偿流程 51](#_Toc451342846)

[7.4 螺距误差补偿 53](#_Toc451342847)

[7.4.1 螺距误差补偿参数 53](#_Toc451342848)

[7.4.2 螺距误差补偿装饰类 54](#_Toc451342849)

[7.4.3 螺距误差补偿流程 56](#_Toc451342850)

[7.5 调用说明 59](#_Toc451342851)

[7.6 螺距补偿原理 59](#_Toc451342852)

[7.6.1 正向补偿 59](#_Toc451342853)

[7.6.2 反补偿 61](#_Toc451342854)

# 文档概述

## 版本更新内容

表 1 版本更新情况

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 作者 | 更新状况 |
| R1.0 | 2016.05 | 何敏聪 | 创建汇总 |
|  |  |  |  |

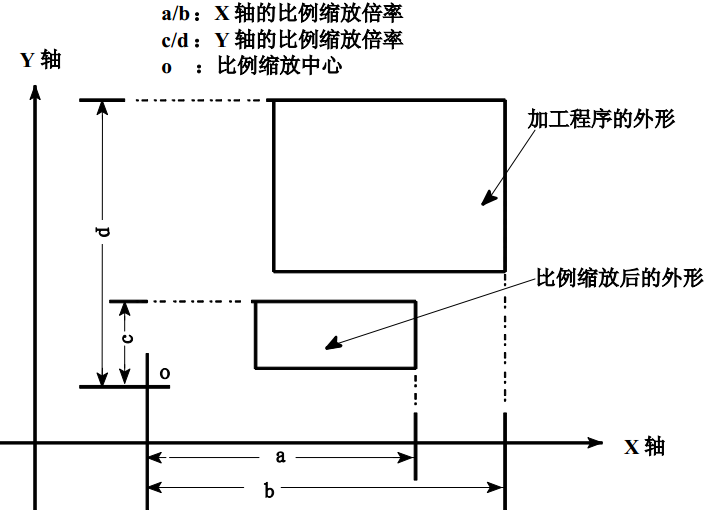
## 文档内容介绍

AE8100的算法相关功能，按处理顺序，包括坐标变化，刀补，过渡，前瞻，插补和补偿功能。本文档主要从设计原理上对以上功能进行说明。

# 坐标变换

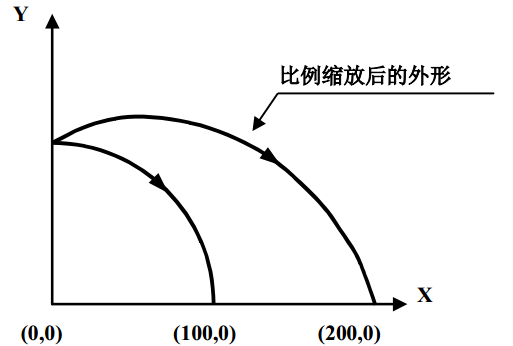
## 比例缩放

起点不变，终点、半径、圆心矢量相对各轴按指定比例进行缩放。



每个轴按比例缩放

圆弧按比例缩放时，由于终点改变，所以会变成螺旋线轨迹，按螺旋线插补

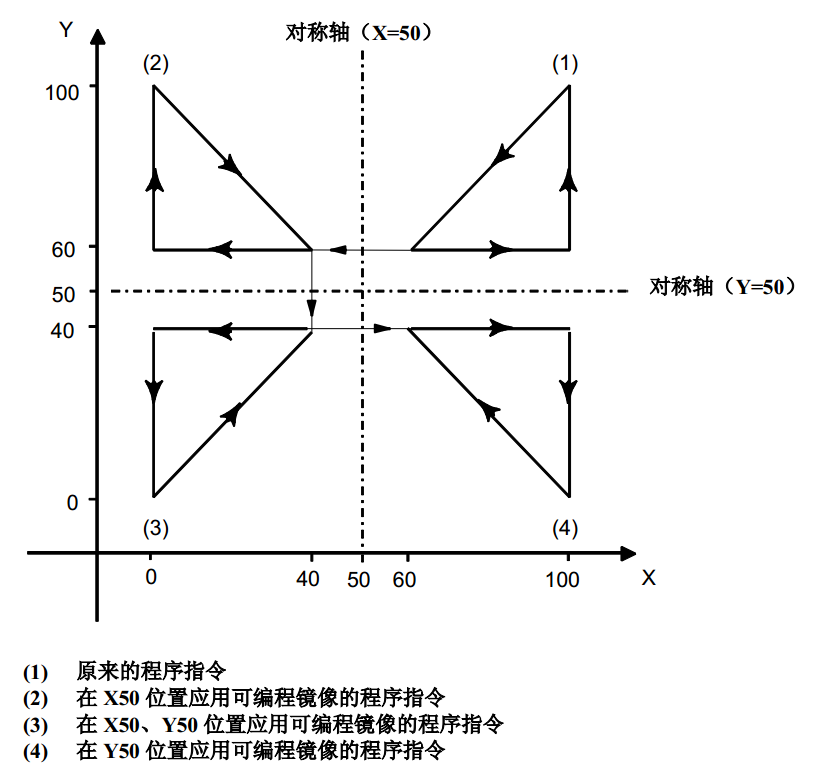


圆弧比例缩放

比例值可取负值，当缩放比例为负时，缩放同时，相对于同一坐标轴进行镜像处理。

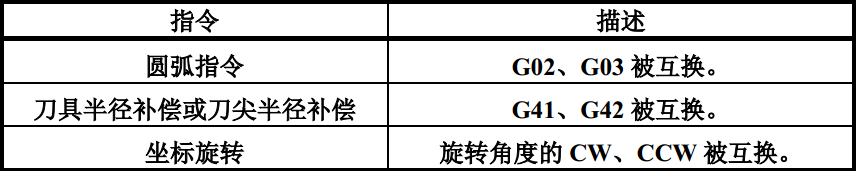
## 镜像

相对指定轴作镜像处理



镜像

进行镜像时，下列指令会被改变。



## 极坐标

极坐标功能具体使用可参见西门子手册，AE8100的极坐标使用参照西门子的做法，但是西门子手册中没有对一些细节进行进一步解释，经过在西门子808D上验证，增加了极坐标的补充说明。

### 极坐标增量指令

#### 增量模式下指定极径指令

如果当前是增量模式，指令中指定了极径，则以当前点作为极坐标的中心。



N1 G90X100.Y45.

N2 G91X100.Y15.

如果当前是增量模式，指令中没有指定极径，则极坐标中心立即变回原点。



N1 G90X100.Y45.

N2 G91X100.Y15.

N3 Y30.

#### 增量模式下指定极角指令

如果当前是增量模式，指令中指定了极角，则指定的极角值作为极角增量使用。



N1 G90X100.Y45.

N2 G91X100.Y15.

如果当前是增量模式，指令中没有指定极角，则上一个指令中的极角值作为本次指令的增量值使用。



N1 G90X100.Y45.

N2 G91X100.

### 极坐标绝对指令

在任何情况下，只要出现G90指令，则极径和极角立即变为绝对指令。



N1 G90X100.Y45.

N2 G91X100.Y15.

N3 G90X100.

# 刀具补偿功能

## 刀具长度补偿

刀具长度补偿功能是将刀具实际长度值和假定的刀具长度值的差值作为刀具长度方向的补偿值，对加工工件进行加工。这样在更换刀具时也不必对加工程序进行修改，如图2-1所示。

图 2‑1

差值为偏置值

## 刀具半径补偿

### C类刀具半径补偿

刀具半径补偿分为B类刀具半径补偿（简称B刀补）和C类刀具半径补偿（简称C刀补）两种。其中B刀补根据本段程序的轮廓尺寸进行刀补，不能解决程序段之间的过渡问题，编程人员必须先估计可能出现的间断点和交叉点进行人为处理，加工编程非常不方便。在本系统中采用C刀补，以下不说明的情况，刀补即为C刀补。

针对B刀补的缺陷，C刀补采用提前读取下一段程序，根据下一段程序的转接情况，对本段的刀补轨迹做适当的修正，得到正确的加工轨迹。根据转接角的类型，C刀补可以分为三种：缩短型，伸长型和插入型。以下以直线与直线的转接为例进行说明。对于直线与圆弧，以及圆弧与圆弧的转接角的计算，只需要再转接处用圆弧的切线方向代替圆弧方向即可。

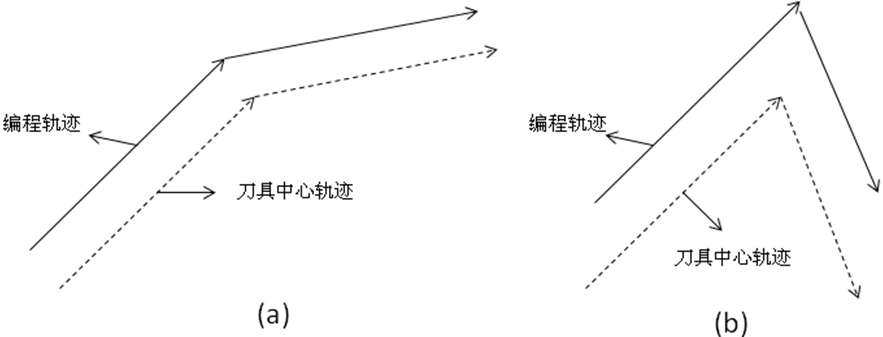
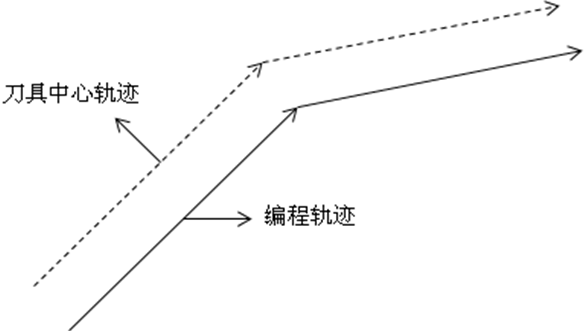
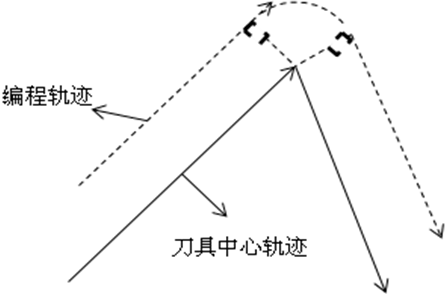
如图所示，加工内轮廓归为缩短型；加工外轮廓并且前后两段轨迹向量夹角小于时为伸长型；加工外轮廓并且前后两段轨迹向量夹角大于时为插入型，在转接处需要插入一段圆弧作为过渡。

图 2‑4 插入型

图 2‑2 缩短型

图 2‑3 伸长型

### 刀具半径补偿的建立和取消

刀补的建立和取消分为A，B，C三种，首先介绍A类型刀补建立方法。

A类型刀具半径补偿的建立：

|  |  |  |
| --- | --- | --- |
| **转接类型** | **直线🡪直线** | **直线🡪圆弧** |
| **缩短型** | 刀具中心轨迹  编程轨迹 | 刀具中心轨迹  编程轨迹 |
| **伸长型** | 编程轨迹  刀具中心轨迹 | 编程轨迹  刀具中心轨迹 |
| **插入型** | 编程轨迹  刀具中心轨迹 | 编程轨迹  刀具中心轨迹 |

A类刀补取消方法：

|  |  |  |
| --- | --- | --- |
| **转接类型** | **直线🡪直线** | **圆弧🡪直线** |
| **缩短型** | 刀具中心轨迹  编程轨迹 | 编程轨迹  刀具中心轨迹 |
| **伸长型** | 编程轨迹  刀具中心轨迹 | 刀具中心轨迹  编程轨迹 |
| **插入型** | 编程轨迹  刀具中心轨迹 | 编程轨迹  刀具中心轨迹 |

## 刀具补偿流程设计

### 刀具补偿输入输出接口

#### 输入接口

刀补类ToolCompensation被坐标变换模块调用，对外接口为uint8\_t ToolCompensate(CompileData\* data, double start\_point\_update[], uint8\_t proc\_type)，在此函数中对CompileData数据进行刀补补偿修改其终点坐标值等相关信息，然后传递给圆弧过渡模块。

在CompileData结构体中，成员变量ToolCompensationInfo tool\_compensation\_info包含刀补的主要输入信息：

|  |  |  |
| --- | --- | --- |
| **变量名** | **类型** | **详细** |
| r\_compensation\_type | uint16\_t | 半径补偿指令，对应枚举指令G40/G41/G42 |
| l\_compensation\_type | uint16\_t | 长度补偿指令，对应枚举指令G43/G44/G49 |
| tool\_no | uint32\_t | 刀具号，据此判断应该用何种刀具补偿算法 |
| h\_code | uint32\_t | 长度偏置号，据此从刀库中得到长度偏置值 |
| d\_code | uint32\_t | 半径补偿值号，据此从刀库中得到刀具半径值 |

在编译器解析每行G代码时，上述变量都有指定值。如果G代码中没有指令的变化，则这些变量值默认和上一行G代码中的值相同。

除以上刀补指令外，刀补模块还会用到CompileData结构体中的其他成员变量信息，如CoordInfo coord\_info，指明当前加工所在平面，用来判断刀轴的方向。

#### 输出接口

刀补模块根据输入接口数据CompileData得到当前刀补状态，刀具长度信息和半径信息，以及刀轴方向，然后根据这些信息进行相应的刀补计算。重新计算后用刀具中心的坐标轨迹代替编程轨迹，发送至下个模块。具体刀补模块修改CompileData中以下变量：

|  |  |  |
| --- | --- | --- |
| 变量名称 | 类型 | 详细 |
| CompileData. axis\_mov\_data. dest\_pos | double[MAX\_AXIS\_NUM] | 编程轨迹的终点坐标 |
| CompileData. axis\_mov\_data. dist | double | 曲线轨迹的长度 |
| CompileData. extra\_mov\_data. extra\_data[I] | double | 仅圆弧插补：圆弧圆心的X轴坐标 |
| CompileData. extra\_mov\_data. extra\_data[J] | double | 仅圆弧插补：圆弧圆心的Y轴坐标 |
| CompileData. extra\_mov\_data. extra\_data[K] | double | 仅圆弧插补：圆弧圆心的Z轴坐标 |
| CompileData. extra\_mov\_data. extra\_data[I1] | double | 仅圆弧插补：圆弧所在平面的法向量X坐标 |
| CompileData. extra\_mov\_data. extra\_data[J1] | double | 仅圆弧插补：圆弧所在平面的法向量Y坐标 |
| CompileData. extra\_mov\_data. extra\_data[K1] | double | 仅圆弧插补：圆弧所在平面的法向量Z坐标 |
| CompileData. extra\_mov\_data. extra\_data[R] | double | 仅圆弧插补：圆弧的半径 |
| CompileData. extra\_mov\_data. extra\_data[AR] | double | 仅圆弧插补：圆弧的张角 |

### 刀补模块类图



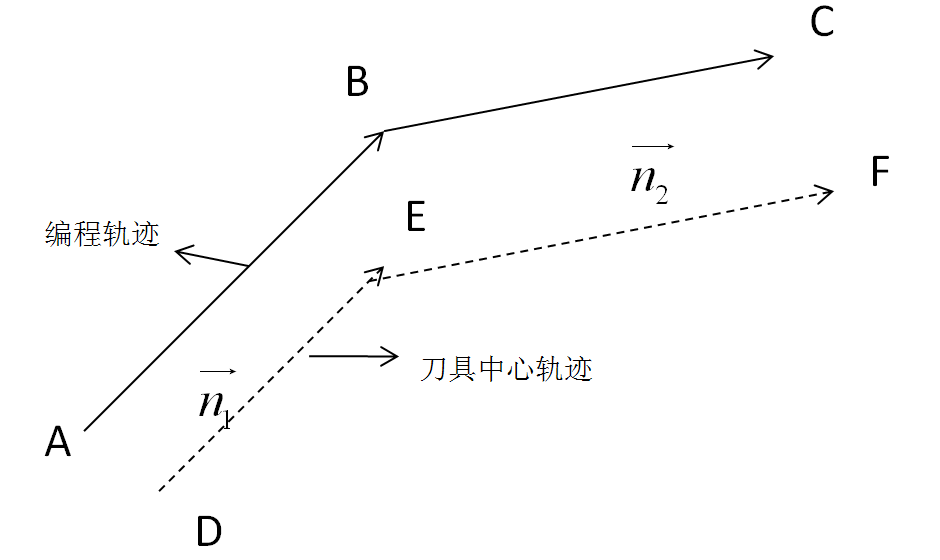
## 刀具补偿算法设计

刀具补偿算法主要在ToolCompAlgorithm类及其子类中调用。在伸长型和缩短型转接处，直接计算两条曲线刀补后轨迹的交点；在插入型转接处，在两条曲线刀补后轨迹中间插入一条圆弧。

插入圆弧时计算量较少，圆弧的起点和终点即为第一条曲线终点刀补后位置和第二条曲线起点刀补后位置，圆心即为两条曲线刀补前交点，在此不做详细介绍。下面主要介绍如何计算两条曲线的交点。

### 直线与直线刀补后交点计算

如图所示，三维空间两条直线AB和BC，交于B点，它们刀补后的轨迹分别为直线DE和EF，交于E点。求E点坐标。



根据A，B，C点坐标可以计算出两条直线的单位方向向量和。则

 公式 1

 公式 2

其中，。根据公式1和公式2可得：

 公式 3

由公式3得：

公式 4

即：

 公式 5

设：

 公式 6

公式6计算可得：

 公式 7

如果，则两条直线共线。如果方向相同，则刀补后交点为第一条直线终点刀补后的位置，如果方向相反，为插入型转接，需要在转接处出入圆弧。

根据求得s的值代入公式1即可求出E点坐标。

### 直线与圆弧刀补后交点计算

如图所示，直线AB与圆弧BC交于B点，圆弧的圆心为O点。刀补后直线与圆弧交于F点，过O点向直线AB做垂线交于D点，交AB刀补后的直线与D点。求F点坐标。

（1）求向量。首先求向量单位向量。，其中为平面法向量，也为刀轴矢量，由刀补信息给出。为的单位方向矢量。和夹角应小于，即如果，则应取反向。

O

A

B

C

D

E

F



由，求出OE的长度即可得到向量。，向量可求，最终得到向量。

（2）求向量。的方向即为直线刀具半径补偿方向，此方向可以根据直线运动方向和左刀补还是右刀补判断得到。的长度ED为直线刀补半径。根据可求得向量。

（3）求DF长度。。

（4）求EB的方向。。

（5）求向量。。向量和向量同向，如果，则取反。

（6）求F点坐标。。

对于圆弧与直线刀补后的交点计算，与直线与圆弧刀补后交点计算相似，在此不做赘述。

### 圆弧和圆弧刀补后交点计算

如图，圆弧MA与圆弧AN交于A，其刀补后的轨迹交于D。B和C分别为A和D对应的另外一个圆弧交点。AB的中点为E，CD的中点为F。求D点坐标。

1. 求向量。首先求OF的长度。如果向量和向量的夹角小于，则，否则。根据几何关系可知：，即。据此可以求出OF的长度。

A

B

C

D

O

P

E

F



M

N

然后求OF的方向。容易求出OP的单位方向向量。如果向量和向量的夹角小于，则与同向；否则二者反向。

1. 求向量。首先求平行于FD的单位方向向量。与的夹角应小于，如果，则取反向。

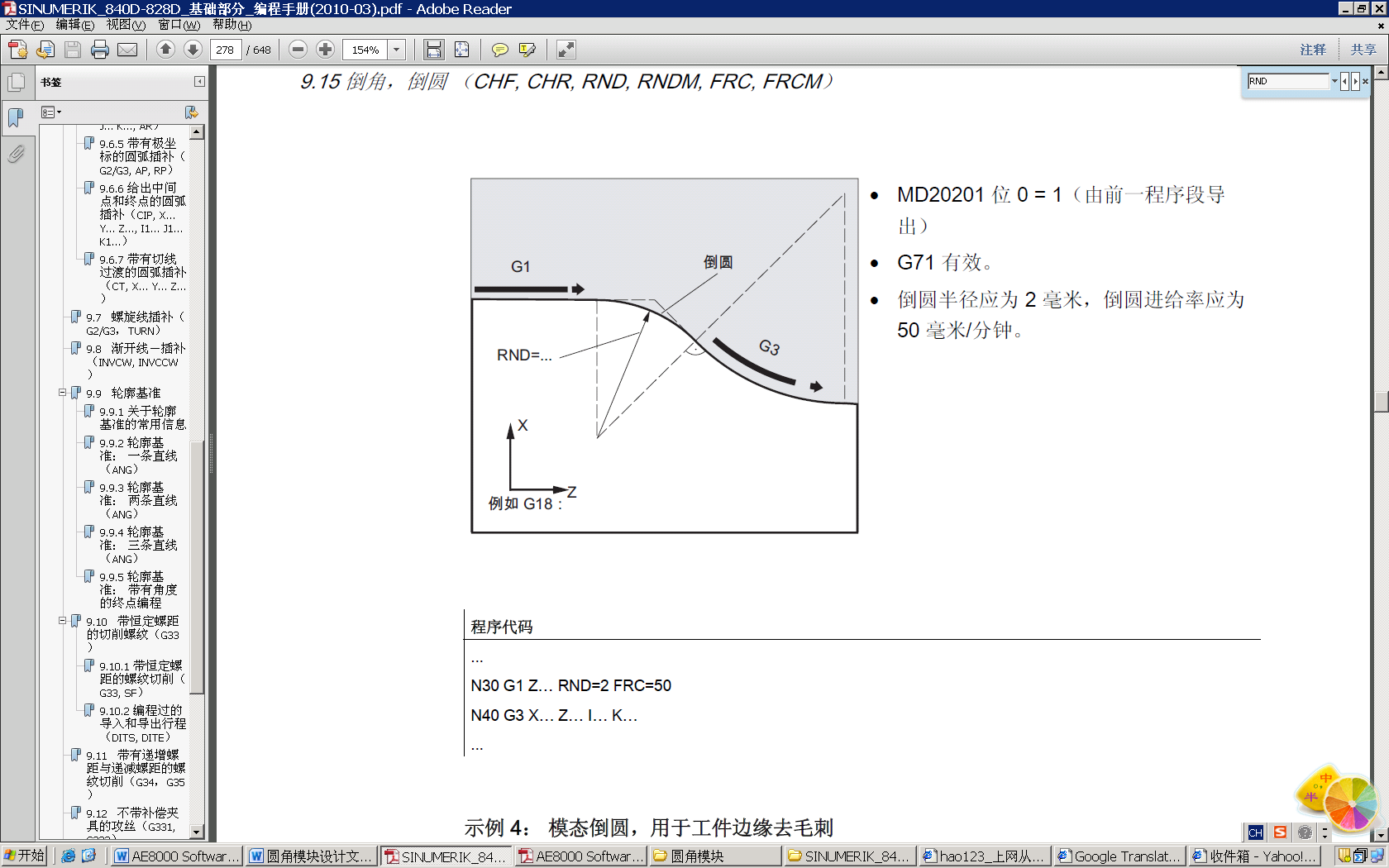
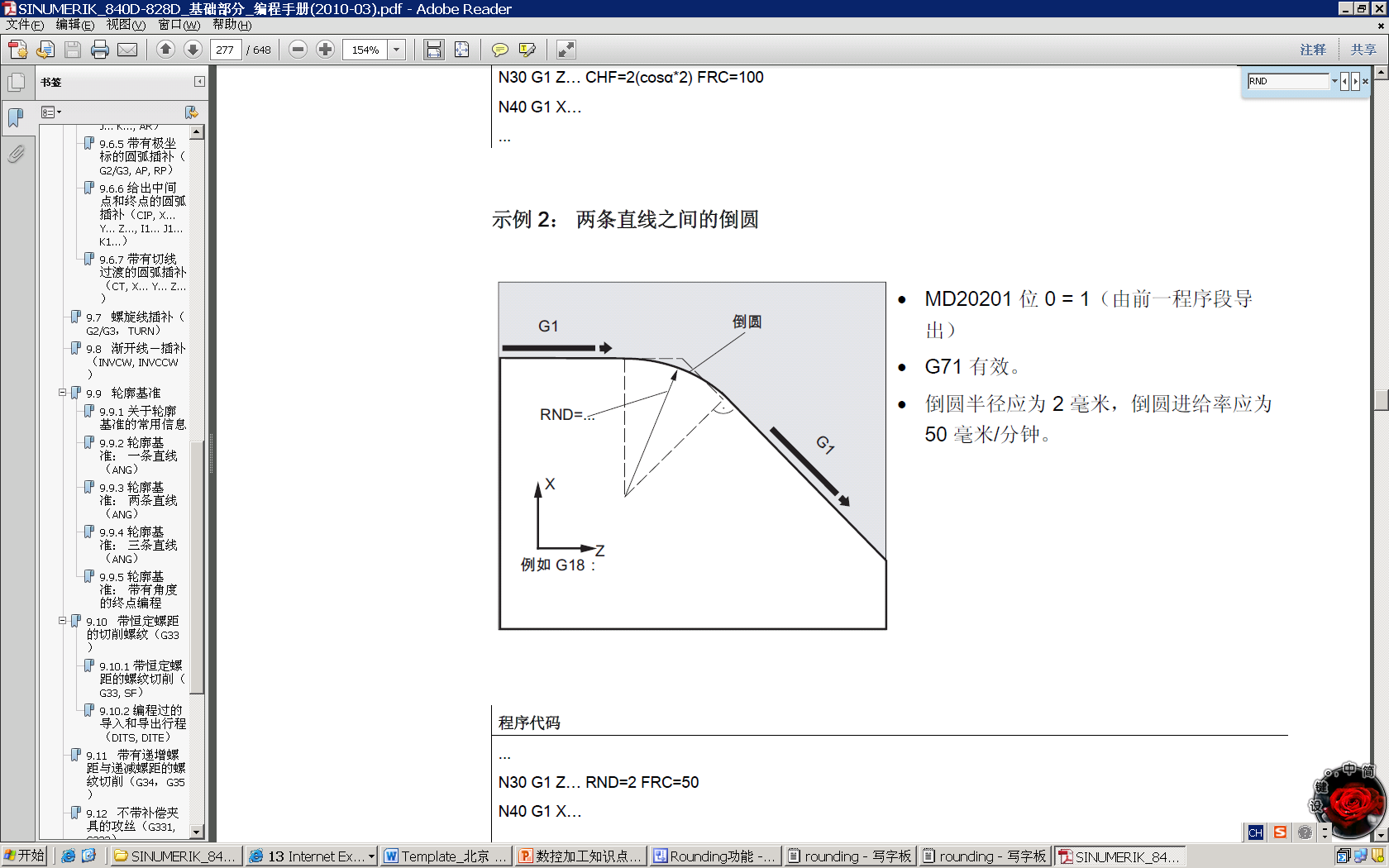
然后求FD的长度。。

根据求得向量。

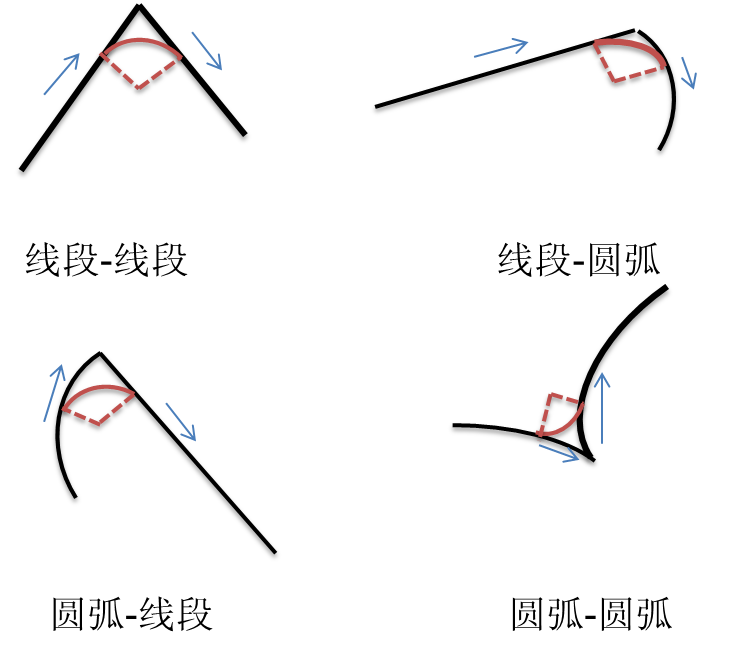
1. 最后求D点坐标。

# 圆弧过渡

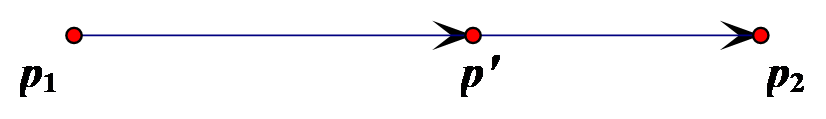
圆弧过渡主要用于实现加工段之间的圆滑过渡。做法是将加工路径上的尖角用圆角替换，从而使零件轮廓圆滑，表面质量提高，同时避免反复的加减速过程，从而提高加工效率和加工表面质量。



过渡分类：



点分线段成比例：





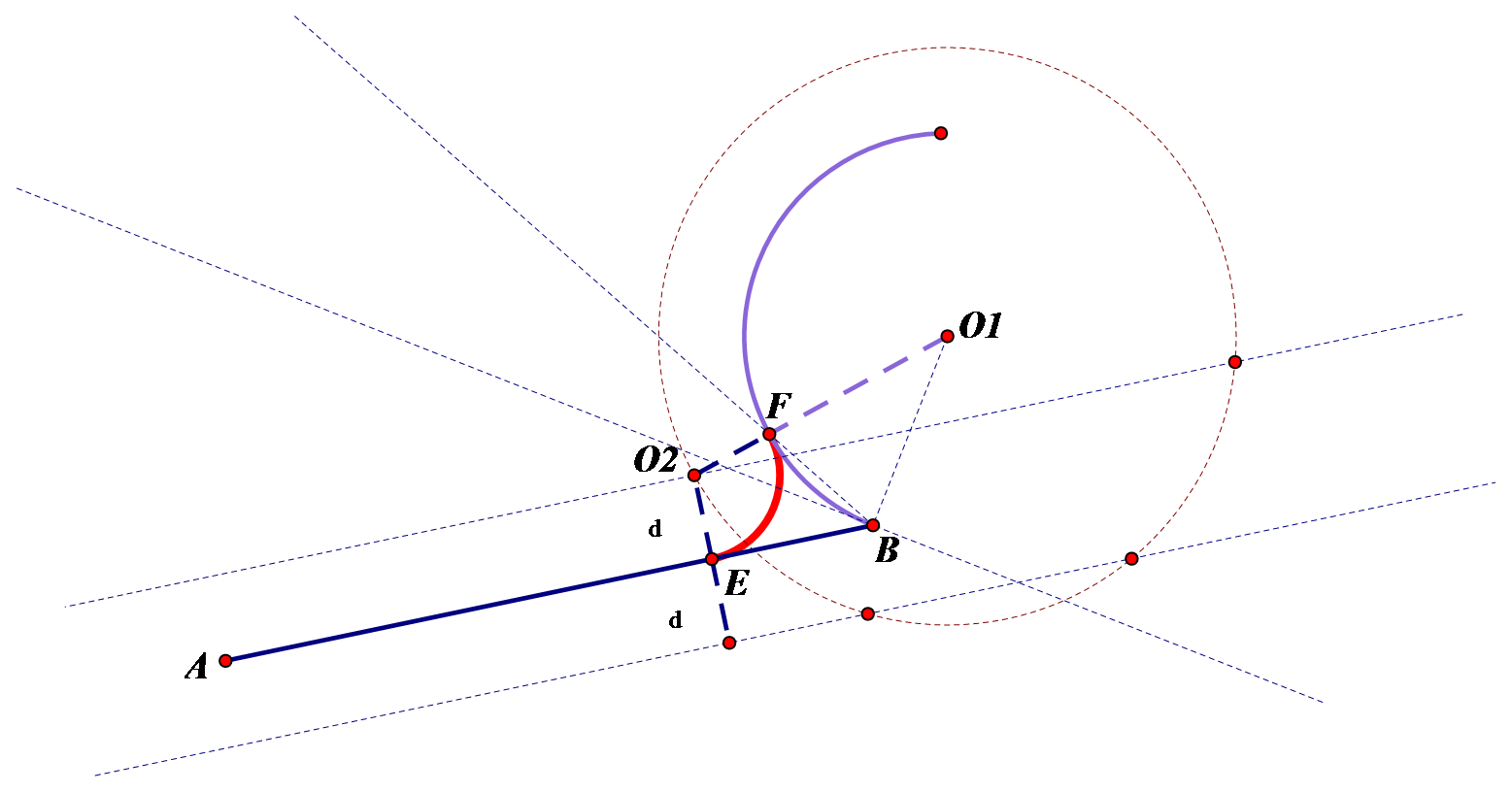
## 线段-线段过渡



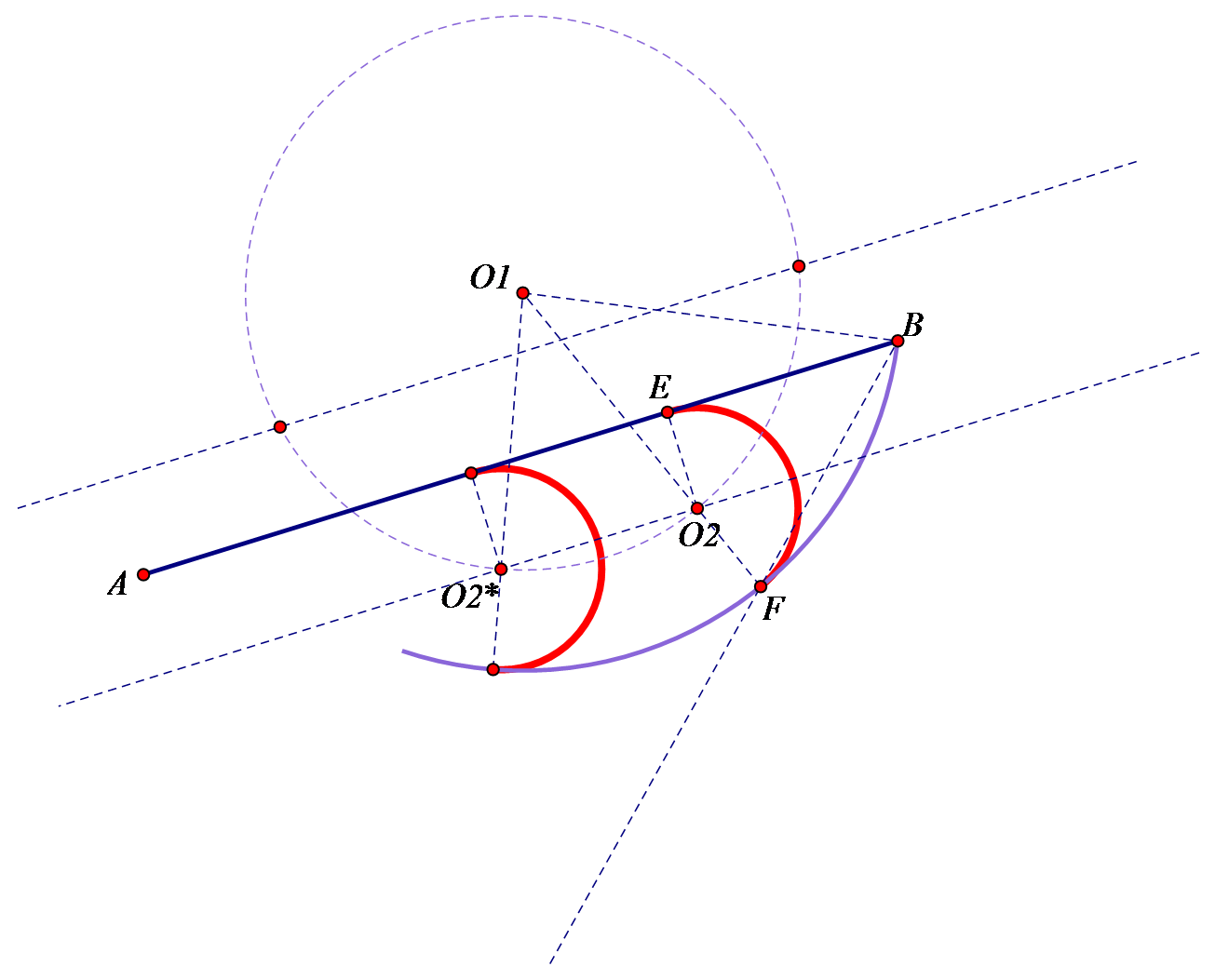


可以证明过渡圆弧所对的圆心角。

## 线段-圆弧过渡：



外接



内切

分为外接和内切两种情况：

外接:

判断条件：



圆心距：



过渡圆弧从与直线的切点开始到与圆弧的切点结束，方向与已知圆弧方向相反。

内切：

判断条件：



圆心距：



公共条件：

过渡圆心到直线的距离：



过渡圆弧从与直线的切点开始到与圆弧的切点结束，方向与已知圆弧方向相同。

过渡圆圆心坐标为未知量，其余均为已知量，联立圆心距与公共条件两个方程，可以解出过渡圆圆心坐标共四组解。

筛选条件：

连结，以弧上从B到C的方向为正方向，取构成的角最小的解即为所求。

根据圆心坐标，可以求出切点E，F的坐标。可以证明过渡圆弧所对的圆心角。以为圆心，E，F为端点作劣弧，即为所求。

无解情况：

1. 求得BE长度大于AB，证明线段AB过短，对于制定过渡半径无法完成圆弧过渡。
2. 求得 圆弧上切点F，但圆心角大于给定圆弧所对的圆心角，证明给定圆弧过短，对于制定过渡半径无法完成圆弧过渡。
3. 对于内切过渡而言，若给定圆弧被给定线段所截形成的弓形高度小于2倍过渡圆半径，则没有足够的空间完成圆弧过渡。这里又分成两种情况。

## 圆弧-线段过渡

与线段圆弧类似，可以将加工顺序取反，然后计算过渡圆弧，然后将圆弧方向再取反。

## 圆弧-圆弧过渡

### 外接

判断条件：

1. 选定任意方向经过两端圆弧，则两段圆弧旋向相同。
2. 的旋向与的旋向相反。

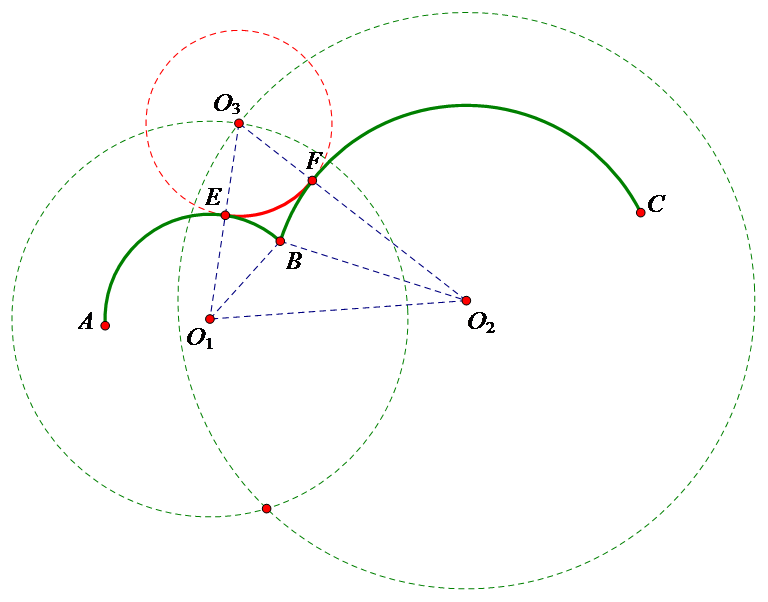
计算方程：

圆心距：





求出两组解，取与交点B距离较近的一组。



过渡圆与两段圆弧外接

### 内接

判断条件：

1. 选定任意方向经过两端圆弧，则两段圆弧旋向相同。
2. 的旋向与的旋向相同。

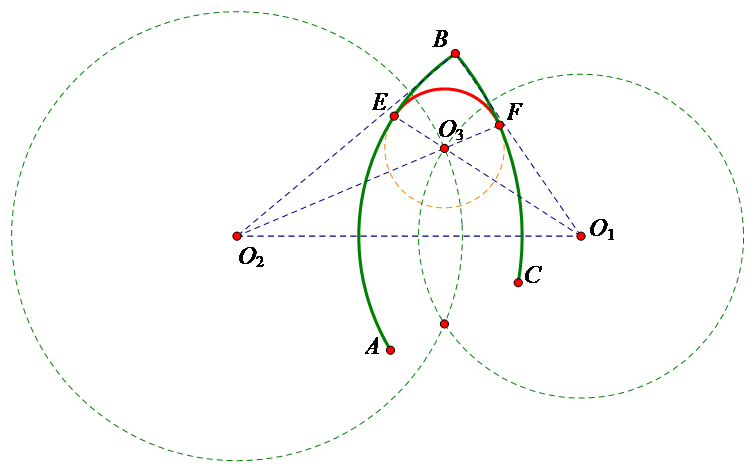
计算方程：

圆心距：





最多求出两组解，取与交点B距离较近的一组。



过渡圆与两段圆弧内切

### 混合（内外）连接

判断条件：

1. 选定任意方向经过两端圆弧，则两端圆弧旋向相反。

计算方程：这种情况需首先判断，顶圆弧与底圆弧。顶圆弧为与过渡圆内切的圆弧，底圆弧为与过渡圆弧外接的圆弧。这需要借助圆弧的凸向来判断。首先给出圆弧凸向的定义：

以为例，在其上除端点外任选一点M，连结BM，BC。若BM所在的位置可由BC顺时针旋转获得，则称对于点B为顺时针凸，或对于点B的凸向为顺时针。显然，在混合连接的情况下，两段圆弧对于交点B的凸向是相同的。

对于两段邻接的圆弧（圆心为O1，半径R1），（圆心为O2，半径R2），圆弧连结O1B, O2B,若的方向（旋转方向）与两段圆弧对于交点B的凸向一致，则前一段圆弧为过渡圆的底圆弧，后一段圆弧为顶圆弧，过渡圆与两端圆弧圆心距L13，L23和过渡圆半径R3，弧半径R1,R2关系如下：



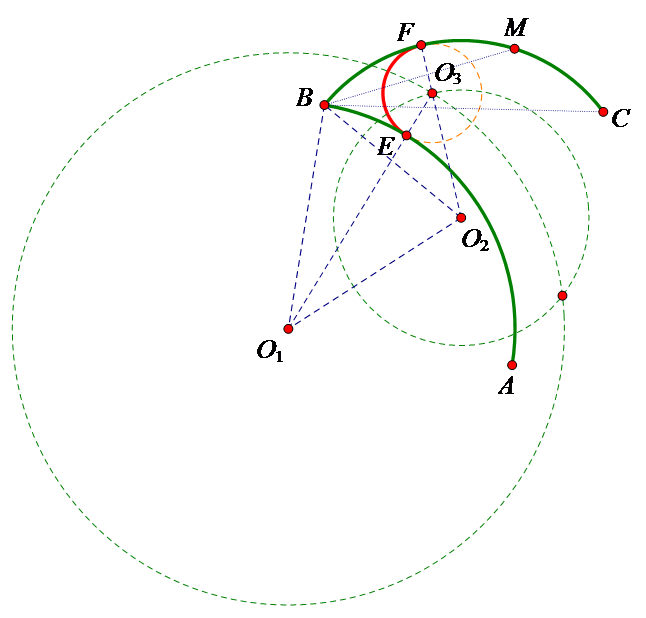


反之:





以上方法最多可求出两组解，取与交点B距离较近的一组。



过渡圆与两段圆弧混合连接

### 无解情况

1. 内接或者混合的情况中，如果过渡圆半径设置过大，会出现复数解，证明此时无解，需调整过渡圆弧半径。

# 前瞻

## 前瞻目的

实现速度的合理规划，合理地分配不同几何元素交点处的通过速度。得到各段起末点最优速度，在保证加工质量的前提下尽可能高的提高系统效率。

## 需要考虑的主要因素

1.加工效率，即满足其他条件的情况下，速度尽可能快；

2.系统最大加减速能力，以及系统最大加工速度；

3.相邻段夹角，影响该点最大通过速度；

4.段长因素；

5.圆弧弓高误差限制（针对圆弧），影响该段最大进给速度；

6.系统最大向心加速度，影响该段最大进给速度。

## 前瞻算法描述

前瞻处理主要包括两个阶段：预处理和速度校核。在预处理阶段，主要计算该段的段长，该段末点的最大预估速度；速度校核阶段，主要有两个过程，即前向校核和后向校核，其中前向校核是根据上一个已校核完成的速度和当前规划段的段长来对当前段末点的速度进行限制，后向校核则是根据前向校核的结果计算此速度下减速到0所需的距离，然后将包含在此距离内的后续段提取出来，依次对当前校验段到这些后续段进行速度校验，如果当前点速度大了，降速无法到达其中某点，则降低当前校验点的速度，使之对于提取的后续段均可达。进而完成校验。

参数定义：

第i段微线段起始速度为vi-1，终止速度为vi，加速度为ai，该微线段运行时间为Ti,微线段终点相对于起点的坐标为(∆xi， ∆yi ， ∆zi)，微线段长度为Li，与第i+1段微线段夹角为αi，系统允许的最大速度和最大加速度分别为vmax，amax。

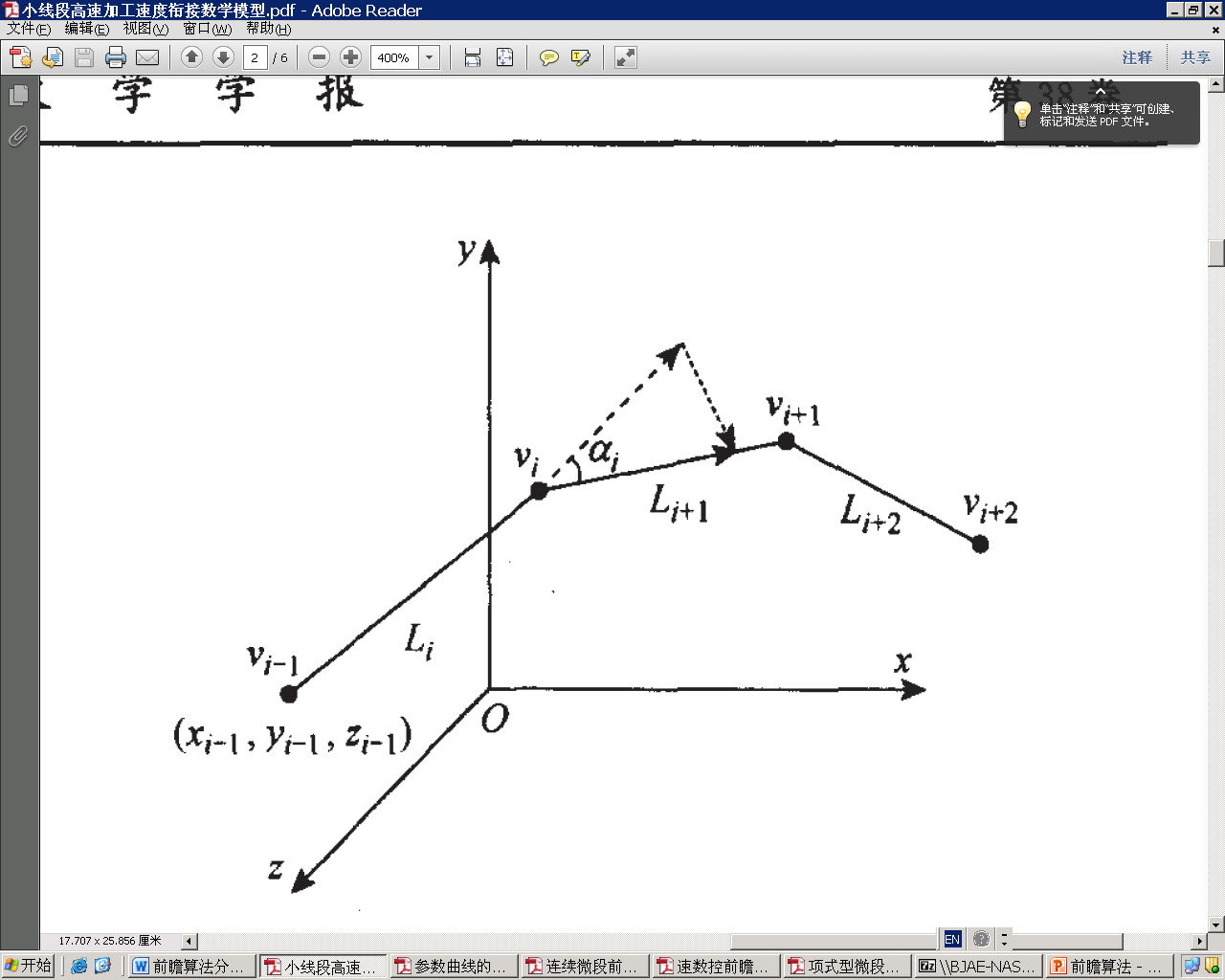


图 1

### 预处理

1.段长的计算也需要分别对待，主要有以下两种形式，线段类和圆弧类，其中把螺旋线、圆锥线也归为圆弧类，处理方法与普通圆弧稍有不同。

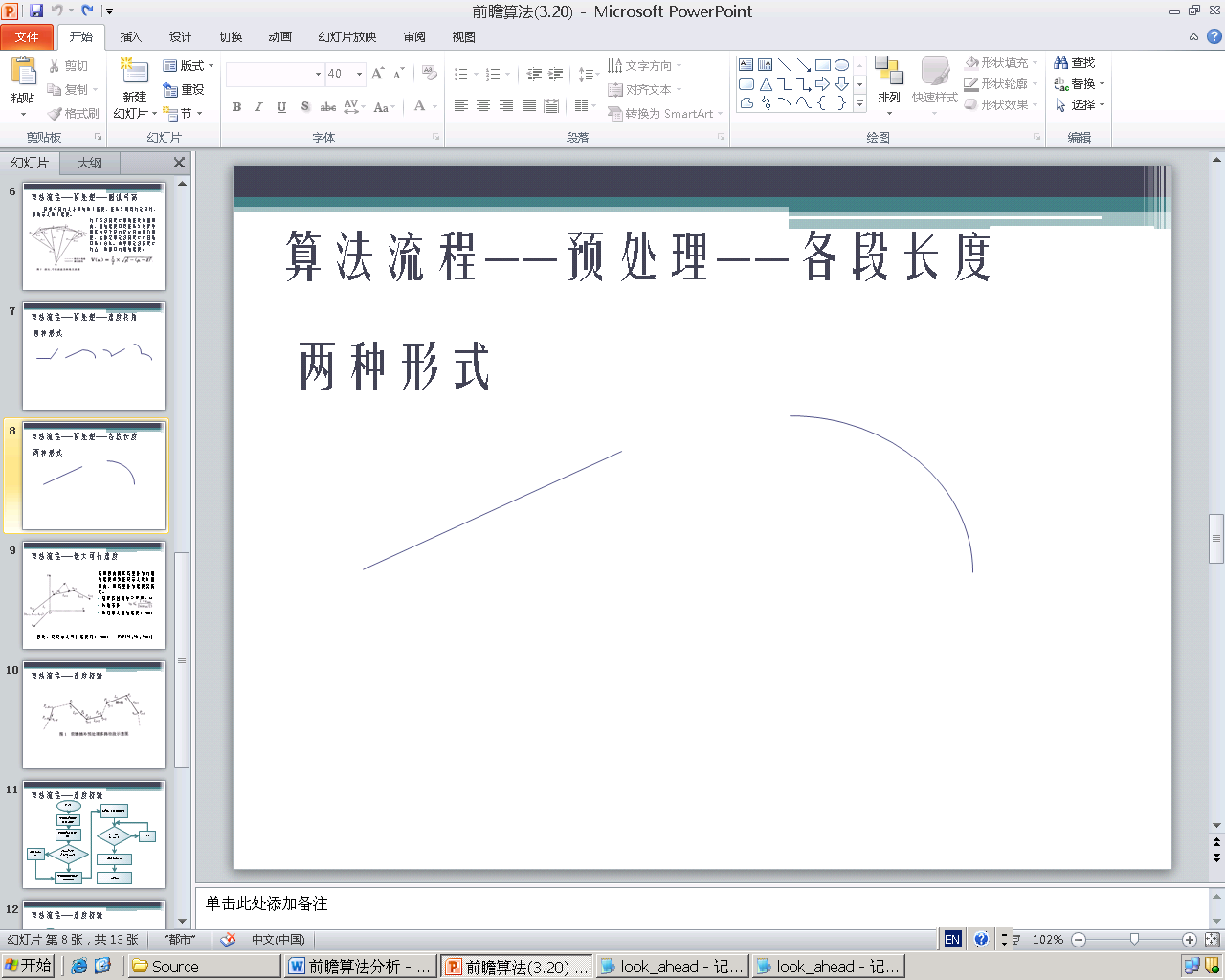


图2

2. 最大预估速度：

系统最大进给速度：vmax

相邻段转角条件限制：

相邻段之间的夹角形式有以下四种，分别为两段线段相邻、线段和圆弧相邻、圆弧和线段相邻、圆弧和圆弧相邻。

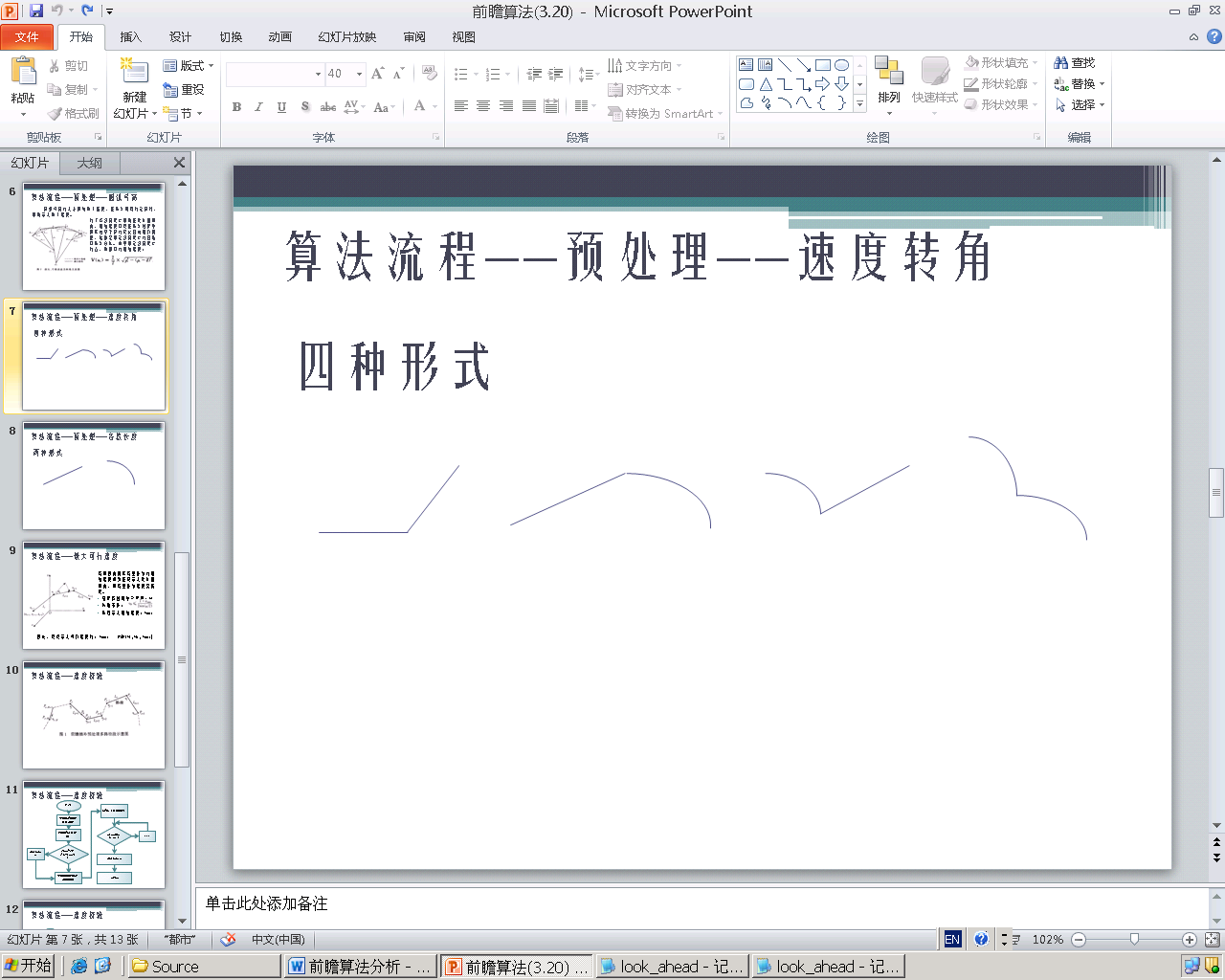


图3

为了将弦高误差限制在允许范围内，进给速度应该在插补过程中根据曲率半径的变化自动进行调整。如果限定弦高误差为占，则相应的进给速度：

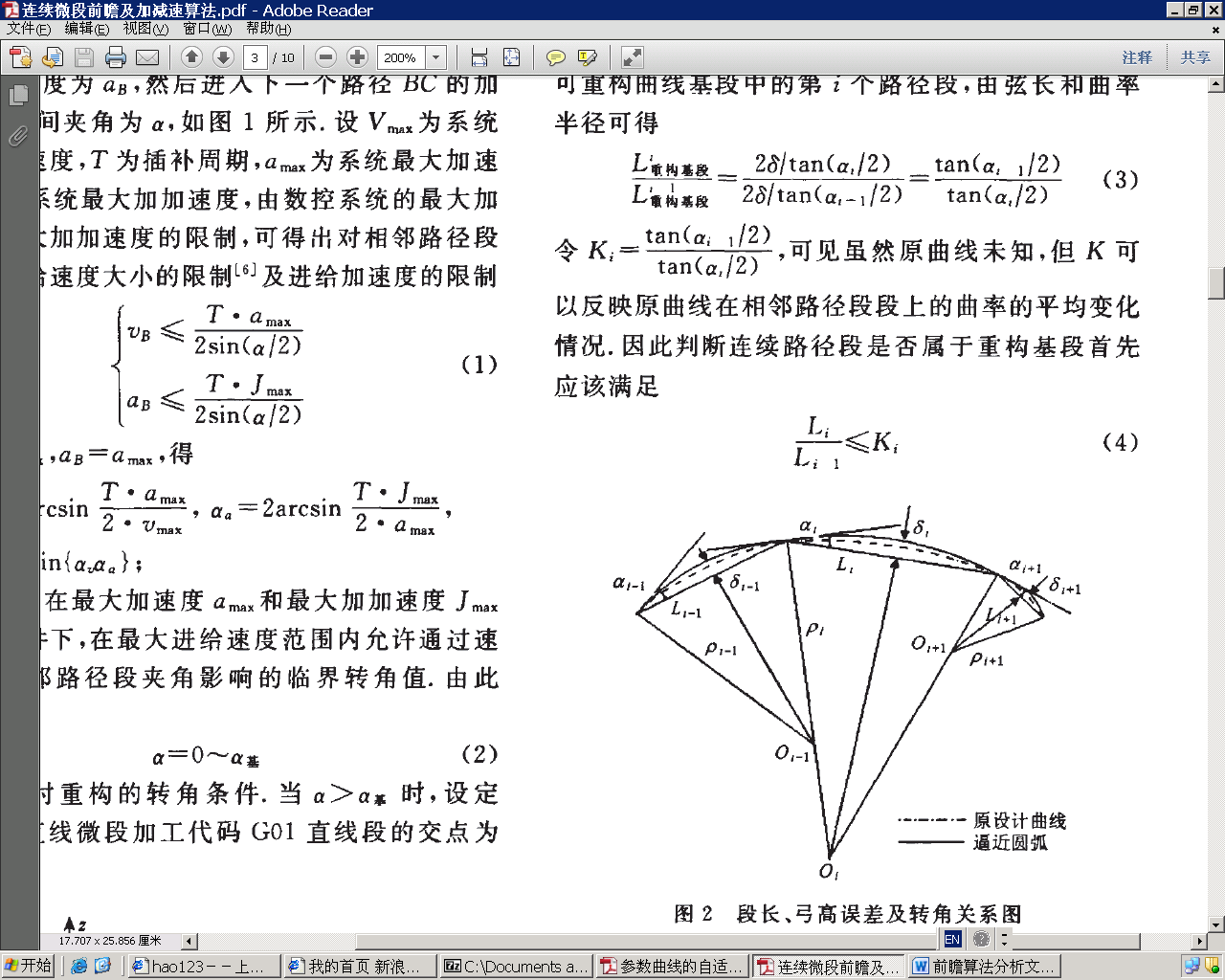


图4

最大向心加速度限制加工速度： 

编译器输出目标速度：

段末点最大预估速度：

圆弧段的目标速度为：

通过对以上数据的计算，完成最大预估速度计算和目标速度修正，然后将计算结果存入缓冲区。

### 速度校验

1. 第0点的速度为0；

2. 第i点前向速度校验：此时假定第i点速度就为最大预估速度vimax ，首先判断从第i-1点经过Li的距离可否速度变为vimax ；若能则第i点速度为vimax，否则降低到系统可承受的值。

3. 第i点后向速度校验：按照上一步得到第i步的速度vi，按照系统最大减速能力（应该设有一个安全系数）推算，减速到0最少所需的加工长度di。按照此加工长度推算，得到此长度内包含后向段数为n；从第i+1点开始，验算第i点可否在该段长度内速度由vi降为v（i+1）max（若v（i+1）max比vi大则不可虑可否加速到达，因为在校验i+1点时会进行前向速度校验，进行降速处理），若可以则继续验算i+2点，否则降低vi至可以达到降速要求的值，如此继续验证，直至n个点验证完毕（各点速度按照最大可行速度计算）。

4. 按上一步的方法继续验证i+1点速度，直至所有缓冲区数据处理完毕。

## 前瞻算法公式

### 最大加速度和停止加速度条件

加速到达应满足如下条件：

V1^2 <= V0^2+2\*a\*L

其中V1是末速度，V0是起始速度，a是系统最大加速度，L是线段长度。

减速停止应满足如下条件：

V1^2 >= V0^2-2\*a\*L

其中V1是末速度，V0是起始速度，a是系统最大停止加速度，L是线段长度。

### 相邻段夹角条件

相邻段速度应满足如下条件：

V <= T＊a／(2\*sin(Ѳ/2）)

其中V是交点速度，T是插补周期，a是系统最大加速度，Ѳ 是相邻段的夹角。

### 弓高误差条件

圆弧速度应满足如下弓高误差条件：

V <= 2＊sqrt(2\*r\*e-e^2)／T

其中V是圆弧速度，T是插补周期，r是圆弧半径，e是弓高误差。

### 向心加速度条件

圆弧速度应满足如下向心加速度条件：

V <= sqrt(r＊a)

其中V是圆弧速度，r是圆弧半径，a是向心加速度。

## 前瞻处理

### 前瞻处理分类

前瞻处理主要包括两个阶段：预处理和综合处理。

#### 预处理

预处理阶段对不需多段综合分析才能得出结果的数据进行计算，主要包括：线段长度，相邻段夹角的最大速度，弓高误差允许的最大速度，向心加速度允许的最大速度。

#### 综合处理

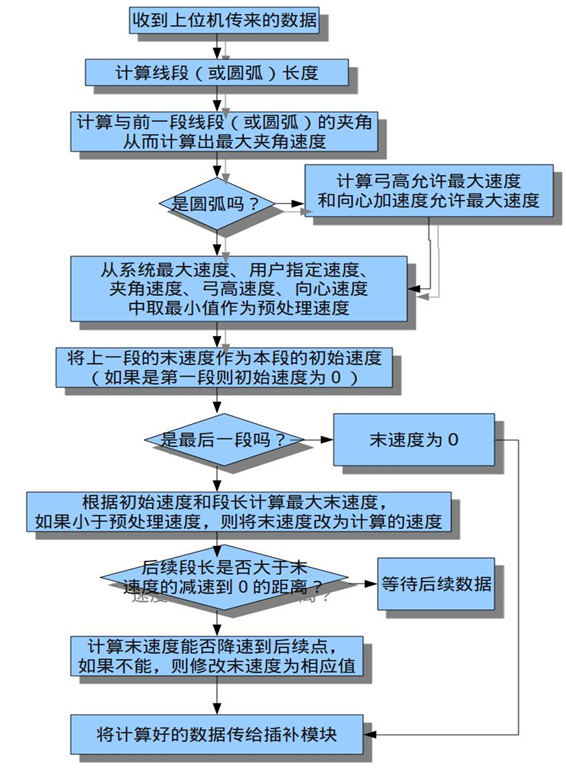
综合处理阶段对需多段综合分析才能得出结果的数据进行计算，对线段进行综合处理的前提条件是：总线段长度已经达到线段减速到0所需的距离，或前瞻缓冲区已满。

综合处理分为能否加速到达和能否减速停止两部分：

能否加速到达的判断是根据上一个点综合处理后的未速度、系统最大加速度、段长，计算出线段的最大未速度；

能否减速停止的判断是首先计算出的此速度下减速到0所需的距离，然后将包含在此距离内的后续段提取出来，依次对当前段到这些后续段进行速度校验，如果当前点速度大了，降速无法到达其中某点，则降低当前点的速度，使之对于后续段均可到达。

## 流程图



## 上下文接口

前瞻处理位于上位机处理之后，插补处理之前。

当上位机将G代码编译处理完成之后，将数据通过SysLink发送给下位机，下位机收到数据后调用前瞻模块的LookAhead::Process函数将数据传给前瞻模块处理，它的原型如下：

int16\_t LookAhead::Process(LineData\* line\_data)；

当前瞻模块处理完毕之后，通过 Interpolation::InsertInterpolationData 函数将数据传给插补模块进行处理，它的原型如下：

int16\_t Interpolation::InsertInterpolationData(LineData\* data)；

## 前瞻函数及其说明

### Process函数

函数的原型是：int16\_t Process(LineData\* line\_data)；

下位机收到数据后调用Process函数将数据传给前瞻模块，该函数收到数据后，如果该数据是插补数据或有程序结束标志，则放入前瞻缓冲区m\_data\_buffer中并调用LookAheadHandle函数进行前瞻处理，若是其它非插补数据则不放入前瞻缓冲区m\_data\_buffer中。

### LookAheadHandle函数

函数的原型是： void LookAheadHandle()；

该函数对数据进行前瞻处理，它首先查询m\_data\_buffer缓冲区，如果缓冲区中有新数据则对新数据进行如下处理：

(a)、若该数据是插补数据，则调用PrepareProcess函数进行前瞻预处理；

(b)、若该数据有程序结束标志，则调用EndProcess函数对缓冲区中的数据进行前瞻计算，完成前瞻；

(c)、若该数据的线段长度大于上一条线段末速度停下来为0的最小长度，则调用GetLookAheadData函数对该数据之前面的数据进行前瞻计算，完成该数据前面数据的前瞻；

(d)、若前瞻缓冲区m\_data\_buffer的数据已经达到系统前瞻条数m\_lookahead\_count，则不论是否前瞻条件满足，都调用ForceOneProcess函数强制进行前瞻计算；若缓冲区没有达到系统前瞻条数，则调用NormalProcess函数进行前瞻处理，该函数只有在前瞻条件满足时才进行前瞻计算。

### PrepareProcess函数

函数的原型是： void PrepareProcess(LineData\* pre\_data, LineData\* line\_data)；

该函数对数据进行前瞻预处理，它进行下面的操作：

(a)、若需要前瞻计算圆心，则调用FillCircleData函数计算圆心和圆弧角度；

(b)、若需要前瞻计算长度，则调用Distant计算线段和圆弧长度；

(c)、调用CalcSpeed函数计算向心加速度和弓高误差限制及用户指定速度和系统最大速度，选择其中的最小值作为当前速度；

(d)、调用AngleSpeed函数计算夹角速度，选择当前速度与夹角速度中的最小值作为末速度。

### EndProcess函数

函数的原型是： void EndProcess(int16\_t len)；

该函数对结束前瞻缓冲区m\_data\_buffer中前len个数据的前瞻处理，并将第len个数据的末速度设置为0。它首先将将第len个数据的末速度设置为0，然后调用GetLookAheadData函数对前len个数据进行前瞻计算。

### GetLookAheadData函数

函数的原型是： void GetLookAheadData(int16\_t len, bool end)；

该函数对前瞻缓冲区m\_data\_buffer中前len个数据进行前瞻处理，计算出每个数据的末速度，计算方法如下：

（a）、从上次前瞻的最后点的末速度开始，依次对缓冲区中前len个数据的末速度进行比较，查看是否能加速到达该点，如果不能到达，则修改该点的末速度为能到达的最大速度；

（b）、从第len个数据的末速度开始，从后向前依次对len个数据的末速度进行比较，查看是否能减速停到该点，如果不能停下来，则修改该点的末速度为能停下来的最大速度；

前瞻计算完成后，调用InsertDataToInterpolation函数将前len个数据发送给插补模块。

### ForceOneProcess函数

函数的原型是： void ForceOneProcess()；

该函数对第一条线段进行前瞻处理，而无论该线段的前瞻条件是否满足。当前瞻缓冲区满时或前瞻条件满足时，都会调用该函数进行前瞻处理，该函数的前瞻计算方法如下：

（a）、从上次前瞻的最后点的末速度开始，计算能否加速到达第一条线段的末速度，如果不能到达，则修改第一条线段的末速度为能到达的最大速度；

（b）、用第一条线段的末速度，依次对后续线段进行比较，查看是否能减速停到该点，如果不能停下来，则修改第一条线段的末速度为能停下来的最大速度；

前瞻计算完成后，调用InsertDataToInterpolation函数将第一条线段发送给插补模块。

### NormalProcess函数

函数的原型是： int16\_t NormalProcess()；

该函数判断第一条线段的前瞻条件是否满足，如果不满足则不进行任何处理，如果满足则调用**ForceOneProcess** 函数对第一条线段进行前瞻处理。前瞻条件是否满足只需判断后续线段的长度是否大于第一条线段减速停止到0的最小长度，若大于则能对第一条线段进行前瞻，否则不进行前瞻。

### CalcSpeed函数

函数的原型是： double CalcSpeed(LineData\* data)；

该函数首先判断数据是否是圆弧，如果是圆弧则先计算向心加速度限制下的最大速度，再计算弓高误差限制下的最大速度，同时，不论数据是否圆弧都计算用户指定最大速度和系统最大速度，然后取这四个速度的最小值做为函数返回值。

### AngleSpeed函数

函数的原型是： double AngleSpeed(const LineData\* point1, LineData\* point2)；

该函数计算线段的方向向量，和在方向向量限制下的各个轴的速度，同时计算上一条线段与本线段的夹角限制速度，该函数的计算方法如下：

（a）、如果数据是直线，则当前点坐标减去上一点坐标就是该直线的方向向量；

（b）、如果数据是二维圆弧，则需要计算起点和终点的切向量，计算方法是先计算圆心到起点（或终点）的方向向量，再计算与该向量垂直的向量，垂直向量就是该点的切向量；

（c）、如果数据是三维圆弧，则调用Get3DArcVector函数计算起点与终点的切向量；

（d）、获得方向向量后，用速度乘以每个轴的分量，就是该轴的速度，如果速度大于轴最大速度，则减少总速度，使各轴速度不大于各轴的最大速度；

（e）、计算上一线段的方向向量与本线段的夹角，根据夹角计算出夹角最大速度，若该速度比上一条线段的末速度小，则修改上一条线段的末速度。

### Get3DArcVector函数

函数的原型是： void Get3DArcVector(const double\* point1, const double\* point2, const double\* center, double\* start\_v, double\* end\_v)；

该函数根据起点、终点和圆心计算三维圆弧的起点切向量和终点切向量，它首先根据三点求出平面的法向量，然后根据法向量和圆心到起点（或终点）的方向向量，计算出切向量。

### Distant函数

函数的原型是：double Distant(const LineData\* point1,const LineData\* point2)；

该函数计算数据的长度，该函数的计算方法如下：

（a）、如果数据是直线，则计算起点到终点的距离；

（b）、如果数据是圆弧，则调用DistantCircle函数计算圆弧长度。

### DistantCircle函数

函数的原型是：double DistantCircle(const LineData\* point1,const LineData\* point2)；

该函数计算圆弧数据的长度，该函数的计算方法如下：

（a）、如果数据是平面或三维圆弧，则半径\*夹角就是圆弧的长度；

（b）、如果数据是螺旋线，则sqrt((半径\*夹角)^2 + 高度^2)就是螺旋线的长度。

# 插补

## 插补模块结构和流程

### 插补模块结构图

插补的主要结构如图1，详细参考文档插补结构图.vsd

其中Interpolation类中的Interpolate接口由前瞻调用，主要作用是将前瞻处理的数据写入插补缓冲区类。在InterpolationThread中，Start()函数启动插补线程，在每个插补周期调用ManageInterpolation方法，对插补指令进行相应的处理。

在ManageInterpolation方法中，如果在正常插补的情况下，调用DoInterpolation方法，此方法中首先对插补线段进行速度规划（在此有两种速度规划方式，S型和T型），然后根据速度规划计算下一个插补周期需要移动的距离，根据插补类型以及几何形状计算每个轴上应该到达的位置，并传给下一个模块。

插补类型目前实现的一共有七种：快速移动，直线插补，平面圆弧插补，三维圆弧插补，螺旋插补（支持两个直线轴），螺旋插补B，螺线插补和圆锥插补。其中快速移动（G00）由类QuickMoveInterpolation实现；直线插补（G01）由LineInterpolation实现；平面圆弧插补，三维圆弧插补，螺旋插补（支持两个直线轴），螺旋插补B，螺线插补和圆锥插补（G代码都是G02，G03或者G02.4，G03.4）都由CircleInterpolation实现。

从插补缓冲区读取数据后，根据interpolation\_type判断运动指令中的插补类型，根据插补类型配置相应的插补算法，以及对数据做相应的预处理。

### 插补模块流程图

插补模块的主要流程如图2所示，参考插补模块流程图.vsd文档。

图2所示插补流程是在插补的一个线程（InterpolationThread）中运行的，当线程启动时，插补模块在每一个插补周期开始会检查插补的缓冲区内是否有数据剩余，如果有数据，则取出数据，进行分析，做出相应的插补动作；如果没有数据，则等待下一个插补周期到来。

插补缓冲区（m\_interpolation\_buffer）是专为插补类建立的一个循环缓冲区，因为一般情况下，由于插补的功能是将一段曲线进行密化，并实时计算插补数据，一段程序中插补所用的时间远远比其他模块消耗的时间大得多，所以需要一段专门的缓冲区来存储插补数据。它的写入是由插补的上层模块——前瞻来完成的。前瞻模块以及其他的编译，刀补，坐标转换等模块都是在另一个线程中由ChannelControl模块来统一调度的。前瞻模块把当前处理完的数据写入插补的缓冲区内，插补模块从缓冲区取出数据进行处理，如图3所示。

**图表 1 插补结构图**



图表 2 插补流程图

**ChannelControl**

**编译**

**参数归一化**

**刀补**

**前瞻**

**写入插补缓冲区**

**用户数据输入**

**取出数据**

**插补**

**多轴处理**

**坐标转换**

**向电机输出数据**

**. . .**

图表 3 插补缓冲区

## 速度规划

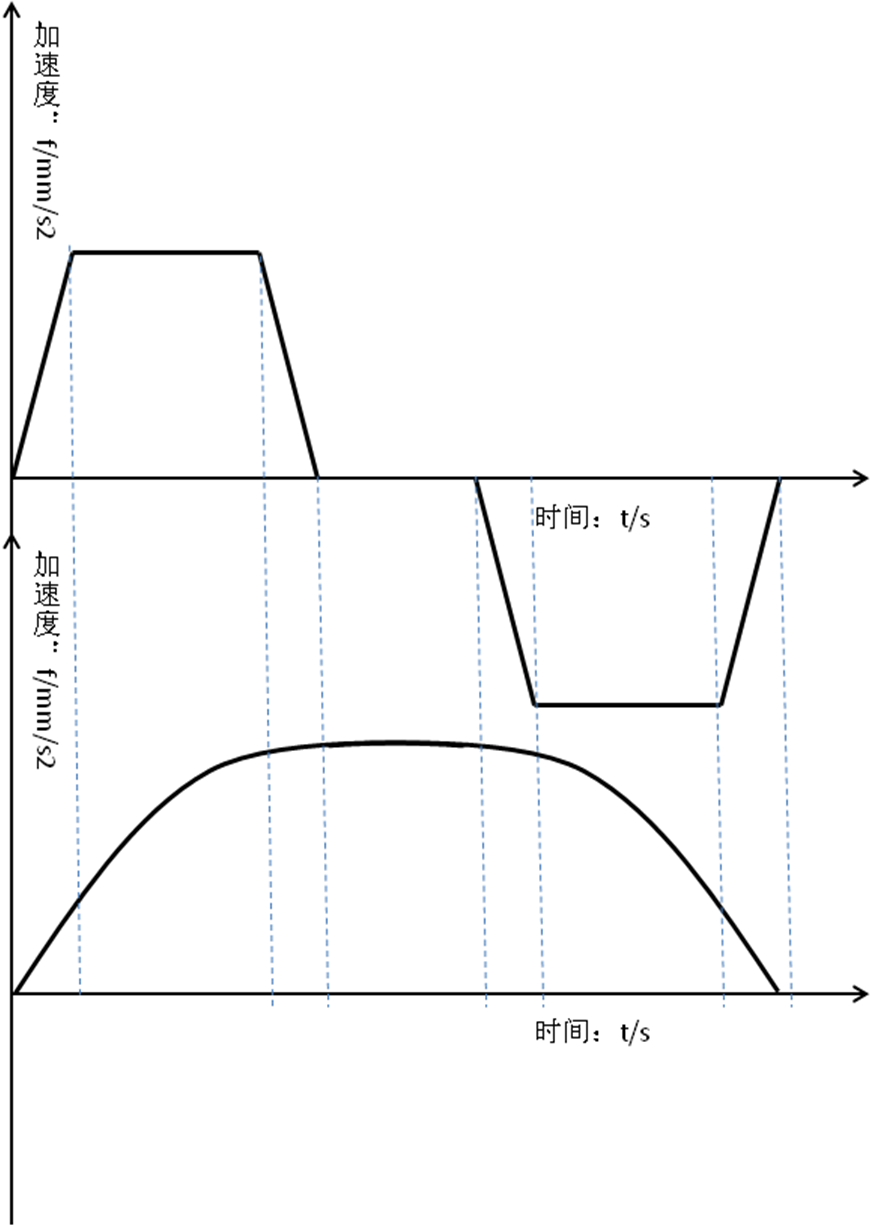
AE8000中速度规划有两种模式，S型曲线规划和T型曲线规划。如图4所示

时间：t/s

速度：f/mm/s

时间：t/s

加速度：f/mm/s2



图表 4 速度规划，左：T型速度规划；右：S型速度规划

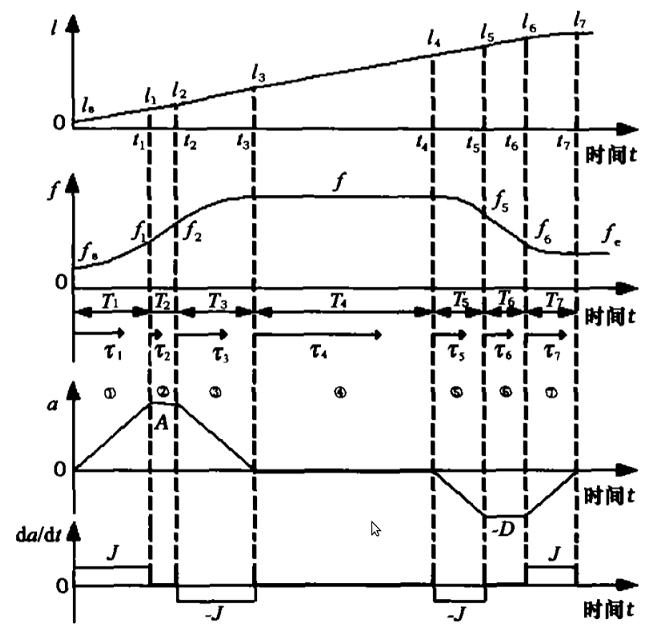
由图可以看出,一方面，S型曲线加速度规划比T型曲线速度规划更平滑，它的加速度也呈线性变化，不像T型曲线规划的加速度会有较大的突变。加速度的突变在一定情况下会产生震荡，对机床造成较大的磨损，加工的平滑度也会降低。另一方面，T型曲线规划比S型曲线规划计算简单，因此更加节省计算时间。在选择加速度规划时同时考虑以上两方面的原因。

### S型曲线速度规划

#### S型曲线参数

S型曲线速度规划分为7个阶段，如图5。t 为时间坐标；t1~t7为每个阶段终点的时间坐标；T1~T4为每个阶段时间长度；τ1~τ7为局部时间坐标，即以每一个阶段的时间起点为零点，τi = t - ti-1。

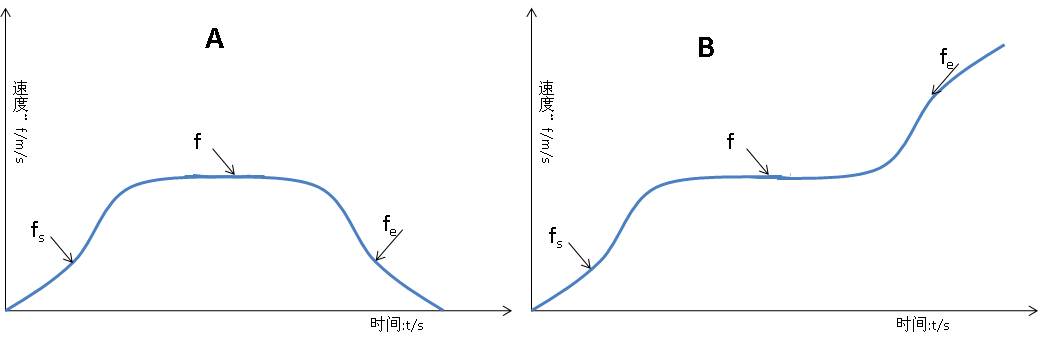
S型曲线规划需要用到的系统参数：A­——系统最大加速度，D——系统最大减速度，J——系统加加速度。另外，fs是曲线初始进给速度，fe为终点速度，f是目标速度，表示曲线速度稳定后（匀速阶段）达到的速度。ls是曲线初始位置。

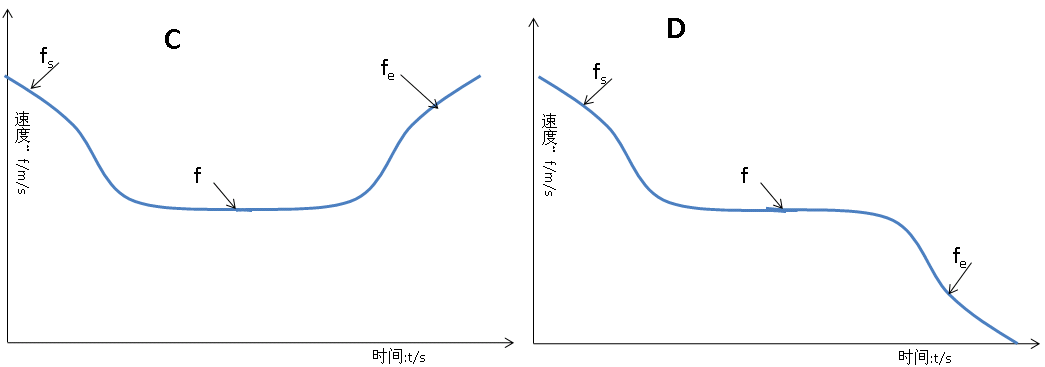


图表 5 S型曲线速度规划

#### 速度规划类型

根据初始速度fs,终点速度fe，以及目标速度f之间的关系，S型曲线速度规划有以下4种类型，如图6。其中A图表示fs<f, fe<f的情况；B图表示fs<f<fe情况；C图表示fs>f, fe>f的情况；D图表示fs>f>fe的情况。



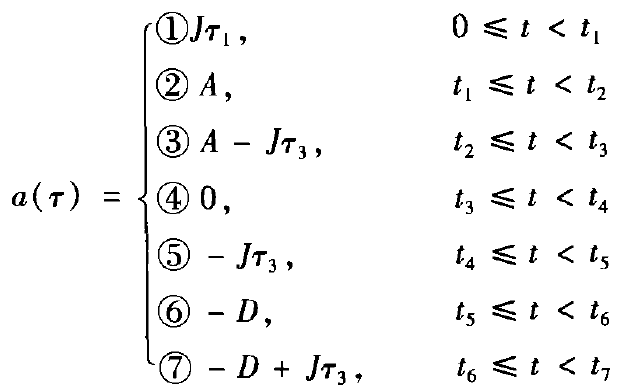


图表 6 4种S型曲线规划

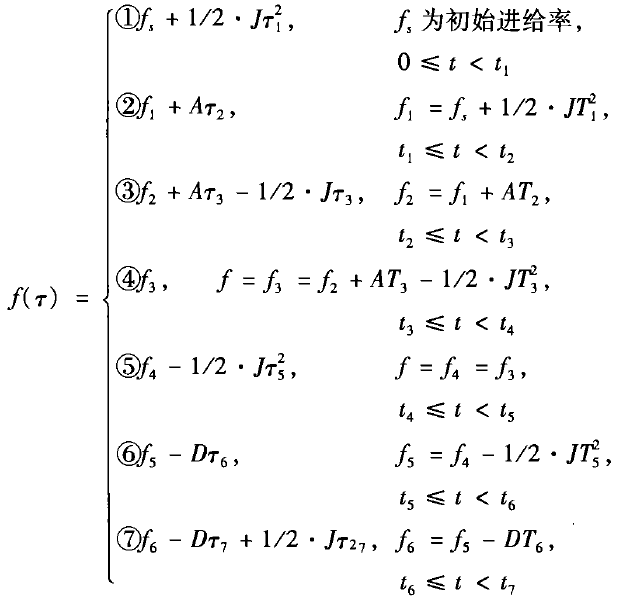
在处理这四种情况时，后三种情况都视作第一种情况来处理，即所有运动都看做先加速，后匀速，最后减速的过程。例如B图的情况，可以看做刚开始进行一段加速度为正的加速运动，后匀速，然后再以一个负的减速度进行减速。这样在进行速度规划时只需先判断fs，f和fe的关系，确定A和D的正负，就可以把所有运动情况当成第一种来处理。

#### S型曲线规划公式表达

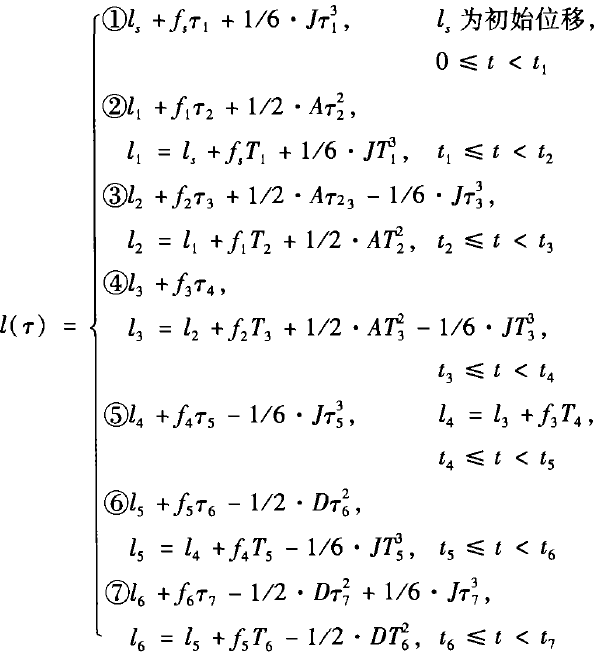
在图5中，da/dt在每个阶段为一常量，J或者0，积分可以得到加速度的表达式。如下为加速度，速度，以及长度的表达式。

 公式（1）

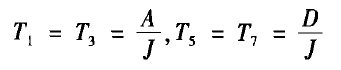
设fs为初始速度，将加速度a进行积分得到速度关于时间的公式：

 公式（2）

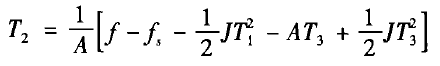
设ls为初始距离，将速度进行积分得到距离关于时间的公式：

 公式（3）

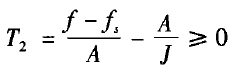
电机速度从零加速到最大值和从最大值减速到零的时间相同，则有：

 公式（4）

由于在第三阶段f3 = f，则

 公式（5）

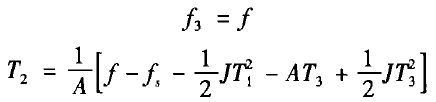
如果存在匀速阶段，意味着T2 不小于零，公式（5）可以简化为：

 公式（6）

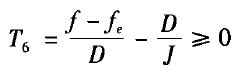
如果不存在匀速阶段，则将T2设置为0，而加速度必须减少到它的最大可能值：

 公式（7）

同时考虑到在第七阶段结束时达到最终进给速率fe，根据公式（2）得到：

 公式（8）

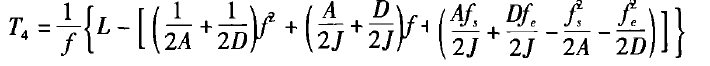
同理，如果匀减速阶段存在，则：

 公式（9）

如果匀减速阶段不存在，将T6的值设为0，减速度极限减少为：

 公式（10）

根据上述公式（4），（6），（9）求得T1，T2，T3，T5，T6代入公式（3）可以得到总的移动长度，即L = le – ls = 。L为已知总长度，从而得到T4：

公式（11）

如果位移长度足够大，包含恒进给阶段，那么T4大于等于0。否则，如果刀具路径太短不能达到目标进给速率f，那么T4 = 0。进给速率减少到由公式（11）解出来的最大可能值。对于新的目标进给速率，还需要通过公式（6）和（9）重新计算T2­­和T6，以及对参数A和D进行重新调整。

### T型曲线速度规划

当S型曲线规划J为无穷大的时候，便退化为T型曲线规划。T型曲线规划的计算可以看做S型曲线规划的一种，在此不再赘述。

## 核心算法说明

在进行曲线的插补之前，先对曲线的长度进行速度规划，保证合成进给速率平滑的过渡，然后根据速度规划值计算出下一个周期合成速度的步长，最后根据合成速度的步长，计算分配到每个轴上的步长。不同的插补类型，补偿与每个轴上移动的位置之间的几何关系不同，下面主要介绍不同的插补类型，在速度规划之后，对于每个轴上位置的计算。

### 快速移动

快速移动是指机床的每个轴用系统最大的速度移动到指定的位置。快速移动的目标速度是由系统参数max\_velocity决定的，其终点速度为0，起点速度fs由前一段曲线的终点速度决定。

快速移动分为两类：直线型快速移动和非直线型快速移动。直线型快速移动是指轴之间合成运动为直线；非直线型快速定位是指每个轴独立运动，直到各自运动到指定的位置。

非直线插补

非直线插补

图表 7 快速移动的两种类型

非直线型快速移动由合成步长计算每个轴上的位置比较简单，因为快速移动每个轴的运动都是独立的，首先先对每个轴做速度规划，然后计算每个轴上的步长即可。

直线型快速移动参考直线插补的算法，其和直线插补的位移区别就是目标速度由系统参数事先设定，终点速度为0。

### 直线插补

∆l

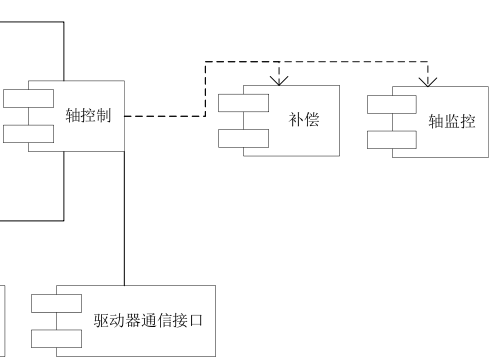
∆x

∆y

k

# 补偿

补偿功能由轴控制模块调用，处在插补之后，私服输出之前。

补偿代码采用装饰模式，所有补偿算法（反向间隙、螺距误差补偿）都是单独一个装饰类挂在装饰链中，所有补偿方法可以实现叠加。当需要增加或修改补偿时，只需在装饰链上增加或修改装饰类。以下，建立Compensation为实体类，但不实现补偿算法；然后建立装饰类基类CompDecorator 继承Compensation，用户装饰链上装饰类的叠加；最后继承CompDecorator创建各个补偿装饰类，实现各个补偿算法。

## 补偿实体类

* class Compensation

补偿实体类存储了轴号、轴类型、位置信息和参考状态，以及定义补偿功能的各个函数，但没有实现的补偿方法，而是由挂在实体类之上的各个装饰类实现。

成员变量表

|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| m\_axis\_id | int16\_t | 轴号 |
| m\_current\_pos | double | 当前轴位置 |
| m\_current\_mot\_pos | double | 当前电机位置 |
| m\_is\_axis\_run\_home | char | 轴是否已回参考点，0为否，1为是 |
| m\_axis\_type | uint8\_t | 轴类型 |

成员函数表

|  |  |  |  |
| --- | --- | --- | --- |
| 成员函数 | 输入 | 输出 | 描述 |
| SetAxisIndex | 轴号 | 处理结果 | 设置轴号 |
| SetAxisHomeStatus | 轴是否已回参考点，0为否，1为是 | 无 | 设置轴是否已回参考点 |
| Init（对外函数接口） | 无 | 处理结果 | 初始化,在参数配置和通道控制实例化和初始化后执行 |
| Reset（对外函数接口） | 无 | 无 | 复位 |
| Compensate | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 补偿方法；根据输入的轴目标位置，输出电机目标位置 |
| CancelCompensation | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 取消补偿方法，回参考点前，开机或复位时调用；根据存储的当前电机位置位置，计算并输出轴目标位置 |
| BuildCompensation | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 建立补偿方法，回参考点后调用；根据存储的当前电机位置位置，计算轴目标位置 |
| InputCmd | NC命令 | 处理结果，和NC命令（同输入） | 为补偿模块上下层模块提供命令输入接口，并对各种命令进行处理，处理下层模块发送过来的命令 |

Nc命令

|  |  |  |  |
| --- | --- | --- | --- |
| 命令 | 输入 | 输出 | 描述 |
| SET\_COMP\_AXIS\_INDEX | 轴号 | 无 | 设置补偿器中的轴号 |
| SET\_COMP\_HOME\_STATUS | 轴是否已回参考点，0为否，1为是 | 无 | 设置补偿器中是否回参考点标志 |
| INIT\_COMPENSATION | 无 | 处理结果 | 初始化补偿器 |
| RESET\_COMPENSATION | 无 | 无 | 复位补偿器 |
| BUILD\_COMPENSATION | 无 | 处理结果，轴目标位置 | 建立补偿 |
| CANCEL\_COMPENSATION | 无 | 处理结果，轴目标位置 | 取消补偿 |
| COMPENSATE | 轴目标位置 | 处理结果，电机目标位置 | 补偿 |

## 补偿装饰类基类

* class CompDecorator:public Compensation

补偿装饰类基类存储了补偿装饰链上前一元素的指针，以及实现对装饰链上前一元素各个函数的调用，但没有实现的补偿方法，而是由继承装饰类基类的各个装饰类实现。

成员变量表

|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| m\_pComponent | Compensation \* | 装饰链的上一元素 |

成员函数表

|  |  |  |  |
| --- | --- | --- | --- |
| 成员函数 | 输入 | 输出 | 描述 |
| SetAxisIndex | 轴号 | 处理结果 | 设置轴号，并调用装饰链上一元素的同一函数 |
| SetAxisHomeStatus | 轴是否已回参考点，0为否，1为是 | 无 | 设置轴是否已回参考点，并调用装饰链上一元素的同一函数 |
| Init | 无 | 处理结果 | 初始化，并调用装饰链上一元素的同一函数 |
| Reset | 无 | 无 | 复位，并调用装饰链上一元素的同一函数 |
| Compensate | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 调用装饰链上一元素的同一函数 |
| CancelCompensation | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 调用装饰链上一元素的同一函数 |
| BuildCompensation | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 调用装饰链上一元素的同一函数 |
| Decorate | 装饰链中上一元素的指针Compensation \* | 当前元素的指针Compensation \* | 将当前元素接到装饰链中， |

## 反向间隙补偿

### 反向间隙参数

反向间隙DSP全局变量表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变量 | 类型 | 范围 | 单位 | 描述 |
| backlash | Double | [0,MAX\_BACKLASH\_VALUE] | m | 各轴反向间隙补偿量；修改后，回参考点后生效；为0时，不补偿 |
| backlash\_comp\_step | Double | [0,backlash] | m | 各轴反向间隙补偿步长；修改后，回参考点后生效；为0时，不补偿 |
| MAX\_BACKLASH\_VALUE | 宏定义 | 0.1 | m | 最大反向间隙补偿量为0.1m |

### 反向间隙补偿装饰类

* class BacklashCompensation:public CompDecorator

反向间隙补偿装饰类存储了轴补偿参数、已补偿量和补偿使能变量。并实现了补偿初始化、复位、建立、取消和补偿方法。反向间隙补偿间隙在装饰链中处于螺距误差补偿之后。

成员变量表

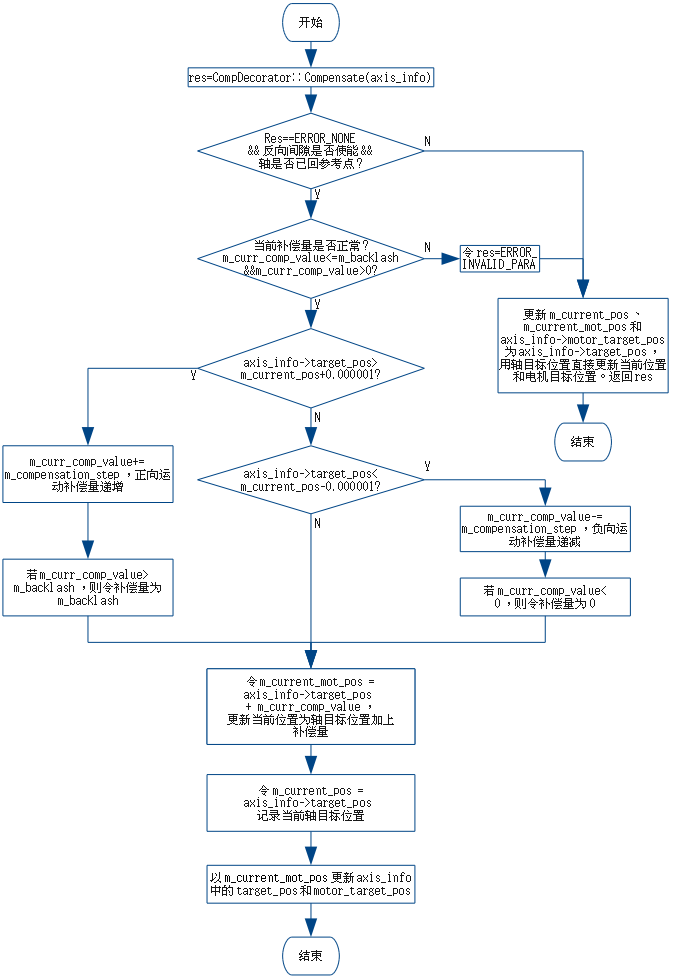
|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| m\_is\_backlash\_comp\_enabled | bool | 反向间隙补偿是否使能，TRUE为是，FALSE为否，补偿初始化、复位、建立、取消时更新 |
| m\_curr\_comp\_value | double | 已经完成的补偿量，初始化和复位时置零 |
| m\_backlash | double | 反向间隙值，初始化和复位时根据轴参数更新 |
| m\_compensation\_step | double | 补偿步长，初始化和复位时根据轴参数更新 |

成员函数表

|  |  |  |  |
| --- | --- | --- | --- |
| 成员函数 | 输入 | 输出 | 描述 |
| ReadParam | 无 | 处理结果 | 从g\_config中读取反向间隙补偿的相关参数，并判断数值是否有误，由Init()和Reset()调用 |
| Init | 无 | 处理结果 | 初始化反向间隙补偿；并调用装饰链上一元素的同一函数 |
| Reset | 无 | 无 | 复位反向间隙补偿；并调用装饰链上一元素的同一函数 |
| Compensate | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 进行反向间隙补偿；并调用装饰链上一元素的同一函数 |
| CancelCompensation | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 取消反向间隙补偿；并调用装饰链上一元素的同一函数 |
| BuildCompensation | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 建立反向间隙补偿；并调用装饰链上一元素的同一函数 |

### 反向间隙补偿流程

* Compensate函数，根据axis\_info的轴目标位置target\_pos，更新axis\_info的轴目标位置target\_pos（给下一补偿使用）和电机目标位置motor\_target\_pos。



反向间隙补偿函数流程图

1. 调用基类补偿函数，先进行补偿链中上一级的补偿，并令res = CompDecorator::Compensate(axis\_info)。
2. 如果返回res为ERROR\_NONE，且反向间隙补偿已使能且轴已返回参考点，则转到e；否则转到d。
3. 检测当前补偿量是否正常，即检测m\_curr\_comp\_value是否处于[0, m\_backlash]范围中，若是转到e；否则令res=ERROR\_INVALID\_PARA，并转到d。
4. 更新m\_current\_pos = m\_current\_mot\_pos = axis\_info->motor\_target\_pos = axis\_info->target\_pos，并return res。
5. 比较 axis\_info->target\_pos与 m\_current\_pos，若axis\_info->target\_pos比 m\_current\_pos大0.000001（1um）则转到f；若axis\_info->target\_pos比 m\_current\_pos小0.000001（1um）则转到g；否则转到h。
6. 轴向正侧运动，令m\_curr\_comp\_value = m\_curr\_comp\_value + m\_compensation\_step；若m\_curr\_comp\_value > m\_backlash，令m\_curr\_comp\_value = m\_backlash，转到i。
7. 轴向负侧运动，令m\_curr\_comp\_value = m\_curr\_comp\_value – m\_compensation\_step；若m\_curr\_comp\_value < 0，令m\_curr\_comp\_value = 0，转到i。
8. 轴停止或刚起步，此时也不让电机动，则不更新m\_curr\_comp\_value。
9. 更新 m\_current\_mot\_pos 为axis\_info->target\_pos + m\_curr\_comp\_value，加入补偿量。
10. 保存当前轴位置，令m\_current\_pos = axis\_info->target\_pos；然后以m\_current\_mot\_pos 更新axis\_info中的target\_pos和motor\_target\_pos，结束。

## 螺距误差补偿

### 螺距误差补偿参数

螺距误差DSP全局变量表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变量 | 类型 | 范围 | 单位 | 描述 |
| axis\_home\_pos | Double |  | m | 各轴参考点偏移位置，回零时，用户先要显示的坐标位置，即令坐标偏移 |
| screw\_comp\_home\_no | int16\_t | [-500,1500] |  | 在各轴螺补表中，参考点对应的补偿编号，从零开始计数，可不在补偿表范围内；回参考点并Reset()或重启回参考点后生效 |
| screw\_comp\_num | int16\_t | [0,SCREW\_COMP\_TABLE\_MAX\_LEN] |  | 各轴螺距误差补偿点数；回参考点并Reset()或重启回参考点后生效 |
| screw\_comp\_rate | Double | [0,MAX\_SCREW\_COMP\_RATE] |  | 各轴螺距误差补偿倍率，0相当于1；回参考点并Reset()或重启回参考点后生效 |
| screw\_comp\_interval | Double | [0,0.1] | m | 各轴螺距误差补偿间距；回参考点并Reset()或重启回参考点后生效 |
| screw\_comp\_table | int16\_t | [-MAX\_SCREW\_COMP\_VALUE  ,MAX\_SCREW\_COMP\_VALUE] | um | 各轴螺距误差补偿表；回参考点并Reset()或重启回参考点后生效 |
| SCREW\_COMP\_TABLE\_MAX\_LEN | 宏定义 | 1000 |  | 螺距误差表最大长度 |
| SCREW\_COMP\_TABLE\_LEN | 宏定义 | SCREW\_COMP\_TABLE\_MAX\_LEN+2 |  | 螺补装饰类的补偿表长度，相对于g\_config中的补偿表加入了头尾两个补偿值为0的点，所以补偿表长度加2 |
| MAX\_SCREW\_COMP\_RATE | 宏定义 | 10 |  | 最大螺距误差补偿倍率 |
| MAX\_SCREW\_COMP\_VALUE | 宏定义 | 100 | um | 设置最大螺距误差补偿量为100um，乘倍率后为1mm |
| DIV\_SCREW\_COMP\_UNIT\_TRANS | 宏定义 | 1000000 | um/m | 螺补数据的单位转换，由存储单位um转换为DSP的统一长度单位m |

### 螺距误差补偿装饰类

* class ScrewCompensation:public CompDecorator

螺距误差补偿装饰类存储了轴补偿参数和补偿使能变量。并实现了补偿初始化、复位、建立、取消和补偿方法。

成员变量表

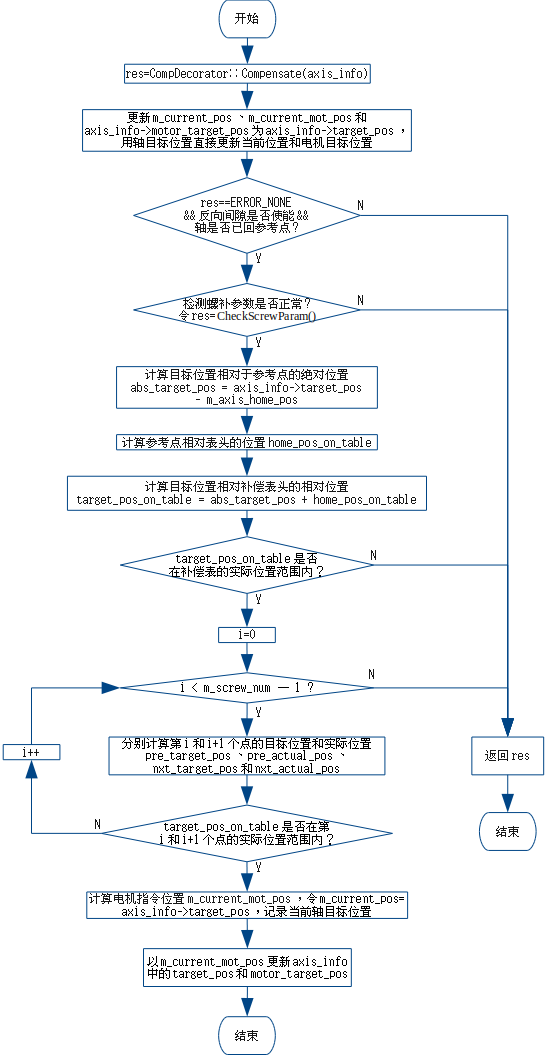
|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| m\_is\_screw\_comp\_enabled | bool | 螺距误差补偿是否使能，TRUE为是，FALSE为否，补偿初始化、复位、建立、取消时更新 |
| m\_screw\_home\_no | int16\_t | 参考点对应补偿表中编号，从０开始计数 |
| m\_screw\_num | int16\_t | 补偿点数 |
| m\_compensation\_rate | double | 补偿倍率 |
| m\_screw\_interval | double | 补偿间距 |
| m\_axis\_home\_pos | double | 参考点偏移位置 |
| m\_screw\_comp\_table | int16\_t | 螺距补偿表 |

成员函数表

|  |  |  |  |
| --- | --- | --- | --- |
| 成员函数 | 输入 | 输出 | 描述 |
| ReadParam | 无 | 处理结果 | 从g\_config中读取反向间隙补偿的相关参数，并判断数值是否有误，由Init()和Reset()调用 |
| Init | 无 | 处理结果 | 初始化螺距误差补偿；并调用装饰链上一元素的同一函数 |
| Reset | 无 | 无 | 复位螺距误差补偿；并调用装饰链上一元素的同一函数 |
| Compensate | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 进行螺距误差补偿；并调用装饰链上一元素的同一函数 |
| CancelCompensation | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 取消螺距误差补偿；并调用装饰链上一元素的同一函数 |
| BuildCompensation | 轴位置信息 | 处理结果，和轴位置信息（同输入） | 建立螺距误差补偿；并调用装饰链上一元素的同一函数 |
| CheckScrewParam | 无 | 处理结果 | 检查螺补参数是否正常 |

### 螺距误差补偿流程

* Compensate函数，根据axis\_info的轴目标位置target\_pos，更新axis\_info的轴目标位置target\_pos（给下一补偿使用）和电机目标位置motor\_target\_pos。



螺距误差补偿函数流程

1. 调用基类补偿函数，先进行补偿链中上一级的补偿，并令res = CompDecorator::Compensate(axis\_info)。
2. 更新m\_current\_pos = m\_current\_mot\_pos = axis\_info->motor\_target\_pos = axis\_info->target\_pos。
3. 如果返回res为ERROR\_NONE，且螺距误差补偿已使能且轴已返回参考点，则转到e；否则转到d。
4. 返回res，结束。
5. 调用函数CheckScrewParam()，检测螺距误差参数是否正常，若是则转到f，否则转到d。
6. 计算目标位置 axis\_info->target\_pos相对参考点的距离abs\_target\_pos，即去除参考点偏移量：abs\_target\_pos = axis\_info->target\_pos – m\_axis\_home\_pos。
7. 计算参考点相对表头的位置home\_pos\_on\_table = m\_screw\_home\_no \* m\_screw\_interval。
8. 根据以上两步获得目标位置相对补偿表头的相对位置target\_pos\_on\_table = abs\_target\_pos + home\_pos\_on\_table。
9. 检测target\_pos\_on\_table是否处于补偿表的实际位置范围内，若是则转到j，否则转到d，而不对补偿表进行遍历。
10. 令i=0。
11. 检测i是否小于m\_screw\_num – 1，即不能超过倒数第二个点，若是则转到l，否则，转到d。
12. 计算第i点的目标位置pre\_target\_pos=(i-1)\*补偿间隔，及其实际位置pre\_actual\_pos为pre\_target\_pos减去第i点的补偿量；同时，计算第i+1点的目标位置nxt\_target\_pos和实际位置nxt\_actual\_pos。
13. 检测目标位置target\_pos\_on\_table是否处于第i点与第i+1点的实际位置范围里，即检测target\_pos\_on\_table是否大于等于pre\_actual\_pos，且小于nxt\_actual\_pos，若是则转到n，否则，i++，转到k继续循环。
14. 计算电机指令相对于补偿表头的位置：temp\_pos = pre\_target\_pos + m\_screw\_interval \* (target\_pos\_on\_table – pre\_actual\_pos) / (nxt\_actual\_pos - pre\_actual\_pos)。
15. 加上偏移量计算电机指令位置：m\_current\_mot\_pos = temp\_pos - home\_pos\_on\_table + m\_axis\_home\_pos。
16. 保存当前轴位置，令m\_current\_pos = axis\_info->target\_pos；然后以m\_current\_mot\_pos 更新axis\_info中的target\_pos和motor\_target\_pos，结束。

## 调用说明

插补后，通道控制把轴目标位置分别发到轴控制中，由轴控制调用补偿装饰链对轴目标位置进行补偿，最后获得电机目标位置。其中，轴控制类中定义了三个成员变量：

* Compensation\* m\_compensator，补偿实体类，没有实现补偿方法
* ScrewCompensation\* m\_1st\_compensator，螺距误差补偿，标记为补偿装饰链中第一个补偿方法
* BacklashCompensation\* m\_2nd\_compensator，反向间隙补偿，标记为补偿装饰链中第二个补偿方法

轴控制初始化时，同时对装饰链进行初始化，主要完成下列步骤：

1. 建立补偿装饰链（其中螺补必须在反向间隙之前）：

m\_2nd\_compensator->Decorate(m\_1st\_compensator->Decorate(m\_compensator))

1. 调用m\_2nd\_compensator的InputCmd函数设置装饰链的轴号，命令为SET\_COMP\_AXIS\_INDEX。
2. 调用m\_2nd\_compensator的InputCmd函数初始化装饰链，命令为INIT\_COMPENSATION。

轴控制接收到SEND\_TARGET\_POS命令时，则调用m\_2nd\_compensator的InputCmd函数，进行补偿计算，命令为COMPENSATE。由于补偿前，必须在回参考点后，建立补偿并设置已回参考点状态，这两步暂时放在轴控制初始化函数中，即默认已回参考点。

## 螺距补偿原理

### 正向补偿

假设螺距误差补偿参数如下表：

螺距误差补偿相关参数表

|  |  |  |
| --- | --- | --- |
|  | 名称 | 描述 |
|  | 螺距误差补偿点数 | 丝杠上螺距误差误差测量位置的数量，对应螺距误差补偿表的长度 |
|  | 补偿间隔 | 补偿点之间的间隔 |
|  | 参考点补偿位置（偏差号） | 参考点对应补偿表中的点编号 |
|  | 最负侧螺距误差误差补偿点号 | 轴设定的最负侧螺距误差误差补偿点号 |
|  | 最正侧螺距误差误差补偿点号 | 轴设定的最正侧螺距误差误差补偿点号 |

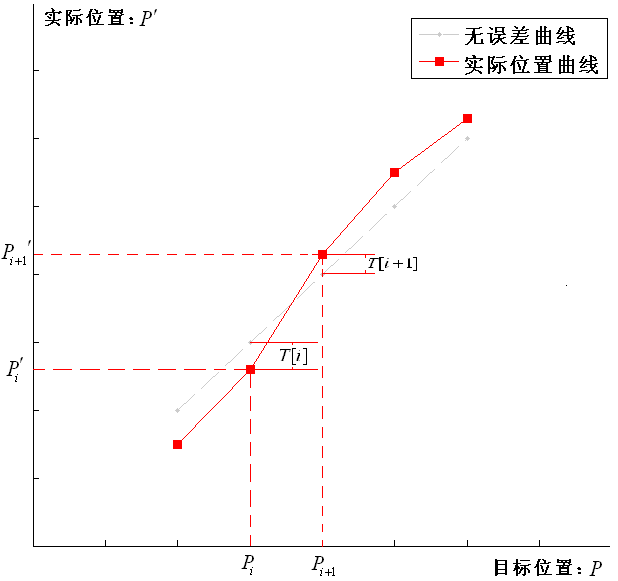
系统需维护一个点数为的螺距误差补偿数据表：

螺距误差补偿数据表

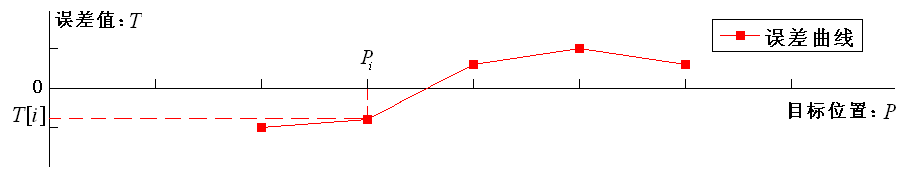
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 点号 |  |  |  | …… |  | …… |  |  |
| 补偿值 |  |  |  | …… |  | …… |  |  |

表中数据将使用激光干涉仪或其他工具测量得到。其中，表格中填写的补偿值为目标位置（）与实际位置之差：。轴实际位置与目标位置，误差与目标位置之间的关系曲线如图1和2，补偿点之间的补偿值都通过线性插值式（1）获得：

 （1）

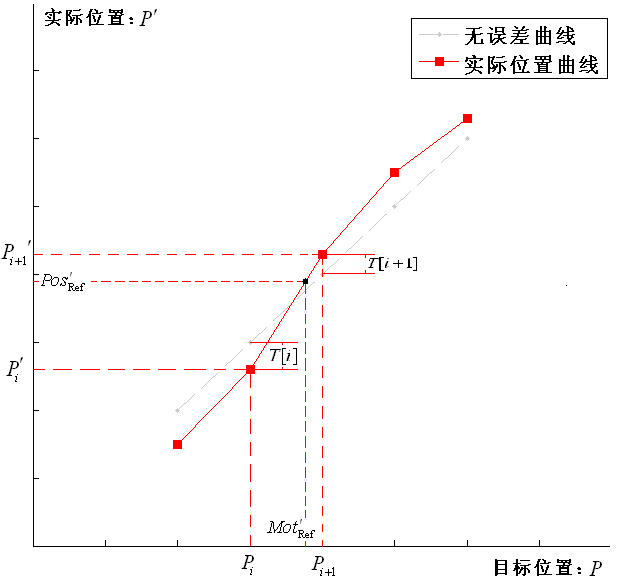


轴实际位置与目标位置关系图



误差与目标位置之间的关系图

在控制过程中，若要控制轴移动到位置（指令位置），则需通过轴实际位置与目标位置关系曲线，计算输出到电机的电机指令位置，如下图。



根据轴实际位置与目标位置关系曲线获取电机指令位置

具体步骤如下：

1. 计算相对于螺距误差补偿表原点的位置：



1. 计算处于哪两个补偿点的实际位置之间，即搜索，使得：

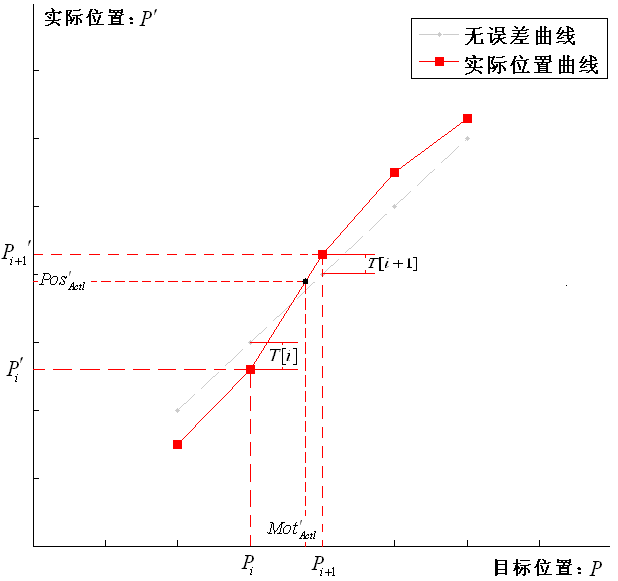


其中，。若超出的范围，则转到3，否则转到4。

1. 令，即无需进行螺距误差补偿，结束。
2. 把代入式（1）的反函数中计算电机指令相对螺距误差补偿表原点的位置：
3. 
4. 计算电机指令位置：，结束。

### 反补偿

相反的，在控制过程中，若根据反馈获取到电机的实际位置，则需通过轴实际位置与目标位置关系曲线，计算轴的实际位置，这个计算过程为螺距误差反补尝，如下图。



根据轴实际位置与目标位置关系曲线获取轴实际位置

具体步骤如下：

1. 计算相对于螺距误差补偿表原点的位置：



1. 计算处于哪两个补偿点之间：，即搜索，使得：



若超出的范围，则转到3，否则转到4。

1. 令，即无需进行螺距误差补偿，结束。
2. 把代入式（1）中计算轴实际位置相对置螺距误差补偿表原点的位置：
3. 
4. 计算电机指令位置：，结束。