GOOGOL TECHNOLOGY

CPAC OtoStudio

Advanced Programming Manual of Motion Controller

GUC-ECAT Series

V1.0

2014.10

www.googoltech.com

Copyright Statement

CAUTION	Machinery in motion can be dangerous! It is the user's responsibility to design effective error handling and safety protection methods as part of the machinery. Googol Technology shall not be liable or responsible for any incidental or consequential damages.

Contact Us

Googol Technology (Shenzhen) Ltd.

Address: 2nd Floor, West Wing, IER Building (PKU-HKUST Shenzhen Hong Kong Institution) High-tech Industrial Park, Nanshan, Shenzhen, PRC
Postal Code: 518057
Tel.: +(86) 755-26970817, 755-26970824, 755-26737236
Fax: +(86) 755- 26970821
E-mail: support@googoltech.com
URL: http://www.googoltech.com.cn

Googol Technology (HK) Ltd.

Address: Unit 1008-09, 10/F C-Bons International Center, 108 Wai Yip Street, Kwun Tong, Kowloon, Hong Kong

Tel.: +(852) 2358-1033
Fax: +(852)2719-8399
E-mail: info@googoltech.com
URL: http://www.googoltech.com

Document Version

Version	Date
1.0	2014-10-31

Foreword

Thank you for selecting Googol Technology motion controller

To repay user, we will help you establish your own control system, by providing our first-class motion controller, perfect after-sale services, and high-efficiency technical support.

More information about products of Googol Technology

Googol Technology's web site is http://www.googoltech.com. You can get more information about the company and products on our website, including company profile, product introduction, technical support, products recently released.

You can also get more information about the company and products through the phone: +(86) 755-26970817.

Technical support and after-sale services

To get our technical support and after-sale services:

E-mail: support@googoltech.com

Tel.: +(86) 755 2697-0843

Addr: Googol Technology (SZ) Ltd

2nd Floor, West Wing, IER Building (PKU-HKUST Shenzhen Hong Kong Institution) High-tech Industrial Park, Nanshan, Shenzhen, PRC.

Postal Code: 518057

Usage of this Programming Manual

By reading this manual, you will know the control functions of GUC-ECAT series motion controller, learn the usage of motion functions, and become familiar with programming of specific control function. Finally, you can program your application for controlling according to your specific control system.

User of this Programming Manual

This manual is applicable to those engineering developers who have the base knowledge of programming in C or other using Dynamic Link Library (DLL) in Windows environment, with certain work experience in motion control and understanding of the basic architecture of servo or step control.

Main Contents of this Programming Manual

This manual consists of five chapters, which introduced the **advanced motion control functions and programming of the GUC-ECAT series motion controller in detail**.

Relevant Documents

For installing and debugging of GUC-ECAT series motion controller, please refer to *User's Guide of GUC-ECAT series Motion Controller* provided together with our product.

Contents

Chapter 1 Use of Motion Function Library in OtoStudio

1.1 Use of OtoStudio software library

Ethercat bus using motion controller in CPAC software platform must call Ethercat private library, and the method is consistent with the method of using CPAC-X00-TPX.lib, and CPAC-X00-TPX.lib could be used at the same time. The library file name of CPAC-OtoBox controller is CPAC GUC-X00-TPX ECAT.lib. The same method for using CPAC GUC-X00-TPX ECAT.lib could be used to call the advanced function library, and the library file name is GUC-X00-TPX-Addition 2.2.lib.

1.1.1 Usage of the library in OtoStudio

1. Start the OtoStudio.exe，and create a new project;
2. The system adds CPAC GUC-X00-TPX.lib automatically;
3. Manually add CPAC GUC-X00-TPX ECAT.lib and CPAC GUC-X00-TPX-Addition 2.2.lib to the library file manager;

   Now, users can call any commands in DLL and program their application programs in OtoStudio.

Chapter 2 Return Values of Commands and Their Meanings

2.1 Return values of commands

CPAC controller works according to the motion controller commands sent by the host. These commands are encapsulated in DLL. User can call GUC-X00-TPX-Addition 2.2.lib in the library of CPAC controller to operate the motion controller when the user writes program to the host PC.

When receiving commands from the host, CPAC controller will give a feedback after checking and verifying the commands. The definitions of return values are listed in Tab 2-1.

Tab 2-1 Definition of Return Values of Motion Controller

Value	Meanings	Processing Methods
0	Command executed successful	
1	Command error	1. Check the execution condition of the current command
7	Command parameters error	1. Check the value of current command parameters
2	License error	1. If user needs this function, please contacts with Googol
-1	Error in communication.	1. Check drive of motion controller; 2. Check connection between motion controller and host PC; 3. Change host PC; 4. Change motion controller
-6	Failure in opening the card.	1. Check drive of motion controller 2. Check if GT_Open() was called twice. 3. Check if the card was opened in another program
-7	No response of motion controller	1. Change another motion controller

It is suggested to check each command return value in user program to confirm if the execution of the command is successful, and establish necessary error treatment mechanism to assure the safety and reliability of the program.

Chapter 3 System Configuration

Except OtoStudio configuration tools, system configuration also supports the instructions of CPAC GUC-X00-TPX-Addition 2.2.lib to configure in program run.

3.1 Command to modify configuration information

Except configuration file, User can use GT commands to initialization of control system

3.1.1 Commands summary

Tab 3-1 Summary of configuration commands

Commands	Description
GT_AlarmOff	Disable the drive alarm
GT_AlarmOn	Enable the drive alarm
GT_LmtsOn	Enable the limit
GT_LmtsOff	Disable the limit
GT_ProfileScale	Set the profile scale of axis
GT_EncScale	Set the encoder scale of axis
GT_SetStopDec	Set smooth stop decelerability and emergency stop decelerability
GT_GetStopDec	Get smooth stop decelerability and emergency stop decelerability
GT_SetStopIo	Set the input type of smooth stop and emergency stop
GT_GpiSns	Specify the effective electrical lever for digital input
GT_SetAdcFilter	Set the filter time parameter of adc input(for GTS-400-PX only)

Tab 3-2 Definition of configuration commands

GT_AlarmOff(axis)	
Axis:INT	Axis NO
GT_AlarmOn(axis)	
Axis:INT	Axis NO
GT_LmtsOn(axis, limitType)	
Axis:INT	Axis NO
LimitType:INT	Enable limit type MC_LIMIT_POSITIVE：Enable the positive limit of axis MC_LIMIT_NEGATIVE：Enable the negative limit of axis -1：Enable both of positive and negative limit of axis, default value
GT_LmtsOff(axis, limitType)	
Axis:INT	Axis NO
LimitType:INT	Disable limit type MC_LIMIT_POSITIVE：Disable the positive limit of axis MC_LIMIT_NEGATIVE：Disable the negative limit of axis

	-1：Disable both of positive and negative limit of axis, default value
GT_ProfileScale(axis, alpha, beta)	
Axis:INT	Axis NO
Alpha:INT	Alpha value of profile scale,ranging in [-32768,32767]，please refer to ii.1
Beta:INT	Beta value of profile scale,ranging in [-32768,32767]，please refer to ii.1
GT_EncScale(axis, alpha, beta)	
Axis:INT	Axis NO
Alpha:INT	Alpha value of encoder scale,ranging in [-32768,32767]，please refer to ii.1
Beta:INT	Beta value of encoder scale,ranging in [-32768,32767]，please refer to ii.1
GT_SetStopDec(profile, decSmoothStop, decAbruptStop)	
Profile:INT	Profile No
DecSmoothStop:LREAL	smooth stop decelerability,range in (0,32767]
DecAbruptStop:LREAL	emergency stop decelerability,range in (0,32767]
GT_GetStopDec(profile, pDecSmoothStop, pDecAbruptStop)	
Profile:INT	Profile No
PDecSmoothStop:POINTER TO LREAL	smooth stop decelerability
PDecAbruptStop:POINTER TO LREAL	emergency stop decelerability
GT_SetStoplo(axis, stopType,inputType, inputIndex)	
Axis:INT	Axis No，range of value: [1,8]
StopType:INT	Stop mode 0: emergency stop 1: smooth stop
InputType:INT	Digital input MC_LIMIT_POSITIVE(this macro is defined 0) positive limit MC_LIMIT_NEGATIVE(this macro is defined 1) negative limit MC_ALARM(this macro is defined 2) drive alarm MC_HOME(this macro is defined 3) home MC_GPI(this macro is defined 4) general input MC_ARRIVE(this macro is defined 5) motor arrive(GTS-400-PX only)
InputIndex:INT	Index of digital input, its range depends on inputType If inputType= MC_LIMIT_POSITIVE its value ranging in [1, 8]； If inputType= MC_LIMIT_NEGATIVE its value ranging in [1, 8]； If inputType= MC_ALARM its value ranging in [1, 8]； If inputType= MC_HOME its value ranging in [1, 8]； If inputType= MC_GPI its value ranging in [1, 16]。 If inputType= MC_ARRIVE its value ranging in [1,8]
GT_GpiSns(sense)	
Sense:UINT	Set the digital input level by bit, bit0 to bit15 represents general input 1 to general input 16. 0: original input level, the return of GT_GetDi() is same as the input level, 0 means the input is low level, 1 means the input is high level. 1: reverse input level, the return of GT_GetDi() is same as the reversed value of input level, 0 means the input is high level, 1 means the input is low level.

GT_SetAdcFilter(adc, filterTime)	
Sdc:INT	Adc No. its value ranging in [1,8]
FilterTime:INT	Time parameter of the digital input filter,its ranging in[1,50]

3.1.2 Highlights

(1) Set the direction of encoder

GT_EncSns() command can modify the counting direction of encoder, if corresponding bit of parameter is set as 1, counting direction of the encoder of corresponding axis is reversed. The definition of status bit of the parameter as show in Tab 3-3.

Tab 3-3 Definition of status bit of GT_EncSns()

Status bit	8	7	6	5	4	3	2	1	0
encoder	AUX encoder	Enc8	Enc7	Enc6	Enc5	Enc4	Enc3	Enc2	Enc1

(2) Set effective electrical level for limit switch

The default limit switch is normally closed switch. In normal status, the signal of limit switch is at low level, and when a high level input, the switch will be triggered. If normally opened switch is used, user need to call GT_LmtSns() command to change effective electrical level for limit switch.

The parameter of GT_LmtSns() indicates the effective level of the positive/negative limit switch of each axis. When one status bit of the parameter is set as 0, it means that the trigger level of corresponding limit switch is high level active. Whereas, if it is set to 1, it means that the input signal of limit switch is low level active. The corresponding relationship between the status bit of parameter and limit switch is as show in Tab 3-4.

Tab 3-4 Definition of status bit of GT_LmtSns()

Status bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Limit	Axis 8		Axis 7		Axis 6		Axis 5		Axis 4		Axis 3		Axis 2		Axis 1	
	—	+	—	+	—	+	—	+	—	+	—	+	—	+	—	+

Chapter 4 Motion Mode

Each axis of GUC-800-TPX controller can operate at Point to Point motion mode, Jog motion mode, PT motion mode, electronic gear motion mode, follow motion mode and other motion mode.This chapter describes CPAC GUC-X00-TPX-Addition 2.2.lib achieve PVT motion mode and Crd motion mode.

4.1 Crd Motion Mode

Interpolation motion mode could realize multi-shaft coordination motion, thus to complete certain motion track. Therein, the interpolation motion mode has following functions: it could realize linear interpolation and circular interpolation; it could conduct the interpolation motion for two coordinate systems at the same time; each coordinate system has two caches and could realize such functions as suspension, recovery, etc.; it has the function of delaying the cache and outputting the digital quantity of the cache; and it also has the look-ahead preprocessing function and could realize the high-speed and smooth motion on short segment with continuous track.

4.1.1 Commands summary

Tab 4-1 Summary of Crd motion mode commands

Commands	Description
GT_SetCrdPrm	Set parameters of the coordinate system
GT_GetCrdPrm	Get parameters of the coordinate system
GT_CrdData	Add Crd data to FIFO
GT_LnXY	Buffer command,two-dimensional linear interpolation
GT_LnXYZ	Buffer command,three-dimensional linear interpolation
GT_LnXYZA	Buffer command,four-dimensional linear interpolation
GT_LnXYG0	Buffer command,two-dimensional linear interpolation with symmetry profile of velocity planning
GT_LnXYZG0	Buffer command,three-dimensional linear interpolation with symmetry profile of velocity planning
GT_LnXYZAG0	Buffer command,four-dimensional linear interpolation with symmetry profile of velocity planning
GT_ArcXYR	Buffer command,Circular interpolation in XY plane described with end point position and radius
GT_ArcXYC	Buffer command,Circular interpolation in XY plane described with end point position and circle center position
GT_ArcYZR	Buffer command,Circular interpolation in YZ plane described with end point position and radius
GT_ArcYZC	Buffer command,Circular interpolation in YZ plane described with end point position and circle center position
GT_ArcZXR	Buffer command,Circular interpolation in ZX plane described with end

	point position and radius
GT_ArcZXC	Buffer command,Circular interpolation in ZX plane described with end point position and circle center position
GT_BufIO	Buffer command, set digital output value
GT_BufDelay	Buffer command, set buffer delay time
GT_BufDA	Buffer command, coordinate buffer DA output
GT_BufLmtsOn	Buffer command, enable limit switch in coordinate buffer
GT_BufLmtsOff	Buffer command,disable limit switch in coordinate buffer
GT_BufSetStopIo	Buffer command,set stop IO information of axis in coordinate buffer
GT_BufMove	Buffer command, realize the Cutter direction following function. Start some axis's point to point move.
GT_BufGear	Buffer command, realize the Cutter direction following function. Start some axis's following move.
GT_SetUserSegNum	Buffer command, set the user defined segment number.
GT_GetUserSegNum	Get the user defined segment number.
GT_GetRemainderSegNum	Get the number of unfinished segments.
GT_CrdSpace	Get the free space of the FIFO
GT_CrdClear	Clear Crd data in the FIFO
GT_CrdStart	Start Crd motion
GT_CrdStatus	Get the status of coordinate system
GT_SetOverride	Set the Programming resultant velocity
GT_SetCrdStopDec	Set the resultant acceleration of smooth stop and emergency stop
GT_GetCrdStopDec	Get the resultant acceleration of smooth stop and emergency stop
GT_GetCrdPos	Get the specified coordinate of the coordinate system
GT_GetCrdVel	Get the resultant velocity of the coordinate system
GT_InitLookAhead	Initialize the FIFO of look-ahead

Tab 4-2 Definition of Crd motion mode commands

GT_SetCrdPrm(crd, pCrdPrm)	
Crd:INT	Coordinate system ID, ranging in [1,2].
PCrdPrm:POINTER TO TCrdPrm	Set parameters of the coordinate system. TYPE TCrdPrm : STRUCT dimension:INT; profile:ARRAY[0..7] OF INT; synVelMax:LREAL; synAccMax:LREAL; evenTime:INT; setOriginFlag:INT; originPos:ARRAY[0..7] OF DINT; END_STRUCT END_TYPE Dimension: dimension of the coordinate system, ranging in [1,4].

	Profile[8]: mapping relation of coordinate system and profile.Each element ranges in [0,4]. synVelMax: maximum synthetical velocity of the coordinate system. ranging in (0,32767), unit: pulse/ms synAccMax: maximum resultant acceleration of the coordinate system. ranging in (0,32767), unit: pulse/(ms*ms). evenTime:.minimum uniform period of each interpolation. ranging in [0,32767),unit:ms. setOriginFlag: it indicates whether the profile origin position needs to be specified, the parameter is used to set machining coordinate system which is from machine coordinate system. 0: no, origin is the current profile position; 1: yes, origin is the profile position given by originPos. originPos[8]: specify the profile position of origin.
GT_GetCrdPrm(crd, pCrdPrm)	
Crd:INT	Coordinate system ID, ranging in [1,2]
PCrdPrm:POINTER TO TCrdPrm	Get parameters of the coordinate system. Please refer to GT_SetCrdPrm for definitions of the structure members.
GT_CrdData(crd, pCrdData, fifo)	
Crd:INT	Coordinate system ID, ranging in [1,2]
PCrdData:POINTER TO TCrdData	Crd data TYPE TCrdData : STRUCT motionType:INT; circlePlat:INT; pos:ARRAY[0..3] OF DINT; radius:DINT; circleDir:INT; center:ARRAY[0..1] OF DINT; vel:LREAL; acc:LREAL; velEndZero:INT; operation:TCrdBufOperation; cosI:ARRAY[0..3] OF LREAL; velEnd:LREAL; velEndAdjust:LREAL; SchyR:LREAL; END_STRUCT END_TYPE pos: coordinate of end-point position.range: [-1073741823, 1073741823], unit: pulse. vel: target velocity of the crd segment. range: (0,32767), unit: pulse/ms. acc: acceleration of the crd segment. range: (0,32767) , unit: pulse/(ms*ms). velEndZero: indicates whether the end-point velocity is forced to 0. operation: operation of delay and digital output in buffer.

	cos: an interior parameter of crd module. velEnd: end velocity of the crd segment,range: [0,32767),unit:pulse/ms, velEndAdjust: an interior parameter of crd module. r: an interior parameter of crd module. other variables: all interior parameters which is not valid to user. TYPE TCrdBufOperation : STRUCT flag:INT; delay:UINT; doType:INT; doMask:UINT; doValue:UINT; dataExt:ARRAY[0..1] OF UINT; END_STRUCT END_TYPE Flag: indicates whether there are delay or digital output operations in the crd segment, range: [0,1], 0: no; 1: yes. Delay: delay time of the delay operation ,range: [0,16383],unit:ms doType: type of digital output. 0:no digital output operations MC_ENABLE: output drive enable MC_CLEAR: output drive alarm clear. MC_GPO: general purpose I/O. doMask: bit0~bit15 indicate whether there are digital output operation in the corresponding channel. 0: no; 1: yes. doValue: bit0~bit15 are the digital output value of the corresponding channel. dataExt: an interior parameter which is not valid to user.
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_LnXY(crd, x, y, synVel, synAcc, velEnd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
X:DINT	x-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
Y:DINT	y-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
VelEnd:LREAL	End velocity of the crd segment,range: [0,32767),unit:pulse/ms, The parameter is valid when look-ahead processing isn't used, otherwise is invalid, default:0
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_LnXYZ(crd, x, y, z, synVel, synAcc, velEnd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
X:DINT	x-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
Y:DINT	y-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.

Z:DINT	z-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
VelEnd:LREAL	End velocity of the crd segment,range: [0,32767),unit:pulse/ms, The parameter is valid when look-ahead processing isn't used, otherwise is invalid,default:0
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_LnXYZA(crd, x, y, z, a, synVel, synAcc, velEnd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
X:DINT	x-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
Y:DINT	y-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
Z:DINT	z-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
A:DINT	a-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
VelEnd:LREAL	End velocity of the crd segment,range: [0,32767),unit:pulse/ms, The parameter is valid when look-ahead processing isn't used, otherwise is invalid,default:0
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_LnXYG0(crd, x, y, synVel, synAcc, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
X:DINT	Coordinate system ID, ranging in [1,2]
Y:DINT	x-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
SynVel:LREAL	y-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
SynAcc:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_LnXYZG0(crd, x, y, z, synVel, synAcc, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
X:DINT	x-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
Y:DINT	y-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
Z:DINT	z-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_LnXYZAG0(crd, x, y, z, a, synVel, synAcc, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
X:DINT	x-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.

Y:DINT	y-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
Z:DINT	z-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
A:DINT	a-Axis end-point, ranging in [-1073741823, 1073741823], unit: pulse.
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_ArcXYR(crd, x, y, radius, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
X:DINT	arc Segment x-axis end-point,range: [-1073741823, 1073741823],unit:pulse
Y:DINT	arc Segment y-axis end-point,range: [-1073741823, 1073741823],unit:pulse
Radius:LREAL	Radius of the arc segment, range: [-1073741823, 1073741823],unit:pulse Radius>0:the arc segment ≤180° Radius<0: the arc segment ＞180° This description mode cannot represent the whole circle.
CircleDir:INT	Arc direction 0:CW 1:CCW
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
VelEnd:LREAL	End velocity of the crd segment,range: [0,32767),unit:pulse/ms,default:0
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_ArcXYC(crd, x, y, xCenter, yCenter, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
X:DINT	arc Segment x-axis end-point,range: [-1073741823, 1073741823],unit:pulse
Y:DINT	arc Segment y-axis end-point,range: [-1073741823, 1073741823],unit:pulse
XCenter:LREAL	Arc center x-axis offset beside the start point
YCenter:LREAL	Arc center y-axis offset beside the start point
CircleDir:INT	Arc direction 0:CW 1:CCW
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
VelEnd:LREAL	End velocity of the crd segment,range: [0,32767),unit:pulse/ms,default:0

Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_ArcYZR(crd, y, z, radius, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Y:DINT	arc Segment y-axis end-point,range: [-1073741823, 1073741823],unit:pulse
Z:DINT	arc Segment z-axis end-point,range: [-1073741823, 1073741823],unit:pulse
Radius:LREAL	Radius of the arc segment, range: [-1073741823, 1073741823],unit:pulse Radius>0:the arc segment ≤180° Radius<0: the arc segment ＞180° This description mode cannot represent the whole circle.
CircleDir:INT	Arc direction 0:CW 1:CCW
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
VelEnd:LREAL	End velocity of the crd segment,range: [0,32767),unit:pulse/ms,default:0
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_ArcYZC(crd, y, z, yCenter, zCenter, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Y:DINT	arc Segment y-axis end-point,range: [-1073741823, 1073741823],unit:pulse
Z:DINT	arc Segment z-axis end-point,range: [-1073741823, 1073741823],unit:pulse
YCenter:LREAL	Arc center y-axis offset beside the start point
ZCenter:LREAL	Arc center z-axis offset beside the start point
CircleDir:INT	Arc direction 0:CW 1:CCW
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
VelEnd:LREAL	End velocity of the crd segment,range: [0,32767),unit:pulse/ms,default:0
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_ArcZXR(crd, z, x, radius, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Z:DINT	arc Segment z-axis end-point,range: [-1073741823, 1073741823],unit:pulse
X:DINT	arc Segment x-axis end-point,range: [-1073741823, 1073741823],unit:pulse

Radius:LREAL	Radius of the arc segment, range: [-1073741823, 1073741823],unit:pulse Radius>0:the arc segment ≤180° Radius<0: the arc segment ＞180° This description mode cannot represent the whole circle.
CircleDir:INT	Arc direction 0:CW 1:CCW
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
VelEnd:LREAL	End velocity of the crd segment,range: [0,32767),unit:pulse/ms,default:0
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_ArcZXC(crd, z, x, zCenter, xCenter, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Z:DINT	arc Segment z-axis end-point,range: [-1073741823, 1073741823],unit:pulse
X:DINT	arc Segment x-axis end-point,range: [-1073741823, 1073741823],unit:pulse
ZCenter:LREAL	Arc center z-axis offset beside the start point
XCenter:LREAL	Arc center x-axis offset beside the start point
CircleDir:INT	Arc direction 0:CW 1:CCW
SynVel:LREAL	Programming resultant velocity of the Crd segment, ranging in (0,32767), unit: pulse/ms.
SynAcc:LREAL	acceleration of the Crd segment, ranging in (0,32767), unit: pulse/(ms*ms)
VelEnd:LREAL	End velocity of the crd segment,range: [0,32767),unit:pulse/ms,default:0
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_BufIO(crd, doType, doMask, doValue, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
DoType:UINT	type of digital output. MC_ENABLE: output drive enable. MC_CLEAR: output drive alarm clear. MC_GPO: general purpose I/O.
DoMask:UINT	bit0~bit15 indicate whether there are digital output operation in the corresponding channel. 0: no; 1: yes.
DoValue:UINT	bit0~bit15 are the digital output value of the corresponding channel.
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_BufDelay(crd, delayTime, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
DelayTime:UINT	delay time of the delay operation, ranging in: [0,16383], unit: ms.

Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_BufDA(crd, chn, daValue, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Chn:INT	DA channel No., ranging in [1,8]
DaValue:INT	DA output value, ranging in: [-32768,32767], and the corresponding voltage rang is -10V~+10V
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_BufLmtsOn(crd, axis, limitType, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Axis:INT	Axis No. of which limit be enabled, ranging in [1,8]
LimitType:INT	Enable limit type MC_LIMIT_POSITIVE: Enable the positive limit of axis MC_LIMIT_NEGATIVE: Enable the negative limit of axis -1:Enable both of positive and negative limit of axis, default value
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_BufLmtsOff(crd, axis, limitType, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Axis:INT	Axis No. of which limit be disabled, ranging in [1,8]
LimitType:INT	Disable limit type MC_LIMIT_POSITIVE: Disable the positive limit of axis MC_LIMIT_NEGATIVE: Disable the negative limit of axis -1:Disable both of positive and negative limit of axis, default value
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_BufSetStopIo(crd, axis, stopType, inputType, inputIndex, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Axis:INT	Axis No. of which stop IO information be changed, ranging in [1,8]
StopType:INT	Stop Type of which information be setted 0: emergency stop 1:smooth stop
InputType:INT	Type of digital input. MC_LIMIT_POSITIVE (this macro is defined 0): positive limit. MC_LIMIT_NEGATIVE (this macro is defined 1): negative limit. MC_ALARM (this macro is defined 2): drive alarm. MC_HOME (this macro is defined 3): home. MC_GPI (this macro is defined 4): general input.
InputIndex:INT	Digital input index,rang according to type of digital input inputType= MC_LIMIT_POSITIVE,ranging in[1,8] inputType= MC_LIMIT_NEGATIVE,ranging in[1,8] inputType= MC_ALARM,ranging in[1,8] inputType= MC_HOME,ranging in[1,8] inputType= MC_GPI,ranging in[1,16]
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_BufMove(crd,moveAxis,pos,vel, acc,modal,fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]

MoveAxis:INT	Axis ID, ranging in [1,8], this axis can not be anyone of the coordinate axes.
Pos:DINT	Target position of point to point movement, unit: pulse
Vel:lreal	Target velocity of point to point movement, unit: pulse /ms
Acc:lreal	Acceleration of point to point movement, unit: pulse /(ms*ms)
Modal:INT	Command mode: 0: this command is not a modal command, which would not block the execution of the following command. 1: this command is a modal command, which would block the execution of the following command.
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_BufGear(crd,gearAxis, pos,fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
GearAxis:INT	Axis ID, ranging in: [1,8], this axis cannot be anyone of the coordinate axes
Pos:DINT	The displacement of follow movement. Unit:pulse.
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_SetUserSegNum(crd, segNum,fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
SegNum:DINT	The user defined segment number
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_GetUserSegNum(crd,pSegment, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
pSegment:POINTER TO DINT	Get the user defined segment number
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_GetRemainderSegNum(short crd,long *pSegment,short fifo=0)	
Crd:INT	Coordinate system ID, ranging in[1,2]
pSegment:POINTER TO DINT	Get the number of the unfinished Crd segments.
FifoINT	FIFO ID, ranging in [0,1], default is 0.
GT_CrdSpace(crd, pSpace, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
PSpace:POINTER TO DINT	Query free space of the FIFO
Fifo:INT	FIFO ID, ranging in [0,1], default is 0.
GT_CrdClear(crd, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Fifo:INT	FIFO ID to be cleared, ranging in [0,1].
GT_CrdStart(mask, option)	
Mask:INT	bit0~bit1 indicate the coordinate systems to be started. bit0 corresponds to coordinate system 1, and bit1 corresponds to coordinate system 2.0: do not start; 1: start.
Option:INT	bit0~bit1 indicate the FIFO number to be started in the corresponding coordinate system.bit0 corresponds to coordinate system 1, and bit1 corresponds to coordinate system 2. 0: start FIFO0; 1: start FIFO1.

GT_CrdStatus(crd, pRun, pSegment, fifo)	
Crd:INT	Coordinate system ID, ranging in[1,2]
PRun:POINTER TO INT	Get the motion status 0: the FIFO specified by the coordinate system is not in motion. 1: the FIFO specified by the coordinate system is in Crd motion mode.
PSegment:POINTER TO DINT	Get the numbe of segments that have been executed. After the re-establishment of the coordinate system or call GT_CrdClear instruction , the value will be cleared
Fifo:INT	The number of fifo that to be queried, ranging in [0,1], default is 0.
GT_SetOverride(crd, synVelRatio)	
Crd:INT	Coordinate system ID, ranging in[1,2]
SynVelRatio:LREAL	Target velocity rate, ranging in (0,1], default is 1.
GT_SetCrdStopDec(crd, decSmoothStop, decAbruptStop)	
Crd:INT	Coordinate system ID, ranging in[1,2]
DecSmoothStop:LREAL	Set resultant smooth stop acceleration, ranging in (0,32767), unit: pulse/(ms*ms).
DecAbruptStop:LREAL	Set resultant emergency stop acceleration, ranging in (0,32767), unit: pulse/(ms*ms).
GT_GetCrdStopDec(crd, pDecSmoothStop, pDecAbruptStop)	
Crd:INT	Coordinate system ID, ranging in[1,2]
PDecSmoothStop:POINTER TO LREAL	Get resultant smooth stop acceleration, unit: pulse/(ms*ms).
PDecAbruptStop:POINTER TO LREAL	Get resultant emergency stop acceleration, unit: pulse/(ms*ms).
GT_GetCrdPos(crd, pPos)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Ppos:POINTER TO LREAL	Returned profile position of “crd” coordinate, unit: pulse. This parameter is a pointer to the first element of an array. The number of elements in the array is decided by the dimension of the coordinate system.
GT_GetCrdVel(crd, pSynVel)	
Crd:INT	Coordinate system ID, ranging in[1,2]
PSynVel: POINTER TO LREAL	Returned resultant velocity, unit: pulse/ms.
GT_InitLookAhead(crd, fifo, T, accMax, n, pLookAheadBuf)	
Crd:INT	Coordinate system ID, ranging in[1,2]
Fifo:INT	FIFO ID, ranging in [0,1].
T:LREAL	Turning time .unit: ms
AccMax:LREAL	Maximum acceleration. Unit: pulse/ms2
N:INT	Size of look-ahead buffer, the value ranging in[0,32767]
PLookAheadBuf:POINTER TO TCrdData	Pointer of look-ahead buffer.

4.1.2 Highlights

(1) Establish coordinate system

In the initialization status, all the profile axes are in the single-axis motion mode and the two coordinate systems are invalid. Therefore, the coordinate systems should be established first to implement Crd motion, and then map the profile axes to the corresponding coordinate system. Each coordinate system supports at most four dimensions (X-Y-Z-A). Users can use two-dimensional (X-Y) and three-dimensional (X-Y-Z) coordinate system to describe the motion trajectory.

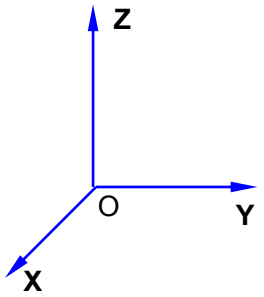

Fig 4-1 Right-handed coordinate system

The motion described in the coordinate system is mapped to the corresponding profile axis by calling GT_SetCrdPrm(). The motion controller controls the axes according to the mapping relation to implement desired motion trajectory. All the profile axes mapped should be in static status when calling GT_SetCrdPrm().

Example: establish coordinate system

```
……
Rtn:INT;
CrdPrm: TCrdPrm;                    (*Define the structure variable for coordinate system *)
First:BOOL:=TRUE;
-----------------------------------------------------------------------------------------
IF First THEN
SysMemSet(ADR(crdPrm),0,sizeof(crdPrm));          (*Initialize variables to 0 *)
crdPrm.dimension:=2;                 (*two-dimensional coordinate system *)
crdPrm.synVelMax:=500;               (*Maximum resultant velocity: 500pulse/ms *)
crdPrm.synAccMax:=1;                 (*Maximum resultant acceleration: 1pulse/ms^2*)
crdPrm.evenTime := 50;               (*Smooth time: 50ms *)
crdPrm.profile[0] := 1;              (*map profiler1 to X-axis *)
crdPrm.profile[1] := 2;              (*map profiler2 to Y-axis *)
crdPrm.setOriginFlag := 1;           (*Specify origin *)
```

```
crdPrm.originPos[0] := 100;

crdPrm.originPos[1] := 100;

rtn := GT_SetCrdPrm(1,ADR(crdPrm));        (*Establish coordinate system 1 and set parameters *)

First:=FALSE;

END_IF

……
```

Illustration:

dimension: dimension of the coordinate system, ranging in [1,4]. The coordinate system established in the example program is two-dimensional, i.e. X-Y coordinate system.

synVelMax: the maximum resultant velocity that the coordinate system supports. If the target velocity given by users is greater than the maximum, then it is set to the maximum.

synAccMax: the maximum resultant acceleration that the coordinate system supports. If the acceleration given by user is larger than the maximum, then it is set to the maximum.

evenTime: minimum even pace time of each Crd segment. If the Crd segment is short and the target velocity is high, the resultant velocity looks like the curve illustrated in Fig 4-2. There are only accelerating segment and decelerating segment, and form an angle. The value of acceleration changes from positive to negative instantly thus causes a big impulse. By setting evenTime, the target velocity can be decreased. The velocity curve will look like that illustrated in Fig 4-3, thus the impulse is reduced.

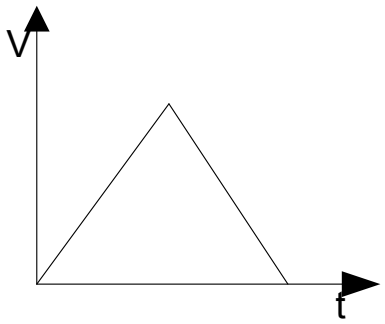

Fig 4-2 evenTime=0

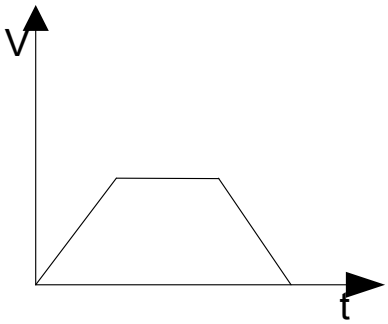

Fig 4-3 evenTime>0

profile[x]: mapping relation of coordinate system and profile. Profile[0..7] mapped to 1~8 profile axis. For example, if profile axisl is not used in coordinate system, the value of profile[x] is 0; if profile axis1 is used as X axis, the value of profile[x] is 1. Similary,Y axis is 2, Z axis is 3, A axis is 4. Multi-profile axes is to be mapped to the same coordinate axis is not allowed. And a profile axis be mapped to different axes is not allowed too, otherwise ,the command return error. Coordinate system's axis number ranges in [1,4].

setOriginFlag: it indicates whether the profile origin position needs to be specified, the parameter is used to set machining coordinate system which is from machine coordinate system. 0: origin is the current profile position; 1: origin is the profile position given by originPos.

originPos[x] : specify the profile position of origin, that is offset relative to machine coordinate system. Establishing a machining coordinate system as shown in Fig 4-4.

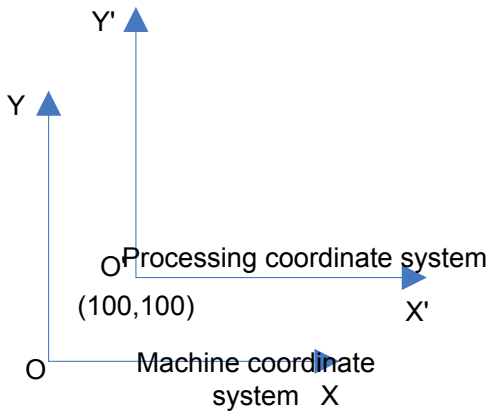

Fig 4-4 Offset of the machining coordinate system

(2) Coordinate movement

To realize coordinate movement, the motion controller provides a buffer with. It allows user to push several motion path and parameter commands into the buffer, and then start motion,the controller will execute the data in order, till all the data was executed.

There are two Crd FIFOs (FIFO0 and FIFO1) in each coordinate system, FIFO0 is the principal motion FIFO, and FIFO1 is auxiliary motion FIFO, and there are 4096 Crd segments in each FIFO.

FIFO support dynamic management, that is, after the crd motion, the FIFO will be released automatically, users are allowed to transport new data. In this way, it can support more than 4096 Crd segments。

Linear interpolation program：

```
…… …… ……
Rtn:INT;
Run:INT;                          (*define query variable of motion status for coordinate system *)
Segment:DINT;                     (*the number of finished segment *)
First:BOOL:=TRUE;
----------------------------------------------------------------------------------------------------
IF First THEN
rtn := GT_AxisOn(1);
rtn := GT_AxisOn(2);

rtn := GT_CrdClear(1,0);          (*clear the data in FIFO0 of coordinate 1*)
(*The first Crd data *)
rtn := GT_LnXY(1,200000,0,100,0.1,0,0);
                                  (*transfer linear interpolation data to FIFO0 of coordinate 1 *)
                                  (*Coordinate of end-point: (200000,0)*)
```

```
                                        (*target velocity : 100pulse/ms*)
                                        (*acceleration : 0.1pulse/ms^2*)
                                        (*velocity at end-point :0*)
    (*The second Crd data*)
    rtn = GT_LnXY(1,100000,173205,100,0.1,0,0);
    (*buffer digital output*)
    rtn = GT_BufIO(1,MC_GPO,16#ffff,16#55,0);   (*Type of digital output: general purpose *)
                                        (*All the bit0~bit15 output *)
                                        (*Output value is 0x55*)
    (*The third Crd data*)
    rtn := GT_LnXY(1,-100000,173205,100,0.1,0,0);
    (*buffer digital output*)
    rtn := GT_BufIO(1,MC_GPO,16#ffff,16#aa,0);
    (*The fourth Crd data*)
    rtn := GT_LnXY(1,-200000,0,100,0.1,0,0);
    (*buffer delay*)
    rtn := GT_BufDelay(1,400,0);                (*delay 400ms *)
    (*The fifth Crd data*)
    rtn := GT_LnXY(1,-100000,-173205,100,0.1,0,0);
    (*buffer digital output *)
    rtn := GT_BufIO(1,MC_GPO,16#ffff,16#55,0);
    (*buffer delay *)
    rtn := GT_BufDelay(1,100,0);
    (*The sixth Crd data*)
    rtn := GT_LnXY(1,100000,-173205,100,0.1,0,0);
    (*The seventh Crd data*)
    rtn := GT_LnXY(1,200000,0,100,0.1,0,0);
    rtn := GT_CrdSpace(1,ADR(space),0);     (*Get free space of FIFO0 of coordinate system1*)
    rtn := GT_CrdStart(1,0);                (*Start Crd motion of FIFO0 of coordinate system1*)
    First:=FALSE;
    END_IF
```

```
(*The value of run is 1 if the coordinate system is in motion *)
rtn := GT_CrdStatus(1,ADR(run),ADR(segment),0);
…… …… ……
```

Example specification：

Data is transmitted to the interpolation cache through GT_LnXY() instruction which includes terminal coordinate, acceleration and target speed, and GT_BufIO() instruction could be called to realize the digital quantity output in the cache and GT_BufDelay() instruction could be called to realize the delay operation in the cache. In this routine, totally 7 sections of motion data are transmitted to the interpolation cache, and the running result of the routine is a hexagon as shown in Fig 4-5. Therein, cache delay and digital quantity output are conducted during the motion..

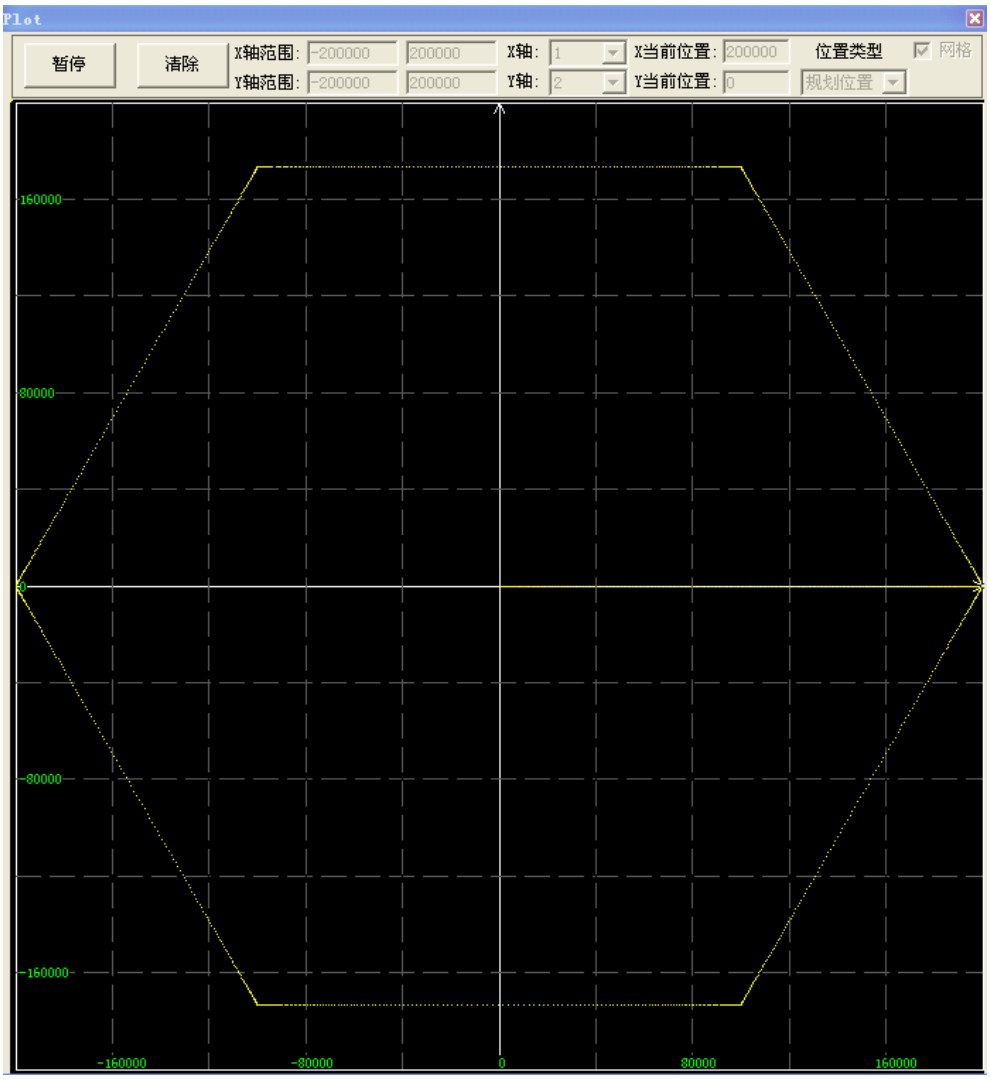

Fig 4-5 Trajectory of linear interpolation example

GT_CrdStatus() instruction could be used to inquire FIFO motion state (moving or stationary) designated by coordinate system, number of segments that have currently completed the interpolation motion, wherein the number of segments is accumulatively counted after the first GT_CrdStart() is called since the establishment of the coordinate system and is reset till the coordinate system is destroyed or

GT_CrdClear() is called.

Circular interpolation program：

```
Rtn:INT;
Run:INT;                                    (*Define motion status query variable*)
Segment:DINT;                               (*Define query variable for the number of segments finished *)
First:BOOL:=TRUE;
-----------------------------------------------------------------------------------------------------
IF First THEN
rtn := GT_AxisOn(1);
rtn := GT_AxisOn(2);

rtn := GT_CrdClear(1,0);                    (*clear the data in FIFO0 of coordinate 1*)

(*linear interpolation data*)
rtn := GT_LnXY(1,200000,0,100,0.1,0,0);

(*circulaer interpolation data *)
rtn := GT_ArcXYC(1,200000,0,-100000,0,0,100,0.1, 0,0);
                                            (*describe the whole circle with center point mode *)
                                            (*center point: (100000,0)*)
                                            (*coincide with the end-point and start-point: (200000,0)*)
                                            (*CW circle*)
                                            (*target velocity: 100pulse/ms*)
                                            (*acceleration: 0.1pulse/ms^2*)
                                            (*end velocity: 0*)
(*circulaer interpolation data *)
rtn := GT_ArcXYR(1,0,200000,200000,1,100,0.1,0,0);
                                            (*1/4 circle described by radius mode *)
                                            (*end point: (0,200000)*)
                                            (*radius: 200000*)
                                            (*CCW circle *)
```

```
rtn := GT_LnXY(1,0,0,100,0.1,0,0);          (*back to the origin *)
rtn := GT_CrdSpace(1, ADR(space),0);        (*Get free space of FIFO0 of coordinate system1*)
rtn := GT_CrdStart(1,0);                    (*Start Crd motion of coordinate system1*)
First:=FALSE;
END_IF

rtn := GT_CrdStatus(1,ADR(run),ADR(segment),0);
(*Get motion status of FIFO0 of coordinate system1*)
(*The value of run is 1 if the coordinate system is in motion *)
IF run=1 THEN
…… …… ……
```

The trajectory of circular interpolation example is illustrated in Fig 4-6.

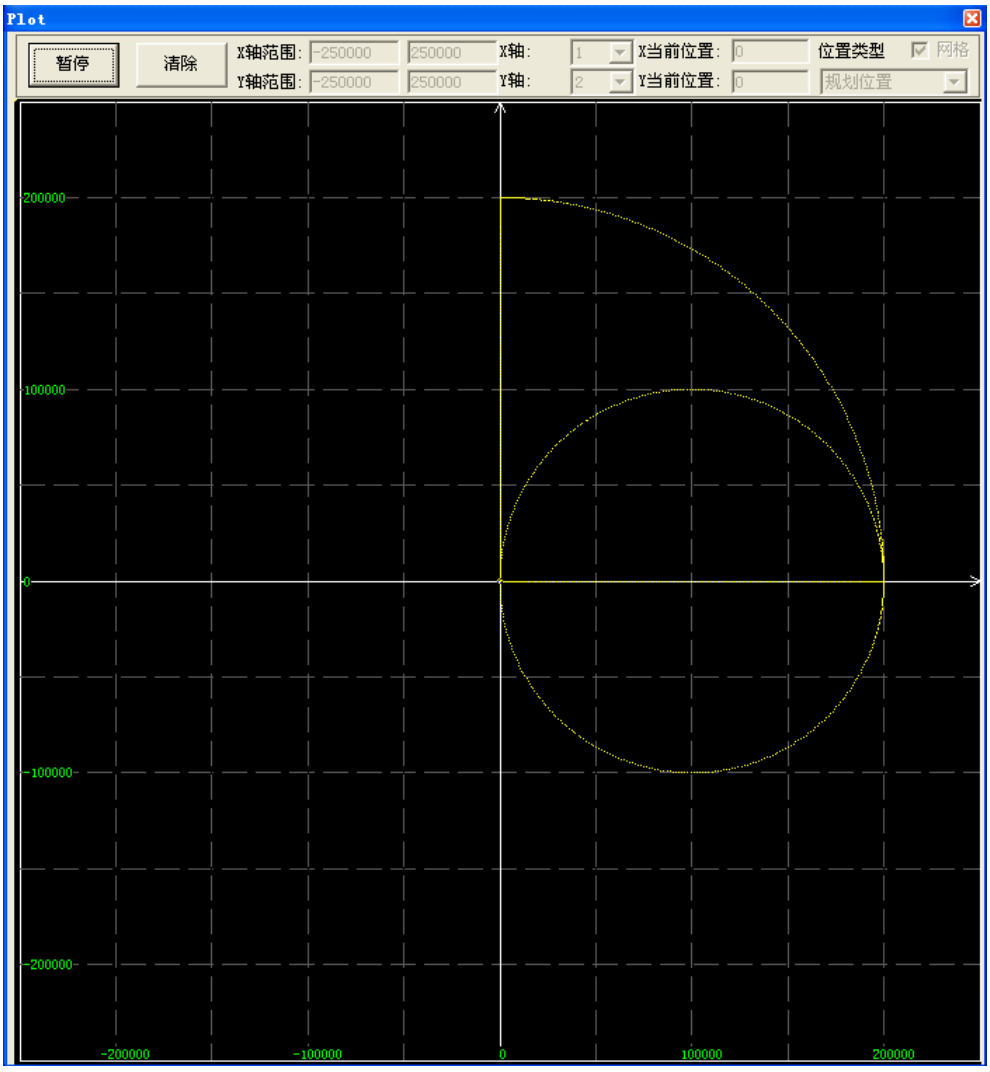

Fig 4-6 Trajectory of circular interpolation example

Controller supports circular interpolation on XY plane, YZ plane and ZX plane, arc rotating direction is defined according to the right-hand rotation rule. Form the “top” of two-dimensional coordinate plane(i.e. the positive direction of the third axis which is vertical to the two-dimensional coordinate plane), to confirm the CW direction and CCW direction. In short to remember: Extend the thumb of right hand, and make fist with the other four fingers; the thumb points to the positive direction of the third axis and the direction of the other four fingers is the CCW rotating direction. When the coordinate system is a two-dimensional system(X-Y), the CCW direction of circular interpolation in XOY coordinate plane is defined as the same way. As illustrated in Fig 4-7.

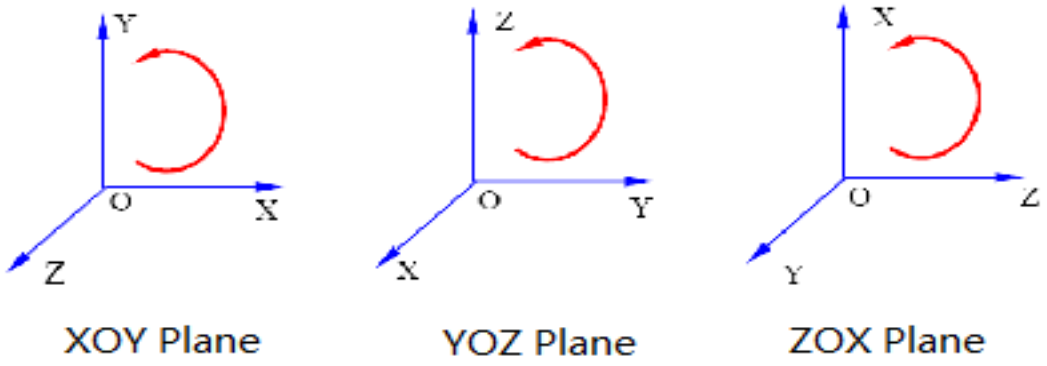

Fig 4-7 CCW direction of circular interpolation

Controller provides two mode to describe circular interpolation: radius mode and center point mode. User can choose the suitable mode to program. Both follow the G code standard.

1) Radius mode

Radius mode use GT_ArcXYR(),GT_ArcYZR() and GT_ArcZXR() command to describe the circular interpolation. User need to input the end point,radius,circular rotating direction, velocity and acceleration and so on. Radius can be positive or negative, Its absolute value is the radius of arc. If radius > 0, means the arc ≤180 degree, if radius < 0, means the arc > 180 degree. This mode cannot describe a whole circle. As illustrated in Fig 4-8.

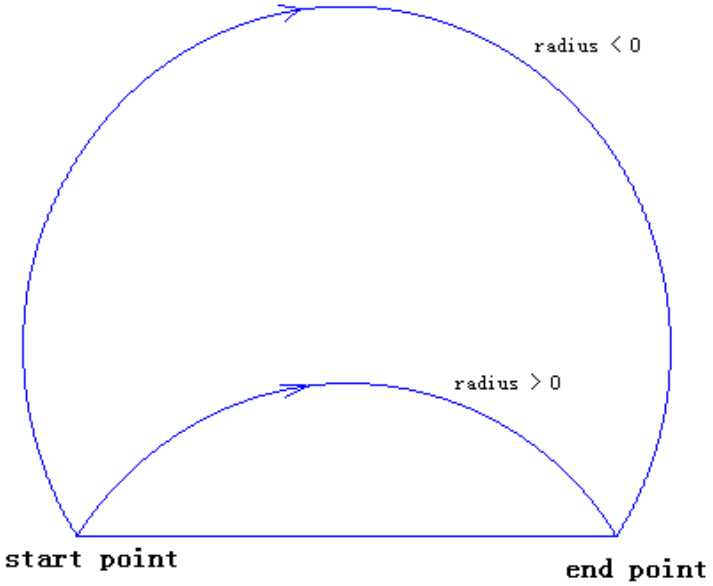

Fig 4-8 Schematic diagram of radius > 0 and radius < 0

2) Center point mode

Center point mode use GT_ArcXYC(),GT_ArcYZC() and GT_ArcZXC() command to describe the circular interpolation. User need to input end point, offset between the center point and start point, circular rotating direction, velocity and acceleration. Offset between the center point and start point show as Fig 4-9.

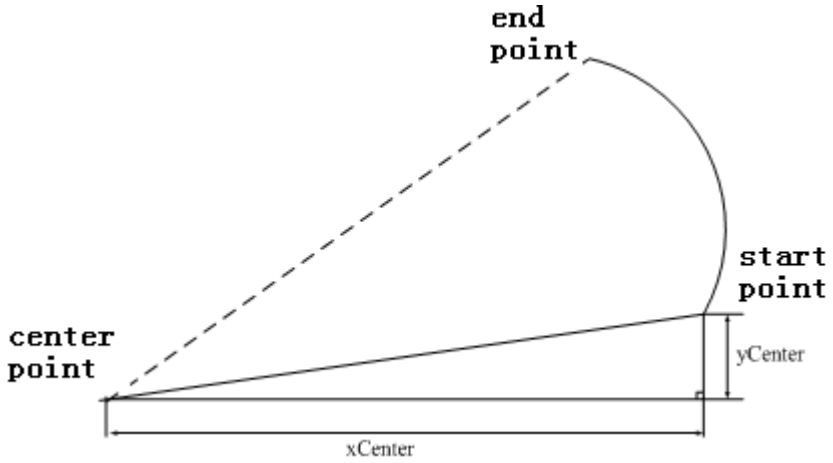

Fig 4-9 Center point description

The parameter of center point is signed offset to the start point. If the start point is (xStart, yStart), the center point parameter set by user is (xCenter, yCenter), then the center point is (xStart+xCenter, yStart+yCenter). If start point is coincide with end point, there will be a whole circle motion.

User need ensure the parameter can describe an arc correctly, if not, command result will be wrong (return value is 7).

(3) Look-ahead processing

Features of short segment interpolation processing: In order to ensure the smoothness of the contact surface between the tools and the processing piece, it is necessary to ensure the constancy of the tangential speed during the track motion and meanwhile ensure the track processing precision.

According to Fig 4-10, we could know that the terminal point of each segment in the Figure has an inflection point (the track has obviously changed point), and the speed must be reduced, but whether the speed shall be reduced to 0 depends on the length, speed, acceleration, speed change limit of the inflection point of the segment, etc. and the terminal speed of various segments calculated according to relevant processing parameters.

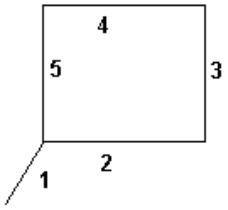

Fig 4-10 X-Y Plane Multi-segment Track Chart

According to Fig 4-11, we could know that the segment combination of the fitted curve track on the basis

of short segments shall have the constant tangential speed during processing, but the speed must be reduced to a rational value (rational terminal speed) at the inflection point (the 8th point) in order to ensure that the processing actuator (the basic machine and the motor) could bear the speed variation caused by the track feature change at the inflection point.

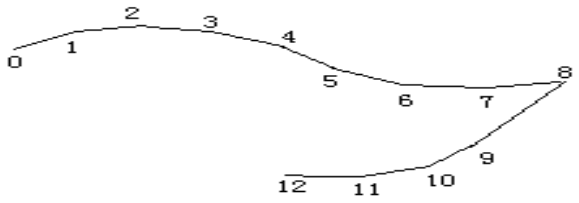

Fig 4-11 –Y Plane Short Segment Track Chart

In order to solve the conflict between high speed and high precision, the look-ahead preprocessing mode is adopted for the motion controller to control the speed during the motion process.

Users could call the look-ahead preprocessing module provided by the motion controller according to the technological characteristic parameter (pulse equivalent, target speed, maximum acceleration, allowable turning time, etc.) to give out the terminal speed of each segment. Meanwhile, the motion controller executes the acceleration and deceleration operations strictly according to the terminal speed of each segment. In the motion controller, the set of instructions for realizing the speed preprocessing function is called look-ahead preprocessing instruction (also called LookAhead).

From Fig 4-12 we can get that the velocity increases obviously in the process of micro-segment machining by using the look-ahead processing module.

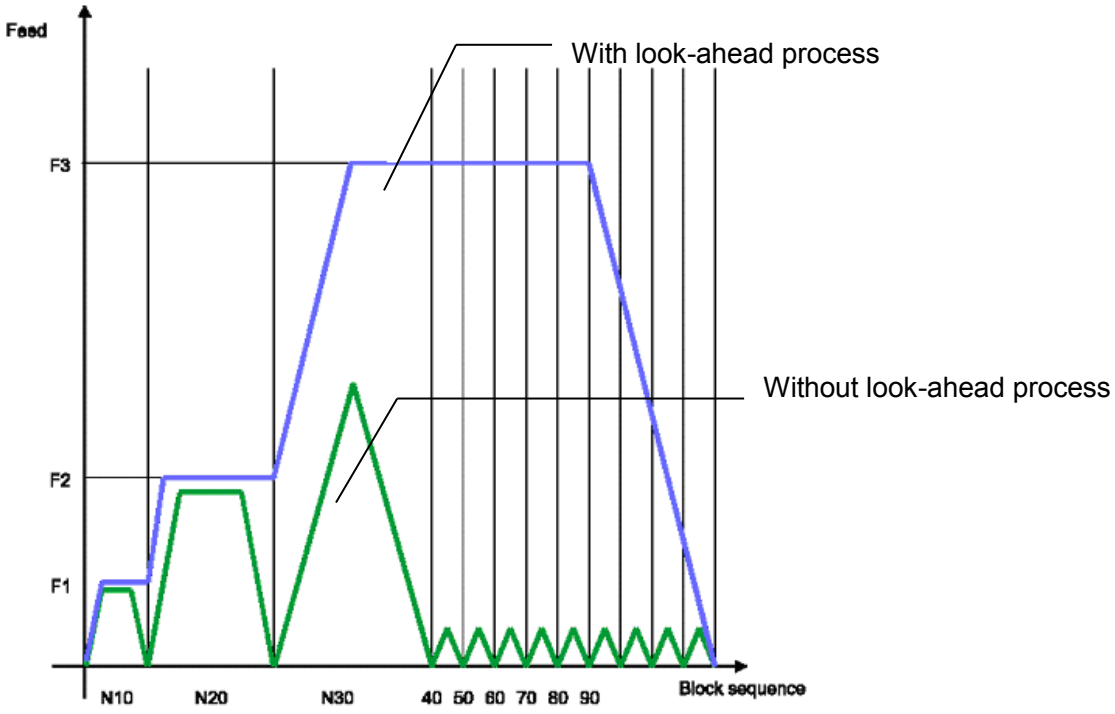

Fig 4-12 Velocity curve comparison between with and without look-ahead process

Example of look-ahead processing:

```
Rtn:INT;
I:INT;
    CrdDataSend: TCrdData;
CrdData:ARRAY[0..199] OF TCrdData;                    (*define buffer for look-ahead processing *)
PosTest:ARRAY[0..1] OF DINT;
First:BOOL:=TRUE;
IF First THEN
rtn := GT_AxisOn(1);
rtn := GT_AxisOn(2);

    rtn := GT_InitLookAhead(1,0,5,1,200,crdData);
                                    (*Initialize look-ahead module of FIFO0 of coordinate system1*)
    (*Insert Crd data: micro-segment machining *)
    posTest[0] := 0;
    posTest[1] := 0;
FOR I:=1 TO 300 BY 1 DO
        rtn := GT_LnXY(1,8000+posTest[0],9000+posTest[1],100,0.8,0,0);
        posTest[0] := posTest[0]+1600;
        posTest[1] := posTest[1]+1852;
END_FOR
rtn := GT_CrdData(1,0,0);              (*insert the data in look-ahead buffer to Crd buffer *)
rtn := GT_CrdStart(1,0);
First:=FALSE;
END_IF
……
```

Illustration:

Turning time (T): the third parameter of GT_InitLookAhead(), unit: ms. The empirical range of T is 1ms~10ms. The greater the T is, the higher the terminal velocity calculated is, but the machining precision decreases. On the contrary, the machining precision increases but the terminal velocity calculated is low. Therefore the value of T should be chosen suitably.

Maximum acceleration (accMax): the forth parameter of GT_InitLookAhead(), unit: pulse/(ms*ms). The

maximum acceleration the system supports. Its value varies for different mechanism and motor drive.

Size of look-ahead cache and memory pointer of look-ahead cache: this look-ahead module is composed of look-ahead cache memory provided by user, so the user could properly define the size of the cache according to the needs thereof and the computer conditions, the larger the look-ahead cache is, the larger the memory occupies. In detail, the user needs to firstly define an interpolation data array variable and applies certain cache, and then transfer the memory pointer to the look-ahead module of the motion controller through GT_InitLookAhead() instruction. During the look-ahead processing, the user could not operate the cache, or else the data in the look-ahead cache could be destroyed, thus causing data error.

If the number of segments in the look-ahead buffer is not zero, the Crd data transferred by buffer command enter the look-ahead buffer. After the look-ahead buffer is full, the data entered first into the look-ahead buffer will enter Crd buffer if there are new data inserted.

If all the Crd data have been input, and there are still data in look-ahead buffer that have not entered Crd buffer, then the GT_CrdData(1,NULL,0) should be called to transfer the data in the look-ahead buffer to the Crd buffer until the look-ahead buffer is clear.

If the data volume is large, users should use GT_CrdSpace() to get the free space of Crd buffer. If there are free space, calling buffer command will transfer data, otherwise buffer command will return error which indicates failure of data transfer and the data should be transferred again.

If the look-ahead preprocessing function is not used, then the motion controller will not optimize the terminal speed and the target speed of the interpolation segment but control the speed strictly according to the target speed and the terminal speed designated by the user. If the user calls the instruction of GT_LnXYG0(), GT_LnXYZG0() and GT_LnXYZAG0(), then the motion instruction will finish a complete acceleration and deceleration process. In other words, the resultant speed of each segment is started from 0 and is also ended at 0. In case of calling other interpolation motion instructions (including linear interpolation and circular interpolation instructions), the user could designate the target speed and the terminal speed of the interpolation segment and meanwhile the motion controller will control the speed strictly according to the target speed and the terminal speed designed by the user.

After the look-ahead preprocessing function is used, the controller will set a rational value which is not always 0 for the terminal speed of each segment according to the parameter set by the user. If the terminal speed of certain interpolation data is required to be 0 according to user's technological requirement, then it is necessary to call GT_LnXYG0(), GT_LnXYZG0() or GT_LnXYZAG0() to set the terminal speed of the linear interpolation segment as 0. In case of calling other interpolation motion instructions (including linear interpolation and circular interpolation instructions), the terminal speed set by the user will be invalid and the actual terminal speed will be a rational terminal speed calculated by the look-ahead preprocessing module according to the look-ahead preprocessing parameter set by the user and the motion track. In addition, if the cache delay exists between certain interpolation motion data and the next interpolation motion data, then the terminal speed of the interpolation motion will be set as 0.

Look-ahead processing only supports 3-axis or less than 3 axes Crd motion. If the coordinated system established is more than 3 axes, it will return error when using look-ahead processing, and 7 (parameter error) is returned when calling buffer command.

The curve of the resultant speed without look-ahead preprocessing is as shown in Fig 4-13, and the

resultant speed will ceaselessly change. The curve of the resultant speed with look-ahead preprocessing is as shown in Fig 4-14. Since the interpolation motions in the routine are in the same straight line, the speed could be kept at the target speed, thus to greatly improve the processing efficiency.

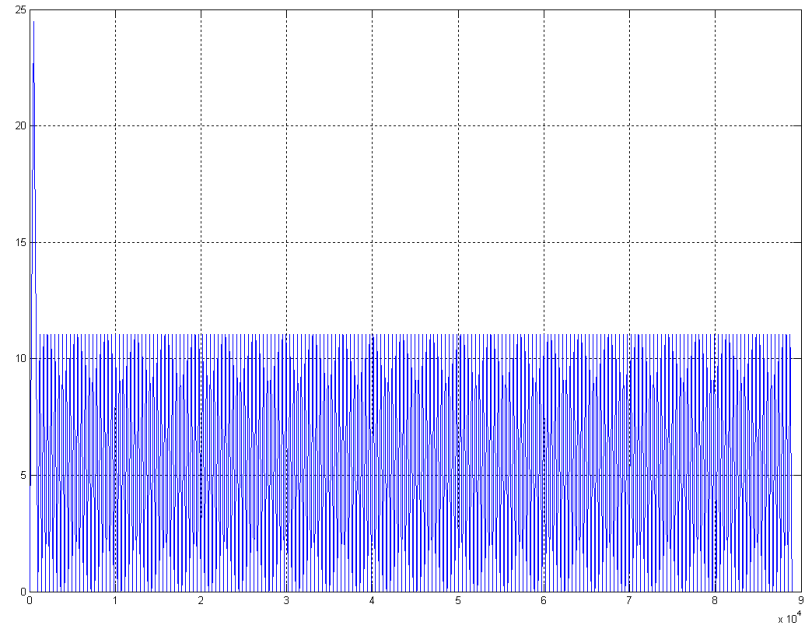

Fig 4-13 Velocity curve without look-ahead processing

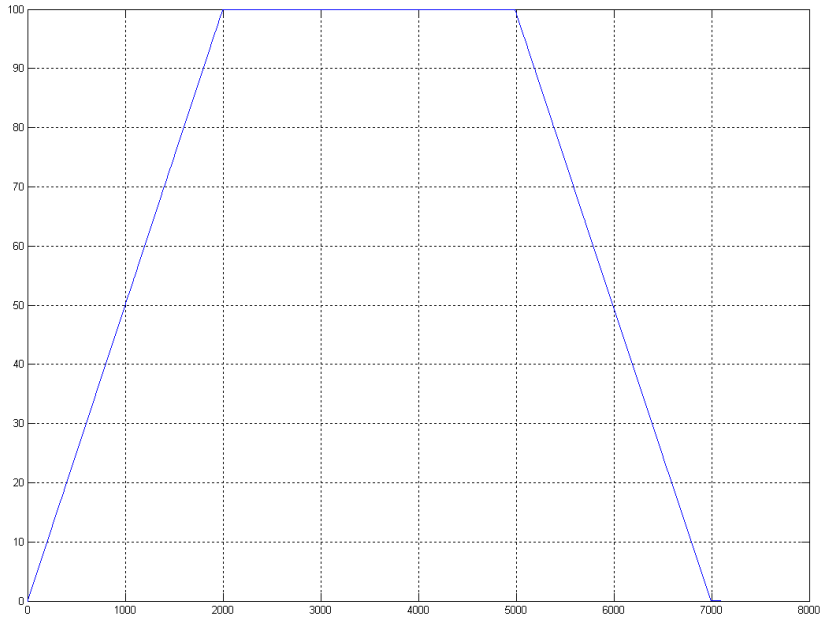

Fig 4-14 Velocity curve without look-ahead processing

(4) Suspension and resumption of FIFO

During cache interpolation process, user may stop processing and check the processing effect or conduct other operations such as tool changing, etc. after stopping processing. The motion controller supports the above operations. In order to realize the above operation process, two interpolation caches are provided to each coordinate system: FIFO0 and FIFO1. Both the two caches have 4096 segments of interpolation caches and could independently set their own look-ahead preprocessing caches.

FIFO0 is the principal FIFO, and the Crd data of principal Crd motion should be stored in FIFO0. The Crd motion of FIFO0 can be suspended to execute the Crd motion of FIFO1. After the motion of FIFO1 finished, the motion of FIFO0 can resume from the breakpoint.

FIFO1 is the auxiliary FIFO, and the Crd data of auxiliary Crd motion can be stored in FIFO1. Only if the motion of FIFO0 stops, the data in FIFO1 can be transferred. If the motion of FIFO0 is executing, transferring data to FIFO1 will cause error. The motion of FIFO1 can also be suspended and resumed, but in the interval of suspension and resumption the motion of FIFO0 cannot execute, otherwise FIFO1 will be cleared and the motion of FIFO1 cannot be resumed.

If the motion of FIFO0 needs to be resumed after the motion of FIFO1 finishes, the coordinate location value must be the same before and after the suspension of FIFO0. Otherwise FIFO0 cannot start motion and return error by calling GT_CrdStart().

(5) Cutter direction following

Tool following refers to the technology that part of shaft changes along with the change of the resultant displacement of the interpolation motion during the interpolation motion process, thus to enable the tool to be always at the suitable processing direction during the processing. The interpolation module of this controller has two instructions to realize this technology: GT_BufMove() and GT_BufGear(), wherein GT_BufMove() instruction could be used to interpolate modal and modeless point motions during the interpolation motion process while GT_BufGear() instruction could be used to realize the motion of other shafts following the interpolation resultant displacement

1) Point to point motion in interpolation process

In the interpolation process Point to Point movement of other axis in the interpolation process is realized by pushing GT_BufMove() command into command buffer. It should be particularly noted that the second parameter of this command is NO. of axis to motion in Point to Point mode. The axis to move in point to point mode cannot be axes in the coordinate system; what's more, if the axis is not in the point to point mode and is moving, this command cannot be executed correctly. The third parameter is the target position of point to point movement, which is the absolute position relative to the zero point of the machine. The forth parameter is the target velocity of the point to point movement, which must be a positive value. The fifth parameter is acceleration of the move, which must be a positive value. The sixth parameter set the command to be modal or non-modal. Modal command means while executing this point to point movement, the following commands in the interpolation buffer would wait until it's finished; **non-modal command** means: after motion controller starts the point to point movement, it execute the next command in the command buffer without waiting for the end of the point to point movement. The example is illustrated as follows:

```
PROGRAM PLC_PRG
VAR
  Enable : BOOL;
  Rtn : INT;
  Run : INT; (*define query variable of motion states of coordinate system *)
  Segment : DINT; (*define variable for the number of finished segment *)
```

```
END_VAR

---------------------------------------------------------------------------------------------

If Enable THEN

    rtn := GT_CrdClear(1,0); (*clear the data in FIFO0 of coordinate 1*)

    rtn := GT_LnXY(1,200000,200000,100,0.1,0,0); (*linear interpolation command*)

rtn := GT_BufMove(1,4,50000,30,0.1,0,0); (* point to point mode command in buffer
                                            ID of axis: 4
                                            target position: 50000 pulse
                                            target velocity: 100 pulse/ms
                                            target acceleration: 0.1 pulse/(ms*ms)
                                            non-modal command *)
    rtn := GT_LnXY(1,200000,0,100,0.1,0,0); (*linear interpolation command*)
    rtn := GT_BufMove(1,4,100000,30,0.1,1,0); (*point to point mode command in buffer
                                            ID of axis: 4
                                            target position: 100000 pulse
                                            target velocity: 100 pulse/ms
                                            target acceleration: 0.1 pulse/(ms*ms)
                                            modal command *)
    rtn := GT_ArcXYC(1,-200000,0,-200000,0,0,100,0.1,0,0); (*circular interpolation command *)

Enable := FALSE;

END_IF
```

The result of the example code is as Fig 4-15.

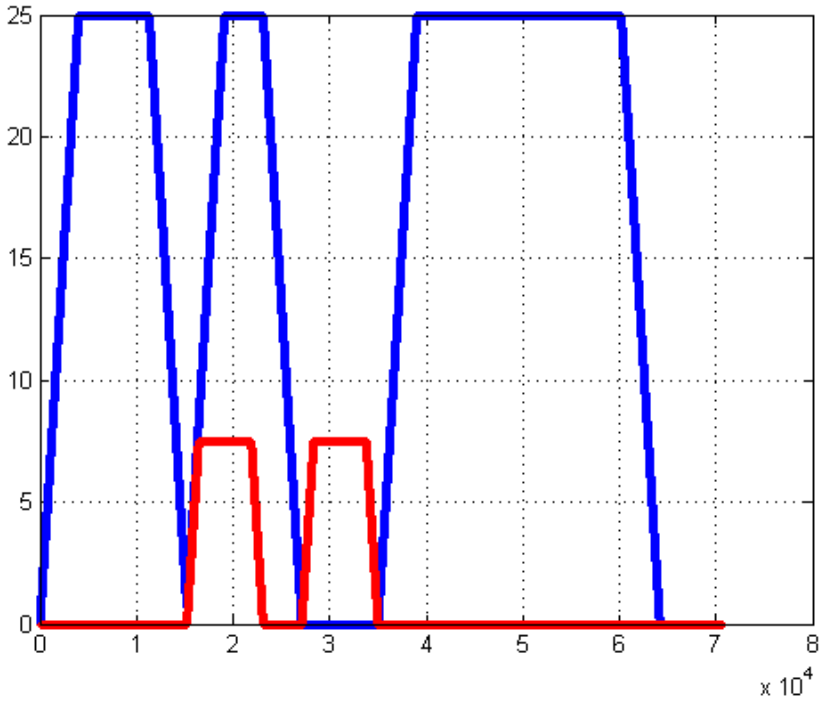

Fig 4-15 Point to Point motion trajectory of interpolation buffer

The blue line is the resultant velocity of interpolation movement; the red line is the resultant velocity of the point to point moving axis. According to the figure, the first point to point movement is a non-modal command, which is moving with the interpolation at the same time; the second point to point movement is a modal command, it blocks the interpolation, the interpolation resumes after the point to point movement finished. When using the point to point movement function of the interpolation module, pay attention to the following matters:

a) The target position of point to point movement is absolute position relative to the zero point of the machine.

b) If the last point to point movement in the buffer isn't finished, while a new point to point movement is sent, then the controller would profile according to the new point to point movement command, which means the target position the target velocity in the interpolation buffer can be modified.

c) The point to point movement would not stop when user stops the movement of the interpolation buffer. To stop the point to point movement, execute GT_Stop() to stop the axis's moving. When the buffer movement resumes, user must check the point to point mode axis to ensure it's at the proper position

d) In the modal point to point moving process, if the axis stops by exceptional results such as triggered a limit switch, the remaining command in the interpolation buffer would not be executed any more. In this situation, user has to check the exceptional results, and reset the relative parameters, and make the system normal before resume working

2) Follow motion in interpolation process

Following movement of other axis in the interpolation process is realized by pushing GT_BufGear() command into command buffer. The second parameter of this command is the axis ID which to move in

following mode.It should be noted that, this axis can not be an axis in the coordinate system; if this axis is moving when the GT_BufGear() is sent, the command cannot be executed correctly. The third parameter is the distance of the follow movement, this value is a relative value, which is the following axis has to move in the next interpolation moving process.

Example is illustrated as follows:

```
PROGRAM PLC_PRG
VAR
   Enable : BOOL;
   Rtn : INT;
   Run : INT; (*define query variable of motion states of coordinate system *)
   Segment : DINT; (*define variable for the number of finished segment *)
END_VAR
----------------------------------------------------------------------------------------
If Enable THEN
      rtn = GT_CrdClear(1,0); (*clear the data in FIFO0 of coordinate 1*)
      rtn = GT_LnXY(1,200000,200000,100,0.1,0,0); (*linear interpolation command *)
      rtn = GT_BufGear(1,4,50000, 0); (*following mode command in buffer
                                        axis ID: 4
                                        follow distance: 50000 pulse *)
      rtn = GT_LnXY(1,200000,0,100,0.1,0,0); (*linear interpolation command *)
      rtn = GT_BufGear(1,4,50000,0); (*following mode command in buffer
                                        axis ID: 4
                                        follow distance: 50000 pulse *)
      rtn = GT_ArcXYC(1,-200000,0,-200000,0,0,100,0.1,0,0); (*circular interpolation command *)
      Enable := FALSE;
END_IF
```

The result of the example code is as Fig 4-16.

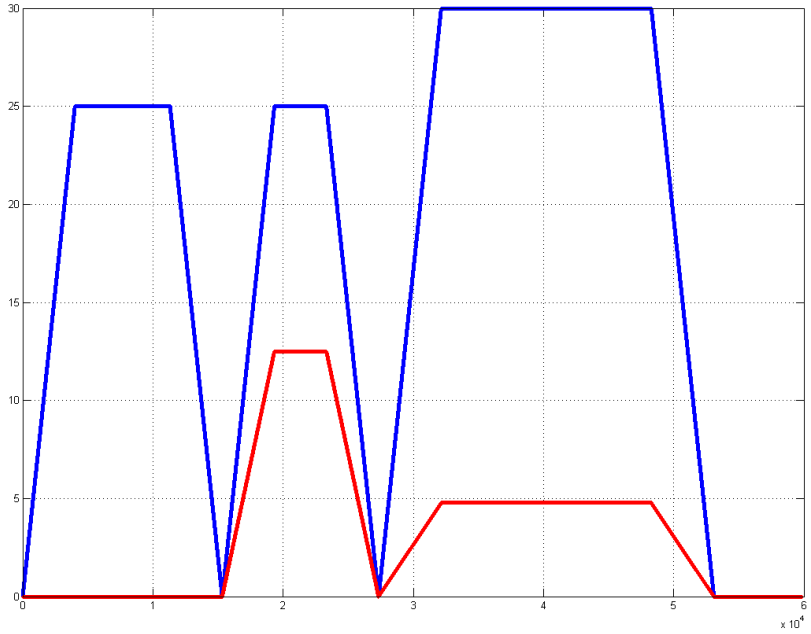

Fig 4-16 Follow motion velocity curve of interpolation buffer

The blue line is the resultant velocity of interpolation movement; the red line is the resultant velocity of the point to point moving axis. The velocity of following axis changes along with the resultant velocity of interpolation.When using the point to point movement function of the interpolation module, pay attention to the following matters:

a) GT_BufGear() command must place right before the interpolation segment needs to be follow, do not insert any other kind commands, more than one GT_BufGear() command can be executed for multi-axis following movement.

b) When coordinate system paused, as the resultant interpolation velocity decelerates to 0, the follow axis would decelerate to 0, too. If wants to resume the follow axis's following movement as the coordinate system resumes, do not call the GT_Stop() to stop the follow axis, otherwise the follow axis cannot finish the last follow movement when restart the interpolation movement.

4.2 PVT Motion Mode

4.2.1 Commands summary

Tab 4-3 Summary of PVT mode commands

Commands	Description
GT_PrfPvt	Set the axis as PVT mode
GT_SetPvtLoop	Set the number of loops
GT_GetPvtLoop	Get the number of loops
GT_PvtTable	Transfer data to the table in PVT description mode
GT_PvtTableComplete	Transfer data to the table in Complete description mode
GT_PvtTablePercent	Transfer data to the table in Percent description mode
GT_PvtPercentCalculate	Calculate the velocity in Percent description mode

GT_PvtTableContinuous	Transfer data to the table in Continuous description mode
GT_PvtContinuousCalculate	Calculate the time in Continuous description mode
GT_PvtTableSelect	Select the table
GT_PvtStart	Start Pvt motion
GT_PvtStatus	Read status

(1) GT_PrfPvt

Tab 4-4 Set the axis as PVT mode

GT_PrfPvt(profile)	
Profile:INT	Profile No.

(2) GT_SetPvtLoop

Tab 4-5 Set the number of loops

GT_SetPvtLoop(profile, loop)	
Profile:INT	Profile No.
Loop:DINT	Number of loops 0 denotes infinite loop

(3) GT_GetPvtLoop

Tab 4-6 Get the number of loops

GT_GetPvtLoop(profile, pLoopCount, pLoop)	
Profile:INT	Profile No.
PLoopCount: POINTER TO DINT	Get the number of loops executed
PLoop: POINTER TO DINT	Get the total number of loops

(4) GT_PvtTable

Transfer data to the table in PVT description mode

Tab 4-7

GT_PvtTable(tableId, count, pTime, pPos, pVel)	
TableId:INT	Table ID
Count:DINT	Number of data points. 1024 memory blocks each table, and 1 memory block for 1 data point.
PTime:POINTER TO LREAL	Time array of data points, unit: ms, length of array: count
PPos:POINTER TO LREAL	Position array of data points, unit: pulse, length of array: count
PVel:POINTER TO LREAL	Velocity array of data points, unit: pulse/ms, length of array: count

(5) GT_PvtTableComplete

Transfer data to the table in Complete description mode

Tab 4-8

GT_PvtTableComplete(tableId,count, pTime, pPos, pA, pB, pC, velBegin, velEnd)	
TableId:INT	Table ID
Count:DINT	Number of data points. 1024 memory blocks each table, and 1 memory

	block for 1 data point.
PTime:POINTER TO LREAL	Time array of data points, unit: ms, length of array: count
PPos:POINTER TO LREAL	Position array of data points, unit: pulse, length of array: count
pA、pB、pc:POINTER TO LREAL	Working array, for internal use, length of array: count Users need not assign for the array
velBegin:LREAL	Start velocity, unit: pulse/ms
velEnd:LREAL	Terminal velocity, unit: pulse/ms

(6) GT_PvtTablePercent

Transfer data to the table in percent description mode

Tab 4-9

GT_PvtTablePercent(tableId, count, pTime, pPos, pPercent, velBegin)	
TableId:INT	Table ID
Count:DINT	Number of data points. 1024 memory blocks each table, and 1-3 memory blocks for 1 data point.
pTime:POINTER TO LREAL	Time array of data points, unit: ms, length of array: count
pPos:POINTER TO LREAL	Position array of data points, unit: pulse, length of array: count
pPercent:POINTER TO LREAL	Percent array of data points, length of array: count Range of percent: [0,100]
velBegin:LREAL	Start velocity, unit: pulse/ms

(7) GT_PvtPercentCalculate

Calculate the time in Continuous description mode

Tab 4-10

GT_PvtPercentCalculate(count, pTime, pPos, pPercent, velBegin, pVel)	
Count:DINT	Number of data points. This command is used to calculate the time of each data point. The data points are not downloaded to the motion controller.
pTime:POINTER TO LREAL	Time array of data points, unit: ms, length of array: count
pPos:POINTER TO LREAL	Position array of data points, unit: pulse, length of array: count
pPercent:POINTER TO LREAL	Percent array of data points, length of array: count Range of percent: [0,100]
velBegin: LREAL	Start velocity, unit: pulse/ms
pVel:POINTER TO LREAL	Return velocity of each data point, unit: pulse/ms

(8) GT_PvtTableContinuous

Transfer data to the table in Continuous description mode

Tab 4-11

GT_PvtTableContinuous(tableId, count, pPos, pVel, pPercent, pVelMax, pAcc, pDec, timeBegin)	
TableId:INT	Table ID
Count:DINT	Number of data points. 1024 memory blocks each table, and 1-8 memory blocks for 1 data point.

pPos:POINTER TO LREAL	Position array of data points, unit: pulse, length of array: count
pVel:POINTER TO LREAL	Velocity array of data points, unit: pulse/ms, length of array: count
pPercent:POINTER TO LREAL	Percent array of data points, length of array: count Range of percent: [0,100]
pVelMax:POINTER TO LREAL	Max. velocity array of data points, unit: pulse/ms, length of array: count
pAcc:POINTER TO LREAL	Acceleration array of data points, unit: pulse/ms^2, length of array: count
pDec:POINTER TO LREAL	Deceleration array of data points, unit: pulse/ms^2, length of array: count
TimeBegin:LREAL	Start time, unit: ms

(9) GT_PvtContinuousCalculate

Calculate the velocity in Percent description mode

Tab 4-12

GT_PvtContinuousCalculate(count, pPos, pVel, pPercent, pVelMax, pAcc, pDec, pTime)	
Count:DINT	Number of data points. This instruction is used to calculate the velocity of each data point. The data points are not downloaded to the motion controller.
pPos:POINTER TO LREAL	Position array of data points, unit: pulse, length of array: count
pVel:POINTER TO LREAL	Time array of data points, unit: ms, length of array: count
pPercent:POINTER TO LREAL	Percent array of data points, length of array: count Range of percent: [0,100]
pVelMax:POINTER TO LREAL	Max. velocity array of data points, unit: pulse/ms, length of array: count
pAcc:POINTER TO LREAL	Acceleration array of data points, unit: pulse/ms^2, length of array: count
pDec:POINTER TO LREAL	Deceleration array of data points, unit: pulse/ms^2, length of array: count
pTime:POINTER TO LREAL	Return time of each data point, unit: ms

(10) GT_PvtTableSelect

Tab 4-13 Description fo select table command

GT_PvtTableSelect(profile, tableId)	
Profile:INT	Profile No.
TableId:INT	Table ID PVT mode provides 32 tables, range: [1,32]

(11) GT_PvtStart

Tab 4-14 Description fo start motion command

GT_PvtStart(mask)	
Mask:DINT	“mask” represents the axis No., which will be started in Pvt mode by bit. bit0 corresponds to 1st axis, bit1 corresponds 2st axis,…… When bitX is 1, it means that the controller will start the corresponding axis Before starting motion, GT_PvtTableSelect can be called to select a table.Default is table1 if no tables are selected.If the table is null, motion start fails.

(12) GT_PvtStatus

Tab 4-15 Description fo start motion command

GT_PvtStatus(profile, pTableId, pTime, count)	
Profile:INT	Profile No.
PTableId:POINTER TO INT	ID of the table in use
pTime:POINTER TO LREAL	Motion time of the axis, unit: ms
Count:INT	Number of axis read

4.2.2 Highlights

M otion patterns are described by parameters of data points such as position, velocity and time in the PVT mode.Position, velocity and time satisfy the following equFtuations.

$$p = at^3 + bt^2 + ct + d$$

$$v = \frac{dp}{dt} = 3at^2 + 2bt + c$$

Given the parameters of position, velocity and time of two consecutive data points, the following equations can be established.

$$\begin{cases} at_1^3 + bt_1^2 + ct_1 + d = p_1 \\ 3at_1^2 + 2bt_1 + c = v_1 \\ at_2^3 + bt_2^2 + ct_2 + d = p_2 \\ 3at_2^2 + 2bt_2 + c = v_2 \end{cases}$$

a, b, c and d can be obtained by solving these equations, thus the motion pattern of the two consecutive data points can be determined.

The motion controller provides 32 tables to store data points. Each table has 1024 memory blocks. The tables and the axes are independent, and one table can serve for more than one axis.

User can transfer data to the tables by calling GT_PvtTable(), GT_PvtTableComplete(), GT_PvtTablePercent() or GT_PvtTable(). These instructions will delete the original data in the table, so all data should be transferred at one time. If the axis using the table is in motion, the table should not be update.

GT_PvtTableSelect() is for table selection. Tables can be switched in motion status, but the switch is not executed immediately. Only the data in the specified table is finished, it will switch to the new table.

User should start motion by calling GT_PvtStart(). After the motion starts, time of all axes is set to 0. The motion starts immediately if the time of the first data point is 0, otherwise the motion starts with time delay and the delay time is the time of the first data point.

The table can be executed repeatedly. Set the number of loop by calling GT_SetPvtLoop(), and 0 denotes infinite loop. After table be executed, the time is initialized to the first data point instead of 0.

Suppose there are four data points as illustrated in Tab 4-16, and PVT mode is used to describe the motion pattern.

Tab 4-16

Data points	Time (ms)	Position (pulse)	Velocity (pulse/ms)
P1	1,000	0	0
P2	2,000	5,000	10
P3	3,000	15,000	10
P4	4,000	20,000	0

1. Call GT_PrfPvt to switch to PVT mode
2. Call GT_PvtTable to transfer the 4 data points to the table
3. Call GT_SetPvtLoop to set loop
4. Call GT_PvtStart to start motion

Since the time of P1 is 1000 ms, the motion starts after calling GT_PvtStart ()with 1000 ms time delay. Return to P1 after it moves to P4 for the motion is in loop. The velocity curve is illustrated in Fig 4-17.

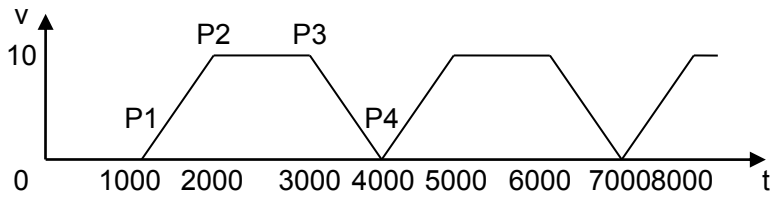

Fig 4-17 Executing table in loop

There are 4 modes to describe motion patterns in the PVT mode, PVT, Complete, Percent and Continuous. The following sections describe these modes in detail.

(1) PVT description mode

PVT description mode defines position, velocity and time of data points directly. In the interval of two consecutive data points, the motion controller uses 3-order polynomial to interpolate positions, and uses 2-order polynomial to interpolate velocity. Therefore the motion pattern is determined after the parameters of position, velocity and time are given.

There are 4 sets of data points are described in PVT description mode in Tab 4-17.

Tab 4-17

Data points	Time (ms)	Position (pulse)	Velocity (pulse/ms)
P1	0	0	0
P2	1,000	5,000	10
P3	2,000	15,000	10
P4	3,000	20,000	0
Data points	**Time (ms)**	**Position (pulse)**	**Velocity (pulse/ms)**
P1	0	0	0
P2	1,000	5,000	9
P3	2,000	15,000	9

P4	3,000	20,000	0
Data points	**Time (ms)**	**Position (pulse)**	**Velocity (pulse/ms)**
P1	0	0	0
P2	1,000	5,000	7.5
P3	2,333	15,000	7.5
P4	3,333	20,000	0
Data points	**Time (ms)**	**Position (pulse)**	**Velocity (pulse/ms)**
P1	0	0	0
P2	750	1,667	6.6669
P3	2,250	18,333	6.6669
P4	3,000	20,000	0

The motion patterns of these 4 sets of data points are illustrated in Fig 4-18.

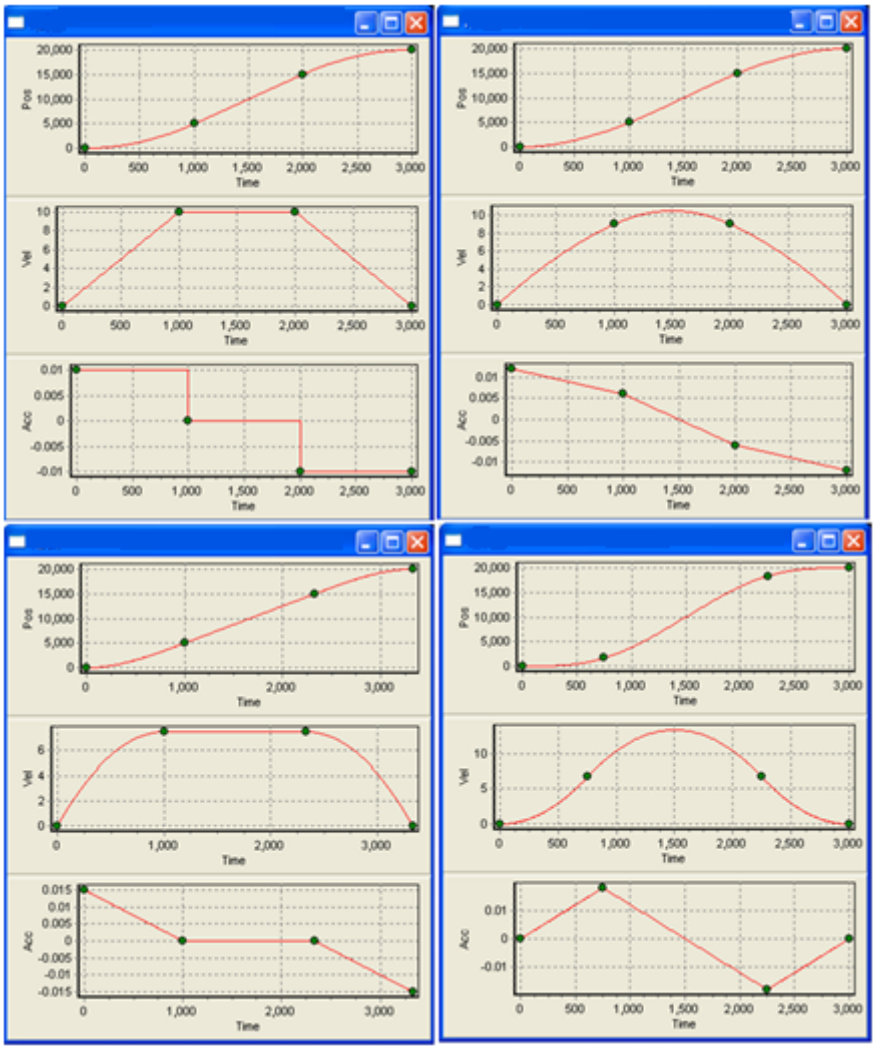

Fig 4-18 Proper data point's parameters in PVT description mode

It can be found that the PVT description mode is very flexible. Given the data point parameters of position, velocity and time the motion patterns can be obtained.

It should be noticed that the parameters of data points should be designed carefully; otherwise ideal motion pattern cannot be obtained.

For example, improper parameter setting of 2 sets of data points in Tab 4-18, becauses the velocity curve obtained not smooth.

Tab 4-18

Data points	Time (ms)	Position (pulse)	Velocity (pulse/ms)
P1	0	0	0
P2	1,000	5,000	15
P3	2,000	15,000	15
P4	3,000	20,000	0
Data points	**Time (ms)**	**Position (pulse)**	**Velocity (pulse/ms)**
P1	0	0	0
P2	1,000	5,000	5
P3	2,000	15,000	5
P4	3,000	20,000	0

Motion patterns of the 2 sets of data points are illustrated in Fig 4-19.

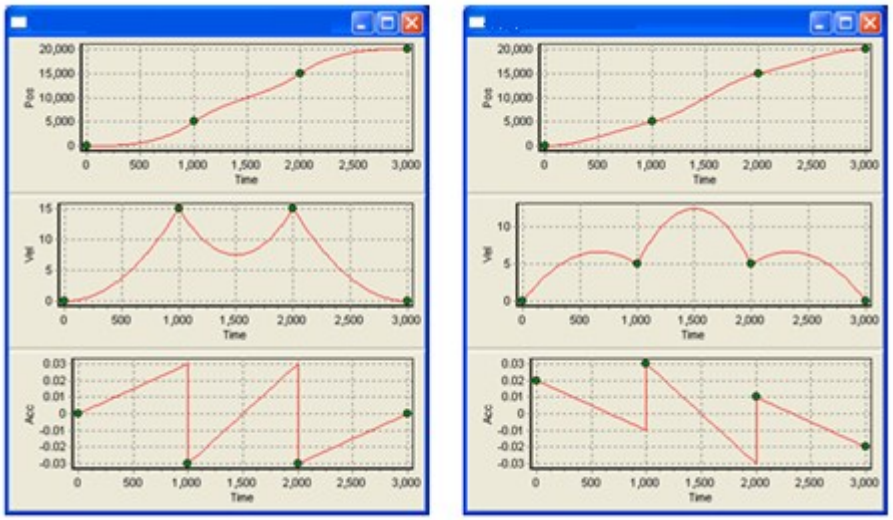

Fig 4-19 Improper data points parameter in PVT description mode

(2) Complete description mode

Complete description mode defines position and time of data points, start velocity and terminal velocity. Complete description mode only defines start velocity and terminal velocity. The motion controller calculates velocity of intermediate points according to the position and time of data points to ensure the continuity of velocity and acceleration. As in Tab 4-19, the set of data points is described in Complete mode, and the velocity curve obtained is smooth.

Tab 4-19

Data points	Time (ms)	Position (pulse)	Velocity (pulse/ms)
P1	0	0	0
P2	1,000	5,000	-
P3	2,000	15,000	-
P4	3,000	20,000	0

The motion patterns of this set of data points are illustrated in Fig 4-20.

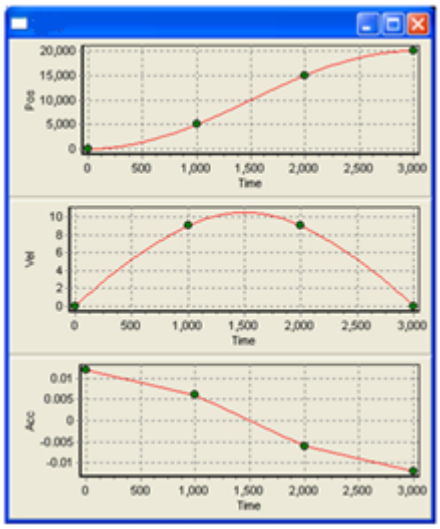

Fig 4-20 Complete description mode

Complete mode is always used to describe smooth velocity curve, such as trigonometric functions.

Suppose the relation of position and time is determined by P=50000*sin2 (π/2000*t). Select 5 time points in a function period [0,2000] to calculate their positions as illustrated in Tab 4-20.

Tab 4-20 Complete description mode

Data points	Time (ms)	Position (pulse)	Velocity (pulse/ms)
P1	0	0	0
P2	500	25,000	-
P3	1,000	50,000	-
P4	1,500	25,000	-
P5	2,000	0	0

The motion patterns of this set of data points are illustrated in Fig 4-21.

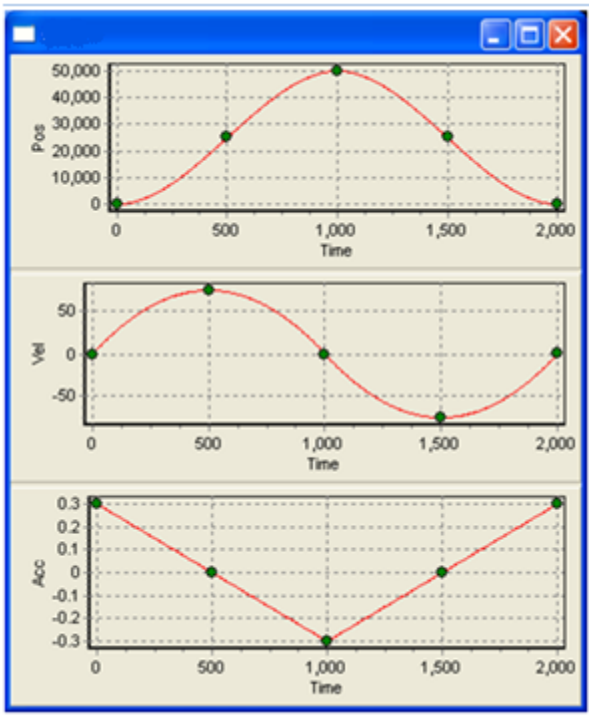

Fig 4-21 Trigonometric functions described in Complete mode

Increasing the number of data points will decrease the approximation error to P=50000*sin2 (π/2000*t). Fig 4-22 shows the position error when the number of data points is 5, 10 and 50.

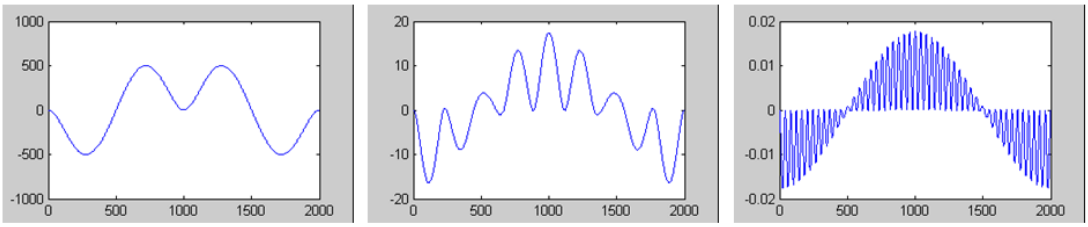

Fig 4-22 Position error when the number of data points is 5, 10 and 50

(3) Percent description mode

Percent description mode defines position, time and percent of each data point, and start velocity.

Percent description mode can define distance, velocity and time of accelerating segment, constant speed segments and decelerating segment precisely.

Percent description mode supposes the velocity variation between two consecutive points is linear. The following equation is used to calculate the velocity of data points with the start velocity and the parameters of position and time.

$$v_{i+1} = \frac{2(p_{i+1} - p_i)}{t_{i+1} - t_i} - v_i$$

Therefore, the velocity of each data point is determined after the position and time of data points are specified. The parameter of percent can adjust the smooth degree of a velocity curve. Percent of a data point denotes the percentage of acceleration variation time to velocity variation time between two

consecutive data points. Fig 4-23 is used for illustration.

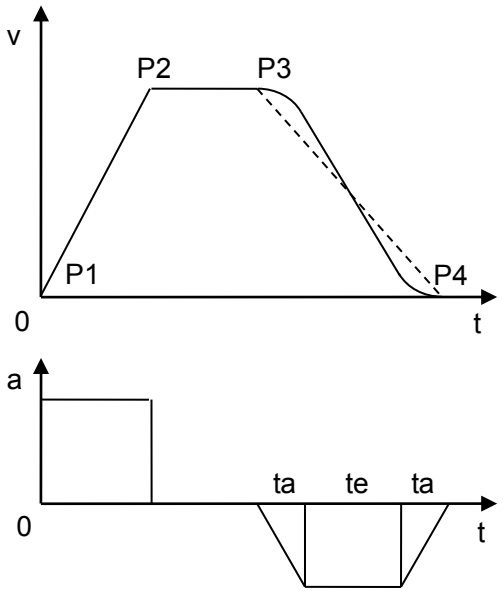

Fig 4-23 Definition of percent

Acceleration between P1 and P2 is constant, so the percent of P1 is 0.

Acceleration between P2 and P3 is constant, so the percent of P2 is 0.

Acceleration variation time between P3 and P4 is 2ta, and motion time is 2ta+te, so the percent of P3 is 2ta/(2ta+te)*100%.

Tuning percent will not affect the parameters of position and time of data points. In Fig 4-23 the velocity curve between P3 and P4 is a dashed line if percent of P3 is 0; otherwise the velocity curve is a solid line.

The set of data points is described in Percent mode in Tab.

Tab 4-21

Data points	Time (ms)	Position (pulse)	Percent	Velocity (pulse/ms)
P1	0	0	60	0
P2	1,000	5,000	0	-
P3	2,000	15,000	100	-
P4	3,000	20,000	0	-

The motion patterns of this set of data points are illustrated in Fig 4-24.

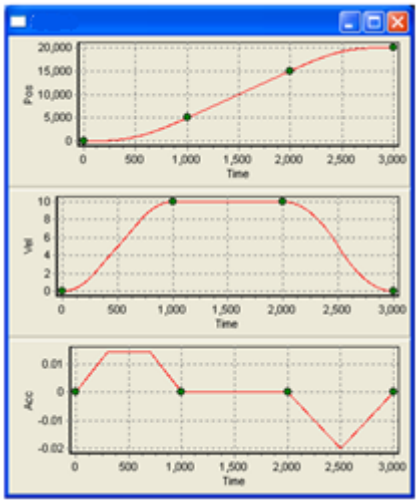

Fig 4-24 Percent description mode

(4) Continuous description mode

Continuous description mode defines position, velocity, maximum velocity, acceleration, deceleration and percent of data points. Time of data points needs not be defined. According to these parameters, the motion controller separates the segment between two consecutive points into accelerating segment, constant speed segments and deceleration segment.

Maximum acceleration of Pi is the upper limit of velocity between Pi and Pi+1.

Acceleration of Pi is the acceleration in the accelerating segment between Pi to Pi+1.

Deceleration of Pi is the deceleration in the decelerating segment between Pi to Pi+1.

Percent of Pi is the percentage of acceleration variation time to velocity variation time in the accelerating/decelerating segment between Pi to Pi+1.

The number of segments separated between two consecutive data points has relation with the parameters of these two points. Fig 4-25 shows some examples.

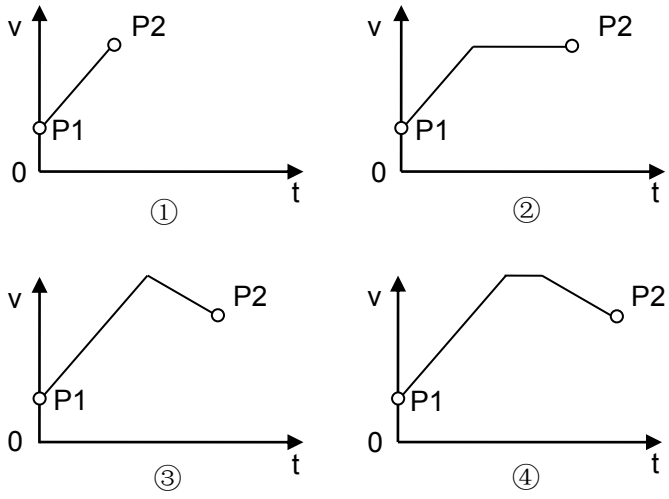

Fig 4-25 Continuous description mode

As in Tab 4-22 the two sets of data points are described in Continuous mode

Tab 4-22

Data points	Position (pulse)	Velocity (pulse/ms)	Max Vel.	Acceleration	Deceleration	Percent
P1	0	0	10	0.01	0.01	60
P2	20,000	0	10	0.01	0.01	0
Data points	**Position (pulse)**	**Velocity (pulse/ms)**	**Max Vel.**	**Acceleration**	**Deceleration**	**Percent**
P1	0	0	10	0.01	0.01	60
P2	19,800	2	2	0.02	0.02	0
P3	21,800	0	2	0.02	0.02	0

The motion patterns of these two sets of data points are illustrated in Fig 4-26.

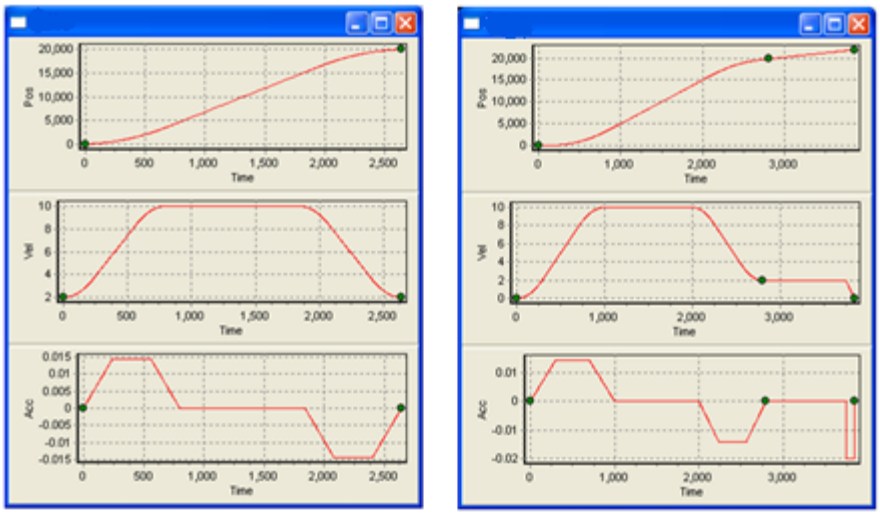

Fig 4-26 Continuous description mode

4.2.3 Examples

(1) PVT description mode

The velocity curve consists of 5 segments with takeoff velocity.

In the 1st segment, velocity increases and acceleration is constant.

In the 2nd segment, velocity increases and acceleration decreases.

In the 3rd segment, velocity is constant and acceleration is 0.

In the 4th segment, velocity decreases and acceleration increases.

In the 5th segment, velocity decreases and acceleration is constant.

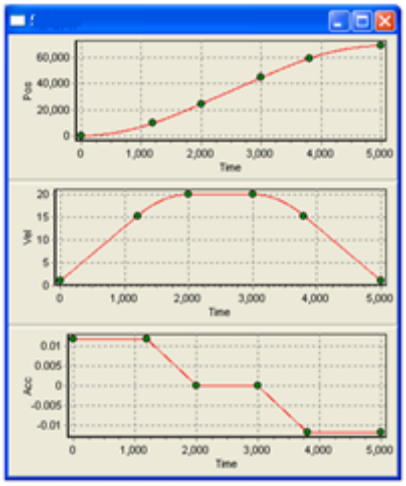

Fig 4-27 Velocity curve of PVT example program

A set of data points that satisfy the above condition is as Tab 4-23.

Tab 4-23

Data points	Time (ms)	Position (pulse)	Velocity (pulse/ms)
P1	0	0	1
P2	1,200	9,750	15.25
P3	2,000	24,483	20
P4	3,000	44,483	20
P5	3,800	59,216	15.25
P6	5,000	68,966	1

```
VAR_GLOBAL CONSTANT
AXIS:INT:=1
TABLE:INT:=1
END_VAR

PROGRAM MAIN
VAR
    Rtn:INT;
    Mask:DINT;
    (*Parameter of data points in X-axis *)
    AlTime:ARRAY[0..5] OF LREAL:=0,1200,2000,3000,3800,5000;
    Pos:ARRAY[0..5] OF LREAL:=0,9750,24483,44483,59216,68966;
    Vel:ARRAY[0..5] OF LREAL:=1,15.25,20,20,15.25,1;
```

```
    prfVel,prfPos,t:LREAL;
    tableId:INT;
    First:BOOL:=TRUE;
END_VAR
-----------------------------------------------------------------------------------------------------------
IF First THEN
    rtn := GT_AxisOn(AXIS);

    (*Set to PVT mode*)
    rtn := GT_PrfPvt(AXIS);

    (*Transfer data*)
    rtn := GT_PvtTable(TABLE,6,ADR(alTime[0]),ADR(pos[0]),ADR(vel[0]));

    (*Select data table *)
    rtn := GT_PvtTableSelect(AXIS,TABLE);

    mask := SHL(1,(AXIS-1));
    rtn := GT_PvtStart(mask);

    First:=FALSE;
END_IF

(*Read data table and motion time *)
rtn := GT_PvtStatus(AXIS,ADR(tableId),ADR(t),1);

(*Read profile velocity *)
rtn = GT_GetPrfVel(AXIS,ADR(prfVel),1,0);

(*Read profile position *)
rtn = GT_GetPrfPos(AXIS,ADR(prfPos),1,0);
```

(2) Complete description mode

Suppose the relation of position and time is determined by P=40000*sin2 (π/2000*t). The motion should be cyclic and the amplitude increases 50% when pressing the button of A and decreases 50% when pressing the button of B.

```
VAR_GLOBAL CONSTANT
   AXIS:INT:=1
   TABLE1:INT:=1
   TABLE2:INT:=2
   PI:LREAL:=3.1415926
END_VAR
```

Fig 4-28 Velocity curve of Complete example program

```
FUNCTION Calculate : INT
VAR_INPUT
   Amplitude:LREAL;
   n:DINT;
   pTime:POINTER TO LREAL;
   pPos:POINTER TO LREAL;
END_VAR

VAR
i:DINT;
END_VAR
-----------------------------------------------------------------------------------------------------------
FOR i:=0 TO n-1 BY 1 DO
      pPos[i] := amplitude*sin(PI/2000*pTime[i])*sin(PI/2000*pTime[i]);
END_FOR



PROGRAM MAIN
```

```
VAR
    Rtn:INT;
    Mask:DINT;
    (*Parameter of data points in X-axis *)
    AlTime:ARRAY[0..4] OF LREAL:= 0,500,1000,1500,2000;
    Pos:ARRAY[0..4] OF LREAL;
    a,b,c: ARRAY[0..4] OF LREAL;
    prfVel,prfPos,t:LREAL;
    tableId:INT;
    amplitude:LREAL:=40000;
    table:INT:= TABLE1;
    key:STRING(1);
    First:BOOL:=TRUE;
END_VAR
-----------------------------------------------------------------------------------------------------------
IF First THEN
    rtn := GT_AxisOn(AXIS);

    (*Set to PVT mode *)
    rtn := GT_PrfPvt(AXIS);

    Calculate(amplitude,5,ADR(alTime[0]),ADR(pos[0]));

    (*Transfer data *)
rtn:=GT_PvtTableComplete(table,5,ADR(alTime[0]),ADR(pos[0]),ADR(a[0]),ADR(b[0]),ADR(c[0]),0,
0);

    (*Select data table *)
    rtn := GT_PvtTableSelect(AXIS,table);

    (*Set loop *)
```

```
        rtn := GT_SetPvtLoop(AXIS,0);
        mask := SHL(1,(AXIS-1));
        rtn := GT_PvtStart(mask);
        First:=FALSE;
    END_IF

    (*Read data table and motion time *)
    rtn := GT_PvtStatus(AXIS,ADR(tableId),ADR(t),1);

    (*Read profile velocity *)
    rtn = GT_GetPrfVel(AXIS,ADR(prfVel),1,0);

    (*Read profile position *)
    rtn = GT_GetPrfPos(AXIS,ADR(prfPos),1,0);

    If key<>'' THEN
        IF ( 'A' = key ) THEN
            amplitude := amplitude *1.5;
        END_IF
        IF ( 'B' = key) THEN
            amplitude := amplitude *0.5;
        END_IF
    IF ( 'A' = key ) OR ( 'B' = key ) THEN
            Calculate(amplitude,5,ADR(alTime[0]),ADR(pos[0]));
            table := TABLE1 + TABLE2 - tableId;

        (*Transfer data *)
        rtn := GT_PvtTableComplete(table, 5, ADR(alTime[0]), ADR(pos[0]), ADR(a[0]), ADR(b[0]),
ADR(c[0]), 0, 0);

            (*Select data table *)
```

```
            rtn := GT_PvtTableSelect(AXIS,table);
        END_IF

        IF ( 'Q' = key) THEN
            (*stop axis motion*)
            mask := SHL(1,(AXIS-1));
            GT_Stop(mask,0);
        END_IF
    END_IF
```

(3) Percent description mode

Reciprocating motion executes in X-axis and positive feeding executes in Y-axis. Feeding starts in Y-axis when it accelerates/decelerates in X-axis. Y-axis keeps motionless when X-axis moves in even pace.

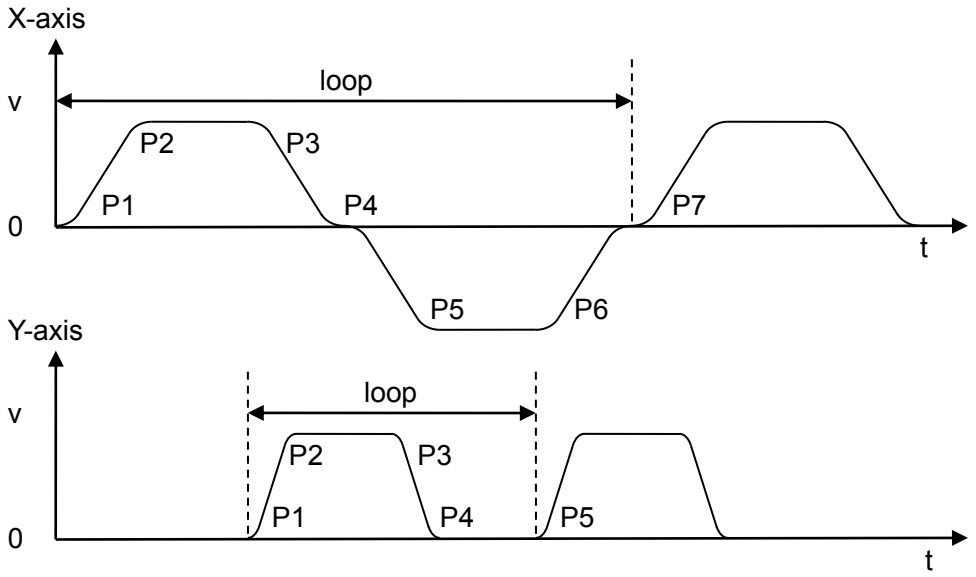

Fig 4-29 Motion patterns of X-axis and Y-axis

Select 7 data points in X-axis, and set to loop mode. Parameters of data points are as Tab 4-24 and Tab 4-25.

Tab 4-24

Data points	Time (ms)	Position (pulse)	Percent	Velocity (pulse/ms)
P1	0	0	60	0
P2	1,000	5,000	0	-
P3	2,000	15,000	60	-
P4	3,000	20,000	60	-
P5	4,000	15,000	0	-
P6	5,000	5,000	60	-
P7	6,000	0	0	-

Velocity of each data point can be calculated according to the parameters of data points in X-axis. Percent has no effect on the calculation of velocity.

Tab 4-25

Data points	Time (ms)	Position (pulse)	Velocity (pulse/ms)
P1	0	0	0
P2	1,000	5,000	2(5000-0)/(1000-0)-0=10
P3	2,000	15,000	2(15000-5000)/(2000-1000)-10=10
P4	3,000	20,000	2(20000-15000)/(3000-2000)-10=0
P5	4,000	15,000	2(15000-20000)/(4000-3000)-0=-10
P6	5,000	5,000	2(5000-15000)/(5000-4000)-(-10)=-10
P7	6,000	0	2(0-5000)/(6000-5000)-(-10)=0

Select 5 data points in Y-axis, and set to loop mode. After X-axis moves to P3, start Y-axis. Thus time of the first data point is set to be 2000ms. Y-axis returns to P1 for loop after it reaches P5. Parameters of data points are as Tab 4-26 and Tab 4-27.

Tab 4-26

Data points	Time (ms)	Position (pulse)	Percent	Velocity (pulse/ms)
P1	2,000	0	60	0
P2	2,500	2,500	0	-
P3	3,500	12,500	60	-
P4	4,000	15,000	0	-
P5	5,000	15,000	0	-

Velocity of each data point can be calculated according to the parameters of data points in Y-axis. Percent has no effect on the calculation of velocity.

Tab 4-27

Data points	Time (ms)	Position (pulse)	Velocity (pulse/ms)
P1	2,000	0	0
P2	2,500	2,500	2(2500-0)/(2500-2000)-0=10
P3	3,500	12,500	2(12500-2500)/(3500-2500)-10=10
P4	4,000	15,000	2(15000-12500)/(4000-3500)-10=0
P5	5,000	15,000	2(15000-15000)/(5000-4000)-0=0

When X-axis loops n times, Y-axis loops 2n-1 times. Fig 4-30 shows the XY position after X-axis loops 2 times and Y-axis loops 3 times. Horizontal axis is the position of X-axis, and vertical axis is the position of Y-axis. The yellow line is the real trajectory.

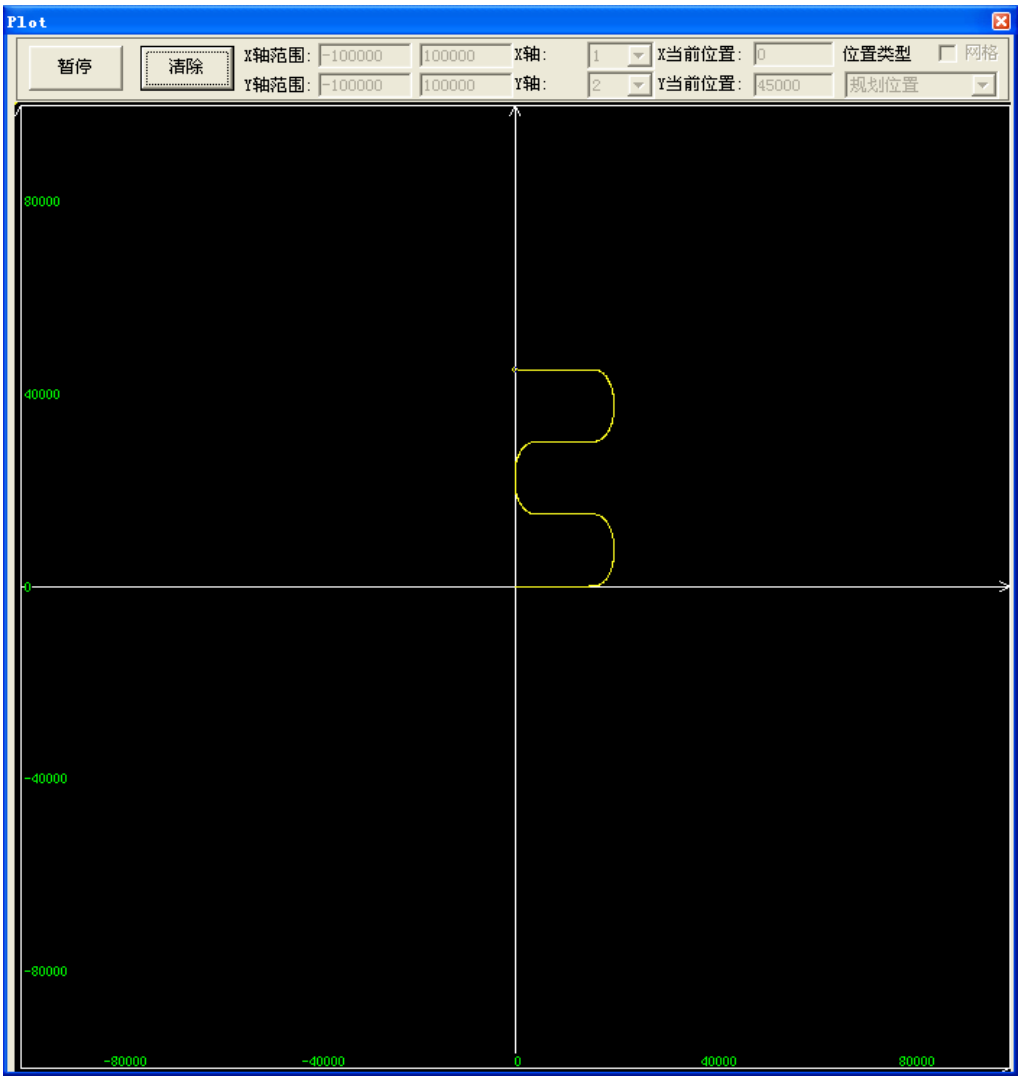

Fig 4-30 X-Y position in Percent description mode

```
VAR_GLOBAL CONSTANT
   AXIS_X:INT:=1
   AXIS_Y:INT:=2

   TABLE_X:INT:=1
   TABLE_Y:INT:=2
   LOOP_COUNT:INT:=2;
END_VAR

PROGRAM MAIN
VAR
      Rtn:INT;
```

```
        Mask:DINT;
        (*Parameters of data points in X-axis *)
        Time_x:ARRAY[0..6] OF LREAL:= 0,1000,2000,3000,4000,5000,6000;
        pos_x:ARRAY[0..6] OF LREAL:= 0,5000,15000,20000,15000,5000,0;
        percent_x:ARRAY[0..6] OF LREAL:= 60,0,60,60,0,60,0;
        Time_y:ARRAY[0..4] OF LREAL:= 2000,2500,3500,4000,5000;
        pos_y:ARRAY[0..4] OF LREAL:= 0,2500,12500,15000,15000;
        percent_y:ARRAY[0..4] OF LREAL:= 60,0,60,0,0;

        prfVel, prfPos, alTime:ARRAY[0..1] OF LREAL;
        tableId:INT;
        First:BOOL:=TRUE;
    END_VAR
    ------------------------------------------------------------------------------------------------------------
    IF First THEN

        rtn := GT_AxisOn(AXIS_X);
        rtn := GT_AxisOn(AXIS_Y);

        (*Set X-axis as PVT mode *)
        rtn := GT_PrfPvt(AXIS_X);

        (*Set Y-axis as PVT mode *)
        rtn := GT_PrfPvt(AXIS_Y);

        (*Transfer data to X-axis data table *)
    rtn := GT_PvtTablePercent(TABLE_X,7,ADR(time_x[0]),ADR(pos_x[0]),ADR(percent_x[0]),0);

        (*Transfer data to Y-axis data table *)
    rtn := GT_PvtTablePercent(TABLE_Y,5,ADR(time_y[0]),ADR(pos_y[0]),ADR(percent_y[0]),0);
```

```
        (*Select data table TABLE_X for X-axis *)
        rtn := GT_PvtTableSelect(AXIS_X,TABLE_X);

        (*Select data table TABLE_Y for Y-axis *)
        rtn := GT_PvtTableSelect(AXIS_Y,TABLE_Y);

        (*Set cycle number *)
        rtn := GT_SetPvtLoop(AXIS_X,LOOP_COUNT);

        (*Set cycle number *)
        rtn := GT_SetPvtLoop(AXIS_Y,2*LOOP_COUNT-1);

        (*start X axis and Y axis simultaneously *)
        (*Since time of the first data point in Y-axis is 2000ms*)
    (*Y-axis starts 20000ms after X-axis starts *)
        mask := SHL(1,(AXIS_X-1));
        mask := mask OR SHL(1,(AXIS_Y-1));
        rtn := GT_PvtStart(mask);
        First:=FALSE;
    END_IF

    (*Read data tables and motion time *)
    rtn := GT_PvtStatus(AXIS_X,ADR(tableId[0]),ADR(alTime[0]),1);
    rtn := GT_PvtStatus(AXIS_Y,ADR(tableId[1]),ADR(alTime[1]),1);

    (*Read profile velocity *)
    rtn := GT_GetPrfVel(AXIS_X,ADR(prfVel[0]),1,0);
    rtn := GT_GetPrfVel(AXIS_Y,ADR(prfVel[1]),1,0);

    (*Read profile position *)
    rtn := GT_GetPrfPos(AXIS_X,ADR(prfPos[0]),1,0);
```

```
rtn := GT_GetPrfPos(AXIS_Y,ADR(prfPos[1]),1,0);
```

(4) Continuous description mode

X-axis moves from A to B, and Y-axis moves from C to D. When X-axis reaches B, Y-axis must reaches D.

Motion time of X-axis t_x and motion time of Y-axis t_y are calculated by calling GT_PvtContinuousCalculate(). This instruction does not transfer data points to the motion controller. If $t_x>t_y$, Y-axis starts with a time delay of t_x-t_y. If $t_x<t_y$, X-axis starts with a time delay t_y-t_x.

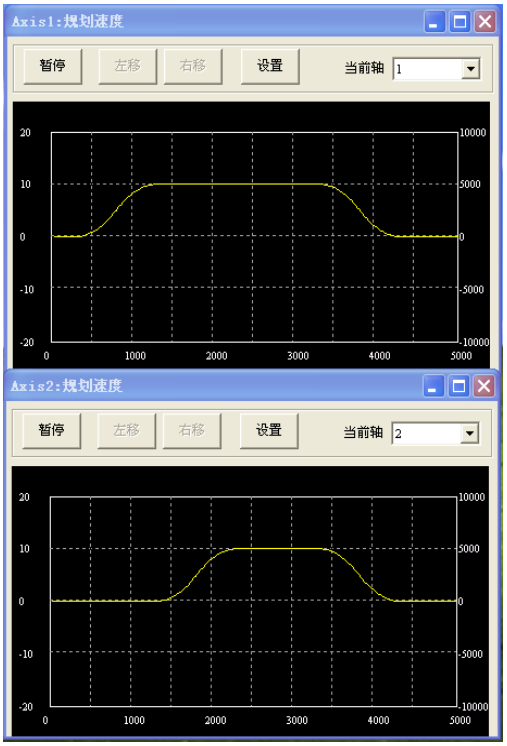

Fig 4-31 Velocity curves in X-axis and Y-axis

```
VAR_GLOBAL CONSTANT
   AXIS_X:INT:=1
   AXIS_Y:INT:=2

   TABLE_X:INT:=1
   TABLE_Y:INT:=2
END_VAR
--------------------------------------------------------------------------------
PROGRAM MAIN
VAR
```

```
        Rtn:INT;
        Mask:DINT;
        (*parameters of data point in X axis *)
        pos_x:ARRAY[0..1] OF LREAL:= 0,30000;
        vel_x:ARRAY[0..1] OF LREAL:= 0,0;
        velMax_x:ARRAY[0..1] OF LREAL:= 10,10;
        percent_x:ARRAY[0..6] OF LREAL:=100,100;
        acc_x:ARRAY[0..1] OF LREAL:= 0.01,0.01;
        dec_x:ARRAY[0..1] OF LREAL:= 0.01,0.01;
        time_x:ARRAY[0..1] OF LREAL;
    timeBegin_x:LREAL;

        (*parameters of data point in Y axis *)
        pos_y:ARRAY[0..1] OF LREAL:= 0,20000;
        vel_y:ARRAY[0..1] OF LREAL:= 0,0;
        velMax_y:ARRAY[0..1] OF LREAL:= 10,10;
        percent_y:ARRAY[0..6] OF LREAL:=100,100;
        acc_y:ARRAY[0..1] OF LREAL:= 0.01,0.01;
        dec_y:ARRAY[0..1] OF LREAL:= 0.01,0.01;
        time_y:ARRAY[0..1] OF LREAL;
    timeBegin_y:LREAL;

        prfVel, prfPos, alTime:ARRAY[0..1] OF LREAL;
        tableId: ARRAY[0..1] OF INT;
        First:BOOL:=TRUE;
    END_VAR
    ----------------------------------------------------------------------------------------------------
    IF First THEN
        rtn := GT_AxisOn(AXIS_X);

        rtn := GT_AxisOn(AXIS_Y);
```

```
(*set X axis as PVT mode *)
rtn := GT_PrfPvt(AXIS_X);

(*set Y axis as PVT mode *)
rtn := GT_PrfPvt(AXIS_Y);

(*Calculate motion time of  X axis *)
rtn :=GT_PvtContinuousCalculate(2,ADR(pos_x[0]),ADR(vel_x[0]),ADR(percent_x[0]),ADR(velMax_x[0]), ADR(acc_x[0]), ADR(dec_x[0]), ADR(time_x[0]));

(*Calculate motion time of  Y axis *)
rtn := GT_PvtContinuousCalculate(2, ADR(pos_y[0]), ADR(vel_y[0]), ADR(percent_y[0]), ADR(velMax_y[0]), ADR(acc_y[0]), ADR(dec_y[0]), ADR(time_y[0]));

(*Calculate start time delay *)
IF time_x[1] < time_y[1] THEN
    timeBegin_x := time_y[1] - time_x[1];
    timeBegin_y := 0;
ELSE
    timeBegin_x := 0;
    timeBegin_y := time_x[1] - time_y[1];
END_IF

(*Transfer data points in X-axis *)
rtn := GT_PvtTableContinuous(TABLE_X,2, ADR(pos_x[0]), ADR(vel_x[0]), ADR(percent_x[0]), ADR(velMax_x[0]), ADR(acc_x[0]), ADR(dec_x[0]),timeBegin_x);

(*Transfer data points in Y-axis *)
rtn := GT_PvtTableContinuous(TABLE_Y,2, ADR(pos_y[0]), ADR(vel_y[0]), ADR(percent_y[0]), ADR(velMax_y[0]), ADR(acc_y[0]), ADR(dec_y[0]),timeBegin_y);
```

```
        (*Select data table TABLE_X for X-axis *)
        rtn := GT_PvtTableSelect(AXIS_X,TABLE_X);

        (*Select data table TABLE_Y for Y-axis *)
        rtn := GT_PvtTableSelect(AXIS_Y,TABLE_Y);

        (*Start X-axis and Y-axis simultaneously *)
        (*Since time of the first data point in Y-axis is 2000ms *)
    (*Y-axis starts 20000ms after X-axis starts *)
        mask := SHL(1,(AXIS_X-1));
        mask := mask OR SHL(1,(AXIS_Y-1));
        rtn := GT_PvtStart(mask);
        First:=FALSE;
    END_IF

    (*Read data tables and motion time *)
    rtn := GT_PvtStatus(AXIS_X,ADR(tableId[0]),ADR(alTime[0]),1);
    rtn := GT_PvtStatus(AXIS_Y,ADR(tableId[1]),ADR(alTime[1]),1);

    (*Read profile velocity *)
    rtn := GT_GetPrfVel(AXIS_X,ADR(prfVel[0]),1,0);
    rtn := GT_GetPrfVel(AXIS_Y,ADR(prfVel[1]),1,0);

    (*Read profile position *)
    rtn := GT_GetPrfPos(AXIS_X,ADR(prfPos[0]),1,0);
    rtn := GT_GetPrfPos(AXIS_Y,ADR(prfPos[1]),1,0);
    ;
```

Chapter 5 Motion Program

5.1 Introduction

In order to facilitate the expression, we name programs which call windows DLL (Dynamic Linkage Library) and execute in PC as "Application Program", while the program which is downloaded into and execute in motion controller as "Motion Program".

Motion program is compiled with C language and then downloaded into the motion controller for execution. Motion program can be executed independently from PC, so PC can be free from complex logical management work.Firstly, PC can arrange CPU resource to do other tasks; Sencondly, the motion program can use motion controller resource directly without communication with PC, which makes better real time capability.

When needed, PC can send commands to motion controller and receive data from controller, even when the motion program is running in the controller. Watch out, when the axis is controlled by both PC command (Application program) and motion program at the same time, please design the motion logic carefully in case of confusion.

The motion controller allows 32 motion programs running on it at the same time.

The thread scheduling mechanism built-in motion controller ensures that all execution of commands is complete in multi-tread environment.

In multi-tread environment, it’s possible that one thread interrupted by the other thread, please consider the influence carefully between threads during the execution of motion program.

5.2 Programming motion program

5.2.1 Commands summary

Tab 5-1 Summary of Motion program commands

Command	Description
GT_Download	Download motion program to controller.
GT_GetFunId	Read a function ID of motion program.
GT_GetVarId	Read a variable ID of motion program.
GT_Bind	Binding thread, function and data page
GT_RunThread	Start a thread
GT_StopThread	Stop a running thread
GT_PauseThread	Pause the running thread
GT_GetThreadSts	Read a thread’s running status
GT_SetVarValue	Set a variable value of motion program
GT_GetVarValue	Get a variable value of motion program

(1) GT_Download

Tab 5-2 Download motion program to controller

GT_Download(pFileName)	
pFileName:POINTER TO STRING	The file name downloaded into motion controller core.

(2) GT_GetFunId

Tab 5-3 Read a function ID of motion program

GT_GetFunId(pFunName, pFunId)	
pFunName:POINTER TO STRING	Function name of motion program
pFunId:POINTER TO INT	ID of function searched by function name

(3) GT_GetVarId

Tab 5-4 Read a variable ID of motion program

GT_GetVarId(pFunName, pVarName, pVarInfo)	
pFunName:POINTER TO STRING	If pVarName is a global variable, this parameter is Null If pVarName is a local variable, this parameter is the corresponding function name
pVarName:POINTER TO STRING	Variable name of motion program.
pVarInfo:POINTER TO TVarInfo	Variable information searched by function name and variable name.

(4) GT_Bind

Tab 5-5 Binding thread, function and data page

GT_Bind(thread, funId, page)	
thread:INT	Thread number, and its value ranging in [0,31].
funId:INT	Function id which can be checked using GT_GetFunId.
page:INT	data page ID, and its value ranging in [0,31].

(5) GT_RunThread

Tab 5-6 Start a thread

GT_RunThread(thread)	
thread:INT	Thread number, and its value ranging in [0,31].

(6) GT_StopThread

Tab 5-7 Stop a running thread

GT_StopThread(thread)	
thread:INT	Thread number, and its value ranging in [0,31].

(7) GT_PauseThread

Tab 5-8 Pause the running thread

GT_PauseThread(thread)	
thread:INT	Thread number, and its value ranging in [0,31].

(8) GT_GetThreadSts

Tab 5-9 Read a thread's running status

GT_GetThreadSts(thread, pThreadSts)	
thread:INT	Thread number, and its value ranging in [0,31].

pThreadSts:POINTER TO TThreadSts	Read the thread status.
	typedef struct ThreadSts
	{
	short run; // running status
	short error; // command return value
	double result; // function return value
	short line; // current command line number
	} TThreadSts;

(9) GT_SetVarValue

Tab 5-10 Set a variable value of motion program

GT_SetVarValue(page, pVarInfo, pValue, count=1)	
page:INT	Data page ID Global variable: -1 local variable ranging in [0,31]
pVarInfo:POINTER TO TVarInfo	Variable information.
pValue:POINTER TO DINT	Value of variable which will be writen.
count:INT	Number of variable which will be written, ranging in [1,8]

(10) GT_GetVarValue

Tab 5-11 Get a variable value of motion program

GT_GetVarValue(page, pVarInfo, pValue, count=1)	
page:INT	Data page ID, Global variable: -1; local variable range[0,31]
pVarInfo:POINTER TO TVarInfo	Variable information
pValue:POINTER TO DINT	Value of read variable
count:INT	Number of variable which to be read, its value ranging in [1,8]

5.2.2 Highlights

Method of motion program:

1) Programming motion program use a text editor with C language.
2) Compiling Motion program with MCT2008 will generate an object program files (*. Bin) and symbol files (ini).
3) Calling GT_Download to download target file into controller's SDRAM.
4) Calling GT_GetFunId() to get function ID.

5) Calling GT_GetVarId() to get variable ID.

6) Calling GT_Bind() to bind thread, function and data page.

7) Calling GT_SetVarValue() to update local variable and global variable.

8) Calling GT_RunThread() to start a thread.

9) Calling GT_GetThreadSts() to check running status of thread, or calling GT_GetVarValue() to check variable.

Calling GT_Download can download target file (*.bin) into controller's SDRAM. When a new motion program is downloaded, the previous motion program would be overrided.The motion program should be re-downloaded whenever the motion controller is powered.When releasing application program, we should release the target files and symbol files simultaneously, if not the motion program can not be downloaded and executed correctly.

When the motion program was downloaded into controller, the program cannot be executed at once, user must call GT_Bind() to bind thread, function and data page, and then call GT_Bind() to start a thread. Motion controller allowed 32 threads running at the same time, each thread have only one function, but one function can be used by more than one thread at the same time, for example, when multi-axes homing, user can bind more than one thread to one homing function, then start these threads at the same time, which can realize multi-axes homing action. During the running of thread, it's not allowed to bind a new function with this thread until the thread is done.

The local variables of each function have independent data pages. Motion controller provided 32 data pages. When binding thread and function, you must notice the used data pages. One data page can only be arranged to one thread, but one thread can use many data pages. The thread can change the data page during its running.Application can call GT_GetThreadSts to check running status of thread any time.

Application can call GT_SetVarValue to update the value of every variable in motion program any time.

Application can call GT_GetVarValue to get the value of every variable in motion program any time.

5.2.3 Example

(1) Addition in single thread

In programming the motion program for addition task , the global variable "sum" is defined for saving result of addition, and the local variable "begin" is defined for saving the start point of addition, while the local variable "end" means saving end point of addition.. When the addition task is completed, the program would be ended.

```
//-----------------------------------------------
// Addition
// begin
// end
//-----------------------------------------------
```

```
int sum;


int add(int begin,int end)
{
    int i;
    int cc;


    i=begin;
lbl_loop:
    cc = i > end;
    if(cc) goto lbl_end;
    sum = sum + i;
    i = i + 1;
    goto lbl_loop;
lbl_end:
    return sum;
}


// Application program is responsible for compiling,
//downloading, initialling, and running of motion program.
PROGRAM MAIN
VAR
    rtn: INT;
    compile: TCompileInfo;
    filename:STRING:='sum.c';
    downloadfilename:STRING:='sum.bin';
    funname:STRING:='add';
    funid: INT;
    varname:STRING:='sum';
    sum:TVarInfo;
    begin:TVarInfo;
```

```
    end:TVarInfo;
    value:DINT;
    thread:TThreadSts;
    First: BOOL := TRUE;
END_VAR
-------------------------------------------------------------------------------
IF First THEN
    (*Reset the motion controller *)
    rtn:=GT_Reset();

    (*compile sum.c*)
    (*generate sum.bin and sum.ini after compiling*)
    (*Ensure that error.ini locates in project file folder *)
    rtn:=GT_Compile(ADR(filename), ADR(compile));

    (*Download the motion program sum.bin *)
    rtn:=GT_Download(ADR(downloadfilename));

    (*Get the ID of function *)
    rtn:=GT_GetFunId(ADR(funname), ADR(funid));

    (*Get the ID of global variable “sum”*)
    rtn:=GT_GetVarId(0, ADR(varname), ADR(sum));

    (*Get the ID of local variable “begin”*)
    varname:='begin';
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(begin));

    (*Get the ID of local variable “end”*)
    varname:='end';
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(end));
```

```
        (*bind thread, function and data page *)
        rtn:=GT_Bind(0, funId, 0);

        value:=0;
        (*Initial the global variable “sum” of motion program *)
        rtn:=GT_SetVarValue(-1, ADR(sum), ADR(value), 1);

        value:=1;
        (*Initial the local variable “begin” of motion program *)
        rtn:=GT_SetVarValue(0, ADR(begin), ADR(value), 1);

        value:=100;
        (*Initial the local variable “end” of motion program *)
        rtn:=GT_SetVarValue(0, ADR(end), ADR(value), 1);

        (*Start up thread *)
        rtn:GT_RunThread(0);

        First:=FALSE;
    END_IF

    (*Check the status of thread *)
    rtn:=GT_GetThreadSts(0,ADR(thread));

    (*Check the value of global variable “sum”*)
    rtn:=GT_GetVarValue(-1, ADR(sum), ADR(value), 1);
```

(2) Addition in multi-thread

Motion program for addition in multi-thread is as same as example program for addition in single thread.

Application program can compile, download, initialize, and run of motion program. In contrast to example (1), it used 2 threads to complete addition calculation task in example (2).

```
PROGRAM MAIN
VAR
    rtn:INT;
    compile:TCompileInfo;
    funId:INT;
    sum,begin,end:TVarInfo;
    value:DINT;
    thread:TThreadSts;

    First: BOOL:=true;
    filename: STRING:='sum.c';
    downloadfilename: STRING:='sum.bin';
    funname: STRING:='add';
    varname:STRING:='sum';
END_VAR
-------------------------------------------------------------------------------
IF First THEN
    (*Reset the motion controller *)
    rtn:=GT_Reset();

    (*compile sum.c*)
    (*generate sum.bin and sum.ini after compiling*)
    (*Ensure that error.ini locates in project file folder *)
    rtn:=GT_Compile(ADR(filename), ADR(compile));

    (*Download the motion program sum.bin *)
    rtn:=GT_Download(ADR(downloadfilename));

    (*Get the ID of function *)
    rtn:=GT_GetFunId(ADR(funname), ADR(funId));
```

```
(*Get the ID of global variable “sum”*)
rtn:=GT_GetVarId(0, ADR(varname), ADR(sum));

(*Get the ID of local variable “begin”*)
varname:='begin';
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(begin));

(*Get the ID of local variable “end”*)
varname:='end';
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(end));

(*bind thread, function and data page *)
rtn:=GT_Bind(0, funId, 0);
rtn:=GT_Bind(1, funId, 1);

(*Initial the global variable “sum” of motion program *)
value:=0;
rtn:=GT_SetVarValue(-1, ADR(sum), ADR(value), 1);

(*Initial the local variable “begin” of motion program in thread 0*)
value:=1;
rtn:=GT_SetVarValue(0, ADR(begin), ADR(value), 1);

(*Initial the local variable “end” of motion program in thread 0*)
value:=50;
rtn:=GT_SetVarValue(0, ADR(end), ADR(value), 1);

(*Initial the local variable “begin” of motion program in thread 1*)
value:=51;
rtn:=GT_SetVarValue(1, ADR(begin), ADR(value), 1);
```

```
        (*Initial the local variable “end” of motion program in thread 1*)
        value:=100;
        rtn:=GT_SetVarValue(1, ADR(end), ADR(value), 1);

        (*Start thread 0 and 1*)
        rtn:=GT_RunThread(0);
        rtn:=GT_RunThread(1);

        First:=FALSE;
    END_IF

    (*Check the status of thread *)
    rtn:=GT_GetThreadSts(0, ADR(thread));
    rtn:=GT_GetThreadSts(1, ADR(thread));

    (*Check the value of the global variable “sum”*)
    rtn:=GT_GetVarValue(-1, ADR(sum), ADR(value), 1);
    … …
```

(3) Combined motion

This routine realizes the coordination motion of three motion shafts and is applied for semiconductor processing equipment, wherein the three motion shafts are respectively defined as arm shaft, piece shaft and glue shaft. The motion relation of these shafts is described as follows.

Arm shaft: when the glue shaft leaves the working area, the arm shaft is started to move towards the working area; after completing relevant work, the arm shaft could immediately leave the working area, without the need to wait any signal.

Feed shaft: when the arm shaft leaves the working area, the piece shaft could be started.

Glue shaft: after the arm shaft leaves the working area, the glue shaft is started to move towards the working area; when the piece shaft is in place, the glue shaft has completed the gluing work and then could leave the working area.

Three functions are used in the motion program to realize the motion of the three shafts, wherein ArmMotion is used to control the arm shaft, GlueMotion is used to control the glue shaft and PieceMotion is used to control the piece shaft. Meanwhile, the mutual exclusion and the correlation of these motions

are realized through four global synchronization variables.

pieceArrival: the piece shaft arrives, and the glue shaft could be started to leave the working area.

pieceStart: piece shaft starting flag, indicating that the piece shaft could be started.

glueStart: glue shaft starting flag, indicating that the glue shaft could be started to move towards the working area.

armStart: arm shaft starting flag, indicating that the arm shaft could be started to move towards the working area.

ArmMotion function is bound with thread 0 and data page 0; GlueMotion is bound with thread 1 and data page 1; PieceMotion function is bound with thread 2 and data page 2. Each thread independently controls the motion of one shaft, and the initial value of the variable of the motion program is set as follows:

Global variable pieceArrival: 0 (the piece shaft does not arrive)

Global variable armStart: 0 (under initial state, the arm shaft motion condition is not available)

Global variable pieceStar: 1 (under initial state, the piece shaft motion condition is available)

Global variable glueStart: 1 (under initial state, the glue shaft motion condition is available)

In Fig 5-1, axis 1 is the planning speed of the arm shaft, axis 2 is the planning speed of the glue shaft and axis 3 is the planning speed of the piece shaft.

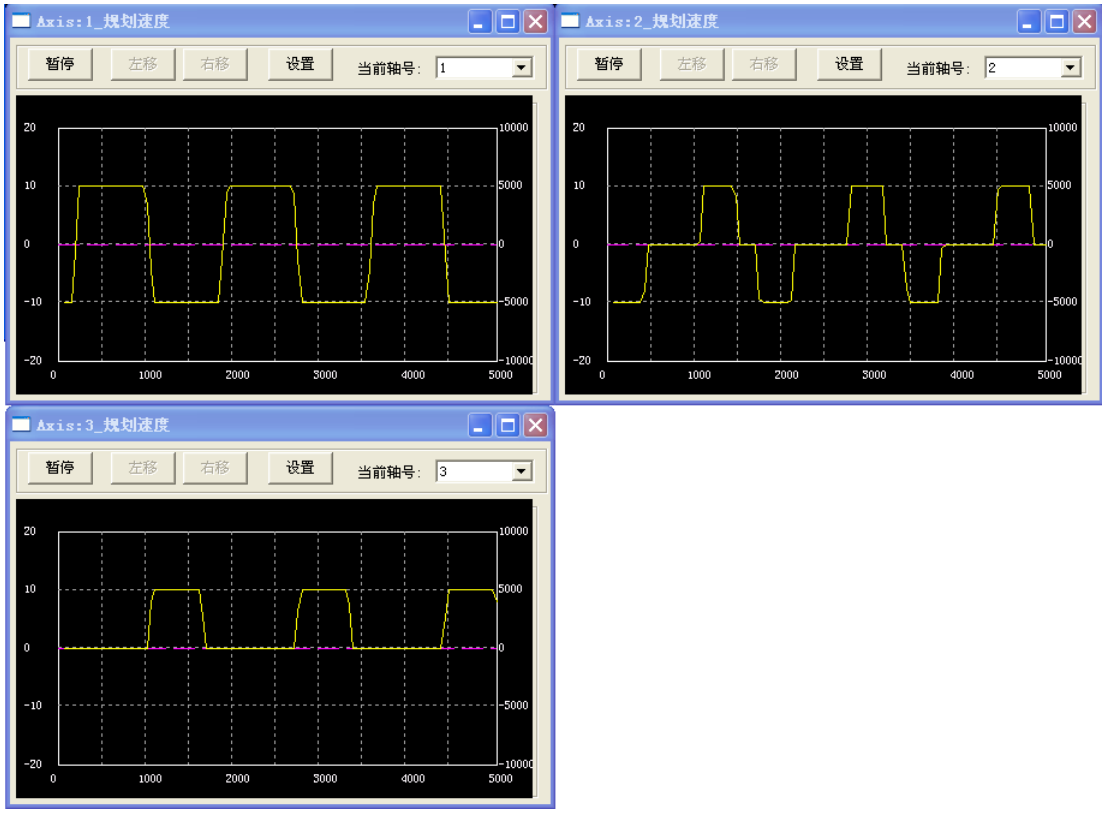

Fig 5-1 Timing diagram of combined motion

Motion program is illustrated as follows：

```
//-----------------------------------------------

// Combined motion

//-----------------------------------------------


int pieceArrival;          //Piece shaft arrive flag

int pieceStart;             //Piece shaft start flag

int glueStart;             // Glue shaft start flag

int armStart;              // Arm shaft start flag


//Arm shaft motion

void ArmMotion(short arm)

{

    long armStep;

    short cc;

    long clock;

    long sts;

    short axis;


lbl_loop:

    axis=arm-1;

    axis=1<<axis;


    //Waiting for the start flag of   arm shaft.

lbl_wait_for_arm_start:

    cc = !armStart;

    if(cc) goto lbl_wait_for_arm_start;


    // Clear arm shaft start flag

    armStart = 0;


    //Arm shaft motion
```

```
        GT_SetPos(arm,armStep);
        GT_Update(axis);

        //Waiting for the stop of arm shaft.
    lbl_wait_for_arm_stop:
        GT_GetSts(arm,&sts,1,&clock);
        cc = sts & 0x400;
        if(cc) goto lbl_wait_for_arm_stop;

        //Arm shaft returns
        GT_SetPos(arm,0);
        GT_Update(axis);

        // Set the glue shaft start flag
        glueStart = 1;

        // Set the piece shaft start flag
        pieceStart = 1;

        // Waiting for the stop of arm shaft.
    lbl_wait_for_arm_return:
        GT_GetSts(arm,&sts,1,&clock);
        cc = sts & 0x400;
        if(cc) goto lbl_wait_for_arm_return;

        goto lbl_loop;
    }

    // Glue shaft motion
    void GlueMotion(short glue)
    {
```

```
    long glueStep;
    short cc;
    long clock;
    long sts;
    short axis;

lbl_loop:
    axis=glue-1;
    axis=1<<axis;

    // Waiting for the start flag of glue shaft.
lbl_wait_for_glue_start:
    cc = !glueStart;
    if (cc) goto lbl_wait_for_glue_start;

    // Clear glue shaft start flag
    glueStart = 0;

    // Glue shaft motion
    GT_SetPos(glue,glueStep);
    GT_Update(axis);

    // Waiting for the stop of glue shaft
lbl_wait_for_glue_stop:
    GT_GetSts(glue,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_glue_stop;

    //Waiting for piece shaft arrival
lbl_wait_for_piece_arrival:
    cc = !pieceArrival;
```

```
        if(cc) goto lbl_wait_for_piece_arrival;

        //Glue shaft returns
        GT_SetPos(glue,0);
        GT_Update(axis);

        // Set the arm shaft start flag
        armStart = 1;

        // Waiting for the stop of motion
    lbl_wait_for_glue_return:
        GT_GetSts(glue,&sts,1,&clock);
        cc = sts & 0x400;
        if(cc) goto lbl_wait_for_glue_return;

        goto lbl_loop;
    }

    // Piece shaft motion
    void PieceMotion(short piece)
    {
        long pieceStep;
        short cc;
        long clock;
        long sts;
        long pos;
        short axis;
        pos = 0;

    lbl_loop:
        axis=piece-1;
```

```
        axis=1<<axis;

        //Waiting for the start flag of piece shaft.
lbl_wait_for_piece_start:
        cc = !pieceStart;
        if (cc) goto lbl_wait_for_piece_start;

        // Clear the start flag of piece shaft.
        pieceStart = 0;

        // Clear the arrival flag of piece shaft.
        pieceArrival = 0;

        //Start up the motion of piece shaft.
        pos = pos + pieceStep;
        GT_SetPos(piece,pos);
        GT_Update(axis);

        // Waiting for the stop of piece shaft motion
lbl_wait_for_piece_stop:
        GT_GetSts(piece,&sts,1,&clock);
        cc = sts & 0x400;
        if(cc) goto lbl_wait_for_piece_stop;

        //Set the arrival flag of piece shaft.
        pieceArrival = 1;

        goto lbl_loop;
}

// Application program is responsible for compiling, downloading,
```

```
// initialling, and running of motion program.
PROGRAM MultiMotion
VAR
    ARMAxis:INT:=1;
    GLUEAxis:INT:=2;
    PIECEAxis:INT:=3;
    ARM_VEL:LREAL:=10;
    GLUE_VEL:LREAL:=10;
    PIECE_VEL:LREAL:=10;
    ARM_STEP:DINT:=6000;
    GLUE_STEP:DINT:=4000;
    PIECE_STEP:DINT:=8000;

    rtn:INT;
    trap:TTrapPrm;
    compile:TCompileInfo;
    armMotion,glueMotion,pieceMotion:INT;
    pieceArrival,pieceStart,glueStart,armStart:TVarInfo;
    arm,armStep,glue,glueStep,piece,pieceStep:TVarInfo;
    value:DINT;
    prfPos:ARRAY [1..8] OF DINT;

    cmpfilename:STRING:='led.c';
    dlfilename:STRING:='led.bin';
    funname:STRING;
    varname:STRING;

    First: BOOL:=TRUE;
END_VAR
-------------------------------------------------------------------------------
IF First THEN
```

```
(*Reset the motion controller *)
rtn:=GT_Reset();

(*Clear the alarm and limit*)
rtn:=GT_ClrSts(1,8);

(*Set the motion parameter of ARM *)
rtn:=GT_PrfTrap(ARMAxis);
rtn:=GT_GetTrapPrm(ARMAxis,ADR(trap));
trap.acc:=0.25;
trap.dec:=0.25;
rtn:=GT_SetTrapPrm(ARMAxis,ADR(trap));
rtn:=GT_SetVel(ARMAxis,ARM_VEL);
rtn:=GT_Update(SHL(DWORD#1,ARMAxis-1));

(*Set the motion parameter of GLUE *)
rtn:=GT_PrfTrap(GLUEAxis);
rtn:=GT_GetTrapPrm(GLUEAxis, ADR(trap));
trap.acc:=0.25;
trap.dec:=0.25;
rtn:=GT_SetTrapPrm(GLUEAxis, ADR(trap));
rtn:=GT_SetVel(GLUEAxis, GLUE_VEL);
rtn:=GT_Update(SHL(DWORD#1,GLUEAxis-1));

(*Set the motion parameter of PIECE*)
rtn:=GT_PrfTrap(PIECEAxis);
rtn:=GT_GetTrapPrm(PIECEAxis, ADR(trap));
trap.acc:=0.25;
trap.dec:=0.25;
rtn:=GT_SetTrapPrm(PIECEAxis, ADR(trap));
rtn:=GT_SetVel(PIECEAxis, PIECE_VEL);
```

```
rtn:=GT_Update(SHL(DWORD#1,PIECEAxis-1));

(*compile sum.c*)
(*generate sum.bin and sum.ini after compiling*)
(*Ensure that error.ini locates in project file folder *)
rtn:=GT_Compile(ADR(cmpfilename), ADR(compile));

(*Download the motion program *)
rtn:=GT_Download(ADR(dlfilename));

(*Get the ID of function *)
funname:='armMotion';
rtn:=GT_GetFunId(ADR(funname), ADR(armMotion));
funname:='glueMotion';
rtn:=GT_GetFunId(ADR(funname), ADR(glueMotion));
funname:='pieceMotion';
rtn:=GT_GetFunId(ADR(funname),ADR(pieceMotion));

(*Get the ID of global variable*)
varname:='pieceArrival';
rtn:=GT_GetVarId(0, ADR(varname), ADR(pieceArrival));
varname:='pieceStart';
rtn:=GT_GetVarId(0, ADR(varname), ADR(pieceStart));
varname:='glueStart';
rtn:=GT_GetVarId(0, ADR(varname), ADR(glueStart));
varname:='pieceStart';
rtn:=GT_GetVarId(0, ADR(varname), ADR(pieceStart));

(*Get the ID of local variable *)
funname:='ArmMotion';
varname:='arm';
```

```
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(arm));
varname:='armStep';
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(armStep));

funname:='GlueMotion';
varname:='glue';
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(glue));
varname:='glueStep';
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(glueStep));

funname:='PieceMotion';
varname:='piece';
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(piece));
varname:='pieceStep';
rtn:=GT_GetVarId(ADR(funname), ADR(varname), ADR(pieceStep));

(*bind thread, function and data page *)
rtn:=GT_Bind(0, armMotion, 0);
rtn:=GT_Bind(1, glueMotion, 1);
rtn:=GT_Bind(2, pieceMotion, 2);

(*Initial the global variable of motion program *)
value:=0;
rtn:=GT_SetVarValue(-1, ADR(pieceArrival), ADR(value), 1);
rtn:=GT_SetVarValue(-1, ADR(armStart), ADR(value), 1);

value:=1;
rtn:=GT_SetVarValue(-1, ADR(pieceStart), ADR(value), 1);
rtn:=GT_SetVarValue(-1, ADR(glueStart), ADR(value), 1);

(*Initial the local variable of motion program *)
```

```
        value:=ARMAxis;
        rtn:=GT_SetVarValue(0, ADR(arm), ADR(value), 1);
        value:=ARM_STEP;
        rtn:=GT_SetVarValue(0, ADR(armStep), ADR(value), 1);

        value:=GLUEAxis;
        rtn:=GT_SetVarValue(1, ADR(glue), ADR(value), 1);
        value:=GLUE_STEP;
        rtn:=GT_GetVarValue(1, ADR(glueStep), ADR(value), 1);

        value:=PIECEAxis;
        rtn:=GT_SetVarValue(2, ADR(piece), ADR(value), 1);
        value:=PIECE_STEP;
        rtn:=GT_SetVarValue(2, ADR(pieceStep), ADR(value), 1);

        (*Start a thread *)
        rtn:=GT_RunThread(0);
        rtn:=GT_RunThread(1);
        rtn:=GT_RunThread(2);

        First:=FALSE;
    END_IF

    GT_GetPrfPos(1, ADR(prfpos), 8 ,0);
```

5.3 Language elements

5.3.1 Data type

Int and float data type are supported.

Int data type use 32 bits, range is [-2,147,483,648, 2,147,483,648].

Float data type use fix-pointer, 32 bits in integral part, 16 bits in fractional part. The resolution is (1/2)^16=0.0000152587890625。

5.3.2 Constant

The immediate and macro are allowed, the immediate can be described as Decimal, Hexadecimal or float number.

5.3.3 Variable

User can define local variables and global variables. For each function, maximal number of local variables is 1024; maximal number of global variables is 1024. Integral type declaration symbol is "int", and float type declaration symbol is "double".

5.3.4 Array

One dimension array and constant index and variable index are supported.

Multi-dimension array and using array element as index are not supported.

5.3.5 Function

The return data type and input data type of a function can be declared. It is not allowed to call self-defined function but GT commands in function.

5.3.6 Data type conversion

F orced data type conversion is supported. The conversion symbols are "int, double".

1. The data type conversion symbol with (), like a=(int)b
2. The data type conversion symbol cannot change the data type definition of the variable itself.

5.3.7 Operators

Motion program supports arithmetic operation, logical operation, relational operation, bitwise operation. The grammar rule is the same as that of C language. But it is not allowed for complex operations, only allowed 2 operands in an operation expression, and the data type of these two operands must be same.

Notice: Please do not use high precise float calculation, since there is only 16 bit fraction accuracy for float data type in motion program.

5.3.8 Arithmetic operator

Arithmetic operators includes plus (+), minus (-), multiply (*), divide (/), mode (%).

5.3.9 Logical operator

L ogical operators includes logical AND (&&), logical OR (||), and logical NOT (!). The parameter can be INT variable or INT constant.

5.3.10 Relational operator

Relational operators are used for comparison operation, includes less than(>), larger than(<), equal to(==), larger than or equal to(>=), less than or equal to(<=) and not equal to(!=). The parameter data type for comparison must be same.

5.3.11 Bitwise operator

B itwise operation is done by bits, includes bitwise AND(&), bitwise OR(|), bitwise NOT(~), bitwise XOR(^), bitwise left shift(<<), bitwise right shift(>>).

5.4 Sequential control

Motion program supports Jump by condition; return by condition and grammar rules are the same as that of C language.

1. Conditional jump is used as following,

   if(var) goto label；
   When the condition variable “var” is not 0, it jumps to the command which with symbol “label”.
   Using an expression as jump condition is not supported.

2. Conditional return is used as following,

   if(var) return value；
   When the condition variable “var” is not 0, the program return, return value is "value"
   Using an expression as return condition is not supported.