GOOGOL TECHNOLOGY (HK) LTD

Programming Manual

For

GT Series Motion Controller

Revision 1.10 June, 2003

Part#: GT400-M-E-R1100603-002

Copyright Statement

Googol Technology Ltd. (Googol Technology hereafter) reserves the right of modifying the products and product specifications described in this manual without notification in advance.

Googol Technology is not submitted to any direct, indirect, special, incidental or consequent loss or liability caused by using this manual or product incorrectly.

Googol Technology owns the patent, copyright or any other intellectual property right of this product and its software. No one shall directly or indirectly duplicate, produce, process or use this product and its relevant parts.

注意 **Notice**	***There is danger in a controller in motion! User has the duty to design an effective error treatment and safety protection system for the machine. Googol Technology has no such obligation or liability to be responsible for any incidental or consequent loss caused accordingly.***

Foreword

Thank you for using Googol Technology motion controller

To repay customers, we will help you establish your own control system with our first-class motion controller, perfect after-sale service, and high-efficiency technical support.

Technical Support and After-sale Service

You may get our technical support and after-sale service through the following approaches:

- E-mail: *support@googoltech.com*
- Tel.: (0755) 2697-0823, 2697-0835, 2697-0839
- Post: Googol Technology (Shenzhen) Co., Ltd.
  W211, 2/F, West Building, SZHK Industry, Education and Research Base, South Area, High-tech Industry Park, Shenzhen, 518057

Use of this Programming Manual

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

User of this Programming Manual

This manual is applicable to those engineering developers having the base knowledge of programming in C or using Dynamic Link Library (DLL) in Windows environment, and 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 is divided into two parts. The first part is programming specification, mainly introducing various control functions and corresponding programming approaches of GT series motion controller. The second part is interface library function specification, mainly introducing the function prototype, description and evoking of interface functions of GT series motion controller.

Usage Description of this Programming Manual

Except stated specially, among all the samples listed in this manual, all DOS samples are written in Borland C3.1 and all Windows samples in VC++.

Relevant Documents

For the installation and debugging of GT series motion controller, please refer to User's Guide of GT Series Motion Controller provided together with product.

Contents

Copyright Statement ----------1

Foreword ----------1

Thank you for using Googol Technology motion controller ----------1
Technical Support and After-sale Service ----------1
Use of this Programming Manual----------1
User of this Programming Manual----------1
Main Contents of this Programming Manual----------1
Usage Description of this Programming Manual ----------2
Relevant Documents----------2

Contents ---------- I

Chapter One Use of Function Library of Motion Controller -----1

1.1 Function Library in DOS----------1
1.2 DLL in Windows----------1
1.2.1 Usage in VC----------2
1.2.2 Usage in VB----------2
1.2.3 Usage in Delphi----------2

Chapter Two Command Return Values and their Meaning ----------3

2.1 Command (Function Library) Return Values ----------3
2.2 Command Status Register ----------4

Chapter Three Initialization of Control System ----------7

3.1 Initialization of Motion Controller ----------7
3.1.1 Command Summary ----------7
3.1.2 Example ----------7
3.1.3 Notes of Main Point----------8
3.2 Specify Parameters for Dedicated Input Signal----------9
3.2.1 Command Summary ----------9
3.2.2 Example ----------9
3.2.3 Notes of Main Point---------- 10
3.3 Initialization of Motion Control Axis ---------- 11
3.3.1 Function List ---------- 11
3.3.2 Example ---------- 11
3.2.3 Notes of Main Point---------- 13

Chapter Four Independent Axis Motion---------- 17
4.1 Modes and Parameters of Motion ---------- 17
4.1.1 Independent Positioning: S-curve ---------- 17
4.1.2 Independent Positioning: T-curve ---------- 19
4.1.3 independent jogging---------- 21
4.1.4 Electronic Gearing ---------- 22
4.2 Stop Motion ---------- 23
4.2.1 Command Summary ---------- 23
4.2.2 Example ---------- 24
4.2.3 Notes of Main Point ---------- 24
4.3 Specify and Update Parameters of Specified Axis ---------- 25
4.3.1 Normal Updating---------- 26
4.3.2 Self-Updating at Breakpoint ---------- 27
4.4 Specifying Target Position and Actual Position for Specified Axis---------- 31
4.4.1 Command Summary ---------- 31
4.4.2 Notes of Main Point ---------- 31
4.5 Axis Status---------- 32
4.5.1 Axis Status Register---------- 32
4.5.2 Axis Mode Register ---------- 34
Chapter Five Coordinated Motion(The series of GT_PX do not contain)---------- 36
5.1 Coordinate Mapping ---------- 36
5.2 Set Vector Velocity and Acceleration of Coordinated Motion---------- 40
5.2.1 Command Summary ---------- 40
5.2.2 Example ---------- 40
5.2.3 Notes of Main Point ---------- 41
5.3 Set Motion Path in Coordinate System ---------- 42
5.3.1 Command Summary ---------- 42
5.3.2 Example ---------- 42
5.3.3 Notes of Main Point ---------- 43
5.4 Realization of Multi-segment Path Continuous Motion ---------- 43
5.4.1 Push Motion Path and Parameters Commands into Buffer ---------- 44
5.4.2 commands of Starting and stopping coordinated motion in the buffer - 46
5.4.3 Planning Strategy of Vector Velocity in Multi-segment path ---------- 48
5.4.4 Breakpoint information in Multi-segment path Continuous Motion---- 49
5.4.5 Coordinate System Status Register ---------- 50
Chapter Six High velocity Home/Index Capture---------- 52
Chapter Seven Safety Mechanism ---------- 55
7.1 Monitor Axis Motion Error and Restore Status ---------- 55
7.2 Treat Axis Driver Alarm---------- 56

7.3 Treat Limit Status---------- 56

Chapter Eight Interrupt ---------- 58

8.1 Interrupt Treatment in DOS---------- 58
8.2 Interrupt Treatment in WINDOWS98/2000/NT---------- 61
8.2.1 Event Synchronization Mechanism ---------- 61
8.2.2 Interrupt Preprocessing Mechanism ---------- 63

Chapter Nine General Purposed I/O ---------- 66

Chapter Ten List of Functions ---------- 68

Chapter Eleven Description of Functions ---------- 72

GT_AbptStp ---------- 72
GT_AuStpOff ---------- 72
GT_AuStpOn---------- 73
GT_AuUpdtOff---------- 73
GT_AuUpdtOn---------- 74
GT_Axis---------- 74
GT_AxisI---------- 74
GT_AxisOff ---------- 75
GT_AxisOn---------- 76
GT_BrkOff---------- 76
GT_CaptHome ---------- 76
GT_CaptIndex---------- 77
GT_CaptProb---------- 77
GT_Close ---------- 78
GT_CloseLp ---------- 79
GT_ClearInt ---------- 79
GT_ClrEncPos ---------- 80
GT_ClrSts---------- 80
GT_CtrlMode---------- 81
GT_DrvRst---------- 81
GT_EncPos ---------- 82
GT_EncSns ---------- 82
GT_EncVel---------- 83
GT_EStpMtn ---------- 84
GT_EvntIntr ---------- 84
GT_ExInpt---------- 85
GT_ExOpt ---------- 85
GT_ExtBrk---------- 86
GT_GetAcc ---------- 86
GT_GetAdc---------- 87
GT_GetAddr---------- 87
GT_GetAtlErr ---------- 88

GT_GetAtlPos ---- 88
GT_GetBgCommandResult ---- 89
GT_GetBrkCn ---- 89
GT_GetCapt ---- 89
GT_GetCmdSts ---- 90
GT_GetCurrentCardNo ---- 92
GT_GetEncCapt ---- 92
GT_GetEncSts ---- 92
GT_GetILmt ---- 92
GT_GetIntgr ---- 93
GT_GetIntr ---- 93
GT_GetIntrMsk ---- 93
GT_GetIntrTm ---- 93
GT_GetJerk ---- 94
GT_GetKaff ---- 94
GT_GetKd ---- 94
GT_GetKi ---- 94
GT_GetKp ---- 95
GT_GetKvff ---- 95
GT_GetLmtSwt ---- 95
GT_GetMAcc ---- 96
GT_GetMode ---- 96
GT_GetMtrBias ---- 96
GT_GetMtrCmd ---- 97
GT_GetMtrLmt ---- 97
GT_GetPos ---- 97
GT_GetPosErr ---- 97
GT_GetRatio ---- 98
GT_GetSmplTm ---- 98
GT_GetSts ---- 98
GT_GetVel ---- 98
GT_HardRst ---- 99
GT_Home ---- 99
GT_HookIsr ---- 100
GT_Index ---- 100
GT_LmtSns ---- 101
GT_LmtsOff ---- 102
GT_LmtsOn ---- 102
GT_MltiUpdt ---- 102
GT_MtnBrk ---- 103
GT_NegBrk ---- 103
GT_OpenLp ---- 104
GT_Open ---- 104
GT_Open ---- 104
GT_PosBrk ---- 104

GT_PrflG----------105
GT_PrflS----------105
GT_PrflT----------106
GT_PrflV----------106
GT_Reset----------107
GT_RstIntr----------107
GT_RstSts----------108
GT_SetAcc----------108
GT_SetAdcChn----------108
GT_SetAddr----------108
GT_SetAtlPos----------109
GT_SetBgCommandSet----------109
GT_SetBrkCn----------111
GT_SetEncCapt----------111
GT_SetILmt----------111
GT_SetIntrMsk----------111
GT_SetIntrTm----------112
GT_SetIntSyncEvent----------112
GT_SetJerk----------113
GT_SetKaff----------114
GT_SetKd----------114
GT_SetKi----------114
GT_SetKp----------114
GT_SetKvff----------115
GT_SetMAcc----------115
GT_SetMtrBias----------115
GT_SetMtrCmd----------115
GT_SetMtrLmt----------116
GT_SetPos----------116
GT_SetPosErr----------116
GT_SetRatio----------116
GT_SetSmplTm----------117
GT_SetTime----------117
GT_TmrIntr----------117
GT_SetVel----------117
GT_SmthStp----------118
GT_StepDir----------118
GT_StepPulse----------118
GT_SwitchtoCardNo----------118
GT_SynchPos----------119
GT_UnhookIsr----------119
GT_Update----------120
GT_ZeroPos----------120

Contents

Part One

Programming Specification

Chapter One	Use of Library Functions of Motion Controller
Chapter Two	Command Return Values and their Purposes
Chapter Three	Initialization of Control System
Chapter Four	Single-spindle Motion
Chapter Five	Multiple-spindle Coordination Motion
Chapter Six	Home and Index Capture
Chapter Seven	Safety Mechanism and Corresponding Treatment
Chapter Eight	Interrupt
Chapter Nine	Universal Digital Values and I/O Operations

Chapter One Use of Function Library of Motion Controller

The motion controller provides motion function library in DOS and DLL in Windows. Just by calling the command in library, the user can realize various functions of motion controller. The use of library function in DOS and Windows will be described respectively as follow.

1.1 Function Library in DOS

The motion function library in DOS exists in the directory of DOS\UserLib in the CD provided with product, including seven files:

userlib.h	Head file
userlibt.lib	Function library of micro mode
userlibs.lib	Function library of small mode
userlibm.lib	Function library of middle-sized mode
userlibc.lib	Function library of compact mode
userlibl.lib	Function library of large mode
userlibh.lib	Function library of huge mode

This library is compiled in Borland C3.1. Developers can link it in Borland C3.1 or higher versions of developing environment.

Methods of using the library:

1. Select Project – Open Object to create a project file, in Borland C3.1 developing environment.
2. Add<＃include “userlib.h”> into the program developed.
3. Select Project – Add Item in Borland C environment and add the c or cpp files developed into the project file.
4. Copy the userlib.h and library files needed into the directory of the project file.
5. Select Project – Add Item in Borland C environment and add the library files into the project file.
6. Select Option – Compiler – Code Generation in Borland C environment and select the compiling mode corresponding to the compiling mode of library.
7. Then the user can call any command in the library. For the detailed usage and definition of the functions, please refer to part two of this manual.

1.2 DLL in Windows

The DLL in Windows exists in the directory of Windows\Dll in the CD provided with product, including several files, GTDLL.h, GTDLL.lib and GTDLL.dll for ISA card, and GT400.h, GT400.lib and GT400.dll for PCI card, which are written in VC++. Regarding the advanced programming languages VC, VB and Delphi frequently used by programmers today, the use of DLL in such languages will be described one by one as follows.

1.2.1 Usage in VC

ISA Card:

1. Add the following statement into the user program:
   #include “GTDLL.h"
2. From the menu of VC environment, select project-setting setting – link. Input GTDLL.lib into Object/library modules. Then user can call any command in the DLL.

PCI Card:

1. Add the following statement into the user program:
   #include “GT400.h"
2. From the menu of VC environment, select project – setting – link. Input GT400.lib into Object/library modules. Then user can call any command in DLL.

1.2.2 Usage in VB

According to the type of the bus on board, you can use GTDeclarISA.bas or GTDeclarPCI.bas directly by copying them from the directory of Windows\VB in the CD into your project.

1.2.3 Usage in Delphi

You can use GTFunc.pas directly by copying it from the directory of Windows\Delphi in the CD into your project.

Chapter Two Command Return Values and their Meaning

2.1 Command (Function Library) Return Values

Command and Library Function: The motion controller works according to the ***motion controller commands*** sent by the host. It has C function library (in DOS) and DLL (in Windows). The user can call corresponding ***function in the library,*** when he is programming in the host, that is, send ***motion controller command*** (***command*** for short) to the controller.

After receiving a command from the host, the motion controller will give a feedback after checking and verifying it. This feedback is the ***command (library function) return value***. The definitions of return values are listed in Table 2-1.

Table 2-1 Definition of Return Values

Value	Meaning	Processing Method
-1	Error in communication.	Check whether the controller works normally, according to the "Fault Processing" in User's Manual of GT series Motion Controller.
0	Command execution succeeded.	Operation can go on.
1	Error in command.	Call GT_GetCmdSts(). Find causes and correct them.
2	Wrong circular interpolation radius.	This value can only be returned after GT_ArcXY, GT_ArcYZ or GT_ArcZX command is called. Check the command parameters and send the command again after correction.
3	Linear interpolation length is zero, negative or exceeds the motion bound of the controller.	This value can only be returned after GT_LnXY, GT_LnXYZ or GT_LnXYZA command is called. Check the command parameters and send the command again after correction.
4	In coordinated motion mode, the vector velocity (acceleration) is zero, negative or exceeds the motion bound of the controller.	This value can only be returned after GT_SetSynVel or GT_SetSynAcc command is called. Check the command parameters and send the command again after correction.

Value	Meaning	Processing Method
5	Wrong position of end point or radius description in circular interpolation.	This value can only be returned after GT_ArcXYP, GT_ArcYZPor or GT_ArcZXP command is called. Check the command parameters and send the command again after correction.
6	Coordinate mapping failed.	This value can only be returned after GT_MapAxis command is called. Check whether the mapping is correct, or conflicts with the mapping command executed before. Send the command again after correction.
7	General parameter error.	Check whether the command parameters are reasonable or exceed the limits. Send the command again after correction.

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

If the return value is -1 and it returns several times when being called repeatedly, it means that the communication of the motion controller has fault and the controller doesn't receive the command from the host correctly, or the function library doesn't work normally, unable to process the command from the host correctly. At this time, user should stop executing the program and take relevant processing method according to Appendix D – Fault Process in User's Guide of GT Series Motion Controller.

If the return value is 1, it means the commands called by user are illegal and the motion controller neglects these illegal commands. If the illegal command is a ***coordinated motion command***, it will cause bit3 in the coordinated motion status to be set. If the illegal command is an ***independent axis motion command***, it will cause bit7 in the current axis status to be set. Meanwhile, The ***command status register*** of the motion controller provides the detailed reason for command error. The host can call the command GT_GetCmdSts() to get the causes of command error.

2.2 Command Status Register

The command status register of the motion controller provides the detailed causes of command errors, the user can call the command GT_GetCmdSts() to get the status. The register is a 16-bit register (among which, bit0-bit 11 are mainly for the commands of independent axis, and bit12-bit15 shows the causes for commands of coordinate system.). The definition for each bit is listed in Table 2-2.

The following example is a return value processing function. The user can program own

function to process return value refer to the example.

Sample 2-1: Return Value Processing Function

```
void error (short rtn)          //Return value processing function, rtn is the return
                                value of the command.
{
    switch (rtn)
    {
        case -1:
             printf("error: communication error\n");   break;
        case 0:
             /*no error, continue */     break;
        case 1:
             printf("error: command error\n");    break;
        case 2:
        case 3:
        case 4:
        case 5:
        case 7:
             printf("error: parameter error\n");   break;
        case 6:
             printf("error: map is error\n");    break;
        default:
            break;
     }

    }
```

Table 2-2 Definition of Command Status Register

Bit	Definition
Bit0	1: The motion parameter is overflow. Commands causing this error are GT_SetPos, GT_SetVel, GT_SetAcc and GT_SetAtlPos, etc.
Bit1	1: The control parameter is illegal. Commands causing this error are GT_SetVel, GT_SetAcc, GT_SetJerk, GT_SetMAcc, GT_SetMtrLmt, GT_SetKp, GT_SetKi, GT_SetKd, GT_SetKvff, GT_SetKaff, GT_SetILmt and GT_SetPosErr, etc.
Bit2	1: The host call GT_MltiUpdt (value), but the value=0.
Bit3	1: The illegal use of GT_DrvRst. When the current axis is in activating status, the host calls this command.
Bit4	1: The controller doesn’t generate event interrupt, but the host calls a command of interrupt.
Bit5	Reserved
Bit6	1: When the specified axis is in motion, the host calls a command to change the motion mode of specified axis (except that specified axis is in electronic gearing mode).
Bit7	1: The sign of the position captured of the current axis status register is 1, or the command GT_CaptIndex（GT_CaptHome）has been called to specify

Bit	Definition
	the capture mode and before the position is captured, the host calls the command GT_CaptIndex again.
Bit8	1: The sign of the position captured of the current axis status register is 1, or the command GT_CaptIndex（GT_CaptHome）has been called to specify the capture mode and before the position is captured, the host calls the command GT_CaptHome again.
Bit9	1: When the driver alarm signal of the current axis status register is 1, and the host calls the command GT_AxisOn.
Bit10	Reserved
Bit11	1: When the current axis is in motion, the host calls the command GT_ZeroPos() , or call the command GT_Update() (GT_MltiUpdt()) to modify some parameter which can not be changed during the current axis is in motion; When the current axis motion mode is the independent jogging mode, the host calls the command GT_SynchPos and makes it valid by GT_Update(); (For example, in the S-curve motion mode, when the motor is in motion, the host calls the command GT_SetVel and GT_Update or GT_MltiUpdt.)
Bit12	1: The command of coordinate system is illegal, including: When setting up the coordinate system, the mapped axis is in motion. When coordinated motion command in the buffer is in execution, call GT_StrtMtn(), GT_StrtList(). When input status of the buffer command is not end, call GT_AddList. When it is in single-segment path coordinated motion, and the motion is not finished yet, specify a new motion path command.
Bit13	1: Errors in calling GT_MvXY, GT_MvXYZ and GT_MvXYZA.
Bit14	Reserved
Bit15	1: The buffer is full. Since the buffer is full, the motion path and parameters command called last time cannot be received by the motion control. The host needs to repeat calling this command till it is received.

Chapter Three Initialization of Control System

3.1 Initialization of Motion Controller

3.1.1 Command Summary

Table 3-1 Initialization Command Summary of Motion

Function	Description
GT_Open()	Open the motion controller (in other application conditions except for that ISA card is used in DOS).
GT_SetAddr()	Set communication base address (only for ISA card being used in DOS).
GT_SwitchtoCardNo()	Specify the current control card (only for PCI multi-card control system).
GT_Reset()	Reset the motion controller.
GT_SetSmplTm()	Set control period.

3.1.2 Example

Example 3-1 Initialization Function of Motion Control Card (ISA Card, for DOS)

```
void GTInitial()              //Initialization function of motion controller
{
    short rtn;
    unsigned long PortBase=0x300;   //Evaluate base address.

    rtn=GT_SetAddr(PortBase); error(rtn); //Set base address.
    rtn=GT_Reset( );       error(rtn);       //Reset motion controller.
    rtn=GT_SetSmplTm(200);    error(rtn); //Set control period as 200us.
}
```

Example 3-2 Initialization Function of Motion Control Card (ISA Card, for Windows)

```
void GTInitial()              //Initialization function of motion controller
{
    short rtn;
    unsigned long PortBase=0x300;   //Evaluate base address.
    unsigned long irq=0;            //Evaluate interrupt number.

    rtn=GT_Open(PortBase;irq );  error(rtn);   //Open the motion controller.
    rtn=GT_Reset( );   error(rtn);          //Reset motion controller.
    rtn=GT_SetSmplTm(200);  error(rtn);   //Set control period as 200us.
}
```

Example 3-3 Initialization Function of Motion Control Card (PCI Card)

```
void GTInitial()            //Initialization function of motion controller
{
```

```
    short rtn;
    rtn=GT_Open( );    error(rtn);      //Open the motion controller.
    rtn=GT_Reset( );   error(rtn);      //Reset motion controller.
    /* Set No. 1 card as the current card (only for multi-card system, and this line can
      be cancelled for single-card system.).*/
    rtn=GT_SwitchtoCardNo(1); error(rtn);
    rtn=GT_SetSmplTm(200); error(rtn);   //Set control period as 200us.
    for(int i=1; i<5; i++;)              //Screen each axis interrupt.
    {
        rtn=GT_Axis(i);
        rtn=GT_SetIntrMsk(0);
    }
}
```

3.1.3 Notes of Main Point

Specify the communication base address of the motion controller (only for ISA card in DOS)

The motion controller communicates with the host through ISA bus of PC. In fact, the base address is to tell the host about the site of the motion controller and enables the communication between both.

The command of specifying communication base address is short GT_SetAddr(unsigned short BaseAddr). The parameter of BaseAddr should correspond to the “Base Address jumper” on the motion control board (See ***Set jumper on Motion Control Card*** in User’s Guide of GT Series Motion Controller.). The default base address of motion controller is 0x300.

If the return value of the command is 0, it means a successful communication between the host and motion controller. If the return value is –1, it means a failed communication.

When initializing the controller, if the base address of the controller is not the default one, user should call this command at first to establish successful communication between the host and controller.

Open motion controller (only for ISA card in Windows)

short GT_Open(unsigned long PortBase, unsigned long irq) must be called to open the controller when the user program in Windows. PortBase is the base address of controller, and irq is the interrupt number of controller. When selecting the base address and interrupt number, the user must pay attention to avoid collision with other equipment. Otherwise, the command execution will be failed. The return value = 0 means success, otherwise means failure.

Prompt	*If interrupt numbers 10-15 of the host are occupied, or user doesn't need interrupt, the value of irq should be set as 0 to indicate no interrupt.*

The command relative to GT_Open is short GT_Close(void). It must be called when the user's program is end and exit, and motion controller is closed. The return value = 0 means success, otherwise means failure.

Specify Sample Time

The motion controller controls output at a specific sample time. User can change the sample time according to system requirements. Command to specify the sample time is short GT_SetSmplTm(double Timer) and the unit of parameter Timer is microsecond.

Since the motion controller has to finish necessary control calculation within one sample time and the period cannot be too shorter, the scope should be set between 48-1966.08 microseconds. The default sample time is 200 microseconds, which can make the controller work safely and reliably for common users.

We suggest that user do not modify the sample time shorter than the default value above, otherwise, it may cause the controller to work abnormally. If an actual system requires to reduce the sample time, the command GT_GetCrdSts() should be called frequently during program running to get the information about whether the new sample time is appropriate or not (For detailed description of this command, see 1.7 Status Monitoring). If bit2 is 1, it means the sample time is too short and motion controller works abnormally. At this time, please reset the controller immediately and restore to the default sample time value or prolong the sample time.

Notice	*Specifying sample time can only be done during initialization. It is suggested that the user do not modify the sample time during program running. Otherwise, it will cause unpredictable result.*

3.2 Specify Parameters for Dedicated Input Signal

3.2.1 Command Summary

Table 3-2 Command Summary of Specifying Parameters for Dedicated Input Signal

Function	Description
GT_LmtSns()	Specify the effective electrical level for limit switch.
GT_EncSns()	Specify the direction of encoder (only for SV card).

3.2.2 Example

Example 3-4 Specify Parameters for Dedicated Input Signal

```
void InputCfg()                    //Specify parameters for dedicated input
```

signals.

```
{
    short rtn;
    unsigned short LmtSense = 0;   //set the parameter of limit switch as high-level
  triggering.
    unsigned int EncSense = 0xF;    /* Opposite the direction of 1-4 axes encoders
                                       to original direction, 1 and 2 auxiliary
                                       encoder keep the original direction. */

    rtn=GT_LmtSns(LmtSense); error(rtn); /* Specify 1-4 axes positive and
negative limit switches as high-level triggering. */
    rtn=GT_EncSns(EncSense); error(rtn); //Specify the direction of encoder.

    }
```

3.2.3 Notes of Main Point

Set effective electrical level for limit switch.

The motion controller uses two (positive and negative) limit switches to set the motion scope of the control axis automatically. Once the limit switch is triggered, the controller will prohibit automatically the motion of control axis towards the limit.

The default limit switch is the **permanently closed switch**. In normal work, the signal of limit switch is at low level, and when switch is triggered, it is at high level. If the commonly used switch is used, or the level of the switch is opposite to the above default status caused by connection, user needs to use a command to change the effective level, which is short GT_LmtSns(unsigned short Sense).

The parameter Sense indicates the effective level status of the positive (or negative) limit switch of each axis. The default values of motion controller are all “0”. For detailed explanation, please see the definition of this function.

Set the direction of coder (only for SV card).

The default set of controller considers the positive direction of controlling the rotating of motor (i.e. the motor control voltage is the rotating direction of motor at positive direction) is the same as the positive direction of coder calculation (i.e. the increasing direction of pulse calculation). But in actual application, it may be due to the unmatched set of motor and coder or wrong connection that the positive rotating of motor is opposite to that of coder, forming positive feedback and causing controller abnormal in work. Now, user can set the positive direction of the coder calculation with a command short GT_EncSns(unsigned int Sense).

The corresponding bit of the parameter Sense means whether to change the calculation direction of the corresponding axis coder. The default values of motion control are all “0”. For detailed explanation, please see the definition of this function.

3.3 Initialization of Motion Control Axis

3.3.1 Function List

Table 3-3 Initialization Function List of Motion Controller

Function		Description
GT_Axis()		Set the current axis.
GT_ClrSts()		Clear the current axis status.
GT_StepDir()		Set the output mode of the current axis in the pulse output mode as “Pulse/Direction”.
GT_StepPulse()		Set the output mode of the current axis in the pulse output mode as “Positive Pulse/Negative Pulse”.
GT_AxisOn()		Activate drive.
Only for SV card	GT_CtrlMode()	Set output analog amount/pulse amount.
	GT_CloseLp()	Set as close loop control.
	GT_OpenLp()	Set as open loop control.
	GT_SetKp()	Set the servo filter percentage gain of the current axis.
	GT_SetKi()	Set the servo filter integral gain of the current axis.
	GT_SetKd()	Set the servo filter differential gain of the current axis.
	GT_SetKvff()	Set the servo filter velocity feedback gain of the current axis.
	GT_SetKaff()	Set the servo filter acceleration feedback gain of the current axis.
	GT_SetILmt()	Set the servo filter error integral saturation of the current axis.
	GT_SetMtrLmt()	Set the servo filter output saturation of the current axis.
	GT_SetMtrBias()	Set the servo filter output zero point excursion of the current axis.

3.3.2 Example

Example 3-5 Initialization Function of Axes (For SD, SE, SG and SP Cards)

```
void AxisInitial()              //Initialization function of axes.
{
    short rtn;

    for(int i=0;i<4;i++)
    {
      rtn=GT_Axis(i);       error(rtn);         //Set the i axis as current axis.
      rtn=GT_ClrSts();      error(rtn);   //Clear the wrong status of current axis.
      rtn=GT_StepPulse();   error(rtn);   //Set to output positive and negative
  pulse singal.
      rtn=GT_AxisOn();      error(rtn);         //Activate this axis.
    }
```

```
}

void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  }
```

Example 3-6 Initialization Function of Axes (For SV Card)

```
void AxisInitial()                  //Initialization function of control axis.
{
    short rtn;

    rtn=GT_Axis(1);       error(rtn);        //Set the first axis as current axis.
    rtn=GT_ClrSts();      error(rtn);   //Clear the wrong status of current axis.
    rtn=GT_CtrlMode(0);   error(rtn);   //Set to output analog amount.
    rtn=GT_CloseLp();   error(rtn);        //Set as close loop control.
         rtn=GT_SetKp(20);       error(rtn);       //Set percentage gain as20.
         rtn=GT_SetKi(0);        error(rtn);       //Set integral gain as 0.
         rtn=GT_SetKd(10);       error(rtn);        //Set differential gain as 10.
         rtn=GT_SetKvff(0);      error(rtn);       //Set velocity feedback as 0.
         rtn=GT_SetKaff(0);      error(rtn);  //Set acceleration feedback as 0.
         rtn=GT_SetMtrBias(10); error(rtn);  //Set output zero point excursion as 10.
         rtn=GT_Update();        error(rtn);       //Update parameter (Validate
    parameter).
    rtn=GT_AxisOn();        error(rtn);        //Activate drive.

    rtn=GT_Axis(2);       error(rtn);        //Set the second axis as current axis.
    rtn=GT_ClrSts();      error(rtn);     //Clear the wrong status of current axis.
    rtn=GT_CtrlMode(0);   error(rtn);        //Set to output analog amount.
    rtn=GT_OpenLp();    error(rtn);        //Set as close loop control.
    rtn=GT_AxisOn();      error(rtn);        //Activate drive.

    rtn=GT_Axis(3);       error(rtn);        //Set the third axis as current axis.
    rtn=GT_ClrSts();      error(rtn);     //Clear the wrong status of current axis.
    rtn=GT_CtrlMode(1);   error(rtn);        //Set to output analog amount.
    rtn=GT_StepDir();     error(rtn);         //Set to output pulse + direction
    signal.
    rtn=GT_AxisOn();      error(rtn);        //Activate drive.

    rtn=GT_Axis(4);       error(rtn);        //Set the fourth axis as current axis.
    rtn=GT_ClrSts();      error(rtn);     //Clear the wrong status of current axis.
    rtn=GT_CtrlMode(1);   error(rtn);        //Set to output analog amount.
    rtn=GT_StepPulse();   error(rtn);    //Set to output positive and negative
pulse signal.
    rtn=GT_AxisOn();      error(rtn);        //Activate drive.
  }
```

```
void main()
{
  GTInitial();
 InputCfg();
  AxisInitial();

  }
```

3.2.3 Notes of Main Point

Set current axis

The motion controller can control four control axes at the same time, each of them able to set parameters independently. To improve the communication efficiency between the host and controller, the controller applies the strategy of address searching for control axis.

The single-axis commands evoked by user program all act on the ***current axis***. The default current axis is the first axis. To send command to another axis, first evoke the ***set current axis*** command short GT_Axis(unsigned short num).

GT_Axis() sets the axis specified by parameters as the current axis. ***The following single-axis commands evoked are all for the current axis until this function is evoked again to set the current axis as another axis.*** The parameter num indicates the axis number specified, from 1, 2, 3 to 4, representing the first, second, third and fourth axis respectively.

Specify pulse output mode of "Pulse+Direction" or "Positive and Negative Pulse"

For SG, SE, SD cards and the SV card working in pulse output mode, by default the controller outputs "Pulse+Direction" signal. User can evoke the function GT_StepPulse to set the controller to output "Positive and Negative Pulse" signal. Evoking the function GT_StepDir to set the controller to output "Pulse+Direction" signal.

Specify output mode of analog voltage output or pulse output (The following explanation is only for SV card. Other users may skip it.)

The SV controller can output analog amount and also the pulse amount (but not at the same time). The default output is analog amount. User can evoke the function short GT_CtrlMode(int mode) to set the output mode. The parameter Mode = 0 means the analog amount output mode, Mode = 1 means the pulse amount output mode.

When the output is set as pulse amount output, evoke GT_StepDir and GT_StepPulse to set the output mode. The default output is pulse/direction mode.

Set as close cycle/open loop control (only for SV card).

The SV controller has two control modes of ***close loop*** and ***open loop***.

Evoke the command GT_CloseLp() to set the current axis to work in close loop mode.

The controller will send the motion position, velocity and acceleration planned currently into the digital servo filter and compare them with the actual position fed back to get the control output signal. This mode enables to realize accurate position control. The default control mode of the SV controller is close loop control mode.

Evoke the command GT_OpenLp() to set the current axis to work in open loop mode, allowing the host to directly set the control axis output signal of the controller with the command GT_SetMtrCmd(). This mode is mainly used in the motion or calibration driver only requiring torque control. The motion controller cannot realize the accurate position control.

Set digital servo filtering parameters (only for SV cards).

The digital servo filter is used for PC to output signal. The SV controller applies ***PID*** filter, together with the velocity and acceleration feedback, i.e. PID+K_{vff}+K_{aff} filter. By adjusting each parameter, the filter can perform accurate and stable control on most systems. The parameters of the servo filter can be set in the host. Table 3-4 is the parameter definition of digital servo filter. The principle of PID+K_{vff}+K_{aff} filter is illustrated in Fig. 3-1.

Table 3-4 Parameter Definition of Digital Servo Filter

Symbol	Name	Parameter Setting Command	Parameter Value Scope	Default Value
K_p	Percentage gain	GT_SetKp()	0 to 32,767	0
K_i	Integral gain	GT_SetKi()	0 to 32,767	0
K_d	Differential gain	GT_SetKd()	0 to 32,767	0
Lim	Error integral gain	GT_SetILmt()	0 to 32,767	32,767
K_{vff}	Velocity feedback gain	GT_SetKvff()	0 to 32,767	0
K_{aff}	Acceleration feedback gain	GT_SetKaff()	0 to 32,767	0
B	Net output difference compensation	GT_SetMtrBias()	-32768 to 32,767	0

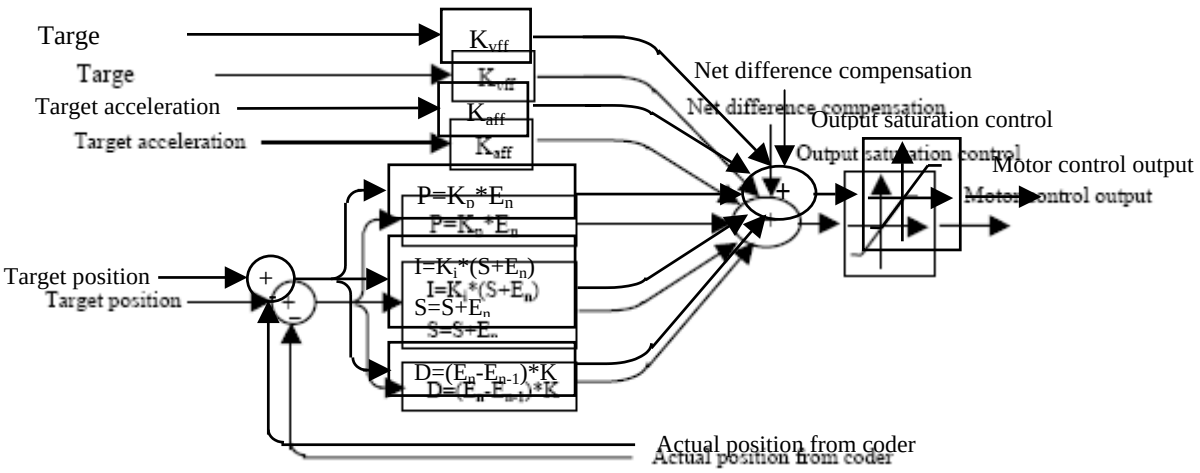

Fig. 3-1 Principle Diagram of Digital Servo Filter

Output calculation formula of digital servo filter:

$$E_n = \left(P_{target}\right)_n - \left(P_{actual}\right)_n$$

$$U_n = E_n K_p + \left(E_n - E_{n-1}\right)K_d + \left(\sum_n E_n\right)K_i / 256 + V_{target}K_{vff} + ACC_{target}K_{aff} + B$$

The meanings of variables are listed in Table 3-5.

Table 3-5 Meanings of Variables

Variable	Meaning
U_n	The output value of digital servo filter
E_n	The position error of the n sampling time
P_{target}	The target position of the n sampling time
P_{actual}	The actual position of the n sampling time
$\left(\sum_n E_n\right)$	The accumulated error value of the n sampling time
V_{target}	The current target velocity with a unit of Count value/Sampling period
ACC_{target}	The current target acceleration with a unit of Count value/Sampling period^2
B	Net difference compensation of motor

Be careful when setting Ki gain at first time. If the system is running and the accumulated value is unknown, setting Ki as a value that is not 0 will cause abrupt "jump". To avoid this situation, it needs to set LIM (Accumulated limit) as 0 and Ki as the expected value. Then set LIM as the expected accumulated limit. Thus, all previous accumulated values will be cleared, and the accumulation will start from the previous point smoothly.

The net output compensation of digital servo filter is mainly used to compensate the influence of outside force at a single direction against the control axis, such as the gravity of the vertical axis of a lathe. The compensation of filter can be set by host

command, with a scope of –32768 to +32767. The default compensation is 0.

The maximum output values of the digital servo filter are $+/-2^{15}$. The corresponding analog amount output is +/-10V. But the actual output value scope is limited by the control saturation value, which (0-32767) decides the valid output scope of the filter. The host can send the command GT_SetMtrLmt() to modify the control saturation value, so as to control the valid output scope of filter. The default control saturation value is 32767.

Chapter Four Independent Axis Motion

4.1 Modes and Parameters of Motion

For Independent Axis motion, The GT provides several modes of motion: ***independent positioning (including S-curve and T-curve), independent jogging*** and ***electronic gearing***.

The specified axes will keep their original modes until another mode is valid. Except electronic gearing, the user can specify another mode of the axis correctly just after current motion of this axis is end. The motion between the specified axes is independent, and each axis follows its own profile. The description in details below will guide you to the appropriate mode and parameters of motion.

4.1.1 Independent Positioning: S-curve

4.1.1.1 Command Summary

Table 4-1 Command Summary of S-curve

Command	Description
GT_PrflS()	Specifies motion profile of current axis as S-curve
GT_SetJerk()	Specifies the jerk of current axis
GT_SetMAcc()	Specifies the maximum acceleration of current axis.
GT_SetVel()	Specifies the velocity of current axis.
GT_SetPos()	Specifies the absolute position of current axis.

Table 4-2 Span of the Parameters (ST: Sample Time)

Parameter	Span	Unit
Target Position	-1,073,741,824 ~+1,073,741,823	Pulse
Jerk	0~0.5（without 0.5）	Pulse/ST^3
Max. Acceleration	0~0.5（without 0.5）	Pulse/ST^2
Max. Velocity	0~16384	Pulse/ST

4.1.1.2 Example

Example 4-1 S-curve Profile Motion

```
void SMotion()                          //Motion function of S-curve profile
{
short rtn;
```

```
  rtn=GT_PrflS();       error(rtn);  // Specifies motion profile of S-curve for
current axis
  rtn=GT_SetJerk(0.00000002); error(rtn);  /* Specifies jerk of 0.00000002.
                                                */
  rtn=GT_SetMAcc(0.004); error(rtn);      // Specifies max. acceleration of
                                             0.004.
  rtn=GT_SetVel(4);      error(rtn);      // Specifies velocity of 4.
  rtn=GT_SetPos(80000);  error(rtn);      // Specifies position of 80000.
  rtn=GT_Update();       error(rtn);      // Update parameters.
  }

  void main()
  {
    GTInitial();
   InputCfg();
    AxisInitial();
    SMotion();
  }
```

4.1.1.3 Notes of Main Point

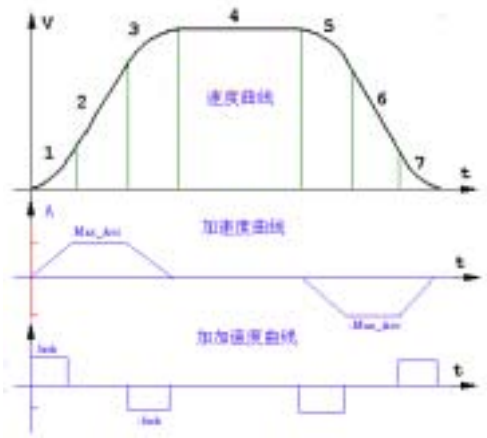

Fig4-1

Fig. 4-1 shows the velocity, acceleration and jerk profile of a typical S-curve. The process of motion is described as below.

- In the first stage, Specified axes increase the acceleration from 0 to the max. acceleration by the Jerk ratio.
- In the second stage, the jerk is zero. Specified axes Accelerate by the max. acceleration.
- In the third stage, Specified axes decrease the acceleration to zero by jerk, so that the specified velocity is reached..
- In the fourth stage, Specified axes move at this constant Specified velocity both of the acceleration and jerk are zero.
- The fifth, sixth and seventh stages are similar with the first, second and third

stages. In these stages, Specified axes decelerate to zero.

In S-curve, the user can modify the target position at any time, other parameters cannot be modified during motion. All the valued of the target velocity, maximum acceleration and jerk are positive. The motion direction of the specified axis is decided by the commanded position. Normally S-curve is symmetrical, but it is allowed to be lack of some stage. For example, the S-curve in *Fig. 4-2* is lack of the fourth stage.

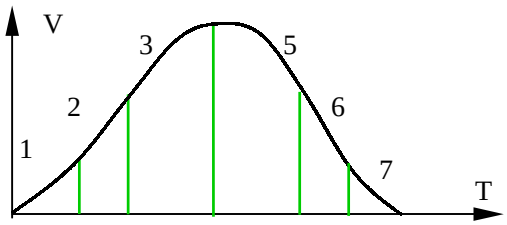

Fig. 4-2 Deformed S-curve

4.1.2 Independent Positioning: T-curve

4.1.2.1 Command Summary

Table 4-3 Command Summary of T-curve ~~Mode~~

Function	Description
GT_PrflT()	Specifies motion profile of current axis as T-curve
GT_SetAcc()	Specifies the acceleration of current axis.
GT_SetVel()	Specifies the velocity of current axis.
GT_SetPos()	Specifies the absolute position of current axis.

Table 4-4 Span of the Parameters (ST: Sample Time)

Parameter	Span	Unit
Acceleration	0~16384	Pulse/ST ^{2}
Velocity	0~16384	Pulse/ST
Absolute Position	-1,073,741,824 ~1,073,741,823	Pulse

4.1.2.2 Example

Example 4-2 T-curve Profile Motion

```
void TMotion()                          //Motion function of T-curve profile
{
   short rtn;
   rtn=GT_PrflT();       error(rtn); // Specifies motion profile of T-curve for
current axis
   rtn=GT_SetAcc(0.01); error(rtn); //Specifies the ~~max.~~ acceleration of 0.01.
```

```
  rtn=GT_SetVel(1);     error(rtn); //Specifies target velocity of 1.
  rtn=GT_SetPos(80000);error(rtn); // Specifies absolute position of 80000.
  rtn=GT_Update();      error(rtn); //Update parameter.
}

void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  TMotion();
}
```

4.1.2.3 Notes of Main Point

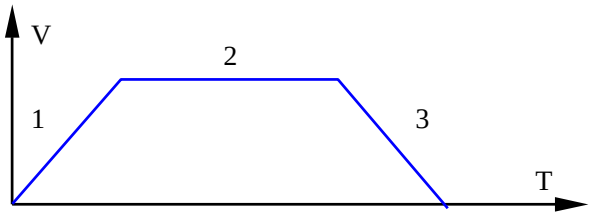

Fig. 4-3 Velocity Profile of T-curve

Fig. 4-3 describes the velocity profile of T-curve. A typical velocity profile of T-curve is described as below.

- The first stage: The specified axes accelerate from zero to the commanded velocity by the acceleration.
- The second stage: The specified axes move at this constant velocity.
- The third stage: The specified axes decelerate such that the final position agree with the command position..

In some situations, The specified axes may decelerate without the second stage. In T-curve, the commanded velocity and position can be changed at any time. The velocity profile is ~~as~~ illustrated in *Fig. 4-4*.

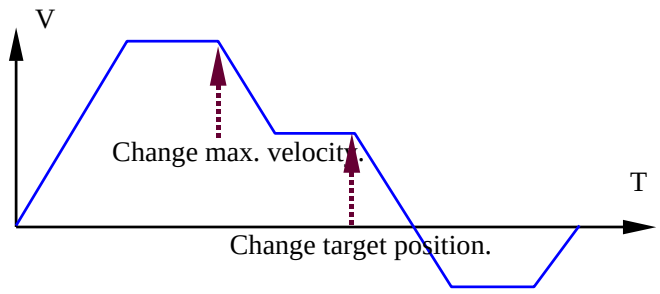

Table 4-4 Velocity Profile in T-curve after Changing Velocity and Position Changed

4.1.3 independent jogging

4.1.3.1 Command Summary

Table 4-5 Command Summary of independent jogging

Function	Description
GT_PrflV()	Set the motion mode of current axis as velocity control mode. Specifies motion mode of current axis as independent jogging
GT_SetAcc()	Specifies the acceleration of current axis.
GT_SetVel()	Specifies the maximum velocity of current axis.

Table 4-6 Span of the Parameters (ST: Sample Time)

Parameter	Scope	Unit
Velocity	-16384~16384	Pulse/ST
Acceleration	0~16383	Pulse/ST ^{2}

4.1.3.2 Example

Example 4-3 Jog Mode of Motion

```
void VMotion()                    //Motion function of independent jogging mode
{
   short rtn;
   rtn=GT_PrflV();       error(rtn); // Specifies the jog mode of motion for
current axis
   rtn=GT_SetAcc(0.01); error(rtn); //Specifies the acceleration of 0.01.
   rtn=GT_SetVel(1);     error(rtn); // Specifies the target velocity of 1.
   rtn=GT_Update();      error(rtn); //Update parameter.
}

void main()
{
   GTInitial();
  InputCfg();
   AxisInitial();
   VMotion();
}
```

4.1.3.3 Notes of Main Point

The jog mode of motion is flexible because velocity, acceleration and direction can be changed during motion. The user specifies the target velocity and acceleration for current axis, the direction of motion is specified by the sign of the command velocity. In this mode, when motion begins, the specified axis will accelerate up to the commanded velocity and continue to move at this velocity until a new velocity or stop command is issued. *Table 4-5* lists the command summary for independent jogging mode of motion.

Table 4-6 lists the span of the parameters in the command.

4.1.4 Electronic Gearing

4.1.4.1 Command Summary

Table 4-7 Command Summary of Gearing

Command	Description
GT_PrflG()	Set the motion mode of current axis as the electronic gear control mode. Specifies motion mode of current axis as electronic gearing, and specifies which axis is the master axis.
GT_SetRatio()	Specifies the gear ratio of current axis.

Table 4-8 Span of the Parameters in Electronic Gearing (ST: Sample Time)

Parameter	Span	Unit
Drive Axis Number	1～6	The axes 1 to 4 are controlled axes. The axes 5 and 6 are auxiliary encoders.
Electronic Gear Ratio	-16384～16384	Positive value means the same direction with the master axis. Negative value means the opposite direction against the master axis.

4.1.4.2 Example

Example 4-4 Electronic Gearing Mode of Motion

```
void GMotion()                  //Motion function of electronic gearing mode of motion.
{
  short rtn;
  rtn=GT_Axis(1);       error(rtn); // Specifies axis #1 of current axis
  rtn=GT_PrflG(2);      error(rtn); /* Specifies electronic gearing mode of motion for current axis, and specifies the master axis of axis #2 */
  rtn=GT_SetRatio(-1); error(rtn); //Specifies the gear ratio of –1.
  rtn=GT_Update();      error(rtn); //Update parameter.
}

void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  GMotion();
}
```

4.1.4.3 Notes of Main Point

When Calling GT_PrflG(), the controller specifies the current axis of electronic gearing

mode (i.e. the current axis is a slave axis.), and specifies another axis or auxiliary encoder of the master axis.

The slave axis will follow the motion of the master axis at the specified gear ratio. The mode of motion for the master axis can be any sort of mode. (If the master axis is in the mode of interpolation motion, the slave axis will follow the motion of the master axis, not the vector motion of the interpolation). The incremental position of current axis is in proportion to the incremental position of the master axis by gear ratio.

In fact, the electronic gearing mode is coordinate motion of several axes. The effect of the motion is similar with the joggle motion of two mechanical gears.

The command GT_SetRatio() is to specifies the gear ratio. The controller allows one master axis with several slave axes, or an axis as a drive axis to drive secondary axis. If there were any abnormal status happened in the slave axis (such as limit switch action, drive alarming, etc.), the master axis would stop motion. But if the master axis is in open loop mode or an auxiliary encoder, the motion of the master axis will not be stopped under the abnormal situation of the slave axis.

 Notice	*When the current axis is in the electronic gearing mode, no matter the master axis is in motion or not, the status of the current axis will always show that the current axis is in motion (i.e. the BIT10 of the current axis status is always set.) until the user changes the current axis to another mode of motion, or there are some abnormal situation happened in the current axis.*

4.2 Stop Motion

Sometimes for the purpose of safety or having some specific motion profile, it is required to stop the motion of current axis at special position or time. For the independent axis motion, the controller provides two ways for stopping: ***Abrupt Stop*** and ***Smooth Stop***.

4.2.1 Command Summary

Table 4-9 Stopping Command Summary of Indepent Axis Motion

Command	Description
GT_SmthStp()	Stop the motion of current axis smoothly.
GT_AbptStp()	Stop the motion of current axis abruptly.

4.2.2 Example

Example 4-5 Stop the motion of the first axis abruptly when the EXI15 input port of external IO is at high level.

```
void main()
{
    short rtn,ex_data;
    rtn=GT_ExInpt(&ex_data);    error(rtn); //Read the status of input port.
    rtn=GT_Axis(1);             error(rtn); //Specify the first axis as the current axis.
     if(ex_data&0x8000)                     //Check if EXI15 is at high level.
    {
        rtn=GT_AbptStp();       error(rtn); //Stop motion abruptly.
        }
}
```

4.2.3 Notes of Main Point

Command of Abrupt Stop

The command GT_AbptStp() stop the motion of current axis abruptly, and specifies the target velocity and real running velocity of 0 without deceleration..

The command GT_AbptStp() can be called in all modes of independent axis motion.

注意 Notice	*Be careful in using the command GT_AbptStp(). Because this command will stop the motor abruptly, generate large impact to the motor, and reduce the service life of the motor and system.*

Command of Smooth Stop

The command GT_SmthStp() stop the motion of current axis smoothly, i.e. decelerates the velocity to zero by commanded acceleration. The deceleration and acceleration stages are symmetrical. *Diagram 4-5* shows the velocity profile of smooth stop in S-curve and T-curve.

The command GT_SmthStp() will not be valid until the user calls the command GT_Update() or GT_MltiUpdt(). GT_SmthStp() can be used in several modes of motion but electronic gearing.

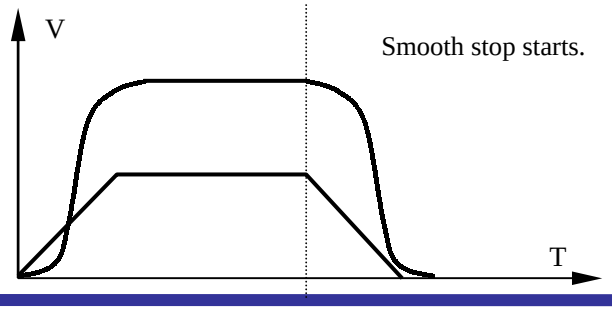

Diagram 4-5 Velocity Profile of Smooth Stop in S-curve and T-curve Modes

4.3 Specify and Update Parameters of Specified Axis

Parameter Updating is used in *3.3 Initialization of Motion Control Axis*, *4.1 Mode and Parameter of motion* and *4.2 Stop Motion*. Parameter specifying and updating will be described in details below. Parameter updating includes normal updating and self-updating at breakpoint.

To update several parameters of current axis or parameters of several axes in synchronism, the motion controller uses ***double-buffered mechanism*** to realize the specifying and updating of specified axes.

The double-buffered mechanism means, when the host sends the command to specify the motion and control parameters of independent axis, the command will be downloaded into the motion controller. Before ***parameter updating***, parameters and motion commands are all waiting for validation. After ***parameter updating***, the controller will copy these parameters and commands into the valid registers in the next sample time, and making them valid at the same time. The commands in command summary *Table 4-10* is all suitable for double-buffered mechanism.

Table 4-10 Command Summary of Double-buffered Commands

Function	Description
GT_SetPos()	Set the target position of current axis.
GT_SetBrkcn()	Set the comparison value of breakpoint position of current axis.
GT_SetVel()	Set the target velocity of current axis.
GT_SetAcc()	Set the acceleration of current axis.
GT_SetMAcc()	Set the maximum acceleration of current axis.
GT_SetJerk()	Set the jerk of current axis.
GT_SetRatio()	Set the electronic gear transmission ratio of current axis.
GT_SetMtrLmt()	Set the servo filter output limit of current axis.
GT_SetMtrBias()	Set the servo filter output zero point bias value of current axis.
GT_SetKp()	Set the servo filter percentage gain of current axis.
GT_SetKi()	Set the servo filter integral gain of current axis.
GT_SetKd()	Set the servo filter differential gain of current axis.
GT_SetKvff()	Set the servo filter velocity feedback gain of current axis.

Function	Description
GT_SetKaff()	Set the servo filter acceleration feedback gain of current axis.
GT_SetILmt()	Set the servo filter error differential limit of current axis.
GT_SetPosErr()	Set the servo filter position error limit of current axis.
GT_SmthStp()	Stop the motion of current axis smoothly.
GT_SynchPos()	Synchronize the actual position and target position of current axis.

4.3.1 Normal Updating

4.3.1.1 Command Summary

Table 4-11 Command Summary of Normal Parameter Updating

Function	Description
GT_Update()	Update parameter.
GT_MltiUpdt()	Update multiple-axis parameter.

4.3.1.2 Example

Example 4-6 Normal Parameters Updating

```
void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();

  rtn=GT_Axis(1);  error(rtn);      //Specify the axis #1 of current axis.
  VMotion();                        //Specify the mode of motion as independent jogging.
  delay(1000);                      //Delay by one second.
  rtn=GT_SmthStp(); error(rtn);     //Smooth stop.
  rtn=GT_Update();  error(rtn);     //Update parameter (valid command of smooth stop.).

  rtn=GT_Axis(2);   error(rtn);     //  Specify the axis #2 of current axis.
  rtn=GT_SetAcc(0.01); error(rtn);  //Acceleration = 0.01 pulse/Control period2
  rtn=GT_SetVel(1);    error(rtn);  //Velocity = 1 pulse/Control period
  rtn=GT_SetPos(20000); error(rtn); //20000 pulse in target position.

  rtn=GT_Axis(3);   error(rtn);     //  Specify the axis #3 of current axis.
  rtn=GT_SetAcc(0.01); error(rtn);  //Acceleration = 0.01 pulse/Control period2
  rtn=GT_SetVel(1);    error(rtn);  //Velocity = 1 pulse/Control period
  rtn=GT_SetPos(20000); error(rtn); //20000 pulse in target position.

  rtn=GT_MltiUpdt(0x6); error(rtn);  /* Updating the parameters of
```

```
                                                    several axes (The second and third
                                                    axes parameters become valid at
                                                    the same time.) */
}
```

4.3.1.3 Notes of Main Point

The command GT_Update() updates the parameters of current axis immediately.

The command GT_MltiUpdt() updates the parameters of multi-axes immediately.

4.3.2 Self-Updating at Breakpoint

Except normal updating, the motion controller also provides self-updating at breakpoint to update parameters automatically. The host can specify a condition (called ***breakpoint***) that, when the motion of specified axis satisfies this condition, the controller will automatically update parameters (commands) of this axis. The command related to the self-updating at breakpoint is listed in *Table 4-12*.

Table 4-12 Command Summary of Self-updating at Breakpoint

Command	Description
GT_AuUpdtOn()	Validate the self-updating parameters of current axis.
GT_AuUpdtOff()	Invalidate the self-updating parameters of current axis.
GT_GetBrkCn()	Get the breakpoint position of current axis.
GT_PosBrk()	Specify the breakpoint triggering condition of current axis as positive position breakpoint.
GT_NegBrk()	Specify the breakpoint triggering condition of current axis as negative position breakpoint.
GT_ExtBrk()	Specify the breakpoint triggering condition of current axis as the home signal triggering breakpoint.
GT_MtnBrk()	Specify the breakpoint triggering condition of current axis as the motion completion event breakpoint.
GT_BrkOff()	Clear the breakpoint of current axis, and invalidate breakpoint condition.

The default status of self-updating at breakpoint is invalid. The host can use the commands GT_AuUpdtOn() and GT_AuUpdtOff() to validate and invalidate this function. The user can tell if this function is valid or invalid by calling GT_GetMode() and read relative bit (For details, see *4.5.2*).

After the breakpoint of specified axis is triggered, the controller will clear it. The user must reset the breakpoint condition again if he needs to trigger breakpoint again. Meanwhile, after specifying breakpoint condition, the host can cancel it by calling the command GT_BrkOff() before it is triggered.

The motion controller provides four kinds of breakpoints, the positive position breakpoint, negative position breakpoint, axis motion completed breakpoint, and home switch triggered breakpoint. The following will describe in details the usage of self-updating at four kinds of breakpoint parameters.

4.3.2.1 Positive Position Breakpoint

At first, the user calls the command GT_SetBrkCn() to specify the breakpoint position of current axis, and uses the command GT_Update() or GT_MltiUpdt() to validate it. Then, the user calls the command GT_PosBrk() to specify the breakpoint triggering condition of current axis of the positive position breakpoint. When the actual position of the current axis is larger than or equal to the breakpoint position, the breakpoint will be triggered and the sign indicating the mode of breakpoint triggering of current axis will be cleared, then the controller will search the sign of self-updating parameters in this axis's mode register. If the sign bit = 1, the new parameter for this axis will be updated automatically. no matter the self-updating parameters is allowed or not, the sign of breakpoint triggered will be set by the controller。

Example 4-7 Specifying and Triggering the Positive Position Breakpoint

In this example, in the process of moving the axis #1 to the position 50000, the controller will trigger the breakpoint at the position 20000, and change the valid velocity from 1 (Pulse/ST) to 4 (Pulse/ST).

```
void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();

  rtn=GT_Axis(1);       error(rtn);  //Specify the axis #1 of the current axis.
  rtn=GT_ClrSts();      error(rtn);  //Clear status.
  rtn=GT_SetVel(1);     error(rtn);  //Specify velocity of 1 (Pulse/ST).
  rtn=GT_SetAcc(0.1);     error(rtn);     //Specify acceleration of 0.1 (Pulse/ST^2)
  rtn=GT_SetPos(50000); error(rtn);  //Specify target position of 50000.
  rtn=GT_AuUpdtOn();       error(rtn);       //Validate the function of self-updating parameters at breakpoint position.
  rtn=GT_SetBrkCn(20000); error(rtn);  //Specify breakpoint position.
  rtn=GT_Update();        error(rtn);       //Update parameter (Breakpoint position becomes effective.).
  rtn=GT_PosBrk();      error(rtn);     //Specify the breakpoint triggering condition of positive position breakpoint.
  rtn=GT_SetVel(4);       error(rtn);  //Specify velocity of 4 (Wait for
```

self-updating.).
}

4.3.2.2 Negative Position Breakpoint

The user calls the command GT_SetBrkCn() to specify the breakpoint position of current axis at first, and uses the command GT_Update() or GT_MltiUpdt() to validate it. Then, the user calls the command GT_NegBrk() to specify the breakpoint triggering condition of current axis of the negative position breakpoint. When the actual position of the current axis is smaller than or equal to the breakpoint position, the breakpoint will be triggered and the sign indicating the mode of breakpoint triggering of current axis will be cleared, then the controller will search the sign of self-updating parameters in this axis's mode register. If the sign bit = 1, the new parameter of this axis will be updated automatically. no matter the self-updating of parameters is allowed or not, the sign of breakpoint triggered will be set by the controller.

Example 4-7 Specifying and Triggering the Negative Position Breakpoint

In this example, in the process of moving the axis #1 from 0 to the position -50000, the controller will trigger the breakpoint at the position -20000, and change the valid velocity from 1 (Pulse/ST) to 4 (Pulse/ST).

```
void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();

  rtn=GT_Axis(1);       error(rtn);  // Specify the axis #1 of the current axis.
  rtn=GT_ClrSts();      error(rtn);  //Clear status.
  rtn=GT_SetVel(1);     error(rtn);  //Specify velocity of 1 (Pulse/ST).
  rtn=GT_SetAcc（0.1）; error(rtn);  //Specify acceleration of 0.1 (Pulse/ST^2)
  rtn=GT_SetPos(-50000); error(rtn); //Specify target position of -50000.
  rtn=GT_AuUpdtOn();      error(rtn);       // Validate the function of self-updating parameters at breakpoint position.
  rtn=GT_SetBrkCn(-20000); error(rtn);  //Specify breakpoint position.
  rtn=GT_Update();        error(rtn);       //Update parameter (Breakpoint position becomes effective.).
  rtn=GT_NegBrk();      error(rtn);     // Specify the breakpoint triggering condition of positive position breakpoint.
  rtn=GT_SetVel(4);       error(rtn);  //  Specify velocity of 4 (Wait for self-updating.).
}
```

4.3.2.3 Axis Motion Completion Event Breakpoint

The user calls the command GT_MtnBrk() to specify the current axis breakpoint triggering condition of motion completion event breakpoint. When the motion of the current axis is completed and the sign of motion completion in the status register of current axis is 1, the breakpoint will be triggered and the sign indicating the mode of

breakpoint triggering of current axis will be cleared, then the controller will search the sign of self-updating parameters in the current axis's mode register. If the sign bit = 1, the new parameter of this axis will be updated automatically. no matter the self-updating parameters is allowed or not, the sign of breakpoint triggered will be set by the controller。

Example 4-9 Specifying and Triggering the Motion Completion Event Breakpoint

In this example, after the motion of axis #1 reaching the position 20000 at a velocity of 1 (Pulse/ST), the controller will self-updating the velocity as 4 (Pulse/ST) and the position as 0.

```
void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();

  rtn=GT_Axis(1);       error(rtn);  //  Specify the axis #1 of current axis as the first axis.
  rtn=GT_ClrSts();      error(rtn);  //Clear status.
  rtn=GT_SetVel(1);     error(rtn);  //Specify velocity of 1 (Pulse/ST).
  rtn=GT_SetAcc(0.1);   error（rtn）; //Specify acceleration of 0.1 (Pulse/ST^2)
  rtn=GT_SetPos(20000); error(rtn);  //Specify target position of 20000.
  rtn=GT_AuUpdtOn();       error(rtn);       //Validate the function of self-updating parameters at breakpoint position.
  rtn=GT_MtnBrk();   error(rtn);     //Specify the breakpoint triggering condition ofmotion completion event breakpoint.
  rtn=GT_SetVel(4);       error(rtn);  //Specify velocity of 4 (Wait for self-updating.).
  rtn=GT_SetPos(0);      error(rtn);  //Specify target position of 0 (Wait for self-updating.).
}
```

4.3.2.4 Home Signal Triggering Event Breakpoint

The user calls the command GT_MtnBrk() to specify the current axis breakpoint triggering condition of home signal triggering event breakpoint. After the host calls the home signal capture command and when the sign of Index/Home captured in the axis motion status register is 1, the breakpoint will be triggered and the sign indicating the mode of breakpoint triggering of current axis will be cleared, then the controller will search the sign of allowing self-updating parameters in the current axis mode register. If the sign bit = 1, the new parameter of this axis will be updated automatically. no matter the self-updating of parameters is allowed or not, the sign of breakpoint triggered will be set by the controller。

Example 4-9 Specifying and Triggering the Home Signal Triggering Event Breakpoint

This program will enable axis #1 to capture the home signal in the motion. After the

home signal of current axis is captured, the breakpoint will be triggered and the smooth stop command of this axis will be validated automatically, and make the current axis stop smoothly. Meanwhile, the home position captured will be saved in the position captured register.

```
void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  rtn=GT_Axis(1);       error(rtn);  // Specify the axis #1 of the current axis.
  rtn=GT_ClrSts();      error(rtn);  //Clear status.
  rtn=GT_AuUpdtOn();    error(rtn); //Validate the function of self-updating parameters at breakpoint position.
  rtn=GT_CaptHome();    error(rtn);     //Specify the capture mode of home signal capture.
  rtn=GT_ExtBrk();      error(rtn);    //Specify the home signal triggering event breakpoint.
  rtn=GT_PrflV();         error(rtn);       //Specify the motion mode of independent jogging.
  rtn=GT_SetVel(1);     error(rtn);      //Specify velocity of 1 (Pulse/ST).
  rtn=GT_Update();      error(rtn);      //Update parameter.
  rtn=GT_SmthStp();     error(rtn);   //Smooth stop (Wait for self-updating.).
  }
```

4.4 Specifying Target Position and Actual Position for Specified Axis

4.4.1 Command Summary

Table 4-13 Command Summary of Setting Target Position and Actual Position for Specified Axis

Function	Description
GT_SetPos()	Set the target position of current axis.
GT_ZeroPos()	Reset the actual and target position of current axis to zero.
GT_SynchPos()	Set the target position of current axis same as the actual position.
GT_SetAtlPos()	Set the actual position of current axis.

4.4.2 Notes of Main Point

Specify the target position of current axis

The user can call the command GT_SetPos (long Pos) to specify the target position of current axis in S-curve and T-curve.

The value of parameter Pos is between –1073741824 and 1073741823. The unit is ***pulse***. The parameters value specified by this command can be valid only by calling the commands GT_Update() or GT_MltiUpdt().

Specify actual and target position of zero.

The command GT_ZeroPos() can be called to specify the actual position, target position and the profiled position of current axis of zero. This command is invalid when current axis is in motion. So this command is invalid when current axis is in the electronic gearing mode.

Specify the target position of current axis is equal to the actual position of current axis.

The command GT_SynchPos(void) can be called to specify the target position and the profiled position of current axis is equal to the actual position. This command is applicable when current axis is in S-curve and T-curve. When the motion encounters error and needs to be restarted, the user can use this command to correct the error. Meanwhile, the user can use this command to smooth the changing of the motor when current axis is being activated (inactivated or) or in closed loop (in open loop). This command needs to be valid by the command GT_Update() or GT_MltiUpdt().This function is invalid when current axis is in electronic gearing.

Specify the actual position of current axis.

The command GT_SetAtlPos (long actl_pos) can be used to modify the actual position, target position and the profiled position of current axis to a specified value (actl_pos). This command is invalid when current axis is in motion. This command is invalid when current axis is in electronic gearing mode.

4.5 Axis Status

4.5.1 Axis Status Register

The motion controller provides a 16-bit status register for each axis. The user can use command GT_GetSts() to get the status of current axis.

4.5.1.1 Definition of Bit in Register

Table 4-14 Definition of Bit in Axis Status Register

Bit	Definition
0	Sign of Motion completion. If the motion of current axis is completed, this bit will be 1. This sign is invalid in electronic gearing mode.
1	Alarm signal of Motor driver. If the driver of current axis gives alarm signal, this bit will be 1.
2	Sign of Breakpoint arrival. If the specified breakpoint condition is satisfied, this bit will be 1.
3	Sign of Index/Home capture. After specifying the position capture function, when the controller detects the specified Index/Home capture happened, this bit will be 1.
4	Sign of Motion error. If position error overreaches the error band (Refer to

Bit	Definition
	4.4.7.3 Description), this bit will be 1. Only when current axis is not in motion error status, this sign can be reset.
5	Sign of Positive limit switch triggering. If the positive limit switch is triggered, this bit will be 1.
6	Sign of Negative limit switch triggering. If the negative limit switch is triggered, this bit will be 1.
7	Sign of Command error. If last operating command (not reading command) is invalid, this bit will be 1.
8	Motor is in open loop/close loop (1 means close loop and 0 means open loop.).
9	Motor is activated/ inactivated (1 means activated and 0 means inactivated).
10	Current axis is in motion or not. If current axis is in motion, the bit will be 1. Otherwise, it will be 0.
11	Limit switch status detecting is permitted /prohibited (1 means permitted and 0 means prohibited).
12 13	Current axis number (13 bit = high bit, 12 bit = low bit). The encode table of current axis number is as follows. Bit 13 Bit 12 Axis 0 0 1 0 1 2 1 0 3 1 1 4
14	Sign of Specifying Home switch signal capture.
15	Sign of Specifying Index signal capture.

4.5.1.2 Description

The definition of each bit of status register is listed in *Table 4-14*. The bit 8-15 indicates the motion status and current axis number, which cannot be reset by the user. The bit 0-7 indicates different event status of current axis. Once these events happen, the corresponding bit will be set as 1 and kept until the user call the command GT_ClrSts() and GT_RstSts() to clear all bits or the corresponding bit. In addition, the bit 0-6 can trigger the interrupt request to the host.

To simplify the programming of motion, there are two bits to indicate the motion status in the status register. One represents whether current axis motion completion event happens or not (bit 0). The other represents the current motion status of current axis (bit 10.).

Both of the host and motion controller can modify the sign of motion completion. After the motion of specified axis is completed, the controller sets the sign of motion completion. Then the host can check this bit to tell the motion is completed or not. If needed, the user can also program in the host to let the motion controller trigger the interrupt request to the host when the motion of specified axis is completed. In any one of the methods above, once the host tell the motion of specified axis is end by the sign, it should clear this sign immediately, such that the motion completion sign of next motion can be set correctly. The controller may set the completion sign in the following situations.

- Motion arrives.
- In independent jogging mode, both of the actual velocity and commanded velocity of specified axis are 0.
- After the command GT_SmthStp() becomes valid and the actual velocity of specified axes reaches 0.
- After sending the command GT_AbptStp().
- When the limit switch status is being detected, and the limit switch trigger.
- In the mode of allowing auto stop when position error overreach error band, and this situation has happened.
- The driver of specified axis alarms.
- Call the command GT_AxisOn() to activate the driver of current axis.

The sign of axis in motion is similar with the sign of motion completion. The difference is that the former indicates the motion status of specified axis and can be checked by the host at any time, but cannot trigger interrupt request and modified by the host.

The sign of axis in motion and the sign of motion completion only represent the status of profiled motion in the controller, not the status of actual motion. That is, they only represent whether the motion planning completes. Whether the actual position of specified axes agree with the target position depends on the lag of control, the stability and other conditions of the whole system.

The sign of motion completion is invalid in the electronic gearing mode. And the sign of axis in motion is always set after specified axis enters the electronic gearing mode, no matter whether the specified axis is in motion or not.

4.5.2 Axis Mode Register

The motion controller provides a mode register to represent the working modes. The command GT_GetMode() can be used to get the value of the register. The definition of bits is listed in *Table 4-15*.

Table 4-15 Definition of Axis Mode Register

Bit	Definition
0-6	Reserved.
7	Sign of allowing auto stop when position error overreach error band. The commands GT_AuStpOn() and GT_AuStpOff() can modify this sign. The bit = 1 means that, when position error of specified axis overreach error band, the controller will auto stop this axis and inactivate the driver of this axis
8-9	Reserved.
10	Sign of self-updating. The commands GT_AuUpdtOn() and GT_AuUpdtOff() can modify this sign. The bit = 1 means that, the motion controller will auto update the specified axis parameters when the breakpoint condition is satisfied.
11-13	The coding of specified axis motion mode: Bit13 Bit12 Bit11 Mode of Motion 0 0 0 T-curve of independent positioning

Bit				Definition
	0	0	1	Independent jogging
	0	1	0	S-curve of independent positioning
	0	1	1	Electronic gearing
	1	0	1	Coordinated motion
14-15	Reserved			

Chapter Five Coordinated Motion(The series of GT_PX do not contain)

The motion controller can implement the coordinated motion with two kinds of path, linear interpolation and circular interpolation. The simplest way to describe complicated coordinated motion path is to use the coordinate system, where it is easy to describe the motion path of motion object. .

The motion controller changes the motion mode of specified axes from independent axis motion to the coordinated motion by ***coordinate mapping***. In the coordinated motion mode, the controller can implement ***single-segment path motion*** and ***multi-segment continuous path motion***. As to multi-segment continuous path motion, the motion controller supplies a ***data buffer*** to realize high speed and stability continuous motion.

5.1 Coordinate Mapping

The motion controller uses a 4D coordinate system (X-Y-Z-A) to describe the linear and circular interpolation path. When the circular interpolation command is applied, the three axes X-Y-Z will form a right-hand coordinate system as illustrated in *Fig. 5-1*. User can also use 2D (X-Y) and 3D (X-Y-Z) coordinate system to describe the motion path.

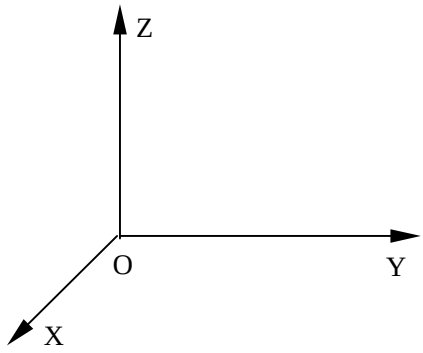

Fig. 5-1 Right-hand Coordinate System

User can use the command GT_MapAxis() to map the motion described in a coordinate system to the corresponding axes through mapping relationship, so as to establish a kinematics transferring relationship between the motion of each axis and the required motion path in the coordinate system. Through mapping relationship, the motion between the axes is coordinate to maintain the specified vector speed,

acceleration and deceleration along with the specified path. The command GT_MapAxis() can be called to specified axis unless this axis is not in motion.

The prototype of coordinate mapping command:

short GT_MapAxis(short Axis_Num, double * map_count)

Axis_Num describes the number of specified axis (1, 2, 3 or 4). After the coordinate mapping command is called, this axis will work in coordinated motion mode. The actual position of this axis is marked as *Axis_N*, with a unit of pulse. The array map_count includes five elements, marked in order as C_x, C_y, C_z, C_a and C. The coordinates corresponding to the coordinate axes of X, Y, Z and A are marked as x, y, z and a. The mapping relationship described by the command above can be simply described with the following formula:

$$Axis\_N = C_x \times x + C_y \times y + C_z \times z + C_a \times a + C$$

From this formula, we can see that the motion of specified axis mapped is a linear combination of coordinate X, Y, Z and A.

Example 5-1 The Simplest Coordinate Mapping

This example can realize the simplest coordinate mapping:
Axis 1 = X
Axis 2 = Y
Axis 3 = Z
Axis 4 = A

```
void MapAxis()                          //Coordinate mapping function
{
    short rtn;
    double cnt1[5]={1,0,0,0,0};   /* Set the coordinate mapping array
  according to the system. */
    double cnt2[5]={0,1,0,0,0};   /* Set the coordinate mapping array
   according to the system.  */
    double cnt3[5]={0,0,1,0,0};   /* Set the coordinate mapping array
   according to the system.  */
    double cnt4[5]={0,0,0,1,0};   /* Set the coordinate mapping array
   according to the system.  */
    rtn=GT_MapAxis(1,cnt1);  error(rtn);  /* Map axis 1 to axis X. */
    rtn=GT_MapAxis(2,cnt2);  error(rtn);  /* Map axis 2 to axis Y.  */
    rtn=GT_MapAxis(3,cnt3);  error(rtn);  /* Map axis 3 to axis Z.  */
    rtn=GT_MapAxis(4,cnt4);  error(rtn);  /* Map axis 4 to axis A.  */
}

void main()
```

```
{
  GTInitial();
  InputCfg();
  AxisInitial();
  MapAxis();
  }
```

Example 5-2 Coordinate Mapping of Unit Conversion

This example describes the coordinate mapping of a unit conversion. Suppose thepulse count per rotation of the motor is 8000, and the pitch of the lead screw is 4mm. the several motor axes form an orthogonal coordinate system. Then the coordinate mapping relationship can be described as below:

```
Axis 1 = 2000 X
Axis 2 = 2000 Y
Axis 3 = 2000 Z
Axis 4 = 2000 A
```

In this case, the unit of each coordinate axis is 1mm. The motion controller realizes automatically the unit conversion.

```
void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  double cnt1[5]={2000,0,0,0,0};  /* Set the coordinate mapping array
   according to the system. */
  double cnt2[5]={0,2000,0,0,0};  /* Set the coordinate mapping array
   according to the system. */
  double cnt3[5]={0,0,2000,0,0};  /* Set the coordinate mapping array
   according to the system. */
  double cnt4[5]={0,0,0,2000,0};  /* Set the coordinate mapping array
   according to the system. */
  rtn=GT_MapAxis(1,cnt1);  error(rtn);/* Map axis 1 to axis X. */
  rtn=GT_MapAxis(2,cnt2);  error(rtn);/* Map axis 2 to axis Y. */
  rtn=GT_MapAxis(3,cnt3);  error(rtn);/* Map axis 3 to axis Z. */
  rtn=GT_MapAxis(4,cnt4);  error(rtn);/* Map axis 4 to axis A. */
  }
```

Example 5-3 Coordinate Mapping of non-orthogonal axes

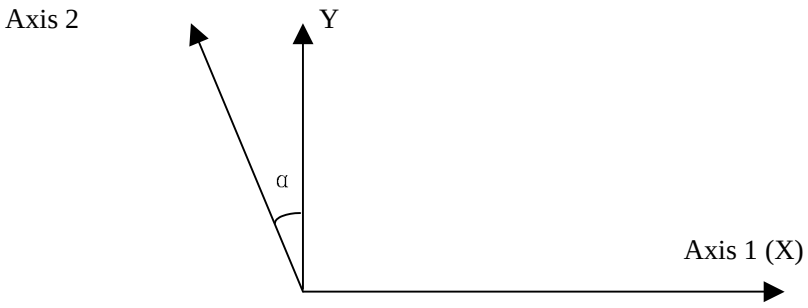

Fig. 5-2 Coordinate Mapping Example

This example describes another application of the coordinate mapping.

As illustrated in Fig. 5-2, if the motion direction of axis 1 is non-orthogonal with the motion direction of axis 2, in order to use an orthogonal coordinate system X-O-Y to describe the motion path, user can establish the following coordinate mapping relationship:

Axis 1 = X + Y tan α

Axis 2 = Y/cos α

With the mapping relationship above, user can describe the motion path in the orthogonal coordinate system easily and directly. This example can be used to compensate the installation error between two vertical guide rails.

```
void MapAxis()                          //Coordinate mapping function
{
    short rtn;
    double cnt1[5]={1,0,0,0,0};
    double cnt2[5]={0,0,0,0,0};
    cnt1[1]=tan(3),cnt2[1]=1/cos(3);    /* The lean angle is 3^0. */
    rtn=GT_MapAxis(1,cnt1);     error(rtn);
    rtn=GT_MapAxis(2,cnt2);     error(rtn);
}

void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  MapAxis();
  }
```

The user may understand the basic principle of coordinate mapping from the three example above. So, such calculation, such as coordinate translation, coordinate rotation and transformation of coordinate scale (which can easily convert the length unit) etc., can be realized through coordinate mapping.

Relationship between the coordinated motion mode and independent axis motion modes:

- The coordinated motion has its own motion commands. Most of the commands of independent axis motion are invalid.
- The modes of motion can be switched with each other. The precondition is that the axis in mode changing is not in motion.
- The coordinate mapping command doesn't modify the specified axis to

current axis; the current axis can be changed only by command GT_Axis() or GT_AxisI().

 **Attention**	*If user want to establish a contradictory mapping relationship, such as mapping two motor axes to Axis X in the coordinate system, the motion controller will calculate the coordinate position according to the mapping relationship with the prior axis, then change the actual position of the other axis according to the coordinate mapping relationship. This may result in the abrupt motion of the other axis. (Axis 1 is prior to axis2, axis 2 is prior to axis 3, and so on)*

 **Notice**	*To avoid this contradictory coordinate mapping causing abrupt motion of the motor, it is suggested that the coordinate mapping relationship agree with independent and unrelated condition.*

 **Notice**	*As to complicated coordinate mapping relationship, please contact Googol Technology. We can customize special interface function as user's requirement, to meet specific application.*

5.2 Set Vector Velocity and Acceleration of Coordinated Motion

5.2.1 Command Summary

Table 5-1 Command Summary of Setting Vector Velocity and Acceleration of Coordinate System Motion

Function	Description
GT_SetSynVel()	Set vector velocity of coordinate motion.
GT_SetSynAcc()	Set vector acceleration of coordinate motion.

5.2.2 Example

Example5-4 Setting Vector Velocity and Acceleration of Coordinated Motion

This example describes the specifying and unit description of the vector velocity and acceleration of a coordinated motion.

This example uses the coordinate mapping of unit conversion in Sample 5-2 to map the unit of coordinate system as 1mm. Suppose the vector velocity is 3m/min, vector acceleration is 0.9m/min^2 and the default sample time is 200us.

3m/min＝3000/300000(mm/ST)＝0.01mm/ST

$0.9m/min^2 = 900/(9x10^{10})(mm/ST^2) = 1x10^{-8}(mm/ST^2)$

```
void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  double cnt1[5]={2000,0,0,0,0};   /* Set coordinate mapping array according to the system. */
  double cnt2[5]={0,2000,0,0,0};   /* Set coordinate mapping array according to the system. */
  double cnt3[5]={0,0,2000,0,0};   /* Set coordinate mapping array according to the system. */
  double cnt4[5]={0,0,0,2000,0};   /* Set coordinate mapping array according to the system. */
  rtn=GT_MapAxis(1,cnt1);  error(rtn);/* Mapp Axis 1 to Axis X. */
  rtn=GT_MapAxis(2,cnt2);  error(rtn);/* Mapp Axis 2 to Axis Y. */
  rtn=GT_MapAxis(3,cnt3);  error(rtn);/* Mapp Axis 3 to Axis Z. */
  rtn=GT_MapAxis(4,cnt4);  error(rtn);/* Mapp Axis 4 to Axis A. */
  rtn=GT_SetSynAcc(0.00000001); error(rtn); /* Set vector acceleration as 0.9 (m/min^2) */
  rtn=GT_SetSynVel(0.01); error(rtn); /* Set vector velocity as 3 (m/min) */
  }
```

5.2.3 Notes of Main Point

The command short GT_SetSynVel (double Vel) specifies the target vector velocity in coordinated motion. The parameter Vel is the target vector velocity value. **Its unit is the length unit of coordinate system per sample time.** It will act on the velocity of all the linear interpolation and circular interpolation commands called later, until it is called again.

Vector velocity: $V = \sqrt{V_x^2 + V_y^2 + V_z^2 + V_A^2}$

The command short GT_SetSynAcc(double Accel) specifies the vector acceleration in coordinated motion. The parameter Accel is the vector acceleration value. **Its unit is the length unit of coordinate system/ sample time^2.** It will act on the acceleration of all the linear interpolation and circular interpolation commands called later, until it is called again.

Vector acceleration: $Acc = \sqrt{Acc_x^2 + Acc_y^2 + Acc_z^2 + Acc_A^2}$

*The unit of vector velocity and acceleration (i.e. **Length Unit of Coordinate Syste**m) is related to the coordinate mapping parameter set by coordinate mapping, i.e. the setting of vector velocity and acceleration*

Notice	*corresponds to coordinate axis X, Y, Z and A, not motor axis 1, 2, 3 and 4.*

5.3 Set Motion Path in Coordinate System

5.3.1 Command Summary

Table 5-2 Command Summary of Motion Path in Coordinate System

Command	Description
GT_LnXY()	2D linear interpolation
GT_LnXYZ()	3D linear interpolation
GT_LnXYZA()	4D linear interpolation
GT_ArcXY()	circular interpolation in XY plane (The parameters are the circle center position and angle.)
GT_ArcXYP()	circular interpolation in XY plane (The parameters are the position of end point and radius.)
GT_ArcYZ()	Circular interpolation in YZ plane (The parameters are the circle center position and angle.)
GT_ArcYZP()	Circular interpolation in YZ plane (The parameters are the position of end point and radius.)
GT_ArcZX()	Circular interpolation in ZX plane (The parameters are the circle center position and angle.)
GT_ArcZXP()	Circular interpolation in ZX plane (The parameters are the position of end point and radius.)

5.3.2 Example

Example 5-5 Realization of Coordinated motion

In this example, based on Sample 5-4, the command of setting motion path is called to realize the single-segment path motion immediately in coordinate system.

```
void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  double cnt1[5]={2000,0,0,0,0};
  double cnt2[5]={0,2000,0,0,0};
  double cnt3[5]={0,0,2000,0,0};
  double cnt4[5]={0,0,0,2000,0};
  rtn=GT_MapAxis(1,cnt1);  error(rtn);
  rtn=GT_MapAxis(2,cnt2);  error(rtn);
```

```
rtn=GT_MapAxis(3,cnt3);  error(rtn);
rtn=GT_MapAxis(4,cnt4);  error(rtn);
rtn=GT_SetSynAcc(0.00000001); error(rtn);
rtn=GT_SetSynVel(0.01); error(rtn);
//The above refers to Sample 5-4.
 rtn=GT_LnXY(10,10);       //The linear interpolation    in plane (10mm, 10mm).
}
```

5.3.3 Notes of Main Point

According to the right-hand rotation rule, the rotating direction of circular interpolation is defined as that, from the “top” of 2-D coordinate plane (i.e. the positive direction of the third axis which is vertical to the 2-D coordinate plane), the counter-clockwise is positive (Fig. 1-9). In short to remember: Extend the thumb of right hand, and make a 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 positive rotating direction. When the mapping coordinate system is a 2D system (X-Y), the positive direction of circular interpolation in XOY coordinate plane is defined as the same way.

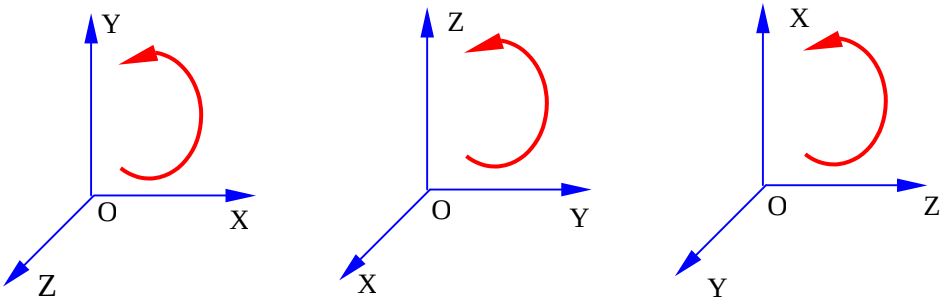

Fig. 5-3 Positive Direction of Arc Interpolation

5.4 Realization of Multi-segment Path Continuous Motion

In example 5-5, if user wants to realize a multi-segment path continuous motion, following the last line rtn=GT_LnXY(10,10), add one more command, such as “rtn=GT_LnXYZ(20,20,20)”. After running the program, the user will find that the line motion added is not realized, and the return value of the command added is 1 (wrong command). This is due to that, when the coordinated motion is uncompleted, and it is in the single-segment path motion status, the motion controller doesn’t receive new motion path command. Only after the previous path motion is completed, the new command can be received.

To realize multi-segment path continuous motion in coordinate system, the motion controller provides a ***buffer*** with a size of 4k. It allows user to push several motion path and parameter commands into the buffer, and then start motion. During the controller

execute the motion path command saved in the buffer, the host can push the motion path and parameter commands into the buffer on and on. By this way, the requirement for real-time communication between the host and the controller is not very strict, and communication efficiency is improved. At the same time, the motion controller can achieve good continuous motion effect by preprocessing the motion path in the buffer.

The following will describe the realization of multi-segment path continuous motion by two parts: how to push motion path and parameters commands into buffer and how to execute coordinated motion in the buffer.

5.4.1 Push Motion Path and Parameters Commands into Buffer

5.4.1.1 Command Summary

Table 5-3 Command Summary of Buffer Management Command

Function	Description
GT_StrtList()	Open and clear the buffer.
GT_MvXY()	Specify the position of start point in buffer (2D).
GT_MvXYZ()	Specify the position of start point in buffer (3D).
GT_MvXYZA()	Specify the position of start point in buffer (4D).
GT_AddList()	Reopen the buffer.
GT_EndList()	Close the buffer.

The commands able to be pushed into the buffer are listed in Table 5-4.

Table 5-4 Command Summary of Commands able to be pushed into Buffer

Function	Description
GT_SetSynVel()	Specify the vector velocity of coordinated motion.
GT_SetSynAcc()	Specify the vector acceleration of coordinated motion.
GT_LnXY()	2D linear interpolation
GT_LnXYZ()	3D linear interpolation
GT_LnXYZA()	4D linear interpolation
GT_ArcXY()	Circular interpolation in XY plane (The parameters are the circle center position and angle.)
GT_ArcXYP()	Circular interpolation in XY plane (The parameters are the end point position and radius.)
GT_ArcYZ()	Circular interpolation in YZ plane (The parameters are the circle center position and angle.)
GT_ArcYZP()	Circular interpolation in YZ plane (The parameters are the end point position and radius.)
GT_ArcZX()	Circular interpolation in ZX plane (The parameters are the circle center position and angle.)
GT_ArcZXP()	Circular interpolation in ZX plane (The parameters are the end point position and radius.)

5.4.1.2 Notes of Main Point

Open and clear the buffer, Move to the position of start point in the buffer.

The command GT_StrtList() can be called to open the buffer and clear the commands unexecuted in the buffer. When user specifies the motion of coordinated motion for the first time after the motion controller is powered, if the user wants to realize multi-segments path continuous motion by the command buffer strategy, he must call the command GT_StrtList() at first to enter the status of inputting buffer command.

On the neck of calling the command GT_StrtList() to open and clear the buffer (GT_StrtList), the user **must** call the command (GT_MvXY, GT_MvXYZ or GT_MvXYZA) to specify the position of start point in the buffer. After starting coordinated motion, the motion controller will move from the current position to the position of start point specified by this command along the path of linear interpolation, during the process, it will accelerate up to target vector velocity and decelerate vector velocity to zero. Then the controller will execute the following commands of coordinated motion in buffer in order.

The commands GT_MvXY(), GT_MvXYZ() and GT_MvXYZA() contain vector velocity and acceleration parameters. These parameters will be used as vector velocity and acceleration parameters for other motion path commands pushed into the buffer later, till new vector velocity and acceleration commands are specified.

Close the buffer.

The command GT_EndList() can be called to close the buffer. This command is valid when the buffer is open.

Reopen the buffer.

The command GT_AddList() can be called to open a closed buffer that was opened before, after that, the motion path and parameters commands can be pushed into buffer again.

When the process of inputting motion path and parameters commands to the buffer is over, the command GT_EndList() can be called again to close the buffer. User can call the command GT_AddList() several times.

Notes of pushing coordinated motion command into the buffer

After calling GT_StrtList(), user can call GT_EndList() at any time to finish the status of inputting buffer command. GT_AddList() can be called to reenter the status of inputting buffer command. Then, GT_EndList() can be called again to finish the status of inputting buffer command. The combination of GT_AddList() and GT_EndList() can be used for any times. When the description of motion path is completed, user must call GT_EndList() to inform the motion controller.

Mechanism of the motion controller processing the coordinated motion commands in the buffer

User can input coordinated motion commands into the buffer continuously until it is full.

The buffer in the motion controller is a 4096×16Bit loop buffer. After the buffer is full, the motion controller will refuse the motion path and parameters commands and return a status that the buffer is full. After the motion described by the commands in the buffer are started, there will be new space in the buffer, allowing more commands to be pushed into.

5.4.2 commands of Starting and stopping coordinated motion in the buffer

5.4.2.1 Command Summary

Table 5-5 Command Summary of Starting and Stopping Commands

Function	Description
GT_StrtMtn()	Start executing coordinated motion command in the buffer.
GT_StpMtn()	Stop coordinated motion smoothly.
GT_EStpMtn()	Stop coordinated motion abruptly.

5.4.2.2 Example

Example 1-21 Realizing multi-segment path continuous motion

This example realizes the continuous motion of three segments path, which can be referred to realize the continuous motion of more segments path.

```
void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  double cnt1[5]={2000,0,0,0,0};
  double cnt2[5]={0,2000,0,0,0};
  double cnt3[5]={0,0,2000,0,0};
  double cnt4[5]={0,0,0,2000,0};
  rtn=GT_MapAxis(1,cnt1);  error(rtn);
  rtn=GT_MapAxis(2,cnt2);  error(rtn);
  rtn=GT_MapAxis(3,cnt3);  error(rtn);
  rtn=GT_MapAxis(4,cnt4);  error(rtn);
  //The above refers to sample 5-2.
  rtn=GT_MvXYZA(0,0,0,0,0.01,0.00000001); error（rtn）;
  /* Specify the position of start point (0mm, 0mm, 0mm, 0mm) in the buffer,
    specify vector velocity of 3m/min and vector acceleration of 0.9m/min^2 */
rtn=GT_LnXY(10,10);     error(rtn);  //Specify 2D linear interpolation, the
                                     end point is at 10mm,
                                     10mmrtn=GT_ArcXY(0,0,1
```

```
                                                    23); error(rtn);
// Specify circular interpolation in XY plane, the circle center position is at (0, 0)
   and the circular angle is 123 degree.
rtn=GT_LnXYZA(0,0,10,12); error(rtn); //Specify 4D linear vector, the end
                                      point is at (0,0,10,12)
rtn=GT_EndList();        error(rtn);  //Close the buffer.
rtn=GT_StrtMtn();         error(rtn);  //Start motion described by the
                                      commands above.

  }
```

5.4.2.3 Notes of Main Point

Start motion described by the commands in buffer.

The command GT_StrtMtn() can be called to start motion described by the commands in the buffer in order. Before calling the command, user must confirm that there are motion path commands in the buffer that can be executed, that is, the host has called at least the commands GT_StrtList() and GT_MvXY(GT_MvXYZ, GT_MvXYZA) before.

After the command GT_StrtMtn is called, and before the command GT_EndList is called, the motion controller will, and execute the commands added in the buffer on and on.

Stop coordinated motion

If the motion described by the commands in the buffer has been started and the user wants to stop it, user may call the commands GT_StpMtn() and GT_EStpMtn(). The command GT_StpMtn() is similar with the command GT_SmthStp() for independent axis motion stopping, stopping the coordinated motion smoothly. While the command GT_EStpMtn() is similar with the command GT_AbptStp() for independent axis motion stopping, stopping the coordinated motion abruptly.

In addition to stopping the coordinated motion in the buffer, the commands GT_StpMtn() and GT_EStpMtn() also close the buffer at the same time, just like calling the command GT_EndList().

After the commands GT_StpMtn() or GT_EStpMtn() is called, the motion path command sent by the user will be seemed as command of single-segment path motion, and executed immediately.

In the status of “coordinated motion completed”, if there are still some motion path commands in buffer, user can call the command GT_StrtMtn() to restart motion described by the commands in buffer. Then the motion controller will start coordinated motion from current position to the position where the command GT_StpMtn() or GT_EStpMtn() stopped at in terms of the vector velocity specified in the command GT_MvXY (GT_MvXYZ, GT_MvXYZA) , and decelerate the velocity to zero, and then continue to execute commands in the buffer.

注意 **Notice**	*User cannot call the stop command during the execution of command GT_MvXY(GT_MvXYZ,GT_MvXYZA) or the execution of back to the position of breakpoint in coordinated motion , i.e. the process of moving to the start point of the buffer cannot be stopped. Now, sending stop command will cause motion error.*

5.4.3 Planning Strategy of Vector Velocity in Multi-segment path

The vector velocity of along the path applies the T-curve acceleration/deceleration strategy. The commands GT_SetSynVel() and GT_SetSynAcc() specify the corresponding target vector velocity and acceleration respectively.

For the motion path commands in buffer (except GT_MvXY(), GT_MvXYZ() and GT_MvXYZA()), the vector velocity the path will accelerate from zero to the target velocity at the first segment path (linear interpolation or circular interpolation), and decelerate to zero to agree with the end position of the last segment. While at the joint of each linear interpolation segment or circular interpolation segment along the whole path, the controller will reduce the change of vector velocity along the path as much as possible, to obtain continuous and stable motion specialty.

While at this time, the acceleration characteristic of each motor axis should be considered to prevent higher acceleration from causing path distortion. Therefore, there is a command named GT_SetAccLmt() (a command for each motor axis) can be used to specify the acceleration limit of each axis according to its actual mechanical and electrical characteristics, and applied to the following strategy to decide the vector velocity at joint.

Strategy One: At the joint of two segments, the natural transition of the two segments should be considered at first. The vector velocity should accelerate and decelerate according to specified velocity and acceleration. At the same time, it is necessary to confirm that the transformation of velocity in each relevant motor axis doesn't exceed the acceleration limit specified for each axis. Otherwise, another strategy should be applied (strategy two). If the vector velocity for each segment in two segments specified by the user is different, such strategy should be applied (*Figure 5-4(a)(b)(c)*): the deceleration part is happened in the first segment, and the acceleration part is happened in the second segment. This will assure that, the motion vector velocity at any segment will not exceed the target vector velocity specified by user.

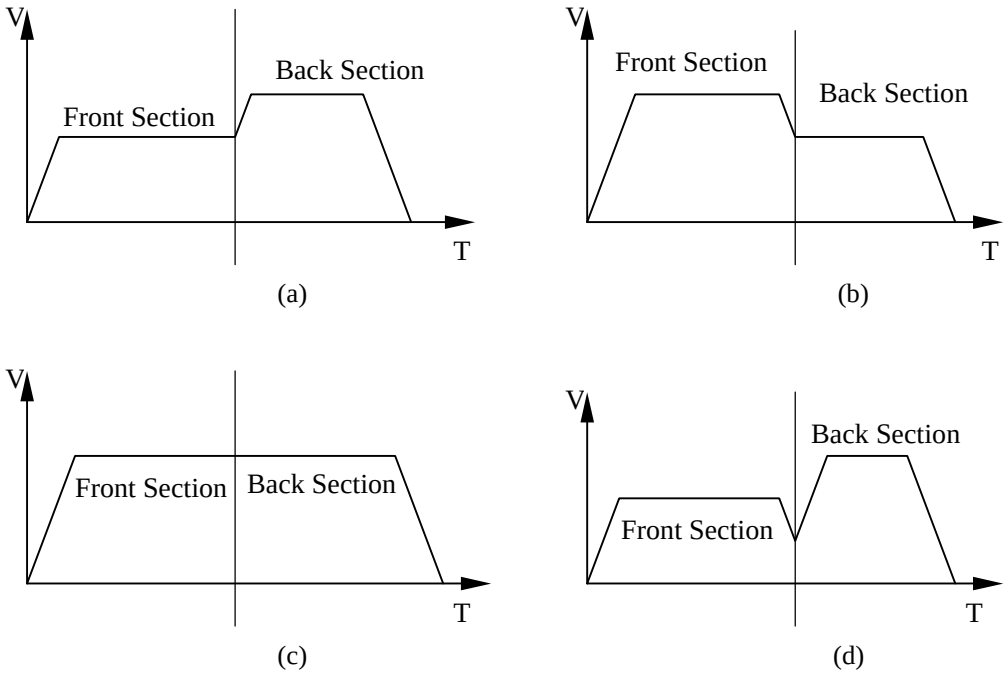

Fig. 5-4 Velocity Strategy at Track

Strategy Two: At the joint of two segments, if the situation described in Strategy One is happened, an appropriate vector velocity at the turning point should be calculated, and decelerate the vector velocity as little as possible at the premise of satisfying the acceleration limit of each axis at turning point, to avoid large velocity transformation at the joint to the best. At this situation, the velocity transformation strategy is also like the description above *Figure 5-4(d)*): the deceleration part is happened in the first segment, and the acceleration part is happened in the second segment.

Notices: When specifying velocity and acceleration for path planning, user should make sure that the deceleration part be completed in one segment, i.e. achieving the vector velocity at the turning point in one segment.

5.4.4 Breakpoint information in Multi-segment path Continuous Motion

After the host calls the commands GT_StpMtn() and GT_EStpMtn(), calling the commands GT_GetBrkPnt() and GT_GetMtnNm() can get the relevant information about breakpoint. GT_GetBrkPnt() can be called to get the coordinate position of breakpoint and GT_GetMtnNm() can be called to get the number of segment at breakpoint. The number of segments is defined as the following default rule.

After starts the coordinated motion in buffer, the segment number will increase by

degrees. GT_MvXY() (or GT_MvXYZ() or GT_MvXYZA()) doesn't have a segment number, i.e. the segment number is 0 when this command is executed, then the following segment number will be increased in order.

When the execution of the segments in the buffer is completed, but user doesn't call GT_EndList() to close the buffer. The motion controller will consider the coordinated motion is not completed yet. At this time, the bit0 and bit1 for the status of coordinate system will not be set, and the segment number will remain as the current segment number executed. The user also can call GT_GetMtnNm() during the motion process; it will return the current segment number being executed.

The increasing rule of segment numbers is:

1. When the motion of current segment path is completed, if there is other command in the buffer, the segment number will increase by 1, if there is none, the segment number will remain as the current value.
2. During the coordinated motion in buffer, when GT_StpMtn() or GT_EStpMtn() are used to stop the coordinated motion (which will cause the Bit0 and Bit1 of the coordinate system status register to be set), the segment number will remain as the current value.
3. When the user calls the command GT_StrtList(), the segment number will be set to zero.
4. The segment number is increased from zero. When it reaches the maximum number (65536), the numbers will overflow and increase from zero again.

5.4.5 Coordinate System Status Register

The motion controller provides a status register to describe the status of planning coordinated motion. User may use GT_GetCrdSts() to access this 16-bit register. The definition of each sign is listed in *Table 5-6*. The symbol bits in the status register only indicates the status of current coordinated motion, and are managed by the motion controller. User cannot interfere the status of this register.

Table 5-6 Definition of Bits in Coordinate System Status Register

Bit	Definition
0	1 = the coordinated motion are finished, or there is no coordinated motion.
1	1 = The status of pushing motion path and parameters commands into buffer is finished (When GT_EndList(), GT_StpMtn() or GT_EStpMtn() are called, this bit is set.).
2	1 = The sample time (interpolation period) is too short to cause motion error (When this bit = 1, please modify the sample time immediately.).
3	1 = The last command for coordinated motion is error.
4	1 = current segment path motion is completed.
5	1 = Alarm of exceeding acceleration limit (No matter which coordinate axis exceeds limit, this bit will be set.).
6	1 = Allow auto stop for coordinated motion at abnormal situation (When there is alarm signal or axis inactive sign or open loop sign given by the

Bit	Definition
	motor axis relevant to the coordinated motion s the coordinated motion will be stopped automatically).
7	1 = in status of single-segment path motion, 0 = in status of multi-segments path continuous motion in buffer.
8	Reserved.
9	1 = abnormal situation occurs in the motor axis relevant to the coordinate system and the coordinated motion is stopped automatically.
10	1 = abnormal error occurs in the coordinated motion (In normal, this bit will not be set, unless abnormal situation occurs in the motion controller. If this bit is set, please stop using the controller immediately, turn off the power and restart the controller.).
11～12	Reserved. The default status is 0.
13	1 = The buffer is empty (After the buffer motion starts, if there is no motion path command in the buffer, this bit will be set.).
15	Reserved. The default status is 0.

When Bit6 of the coordinate system status register is set to 1, if the motion of any axis mapped in the coordinate system encounters abnormal situation (including servo alarms, axis inactivation, open loop or triggering limit switch.), the motion controller will stop the coordinated motion and planning. This will assure the integrality of the whole motion path and the safety of the motor axis that take part in the coordinated motion. If Bit6 = 0, when the motion of any axis encounters abnormal situation, the motion controller will only stop the motion of this axis, but continue to plan the coordinated motion. The motion of other motor axes mapped in the coordinate system will continue as they were, **but it is very dangerous**. Bit6 = 1 or = 0 are done by GT_CrdAuStpOn() and GT_CrdAuStpOff(), the default value of bit6 in the controller is 1.

Chapter Six High velocity Home/Index Capture

The motion controller provides a high-velocity position capture register for each axis, to save the axis's actual position exactly at the time of the external signal triggering. GT-400-SV allows the user to use the C (Index) signal of incremental encoder or home switch signal as the triggering signal of capturing the axis position. SD, SE and SG allow the home switch signal to be used as the triggering signal of capturing the axis position. The controller can use the command GT_CaptIndex() (only for SV card) and GT_CaptHome() to select the kind of signal to capture position and permit to capture.

After the controller captures the Index or Home signal needed, the sign of Index/Home captured in the status register of specified axis will be set to 1, and the sign bit of setting Index capture or Home capture in the status register (15 bit or 14 bit) will be cleared. If the user wants to trigger next capture of position, he must give the permission of the position capture again and clear the sign of Index/Home captured in the status register of specified axis before the next position capture happens. The position captured by the controller is the actual position of specified axis when the capture signal triggers. The precision of captured position is +/-1 pulse. The motion controller uses hardware to capture position, so the motion velocity of the controller will not affect the capture precision.

The high-velocity position capture function of the controller is mainly used to fix the home position. For those users who require high precision for the repeated positioning, they may use **Home+Index method to fix the home position**, that is, after capturing Home signal, capture position of the Index signal closest to Home signal.

Example 6-1 Home+Index Programming (Only for SV Card)

This example fixes the home position and set to zero.

```
void home(long pos)
{
   unsigned short status;
   long actl_pos;
   rtn=GT_ClrSts();        error(rtn);   //Clear status.
   rtn=GT_CaptHome();       error(rtn);   //Specify the capture mode of Home capturing.
   rtn=GT_PrflT();         error(rtn);   //Specify T-curve of current axis.
   rtn=GT_SetVel(4);       error(rtn);   //Specify velocity of 4 pulse/ST.
   rtn=GT_SetAcc(1);          error(rtn);    //Specify acceleration of 1 pulse/ST2.
   rtn=GT_SetPos(pos);     error(rtn);   //Specify target position.
   rtn=GT_Update();        error(rtn);   //Update parameter.
   rtn=GT_GetSts(&status);  error(rtn);   //Get axis status.
```

```
        while(!(status&0x8))                    //Wait for Home capturing.
        {
          if(status&0x1) return;  //If the motion is completed but Home is not triggered,  exit.
          rtn=GT_GetSts(&status); error(rtn); //Get axis status.
         }
        rtn=GT_GetCapt(&pos);     error(rtn);   //Get position captured.
        rtn=GT_SetPos(pos);        error(rtn);   //Specify capture position of target position.
        rtn=GT_Update();          error(rtn);   //Update parameter.
        rtn=GT_ClrSts();          error(rtn);   //Clear status.
        rtn=GT_GetSts(&status);   error(rtn);   //Get status value.
        while(!(status&0x1))       //Wait until the motion is completed.
        {
          rtn=GT_GetSts(&status); error(rtn);   //Get status.
          }
        rtn=GT_ClrSts();          error(rtn);   //Clear status.
        rtn=GT_CaptIndex();       error(rtn);   //Specify the capture mode of Index capturing.
        pos=pos+8000;                           //Captured position + 8000 pulse
        rtn=GT_SetPos(pos);          error(rtn);   //Move 8000 pulse from captured position.
        rtn=GT_Update();          error(rtn);
        rtn=GT_GetSts(&status);   error(rtn);   //Get status.
        while(!(status&0x8))         //Get status and wait for index capturing.
        {
          rtn=GT_GetSts(&status); error(rtn);
        }
        rtn=GT_ClrSts();          error(rtn);    //Clear status.
        rtn=GT_GetCapt(&pos);     error(rtn);    //Get position captured.
        rtn=GT_SetPos(pos);       error(rtn);    //Specify the captured position of target position.
        rtn=GT_Update();          error(rtn);
        rtn=GT_GetSts(&status);   error(rtn);    //Get status.
        while(!(status&0x1))         //Wait until the motion is completed.
        {
          rtn=GT_GetSts(&status); error(rtn);
         }
        rtn=GT_ClrSts();          error(rtn);    //Clear status.
        rtn=GT_ZeroPos();          error(rtn);    //Specify actual and target position of zero.
    }

    void main()
    {
      short rtn;
      GTInitial();
      InputCfg();
      AxisInitial();
      rtn=GT_Axis(1); error(rtn);
      Home(200000);
      }
```

Example 6-2 Home Programming

This example is to go back Home, i.e. return to home point.

```
void home(long pos)
{
   unsigned short status;
   long actl_pos;
   rtn=GT_ClrSts();          error(rtn);   //Clear status.
   rtn=GT_CaptHome();        error(rtn);   // Specify the capture mode of Home capturing.
   rtn=GT_PrflT();           error(rtn);   //Specify the independent motion mode of T-curve.
   rtn=GT_SetVel(4);         error(rtn);   //Specify velocity of 4 pulse/ST.
   rtn=GT_SetAcc(1);           error(rtn);   //Specify acceleration of 1 pulse/ST2.
   rtn=GT_SetPos(pos);       error(rtn);   //Specify target position.
   rtn=GT_Update();          error(rtn);   //Update parameter.
   rtn=GT_GetSts(&status);   error(rtn);   //Get axis status.
   while(!(status&0x8))                    //Wait for Home capturing.
   {
     if(status&0x1)  return; //If the motion is completed but Home is not triggered, exit.
     rtn=GT_GetSts(&status); error(rtn); //Get axis status.
    }
   rtn=GT_GetCapt(&pos);     error(rtn);   //Get position captured.
   rtn=GT_SetPos(pos);        error(rtn);   //Specify capture position of target position.
   rtn=GT_Update();          error(rtn);   //Update parameter.
   rtn=GT_ClrSts();          error(rtn);   //Clear status.
   rtn=GT_GetSts(&status);   error(rtn);   //Get status.
   while(!(status&0x1))      //Wait until the motion is completed.
   {
     rtn=GT_GetSts(&status); error(rtn);   //Clear status.
     }
   rtn=GT_ZeroPos();           error(rtn);    //Specify actual and target position of  zero.
}

void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  rtn=GT_Axis(1); error(rtn);
  Home(200000);
  }
```

Chapter Seven Safety Mechanism

7.1 Monitor Axis Motion Error and Restore Status

The motion controller provides the function of monitoring axis motion error.

For the close loop control, sometimes the actual position of motor may be far from the target position. This situation usually means some dangers existing, such as motor fault, reversed connection or disconnection of A and B signals of encoder, stemming of motor caused by too large mechanical friction or mechanical fault. In order to detect this situation, to improve the safety of the system and prolong the service life of the equipment, GT-400-SV controller sets programmable and modifiable position error limit of specified axis.

The command GT_SetPosErr() can be called to specify the position error limit and the command GT_GetPosErr() can be called to get it.

In each sample time, the motion controller compares the position error limit with the actual position error to check if a motion error occurs. If the actual position error of specified axis overreaches the error limit, the controller will consider this axis has motion error.

When the motion error of specified axis occurs, the controller will generate the following events.

- The sign of motion error in status register = 1.
- If the sign of allowing auto stop when position error overreach error band = 1, the motion of specified axis will be stopped, the sign of motion completed will be set to 1, and motor of specified axis will be inactive. Otherwise, only the sign of motion error in the status register of specified axis will be set to 1. The user can use the commands GT_AuStpOn() or GT_AuStpOff() to set or clear the sign of allowing auto stop when position error overreach error band. The default value of this sign is 0, i.e. the motion of specified axis will not be stopped automatically when motion error occurs.

To restore from the error status to normal status, the user must take the following steps.

- Confirm the causes of motion error and correct them.
- Clear the sign of motion error in status register.
- Use GT_SynchPos(), GT_SetAtlPos() or GT_ZeroPos() to synchronize the actual and target position of the axis in error, or reset the actual and target

position to zero.

- Use the command GT_AxisOn() to reactivate this axis.
- After the steps above are completed, this axis will reenter the normal status and wait for next motion.

7.2 Treat Axis Driver Alarm

For the purpose of safety and longer service life of motor and driver, some motor drivers set the detecting and protecting of system fault (such as input voltage limit of driver, protection of too large change for input signal). When the driver detects abnormal or fault situation, it will trigger the fault protection function and output an alarm signal.

The controller provides a dedicated input of driver alarm signal. When it detects the input signal of motor driver alarm, the controller will set the sign bit of driver alarm and the sign bit of motion completed in the status register of specified axis as 1, and inactivates the motor driver of this axis (same as the effect of the command GT_AxisOff()).

To restore from the driver alarm status to normal status, the user must take the following steps:

- Confirm the causes of the alarm signal and correct them.
- Use the command GT_DrvRst() to reset the driver.
- Use the command GT_ClrSts() or GT_RstSts() to clear the sign of driver alarm in the status register.
- Use the command GT_SynchPos(), GT_SetAtlPos() or GT_ZeroPos() to synchronize the actual and target position of the axis in error, or reset the actual and target position to zero.
- Use the command GT_AxisOn() to reactivate this axis

After the steps above are completed, this axis will reenter the normal status and wait for next motion.

7.3 Treat Limit Status

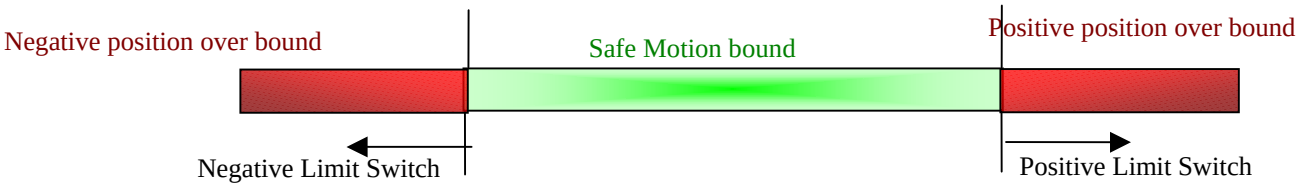

Fig. 7-1 Definition of Motion bound of Specified Axis

The limit switch can be used to indicate automatically the motion bound of axes. The “safe” motion bound of axis with limit switch is illustrated in *Fig. 7-1*.

After receiving the signal triggered by limit switch, the controller will operate as follow:

- The controller will set the sign of relevant limit switch triggering in the status register of the axis in limit switch, and notify the host to take proper actions to process.
- Meanwhile, the controller will stop the motion of this axis immediately to prevent it from moving further towards the area of over the bound.

Once a controlled axis triggers a limit switch, this axis will only be allowed to move towards the opposite direction. For example, suppose that the positive limit switch of a controlled axis be triggered, the controller will only allow the axis to move towards the negative direction, so as to return back to the safe motion range.

After the controlled axis returns back to the safe motion bound, the host must call the command GT_ClrSts() or GT_RstSts() to clear the relevant status bit, to restore the status of controlled axis from the status of over bound to normal status.

If a specified axis works in electronic gearing mode, this axis triggers the limit switch, and if the master axis is inactivated status, its motion direction will be also restricted by the limit switch status of slave axis. For example, when the negative limit switch of the slave axis is triggered, the slave axis will only be allowed to mover towards the positive direction and the master axis will only be allowed towards a certain direction: if the gear ratio is negative, the master axis will only be allowed towards the negative direction. This relation restricting each other will keep until the secondary axis returns back to the safe area.

Chapter Eight Interrupt

The motion controller can send interrupt request to the host, so that the host can process the events during the motion of each control axis in time. This interrupt method is generally more convenient and effective than that the host inquiring status of each control axis.

The motion controller provides two interrupt method. One is called **Event Interrupt**, mainly to treat the events during the motion of control axes in time. The other is **Time Interrupt**, that the controller sends time interrupt to the host in certain period. When the motion controller is included in a system, this interrupt can be used as the system timer.

Since the two interrupt methods share one interrupt request signal of the host computer the host in the system only allows to evoke the commands GT_TmrIntr() and GT_EvntIntr() to select one interrupt method. The command GT_TmrIntr() sets the interrupt of control axis as time interrupt, whose period is decided together by the control cycle of controller and the set value of the command GT_SetIntrTm(). For example, if the control cycle of controller is 200 microseconds and the host uses the command GT_SetIntrTm() to set a value of 10, the period of time interrupt = 10*200 milliseconds. The command GT_EvntIntr() closes the time interrupt and turns to the event interrupt. The default interrupt by the controller is the event interrupt.

The events of control axis corresponding to bit0-bit6 in the status register of control axis listed in *Table 4-14* can all trigger the event interrupt request. For each control axis, the host uses the command GT_SetIntrMsk() to set the interrupt mask register of the control axis, and to determines whether an event can cause an interrupt request to the host or not.

8.1 Interrupt Treatment in DOS

If a control axis has the above interrupt situation and activates the interrupt request signal, the host should respond to the interrupt request according to the actual situation and perform proper treatment. After interrupt treatment, the host shall send the instruction GT_RstIntr() to clear the interrupt request condition of the current motor, to enable the next interrupt to happen. This command has a parameter of “Clear relevant mask symbol”.

The host can only respond to one interrupt of control axis one time. For example, when user is setting the parameters of current axis #1, the control axis #3 sends an interrupt request signal to the host. Now, to treat this interrupt, the host must set #3 as the current axis (by the command GT_AxisI()). If several control axes send interrupt request at the same time, the control axis with smallest number will have the highest interrupt priority.

The following lists a typical interrupt treatment sequence to describe how the host responds to an interrupt. Suppose the control axis #3 (The current axis is #1.) is in a status of exceeding limit due to instant motion error caused by “abrupt stop”, and at the same time, suppose the interrupt mask register of the control axis allows the host to respond to the above interrupt request signal sent from the motor. Now, the responding of the host to the interrupt is listed in *Table 8-1*. At the last of the above process, all the status bits will be cleared, the interrupt request signal of the host is restored and there is no interrupt waiting for treatment.

Table 8-1 Treatment of the Host Responding to Interrupt

Interrupt Event	Responding of the Host
Motion error and exceeding limit cause interrupt.	The host sends the command GT_AxisI().
The controller returns the status of the control axis that causes the interrupt request and set it as current axis.	The host finds that the motion error and limit exceeding symbol bit is 1, and treat the motion error first. The host sends the instruction GT_RstIntr(0xEF) to clear the motion error symbol bit.
The controller clears the motion error bit and restores the host interrupt request signal to low level.	-
Since the limit exceeding situation of axis is still effective, the controller sends limit exceeding interrupt request to the host immediately.	The host sends the instruction GT_AxisI().
The controller returns the status of the control axis that causes the interrupt request and set it as current axis.	The host finds that the limit exceeding bit is 1 and executes relevant treating program. Then the host sends the instruction GT_RstIntr(00DF) to clear the limit exceeding symbol bit.
The controller clears the limit exceeding symbol bit and restores the host interrupt source to low level.	-

The instructions GT_RstIntr() and GT_AxisI() are only effective when there is a interrupt request. If there is no interrupt, an inquiry command, such as GT_GetSts() can be used to check the status of axis.

For a control axis requesting an interrupt, the host can only respond to one event interrupt. When several events request interrupt at the same time, it is not necessary to treat these requests as described in the above example. The controller can send only one interrupt request to the host. During the processing of the interrupt, the host can use the command GT_GetIntr() to get the status of the axis sending interrupt request to judge and proceed relevant treatment. Then the host can use the command GT_RstIntr() to clear all the interrupt events (GT_RstIntr() and GT_AxisI() can only be used in an interrupt service routine. If GT_GetIntr() is evoked when there is no interrupt, the returned status will be the status of current axis.).

When several axes request interrupt at the same time, the host can also use the same

treating method. When responding to the interrupt of a control axis with smallest number, the host uses the command GT_GetSts() to view the status of other control axes and proceeds relevant treatment. Then, the host uses the command GT_RstSts() or GT_ClrSts() to clear relevant status symbols.

When an axis is disabled, except the motion error and drive alarm status, the other status cannot cause event interrupt. Please pay attention when using the controller.

Sample 8-1 Event Interrupt Sample (For ISA Bus Card)

```
void interrupt handler(…)
{
   disable();        //Close interrupt.
   GT_AxisI();    //Set the axis with interrupt as current axis.
   GT_GetIntr(&event);    //Get the status register of the axis with interrupt.
   If(event&0x8)              //Judge whether it is a HOME triggering interrupt.
     (
       GT_GetCapt(&HomePos);      //Get Home captured position.
       GT_SetPos(HomePos);          //Set target position as Home captured position.
       GT_Update();                    //Update parameter.
     )
   GT_RstIntr(0);                    //Clear the interrupt of moton control card.
   outportb(0x20,0x20);           //Send EOI to master 8259.
   outportb(0xa0,0x20);           //Send EOI to slave 8259.
   enable();                          //Open interrupt.
}
```

Sample 8-2 Time Interrupt Sample (For PCI Bus Card)

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "userlib.h"

GT_ISR oldisr;
int count;
void interrupt OnInterrupt(...)
{
   GT_ClearInt(0);          //Clear interrupt.
   count++;                  //Count + 1. User may set one's own code.
   outportb(0x20,0x20);    // Send EOI to master 8259.
   outportb(0xa0,0x20);    // Send EOI to slave 8259.
}
main()
{
   short nret=GT_Open();    //Open the motion controller.
   if(nret)                     //Check the return value.
   {
        printf("Open fail\n");
        return 0;
   }
   oldisr=GT_HookIsr(OnInterrupt);    //Hook interrupt.
   nret=GT_SetIntrTm(500);              //Set time interrupt period as 500*200
microseconds.
   if(nret)
   {
```

```
            printf("Set Interrupt Time fail\n");
            return 0;
        }
        nret=GT_TmrIntr();                  //Set time interrupt.
        while(!kbhit())
        {
            printf("Count:%d\n",count);     //Print count.
            delay(500);
        }
        getch();
        nret=GT_EvntIntr();                 //Set event interrupt.
        GT_UnhookIsr(oldisr);               //Cancel hooked interrupt and restore the
original interrupt.
        return 0;
    }
```

8.2 Interrupt Treatment in WINDOWS98/2000/NT

When developing program in DOS, user can modify interrupt vector and hook the interrupt service routine (ISR for short) directly to respond to the interrupt request generated by the motion controller. However, in WINDOWS, the operating system separates the system kernel from applications and no longer allows user program to modify ISR. So, GT-400-PCI driver program initiates two kinds of interrupt treatment mechanism concerning user program and equipment ISR, called ***Event Synchronization Mechanism*** and ***Interrupt Preprocessing Mechanism***. The former is applicable to treat event interrupt and time interrupt. The latter is only applicable to treat event interrupt. This two kinds of mechanism can be used together or independently.

8.2.1 Event Synchronization Mechanism

The principle is to allow equipment ISR and top-layer user program to share an event, to synchronize of user program and equipment ISR, so as to enable the user program to respond to the event from hardware equipment.

The detailed approach is to create a synchronization event in user program, and use the API function GT_SetIntSyncEvent(HEVENT hIntEvent) provided by the controller driver to set synchronization event for equipment ISR. Thus, user program and equipment ISR become two common processes sharing a synchronization event. Generally, user program will start a new thread to avoid blocking itself. It is effective to evoke WaitForSingleObject() to wait for event in the new thread. Once equipment interrupt occurs, equipment ISR will activate interrupt synchronization event. Now, the user thread is activated at WaitForSingleObject() and starts executing the following part of the thread.

Please pay attention that, before evoking CloseHandle() to close event and equipment, user program must evoke GT_SetIntSyncEvent(NULL) to notify equipment ISR to

clear the interrupt synchronization event.

During this process, user must have a clear understanding that, user thread is only notified that an interrupt happened just now, rather than it is in the process of treating interrupt; In fact, the interrupt has been cleared by the kernel. In addition, due to the influence of the system efficiency and interrupt frequency, it is not guaranteed that each and every interrupt will activate user thread. When the interrupt frequency is very high and the thread calling in the operating system is slow, there may be interrupt stacking for several times. Normally, the required interrupt frequency <=10KHz.

For the detailed programming, please refer to the following codes (All the Windows programs are written in VC++ environment.).

Sample 8-3 Interrupt Event Synchronization Mechanism

```
//Overall Variable
HANDLE hSyncEvent;      //Synchronization event handle
bool stopflag;          //Thread stop flag

//Codes in main function
HANDLE hSubThread;
DWORD idSubThread;
//……
   //Create a synchronization event.
   hSyncEvent=CreateEvent(NULL,true,false,NULL); //WIN32 API function
   if(hSyncEvent==INVALID_HANDLE_VALUE)
   {
        //..Check validity here.
   }
   //Set synchronization event for equipment ISR, to realize event sharing.
    nret=GT_SetIntSyncEvent(hSyncEvent);
    //API function provided by GT400.DLL.
   if(nret)
   {
        //Check function execution.
   }
   stopflag=false;
   //In order not to block oneself, create and start a new thread.
   hSubThread=CreateThread(NULL,
                           0,
                           intproc, //New thread function.
                           NULL,//Parameter used in new thread function.
                           0,
                           &idSubThread);
   //Until now, the task of main function is completed.

//The following is a thread function to wait for synchronization event.
```

```
DWORD WINAPI intproc(LPVOID param)
{
   ResetEvent(hSyncEvent); //Assure that the event is in non-signaled status.
   while(1)
   {
        //waiting for interrupt happen
        WaitForSingleObject(hSyncEvent,INFINITE);//Wait for event.
        if(stopflag)
             break;
        //add your code for handling intrerrupt event here
                  //.....

        //reset event state
        ResetEvent(hSyncEvent);//Reset synchronization event.
   }
   GT_SetIntSyncEvent(NULL);//Notify equipment ISR to release event.
   //close Synchronize Event Handle
   CloseHandle(hSyncEvent); //Close synchronization event handle.
   ExitThread(0); //Exit the thread.
   return 0;
}
```

8.2.2 Interrupt Preprocessing Mechanism

The principle is to preset some commands (which refer to the GT commands used by motion control card) for equipment before interrupt is generated, and execute relevant command to the interrupt according to the preset structure when equipment generates a specified interrupt.

The use of preprocessing is relatively simple. Generally, it will assign a certain size of memory space to save the commands to be set for equipment, fill this memory according to the specified structure (defined in gt400data.h) and evoke the API function GT_SetBgCommandSet（） provided by the driver of control card to set commands for equipment. When the equipment generates an interrupt, it will search automatically the data structure, find and execute the command corresponding to the interrupt. The execution result will also be saved in the structure. The user-layer program can use GT_GetBgCommandResult（）to get command execution result. For the functions available for background command set, please refer to Table 5-1. The command format is of adding Intr before the original function. For example, GT_PrflT() will become Intr_GT_PrflT as a background command. For the detailed programming, please refer to the following codes:

Sample 8-4 Interrupt Preprocessing Mechanism

```
//Define the maximum memory size to be used.
#define MAX_SIZE    500
```

```
//Buffer requires a size = 4+4* (the number of interrupt requiring command to be set) + 16* (the sum of all commands).
//In the following codes, the minimum space size needed: 4+4*2 (two kinds of interrupt) +16* (2+1) = 60 bytes.
    PBGCOMMANDSET pBgCmdSet;//Define a pointer.
    pBgCmdSet=(PBGCOMMANDSET)malloc(MAX_SIZE);//Allocate memory.
    if(pBgCmdSet==NULL)
    {
        //Check validity.
    }
    PBACKGROUND_COMMAND pBackCmd;
    PGENERAL_COMMAND pCmd;
    //Specify how many kinds of interrupt require background commands.
    pBgCmdSet->Count=2;                //Set backgrounds commands for two kinds of interrupt.
    //Set a command array for each interrupt respectively. The same interrupt allows several commands.
    pBackCmd=pBgCmdSet->BackgroundCommand;

    pBackCmd->InterruptMask=0x01;          //Specify interrupt source.
    pBackCmd->CommandCount=2;       //The number of commands to be executed when generating the interrupt.
    //Fill each command one by one.
    //The first command
    pCmd=pBackCmd->GenCommand;
    pCmd->usCommand= Intr_GT_SetPos;       //Set characters for command type .
    pCmd->OutputLength=2;
    //How many words (length) of data to be output into DSP when executing the command.
    pCmd->InputLength=0;
    // How many words (length) of data to be input into DSP when executing the command.
    pCmd->in.lData=20000;       //Input data.
    pCmd->out.lData=0;          //Output data.
    //Fill the next command.
    pCmd=PGENERAL_COMMAND((char*)pCmd+sizeof(GENERAL_COMMAND));  //Point to next memory unit.
    pCmd->usCommand= Intr_GT_Update;       //Set command.
    pCmd->OutputLength=0;
    // How many words (length) of data to be output into DSP when executing the command.
    pCmd->InputLength=0;
    // How many words(length) of data to be input into DSP when executing the command.
    pCmd->in.lData=0;           //Input data.
```

```
    pCmd->out.lData=0;              //Output data.

    //Set a command array for another interrupt.
    pBackCmd=
    PGENERAL_COMMAND( (char*)pCmd+sizeof(GENERAL_COMMAND));
    //Point to next memory unit.

    pBackCmd->InterruptMask=0x02;//Specify interrupt source.
     pBackCmd->CommandCount=1;// The number of commands to be executed
     when generating the interrupt.
    // Fill the first command.
    pCmd=pBackCmd->GenCommand;
    pCmd->usCommand= Intr_GT_SetKp;  // Set characters for command type .
    pCmd->OutputLength=1;
    // How many words (length) of data to be output into DSP when executing the
     command.
    pCmd->InputLength=0;
    // How many words (length) of data to be input into DSP when executing the
     command.
    pCmd->in.lData=0;               //Input data.
    pCmd->out.lData=20;             //Output data.
    //Transfer command buffer to equipment ISR.
    short nret=GT_SetBgCommandSet(pBgCmdSet,MAX_SIZE);
    if(nret)
{
    //Check the execution of function.
}
//Then release memory.
free(pBgCmdSet);
```

Chapter Nine General Purposed I/O

The motion controller provides input/output (I/O) ports for general use. The host can use command to control these I/Os.

The port 0 (EXI0) for general purposed input can be used as probe input signal. Use a relevant command to enable capture function of the signal, which may cause the motion controller to capture the actual position of all control axes and auxiliary encoders when the signal is active.

16-bit Output Port: The host uses the command GT_ExOpt(Data) to set the value of this port. The corresponding relation between Data and the general purposed output port EXO0-EXO15 on the connector CN2 of the controller is as follows.

Bit - Definition	Bit - Definition	Bit - Definition	Bit - Definition
Bit0----EXO0	Bit1----EXO1	Bit2----EXO2	Bit3----EXO3
Bit4----EXO4	Bit5----EXO5	Bit6----EXO6	Bit7----EXO7
Bit8----EXO8	Bit9----EXO9	Bit10----EXO10	Bit11----EXO11
Bit12----EXO12	Bit13----EXO13	Bit14----EXO14	Bit15----EXO15

16-bit Input Port: The host uses the command GT_ExInpt(&Data) to get the logic level of this port. The corresponding relation between the returned data Data and the definition of EXI0-EXI15 bits is as follows.

Bit - Definition	Bit - Definition	Bit - Definition	Bit - Definition
Bit0----EXI0	Bit1----EXI1	Bit2----EXI2	Bit3----EXI3
Bit4----EXI4	Bit5----EXI5	Bit6----EXI6	Bit7----EXI7
Bit8----EXI8	Bit9----EXI9	Bit10----EXI10	Bit11----EXI11
Bit12----EXI12	Bit13----EXI13	Bit14----EXI14	Bit15----EXI15

Sample 9-1 If EXI5 is at high level, set EXO0 at high level.

```
void main()
{
    short rtn, ex_inp;
    rtn=GT_ExInpt(&ex_inp);      error(rtn);
    if(ex_inp&0x20)
    {
        rtn=GT_ExOpt(0x1);       error(rtn);
        }
  }
```

Part Two

Description of Library Functions

Chapter Ten List of Functions

Chapter ElevenDescription of Functions

Chapter Ten List of Functions

Function	Description
GT_AbptStp()	Stop the motion of current axis abruptly.
GT_AuStpOff()	Enable automatic stopping when motion error.
GT_AuStpOn()	Disable automatic stopping when motion error.
GT_AuUpdtOff()	Disable automatic updating of parameters and commands of current axis.
GT_AuUpdtOn()	Enable automatic updating of parameters and commands of current axis.
GT_Axis()	Set a control axis as current target axis.
GT_AxisI()	Set the control axis sending interrupt request as current axis(prohibited in Windows environment).
GT_AxisOff()	Make the current axis in servo-off condition.
GT_AxisOn()	Make the current axis in servo-on status.
GT_BrkOff()	Clear the breakpoint of current axis and close breakpoint mode.
GT_CaptHome()	Allow the current axis to capture position when HOME signal is activated.
GT_CaptIndex()	Allow the current axis to capture position when INDEX signal is activated.
GT_CaptProb()	Enable probe capture function.
GT_Close()	Close the motion controller.
GT_CloseLp()	Set the current axis in close loop servo control.
GT_ClearInt	Clear interrupt generated by control card.
GT_ClrEncPos()	Clear the auxiliary encoder position value.
GT_ClrSts()	Clear the status of current axis.
GT_CtrlMode()	Set the control output of current axis as analog output or pulse output.
GT_DrvRst()	Reset the servo driver of current axis.
GT_EncPos()	Get the position of auxiliary encoder.
GT_EncSns()	Set the counting direction of encoder.
GT_EncVel()	Get the velocity of auxiliary encoder.
GT_EStpMtn()	Stop the coordinate system motion abruptly.
GT_EvntIntr()	Set the interrupt of the controller to the host as axis event interrupt.
GT_ExInpt()	Get the value of general purposed input port.
GT_ExOpt()	Set the value of general purposed output port.

Function	Description
GT_ExtBrk()	Set breakpoint triggering mode of the current axis home point signal.
GT_GetAcc()	Get the setting acceleration of current axis.
GT_GetAdc()	Get the AD conversion result.
GT_GetAddr()	Get the communication base address of the motion controller (prohibited in Windows environment).
GT_GetAtlErr()	Get the actual position error of the current axis.
GT_GetAtlPos()	Get the actual position of current axis.
GT_GetBgCommand Result()	Get the execution result of controller background command set.
GT_GetBrkCn()	Get the breakpoint position comparison value of current axis.
GT_GetCapt()	Get the INDEX or HOME captured position of current axis.
GT_GetCmdSts()	Get the execution status of the previous control command.
GT_GetCurrentCard No()	Get the card ID of current control card.
GT_GetEncCapt()	Get the captured value of auxiliary encoder.
GT_GetEncSts()	Get the status of auxiliary encoder.
GT_GetILmt()	Get the servo filter error integral limit of current axis.
GT_GetIntgr()	Get the position error integral of current axis.
GT_GetIntr()	Get the interrupt event of current axis (prohibited in Windows environment).
GT_GetIntrMsk()	Get the interrupt mask word of current axis.
GT_GetIntrTm()	Get the timer set value of time interrupt.
GT_GetJerk()	Get the jerk setting value of current axis.
GT_GetKaff()	Get the servo filter acceleration feedback coefficient of current axis.
GT_GetKd()	Get the servo filter differential coefficient of current axis.
GT_GetKi()	Get the servo filter integral coefficient of current axis.
GT_GetKp()	Get the servo filter gain of current axis.
GT_GetKvff()	Get the servo filter velocity feed coefficient of current axis.
GT_GetLmtSwt()	Get the status of limit switch.
GT_GetMAcc()	Get the setting maximum acceleration of current axis.
GT_GetMode()	Get the mode word of current axis.
GT_GetMtrBias()	Get the setting DAC bias of current axis.
GT_GetMtrCmd()	Get the motor control value of current axis in open loop mode.
GT_GetMtrLmt()	Get the servo filter output limit of current axis.
GT_GetPos()	Get the setting position of current axis.
GT_GetPosErr()	Get the servo filter position error limit of current axis.
GT_GetRatio()	Get the electronic gear ration of current axis.
GT_GetSmplTm()	Get the servo sampling period of controller.

Function	Description
GT_GetSts()	Get the status word of current axis.
GT_GetVel()	Get the setting velocity of current axis.
GT_HardRst()	Reset the motion controller with hardware.
GT_Home()	Start automatic HOME capture.
GT_HookIsr()	Hook interrupt service routine for the controller.
GT_Index()	Set the automatic HOME capture mode (Single HOME or HOME+INDEX).
GT_LmtSns()	Set the triggering level of limit switch.
GT_LmtsOff()	Disable the limit switch of current axis.
GT_LmtsOn()	Make effective the limit switch of current axis.
GT_MltiUpdt()	Update parameters of several control axes.
GT_MtnBrk()	Set the breakpoint triggering mode of motion arrival of current axis.
GT_NegBrk()	Set the negative position breakpoint mode of current axis.
GT_Open()	Open the motion controller.
GT_OpenLp()	Set the current axis in open loop control.
GT_PosBrk()	Set the positive position breakpoint mode of current axis.
GT_PrflG()	Set the motion mode of current axis as electronic gear mode.
GT_PrflS()	Set the motion mode of current axis as S-curve mode.
GT_PrflT()	Set the motion mode of current axis as T-curve mode.
GT_PrflV()	Set the motion mode of current axis as velocity control mode.
GT_Reset()	Reset the motion controller.
GT_RstIntr()	Reset the current axis interrupt event (prohibited in Windows environment).
GT_RstSts()	Reset the status of current axis.
GT_SetAcc()	Set the acceleration of current axis (T-curve and velocity control modes).
GT_SetAdcChn()	Set the number of AD conversion channels.
GT_SetAddr()	Set the communication base address of the motion controller (Prohibited in Windows environment).
GT_SetAtlPos()	Set the actual position of current axis.
GT_SetTime()	Adjust the output pulse width.
GT_SetBgCommand Set()	Set the background execution command set at interrupt.
GT_SetBrkCn()	Set the comparison value of breakpoint position of current axis (used together with the positive or negative position breakpoint mode).
GT_SetEncCapt()	Set INDEX capture of auxiliary encoder.
GT_SetILmt()	Set the servo filter error differential limit of current axis.
GT_SetIntSyncEvent( )	Set synchronization event for the interrupt of motion controller.

Function	Description
GT_SetIntrMsk()	Set the interrupt mask word of current axis.
GT_SetIntrTm()	Set the time constant of time interrupt of controller.
GT_SetJerk()	Set the jerk of current axis (S-curve mode).
GT_SetKaff()	Set the servo filter acceleration feedback gain of current axis.
GT_SetKd()	Set the servo filter differential coefficient of current axis.
GT_SetKi()	Set the servo filter integral coefficient of current axis.
GT_SetKp()	Set the servo filter gain of current axis.
GT_SetKvff()	Set the servo filter velocity feed coefficient of current axis.
GT_SetMAcc()	Set the maximum acceleration of current axis (S-curve mode).
GT_SetMtrBias()	Set the DAC bias value of current axis.
GT_SetMtrCmd()	Set the motor control value of current axis in open loop mode.
GT_SetMtrLmt()	Set the servo filter output limit of current axis.
GT_SetPos()	Set the target position of current axis (S-curve and T-curve modes).
GT_SetPosErr()	Set the servo filter position error limit of current axis.
GT_SetRatio()	Set the electronic gear transmission ratio of current axis (electronic gear mode).
GT_SetSmplTm()	Set the servo sampling period of controller.
GT_SetVel()	Set the target velocity of current axis (S-curve, T-curve and velocity control modes).
GT_SmthStp()	Stop the motion of current axis smoothly.
GT_StepDir()	Set the output way of current axis in pulse output mode as “Pulse + Direction”.
GT_StepPulse()	Set the output way of current axis in pulse output mode as “Positive and Negative Pulse”.
GT_SynchPos()	Set the target position of current axis same as the actual position.
GT_SwitchtoCardNo()	Switch current card.
GT_TmrIntr()	Set the interrupt of the controller to host as time interruption.
GT_UnhookIsr()	Unhook the ISR held by the function GT_HookIsr for the control card and restore the original ISR.
GT_Update()	Update parameters of current axis.
GT_ZeroPos()	Reset the actual and target position of current axis to zero.

Chapter Eleven Description of Functions

 **Notice**	*Each interface function will be described as follows one by one in the alphabetic order.* *Except being stated specially, the* sample *programs are all written in BC3.1 language environment.* *For all the function return values, please refer to Chapter Two Function (Library Function) Return Values.*

GT_AbptStp

Function Prototype: short GT_AbptStp(void)

Description of Function: This function is used to stop the motion of current axis abruptly, and set the parameter of target velocity and actual velocity as zero. After being executed, this function will become effective for abrupt stop. The command GT_AbptStp() is effective in the four motion control modes for control axis.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SmthStp

Function Evoking: In the following sample, when detecting that EXI15 port of external IO is at logic 1, stop the motion of the first axis abruptly.

```
void main()
{
    short rtn,ex_data;
    rtn=GT_ExInpt(&ex_data);    error(rtn); //Get the status of input port.
    rtn=GT_Axis(1);             error(rtn); //Set the first axis as current axis.
     if(ex_data&0x8000)                     //Check EXI15 is at logic 1 or not.
    {
        rtn=GT_AbptStp();      error(rtn); //Abrupt stop.
        }
}
```

GT_AuStpOff

Function Prototype: short GT_AuStpOff(void)

Description of Function: This function clears the Automatic Stop when Error bit of the axis mode register. When this bit is cleared, if the current axis meets the preset conditions for motion error during the servo control process (i.e. the actual position error exceeds the value set by the function GT_SetPosErr()), the controller will only set the error flag of axis motion, rather than stop the servo motor control output of the axis automatically.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_AuStpOn

Function Evoking: The following sample disable the automatic stop function of the motor when the fourth axis has error in motion.

```
void main()
{
    short rtn;
    rtn=GT_Axis(4);       error(rtn);
    rtn=GT_AuStpOff ();   error(rtn);
    }
```

GT_AuStpOn

Function Prototype: short GT_AuStpOn(void)

Description of Function: This function sets the Automatic Stop when Error bit of the axis mode register. If the current axis meets the preset conditions for motion error during the servo control process (i.e. the actual position error exceeds the value set by the function GT_SetPosErr()), the controller will stop the servo motor control output of the axis and stop the motor.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_AuStpOff

Function Evoking: The following sample sets the automatic stop symbol of the motor when the third axis has error in motion.

```
void main()
{
    short rtn;
    rtn=GT_Axis(3);   error(rtn);
    rtn=GT_AuStpOff ();   error(rtn);
    }
```

GT_AuUpdtOff

Function Prototype: short GT_AuUpdtOff(void)

Description of Function: This function clears the Automatic Updating bit of the axis mode register of the current axis. After this bit is cleared, the breakpoint already meeting the triggering condition will only have breakpoint symbol set and parameters of control axis will not updated. This bit will remain until the host evokes the function GT_AuUpdtOn().

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_AuUpdtOn

Function Evoking: This sample clears the Automatic Updating bit of the axis mode register of the second axis.

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);     error(rtn);
```

```
    rtn=GT_AuUpdtOff ();    error(rtn);
}
```

GT_AuUpdtOn

Function Prototype: short GT_AuUpdtOn(void)

Description of Function: This function sets the Automatic Updating bit of the axis mode register of the current axis to 1. After this function becomes effective, the breakpoint meeting the triggering condition will update automatically the parameters of control axis. This bit will remain until the host evokes the function GT_AuUpdtOff().

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_AuUpdtOff

Function Evoking: This sample clears the Automatic Updating bit of the mode register of the first axis.

```
void main()
{
    short rtn;
    rtn=GT_Axis(1);      error(rtn);
    rtn=GT_AuUpdtOn();    error(rtn);
    }
```

GT_Axis

Function Prototype: short GT_Axis(unsigned short num)

Description of Function: This function sets the specified axis as current axis. The commands related to control axis evoked later are all for the specified axis. The current axis will not change until this command of the control axis is evoked again. The parameter “num” means the specified axis number, the value of which is from 1, 2, 3 to 4, representing the first, second, third and fourth axis respectively.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Function Evoking: In the following sample, the second axis is set as current axis and the following commands are all for the second axis.

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);      error(rtn);
    rtn=GT_Setpos(1000);      error(rtn);
    rtn=GT_SetVel(10);       error(rtn);
    rtn=GT_SetAcc(1);      error(rtn);
    }
```

GT_AxisI

Function Prototype: short GT_AxisI(void);

Description of Function: When receiving an event interrupt request, the host can evoke this function to set the axis sending the interrupt request as current axis. The commands evoked later are all for the axis generating the interrupt request, until another current axis assign command is evoked.

If the motion controller is in the mode of allowing event interrupt request to the host, once the controller encounters an event satisfying interrupt condition, it will send the interrupt request to the host immediately, and the host shall treat this request. Now, evoke the command GT_AxisI() and the motion controller will set the control axis sending the interrupt request as the current axis automatically, and not set each axis any more to check whether an interrupt happens, which allows the host to get the interrupt status quickly and perform corresponding treatment.

System: DOS

Applicable Card: All GT series cards

Function Evoking: The following sample treats the positive limit switch event of the axis generating interrupt in the interrupt routine.

```
void interrupt handler(...)
{
    short rtn;
    unsigned short intr_sts;
    long actl_pos, pos;
    disable();
    rtn=GT_AxisI();      if(rtn!=0) return;
    rtn=GT_GetIntr(&intr_sts);      if(rtn!=0) return;
    if (intr_sts & 0x20)    // positive limit switch error
      {
        rtn=GT_GetAtlPos(&actl_pos);    if(rtn!=0) return;
       pos=actl_pos-20000;
        rtn=GT_SetPos(pos);    if(rtn!=0) return;
       rtn=GT_Update();   if(rtn!=0) return;
        rtn=GT_RstIntr(0x9f);  if(rtn!=0) return;
    }
    enable();
    return;
}
```

GT_AxisOff

Function Prototype: short GT_AxisOff(void);

Description of Function: This function sets the current axis in uncontrolled status and disables the driver.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_AxisOn

Evoke Function: This sample sets the second axis as in the uncontrolled status and disable the driver.

```
void main()
{
```

```
    short rtn;
    rtn=GT_Axis(2);   error(rtn);
    rtn=GT_AxisOff();     error(rtn);
   }
```

GT_AxisOn

Function Prototype: short GT_AxisOn(void);

Description of Function: This function sets the current axis in controlled status and enables the driver.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_AxisOff

Evoke Function: This sample sets the second axis as in the controlled status after digital filter parameters are set for it, and activates the axis driver servo.

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);       error(rtn);
    rtn=GT_SetKp(10);     error(rtn);
    rtn=GT_Update();          error(rtn);
    rtn=GT_AxisOn();      error(rtn);
   }
```

GT_BrkOff

Function Prototype: short GT_BrkOff(void);

Description of Function: This function clears the breakpoint already set for the current axis, but not triggered yet.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_BrkOn

Evoke Function: This sample clears the breakpoint set for the first axis, but not triggered yet.

```
void main()
{
    short rtn;
    rtn=GT_Axis(1); error(rtn);
    rtn=GT_BrkOff();      error(rtn);
    }
```

GT_CaptHome

Function Prototype: short GT_CaptHome(void);

Description of Function: This function sets the position capture event of Home signal. After evoking this function, the position capture register will record the actual position when Home signal is activated. Executing GT_CaptHome() once only captures the position information of

Home signal once. To capture the position of next Home signal, user must clear the corresponding status symbol bit, i.e. evoking the function of GT_ClrSts() or GT_RstSts(), and use GT_CaptHome() to set the position capture event of Home signal. The process of capturing Home signal is done by hardware, irrelevant with the velocity of motion axis.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_CaptIndex, GT_CaptProb

Evoke Function: The following sample sets the position capture event of Home signal for the third axis.

```
void main()
{
    short rtn;
    rtn=GT_Axis(3);   error(rtn);
    rtn=GT_ClrSts();  error(rtn);
    rtn=GT_CaptHome();    error(rtn);
    }
```

GT_CaptIndex

Function Prototype: short GT_CaptIndex(void);

Description of Function: This function sets the position capture event of Index signal. After evoking this function, the position capture register will record the actual position when Index signal is activated. Executing GT_CaptIndex() once only captures the position information of Index signal once. To capture the position of next Index signal, user must clear the corresponding status symbol bit, i.e. evoking the function of GT_ClrSts() or GT_RstSts(), and set the position capture event of Index signal. The actual position captured can be used as the accurate coordinate home point of the motion axis. The process of capturing Home signal is done by hardware, irrelevant with the velocity of motion axis.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_CaptHome, GT_CaptProb

Evoke Function: The following sample sets the position capture event of Index signal for the third axis.

```
void main()
{
    short rtn;
    rtn=GT_Axis(3);       error(rtn);
    rtn=GT_ClrSts();      error(rtn);
    rtn=GT_CaptIndex(); error(rtn);
    }
```

GT_CaptProb

Function Prototype: short GT_CaptProb(void);

Description of Function: This function sets the capture probe input signal event. The motion controller uses the channel 0 (EXI0) of general purposed input as probe input. After this

function is evoked, the position capture register of all control axes and the position capture register of auxiliary encoder will record the actual position when the probe signal comes.

When the capture event happens, the symbol (i.e. bit3) indicating that a capture happens in the status characters of all the control axes will be set.

Executing GT_CaptProb() once only captures the position information of one probe input signal. To capture the position of next signal, user must evoke this function again to set the capture probe input signal event. The actual position captured can be used as the accurate coordinate home point of the control axis. The process of capturing Home signal is done by hardware,, irrelevant with the velocity of motion axis.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_CaptHome, GT_CaptIndex

Evoke Function: The following sample sets a capture probe input signal event, detect status during motion, get and print the actual position captured by the second axis when capturing event happens.

```
void main()
{
    short rtn;
    unsigned short status;
    long actl_pos;
    rtn=GT_CaptProb();   error(rtn);
    ….
    rtn=GT_Axis(2); error(rtn);
    rtn=GT_GetSts(&status);   error(rtn);
    while(status&0x400)
    {
        if(status&0x4)
        {
            rtn=GT_GetAtlPos(&actl_pos);        error(rtn);
            printf(“the capture pos of axis 2 is: %ld\n”,actl_pos);
            break;
        }
        rtn=GT_GetSts(&status);       error(rtn);
    }
}
```

GT_Close

Function Prototype: short GT_Close(void);

Description of Function: Close the motion controller. Generally this function this command is evoked at the end of user program.

System: DOS (PCI Bus), WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_Open

GT_CloseLp

Function Prototype: short GT_CloseLp(void);

Description of Function: This function sets the current axis as in close loop control. When the system is powered, the default status of each control axis is close loop control. If the current axis has been activated (i.e. the function GT_AxisOn() was evoked.), but is still in open loop status, evoking this function may cause the motor to stop abruptly sometimes. This is due to that, in the current open loop, the motor may float, causing a quite large deviation between the actual position and target position of the motor. So, when the current axis has been activated, before evoking this function, it is better to prohibit the current axis activation (i.e. evoking the function GT_AxisOff()) first, and then evoke the function GT_SynchPos() to make the target position equal to the actual position.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_OpenLp

Evoke Function: Without any danger, the following sample sets the first axis as in close loop control status .

```
void main()
{
    short rtn;
    rtn=GT_Axis(1);       error(rtn);
    rtn=GT_AxisOff();     error(rtn);
    rtn=GT_SynchPos();    error(rtn);
    rtn=GT_Update();          error(rtn);
    rtn=GT_ClosLp();          error(rtn);
    }
```

GT_ClearInt

Function Prototype: short GT_ClearInt(unsigned short CardNo)

Description of Function: Use this function to clear the interrupt state of the control card. In the end of the interrupt service routine (ISR), user must evoke this function to recover the state of control card from interrupt to normal.

Function Parameter: CardNo is the corresponding control card number. Please refer to GT_SwitchtoCardNo for detail. When a single card is in use, this value will always be 0.

Function Return: 0 means success and -1 means failure.

Function Evoking: Refer to DOS interrupt treatment.

Applicable Card: PCI Bus Card

System: DOS

Function Evoking: GT_ClearInt is only effective in the interrupt service routine. The typical use is as follows:

```
void interrupt    My_Isr(...) //ISR to be used.
{
    ....
```

```
    ...//User code
    GT_ClearInt(0);
    outportb(0x20,0x20);
    outportb(0xa0,0x20); //State interrupt service ending to interrupt controller.
}
```

GT_ClrEncPos

Function Prototype: short GT_ClrEncPos(unsigned short EncNum)

Description of Function: Clear the value in the auxiliary encoder specified by EncNum.

Function Parameter: The parameter Enc_Num indicates the auxiliary encoder number requiring to clear its position. The motion controller has two auxiliary encoders, No. 1 and No. 2 .

Function Return: 0 means success and -1 means failure.

Applicable Card: The card with this selectable function.

System: DOS, WINDOWS

GT_ClrSts

Function Prototype: short GT_ClrSts(void);

Description of Function: This function clears the event status bit of the current axis, but doesn't affect the interrupt triggered by event of the motion controller. It just clears the event related bit of the current axis status word, and enables the setting of a bit when the next new event happens. For the definition of each bits of status word, please refer to *Table 11-1*.

After executing GT_ClrSts(), Bit0~Bit7 is cleared. When new event happens next time, the corresponding bit will be set to "1". Bit8-bit15 is controlled wholly by the motion controller, which this function doesn't affect.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Function Evoking: The following sample clears the event status bit of the fourth axis.

```
void main()
{
    short rtn;
    rtn=GT_Axis(4);   error(rtn);
    rtn=GT_ClrSts();    error(rtn);
    }
```

Table 11-1 Defintion of Each Bit in Status Register

Bit	Definition
0	Motion completion symbol bit. If the motion of control axis is completed, this bit = 1. This symbol is ineffective in electronic gear motion mode.
1	Alarm bit of motor servo driver. If the driver of control axis alarms, the bit = 1.
2	Breakpoint arrival bit. If the breakpoint is enabled, when any conditions are met, this bit is set to 1.
3	Index/Home triggering bit. After setting a position capture command, when the controller detects required Index/Home capture conditions, this bit = 1.

Bit	Definition
4	Motion error bit. If the position error exceeds the allowed scope (Refer to 4.4.7.3 Description.), the controller will set this bit as 1. Only when the controller is not in motion error status any more, can this bit be cleared.
5	Triggering bit of positive limit switch. If the switch is triggered, this bit = 1.
6	Triggering bit of negative limit switch. If the switch is triggered, this bit = 1.
7	Command error bit. If there is error command, the controller will set it as 1.
8	Open loop/close loop status of motor (1 means close loop and 0 means open loop.)
9	Motor servo activation/prohibit status (1 means activation and 0 means prohibit.)
10	Motion status symbol bit. It indicates continuously whether the control axis is in motion. If it is in motion, the bit = 1. If in static, the bit = 0.
11	Limit switch activation/prohibit status (1 means activation and 0 means prohibit.)
12 13	The number symbol of current axis (13bit = high level and 12bit = low level.). The coding of the current axis numbers is as follows. Bit 13 Bit12 Axis 0 0 1 0 1 2 1 0 3 1 1 4
14	Set Home switch signal capture symbol.
15	Set Index signal capture symbol.

GT_CtrlMode

Function Prototype: short GT_CtrlMode(int mode);

Description of Function: This function sets the output mode of current axis as analog voltage output or pulse output.

Function Parameter: "mode" means motion control output mode. There are two modes, 0 means the analog voltage output mode and 1 means pulse output mode. The default status of the motion controller is the analog voltage output mode.

System: DOS, WINDOWS

Applicable Card: SV

Function Evoking: The following sample sets the first axis as in pulse output mode to control the step motor.

```
void main()
{
    short rtn;
    rtn=GT_Axis(1);   error(rtn);
    rtn=GT_CtrlMode(1);     error(rtn);
    }
```

GT_DrvRst

Function Prototype: short GT_DrvRst(void);

Description of Function: When the driver of current axis sends fault alarm, after evoking this

function, the controller will send resetting signal to the axis driver to reset the driver. Before evoking this function, the current axis has to be in drive prohibition status. Otherwise, the command is ineffective.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Function Evoking: In the following sample, if the fourth axis generates drive alarm signal, reset the driver and clear alarm signal.

```
void main()
{
    short rtn;
    unsigned short status;
    rtn=GT_Axis(4);   error(rtn);
    rtn=GT_GetSts(&status);   error(rtn);
    if(status&0x2)
    {
        rtn=GT_DrvRst();      error(rtn);
        rtn=GT_RstSts(0xfffd);      error(rtn);
     }
}
```

GT_EncPos

Function Prototype: short GT_EncPos(short Enc_Num, long* Actl_pos);

Description of Function: This function is to get the actual position of auxiliary encoder.

The parameter Enc_Num means the auxiliary encoder number to be gotten. The motion controller has two routes of auxiliary encoder, No. 1 and No. 2. The parameter *Actl_pos gets the actual position of specified auxiliary encoder.

System: DOS, WINDOWS

Applicable Card: SV, SG, SP

Relevant Function: GT_EncVel

Function Evoking: The following sample gets and prints the position of auxiliary encoder No. 2.

```
void main()
{
    short rtn;
    long actl_pos;
    rtn=GT_EncPos(2, &actl_pos);        error(rtn);
    printf("the actual position of assistant encoder 2 is: %ld\n", actl_pos);
 }
```

GT_EncSns

Function Prototype: short GT_EncSns(unsigned int Sense);

Description of Function: The motion controller requires the positive direction of the control axis (motor) consistent with that of the corresponding encoder, so as to form correct negative feedback. If the wrong connection or other reasons causes the two directions opposite to each

other, user can evoke this function to use a software method to verify the positive counting direction of encoder, to make it consistent with the positive direction of the motor motion.

Bit0 to bit6 in the parameter Sense indicate that whether the counting direction of each corresponding axis encoder needs to be revised, as listed in Table 2-3. In the table, if a corresponding bit is set to 0, it means that the counting direction of corresponding axis encoder will not be modified. Whereas, if it is set to 1, it means that the counting direction of corresponding axis encoder will be modified. All the bits are 0 by default in register.

System: DOS, WINDOWS

Applicable Card: SV

Function Evoking: The following sample reverses the direction of the first and second axes, and keeps the direction of the third and fourth axis encoder, and auxiliary encoder No. 1 and No.2 the same.

```
void main()
{
    short rtn;
    rtn=GT_EncSns(3);    error(rtn);
    }
```

Table 11-2 Definition of Bits in Encoder Register

Bit	Description	Definition
6-15	Not used. Set to 0.	
5	Auxiliary encoder No. 2	0：No change 1：Reversed
4	Auxiliary encoder No. 1	0：No change 1：Reversed
3	Axis#4	0：No change 1：Reversed
2	Axis#3	0：No change 1：Reversed
1	Axis#2	0：No change 1：Reversed
0	Axis#1	0：No change 1：Reversed

GT_EncVel

Function Prototype: short GT_EncVel(short Enc_Num, double* Actl_vel);

Description of Function: This function is to get the actual velocity of auxiliary encoder.

The parameter Enc_Num indicates the number of auxiliary encoder to be gotten. The motion controller has two routes of auxiliary encoder, No. 1 and No. 2. The parameter *Actl_vel gets back the actual position of specified auxiliary encoder.

System: DOS, WINDOWS

Applicable Card: SV, SG, SP

Relevant Function: GT_EncPos

Function Evoking: The following sample gets and prints the velocity of auxiliary encoder No. 2.

```
void main()
{
    short rtn;
    double actl_vel;
    rtn=GT_EncVel(2, &actl_vel);    error(rtn);
```

```
        printf("the actual velocity of assistant encoder 2 is: %f\n", actl_vel);
}
```

GT_EStpMtn

Function Prototype: short GT_EStpMtn(void);

Description of Function: This function stops abruptly the multiple-axis coordination motion command based on the coordinate system. After the function is executed, there is no velocity decelerating course, and the synthesized velocity of current motion and the target synthesized velocity becomes zero immediately. In the buffer command execution status, after the motion stops, it will save necessary information when the motion scope, so as to evoke the function GT_StrtMtn () to continue the execution of buffer command. In the abrupt command input execution status, after the motion stops, the information of current motion will be abandoned.

User cannot send interrupt command when executing positioning instruction, that is, the process moving to the positioning point cannot be interrupted. Now, sending interrupt command will cause motion error.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_StpMtn

Function Evoking: The following sample stops abruptly the motion in coordiate system in executing the cmmmand in buffer.

```
void main()
{
    short rtn;
    rtn=GT_StrtList();          error(rtn);
    rtn=GT_MvXYZ(0,0,0,16,3.7);     error(rtn);
    rtn=GT_LnXYZ(1234,5678,9013);   error(rtn);
    rtn=GT_StrtMtn();           error(rtn);
    rtn=GT_ArcXY(2345,6789,360);    error(rtn);
    ……
    rtn=GT_EndList();   error(rtn);
    rtn=GT_EStpMtn();   error(rtn);
    }
```

GT_EvntIntr

Function Prototype: short GT_EvntIntr(void);

Description of Function: This function sets the interrupt request by the motion controller to the host as axis event interrupt.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_TmrIntr

Function Evoking: GT_EvntIntr()

GT_ExInpt

Function Prototype: short GT_ExInpt (unsigned short * Data);

Description of Function: This function gets the status of general purposed input of the motion controller.

Function Parameter: *Data returns the status. The corresponding relation of each bit to general purposed input port is as follows.

Bit-Definition	Bit-Definition	Bit-Definition	Bit-Definition
Bit0----EXI0	Bit1----EXI1	Bit2----EXI2	Bit3----EXI3
Bit4----EXI4	Bit5----EXI5	Bit6----EXI6	Bit7----EXI7
Bit8----EXI8	Bit9----EXI9	Bit10----EXI10	Bit11----EXI11
Bit12----EXI12	Bit13----EXI13	Bit14----EXI14	Bit15----EXI15

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_ExOpt

Function Evoking: The following sample gets the status of general purposed input. If EXI8 is at logic 1, stop the motion in coordinate system.

```
void main()
{
    short rtn, ex_inp;
    rtn=GT_ExInpt(&ex_inp);    error(rtn);
    if(ex_inp&0x100)
    {
        rtn=GT_StpMtn();      error(rtn);
        }
}
```

GT_ExOpt

Function Prototype: short GT_ExOpt(unsigned short Data);

Description of Function: This function sets the status of general purposed input of the motion controller.

Function Parameter: Data is the status to be set. The corresponding relation of each bit to general purposed input port is as follows.

Bit-Definition	Bit-Definition	Bit-Definition	Bit-Definition
Bit0----EXO0	Bit1----EXO1	Bit2----EXO2	Bit3----EXO3
Bit4----EXO4	Bit5----EXO5	Bit6----EXO6	Bit7----EXO7
Bit8----EXO8	Bit9----EXO9	Bit10----EXO10	Bit11----EXO11
Bit12----EXO12	Bit13----EXO13	Bit14----EXO14	Bit15----EXO15

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_ExInpt

Function Evoking: The following sample sets the general purposed output EXO0 at logic 1 if the general purposed input EXI5 is at logic 1.

```
void main()
{
    short rtn, ex_inp;
    rtn=GT_ExInpt(&ex_inp);      error(rtn);
    if(ex_inp&0x20)
    {
        rtn=GT_ExOpt(0x1);       error(rtn);
        }
}
```

GT_ExtBrk

Function Prototype: short GT_ExtBrk(void);

Description of Function: This function sets the breakpoint mode of current axis as home point switch triggering breakpoint mode. After evoking the function, when the controller detects that the axis home point switch is effective, trigger the breakpoint to set the breakpoint symbol of axis status register as 1 and clear the breakpoint mode, i.e. except the host sets the breakpoint again, the host will not trigger this breakpoint any more. Meanwhile, when the automatic update symbol of axis mode status register is 1, the controller will update all the dual-buffer structure parameters and commands of the current axis.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Function Evoking: The following sample sets the breakpoint mode of the second axis as the home point switch triggering breakpoint mode, and automatically updates the command of smooth stop when the home point switch is triggered.

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);          error(rtn);
    rtn=GT_AuUpdtOn();       error(rtn);
    rtn=GT_CaptHome();       error(rtn);
    rtn=GT_ExtBrk();             error(rtn);
    rtn=GT_PrflT();          error(rtn);
    rtn=GT_SetPos(87654);    error(rtn);
    rtn=GT_SetVel(32);       error(rtn);
    rtn=GT_SetAcc(1.1);          error(rtn);
    rtn=GT_Update();             error(rtn);
    rtn=GT_SmthStp();        error(rtn);
   }
```

GT_GetAcc

Function Prototype: short GT_GetAcc(double * Acc);

Description of Function: This function gets the acceleration of current axis set by the function GT_SetAcc().

Function Parameter: *Acc returns the acceleration.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetAcc

Function Evoking: The following sample gets the acceleration set for the third axis.

```
void main()
{
    short rtn;
    double acc;
    rtn=GT_Axis(3);        error(rtn);
    rtn=GT_GetAcc(&acc);   error(rtn);
    }
```

GT_GetAdc

Function Prototype: short GT_GetAdc(short Channel, short* Adc_Data);

Description of Function: This function gets the AD conversion result. The sampling frequency of eight channels under AD conversion is 770Hz.

Function Parameter: The parameter Channel sets AD channel number, with a value range of 1-8. The parameter *Adc_Data returns AD conversion result, with a value range of 0-2047 (indicating positive voltage: 0V to 10V) and 2048~4095 (indicating negative voltage: -0V to -10V). For evoking it, please refer to the sample.

System: DOS, WINDOWS

Applicable Card: Card with A/D function

Function Evoking: This function can be evoked at any time. The following sample gets the AD conversion result of channel two.

```
void main()
{
    short rtn;
    short Adc_Data;
    rtn=GT_GetAdc(2,&Adc_Data);
    if(rtn!=0) return;
    if(Adc_Data>2047)    Adc_Data = -10*(Adc_Data-2048)/2048;
    printf("the Adc_Data is: d%", Adc_Data);
 }
```

GT_GetAddr

Function Prototype: short GT_GetAddr(unsigned short * Base_addr);

Description of Function: This function gets the communication base address between the host and the motion controller set by the function GT_SetAddr() or the default base address (0x300) by the host at the time of being powered.

Function Parameter: *Base_addr returns the base address used in current communication. If user used GT_SetAddr() to set a base address, but this address has an error, what the parameter

returns will also be this wrong base address.

Function Return: The difference between this function and other functions is that, because there is no data exchange with the motion controller, the return value of this function will always be 0.

System: DOS

Applicable Card: ISA Bus Card

Relevant Function: GT_SetAddr

Evoke Function: The following sample gets and prints the base address.

```
void main()
{
    short rtn;
    unsigned shor base_addr;
    rtn=GT_GetAddr(&base_addr); error(rtn);
    printf("the base address is: %d\n",base_addr);
}
```

GT_GetAtlErr

Function Prototype: short GT_GetAtlErr (short * Aerr);

Description of Function: This function gets the actual position error of current axis, i.e. the difference between the current planned position and the current actual position. This function is usually used to monitor the servo control position error.

Function Parameter: *Aerr returns the actual position error.

System: DOS, WINDOWS

Applicable Card: SV

Function Evoking: The following sample gets and prints the actual position error of the second axis.

```
void main()
{
    short rtn, actl_err;
    rtn=GT_Axis(2);        error(rtn);
    rtn=GT_GetAtlErr(&actl_err);        error(rtn);
    printf("the actual error is: %d\n",actl_err);
}
```

GT_GetAtlPos

Function Prototype: short GT_GetAtlPos(long * Apos);

Description of Function: This function gets the actual position of current axis.

Function Parameter: *Apos returns the actual position.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetAtlPos

Function Evoking: The following sample gets and prints the actual position of the first axis.

```
void main()
{
```

```
    short rtn;
    long actl_pos;
    rtn=GT_Axis(2); error(rtn);
    rtn=GT_GetAtlPos(&actl_pos);        error(rtn);
    printf("the actual pos is: %ld\n",actl_pos);
}
```

GT_GetBgCommandResult

Function Prototype: short WINAPI GT_GetBgCommandResult (PBGCOMMANDSET BgCmdset, ULONG CmdsetSize);

Description of Function: Get the background command set execution result of the controller.

Function Parameter: pBgCmdset is the start address of the background command set. The variable result of the structure GENERAL_COMMAND saves the command execution result. 0 means success in executing command, -1 means failure in executing relevant command. The parameter CmdsetSize indicates the space size of the background command set, with a unit in byte.

Function Return: 0 means success and -1 means failure.

System: WINDOWS

Applicable Card: PCI Bus Card

Relevant Function: GT_SetBgCommandSet

GT_GetBrkCn

Function Prototype: short GT_GetBrkCn(long * Brk);

Description of Function: This function gets the breakpoint position of the current axis set by the function GT_SetBrkCn().

Function Parameter: *Brk returns the breakpoint position.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetBrkCn

Function Evoking: GT_GetBrkCn()

GT_GetCapt

Function Prototype: short GT_GetCapt (long * Capt);

Description of Function: This function gets the position value where Index or Home signal captured by the current axis appears. This value kept the same in motion situation until the next capture event happens.

Function Parameter: *Capt returns the captured position value by the current axis.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_CaptHome，GT_CaptIndex，GT_CaptProb

Function Evoking: The following sample gets the position value of the point where Home signal is captured by the third axis and keeps the position stay at the point.

```
    void main()
```

```
{
    short rtn;
    unsigned short status;
    long capt_pos;
    rtn=GT_GetSts(&status);   error(rtn);
    if(status&0x8)
    {
        rtn=GT_GetCapt(&capt_pos);   error(rtn);
        rtn=GT_SetPos(capt_pos);     error(rtn);
        }
}
```

GT_GetCmdSts

Function Prototype: short GT_GetCmdSts(unsigned short * Cstatus);

Description of Function: This function gets the contents of the command status register of the controller.

Function Parameter: *Cstatus returns the value of the register. In main user program, detect the return value of each function to tell the execution status of communication and command. If the command return value is 1, it indicates that the command from the host is wrong, and the controller reports error in the command. Once a command error happens, the motion controller will neglect these illegal motion commands. The host can evoke this command to get the reason for command error. This register is a 16-bit register. The definition of each bit is in the following table.

Bit	Definition
Bit0	1: Overflow of control parameters input by command. The commands generating error include GT_SetPos(), GT_SetVel(), GT_SetAcc(), GT_SetAtlPos(), etc.
Bit1	1: Illegal control parameters input by command. The commands generating error include GT_SetVel(), GT_SetAcc(), GT_SetJerk(), GT_SetMAcc(), GT_SetMtrLmt(), GT_SetKp(), GT_SetKi(), GT_SetKd(), GT_SetKvff(), GT_SetKaff(), GT_SetILmt(), GT_SetPosErr(), etc.
Bit2	1: The host sends the command GT_MltiUpdt (value), but value=0.
Bit3	1: Illegal usage of GT_DrvRst(). When the current axis is in the servo activation status, the host sending this command will generate error symbol.
Bit4	1: No event interrupt from the controller, but the host sends the command concerning interrupt.
Bit5	(Blank)
Bit6	1: When the current axis is in motion, the host sends a command to modify the work mode of the current axis (except that the current axis is in the electronic gear mode).
Bit7	1: The position capture completion symbol of the current axis status register is 1, or when the command GT_CaptIndex() (GT_CaptHome()) has been set but the position has not been captured yet, the host sends the command

Bit	Definition
	GT_CaptIndex().
Bit8	1: The position capture completion symbol of the current axis status register is 1, or when the command GT_CaptIndex() (GT_CaptHome()) has been set but the position has not been captured yet, the host sends the command GT_CaptHome().
Bit9	1: When the driver alarm symbol of the current axis status register is 1, the host sends the command GT_AxisOn().
Bit10	(Blank)
Bit11	1: When the “motion status symbol” in the current axis status register is 1, the host sends the command GT_ZeroPos() and sets this bit to 1. When the motion mode of current axis is the velocity control mode, the host sends the command GT_SynchPos() and makes it effective, and this bit is set to 1. When the “motion status symbol” of the current axis status register is 1, use GT_Update() or GT_MltiUpdt() to modify the control parameters of current axis. But these parameters cannot be modified in the current motion mode (for example, in S-curve motion mode and when the motor is in motion, the host sends the command GT_SetVel(), and GT_Update() or GT_MltiUpdt()).
Bit12	1: Illegal commands for coordinate system, including: The physical axis mapped is in motin when the coordinate system is being built. When the buffer command is in execution, evoke the command GT_StrtMtn(). When the buffer command is in execution, evoke the command GT_StrtList(). When the buffer command is being input, evoke the command GT_AddList (). When the immediate command input is in execution and the motion is not yet finished, evoke a new motion description command.
Bit13	1: Error in evoking the commands GT_MvXY (), GT_MvXYZ() and GT_MvXYZA().
Bit14	(Blank)
Bit15	1: The buffer is full. Now, since the buffer is full, the motion description command evoked just now has not be received by the motion controller, and the host has to evoke the command repeatedly until it is received.

System: DOS, WINDOWS
Applicable Card: All GT series cards
Function Evoking: This function can be evoked at any time.

GT_GetCurrentCardNo

Function Prototype: short GT_GetCurrentCardNo(void);

Description of Function: This function gets the current control card number, ranged from 0 to 15.

System: DOS, WINDOWS

Applicable Card: PCI Bus Card

Relevant Function: GT_SwitchtoCardNo

Function Evoking: GT_GetCurrentCardNo (1)

GT_GetEncCapt

Function Prototype: short GT_GetEncCapt(long *value);

Description of Function: This function gets the actual count value of the auxiliary encoder when INDEX signal is captured.

When INDEX signal is not captured, evoke the previous actual count value gotten by this function.

Function Parameter: *value is the actual position of auxiliary encoder when the signal is captured.

System: DOS, WINDOWS

Applicable Card: Select GT series cards with “INDEX signal capture function of auxiliary encoder”.

Relevant Function: GT_GetEncSts, GT_SetEncCapt

GT_GetEncSts

Function Prototype: short GT_GetEncSts(unsigned short *value);

Description of Function: This function gets the capture status of auxiliary encoder.

Function Parameter: Bit3 in the returned parameter means whether INDEX signal is captured or not:

“0” means “not captured”, and “1” means “captured”.

Once the capture succeeds, this symbol bit (bit3) will be set (= 1). It will not be cleared (= 0) to the initial status of 0 until the function GT_SetEncCapt is evoked again.

System: DOS, WINDOWS

Applicable Card: Select GT series cards with “INDEX signal capture function of auxiliary encoder”.

Relevant Function: GT_GetEncCapt, GT_SetEncCapt

GT_GetILmt

Function Prototype: short GT_GetILmt(unsigned short * Ilm);

Description of Function: This function the error integral limit of the current axis set by the function GT_SetILmt ().

Function Parameter: *Ilm returns the error integral limit.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_SetILmt

GT_GetIntgr

Function Prototype: short GT_GetIntgr(short * Intgr);
Description of Function: This function gets the integral error of the current axis servo filter.
Function Parameter: *Intgr returns the integral error.
System: DOS, WINDOWS
Applicable Card: SV

GT_GetIntr

Function Prototype: short GT_GetIntr(unsigned * Status);
Description of Function: This function gets the status character of the axis generating interrupt request. If the command is executed but the motion controller doesn't generate interrupt request, the function GT_GetIntr() will get the status character of the current axis.
Function Parameter: *Status returns the status character. For its meaning, please see *Table 11-1* of the function GT_ClrSts(). Bit 0-bit6 indicates the event related with interrupt. When the interrupt mask character corresponding to an event is set as allowing interrupt, if the interrupt condition is satisfied, the motion controller will send interrupt request to the host. After the host responds to this interrupt, evoke GT_GetIntr() to inquire the type of interrupt event.
System: DOS, WINDOWS
Applicable Card: All GT series cards
Special Notice: In WINDOWS environment, what the function GT_GetIntr gets is the interrupt status character when the previous interrupt is generated by the motion controller. Status = 0: The controller doesn't generate an interrupt, or the previous one is the time event. Status $\neq$ 0: For detailed meaning, please see the following table. Pay attention that, Bit12 and Bit13 in the status indicate the axis generating the previous event interrupt. In the interrupt service routine, evoke this function to tell the interrupt type.

GT_GetIntrMsk

Function Prototype: short GT_GetIntrMsk (unsigned short * Mask);
Description of Function: This function gets the interrupt mask character of the current axis set by the function GT_SetIntrMsk ().
Function Parameter: *Mask returns the interrupt mask character. For its meaning, please refer to the bit definition of the function GT_RstIntr() and description of the function GT_SetIntrMsk().
System: DOS, WINDOWS
Applicable Card: All GT series cards
Relevant Function: GT_SetIntrMsk

GT_GetIntrTm

Function Prototype: short GT_GetIntrTm(unsigned short * Timer);
Description of Function: This function gets the time constant of time interrupt of the motion

controller set by the function GT_SetIntrTm().

Function Parameter: *Timer returns the time constant.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetIntrTm

GT_GetJerk

Function Prototype: short GT_GetJerk (double * Jerk);

Description of Function: This function gets the jerk of the current axis set by the function GT_SetJerk ().

Function Parameter: The double-precision parameter *Jerk returns the jerk.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetJerk

GT_GetKaff

Function Prototype: short GT_GetKaff (unsigned short * Kaff);

Description of Function: This function gets the acceleration feedback gain of the current axis set by the function GT_SetKaff().

Function Parameter: *Kaff returns the acceleration feedback gain.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_SetKaff

GT_GetKd

Function Prototype: short GT_GetKd(unsigned short * Kd);

Description of Function: This function gets the differential gain of the current axis set by the function GT_SetKd().

Function Parameter: *Kd returns the differential gain.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_SetKd

GT_GetKi

Function Prototype: short GT_GetKi (unsigned short * Ki);

Description of Function: This function gets the integral gain of the current axis set by the function GT_SetKi ().

Function Parameter: *Ki returns the integral gain.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_SetKi

GT_GetKp

Function Prototype: short GT_GetKp(unsigned short * Kp);
Description of Function: This function gets the percentage gain of the current axis set by the function GT_SetKp().
Function Parameter: *Kp returns the percentage gain.
System: DOS, WINDOWS
Applicable Card: SV
Relevant Function: GT_SetKp

GT_GetKvff

Function Prototype: short GT_GetKvff (unsigned short * Kvff);
Description of Function: This function gets the velocity feedback gain of the current axis set by the function GT_SetKvff().
Function Parameter: *Kvff returns the velocity feedback gain.
System: DOS, WINDOWS
Applicable Card: SV
Relevant Function: GT_SetKvff

GT_GetLmtSwt

Function Prototype: short GT_GetLmtSwt (unsigned short * Switch);
Description of Function: This function gets the actual status of current limit switch.
Function Parameter: *Switch returns the level of the limit switch input signal, as defined in the following table. If a bit = 1, it means that the input signal of the corresponding switch is at logic 1. If a bit = 0, it means that the input signal is at low level. This function is not related with the function GT_LmtSns(), and only shows the actual status of limit switch.

Bit	Description	Definition
8-15	Not used. Set to 0.	
7	Axis #4: Status bit of negative direction limit switch	0: High level, 1: Low level.
6	Axis #4: Status bit of positive direction limit switch	0: High level, 1: Low level.
5	Axis #3: Status bit of negative direction limit switch	0: High level, 1: Low level.
4	Axis #3: Status bit of positive direction limit switch	0: High level, 1: Low level.
3	Axis #2: Status bit of negative direction limit switch	0: High level, 1: Low level.
2	Axis #2: Status bit of positive direction limit switch	0: High level, 1: Low level.
1	Axis #1: Status bit of negative	0: High level, 1: Low level.

	direction limit switch	
0	Axis #1: Status bit of positive direction limit switch	0: High level, 1: Low level.

System: DOS, WINDOWS

Applicable Card: All GT series cards

GT_GetMAcc

Function Prototype: short GT_GetMAcc(double * Macc);

Description of Function: This function gets the maximum acceleration of the current axis set by the function GT_SetMAcc().

Function Parameter: The double-precision parameter *Macc indicates the maximum acceleration needed.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetMAcc

GT_GetMode

Function Prototype: short GT_GetMode (unsigned short * Mode);

Description of Function: This function gets the control mode character of the current axis.

Function Parameter: *Mode returns the control mode character. Each bit is defined in the following table.

Bit	Description
0-6	For internal use.
7	Motion error stop symbol bit. The commands GT_AuStpOn() and GT_AuStpOff() can modify this symbol bit. The bit = 1 means that, afer the motor encounters motion error, the motor servo activation will be closed automatically and the motor will stop.
8-9	Reserved.
10	Automatic updating of symbol bit. The commands GT_AuUpdtOn() and GT_AuUpdtOff() can modify this symbol bit. The bit = 1 means that, the motion controller will update automatically the control axis parameters after the breakpoint condition is satisfied.
11-13	The coding of control axis motion mode symbols are: Bit13 Bit12 Bit11 Motion Mode 0 0 0 T-curve mode 0 0 1 Velocity control mode 0 1 0 S-curve mode 0 1 1 Electronic gear mode 1 0 1 Coordinate motion mode
14-15	For internal use.

System: DOS, WINDOWS

Applicable Card: All GT series cards

GT_GetMtrBias

Function Prototype: short GT_GetMtrBias(unsigned short * Bias);

Description of Function: This function gets the net difference compensation of the current

axis set by the function GT_SetMtrBias().

Function Parameter: *Bias returns the net difference compensation.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_SetMtrBias

GT_GetMtrCmd

Function Prototype: short GT_GetMtrCmd (short * Mcmd);

Description of Function: In open loop status, this function gets the motor control output value of the current axis set by the function GT_SetMtrCmd().

Function Parameter: *Mcmd returns the motor control output value. When the current axis is in close loop status, the return value of parameter *Mcmd has no meaning.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_SetMtrCmd

GT_GetMtrLmt

Function Prototype: short GT_GetMtrLmt (unsigned short * Mlmt);

Description of Function: This function gets the output limit of the current axis set by the function GT_SetMtrLmt().

Function Parameter: *Mlmt returns the output limit.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetMtrLmt

GT_GetPos

Function Prototype: short GT_GetPos(long * Pos);

Description of Function: This function gets the position value of the current axis set by the function GT_SetPos().

Function Parameter: *Pos returns the target position value set.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetPos

GT_GetPosErr

Function Prototype: short GT_GetPosErr(unsigned short * Perr);

Description of Function: This function gets the position error limit of the current axis set by the function GT_SetPosErr().

Function Parameter: *Perr returns the position error limit set.

Applicable Card: SV

System: DOS, WINDOWS

Relevant Function: GT_SetPosErr

GT_GetRatio

Function Prototype: short GT_GetRatio(double * Ratio);

Description of Function: This function gets the electrical gear deceleration ratio set by the function GT_SetRatio() in the electrical gear work mode.

Function Parameter: *Ratio returns the electrical gear deceleration ratio set.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetRatio

GT_GetSmplTm

Function Prototype: short GT_GetSmplTm(double * Timer);

Description of Function: This function gets the servo period set by the function GT_SetSmplTm().

Function Parameter: *Timer returns the servo period set, ranged from 48(us) to 1966.08 (us).

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetSmplTm

GT_GetSts

Function Prototype: short GT_GetSts (unsigned short * Status);

Description of Function: This function gets the status character of the current axis.

Function Parameter: *Status returns the status character. For its meaning, please see *Table 11-1* for the function GT_ClrSts().

System: DOS, WINDOWS

Applicable Card: All GT series cards

Function Evoking: The following sample detects the status of the first axis. If the axis doesn't move, give a new position value and update it.

```
void main()
{
    short rtn;
    unsigned short status;
    rtn=GT_GetSts(&status);    error(rtn);
    if(status&0x400) return;
    rtn=GT_SetPos(10000);      error(rtn);
    rtn=GT_Update(status);     error(rtn);
 }
```

GT_GetVel

Function Prototype: short GT_GetVel (double * Vel);

Description of Function: This function gets the velocity of the current axis set by the function GT_SetVel().

Function Parameter: *Vel returns the velocity.

System: DOS, WINDOWS
Applicable Card: All GT series cards
Relevant Function: GT_SetVel

GT_HardRst

Function Prototype: void GT_HardRst (void);
Description of Function: This function restores the motion controller. After evoking this function, the host will output a restoring signal. So, in any condition, GT_HardRst() can restore the controller, and reset all the initial status of the control register, which are as follows.

[1] Set all the motion parameters register to 0.
[2] Set all the position capture register to 0.
[3] Clear all the event status bits.
[4] Mask all the interrupt request bits (Reset them.).
[5] Set the position control mode as T-curve control mode.
[6] Set the parameters of each filter to 0.
[7] Limit the integral of four axes to 32767.
[8] Set the position error limit of four axes to 32767.
[9] Set all the breakpoint comparison values to 0.
[10] Prohibit the automatic parameter updating mode.
[11] Set four axes as in close loop mode.
[12] Prohibit the automatic stopping of four axes due to motion error.
[13] The output voltage limit value of four axes is 32767.
[14] The acceleration limit of four axes (for motion in coordinate system) is 16384.
[15] Allow automatic stopping of motion in coordinate system due to motion exception.
[16] The motion in coordinate system is in the immediate command input and execution status.
[17] No mapping relation between the coordinate system and the four motor control axes.

System: DOS, WINDOWS
Applicable Card: All GT series cards
Relevant Function: GT_Reset
Function Evoking: This function is evoked normally when the motion controller has error not restorable.

GT_Home

Function Prototype: short GT_Home(long pos)
Description of Function: Evoking this function to reset the current axis automatically. After receiving this command, the motion controller will move to the target position set by the parameter immediately. During movement, when Home signal is triggered, the motion controller will do different treatment according to whether user sets the command GT_Index or not. If GT_Index has been set and Home+Index are required to be reset automatically, the current axis will continue moving, search and stop at the position triggering Home signal. If GT_Index is not set and only Home is required to be reset, the current axis will stop at the

position triggering Home signal.

When this command is evoked, Home signal will be in triggering status. The current axis will first move at the opposite direction against the target position set, until it exits the signal triggering area. Then it will move towards the target position and stop at Home position (or Home+Index position).

Function Parameter: pos is the target position of the resetting motion, ranged from -1,073,741,824 to 1,073,741,823. The setting of parameter requires that, before the motion arrives at the target position, Home signal can be triggered.

Applicable Card: Select GT series card with the automatic resetting function.

System: DOS WINDOWS

Relevant Function: GT_HomeSense, GT_Index

GT_HookIsr

Function Prototype: GT_ISR GT_HookIsr(GT_ISR gtisr)

Description of Function: Specify an interrupt service routine (ISR) for the interrupt hook of PCI control card.

Function Parameter: gtisr is the start address of ISR.

Function Return: The return value is the start address of the original ISR. It must be reserved for the function GT_UnhookIsr.

Function Evoking: Refer to interrupt treatment in DOS.

Applicable Card: PCI Bus Card

System: DOS

Relevant Function: GT_UnhookIsr

GT_Index

Function Prototype: shortGT_Index(short value)

Description of Function: Set the automatic resetting mode as Home+Index, i.e. after finding the Home position automatically, search the triggering point of Index signal automatically，and stop at the triggering position of Index signal. The changing command is used together with GT_Home to reset accurately the equipment automatically.

Function Parameter: value 0 (default): Do not search Index signal after automatically resetting.

1: Search Index signal after automatically resetting.

Function Evoking: The following sample realizes the automatic resetting accurately.

```
void autohome()
{
   short rtn;
   GT_Index();
   GT_Home(2000000);
   }
```

Applicable Card: Select GT series card with the automatic resetting function.

System: DOS WINDOWS

Relevant Function: GT_Home GT_HomeSense

GT_LmtSns

Function Prototype: short GT_LmtSns(unsigned short Sense);

Description of Function: This function sets the effective level of the limit switch.

Function Parameter: For the meaning of Sense, please see the following table. In the table, if the corresponding bit is set to 0, it means that the input signal of limit switch is the triggering limit bit at high level. Whereas, if it is set to 1, it means that the input signal of limit switch is the triggering limit bit at low level. The bits in the controller are all set to being triggered at high level. When initializing the motion controller, it is necessary to give the effective level of limit switch correctly. Otherwise, it will not assure the normal work of the controller, or it will cause all the positive and negative limit switch symbol bits of all the axes to be set.

Bit	Description	Definition
8-15	Not used. Set to 0.	
7	Axis #4: Status bit of negative limit switch	0: High level, 1: Low level.
6	Axis #4: Status bit of positive limit switch	0: High level, 1: Low level.
5	Axis #3: Status bit of negative limit switch	0: High level, 1: Low level.
4	Axis #3: Status bit of positive limit switch	0: High level, 1: Low level.
3	Axis #2: Status bit of negative limit switch	0: High level, 1: Low level.
2	Axis #2: Status bit of positive limit switch	0: High level, 1: Low level.
1	Axis #1: Status bit of negative limit switch	0: High level, 1: Low level.
0	Axis #1: Status bit of positive limit switch	0: High level, 1: Low level.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Function Evoking: The following sample sets the effective levels reversed of the positive and negative limit switches of the third axis, and clears the error status of triggering the limit switch caused by the previous unconformity of the effective level of limit switch.

```
void main()
{
    short rtn;
    rtn=GT_LmtSns(0x30);     error(rtn);
    rtn=GT_Axis(3);          error(rtn);
    rtn=GT_Rststs(0xff9f);   error(rtn);
 }
```

GT_LmtsOff

Function Prototype: short GT_LmtsOff (void);

Description of Function: This function prohibits the motion controller to monitor the limit switch status of current axis. But the status is still reflected in the limit switch status register of the controller. User can evoke GT_GetLmtSwt() to inquire the status of limit switch. The change of the status will not cause the controller to perform any operation.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_LmtsOn

GT_LmtsOn

Function Prototype: short GT_LmtsOn(void);

Description of Function: This function makes the motion controller to start monitoring the limit switch status of current axis. If the interrupt request is allowed, no matter it is the positive or negative limit switch is pressed down, the controller will send the interrupt request to the host, and the corresponding bit of the limit switch shown in *Table 11-1* will be set to 1. If the interrupt is not allowed, user may use the inquiry method to evoke the command GT_GetLmtSwt() or GT_GetSts() to inquire the status of limit switch. The default status of the controller is the evoking of GT_LmtsOn().

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_LmtsOff

GT_MltiUpdt

Function Prototype: short GT_MltiUpdt(unsigned short Mask)；

Description of Function: This function makes the buffer command and parameter set by several axes to become effective immediately. The difference is that, the function GT_Update() only makes the parameter and command set for the current axis effective.

Function Parameter: The meaning of each Mask bit is listed in the following table. If user 各 doesn't want the parameter set for an axis to become effective immediately, set the corresponding bit to 0. Otherwise, set it to 1.

Bit	Description
0	Axis 1
1	Axis 2
2	Axis 3
3	Axis 4
4-15	Not used.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_Update

Function Evoking: The following sample updates the parameter Kp for the first and third axes at the same time.

```
void main()
```

```
{
    short rtn;
    rtn=GT_Axis(1);             error(rtn);
    rtn=GT_SetKp(10);           error(rtn);
    rtn=GT_Axis(3);             error(rtn);
    rtn=GT_SetKp(15);           error(rtn);
    rtn=GT_MltiUpdt(0x5);       error(rtn);
 }
```

GT_MtnBrk

Function Prototype: short GT_MtnBrk(void);

Description of Function: This function sets the breakpoint mode of current axis as motion completion mode. After this function is evoked, and when the motion completion symbol bit of the controller in the axis status register is set to 1, a breakpoint will be evoked, to set the breakpoint symbol bit in the axis status register to 1 and clear the breakpoint mode, i.e. the controller will not trigger this breakpoint any more unless the host resets it. Meanwhile, when the automatic updating symbol of the axis mode status register is set to 1, the controller will update all the dual-buffer structure parameters and commands of the axis automatically.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Function Evoking: In the following sample, set the breakpoint mode of the fourth axis to the motion completion triggering breakpoint mode, and update a new target position automatically upon motion completion, to cause a reversed motion.

```
void main()
{
    short rtn;
    rtn=GT_Axis(1);         error(rtn);
    rtn=GT_AuUpdtOn();      error(rtn);
    rtn=GT_MtnBrk();        error(rtn);
    rtn=GT_PrflT();         error(rtn);
    rtn=GT_ClrSts();        error(rtn);
    rtn=GT_SetPos(10000);   error(rtn);
    rtn=GT_SetVel(7);       error(rtn);
    rtn=GT_SetAcc(0.347);   error(rtn);
    rtn=GT_Update();        error(rtn);
    rtn=GT_SetPos(0);       error(rtn);
    }
```

GT_NegBrk

Function Prototype: short GT_NegBrk(void);

Description of Function: This function sets the breakpoint mode of current axis as breakpoint less than the target position. Before evoking this function, first use the function GT_SetBrkCn() to set the breakpoint position, and evoke GT_Update() or GT_MltiUpdt() to make it effective. Then, after evoking this function, the controller will compare the actual position of the axis

with the breakpoint position in each servo period. When the actual position is no more than the breakpoint position, it will set the breakpoint symbol bit in the axis status register to 1 and clear the breakpoint mode. The controller will not trigger this breakpoint any more unless the host resets it. Meanwhile, when the automatic updating symbol of the axis mode status register is set to 1, the controller will update all the dual-buffer structure parameters and commands of the axis automatically.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_PosBrk

GT_OpenLp

Function Prototype: short GT_OpenLp(void);

Description of Function: This function sets the current axis as in the servo open loop control status. It is mainly used in controlling the motor directly. Generally, the servo driver shall be in the close loop control status.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_CloseLp

GT_Open

Function Prototype: short GT_Open(unsigned long PortBase, unsigned long irq);

Description of Function: This function opens the motion controller. It must be evoked when user program starts.

Function Parameter: PortBase is the base address of motion controller. The parameter irq is the interrupt number of controller (*If Interrupt 10 to 15 in the host are all filled, or user doesn't need interrupt, the parameter irq can be set to 0, indicating that there is no interrupt.*). When selecting the base address and interrupt number, pay attention that, do not interfere with other equipment. Or, this function will fail.

System: WINDOWS

Applicable Card: ISA Bus Card

Relevant Function: GT_Close

GT_Open

Function Prototype: short GT_Open();

Description of Function: This function opens the motion controller. Evoke this function when user program starts.

System: DOS, WINDOWS

Applicable Card: PCI Bus Card

Relevant Function: GT_Close

GT_PosBrk

Function Prototype: short GT_PosBrk(void);

Description of Function: This function sets the breakpoint mode of current axis as breakpoint

at positive target position. Before evoking this function, first use the function GT_SetBrkCn() to set the breakpoint position, and evoke GT_Update() or GT_MltiUpdt() to make it effective. Then, after evoking this function, the controller will compare the actual position of the axis with the breakpoint position in each servo period. When the actual position is no less than the breakpoint position, it will set the breakpoint symbol bit in the axis status register to 1 and clear the breakpoint mode, i.e. the controller will not trigger this breakpoint any more unless the host resets it. Meanwhile, when the automatic updating symbol of the axis mode status register is set to 1, the controller will update all the dual-buffer structure parameters and commands of the axis automatically.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_NegBrk

GT_PrflG

Function Prototype: short GT_PrflG(unsigned short Master)；

Description of Function: This function sets the motion control mode of current axis as electrical gear mode, and specifies the main axis number followed.

The parameter Master can be 1, 2, 3 or 4, indicating that Axis 1, 2, 3 and 4 are the main axes of current axis. It can also be 5 or 6, indicating that one of the two auxiliary encoders is specified as the main axis (The auxiliary encoder is an optional function.).

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_PrflT, GT_PrflS, GT_PrflV

Function Evoking: When the current axis is not in motion, evoke this function. In the following sample, set the first axis to follow the T-curve motion of the fourth axis at a scale of –1.5, but the third axis to follow the motion of the first axis at a scale of 0.5.

```
void main()
{
    short rtn;
    rtn=GT_Axis(4);          error(rtn);
    rtn=GT_PrflT();          error(rtn);
    rtn=GT_Axis(1);          error(rtn);
    rtn=GT_PrflG(4);         error(rtn);
    rtn=GT_SetRatio(-1.5); error(rtn);
    rtn=GT_Axis(3);          error(rtn);
    rtn=GT_PrflG(1);         error(rtn);
    rtn=GT_SetRatio(0.5); error(rtn);
    rtn=GT_MltiUpdt(0x5); error(rtn);
 }
```

GT_PrflS

Function Prototype: short GT_PrflS(void);

Description of Function: This function sets the motion control mode of current axis as S-curve mode.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_PrflT, GT_PrflG, GT_PrflV

Function Evoking: When the current axis is not in motion, evoke this function. The setting of the S-curve control mode is as follows.

```
void main()
{
    short rtn;
    rtn=GT_Axis(1);             error(rtn);
    rtn=GT_PrflS();             error(rtn);
    rtn=GT_SetPos(12345);       error(rtn);
    rtn=GT_SetVel(3.21);        error(rtn);
    rtn=GT_SetMAcc(0.345);      error(rtn);
    rtn=GT_Jerk(0.087);         error(rtn);
    rtn=GT_Update();                error(rtn);
    }
```

GT_PrflT

Function Prototype: short GT_PrflT(void);

Description of Function: This function sets the motion control mode of current axis as T-curve mode.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_PrflS, GT_PrflG, GT_PrflV

Function Evoking: When the current axis is not in motion, evoke this function. The setting of the T-curve control mode is as follows.

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);             error(rtn);
    rtn=GT_PrflT();             error(rtn);
    rtn=GT_SetPos(100000);      error(rtn);
    rtn=GT_SetVel(5.7);         error(rtn);
    rtn=GT_SetAcc(0.67);        error(rtn);
    rtn=GT_Update();                error(rtn);
 }
```

GT_PrflV

Function Prototype: short GT_PrflV(void);

Description of Function: This function sets the motion control mode of current axis as velocity control mode.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_PrflS，GT_PrflG，GT_PrflT

Function Evoking: When the current axis is not in motion, evoke this function. The setting of the velocity control mode is as follows.

```
void main()
{
    short rtn;
    rtn=GT_Axis(3);       error(rtn);
    rtn=GT_PrflV();       error(rtn);
    rtn=GT_SetVel(17);    error(rtn);
    rtn=GT_SetAcc(1.1);       error(rtn);
    rtn=GT_Update();          error(rtn);
    }
```

GT_Reset

Function Prototype: short GT_Reset (void)

Description of Function: This function enables the host to reset the motion controller with a command. Its result is the same as that of the function GT_HardRst().

Notice: When this function is evoked, if a motor axis is in the control axis (i.e. use GT_AxisOn() to open the control axis.), it may act. This is because that, when resetting the motion controller, the resetting instruction also sets all the external output of the controller as in the initial status (including closing the motor drive activation). But, due to the delayed response of the motor itself to this closing signal, it makes the motor still be in the status of receiving servo control command when the controller has closed the control, so as to make the motor act. Therefore, before evoking this function, use the instruction GT_AxisOn() to close the control axis of all the motor axes.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_HardRst

GT_RstIntr

Function Prototype: short GT_RstIntr(unsigned int Mask);

Description of Function: This function clears the interrupt symbol bit of current axis. After the controller sends interrupt request to the host, the host shall execute the function GT_RstIntr () when the interrupt service routine finishes, to clear the interrupt symbol of the controller, so that the controller can send interrupt request to the host again.

Function Parameter: For the definition of Mask, please see the following table. When a bit in Mask is set to 1, the interrupt event indicated by this bit is allowed to be reserved. If the bit is set to 0, the interrupt event indicated by this interrupt is cleared.

Bit	Definition
7-15	Not used. Can be set to 0.
6	Positive limit switch
5	Negative limit switch
4	Error in motion

3	Index/Home captured.
2	Breakpoint condition is satisfied.
1	Axis driver alarms.
0	The axis motion is completed.

System: DOS

Applicable Card: All GT series cards

GT_RstSts

Function Prototype: short GT_RstSts(unsigned short Mask);

Description of Function: This function clears the status register of the current axis according to the definition of Mask.

Function Parameter: The definition Mask is consistent with the definition of Bit0 – Bit7 in the attached table of the function GT_RstIntr(). When a bit of Mask is set to 1, this bit is allowed to be reserved. If it is set to 0, this event symbol is to be cleared.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_ ClrSts

GT_SetAcc

Function Prototype: short GT_SetAcc(double Acc);

Description of Function: This function sets the acceleration parameter of the current axis. It is only effective in the T-curve velocity mode and velocity control mode.

Function Parameter: Acc is the acceleration to be set, ranged from 0 to 16384. The unit of acceleration is the **number of pulse/control period^2**. To make the new parameter set effective requires the evoking of the function GT_Update() or GT_MltiUpdt().

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_GetAcc

GT_SetAdcChn

Function Prototype: short GT_SetAdcChn(unsigned short value);

Description of Function: This function sets the number of ADC channels.

Function Parameter: The parameter value sets the number of AD channels, ranged from 1 to 8.

System: DOS, WINDOWS

Applicable Card: Select GT series cards having this function.

GT_SetAddr

Function Prototype: short GT_SetAddr(unsigned short Address);

Description of Function: This function sets the base address of communication between the host and the motion controller.

Function Parameter: Address is the base address value to be set. Its range is the setting range

of base address allowed by the motion controller.

System: DOS

Applicable Card: All GT series cards

Relevant Function: GT_GetAddr

GT_SetAtlPos

Function Prototype: short GT_SetAtlPos(long actl_pos);

Description of Function: Evoke this function to modify the actual and target positions of the current axis and the planned position controlling the current control period to the specified values. This function is only effective when the current axis stops. Otherwise, it will be considered as an illegal command with a motion error symbol generated. This function is ineffective when the current axis is in the electrical gear motion mode.

Function Parameter: actl_pos indicates the actual position to be set.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_GetAtlPos, GT_SynchPos, GT_ZeroPos

Function Evoking: The following sample sets the actual position of the fourth axis to 1000, when it is not in motion.

```
void main()
{
    short rtn;
    unsigned short status;
    rtn=GT_Axis(4);                 error(rtn);
    rtn=GT_GetSts(&status);         error(rtn);
    if(status&0x400) return;
    rtn=GT_SetAtlPos(1000);         error(rtn);
    }
```

GT_SetBgCommandSet

Function Prototype: short WINAPI GT_SetBgCommandSet(PBGCOMMANDSET pBgCmdset, ULONG CmdsetSize);

Description of Function: This function sets the background execution command set for the motion control card when the card is interrupted. It will cover the previous value set. It can set the background commands for several interrupts once, or several background commands for the same interrupt.

Function Parameter: pBgCmdset is the start address of command set to be executed by the control card. CmdsetSize is the memory size for the background command set, with a unit in byte.

Function Return: 0 means success, and –1 means failure.

The drive program of motion controller allows user to specify the GT command set to be executed by the motion controller after it generates interrupt. Since these commands are to be executed by the controller when it generates interrupt, and do not interact with user directly, we call them background commands. The structure of command set is as follows:

```
typedef struct _BACKGROUND_COMMANDSET
{
USHORT    Count;                    //The number of the subitems of command set, i.e. the number of interrupt types to be served.
BACKGROUND_COMMAND BackgroundCommand[1];//A background command group specifying interrupt.
}BGCOMMANDSET,*PBGCOMMANDSET;

typedef struct _BACKGROUND_COMMAND
{
   USHORT              InterruptMask;//The exact interrupt served by the command array.
   USHORT                   CommandCount; //The number of commands in the command group.
   GENERAL_COMMAND     GenCommand[1];    //Command array.
}BACKGROUND_COMMAND,*PBACKGROUND_COMMAND;

typedef struct _GENERAL_COMMAND //The structure of a single command.
{
   USHORT      usCommand; //Command character.
   USHORT      InputLength; //The length of data to be input by the command, with a unit in character.
   USHORT      OutputLength;//The length of data to be returned by the command, with a unit in character.
   USHORT      usResult; //The command execution result.

   union
   {
        USHORT      sData[2];
        ULONG       lData;
   }in; //Data input by command.

   union
   {
        USHORT      sData[2];
        ULONG       lData;
   }out; //Data returned by command.

}GENERAL_COMMAND,*PGENERAL_COMMAND;
```

System: WINDOWS

Applicable Card: PCI Bus Card

Relevant Function: GT_SetIntSyncEvent

Function Evoking: Refer to the interrupt treatment in Windows environment.

GT_SetBrkCn

Function Prototype: short GT_SetBrkCn(long Brk);

Description of Function: This function sets the breakpoint position comparison value of the current axis.

When evoking the functions GT_PosBrk( ) and GT_NegBrk( ), since these two commands will become effective immediately, user must evoke the function GT_SetBrkCn( ) before evoking them, and evoke the function GT_Update() or GT_MltiUpdt() to make effective the newly set breakpoint position.

Function Parameter: Brk indicates the breakpoint position of current axis to be set, ranged from –1073741824 to 1073741823. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_GetBrkCn

GT_SetEncCapt

Function Prototype: short GT_SetEncCapt(void);

Description of Function: This function sets the INDEX signal capture of the auxiliary encoder No. 1.

When using this instruction, it will clear the INDEX signal capture mode of auxiliary encoder already set, and reset the capture mode. Meanwhile, it also clears the INDEX signal captured status of the auxiliary encoder No. 1.

To realize the INDEX capture function of auxiliary encoder, use this function to start the capture mode.

System: DOS, WINDOWS

Applicable Card: Select the GT series cards having the function of “INDEX signal capture of auxiliary encoder”.

Relevant Function: GT_GetEncSts, GT_GetEncCapt

GT_SetILmt

Function Prototype: short GT_SetILmt(unsigned short Ilm);

Description of Function: This function sets the error integral limit of the current axis servo filter.

Function Parameter: Ilm is the error integral limit to be set, ranged from 0 to 32767. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt().

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetILmt

GT_SetIntrMsk

Function Prototype: short GT_SetIntrMsk(unsigned short Mask);

Description of Function: This function sets the interrupt mask character.

Function Parameter: For the meaning of each Mask bit, please refer to the following table of the function GT_RstIntr(). When a bit in interrupt mask character is set to 1, the interrupt event indicated by this bit is allowed to send interrupt request to the host. If it is set to 0, the sending is not allowed.

Bit	Definition
7-15	Not used. Can be set to 0.
6	Positive limit switch
5	Negative switch limit
4	Error in motion.
3	Index/Home captured
2	Breakpoint condition is satisfied.
1	The axis driver alarms.
0	The motion of axis is completed.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_GetIntrMsk

GT_SetIntrTm

Function Prototype: void GT_SetIntrTm(short Timer);

Description of Function: This function sets the time constant of time interrupt of the controller. The time during the time interrupt of controller is decided together by the control period of controller and the Timer value set by this function.

Function Parameter: Timer is the multiple of control period, ranged from 0 to 32767.

For example, the control period is 200 microseconds, and the host uses the function GT_SetIntrTm() to set the value of Timer to 10. Then, the period of time interrupt = 10*200 microseconds, i.e. 2 milliseconds.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_GetIntrTm

GT_SetIntSyncEvent

Function Prototype: short WINAPI GT_SetIntSyncEvent(HANDLE hEvent);

Description of Function: This function sets the interrupt synchronization event for PCI control card, and clears the previous value set.

Function Parameter: hEvent is the handle of event. When it is NULL, the function only resets the previous value set.

This function can treat the time interrupt and also the event interrupt generated by the motion controller. User may evoke GT_GetIntr(unsigned * Status) to tell which kind of interrupt is the current interrupt. Status = 0: Time interrupt. Status $\neq$ 0: For detailed meaning, please refer to the following table.

Bit	Definition
15	Activate the Index signal capture (1 means activation).
14	Activate the home point switch capture (1 means activation).
12-13	Bit13 Bit12 Axis Generating Interrupt 0 0 1 0 1 2 1 0 3 1 1 4
11	Activate axis limit switch (1 means activation)
10	Motion status symbol bit (1 means being in motion.)
9	Activate/close control axis (1 means activation)
8	Open loop/Close loop (1 means close loop.)
7	Error in host command (1 means error.)
6	Negative limit switch action (1 means action.)
5	Positive limit switch action (1 means action.)
4	Error in motion (1 means error.)
3	Index/Home position capture arrival (1 means arrival.)
2	Breakpoint arrival (1 means arrival.)
1	Servo driver alarm (1 means alarm.)
0	Motion completion symbol bit (1 means completion.)

PCI bus motion control drive program can use the event synchronization mechanism to synchronize with user program. When the control card generates an interrupt, it will set the event object represented by hEvent as setting status. For the detailed event synchronization mechanism, please see the relevant document about the developing of progress synchronization mechanism.

Function Return: 0 means success and –1 means failure.

System: WINDOWS

Applicable Card: PCI Bus Card

Relevant Function: GT_SetBgCommandSet

Function Evoking: See the interrupt treatment in Windows environment.

GT_SetJerk

Function Prototype: short GT_SetJerk(double Jerk);

Description of Function: This function is used to set the jerk of current axis in S-curve mode. The unit of jerk is plus/control period^3.

Function Parameter: Jerk is the jerk to be set, ranged from 0 to 0.5. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_GetJerk

GT_SetKaff

Function Prototype: short GT_SetKaff(unsigned short Kaff);

Description of Function: This function sets the acceleration feedback gain of the current axis servo filter.

Function Parameter: Kaff is the acceleration feedback gain to be set, ranged from 0 to 32767. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetKaff

GT_SetKd

Function Prototype: short GT_SetKd(unsigned short Kd);

Description of Function: This function sets the differential gain of the current axis servo filter.

Function Parameter: Kd is the differential gain to be set, ranged from 0 to 32767. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetKd

GT_SetKi

Function Prototype: short GT_SetKi(unsigned short Ki);

Description of Function: This function sets the integral gain of the current axis servo filter.

Function Parameter: Ki is the integral gain to be set, ranged from 0 to 32767. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetKi

GT_SetKp

Function Prototype: short GT_SetKp(unsigned short Kp);

Description of Function: This function sets the percentage gain of the current axis servo filter.

Function Parameter: Kp is the percentage gain to be set, ranged from 0 to 32767. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetKp

GT_SetKvff

Function Prototype: short GT_SetKvff (unsigned short Kvff);

Description of Function: This function sets the velocity feedback gain of the current axis servo filter.

Function Parameter: Kvff is the velocity feedback gain to be set, ranged from 0 to 32767. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetKvff

GT_SetMAcc

Function Prototype: short GT_SetMAcc(double Macc);

Description of Function: When the control mode is S-curve mode, evoke this function to set the maximum acceleration of the current axis.

Function Parameter: Macc is the maximum acceleration to be set, ranged from 0 to 0.5 (excluding 0.5), with a unit of **Pulse/Control Period^2**. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_GetMAcc

GT_SetMtrBias

Function Prototype: short GT_SetMtrBias(short Bias);

Description of Function: This function sets the bias compensation of the control output of the current axis servo filter.

Function Parameter: Bias is the bias compensation to be set, ranged from –32768 to 32767. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first. This parameter is only effective for close loop control. The default value in the controller is 0.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetMtrBias

GT_SetMtrCmd

Function Prototype: short GT_SetMtrCmd(short Mcmd);

Description of Function: This function sets the voltage output control value in the open loop control status.

Function Parameter: Mcmd is the motor output control value to be set, ranged from –32767 to 32767. Here, -32767 means the maximum negative voltage value, and 32767 means the maximum positive voltage value.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetMtrCmd

GT_SetMtrLmt

Function Prototype: short GT_SetMtrLmt(unsigned short Mlmt);

Description of Function: This function sets the limit of the current axis control output.

Function Parameter: Mlmt is the limit to be set, ranged from 0 to 32767. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first. This parameter is only effective for close loop control. The default value in the controller is 0.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetMtrLmt

GT_SetPos

Function Prototype: short GT_SetPos (long Pos);

Description of Function: This function sets the target position of current axis in the S-curve and T-curve modes.

Function Parameter: Pos is the target position value to be set, ranged from –1073741824 to 1073741823. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first. The unit of position value is the **number of pulse**.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_GetPos

GT_SetPosErr

Function Prototype: short GT_SetPosErr(unsigned short Perr);

Description of Function: This function sets the position error limit of current axis.

Function Parameter: Perr is the position error limit to be set, ranged from 0 to 32767. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: SV

Relevant Function: GT_GetPosErr

GT_SetRatio

Function Prototype: short GT_SetRatio(double Ratio);

Description of Function: In the electrical gear mode, this function sets the electrical gear ratio of the current axis, to make the primary axis and secondary axis to move at the specified ratio.

Function Parameter: Ratio is the electrical gear ratio to be set, ranged from –16384 to 16384. When Ratio is positive, the primary axis and secondary axis move at the same direction. When Ratio is negative, the primary axis rotates at a reversed direction against that of the secondary axis. To make effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: All GT series cards
Relevant Function: GT_GetRatio

GT_SetSmplTm

Function Prototype: short GT_SetSmplTm(double Timer);
Description of Function: This function sets the servo period of the motion controller.
The parameter Timer is the servo period to be set, with a unit in microsecond, ranged from 48 to 1966.08 microseconds. The recommended servo period by the controller is 200 microseconds.
System: DOS, WINDOWS
Applicable Card: All GT series cards
Relevant Function: GT_GetSmplTm

GT_SetTime

Function Prototype: short GT_SetTime(unsigned long value);
Description of Function: This function adjusts the width of output pulse.
Function Parameter: value is the time of pulse width, with a unit in millisecond, ranged from 0 to 65535. The corresponding pulse width is 0 to 16ms. The output signal is at Pin23 of CN7.
System: DOS, WINDOWS
Applicable Card: SE

GT_TmrIntr

Function Prototype: short GT_TmrIntr (void);
Description of Function: This function sets the interrupt requested by the motion controller to the host as time interrupt. The period of time interrupt is decided together by the servo updating period of the controller and the set value of the function GT_SetIntrTm().
System: DOS, WINDOWS
Applicable Card: All GT series cards
Relevant Function: GT_EvntIntr
Function Evoking: The following sample sets the interrupt requested by the motion controller to the host as time interrupt.

```
void main()
{
    short rtn;
    rtn=GT_TmrIntr();     error(rtn);
    }
```

GT_SetVel

Function Prototype: short GT_SetVel(double Vel);
Description of Function: This function sets the target velocity parameter of the current axis.
Function Parameter: Vel is the target velocity value to be set. In the T-curve and S-curve modes, the velocity ranges from 0 to 16384. In the velocity control mode, the velocity ranges from –16384 to 16384. The velocity unit is the **number of pulses/control period**. To make

effective the newly set parameter requires to evoke the function GT_Update() or GT_MltiUpdt() first.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetVel

GT_SmthStp

Function Prototype: short GT_SmthStp(void);

Description of Function: This function is to stop the motion of current axis. The difference from GT_AbptStp() is that, GT_SmthStp() will decelerate until stop the motion of current axis at the acceleration parameter set, rather than that GT_AbptStp() stops the motion abruptly. This function can only be effective when being used together with the function GT_Update() or GT_MltiUpdt(). GT_SmthStp() is used in the three motion control modes for the control axis, except the electrical gear mode.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_AbptStp

GT_StepDir

Function Prototype: short GT_StepDir (void);

Description of Function: If the control output mode of the current axis is the pulse control mode, evoking this function can set the pulse output method as “Pulse + Direction”.

When user evokes the function GT_CtrlMode() to set the control output mode as pulse output mode, the motion controller will take “Pulse + Direction” as the default pulse output mode.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetPulse

GT_StepPulse

Function Prototype: short GT_StepPulse (void);

Description of Function: If the control output mode of current axis is the pulse output mode, evoking this function can set the pulse output method as “Positive and Negative Pulse”.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SetDir

GT_SwitchtoCardNo

Function Prototype: short GT_SwitchtoCardNo(short card_no);

Description of Function: This function is to switch the current card. When a PC system uses several control cards, this function can be used to specify the current control card. When the execution of the function succeeds, all the followed GT functions can only operate on the current card.

Function Parameter: card_no is the card number of the current card to be set, ranged from 0

to 15.

Function Return: 0 means success and –1 means failure.

In a multiple-card system, each control card will be allocated with a card number (0-15) when the operating system starts. This card will keep effective before the system is restarted to differentiate each control card. The number allocation principle follows the PNP principle that, the first card recognized by the system will be the card 0 forever. Therefore, if there is no change in hardware configuration, the numbers allocated by the system each time will be consistent.

System: DOS、WINDOWS

Applicable Card: PCI Bus Card

Relevant Function: GT_GetCurrentCardNo

Function Evoking: GT_SwitchtoCardNo(1)

GT_SynchPos

Function Prototype: short GT_SynchPos(void);

Description of Function: This function sets the target position register of the current control axis and the planned position register of the current servo period as the position value of the actual position register. It is applicable to the S-curve and T-curve control mode. It can be used to set the value of the current servo position control register equal to the value of the actual position register, when the motion control generates error and restarting is needed. Meanwhile, when the current axis switches in the servo activation (prohibit activation) status or the close loop (open loop) status, this function can also assure the smooth transition of the motor. This function can only be effective when being used together with the function GT_Update() or GT_MltiUpdt(). This function is ineffective when the current axis is in the electrical gear motion mode.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_ZeroPos, GT_SetAtlPos

GT_UnhookIsr

Function Prototype: short GT_UnhookIsr(GT_ISR old_isr)

Description of Function: Unhook the ISR hooked by the function GT_HookIsr for GT400 control card, and restore the old ISR.

Function Parameter: The parameter old_isr is the start address of the old ISR, returned by GT_HookIsr.

Function Return: 0 means success and –1 means failure.

Applicable Card: PCI Bus Card

System: DOS

Relevant Function: GT_HookIsr

Function Evoking: Refer to the interrupt treatment in DOS.

Note: GT_HookIsr and GT_UnhookIsr must be used together, i.e. after using GT_HookIsr, evoke GT_UnhookIsr before the whole program finishes to restore the old ISR. Otherwise, the system will break down.

GT_Update

Function Prototype: short GT_Update(void);

Description of Function: Some parameter setting functions and command functions of the motion controller work in the dual-buffer register mode. The dual-buffer parameters and commands can only become effective after the function GT_Update() is evoked.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_MltiUpdt

GT_ZeroPos

Function Prototype: short GT_ZeroPos (void),

Description of Function: Evoke this function to set the actual position register and target position of the current control axis and the planned position register of the current servo period to 0. This function is only effective when the current axis stops. Otherwise, it will be considered as an illegal command and generates error symbol. This function is ineffective when the current axis is in the electrical gear motion mode.

System: DOS, WINDOWS

Applicable Card: All GT series cards

Relevant Function: GT_SynchPos, GT_SetAtlPos

Googol Technology (HK) Ltd

Address: Room 3639, Annex Building
Hong Kong University of Science and Technology
Hong Kong
Tel: (852) 2358-1033
Fax: (852) 2358-4931
E-mail: info@googoltech.com
Web: http://www.googoltech.com/

Googol Technology (SZ) Ltd.

Address: Room W211, IER Building, South Area,
Shenzhen Hightech Industrial Park,
Shenzhen, PRC
Tel.: (0755) 2697-0823, 2697-0819, 2697-0824
Fax: (0755) 2697-0821
E-mail: support@googoltech.com
Web: http://www.googoltech.com.cn/