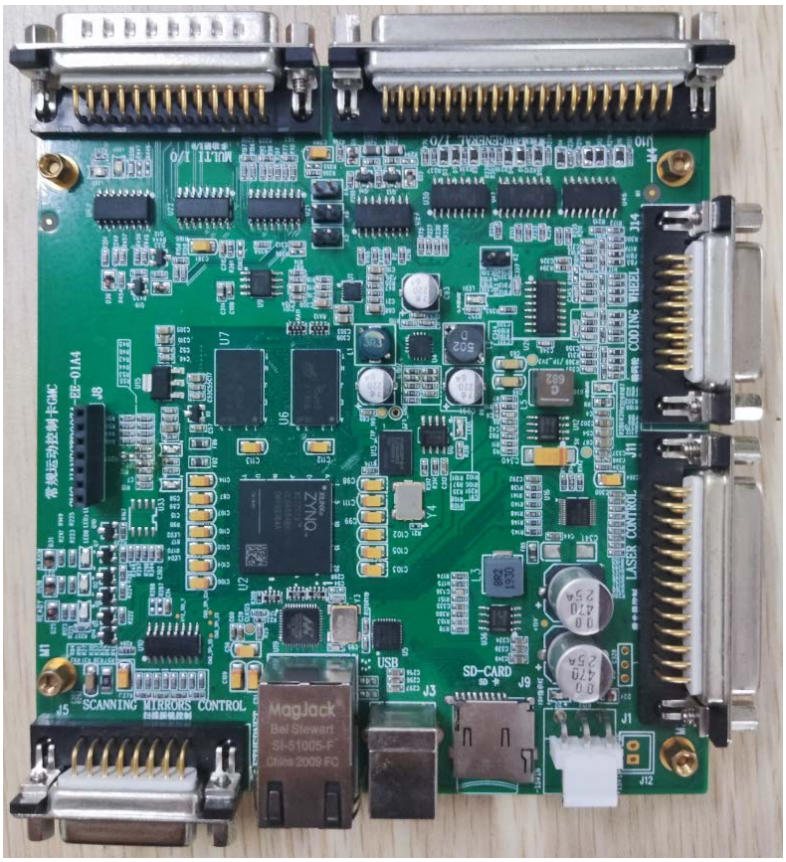
GMC Control Card Further Development Software

Instruction Manual V2.2



Scanner Optics Co., Ltd.

Add: 4F, Building 4, Han's Laser Production Base, No.128 Chongqing Rd., Fuyong Street, Baoan District, Shenzhen City, Guangdong, P.R.China 518103.

Web: www.scanneroptics.com

ScannerOptics 思特光学 | 深圳市思特光学科技有限公司 Scanner Optics Co.,Ltd.

Table of contents

Chapter 1 Overview

1.1 Overview of the GMC Control Card Functions

GMC control card is a galvo motion control card independently developed by Scanner Optics Co., LTD. At present, the main functions of the control card are as follows:

1. Support 2D, 3D, marking on-the-fly (customization), four-axis linkage (customization) and other functions;

2. Support three kinds of galvanometer data communication protocols, namely XY2-100, SPI and SL2 communication protocols;

3, Using TCP\IP communication protocol, Max 200 control cards can be connected and controlled at the same time, users only need to change the computer IP, without any other configuration;

4, Provide C++, C# two further development languages;

5, Support fiber laser (such as IPG, SPI, Raycus, JPT, etc.), CO2 laser, ultraviolet laser, green laser, provide 8 channels of digital energy control signal and two channels of analog control (0~10V);

6. With its own high-precision calibration software, after once calibration, it can be used permanently, without the need for further developers to frequently load calibration files. The calibration file of the control card is saved in the control card. It will not be lost after power failure and restart. No other software is required to load the calibration file again;

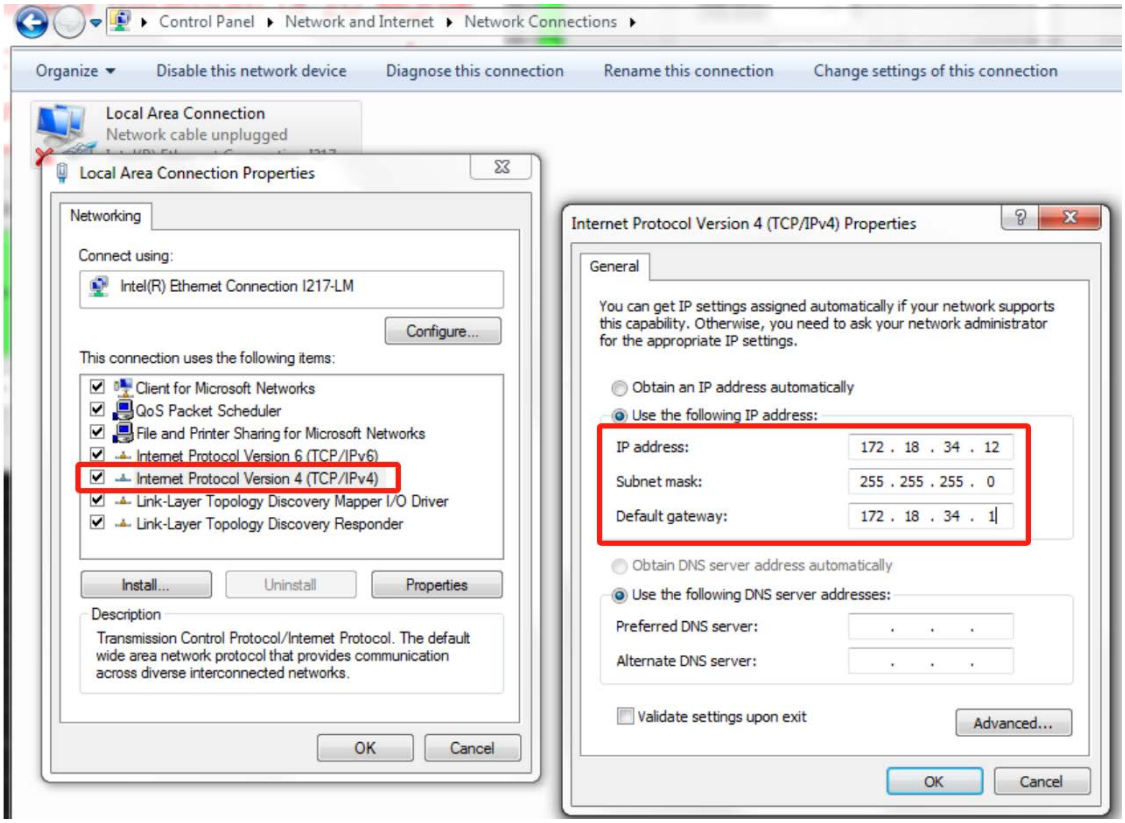
7, Support double calibration table saving, can control the scan heads with two different calibration tables to work;

8. One-way coding wheel reads the signals, two-way servo motors differential control signals, 15-way general IO signals;

9, Support single document, multi-document offline marking.

Chapter 2 IP Setting

Because this control card communicates with the computer via Ethernet, you need to set the computer's IP before using this control card. The IP address can be set in the network segment of 172.18.34.2~172.18.34.123, and the subnet mask is 255.255.255.0. However, avoid the computer's IP being the same as the card's IP. The initial IP address of the control card is 172.18.34.227 (this IP address can be changed, but the network segment cannot be changed). In addition, the control card will generate a temporary address of 172.18.34.226 when it is powered on. This IP will disappear after the card is powered on and initialized. Avoid using this initialized IP address. Therefore, except for the card's IP (172.18.34.227) and the initialization IP (172.18.34.226), other IPs can be set at will.



In addition, this control card can support the function of controlling multiple control cards to print different graphics with one host computer software at the same time. To achieve this function, you need to change the default 172.18.34.227 address of the control card to 172.18.34.228, 172.18.34.229~172.18.34.254, etc. If the IP addresses of multiple cards are the same, the host computer software cannot distinguish different cards. In the case of multiple control cards, you need to use a router or switch. Inserting different network cards on the computer may cause problems. Therefore, try to use a switch or router. After completing the above IP settings, you can use the built-in marking software to connect the control card.

Chapter 3 Control Card Hardware Introduction

3.1 GMC2 and GMC4

At present, the two commonly used versions of the control card are mainly GMC2 and GMC4. Compared with GMC2, GMC4 has made the following three changes:

① The general IO interface has been changed from DB25 to DB37;

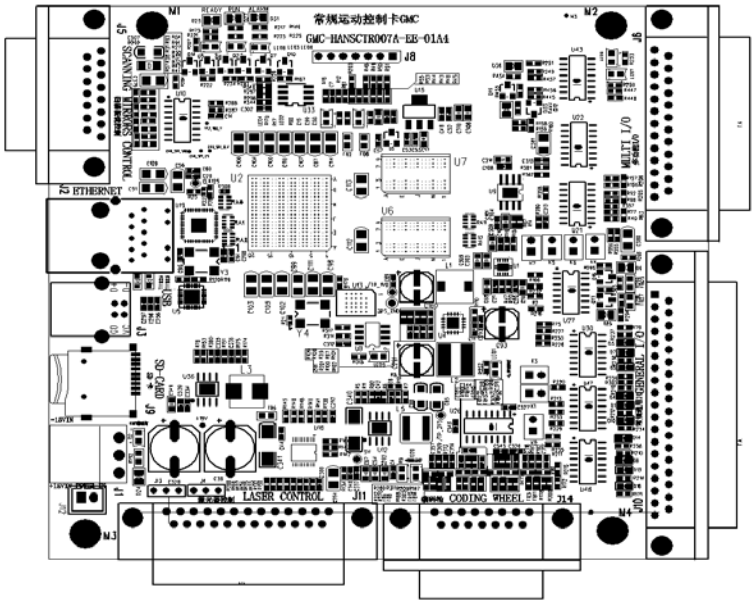
② The differential signal acquisition has been increased from one to two, that is, GMC2 can only realize the single-axis flight marking function, while GMC4 can realize dual-axis marking control, which is generally used in four-axis linkage systems;

③ The Z-axis control has been changed from DB37 port to DB25 port.

④ The laser alarm pin feedback, the pin is designed according to the one-to-one definition

This manual only introduces the hardware part of GMC4, and the GMC4 function is fully compatible with GMC2.

3.2 Hardware structure layout



The above picture is the control card layout diagram. "GMC-HANSCTR007A-EE-01A4" is printed in the middle of the board, indicating that this card is a GMC4 control card.

Among them,

J1: power interface, 3PIN socket, used to power the control card;

J2: network port, used for data communication with PC;

J3: USB interface, used for USB to UR232 signal;

J5: marking command output port, DB15 female interface;

J6: multi-function IO port DB25 male interface;

J8: program download and debug interface;

J9: SD card socket;

J10: general I/O, DB37 female interface;

J11: DB25 female control interface of laser;

J12: 5V power output port;

J14: encoder wheel control signal, DB15 female interface;

3.2.1 J1 Power port

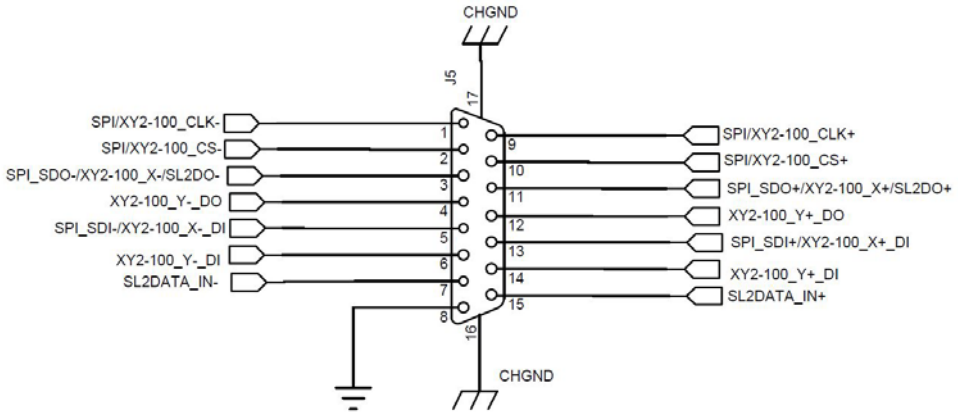
The control card requires a ±15V DC power supply. It is recommended to use a 15V/3A DC power supply. The pin definitions are as follows:

J1 PIN	Name	Instructions
1	+15VIN	+ 15V, the positive terminal of the power supply

2	GND	Ground, power ground
3	-15VIN	-15V, the negative terminal of the power supply

3.2.2 J5 J5 Galvanometer command output port

This port uses a DB15 female connector. The galvanometer control signal is a digital signal and can be directly connected to a digital galvanometer. Since the digital signal transmission protocols used by digital galvanometers are not exactly the same, it is necessary to confirm which transmission protocol the digital galvanometer uses. Our company also provides a digital-to-analog converter board, which can also be used to convert the digital signal output to an analog galvanometer. The pin definitions are as follows:



PIN	Name	Instructions
1，9	SPI/XY2-100_CLK- SPI/XY2-100_CLK+	Clock signal- Clock signal+
2，10	SPI/XY2-100_CS- SPI/XY2-100_CS+	Sync signal- Sync signal+
3，11	SPI_SDO-/XY2-100_X-/SL2DO- SPI_SDO+/XY2-100_X+/SL2DO+	Galvo X position command output signal- Galvo X position command output signal+
4，12	XY2-100_Y-_DO XY2-100_Y+_DO	Galvo Y position command output signal- Galvo Y position command output signal+
5，13	SPI_SDI-/XY2-100_X-_DI SPI_SDI+/XY2-100_X+_DI	Galvo X position command feedback signal- Galvo X position command feedback signal+
6，14	XY2-100_Y-_DI XY2-100_Y+_DI	Galvo Y position command feedback signal-

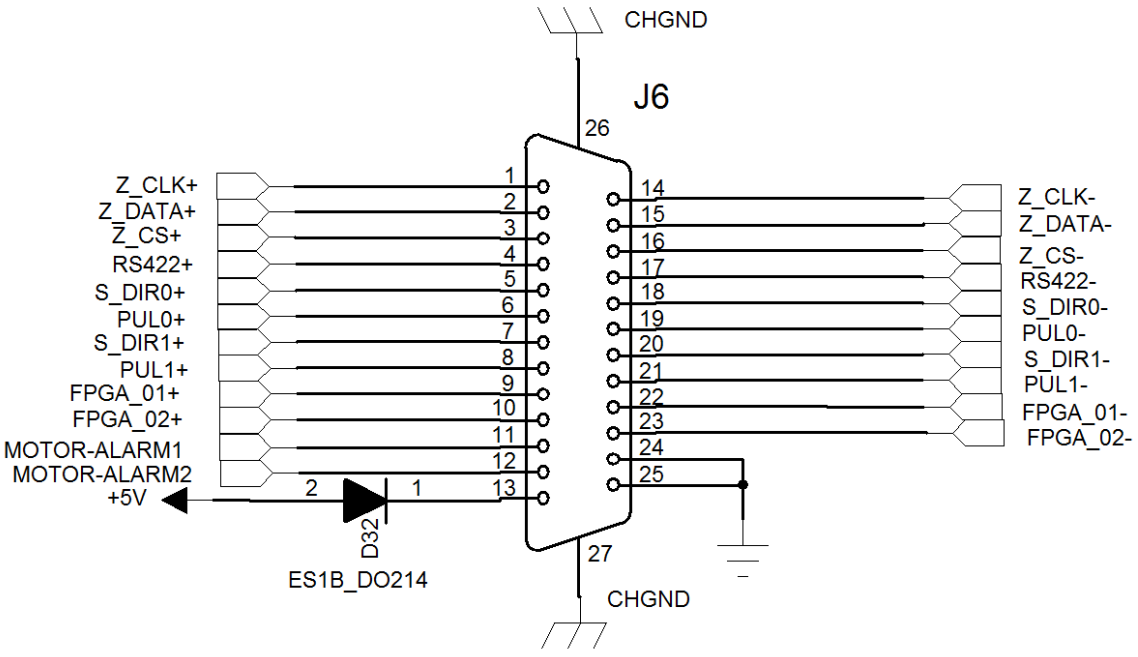
		Galvo Y position command feedback signal+
7，15	SL2DATA_IN- SL2DATA_IN+	SL2 Feedback signal- SL2 feedback signal +
8	GND	GND

This control card can support three galvanometer protocols, namely XY2-100 protocol, SPI protocol and SL2 protocol. Among them, XY2-100 protocol is generally a common protocol internationally, and most galvanometer square heads adopt this protocol; SPI protocol is a protocol defined by Han's Laser itself, and only some galvanometer square heads generated by Han's Laser support this protocol; SL2 protocol is a protocol customized by SCANLAB. Our company can produce this protocol square head by ourselves, and can also control the galvanometer of SCANLAB's SL2 protocol. The marking protocol can be switched in the program through the UDM_SetProtocol() function.

When wiring, you need to select different pins for connection according to the galvanometer square head protocol. It is recommended to use a shielded twisted pair for digital signal connection.

3.2.3 J6 multi-function IO port

This port uses a DB25 male interface, which is mainly used for Z-axis control and other servo motor axis control functions. Its definition is as follows:



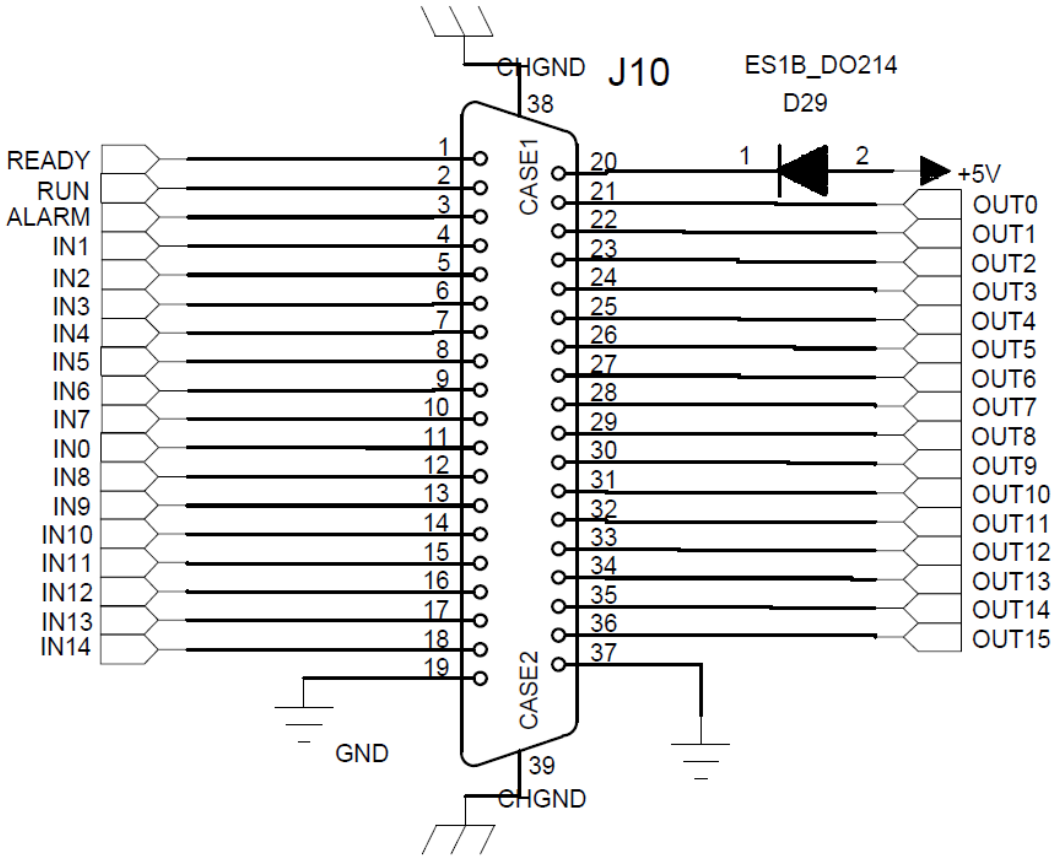
PIN	Signal name	Illustrate
1,14	Z_CLK+/ Z_CLK-	Z-axis clock signal +/ Z-axis clock signal -
2,15	Z_DATA+/ Z_DATA-	Z-axis data signal +/ Z-axis data signal -
3,16	Z_CS+/ Z_CS-	Z-axis chip select signal +/ Z-axis chip select signal -
4,17	RS422+/RS422+	RS422+/RS422+
5,18	S_DIR0+、S_DIR0-	Stepper motor DIR0+, stepper motor DIR0-

6,19	PUL0+/PUL0-	Stepper motor PUL0+, stepper motor PUL0-
7,20	S_DIR1+/ S_DIR1-	Stepper motor DIR1+, stepper motor DIR1-
8,21	PUL1+/PUL1-	Stepper motor PUL1+, stepper motor PUL1-
9,22	FPGA_01+/ FPGA_01-	Reserved differential pair signal: FPGA_01+/FPGA_01-
10,23	FPGA_02+/ FPGA_02-	Reserved differential pair signal: FPGA_02+/FPGA_02-
11,12	MOTOR-ALARM1,ALARM2	Motor drive alarm input signal
13	+5V	5V output
24,25	GND	GND

In 3D marking applications, because the Z-axis needs to be controlled, it is necessary to connect the Z-axis clock signal ±, Z-axis data signal ±, and Z-axis chip select data ± in this definition. The protocol for controlling the Z-axis adopts the XY2-100 protocol.

3.2.4 J10 General purpose IO ports

This port uses a DB37 female interface, which functions as a general IO and can be used to receive input signals and control output
signals.

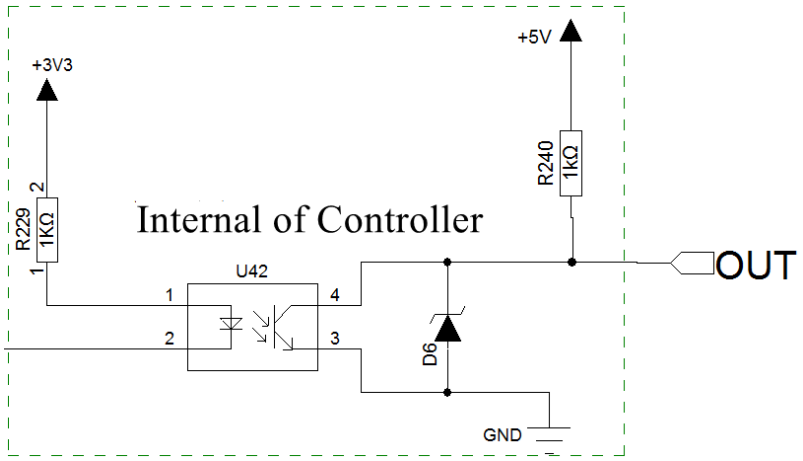


① Output port (the voltage between GND and high level is 5V)

Wherein, READY, RUN, and ALARM are the output signals of the control card status:

After the control card is powered on successfully, READY outputs 5V voltage, indicating that it is in the ready or idle state at this time, and the control card indicator light is displayed in orange;
In the marking operation state, RUN outputs 5V voltage, indicating that it is in the marking state at this time, and the control card indicator light is displayed in green;
When the control card alarms, ALARM outputs 5V voltage. Under normal circumstances, there will be alarm output only in special applications such as flight marking, four-axis linkage, and customization. There will be basically no alarm output in conventional marking.
OUT0~OUT15 are 16-way general output ports with an output voltage of 5V. The output pin can be controlled to be pulled high or low through the UDM_SetOutputOn_GMC4 and UDM_SetOutputOff_GMC4 functions.

Its internal circuit diagram is as follows:



② Input port (the designated input pin is turned on when shorted to +5V)

IN0 to IN14 are 15 general input ports. In general, IN0 is often used as a trigger port. For example, the foot trigger function is enabled in the software, or the UDM_FootTrigger() function is called in further development. After starting marking, IN0 must be externally triggered to start marking, otherwise it will be in a waiting state.

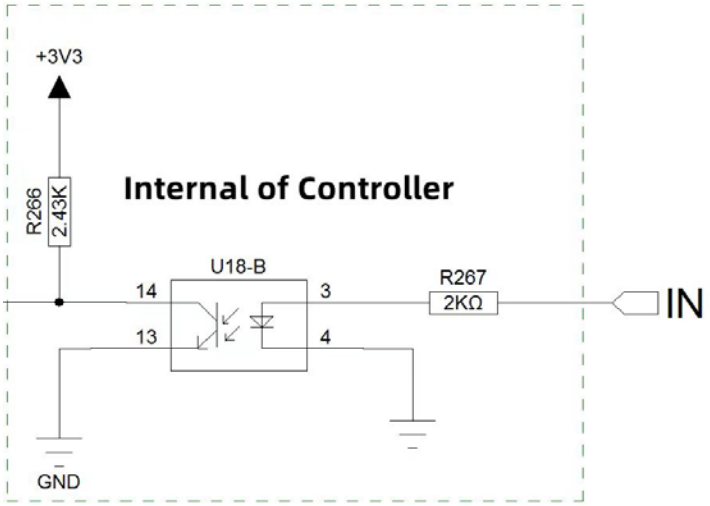
Calling the HM_GetInput_GMC4 function can obtain all pin input information.

IN1~IN4 are often used as file selection ports. For example, in offline marking or signal marking, different files can be selected for marking according to these four signals. These four pins form a binary number, "0000"~"1111" a total of 16 different file numbers，

IN4	IN3	IN2	IN1	Drawing file number
0	0	0	0	No. 0
0	0	0	1	No.1
0	0	1	0	No. 2
0	0	1	1	No. 3
0	1	0	0	No. 4
1	0	0	0	No. 8
1	1	1	1	No.15

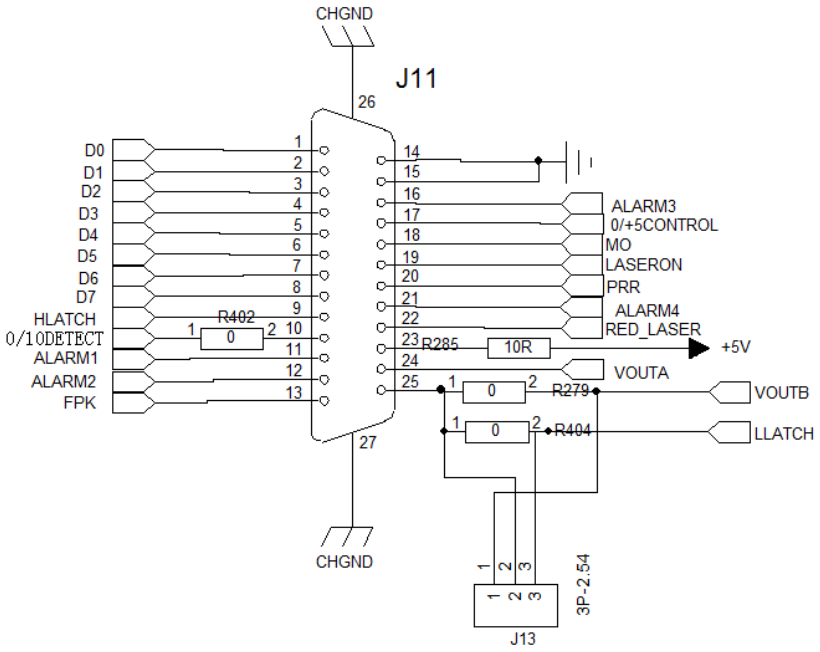
When giving a trigger signal externally, you should follow the principle of setting the values of IN1~IN4 first, and then give the IN0 signal, and then you can start marking. That is, IN4~IN1 controls which graphic to mark, and IN0 controls when to start marking.

Its internal circuit diagram is as follows:



3.2.5 J11 Laser control interface

This port uses a DB25 female port and is mainly used for laser control.



D0~D7：8-bit digital output is mainly used for energy control of lasers. Common lasers such as IPG, JPT, and Raycus are all controlled through these eight pins. We can understand that the 8 pins are a binary number. Binary 11111111, decimal 255, is also 100% energy. Binary 01111111, decimal 127, is about 50% energy. In the program, this value can be changed through the LaserPower variable in the structure MarkParameter. Input 50 represents 50% energy output, and input 100 represents 100% energy output.

Pin9 HLATCH，Energy latch signal output. This signal is used to tell the laser that the energy needs to be changed. When we change the energy in the software, this pin will change automatically. Generally, this signal will change with D0~D7.

Pin10 Pulse width enable signal，This signal can be ignored. For the convenience of customer

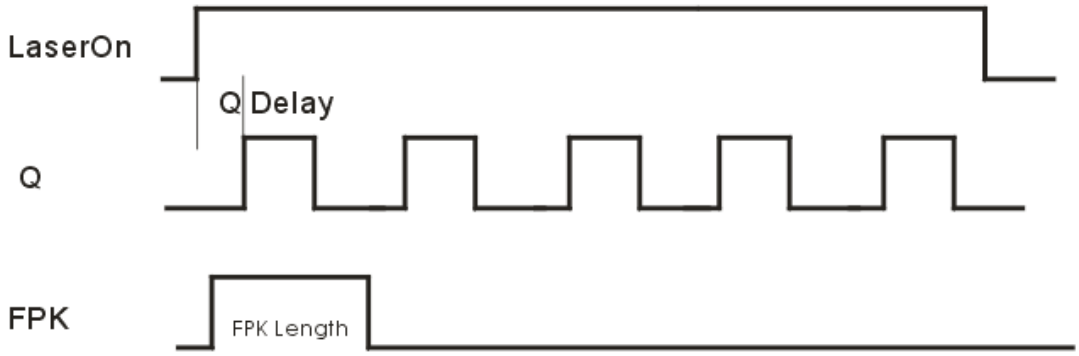
wiring, it can be replaced by pin 22 (pin 22 of the laser is generally the pulse width enable pin);

Pin11、Pin12、Pin16、Pin21，That is, ALARM1, 2, 3, 4 signals are input signals used to receive laser alarm status input. In the laser, these four pins can form 16 laser states, and the laser state at the moment can be judged based on the four input pins. **Different combinations of lasers from different manufacturers and models have different meanings**， You need to check the laser manual for details. Currently, the general control card only accepts these four input states, but does not process them. Users can call the HM_GetLaserInput() function to obtain the status information, and then determine whether the laser alarm is on according to the laser manual. The following is the status information of a YLPN 19 rack mountable series of IPG lasers for reference, **the status feedback of other models may be different from this!**

Message	Type	Signal status			
		Status 0	Status 1	Status 2	Status 3
		Pin 16	Pin 21	Pin 11	Pin 12
Error: Temperature	Fault	LOW	LOW	LOW	LOW
Error: Back reflection	Fault	HIGH	LOW	LOW	LOW
Laser waiting for emission	Status	LOW	HIGH	LOW	LOW
System alarm	Fault	HIGH	HIGH	LOW	LOW
Fiber defective	Fault	LOW	LOW	HIGH	LOW
Safety interlock open	Fault	HIGH	LOW	HIGH	LOW
Laser not ready for emission	Fault	LOW	HIGH	HIGH	LOW
Error: Opt. connection YLP-OLRC	Fault	HIGH	HIGH	HIGH	LOW
Critical alarm	Fault	LOW	LOW	LOW	HIGH
Unexpected interruption of emission	Fault	HIGH	LOW	LOW	HIGH
Laser power supply is OFF	Status	LOW	HIGH	LOW	HIGH
Emission ON	Status	HIGH	HIGH	LOW	HIGH
Safety circuit blocked	Fault	LOW	LOW	HIGH	HIGH
Voltage opt. remote control	Fault	HIGH	LOW	HIGH	HIGH

Table 13: Messages of the parallel interface I/O

Pin13 FPK（First Pulse Killer）The first pulse suppression signal is output. This signal mainly solves the problem of excessive energy at the first point of the laser. Through this signal, the energy of the first laser pulse can be suppressed to ensure that its light output energy is always in a relatively stable state. Its timing is shown in the figure below.



LaserOn, Q, FPK relationship

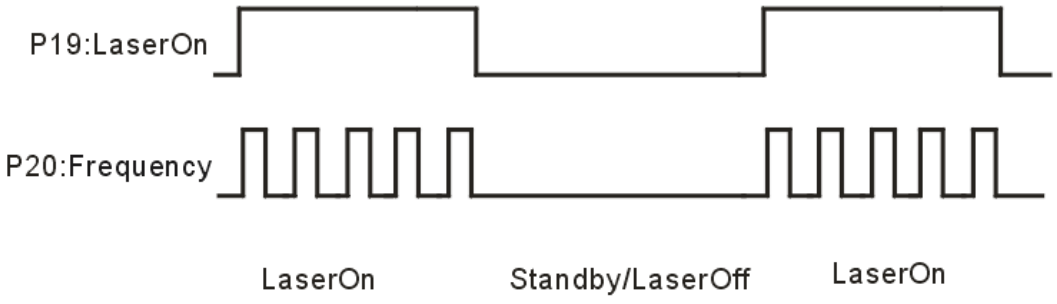
Pin14、Pin15，Control card reference ground

Pin17 +5V Voltage output, this signal is generally used to provide 5V voltage to the 17th pin of the IPG laser, otherwise the laser will alarm.

Pin18 The laser enable output pin is called MO, enable, etc. Before the laser starts to emit light, this signal will be set to a high level in advance.

Pin19 The output signal of the laser pin is called PA, LaserOn, Gate, modulation and so on. The essence of the signal is that the laser is emitted when the pin is at a high level (5V) and the laser is turned off when the pin is at a low level (0V). The laser is controlled by switching between high and low levels.

Pin20 The frequency output pin is mainly used to output TTL type pulse signals. In a pulsed laser, the laser will emit a laser point when it receives a pulse signal. The higher the frequency, the more laser points will be emitted.。**CO2 or UV laser,** in general, you only need to connect 14 (GND) and Pin20 (Q frequency). In some cases, you may also need to connect Pin19 to switch the laser. In the MarkParameter structure, you can control it through these four parameters: DutyCycle, Frequency, StandbyFrequency, and StandbyDutyCycle. DutyCycle represents the duty cycle of the frequency when the laser is marking, Frequency represents the frequency (kHz) when the laser is marking, StandbyFrequency represents the frequency (kHz) after the marking is completed or when jumping, and StandbyDutyCycle represents the frequency duty cycle after the marking is completed or when jumping.



Pin22 Red light output pin, or pulse width enable pin. When used as red light, directly call HM_SetGuidLaser function to directly control the opening or closing of this output pin. If the parameter of PulseWidthMode in the structure MarkParameter is set to 1 in the program, then this pin will be used as a pulse width enable pin.

Pulse width control function pin

Pins 2/3/22 of DB25 are not only used for basic control of the laser, but also for laser pulse width control.

Table 10 Fiber laser pulse width control pins and signals

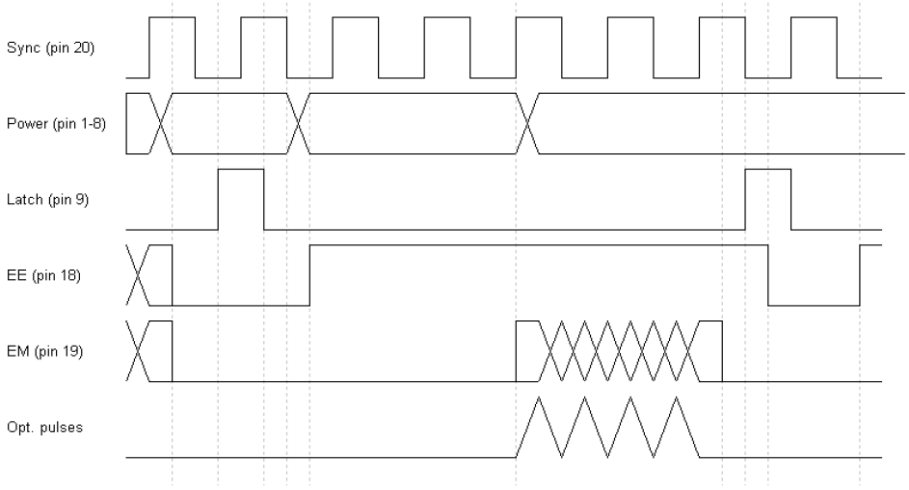
PIN NO.	Name	Describe
2	Serial Input	Laser serial input, the data bit is set synchronously with the rising edge of the serial clock
3	Serial Clock	Serial digital clock, 8kHz≤clock frequency≤10kHZ, 10kHZ is recommended
22	Enable	Pulse width control enable, High: Enable, pins 2 and 3 control the pulse width Low or not connected: Disable

P23 +5V High level output, after the control card is powered on, the default output is +5V.

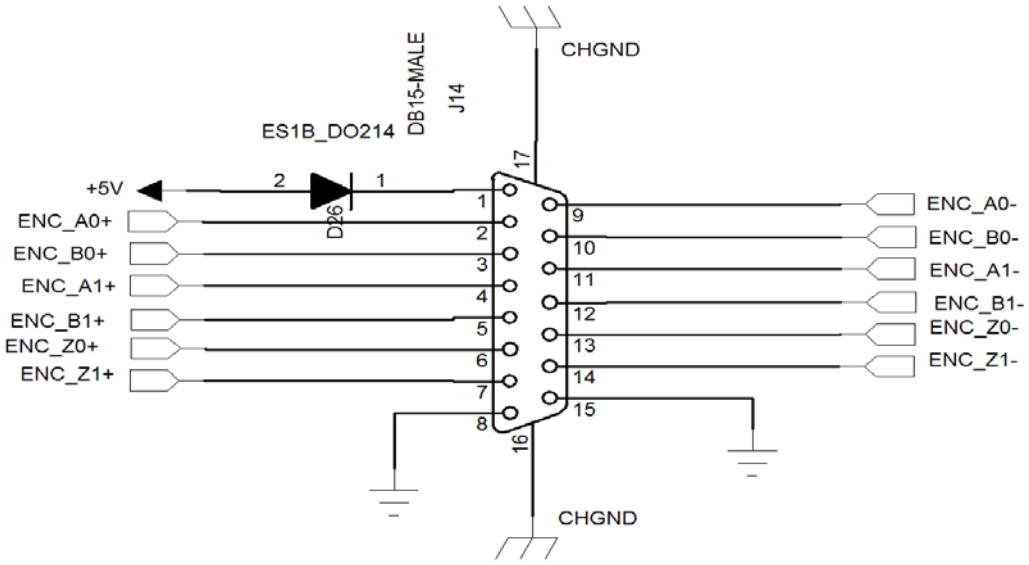
P24，VoutA Analog output，Its output voltage is 0~10V. If the parameter of AnalogMode in the structure MarkParameter is set to 1 in the program, this pin will be used as an energy output pin, with 10V output at 100% power and 5V output at 50% power. The value of this analog quantity can also be set through the VoutA parameter in the HM_SetAnalog (float VoutA, float VoutB) function. It should be noted that when AnalogMode=1, the HM_SetAnalog function is invalid for VoutA.

P25，VoutB Analog output，Its output voltage is 0~10V. The value of this analog quantity can be set through the VoutB parameter in the HM_SetAnalog(float VoutA,float VoutB) function. When VoutB=0.8, the output is 8V.

The overall timing diagram of the laser signal is as follows:



3.2.6 Codewheel input signal



PIN	Signal name	Illustrate
2，9	ENC_A0+ ENC_A0-	Encoder wheel A0+ input signal (horizontal axis) Encoder wheel A0- input signal (horizontal axis)
3，10	ENC_B0+ ENC_B0-	Encoder wheel B0+ input signal (horizontal axis) Encoder wheel B0- input signal (horizontal axis)
4，11	ENC_A1+ ENC_A1-	Encoder wheel A1+ input signal (vertical axis) Encoder wheel A1- input signal (vertical axis)
5，12	ENC_B1+ ENC_B1-	Encoder wheel B1 + input signal (vertical axis) Encoder wheel B1 - input signal (vertical axis)
1	+5V	5V The positive terminal of the output power supply
8、15	GND	The negative polarity end (ground signal) of the 5V output power supply is the ground signal of the control card.
10,11	CHGND	Socket housing

This port is mainly used to collect external pulse signals and calculate the distance based on the number of pulse signals. If it is a normal single-axis flight, you only need to connect the horizontal axis, and the external encoder wheel needs to use a differential input signal.

In a four-axis linkage application, both the horizontal axis and the vertical axis need to be wired.

Chapter 4 Further Development Description

4.1 Basic file description

In addition to providing supporting laser marking software, the GMC control card also provides users with a dynamic link library for further development. The files used include:

HM_Comm.dll： This DLL is mainly responsible for data exchange between the host computer software and the control card, and is also the connection library that HM_HashuScan.dll relies on;

HM_HashuScan.dll： This file has two main functions: ① Generate the UDM.BIN file required for marking; ② Interact with the control card and transmit data, etc.

Head File HM_HashuUDM.h： It is mainly used to generate marking data, including setting square head protocol, square head motor parameters, laser parameters, basic graphics (points, lines, polygons, circles, etc.) and other function interfaces;

Head File 件 HM_HashuScan.h： It mainly includes function interfaces such as finding the control card IP, connecting to the control card, sending data, starting marking, and stopping marking;

System.ini： Configuration file, users only need to put it in the directory where EXE is located. The lack of this configuration file may cause the card's IP address to be unable to be found

It also contains two corresponding LIB files HM_HashuScan.lib and HM_Comm.lib;

This software basic development library is compiled with VS2010. Users are advised to use VS version higher than 2010 for development. It supports C++ and C# languages.

Notice: HM_HashuScan.dll, HM_Comm.dll, System.ini and EXE program must be placed in the same directory.

If users want to quickly apply, they can skip this chapter and go directly to the DEMO Development Guide in Chapter 4. They can quickly develop applications based on the DEMO. The function descriptions in this chapter can be reviewed again when they do not understand.

4.2 HM_HashuScan.h header file

4.2.1 Parameter address definition description

In HM_HashuScan.dll, all functions (except some functions) return a 32-bit integer.
The general definition is as follows:

```
#define HM_OK                   0x00000000    // Success
#define HM_FAILED               0x00000001    // Fail
#define HM_UNKNOWN              0xFFFFFFFF    // Unknown
```

The device connection status is defined as follows:

```
#define HM_DEV_Connect          0x00000000    // Connection status
#define HM_DEV_Ready            0x00000001    // The state is Ready
```

#define HM_DEV_NotAvailable 0x00000002 // The status is offline

The message is defined as follows:

#define HM_MSG_DeviceStatusUpdate 5991// Device status message, search IP address, etc

#define HM_MSG_StreamEnd 6012//(stream) marking file UDM.BIN Download completion message

#define HM_MSG_UDMHalt 6035// Marking completion message

#define HM_MSG_ExecProcess 6037// Marking execution progress, that is, marking percentage

4.2.2 Definition of marking position point structure

```
typedef struct structUdmPos
{
    float x;    //x coordinate
    float y;    //y coordinate
    float z;    //z coordinate
    float a;    // reserve
}structUdmPos;
```

structUdmPos is the data structure used by the motion instruction function parameters. x, y, and z represent the positions where the galvanometer is to move. In two-dimensional marking, z can be 0. a is a reserved position, which is temporarily reserved and can be set to 0.

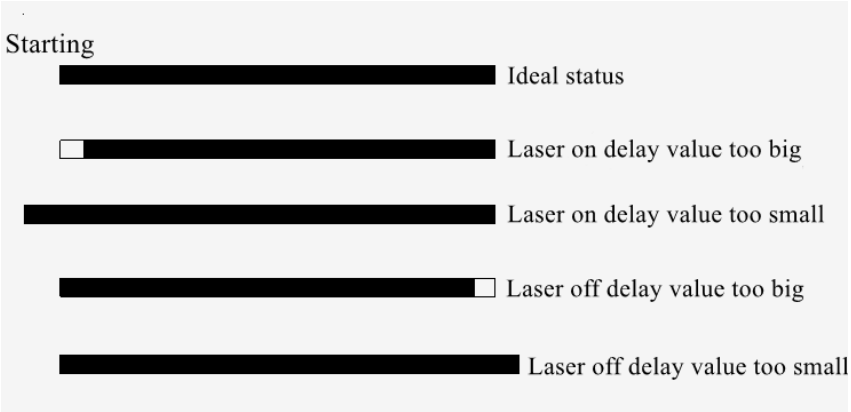
4.2.3 Marking parameter data structure definition

The MarkParameter structure contains the galvanometer parameters and laser parameters, and its structure definition is as follows:

```
typedef struct MarkParameter
{
unsigned int MarkSpeed;//Marking speed (mm/s)
unsigned int JumpSpeed;//Jump speed (mm/s)
unsigned int MarkDelay;//Marking delay (us)
unsigned int JumpDelay;//Jump delay (us)
unsigned int PolygonDelay;//Turn delay (us)
unsigned int MarkCount;//Marking times
float LaserOnDelay;//Laser on delay (unit us)
float LaserOffDelay;//Laser off delay (unit us)
float FPKDelay;//First pulse suppression delay (unit us)
float FPKLength;//First pulse suppression length (unit us)
float QDelay;//Light Q frequency delay (unit us)
float DutyCycle;//Duty cycle when emitting light, (0~1)
float Frequency;//Frequency when emitting light kHz
float StandbyFrequency;//Frequency when not emitting light Q (unit kHz);
float StandbyDutyCycle;//Duty cycle when not emitting light Q (0~1);
```

float LaserPower;//Laser energy percentage (0~100), 50 represents 50%
unsigned int AnalogMode;//1 represents the use of analog output to control laser energy (0~10V)
unsigned int Waveform;//SPI laser waveform number (0~63)
unsigned int PulseWidthMode;//0, do not turn on MOPA pulse width enable mode, 1, turn on MOPA laser pulse width enable
unsigned int PulseWidth;//MOPA laser pulse width value unit (ns)}MarkParameter;

Marking speed: the speed of the laser part. The higher the speed, the shorter the time, but the effect may be worse.
Jump speed: the speed of jumping from one graphic to another, which also indicates the speed of the laser trajectory movement.
Laser energy: adjust the laser power. The stronger the energy, the more obvious the pattern marks. The value range is 0~100.
Q frequency: the frequency of the laser light. The higher the frequency, the more points the laser produces. The value range is 0~10000.
Laser on delay: how long does the laser delay before emitting light when starting marking, the value range is -320~＋∞**;**
Too big: the galvanometer moves back too much, the laser is turned on, and the beginning of the stroke will be missing.
Too small: the galvanometer emits light before it is in place, that is, the light is emitted in advance.
Laser off delay: how long does it take to turn off the light after the track is marked, the value range is -320~＋∞;
Too big: the galvanometer track is in place and the next track is started, the laser is still on, which will cause tailing or multiple marking.
Too small: the galvanometer is not fully rotated, the laser is turned off, and the end of the stroke will not be long enough.



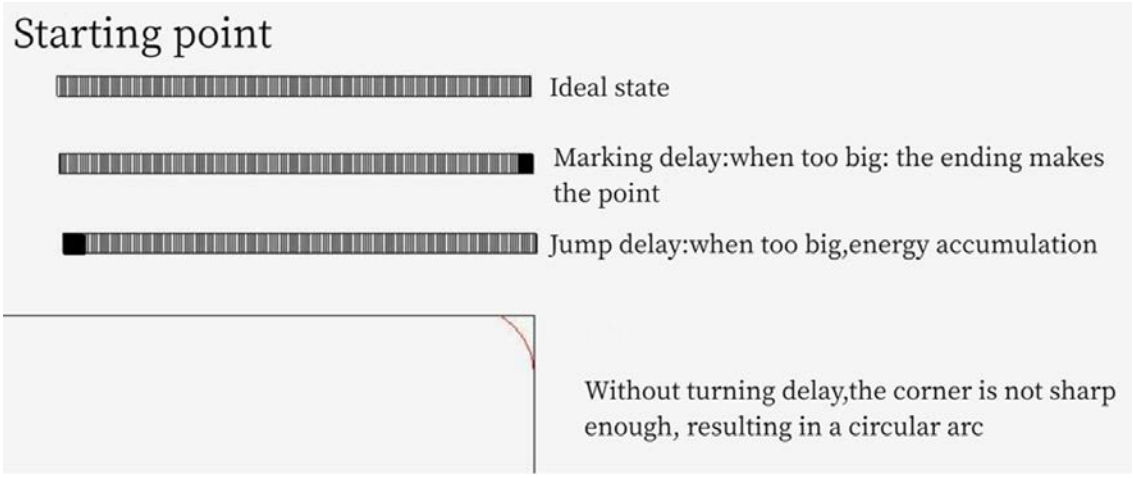
Marking delay (pen delay): the delay time of the galvanometer after the end of marking, the value range is 0~＋∞;
Too large: the end forms a focus.
Too small: the end is deformed and the figure cannot be closed.
Jump delay: the delay time of the galvanometer after the jump, which can also be understood as the delay time of the galvanometer before the light is emitted, the value range is 0~＋∞.
Too large: the starting point causes energy accumulation due to slow speed, forming a focus.

Too small: the starting point is deformed, which may cause the figure to be unable to close.

Turning delay: the delay time of the galvanometer at the turning, the value range is 0~＋∞;

Too large: there is a focus at the turning.

Too small: the turning is not sharp enough, resulting in an arc shape.



4.2.4 HM_HashuScan class function definition

The functions in the HM_HashuScan.h file are mainly used to interact with the control card, that is, to control the marking card online, including sending marking files, starting marking, stopping marking, online reading and setting of IO signals, etc. Because the communication protocol adopts the form of TCP, multiple control cards can be connected in the same network segment. When connecting multiple control cards, the IP address of each card needs to be set differently. The default IP of the control factory is "172.18.34.227". When there are multiple control cards, we can control the specified card through the card index.

Note: The IP index of the card is not unique. The card that is powered on first has an earlier index, and the card that is powered on later has a later index. However, the IP address of each card (172.18.34.227) is unique. You can use the HM_GetIndexByIpAddr() function to get the IP index of the current IP address.

All functions in this header file have an ipIndex parameter, which is the control card IP index. If and only if there is only one card, ipIndex is 0. The meaning of this parameter will not be repeated in the following text.

The following are commonly used function interfaces:

Function name：HM_InitBoard

Purpose： The control card communication is initialized and the network connection is established.

Grammar： HM_InitBoard(HWND hWnd)；

hWnd_ It is the window handle for receiving device messages;

Describe： Initialize the control card communication and specify the message receiving window handle and the agreed device message update ID.

Return value： Initialization success: HM_OK, failure: HM_FAILED;

Function name：HM_ConnectTo

Purpose： Connect to the control card with the specified IP index.

Grammar：HM_ConnectTo(int ipIndex)；

ipIndexControl card IP index；

Describe：Connects to the device specified by index number in the device list.

Return value：Connection successful: HM_OK, connection failed: HM_FAILED;

Function name：HM_ConnectByIpStr(char* pIp);

Purpose：Connect to the control card via IP address.

Grammar：HM_ConnectByIpStr(char* pIp);

pIpControl card IP string, for example”172.18.34.227”；

Describe：Connects to the device specified by index number in the device list.

Return value：Connection successful: HM_OK, connection failed: HM_FAILED;

Function name：HM_DisconnectTo

Purpose：Disconnect the device.

Grammar：HM_Disconnect(int ipIndex);

Describe：Disconnect the control card with the current IP index;

Return value：Connection successful: HM_OK, connection failed: HM_FAILED;

Function name：HM_GetIndexByIpAddr

Purpose：Get the IP index through the specified IP address.

Grammar：HM_GetIndexByIpAddr(char* strIP);

strIP control card IP string, for example "172.18.34.227";

Describe：Get the IP index of the IP address strIP;

Return value：Return the control card IP index;

Function name：HM_GetConnectStatus

Purpose：Get the connection status of the specified control card.

Grammar：HM_GetDeviceStatus(int ipIndex);

Describe：Get the connection status of the device specified by the index number;

Return value：HM_DEV_Connect：The current device is connected;
HM_DEV_Ready： The current device can be connected;
HM_DEV_NotAvailable：The current device is not available;

Function name：HM_DownloadMarkFile

Purpose：Download the UDM.BIN marking file to the temporary storage area of the control card;

Grammar：HM_DownloadMarkFile(int ipIndex,char *filePath,HWND hWnd);

filePath，Marking file path string;

hWnd Receive the window handle of the download progress;

Describe：Download the generated UDM.BIN marking file to the control card. After the download is complete, you can start marking. It should be noted that this function is non-blocking. Calling this function does not mean that the marking file has been sent. You need to wait for the download completion message in the message function before marking.

Return value：Start download successfully HM_OK, start download failed HM_FAILED;

Function name：HM_DownloadMarkFileBuff

Purpose：Download the marking file to the temporary storage area of the control card through the

memory;

Grammar：HM_DownloadMarkFileBuff(int ipIndex,char *pUDMBuff,int nBytesCount,HWND hWnd);

pUDMBuff，Marking data UDM The starting address in memory;

nBytesCount The number of bytes of the downloaded UDM code;

hWnd Receive the window handle of the download progress;

Describe： Use the UDM_GetUDMBuffer function to download the generated UDM data to the control card through the memory. After the download is complete, you can start marking. It should be noted that this function is non-blocking. Calling this function does not mean that the marking file has been sent. You need to wait for the download completion message in the message function before marking.

You can choose either this function or the HM_DownloadMarkFile function. It is recommended to use the HM_DownloadMarkFile function because it can save data to the hard disk and view the data. It is sometimes used to troubleshoot error information or historical records.

Return value：Start download successfully HM_OK, start download failed HM_FAILED;

Function name：**HM_BurnMarkFile**

Purpose：Solidify or clear offline marking files;

Grammar：HM_BurnMarkFile(int ipIndex,bool enable);

enable= true The files are saved to the control card, and the data will not be lost after power failure and restart, thus achieving offline operation;

enable= false Erase offline files;

Describe：When offline marking is required, call this function to solidify the marking file into the flash area of the control card. It can be stored permanently and the data still exists after power failure. It is often used for offline marking function.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_ExecuteProgress;**

Purpose：Check the marking progress;

Grammar：HM_ExecuteProgress(int ipIndex);

Describe：Set a timer. Each time this function is called, the control card will return the marking progress percentage (0~100) through a message.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_StartMark**

Purpose：Start marking;

Grammar：HM_StartMark(int ipIndex);

Describe：After the marking file is downloaded successfully, call this function to start marking;

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_StopMark**

Purpose：Stop marking;

Grammar：HM_StopMark(int ipIndex)

Describe：Stop marking;

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_PauseMark**

Purpose：Pause marking;

Grammar：HM_PauseMark(int ipIndex);

Describe：Pause marking;

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_ContinueMark**

Purpose：Continue marking；

Grammar：HM_ContinueMark(int ipIndex);

Describe：Continue marking after a pause;

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_SetOffset**

Purpose：Set the overall offset of the marking graphic;

Grammar：HM_SetOffset(int ipIndex,float offsetX, float offsetY, float offsetZ);

offsetX X Position offset；

offsetY Y Position offset；

offsetZ Z Position offset；

Describe：Set the offset of all graphics in the marking file online, in mm. After setting the offset, the starting point and end point of the marking are both offset points, that is, the motor zero position becomes the set offset position. Usually, the marking graphic is made at the origin of the coordinate system first, and then the offset is set to offset the marking graphic to the target position.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_SetRotates**

Purpose：Set the overall rotation amount of the marking graphic;

Grammar：HM_SetRotates(int ipIndex,float angle, float centryX, float centryY);

angle Rotation angle;

centryX The X coordinate of the rotation center;

centryY The Y coordinate of the rotation center;

Describe：Set the rotation angle of all graphics in the marking file online, in degrees. Usually, the marking graphic is made at the origin of the coordinate system first, and then the angle is set to rotate the marking graphic to the target position. Usually, this function needs to be used together with HM_SetOffset, first rotating at (0, 0), and then offsetting.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_SetGuidLaser**

Purpose：Turn on or off the red light, corresponding to the P22 pin on the J11 port of the control card

Grammar：HM_SetGuidLaser(int ipIndex,bool enable);；

enable= true Turn on the red light and the 22-pin high level 5V output,

enable= falseTurn off the red light and the 22-pin low level 0V output

Describe：Laser red light switch control;

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：HM_ScannerJump；

Purpose： Move the galvanometer to a specified position;

Grammar： HM_ScannerJump(int ipIndex,float X, float Y, float Z);

Describe： Move the galvanometer to the specified xyz position and keep it there. No laser is emitted during the movement.

Return value： Success: HM_OK, failure: HM_FAILED;

Function name：HM_GetWorkStatus

Purpose： Get the current marking status of the control card;

Grammar： HM_GetWorkStatus(int ipIndex);

Describe： Return to get the marking status,1 ready，2 run，3 Alarm

Return value： Return to marking status

Function name：HM_SetCoordinate；

Purpose： Set the coordinate system;

Grammar： HM_SetCoordinate(int ipIndex, int coordinate);

coordinate，Coordinate system mode, 0~7 Eight types in total，0：XY，1：X-Y，2：-XY，3：-X-Y，4：YX，5：Y-X，6：-YX，7：-Y-X

Describe： Setting the coordinate system during calibration；

Return value： Success: HM_OK, failure: HM_FAILED;

Function name：HM_SetMarkRegion；

Purpose： Set the marking range of the current card;

Grammar： HM_SetMarkRegion(int ipIndex, int region);

Region，Marking range, generally you need to specify the marking range during calibration

Describe： Set the marking range of the current card;

Return value： Success: HM_OK, failure: HM_FAILED;

Function name：HM_GetMarkRegion；

Purpose： Get the marking range of the current card;

Grammar： HM_GetMarkRegion(int ipIndex);

Describe： Get the marking range of the current card, and ensure that the input point coordinates are within ±region/2;

Return value： Marking range;

Function name：HM_DownloadCorrection；

Purpose： Download the correction table to the temporary storage area;

Grammar： HM_DownloadCorrection(int ipIndex,char *filePath,HWND hWnd);

filePath Correction table path

hWnd Window handle

Describe： Download the correction table in the specified path to the control card. It will take effect immediately after setting, but the data will be lost after power failure and restart.

Return value： Success: HM_OK, failure: HM_FAILED;

Function name：HM_BurnCorrection；

Purpose：Curing correction table;

Grammar：HM_BurnCorrection(int ipIndex,char *filePath,HWND hWnd);

filePath Correction table path

hWnd Window handle

Describe：The correction table under the specified path is solidified into the flash memory of the control card. The data still exists after power failure and restart.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_SelectCorrection**；

Purpose：Selecting a calibration table；

Grammar：HM_SelectCorrection(int ipIndex,int crtIndex);

crtIndex Correction table index, 0 or 1

Describe：The control card supports dual correction table storage. In the dual-galvanometer application on one card, different galvanometers can be controlled by changing the correction table index. Correction table 0 is used by default.。

Function name：Success: HM_OK, Failure: HM_FAILED；

Function name：**HM_GetInput_GMC2**；

Purpose：Get the external input status of the second-generation GMC2 control card;

Grammar：HM_GetInput_GMC2(int ipIndex);

Describe：Returns the input information of the control card, and the return value is converted to binary. For example, "1100" means IN3 and IN2 are conducting. IN1 and IN0 are not conducting (starting from IN0)

Return value：Returns the current IO input status of the control card

Function name：**HM_GetInput_GMC4**；

Purpose：Get the external input status of the fourth-generation GMC4 control card;

Grammar：HM_GetInput_GMC4(int ipIndex);

Describe：Returns the input information of the control card, and the return value is converted to binary. For example, "1100" means IN3 and IN2 are conducting. IN1 and IN0 are not conducting (starting from IN0)

Return value：Returns the current IO input status of the control card

Function name：**HM_GetLaserInput**

Purpose：Get the laser alarm input status;

Grammar：HM_GetLaserInput(int ipIndex);

Describe：Returns the laser alarm status input information, and the return value is converted to binary. For example, "1100" means Alarm4, Alarm3 are high, Alarm2, Alarm1 are low (commonly used for IPG, Raycus, MOPA lasers)

Return value：Returns the laser alarm input status

Function name：**HM_SetOutputOn_GMC2**；

Purpose：Set the GMC2 control card to a high level 5V output pin online

Grammar：HM_SetOutputOn_GMC2(int ipIndex,int nOutIndex);

nOutIndex，The output signal index value to be pulled high

Describe：Set the control card output signal online. For example, writing 1 means pulling up the OUT1 output signal, and writing 2 means pulling up the OUT2 output signal. 5V is a high level and

0V is a low level.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_SetOutputOn_GMC4**；

Purpose：Set the GMC4 control card to specify the output pin as high level 5V online

Grammar：HM_SetOutputOn_GMC4(int ipIndex,int nOutIndex);

nOutIndex，The output signal index value to be pulled high

Describe：Set the control card output signal online. For example, writing 1 means pulling up the OUT1 output signal, and writing 2 means pulling up the OUT2 output signal. 5V is a high level and 0V is a low level.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_SetOutputOff_GMC2**；

Purpose：Set the GMC2 control card to specify the output pin as high or low level 0V online

Grammar：HM_SetOutputOff_GMC2(int ipIndex,int nOutIndex);

nOutIndex，The output signal index value to be pulled low

Describe：Set the control card output signal online. For example, writing 1 means pulling down the OUT1 output signal, and writing 2 means pulling down the OUT2 output signal. 5V is a high level, and 0V is a low level.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_SetOutputOff_GMC4**；

Purpose：Set the GMC4 control card to specify the output pin as high or low level 0V online

Grammar：HM_SetOutputOff_GMC4(int ipIndex,int nOutIndex);

nOutIndex，The output signal index value to be pulled low

Describe：Set the control card output signal online. For example, writing 1 means pulling down the OUT1 output signal, and writing 2 means pulling down the OUT2 output signal. 5V is a high level, and 0V is a low level.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_SetAnalog**；

Purpose：Set the analog voltages VoutA and VoutB online

Grammar：HM_SetAnalog(int ipIndex,float VoutA,float VoutB);

VoutA，Set the analog value of the VoutA pin

VoutB，Set the analog value of VoutB pin

Describe：Set two analog voltages online. The voltage of these two signals is in the range of 0~10V. If the parameter is set to 0.5, the voltage output is 5V, if it is set to 0.8, the output is 8V, and if it is set to 1.0, the output is 10V.

Return value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_GetFeedbackPosXY**；

Purpose：Get XY galvanometer position feedback

Grammar：HM_GetFeedbackPosXY(int ipIndex,short *fbX,short *fbY);

fbX，X Motor actual position feedback

fbY，YMotor actual position feedback

Describe：This function can be used to obtain the actual position feedback of the XY motor

Hint：The galvanometer needs to have position feedback function, otherwise this function is

meaningless.

Return Value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_GetCmdPosXY**；

Purpose：Get the position command sent to the XY galvanometer at the current moment

Grammar：HM_GetCmdPosXY(int ipIndex,short *cmdX,short *cmdY);

cmdX，Send position command to X motor

cmdY，Send position command to Y motor

Describe：This function can be used to obtain the actual position feedback of the XY motor

Hint：This only represents the command sent by the control card. It still has value when the galvanometer motor is not connected.

Return Value：Success: HM_OK, Failure: HM_FAILED；

Function name：**HM_ClearCloseLoopAlarm**；

Purpose：When using the closed loop function, this function can clear the system alarm

Grammar：HM_ClearCloseLoopAlarm(int ipIndex);

Describe：Clear the alarm and reset the system. If the galvanometer motor itself has an alarm, it cannot be cleared here. Please check the status of the galvanometer motor.

Hint：This function needs to be used together with HM_GetGalvoStatusInfo and UDM_SetCloseLoop functions.

Return Value：Success: HM_OK, failure: HM_FAILED;

Function name：**HM_GetGalvoStatusInfo**；**needs customization**

Purpose：When using the closed-loop function, obtain feedback information on the status of each galvanometer motor

Grammar：HM_GetGalvoStatusInfo (int ipIndex,int galvoType);

galvoType Motor type: 0 analog square head or square head with position feedback only, 1 digital photoelectric square head, 2 digital grating square head

Describe：When the closed loop function is enabled, this function can be used to read the galvanometer motor head information.

Hint：When using this function, please make sure that the galvanometer has a closed-loop function and can provide feedback such as temperature, current and other alarm functions.

This function needs to be used together with HM_ClearCloseLoopAlarm and UDM_SetCloseLoop functions.

Function name：Returns status information. When the return value is converted to binary, each bit has the following meaning, starting from the low bit to the high bit.

Different types of galvanometer motors have different meanings for the returned values.

When Galvo = 0, the return value has the following meanings:

bit 0: Fixed-point light alarm, 0 is normal, 1 is abnormal. The control card keeps sending position instructions during marking, and the control card will constantly monitor the actual position feedback of the galvanometer. If the galvanometer is always in the same position, an alarm will be triggered.

bit 1: Following error alarm, 0 is normal, 1 is abnormal. The control card keeps sending position instructions during marking, and the control card will constantly monitor the actual position feedback of the galvanometer. If the difference between the sent command position and the feedback position is too large, an alarm will be triggered. For example, if we originally wanted to mark a square, but the actual result is a circle, it is considered that the difference between the command and the actual is too large.

When Galvo = 1, the return value has the following meanings:

bit 0: transmission status, 0 normal, 1 abnormal

bit 1, 2: reserved

bit 3: Y zeroing success, 0 abnormal unsuccessful, 1 normal zeroing success
bit 4, 5: reserved
bit 6: Y motor overtemperature, 0 normal, 1 abnormal
bit 7: Y motor overcurrent, 0 normal, 1 abnormal
bit 8, 9, 10: reserved
bit 11: X zeroing success, 0 abnormal unsuccessful, 1 normal zeroing success
bit 12, 13: reserved
bit 14: X motor overtemperature alarm, 0 normal, 1 abnormal
bit 15: X motor overcurrent alarm, 0 normal, 1 abnormal
bit 16: fixed-point light alarm, 0 normal, 1 abnormal. When marking, the control card keeps sending position instructions, and the control card will constantly monitor the actual position feedback of the galvanometer. If the galvanometer is always in the same position and does not move, an alarm will be triggered.
bit 17: Following error alarm, 0 is normal, 1 is abnormal. During marking, the control card keeps sending position commands, and the control card will constantly monitor the actual position feedback of the galvanometer. If the difference between the sent command position and the feedback position is too large, an alarm will be triggered. For example, if we originally wanted to mark a square, but the actual result is a circle, it is considered that the difference between the command and the actual is too large.

When Galvo = 2, the return value has the following meanings:
bit 0: galvanometer motor status line connection status, 0 normal, 1 abnormal
bit 1: galvanometer motor initialization status, 0 normal, 1 abnormal
bit 2, 3: reserved
bit 4: Y motor initialization, 0 normal, 1 abnormal
bit5: Y motor encoder, 0 normal, 1 abnormal
bit 6: Y motor overtemperature, 0 normal, 1 abnormal
bit 7: Y motor overcurrent, 0 normal, 1 abnormal
bit 8, 9, 10, 11: reserved
bit 12: X motor initialization, 0 normal, 1 abnormal
bit 13: X motor encoder, 0 normal, 1 abnormal
bit 14: X motor overtemperature alarm, 0 normal, 1 abnormal
bit 15: X motor overcurrent alarm, 0 normal, 1 abnormal
bit 16 : Fixed-point light alarm, 0 is normal, 1 is abnormal. The control card keeps sending position instructions during marking, and the control card will constantly monitor the actual position feedback of the galvanometer. If the galvanometer is always in the same position, an alarm will be triggered.
bit 17 : Following error alarm, 0 is normal, 1 is abnormal. The control card keeps sending position instructions during marking, and the control card will constantly monitor the actual position feedback of the galvanometer. If the difference between the sent command position and the feedback position is too large, an alarm will be triggered. For example, if we originally want to mark a square, but the actual result is a circle, it is considered that the difference between the command and the actual is too large.

4.3 HM_HashuUDM.h Head file library

This instruction library is mainly used to generate marking data, and finally generates a UDM.BIN

file. This file contains marking parameters, laser control parameters, graphic data point coordinates, input and output signals, etc. All control instructions are included in this file, and finally sent to the control card through the communication library function mentioned above. The control card can control the movement of the galvanometer and emit laser.

Note: During the marking process, the contents of the UDM.BIN file are actually read sequentially from the beginning to the end. When a line of content is read, a line of instruction is executed. Therefore, the order in which these instruction functions are called represents the order of execution during marking.

Function name：UDM_NewFile

Purpose：Create a new UDM file；

Grammar：int UDM_NewFile();

Describe：Create a new UDM file；

Return Value：0 - success; 1 - failure

Note: This function must be called before other functions can be called;

Function name：UDM_SaveToFile

Purpose：Save the UDM executable file

Grammar：int UDM_SaveToFile(char* strFilePath);

Describe：Save the UDM executable file to the file path specified by strFilePath；

Return Value：0 - success; 1 - failure

Note: After calling this function, the generation ends. You need to call UDM_NewFile before using it again.

Function name：UDM_GetUDMBuffer

Purpose：Get the pointer and byte number of the UDM marking file in the memory, and use it with the HM_DownloadMarkFileBuff function

Grammar：int UDM_GetUDMBuffer(char** pUdmBuffer, int* nBytesCount);

Describe：Get the pointer of the generated UDM data in the memory and the byte length of the code. You can choose between this function and the UDM_ SaveToFile function. It is recommended to use the UDM_ SaveToFile function because it can save the data to the hard disk and view the data. It is sometimes used to troubleshoot error information or historical records.

Example：
```
UDM_EndMain(); //End of UDM file
char* pBuffer;
int     nCount;
UDM_GetUDMBuffer(&pBuffer, &nCount);
HM_DownloadMarkFileBuff(ipIndex,pBuffer,nCount,this->m_hWnd);
```

Return Value：0 - success; 1 - failure

Note: After calling this function, the generation ends. You need to call UDM_NewFile before using it again.

Function name：UDM_Main

Purpose：The main function instruction of the program starts

Grammar：int UDM_Main();

Describe：Provides the main function entry of the UDM program；

Return Value：The starting address of the instruction

Function name：UDM_EndMain

Purpose：The main function of the program ends

Grammar：int UDM_EndMain();

Describe：The main function of the UDM program ends；

Return Value：The starting address of the instruction

Function name：UDM_SetProtocol

Purpose：Marking card protocol selection

Grammar：int UDM_SetProtocol(int nProtocol,int nDimensional);

Describe： nProtocol 0 means SPI protocol, 1 means XY2-100 protocol, 2 means SL2 protocol;

nDimensional 0 means flat 2D marking, 1 means 3D marking

Note: Common Scanheads are almost all XY2-100 open-loop protocols.

Return Value：The protocol command value to be set

Function name：UDM_RepeatStart

Purpose：Program limited loop start

Grammar：int UDM_RepeatStart(int repeatCount);

Describe：Set the number of cycle marking

repeatCount Cycles；

Return Value：The starting address of the instruction

Function name：UDM_RepeatEnd

Purpose：The program ends after a limited number of cycles

Grammar：int UDM_RepeatEnd(int startAddress);

Describe：End of loop

startAddress The loop target address given by the UDM_RepeatStart function;

Note: You cannot nest loops in the middle of loops. Instructions between start and end can be looped. After the current loop ends, you can call start loop and end loop again.

Return Value：The starting address of the instruction

Function name：UDM_Jump

Purpose：Jump Instructions

Grammar：int UDM_Jump (float x, float y, float z);

Describe：The galvanometer jumps from the current position to the (x, y, z) position, and no laser is emitted during the jump;

The difference from HM_ScannerJump is that after calling the HM_ScannerJump function, the jump instruction will be executed immediately，The UDM_Jump function needs to be written into the marking file, and the jump will be executed when it runs to this point.

Return Value：The starting address of the instruction

Function name：UDM_Wait

Purpose：The program waits for a specified time

Grammar：int UDM_Wait(float msTime);

Describe：When the command position is reached, wait and proceed to the next step after the time is up;

msTime waiting time, in ms

Return Value：The starting address of the instruction

Function name：UDM_SetGuidLaser

Purpose：Turn the red light on or off in command.

Grammar：int UDM_SetGuidLaser(bool enable);

Describe：enable turns on the red light when true, and turns off the red light when false;

Note: It needs to be used together with the online red light setting function HM_SetGuidLaser. When calling the HM_SetGuidLaser function, the red light can be turned on or off immediately. When generating a marking file, calling UDM_SetGuidLaser can continue to emit red light during marking.

Return Value：The starting address of the instruction

Function name：UDM_SetInput()

Purpose：Set to wait for input signal

Grammar：int UDM_SetInput(unsigned int uInIndex)

Describe：uInIndex Input signal index value. Adding this function will wait for external input signal, otherwise it will be stuck and will not run further. Currently only IN0~IN7 support this function.

Return Value：The starting address of the instruction

Function name：UDM_SetOutPutAll();

Purpose：Set the output signal in the instruction

Grammar：int UDM_SetOutPutAll(unsigned int uData);

Describe：uData controls all output signals with one click (this information will be written to the UDM.BIN file) "Binary 1111 means all four outputs are pulled high, binary 0011 means out0 and out1 are pulled high, out2/out3 are pulled low"

Return Value：The starting address of the instruction

Function name：UDM_SetOutPutOn(unsigned int nOutIndex);

Purpose：Pull high the output signal of the specified index in the instruction

Grammar：int UDM_SetOutPutOn(unsigned int nOutIndex);

Describe：nOutIndex pulls up the output signal of the specified index separately (this information will be written to the UDM.BIN file)

If written as 1, it means pulling up OUT1 and outputting 5V

Return Value：The starting address of the instruction

Function name：UDM_SetOutPutOff(unsigned int nOutIndex);

Purpose：Pull high the output signal of the specified index in the instruction

Grammar：int UDM_SetOutPutOff(unsigned int nOutIndex);

Describe：nOutIndex pulls down the output signal of the specified index individually (this information will be written to the UDM.BIN file)

If written as 1, it means pulling down OUT1 and outputting 0V

Return Value：The starting address of the instruction

Function name：UDM_SetOutPutOn_GMC4(unsigned int nOutIndex);

Purpose：Pull high the output signal of the specified index in the instruction

Grammar：int UDM_SetOutPutOn(unsigned int nOutIndex);

Describe：nOutIndex pulls up the output signal of the specified index separately (this information will be

written to the UDM.BIN file)

If written as 1, it means pulling up OUT1 and outputting 5V

Return Value：The starting address of the instruction

Function name：UDM_SetOutPutOff_GMC4(unsigned int nOutIndex);

Purpose：Pull down the output signal of the specified index in the command

Grammar：int UDM_SetOutPutOff(unsigned int nOutIndex);

Describe：nOutIndex pulls down the output signal of the specified index individually (this information will be written to the UDM.BIN file)

If written as 1, it means pulling down OUT1 and outputting 0V

Return Value：The starting address of the instruction

Function name：UDM_SetAnalogValue；

Purpose：Set the voltage of analog VoutA and VoutB in the instruction

Grammar：UDM_SetAnalogValue(float VoutA,float VoutB)

VoutA，Set the analog value of VoutA pin

VoutB，Set the analog value of VoutB pin

Describe：Two analog voltages are set in the instruction. The voltages of these two signals are in the range of 0~10V. If the parameter is set to 0.5, the voltage output is 5V, if it is set to 0.8, the output is 8V, and if it is set to 1.0, the output is 10V.

Return Value：The starting address of the instruction;

Function name：UDM_SetOffset

Purpose：Set the overall offset of the marking graphic;

Grammar：UDM_SetOffset(float offsetX, float offsetY, float offsetZ);

offsetX X position offset；

offsetY Y position offset;

offsetZ Z position offset;

Describe：In the instruction, set the offset of all graphics in the marking file, in mm. After setting the offset, the starting point and end point of the marking are both offset points, that is, the motor zero position becomes the set offset position. Usually, the marking graphic is made at the origin of the coordinate system first, and then the offset is set to offset the marking graphic to the target position. This offset will be saved in the marking file. If the online offset setting function HM_SetOffset is called, the online setting will be overwritten.

Function name：The starting address of the instruction;

Function name：UDM_SetRotate

Purpose：Set the overall rotation of the marking graphic；

Grammar：UDM_SetRotate(float angle, float centryX, float centryY);

angle Rotation angle；

centryX X coordinate of the rotation center；

centryY Y coordinate of the rotation center；

Describe：Set the rotation angle of all graphics in the marking file in the command, the unit is °. Usually, the marking graphics are made at the origin of the coordinates first, and then the angle is set to rotate the marking graphics to the target position; usually this function needs to be used together with

UDM_SetOffset, first rotate at (0, 0), and then offset. This rotation amount will be saved in the marking file. If the online rotation setting function HM_SetRotates is called, the online setting will be overwritten.

Return Value：The starting address of the instruction；

Function name：UDM_FootTrigger

Purpose：Waiting for foot trigger signal input

Grammar：int UDM_FootTrigger(unsigned int nDelayTime,int nTriggerType);

Describe：When marking, after running to this instruction, the control card will wait for the IN0 input signal until there is a correct IN0 input signal from the outside before continuing to execute the instruction.

After receiving the IN0 signal, nDelayTime will delay execution for nDelayTime time, in ms

nTriggerType, trigger mode, 0 rising edge trigger, 1 level trigger

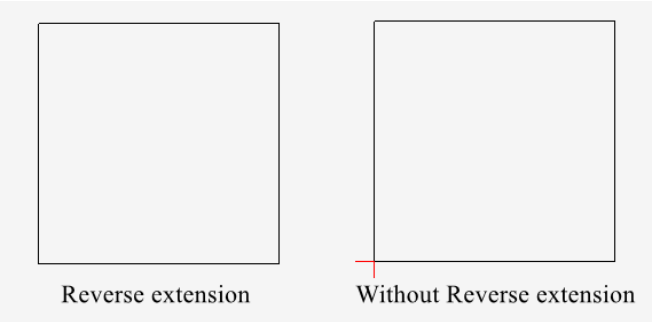
Return Value：The starting address of the instruction

Function name：UDM_SetJumpExtendLen

Purpose：Set the jump extension length to improve the craftsmanship at the starting and ending points of the graphics.

Grammar：int UDM_SetJumpExtendLen(float jumpExtendLen);

Describe：jumpExtendLen Extend the jump length. When this value is set, it is recommended to set the jump delay and marking delay to 0，because when this value is not 0, the galvanometer will jump to the starting point of the figure first, so that the galvanometer is in a constant speed state when emitting light. This function is similar to a simplified version of the SkyWriting function, only for the starting point and the end point. The schematic diagram is as follows, where the red area is the galvanometer jump track.



Return Value：The starting address of the instruction

Calling method：UDM_SetLayersPara() Then call the function

Note: When using this mode, it is forbidden to use it with UDM_SetSkyWritingMode, UDM_SkyWriting() and other functions at the same time. Only one of the three can be used. In addition, the jump delay, marking delay and turning delay need to be set to 0.

Function name：UDM_SetSkyWritingMode

Purpose：Whether to enable the SkyWriting function and set the parameters

Grammar：UDM_SetSkyWritingMode(int enable,int mode,float uniformLen,float accLen,float angleLimit);

When enable is 0, this function is disabled; when enable is 1, this function is enabled.

Mode: 0 uses linear transition and automatically sets acceleration and deceleration at the corners; 1 uses arc transition; 2 uses linear transition and the instruction does not plan the acceleration and deceleration area.

Uniform Len Length of uniform velocity area

accLen Acceleration/deceleration zone length

angleLimitAngle limit, if the angle is less than this, SkyWriting will be performed, if the angle is greater than this, it will not be processed directly

Describe： In some special industries, the graphics that need to be processed are completely in a uniform speed state, which can ensure the consistent spacing between laser points, better process effects, and avoid key points.

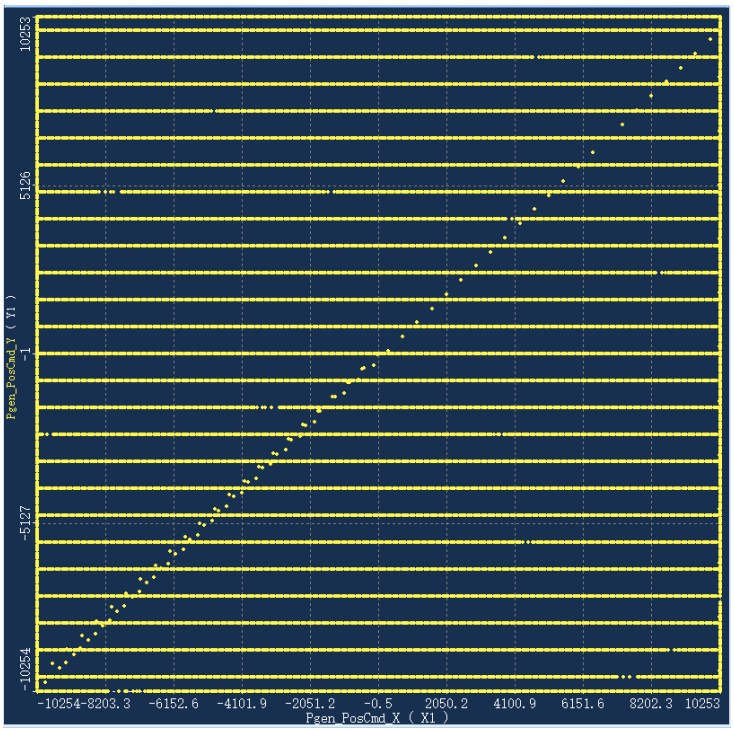
Calling method: Call after UDM_SetLayersPara() function

Note: When using this mode, it is forbidden to use it with UDM_SkyWriting(), UDM_SetJumpExtendLen() and other functions at the same time. Only one of the three can be used. In addition, the jump delay, marking delay, and turning delay need to be set to 0. Currently only effective for 2D marking

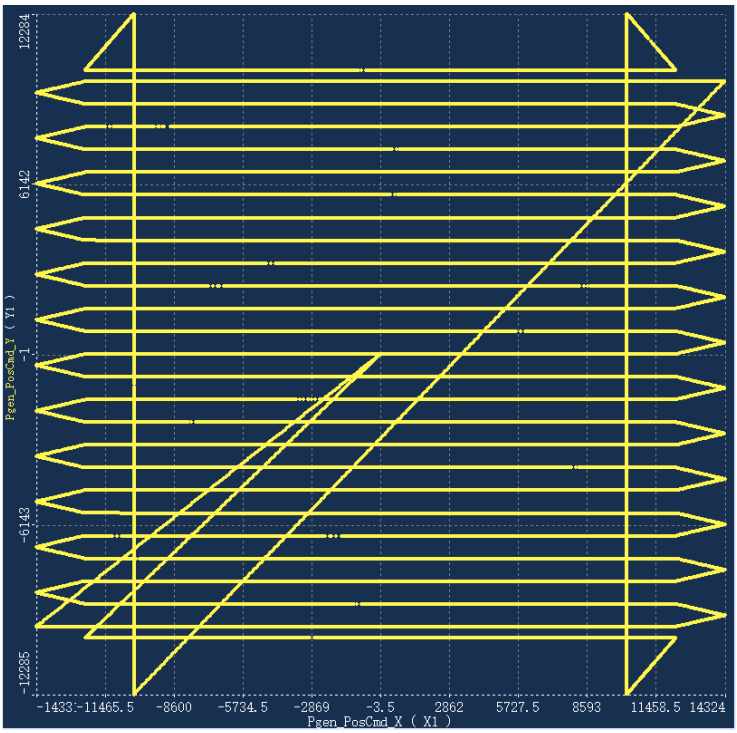
About the three modes：

Mode 0: Straight line transition with acceleration and deceleration

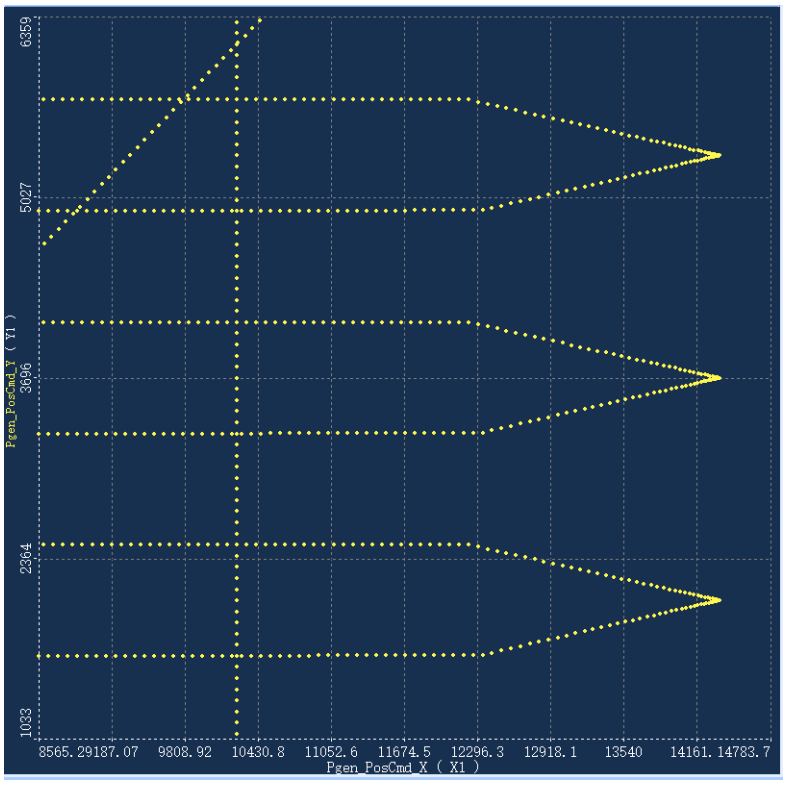
When this mode is not enabled, the graphics instructions are as follows:



The overall instructions when enabling mode 0 are as follows:

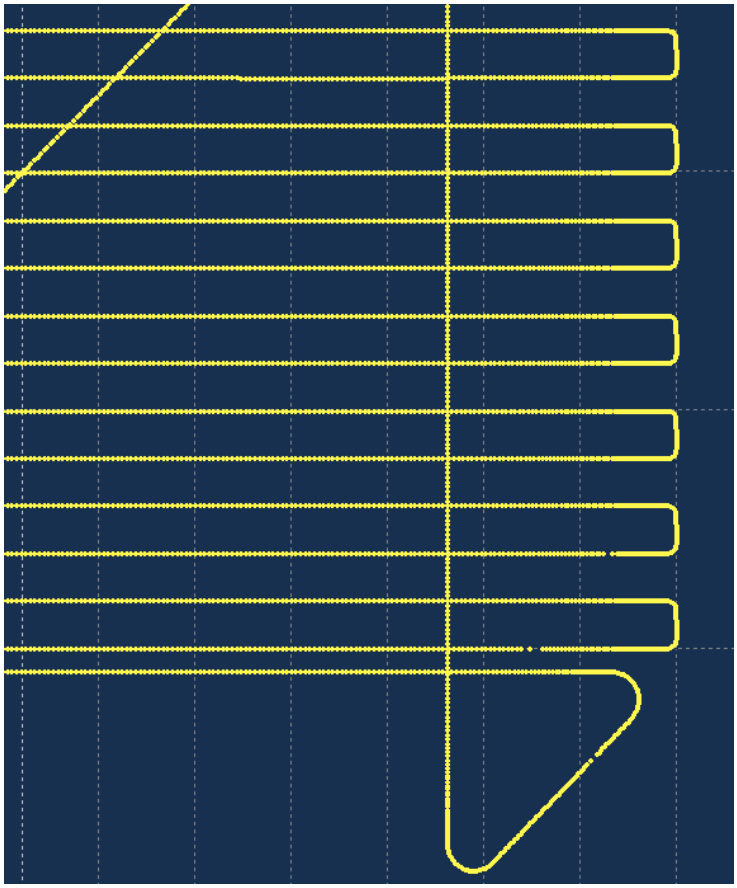


We select a certain position to zoom in locally, and we can see the density of the command points. The denser the points, the slower the planning speed, and the sparser the points, the faster the speed. From the figure below, we can see that the upper and lower line segments are accelerated and decelerated during the jump process. Acceleration and deceleration here can make the galvanometer motor run more stably and increase its service life, but the efficiency is relatively low, as shown in the figure below:

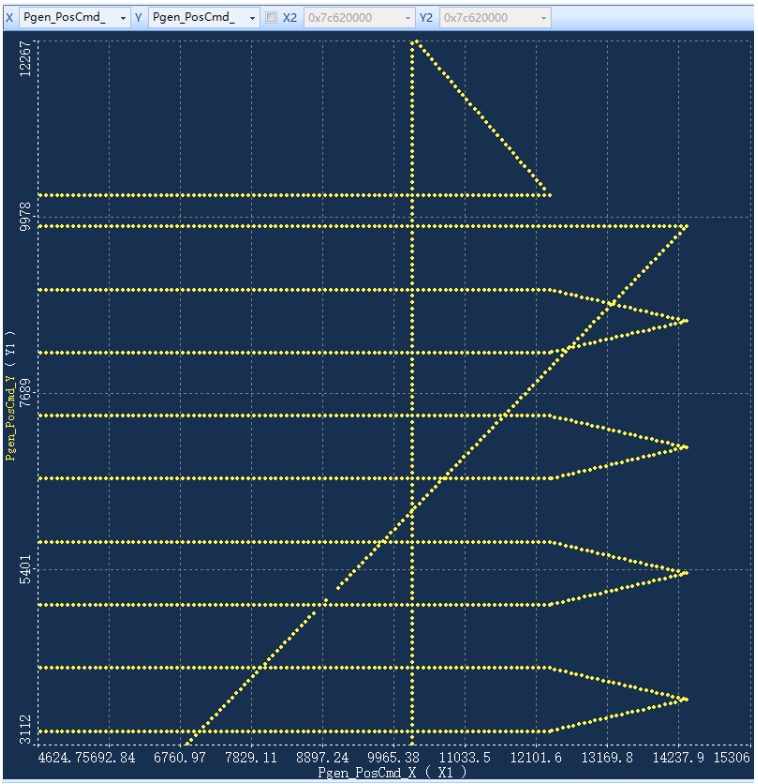


Mode 1 Arc transition zone acceleration and deceleration command effect is shown in the figure

below:

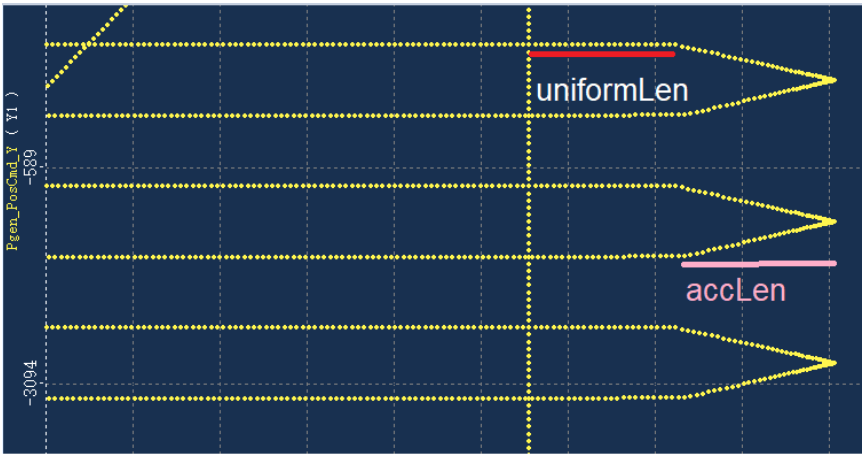


Mode 2: Straight-line transition without acceleration or deceleration. The command effect is shown in the figure below:

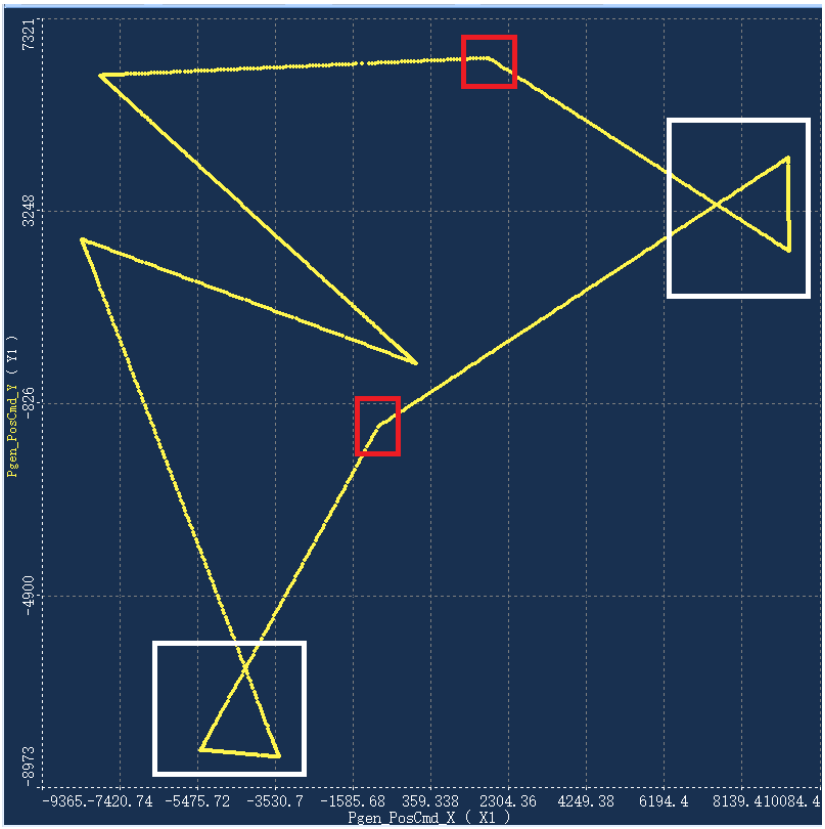


From the above figure, we can see that the density of points at the corners has not changed, which means that the command does not perform acceleration or deceleration. This mode is more efficient.

The explanation of uniformLen and accLen is as follows:



AngleLimit described as follows:



The red area represents an angle greater than angleLimit, and the white area represents an angle less than angleLimit

Function name： UDM_SetLayersPara

Purpose： Set different layer parameters

Grammar： int UDM_SetLayersPara(MarkParameter *layersParameter,int count);

Describe： In marking software, there is generally a concept of layers. Different layers have different marking parameters. This function can pass all the parameters of the layers to

the interface function at once. When writing the coordinates of the graphic track, you only need to pass in the layer index where the track is located.

layersParameter An array of layer parameters, also supports one layer

count，The number of layers must be greater than or equal to 1.

Return Value：The starting address of the instruction

Function name：UDM_AddPoint2D

Purpose：Add a point to the marking file and specify the layer index where this point is located.

Grammar：int UDM_AddPoint2D(structUdmPos pos, float time,int layerIndex);

Describe：pos Point coordinates location;

time The time of light emission at this point, in ms;

layerIndex，The layer index where this point is located.

Return Value：The starting address of the instruction

Function name：UDM_AddPolyline2D

Purpose：Add a continuous 2D polyline track to the marking file and specify the layer index where the track is located.

Grammar：int UDM_AddPolyline2D(structUdmPos *nPos, int nCount,int layerIndex);

Describe：nPos Pointer to the position array of the starting point, turning point and end point on the polyline;

nCount The number of points, the galvanometer jumps to the starting point of the polyline, and marks a continuous polyline along the given points;

layerIndex，The layer index of this track.

Return Value：The starting address of the instruction

Function name：UDM_AddPolyline3D

Purpose：Add a continuous 3D polyline track to the marking file and specify the layer index where the track is located.

Grammar：int UDM_AddPolyline3D(structUdmPos *nPos, int nCount,int layerIndex);

Describe：nPos Pointer to the position array of the starting point, turning point and end point on the polyline;

nCount The number of points, the galvanometer jumps to the starting point of the polyline, and marks a continuous polyline along the given points;

layerIndex，The layer index of this track.

Return Value：The starting address of the instruction

Function name：UDM_AddBreakAndCorPolyline3D

Purpose：Decompose a continuous 3D polyline track into more points, recalculate the z value through the 3D correction table, and specify the layer index of this track.

Grammar：int UDM_AddBreakAndCorPolyline3D(structUdmPos *nPos, int nCount,float p2pGap, int layerIndex)

Describe：nPos Pointer to the position array of the starting point, turning point and end point on the polyline;

nCount The number of points, the galvanometer jumps to the starting point of the polyline, and marks a continuous polyline along the given points;

p2pGap Divide this curve into two equal parts according to the p2pGap interval

layerIndex，The layer index of this track.

Return Value：The starting address of the instruction

Illustrate：This function is different from the UDM_AddPolyline3D function. The UDM_AddPolyline3D function directly writes the data into the UDM file without any processing. The UDM_AddBreakAndCorPolyline3D function goes through three logical processes: ① subdividing the original points, ② recalculating the Z value through the 3D correction parameters, and ③ calling the UDM_AddPolyline3D function again with the point coordinates processed in the previous two steps.

Function name：UDM_Set3dCorrectionPara

Purpose：Set the 3D correction table parameters.

Grammar：UDM_Set3dCorrectionPara(float baseFocal,double *paraK, int nCount)

Describe：baseFocal The focal length of the reference plane, that is, the length of the galvanometer from the focal point when xyz is at 0 during 3D marking;

paraK 3D Correction parameter coefficient value;

nCount，The number of parameter coefficients.

Return Value：The starting address of the instruction

Function name：UDM_GetZvalue

Purpose：Set the xyz value passing through this point, and obtain the value that needs to be sent to the Z-axis motor through the 3D correction table

Grammar：UDM_GetZvalue(float x,float y, float height);

Describe：xThe x coordinate of this point;

y is the y coordinate of this point;

height，The height of this point.

Return Value：The z value after 3D correction,

Risk Warning: Because the Z-axis motor stroke is between -4 and 4, if the return value of this value is outside this range, it is considered abnormal. Possible causes of abnormality: ① abnormal calibration parameters, ② abnormal point coordinates themselves are input

Note: When adding graphic tracks, you can call UDM_AddPoint2D, UDM_AddPolyline2D, UDM_AddPolyline3D and other functions multiple times. Each call represents adding a track to the marking file. When marking, it will be marked in the order of addition. Developers do not need to manage jump instructions. When moving from this track to the next track, the control card will internally determine whether to jump to turn off the light.

Function name：UDM_SetCloseLoop

Purpose：Whether to open the closed loop function (card and square head customization required)

Grammar：int UDM_SetCloseLoop(bool enable, int galvoType, int followErrorMax,int followErrorCount);

Describe：Enable enables the closed loop function when it is true, and disables it when it is false

galvoType Motor Type，0 Analog Scanhead or Scanhead with position feedback only; 1 digital Ultra Scanhead, 2 Extra Scanhead

followErrorMaxThe maximum error between the command and the actual feedback value, the default is 200 counts

followErrorCount，The cumulative number of times the maximum error between the command and the actual feedback value is exceeded, the default value is 1000.

When the difference between the command and the feedback value exceeds followErrorMax, it is

counted once. When the total number reaches followErrorCount, it is considered that there is a following error alarm, that is, the actual running trajectory of the motor is too different from the command trajectory sent.

If the followErrorMax and followErrorCount values are set too small, false alarms may occur. If they are set too large, alarms may become insensitive.

Hint：When using this function, please make sure that the galvanometer has a closed loop function and can feedback alarm functions such as temperature and current. It can be used in conjunction with HM_ClearCloseLoopAlarm(); HM_GetGalvoStatusInfo() functions. It only supports the XY2-100 protocol.

Return Value：The starting address of the instruction

Chapter 5 DEMO Software Usage Guide

In the DEMO software, the entire marking process is written, including finding IP, connecting equipment, setting marking parameters, laser parameters and marking graphics, etc. Users can quickly develop based on the DEMO software. The previous documents can be used as a guiding reference. The main function applications are reflected in the DEMO. If you understand the DEMO process, you can basically carry out customized development of your own project. DEMO provides two development languages C++ and C#, which are explained below. Because the communication between the software and the control card is based on the window handle for signal interaction, it must be developed on the MFC or Form form, otherwise data interaction cannot be carried out and the marking operation cannot be controlled. **In addition, the following three files must be in the executable file directory, none of which can be missing: HM_Comm.dll, HM_HashuScan.dll, and system.ini.**

The entire marking process is as follows:

1. Connect the device;
2. Select the marking protocol and the graphics to be marked;
3. Click the start marking button;
4. After starting marking, the marking parameters and laser parameters will be assigned, and the UDM file will be generated according to the selected graphics;
5. Download the UDM marking file to the control card;
6. After the marking file is successfully downloaded, enter the download completion message function;
7. Call the start marking function HM_StartMark() in the download completion message function;
8. The control card starts marking, and the laser emits laser at the same time;
9. Marking is completed, and the marking completion message function is entered;
10. The entire marking process is completed.

5.1 C++ MFC version DEMO

5.1.1 Window initialization

In the form initialization function OnInitDialog(), the control card initialization function needs to be

called, that is, HM_InitBoard(this->m_hWnd); after calling this function, the communication relationship can be established. When the software is opened for initialization, it does not matter whether the control card is turned on or not. When the control card is powered on and the computer IP is set correctly, the program will automatically enter the message callback function to notify the software that the control card has been found and can be connected.

5.1.2 Message callback function

The following message functions need to be added to the BEGIN_MESSAGE_MAP (CST_MarkDemo_CPPDlg, CDialogEx) message function to obtain the status information of the control card at any time, as follows:

```
// Search for IP Address Device status message map
ON_MESSAGE(HM_MSG_DeviceStatusUpdate,OnMsgDeviceEhco)
// Progress feedback during marking
ON_MESSAGE(HM_MSG_ExecProcess, ExecProcess)
//Download completed download finish message map
ON_MESSAGE(HM_MSG_StreamEnd,OnMsgUDMDownloadEnd)
// Marking completed     executed finished message map
ON_MESSAGE(HM_MSG_UDMHalt, OnMsgUDMRunHalt)
```

When the control card is powered on and the Ready light is on, and the IP address is correctly connected to the network cable, after the software is started, it will first enter the OnMsgDeviceEhco (WPARAM wParam, LPARAM lParam) function, in which you can get the IP index (lParam) and IP value (wParam) returned from the control card. Call the HM_GetConnectStatus function to get the current connection status. When you click the connect button, the connection function is called. When the connection is successful, this message will be entered again, which means the connection is successful, and you can send the UDM marking file. Note that the IP address and index are returned by the control card, and can be defaulted to unknown in advance. **Note: When there are multiple control cards, the IP address is unique and unchanged, but the IP index is related to the power-on sequence of the root control card. Therefore, when you need to control a specific control card, you must first find the corresponding IP index based on the IP address of the card. The function HM_GetIndexByIpAddr can obtain the IP index under the specified IP address.**

```
//*****************************************************//
// The HM_MSG_DeviceStatusUpdate Msg Process Function
// wParamv:  Device IP Address ,such as 0x e3 22 12 ac means 172.18.34.227
// lParam:   Device IpIndex , begin 0。
//*****************************************************//
LRESULT CST_MarkDemo_CPPDlg::OnMsgDeviceEhco(WPARAM wParam, LPARAM lParam)
{
    CString strIP;
    if(HM_DEV_Connect == HM_GetConnectStatus(lParam))
    {
        //The device is connected successfully
        buttomConnect.EnableWindow(FALSE);
        ipIndex = lParam;
    }
    else if(HM_DEV_Ready == HM_GetConnectStatus(lParam))
    {
        //The device is in standby state, that is, the device IP address has been found and can be connected
        ipIndex = lParam;
        BYTE* ipArr = (BYTE*)&wParam;
        strIP.Format(_T("%d.%d.%d.%d"),ipArr[0],ipArr[1],ipArr[2],ipArr[3]);
        comboBoxIpList.AddString(strIP);
        comboBoxIpList.SetCurSel(ipIndex);
        buttomConnect.EnableWindow(TRUE);
    }
    else if (HM_DEV_NotAvailable == HM_GetConnectStatus(lParam))
    {
        //The controller card is powered off or the network cable is not connected
        buttomConnect.EnableWindow(TRUE);
        comboBoxIpList.ResetContent();
    }
    return 0;
}
```

Figure 4.2 IP message search function diagram

After calling the HM_DownloadMarkFile(ipIndex,"D:\\UDM.bin",this->m_hWnd); function, the marking file in the specified path will be downloaded and sent to the control card. At this time, it is forbidden to call the HM_StartMark() function immediately, because if the marking file occupies a large amount of memory, it will take a certain amount of time to send it. Calling the HM_StartMark() function before sending is completed will cause marking abnormalities. This control card uses 100M Ethernet. In theory, it can send about 12.5M data per second, but there will be a little deviation in reality.

When the UDM.BIN marking download is completed, the OnMsgUDMDownloadEnd(WPARAM wParam, LPARAM lParam) function will be entered. When entering this function, it means that the marking file download is completed. The next step is to call the marking start function HM_StartMark() in this function; that is, after the download is completed, the marking starts. **Note: You must wait until the download is complete before marking, otherwise it will cause marking abnormalities.**

```
//After this function is displayed, the UDM.BIN file is downloaded and can be marked
//download finish,next need start mark
LRESULT CST_MarkDemo_CPPDlg::OnMsgUDMDownloadEnd(WPARAM wParam, LPARAM lParam)
{
    SetTimer(1,10,NULL);//When marking starts, the timer is started to query the marking progress
    HM_StartMark(lParam);
    return 0;
}
```

Figure 4.3 Marking file download completed

During the marking process, the control card will return a marking progress message, and then enter the ExecProcess (WPARAM wParam, LPARAM lParam) message function. This message must be used together with the HM_ExecuteProgress (ipIndex) function. Calling the HM_ExecuteProgress

(ipIndex) function once will enter this message once. For this reason, if real-time control is required, a timer needs to be set. Start the timer when starting marking, and continuously call the HM_ExecuteProgress function. After the marking is completed, the timer needs to be closed.

```
void CST_MarkDemo_CPPDlg::OnTimer(UINT_PTR nIDEvent)
{
    //Set a timer to check the marking progress continuously when the marking starts
    switch(nIDEvent)
    {
    case 1:
        HM_ExecuteProgress(ipIndex);
        break;
    default:
        break;
    }

    CDialog::OnTimer(nIDEvent);
}
```

Figure 4.4 Query marking progress bar

After the final marking is completed, the OnMsgUDMRunHalt (WPARAM wParam, LPARAM lParam) function is entered, indicating that the marking is completed.

```
//This function is entered when the marking is complete.
//mark over
LRESULT CST_MarkDemo_CPPDlg::OnMsgUDMRunHalt(WPARAM wParam, LPARAM lParam)
{
    KillTimer(1);//Turn off the marking progress timer when marking is complete
    MessageBox(_T("mark over"));//mark over

    return 0;
}
```

Figure 4.5 Marking completed

If you want to achieve cyclic marking, you only need to call the start marking function again after the marking is completed. There is no need to download the UDM marking file repeatedly. As long as the control card is powered on, the data will not be lost. After calling the HM_DownloadStart_DDR_UDM function again, the data will be overwritten. The cyclic marking is as shown below

```
//This function is entered when the marking is complete.
//mark over
LRESULT CST_MarkDemo_CPPDlg::OnMsgUDMRunHalt(WPARAM wParam, LPARAM lParam)
{
    KillTimer(1);//Turn off the marking progress timer when marking is complete
    MessageBox(_T("mark over"));//mark over

    bool bRecycleMark = false;//Whether to cycle marking,recycle mark
    if (bRecycleMark)
    {
        SetTimer(1,10,NULL);
        HM_StartMark(lParam);
    }
    return 0;
}
```

Figure 4.6 Circular marking

5.1.3 Instructions for generating marking file UDM.BIN

1. First set the marking parameters and laser parameters. Since most applications require different layer parameters to be set according to different graphics, the user can set an array to save the parameters. Two layers are set here, and the number of layers set by the user is not limited.

```
MarkParameter *CST_MarkDemo_CPPDlg::getMarkParameter()
{
    UpdateData(TRUE);
    MarkParameter* para = new MarkParameter[2];
    for (int i = 0;i < 2;i++)
    {
        para[i].MarkSpeed = markSpeed;//markSpeed mm/s
        para[i].JumpSpeed = jumpSpeed;//jumpSpeed mm/s
        para[i].PolygonDelay = polygonDelay;
        para[i].JumpDelay = jumpDelay;
        para[i].MarkDelay = markDelay;
        para[i].MarkCount = markCount;

        para[i].LaserOnDelay = laseronDelay;//laseronDelay
        para[i].LaserOffDelay = laserOffDelay;//laserOffDelay
        para[i].DutyCycle = dutyCycle;//Light duty cycle, (0~1)
        para[i].Frequency = frequency;//Outgoing light frequency KHZ
        para[i].StandbyFrequency = frequency;//No light Q frequency(KHZ);
        para[i].StandbyDutyCycle = dutyCycle;//No light Q duty cycle;
        para[i].LaserPower = power;//Laser energy percentage(0~100), 50 represents 50%
        para[i].AnalogMode = 0;
        if (comboxLaser.GetCurSel() == 1)
        {
            para[i].AnalogMode = 1;//1 represents the use of analog output, at this time you can control analog output (0~10V)
            para[i].Waveform = 0;
        }
        else if (comboxLaser.GetCurSel() == 2)
        {
            para[i].StandbyFrequency = 0;//No light Q frequency(KHZ);
            para[i].StandbyDutyCycle = 0;//No light Q duty cycle;
        }
    }
    return para;
}
```

Figure 4.7 Setting parameters

2,To start creating a UDM file, you need to set the marking protocol in advance. The commonly used one is the XY2-100 open-loop protocol, and then set the layer array parameters, the number of cycle markings, etc.

```
void CST_MarkDemo_CPPDlg::creatUdmBin()//Used to generate the UDM.bin file
{
    UDM_NewFile();
    UDM_Main();
    UDM_SetProtocol(comboxProtocol.GetCurSel(),0);
    UDM_SetLayersPara(getMarkParameter(),2);
    int startAddress = UDM_RepeatStart(markCount);//Set times
```

Figure 4.8 Start creating UDM file

3, Start adding the coordinates of the marking graphics points. In the laser marking industry, all graphics are vector graphics, that is, they are all determined by point coordinates. Generally, any graphics can be composed by using the straight line UDM_AddPolyline2D(). The code shown in the figure below adds three graphics: straight line, circle, and rectangle.

```
if (((CButton*)GetDlgItem(IDC_CHECK_LINE))->GetCheck() == BST_CHECKED)
{
    structUdmPos polyline2d[2] = {-30,30,0,0,30,-30, 0,0,};
    UDM_AddPolyline2D(polyline2d,2,0);
}
if (((CButton*)GetDlgItem(IDC_CHECK_CIRCLE))->GetCheck() == BST_CHECKED)
{
    structUdmPos polyline2d[361];
    float radius = 25;
    double dAngle = 2 * 3.1415 / 360;
    for (int i = 0; i < 361;i++)
    {
        polyline2d[i].x = radius * cos(dAngle * i);
        polyline2d[i].y = radius * sin(dAngle * i);
        polyline2d[i].z = 0;
        polyline2d[i].a = 0;
    }
    UDM_AddPolyline2D(polyline2d,361,0);
}
if (((CButton*)GetDlgItem(IDC_CHECK_RECTANGLE))->GetCheck() == BST_CHECKED)
{
    structUdmPos polyline2d[5] = {-25,-25,0,0,+25,-25, 0,0,+25,+25, 0,0,-25,+25, 0,0,-25,-25, 0,0};
    UDM_AddPolyline2D(polyline2d,5,0);
}
```

Figure 4.9 Adding graphic point coordinates

1. After setting the graphics, enter the final stage and call the following function to generate a complete UDM file

```
UDM_Jump(0, 0, 0);
UDM_EndMain();
UDM_SaveToFile("D:\\UDM.bin");//Generate and save the marked udm.bin file
```

Figure 4.9 Save UDM file

UDM_Jump(0,0,0) means the galvanometer will return to zero after marking. If this line of code is not added, it means the galvanometer will stop at the last point after marking. Using UDM_Jump(20,30,0) means the galvanometer will stop at (20,30) after marking.

5.1.4 Instructions for setting the number of markings

When marking, if a certain graphic needs to be marked multiple times, you can use the following method to handle it.

int startAddress = UDM_RepeatStart(2);

UDM_AddPolyline2D(polyline1,5,0);

UDM_RepeatEnd(startAddress);

startAddress = UDM_RepeatStart(3);

UDM_AddPolyline2D(polyline2,5,0);

UDM_RepeatEnd(startAddress);

startAddress = UDM_RepeatStart(999);

UDM_AddPolyline2D(polyline3,5,0);

UDM_RepeatEnd(startAddress);

In the above code example, polyline1 represents marking 2 times, polyline2 represents marking 3 times, and polyline3 represents marking 999 times. In this way, any number of markings can be achieved. If you only mark once, it is recommended not to call UDM_RepeatStart and UDM_RepeatEnd, which can

save the memory space of the UDM.BIN file. If the number of markings is filled in 0, it will also mark once. If you want to achieve the function of marking 0 times, do not add the graphic data to the code.

Note: Nested loops are prohibited in loop marking. The following method is not allowed:

```
int startAddress = UDM_RepeatStart(2);
    startAddress1 = UDM_RepeatStart(2);
    UDM_AddPolyline2D(polyline1,5,0);
    UDM_RepeatEnd(startAddress1);
UDM_RepeatEnd(startAddress);
```

5.1.5 Marking settings for different layers

In the actual marking process, for the needs of laser technology, graphics are generally marked in layers, that is, different layers of graphics use different marking parameters or laser parameters. You can use the following methods to mark in layers

MarkParameter* para = new MarkParameter[layerCount];

UDM_SetLayersPara(para , layerCount);

This will create layerCount number of layers, and the parameters in each layer can be set differently. When calling the command to add a graphic track, you only need to pass in different layer indexes:

UDM_AddPolyline2D(polyline1,5,0);

UDM_AddPolyline2D(polyline2,5,1);

The above code indicates that the polyline1 track is marked with the parameter of layer index 0, and the polyline2 graphic track is marked with the parameter of layer index 1.

5.1.6 IO set up

When cooperating with automation, IO operations are often required. There are two modes for controlling IO. One is to read input signals online and set output signals online. The other is to write IO information into the UDM.BIN file for dynamic control during the marking process.

① Online IO control description:

HM_GetInput_GMC2() This function reads the input status of the control card IN0~IN14 online. The returned value needs to be converted into binary. The status is judged in order from low to high. 0 represents non-conduction and 1 represents conduction. For example, if the decimal value is 89 and converted to binary as 1011001, the corresponding input status is IN14~IN7 non-conduction, IN6 conduction, IN5 non-conduction, IN4 conduction, IN3 conduction, IN2 non-conduction, IN1 non-conduction, and IN0 conduction. At this time, the software can perform different operations according to these different input states, such as selecting different marking graphics, etc.

HM_SetOutputOn_GMC2 and HM_SetOutputOff_GMC2, calling this function means controlling the output status of the specified pin (OUT0~OUT15) online, ON means the level is pulled up to 5V output, OFF means the level is pulled down to 0V output.

②UDM control IO description

UDM_SetInput, when creating a UDM file, call this function at the specified location, then when the marking runs to this step, it will be in a waiting state until the correct input signal is given externally, and only IN0~IN7 is currently supported.

UDM_SetOutPutOn and UDM_SetOutPutOff functions mean that during the marking process, when it is executed here, the specified output pin will be controlled to be pulled up or down.

③Code example:

UDM_SetInput(0)

UDM_AddPolyline2D(line,2,0);

UDM_SetOutPutOn(0);

UDM_SetInput(1)

UDM_AddPolyline2D(circle,360,0);

UDM_SetOutPutOff(0);

UDM_SetOutPutOn(1);

In the above code, UDM_SetInput(0) is set at the beginning; it means that an external trigger IN0 is needed at this time, otherwise the program will be stuck here and wait for the input signal. When the external PLC gives the input signal, it starts to draw a straight line. When the straight line is completed, the OUT0 pin will be turned to a high level. At this time, the execution continues, and the PLC needs to trigger the IN1 signal. When the control card receives the IN1 signal from the PLC, it starts to draw a circle. After the circle is completed, OUT0 will be set to a low level, and then OUT1 will be turned to a high level.

5.1.7 Red light preview

Some lasers have the function of red light preview. When clicking on the red light preview, call the HM_SetGuidLaser(enable) function. When true is passed in, the red light pin P22 of the J11 port immediately becomes a high-level 5V output. If marking is performed directly at this time, PIN19 of the J11 port continues to maintain a low-level output; if you want to restore normal light marking, you need to set it to false.

In addition, for some MOPA-type lasers, their red light pin and pulse width enable pin are both P22, which results in different functions sharing one pin, and sometimes there will be conflicts. For example, after setting the red light preview HM_SetGuidLaser(true) (P22 becomes a high level), the pulse width enable mode is turned on (PulseWidthMode = 1 in the parameter structure). At this time, you need to pull up P22 first, and then pull down P22 to realize the function of changing the pulse width, but pulling down P22 will turn off the red light. Therefore, we have made a restriction at the application level, that is, the pulse width is not allowed to be changed during the red light preview. That is, when using a MOPA type laser, when generating the UDM.BIN file, you need to inform the underlying layer that this is a red light preview, and you need to shield the function of changing the pulse width. Before setting the marking parameters, call

UDM_SetGuidLaser(true) The function is as follows:

UDM_SetProtocol(comboxProtocol.GetCurSel(),0);

UDM_SetGuidLaser(true);// Turn on red light

UDM_SetLayersPara(getMarkParameter(),2);

Of course, it also needs to be used together with the online red light setting function HM_SetGuidLaser().

5.2 C# Form Version DEMO

Because the further development library is developed in C++, the generated DLL cannot be directly called by C# and needs to be converted. Therefore, there are two more class files in the C# version, namely HM_HashuScanDLL.cs and HM_UDM_DLL.cs. Their functions correspond to the two header

files HM_HashuScan.h and HM_HashuUDM.h in C++, and their usage is the same. In the C# version, the message function is written differently from the C++MFC, but the logic is the same. This chapter mainly describes the usage of the message callback function. Others such as: marking file UDM generation, marking times setting, different layer settings, and IO settings will not be repeated in this section. Please refer to the previous section for C++ version settings.

5.1.1 Window Initialization

In the form initialization function, the control card initialization function needs to be called, that is, HM_InitComm()

HM_MSG_DeviceStatusUpdate was mentioned above. To set the status message, this function is used to establish a connection with the control card's IP.

```
public Form1()
{
    try
    {
        //System.Threading.Thread.CurrentThread.CurrentUICulture = new System.Globalization.CultureInfo("en-US");//english language
        InitializeComponent();
        comboBox_LaserType.SelectedIndex = 0;
        comboBox_Treaty.SelectedIndex = 1;
        //Control card initialization
        HM_HashuScanDLL.HM_InitBoard(this.Handle);
    }
    catch (System.Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

Figure 4.13 C# window initialization

5.1.2 Message callback function

When the control card is powered on and the Ready light is on, and the IP address and the network cable are correctly connected, after the software is started, it will first enter the DefWndProc (ref Message m) function, in which you can get the IP index (lParam) and IP value (wParam) returned from the control card. Call the HM_GetDeviceStatus function to get the current connection status. When you click the connect button, the connection function is called. When the connection is successful, this message will be entered again, which means the connection is successful, and you can send the UDM marking file. Note: The IP address and index are returned by the control card, and can be defaulted to unknown in advance.

```
//Window message queue function to get various status information returned from the card
protected override void DefWndProc(ref Message m)
{
    switch (m.Msg)
    {
        case clsEnumMessageDef.HM_MSG_StreamProgress: //File download progress bar, may or may not be displayed
            {
                progressBar1.Value = (int)m.WParam;
                break;
            }
        case clsEnumMessageDef.HM_MSG_DeviceStatusUpdate://The device IP address is connected or disconnected
            {
                搜索设备IP相关内容
            }
            break;
        case clsEnumMessageDef.HM_MSG_StreamEnd://The marking file UDM.BIN is downloaded
            {
                HM_HashuScanDLL.HM_StartMark((int)m.LParam); //Click whichever card is downloaded
                timer_ExecProcess.Start();//The marking progress timer is displayed
            }
            break;
        case clsEnumMessageDef.HM_MSG_MarkOver://mark over
            {
                Console.WriteLine("mark over");
                timer_ExecProcess.Stop();//Stop querying the marking progress timer
                progressBar1.Value = 100;
            }
            break;
        case clsEnumMessageDef.HM_MSG_QueryExecProcess:
            {
                //The progress bar is updated during marking
                progressBar1.Value = (int)m.WParam;
            }
            break;
        default:
            break;
    }
    base.DefWndProc(ref m);
}
```

Figure 4.14 IP message search function diagram

After calling HM_HashuScanDLL.HM_DownloadStart_DDR_UDM(strFilePath, 0, this.Handle, 0) function, it will enter clsEnumMessageDef.HM_MSG_DeviceStatusUpdate, where wParam represents the download progress percentage, with a value of 0 to 100. Generally speaking, the UDM marking file transfer speed is very fast, so this message function is dispensable and does not matter if it is not displayed.

```
protected override void DefWndProc(ref Message m)
{
    switch (m.Msg)
    {
        case clsEnumMessageDef.HM_MSG_StreamProgress: //File download progress bar, may or may not be displayed
            {
                progressBar1.Value = (int)m.WParam;
                break;
            }
```

Figure 4.15 Download marking file progress bar

When UDM.bin is downloaded, the download completion message function will be entered. When entering this function, it means that the marking file download is complete. The next step is to call the marking start function HM_MarkStart() in this function; that is, marking can only start after the download is complete**. Note: You must wait until the download is complete before marking, otherwise it will cause marking abnormalities.**

```
case clsEnumMessageDef.HM_MSG_StreamEnd://The marking file UDM.BIN is downloaded
    {
        HM_HashuScanDLL.HM_StartMark((int)m.LParam); //Click whichever card is downloaded
        timer_ExecProcess.Start();//The marking progress timer is displayed
    }
    break;
```

Figure 4.16 Marking file download completed

During the marking process, the control card will return a marking progress message, and then enter the clsEnumMessageDef.HM_MSG_QueryExecProcess message function. This message must be used together with the HM_ExecProcess(ipIndex) function. This message will be entered once the HM_ExecProcess(ipIndex) function is called. For this reason, if real-time control is required, a timer needs to be set. Start the timer when marking starts, and continuously call the HM_ExecProcess function. After marking is completed, the timer needs to be closed.

```
private void timer_ExecProcess_Tick(object sender, EventArgs e)
{
    HM_HashuScanDLL.HM_ExecuteProgress(currentIpIndex);//Get the marking progress
}
```

```
case clsEnumMessageDef.HM_MSG_QueryExecProcess:
    {
        //The progress bar is updated during marking
        progressBar1.Value = (int)m.WParam;
    }
    break;
```

Figure 4.17 Query marking progress bar

After the final marking is completed, the clsEnumMessageDef.HM_MSG_MarkOver function is entered, indicating that the marking is completed.

```
case clsEnumMessageDef.HM_MSG_MarkOver://mark over
    {
        Console.WriteLine("mark over");
        timer_ExecProcess.Stop();//Stop querying the marking progress timer
        progressBar1.Value = 100;
    }
    break;
```

Figure 4.18 Marking completed

If you want to achieve cyclic marking, you only need to call the start marking function again after the marking is completed. There is no need to download the UDM marking file repeatedly. As long as the control card is powered on, the data will not be lost. After calling the HM_DownloadStart_DDR_UDM function again, the data will be overwritten. The cyclic marking is as shown below

```
case clsEnumMessageDef.HM_MSG_MarkOver://mark over
    {
        Console.WriteLine("mark over");
        timer_ExecProcess.Stop();//Stop querying the marking progress timer

        bool bRecycleMark = true;
        if (bRecycleMark)
        {
            timer_ExecProcess.Start();
            HM_HashuScanDLL.HM_StartMark((int)m.LParam);
        }
    }
    break;
```

Figure 4.19 Cycle marking

5.3 Final instructions for DEMO

When users use further development, the process should be carried out as much as possible according to the DEMO, so as to avoid unknown exceptions. The process can be simplified into three steps: ① Connect and enter the connection OK message; ② Download the marking file and enter the download completion message after success; ③ After the file is successfully downloaded, start calling the start marking function.

Most of the user's energy should be on how to save the UDM marking file, that is, how to use point coordinates to combine the desired marking graphics.

Because the control card comes with high-precision calibration software, the calibration table has been downloaded to the control card in that software, and it is still effective after power failure and restart, so there is no need for users to load the calibration table again, that is, the software of further developers does not need to operate the calibration table.

Chapter 6 Laser Parameters

6.1 Introduction to Common Lasers

The following is based on my personal experience in using lasers. There may be some errors. It is only for reference. Don't believe it. Generally speaking, lasers are distinguished according to their wavelength. There are four common laser waveforms: ① 1024nm wavelength laser, also called fiber laser; ② 355nm wavelength laser, generally called ultraviolet laser; ③ 532nm laser, generally called green laser; ④ 10240nm laser, also called CO2 laser；

6.1.1 Fiber Laser

At present, fiber lasers are widely used in the market. Common brands include IPG, SPI, JPT, Raycus, and Han's MP lasers. We will not mention the performance for now. They can be divided into two categories based on the energy control method; The first category is IPG, JPT, Raycus, and Han's MP lasers. The energy of the lasers of these manufacturers is controlled by digital quantity, and 8 pins are controlled together, that is, when all 8 pins are high, it represents 100% energy, and when all are low, it represents 0% energy. We can understand that the 8 pins are a binary number, binary 11111111, decimal 255, which is also 100% energy. Binary 01111111, decimal 127, which is about 50% energy. 0 represents low level, and 1 represents high level. It can also be understood as follows: set the TTL signal of PIN1~8, and set the current of the pump laser diode, that is, the output power of the laser, through the combination of TTL signals. Through PIN1~8, you can set the code in the range of 0~255, corresponding to 0%~100% power output (the actual optical power output may not be linearly related to these settings)
Among them, the Han's MP laser and JPT laser can set the pulse width value (unit ns, which is different from the Q pulse width). Under different pulse widths, the same frequency power will produce different energies. It is generally needed when adjusting the process. When PulseWidthMode = 1 in the MarkParameter parameter structure, the function of changing the pulse width is enabled. PulseWidth represents the specific pulse width value in ns. The following figure is a screenshot of the pulse width in

the JPT laser description:

M7 Laser Power Reduction Frequency Value

YDFLP-80-M7-L1-R	
Pulse width(ns)	Down power frequency(kHz)
1	cw
2	1900
4	1300
6	1000
8	600
12	400
20	280
30	200
45	160
60	130
80	115
100	110
150	55
200	50
250	45
350	40
500	40

The second type is SPI type lasers. The characteristic of this type of laser is that its energy control is controlled by analog quantity, and 0~10V represents 0%~100% energy. **(When 0~10V control is required, SPI lasers can be used)**. It can be set by AnalogMode=1 in the MarkParameter structure. When AnalogMode=1, the energy setting value in the structure represents the size of the analog quantity. At this time, the energy percentage will be converted into a voltage value. When 10% energy is output, 1V is output, when 50% energy is output, 5V is output, and when 100% energy is output, 10V is output. When AnalogMode=0, the analog quantity function is turned off, and its energy is controlled by the 8 pins mentioned above. .

There is a concept called waveform on SPI lasers, which is equivalent to adjusting the pulse width. SPI lasers define 64 waveforms, 0~63, and Waveform represents the waveform number (0~63) used. For details, please refer to the SPI laser manual.

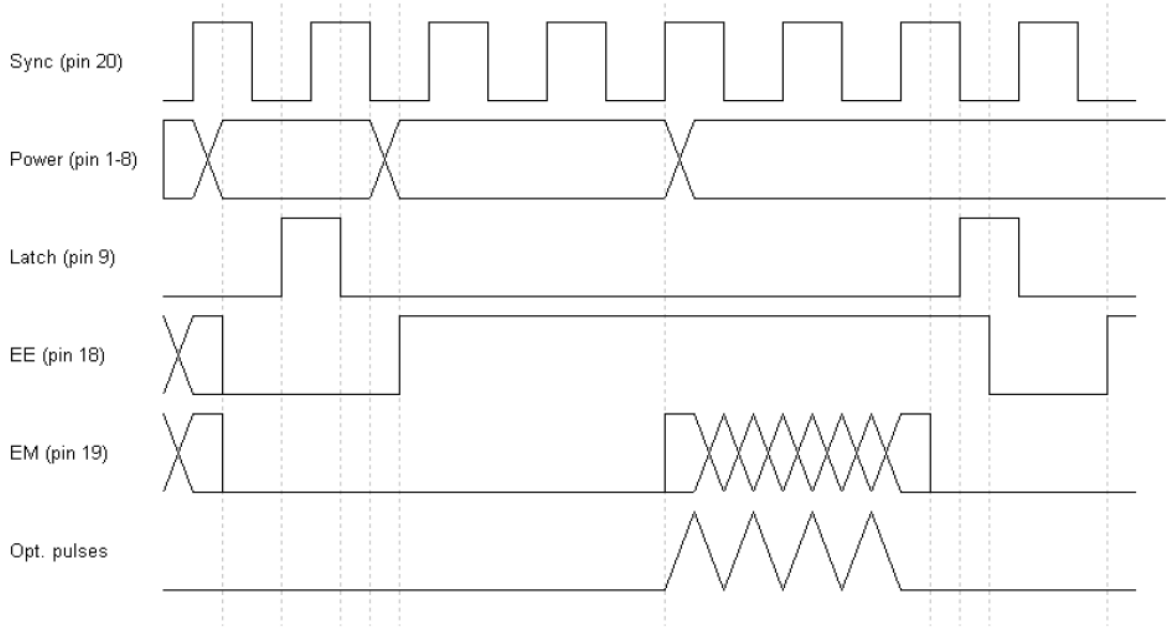
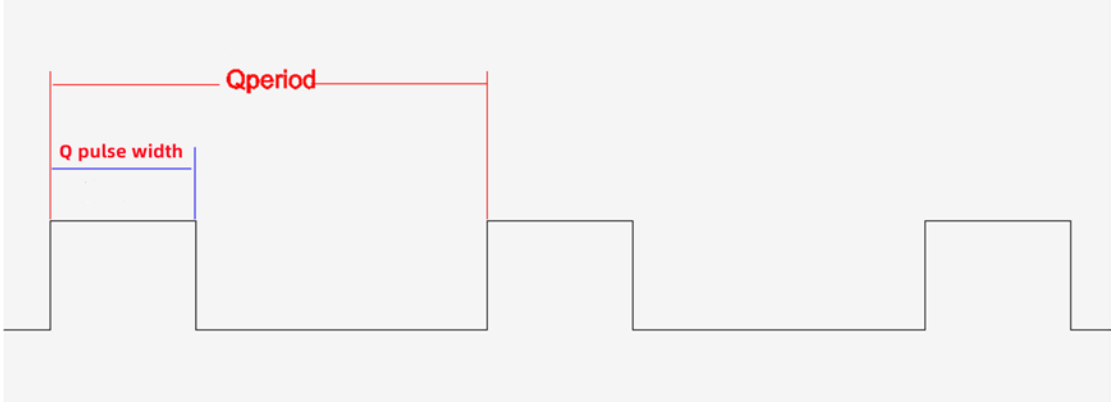
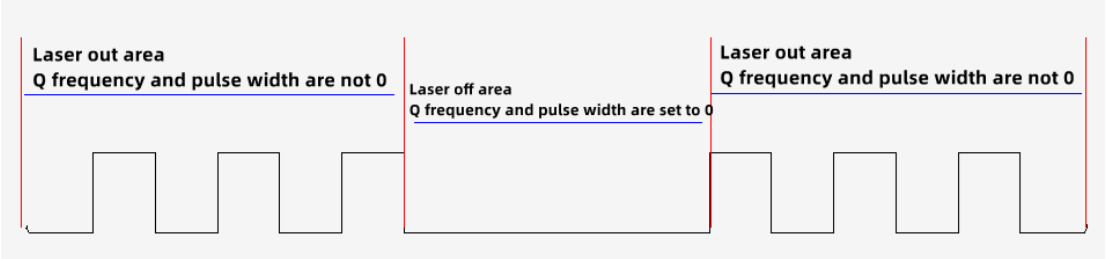


Figure 5.1 Laser light emission timing diagram

6.1.2 UV, CO2, and green lasers

Compared with fiber lasers, these three types of lasers are relatively simple to control. Generally, only one Q frequency signal is needed to control the switching light, and its energy is determined by the duty cycle **(duty cycle = pulse width / cycle)**. When encountering this type of laser, you need to set the non-light Q frequency and non-light Q pulse width to 0, otherwise the laser will continue to emit light and cannot be turned off. **CO2 lasers can burn the skin, so please pay attention to safety.**





6.1.3 Description of laser frequency

For a fiber laser, to control its light emission, more than a dozen signals are required, and light can be emitted only when all signals are met. Therefore, its non-light emitting Q frequency and non-light emitting duty cycle do not need to be set to 0. Try to make the non-light emitting Q frequency equal to the light emitting Q frequency, and the non-light emitting duty cycle equal to the light emitting duty cycle.

Note: If you forcefully set the non-emitting duty cycle to 0, for some lasers with poor performance, the response will be slow, which will cause abnormal starting frequency, laser point spacing and point energy. It is recommended to set them to the same to avoid this phenomenon. In addition, the duty cycle is best to be 0.5, that is, the Q pulse width is half of the Q period.

Lasers like CO2, ultraviolet, and green light only need a Q frequency and Q pulse width signal, so the non-light Q frequency and non-light duty cycle must be set to 0, otherwise the laser cannot be turned off. Of course, some manufacturers' lasers may also require a laseron signal in addition to this signal. In this case, it does not matter whether it is set to 0 or not.

```
para[i].DutyCycle = dutyCycle;//Light duty cycle, (0~1)
para[i].Frequency = frequency;//Outgoing light frequency KHZ
para[i].StandbyFrequency = frequency;//No light Q frequency(KHZ);
para[i].StandbyDutyCycle = dutyCycle;//No light Q duty cycle;
para[i].LaserPower = power;//Laser energy percentage(0~100), 50 represents 50%
para[i].AnalogMode = 0;
if (comboxLaser.GetCurSel() == 1)
{
    para[i].AnalogMode = 1;//1 represents the use of analog output, at this time you can control analog output (0~10V)
    para[i].Waveform = 0;
}
else if (comboxLaser.GetCurSel() == 2)
{
    para[i].StandbyFrequency = 0;//No light Q frequency(KHZ);
    para[i].StandbyDutyCycle = 0;//No light Q duty cycle;
}
```

6.3 Concepts of Q frequency, Q pulse width, and Q period

The Q cycle (T) and Q frequency (F) are inversely related, that is, T = 1/F. The cycle unit is seconds (S), and the frequency unit is Hertz (HZ). Usually in the laser industry, Q frequency is generally used as a unit instead of Q cycle, where duty cycle = pulse width / cycle. From this formula, we can see that the duty cycle range is between 0 and 1。

If it is a fiber laser, the recommended duty cycle is 50%.

If it is a CO2 laser, its energy is determined by the duty cycle, so it is recommended to assign the energy percentage on the interface to the duty cycle. For example, if the user enters 50% energy, then dutyCycle = 50%.

Chapter 7 3D Marking Instructions

7.1 Basic principles of 3D marking

To enable the 3D marking function, you need to set the nDimensional value in the UDM_SetProtocol(int nProtocol,int nDimensional) function to 1, and call the UDM_AddPolyline3D() function when adding a graphic track.

In conventional marking, you only need to control the XY coordinates to mark the desired graphics on the surface of the object. The laser focus is mainly determined by the lens. When the lens is fixed, the focal length is fixed. Generally, the marking range of this lens is small, mostly within 300mm, which cannot meet the needs of a larger range, and its working surface can only be in two-dimensional space and cannot be zoomed at all times, so a 3D marking system is produced. The 3D marking system has a relatively large marking format, because it can change the focal length at any time, so it can mark on curved surfaces. In the 3D system, there is no focusing lens. It drives a concave lens through the Z axis to change the focal length of the light path.

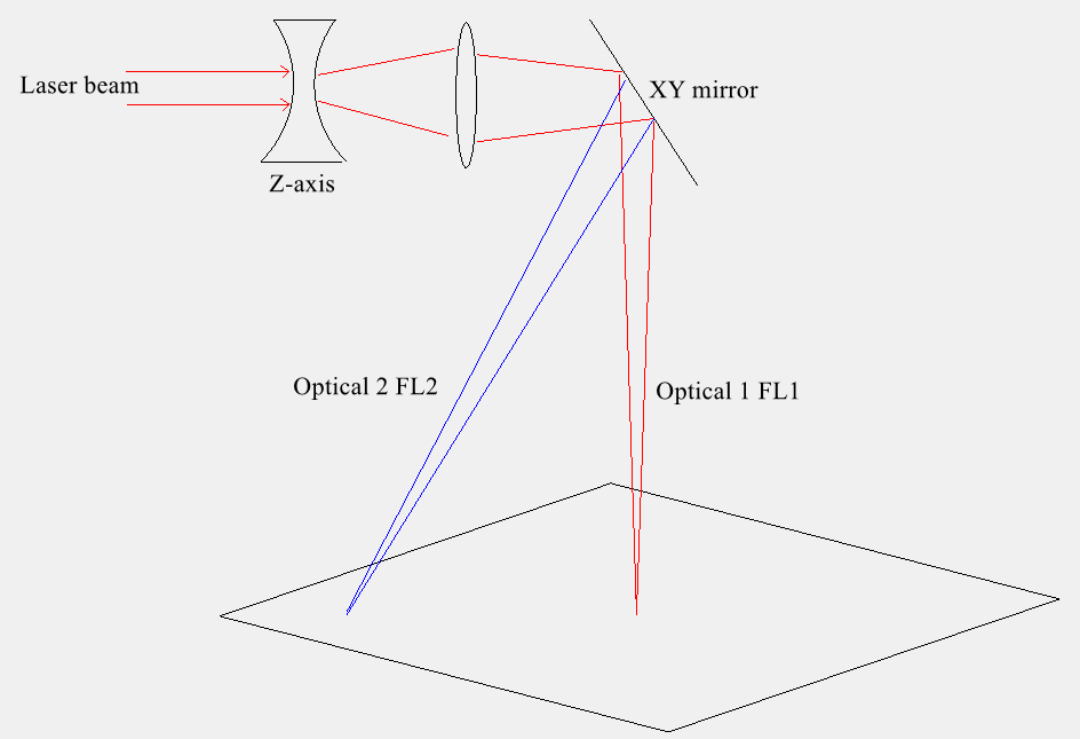


Figure 6.1 Simplified planar 3D optical path

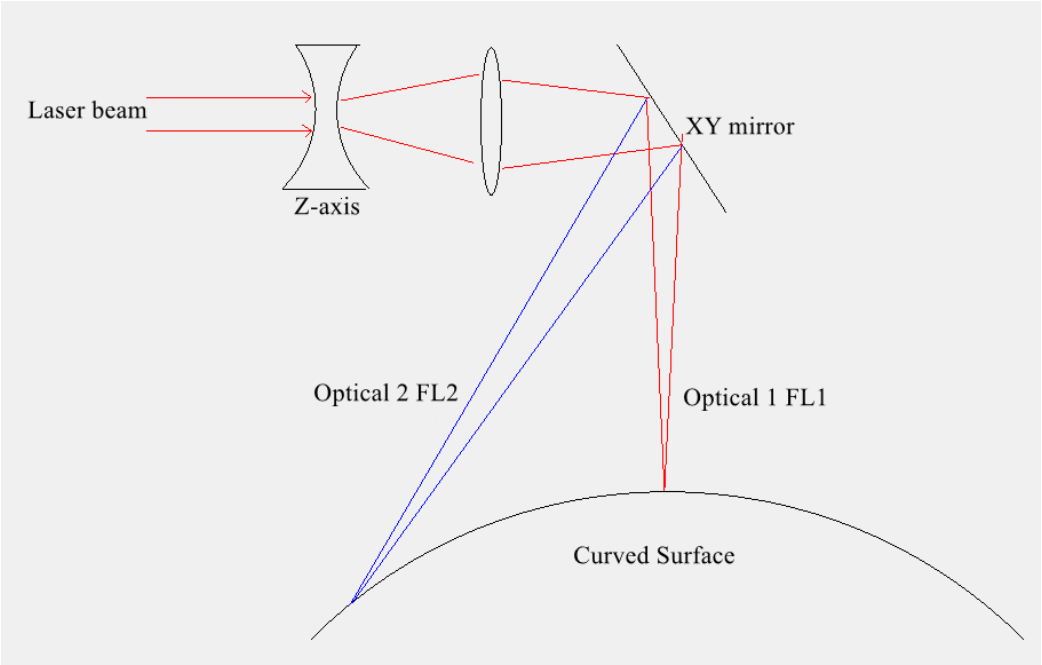


Figure 6.2 Simplified 3D optical path on curved surface

As shown in the figure above, the position of the convex lens is fixed. When the Z axis drives the concave lens to move left and right, the focal length of the laser focus changes, as shown in the optical path 1 and optical path 2 in the figure. The focal length is controlled by Z and the position is controlled by XY. In other words, XY determines the marking position and Z controls the focus. When XY is specified, Z must also be determined, otherwise the focal length will be incorrect and cannot be focused.

7.2 Relationship between the third axis Z value and focal length

In the further development, a point structure **structUdmPos** is defined, which contains three variables xyz. x and y are the position coordinates. The z value is the value of controlling the Z axis, in mm. z=1 means the Z axis moves 1 mm. However, it should be noted that a 1mm movement in the z-axis does not mean that the focal length f of the actual light changes by 1mm. The z value and the focal length f need to establish a functional relationship, that is, we need to know the corresponding change in f when the z value changes by a certain value. The process of determining the relationship between z and f is generally called 3D calibration. The following are common usage methods, which are only provided as reference.

As shown in the figure below, the basic focal length H refers to the height from the bottom of the 3D square head to the reference position of the marking. After the equipment is installed, this height is fixed and can be measured with a ruler. Usually, at the basic surface (0,0), z=0. That is, after the equipment is installed, at the (0,0) position, z is written as 0, which means it can focus.

According to the 3D space coordinates we can calculate:

f1 = sqrt(X1 * X1 + Y1 * Y1 + H * H)

f2 = sqrt(X2 * X2 + Y2 * Y2 + H * H)

...

fn = sqrt(Xn* Xn + Yn * Yn + H * H)

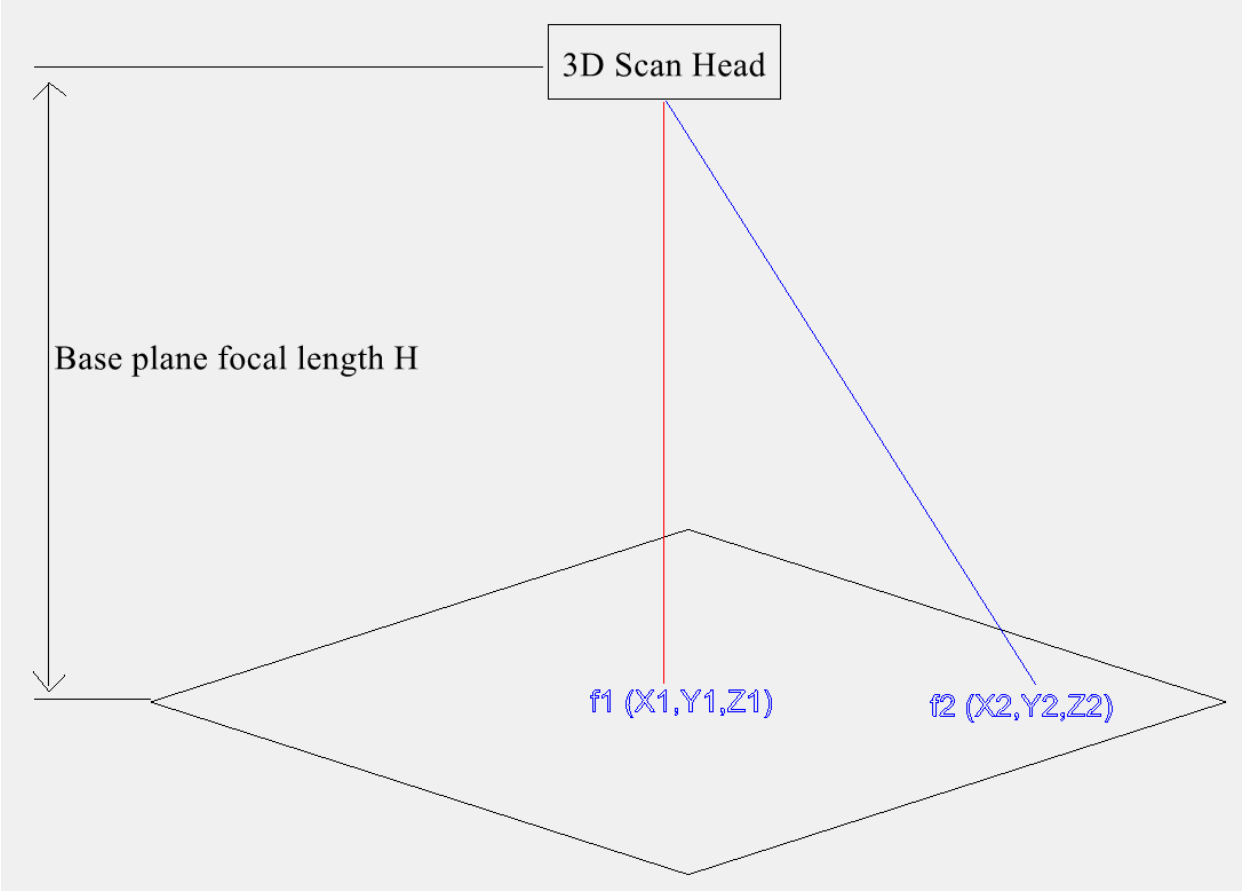


Figure 6.3 Schematic diagram of the relationship between f and z

How to determine the z value? At this time, we can take the following approach:

①First determine a set of (x, y), and make a very small circle with (x, y) as the center (call UDM_AddPolyline3D function to subdivide the circle into many points).

②Assign a z value to the z point of the above figure, such as 0, and then mark it. The human eye checks the laser focus. If it is focused at this time, record the x, y, z values at this time. If it is not focused, change a z value, such as 1, until it can be focused. Finally, record the (x, y, z) that can be focused

③Change a set of (x, y) coordinates, that is, change the marking position, and then repeat step ②.

④According to the situation, you can choose to measure several sets of data, and record all the (x, y, z) values that can be focused.

Figure 6.4 below is the 3D calibration interface of ScanWorld software, which can be used for reference.

In general, the focal length at z=0 is equal to the focal length of the reference plane.

Then, based on a series of (x, y, z) and H, we can find the relationship between z and f, that is,

f1 corresponds to z1, f2 corresponds to z2, f3 corresponds to z3 ... fn corresponds to zn

fn = sqrt(Xn* Xn + Yn * Yn + H * H)

The relationship between z and f can be obtained through the "least squares method" (the least squares algorithm code is available online)

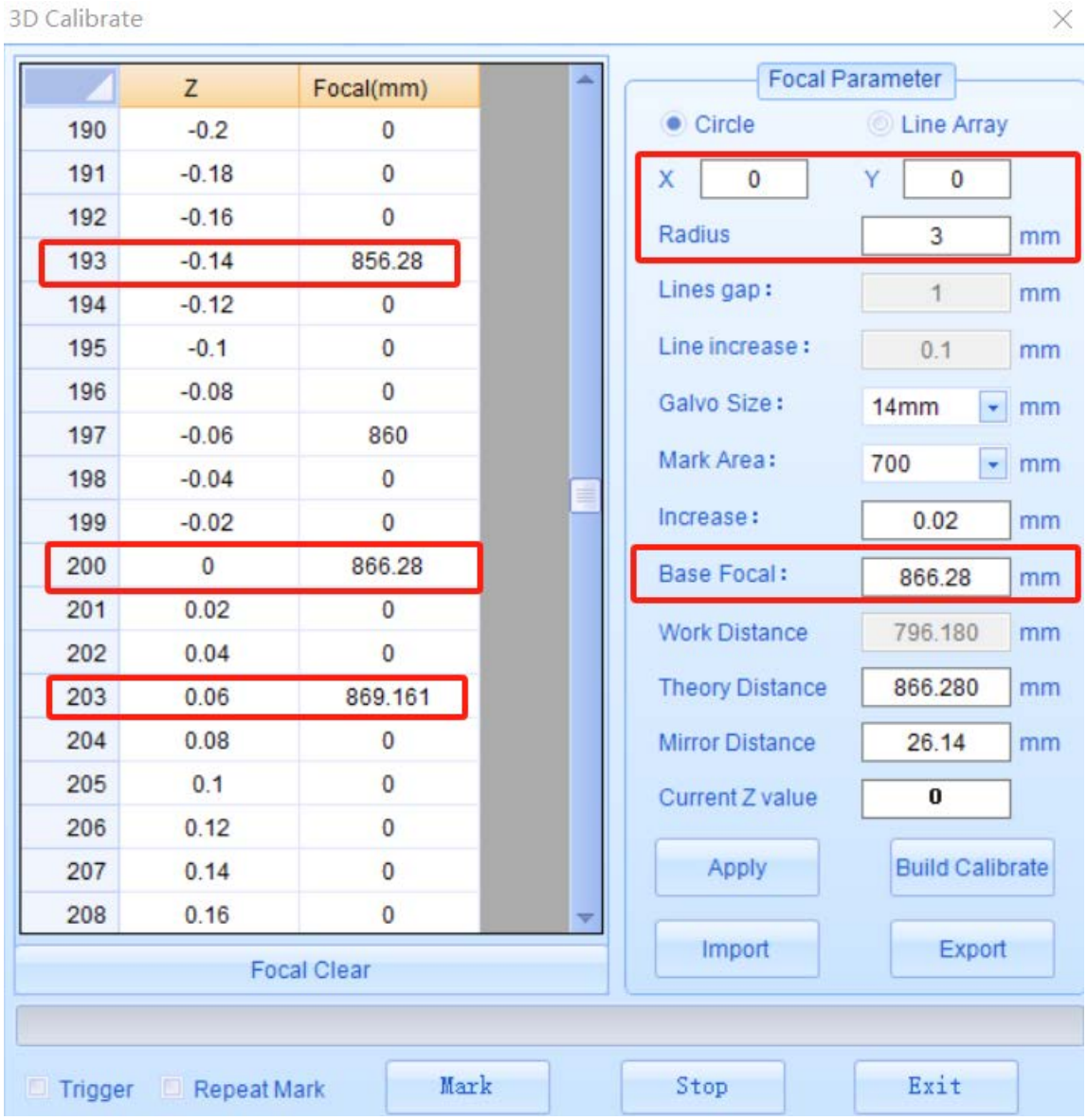


Figure 6.4 3D correction

7.3 Examples

If we measure the following data and put it into Excel, we can select the data in Excel, insert a scatter plot, and add a trend line (which can be linear, quadratic, cubic, multi-function, etc.). In this example, we selected a linear function. As shown in Figure 6.5 below. At this time, the relationship between z and f is z=0.0164 * f - 14.402

Then we calculate the value of z at the reference plane (100, 200). As mentioned above, z=0 is the focal length of the reference plane H(866.28). So at this time f=sqrt(100 * 100 + 200 * 200 + 866.28 * 866.28) = 894.674.

Then z=0.0164 * 894.674 - 14.402 = 0.27. At this time, the complete coordinates are (100,200,0.27).

f	z
686.28	-3.38
706.28	-2.92
726.28	-2.48
746.28	-2.08
776.28	-1.56
806.28	-1
836.28	-0.5
866.28	0
869.28	0.06
911.285	0.64
935.65	0.94
997.718	1.66

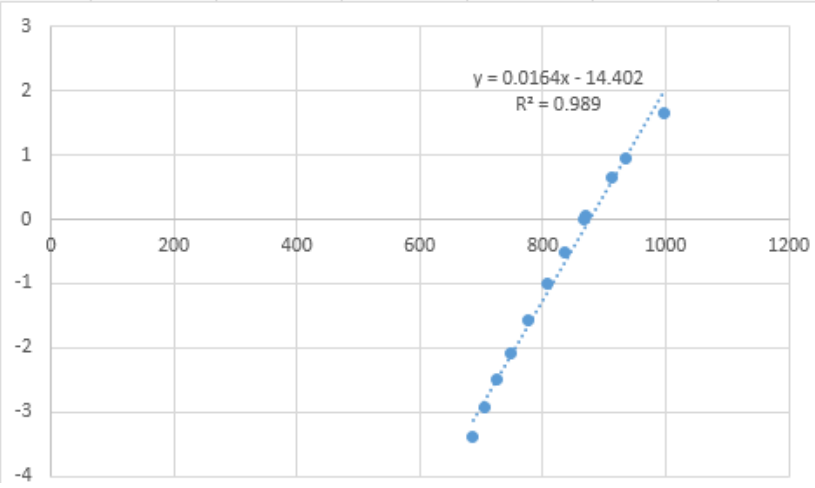


Figure 6.5 Data routine

Due to the special nature of the 3D optical path, the focal lengths are not equal on the same plane. That is, when we mark a straight line, in 2D marking, we only need to send the starting and ending position information. However, in 3D marking, the z value may not be the same everywhere from the starting position to the ending position. Therefore, it is necessary to subdivide a line segment, such as taking a point coordinate every 1mm, and calculating z based on the xy value here. For example, if a line segment is 10mm long, we turn it into a polyline composed of 10 points, where the Z value of each point is calculated based on the relationship of 3D correction. At this time, the UDM_AddPolyline3D() function needs to be called. In addition, in the UDM_SetProtocol(int nProtocol,int nDimensional) function, nDimensional needs to be set to 1. It means that the 3D marking function is enabled.

Note: In the current system -4≤z≤4

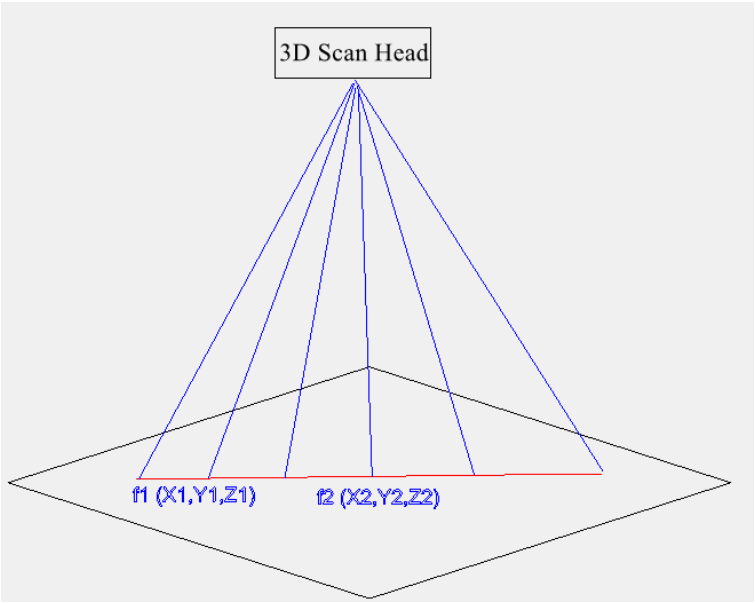


Figure 6.6

7.4 2.5D System

The 2.5D system means that the Z values in the same plane are all equal, and the Z values in

planes at different heights are different. Its optical path system is determined by the 3D system + lens.

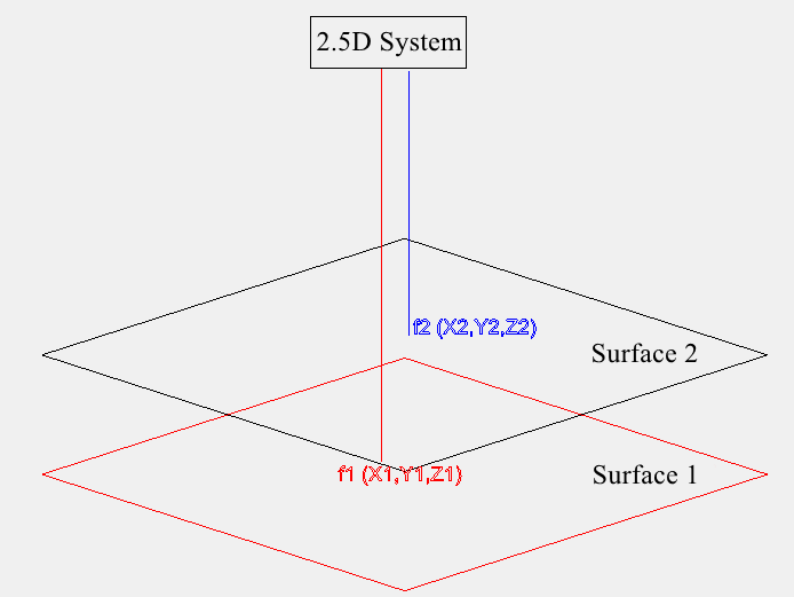


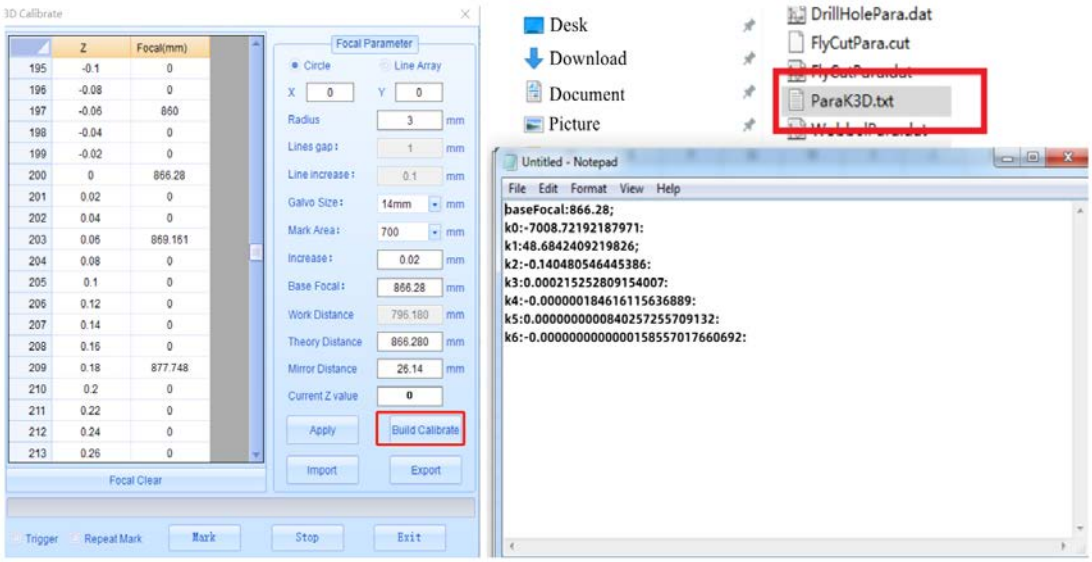
Figure 6.7 2.5D system

As shown in the figure above, in plane 1, the z1 value is equal everywhere. In plane 2, the z2 value is also equal everywhere, but the z value of plane 1 and plane 2 are not the same. Based on this feature, when calibrating the 2.5D system, we take any (x, y) value to mark, and we need to rely on the lifting platform to control the height of the square head and the plane, and then measure with a ruler. A set of equations between f and z can also be obtained, but this time f is measured by the lifting platform, which has nothing to do with the xy value. Unlike the 3D system, the line segments in the same plane of the 2.5D system do not need to be subdivided according to the spacing of 1mm, because the Z value in the same plane is the same, while the Z value in the same plane of the 3D system is different.

7.5 Quick Application Code Examples

7.5.1 3D Basic logic of data call

The above introduces the basic principles of 3D system implementation. It may not be easy to understand for those who are new to this application. In order to reduce the development work of customers, the 3D correction can be directly performed using the ScanWorld software that comes with the control card. After the 3D correction interface is calibrated, a ParaK3D.txt file will be generated in the Parameter Library folder in the root directory of the software. This file stores the functional relationship between the focal length f and the z-axis.

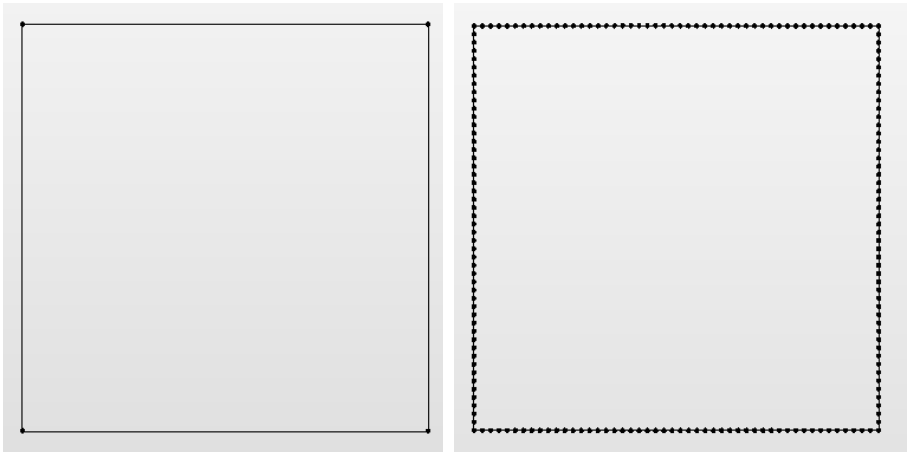


Where baseFocal is the focal height of the reference plane, that is, the height that can be focused at the (0,0) position when the Z axis is at position 0, k0~kn are the coefficients of the multi-order function. At this time, we only need to call the UDM_Set3dCorrectionPara(float baseFocal,double *paraK, int nCount) function to quickly apply the 3D correction table. The code assignment is as follows:

```
double paraK[7];
paraK[0] = -7008.72192187971;
paraK[1] = 48.6842409219826;
paraK[2] = -0.140480546445386;
paraK[3] = 0.000215252809154007;
paraK[4] = -0.000000184616115636889;
paraK[5] = 0.000000000840257255709132;
paraK[6] = -0.000000000000015855701766069 2;
UDM_Set3dCorrectionPara(866.28,paraK, 7);
```

After setting the correction parameters, we can call the UDM_AddBreakAndCorPolyline3D(structUdmPos *nPos, int nCount, float pGap, int layerIndex) function to add the marking track. In this function, we need to understand the z in the midpoint coordinate as the height (mm), that is, the height value of this position in our actual space, rather than the distance the z-axis motor moves. The following three logical processes are performed in this function:

1. Subdivide this polyline into more points according to the spacing of pGap. For example, convert a figure with five points into more points:



2. According to the xy and height values in the point coordinates, obtain the required value of the z-axis through the 3D correction parameters.

For example, there is a point structUdmPos point; point.x = 100; point.y = 100; point.z = 10 (height 10mm).

At this point we need to calculate the value sent to the z motor based on this xyz value using the correction table.

Then point.z = UDM_GetZvalue(x, y, z);

Function UDM_GetZvalue(float x,float y, float height); The internal implementation code is as follows：

```
double UDM_GetZvalue(float x,float y, float height)
{
    double distance = sqrt(x * x + y * y + (baseFocal - height) * (baseFocal - height));
    double zVal = 0;
    for (int i = 0; i < corPara3DCount;i++)
    {
        zVal += (corPara3D[i] * pow(distance,i));
    }
    return zVal;
}
```

3. After the above two steps, the curve is subdivided, and the z value of each point is passed through the 3D correction relationship to obtain the final value sent to the z axis. At this time, UDM_AddPolyline3D(structUdmPos *nPos, int nCount, int layerIndex) is called; finally, the data is written to the udm.bin file.

7.5.2 3D Whole Code Example

```
void creatUdmBin()
{
    UDM_NewFile();
    UDM_Main();
    UDM_SetProtocol(comboxProtocol.GetCurSel(),1);
    UDM_SetLayersPara(getMarkParameter(),2);
    double paraK[7];
    paraK[0] = -7008.72192187971;
    paraK[1] = 48.6842409219826;
```

```
    paraK[2] = -0.140480546445386;
    paraK[3] = 0.000215252809154007;
    paraK[4] = -0.000000184616115636889;
    paraK[5] = 0.000000000840257255709132;
    paraK[6] = -0.000000000000015858557017660692;
    UDM_Set3dCorrectionPara(866.28,paraK, 7);
    if (((CButton*)GetDlgItem(IDC_CHECK_LINE))->GetCheck() == BST_CHECKED)
    {
        structUdmPos polyline2d[2] = {-30,30,0,0,30,-30, 0,0,};
        UDM_AddBreakAndCorPolyline3D(polyline2d,2,1,0);
    }
    if (((CButton*)GetDlgItem(IDC_CHECK_CIRCLE))->GetCheck() == BST_CHECKED)
    {
        structUdmPos polyline2d[61];
        float radius = 25;
        double dAngle = 2 * 3.1415 / 60;
        for (int i = 0; i < 61;i++)
        {
            polyline2d[i].x = radius * cos(dAngle * i);
            polyline2d[i].y = radius * sin(dAngle * i);
            polyline2d[i].z = 0;
            polyline2d[i].a = 0;
        }
        UDM_AddBreakAndCorPolyline3D(polyline2d,61,1,0);
    }
    if (((CButton*)GetDlgItem(IDC_CHECK_RECTANGLE))->GetCheck() == BST_CHECKED)
    {
        float halfSize = 50;
        structUdmPos polyline2d[5] = {-halfSize,-halfSize,10,0,+halfSize,-halfSize,
10,0,halfSize,halfSize, 10,0,-halfSize,halfSize, 10,0,-halfSize,-halfSize, 10,0};
        UDM_AddBreakAndCorPolyline3D(polyline2d,5,10,0);
    }
    UDM_Jump(0, 0, 0);
    UDM_EndMain();
    UDM_SaveToFile("D:\\UDM.bin");
}
```

Chapter 8 Offline Marking

GMC control card supports single document offline marking and multi-document offline marking. After the offline process is completed, there is no need to open the computer or software. Just input a signal to IN0, and the control card can mark automatically. It will mark once every time it is triggered. When marking multiple documents offline, you need to insert an SD memory card, but you don’t need to

insert it when marking a single document offline. Offline marking is a good choice when you need to mark fixed graphics and don't want to equip a computer to save costs.

8.1 Single document offline marking

As the name implies, single-file offline means that the control card can only execute one marking document, that is, it only stores one udm.bin marking file. When the external trigger signal is given to the GMC card IN0, the control card immediately executes the marking action.

Before using offline marking, you need to call the HM_DownloadMarkFile() function to download the udm.bin marking file to the control card. After the marking file is downloaded, call the HM_BurnMarkFile(ipIndex,true) function to solidify the offline file. At this time, after triggering IN0, the control card can mark, and it will still be effective after power off and restart.

When performing offline curing, after downloading the udm.bin file, you can call the HM_BurnMarkFile() function in the OnMsgUDMDownloadEnd(WPARAM wParam, LPARAM lParam) message function, or you can handle it like the following code:

C++ version

```
downOver = false;
HM_DownloadMarkFile(ipIndex,"D:\\udm.bin",this->m_hWnd);
MSG msg;
while(downOver == false)
{
    GetMessage(&msg,NULL,0,0);
}
HM_BurnMarkFile(ipIndex,true);
```

C# version

```
downOver = false;
HM_HashuScanDLL.HM_DownloadMarkFile(currentIpIndex, "D:\\udm.bin", this.Handle);
while (downOver == false)
{
    Application.DoEvents();
}
HM_HashuScanDLL.HM_BurnMarkFile(currentIpIndex,true);
```

This means that before downloading the udm.bin marking file, first set a download completion flag downOver, then call HM_DownloadMarkFile, set downOver to true in the download completion message function, and after the download is complete, call HM_BurnMarkFile() for offline curing.

In addition, when clearing the offline data of a single document, you need to set the parameter in the function to false, such as: HM_BurnMarkFile(ipIndex,false). When you do not use offline marking and need to mark online all the time, it is best to clear the offline data first.

Warning: When marking a single document offline, the size of the udm.bin file is prohibited to exceed 2M, otherwise the control card will be abnormal, and serious cases require return to the factory for repair. If large files need to be marked offline, an SD memory card needs to be inserted and multiple documents can be marked offline.

8.2 Offline marking of multiple documents

To be opened