20

NET-SCAN 系统开发

Royal 板卡系统开发文档

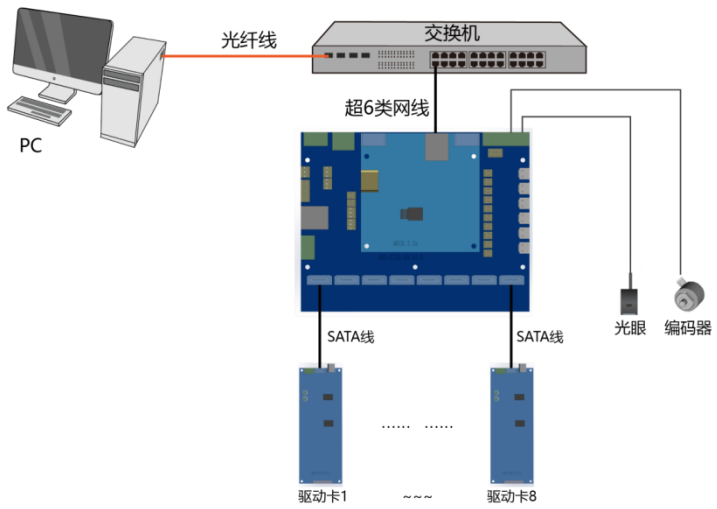
本文档主要介绍板卡系统连接框架，板卡系统架构逻辑定义，参数结构解析和 SDK 接口调用等

Max
ROYAL
2023/1/1

目录

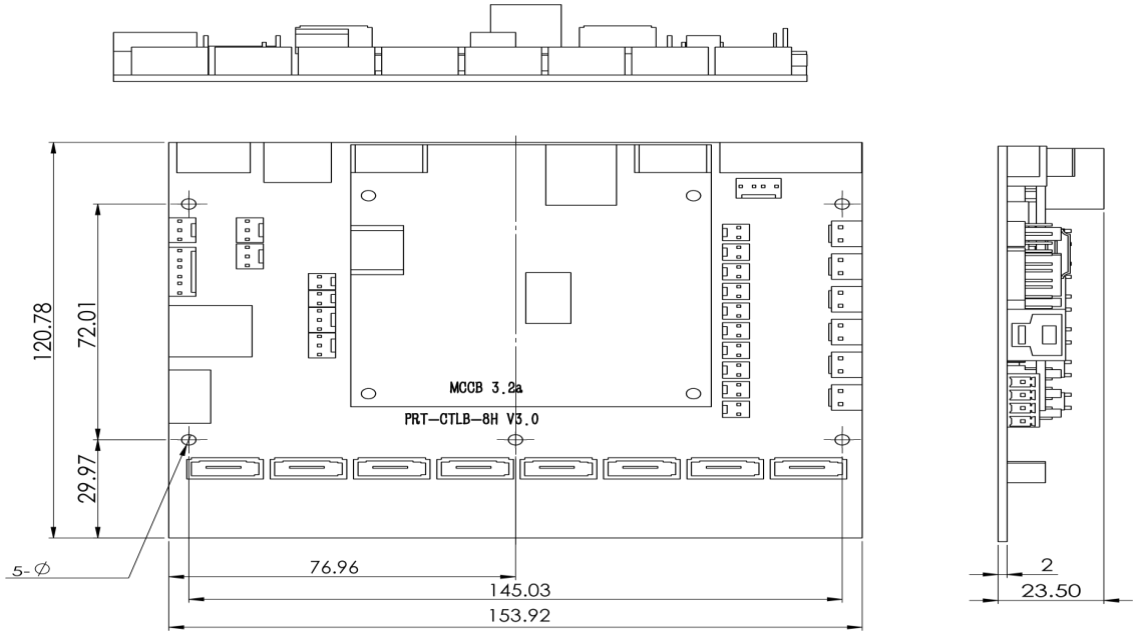
一、硬件系统连接

1.1 板卡连接示意图

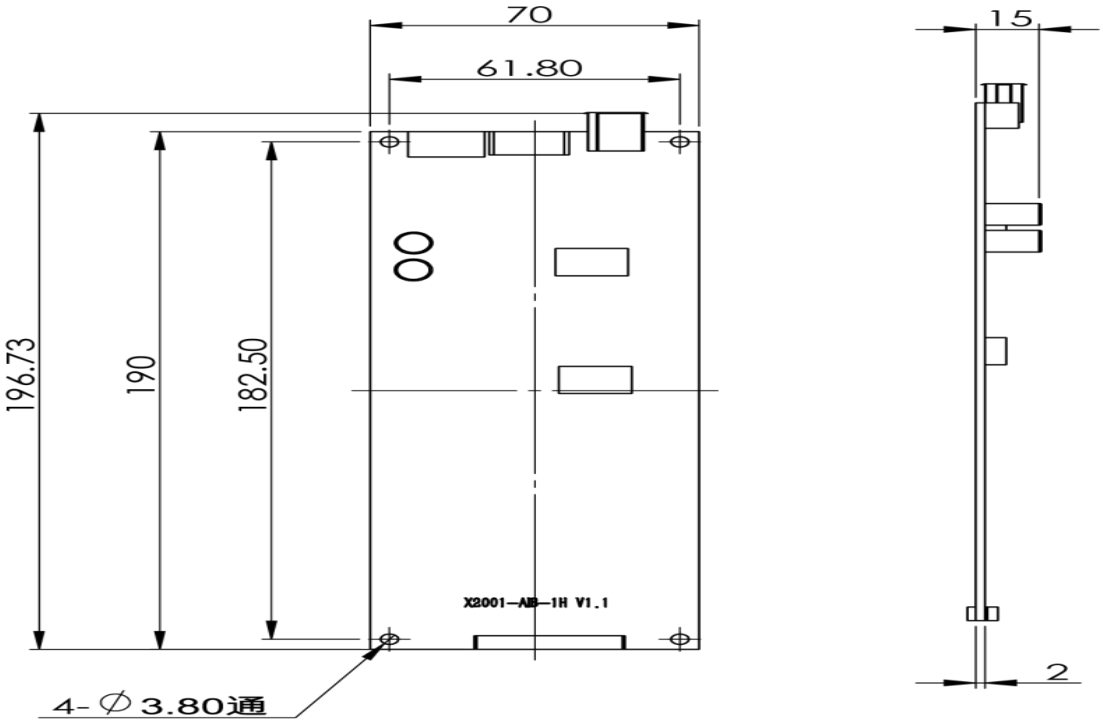


1.2 板卡尺寸图

1.2.1 主板

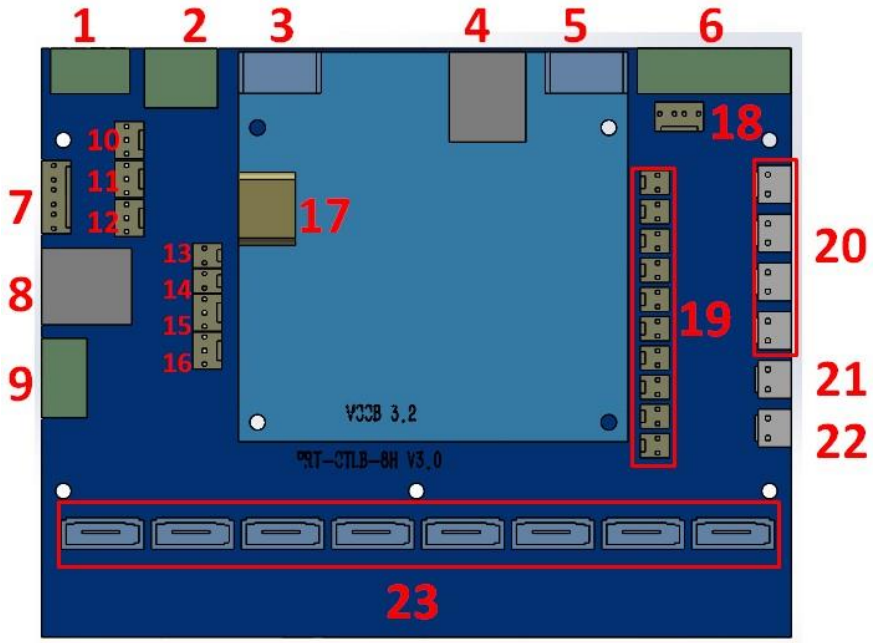


1.2.2 驱动卡



1.3 板卡端口介绍

1.3.1 主板

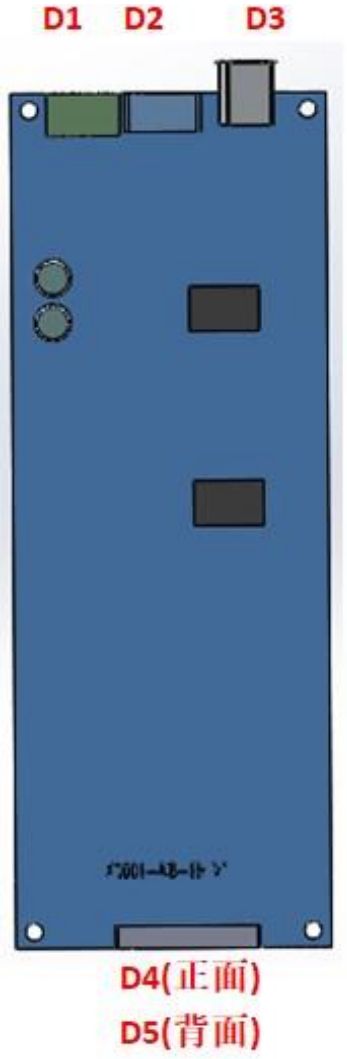


端口介绍：

编号	接口说明
1	电机接口
2	24V/5A(DC)输入电源
3	SATA 主信号接口
4	标准网口，用于实现主机与板卡通讯
5	SATA 从信号接口
6	差分编码器及光眼信号接口
7	4 路光眼接口
8	RJ485 接口，用于实现板卡之间的连接
9	37V 输出电源接口
10	左（上）限位
11	右（下）限位
12	零位
13	温度传感器接口 1
14	温度传感器接口 2
15	气压传感器 1
16	气压传感器 2
17	SD 读卡器接口

18	单端编码器接口
19	8 路液位信号
20	拓展接口(暂为定义)
21	加热输出 1
22	加热输出 2
23	8 路 1394 接口

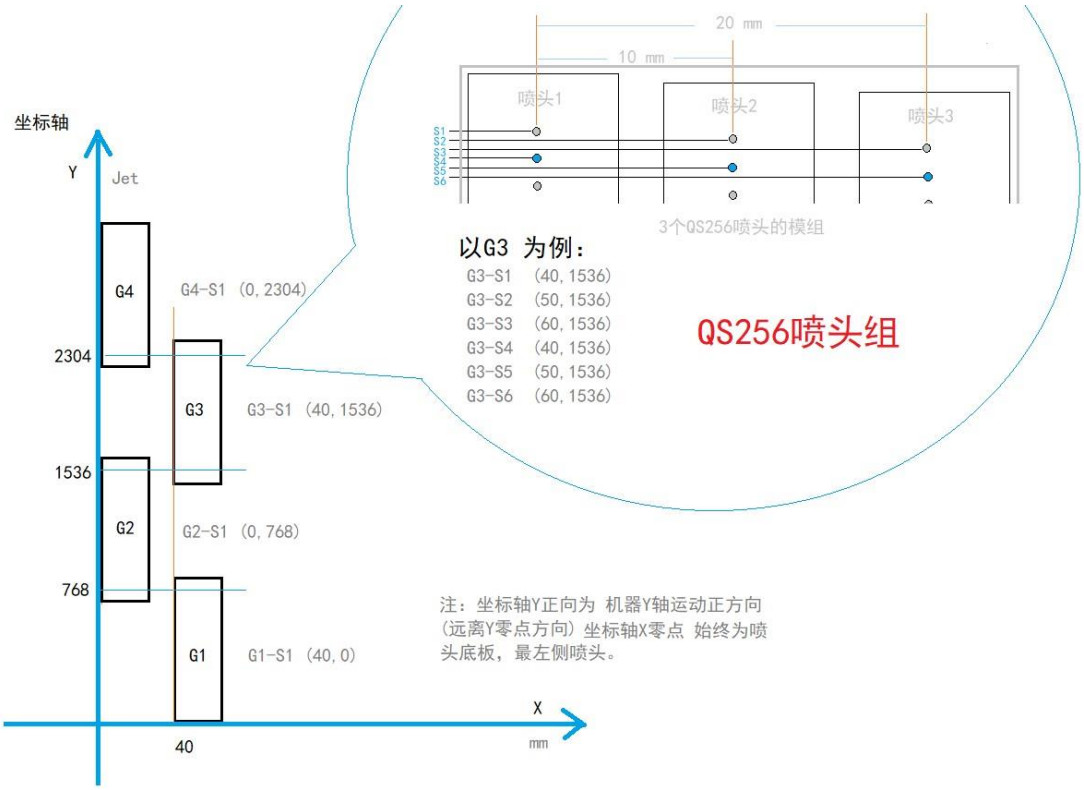
1.3.2 驱动卡



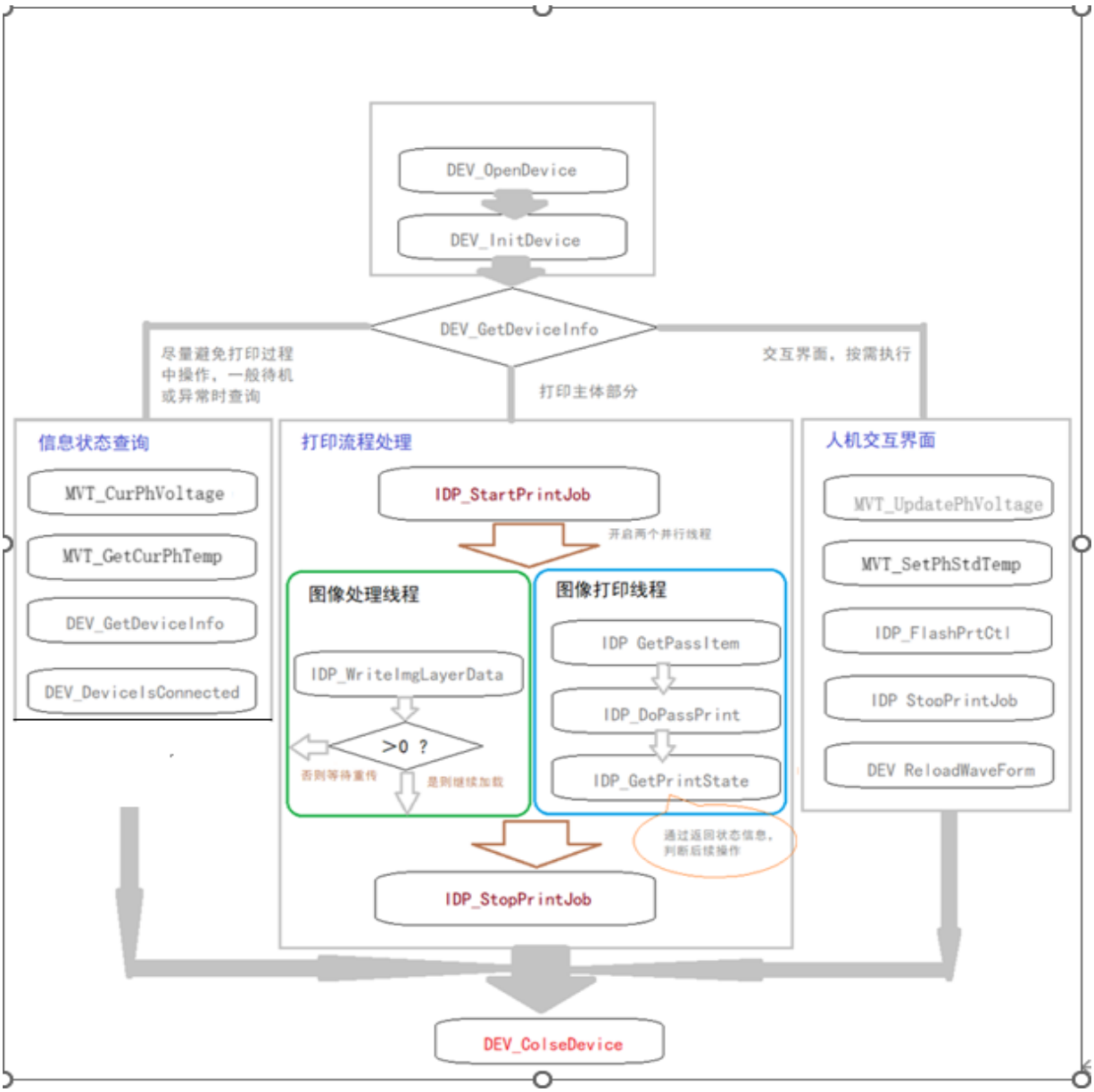
编号	接口说明
D1	输入电源接口
D2	Sata 通讯线接口
D3	1394 通讯接口
D4/D5	喷头接口

二、坐标轴的定义

X 零点始终为喷头底板最左侧喷头，Y 正向为 Y 轴前进的方向，以此为依据建立坐标轴。
（需要注意的是 X 编码计数往坐标轴 X 正向是需要递增的，要求与坐标系方向一致，否则需要通过系统参数来配置。）



三、软件系统框架



四、软件接口调用

接口调用部分主要介绍 RYPrtCtler.dll 的接口操作说明，主体总体分为三个模块，设备初始化、图像打印处理和温度电压控制，以及最终的关闭设备，图像打印处理和温度电压控制是必须设备初始化完成后才能正常执行的，详细模块介绍如下

4.1 设备初始化

Step1 ： 调用 Dev_OpenDevice 使 PC 端与板卡系统建立通讯
Step2 ： 调用 DEV_DeviceIsConnecte 使 PC 端与板卡判断是否成功建立通信
Step3 ： 调用 Dev_InitDevice 初始化设备运行状态，此过程会加载 Step2 中波形参数
Step4 ： 调用 Dev_GetDeviceInfo 查询需要的状态和信息，用来检查初始化设备状态
以上四个步骤完成初始化操作

4.1.1 建立板卡通讯

RYPRTCTLER_API int __stdcall DEV_OpenDevice(void* notifyhand, unsigned char* szDllFilePath);

参数：	notifyhand:上位机窗口句柄，可为 NULL。 szDllFilePath：NULL，使用 RYPrtCtler.dll 所在路径
返回值：	1: 连接正常 -2: pcs 文件加载失败 -4:graycfg 文件加载失败 -3:网络卡连接失败 -5:上位机软件多开

参考示例：	nResult=DEV_OpenDevice(m_hWnd,
	(unsignedchar*)(m_szAppPath.GetBuffer(0)));
	while(!DEV_DeviceIsConnected(0))
	{
	TimeOut++;
	if(TimeOut>20)
	{
	AfxMessageBox("主¡Â 板 ã?0 连¢?接¨®失 º¡ì 败 ã ¨¹");;
	break;
	}
	Sleep(500);
	}
	AfxMessageBox("主¡Â 板 ã?0 连¢?接¨®成¨¦功|");

4.1.2 更新设备参数

RYPRTCTLER_API bool __stdcall DEV_UpdateParam(LPRYUSR_PARAM pParam);

参数：	pParam：RYUSR_PRARAM 结构体指针，详情见附录结构体
返回值：	TRUE:成功，FALSE:失败
参考示例：	RYUSR_PARAM g_sysParam; DEV_UpdateParam(&g_sysParam);

4.1.3 初始化设备运行状态

RYPRTCTLER_API int __stdcall DEV_InitDevice(unsigned int nXEncInitVal);

参数：	nXEncInitVal：系统零位编码预设值
返回值：	0：初始化正常。 -1000001: -1000002

参考示例：	#define ZERO_POSITION 0x100000 nResult = DEV_InitDevice(ZERO_POSITION); //包含加载波形 if (nResult < 0)
	{
	szErrInfo.Format(_T("[%d] >> 初始化设备失败"), nResult);
	AfxMessageBox(szErrInfo);
	}

4.1.4 检查设备初始化信息

RYPRTCTLER_API LPPRINTER_INFO __stdcall DEV_GetDeviceInfo();

参数：	
返回值：	PRINTER_INFO 的结构体指针，详情见附录
参考示例：	LPPRINTER_INFO g_pSysInfo; g_pSysInfo = DEV_GetDeviceInfo();

4.2 打印控制管理

打印管理有 7 个步骤：
Step1 ： 调用 IDP_SartPrintJob 指定打印任务信息，用于启动打印任务，
Step2 ： 调用 IDP_WriteImgLayerData 指定打印图层信息，用于写入图层数据便于缓存，
Step3 ： 调用 IDP_StartLayerPrint 获取 PASS 数
Step4 ： 调用 IDP_GetPassItem 获取 PASS 信息体，用于提供给运动系统的控制
Step5 ： 调用 IDP_DoPassPrint 执行指定 PASS 打印，用于指定 PASS 的打印使能

Step6 ： 调用 IDP_FreeImageLayer 释放每层图片内存

Step7 ： 调用 IDP_GetPrintState 获取当前打印状态，用于判断打印状态和后续操作注：打印时需要满足 X 编码位置与打印起始编码关系，即正向打印时，墨车编码一定小于 pass 信息中的打印起始编码，反向打印时，墨车编码大于 pass 信息中的打印起始编码，否则无法出现剩余列数，无法执行打印流程。

4.2.1 启动任务

RYPRTCTLER_API int __stdcall IDP_SartPrintJob(LPPRTJOB_ITEM pJobItem);

参数:	pJobItem PRTJOB_IITEM 结构体指针，详情见附录
返回值:	-200000：上次打印未结束。 -2019：板卡未连接。 <-200000: 启动任务失败。

使用示例：	g_testJob.nJobID = m_nJobID; // JobID CopyMemory(g_testJob.szJobName,m_szJobName, m_szJobName.GetLength()); // 任务名称 g_testJob.fPrtXPos = m_fXPrtPos; // x 打印起点，单位 mm g_testJob.fClipHeight = m_fClipHeight; // 截图高度 g_testJob.fClipWidth = m_fClipWidth; // 截图宽度 g_testJob.fOutXdpi = m_fXOutDPI; // 输出 XDPI 解析 CLI 文件使用 g_testJob.fOutYdpi = m_fYOutDPI; // 输出 YDPI 解析 CLI 文件使用 g_testJob.nFileType = m_nFileType; // 指定文件类型 g_testJob.nOutPixelBits = m_nGrayBits; // 输出灰度 解析 CLI 文件使用 GetDlgItemText(IDC_MFCEDITBROWSE, m_szJobFilePath); CopyMemory(g_testJob.szJobFilePath,m_szJobFilePath, m_szJobFilePath.GetLength()); // CLI 文件路径 g_testJob.nPrtCtl &= ~0xff; if (m_bWhiteJump) // 跳白 g_testJob.nPrtCtl |= 0x1; if (m_bCycleOff) // 循环喷嘴偏移 g_testJob.nPrtCtl |= 0x2; if (m_bRadomJetOff) // 随机偏移 g_testJob.nPrtCtl |= 0x4; if (m_bXMirror) // X 镜像 g_testJob.nPrtCtl |= 0x10;
	if (m_bYMirror) // Y 镜像 g_testJob.nPrtCtl |= 0x20; if (m_bDoubleYDpi) // 双倍墨量 g_testJob.nPrtCtl |= 0x40; int nResult = IDP_SartPrintJob(&g_testJob); // 启动任务

4.2.2 写入图层数据

RYPRTCTLER_API int __stdcall IDP_WriteImgLayerData(LPPRTIMG_LAYER lpLayerInfo,unsigned char* pSrcBuf[],int nBytes);

参数：	LpLayerInfo:PRTIMG_LATER 指针，详情见附录 SrcBuf：char* 数组，指向图片数据 nBytes:单张图片数据量，字节数。 注意：这个函数需要单独开一个线程进行图像缓存且需要设置一个队列，建议缓存数不大于 20
返回值：	1 正常返回 0 已存在图层 -110001 图层数据为空 -110002 图层序号不满足增量关系 -110004 PC 内存不足 -110005 图层预处理错误

使用示例：	g_PrtImgLayer.nLayerIndex = m_nLayerIndex; // 图层索引
	g_PrtImgLayer.nStartJetOffset = m_nStartJetOff; // 起始喷嘴偏移
	g_PrtImgLayer.nPrtDir = m_nPrtDir; // 打印方向
	g_PrtImgLayer.fPrtXOffet = m_fXPosOff; // 图层相对任务起点 X 偏移
	g_PrtImgLayer.fPrtYOffet = m_fPrtYOffet; // 图层相对任务起点 Y 偏移
	g_PrtImgLayer.nColorCnts = m_nColorCnts; // 图册颜色数
	g_PrtImgLayer.nXDPI = m_fXDPI; // 图层 XDPI g_PrtImgLayer.nYDPI = m_nYDPI; // YDPI
	g_PrtImgLayer.nStartPassIndex = m_nStartPassInedx; // 起始打印 Pass
	g_PrtImgLayer.nFeatherMode = m_nFeatherMode; // 羽化模式
	g_PrtImgLayer.nCustomFeatherJets = m_nCustomFeatherJets; // 自定义羽化嘴
	g_PrtImgLayer.nEdgeScalePixel = m_nEdgeScalePixel; // 边缘收缩像素
	g_PrtImgLayer.fRotateAngle = m_fRotateAngle; // 旋转角度
	g_PrtImgLayer.fDstXScale = m_fDstXScale; // X 缩放比例
	g_PrtImgLayer.fDstYScale = m_fDstYScale; // Y 缩放比例
	g_PrtImgLayer.fLayerDensity = m_fLayerDensity; // 层像素密度
	g_PrtImgLayer.fXScanSpd = m_fXScanSpd; // X 向扫描速度
	g_PrtImgLayer.nScanCtlValue = m_nScanCtlValue;
	g_PrtImgLayer.nImgType = m_nImgType; // 文件类型
	g_PrtImgLayer.nPrtFlag &= ~0xff; if (m_bDoubleDir) // 双向打印

	g_PrtImgLayer.nPrtFlag |= 0x1; if (m_bYReverse) // Y 反向
	g_PrtImgLayer.nPrtFlag |= 0x2; if (m_bXInsert) // X 插点
	g_PrtImgLayer.nPrtFlag |= 0x4; if (m_bSwitchByWhite) // 方向切换由跳白控制
	g_PrtImgLayer.nPrtFlag |= 0x8; if (m_bEdgeNoReduce) // 边缘不抽点
	g_PrtImgLayer.nPrtFlag |= 0x10; if (m_bPrtArea) // 防安装干涉
	g_PrtImgLayer.nPrtFlag |= 0x20; if (m_bGrayBits) // 灰度打印控制
	g_PrtImgLayer.nGrayBits = 2; else g_PrtImgLayer.nGrayBits = 1; LPPRTIMG_LAYER pLayer = &g_PrtImgLayer; nResult = IDP_WriteImgLayerData(pLayer, pBmpFile, nPrtDataSize);

4.2.3 获取 pass 数

RYPRTCTLER_API int __stdcall IDP_StartLayerPrint(int nLayerIndex);

参数：	nLayerIndex ： 图层序号
返回值：	>0 返回 pass 总数 -1 图层数据未就绪 -120000~-120016 初始化失败 -121000 板卡内存分配不足 -122000 启动数据分发线程失败
参考示例：	见 4.2.4 示例

4.2.4 获取 pass 信息

RYPRTCTLER_API LPPassDataItem __stdcall IDP_GetPassItem(unsigned int nLayerIndex,int nPassID);

参数：	nLayerIndex: 图层索引 nPassID： PASS 索引
返回值：	对应图层的 PassDataItem 结构体指针，详情见附录

示例：	for (int k = 0; k < pdlg->m_nJobImgLayerCnts; k++)
	{
	nPassCount = IDP_StartLayerPrint(k);
	while(nPassCount == -1)
	{
	nPassCount = IDP_StartLayerPrint(k);
	
	}
	
	for (int i = 0; i < nPassCount; i++) //
	{
	pPrtPassDes = IDP_GetPassItem(k, i); //获取每 PASS 的执行信息
	if (pPrtPassDes)
	{
	if (!IDP_DoPassPrint(pPrtPassDes)) //执行指定 PASS 打印
	{
	break;
	} do
	{
	IDP_GetPrintState(&RTinfo); //获取打印运行状态，用来判断当前打印的状态

	if (!pdlg->m_bJobStarted) //结束打印
	break;
	Sleep(1);
	} while (RTinfo.nPrtState == 1);
	} else
	{
	break;
	}
	
	}
	Sleep(100);
	IDP_FreeImageLayer(n);
	}
	IDP_StopPrintJob();

4.2.5 执行 PASS 打印

RYPRTCTLER_API bool __stdcall IDP_DoPassPrint(LPPassDataItem pPassItem);

参数：	PassDataItem 结构体指针，详情见附录

返回值：	TREU:成功 FALSE:失败
参考示例：	见 4.2.4 示例

4.2.6 释放图层信息

RYPRTCTLER_API int __stdcall IDP_FreeImageLayer(int nLayerIndex);

参数：	nLayerIndex 图层序号 如果传-1 就是释放所有缓存图层
返回值：	TREU:成功 FALSE:失败
参考示例：	见 4.2.4 示例

4.2.7 获取打印状态

RYPRTCTLER_API bool __stdcall IDP_GetPrintState(LPPrtRunInfo pRTinfo);

参数：	PRTinfo:PrtRunInfo 结构体指针，详情见附录
返回值：	TRUE:成功 FALSE:失败
参考示例：	见 4.2.4 示例

4.2.8 结束打印任务

RYPRTCTLER_API bool __stdcall IDP_StopPrintJob();

参数：	nDevIndex ： 主板编号
返回值：	TRUE:成功 FALSE:失败
参考示例	见 4.2.4

4.2.9 闪喷控制

RYPRTCTLER_API bool __stdcall IDP_FlashPrtCtl(bool bOpen);

参数：	bOpen：TRUE 开启闪喷 FALSE: 关闭闪喷
返回值：	TRUE:成功 FALSE:失败

参考示例：	static bool IsOpenFlash = *FALSE*;
	//while(1)
	{
	//if (IDP_GetPrintState(&runInfo))
	{
	
	//BOOL bFlashState = (runInfo.nPrtState == 3);
	*CWnd** pWnd = *GetDlgItem*(IDC_BUTTON_FLASH);
	if(IDP_FlashPrtCtl(!IsOpenFlash))
	{
	//AfxMessageBox("成¨¦功|");
	}
	else
	{
	AfxMessageBox("失º¡ì败ã¨¹");
	}
	//IsOpenFlash = TRUE;
	if ((IsOpenFlash)&&pWnd)
	{
	IsOpenFlash = *FALSE*;
	pWnd->*SetWindowText*(_*T*("启?动¡¥闪¦ ¨¢喷?"));
	}
	else
	{
	IsOpenFlash = *TRUE*;
	pWnd->*SetWindowText*(_*T*("关?闭 À?闪¦ ¨¢喷?"));
	}
	
	}
	//Sleep(500);
	}

4.2.8 写入图层时传递文件路径

RYPRTCTLER_API int __stdcall IDP_WriteImgLayerFile(LPPRTIMG_LAYER lpLayerInfo,char* szFile);

参数：	LpLayerInfo 图层结构体指针，详情见附录 SzFile：文件路径及名称
返回值：	
参考示例：	IDP_WriteImgLayerFile(lpLayerInfo,”D:\\1.PRT”);

4.3 温度电压控制

温度电压控制：
调用 MVT_SetPhStdTemp 设置喷头温度调用 MVT_UpdatePhVoltage
更新喷头电压
备注：获取喷头 ID 方法可依据配置颜色和组号从 PRINTER_INFO 结构体中的 nPHIDLKT 表查找，
int nPhIDLKT[MAX_COLORS][MAX_GROUP]; //bit0~bit15: PHID bit16:~bit19 subPhID(1 头多色)

4.3.1 设置喷头电压

RYPRTCTLER_API bool __stdcall MVT_UpdatePhVoltage(float* fstdVcom,unsigned int nPhID);

参数：	FstdVcom：设置电压数组 nPhid:喷头 ID
返回值：	TRUE：成功， FALSE:失败
参考示例：	MVT_UpdatePhVoltage(g_sysParam.phctl_param[ph].fVoltage, ph);

4.3.2 设置喷头温度

RYPRTCTLER_API bool __stdcall MVT_SetPhStdTemp(float* fstdTmp,unsigned int nPhID);

参数：	FstdTmp：温度数组 nPhid：喷头 ID
返回值：	TRUE:成功 FALSE:失败
参考示例：	MVT_SetPhStdTemp(g_sysParam.phctl_param[ph].fDestTemp, ph);

4.4 关闭设备

RYPRTCTLER_API bool __stdcall DEV_CloseDevice();

参数：	
返回值：	TRUE:成功，FALSE:失败

参考示例：	DEV_CloseDevice();

4.5 校准打印

调用流程:

Step1: 调用 IDP_StartCaliBration 函数,根据 nAdjType 指定校准图类型；

Step2：调用 IDP_GetPassItem 函数，获取指定 pass 信息；

Step3：调用 IDP_DoPassPrint 函数，打印指定 pass;

RYPRTCTLER_API int __stdcall IDP_StartCalibration(LPRYCalbrationParam pParam);

参数：	RYCalibration 结构体指针，参数详情见附录
返回值:	>0 打印总 pass 数

示例：

```
if (m_bPrinting) {
 AfxMessageBox(_T("其他校准处于打印中..."));  return;
} m_calParam.nAdjType = 5; // 指定校准类型
m_nPassCount = IDP_StartCalibration(&m_calParam);
AfxBeginThread(PrtCaliBrationThread,         this);
PrtRunInfo   RTinfo;   pdlg->m_bPrinting   =   true;
pdlg->m_bStop = FALSE; LPPassDataItem pPrtPassDes;
for (int i = 0; i < pdlg->m_nPassCount; i++) //
{  pPrtPassDes = IDP_GetPassItem(0, i); //获取每 PASS 的执行信息
 if (pPrtPassDes)
 {
  if (!IDP_DoPassPrint(pPrtPassDes)) //执行指定 PASS 打印
  {
   AfxMessageBox(_T("打印失败"));    break;
  }   do
  {
   IDP_GetPrintState(&RTinfo); //获取打印运行状态，用来判断当前打印的状态
   if (!pdlg->m_bStop) //结束打印
    break;
   Sleep(1);
  } while (RTinfo.nPrtState == 1);
 }
 else
 {   break;
 }
}
pdlg->m_bPrinting = false;
AfxMessageBox(_T("打印结束"));
```

4.6 编码重置与获取

4.6.1 获取编码

RYPRTCTLER_API unsigned int __stdcall DEV_GetPrinterEncValue(); //查询当前位置值

参数：	
返回值：	当前位置编码
调用：	UINT nPos = DEV_GetPrinterEncValue();

4.6.2 重置编码

RYPRTCTLER_API bool __stdcall DEV_ResetPrinterEncValue(unsigned int nPos);

参数：	板卡固定为 0X10000; nDevIndex ： 主板编号
返回值：	TRUE:成功 FALSE：失败
调用	DEV_ResetPrinterEncValue(0X10000);

4.7 波形加载

YPRTCTLER_API int __stdcall DEV_ReloadWaveForm(unsigned nClrMask,unsigned int nPHGMask,char* szWaveFile);

参数：	nClrMask 颜色掩码 nPHGMask:喷头组掩码 szWaveFile:波形文件全路径
返回值：	-1700002:更新波形失败 -1700003：波形文件加载失败
调用：	DEV_ReloadWaveForm((1, 1, "D:\\ricoh.rhdat"); 为颜色 1，组 1 的喷头更新波形，波形文件路径及名称 D:\\ricoh.rhdat;

4.8 设置板卡气压温度

RYPRTCTLER_API bool __stdcall MVT_SetAirPressCtlVal(unsigned int nPrtIndex);

参数：	车头卡索引
返回值：	TRUE 成功 FALSE:失败

调用：	for (int i = 0; i < 2; i++) { g_sysParam.fAirCtlPress[0][i] = m_fAirPressCtl[i]; g_sysParam.fSafeAirPress[0][i] = m_fAirPressSafeCtl[i]; g_sysParam.fInkTemp[0][i] = m_fInkTemp[i];
	} if (DEV_UpdateParam(&g_sysParam))
	{

	if (!MVT_SetAirPressCtlVal(0))
	AfxMessageBox(_T("设置温度气压失败"));
	}

4.9 输出控制

RYPRTCTLER_API bool __stdcall MVT_SetOutPut(int nPrtIndex, int nOutMask, bool bEnable);

参数:	nPrtIndex：主卡索引 nOutMask:输出控制 掩码 bEnable:输出使能
返回值：	TRUE：成功 FALSE：失败
示例：	配合 4.10 功能输出使用

4.10 功能输出配置

RYPRTCTLER_API bool __stdcall MVT_SetOutPutConfig(int nPrtIndex)

参数：	nPrtIndex：网络卡索引
返回值：	TRUE:成功 FALSE:失败

示例：	if (m_bEnableSupply)
	g_sysParam.nIoOption[0] |= 0x1; // 自动供墨时，手动控制输出无效
	else
	g_sysParam.nIoOption[0] &= ~0x1;// 关闭自动供墨;
	DEV_UpdateParam(&g_sysParam);更新参数
	MVT_SetOutPutConfig(0); // 控制功能输出
	MVT_SetOutPut(0, 0x2, TRUE); // 开启主卡 2 号口输出
	MVT_SetOutPut(0, 0x2, FALSE);// 关闭主卡 2 好端口输出

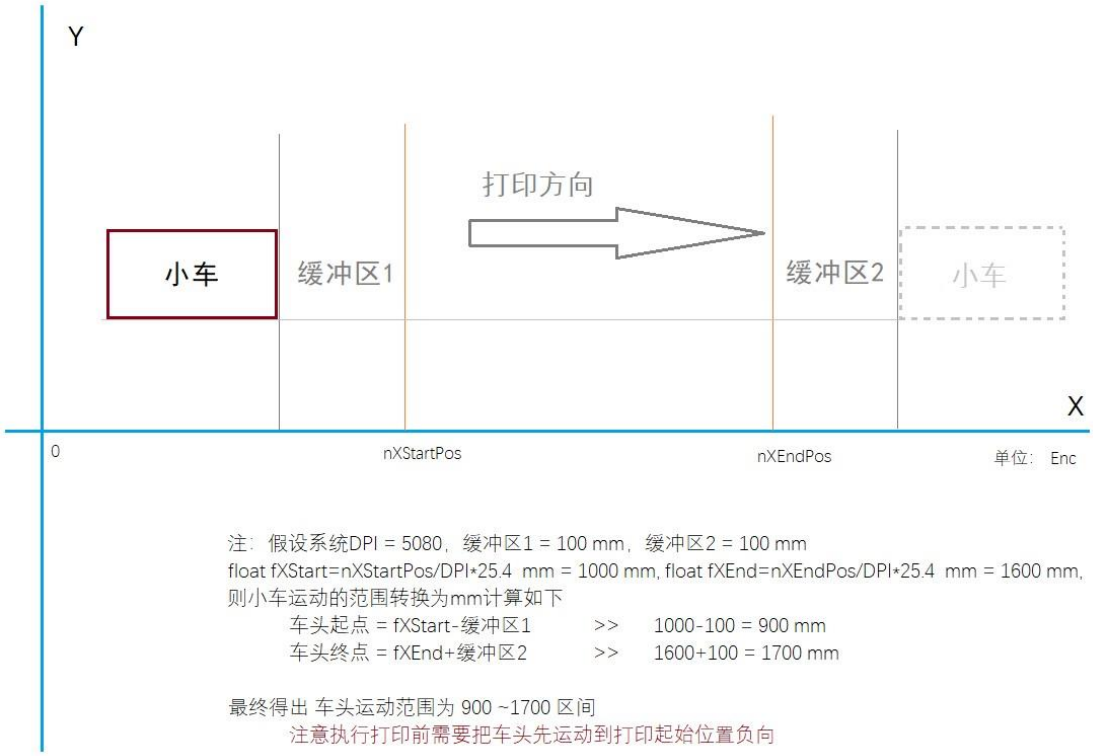
4.11 ADIB 板卡控制

RYPRTCTLER_API bool __stdcall DEV_AdibControl(LPADIB_PARAM lParam, unsigned int nOption, bool bSetParam, bool* bAbort, unsigned int nPrtIndex);

参数：	lParam：ADIB_PARAM 结构体指针，详请见附录 nPrtIndex：主卡索引 bSetParam：TRUE：设置，FALSE：读取设置状态： nOption：0x4:设置负压 0x10：设置负压阈值 0x40：关闭清洗模式 0x80：打开清洗模式 0x100：供墨 1 0x200：关闭供墨 1
	0x400：供墨 2 0x800：关闭供墨 2 0x1000：清洗泵 1 打开 0x2000：清洗泵 1 关闭 0x4000：清洗泵 2 打开 0x8000：清洗泵 2 关闭 0x80000000：写 I2C，负压值保存到板卡程序 0x40000000：读 I2C，从板卡读取负压值读取状态 dwOption：0x1 ADIB 卡程序版本 0x4：读取当前负压值 0x8：读取输入信号 0x4000：读取负压设置值
返回值：	TRUE：成功 FALSE：失败
示例：	详情见 demo

五、系统参数的计算

5.1 X 运动范围计算

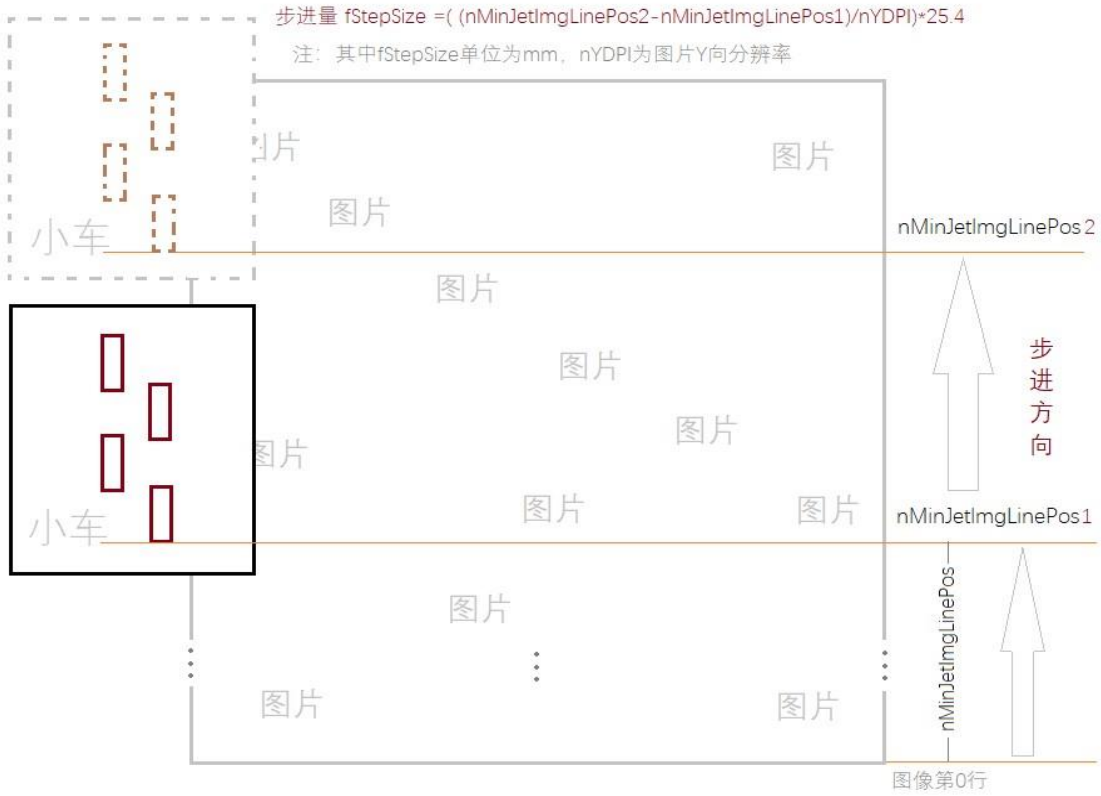


注：假设系统DPI = 5080，缓冲区1 = 100 mm，缓冲区2 = 100 mm
float fXStart=nXStartPos/DPI*25.4 mm = 1000 mm, float fXEnd=nXEndPos/DPI*25.4 mm = 1600 mm,
则小车运动的范围转换为mm计算如下

车头起点 = fXStart-缓冲区1 >> 1000-100 = 900 mm
车头终点 = fXEnd+缓冲区2 >> 1600+100 = 1700 mm

最终得出 车头运动范围为 900 ~1700 区间
注意执行打印前需要把车头先运动到打印起始位置负向

5.2 Y 步进量计算



当前 QS256 系统打印模式也可用以下计算： fStepSize= PASS 有效喷嘴数/300*25.4;

5.3 常用计算公式

5.3.1 mm 转 dot

```
#define MM_TO_DOT(X,DPI)  (int)(((float)(X*DPI))/25.4+0.45f)
```

5.3.2 dot 转 mm

```
static float DOT_TO_MM (int X,float DPI)
{
 //ASSERT(DPI!=0);
int   nresult;   float
nI=0.005f;     if(X<0)
nI=-0.005f;
 nresult= (int)(10*(((float)X*25.4)/DPI+nI));
 return ((float)nresult)/10;
}
```

5.3.3 栅距 μ 转 DPI

先将栅距转换为 mm 单位，则根据 DPI 定义计算如下

```
#define U_TO_DPI (μ) (25.4/(μ))
```

5.3.4 打印分频值

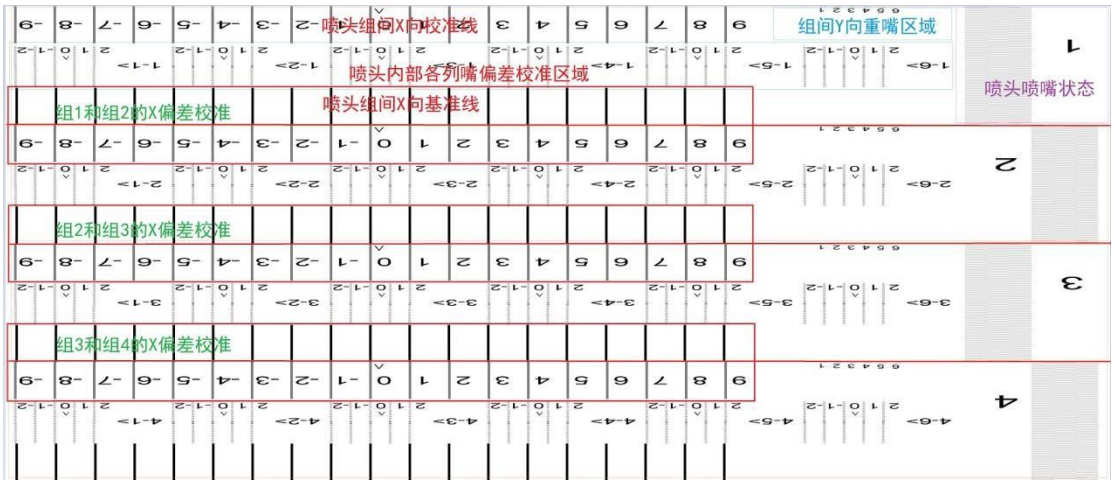
打印图像 X 的分辨率为 nXDPI，系统光栅为 nSysDPI 计算如下

分频值 = nSysDPI/nXDPI 所以打印图像的 XDPI 一般能整除光栅 DPI，这样精度更高

六、喷头校准说明

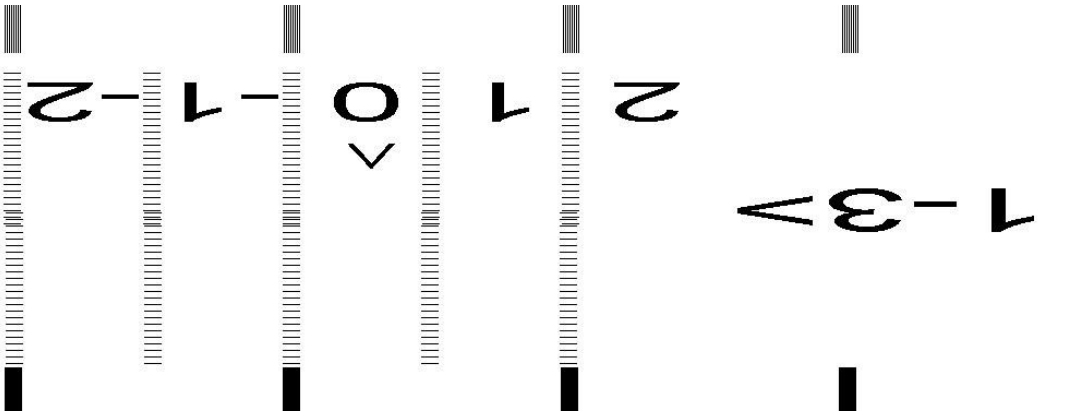
校准图都是游标卡尺原理，假设基准线间距是 100 像素，则校准线间距是 99 像素

6.1 X 向偏差校准图



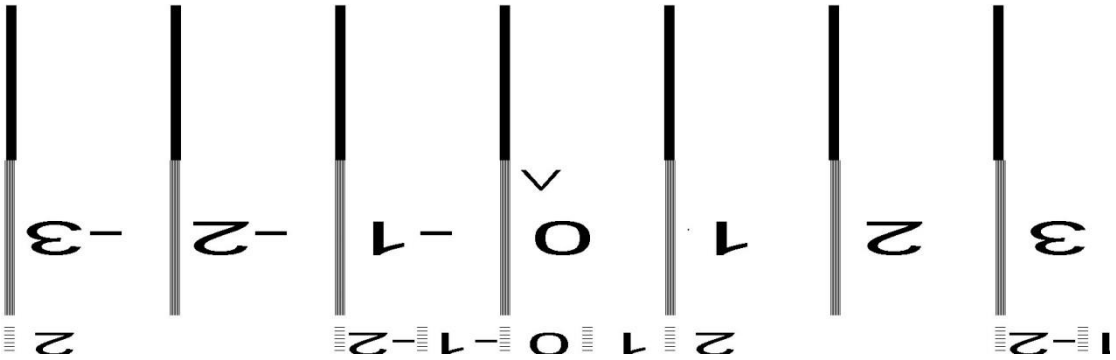
图中彩色线框和文字仅为作说明，并非原图数据，系统打印的原始数据为单色位图数据，打印此校准时不需要产生 Y 向步进，需要上位机开发者注意的是，所有校准的偏差值最终都需要转换为相对组 1 第一列喷嘴的结果。（以组 2 的 2-3 为例，假设组 1 和组 2 的偏差值是 5，2-3 的值是 3，那么 2-3 相对组 1 第一列的偏差值为 5+3=8）

6.1.1 喷头内部 X 校准区域



图示为 喷嘴第 3 列和基准列第 1 列的 X 单向的偏差

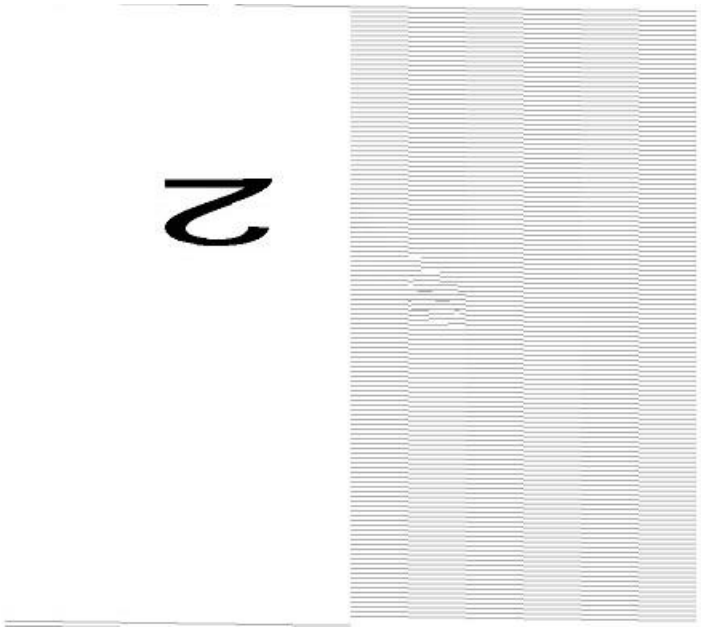
6.1.2 喷头组间 X 校准区域



6.1.3 喷头组间 Y 校准区域

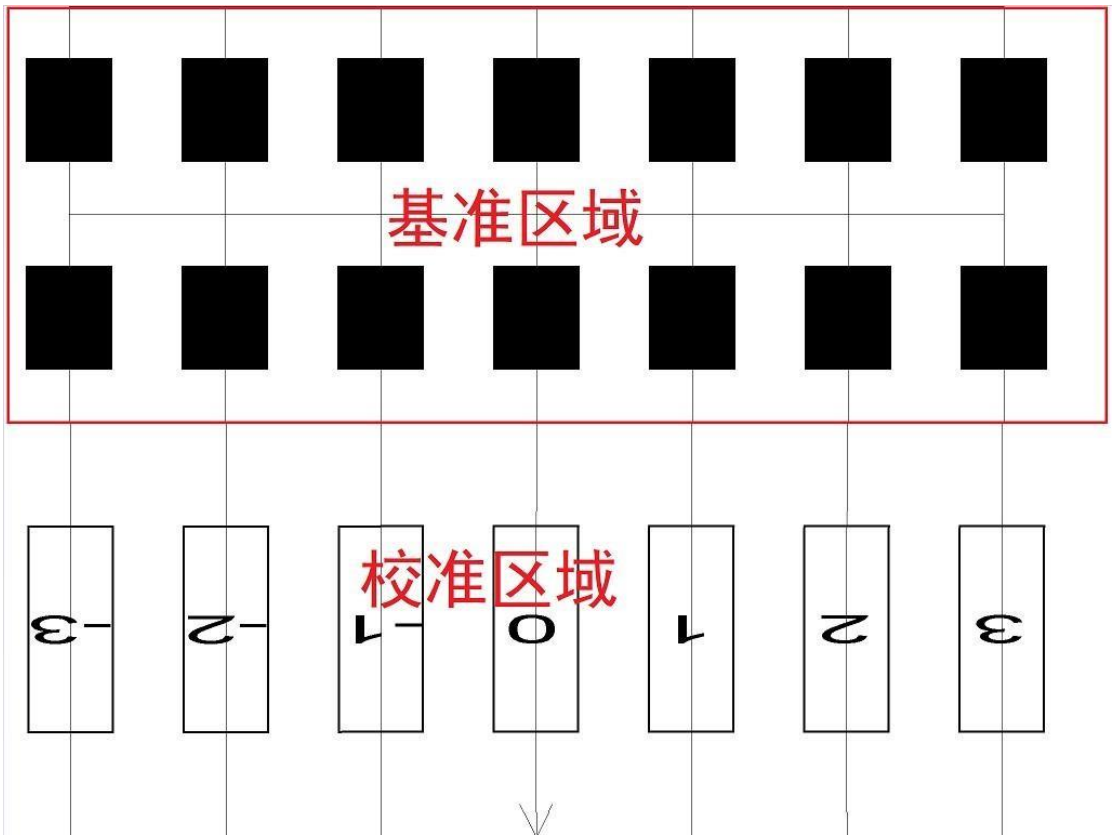
观察喷头组 Y 向喷嘴重叠情况

6.1.4 喷头喷嘴状态区域



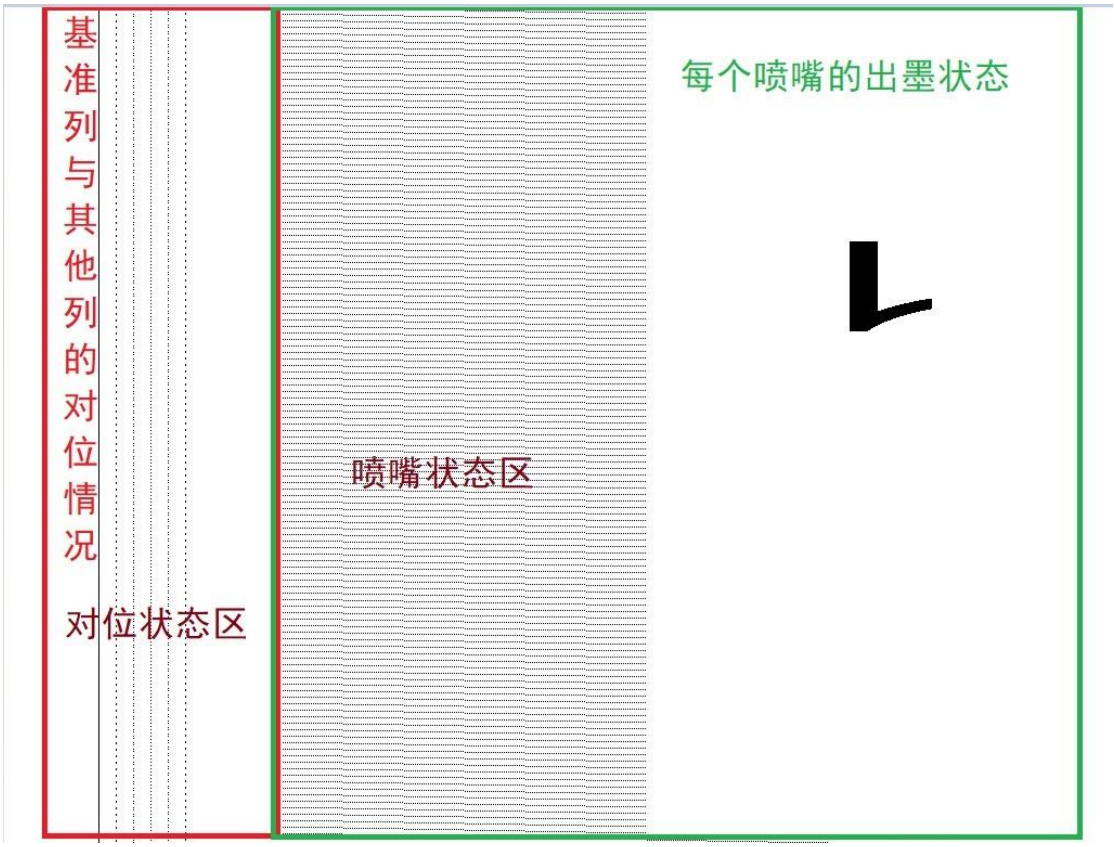
图示为组 2 喷头的各个喷嘴出墨状况

6.2 往返差校准图



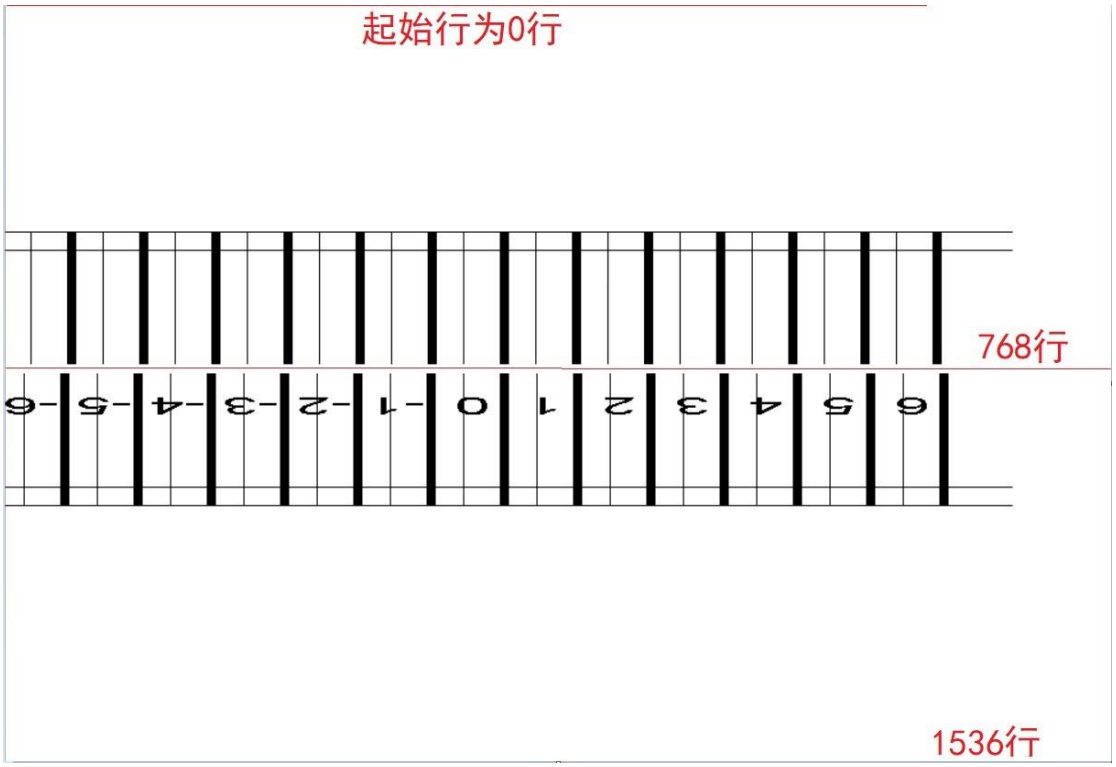
往返差校准可以通过设置打印图层信息 PRTIMG_LAYER 的 nPrtFlag 的 bit[0]置 1 来控制 2PASS 双向打印，第 1pass 打出基准区域（正向打印），第 2pass 打印校准区域（反向打印），注意打印此校准时不需要产生步进.

6.3 喷嘴状态图



图示从左至右 对位状态区 中实线为 6 列嘴的对位状态，虚线从左至右分别是 1-2、13、1-4、1-5、1-6 的对位状态，喷嘴状态区为查看喷头喷嘴情况的，Y 向有多少组头，就有多少个这样的状态，喷嘴状态区的短线呈台阶状

6.4 喷头机械安装



该部分校准原理很简单，只是为了检测喷头组安装的垂直情况，图中数值没实际意义，只能物理调节，需要微动 Y 步进来完成打印，图示为一组头的打印图，图片需要 2PASS 数据行，开发者可自己设计。

七、配置工具说明

SYSCFG 工具是用来描述系统喷头安装关系，其中涉及板卡连接关系，墨水使用关系，喷头位置和驱动卡连接关系；支持以下特性：

1. 支持不同颜色使用不同喷头类型
2. 不同颜色喷头数量可以不同
3. 支持多网络主卡合成多喷头应用
4. 单 PASS 打印和扫描系统都可以兼容使用

参数（左半边）	说明
颜色数量	最多支持 16 个颜色
编码器分辨率（DPI）	走料或车头走一英寸，编码器或者光栅给板卡的复合脉冲数。对于旋转编码器通常的计算=编码器线数*4/编码轮一圈相对的喷头位移长度(inch 单位)；对于直线编码器的计算=直接线数*4。
编码器前置分频数	将编码器的信号分频用，例如设置 1，编码器的分辨率设置就是实际的一半，一般保持默认设置为 0
设备打印方向	如上方框边的黑色箭头指向
应用模式	SP 分页触发：触发一次打印一次 SP 连续循环：连续不断打印扫描式打印：材料不动车头动的打印模式
网络打印控制卡数量	即主板使用数量
网卡序号/识别号/主从模式	识别号根据控制卡连接方式不同，填入后使用网卡序号，勾选方框确认主卡
触发电平反向	电眼信号常开不勾选，常闭勾选
编码器计数反向	正向移动（编码计数减少时）需要勾选
颜色组喷头类型独立	支持每个颜色用不一样的喷头
参数（右半边）	**说明**
色彩名	根据需求自定义，点击颜色名本身，可更改颜色
墨水 ID	根据需求控制墨水种类
X 位置（mm）	设置当前颜色在 X 轴的相对位置
Y 位置（mm）	设置当前颜色在 Y 轴的相对位置
喷头类型	选择喷头控制卡的系列
驱动卡类型	选择对应驱动卡

喷头数量	填入控制卡支持的喷头数量
颜色分配方式	H：喷头，C：颜色
安装方向	正向/反向
喷头起始位置	左侧/右侧
打印触发源	无触发：用内部触发或者扫描的打印模式 Trig1/Trig2：对应板卡的 IN1/IN2
组内距（mm）	改变组内 X 位置的值

按钮说明：

按钮	说明
PreView(预览)	所有设置的效果，点击后将预览在上方的空白方框内
Open(打开)	打开打印设置文件
Save(存储)	保存当前打印设置，文件格式为.tcf
Save As(存储为)	将当前打印设置另存为文件，文件格式为.tcf
Exit(退出)	退出软件

操作方法八、附录

具体信息结构体请参照提供的接口文件为准,以下仅供参考。

PRTIMG_LAYER,*LPPRTIMG_LAYER

```
//图层描述
typedef struct tag_ImgLayer
{
    long        nLayerIndex;        //图层索引
    int         nStartJetOffset;    //图像起始对应喷嘴偏移量，实现不同层同一位置使用不同喷嘴打印
    int         nPrtDir;            //图像打印起始PASS方向//  0 编码负方向 1编码正方向
    int         nXEncOff;           //图层针对任务X起点偏移 单位 X编码
    int         nYJetOff;           //图层针对任务Y起点偏移 单位 喷嘴间距
    int         nColorCnts;         //颜色通道数量
    float       nXDPI;              //图像XIDPI
    long        nYDPI;              //图像YIDPI
    long        nBytesPerLine;      //每行数据字节数
    long        nHeight;            //图像高度  单位：像素
    long        nWidth;             //图像宽度  单位：像素
    long        nGrayBits;          //图像数据灰度位数 1 1bit, 2 2bits
    unsigned int    nStartPassIndex;            //起始PASS，用于一层打印的恢复
    unsigned int    nFeatherMode;               //羽化方式  0 不羽化 1~6分别Level->Level6 羽化程度逐步增加
    unsigned int    nCustomFeatherJets;         //自定义羽化喷嘴数
    int             nEdgeScalePixel;            //图像边界缩放 >0 边缘增加像素 <0 边缘减少像素数 =0 边缘不变化
    float           fRotateAngle;               //以图像中心为参考的角度旋转  弧度制
    float           fDstXScale;                 //X向输出缩放系数
    float           fDstYScale;                 //Y向输出缩放系数
    float           fLayerDensity;              //层像素密度  >=1 100%的输出/ 0<密度<1.0: 0~100%的像素抽点输出
    float           fXScanSpd;                  //X扫描速度 mm/s
    unsigned int    nScanCtlValue;              //扫描计算方式 0 PASS逐步叠加  1:喷头重复覆盖+微步进 2
    unsigned int    nImgType;                   //源文件图片类型 0：BMP 1:PRT 2:CLI（启动任务时给定了CLI文件路径，层打印不需要再指定数据）
    unsigned int    nPrtFlag;                   //bit0:  双向打印 bit1: Y反方向打印 bit2:X向插点   bit3: 方向切换依据跳白计算  bit4:边缘不抽点 bit5:喷头覆盖
    long            nReserved[8];
}PRTIMG_LAYER,*LPPRTIMG_LAYER;
```

PassDataItem,*LPPassDataItem

```
//PASS描述
typedef struct tag_PassDataItem
{
    unsigned int    nLayerIndex;        //所属图层号
    unsigned int    nLayerPassCount;    //所属图层PASS总数
    unsigned int    nLayerPassIndex;    //PASS序号
    unsigned int    nValidPassJets;     //PASS有效喷嘴数    20180818 新增
    unsigned int    nProcState;         //图层处理状态 //0 数据处理阶段  1数据处理就绪  2 写数据到硬件  3 数据已写入硬件  4 已添加到打印队列  5打印已结束 6
    int             nMinJet0ImgLinePos; //Y向偏差最小喷头的0号喷嘴相对图像的行位置
    bool            bPrtDir;            //打印方向  0 编码负方向 1正方向
    int             nStpVector;         //打印此PASS前需要的步进量  单位 um
    int             nValidStartCols;    //扫描的X起始数据列，相对扫描正方向第一列，用于X向跳白动作计算
    int             nValidEndCols;      //扫描的X结束数据列，相对扫描正向最后一列，用于X向跳白动作计算
    unsigned int    nPrtCols;           //总打印列数，可以理解为X像素数
    unsigned int    nStartEncPos;       //起始位置 ，编码值
    float           fPrtPrecession;     //打印分频值
    int             nErrorResult;       //出错码
    int             nReverse[32];
    void*           pInnerParam;                    //内控参数地址，
    tag_PassDataItem* pNextItem;
}PassDataItem,*LPPassDataItem;
```

PrtRunInfo,*LPPrtRunInfo

```
//打印运行信息
typedef struct tag_PrtRunInfo
{
    bool            bJobPrtRuning;          //作业打印运行中  true 运行中
    bool            bLayerPrtIsOver;        //当前图层打印完成标志 true 当前图层打印完成
    int             nPrtState;              //板卡打印控制器状态
    int             nPrintLayerIndex;       //当前打印的图层
    int             nLayerPassCount;        //当前打印图层PASS总数
    int             nPrintPassIndex;        //当前打印的PASS
    int             nCurPrtDir;             //当前的打印方向
    unsigned int    nRevPrtCols;            //当前剩余的打印列数
    int             nProcLayerIndex;        //当前数据处理的图层
    int             nDTLayerIndex;          //当前数据传输的图层
    int             nReverse[16];
}PrtRunInfo,*LPPrtRunInfo;
```

PRTJOB_ITEM,*LPPRTJOB_ITEM

```
//任务描述
typedef struct tag_PrtJobItem
{
    unsigned int    nJobID;
    char            szJobName[64];
    char            szJobFilePath[260]; //涉及需处理的3D文件路径 用于CLI1类似文件解析
    unsigned int    nFileType;          //文件类型 0，任务无文件指定，通过图层增加  1，ＣＬＩ文件
    unsigned int    nPrtXEncPos;        //任务的X向起打位置
    float           fPrtYPos;           //保留 离0位位置
    float           fClipWidth;         //打印截图尺寸 Width  unit:mm
    float           fClipHeight;        //打印截图尺寸 Height unit:m
    float           fOutXdpi;           //需要数据解析文件的XDPI
    float           fOutYdpi;           //需要数据解析文件的YDPI
    unsigned int    nOutPixelBits;      //需要数据解析文件灰度指定
    unsigned int    nPrtCtl;            //bit0 跳白支持  bit1 循环喷嘴偏移  bit2 每层起点随机偏移嘴 bit4 X镜像 bit5 Y镜像 bit6 两倍YDPI打印
}PRTJOB_ITEM,*LPPRTJOB_ITEM;
```

RYUSR_SYSPARAM,*LPRYUSR_SYSPARAM

```
typedef struct tag_RYUSR_SYSPARAM
{
    long            nParamVer;          //参数版本
    long            nParamSize;         //参数字节数
    unsigned int    nPhgValidCtl[MAX_COLORS][MAX_GROUP];        //每喷头组打印允许
    int             nPhgXGroupOff[MAX_COLORS][MAX_GROUP][DIR_COUNT];      //喷头组套色偏移 //Dir 0 负方向 1正反向  单位:打印编码器数值
    int             nPhgJetRowOff[MAX_COLORS][MAX_GROUP][MAX_JETROW][DIR_COUNT]; //组内套色偏移 [MAX_JETROW逻辑嘴] 单位:打印编码器数值
    unsigned int    nLackJetCount[MAX_COLORS][MAX_GROUP];           //缺嘴数量
    unsigned int    nLackJetTbl[MAX_COLORS][MAX_GROUP][32];         //缺嘴索引表，最多32个缺嘴
    float           fPrtScanSpd;            //打印扫描速度 mm/s
    PHCTL_PARAM     phctl_param[MAX_PH_CNT];
    colorbar_param  clrbar_param;
    int             nBiDirEncPrtOff;          // 当前打印使用的双向偏差 编码
    unsigned int    nMicroJetUint;          //PASS微动时变化单位 Jet
    unsigned int    nMicroJetCount;         //PASS微动嘴 Jet
    unsigned int    nMicroStpJet;           //20220103 微步进扫描喷嘴数
    unsigned int    nOverlapJetProcType[MAX_COLORS];            //重叠嘴处理方式  0 交叉 1 后喷头舍弃与前喷头重叠部分
    float           fBrustCycleSec[MAX_COLORS];                 //闪喷的周期
    float           fBrustValidSec[MAX_COLORS];                 //闪喷的有效时间
    float           fFlashPrtFrequecy[MAX_COLORS];              //闪喷频率
    unsigned int    nSysFunOption;              //bit0 :待机自动闪喷0数据 bit1:编码计数器反向（由AB相位决定 需要保证X正向计数增大）
    char            szLogPath[260];             //日志文件路径
    unsigned int    nValidColorMask;            //允许输出的颜色掩码
    unsigned int    nIoOption;                  //
    unsigned int    nRevParam[96];              //20211007 20200819 20191101 126->110->109->108->92->91
}RYUSR_PARAM,*LPRYUSR_PARAM;
#endif
```

PRINTER_INFO,*LPPRINTER_INFO

```
//打印机设备信息
typedef struct tag_Printer_Info
{
    unsigned int    nVersion;                       //打印机软件系统版本号
    unsigned int    nCustomIerD;                    //客户号
    unsigned int    nXSysEncDPI;                    //系统光栅DPI
    unsigned int    nMcpCount;                      //网络打印卡数量
    unsigned int    nMaiMcpIndex;                   //网络主卡索引
    unsigned int    nMcpValidMask;                  //网络打印卡卡的掩码
    unsigned int    nMcpDeviceID[MAX_MCP_CNT];      //网络打印卡设备号
    unsigned int    nMcpFpgaVer[MAX_MCP_CNT];       //网络打印卡FPGA版本
    unsigned int    nMcpMcuVersion[MAX_MCP_CNT];    //网络打印卡卡MCU版本
    unsigned int    nMcpDrvLinkCount[MAX_MCP_CNT];  //驱动卡连接数量
    unsigned int    nMcpDrvLinkMask[MAX_MCP_CNT];   //驱动卡连接掩码
    unsigned int    nMcpConfigPhMask[MAX_MCP_CNT];  //每个主卡的喷头配置码
    unsigned int    nSysStatus;             //系统状态
    unsigned int    nPrintStatus;           //0 待机  1打印 2 暂停
    unsigned int    nSysColors;                             //系统配置颜色数
    unsigned int    nMaxFireFreq;                       //最大扫描频率
    unsigned int    nLicenseResult;                     //系统授权码
    unsigned int    nClrPHGroups[MAX_COLORS];           //每个颜色的喷头数
    unsigned int    nCMHomgPosition;                        //车头零位位置   0:在左边 1:在右边
    int             nPhIDLKT[MAX_COLORS][MAX_GROUP];        //喷头的ID查找表   //bit0~bit15: PHID bit16:~bit19 subPhID(1头多色)
    int             nHWInitResult;                          //硬件初始化结果
    DRVINFO         sysDrvInfo[MAX_MCP_CNT][MCP_DRV_CNT];   //驱动卡状态信息
    PrtRunInfo      prt_rtinfo;
}PRINTER_INFO,*LPPRINTER_INFO;
```

DRVINFO,*LPDRVINFO

```
typedef struct tag_DRVINFO
{
    unsigned int    nState;         //连接状态 1 表示已连接
    unsigned int    nNextState;     //连级下个驱动卡连接状态
    unsigned int    nSignature;     //驱动卡序列号
    unsigned int    nPCBVersion;    //驱动卡PCB版本
    unsigned int    nFMVersion;     //驱动卡MCU版本
    unsigned int    nFpgaVersion;   //驱动卡控制程序版本
    unsigned int    nPtvwarnState;  //1喷头不存在 2,8喷头温度超过控制范围
    unsigned int    nPhLinkCount;           //连接的喷头数
    unsigned int    nPhLinkMask;            //喷头连接的通道掩码
    unsigned int    nDrvType;               //驱动卡类型
    PHRUN_INFO      phinfo[4];              //喷头运行信息
    unsigned int    nRevParam[16];
}DRVINFO,*LPDRVINFO;
//打印运行信息
```

RYCalbrationParam,*LPRYCalbrationParam;

```
//校准参数
typedef struct tag_CalbrationParam
{
    unsigned int  nAdjType;   //0 喷嘴状态图 1垂直校准 2步进 3 X喷头组套色 4 X内距套色 5往返套色
    unsigned int  nCtr1Value; //校准控制字;
    float         fxadjdp1;           //校准打印的DPI
    int           nGrayBits;          //采用灰度位数
    int           nPrtDir;            //校准图打印方向 0 负反向 1正方向
    float         fStpSize;           //步进校准单位量，Y向的步进长度，单位mm
    float         fXRunSpd;           //mm/s  车头运行速度
    float         fXMaxPrtWidth;      //设备最大打印宽度
    float         fYMaxPrtHeight;     //设备最大打印高度
}
RYCalbrationParam,*LPRYCalbrationParam;
//用户表参数
```