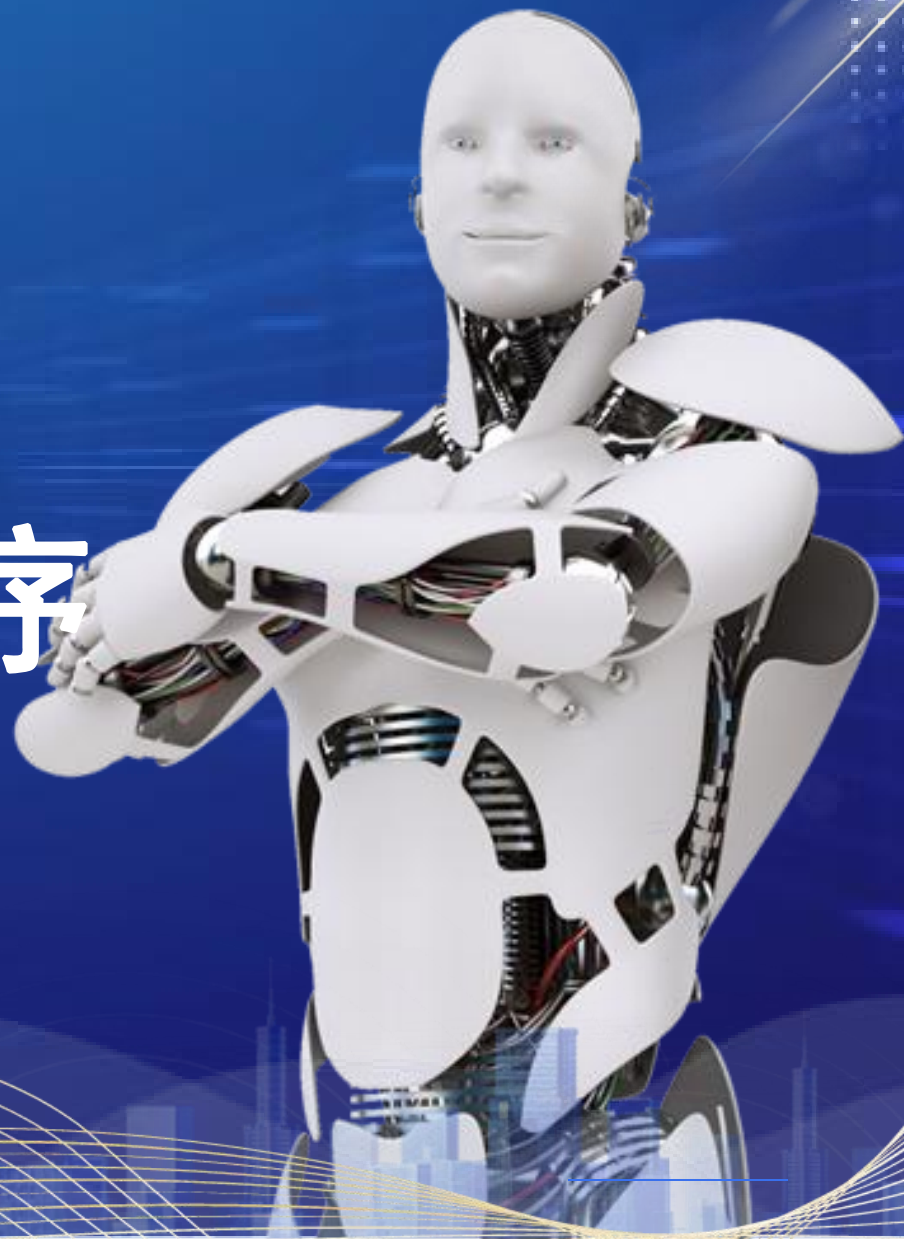
轴运动状态检测程序



01

轴运动状态检测

1. 硬件连接

将运动控制卡、PC机、端子板、驱动器、单轴模组正确连接。

2. 配置驱动器和运动控制器

设置驱动器参数，对运动控制器进行配置，并保存配置文件。

3. 新建控制台项目

在Visual Studio中新建项目工程。

4. 调用库及配置文件

将工程中需要使用的动态链接库、头文件以及控制器配置文件拷贝到项目的源文件目录下。

5. 添加库文件

在项目-属性-链接器-输入-附加依赖项中添加gts.lib库文件；添加库文件的另一种方法是，在程序中使用#pragma comment(lib, "gts.lib")。

01

轴运动状态检测

6. 添加头文件

将代码中需要使用到的指令的头文件包含到程序中。

7. 程序实现

获取轴1的轴状态、运动模式、位置、速度和加速度，代码如下。

```
#define AXIS 1 // 定义轴号
int main(int argc, char* argv[])
{
short bFlagAlarm = false; // 伺服报警标志
short bFlagMError = false; // 跟随误差越限标志
short bFlagPosLimit = false; // 正限位触发标志
short bFlagNegLimit = false ; // 负限位触发标志
short bFlagSmoothStop = false; // 平滑停止标志
short bFlagAbruptStop = false; // 急停标志
short bFlagServoOn = false; // 伺服使能标志
short bFlagMotion = false; // 规划器运动标志
double dPrfPos; // 规划位置
double dPrfVel; // 规划速度
double dPrfAcc; // 规划加速度
```

01

轴运动状态检测

```
long lPrfMode; // 运动模式
char chPrfMode[20]; // 运动模式，字符串变量
short sRtn; // 指令返回值变量
long lAxisStatus; // 轴状态
// 打开运动控制器
sRtn = GT_Open();
// 指令返回值检测
commandhandler("GT_Open", sRtn);
// 系统复位
sRtn = GT_Reset();
commandhandler("GT_Reset", sRtn);
// 读取轴状态
sRtn = GT_GetSts(AXIS, &lAxisStatus);
commandhandler("GT_GetSts", sRtn);
// 伺服报警标志
if (lAxisStatus& 0x2)
{
bFlagAlarm = true;
printf("伺服报警\n");
}
else
{
bFlagAlarm = false;
printf("伺服正常\n");
}
// 跟随误差越限标志
if (lAxisStatus& 0x10)
{
bFlagMError = true;
printf("运动出错\n");
}
else
{
bFlagMError = false;
printf("运动正常\n");
}
```

01

轴运动状态检测

```
// 正向限位
if (lAxisStatus& 0x20)
{
bFlagPosLimit = true;
printf("正限位触发\n");
}
else
{
bFlagPosLimit = false;
printf("正限位未触发\n");
}
// 负向限位
if (lAxisStatus& 0x40)
{
bFlagNegLimit = true;
printf("负限位触发\n");
}
else
{
bFlagNegLimit = false;
printf("负限位未触发\n");
}
// 平滑停止
if (lAxisStatus& 0x80)
{
bFlagSmoothStop = true;
printf("平滑停止触发\n");
}
else
{
bFlagSmoothStop = false;
printf("平滑停止未触发\n");
}
```

01

轴运动状态检测

```
// 急停标志
if (lAxisStatus& 0x100)
{
bFlagAbruptStop = true;
printf("急停触发\n");
}
else
{
bFlagAbruptStop = false;
printf("急停未触发\n");
}
// 伺服使能标志
if(lAxisStatus& 0x200)
{
bFlagServoOn = true;
printf("伺服使能\n");
}
else
{
bFlagServoOn = false;
printf("伺服关闭\n");
}
// 规划器正在运动标志
if (lAxisStatus& 0x400)
{
bFlagMotion = true;
printf("规划器正在运动\n");
}
else
{
bFlagMotion = false;
printf("规划器已停止\n");
}
```

01

轴运动状态检测

```
// 读取运动数据
sRtn = GT_GetPrfPos(AXIS, &dPrfPos);
commandhandler("GT_GetPrfPos", sRtn);
printf("规划位置 %8.2f\n", dPrfPos);
sRtn = GT_GetPrfVel(AXIS, &dPrfVel);
commandhandler("GT_GetPrfVel", sRtn);
printf("规划速度 %8.2f\n", dPrfVel);
sRtn = GT_GetPrfAcc(AXIS, &dPrfAcc);
commandhandler("GT_GetPrfAcc", sRtn);
printf("规划加速度 %8.2f\n", dPrfAcc);
// 读取运动模式
sRtn = GT_GetPrfMode(AXIS, &lPrfMode);
commandhandler("GT_GetPrfMode", sRtn);
// 清空字符串
memset( chPrfMode, '\0', 20);
switch(lPrfMode)
{
case 0:
sprintf(chPrfMode, "Trap");
break;
case 1:
sprintf(chPrfMode, "Jog");
break;
case 2:
sprintf(chPrfMode, "PT");
break;
case 3:
sprintf(chPrfMode, "Gear");
break;
case 4:
sprintf(chPrfMode, "Follow");
break;
case 5:
sprintf(chPrfMode, "Interpolation");
break;
```

01

轴运动状态检测

```
case 6:
sprintf(chPrfMode, "PVT");
break;
default:
break;
}
printf("运动模式 %s\n", chPrfMode);
return true;
}
```

谢　谢　观　看