

# PVTContinuous

## 功能说明

### 文件版本

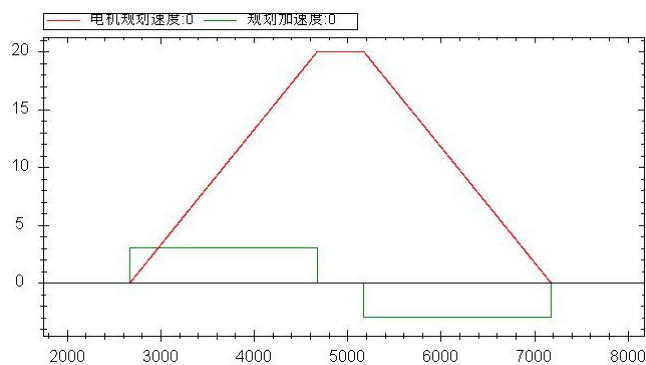
| 版本  | 时间         | 说明 |
|-----|------------|----|
| 1.0 | 2025.04.10 | 初稿 |

## 功能说明及使用场景

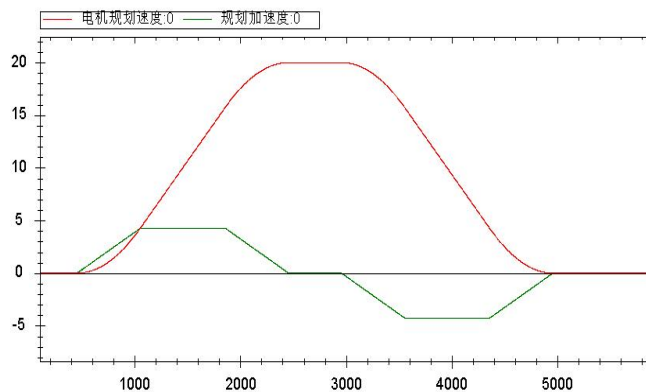
PVTContinuous 模式是 PVT 的一种变形，要求输入各个点的“位置/速度/最大速度/加速度/减速度/百分比”参数。控制模块根据各个点的参数将运动过程分解为连续的点，然后再将各个点按照百分比参数转换为等效的 S 段。

可以比较方便地让点位运动转换为对应的 PVT 运动，而不需要用户做复杂计算，既可实现比较平滑的规划。可以用于点位运动的性能优化场合。

比如下面是单点运动的规划曲线对比(图线中加速度被放大 300 倍)，运动距离为 50000，最大速度为 20pulse/ms,加速度和减速度都是 0.01:



梯形规划



PVTContinuous (percent=60)

## 接口及说明

相关的功能接口如下:

### 接口定义及说明

// 按照 continues 的方式计算 PVT 各个数据点，并将计算结果返回

```
// count:数据点个数
// pPos:输入的位置数组
// pVel:输入的速度数组
// pPercent:输入的数据点百分比数组, [0,100]
// pVelMax:输入的最大速度数组
// pAcc:输入的加速度数组
// pDec:输入的减速度数组
// pResultDataCnt:计算得到的数据点个数
// pResultPvtPArray:计算得到的位置数组
// pResultPvtVArray:计算得到的速度数组
// pResultPvtTArray:计算得到的时间数组
MTN_API short __stdcall OEM_PvtContinuousCalc(Int32 count, double* pPos, double* pVel,
double* pPercent, double* pVelMax, double* pAcc, double* pDec, Int16* pResultDataCnt,
double* pResultPvtPArray, double* pResultPvtVArray, double* pResultPvtTArray);

// 按照 continues 的方式计算 PVT 各个数据点, 并将计算结果压入 PVT 缓冲区
// pPos: 相对当前位置的移动位置, 单位:pulse
MTN_API short __stdcall OEM_PvtContinuousData(HAND axisHandle, Int32 count, double* pPos,
double* pVel, double* pPercent, double* pVelMax, double* pAcc, double* pDec);

// 按照 continues 的方式计算 PVT 各个数据点, 并将计算结果压入 PVT 缓冲区, 单点模式
MTN_API short __stdcall OEM_PvtContinuousDataPt(HAND axisHandle, double relPos, double
velMax, double acc, double dec, double percent);
```

说明:

1. 提供了三条相关的指令, 实际上都是一种计算的实现。

OEM\_PvtContinuousCalc 仅提供计算功能, 用户输入要求的点参数后, 将最终计算出来的 PVT 点列表返回给用户。

OEM\_PvtContinuousData 则是完成计算后, 直接将数据下发给控制器了。对于用户的输入点数较大时, 则可能计算出来的结果点数大于了控制器 PVT 的缓冲区最大值, 这个时候, 则不能使用该接口。

OEM\_PvtContinuousDataPt 是单点的实现。

运行要求如下:

| 项   | 要求            |
|-----|---------------|
| 控制器 | 无要求           |
| 固件  | 要求支持 PVT 运动功能 |
| 驱动  | 无要求           |
| 库   | 需要更新支持的库      |

# 例程及说明

下面的例程演示该功能的使用：

```
void Test1()
{
    short ax = 0;

    // 1. 将轴配置为 PVT 模式
    rtn = NMC_MtSetPrfMode(g_hAx[ax], MT_PVT_PRF_MODE);
    if (rtn != 0) { AddLog(_T("NMC_MtSetPrfMode 错误"), rtn); return; }

    // 2. 设置 PVT 运动的参数
    TPvtPara pvtPrm;
    pvtPrm.abruptDec = 1;
    pvtPrm.smoothDec = 2;
    pvtPrm.dataMode = 1;
    pvtPrm.loopCount = 0;
    rtn = NMC_MtPvtSetPara(g_hAx[ax], &pvtPrm);
    if (rtn != 0) { AddLog(_T("NMC_MtPvtSetPara 错误"), rtn); return; }

    // 3. 清除缓冲区
    rtn = NMC_MtPvtBufClr(g_hAx[ax]);
    if (rtn != 0) { AddLog(_T("NMC_MtPvtBufClr 错误"), rtn); return; }

    // 4. 准备数据, 点数较少, 直接调用 OEM_PvtContinuousData 接口
    short count = 3;
    double pvtCPos[8] = { 0, 19800, 21800 };
    double pvtCVel[8] = { 0, 2, 0 };
    double pvtCPercent[8] = { 60, 60, 0 };
    double pvtCVelMax[8] = { 10, 2, 2 };
    double pvtCAcc[8] = { 0.01, 0.02, 0.02 };
    double pvtCDec[8] = { 0.01, 0.02, 0.02 };
    rtn = OEM_PvtContinuousData(g_hAx[ax], count, pvtCPos, pvtCVel,
        pvtCPercent, pvtCVelMax, pvtCAcc, pvtCDec); if (rtn != 0) {
        AddLog(_T("OEM_PvtContinuousData 错误"), rtn);
        return;
    }

    // 5. 启动 PVT 运动
    rtn = NMC_MtPvtStartMtn(g_hAx[ax], 0, NULL);
    if (rtn != 0) { AddLog(_T("NMC_MtPvtStartMtn 错误"), rtn); return; }
}
```

运行速度及加速度曲线如下图所示：

