

安川 7 代码解析

目录

一、电流环.....	2
1.1 反馈电流计算.....	3
1.2 相位计算及补偿.....	6
1.2.1 电角度的计算.....	7
1.2.2 基于时间补偿的相位角度.....	7
1.2.3 基于最大速度转换的相位角补偿.....	11
1.3 电流观测器.....	15
1.4 弱磁控制.....	20
1.5 D 轴电流环计算.....	33
1.6 q 轴电流环.....	35
1.7 电压补偿.....	42
1.8 过调制.....	48
1.9 死区补偿.....	54
1.10 转矩给定滤波器.....	62
1.10.1 陷波滤波器.....	62
二、速度环.....	67
2.1 PI 控制.....	67
2.1.1 典型二型系统的设计.....	70
2.1.2 转速调节器.....	71
2.1.3 饱和作用.....	72
2.1.4 P-PI 切换.....	73
2.2 速度环输出低通滤波器.....	75
2.3 速度脉动补偿.....	80
2.4 摩擦补偿.....	81
2.5 速度反馈计算.....	85
2.5.1 相位补偿速度观测器.....	85
2.6 摩擦补偿（扰动观测器）.....	88
2.6.1 扰动观测器原理分析.....	89
2.7 A 型振动抑制（中频振动抑制）.....	92
三 位置环.....	93
3.1 位置指令.....	93
3.1.1 低频振动抑制.....	95
3.2 位置环计算.....	99
四、重要功能模块.....	100
4.1 模型追踪控制.....	101
4.2 基本中间参数计算处理.....	105
4.3 最大电流的计算.....	105

4.4 标么化基值选取.....	105
4.5 免调整功能.....	106

一、电流环

电流环采用了零极点对消 PI 整定，电压方程解耦，弱磁，死区补偿，相位补偿，过调制，滤波器，电流观测器等技术。

电流环参数 PI 整定，基本是根据电流环数学模型，将零极点最大时间常数进行对消，简化划为I型系统[1]

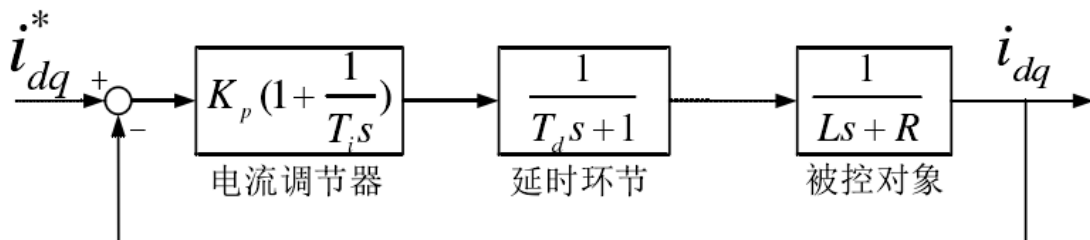
$$W(s) = \frac{K}{s(Ts + 1)}$$

其闭环传递函数可写成：

$$W_c(s) = \frac{W(s)}{W(s) + 1} = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}$$

其中 $\omega_n = \sqrt{\frac{K}{T}}$ 固有角频率；

$\xi = \frac{1}{2}\sqrt{\frac{1}{KT}}$ 阻尼比或称衰减系数。



则电流环开环传递函数为：

$$G(s) = \frac{K_p K_m (T_i s + 1)}{T_i s (T_d s + 1) (T_m s + 1)}$$

式中 K_m 和 T_m 分别为电机电枢回路的增益和电气时间常数， $K_m=1/R$ ， $T_m=L/R$ ；由于 $T_d \ll T_m$ ，采用零极点对消，取积分时间常数为 T_m 。

$$\begin{cases} T_i = T_m = \frac{L}{R} \\ K_p = L\omega_c \end{cases}$$

ω_c 为截止频率，只需设置截止频率，就可得到电流环控制器初始参数，然后给定周期性的激励指令进行堵转，根据评价指标，进行迭代，选出最优截止频率。

电流环在 `MpIntHost(void)` 中处理。

1.1 反馈电流计算

反馈电流取值在此函数中实现 `inline void ADConvDataLoad(MICRO_AXIS_HANDLE *AxisRsc)`。

计算方式为

```
IntAdV.IuInData = IntAdP.Kcu * ( IUS + IntAdV.IuOffset ) / 2^8
IntAdV.IvInData = IntAdP.Kcv * ( IVS + IntAdV.IvOffset ) / 2^8
```

Kcu 为采样系数, IuOffset 为零漂检测值, IuInData 为电流标么化值

```
int chess_storage(PFREG:0x6D0) IuAD;
swk = mulshr(IuAD, ONE, 2);
AxisRsc->IntAdV.IuInData = mulshr((swk + AxisRsc->IntAdV.IuOffset),
AxisRsc->IntAdP.Kcu, 8 );
```

IuAD 为寄存器的值, AD 采样不知道是否为 16 位采样, IuOffset, Kcu 有在 `void MpIntHost(void)` 中处理

```
AxisRscH->IntAdV.IuOffset = AxisRscH->AdinV.IuOffsetIn;
AxisRscH->IntAdV.IvOffset = AxisRscH->AdinV.IvOffsetIn;
AxisRscH->IntAdP.Kcu = AxisRscH->AdinV.KcuIn;
AxisRscH->IntAdP.Kcv = AxisRscH->AdinV.KcvIn;
```

`static LONG LpxLcdCurrentManAdjExe(FUNMSG *Fmsg) FnMotorCurrentAdj.c`
零漂检测函数, KcuIn 类似于功能码进行设置。

```
AxisRscH = &AxisHdl[ax_noH];
```

```
AxisRscH->IntAdV.IqMon = AxisRscH->IntAdV.IqRef; /* for CPU monitor */
/*-----*/
/*      キャリア周波数切り替え処理  载波频率切换处理      <V057> <V075> */
/*-----*/
if ( AxisRscH->IntAdP.CrFreq != AxisRscH->IntAdV.CrFreqW )
{
    AxisRscH->IntAdV.CrFreqW = AxisRscH->IntAdP.CrFreq; /* Carrier Buffer Change*/
    AxisRscH->SvIpRegW->CRFRQ = AxisRscH->IntAdV.CrFreqW; /* Carrier Freq. Change */
}
}
```

此段代码用于 q 轴电流显示, 载波频率改变调节。

```
/*-----*/
/*      input from host */
/*-----*/
/* Check Current Ajust Request */
ActiveAxis = 0;
#ifdef MULTI_AXIS /* 多轴处理有效 */
    for( ax_noH = 0; (SHORT)ax_noH < AxisNum; ax_noH++ )
#else
    //ifdef MULTI_AXIS
```

```

    ax_noH = 0;
#endif          // #ifdef MULTI_AXIS
    {
        AxisRscH = &AxisHdl[ax_noH];
//<2>#ifdef PREG_DEF
//          if ( ( CTSTR & RLOCK ) == 0 )
            if ( ( AxisRscH->CtrlStsIn & RLOCK ) == 0 )
            {
                ActiveAxis |= 0x01 << ax_noH; /* ビット登録 判断是否需要零漂检测 */
            }
        }

    if( ActiveAxis != 0 )
    { /* 電流検出調整要求あり 若零漂検出更新相关变量值 */
        /* ★H/Wアクセスが共通のものをまとめたい！！0軸目って書くのが格好悪い★ */
        DIX = 0x0;          /* disable interrupt          <V112>          */

#ifdef MULTI_AXIS          /* 多軸処理有効          */
        for( ax_noH = 0; (SHORT)ax_noH < AxisNum; ax_noH++ )
#else          // #ifdef MULTI_AXIS
        ax_noH = 0;
#endif          // #ifdef MULTI_AXIS
        {
            AxisRscH = &AxisHdl[ax_noH];

            if( 0 != (ActiveAxis & (0x01 << ax_noH)) )
            {
                AxisRscH->IntAdV.IuOffset = AxisRscH->AdinV.IuOffsetIn; /* IntAdV.IuOffset
<-- AdinV.IuOffsetIn          */
                AxisRscH->IntAdV.IvOffset = AxisRscH->AdinV.IvOffsetIn; /* IntAdV.IvOffset
<-- AdinV.IvOffsetIn          */
                AxisRscH->IntAdP.Kcu = AxisRscH->AdinV.KcuIn;          /* IntAdP.Kcu <--
AdinV.KcuIn          */
                AxisRscH->IntAdP.Kcv = AxisRscH->AdinV.KcvIn;          /* IntAdP.Kcv <--
AdinV.KcvIn          */
            }
        }

        /* ★H/Wアクセスが共通のものをまとめたい！！0軸目って書くのが格好悪い★ */
        EIX = 0x0;          /* enable interrupt          <V112>          */
    }
}

```

此段程序目的在于零漂检测判断条件，更新相关变量值

```

#define RLOCK          0x0008          /* bit.3 : Transer register lock status */
USHORT zadjiu;          /* PnE50 : 電流検出ゼロ調(U相) */
USHORT zadjiv;          /* PnE51 : 電流検出ゼロ調(V相) */
DBYTEX gadjiu_v;        /* PnE52 : 電流検出ゲイン調(U,V相) */

/*-----*/
/*      電流検出ゼロ調      */
/*-----*/
/* U相電流ゼロ調 U相电流调零 零漂检测幅值 集成芯片处理零漂*/
kx = (SHORT)(uCfgPrm->zadjiu);
rc |= LpxSetPrmToASIC( &(pAsicMicroReg->AdinV.IuOffsetIn), kx, NO_LMT_CHK );
/*-----*/
/* V相電流ゼロ調 */
kx = (SHORT)(uCfgPrm->zadjiv);
rc |= LpxSetPrmToASIC( &(pAsicMicroReg->AdinV.IvOffsetIn), kx, NO_LMT_CHK );

/*-----*/
/*      電流検出ゲイン調  电流检测增益调整      */
/*-----*/
/* U相電流ゲイン調 */
kx = (CHAR)uCfgPrm->gadjiu_v.b.l + 0x100;
rc |= LpxSetPrmToASIC( &(pAsicMicroReg->AdinV.KcuIn), kx, NO_LMT_CHK );
/*-----*/
/* V相電流ゲイン調 */
kx = (CHAR)uCfgPrm->gadjiu_v.b.h + 0x100;
rc |= LpxSetPrmToASIC( &(pAsicMicroReg->AdinV.KcvIn), kx, NO_LMT_CHK );

零漂检测以及电流采样系数 PnE67 均有集成芯片完成，其检测值不知道，采样系数不知道，但
做法应该是一样的，根据不同的功率等级，选择采样电阻，上电初始化进行零漂检测或者伺服
OFF 再进行一次检测提高精度要求。

AxisRscH = &AxisHdl[ax_noH];
AxisRscH->PhaseV.PhaseH = AxisRscH->AdinV.PhaseHIn;          /* */
AxisRscH->PhaseV.PhaseIp = AxisRscH->PhaseV.PhaseIpIn;      /* 位相補間量 <V112> */
AxisRscH->PhaseV.PhaseIpF = AxisRscH->PhaseV.PhaseIpFIn;    /* 位相補間フラグ <V112> */
AxisRscH->PhaseV.PhaseIpFIn = 1;                             /* 位相補間フラグセット<V112> */
AxisRscH->WeakFV.Vel = AxisRscH->AdinV.VelIn;               /* */
AxisRscH->IntAdV.TLimP = AxisRscH->AdinV.TLimPIn;          /* */
AxisRscH->IntAdV.TLimM = AxisRscH->AdinV.TLimMIn;          /* */
AxisRscH->IntAdP.Kvv = AxisRscH->IntAdP.KvvIn;              /* for AVR */
AxisRscH->VcmpV.VdRef = AxisRscH->AdinV.VdRefIn;           /* */
AxisRscH->VcmpV.VqRef = AxisRscH->AdinV.VqRefIn;           /* */
AxisRscH->IntAdV.IqDist = AxisRscH->IntAdV.IqDistIn;        /* <V224> */
AxisRscH->WeakFV.WfKpV.s[0] = AxisRscH->WeakFV.WfKpVLIIn; /* 電圧FB比例ゲイン(下位16bit)
<V214> */
AxisRscH->WeakFV.WfKpV.s[1] = AxisRscH->WeakFV.WfKpVHIIn; /* 電圧FB比例ゲイン(上位
16bit) <V214> */

```

```

AxisRscH->WeakFV.WfKiV.s[0] = AxisRscH->WeakFV.WfKiVLIn; /* 電圧FB積分ゲイン(下位
16bit) <V214> */
AxisRscH->WeakFV.WfKiV.s[1] = AxisRscH->WeakFV.WfKiVHIn; /* 電圧FB積分ゲイン(上位
16bit) <V214> */
AxisRscH->WeakFV.WfV1Max = AxisRscH->WeakFV.WfV1MaxIn; /* 電圧指令制限値<V214> */
AxisRscH->WeakFV.WfldRefLim = AxisRscH->WeakFV.WfldRefLimIn; /* d 軸電流指令リミット
<V214> */

```

电流环相关变量的赋值处理

1.2 相位计算及补偿

```

//=====
// 位相補間処理 <V112>
//=====

swk10 = AxisRscI->PhaseV.PhaseH + AxisRscI->PhaseV.PhaseIp;
AxisRscI->PhaseV.PhaseIpF = cmove((AxisRscI->PhaseV.PhaseIpF != 1), ONE,
AxisRscI->PhaseV.PhaseIpF);
AxisRscI->PhaseV.PhaseH = cmove((AxisRscI->PhaseV.PhaseIpF != 1), AxisRscI->PhaseV.PhaseH,
swk10);

```

相位角的计算进行补偿处理，根据其时序图，位置反馈周期为 31.25us，电流采样的周期为 15.625us，一倍的关系，中间需要插值补一次上次的偏差值，作为角度。PhaseIpF=0 表示不需要补，=1 表示需要插值一次，PhaseIp 为插值角度，PhaseH 为此次计算的电角度，65536 对应 360°

```

AxisRscR->PhaseV.PhaseH = AxisRscR->AdinV.PhaseHIn;
AxisRscR->PhaseV.PhaseIp = AxisRscR->PhaseV.PhaseIpIn;
AxisRscR->PhaseV.PhaseIpF = AxisRscR->PhaseV.PhaseIpFIn;
AxisRscR->PhaseV.PhaseIpFIn = 1;

```

PhaseH 的计算在包括 void MplntHost(void) 的 31.25us 中断中进行赋值

```

/* 相順:UVW */
SvAsicRegs->AsicMicroReg->AdinV.PhaseHIn = MencV->MotPhaseX;
/* 位相補間量 */
SvAsicRegs->AsicMicroReg->PhaseV.PhaseIpIn = PhaseComp->var.PcmpdItplt;
/* 位相補間フラグ */
SvAsicRegs->AsicMicroReg->PhaseV.PhaseIpFIn = 0;

```

PhaseHIn 的赋值在 void MicrolfOutputCycData(BASE_LOOPCTRLS *BaseLoops, CTRL_LOOP_OUT *CtrlLoopOut, ASICS *SvAsicRegs)子函数中，此函数在 SCANA 中进行。MotPhaseX的赋值在子函数static void BaseLoopMakePhaseCompValue()中进行计算，为此函数 KpxOutputScanA () 的子函数。

```

MotSpd = BaseLoops->MotSpd * Bprm->DirSign; /* Motor Speed [2^24/OvrSpd] */
/* 位相補償(遅れ1) */
PhaseCmp->var.Pcmpd = (SHORT)MlibMulgainNolim( MotSpd, PhaseCmp->conf.Kpcmpd );
MencV->MotPhaseX = MencV->MotPhase + PhaseCmp->var.Pcmpd + PhaseCmp->var.Pcmps;

```

Q15 格式，代表 360° 或者 2π . MotPhase 反馈位置的差分得到当前电角度，Pcmpd 为基于时间的补偿相位角，Pcmps 为基于最大速度对应补偿角的比例调节。

1.2.1 电角度的计算

```
MotPosX = ( SHal_GetCurMotPos( pAsicHwReg ) << MencV->MposSftR );
```

从寄存器读取当前的电机位置进行左移 MposSftR 位，MposSftR 是用来 pulse 转换为 Q32 格式用

```
MencV->MotPosX[1] = MencV->MotPosX[0];          /* 前回補正後位置データ */
```

```
MencV->MotPosX[0] = MotPosX;                    /* 今回補正後位置データ */
```

```
MencV->MotPhase = (USHORT)(( (((MencV->MotPosX[0]) - (MencV->MotOrgX)) >>8) *  
MencV->Kmotphase)>>8);
```

在此函数中 void SencReadPosition ()，这个函数又在 KpxInputScanA () 中，MotOrgX 为初始角度，MotPosX 的单位应该是 pulse。

在 SencInitParamCalcForLinear () 函数中有关于 Kmotphase 的计算

```
/*-----*/  
/*      モータ磁極位相演算係数の計算      */  
/*-----*/  
/*      */  
/*      65536      PulseNo : [pulse/360deg]      */  
/*      Kmotphase = -----      MotPhase [65536/360deg]      */  
/*      PulseNo      */  
/*      */  
/*-----*/  
#if (FLOAT_USE==TRUE)  
    MencV->Kmotphase = 65536.0f / (float)MencV->PulseNo;  
    MencV->Kinvphase = (float)MencV->PulseNo / 65536.0f;  
#else  
    MencV->Kmotphase = MlibScalKxgain( 65536, 1, MencV->PulseNo, NULL, 24 );  
    MencV->Kinvphase = MlibScalKxgain( MencV->PulseNo, 1, 65536, NULL, 24 );  
#endif /* FLOAT_USE */
```

其目的在于将 pulses 电角度转化为 65536/360deg

1.2.2 基于时间补偿的相位角度

```
kx = MlibScalKxgain( C2PAIE7, 1, MencV->PulseNo, &sx, 0 );
```

```
Bprm->Kmotpls = MlibPcalKxgain( kx, 1, C10POW7, &sx, -1 );
```

Kmotpls 单位为 rad/pulse 用来将 pulse/s 转换为 rad/s，其值为 $2\pi/p$, p 为一圈脉冲数

```
/*-----*/  
/*      最大速度[rad/s]の計算      */  
/*-----*/  
/*      */
```

```

/*          2*PAI * MAXVEL * 100 * (10000+PerOvrSpd)  MAXVEL      : [100r/min] */
/*  OvrSpd = -----  PerOvrSpd : [0.01%] */
/*          60 * 10000                                     */
/*                                                     */
/*-----*/
kx = MlibScalKxgain( Bprm->MaxVel, C2PAIE7, C10POW7, &sx, 0 );
kx = MlibPcalKxgain( kx, (Bprm->PerOvrSpd + 10000), 6000, &sx, 0 );
Bprm->OvrSpd = MlibPcalKxskx( kx, Bprm->ExpSpd, 1, &sx, -1 );
此为最大速度 OvrSpd 计算，将 r/m 转变为 rad/s
上述在此函数 BprmRotaryMotorParamC() 中执行
/*-----*/
/*  モータ速度演算ゲイン中間パラメータの計算          Rotary          Linear*/
/*-----*/
/*                                                     */
/*          Kmotpls * 2^24          Kmotpls : [rad/pulse]          [m/pulse] */
/*          Kmotspd = -----  OvrSpd :      [rad/s]          [m/s] */
/*          OvrSpd                                     */
/*                                                     */
/*-----*/
Bprm->Kmotspd = MlibScalKxskx( 0x1000000, Bprm->Kmotpls, Bprm->OvrSpd, &sx, -1 );
HprmCommonMotorParamCal( BPRMDAT *Bprm )进行标么化处理，将速度转换为 Q 格式，最大
速度对应 Q24 格式。
/*-----*/
/*          モータ速度演算ゲインの計算          */
/*-----*/
/*                                                     */
/*          Kmotpls * 2^24          1000000          */
/*          KmotspdX = ----- * -----[2^24/OvrSpd/dMotPos] */
/*          OvrSpd          ScanXcycle [us]          */
/*                                                     */
/*-----*/
Bprm->KmotspdA = MlibScalKskxkx( Bprm->Kmotspd, 1000000000, KPI_SACYCLEN, NULL, 24 );
Bprm->KmotspdB = MlibScalKskxkx( Bprm->Kmotspd, 1000000, Bprm->SvCycleUs, NULL, 24 );
Bprm->KmotspdC = MlibScalKskxkx( Bprm->Kmotspd, 1000000, KPI_SCCYCLEUS, NULL, 24 );
BprmCalcBaseParameters 将 pulse/scanA 转换为 rad/s 的标么化格式。
MotSpd = MlibMulgain( dMotPos, Bprm->KmotspdA ); /* Motor Speed[2^24/OvrSpd] */
/*-----*/
/*          速度補正有効判定          <S067>          */
/*-----*/
if( MencV->SpdCmpEnable == FALSE )
{
    MencV->MotSpd = TimeBasesMotSpdCalc( Vfbtimbase, MencV, dMotPos, MotSpd,
    VcompSts, FALSE );
    MencV->SpdCmpEnable = TRUE;
}

```



```

}
else
{
    MencV->MotSpd = TimeBasesMotSpdCalc( Vfbtimbase, MencV, dMotPos, MotSpd,
    VcompSts, TRUE );
}

/*-----*/
/* 速度検出移動平均フィルタ処理          <V110> */
/*-----*/
if( BaseLoops->MotSpdMafil->manumBit == 0 )
{ /* エンコーダ17bit以上 */
    MotSpd = MencV->MotSpd;
}
else
{ /* エンコーダ16bit以下 */
    MotSpd = LpxSpdDetMaFilter( BaseLoops->MotSpdMafil, MencV->MotSpd );
}
BaseLoops->MotSpd = MotSpd; /* Motor Speed for ScanB [2^24/OvrSpd] */
速度做了补正处理，且可选择经过滑动平均滤波处理，作为反馈计算。

MotSpd = BaseLoops->MotSpd * Bprm->DirSign; /* Motor Speed [2^24/OvrSpd] */
/* 位相補償(遅れ1) */
PhaseCmp->var.Pcmpd = (SHORT)MlibMulgainNolim( MotSpd, PhaseCmp->conf.Kpcmpd );
/* 位相補償(遅れ2) */
PhaseCmp->var.Pcmpd2 = (SHORT)MlibMulgainNolim( MotSpd, PhaseCmp->conf.Kpcmpd2 );
/* 位相補間量の計算 */
PhaseCmp->var.PcmpdItplt = (SHORT)MlibMulgainNolim( MotSpd, PhaseCmp->conf.KpcmpdItplt );
BaseLoopMakePhaseCompValue()进行了相位补偿的计算
/*-----*/
/*
/*
/*          Dx * OvrSpd * Pole * 65536          Dx * OvrSpd * Pole*/
/*  SVOS  Kpcmpd = ----- = -----*/
/*  Rotary      10^6 * 2 * PAI * 2 * 2^24 * Cx      10^6 * 3216.990877 * Cx*/
/*
/*-----*/
/*
/*          */
/*  PerOvrSpd : OS検出レベル [0.01%]          */
/*  Pole      : モータポール数                  */
/*  nos       : OS速度 [r/min],[mm/s]          */
/*  OvrSpd    : OS速度 [rad/s],[m/s]          */
/*  Dx        : 遅れ時間 [us] --> カーネル構成定義(*.cfg)で指*/
/*  Cx        : バグ補正 [--] --> カーネル構成定義(*.cfg)で指定 */
/*  注意(Cx=16) : モータが本条件に合わせてある場合は、バグ修正不可、リニアはバグな*/

```

```

/*                                                                 */
/*-----*/
#define SVCDEF_PCOMP_DLYUS          ( 80 )    /* 位相遅れ補償遅れ時間(エンコー
ダ:50us+ScanA:30us)    */
#define SVCDEF_PCOMP_DLYCX          ( 16 )          /* 位相遅れ補償補正係数
[-]                    */
if( Axis->MencV->AxisMotType == MOTTYPE_ROTARY )
{
    if( ((Prm->dq_sw & 0x0800) == 0 ) || ((Prm->MencP.flg_wf & 0x0001 ) == 0) )
    {
        #if (FLOAT_USE==TRUE)
            fx = Bprm->OvrSpd * (LONG)SVCDEF_PCOMP_DLYUS / 321699088.0f;
            Axis->BaseControls->PcmpCalcData.Kpcmpd = fx * (float)Prm->MencP.oslv_pol.b.h
/ (10.0f * (LONG)SVCDEF_PCOMP_DLYCX);
        #else
            kx = MlibScalKskxkx( Bprm->OvrSpd, (LONG)SVCDEF_PCOMP_DLYUS, 321699088,
&sx, 0 );
            Axis->BaseControls->PcmpCalcData.Kpcmpd =
                MlibPcalKxgain( kx, (LONG)Prm->MencP.oslv_pol.b.h, (10 *
(LONG)SVCDEF_PCOMP_DLYCX), &sx, 24 );
        #endif /* FLOAT_USE */

    }
    else
    {
        #if (FLOAT_USE==TRUE)
            fx = Bprm->OvrSpd * (float)SVCDEF_PCOMP_DLYUS / 321699088.0f;
            Axis->BaseControls->PcmpCalcData.Kpcmpd = fx * (float)Prm->MencP.oslv_pol.b.h
/ 10.0f;
        #else
            kx = MlibScalKskxkx( Bprm->OvrSpd, (LONG)SVCDEF_PCOMP_DLYUS, 321699088,
&sx, 0 );
            Axis->BaseControls->PcmpCalcData.Kpcmpd =
                MlibPcalKxgain( kx,
(LONG)Prm->MencP.oslv_pol.b.h, 10, &sx, 24 );
        #endif /* FLOAT_USE */

    }

    Axis->BaseLoops->PhaseComp.conf.Kpcmpd = Axis->BaseControls->PcmpCalcData.Kpcmpd;
    Axis->BaseLoops->PhaseComp.conf.Kpcmpd2 = Axis->BaseControls->PcmpCalcData.Kpcmpd2;
    Axis->BaseLoops->PhaseComp.conf.Kpcmpsp1 = Axis->BaseControls->PcmpCalcData.Kpcmpsp1;
    Axis->BaseLoops->PhaseComp.conf.PcmpSpd1 = Axis->BaseControls->PcmpCalcData.PcmpSpd1;
    Axis->BaseLoops->PhaseComp.conf.KpcmpdItplt=Axis->BaseControls->PcmpCalcData.KpcmpdItplt;
}

```

$Dx/10^6$ 为延迟时间 us 转换为 s, $OvrSpd/2^{24}$ 为速度标么值转换为实际值 rad/s, $Pole/2$ 为电机极对数, 将机械角速度转换为电角速度, $65536/2\pi$ 为 2π 转换为标么值。 Cx 为相位延迟补偿更正系数, **某些情况下**为延迟时间的 $1/16$, 则延迟补偿时间实际为 $5\mu s$, Dx 为 $80\mu s$, 其中编码器延迟 $50\mu s$, $ScanA$ 延迟为 $30\mu s$, 这个不知道怎么来的。

总体来说相位补偿是基于

$$\theta' = \theta + \omega t$$

θ 为编码器测得的电角度, ω 为当前校正滤波后的电角速度, t 为延迟时间, 中间经过了大量的单位转换以及标么化处理。相位补偿能提高矢量控制系统在高速工况下的电流控制精度, 增强电机控制系统动态性能。随着电机速度的升高, 功率因数恶化, 此方法可显著提高功率因数。

1.2.3 基于最大速度转换的相位角补偿

Pcmps 的补偿

```

/*-----*/
-----*/
/*      Calculate Pcmps[65536/360deg]
      */
/*-----*/
-----*/
/* 位相補償1(速度) */
    if( MotSpd > PhaseCmp->conf.PcmpSpd1 )
    {
        PhaseCmp->var.Pcmps1 =
            (SHORT)MlibMulgainNolim( (MotSpd - PhaseCmp->conf.PcmpSpd1),
PhaseCmp->conf.Kpcmpsp1 );
    }
    else if( MotSpd < -PhaseCmp->conf.PcmpSpd1 )
    {
        PhaseCmp->var.Pcmps1 =
            (SHORT)MlibMulgainNolim( (MotSpd + PhaseCmp->conf.PcmpSpd1),
PhaseCmp->conf.Kpcmpsp1 );
    }
#endif /* FLOAT_USE */
    else
    {
        PhaseCmp->var.Pcmps1 = 0;
    }
/* 位相補償合計(速度) [65536/360deg] */
PhaseCmp->var.Pcmps = PhaseCmp->var.Pcmps1 + PhaseCmp->var.Pcmps2;

```

Pcmps1 的补偿是根据 PcmpSpd1 开始速度阈值进行补偿的, 绝对值在此范围内, 不进行补偿, 大于此阈值, 则当前速度减去此阈值, 再乘以一个系数 Kpcmpsp1; 若小于此阈值, 则当前速度

加上此阈值，再乘以一个系数 Kpcmpsp1。Pcmps2 由于注释掉了，可认为是 0。

```
Bprm->MaxVel = MencP->vrat_max.b.h;
Bprm->PerOvrSpd = CfgPrm->ratmt2_os.b.h * MencP->oslv_pol.b.l;
Bprm->NorOvrSpd = 0x01000000;
Bprm->NorMaxSpd = MlibScalKxgain( Bprm->NorOvrSpd, 10000, (Bprm->PerOvrSpd+10000), NULL,
-24 );
Bprm->NorRatSpd = MlibScalKxgain( Bprm->NorMaxSpd, MencP->vrat_max.b.l, Bprm->MaxVel,
NULL, -24 );
```

PerOvrSpd 为最大速度百分比 0.01%，**NorOvrSpd** 为 Q24，**MaxVel** 为最大速度 100r/min，**NorMaxSpd** 为计算得到的最大速度标么值，默认值应该就是 Q24，**NorRatSpd** 为额定转速的标么值。

```
/*-----*/
/*      位相補償パラメータ計算:位相補償1      */
/*-----*/
/*
/*
/*
/*      PcmpSpd1 = NorMaxSpd * PhsCmpSpd1 / MotMaxSpd      位相補償開始速度      */
/*
/*
/*      PhsCmpDeg1 * 65536      */
/*
/*      Kpcmpsp1 = -----      位相補償演算ゲイン      */
/*      360 * (NorMaxSpd - PcmpSpd1)      */
/*
/*
/*-----*/
/*
/*      PhsCmpDeg1 : 位相補償値1      [deg/MaxSpd]      */
/*      PhsCmpSpd1 : 位相補償開始速度1      [100r/min]      */
/*      MotMaxSpd : モータ最大速度      [100r/min]      */
/*
/*
/*-----*/
kx = MlibScalKxgain( Bprm->NorMaxSpd, Prm->MencP.phscmpM1.b.h, Bprm->MaxVel, NULL, -30 );
Axis->BaseControls->PcmpCalcData.PcmpSpd1 = kx;
```

PcmpSpd1 为计算得到的位相补偿开始速度的标么值。

```
if( Prm->phscmp != 0 )      /* サーボパックパラメータ != 0
*/
{
    PhsCmpDegx = (LONG)Prm->phscmp;      /* サーボパックパラメータ使用
*/
}
else if( Prm->MencP.phscmpM1.b.l == 0xFF )      /* モータパラメータが、F 詰めの場合
*/
{
    PhsCmpDegx = (LONG)Prm->phscmp;      /* サーボパックパラメータ使用
*/
}
else      /* モータパラメータが、F 詰めでない場合      */
```

```

{
    PhsCmpDegx = (LONG)Prm->MencP.phscmpM1.b.l;          /* モーターパラメータ使用 */
}
kx = MlibScalKxgain( PhsCmpDegx, 65536, 360, &sx, 0 );
kx = MlibPcalKxgain( kx, 1, (Bprm->NorMaxSpd - Axis->BaseControls->PcmpCalcData.PcmpSpd1),
&sx, 24 );
Axis->BaseControls->PcmpCalcData.Kpcmpsp1 = kx;
PhsCmpDegx 根据不同的判断选择不同的功能码值，这个值不确定。Kpcmpsp1 的计算则是将
Deg/vmax，转化成 Q15 的格式。

```

$$\theta_1 = \begin{cases} \frac{v - v_0}{v_{\max} - v_0} \theta_{\max} & (v > v_0) \\ 0 & (v_0 \geq v \geq -v_0) \\ \frac{v + v_0}{v_{\max} - v_0} \theta_{\max} & (v < -v_0) \end{cases}$$

其中 θ_1 为根据不同速度补偿的电角度， v_0 为开始速度， v 为当前实际速度， v_{\max} 为最大速度。

其目的在于根据最大速度对应的补偿角度，做一个比例调节，以及一个死区范围，进行补偿角度。其难点在于这个**最大速度的补偿角度是怎么来的，以及开始速度的确定**。也许和功率因数角有关，则不同的电机应该有不同的补偿方式。

1.3 电流变换

```

/*-----*/
/*      theta calculation                      */
/*-----*/

swk0 = AxisRscI->PhaseV.PhaseH;
swk0 = swk0 + 32;          /* TMP3 <-- PhaseV.PhaseH + 2^5 */
swk1 = PI23;
swk2 = swk1 + swk0;      /* TMP4 <-- PhaseV.PhaseH + 2PI/3 */
swk3 = swk0 - swk1;      /* TMP5 <-- PhaseV.PhaseH - 2PI/3 */

```

U,V,W 三相角度计算，U 超前 V $2\pi/3$, V 前 W $2\pi/3$.为什么要加上 32，初始角 β

$$\begin{cases} i_A = I_m \cos(\theta + \beta) \\ i_B = I_m \cos(\theta + \beta - \frac{2\pi}{3}) \\ i_C = I_m \cos(\theta + \beta + \frac{2\pi}{3}) \end{cases}$$

```

/*-----*/
/*      table read and get iu,iv by Id,Iq reference      */
/*-----*/

//      swk1 = swk0 >> 6;          /* TMP1 <-- TMP3 >> 6 */
      swk1 = (USHORT)swk0 >> 6;      /* TMP1 <-- TMP3 >> 6 */
      IxTblSin16( AxisRscI->SinTbl.SinT, swk1 );      /* SinTbl.SinT <-- stable[ TMP1 ] */

```

```

        swk0 = swk0 + PI2;                                /* TMP3 <-- TMP3 + PI/2*/
//      swk1 = swk0 >> 6;                                /* TMP1 <-- TMP3 >> 6*/
        swk1 = (USHORT)swk0 >> 6;                        /* TMP1 <-- TMP3 >> 6*/
        IxTblSin16( AxisRscI->SinTbl.CosT, swk1 ); /* SinTbl.CosT <-- stable[ TMP1 ]*/
//      swk1 = swk3 >> 6;                                /* TMP1 <-- TMP5 >> 6 */
        swk1 = (USHORT)swk3 >> 6;                        /* TMP1 <-- TMP5 >> 6 */
        IxTblSin16( AxisRscI->SinTbl.SinT3, swk1 ); /* SinTbl.SinT3 <-- stable[ TMP1 ] */
        swk3 = swk3 + PI2;                                /* TMP5 <-- TMP5 + PI/2 */
//      swk1 = swk3 >> 6;                                /* TMP1 <-- TMP5 >> 6*/
        swk1 = (USHORT)swk3 >> 6;                        /* TMP1 <-- TMP5 >> 6*/
        IxTblSin16( AxisRscI->SinTbl.CosT3, swk1 ); /* SinTbl.CosT3 <-- stable[ TMP1 ]*/
//      swk1 = swk2 >> 6;                                /* TMP1 <-- TMP4 >> 6*/
        swk1 = (USHORT)swk2 >> 6;                        /* TMP1 <-- TMP4 >> 6*/
        IxTblSin16( AxisRscI->SinTbl.SinT2, swk1 ); /* SinTbl.SinT2 <-- stable[ TMP1 ]*/
        swk2 = swk2 + PI2;                                /* TMP4 <-- TMP4 + PI/2*/
//      swk1 = swk2 >> 6;                                /* TMP1 <-- TMP4 >> 6*/
        swk1 = (USHORT)swk2 >> 6;                        /* TMP1 <-- TMP4 >> 6*/
        IxTblSin16( AxisRscI->SinTbl.CosT2, swk1 ); /* SinTbl.CosT2 <-- stable[ TMP1 ]*/

```

U, V, W 三角函数正余弦角度的计算。

```

//      /* 2012.12.21 Y.Oka 現状初期化必要 */
/*-----*/

AxisRscR->SinTbl.SinT = 0x0000; /* SinTbl.SinT=sin(θ) sin(0)= 0.000 → 0000h */
AxisRscR->SinTbl.CosT = 0x4000; /* SinTbl.CosT=cos(θ) cos(0)= 1.000 → 4000h */
AxisRscR->SinTbl.SinT2 = 0x376D; /* SinTbl.SinT2=sin(θ+2π/3)sin(2π/3)=0.866 → 376Dh */
AxisRscR->SinTbl.CosT2 = 0xE000; /* SinTbl.CosT2=cos(θ+2π/3) cos(2π/3)= -0.500
→ E000h */
AxisRscR->SinTbl.SinT3 = 0xC893; /* SinTbl.SinT3=sin(θ-2π/3) sin(-2π/3)=-0.866 →
C893h */
AxisRscR->SinTbl.CosT3 = 0xE000; /* SinTbl.CosT3=cos(θ-2π/3) cos(-2π/3)=-0.500 →
E000h */

```

按照此初始化程序的计算SIN,COS的标么化应为Q14格式。

```

/*-----*/
/*      dq-trans(UVW to DQ) */
/*-----*/
/*      ID = IntAdP.Kc * ( (SinTbl.CosT-SinTbl.CosT2)*IntAdV.IuInData/2^14 +
(SinTbl.CosT3-SinTbl.CosT2)*IntAdV.IvInData/2^14 ) / 2^9 */
/*      IQ = IntAdP.Kc * ( (SinTbl.SinT2-SinTbl.SinT)*IntAdV.IuInData/2^14 +
(SinTbl.SinT2-SinTbl.SinT3)*IntAdV.IvInData/2^14 ) / 2^9 */
/*-----*/

/* TMP1 <-- cos(th) - cos(th-2pi/3) */
swk1 = AxisRscI->SinTbl.CosT - AxisRscI->SinTbl.CosT2;
/* ACC <-- TMP1 * iu */

```

```

swk2 = mulshr(swk1, AxisRscI->IntAdV.IuInData, 14 );
/* TMP1 <-- cos(th-2pi/3)-cos(th+2pi/3) */
swk1 = AxisRscI->SinTbl.CosT3 - AxisRscI->SinTbl.CosT2;
/* ACC <-- TMP1 * iv */
swk1 = mulshr(swk1, AxisRscI->IntAdV.IvInData, 14 );
/* TMP2 <-- TMP2 + TMP1 */
swk2 = swk1 + swk2;
/* ACC <-- IntAdP.Kc * TMP2 */
AxisRscI->IntAdV.IdInData = mulshr(AxisRscI->IntAdP.Kc, swk2, 9 );
/*-----*/

swk1 = AxisRscI->SinTbl.SinT2 - AxisRscI->SinTbl.SinT;
/* TMP1 <-- sin(th+2pi/3) - sin(th) */
swk2 = mulshr(swk1, AxisRscI->IntAdV.IuInData, 14 ); /* ACC <-- TMP1 * iu */
swk1 = AxisRscI->SinTbl.SinT2 - AxisRscI->SinTbl.SinT3;
/* TMP1 <-- sin(th+2pi/3)-sin(th-2pi/3) */
swk1 = mulshr(swk1, AxisRscI->IntAdV.IvInData, 14 ); /* ACC <-- TMP1 * iv */
swk2 = swk1 + swk2; /* TMP2 <-- TMP2 + TMP1 */
AxisRscI->IntAdV.IqInData = mulshr(AxisRscI->IntAdP.Kc, swk2, 9 ); /* ACC <--
IntAdP.Kc * TMP2 */

```

其计算公式为

$$\begin{aligned}
 \begin{bmatrix} i_d \\ i_q \end{bmatrix} &= \frac{2}{3} \begin{bmatrix} \cos \theta & \cos \left(\theta - \frac{2}{3} \pi \right) & \cos \left(\theta + \frac{2}{3} \pi \right) \\ -\sin \theta & -\sin \left(\theta - \frac{2}{3} \pi \right) & -\sin \left(\theta + \frac{2}{3} \pi \right) \end{bmatrix} \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} = \\
 &\frac{2}{3} \begin{bmatrix} \cos \theta & \cos \left(\theta - \frac{2}{3} \pi \right) & \cos \left(\theta + \frac{2}{3} \pi \right) \\ -\sin \theta & -\sin \left(\theta - \frac{2}{3} \pi \right) & -\sin \left(\theta + \frac{2}{3} \pi \right) \end{bmatrix} \begin{bmatrix} i_A \\ i_B \\ -i_A - i_B \end{bmatrix} = \\
 &\frac{2}{3} \begin{bmatrix} \cos \theta - \cos \left(\theta + \frac{2}{3} \pi \right) & \cos \left(\theta - \frac{2}{3} \pi \right) - \cos \left(\theta + \frac{2}{3} \pi \right) \\ \sin \left(\theta + \frac{2}{3} \pi \right) - \sin \theta & \sin \left(\theta + \frac{2}{3} \pi \right) - \sin \left(\theta - \frac{2}{3} \pi \right) \end{bmatrix} \begin{bmatrix} i_A \\ i_B \end{bmatrix}
 \end{aligned}$$

1.3 电流观测器

是否开启电流观测器

dq_sw 为功能码 **PnE2F** 其默认值根据机种不同，小功率为 **0D0AH**,则默认开启电流观测器

SvAsicRegs->MicroCsw.data = (USHORT)(uCfgrm->**dq_sw** | 0x00F0); /* 制御フラグ */

/* Set to ASIC */

rc |= LpxSetPrmToASIC(&(pAsicMicroReg->**IntAdP.CtrlSw**), SvAsicRegs->**MicroCsw.data**,

```

NO_LMT_CHK );
#define OBSSEL      0x0008  /* bit.3 : Current Observer Select bit  ; <V038> <V076>  */
if( AxisRsc->IntAdV.IqInData >= 0 )
{ /* 0以上のとき */
    /* TMP3 = IntAdV.IqInData */
    AxisRsc->IntAdwk.swk2 = AxisRsc->IntAdV.IqInData;
}
else /* 負のとき */
{
    AxisRsc->IntAdwk.swk2 = ~AxisRsc->IntAdV.IqInData; /* TMP3 = ~IntAdV.IqInData;
    *///110530tanaka21作業メモ、-1掛けるのとどっちが速い？ 取反 相当于取绝对值操作
    AxisRsc->IntAdwk.swk2 = AxisRsc->IntAdwk.swk2 + 1; /* TMP3 = TMP3 + 1 */
}
if( AxisRsc->IntAdwk.swk2 <= 14250 )
{
    AxisRsc->IntAdwk.swk3 = 0; /* TMP4 = 0 ( OverFlowCheck = OK ) */
}
else
{
    AxisRsc->IntAdwk.swk3 = 1; /* TMP4 = 1 ( OverFlowCheck = NG ) */
}

```

判別 q 軸電流是否飽和，置标志位。若飽和，再后续程序中不累加观测電流。

```

/* 2012.10.26 Y.Oka ★電流オブザーバ現状未対応⇒パラメーター括書き込み対応要★ */
/*****
/*
/*      電流オブザーバパラメータ計算
/*
/*
*****/
/*
/*      オブザーバゲイン1 :  $T_s/L$ 
/*
/*       $\frac{scantime[ns]}{10^9} \cdot \frac{1}{Bprm.MotR[\Omega]} \cdot \frac{Bprm.Vdc[Vop]/2}{2^{13}} \cdot \frac{15000}{Bprm.MaxCur[Aop]} \cdot 2^7$ 
/*wk_TSPL = ----- * ----- * ----- * ----- * 2^7
/*      10^9      Bprm.MotR[Ω]      2^13      Bprm.MaxCur[Aop]
/*
/*      オブザーバゲイン2 :  $1 - R \cdot T_s / L - \exp(-2\pi \cdot K_{pi})$ 
/*       $\frac{Bprm.MotR[\Omega]}{scantime[ns]}$ 
/*       $wk\_GOBS = 2^8 - \frac{Bprm.MotLq[H]}{10^9} \cdot \frac{scantime[ns]}{10^9} \cdot 2^8$ 
/*      Bprm.MotLq[H]      10^9
/*
/*
/*       $- 2^8 \cdot \exp(-2\pi \cdot K_{pi}[Hz]) \cdot \frac{scantime[ns]}{10^9}$ 
/*      - 2^8 * exp( -2*pi * Kpi[Hz] * ----- )
/*      10^9
/*

```



```

/*                                                                    */
/*      オブザーバゲイン3 : 1 - R* Ts / L                            */
/*                                                                    */
/*      Bprm.MotR[Ω]      scantime[ns]                               */
/*      wk_GOBS = 2^4 - ----- * ----- * 2^4                      */
/*      Bprm.MotLq[H]      10^9                                       */
/*                                                                    */
/*                                                                    */
/*      フィルタゲイン      : Ts / ( Ts + Tfil )                      */
/*                                                                    */
/*      scantime[ns]                                               */
/*      wk_FILOBS = ----- * 16384                                   */
/*      10^9                                                         */
/*      scantime[ns] + -----                                       */
/*      2*pi * Pwmf[Hz]                                              */
/*                                                                    */
/*      scantime( KPI_MACYCLEN ) : 電流制御周期[ns]                 */
/*      Bprm.MotLq      : q軸モータインダクタンス[H]                */
/*      Bprm.MotR      : q軸モータ抵抗値[Ω]                         */
/*      Bprm.MaxCur   : 最大電流[Aop]                               */
/*      Bprm.Vdc      : 最大電圧[Vop]                                */
/*      Kpi      : q軸電流ループゲインq軸電流環增益[Hz] ( PnE20 : Prm.ignq )*/
/*      Pwmf      : キャリア周波数[Hz] ( PnE2C : Prm.pwmf )*/
/*****
static void KpiPcalCurrentObs( ASICS *SvAsicRegs, USHORT ignq, USHORT pwmf, BPRMDAT
*Bprm, SHORT *MpReg)
KpiPcalCurrentObs ( ) 函数中有关于系数的计算

```

$$TsPerL = \frac{T_s}{L}$$

$$Gobs = 1 - \frac{R \times T_s}{L} - e^{-2\pi Kpi}$$

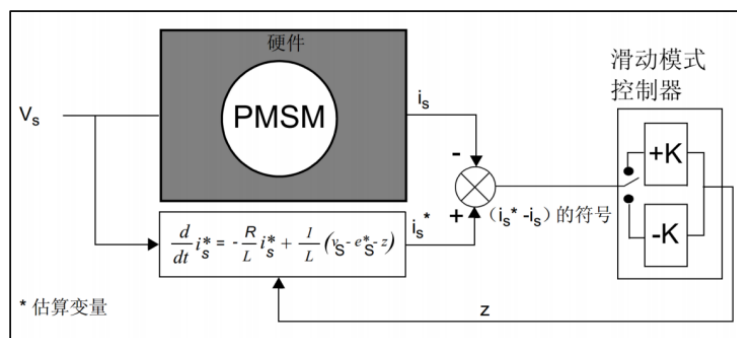
$$RLTs = 1 - \frac{R \times T_s}{L}$$

$$FilObsGain = \frac{T_s}{T_s + T_{PWM}}$$

其差分方程为

$$i_{obs}(n) = (1 - \frac{R \times T_s}{L})i_{obs}(n-1) + \frac{T_s}{L}U(n-1) + (1 - \frac{R \times T_s}{L} - e^{-2\pi Kpi})(i(n) - i_{obs}(n-1))$$

可对比硬石的电流观测器，其结构有相似性，但其观测的是反电动势



通过一个一阶低通滤波器

$$u(n) = i(n) - i_{obs}(n)$$

$$x(n) = x(n-1) + \frac{T_s}{T_s + T_{PWM}} (u(n) - x(n-1))$$

低通滤波器的输出经过一个高通滤波器

$$y(n) = y(n-1) + \frac{T_s}{T_s + T_{PWM}} (x(n) - y(n-1))$$

$$IObsFreq(n) = x(n) - y(n)$$

低通滤波器与高通滤波器的默认滤波时间常数一样，则为无效。

若 q 轴电流饱和，上述高通滤波器输出 $IObsFreq = 0$ ，最终的计算结果为

$$i(n) = i(n) - IObsFreq(n)$$

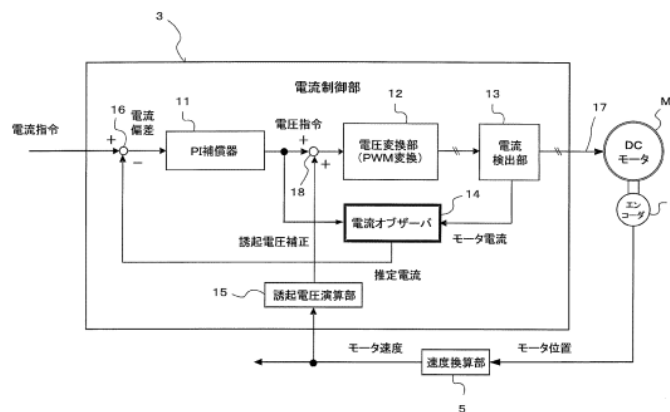
d,q 轴的计算均采用此方式，此处电流观测器的目的是为了提高瞬态的响应还是相当于一个超前校正。串联加入低通滤波器和高通滤波器的作用是抵消掉开关噪声，使电流反馈更接近真实值吗。

在安川的两篇相关专利中有提到电流观测器

其中一篇中

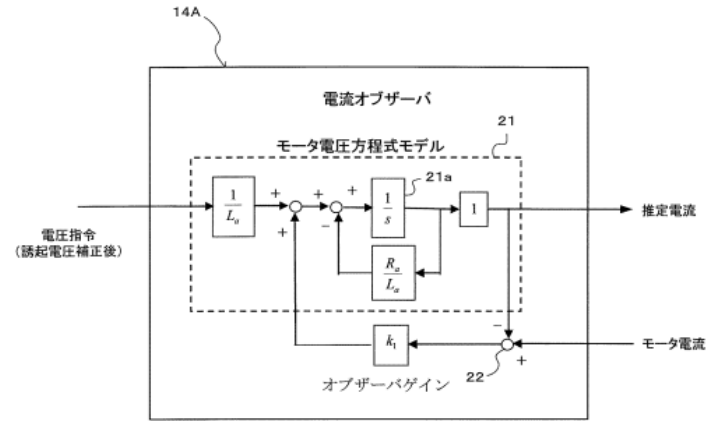
电压方程

$$u_a = R_a i_a + L_a \frac{di_a}{dt} + e_a$$



电流环框图

电压指令以及电流反馈作为输入通过电流观测器观测出电流进行电流环 PI 计算



电流扰动观测器

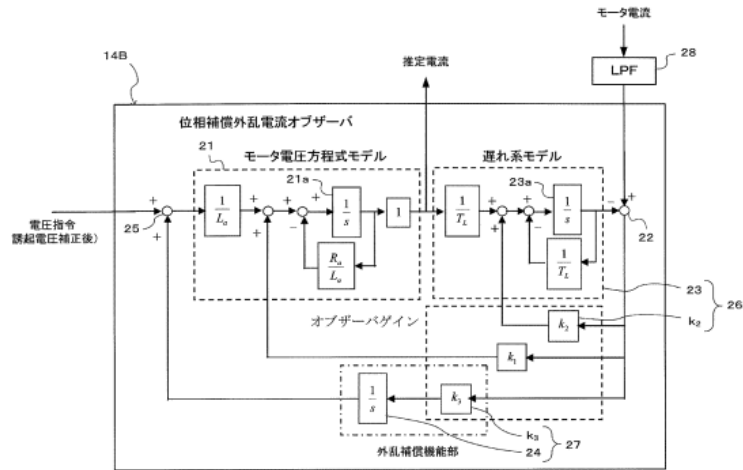
其估算微分方程可表示为

$$\frac{di_{aobs}}{dt} = \frac{1}{L_a} u_a + k_1 (i_a - i_{aobs}) - \frac{R_a}{L_a} i_{aobs}$$

转化为差分方程

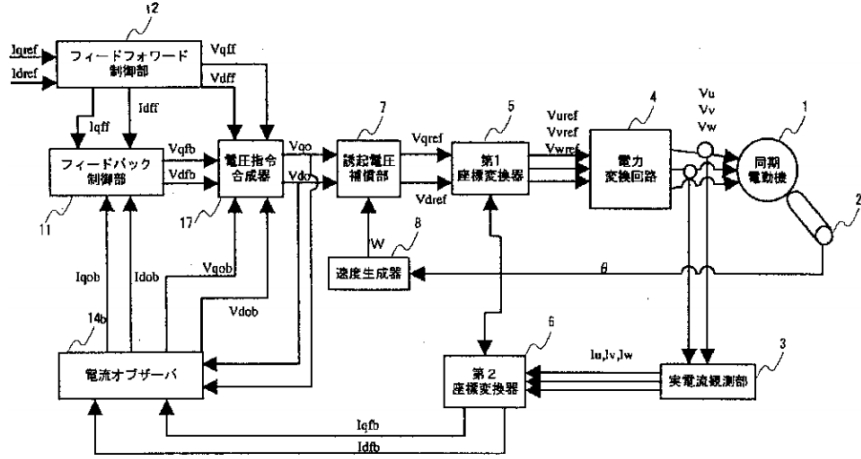
$$i_{obs}(n) = \left(1 - \frac{R \times T_s}{L}\right) i_{obs}(n-1) + \frac{T_s}{L} U(n-1) + k_1 T_s (i(n) - i_{obs}(n-1))$$

与安川的程序很接近，**转化为 k_1 的取值问题。**



相位补偿扰动电流观测器

在另一篇专利文献中



$$Idob * s = -R_d * Idob / L_d + L_{d1} * (Idfb - Idob) + (Vdob + Vdo) / L_d \quad (32)$$

$$Vdob * s = L_{d2} * (Idfb - Idob) \quad (33)$$

$$Iqob * s = -R_q * Iqob / L_q + L_{q1} * (Iqfb - Iqob) + (Vqob + Vqo) / L_q \quad (34)$$

$$Vqob * s = L_{q2} * (Iqfb - Iqob) \quad (35)$$

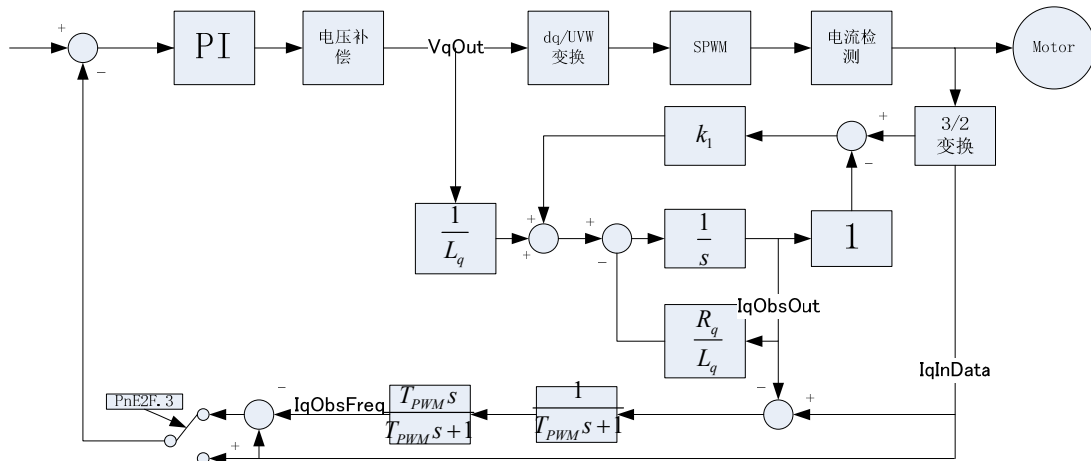
在电压指令合成部 17 中，按如下生成 d 轴第一模拟电压指令 Vdref 和所述 q 轴第一模拟电压指令 Vqref。

$$Vdo = Vdff + Vdfb + Vdob \quad (36)$$

$$Vqo = Vqff + Vqfb + Vqob \quad (37)$$

与第一篇的专利是可以相互佐证的。

但安川的程序似乎只用到了扰动观测器，消除的是扰动部分，从而提高带宽。



1.4 弱磁控制

AxisRsc->AdStop.ADRst = AxisRsc->IntAdP.FccRst;

```

if( AxisRsc->AdStop.ADRst != 0 )
{
    AxisRsc->AdStop.ADRst = 0;
    AxisRsc->IntAdwk.swk6 = AxisRsc->IntAdV.CrFreqW >> 1;
    AxisRsc->IntAdwk.swk5 = AxisRsc->IntAdwk.swk6;
    AxisRsc->IntAdwk.swk4 = AxisRsc->IntAdwk.swk6;

    /*-----*/
}
else if( AxisRsc->StsFlg.CtrlStsRW == BB )
{
    /*-----*/

    AxisRsc->IntAdwk.swk6 = AxisRsc->IntAdV.CrFreqW >> 1;
    AxisRsc->IntAdwk.swk5 = AxisRsc->IntAdwk.swk6;
    AxisRsc->IntAdwk.swk4 = AxisRsc->IntAdwk.swk6;

    /*-----*/
}
在 AD 复位以及控制状态为 BB 模式时，占空比输出为 50%。
/*-----*/
/*      弱め界磁制御用電圧演算用ゲインの計算 (2*ScanBCycle)      */
/*-----*/
/
/*
/*
/*      VdcLevel * 2^13      VdcBase : Base DcVolt (Bprm->Vdc) */
/*      KVdcAsic = -----      VdcLevel : Detect DcVolt Level (PnE70)*/
/*      VdcBase * VdetMax      VdetMax : at Vdet Max      */
/* Attribute of PnE70 : P-N電圧検出レベル 単位V */
/* VdetMax;      Amp Volt detect at max input      [data/MaxV] [data/MaxV] */
/*-----*/
#if (FLOAT_USE==TRUE)
    fw = (float)pnvoltgn * 8192.0F / Bprm->Vdc;
    fw = fw / (float)Bprm->VdetMax;
    weaken_ptr->conf.KVdcAsic = fw;
#else
    kx = MlibScalKxkxks( pnvoltgn, 8192, Bprm->Vdc, &sx, 0 );
    kx = MlibPcalKxgain( kx, 1, Bprm->VdetMax, &sx, 24 );
    weaken_ptr->conf.KVdcAsic = kx;
#endif /* FLOAT_USE */

/*-----*/
/*      主回路電圧の平均値計算[8192/Vdc]      */
/*-----*/

wf_ptr->var.VdcSum += DcVolt;
if ( ++wf_ptr->var.VdcSumCnt >= wf_ptr->var.VdcAvgCnt )
{

```

```

        work1 = wf_ptr->var.VdcSum / wf_ptr->var.VdcSumCnt;
    #if (FLOAT_USE==TRUE)
        work1 = (LONG)((float)work1 * wf_ptr->conf.KVdcAsic);
    #else
        work1 = MlibMulgain( work1, wf_ptr->conf.KVdcAsic );
    #endif /* FLOAT_USE */
    wf_ptr->var.VdcAsic = (SHORT)work1;
    wf_ptr->var.VdcSum = 0;
    wf_ptr->var.VdcSumCnt = 0;
}

LONG   DcVolt;                /* DC電圧 [0.0001V] */
Bprm->Vdc = (float)AmpVtbl[Idx].DcVolt / 10000.0F;
/*-----*/
/*      アンプ主回路検出最大値の設定      */
/*-----*/
Bprm->VdetMax = AmpVtbl[Idx].VdetMax;
初期化在const   AMPVTBL   AmpVtbl[16]中进行，以AC200V为例

AmpVtbl[Idx].DcVolt = 2828427 = 200 *  $\sqrt{2}$  * 10000
AmpVtbl[Idx].VdetMax = 255 为8位数据最大值 具体物理含义不清楚。

/*****
/*
/*      d軸電流指令リミット値計算用定数の計算      */
/*      (1項目の計算)      */
/*
/*****
/*      計算はすべて実効値ではなく、0-pで考える。      */
/*
/*      
$$Id_{lim} = \frac{V_{max}^2 - (K_e \omega_m)^2 - (\omega L)^2 I_{max}^2}{2 * \omega L * K_e \omega_m} - \frac{R}{\omega L} * I_{max}$$

/*      の1項目を計算する。      */
/*
/*      
$$V_{max} = V_{dc} * V_{dcAsic} / 8192 * Prm.v1max / 100 * Prm.vmaxid / 100 / 2 [V_{0-p}]$$

/*      Prm.v1max      :PnEB3[%]      */
/*      Prm.vmaxid     :PnEB4[%]      */
/*      VdcAsic        主回路電圧平均値      */
/*
/*      
$$1項目 = \frac{(V_{dc} * Prm.v1max * Prm.vmaxid)^2 * (V_{dcAsic} / 8192)^2}{4\sqrt{2} * 10^8 * pole * L_x * E_{mf}}$$

/*      の $\omega_m^2$ と(VdcAsic / 8192)^2を除いた部分を計算      */

```

```

/*                                                                    */

/*          (Bprm.Vdc * Prm.v1max * Prm.vmaxid)^2    15000    15000^2    */
/*IdrefLimTerm1 = ----- * ----- * ----- / 2^48 */
/*          4√2 * pole * Lx * Emf                    Imax    Nmax^2    */
/*                                                                    */
/*      1/ωmは、ScanBで2^24/ωmを計算して、掛ける。    */
/*                                                                    */
/*      Vdc  [V]          : DC Voltage    */
/*      Emf  [Vrms/rad/s] : EMF Constant */
/*      Lx   [H]          : Inductance    */
/*      Rx   [ohm]        : Resistance    */
/*      Imax [Ao-p]       : Max. Current  */
/*      Nmax [rad/s]      : Max. Speed    */
/*      pole [-]         : pole No.      */
/*      ωm   [15000/Nmax] : Motor Speed  */
/*                                                                    */
/*****
void PcalIdrefLimTerm1( WEAKENFIELD *WeakenField, ASICS *SvAsicRegs, BPRMDAT *Bprm,
LONG v1max, LONG vmaxid)
{
    LONG      kx,sx;
    LONG      Vdc;
    LONG      Keangle;
    LONG      MotLd;
    LONG      MotEmf;
    LONG      MaxCur;
    LONG      OvrSpd;

    Vdc      = Bprm->Vdc;
    Keangle   = Bprm->Keangle;
    MotLd     = Bprm->MotLd;
    MotEmf    = Bprm->MotEmf;
    MaxCur   = Bprm->MaxCur;
    OvrSpd    = Bprm->OvrSpd;

    /* 1項目を計算 */
    kx = MlibScalKskskx( Vdc, Vdc, 100000000, &sx, 0 );
    kx = MlibPcalKxgain( kx, 10000, 56569*2, &sx, 0 );
    kx = MlibPcalKxkxks( kx, 1, Keangle, &sx, 0 );
    kx = MlibPcalKxkxks( kx, v1max, MotLd, &sx, 0 );
    kx = MlibPcalKxkxks( kx, v1max, MotEmf, &sx, 0 );
    kx = MlibPcalKxkxks( kx, 15000, MaxCur, &sx, 0 );
    kx = MlibPcalKxkxks( kx, 15000, OvrSpd, &sx, 0 );

```

```

kx = MlibPcalKxkxks( kx, 15000, OvrSpd, &sx, 0 );
kx = MlibPcalKxgain( kx, vmaxid, 16777216, &sx, 0 );
kx = MlibPcalKxgain( kx, vmaxid, 16777216, &sx, 48 );

```

```

WeakenField->conf.IdrefLimTerm1 = kx;

```

```

WeakenField->conf.V1Max = (SHORT)(v1max * 8192 / 100);

```

```

}

```

d 轴的电流限制值为

$$I_{d\lim} = \frac{V_{\max}^2 - (K_e \omega_m)^2 - (\omega_e L)^2 I_{\max}^2}{2\omega_e L K_e \omega_m} - \frac{R}{\omega_e L} I_{\max}$$

弱磁控制的原理是根据电压方程

$$u_d = R_s i_d + L_d \frac{di_d}{dt} - \omega_r L_q i_q$$

$$u_q = R_s i_q + L_q \frac{di_q}{dt} + \omega_r L_d i_d + e_0$$

在正弦稳态下，

$$u_d = R_s i_d - \omega_r L_q i_q$$

$$u_q = R_s i_q + \omega_r L_d i_d + e_0$$

若忽略电阻电压以及将反电动势计算可得

$$u_d = -\omega_r L_q i_q$$

$$u_q = \omega_r L_d i_d + e_0$$

根据电压极限方程，以及电流极限方程

$$u_d^2 + u_q^2 \leq V_{\max}^2$$

$$i_d^2 + i_q^2 \leq I_{\max}^2$$

可得

$$I_{d\lim} = \frac{V_{\max}^2 - (K_e \omega_m)^2 - (\omega_e L)^2 I_{\max}^2}{2\omega_e L K_e \omega_m}$$

若不忽略电阻电压降，则有

$$I_{d\lim} = \frac{V_{\max}^2 - (K_e \omega_m)^2 - (\omega_e L)^2 I_{\max}^2}{2\omega_e L K_e \omega_m} - \frac{R^2 I_{\max}^2}{2\omega_e L K_e \omega_m} - \frac{R}{\omega_e L} i_q$$

与安川的公式有差别

/ *-----*/


```

/*      ASICマイクロプログラムへの弱め界磁制御関連変数出力      <S024>      */
/*-----*/
/* 電圧FB比例ゲイン(下位16bit) */
SvAsicRegs->AsicMicroReg->WeakFV.WfKpVLIn = BaseCtrls->WeakenField.var.KpvL;
/* 電圧FB比例ゲイン(上位16bit) */
SvAsicRegs->AsicMicroReg->WeakFV.WfKpVHIn = BaseCtrls->WeakenField.var.KpvH;
/* 電圧FB積分ゲイン(下位16bit) */
SvAsicRegs->AsicMicroReg->WeakFV.WfKiVLIn = BaseCtrls->WeakenField.var.KivL;
/* 電圧FB積分ゲイン(上位16bit) */
SvAsicRegs->AsicMicroReg->WeakFV.WfKiVHIn = BaseCtrls->WeakenField.var.KivH;
/* 電圧指令制限値 */
SvAsicRegs->AsicMicroReg->WeakFV.WfV1MaxIn = BaseCtrls->WeakenField.conf.V1Max;
/* d軸電流指令リミット */
SvAsicRegs->AsicMicroReg->WeakFV.WfIdRefLimIn = BaseCtrls->WeakenField.var.IdrefLim;
弱磁相关变量赋值在MicroIfOutputCycData函数中进行，此函数在SCANA中运行

AxisRscR->WeakFV.WfKpV.s[0] = AxisRscR->WeakFV.WfKpVLIn; /* 電圧FB比例ゲイン(下位16bit) <V214> */
AxisRscR->WeakFV.WfKpV.s[1] = AxisRscR->WeakFV.WfKpVHIn; /* 電圧FB比例ゲイン(上位16bit) <V214> */
AxisRscR->WeakFV.WfKiV.s[0] = AxisRscR->WeakFV.WfKiVLIn; /* 電圧FB積分ゲイン(下位16bit) <V214> */
AxisRscR->WeakFV.WfKiV.s[1] = AxisRscR->WeakFV.WfKiVHIn; /* 電圧FB積分ゲイン(上位16bit) <V214> */
AxisRscR->WeakFV.WfV1Max = AxisRscR->WeakFV.WfV1MaxIn; /* 電圧指令制限値 <V214> */
AxisRscR->WeakFV.WfIdRefLim = AxisRscR->WeakFV.WfIdRefLimIn; /* d軸電流指令リミット <V214> */
弱磁相关变量在 MpIntHost 函数中运行

if( uCfgPrm->flg_wf & 0x0001 ) == 0 )
{ /* モータ側の弱め界磁有効無効チェック */
    /* モータ側が無効の場合、強制的に無効 马达侧是无效的情况，弱磁强制无效 */
    SvAsicRegs->MicroCsw.data &= 0xF7FF;
}

if( SvAsicRegs->MicroCsw.data & 0x0800 ) == 0 )
{
    /* 弱め界磁=OFFの場合，弱め界磁2=OFFにする 弱磁方式1为OFFの場合 关闭弱磁方式2 */
    SvAsicRegs->MicroCsw.data &= 0xDFFF;
}

if( SvAsicRegs->MicroCsw.data & 0x2000 ) != 0 )
{
    /* 弱め界磁2=ONの場合，積分停止機能=ONにする 弱磁方式2打开 停止积分作用 */
    SvAsicRegs->MicroCsw.data |= 0x4000;
}

#define V_FB      0x0800      /* bit.11 電圧 FB 方式弱め界磁選択 */
if( AxisRsc->IntAdP.CtrlSw & V_FB )

```

弱磁标志位判断，若成立则进行以下计算

```

/*-----*/
//       $V_{qmax} = \sqrt{V_{max}^2 - V_d^2}$ 
/*-----*/

AxisRsc->IntAdwk.lwk2 = AxisRsc->WeakFV.WfV1Max * AxisRsc->WeakFV.WfV1Max; /*
IntAdP.Vmax^2
AxisRsc->IntAdwk.lwk4 = AxisRsc->WeakFV.WfVdRef * AxisRsc->WeakFV.WfVdRef; /*  $V_d^2$ 
; 削除 <V309>      復活<V531>  */

#ifdef WIN32
    IxADDSUBLMTCHKRDY( AxisRsc->IntAdwk.lwk2, AxisRsc->IntAdwk.lwk4 );
#endif

    AxisRsc->IntAdwk.lwk2 = AxisRsc->IntAdwk.lwk2 - AxisRsc->IntAdwk.lwk4;
/* IntAdP.Vmax^2 -  $V_d^2$ 
#ifdef WIN32
    IxSUBLMTCHK( AxisRsc->IntAdwk.lwk2 );
#endif

AxisRsc->IntAdwk.lwk2 = IxLmtCBS32( AxisRsc->IntAdwk.lwk2 ); /*
AxisRsc->IntAdwk.lwk2 = IxLIMITUL( AxisRsc->IntAdwk.lwk2, LPX_REG32_MAX,
LPX_REG32_MIN ); /* if (IntAdP.Vmax^2 -  $V_d^2$ ) < 0, then (IntAdP.Vmax^2 -  $V_d^2$ ) = 0 */
AxisRsc->IntAdwk.swk0 = MpSQRT( &AxisRsc->IntAdwk, AxisRsc->IntAdwk.lwk2 ); /*
 $\sqrt{IntAdP.Vmax^2 - V_d^2}$ 
if( AxisRsc->IntAdwk.swk0 > 0x7FFF )
{
    AxisRsc->IntAdwk.swk0 = 0x7FFF; /*
}

AxisRsc->WeakFV.WfVqMax = AxisRsc->IntAdwk.swk0; /*  $V_{qmax} = \sqrt{IntAdP.Vmax^2 - V_d^2}$  */
其计算公式为

```

$$V_{qmax} = \sqrt{V_{max}^2 - V_d^2}$$

```

/*-----*/
//      TMP0 =  $V_{qmax} - V_q$ 
/*-----*/

AxisRsc->IntAdwk.swk1 = AxisRsc->WeakFV.WfVqRef;
if( AxisRsc->IntAdwk.swk1 < 0 )
{
    AxisRsc->IntAdwk.swk1 = (SHORT)ZEROR - AxisRsc->IntAdwk.swk1; /* TMP1 =  $|V_q|$ 
    IxNop( );
}

AxisRsc->IntAdwk.swk0 = AxisRsc->WeakFV.WfVqMax - AxisRsc->IntAdwk.swk1; /* TMP0 =
 $V_{qmax} - V_q = \Delta V_q$ 
AxisRsc->IntAdwk.swk0 = IxLmtCBS16( AxisRsc->IntAdwk.swk0 ); /*
偏差计算

```

$$\Delta V_q = V_{qmax} - V_q$$

```

#define LPX_REG16_MAX      32767
#define LPX_REG16_MIN      -32768

/*-----*/
/*      比例項計算      */
/*-----*/
AxisRsc->IntAdwk.lwk1 = (LONG)AxisRsc->IntAdwk.swk0; /* TMP1,0 = 符号拡張(TMP0) */
// #ifdef WIN32
AxisRsc->IntAdwk.swk2 = (SHORT)IlibASR64(( (INT64)AxisRsc->IntAdwk.lwk1 * (INT64)AxisRsc->WeakFV.WfKpV.I ), 32 );
// #elif defined(ASIP_CC)
//      AxisRsc->IntAdwk.swk2 = asr( AxisRsc->IntAdwk.lwk1 *
AxisRsc->WeakFV.WfKpV.I, 32 );
// #endif
if( AxisRsc->IntAdwk.swk2 > 0 )
{
    AxisRsc->IntAdwk.swk2 = LPX_REG16_MAX; /* 正の最大値 */
}
else if( AxisRsc->IntAdwk.swk2 < (SHORT)0xFF80 )
{
    AxisRsc->IntAdwk.swk2 = LPX_REG16_MIN; /* 負の最大値 */
}
else
{
    // #ifdef WIN32
    AxisRsc->IntAdwk.lwk2 = (LONG)IlibASR64(( (INT64)AxisRsc->IntAdwk.lwk1 *
(INT64)AxisRsc->WeakFV.WfKpV.I ), 16 );
    AxisRsc->IntAdwk.swk2 = (SHORT)IlibASR64(( (INT64)AxisRsc->IntAdwk.lwk2 * (INT64)256 ),
16 );
    // #elif defined(ASIP_CC)
    //      AxisRsc->IntAdwk.lwk2 = asr( AxisRsc->IntAdwk.lwk1 *
AxisRsc->WeakFV.WfKpV.I, 16 );
    //      AxisRsc->IntAdwk.swk2 = asr( AxisRsc->IntAdwk.lwk2 * 256, 16 );
    // #endif
}
比例項の計算

```

$$swk2 = \Delta V_q \times WfKp$$

在函数 **PcmWFPmCalc** 中

```

/*-----*/
/*      1/ω、1/ω^2の計算      */
/*-----*/
/* 速度の正規化を2^24→15000にする。 */
work1 = MlibLABS( MotSpd );

```

```

// work1 = MlibMulhigh32( work1, 38400 ); /* <S068> */
work1 = MlibMulhigh32( work1, 3840000 );

/* 速度の判定 */
if ( work1 < 1000 )
{
    /* 1000以下の時は、1000で計算する。 */
    work1 = 1000;
}

SpdInv = 16777216 / work1;          /* SpdInv = 1/ω */
SpdInv_2 = SpdInv * SpdInv;        /* SpdInv_2 = 1/ω^2 */

/*-----*/
/*      電圧フィードバックループゲイン計算      */
/*-----*/
/* ゲインは桁落ち対策で24bit左シフトされている。ASICで24bit右シフトする。 */
wf_ptr->var.Kpv = MlibMulgain32( SpdInv, wf_ptr->conf.KpvBeforeDiv );
wf_ptr->var.KpvL = (SHORT)(wf_ptr->var.Kpv & 0x0000ffff);
wf_ptr->var.KpvH = (SHORT)((wf_ptr->var.Kpv>>16) & 0x0000ffff);

/*****
/*
/*      電圧フィードバックループ比例ゲイン計算用定数の計算      */
/*
/*
/*****
/*
          2 π * Kpv * Tiv * 10^-6
/*      Kpv(Asic) = -----
/*
                      ωL
/*
          2 π * 10^-6 * Vdc      15000      15000      1
/*      = ----- * ----- * ----- * Kpv * Tiv * ----- * 2^24
/*
          pole * Lx * 8192      Nmax      Imax      ωm
/*      の ωmを除いた部分を計算する。2^40は桁落ち対策。
/*      1/ωmはScanBで行う。
/*
/*      1/ωmは、ScanBで2^24/ωmを計算して、掛ける。
/*
/*
/*      Kpv      [Hz] : 電圧フィードバックループゲイン  电压反馈环路增益
/*      Tiv      [us] : 電圧フィードバックループ積分時定数 电压反馈环路积分时间常数
/*      Lx      [H] : モータインダクタンス
/*      pole     [-] : ポール数
/*      Namx [rad/s] : オーバースピード
/*      ωm [15000/Nmax] : Motor Speed

```

```

/*                                                                 */
/*****
void PcalVFBKp( WEAKENFIELD *WeakenField, ASICS *SvAsicRegs, BPRMDAT *Bprm, USHORT
kv, USHORT tv )
{
    #if (FLOAT_USE==TRUE)

        float fw;

        fw = Bprm->Vdc * 62832.0F / (float)(81920000 * 2);
        fw = fw * (float)kv / Bprm->Keangle;
        fw = fw * (float)tv / Bprm->MotLd;
        fw = fw / 1000000.0F;
        fw = fw * 15000.0F / Bprm->OvrSpd;
        fw = fw * 15000.0F / Bprm->MaxCur;
        WeakenField->conf.KpvBeforeDiv = fw;

    #else

        LONG      kx, sx;

        kx = MlibScalKskxkx( Bprm->Vdc, 62832, 81920000*2, &sx, 0 );
        kx = MlibPcalKxkxks( kx, kv, Bprm->Keangle, &sx, 0 );
        kx = MlibPcalKxkxks( kx, tv, Bprm->MotLd, &sx, 0 );
        kx = MlibPcalKxgain( kx, 1, 1000000, &sx, 0 );
        kx = MlibPcalKxkxks( kx, 15000, Bprm->OvrSpd, &sx, 0 );
        kx = MlibPcalKxkxks( kx, 15000, Bprm->MaxCur, &sx, 24 );

        WeakenField->conf.KpvBeforeDiv = kx;

    #endif /* FLOAT_USE */

}

/*****
/*                                                                 */
/*      電圧フィードバックループ積分ゲインの計算      */
/*                                                                 */
/*****
/*                                                                 */
/*      Ts      2 π * Gain * Tv*10^-6      Ts      */
/*      Kiv(Asic) = Kpv * ----- = ----- * ----- */
/*      Tiv      ωL      Tiv      */
/*      2 π * Ts*10^-9 * Vdc      15000      15000      1      */
/*      = ----- * ----- * ----- * Kpv * ----- * 2^20 */

```

```

/*      pole * Lx * 8192      Nmax      Imax      ωm      */
/*      の ωmを除いた部分を計算する。2^20は桁落ち対策。      */
/*      1/ωmはScanBで行う。      */
/*      */
/*      1/ωmは、ScanBで2^24/ωmを計算して、掛ける。      */
/*      */
/*      Kpv      [Hz] : 電圧フィードバックループゲイン      */
/*      Tiv      [us] : 電圧フィードバックループ積分時定数      */
/*      Lx      [H] : モータインダクタンス      */
/*      pole      [-] : ポール数      */
/*      Namx [rad/s] : オーバースピード      */
/*      ωm [15000/Nmax] : Motor Speed      */
/*      */
/*****

```

```

void PcalVFBKi( WEAKENFIELD *WeakenField, ASICS *SvAsicRegs, BPRMDAT *Bprm, USHORT
kv)
{

```

```

#if (FLOAT_USE==TRUE)

```

```

    float fw;

```

```

    fw = Bprm->Vdc * 62832.0F / (float)(81920000 * 2);
    fw = fw / Bprm->Keangle;
    fw = fw * (float)kv / Bprm->MotLd;
    fw = fw * (float)KPI_MACYCLENS / 1000000000.0F;
    fw = fw * 15000.0F / Bprm->OvrSpd;
    fw = fw * 15000.0F / Bprm->MaxCur;
    fw = fw * 1048576.0F / 16777216.0F;
    Axis->weaken_field_ptr->conf.KivBeforeDiv = fw;

```

```

#else

```

```

    LONG      kx, sx;

```

```

    kx = MlibScalKsxkx( Bprm->Vdc, 62832, 81920000*2, &sx, 0 );
    kx = MlibPcalKxkxks( kx, 1, Bprm->Keangle, &sx, 0 );
    kx = MlibPcalKxkxks( kx, kv, Bprm->MotLd, &sx, 0 );
    kx = MlibPcalKxgain( kx, KPI_MACYCLENS, 1000000000, &sx, 0 );
    kx = MlibPcalKxkxks( kx, 15000, Bprm->OvrSpd, &sx, 0 );
    kx = MlibPcalKxkxks( kx, 15000, Bprm->MaxCur, &sx, 0 );
    kx = MlibPcalKxgain( kx, 1048576, 16777216, &sx, 24 );

```

```

    WeakenField->conf.KivBeforeDiv = kx;

```

```

#endif

```

```
}
```

```
/* **** */
/*      PnEB0 : 電圧フィードバックループ比例ゲイン [Hz]      默认値30      */
/* **** */
```

```
PRM_RSLT pncal_kv(PRM_ACCCMD Cmd, UINT ArrayIdx, AXIS_HANDLE *Axis, LONG *pValue)
```

```
{
```

```
    PRMDATA    *Prm;
```

```
    Prm = Axis->UniPrms->Prm;
```

```
    PcalVFBKp( &(Axis->BaseControls->WeakenField), Axis->SvAsicRegs,
               Axis->UniPrms->Bprm, Prm->kv, Prm->tv );
```

```
    PcalVFBKi( &(Axis->BaseControls->WeakenField), Axis->SvAsicRegs,
               Axis->UniPrms->Bprm, Prm->kv );
```

```
    return PRM_RSLT_SUCCESS;
```

```
}
```

```
/* **** */
/*      PnEB1 : 電圧フィードバック積分時定数 [us]      默认値16      */
/* **** */
```

```
PRM_RSLT pncal_tv(PRM_ACCCMD Cmd, UINT ArrayIdx, AXIS_HANDLE *Axis, LONG *pValue)
```

```
{
```

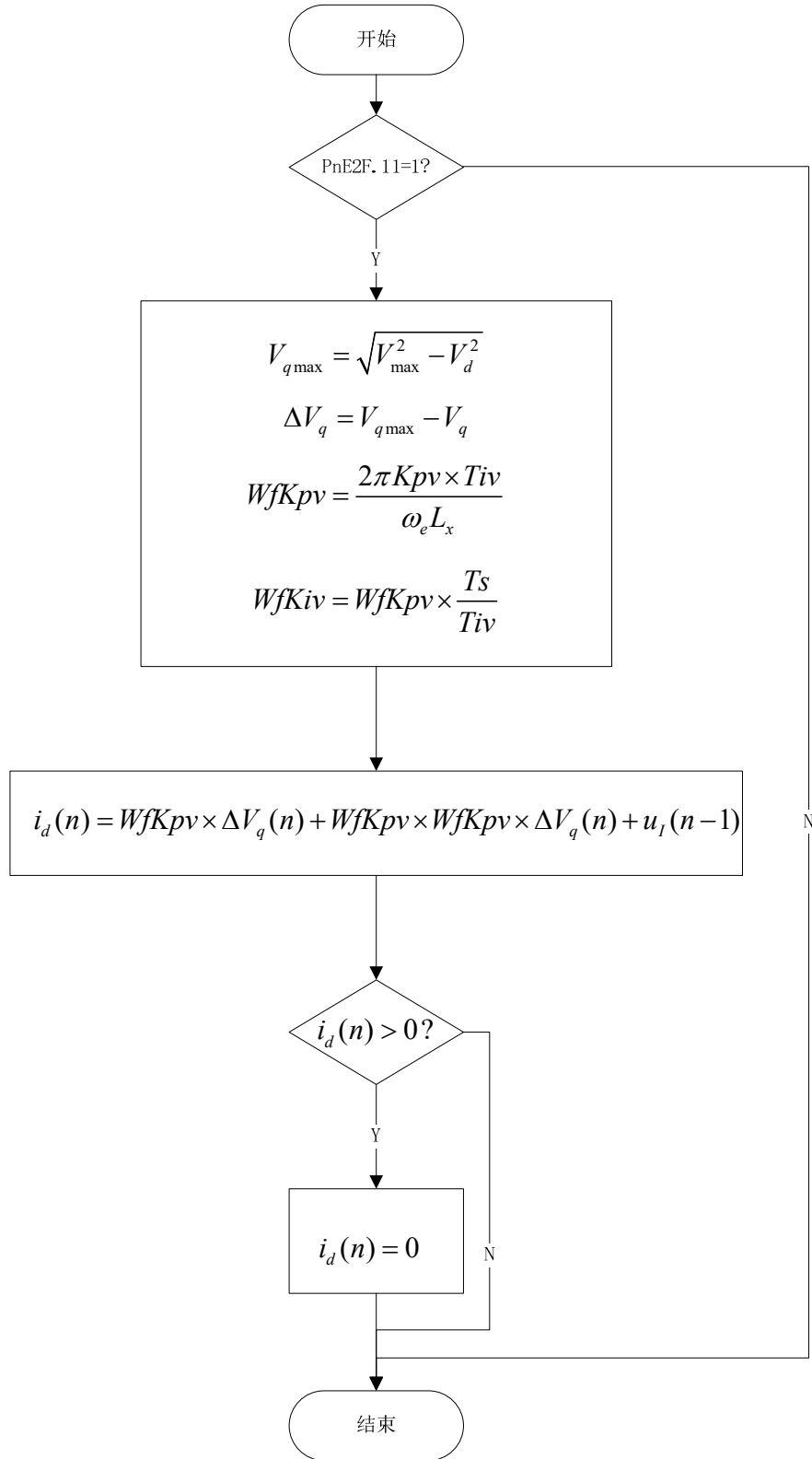
```
    PRMDATA    *Prm;
```

```
    Prm = Axis->UniPrms->Prm;
```

```
    PcalVFBKp( &(Axis->BaseControls->WeakenField), Axis->SvAsicRegs,
               Axis->UniPrms->Bprm, Prm->kv, Prm->tv );
```

```
    return PRM_RSLT_SUCCESS;
```

```
}
```



弱磁控制流程图

弱磁的理论是在基速以上如何分配 **dq** 轴电流，**d** 轴为负值起到弱磁升速的目的。分配 **dq** 轴电流可采用转矩特性测试分配，通过查找表进行；可通过 **PI** 配置转矩角进行分配；可通过逆变器的最大输出电压与实际电压的差值进行 **PI** 调节，输出 **d** 轴电流，进而通过电流极限圆确定 **q** 轴电流，电压反馈的方式，安川采用此方案；也可通过数学模型算出 **d** 轴电流，通

过电压前馈的方式补偿。

1.5 D 轴电流环计算

```
/*          2*PAI * Gain * Derate * MotLd      2^13      Imax * 512 */
/*      Kp = ----- * ----- * ----- */
/*                      100                      Vdc/2          15000 */

/*          Kp * Ts * 2^4 */
/*      Ki = ----- */
/*          1000 * Ti */
/*      Ts [ns] : 電流ループ演算周期 */
/*      Ti [us] : 電流ループ積分時定数 */
/*          Ld[H] */
/*      Tx[us] = ----- × 1000000 */
/*          Rx[0.001ohm] */
```

15000/Imax 其中 Imax 表示驱动器与电机最大电流其中的小值作为最大值，标么化值 15000 对应最大值。电流给定为 16 位的有符号数，为两次的最大值还多，留有余量。

2^14/Udc 表示 Udc 对应标么值 2^14，电流环的输出为 16 位有符号数，对应两次的 Udc。标么化的过程中，充分考虑了留有余量。其他的电阻电感等物理量折算过去就可以。

512 为移位用，后面计算的时候右移了 9 位，保证计算精度。Kp 为 Q9 格式，Ki 在 9 的基础上加 4，为 Q13 格式。Derate 为百分比

```
swk1 = sub_limitf(AxisRscI->WeakFV.IdOut, AxisRscI->IntAdV.IdInData);
/* TMP1 <-- limit( TMP1 , 2^15 - 1 ) */
/*-----*/
/*      TMP2 = limit( IntAdP.KdP * TMP1 / 2^9 , 2^15 - 1 ) */
/*-----*/
swk2 = mulshr_limitf(AxisRscI->IntAdP.KdP, swk1, 9); /* ACC <--
IntAdP.KdP * TMP1 */
/*-----*/
/*      IdIntgl(32) = (IntAdP.KdI * TMP1)<<3 + IdIntgl(32) */
/*      IDIH = limit( IDIH , IntAdP.VdLim ) */
/*-----*/
lwk4 = ((ULONG)AxisRscI->IntAdP.VdLim) << 16; /*
lwk6 = mul(AxisRscI->IntAdP.KdI, swk1 ) << 3; /*
AxisRscI->AcrV.IdIntgl.1 = add_limitf(lwk6, AxisRscI->AcrV.IdIntgl.1);
/* AcrV.IdIntgl <-- limit( AcrV.IdIntgl , 2^31 - 1 ) */
if( LPX_ABS(AxisRscI->AcrV.IdIntgl.1) > LPX_ABS(lwk4) )
{
    AxisRscI->StsFlg.CtrlStsRW = AxisRscI->StsFlg.CtrlStsRW | DLIM;
/*      积分限制值没看明白 */
swk0 = AxisRscI->IntAdP.CtrlSw;
AxisRscI->AcrV.IdIntgl.1 = cmove(((AxisRscI->IntAdP.CtrlSw &
```

```

ICLR) != 0), (LONG)ZEROR, AxisRscI->AcrV.IdIntgl.l);
}
/*-----*/
/*      VcmpV.VdOut = limit( TMP2 + IDIH +TMP3, 2^15 - 1 )      */
/*-----*/
swk1 = add_limitf(AxisRscI->AcrV.IdIntgl.s[1], swk2); /* TMP1 <--
limit( TMP1 , 2^15 - 1 )      */

```

增益输出为 Q15 格式，积分输出为 Q31 格式，其计算为 Q15+Q9+Q4+Q3=Q31,取高 16 位作为 Q15 格式，作为电流环的输出。

增益的计算值为：

$$k_{pwm} = \frac{2^{13}}{U_{dc}/2} \times \frac{I_{max}}{15000}$$

如果采用国际单位值，则该增益系数为 1。

$$\frac{I_{max}}{15000} = \frac{i_{实}}{i_{标}}$$

$$\frac{2^{13}}{U_{dc}/2} = \frac{u_{标}}{u_{实}}$$

$$i_{实} = u_{实}$$

则有

$$\frac{I_{max}}{15000} \times i_{标} = \frac{U_{dc}/2}{2^{13}} u_{标}$$

标么化可理解为实际值与标么值之间的来回转换。

```

/*-----*/
/*      filter : AcrV.VdFil = ( ( ( TMP1 - VDFH ) * IntAdP.Tfil ) << 2 )
+ AcrV.VdFil      */
/*-----*/
swk1 = sub_limitf(swk1, AxisRscI->AcrV.VdFil.s[1]); /* TMP1 <--
limit( TMP1 , 2^15 - 1 )      */
lwk0 = mul(AxisRscI->IntAdP.Tfil, swk1) << 2; /*
AxisRscI->AcrV.VdFil.l = add_limitf(AxisRscI->AcrV.VdFil.l, lwk0);
一阶低通滤波器 滤波时间常数 PnE26(默认为 0,少数为 100us)转换为增益 Tfil,
/*****
*****/
/*
/*      Ts * 2^14      */
/*      Kf = -----      */
/*      (1000*Tf) + Ts      */
/*
/*      Ts [ns] : 电流ループ演算周期      */

```

```

/*      Tf [us] : 電流ループフィルタ時定数  电流环滤波时间常数      */
/*
/*****
*****/

```

```

LONG    PcalDqAcrFilter( LONG Tf )

```

滤波时间常数的增益为 Q14 格式，后续右移 2 位，计算结果为 Q31 格式，取高 16 位数据作为 Q15 格式输出。用到的一阶低通滤波器大致如此。

1.6 q 轴电流环

```

if( AxisRscI->IntAdP.CtrlSw & LPFCDSABL) != 0 )
{
    AxisRscI->IntAdV.IqOut2Lpf.s[1] = AxisRscI->WeakFV.IqOut;    /* disable LPF      */
}
/*-----*/
else
{
    swk0 = sub_limitf(AxisRscI->WeakFV.IqOut, AxisRscI->IntAdV.IqOut2Lpf.s[1]); /* TMP0 <--
limit( TMP0, 2^15 - 1 )                                     */
    lwk2 = mul(AxisRscI->IntAdP.TLpf2, swk0) << 2;
    AxisRscI->IntAdV.IqOut2Lpf.l = add_limitf(AxisRscI->IntAdV.IqOut2Lpf.l, lwk2);
}
/*-----*/
AxisRscI->IntAdV.IqMonFil = AxisRscI->IntAdV.IqOut2Lpf.s[1]; /* IntAdV.IqMonFil:フィルタ後のq
軸電流(モニタ用)  <V224> */
AxisRscI->IntAdV.IqOfRef = add_limitf(AxisRscI->IntAdV.IqOut2Lpf.s[1], AxisRscI->IntAdV.IqDist);
/* IntAdV.IqOfRef <-- limit( IntAdV.IqOfRef , 2^15 - 1 )    <V224>      */

```

电流指令经过 hostint 滤波处理后，可通过选择进行一阶低通滤波器处理。

```

/*-----*/
/*      Torque Limit:          <V214>                                */
/*電圧フィードバック弱め界磁制御でd軸電流指令が作られるので、q軸電流指令は以下の式で*/
/* 电压反馈 弱磁控制 d轴电流指令的形成 q轴电流指令以下的形式形成                                */
/*      求めた値とトルクリミット設定値のいずれか小さい方でリミットする。                                */
/*       $I_q \times \text{リミット値 (極限值)} = \sqrt{I_{\text{max}}^2 - I_d^2}$                                 */
/*-----*/
/*      Id*による（根据）Torque Limit値                                */
/*-----*/
lwk2 = 0x0d693a40; /* 15000^2  电流最大值的平方                                */
swk0 = AxisRscI->IntAdP.CtrlSw;
swk1 = V_FB | V_FB2;
swk0 = swk0 & swk1; /* TMP0のbit11,bit13以外をマスクする                                */

```

```

if( swk0 != V_FB ) /* 控制模式的判断 若不为V_FB的弱磁方式 为实际的输出值 */
{
    lwk4 = mul(AxisRscI->WeakFV.IdOut, AxisRscI->WeakFV.IdOut); /* Idref^2
    ; 削除<V309>復活<V531> */
}
else /* 其他的 id为弱磁限制值 */
{
    lwk4 = mul(AxisRscI->WeakFV.WfldRefLim, AxisRscI->WeakFV.WfldRefLim); /* IdrefLim^2
    ; <V309> */
}
lwk2 = lwk2 - lwk4; /* I_max^2 - Id^2 */
swk0 = MpSQRT( lwk2 ); /*
swk1 = swk0; /* TMP0 = sqrt( I_max^2 - Id^2 ) 计算 iq 的最大值
iq 最大值的计算方式

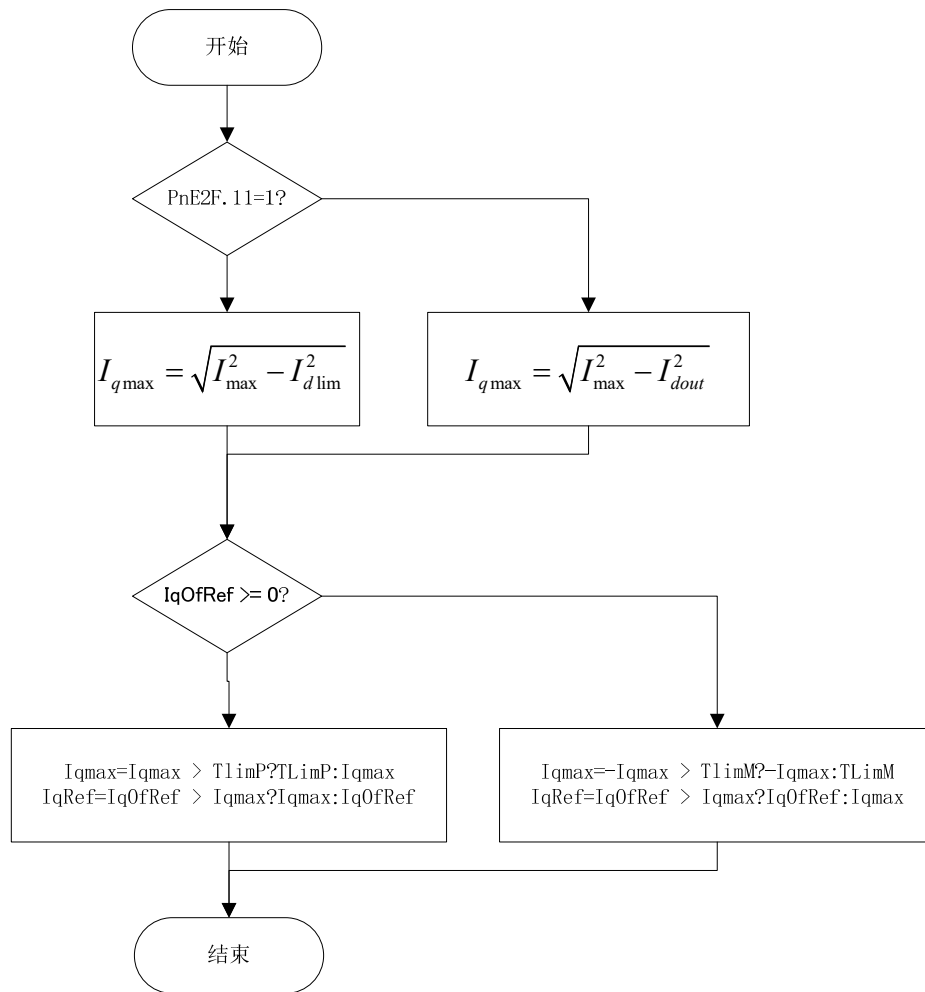
/*-----*/
/* Torque Limit */
/*-----*/
if( AxisRscI->IntAdV.IqOfRef >= 0 )
{
    swk1 = limit( swk1, AxisRscI->IntAdV.TLimP ); /* 正側トルクリミット */
    AxisRscI->IntAdV.IqRef = limit( AxisRscI->IntAdV.IqOfRef, swk1 ); /* <V224> 外乱トルク
加算後のq軸電流指令 */
    swk10 = AxisRscI->StsFlg.CtrlStsRW | TLIM; /* TLIM flag set */
    AxisRscI->StsFlg.CtrlStsRW = cmove((AxisRscI->IntAdV.IqRef == swk1), swk10,
AxisRscI->StsFlg.CtrlStsRW);
}
else
{
    swk1 = limit( swk1, AxisRscI->IntAdV.TLimM ); /* 負側トルクリミット */
    AxisRscI->IntAdV.IqRef = limit( AxisRscI->IntAdV.IqOfRef, swk1 ); /* <V224> 外乱トルク
加算後のq軸電流指令 */
    swk10 = AxisRscI->IntAdV.IqRef + swk1;
    swk11 = AxisRscI->StsFlg.CtrlStsRW | TLIM; /* TLIM flag set */
    AxisRscI->StsFlg.CtrlStsRW = cmove((swk10 == 0), swk11, AxisRscI->StsFlg.CtrlStsRW);
    /* TLIM flag set */
}
转矩限制根据转矩的设置限制值以及弱磁控制的限制值，选取其中的最小值
/*-----*/
/* TMP1 = limit( IntAdV.IqRef - IntAdV.IqInData , 2^15 - 1 ) */
/*-----*/
swk1 = sub_limitf(AxisRscI->IntAdV.IqRef, AxisRscI->IntAdV.IqInData); /* TMP1 <-- limit( TMP1 ,
2^15 - 1 ) */

```

```

/*-----*/
/*      TMP2 = limit( IntAdP.KqP * TMP1 / 2^9 , 2^15 - 1 )      */
/*-----*/
swk2 = mulshr_limitf(AxisRscI->IntAdP.KqP, swk1, 9); /* TMP2 <-- limit( TMP2 , 2^15 - 1 ) */
/*-----*/
/*      AcrV.IqIntgl(32) = (IntAdP.KqI * TMP1)<<3 + AcrV.IqIntgl(32)      */
/*      IQIH = limit( IQIH , IntAdP.VqLim )      */
/*-----*/
/*-积分的判断条件 以及积分饱和的限制 饱和时积分停止-----*/
if( ( (AxisRscI->IntAdP.CtrlSw & INT_ST) == 0) || ( (AxisRscI->StsFlg.IntglFlg & 1) == 0 ) )
{
    lwk6 = mul(AxisRscI->IntAdP.KqI, swk1);/* ACC <-- IntAdP.KqI * TMP1      */
    lwk4 = (ULONG)AxisRscI->IntAdP.VqLim; /*      */
    lwk4 = lwk4 << 16; /*      */
    lwk6 = lwk6 << 3; /*      */
    AxisRscI->AcrV.IqIntgl.l = add_limitf(lwk6, AxisRscI->AcrV.IqIntgl.l); /* AcrV.IqIntgl <--
limit( AcrV.IqIntgl , 2^32 - 1 )      */
    if( LPX_ABS(AxisRscI->AcrV.IqIntgl.l) > LPX_ABS(lwk4) )
    {
        AxisRscI->StsFlg.CtrlStsRW = AxisRscI->StsFlg.CtrlStsRW | QLIM; /* IMM3 <--
STAT | QLIM (imm_16)      */
        swk10 = AxisRscI->IntAdP.CtrlSw & ICLR;
        AxisRscI->AcrV.IqIntgl.l = cmove((swk10 != 0), (LONG)ZEROR, AxisRscI->AcrV.IqIntgl.l);
    }
}
/*-----*/
/*      VcmpV.VqOut = limit( TMP2 + IQIH +TMP3 , 2^15 - 1 )      */
/*-----*/
swk1 = add_limitf(AxisRscI->AcrV.IqIntgl.s[1], swk2); /* TMP1 <-- limit( TMP1 , 2^15 - 1 ) */
/*-----*/
/*      filter : AcrV.VqFil = ( ( ( TMP1 - VQFH ) * IntAdP.Tfil ) << 2 ) + AcrV.VqFil      */
/*-----*/
swk1 = sub_limitf(swk1, AxisRscI->AcrV.VqFil.s[1]); /* TMP1 <-- limit( TMP1 , 2^15 - 1 ) */
lwk0 = mul(AxisRscI->IntAdP.Tfil, swk1 ) << 2; /*      */
AxisRscI->AcrV.VqFil.l = add_limitf(AxisRscI->AcrV.VqFil.l, lwk0);
q 轴电流 PI 控制器，输出经过一个一阶低通滤波器处理。

```



转矩限制值的计算

内部转矩限制的计算会根据内部转矩限制，外部转矩限制，以及紧急停止转矩限制来进行计算，貌似汇川的也参照了此方式

/* 正侧トルク(電流)制限値 */

SvAsicRegs->AsicMicroReg->AdinV.TLimPIn =

(USHORT)MlibMulhigh32(BaseCtrls->TrqLimitData.var.PosTrqLmtOut,

3840000);

/* 負側トルク(電流)制限値 */

SvAsicRegs->AsicMicroReg->AdinV.TLimMIn =

(USHORT)MlibMulhigh32(BaseCtrls->TrqLimitData.var.NegTrqLmtOut,

-3840000);

将转矩设置的限制值由 Q24 格式转换为 15000 格式

/* **** */

/*

*/

/* トルクリミット演算処理 转矩限制计算处理

*/

/*

*/

/* **** */

/*

*/

/* 機能：パラメータで設定される内部トルク制限(Pn402,Pn403)，外部トルク制限(Pn404,Pn405)


```

RvTrqLmt = TrqLimits->conf.EstopTrqLmt;

LpxUvTrqLmtControl( &(TrqLimits->UvTrqLimits), TRUE, AlmMngr ); /* <S145> */
}
else
{
/*-----*/
/*      主回路電源電圧降下時のトルクリミット演算 主回路电源电压下降时的扭矩限制计算
                                          */
/*-----*/
/*-----*/
UvTrqLmt = LpxUvTrqLmtControl( &(TrqLimits->UvTrqLimits), FALSE, AlmMngr ); /*
<S145> */

/*-----*/
/*-----*/
/*      非OT時のトルクリミット演算      非OT时的扭矩限制计算
                                          */
/*-----*/
/*-----*/
    if( TrqLimits->var.PclSignal )
    {
        FwTrqLmt = ( TrqLimits->conf.FwIntTrqLmt < TrqLimits->conf.FwExtTrqLmt ) ?
                    TrqLimits->conf.FwIntTrqLmt:
                    TrqLimits->conf.FwExtTrqLmt;
    }
    else
    {
        FwTrqLmt = TrqLimits->conf.FwIntTrqLmt;
    }

    if( TrqLimits->var.NclSignal )
    {
        RvTrqLmt = ( TrqLimits->conf.RvIntTrqLmt < TrqLimits->conf.RvExtTrqLmt ) ?
                    TrqLimits->conf.RvIntTrqLmt:
                    TrqLimits->conf.RvExtTrqLmt;
    }
    else
    {
        RvTrqLmt = TrqLimits->conf.RvIntTrqLmt;
    }
/*-----*/
/*-----*/

```



```

        if( UvTrqLmt < FwTrqLmt )
        {
            FwTrqLmt = UvTrqLmt;
        }

        if( UvTrqLmt < RvTrqLmt )
        {
            RvTrqLmt = UvTrqLmt;
        }

        /*-----*/
        /*-----*/
        if( (pBaseCtrl->CtrlModeSet.CtrlMode.b.cm != CTRLMODE_TRQ)
            && (pBaseCtrl->CtrlModeSet.CtrlMode.b.cm != CTRLMODE_JOG) )
        {
            if( TrqLimits->conf.TrefTrqLmt || (TrqLimits->conf.TrefTrqLmtCL &&
TrqLimits->var.PclSignal) )
                { /* (Pn002.0=1:TLIMによるトルク制限有効) || ((Pn002.0=3:PCL/NCLによるトルク制
限有効) && (PCL状態)) */
                    lwk = MlibLABS( TrqLimits->var.FwTrqLmt );
                    if( lwk < FwTrqLmt )
                    {
                        FwTrqLmt = lwk;
                    }
                }

            /*-----*/
            /*-----*/
            if( TrqLimits->conf.TrefTrqLmt || (TrqLimits->conf.TrefTrqLmtCL &&
TrqLimits->var.NclSignal) )
                { /* (Pn002.0=1:TLIMによるトルク制限有効) || ((Pn002.0=3:PCL/NCLによるトルク制
限有効) && (NCL状態)) */
                    lwk = MlibLABS( TrqLimits->var.RvTrqLmt );
                    if( lwk < RvTrqLmt )
                    {
                        RvTrqLmt = lwk;
                    }
                }
        }
    }

    /*-----*/
    /*-----*/
    /*      トルクリミット演算結果出力 --> Kernel
        */
    /*-----*/

```

```

-----*/
    TrqLimits->var.PosTrqLmtOut = FwTrqLmt;          /* 正転側トルクリミット
    */
    TrqLimits->var.NegTrqLmtOut = -RvTrqLmt;         /* 逆転側トルクリミット
    */
}

```

1.7 电压补偿

```

if( (AxisRscI->IntAdP.CtrlSw & ISEL) != 0 )
{
    swk1 = AxisRscI->WeakFV.IdOut; /* TMP1 <-- reference ID          */
    swk2 = AxisRscI->IntAdV.IqRef; /*                               */
}
else
{
    swk1 = AxisRscI->IntAdV.IdInData; /* TMP1 <-- feedback ID      */
    swk2 = AxisRscI->IntAdV.IqInData; /* TMP2 <-- feedback IQ     */
}

```

电流的选择，是采用反馈值还是给定值，默认为给定值

```

/*-----*/
/*  TMP4(VcmpV.VdComp) = IntAdP.MotResist*TMP1/2^15 - VcmpV.LqC * TMP2 / 2^15  */
/*-----*/
swk4 = mulshr(AxisRscI->VcmpV.LqC, swk2, 15 ); /* VcmpV.VdComp <-- ACC >> 15 */
swk0 = mulshr(AxisRscI->IntAdP.MotResist, swk1, 15 );
swk4 = swk0 - swk4;
计算

```

$$swk4 = i_d R - \omega_e i_q L_q$$

```

/*-----*/
/*TMP5(VcmpV.VqComp) = VcmpV.LdC * TMP1 / 2^15 + VcmpV.MagC +
IntAdP.MotResist*TMP2/2^15                                     */
/*-----*/
swk3 = mulshr(AxisRscI->VcmpV.LdC, swk1, 15 ); /* TMP3 <-- ACC >> 15          */
swk0 = mulshr(AxisRscI->IntAdP.MotResist, swk2, 15 );
swk3 = swk3 + AxisRscI->VcmpV.MagC;
swk5 = swk3 + swk0; /* VcmpV.VqComp <-- VcmpV.MagC + TMP3 + TMP0 */
计算

```

$$swk5 = \omega_e i_d L_d + e + i_q R$$

```

/*-----*/
/*      if(IntAdP.CtrlSw & DIDTSET) VcmpV.VdComp = TMP4 + KDD * (IntAdV.IdDataP -
IntAdV.IdInData), IntAdV.IdDataP=IntAdV.IdInData      */
/*      VcmpV.VqComp = TMP5 + KQD * (IntAdV.IqDataP - IntAdV.IqRef),
IntAdV.IqDataP=IntAdV.IqRef      */
/*-----*/
if( (AxisRscI->IntAdP.CtrlSw & DIDTSEL) == 0 )
{
    AxisRscI->VcmpV.VdComp = swk4;      /*
    AxisRscI->VcmpV.VqComp = swk5;      /*
}
/*-----*/
/*      filter : I*FL = ( ( ( TMP1 - I*FH ) * IntAdP.Tfil ) << 2 ) + I*FL      */
/*-----*/
else
{
    swk1 = AxisRscI->WeakFV.IdOut;      /*
    swk1 = sub_limitf(swk1, AxisRscI->IntAdV.IdLfil.s[1]);      /*
    lwk0 = mul(AxisRscI->IntAdP.Tfil, swk1) << 2; /*
    AxisRscI->IntAdV.IdLfil.l = add_limitf(AxisRscI->IntAdV.IdLfil.l, lwk0);      /*
/*-----*/
    swk1 = AxisRscI->IntAdV.IqRef;      /*
    swk1 = sub_limitf(swk1, AxisRscI->IntAdV.IqLfil.s[1]);      /*
    lwk0 = mul(AxisRscI->IntAdP.Tfil, swk1) << 2;      /*
    AxisRscI->IntAdV.IqLfil.l = add_limitf(AxisRscI->IntAdV.IqLfil.l, lwk0);      /*
/*-----*/
    swk2 = AxisRscI->IntAdV.IdLfil.s[1] - AxisRscI->IntAdV.IdDataP; /*
    AxisRscI->IntAdV.IdDataP = AxisRscI->IntAdV.IdLfil.s[1];      /*
    swk2 = mulshr_limitf(AxisRscI->IntAdP.L_dIdt, swk2, 9); /* limit( VDL , 2^15 - 1 )
    AxisRscI->VcmpV.VdComp = add_limitf(swk2, swk4);      /* VcmpV.VdComp <--
limit( VcmpV.VdOut , 2^15 - 1 )      */
/*-----*/
    swk2 = AxisRscI->IntAdV.IqLfil.s[1] - AxisRscI->IntAdV.IqDataP; /*
    AxisRscI->IntAdV.IqDataP = AxisRscI->IntAdV.IqLfil.s[1];
    swk2 = mulshr_limitf(AxisRscI->IntAdP.L_dIdt, swk2, 9); /* limit( VQL , 2^15 - 1 )
    AxisRscI->VcmpV.VqComp = add_limitf(swk2, swk5);      /* VcmpV.VqComp <--
limit( VcmpV.VqOut , 2^15 - 1 )      */
}

```

此段程序选择是否将电压方程补全，是先电流低通滤波，再差分，然后乘以一个增益系数，作为前馈电压补偿量与电压方程的电压补偿量进行叠加，作为整个的电压补偿量。

$$u_d = R_s i_d + L_d \frac{di_d}{dt} - \omega_r L_q i_q$$

$$u_q = R_s i_q + L_q \frac{di_q}{dt} + \omega_r (L_d i_d + \psi_f)$$

```

/*****
/*電圧ベクトル補正值計算      <V537> 新弱め界磁制御以外はこの処理をジャンプする  */
/* 电压矢量补正值计算 除了新弱磁控制以外，跳至该处理。 */
/*****
if( AxisRscI->IntAdP.CtrlSw & V_FB2) != 0 )
{
/*****
/*      Get modulation      <V531> 変調率計算を移動      */
/*****
    lwk2 = mul(AxisRscI->VcmpV.VdOut, AxisRscI->VcmpV.VdOut);
    lwk2 = mac((LONG)AxisRscI->VcmpV.VqOut, (LONG)AxisRscI->VcmpV.VqOut, lwk2);
    swk0 = MpSQRT( lwk2 );          /* TMP0 =  $\sqrt{(V_{cmpV.VdOut}^2 + V_{cmpV.VqOut}^2)}$   */
    AxisRscI->IntAdV.V1 = swk0;      /* IntAdV.V1   = TMP0 计算电压矢量的幅值  */
/*****
/* 飽和判断      IntAdV.V1 > 8192*127%(10403.8) -> 飽和状態      */
/*****
    AxisRscI->VcmpV.Vmax2 = 10403; /* VcmpV.Vmax2 = 8192 * 1.27 8192代表什么值  */
    AxisRscI->VcmpV.V12 = AxisRscI->IntAdV.V1; /* VcmpV.V12 =  $\sqrt{(V_{cmpV.VdOut}^2 + V_{cmpV.VqOut}^2)}$   */
    //      swk10 = AxisRscI->VcmpV.Vmax2 >> 1;
    /* VcmpV.Vmax2 = 8192 * 1.27 / 2      */
    //      swk11 = AxisRscI->IntAdV.V1 >> 1;
    /* VcmpV.V12 =  $\sqrt{(V_{cmpV.VdOut}^2 + V_{cmpV.VqOut}^2)} / 2$  */
    swk10 = (USHORT)AxisRscI->VcmpV.Vmax2 >> 1; /* VcmpV.Vmax2 = 8192 * 1.27 / 2  */
    swk11 = (USHORT)AxisRscI->IntAdV.V1 >> 1; /* VcmpV.V12 =  $\sqrt{(V_{cmpV.VdOut}^2 + V_{cmpV.VqOut}^2)} / 2$  */
    /* 根据方向选取不同的值  */
    AxisRscI->VcmpV.Vmax2 = cmove((AxisRscI->IntAdV.V1 < 0), swk10,
AxisRscI->VcmpV.Vmax2);
    AxisRscI->VcmpV.V12 = cmove((AxisRscI->IntAdV.V1 < 0), swk11, AxisRscI->VcmpV.V12);
    if( AxisRscI->VcmpV.Vmax2 < AxisRscI->VcmpV.V12 )
    {
        AxisRscI->IntAdV.V1 = 10403;          /* IntAdV.V1 = IntAdP.Vmax( 8192 * 1.27 )  */
        AxisRscI->StsFlg.IntglFlg = AxisRscI->StsFlg.IntglFlg | 1; /* 積分停止フラグセット  */
    }
#if 1
    lwk2 = mul(AxisRscI->VcmpV.Vmax2, AxisRscI->VcmpV.VdOut);
    AxisRscI->VcmpV.VdOut = lwk2 / (LONG)AxisRscI->VcmpV.V12;
    lwk2 = mul(AxisRscI->VcmpV.Vmax2, AxisRscI->VcmpV.VqOut);
    //      div(lwk2, AxisRscI->VcmpV.V12, (INT*)&swk1, (INT*)&swk0);
    AxisRscI->VcmpV.VqOut = lwk2 / (LONG)AxisRscI->VcmpV.V12;
#else /* JL-076では除算命令が使用できなかったため以下の処理を実施していた */
/*****

```

```

/*      電圧ベクトル補正值計算      */
/*****
/*-----*/
/*      電圧制限テーブルアドレス取得      */
/*-----*/

    lwk2 = mul(AxisRscI->VcmpV.V12, AxisRscI->VcmpV.V12);/* TMP3,2 = VcmpV.V12^2*/
    lwk2 = lwk2 - 0x00400000;          /* TMP3,2 = IntAdV.V1^2 - 2^22      */
//    lwk2 = lwk2 >> 4;                /* TMP3,2 = (VcmpV.V12^2 - 2^22) / 2^4      */
    lwk2 = (ULONG)lwk2 >> 4;          /* TMP3,2 = (VcmpV.V12^2 - 2^22) / 2^4      */
    swk0 = (USHORT)( lwk2 >> 16 );/* TMP0 = (VcmpV.V12^2 - 2^22) / 2^4 / 2^16 = addr*/
    lwk2 = lwk2 & 0x0000ffff; /* TMP2 = { (VcmpV.V12^2 - 2^22) / 2^4 } & 0x0000ffff      */
/*-----*/
/*      電圧制限ベクトル直線補間用データ取得      */
/*-----*/

    lwk4 = 65536;                    /* TMP5,TMP4 = 65536      */
    lwk6 = lwk4 - lwk2;              /* TMP7,6 = 10000h - Table Index (Lo) -> (addr*2^16-low) */
    lwk2 = lwk2 & 0x0000ffff; /* TMP8 : テーブルデータ読み出し(読み出しアドレスaddr)
    /*/* tanaka21,コンパイラ対応待ち      */
    lwk6 = (ULONG)swk8 * lwk6;        /* TMP6 = tblrv(addr)*(2^16-low)      */
    swk0 = swk0 + 1;                /* TMP0 = addr+1      */
    lwk2 = lwk2 & 0x0000ffff; /* TMP8 : テーブルデータ読み出し(読み出しアドレス
    addr+1) /*/* tanaka21,コンパイラ対応待ち      */
    lwk4 = (ULONG)swk8 * lwk2;        /* TMP4 = tblrv(addr+1)*low      */
    lwk0 = lwk6 + lwk4;              /* TMP0 = tblrv(addr)*(2^16-low) + tblrv(addr+1)*low */
/*-----*/
/*      電圧電圧ベクトル補正值計算      */
/*-----*/

    swk8 = AxisRscI->VcmpV.Vmax2;      /* TMP8 = VcmpV.Vmax2      */
    lwk2 = mulshr((ULONG)swk8, lwk0, 28 ); /* TMP2 = MAC / 2^28      */
//    AxisRscI->VcmpV.VdOut = mulshr(swk2, AxisRscI->VcmpV.VdOut, 14 ); /*
VcmpV.VdOut = IntAdP.Vmax / VcmpV.V12 * VcmpV.VdOut * 2^(13+13+16) / 2^(28+14) */
//    AxisRscI->VcmpV.VqOut = mulshr(swk2, AxisRscI->VcmpV.VqOut, 14 );
/* VcmpV.VqOut = IntAdP.Vmax / VcmpV.V12 * VcmpV.VqOut * 2^(13+13+16) / 2^(28+14)
*/
    AxisRscI->VcmpV.VdOut = mulshr((SHORT)lwk2, AxisRscI->VcmpV.VdOut, 14 );/*
VcmpV.VdOut = IntAdP.Vmax / VcmpV.V12 * VcmpV.VdOut * 2^(13+13+16) / 2^(28+14) */
    AxisRscI->VcmpV.VqOut = mulshr((SHORT)lwk2, AxisRscI->VcmpV.VqOut, 14 );/*
VcmpV.VqOut = IntAdP.Vmax / VcmpV.V12 * VcmpV.VqOut * 2^(13+13+16) / 2^(28+14) */
#endif /* JL-076では除算命令が使用できなかったため上記の処理を実施していた */

}
else
{
    AxisRscI->StsFlg.IntglFlg = AxisRscI->StsFlg.IntglFlg & 0xFFFE; /* 積分停止フラグクリ*/

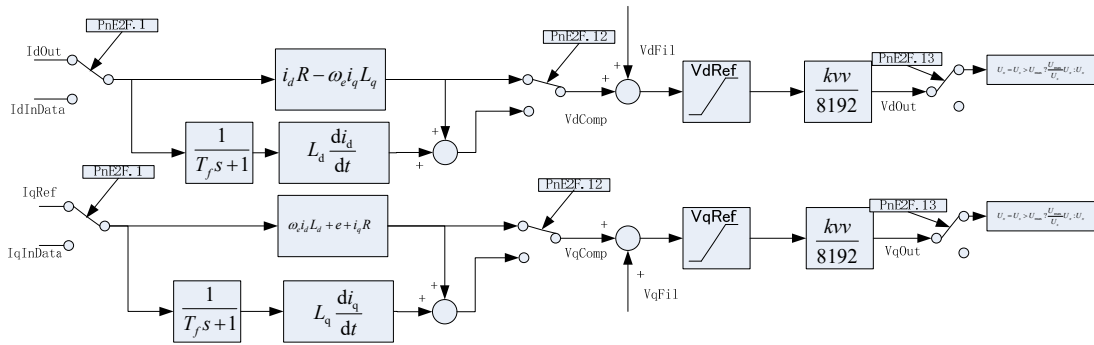
```

}
}

d,q 轴电压饱和的判断，若出现饱和，则 d,q 轴电压输出按比例调节，此处是否相当于过调制处理，增加母线电压利用率？其关键在于 $V_{\max 2}$ 和 V_{12} 的确定计算

$$V_{dout} = \frac{V_{\max 2}}{V_{12}} V_{dout}$$

$$V_{qout} = \frac{V_{\max 2}}{V_{12}} V_{qout}$$



$$U_s = \sqrt{U_d^2 + U_q^2}$$

$$U_{\max} = \frac{U_{dc}}{2} (8192) \times 1.27 = 10403$$

$$U_{dref} = 10430$$

电压补偿方式根据电压方程进行补偿，根据电压饱和的判断，若饱和置饱和标志位，**ACRq** 则不进行积分项。此处电压最大值为何选取 SPWM 的相电压最大值的 1.27 倍，目前不清楚，若采用三次谐波注射法，应该跟 SVPWM 一样是提高 15%。

```

/*****
/*
/*      UVW transform : dq( 2phase ) to UVW( 3phase ) Transform      */
/*
/*
/*-----*/
/*  VcmpV.VuOut = limit( SinTbl.CosT * VcmpV.VdOut / 2^14 - SinTbl.SinT * VcmpV.VqOut /
2^14 , 2^15 - 1 )
/*-----*/
/*
swk4 = AxisRscI->IntAdP.Vmax; /*
swk1 = mulshr(AxisRscI->SinTbl.CosT, AxisRscI->VcmpV.VdOut, 14 ); /* TMP1 <-- ACC >> 14*/
swk2 = mulshr(AxisRscI->SinTbl.SinT, AxisRscI->VcmpV.VqOut, 14 ); /* TMP2 <-- ACC >> 14*/
AxisRscI->VcmpV.VuOut = sub_limitf(swk1, swk2); /* VcmpV.VuOut <-- limit( VcmpV.VuOut , 2^15
- 1 )
/*

```

```

AxisRscI->VcmpV.VuOut = IxLIMIT( AxisRscI->VcmpV.VuOut, swk4 ); /* */
/*-----*/
/*      VcmpV.VvOut = limit( SinTbl.CosT3 * VcmpV.VdOut / 2^14 - SinTbl.SinT3 *
VcmpV.VqOut / 2^14 , 2^15 - 1 ) */
/*-----*/
swk1 = mulshr(AxisRscI->SinTbl.CosT3, AxisRscI->VcmpV.VdOut, 14 ); /* TMP1 <-- ACC >> 14 */
swk2 = mulshr(AxisRscI->SinTbl.SinT3, AxisRscI->VcmpV.VqOut, 14 ); /* TMP2 <-- ACC >> 14 */
AxisRscI->VcmpV.VvOut = sub_limitf(swk1, swk2); /* VcmpV.VvOut <-- limit( VcmpV.VvOut , 2^15
- 1 ) */
AxisRscI->VcmpV.VvOut = IxLIMIT( AxisRscI->VcmpV.VvOut, swk4 ); /* */
/*-----*/
/*      VcmpV.VwOut = limit( - VcmpV.VuOut - VcmpV.VvOut , 2^15 - 1 ) */
/*-----*/
swk1 = (SHORT)ZEROR - AxisRscI->VcmpV.VuOut; /* VcmpV.VwOut <-- - VcmpV.VuOut -
VcmpV.VvOut */
AxisRscI->VcmpV.VwOut = sub_limitf(swk1, AxisRscI->VcmpV.VvOut); /* VcmpV.VwOut
<-- limit( VcmpV.VwOut , 2^15 - 1 ) */
AxisRscI->VcmpV.VwOut = IxLIMIT( AxisRscI->VcmpV.VwOut, swk4 ); /* */
两项旋转 d,q 轴电压到三相静止 U,V,W 的变换

```

$$\begin{bmatrix} V_U \\ V_V \\ V_W \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos \theta & -\sin \theta \\ \cos\left(\theta - \frac{2}{3}\pi\right) & -\sin\left(\theta - \frac{2}{3}\pi\right) \\ \cos\left(\theta + \frac{2}{3}\pi\right) & -\sin\left(\theta + \frac{2}{3}\pi\right) \end{bmatrix} \begin{bmatrix} V_d \\ V_q \end{bmatrix}$$

```

/*****
/*新弱め界磁制御判断処理<V537> 新弱め界磁の場合変調率計算, 飽和判断処理をジャンプす*/
/* 新弱磁場的调制率计算, 跳转饱和判断处理*/
*****/
if( AxisRscI->IntAdP.CtrlSw & V_FB2) == 0 )
{
/*****
/*      Get modulation      <V531> 変調率計算は2相3相変換前にする <V537> 復活 */
*****/
    lwk2 = mul(AxisRscI->VcmpV.VdOut, AxisRscI->VcmpV.VdOut);
    lwk2 = mac((LONG)AxisRscI->VcmpV.VqOut, (LONG)AxisRscI->VcmpV.VqOut, lwk2);
    swk0 = MpSQRT( lwk2 );
    if( (USHORT)swk0 > 0x7FFF )
    {
        swk0 = 0x7FFF; /* √の計算が32767を超えたら、32767にする。

```

```

; <V350> */
}
AxisRscI->IntAdV.V1 = swk0;
/*-----*/
/*      饱和判断                      <V531> <V537> 复活                      */
/*-----*/
AxisRscI->StsFlg.IntglFlg = AxisRscI->StsFlg.IntglFlg & 0xFFFE;          /*
swk10 = AxisRscI->StsFlg.IntglFlg | 1;                                /*
AxisRscI->StsFlg.IntglFlg = cmove((AxisRscI->IntAdV.V1 >= 9421), swk10,
AxisRscI->StsFlg.IntglFlg);
}

```

1.8 过调制

```

/*****
/*      Over modulation type select                      */
/*****
if( AxisRscI->IntAdP.Vmax >= 0x2000 ) /* PnE2B=0x7FFF首先判断最大值是否大于0x2000(8192)
    */
{
    if( AxisRscI->IntAdP.CtrlSw & OVMSEL2) == 0 ) /* bit9=0 默认选择过调制1          */
    {
        /* V1为实际合成矢量电压值的值是否大于0x2000 bit8=1且是否选择了过调制1      */
        if( ( AxisRscI->IntAdV.V1 >= 0x2000 ) && ( AxisRscI->IntAdP.CtrlSw & OVMSEL1) != 0 )
        {
/*****
/*      Over modulation1                      */
/*****
            IxSetCtblAdr( pCtbl, &(OVMODTBLG[0][0]) ); /* gain type 获得增益首地址*/
            MpOVMMODK( &AxisRscI->IntAdP, &AxisRscI->IntAdV, pCtbl ); /*增益的计算 */
            /* 根据增益的调整三相电压的输出 公式为Vx = Kmold/0x2000 * Vx          */
            AxisRscI->VcmpV.VuOut = mulshr_limitf(AxisRscI->VcmpV.VuOut,
AxisRscI->IntAdP.Kmod, 13);
            AxisRscI->VcmpV.VvOut = mulshr_limitf(AxisRscI->VcmpV.VvOut,
AxisRscI->IntAdP.Kmod, 13);
            AxisRscI->VcmpV.VwOut = mulshr_limitf(AxisRscI->VcmpV.VwOut,
AxisRscI->IntAdP.Kmod, 13);
/*-----*/
/*      TMP1 = |VcmpV.VuOut|,      TMP2 = |VcmpV.VvOut|,      TMP3 = |VcmpV.VwOut| */
/*      TMP4 = sign(VcmpV.VuOut), TMP5 = sign(VcmpV.VvOut), TMP6 = sign(VcmpV.VwOut)*/

```


/*---以下程序段目的在于找出最大值 计算补偿值 补偿值的计算为 $V_{x\max} - 0x2000$ ---*/

```

    swk0 = 1;
    swk4 = IxLIMIT( AxisRscI->VcmpV.VuOut, swk0 );
    swk1 = swk4 * AxisRscI->VcmpV.VuOut;
    swk5 = IxLIMIT( AxisRscI->VcmpV.VvOut, swk0 );
    swk2 = swk5 * AxisRscI->VcmpV.VvOut;
    swk6 = IxLIMIT( AxisRscI->VcmpV.VwOut, swk0 );
    swk3 = swk6 * AxisRscI->VcmpV.VwOut;
    if( swk1 >= swk2 )
    {
        if( swk1 >= swk3 )
        {
            swk1 = swk1 - 0x2000; /* TMP1 <-- |VcmpV.VuOut|-2000h */
            IxLmtzImm16( swk1, 0x7fff ); /* zero limit */
            swk0 = swk4 * swk1;
        }
        else
        {
            swk3 = swk3 - 0x2000; /* TMP0 <-- |VcmpV.VwOut|-2000h */
            IxLmtzImm16( swk3, 0x7fff ); /* zero limit */
            swk0 = swk6 * swk3;
        }
    }
    else
    {
        if( swk2 >= swk3 )
        {
            swk2 = swk2 - 0x2000; /* TMP0 <-- |VcmpV.VvOut|-2000h */
            IxLmtzImm16( swk2, 0x7fff ); /* zero limit */
            swk0 = swk5 * swk2;
        }
        else
        {
            swk3 = swk3 - 0x2000; /* TMP0 <-- |VcmpV.VwOut|-2000h */
            IxLmtzImm16( swk3, 0x7fff ); /* zero limit */
            swk0 = swk6 * swk3;
        }
    }
    /* 最终三相电压的输出为  $V_x - \text{补偿值}$  */
    AxisRscI->VcmpV.VuOut = sub_limitf(AxisRscI->VcmpV.VuOut, swk0); /* */
    AxisRscI->VcmpV.VvOut = sub_limitf(AxisRscI->VcmpV.VvOut, swk0); /* */
    AxisRscI->VcmpV.VwOut = sub_limitf(AxisRscI->VcmpV.VwOut, swk0); /* */
    AxisRscI->IntAdV.Vcent = swk0;
}

```

```

    }
    /**-----*/
    /*      Over modulation2                                     */
    /**-----*/
    else
    {
        IxSetCtblAdr( pCtbl, &(OVMODTBLO[0][0]) );    /* offset type */
        MpOVMMODK( &AxisRscI->IntAdP, &AxisRscI->IntAdV, pCtbl );
    }
    /*-----*/
    /*      MAX = TMP1, MIN = TMP2                               */
    /*      OFS = (TMP1+TMP2)/2                                   */
    /*-----以下程序段的目的在于找出三相的最大值和最小值 并求出平均值-----*/
    if( AxisRscI->VcmpV.VuOut >= AxisRscI->VcmpV.VvOut )
    {
        swk1 = AxisRscI->VcmpV.VuOut;
        swk2 = AxisRscI->VcmpV.VvOut;
    }
    else
    {
        swk1 = AxisRscI->VcmpV.VvOut;
        swk2 = AxisRscI->VcmpV.VuOut;
    }
    if( swk1 < AxisRscI->VcmpV.VwOut )
    {
        swk1 = AxisRscI->VcmpV.VwOut;
    }
    else
    {
        if( AxisRscI->VcmpV.VwOut < swk2 )
        {
            swk2 = AxisRscI->VcmpV.VwOut;
        }
    }
    swk0 = add_limitf(swk2, swk1); /*
    swk0 = mulshr(swk0, ONE, 1);
    /*---此处三相的输出为 Vx = Vx - swk0(最小值与最大值的平均值)-----*/
    AxisRscI->VcmpV.VuOut = sub_limitf(AxisRscI->VcmpV.VuOut, swk0); /* */
    AxisRscI->VcmpV.VvOut = sub_limitf(AxisRscI->VcmpV.VvOut, swk0); /* */
    AxisRscI->VcmpV.VwOut = sub_limitf(AxisRscI->VcmpV.VwOut, swk0); /* */
    AxisRscI->IntAdV.Vcent = swk0;
    /*-----*/
    swk0 = 1;
    /*-----*/
    swk0 = IxLIMIT( AxisRscI->VcmpV.VuOut, swk0 ); /* TMP1= -1/0/+1 */

```

```

    swk1 = swk1 | 1;          /* TMP1 = -1/+1 方向判别sign(VcmpV.VuOut)          */
    swk2 = swk1 * AxisRscI->IntAdP.Kmod;
    AxisRscI->VcmpV.VuOut = add_limitf( swk2, AxisRscI->VcmpV.VuOut );/*加增益系数*/
/*-----*/
    swk1 = IxLIMIT( AxisRscI->VcmpV.VvOut, swk0 );
    swk1 = swk1 | 1;          /* sign(VcmpV.VvOut)                      */
    swk2 = swk1 * AxisRscI->IntAdP.Kmod;
    AxisRscI->VcmpV.VvOut = add_limitf( swk2, AxisRscI->VcmpV.VvOut ); /*          */
/*-----*/
    swk1 = IxLIMIT( AxisRscI->VcmpV.VwOut, swk0 );
    swk1 = swk1 | 1;          /* sign(VcmpV.VwOut)                      */
    swk2 = swk1 * AxisRscI->IntAdP.Kmod;
    AxisRscI->VcmpV.VwOut = add_limitf( swk2, AxisRscI->VcmpV.VwOut ); /*          */
}
}

```

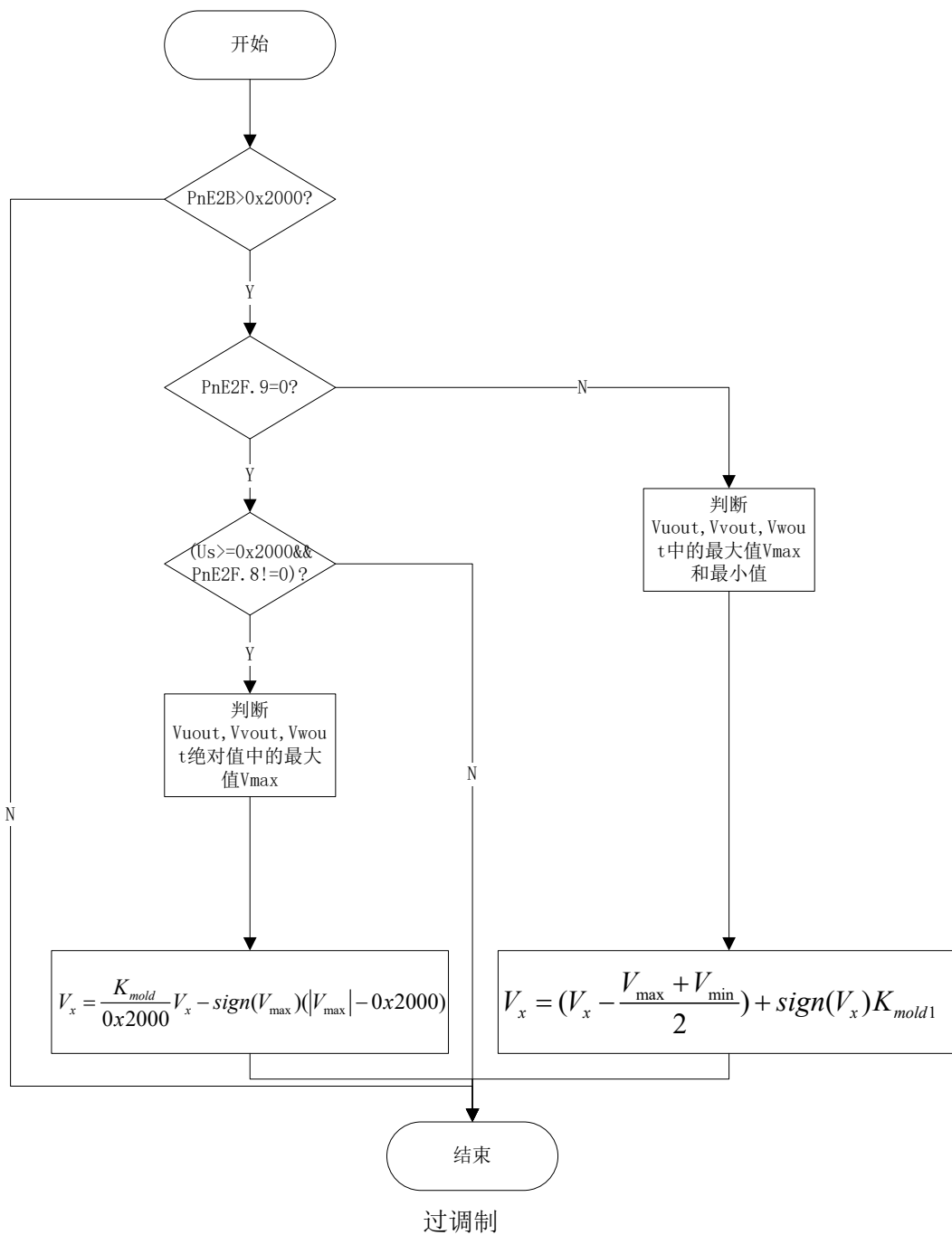
过调制采用了两种方式进行选择，方式 1 的公式为

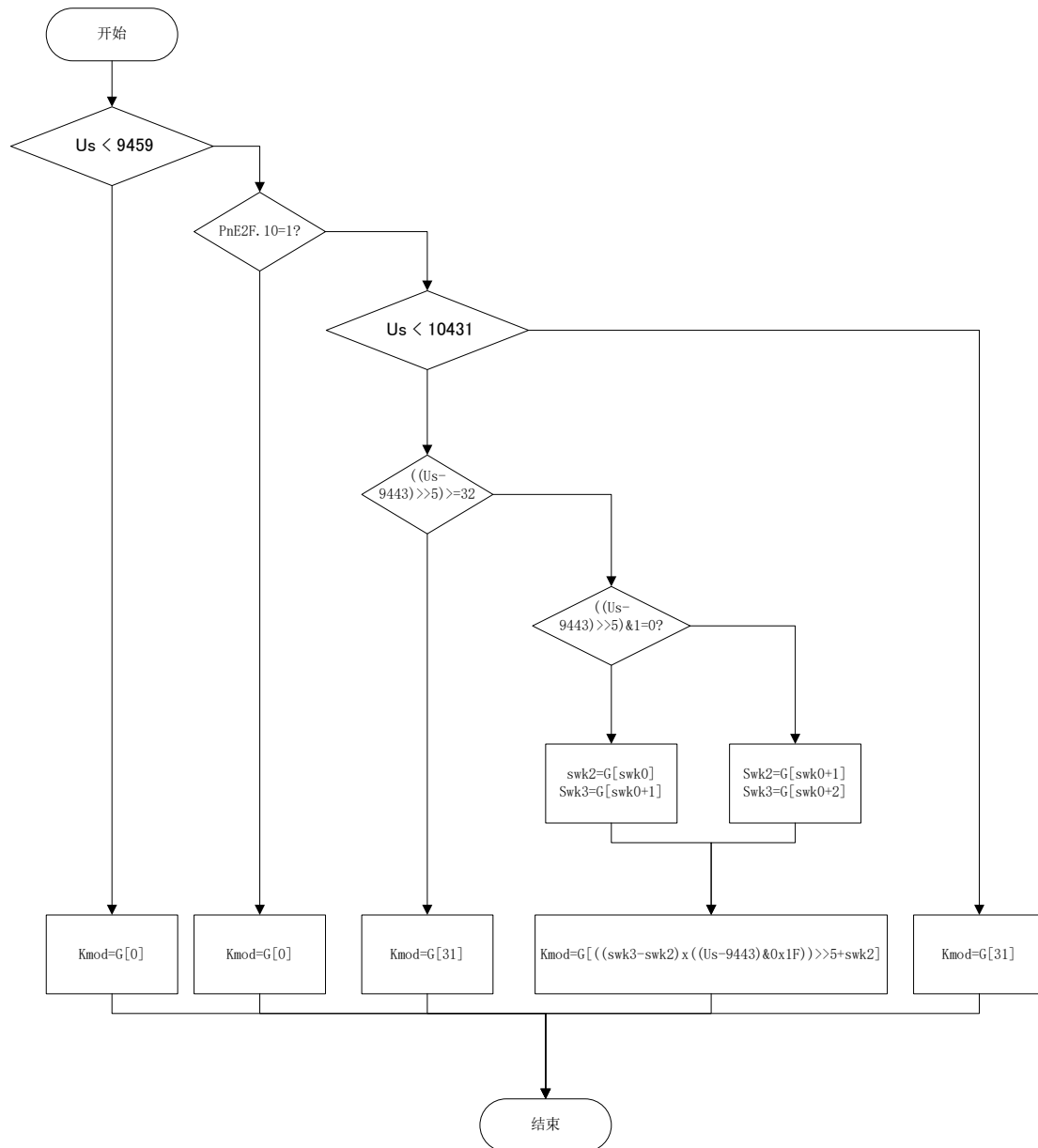
$$V_x = \frac{K_{mold}}{0x2000} V_x - sign(V_{\max})(|V_{\max}| - 0x2000)$$

方式 2 的公式为

$$V_x = (V_x - \frac{V_{\max} + V_{\min}}{2}) + sign(V_x)K_{mold1}$$

条件的判别需要注意





过调制增益的计算

过调制关键在于过调制增益的计算，采用查表的方式进行，共 32 个数据，调制方式 1 从 8192 递增至 14336；调制方式 2 从 0 递增至 8191。

```

/*-----*/
/*      Over Modulation Gain table      */
/*      @OVMODTBLG: OVMOD1 compensation gain    */
/*      @OVMODTBLO: OVMOD2 compensation offset    */
/*-----*/
#pragma arm section rodata = "MICRO_TABLE_OVDATA" /*ELF_LOAD*/
CSHORT OVMODTBLG[2] = {
    { 8192, 8193 },          /* 0, 1          */
    { 8197, 8205 },          /* 2, 3          */
    { 8216, 8229 },          /* 4, 5          */
    { 8244, 8263 },          /* 6, 7          */

```

```

        { 8284, 8308 },          /* 8, 9          */
        { 8335, 8365 },          /* 10,11         */
        { 8399, 8437 },          /* 12,13         */
        { 8479, 8525 },          /* 14,15         */
        { 8577, 8635 },          /* 16,17         */
        { 8699, 8771 },          /* 18,19         */
        { 8851, 8941 },          /* 20,21         */
        { 9044, 9162 },          /* 22,23         */
        { 9298, 9457 },          /* 24,25         */
        { 9648, 9883 },          /* 26,27         */
        { 10184, 10599 },        /* 28,29         */
        { 11263, 14336 },        /* 30,31         */
};
CSHORT OVMDTBLO[2] = {
    { 0, 0 },                    /* 0, 1          */
    { 4, 11 },                  /* 2, 3          */
    { 21, 33 },                 /* 4, 5          */
    { 49, 67 },                 /* 6, 7          */
    { 88, 112 },                /* 8, 9          */
    { 140, 173 },               /* 10,11         */
    { 211, 255 },               /* 12,13         */
    { 306, 368 },               /* 14,15         */
    { 443, 539 },               /* 16,17         */
    { 675, 851 },               /* 18,19         */
    { 1041, 1246 },             /* 20,21         */
    { 1465, 1705 },             /* 22,23         */
    { 1972, 2263 },             /* 24,25         */
    { 2596, 2988 },             /* 26,27         */
    { 3444, 4032 },             /* 28,29         */
    { 4910, 8191 },            /* 30,31         */
};

```

1.9 死区补偿

```

/*****
/*      On-Delay
*****/
/*-----*/
/*  IU, IV reference calc 根据d,q轴电流给定计算U,V,W电流输出值 为什么要这样 */
/*-----  $I_{uout} = I_d \cos \theta - I_q \sin \theta$  -----*/

swk1 = mulshr(AxisRscI->WeakFV.IdOut, AxisRscI->SinTbl.CosT, 14 ); /* TMP1 <-- ACC >> 14*/
swk2 = mulshr(AxisRscI->IntAdV.IqRef, AxisRscI->SinTbl.SinT, 14 ); /* TMP2 <-- ACC >> 14*/

```

```

AxisRscI->IntAdV.IuOut = swk1 - swk2; /* IntAdV.IuOut  <--  TMP1 - TMP2          */

/*-----  $I_{vout} = I_d \cos\left(\theta - \frac{2}{3}\pi\right) - I_q \sin\left(\theta - \frac{2}{3}\pi\right)$  -----*/

swk3 = mulshr(AxisRscI->WeakFV.IdOut, AxisRscI->SinTbl.CosT3, 14 ); /* TMP3 <-- ACC >> 14 */
swk4 = mulshr(AxisRscI->IntAdV.IqRef, AxisRscI->SinTbl.SinT3, 14 ); /* TMP4 <-- ACC >> 14 */
AxisRscI->IntAdV.IvOut = swk3 - swk4; /* IntAdV.IvOut  <--  TMP3 - TMP4          */
/*****
//      if ( |IntAdV.IuInData| < IntAdP.OnDelayLvl ) TMP1 = IntAdV.IuOut  /* Reference */
//      else                                     TMP1 = IntAdV.IuInData
//      if ( |IntAdV.IvInData| < IntAdP.OnDelayLvl ) TMP2 = IntAdV.IvOut  /* Reference */
//      else                                     TMP2 = IntAdV.IvInData
//      if ( |IWD| < IntAdP.OnDelayLvl ) TMP2 = IWO  /* Reference */
//      else                                     TMP2 = IWD
//      根据一个设置值来判断是选择反馈值还是给定值
*****/
swk5 = AxisRscI->IntAdP.OnDelayLvl;
if(LPX_ABS(AxisRscI->IntAdV.IuInData) > LPX_ABS(swk5)) //110530tanaka21作業メモ swk2を以降使わないため代入は行なわない
{
    swk1 = AxisRscI->IntAdV.IuInData; /* TMP1 <-- IntAdV.IuInData          */
}
else
{
    swk1 = AxisRscI->IntAdV.IuOut; /* TMP1 <-- IntAdV.IuOut          */
}
if( LPX_ABS(AxisRscI->IntAdV.IvInData) > LPX_ABS(swk5) ) //110530tanaka21作業メモ swk2を以降使わないため代入は行なわない
{
    swk2 = AxisRscI->IntAdV.IvInData; /* TMP2 <-- IntAdV.IvInData          */
}
else
{
    swk2 = AxisRscI->IntAdV.IvOut; /* TMP2 <-- IntAdV.IvOut          */
}
swk3 = -AxisRscI->IntAdV.IuInData - AxisRscI->IntAdV.IvInData; /* TMP3(IWD) <-- - TMP1 - TMP2          */
if( LPX_ABS(swk3) <= LPX_ABS(swk5) ) //110530tanaka21作業メモ swk4を以降使わないため代入は行なわない
{
    swk3 = -AxisRscI->IntAdV.IuOut - AxisRscI->IntAdV.IvOut; /* TMP3          */
}
swk7 = 0x2000; /* TMP7 <-- 2000h          */
swk5 = 1; /* TMP5 <-- 1          */

```

```

/*-----*/
/*      if(IntAdP.OnDelaySlope != 0) trapezoid type else rectangle type      */
/*-----根据死区等级来选择不同的补偿方式 矩形类型 直接补偿死区时间-----*/
if( AxisRscI->IntAdP.OnDelaySlope == 0 )
{
/*-----*/
/*      TMP1(ONDVU) = sign(IU)*IntAdP.OnDelayComp  根据方向来判别补偿量 */
/*-----*/

    swk6 = IxLIMIT( swk1, swk5 ); /* TMP6 = -1/0/+1 */
    swk1 = AxisRscI->IntAdP.OnDelayComp * swk6;

/*-----*/
/*      TMP2(ONDVU) = sign(IV)*IntAdP.OnDelayComp */
/*-----*/

    swk6 = IxLIMIT( swk2, swk5 );
    swk2 = AxisRscI->IntAdP.OnDelayComp * swk6;

/*-----*/
/*      TMP3(ONDVU) = sign(IW)*IntAdP.OnDelayComp */
/*-----*/

    swk6 = IxLIMIT( swk3, swk5 );
    swk3 = AxisRscI->IntAdP.OnDelayComp * swk6;
}

/*-----*/
/*      trapezoid type 梯形类型 在低电流处做一个斜坡处理来补偿死区时间 */
/*-----*/
else
{
    swk0 = mulshr_limitf(AxisRscI->IntAdP.OnDelaySlope, swk1, 8 ); /* TMP0 <--
IU*IntAdP.OnDelaySlope>>8 */
    swk0 = IxLIMIT( swk0, 8192 ); /* TMP0 = limit(TMP0,8192) */
    swk1 = mulshr(AxisRscI->IntAdP.OnDelayComp, swk0, 13 ); /* TMP1(ONDVU) =
(IntAdP.OnDelayComp*TMP0)>>13 */
/*-----*/

    swk0 = mulshr_limitf(AxisRscI->IntAdP.OnDelaySlope, swk2, 8); /* TMP0= limit(TMP0,2^15-1)*/
    swk0 = IxLIMIT( swk0, 8192 ); /* TMP0 = limit(TMP0,8192) */
    swk2 = mulshr(AxisRscI->IntAdP.OnDelayComp, swk0, 13 ); /* TMP1(ONDVU) =
(IntAdP.OnDelayComp*TMP0)>>13 */
/*-----*/

    swk0 = mulshr_limitf(AxisRscI->IntAdP.OnDelaySlope, swk3, 8); /* TMP0= limit(TMP0,2^15-1)*/
    swk0 = IxLIMIT( swk0, 8192 ); /* TMP0 = limit(TMP0,8192) */
    swk3 = mulshr(AxisRscI->IntAdP.OnDelayComp, swk0, 13 ); /* TMP1(ONDVU) =
(IntAdP.OnDelayComp*TMP0)>>13 */
}

/*-----*/
/*****

```



```

/*      Voltage conversion to Carrier count range      */
/*****/
/*      -2000h..2000h  ---> 0h..4000h  ---> 0h..CRFRQ      */
/*****/
AxisRscI->VcmpV.VuOut = IxLIMIT( AxisRscI->VcmpV.VuOut, swk7 ); /* limit +-2000h      */
AxisRscI->VcmpV.VvOut = IxLIMIT( AxisRscI->VcmpV.VvOut, swk7 );
AxisRscI->VcmpV.VwOut = IxLIMIT( AxisRscI->VcmpV.VwOut, swk7 );

/* for debug 此处应该是转换为载波计数值 swk4 / 0x2000 x CrFreqW */
swk4 = swk7 - AxisRscI->VcmpV.VuOut;
swk4 = mulshr(swk4, AxisRscI->IntAdV.CrFreqW, 14 );
swk5 = swk7 - AxisRscI->VcmpV.VvOut;
swk5 = mulshr(swk5, AxisRscI->IntAdV.CrFreqW, 14 );
swk6 = swk7 - AxisRscI->VcmpV.VwOut;
swk6 = mulshr(swk6, AxisRscI->IntAdV.CrFreqW, 14 );

/*-----*/
/*  Deat-time compensation (timer) : if(Vx == 0 || Vx == IntAdV.CrFreqW) No compensation */
/*-----*/
if( ( swk4 != ZEROR ) && (swk4 != AxisRscI->IntAdV.CrFreqW ) )
{
    swk4 = swk4 - swk1;    /* VcmpV.VuOut <-- VcmpV.VuOut+ONDVU      */
    IxLmtzReg16( swk4, swk4, AxisRscI->IntAdV.CrFreqW );    /* VcmpV.VuOut <--
limitz( VcmpV.VuOut , IntAdV.CrFreqW )      */
}
if( ( swk5 != ZEROR ) && (swk5 != AxisRscI->IntAdV.CrFreqW ) )
{
    swk5 = swk5 - swk2;    /* VcmpV.VvOut <-- VcmpV.VvOut+ONDVV      */
    IxLmtzReg16( swk5, swk5, AxisRscI->IntAdV.CrFreqW );    /* VcmpV.VvOut <--
limitz( VcmpV.VvOut , IntAdV.CrFreqW )      */
}
if( ( swk6 != ZEROR ) && (swk6 != AxisRscI->IntAdV.CrFreqW ) )
{
    swk6 = swk6 - swk3;    /* VcmpV.VwOut <-- VcmpV.VwOut+ONDVW      */
    IxLmtzReg16( swk6, swk6, AxisRscI->IntAdV.CrFreqW );    /* VcmpV.VwOut <--
limitz( VcmpV.VwOut , IntAdV.CrFreqW )      */
}

AxisRscI->PwmV.PwmCntT2 = swk6;
AxisRscI->PwmV.PwmCntT1 = swk5;
AxisRscI->PwmV.PwmCntT0 = swk4;

#define ASIC_CLKMHZ      80      /* ASIC Clock [MHz]      */ /* ** @@暂定 */

```

在函数 `static LONG PcalPwmFrequency(LONG PwmFx)`中有

```
case 10667 : Tc = 3750; break; /* PwmFc = 10.666666... [kHz] */ /*<V324>*/
case 8000 : Tc = 5000; break; /* PwmFc = 8.0 [kHz] */ /*
case 5333 : Tc = 7500; break; /* PwmFc = 5.333333... [kHz] */ /*<V324>*/
case 4000 : Tc = 10000; break; /* PwmFc = 4.0 [kHz] */ /*
case 3556 : Tc = 11250; break; /* PwmFc = 3.555555... [kHz] */ /*<V324>*/
```

选择不同的载波频率，相关的计数值如上所示，可推测 PWM 的时钟为 80Mhz，采用上下计数的方式。

则 CrFreqW 的计数值按 10.66667khz 计算为 3750。

通过计算可得 $80/2/3750 = 10.66667\text{khz}$

```
/* オンディレイ補償時間 */
kx = uCfgPrm->odt_c.b.h * (ASIC_CLKMHZ/2) / 10;
rc |= LpxSetPrmToASIC( &(pAsicMicroReg->IntAdP.OnDelayComp), kx, NO_LMT_CHK );
/* オンディレイ補償切替レベル */
kx = PcalOndelayLevel( uCfgPrm->odlv_slp.b.l );
rc |= LpxSetPrmToASIC( &(pAsicMicroReg->IntAdP.OnDelayLvl), kx, NO_LMT_CHK );
/*-----*/
/* オンディレイ補償傾き */
kx = PcalOndelaySlope( uCfgPrm->odlv_slp.b.h );
rc |= LpxSetPrmToASIC( &(pAsicMicroReg->IntAdP.OnDelaySlope), kx, NO_LMT_CHK );

DBYTEX odt_c; /* PnE2D : 下位:オンディレイ時間, 上位:オンディレイ補償定数 */
DBYTEX odlv_slp; /* PnE2E : 下位:オンディレイ補償変更レベル, 上位:補償傾き */
/* PnE2D : 下位:延迟时间, 上位:延时补偿常数 */
/* PnE2E : 下位:延时补偿变更等级, 默认 100%, 上位: 补偿斜率 5 */
以 400w 机型为例, PnE2D = 0x1E1E, PnE2E = 0x0564, 则
补偿计数值 OnDelayComp = 30 * 40/2/10 = 60 ,补偿时间为 3us
/*****
/*
/* オンディレイレベルの計算 接通延迟电平的计算 */
/*
/*
/*****
/* Level * 15000 */
/* kx = ----- → = limit( Level *150 ) :0 ~15000でリミット*/
/* I_max */
/*
/* Level [%] : オンディレイ切り替えレベル */
/* I_max [%] : 最大電流[%] (=最大トルク[%]) */
/* 此函数的功能是将数值限制在0---15000之间, 为标么值 */
/*
/*
/*****
static LONG PcalOndelayLevel( LONG Level )
{
```

```

    LONG    kx;

    kx = MlibLimitul( (Level * 150), 15000, 0 );

    return( kx );
}

```

补偿层级切换 OnDelayLvl = 0x64 * 150 = 15000，即为最大电流对应的标么值

```

/*****
/*
/*      オンディレイ補償台形傾きの計算  接通延迟梯形倾斜的计算
/*
/*
/*
/*****
/*
/*      
$$k_x = \frac{2^8 * 2^{13}}{15000 * \text{Slope} / 1000} = \frac{2^8 * 2^{13} * 1000}{15000 * \text{Slope}} = \frac{139810}{\text{Slope}}$$

/*
/*      Qdslp [0.1%]: 補償が100%となる電流  补偿为100%的电流
/*      此函数中Q8, Q13是为后续移位用，目的在于提高精度
/*
/*
/*****
static    LONG    PcalOndelaySlope( LONG slope )
{
    #if (FLOAT_USE==TRUE)
        float fw;

        if( slope != 0 )
        {
            fw = 139810.0F / (float)slope;
        }
        else
        {
            fw = 139810.0F;
        }

        return( fw );

    #else
        LONG    kx;

        if( slope != 0 )

```

```

{
    kx = 139810 / slope;
}
else
{
    kx = 139810;
}

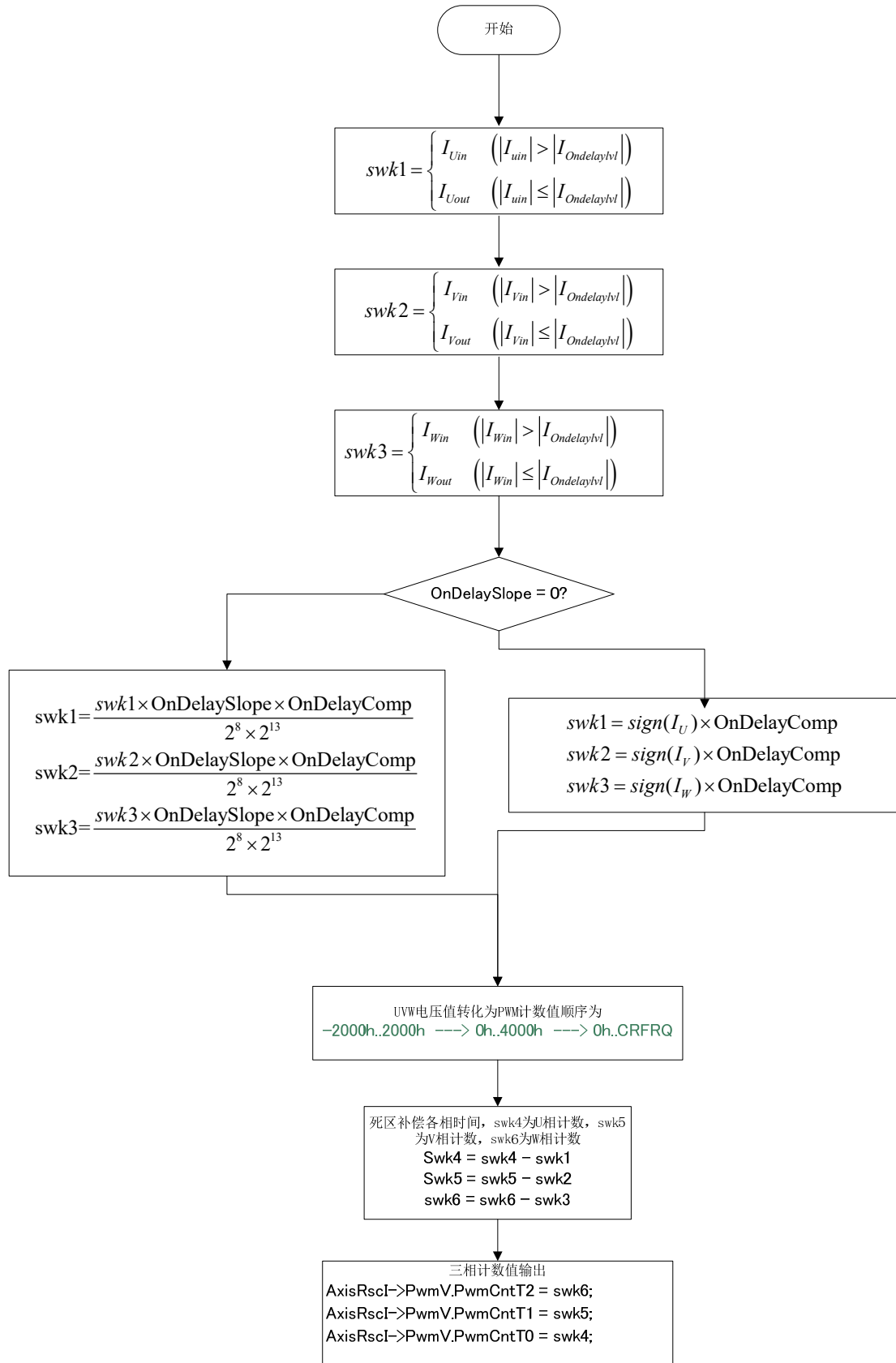
return( kx );

#endif /* FLOAT_USE */
}

```

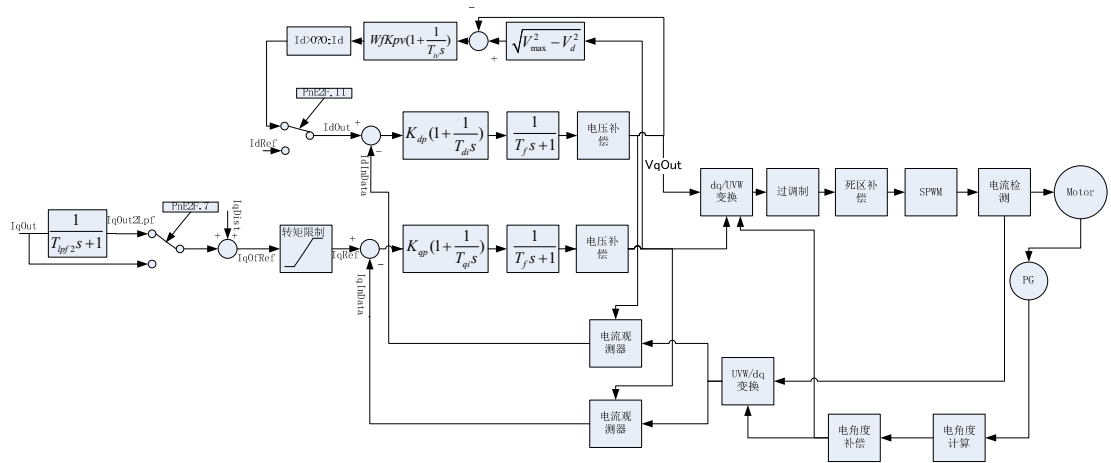
$$OnDelaySlope = \frac{2^8 \times 2^{13}}{15000 \times Slope / 1000} = \frac{139810}{Slope}$$

补偿增益斜坡 $OnDelaySlope = 139810 / 5 = 27962$ ，不同电机补偿斜率不一样



死区补偿的原理是直接补偿三相的时间，根据三相电流的极性进行补偿，在默认为最大电流值内，采用三相电流给定值而不是三相反馈值，采用斜坡补偿的方式，具体要弄清楚什么范

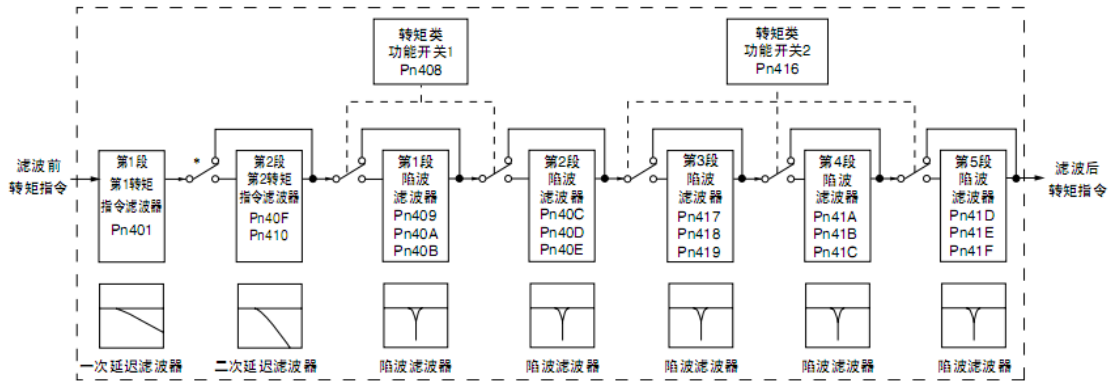
国内采用斜坡补偿，以及不同电机采用不同的斜率进行补偿的方式。任何形式的死区补偿都是一种近似补偿方法。



电流环框图

1.10 转矩给定滤波器

MpIntHost



1.10.1 陷波滤波器

```

/*****
/*
/*      2次ノッチフィルタのパラメータ計算 二阶陷波滤波器的参数计算      */
/*
/*****
/*

```

```

/*          2*(wx^2 - hx^2)          hx^2 + dx*wx*hx/qx + wx^2          */
/*      K1 = -----      K2 = -----      */
/*          hx^2 + wx*hx/qx + wx^2          hx^2 + wx*hx/qx + wx^2          */
/*                                          */
/*                                          */
/*                                          */
/*          hx^2 - wx*hx/qx + wx^2          hx^2 - dx*wx*hx/qx + wx^2          */
/*      K3 = -----      K4 = -----      */
/*          hx^2 + wx*hx/qx + wx^2          hx^2 + wx*hx/qx + wx^2          */
/*                                          */
/*                                          */
/*                                          */
/*****
void KpiPcalMicroNotchFilter2(          /* 2次ノッチフィルタのパラメータ計算          */
    ASICS    *SvAsicRegs,          /* JL-076アクセス用構造体          */
    LONG     Hz,          /* フィルタ周波数  频率      [Hz]          */
    LONG     qx,          /* フィルタQ定数  宽度[0.01]          */
    LONG     dx,          /* フィルタ深さ  深度      [0.001]*          */
    SHORT    *MpReg )/* フィルタパラメータ設定先頭アドレス  濾波器参数设置起始地址*/
{
    LONG     ts;
    LONG     Hx;
    LONG     wx;
    LONG     cx[3];
    LONG     freq;
    USHORT   kf[4];

/*-----*/
/*      パラメータ計算参数计算          */
/*-----*/

    if( qx < 50 )
    { /* Lower Limit for qx */
        qx = 50;
    }
    if( dx > 1000 )
    { /* Upper Limit for dx */
        dx = 1000;
    }

/*-----*/
/*      フィルタ演算周期設定  62500/100=625(0.1us)*          */
/*-----*/
    ts = KPI_MBCYCLEN/100;

/*-----*/
/*      周波数補正      频率更正 原理是什么?          */
/*-----*/

```

```

if ( hz > 2000 )
{
    freq = (LONG)( (SHORT)hz * (SHORT)(3 * hz - 1894) + 11788000) / 10000;
}
else
{
    freq = hz;
}

/*-----*/
hx = 10000000 / ts; /*  $h_x = (2/ts) * (1/2)$  采样频率 hz */
wx = (freq * 12868) >> 12; /*  $w_x = (hz * 2 * \text{PAI}) * (1/2)$   $2\pi * 4096 / 2 = 12868$  */
/*-----*/

while( hx > 23170 )
{ /* Scaling :  $h_x^2 < 2^{29}$  */
    hx = hx >> 1; wx = wx >> 1;
}

while( wx > 23170 )
{ /* Scaling :  $w_x^2 < 2^{29}$  */
    hx = hx >> 1; wx = wx >> 1;
}

/*-----*/
cx[0] = MlibScalKxgain( 100*wx, hx, qx, NULL, -30 ); /*  $c_x[0] = w_x * h_x / q_x$  */
cx[1] = MlibScalKxgain( cx[0], dx, 1000, NULL, -30 ); /*  $c_x[1] = dx * w_x * h_x / q_x$  */
cx[2] = hx*hx + cx[0] + wx*wx; /*  $c_x[2] = h_x * h_x + w_x * h_x / q_x + w_x * w_x$  */
/*-----*/

kf[0] = (USHORT)MlibScalKxgain( 2*(wx*wx - hx*hx), (1<<13), cx[2], NULL, -24 ); /* K1 */
kf[1] = (USHORT)MlibScalKxgain( (hx*hx + cx[1] + wx*wx), (1<<13), cx[2], NULL, -24 ); /* K2 */
kf[2] = (USHORT)MlibScalKxgain( (hx*hx - cx[0] + wx*wx), (1<<13), cx[2], NULL, -24 ); /* K3 */
kf[3] = (USHORT)MlibScalKxgain( (hx*hx - cx[1] + wx*wx), (1<<13), cx[2], NULL, -24 ); /* K4 */
/*-----*/
/*      マイクロIFレジスタへの書き込み      */
/*-----*/

#if 0 /* 2012.09.04 Y.Oka ★  $\mu$  プログラムへのパラメーター括書き込み機能追加必要★ */
    MicroTranslatePrmReq( SvAsicRegs, kf, MpReg, 4 );
#else /* 暫定対応 */
    MpReg[0] = (USHORT)kf[0]; /* Write k[0] to ASIC Micro Register */
    MpReg[1] = (USHORT)kf[1]; /* Write k[1] to ASIC Micro Register */
    MpReg[2] = (USHORT)kf[2]; /* Write k[2] to ASIC Micro Register */
    MpReg[3] = (USHORT)kf[3]; /* Write k[3] to ASIC Micro Register */
#endif

/*-----*/
}

```

传递函数为:

$$G(s) = \frac{s^2 + 2\pi pks + (2\pi f)^2}{s^2 + 2\pi ks + (2\pi f)^2}$$

另一种表达式为

$$G(s) = \frac{s^2 + 2\zeta_2\omega_n s + \omega_n^2}{s^2 + 2\zeta_1\omega_n s + \omega_n^2} = \frac{s^2 + 2\zeta_2 2\pi f s + (2\pi f)^2}{s^2 + 2\zeta_1 2\pi f s + (2\pi f)^2}$$

相关系数可进行换算

$$k = 2\zeta_1 f$$

$$p = \frac{\zeta_2}{\zeta_1}$$

本文采用双线性变换的方法，对模拟陷波滤波器的传递函数做离散化处理，对表达式进行处理。

$$s = \frac{2(z-1)}{T(z+1)}$$

可得：

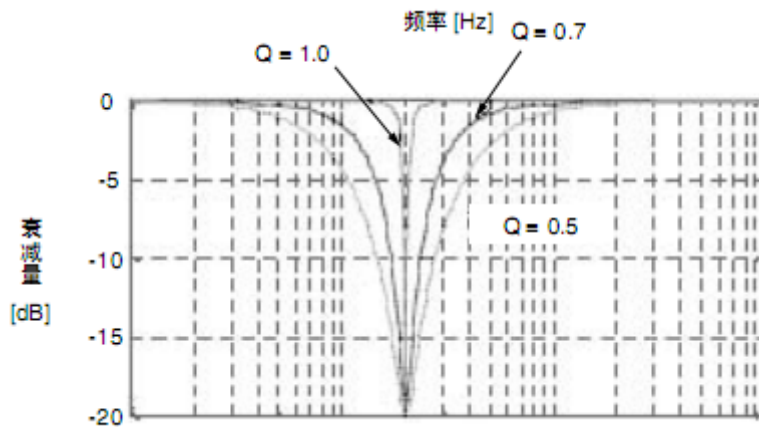
$$\frac{y(n)}{u(n)} = \frac{d_2 + d_1 z^{-1} + d_0 z^{-2}}{1 + c_1 z^{-1} + c_0 z^{-2}}$$

而根据安川的相关系数计算，有

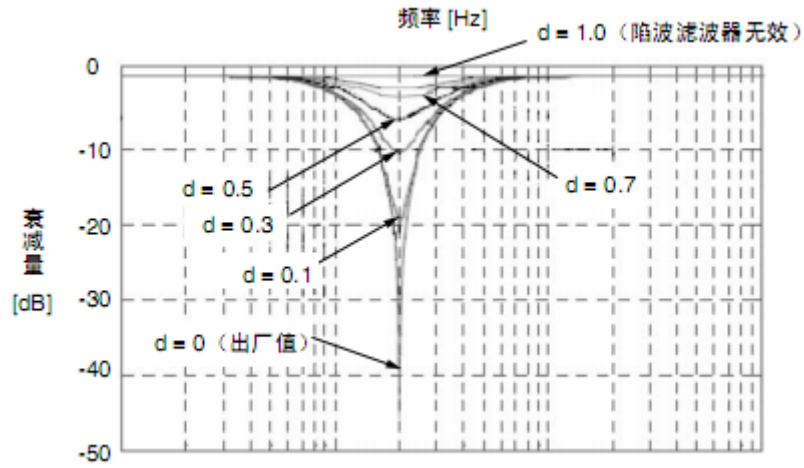
$$k = \frac{100}{qx} f$$

$$p = \frac{dx}{1000}$$

其中 qx 为功能码陷波滤波器 Q 值



其中 dx 为功能码为陷波滤波器深度



共用到了 6 个陷波滤波器，其中 5 个是开放给客户用的，一个是做为自整定的，其功能码为

PnF1C 检测振动频率，PnE3B 为 Q 值，深度默认为 0

中间插了一个一阶低通滤波器，默认时间常数为 0，采样周期 15.625us，也就是无效。另外

一个 Pn401 低通滤波器，一个 Pn40F,Pn410 二阶低通滤波器估计是放在转矩给定后面，未在

此中断中。

此中断程序还包括一些系数的赋值，供后续时序电流环计算使用

二. 速度环

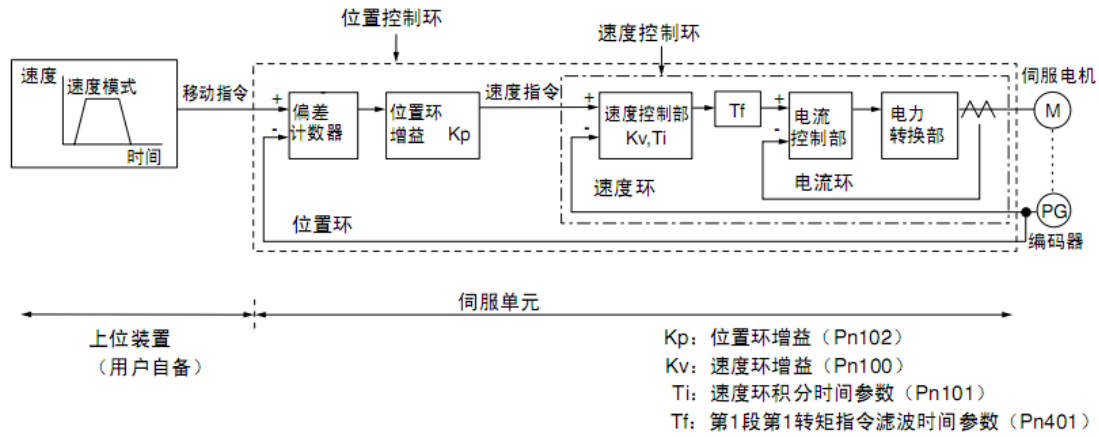


图 8.1 位置控制时的控制框图

2.1 PI 控制

```

/*-----*/
/*速度制御ゲイン中間パラメータの計算                                Rotary      Linear*/
/*-----*/
/*
/*          2*PAI * OvrSpd * Jmot          OvrSpd : [rad/s]          [m/s]*/
/*      Kvx = -----          Jmot   : [kg*m^2]          [kg] *//
/*          MaxTrq                        MaxTrq : [Nm]          [N] *//
/*-----*/

```

```

#if (FLOAT_USE==TRUE)

```

```

    fw = Bprm->OvrSpd * (float)C2PAIE7 / (float)C10POW7;

```

```

    Bprm->Kvx = fw * Bprm->Jmot / Bprm->MaxTrq;

```

```

#else

```

```

    kx = MlibScalKskxkx( Bprm->OvrSpd, C2PAIE7, C10POW7, &sx, 0 );

```

```

    kx = MlibPcalKxksks( kx, Bprm->Jmot, Bprm->MaxTrq, &sx, -1 );

```

```

    Bprm->Kvx = kx;

```

```

#endif /* FLOAT_USE */

```

其中 C2PAIE7 为 62831853, C10POW7 为 10000000, 62831853/10000000=6.2831853=2pi.

```

GselGains->Kv = Bprm->Kvx * (( 100.0f + jrate ) * Loophz ) / 1000.0f;

```

```

GselGains->Kvi = GselGains->Kv * (float)KPI_MBCYCLEUS / ( 10.0f * Pitime );

```

```

GselGains->Kv2 = 1;

```

```

GselGains->Kv2Inv = 1;

```

Loophz 为功能码设置截止频率，单位 0.1hz, jrate 的单位为%，所以要除以 1000；Pitime 为功能码设置的积分时间常数，单位为 0.01ms, 乘以 10 转换为 us 进行换算。

整个速度环增益 Kv 的计算为

$$K_v = \frac{2\pi f \times OverSpd \times J}{T_{\max}}$$

$$K_{vi} = K_v \frac{T}{T_i}$$

f 为截止频率， J 为总的转动惯量， T_{\max} 为最大转矩， T 为刷新周期， T_i 为积分时间。

只不过在实际的计算中，进行了单位的转换。

在 **BaseSpeedControl ()** 中进行

```
if( BaseLoops->BaseCtrls->BBRefClrReq )
{ /* (BaseBlock && OTではない) */
/* 速度指令クリア */
    BaseLoops->SpdRefo = 0;
}
else if( BaseLoops->CtrlMode == CTRLMODE_SPD ) || (BaseLoops->CtrlMode ==
CTRLMODE_JOG )
{
    /* ソフトスタート演算 */ /* ScanBからソフトスタートを移動する必要あり @@CHECK */
    //    BaseLoops->SpdRefo = BaseSoftStartSpdRef( SpdRef, BaseLoops->SpdRefo );
    BaseLoops->SpdRefo = BaseLoops->BaseCtrls->CtrlCmdMngr.SpdRefo;
}
else if( BaseLoops->CtrlMode == CTRLMODE_ZSRCH ) /* <S050> */
{
    BaseLoops->SpdRefo = BaseLoops->BaseCtrls->CtrlCmdMngr.SpdRefo;
}
else
{
    /* 位置制御時はソフトスタートなし */
    BaseLoops->SpdRefo = BaseLoops->SpdRefi;
}
```

```
BaseLoops->SpdRefSum = MlibSatAdd32(BaseLoops->SpdRefo, BaseLoops->SpdFFCx);
```

根据不同的操作模式以及是否有清除请求，速度给定进行不同的赋值，速度给定为最终值。

SpdFFCx 为速度给定补偿值，没具体看是怎么来的。SpdRefo 的单位为超速的 Q24 格式。

后续经过了位相补偿处理，暂且放一边。

```
/*-----*/
/*      モードスイッチ演算 <S0BA>      */
```

```

/*-----*/
ModeSwitch( &BaseLoops->BaseCtrls->ModeSwData,
            BaseLoops->TrqRefo,
            (SpdRefx + SpdFFC),
            PosMngV->var.PosErr );

```

此函数是若选择 I-P 切换模式，选择不同的积分分离限幅值进行积分分离，汇川有类似的功能。

```

SpdErrP = MlibMulgainNolim( SpdRefx, BaseLoops->BaseCtrls->CtrlCmdPrm.PI_rate);
SpdErrP = SpdErrP - BaseLoops->SpdCtrl.V.SpdFbFilo + SpdFFC;
BaseLoops->SpdCtrl.V.PacOut = MlibMulgain( SpdErrP, BaseLoops->GselGains->Kv );

```

此为速度环增益输出，其中有一个比例因子 **PI_rate**，在 **PI** 控制中设为 1，而在 **P-PI** 控制中默认参数计算后得到为 0。若设置为 (0,1)，其实质就是一个 **PDFF** 控制器。

```

/*-----*/
SpdErrI = SpdRefx - BaseLoops->SpdCtrl.V.SpdFbFilo + SpdFFC;
/*-----*/

```

```

/* トルク制限中ではない、または速度偏差とトルク指令の符号が異なる場合に積分する */
if ( (FALSE == BaseLoops->TrqClamp) || (neri_calc_on == TRUE) )
{
    // #if (FLOAT_USE==TRUE)
    //          BaseLoops->SpdCtrl.V.IacOut = FlibIntegral( SpdErrI, BaseLoops->GselGains->Kvi,
    &BaseLoops->SpdCtrl.V.Ivar64 );
    // #else
    BaseLoops->SpdCtrl.V.IacOut = MlibIntegral( SpdErrI, BaseLoops->GselGains->Kvi,
    BaseLoops->SpdCtrl.V.Ivar64 );
    // #endif /* FLOAT_USE */
}

```

此为积分的计算，若转矩限制则抗积分饱和处理。

```

/* 電流制限フラグ：ベースイネーブル && μプログラム電流制限ステータス */
BaseControls->CtrlCmdMgr.TrqClamp = (SeqCtrlOut->BeComplete & CtrlLoopOut->TrqClamp);
BaseLoops->TrqClamp = pBaseCtrl->CtrlCmdMgr.TrqClamp; /* <S04B> */

```

转矩限制的计算方式其跳转有好多层：

```

if ( (SpdRefx - BaseLoops->SpdObsFbki >= 0) && (BaseLoops->SpdCtrl.V.TrqRef < 0) )
    || ( (SpdRefx - BaseLoops->SpdObsFbki < 0) && (BaseLoops->SpdCtrl.V.TrqRef > 0) ) )
{ /* 速度偏差とトルク指令の符号が異なる */
    neri_calc_on = TRUE;
}
else
{ /* 速度偏差とトルク指令の符号が同一(トルク指令 = 0含む) */
    neri_calc_on = FALSE;
}

```

若反馈速度与给定转矩方向相反，则进行积分处理。

```

/*-----*/

```

```

/*      速度制御出力      */
/*-----*/
TrqRef0 = BaseLoops->SpdCtrl.V.PacOut + BaseLoops->SpdCtrl.V.IacOut;
#if (FLOAT_USE==TRUE)
    BaseLoops->SpdCtrl.V.TrqRef = ( TrqRef0 * BaseLoops->GselGains->Kv2 );
#else
    BaseLoops->SpdCtrl.V.TrqRef = MlibMulgain29( TrqRef0, BaseLoops->GselGains->Kv2 );
#endif
    return( BaseLoops->SpdCtrl.V.TrqRef );
}

最终的速度环输出处理，为 PI 控制和 P-PI 控制共有，PI 控制中 Kv2 设置为 1，若为 P-PI 控制，
则是 PI 控制中的 Kv1 和 Kv2 的值进行互换。没弄清楚互换的目的。
其输出值为给定转矩值，为 Q24 格式，因为给定速度和反馈速度均为 Q24 格式，转化成了转矩
的 Q24 格式。
/* 速度制御演算 */
BaseLoops->TrqRefI = BaseSpeedControl( BaseLoops,
                                        0,
                                        BaseLoops->SpdFBCx,
                                        BeComplete );

/* トルクフィルタ */
BaseLoops->TrqRefo = BaseTorqueFilterForPSCtrl( BaseLoops, /* <S0B8> */
                                                BaseLoops->TrqRefI,
                                                BaseLoops->TrqFBCx,
                                                BaseLoops->TrqFFCx,
                                                BeComplete );

```

2.1.1 典型二型系统的设计

转速环我们一般将其校正成典型二型系统，其开环传递函数为

$$G(s)H(s) = \frac{K(\tau s + 1)}{s^2(Ts + 1)}$$

若要保证系统的稳定需要保证 $\tau > T$ 。

其开环对数频率特性如下图所示，其中 h 是斜率为-20dB/dec 的中频段宽度，称作“中频宽”。

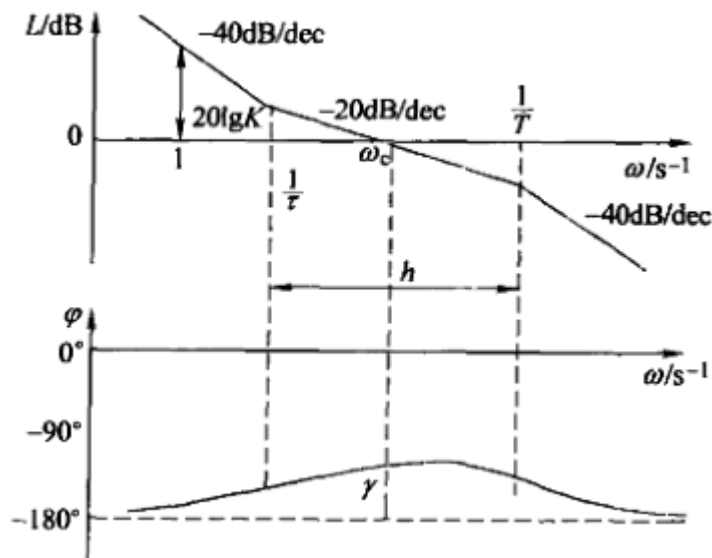


图 1 典型二型系统的对数幅频/相频曲线

由此可见 h 的选取决定了系统的动态性能。

2.1.2 转速调节器

在设计转速调节器时将电流环等效成一个一阶惯性环节，惯性环节的时间常数为 T_Σ 。角速度跟力矩之间的关系

$$\omega = \frac{1}{Js} T$$

其中 s 为拉普帕斯算子。

设计一个 PI 调节器

$$W_{ASR} = \frac{K_P(\tau s + 1)}{\tau s}$$

则系统的开环传递函数为

$$G(s)H(s) = \frac{K_P(\tau s + 1)}{\tau s} * \frac{1}{Js} * \frac{1}{T_\Sigma s + 1}$$

注：式中 s 为拉普拉斯算子，且传递函数未考虑到个环节的转换系数，例如力矩系数等等。

在不考虑采样延时等情况下，系统的模型可以表示为：

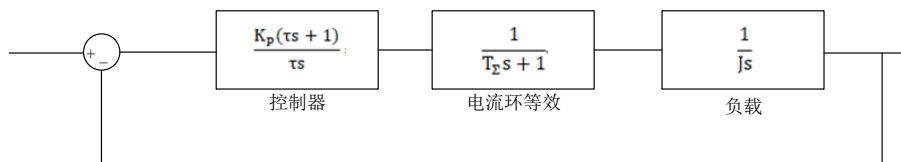


图 2：速度环的等效模型

传递函数简化为

$$G(s)H(s) = \frac{K_p(\tau s + 1)}{\tau J s^2 * (T_\Sigma s + 1)}$$

参数中， τ 和 T_Σ 决定了系统的中频宽 h 。J 为系统惯量，若令 $K_p = 2\pi f$ ，则系统的传递函数又可以简化成

$$G(s)H(s) = \frac{\omega_x(\tau s + 1)}{\tau s^2 * (T_\Sigma s + 1)}$$

又在图 1 中

$$20\lg K = 40(\lg \omega_1 - \lg 1) + 20(\lg \omega_c - \lg \omega_1) = 20\lg \omega_1 \omega_c$$

则

$$K = \omega_1 \omega_c$$

其中 $K = \omega_x / \tau$; $\omega_1 = 1/\tau$;

最终得到

$$\omega_x = \omega_c$$

即 ω_x 能反应系统的开环截止频率。

经过上述分析后，我们可以得到， k_i 决定了系统的中频带宽，决定了系统的动态性能及低频抗扰性能。

K_p 在关联了惯量后，系统被 Scale，此时的 K_p 代表了系统的开环截止频率，在惯量比设置正确的系统在不考虑非线性的情况下，能获得同样的性能。

2.1.3 饱和和作用

如果转速调节器没有饱和限幅的约束，调速系统可以在很大范围内线性工作，则双闭环系统启动时的转速过渡过程会产生较大的超调量，实际上，突加给定电压后，转速调节器很快就进入饱和状态，转速按照线性规律增长。虽然这时的启动过程要比调节器没有限幅时慢得多，但是为了保证启动电流不超过允许值，这是必须的。

如图 2 所示，当转速上升到 O' 后，转速指令减去转速反馈才为负值，此后 ASR 退出饱和，但是由于电机的 I_d 仍是大于 I_{d1} 的，此时电机仍然在加速，直到 $I_d < I_{d1}$ 时，转速才降低下来。在启动的过程中转速必然超调。但是，这已经不是按照线性系统规律的超调，而是经历了饱和和非线性区域之后的超调，可以称作“退饱和超调”。

从理论上分析二型系统的退饱和过程较为复杂，这里给出结论，退饱和超调量的公式为

$$\sigma_n = \frac{\Delta C_{\max}}{C_b} * \frac{\Delta n_b}{n}$$

其超调量是小于在没有饱和超调 $\frac{\Delta C_{\max}}{C_b}$ ，但是其超调量跟转速有关，转速越大退饱和超调量越小。

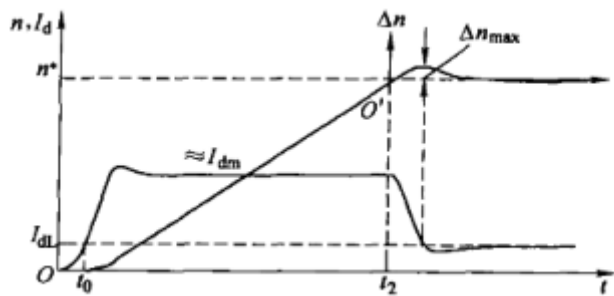


图 2

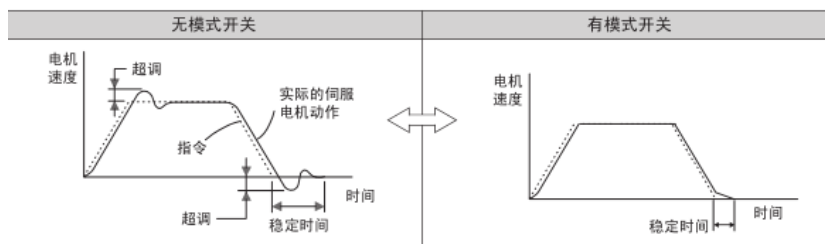
2.1.4 P-PI 切换

P-PI 切换同样是一种非线性的控制，这里没有找到相关的理论依据，但是经过测试改功能很有效果，安川默认开启 P-PI 切换，切换的条件是转矩指令的 200%，如下图所示，Pn108 为 P/PI 切换的选择开关，出厂默认值为以内部转矩指令为条件，且切换的转矩指令为额定转矩的 200%。

设定模式开关（P 控制 /PI 控制切换）

模式开关是自动进行 P 控制、PI 控制切换的功能。
通过 Pn10B.0 设定切换条件，通过 Pn10C、Pn10D、Pn10E、Pn10F 设定切换条件值。

如果设定了切换条件和条件值，则可抑制加减速时的超调并缩短稳定时间。



相关参数

通过 Pn10B.0 选择模式开关的切换条件。

参数	选择模式开关	设定条件值的参数	生效时刻	类别
Pn108	n. 0000 [出厂设定]	以内部转矩指令为条件。	即时生效	基本设定
	n. 0001	以速度指令为条件。		
	n. 0002	以加速度为条件。		
	n. 0003	以位置偏差为条件。		
	n. 0004	不选择模式开关。		

图 3

该功能能有效的解决控制中的快速性与超调之间的矛盾关系，具体的测试波形如下图所示，两张图的参数为安川速度环 $K_p=240$, $T_i=3.3ms$ 时的空载情况下的测试波形，速度指令为 0~3000 的一个阶跃，左图没有开启 P/PI 切换，右图开启了 P/PI 切换，从两张图的对比效果来看，在未牺牲快速性的前提下，超调得到了有效的解决。

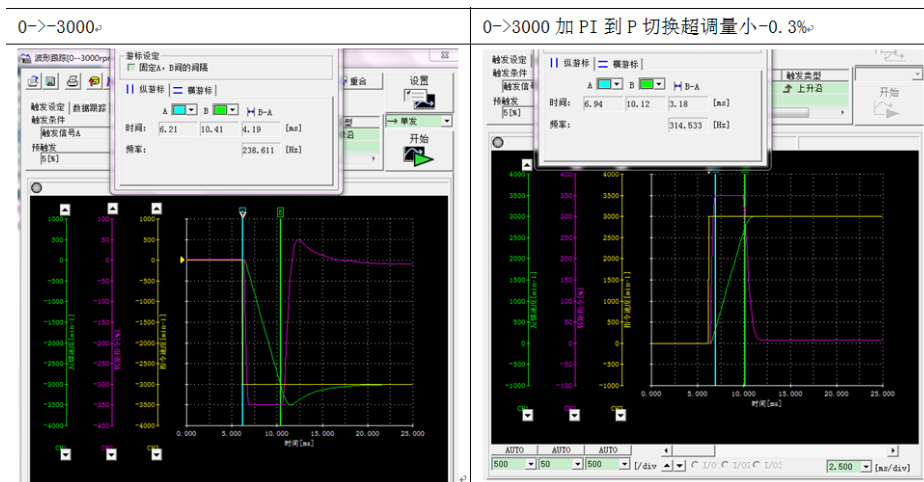


图 4

左图是典型的退饱和超调，加速阶段，电机很快达到了其最大力矩，积分饱和，当实际转速大于给定转速时，积分开始退饱和，由于是空载情况， Idl 较小，此时电机继续加速，直到 $Idm < Idl$ ，转速开始回落。虽然说积分饱和的过程解决了很大一部分的系统超调和振荡，但是仍然会存在一定程度的超调。

右图是加入了 P/PI 切换的过程，当转矩指令达到 200% 时控制模式切换到了 P 控制，但是此时电机仍按照最大力矩出力，速度的加速曲线的斜率跟 PI 控制一样，当转速快要达到给定速度时，此时的 $TrqRef = Kp(V^* - Vx)$ ，当误差较小时方程进入线性区，力矩开始减小，当力矩降低到 200% 以下时，开始进行 PI 控制，开始进行 PI 控制时的积分值为 0（非绝对，下面会在代码中分析），这种情况下可以将超调量降低到很低。PI 控制能消除稳态误差，提供稳态和低频刚度，也即图 1 中的系统频率小于 $1/\tau$ 的部分，提高系统的“低频带宽”。

2.1.5 PI 调节物理含义

由此可见，安川中的速度环调节器的传递函数为：

$$W_{ASR} = \frac{K_p(\tau s + 1)}{\tau s}$$

代码是这么实现的，那么把代码中的实现方式代入到理论公式中去是什么样子？

我们在 1.2 节推导出了速度环的传递函数为

$$G(s) = \frac{K_p(\tau s + 1)}{\tau s^2 * (T_\Sigma s + 1)}$$

将 $K_p = 2\pi * J * \text{LoopHz}$ 代入，则系统的传递函数为

$$G(s) = \frac{\omega_x(\tau s + 1)}{\tau s^2 * (T_\Sigma s + 1)}$$

其中 LoopHz 的单位为 Hz； $\omega_x = 2\pi * \text{LoopHz}$ 。

又在图 1 中

$$20\lg K = 40(\lg w_1 - \lg 1) + 20(\lg w_c - \lg w_1) = 20\lg w_1 w_c$$

则

$$K = \omega_1 \omega_c$$

其中 $K = \omega_x / \tau$ ； $\omega_1 = 1/\tau$ ；

即通过以上方式对 PI 参数进行整定，最终 K_p 代表系统的开环截止频率，又根据开环截止频率和闭环截止频率的同向性，因此这种方法整定后的 K_p 能反映出速度环的带宽。

2.2 速度环输出低通滤波器

```

/*****
/*
/*      トルクフィルタ:ローパスフィルタパラメータ計算      */
/*
/*
/*****
/*
/*      補足:トルク制御モード時のトルクフィルタは、第1トルクフィルタのみ対応      */
/*
/*
/*****
void PcalBaseTrqLpassFilter( TRQFIL *TrqFilData, BASE_CTRL_HNDL *BaseCtrlData,
                           LONG trqfil11, LONG trqfil12,
                           LONG GselNo )
{
    LONG          cyc_time;
    GSELGAINS      *GselGains;
//  CTRL_CMD_PRM  *CtrlCmdPrm;//<S0C7>

    GselGains= &(BaseCtrlData->GainChange.GselGains[GselNo]);
//  CtrlCmdPrm  = &(BaseCtrlData->CtrlCmdPrm);//<S0C7>
    cyc_time  = KPI_MBCYCLEUS;
    #if (FLOAT_USE==TRUE)
        TrqFilData->A.Klpf = FlibPcalKf1gain( trqfil11 * 10, cyc_time, 0 );
        GselGains->Klpf = FlibPcalKf1gain( trqfil12 * 10, cyc_time, 0 );
    #else
        TrqFilData->A.Klpf = MlibPcalKf1gain( 10 * trqfil11, cyc_time, 0 );
        GselGains->Klpf = MlibPcalKf1gain( 10 * trqfil12, cyc_time, 0 );
    #endif /* FLOAT_USE */
}

引用部分Pn401 转矩指令低通滤波时间常数
/* トルクフィルタ */
PcalBaseTrqLpassFilter( &(BaseLoops->TrqFil), BaseCtrlData, Prm->trqfil11, Prm->trqfil11, 5 );

```

```

/*****
/*
/*      トルクフィルタ for 位置ループ & 速度ループ演算      */
/*
/*****

static KSGAIN BaseTorqueFilterForPSCtrl( BASE_LOOPCTRLS *BaseLoops, KSGAIN TrqRefi,
                                         KSGAIN TrqFBC, KSGAIN TrqFFC, KSGAIN
BaseEnable )/*<S00A>*//* <S0B8> */
{
    TRQFIL *TrqFil;
    #if (FLOAT_USE==TRUE)
        float TrqRefx;
        float DisTrqRef1 = 0;          /* トルク外乱指令入力1 */
        float DisTrqRef2 = 0;          /* トルク外乱指令入力2 */
    #else
        LONG   TrqRefx;
        LONG   DisTrqRef1 = 0;          /* トルク外乱指令入力1 扰动转矩指令输入1 */
        LONG   DisTrqRef2 = 0;          /* トルク外乱指令入力2 */
    #endif /* FLOAT_USE */

    TrqFil = &BaseLoops->TrqFil;

    /*-----*/
    /*      テーブル運転時のトルク外乱指令入力処理      */
    /*-----*/

    BaseLoops->DisTrqRef3 = 0;          /* トルク外乱指令入力*/
    #if (FLOAT_USE==TRUE)
        //  switch( TrqFil->V.TrqInputPos ) /* <S0B7> */
        switch( BaseLoops->BaseCtrls->TblDrive.var.DisTrqInTiming )
            /* <S0B7> */
        {
            case 0x01:                      /* トルク指令フィルタ前      */
                DisTrqRef1 = (BaseLoops->BaseCtrls->TblDrive.var.TrqRef);
                break;
            case 0x02:                      /* ローパスフィルタ後、ノッチフィルタ前      */
                DisTrqRef2 = (BaseLoops->BaseCtrls->TblDrive.var.TrqRef);
                break;
            case 0x03:                      /* トルク指令フィルタ後      */
                BaseLoops->DisTrqRef3 = (BaseLoops->BaseCtrls->TblDrive.var.TrqRef);
                break;
            case 0x00:                      /* なし      */
            default :
                break;
        }
    }
}

```

```

#else
    switch( BaseLoops->BaseCtrls->TblDrive.var.DisTrqInTiming )
        /*扰动转矩指令位置 <S0B7> */
    {
        case 0x01:          /* トルク指令フィルタ前          转矩指令滤波前          */
            DisTrqRef1 = BaseLoops->BaseCtrls->TblDrive.var.TrqRef;
            break;
        case 0x02:/* ローパスフィルタ後、ノッチフィルタ前低通滤波器之后，陷波滤波器之前*/
            DisTrqRef2 = BaseLoops->BaseCtrls->TblDrive.var.TrqRef;
            break;
        case 0x03:          /* トルク指令フィルタ後 转矩指令滤波器后 */
            BaseLoops->DisTrqRef3 = BaseLoops->BaseCtrls->TblDrive.var.TrqRef;
            break;
        case 0x00:          /* なし          */
        default :
            break;
    }
#endif /* FLOAT_USE */

/*-----*/
/*      ベースブロック時の処理          基本模块处理          */
/*-----*/

if( BaseEnable == FALSE )
{
    TrqFil->V.TrqFilClrCmd = FALSE; /* 積分初期化コマンドクリア 清除积分初始化命令*/
    TrqFil->V.FilOut = 0;
    TrqFil->V.FilOut3 = 0;
    TrqFil->V.LpFil2[0] = 0;
    TrqFil->V.LpFil2[1] = 0;
    TrqFil->V.LpFil2[2] = 0;
    TrqFil->V.LpFil2[3] = 0;
    return( 0 );
}

#if (FLOAT_USE==TRUE)
    TrqRefx = FlibLimitul( (TrqRefi + DisTrqRef1), (float)0x01000000, (float)-0x01000000 );
#else
    TrqRefx = MlibLimitul( (TrqRefi + DisTrqRef1), 0x01000000, -0x01000000 );
#endif /* FLOAT_USE */

/*-----*/
/*      1次ローパスフィルタ積分初期化处理(拡張制御ON/OFF時に使用する)      一阶低通滤波器积分初始化处理          */
/*-----*/

if(TrqFil->V.TrqFilClrCmd == TRUE)          /* 積分初期化コマンドあり */
{

```

```

        TrqFil->V.FilOut = TrqRefx; /* トルクフィルタ積分値を現入力値で初期化 */
        TrqFil->V.TrqFilClrCmd = FALSE; /* 積分初期化コマンドクリア*/
    }

/*-----*/
/*      1次ローパスフィルタ処理      Pn401      */
/*-----*/
#if (FLOAT_USE==TRUE)
    TrqFil->V.FilOut = FlibLpfilter1( TrqRefx, BaseLoops->GselGains->Klpf, TrqFil->V.FilOut );
#else
    TrqFil->V.FilOut = MlibLpfilter1( TrqRefx, BaseLoops->GselGains->Klpf, TrqFil->V.FilOut );
#endif /* FLOAT_USE */

/*-----*/
/*      トルクFB補償処理      转矩前馈 摩擦补偿 扰动补偿
                                   */
/*-----*/
#if (FLOAT_USE==TRUE)
    TrqRefx = FlibLimitul( (TrqFil->V.FilOut + TrqFFC - TrqFBC + DisTrqRef2), 0x01000000,
    -0x01000000 );
#else
    TrqRefx = MlibLimitul( (TrqFil->V.FilOut + TrqFFC - TrqFBC + DisTrqRef2), 0x01000000,
    -0x01000000 );
#endif /* FLOAT_USE */

/*-----*/
/*      1次ローパスフィルタ処理(トルク補償後) 一階低通滤波処理(转矩补偿后)      */
/*      注意:トルク補償後のフィルタは調整レスのロバスト性を悪化させるので、フィルタは
      由于转矩补偿后的滤波器未经调整会降低鲁棒性, 因此应将滤波器控制在最小限度*/
/*      最小限にすること。      */
/*-----*/
    if(BaseLoops->BaseCtrls->CtrlCmdPrm.LpassFil3 == TRUE)
    { /* トルク補償後トルクフィルタ有効 扭矩补偿后启用扭矩滤波器 Pn170低四位判断
      默认为1401H 为有效 */
        if(BaseLoops->BaseCtrls->TuneLessCtrl.conf.TuningLessUse )
        { /* 調整レス有効 无调整有效 */
            #if (FLOAT_USE==TRUE)
                TrqFil->V.FilOut3 = FlibLpfilter1( TrqRefx,

(BaseLoops->BaseCtrls)->CtrlCmdPrm.TLPm.Klpf3,

                TrqFil->V.FilOut3 );
            #else
                TrqFil->V.FilOut3 = MlibLpfilter1( TrqRefx,

(BaseLoops->BaseCtrls)->CtrlCmdPrm.TLPm.Klpf3,

```

```

TrqFil->V.FilOut3 );

#endif /* FLOAT_USE */

    TrqRefx = TrqFil->V.FilOut3;
}
else
{
    TrqFil->V.FilOut3 = 0;
}
}
else
{
    TrqFil->V.FilOut3 = 0;
}

/*-----*/
/*      2次ローパスフィルタ処理      */
/*-----*/

if(FALSE == BaseLoops->BaseCtrls->CtrlCmdPrm.LpassFil2)
{
    TrqFil->V.LpFil2[0] = 0;
    TrqFil->V.LpFil2[1] = 0;
    TrqFil->V.LpFil2[2] = 0;
    TrqFil->V.LpFil2[3] = 0;
}
else
{
    #if (FLOAT_USE==TRUE)
        TrqRefx = FlibLowPassfilter2( TrqRefx,
                                        (BaseLoops->BaseCtrls)->CtrlCmdPrm.Klpf2,
                                        (BaseLoops->BaseCtrls)->CtrlCmdPrm.Klpf2,
                                        /* S036 */
                                        TrqFil->P.Klpf2,
                                        /*
S036 */ /* TODO:計算結果未確認 */
                                        TrqFil->V.LpFil2 );
    #else
        TrqRefx = MlibLowPassfilter2( TrqRefx,
                                        (BaseLoops->BaseCtrls)->CtrlCmdPrm.Klpf2,
                                        /* S036 */
                                        TrqFil->P.Klpf2,
                                        /* S036 */
                                        TrqFil->V.LpFil2 );
    #endif /* FLOAT_USE */
}

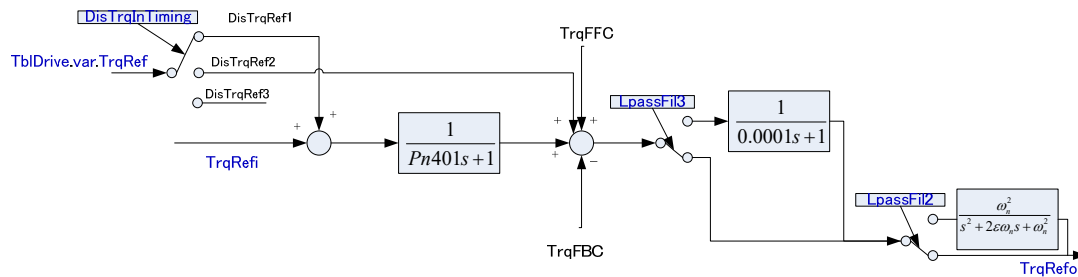
/*-----*/
/*      BoutV.TrcCompTrqRef = TrqRefx;      /* 外乱入力前トルク指 */

```

```

/*-----*/
#if (FLOAT_USE==TRUE)
    return( FlibLimitul( TrqRefx, (float)0x01000000, (float)-0x01000000 ) );
#else
    return( MlibLimitul( TrqRefx, 0x01000000, -0x01000000 ) );
#endif /* FLOAT_USE */
}

```



转矩指令滤波器（CPU）

Pn401	第 1 段第 1 转矩指令滤波时间参数				速度	位置	转矩
	设定范围	设定单位	出厂设定	生效时间	类别		
	0 ~ 65535	0.01ms	100	即时生效	调整		
Pn40F	第 2 段 2 次转矩指令滤波频率				速度	位置	转矩
	设定范围	设定单位	出厂设定	生效时间	类别		
	100 ~ 5000	1Hz	5000*	即时生效	调整		
Pn410	第 2 段 2 次转矩指令滤波器 Q 值				速度	位置	转矩
	设定范围	设定单位	出厂设定	生效时间	类别		
	50 ~ 100	0.01	50	即时生效	调整		

* 设定为 5000 时，滤波器变为无效。

转矩指令滤波器 CPU 一端位置环，速度环有经过一个一阶低通滤波器和一个二阶低通滤波器，二阶低通滤波默认未打开，根据程序其真实的滤波频率为（100,3000），做了处理。其中还包括一个免调整一阶滤波器选择

/* 調整レス用トルク補償後トルクフィルタ選択 转矩补偿后的免调整转矩滤波器选择*/

```

Axis->BaseControls->CtrlCmdPrm.LpassFil3 = TuneLessSetTrqFil3(
    Axis->BaseControls->TuneLessCtrl.conf.TuningLessUse,
    Axis->BaseControls->TuneLessCtrl.conf.TuningLessEx,
    (Axis->UniPrms->Prm)->MencP.flg_wf,
    Axis->MencV->TuningLessLowGainMotor );

```

2.3 速度脉动补偿

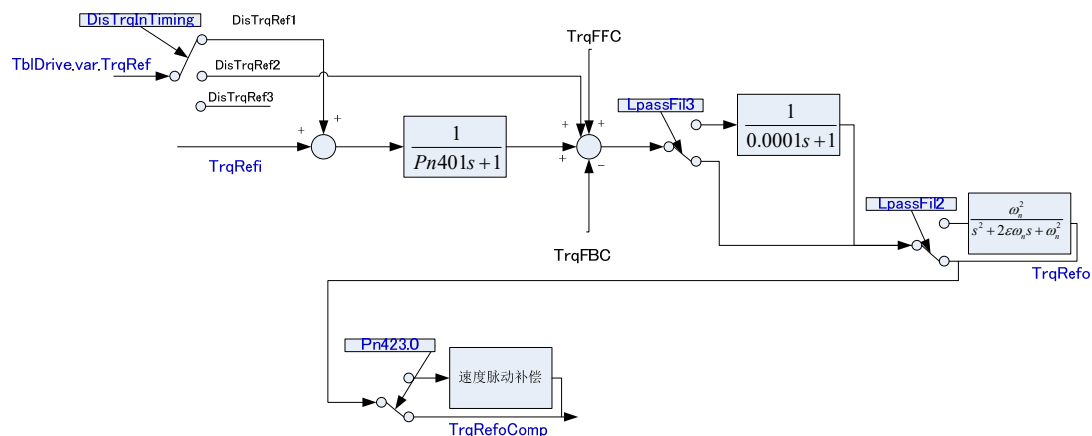
相关功能码

Pn423 速度脉动补偿开关

Pn427 速度脉动补偿有效速度

根据正弦波叠加而成补偿速度脉动，其原理不是很清楚，速度脉动补偿正弦曲线应该是事先设置好的，六个点，其具体值是多少不清楚

$$\text{TrqRefoComp} = \sum A[i] \sin(\omega_e T[i] + \delta_0[i])$$



2.4 摩擦补偿

此处的摩擦补偿为另一种形式的摩擦补偿，相关功能码为

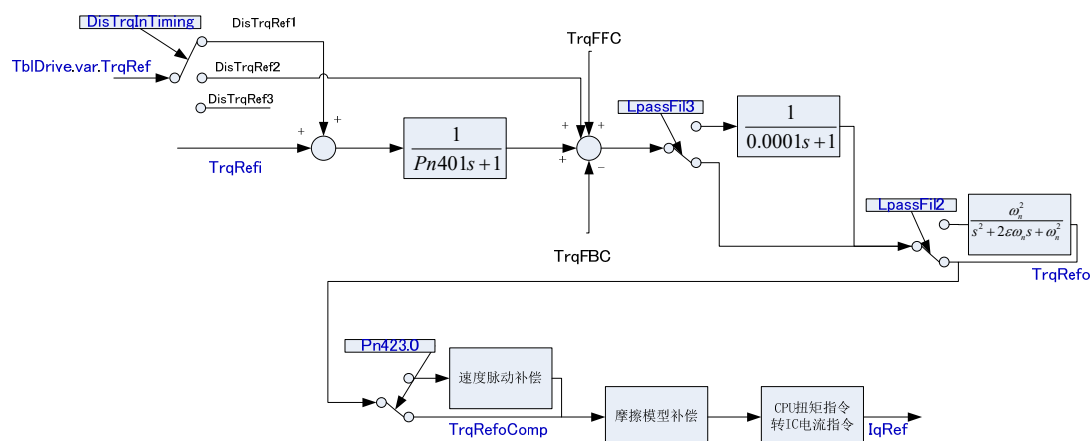
Pn470 重力力矩

Pn471 正向库伦摩擦力矩

Pn472 反向库伦摩擦力矩

Pn473 粘性摩擦力矩

根据此三种摩擦力矩进行补偿。



```
/* TrqRefo --> AinK.IqRef */
```

```
KaiOutputTrqRef( BaseLoops, BaseLoops->TrqRefoComp, 0, BaseLoops->DisTrqRef3 );
```

此函数用来将给定扭矩标么值转化为给定电流标么值，经过摩擦补偿力矩。

```
/******
```

```
/*
```

```
/*      トルク指令出力 : Torque Reference --> ScanA AinK.IqRef/I_dref      */
```

```

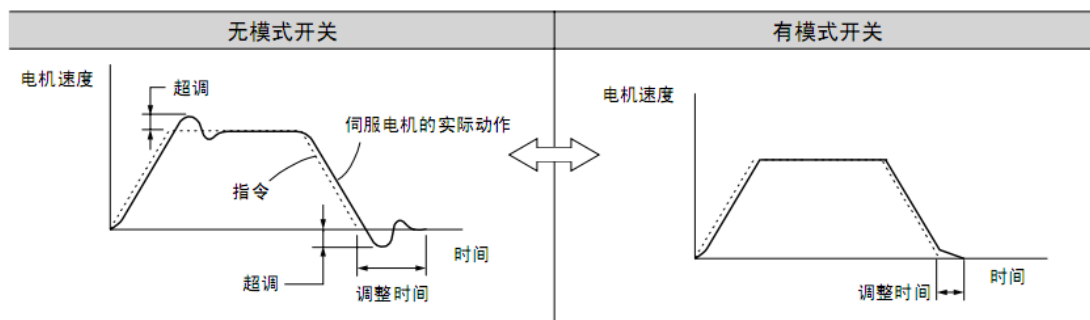
/*                                                                    */
/*****                                                                    */
/*                                                                    */
/*  位相補償方式A：従来方式                                                                    */
/*                                                                    */
/*      IdRef = 0;                                                                    */
/*      15000      TrqRef * 15000 * 2^8                                                                    */
/*      IqRef = TrqRef * ----- = -----                                                                    */
/*      2^24      2^32                                                                    */
/*                                                                    */
/*****                                                                    */
/*                                                                    */
/*  位相補償方式B：IdRef/IqRef方式                                                                    */
/*                                                                    */
/*      15000      MlibFASTSINS16( Pcmps + 0 )      TrqRef * 1875 * SinX  */
/*      IdRef = TrqRef * ----- * ----- = ----- */
/*      2^24      16384      2^32 * 2^3      */
/*                                                                    */
/*      15000      MlibFASTSINS16( Pcmps + 256 )      TrqRef * 1875 * CosX  */
/*      IqRef = TrqRef * ----- * ----- = ----- */
/*      2^24      16384      2^32 * 2^3      */
/*                                                                    */
/*****                                                                    */
static void KaiOutputTrqRef(BASE_LOOPCTRLS *BaseLoops, KSGAIN TrqRef, KSGAIN IdRef,
KSGAIN DisTrq )/*<S00A>*/

```

从此计算公式来看，分为两种，一种是最大转矩控制，一种是弱磁控制，dq 轴电流进行分配。在转矩转化过程中直接转化为了电流，转矩系数都免了，可见标么化的好处。只要明确最大转矩对应 Q24，而最大电流对应 15000，中间的转矩系数都免了，此为真正的标么化，汇川的在速度环增益计算时，就乘以了转矩系数进行转化。

模式开关是自动进行 P 控制、PI 控制切换的功能。

利用参数设定切换条件和切换条件的等级后，可抑制加减速时的超调，缩短整定时间。



通过 Pn10B = n.□□□X 选择模式开关的切换条件。

参数		模式开关的选择	设定等级的参数		有效时间	类别
			旋转型	直线		
Pn10B	n.□□□0 [出厂设定]	以内部转矩指令为条件。	Pn10C		即时生效	设定
	n.□□□1	以速度指令为条件。	Pn10D	Pn181		
	n.□□□2	以加速度为条件。	Pn10E	Pn182		
	n.□□□3	以位置偏差为条件。	Pn10F			
	n.□□□4	不选择模式开关。	-			

补充说明

关于速度环控制方法的选择（PI 控制 /I-P 控制）

一般地，在高速定位以及高速、高精度加工应用中，I-P 控制更为有效。如果位置环增益比 PI 控制时还低，则可缩短定位时间以及降低圆弧半径的缩小。但是，要通过模式开关等充分利用与 P 控制之间的切换以达到上述目标时，一般使用 PI 控制。

■ 将模式开关的切换条件作为转矩指令时 [出厂设定]

转矩指令超出模式开关（转矩指令）（Pn10C）中设定的转矩时，速度环将切换为 P 控制。
出厂时转矩指令值被设定为 200%。

```

if( (BaseLoops->CmdCtrlBit & ENBPCTRL_BIT)
    || ((BaseLoops->BaseCtrls->ModeSwData.var.ModeSWFlag
        && (!BaseLoops->BaseCtrls->CtrlCmdPrm.IPSpdControl))) )
{ /* 比例制御時 */
//      BaseLoops->uCycInputs.MREG_BITDAT |= BITDAT_PCONACTFLG;
      BaseLoops->PConActFlg = TRUE; /* <S064> */
#if (FLOAT_USE==TRUE)
      BaseLoops->SpdCtrl.V.SpdFbFilo = FlibLpfilter1( BaseLoops->SpdObsFbki + SpdFBC,

      BaseLoops->BaseCtrls->CtrlCmdPrm.KVfbFil,

      BaseLoops->SpdCtrl.V.SpdFbFilo );
      SpdErrP = SpdRefx - BaseLoops->SpdCtrl.V.SpdFbFilo + SpdFFC;
      BaseLoops->SpdCtrl.V.PacOut = SpdErrP * BaseLoops->GselGains->Kv ;
#else
      BaseLoops->SpdCtrl.V.SpdFbFilo = MlibLpfilter1( BaseLoops->SpdObsFbki + SpdFBC,

      BaseLoops->BaseCtrls->CtrlCmdPrm.KVfbFil,

      BaseLoops->SpdCtrl.V.SpdFbFilo );
      SpdErrP = SpdRefx - BaseLoops->SpdCtrl.V.SpdFbFilo + SpdFFC;
      BaseLoops->SpdCtrl.V.PacOut = MlibMulgain( SpdErrP, BaseLoops->GselGains->Kv );
#endif /* FLOAT_USE */
      /*-----*/
if( !BaseLoops->BaseCtrls->CtrlCmdPrm.SpdIctrlKeep )
{ /* 速度制御積分保持ではない場合 */
      /* 時定数を持って積分クリア */
#if (FLOAT_USE==TRUE)
      BaseLoops->SpdCtrl.V.IacOut = FlibLpfilter1( 0.0f,

```

```

BaseLoops->SpdCtrl.P.KVintegFil,

BaseLoops->SpdCtrl.V.IacOut );
BaseLoops->SpdCtrl.V.Ivar64[1] = BaseLoops->SpdCtrl.V.IacOut *2.0f;
#else
BaseLoops->SpdCtrl.V.IacOut = MlibLpfilter1( 0,

BaseLoops->SpdCtrl.P.KVintegFil,

BaseLoops->SpdCtrl.V.IacOut );
BaseLoops->SpdCtrl.V.Ivar64[1] = BaseLoops->SpdCtrl.V.IacOut << 1;
#endif /* FLOAT_USE */
BaseLoops->SpdCtrl.V.Ivar64[0] = 0;
}
}

```

此处要弄清楚 P 控制的条件以及积分分离的阈值点。

```

/*-----*/
/*      PnE0B : 速度ループ制御選択設定                      */
/*      BitC : 速度ループ制御選択設定                        */
if( Prm->syssw1 & 0x1000 )
{
    Axis->BaseControls->CtrlCmdPrm.SpdCtrlKeep = TRUE;
}
else
{
    Axis->BaseControls->CtrlCmdPrm.SpdCtrlKeep = FALSE;
}

```

SpdCtrlKeep 的值取决于功能码 PnE0B 的 bit12 位的值。

```
BaseLoops->CmdCtrlBit = CmdCtrl->CmdCtrlBit; /* <S050> */
```

来自于此赋值，为控制命令，bit2 是否置位进行判断，估计是上位机发出的指令，具体不详。

第二个判断选择模式开关是否满足条件，以及功能码 Pn10B.1 设置成了 P-PI 模式

```

/*-----*/
/*      Pn10B.1 : 速度ループ制御選択設定                      */
/*-----*/
switch(( Prm->gnmode >> 4 ) & 0x0F )
{
    case 0x00:
        Axis->BaseControls->CtrlCmdPrm.IPSpdControl = FALSE;
        break;
    case 0x01:
        Axis->BaseControls->CtrlCmdPrm.IPSpdControl = TRUE;
        break;
    default :

```

```

        ALMSetPrmError( Axis->AlmManager, pndef_gnmode.RegNumber );
        break;
    }

```

说明书上Pn10B.1设置为0,为PI模式,则IPSpdControl=0; 设置为1为P-PI切换模式,IPSpdControl=1; 这个条件((BaseLoops->BaseCtrls)->ModeSwData.var.ModeSWFlag

&& (!BaseLoops->BaseCtrls->CtrlCmdPrm.IPSpdControl))岂不是不满足。难道说明数搞反了。

以默认的转矩限制为例

```

/*      Pn10C : モードスイッチ(トルク指令) [%]                                     */
ModeSwData->conf.MSWTrqLevel
        = BprmTorqueLevelCal( Axis->UniPrms->Bprm, ( 100 * Prm->mdswlv ), 0 ); /*
[%] --> [2^24/MaxTrq] */

```

当给定转矩大于设置的转矩时, 默认为 200%, 则只进行 P 控制, 积分不累加;

2.5 速度反馈计算

```

LpxPosSpdFeedbackCal( BaseLoops ); /* モータ位置&速度FB演算                                     */
是否使能位相补偿速度观测器, 根据Pn10B.3=5以及未开启免调整功能时有效
/*-----*/
/*      Pn10B.3 : 位相補償速度オブザーバ設 只数码管位的第三位                                     */
/*-----*/
PrmSetting = EhVobsCalculatePrmSW( &(Axis->BaseControls->EhVobsCtrl.conf.EhVobsUse),
Axis->BaseControls->TuneLessCtrl.conf.TuningLessUse, (Prm->gnmode >> 12) );

```

2.5.1 相位补偿速度观测器

速度观测器的计算, 有两个地方可进行速度观测器的计算

```

static void tuneLessEhSpeedObserver( TUNELESS_EHVOBS *EhVobs, LONG MotSpd, LONG
TrqRef )
BaseLoops->SpdObsFbki = EhSpeedObserver( EhVobsCtrl,
BaseLoops->SpdFbki,
BaseLoops->TrqRef );

```

输入为速度反馈以及速度环输出的转矩, 未经滤波处理的。

Pn127 速度观测器增益 40hz

Pn128 速度观测器位置补偿增益 150%

Pn13B 速度观测器低通滤波器时间常数 6.01ms

Pn401 扭矩低通滤波器时间常数 1.00ms

相关系数变量

```

/*-----*/
/*      位相補償速度オブザーバ(VOBS)変数定*/

```

```

/*-----*/
typedef struct EHVOBS_PRM { /* 位相補償速度オブザーバ用パラメータ定義 */
#ifdef (FLOAT_USE==TRUE)
    float      Kstf1;          /* 位相補償VOBSゲイン1(Pn127) */
    float      Kstf2;          /* 位相補償VOBSゲイン2(Pn127) */
    float      Kstf3;          /* 位相補償VOBSゲイン3(Pn127) */
    float      Kj;             /* 位相補償VOBS位相補償ゲイン(Pn128) */
    float      Lpf;            /* 位相補償VOBSローパスフィルタゲイン(Pn13B+Pn401) */
    float      Tf;             /* 位相補償VOBSローパスフィルタ時定数 */
    float      KTrqReflpf;      /* VOBSトルク指令ローパスフィルタゲイン */
#else
    LONG       Kstf1;          /* 位相補償VOBSゲイン1(Pn127) */
    LONG       Kstf2;          /* 位相補償VOBSゲイン2(Pn127) */
    LONG       Kstf3;          /* 位相補償VOBSゲイン3(Pn127) */
    LONG       Kj;             /* 位相補償VOBS位相補償ゲイン(Pn128) */
    LONG       Lpf;            /* 位相補償VOBSローパスフィルタゲイン(Pn13B+Pn401) */
    LONG       Tf;             /* 位相補償VOBSローパスフィルタ時定数 */
    LONG       KTrqReflpf;      /* VOBSトルク指令ローパスフィルタゲイン */
#endif
} EHVOBS_PRM;

```

速度観測器の系数計算有三个地方

```

void PcalEhSpeedObserver( EHVOBS *EhVobs, PRMDATA *Prm, HPRMDAT *Hprm )
static  BOOL  tuneLessCalEhSpdObsPrm( TUNELESS_CTRL *TuneLessCtrl, USHORT trqfil11,
BPRMDAT *Bprm, TRANSDUCE_CTRL *TransduceCtrl )
BOOL  EhVobsCalculatePrm( EHVOBS_CTRL *EhVobsCtrl, BPRMDAT *Bprm,
EHVOBS_CFGPRM *CfgPrm, TRANSDUCE_CTRL *TransduceCtrl )

```

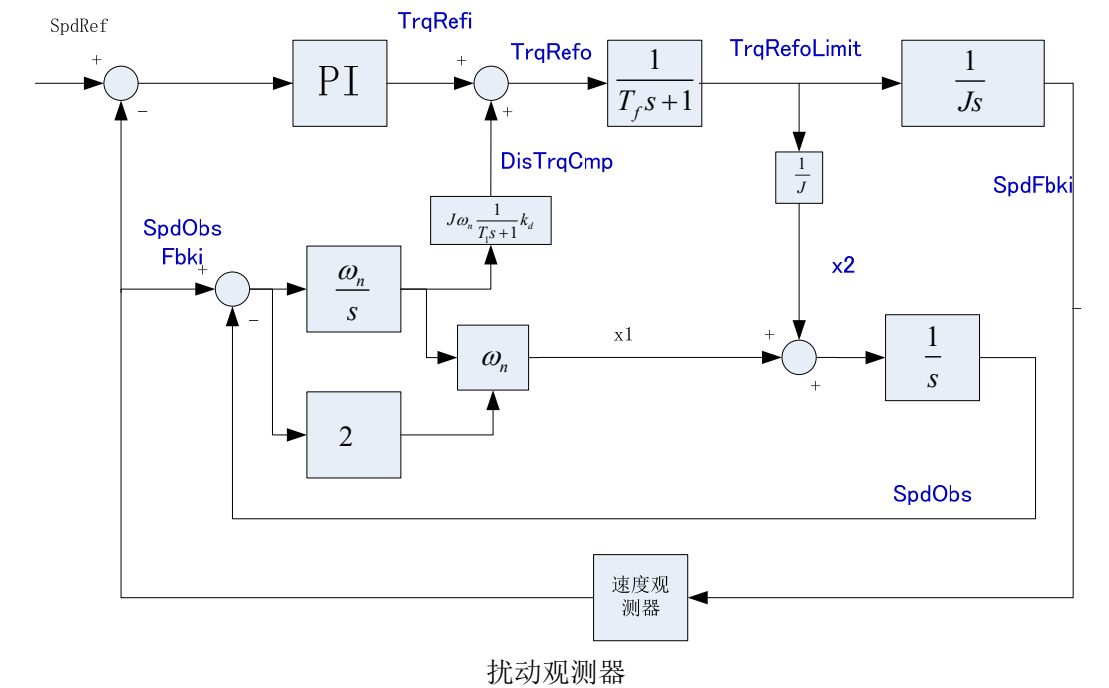

2.6 摩擦补偿（扰动观测器）

摩擦补偿有很多种方式，有通过建立摩擦模型来展开的，也有通过扰动观测器来实现的，这里我们先分析下安川的摩擦补偿的方式，然后再去做进一步的原理推导。

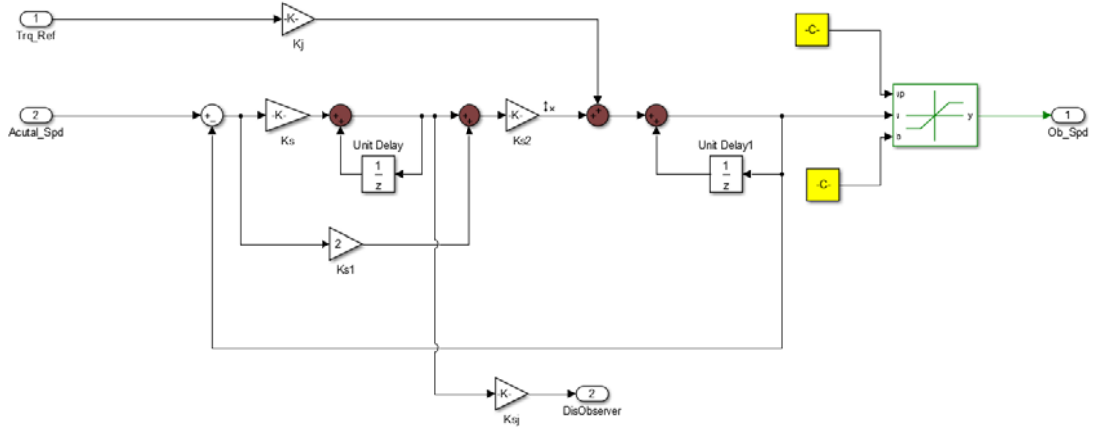
安川摩擦补偿的功能码如下图所示，通过功能码能很快的定位到相关的代码，通过代码可以分析出来安川的摩擦补偿是通过扰动观测器来实现的。

Pn408.3 是否开启摩擦补偿功能

Pn121	2	摩擦补偿增益	10 ~ 1000	1%	100	即时生效	调谐	6.8.2
Pn122	2	第2摩擦补偿增益	10 ~ 1000	1%	100	即时生效	调谐	
Pn123	2	摩擦补偿系数	0 ~ 100	1%	0	即时生效	调谐	
Pn124	2	摩擦补偿频率补偿	-10000 ~ 10000	0.1Hz	0	即时生效	调谐	
Pn125	2	摩擦补偿增益补正	1 ~ 1000	1%	100	即时生效	调谐	



通过代码建立如下的 Matalab 模型。



$$k_j = \frac{t_s}{10^9} \frac{Pn125}{100} \frac{1}{J}$$

$$f = Pn100 \text{ or } Pn104$$

$$k_s = 2\pi f \frac{Pn121}{100} \frac{t_s}{10^9}$$

$$k_{sj} = \frac{2\pi f J \frac{Pn121}{100}}{\frac{Pn125}{100}}$$

$$k_d = \frac{Pn123}{100}$$

$$\omega_k = Pn124 + 4 \times f$$

$$T_1 = \frac{1}{2\pi\omega_k}$$

在摩擦补偿使能切换的过程中,为了避免大的抖动,相当于进行了一个一阶低通滤波器处理,在速度环积分输出环节添加一半的摩擦补偿量,进行抵消。

2.6.1 扰动观测器原理分析

电机的运动学方程:

$$J \frac{d\omega}{dt} = T_e - T_l$$

我们这里可以把 T_l 当做负载和扰动的总和。

以电机转速 ω 和负载扰动 $-T_l$ 作为观测变量,构建电机的模型矩阵如下所示:

$$\begin{aligned} x_1 &= \omega \\ x_2 &= -T_l \\ \dot{x}_1 &= \frac{d\omega}{dt} = \frac{1}{J}(T_e + x_2) \end{aligned}$$

$$\dot{x}_2 = 0$$

则可以得到电机的运动方程矩阵如下所示：

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u$$

$$y = \mathbf{C}\mathbf{x}$$

有

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{J} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1/J \\ 0 \end{bmatrix} T_e$$

$$y = [1 \ 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

其中， $\mathbf{A} = \begin{bmatrix} 0 & \frac{1}{J} \\ 0 & 0 \end{bmatrix}$ ， $\mathbf{B} = \begin{bmatrix} \frac{1}{J} \\ 0 \end{bmatrix}$ ， $\mathbf{C} = [1 \ 0]$ 。

显然系统可控可观测，根据上述矩阵建立全阶状态观测器：

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}u - \mathbf{L}(\hat{y} - y)$$

$$\hat{y} = \mathbf{C}\hat{\mathbf{x}}$$

由上两式可得

$$\dot{\hat{\mathbf{x}}} = (\mathbf{A} - \mathbf{L}\mathbf{C})\hat{\mathbf{x}} + \mathbf{B}u + \mathbf{L}y$$

其中

$$\mathbf{L} = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix}$$

$$\mathbf{A} - \mathbf{L}\mathbf{C} = \begin{bmatrix} \lambda + L_1 & -\frac{1}{J} \\ L_2 & \lambda \end{bmatrix}$$

则其特征方程为 $\lambda^2 + L_1\lambda + \frac{1}{J}L_2 = 0$ ，设计观测器时选取特征根为二重根情况下的特征

方程为 $\lambda^2 + 2\omega_n\lambda + \omega_n^2 = 0$ ，则可以得到观测矩阵增益值为：

$$L_1 = 2\omega_n$$

$$L_2 = \omega_n^2 J$$

将观测器增益矩阵代入到扰动观测器方程中，可以得到。

$$\dot{\hat{x}}_1 = \frac{1}{J}x_2 + \frac{1}{J}T_e + 2\omega_n(y - \hat{y})$$

$$\dot{\hat{x}}_2 = J\omega_n^2(y - \hat{y})$$

则可以得到 x 的计算公式：

$$x_1 = \int \left(\int \omega_n^2(y - \hat{y}) + 2\omega_n(y - \hat{y}) + \frac{1}{J}T_e \right)$$

$$x_2 = \int J\omega_n^2(y - \hat{y})$$

则由 x1 和 x2 的计算公式，可以得到扰动观测器的算法，和安川做法一致。

代码实现如下所示：

```
1. u1 = MotSpd - Dobs.V.SpdObs;
2. /*积分项计算，积分增益为 w^2*/
3. u2 = MlibIntegral( u1, Dobs.P[0].Ks, Dobs.V.Ivar64 );
4. /*比例加积分项的计算，比例增益为 2*w*/
5. u1 = u2 + ( u1 << 1 );
```

```

6. x1 = MlibMulgainNolim( u1, Dobs.P[0].Ks );
7. x2 = MlibMulgain( TrqRef, Dobs.P[0].Kj );
8. Dobs.V.SpdObs = MlibLimitul( ( Dobs.V.SpdObs + x1 + x2 ), 0x1000000, -0x1000000 );
9. Dobs.V.DisTrq = MlibMulgain( u2, Dobs.P[0].Ksj );

```

这里在原理分析的基础上，结合安川的代码及接口，分析下其工程的实现细节。

① 程序刷新机制及接口


- 扰动观测器只有在免调整关闭后才能被启动
- 扰动观测器的刷新为周期 62.5us，当获取了速度信息后，立刻进行扰动观测器的计算，得到的值可以在本周期立刻更新，不存在滞后。
- 观测出来的扰动信息经过滤波及调幅后直接在当前周期加到转矩指令上，当做转矩前馈来使用。

② 惯量比

从观测器模型来看，观测器模型的准确性受到了惯量比的影响很大，所以设置对的惯量比较为关键，所以在安川的说明书中有相关的说明，如下图所示。

(2) 摩擦补偿功能的操作步骤

摩擦补偿功能的操作步骤如下所示。

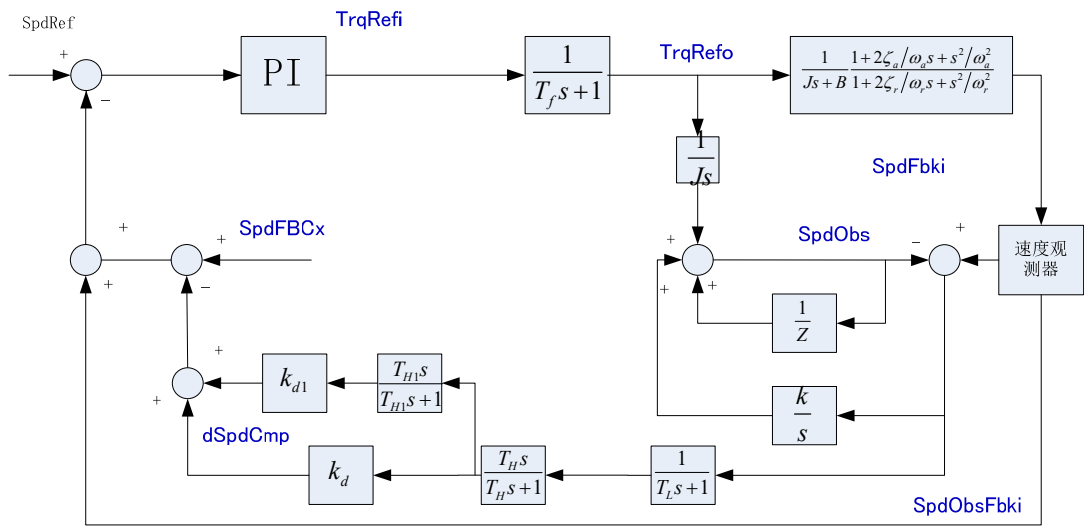
 注意
• 使用摩擦补偿功能时，请尽可能正确地设定转动惯量比（Pn103）。如果转动惯量比设定错误，可能会引起振动。

③ 低通滤波器

在观测出了扰动转矩后，加入低通滤波器，来去除一些观测出来的噪声，那么安川在滤波器这里如何选择？

滤波器的截止频率为 $Pn124 + 4 * \omega_n$ ，其中 Pn124 为摩擦补偿滤波频率补偿，当 Pn124 为 0 的情况下，滤波器的截止频率为观测器增益的 4 倍。

2.7 A 型振动抑制（中频振动抑制）



相关功能码

参数	名称	自动设定的有无
Pn160	防振控制类开关	有
Pn161	A 型抑振频率	有
Pn162	A 型抑振增益补偿	无
Pn163	A 型抑振阻尼增益	有
Pn164	A 型抑振滤波时间参数 1 补偿	无
Pn165	A 型抑振滤波时间参数 2 补偿	无

Pn166 惯性制振阻尼增益

Pn167 惯性制振振动频率

相关系数的计算在此函数中进行 `void ResVibCalculatePrm(RESVIB *ResVib, RESVIB_CFG_PRM *Prm, KSGAIN Kv, LONG ScanTimeNs, ASICS *SvAsicRegs)`

$$k_s = \frac{2\pi f}{2} = \frac{2\pi Pn161}{2}$$

$$T_l = \frac{2}{2\pi f} + Pn164$$

$$T_h = \frac{2}{2\pi f} + Pn165$$

$$k_d = \frac{Pn163}{100}$$

$$T_{h1} = \frac{1}{2\pi Pn167}$$

$$k_d = \frac{Pn166}{100}$$

原理：

其实际模型为一个二惯量振动模型，包括纯刚性部分以及振动部分。观测器只选取纯刚性部分，且忽略摩擦。则实际速度减去观测速度为振动部分速度，通过一阶低通滤波器，高通滤波器以及比例控制部分整形调节输出给反馈速度相加，从而消除振动部分。

等效刚体忽略摩擦，扰动，其数学模型为

$$\frac{d\omega}{dt} = \frac{1}{J} T_e$$

状态方程为

$$\begin{aligned}\dot{x}(t) &= ax(t) + bu(t) \\ y(t) &= cx(t)\end{aligned}$$

其中 $x(t) = v(t)$, $a = 0$, $b = \frac{1}{J}$, $c = 1$

观测器的状态方程

$$\dot{\hat{x}}(t) = (a - kc)\hat{x}(t) + ky(t) + bu(t)$$

三 位置环

3.1 位置指令

在函数 LpxSvPositionManager () 中

```
/*-----*/
/*位置指令更新(125usに1回位置指令が更新される。それを62.5usで位置制御する為の処置)*/
/*-----*/
```

```

if( PosManager->RefRenewCntA != PosManager->RefRenewCntB )
{ /* ScanBで位置指令が更新された後の1回目 */
    /* 1回目ScanAの位置指令を作成 */
    BaseLoops->dPosRefi = PosManager->CompdPosRefi >> 1;    //这个目前不知道什么用
    BaseLoops->dPcmda    = PosManager->CompdPcmda    >> 1; //这个暂时不知道是做什么
    用的 单位为pulse/scan

    if( PosManager->RefRenewalChkCnt == 0 )
    { /* 2回目ScanAが走らなかった場合の処置 SCANA SCANB时序的异常处理*/
        /* 前回の2回目ScanAの位置指令を加算 */
        BaseLoops->dPosRefi  += BaseLoops->NextdPosRefi;
        BaseLoops->dPcmda    += BaseLoops->NextdPcmda;
    }

    /* 2回目ScanAの 位置指令を作成 */
    BaseLoops->NextdPosRefi = PosManager->CompdPosRefi - (PosManager->CompdPosRefi >>
1);
    BaseLoops->NextdPcmda    = PosManager->CompdPcmda    -
(PosManager->CompdPcmda    >> 1); //下一次运算的给定位置偏置值 为余数

    PosManager->RefRenewalChkCnt = 0;
}
else
{ /* ScanBで位置指令が更新された後の2回目 */
    if( PosManager->RefRenewalChkCnt == 0 )
    {
        /* 2回目ScanAの位置指令を作成 */
        BaseLoops->dPosRefi  = BaseLoops->NextdPosRefi;
        BaseLoops->dPcmda    = BaseLoops->NextdPcmda;
    }
    else
    { /* ScanAが3回以上走った場合の処置 */
        /* 3回目以降は指令の作成を行わない */
        BaseLoops->dPosRefi = 0;
        BaseLoops->dPcmda    = 0;
    }
    PosManager->RefRenewalChkCnt++;
}
PosManager->RefRenewCntA = PosManager->RefRenewCntB;
位置给定指令是CompdPcmda在SCANB中计算的，转到SCANA第一次要进行又移1位
处理，第二次要进行又移1位处理，且余数要进行补偿。dPcmda为最终SCANA的位置
给定偏差值，指令单位/SCANA. dPosRefi为位置指令差分，单位pulse/scan

```

3.1.1 低频振动抑制

Pn140	n.□□X□	振动抑制选择		参照章节						
		0	不进行振动抑制。	8-64 页						
		1	对特定频率附加振动抑制功能。							
		2	对 2 种不同的频率附加振动抑制功能。							
Pn145	2	振动抑制 1 频率 A	10 ~ 2500	0.1Hz	500	通用	即时生效	调整	-	
Pn146	2	振动抑制 1 频率 B	10 ~ 2500	0.1Hz	700	通用	即时生效	调整	-	
Pn14A	2	振动抑制 2 频率	10 ~ 2000	0.1Hz	800	通用	即时生效	调整	-	
Pn14B	2	振动抑制 2 补偿	10 ~ 1000	1%	100	通用	即时生效	调整	-	

```

/*****
/*
/*      振動抑制フィルタ演算      振动抑制滤波计算      */
/*
/*      概要： 振動抑制フィルタは，設定された周波数に対するノッチ特性を持
/*      ったフィルタである。      概要： 振动抑制滤波器是具有相对于设定频率的陷
/*      波特性的滤波器。 */
/*      機台振動など位置決め時の振動を抑制する。 抑制机台振动等定
/*      位时的振动。 */
/*      移動平均等位置指令フィルタ，MFCとも直列に接続可能。 */
/*      移动平均等位置指令滤波器、MFC也可串联连接。 */
/*****
LONG  PcmdFilVibSupFilter( VIBSUPFIL *VibSupFil, LONG dPcmd, BOOL
*RefZSignal )
{
    LONG  AvffFili;
    LONG  AvffFilo;
    LONG  x1;
    LONG  x2;
    LONG  x3;
    LONG  wk = 0;

/*-----*/
/* 指令完了&払い出し完了チェック 指令完成&支付完成检查 */
/*-----*/
    if( (dPcmd == 0) && (VibSupFil->var.Buf == 0) )
    { /* 指令完了 && フィルタ溜まりなし->払い出し完了 */
        VibSupFil->conf.Pexe = VibSupFil->conf.VibSupPrm;
/* パラメータ切替え実行 执行参数切换 */
    }
    AvffFili = dPcmd;
/* 振動抑制フィルタ←位置指令パルス振动抑制滤波←位置指令脉冲 */

```

```

/*-----*/
/*      振動抑制フィルタ演算  振动抑制滤波计算      */
/*-----*/
    if( VibSupFil->conf.Pexe.enable )
    { /* 機能有効 */
        x1 = MlibPfbkxremNolim( AvfffFili, VibSupFil->conf.Pexe.Kff1,
&VibSupFil->var.rem1 );
        x2 = MlibPfbkxremNolim( VibSupFil->var.wkbf1,
VibSupFil->conf.Pexe.Kff2, &VibSupFil->var.rem2 );
        x3 = MlibPfbkxremNolim( VibSupFil->var.wkbf2,
VibSupFil->conf.Pexe.Kff3, &VibSupFil->var.rem3 );

        VibSupFil->var.wkbf1 += (x1 - x3 - AvfffFili);
        VibSupFil->var.wkbf2 += (x1 - x3 + x2);

        AvfffFilo = x1 - x3; /* 振動抑制フィルタ出力 振动抑制滤波输出 */

/*-----*/
/*      フィルタ入出力偏差(溜まり) 过滤器输入输出偏差（累积）      */
/*-----*/
        VibSupFil->var.Buf = MlibPerrcalx( AvfffFili, AvfffFilo,
VibSupFil->var.FilioErr );
        if( VibSupFil->var.Buf != 0 )
        { /* 位置指令払い出し 未完了位置指令 支付未完成 */
            *RefZSignal = FALSE;
        }

/*-----*/
/*      振動抑制フィルタ変数初期化 振动抑制滤波变量初始化 */
/*-----*/
        if( (AvfffFilo == 0) && (VibSupFil->var.wkbf1 == 0) &&
(VibSupFil->var.wkbf2 == 0) )
        { /* 振動抑制フィルタ出力=0 && 振動抑制フィルタBUF=0 */
            /* 振動抑制フィルタ変数初期化 */
            MlibResetLongMemory( &VibSupFil->var,
sizeof(VibSupFil->var)/4 );
        }

/*-----*/
/*      フィルタ出力処理      */
/*-----*/
        wk += AvfffFilo; /* 機能無効状態時は */
        VibSupFil->var.Filo = wk; /* フィルタ出力+指令入力を出力 */
    }

```



```

else
{
    /* 機能無効 */
    wk = AvffFili; /* 入力をそのまま出力  输入按原样输出 */
    VibSupFil->var.Filo = wk;
}

return( wk );
}

```

```

/*****
/*
/*      振動抑制フィルタパラメータ計算  振动抑制滤波参数计算 */
/*
/*
/*****
void PcmdFilCalculatePrmVibSupFil( VIBSUPFIL *VibSupFil, ULONG ff_frq,
ULONG ff_fil, LONG ScanTimeNs )
{
    VIBSUPFILPRM  wrkp;
    LONG          ul;
    LONG          s, sx;
    LONG          kx;
    LONG          wk;
    LONG          omega2;
// LONG          scantime;

/*-----*/
/*      演算周期設定          */
/*-----*/
// scantime = AVFILCALCYCLE; /* 演算サイクルタイム    [ns] */

/*-----*/
/*      機能選択    功能选择          */
/*-----*/
    wrkp.enable = VibSupFil->conf.VibSupSetting;
    ul = ff_fil;

/*-----*/
/*      ω 2 最大値          */
/*
/*
/*      (ff_frq/10) [Hz]*(ff_fil[%]/100)*(80/100)*10^9 */
/* omega2_limit =-----*/
/*
/*      2 * PAI * CycleNs          */

```

```

/*-----*/
    wk = (LONG)ff_frq * (LONG)ff_fil;
/* パラメータ設定 ω2*1000[Hz] 参数设定 */
    omega2 = MlibMIN( wk, (MAXOMEGA2(ScanTimeNs) * 1000) );
/* *1000[Hz]リミットとパラメータ値の最小値 极限和参数值的最小值 */

/*-----*/
/*      振動抑制フィルタゲイン1 振动抑制滤波器增益1 Avff.P.Kff1 */
/*-----*/
/*      
$$Kff1 = \frac{ff\_fil^2}{10000}$$
 */
/*-----*/
    wrkp.Kff1 = MlibScalKxgain( u1 * u1, 1, 10000, NULL, 24 );

/*-----*/
/*      振動抑制フィルタゲイン2 Avff.P.Kff2 */
/*-----*/
/*      振動抑制フィルタゲイン3 Avff.P.Kff3 */
/*-----*/
/*      
$$Kff2 = \frac{PAI * omega2 * CycleNs}{1000 * 10^9}$$
 */
/*      
$$Kff3 = \frac{4*PAI * omega2 * CycleNs}{1000 * 10^9}$$
 */
/*-----*/
    kx = MlibScalKxgain( omega2, ScanTimeNs, C10POW9, &s, 0 );
    kx = MlibPcalKxgain( kx, 1, 1000000000, &s, 0 );
    sx = s;

    wrkp.Kff2 = MlibPcalKxgain( kx, 3141593, 1, &s, 24 );
/* 3141593 = PAI*1000000 */
    wrkp.Kff3 = MlibPcalKxgain( kx, 12566371, 1, &sx, 24 );
/* 12566371 = 4*PAI*1000000 */

/*-----*/
/*      Set Parameters */
/*-----*/
// KPI_DI(); /* Disable Interrupt */
    VibSupFil->conf.VibSupPrm = wrkp;
// KPI_EI(); /* Enable Interrupt */

```

}

$$\omega_2 = \frac{f}{2\pi} \frac{Pn148}{100} \frac{80}{100} \frac{1}{T_s}$$

$$k_1 = \left(\frac{Pn148}{100} \right)^2$$

$$k_2 = \pi \omega_2 T_s = \frac{f}{2} \frac{Pn148}{100} \frac{80}{100} \frac{1}{1000}$$

$$k_3 = 4\pi \omega_2 T_s = 2f \frac{Pn148}{100} \frac{80}{100} \frac{1}{1000}$$

滤波器差分方程

$$Y_1(n) = Y_1(n-1) + (k_1 x(n) - k_3 Y_2(n-1) - x(n))$$

$$Y_2(n) = Y_2(n-1) + (k_1 x(n) - k_3 Y_2(n-1) + k_2 Y_1(n-1))$$

$$Y(n) = k_1 x(n) - k_3 Y_2(n-1)$$

3.2 位置环计算

在此函数中BasePositionControlA

```

/*-----*/
/*      位置偏差クリア処理 位置偏差清除处理      */
/*-----*/
if( PerClrCmd )
{
    PosCtrl->V.PerIvar64[0] = 0;
    PosCtrl->V.PerIvar64[1] = 0;
    PosCtrl->V.LastPacOut    = 0;
    return( 0 );
}
/*-----*/
/*      位置制御積分クリア処理      位置控制积分清除处理      */
/*-----*/
if( IcvClrCmd )
{
    PosCtrl->V.PerIvar64[0] = 0;

```

```

    PosCtrl->V.PerIvar64[1] = 0;
}
PosMngV->var.PosErr = MlibPerrcalx( BaseLoops->dPosRefi, BaseLoops->dPosFbki,
PosMngV->var.Per64 ); //总的位置偏差计算

```

```

PosCtrl->V.PacOut = MlibMulgain( PosMngV->var.PosErr, Kp ); //位置比例计算

```

位置比例的计算

```

/*-----*/
/*  FBパルス変換係数[rad/pulse]の計算      位置变换系数的计算 由pulse转换为rad      */
/*-----*/
#if (FLOAT_USE==TRUE)
    fw = (float)C2PAIE7 / (float)Bprm->FbPulse;
    Bprm->Kfbpls = fw / (float)C10POW7;
#else
    kx = MlibScalKxgain( C2PAIE7, 1, Bprm->FbPulse, &sx, 0 );
    Bprm->Kfbpls = MlibPcalKxgain( kx, 1, C10POW7, &sx, -1 );
#endif /* FLOAT_USE */

```

其中C2PAIE7为62831853，FbPulse为一圈脉冲数，其计算式为

$$K_{fbpls} = \frac{62831853}{10000000 \times FbPulse} = \frac{2\pi}{FbPulse}$$

```

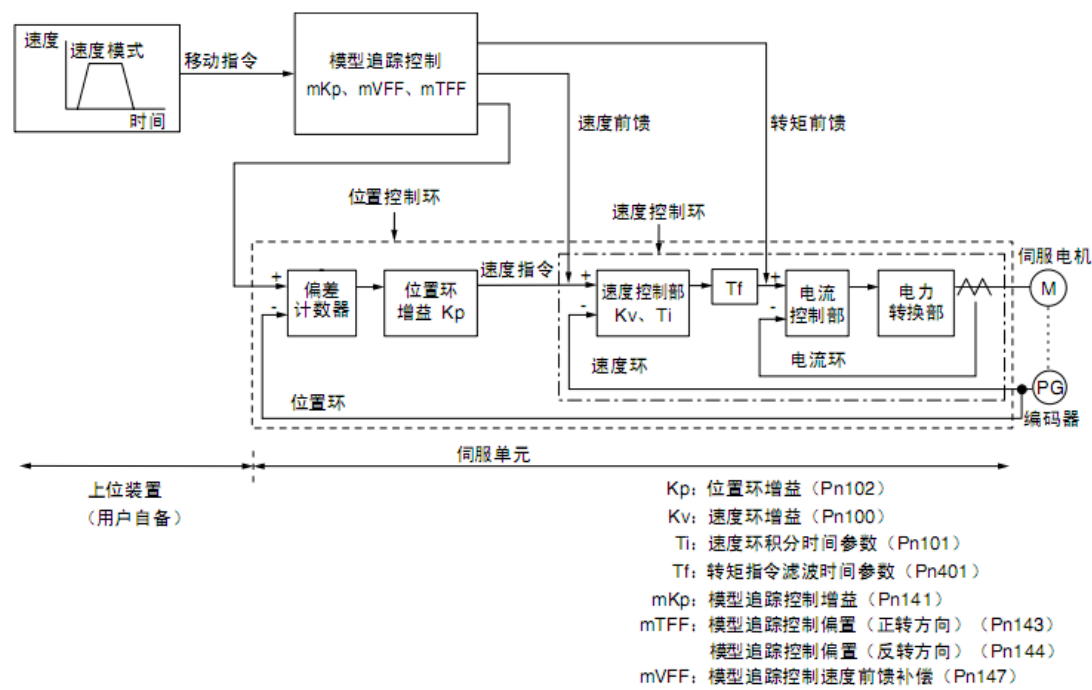
/*-----*/
/*  位置制御ゲイン中間パラメータの計算      Rotary      Linear      */
/*-----*/
/*
/*          Kfbpls * 2^24          Kfbpls : [rad/pulse]      [m/pulse]      */
/*      Kpx = -----          OvrSpd : [rad/s]      [m/s]      */
/*          OvrSpd
/*
/*-----*/
#if (FLOAT_USE==TRUE)
    Bprm->Kpx = (float)0x1000000 * Bprm->Kfbpls / Bprm->OvrSpd;
#else
    Bprm->Kpx = MlibScalKxksks( 0x1000000, Bprm->Kfbpls, Bprm->OvrSpd, &sx, -1 );
#endif /* FLOAT_USE */

```

从中可看出位置环的输出已经转换为了速度环的给定，其单位为rad/s的Q格式， $2^{24}/OvrSpd$ 。

四．重要功能模块

4.1 模型追踪控制



模型追踪功能码

- Pn140 (模型追踪控制类开关)
- Pn141 (模型追踪控制增益)
- Pn143 (模型追踪控制偏置 (正转方向))
- Pn144 (模型追踪控制偏置 (反转方向))
- Pn147 (模型追踪控制速度前馈补偿)

■ 模型追踪控制类开关

通过 Pn140 = n.□□□X 选择使用 / 不使用模型追踪控制。

Pn14F 模型追踪类型 0刚体 1两惯量模型

Pn140---Pn149

相关系数的计算void MfcCalculatePrm(MFCTRL *MFCtrl, MFC_CFG_PRM *MfcCfgPrm, USHORT jrate, BPRMDAT *Bprm, LONG cfrc, LONG GselNo)

$$\omega_n = 4\xi k_m = 4 \frac{Pn142}{1000} \frac{Pn141}{10}$$
$$f = \frac{\omega_n}{2\pi}$$

$$\frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

パラメータ演算式	備 考
$Kf[0] = \frac{T_s}{T_f + T_s} = \frac{(dx*hz*T_s)}{(10^{10}/4\pi) + (dx*hz*T_s)}$	Ts: フィルタ演算周期 [μs] hz: フィルタ周波数 [0.1Hz] dx: フィルタ減衰係数ζ [0.001]
$Kf[1] = \frac{\omega_n*T_s}{2\zeta*10^6} = \frac{100*2\pi*hz*T_s}{2*dx*10^6}$	$\omega_n = 2\pi*hz/10$ $\zeta = dx/1000$ $T_f = 10^6/(2\zeta\omega_n) = 10^{10}/(2*dx*2\pi*hz)$ 注)演算時間短縮のため、Kf[0]は、2 ²⁴ 倍しておく

$$Kf2 = \frac{Kf[0]}{0x1000000}$$

$$Kf3 = \frac{Kf[0]}{T_s}$$

/*モータOS速度パルス計算 [Pluse/Scan] 马达OS速度脉冲计算 Bprm.MaxPspd:[Pulse/sec]*/
maxpls = (scantime * Bprm->MaxPspd / (float)C10POW9);

$$Fsft = 2^{30} / (\max pls / Kf[1])$$

$$InvFsft = \frac{1}{Fsft}$$

机械系数増益

$$Kj1 = J_{\text{总}}$$

$$Kj2 = \frac{1}{(2\pi Pn145)^2 \times Fsft \times T_s}$$

$$Kj2_sft = \frac{1}{(2\pi Pn145)^2 \times T_s}$$

$$Kj3 = \frac{1}{(2\pi Pn146)^2 \times Fsft \times T_s}$$

DTR速度前馈増益

$$kvff = kpx \frac{Pn147}{1000} \frac{1}{T_s}$$

DTR转矩前馈増益

正转

$$k_{tff1} = \frac{Pn143}{1000}$$

反转

$$k_{tff2} = \frac{Pn144}{1000}$$

库伦摩擦补偿扭矩: 0.1%

`wrkp.Cfric = cfric*10 * Bprm->Ktrqlvl;`

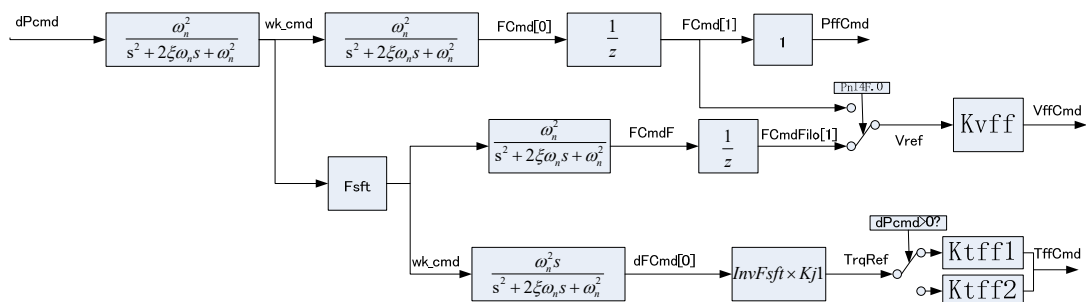
库伦摩擦补偿滤波器增益

$$k_{fcf} = \frac{T_s}{\frac{5000000}{Pn141} + T_s}$$

```

/*****
/*
/*      刚体MFC演算
/*
/*
/*      概要:      刚体MFC演算を行う      执行刚体MFC操作
/*      2次フィルタ×1段を通し, MFC後位置指令を作成。      通过二阶滤波器 x 1阶段创建MFC后位置命令
/*      FF用1階微分は, 分解能UPのため, 別の2次フィルタを使用する      由于分辨率UP, FF的一阶导数使用另一个二阶滤波器。
/*
/*
/* arg      : LONG   dPcmd      :位置差分指令 [pluse]
/* out      :
/* g-out    :
/*
/*
*****/
static void mfcControlOneMass( MFCtrl *MFCtrl, LONG dPcmd )

```



```

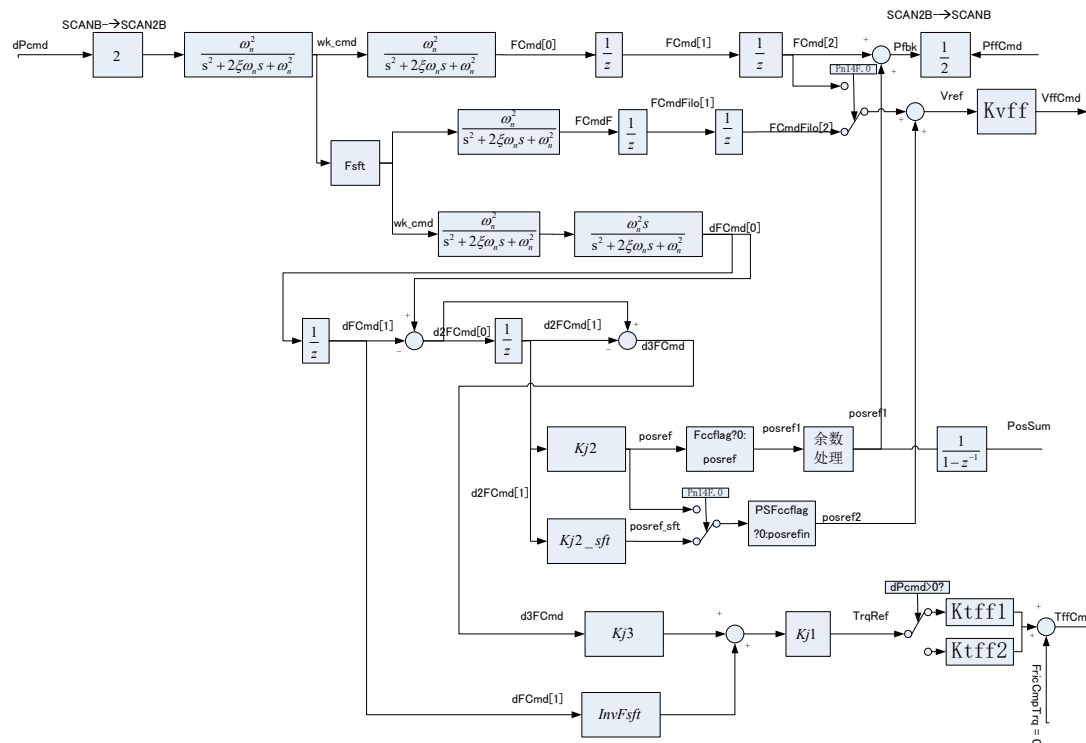
/*****
/*
/*      二慣性, 剛体+機台MFC演算
/*
/*
*****/

```

```

/* 概要: 二慣性MFC演算を行う */
/* 2次フィルタ×2段を通し, MFC後位置指令を作成。 */
/* FF用1階微分は, 分解能UPのため, 別の2次フィルタを使用する */
/*
/* arg : LONG dPcmd :位置差分指令 [pluse] */
/* out : */
/* g-out : */
/*
/*
/*****
static void mfcControlTwoMass( MFCTRL *MFControl, LONG dPcmd )

```



其刷新周期为 250us, 分两个时间片 125us 完成。是否是考虑程序执行时间, 才做出了这个取舍。

```

/*****
/*
/* 1階微分値算出用2次フィルタ演算用于计算一阶微分值的二阶滤波器计算 */
/*
/*****
static LONG mfcDtrCmdFilter( MFCTRL *MFControl, LONG pcmd )
滤波器的计算, 此函数的作用是否为一阶微分后再加一个二阶低通滤波器

```

$$dFilwk(n) = dFilwk(n-1) + Kf2 \times [(x(n) - Filo(n-1)) - dFilwk(n-1)]$$

$$Filo(n) = Filo(n-1) + Kf[1] \times dFilwk(n)$$

4.2 基本中间参数计算处理

在BprmCalc.c文件中计算有关伺服驱动器的中间变量值，目的是单位转换，标么化，节省程序运行时间。

包括位置指令用户单位，电机单位，速度指令，速度反馈，转矩百分比，转矩，电流，电压等单位之间的换算，以及刷新时间，周期，监控参数换算等。

4.3 最大电流的计算

最大电流取：min[电机最大电流，驱动器最大电流]。
PnFOA 电机瞬时最大电流，不知道其表示的有效值还是峰值
PnE15 伺服驱动器最大电流，不知道其表示的有效值还是峰值
最大电流作为基准值进行标么化计算。若选取的是电机最大电流，则比例系数Kmottrq为1；若选取的是驱动器最大电流，则比例系数Kmottrq为：电机最大电流/驱动器最大电流。
按道理在转矩标么值转换为电流标么值的时候应乘此系数，但程序并没有用到，是没考小马拉大车吗？

4.4 标么化基值选取

基准值	定点型标么值	浮点型标么值		
OvrSpd	2 ²⁴	100.0f		
MaxTrq	2 ²⁴			
MaxTrq	15000			
MaxCur	15000			
Udc	2 ¹⁴			
演变标么值				
名称	计算	定 点 型 标	浮 点 型 标 么 值	说明

		么值		
NorMaxSpd	$\text{NorOvrSpd} \times \frac{10000}{10000 + \text{PerOvrSpd}}$	2^{24}	100.0f	超速标么值是在最大速度往上调一点
NorRatSpd	$\text{NorMaxSpd} \times \frac{vrat}{\text{MaxVel}}$	2^{24}	100.0f	电机最大速度，额定速度为电机设定参数
pNorOvrSpd			100.0f	
nNorOvrSpd			-100.0f	
NorSuspSpd	$\text{NorMaxSpd} \times \frac{1}{10}$			

4.5 免调整功能

出厂设定中，免调整功能为“有效”，无需操作。免调整功能的有效、无效通过以下参数来选择。

参数	含义	生效时间	类别
Pn170	n.□□□0	再次接通电源后	设定
	n.□□□1 [出厂设定]		
	n.□□0□ [出厂设定]		
	n.□□1□		

免调整功能有效时，可选择免调整型。通常请设为 Pn14F = n.□□2□（免调整 3 型）[出厂设定]。只有需要与以往产品兼容时，设为 Pn14F = n.□□0□（免调整 1 型）或 = n.□□1□（免调整 2 型）。

参数	含义	生效时间	类别
Pn14F	n.□□0□	再次接通电源后	调整
	n.□□1□		
	n.□□2□ [出厂设定]		

Pn170.1=0 速度控制时位相补偿无效 Pn170.1=1速度控制时位相补偿有效

在ExControls.c

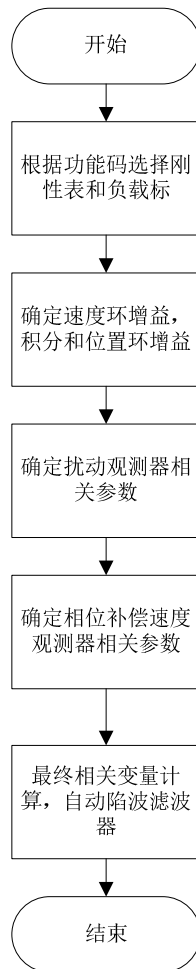
ExCtrlPrmCalc.c

/*****

```

/*                                     */
/*      調整レス関連ワンパラパラメータ計算      免調整相关的参数计算      */
/*                                     */
/*****
BOOL    TuneLessCalculatePrm( BASE_LOOPCTRLS *BaseLoops, BPRMDAT *Bprm,
                                TUNELESS_SETPRM    *Prm,    BASE_SPDLOOP_PRM
                                *SpdCtrlPrm )

```



速度环增益根据刚性表进行设置，其中还有一个百分比参数进行速度环增益调节。

PI控制时

$$T_i = \frac{5}{2\pi K_v}$$

$$K_p = K_v$$

I-P控制时

$$T_i = \frac{2}{2\pi K_v}$$

$$K_p = \frac{25}{32} K_v$$

旋转型电机扰动观测器相关参数

$$f1 = PnEDA$$

$$f2 = 8000$$

$$Stiff = PnEDB$$

转矩补偿后的转矩滤波器时间常数

$$Klpf = 0.1ms$$

鲁棒补偿器系数

$$kd_dist = 99$$

扰动观测器参数

$$kj_dist = 100$$

相位补偿速度观测器相关参数

相位补偿速度观测器增益

$$ks_vobs = 60$$

相位补偿速度观测器惯量增益

$$kj_vobs = 100$$

相位补偿速度观测器LPF时间常数[0.01ms]

$$vobsLpf = \frac{64000000}{VOBSLPFGNHZ \times kv} = \frac{64000000}{DISOBSFSTD/TUNELESSJRAT \times kv} = \frac{64000000}{kv \times 6000/30}$$

前馈相关参数

相位补偿滤波器频率1 [0.1Hz]

$$vffPhCmpFrq1 = \frac{12800}{vobsLpf}$$

相位补偿滤波器频率2 [0.1Hz]

$$vffPhCmpFrq2 = 10000$$

扭矩滤波器相关参数

包括两个一阶低通滤波器，一个时间常数时Pn401，一个时间常数是0.1ms，可参考方框图

扰动观测器相关参数

$$kj_dist2 = kj_dist \frac{Stiff}{100}$$

校正后的干扰观测器低通滤波器1频率 [0.1Hz]

$$\text{flComp} = f1 \frac{\text{Flrate}}{100}$$

补正后相位补偿速度观测器惯量增益 [%]

然后运行tuneLessCalEhSpdObsPrm子函数

$$kf = \frac{ts}{tx + ts} = \frac{ts}{ts + \frac{1}{2\pi f}}$$

相位超前补偿增益

自动陷波滤波器幅值

```
AutoNotchCalculatePrm( Prm->ANotchSeq, TuneLessPrm->kv, TuneLessPrm->jrate );
```

振动检测参数设定

```
DetVibObsCalculateGains( &(Prm->DetVib->DetVibObs),
                        VIBOBS_KS, DETVIBLP, DETVIBHP,
                        Bprm->SvCycleNs );
```

其中振动检增益VIBOBS_KS=80.0hz，低通滤波器时间常数DETVIBLP=3183us,相当于50hz,高通滤波器时间常数DETVIBHP=320us，**后续弄清楚振动检测方式**

109

```
static  BOOL    tuneLessCalRobustCompPrm( TUNELESS_CTRL *TuneLessCtrl, BOOL
LpassFil3,          /* <S066> */
                                BPRMDAT *Bprm, TUNELESS_SETPRM *Prm,
TRANSDUCE_CTRL *TransduceCtrl )
```

当免调制类型Pn14F.1设为0和2时，低通滤波器不起作用；当设置为1时，若当设置了低鲁棒性并且连接了SGMAV和SGMJV电机时，编码器分辨率为13位以下时，调整量较小，低通滤波器不起作用，其余有

$$\text{TuneLessTrqLpf2} = PnEDC$$

$$\text{TuneLessVfbLpf2} = PnEDD$$

速度反馈滤波器选择

选择较大的PnEA8和PnEDD滤波时间常数，再计算KVfbFil，为一阶低通滤波系数

扭矩滤波器的选择根据免调制类型以及低增益类型电机的选择，选择是否开通。若开通，则选择较大的PnEA8和PnEDC时间常数。然后计算一阶低通滤波系数KCmpOutlpf。

鲁棒补偿器低通滤波器增益1

$$\text{Krlpf1} = \frac{ts}{ts + \frac{1}{2\pi \times f1 \text{Comp}}}$$

鲁棒补偿器低通滤波器增益2

$$\text{Krlpf2} = \frac{ts}{ts + \frac{1}{2\pi \times f2}}$$

鲁棒补偿器系数

$$\text{Krd} = \frac{kd_dist}{100}$$

```
/*-----*/
/*      ロバスト補償器トルク補正ゲイン f1[0.1Hz], kj [%] : Drcomp.conf.Krsj      鲁棒补偿器
扭矩校正增益      */
/*
/*      OvrSpd * Jmot * (100+Prmjrate)/100 * 2*PAI * ObsJGain[%] * Lpf[0.1Hz] */
/*      = ----- */
/*      MaxTrq */
/*
/*      = kvx * (100+Prmjrate)/100 * ObsJGain[%] * Lpf[0.1Hz] */
/*
/*-----*/
```

$$\text{Krsj} = Jmot \times \frac{100 + jrate}{100} \times 2\pi \times f1 \text{Comp} \times \frac{kj_dist2}{100}$$

鲁棒补偿器的选择

TuneLessInitRobustCompensator(TuneLessCtrl, &wrkp);

可选择两种方式

tuneLessRobustCompensator和tuneLessRobustCompensatorEx

```

/*****
*****/
/*
    */
/*      調整レスロバスト補償器演算(SGDV互換方式)      免調整鲁棒补偿器计算
    */
/*
    */
/*****/
/*
    */
/* 概要： トルク指令および差分速度に基づき変動トルクを推定し、      根据扭矩指令和
差速估算波动扭矩      */
/*      トルク指令に足して新たなトルク指令とする。      添加到扭矩命令
以创建新的扭矩命令      */
/*       $Tr1 = Tr0 + K_f * LPF\{Tr0 - LPF(J * \frac{d\omega_m}{dt})\}$       */
/*      */
/*  arg  : LONG   MotSpd      : 速度フィードバック      [2^24/OvrSpd]速度反馈      */
/*      LONG   TrqRef      トルク指令      [2^24/MaxTrq]      */
/*  out   :      */
/*  g-out      */
/*      */
/*****/
//<S1A5> static void tuneLessRobustCompensator( TUNELESS_DRCOMP *Drcomp, LONG
MotSpd, LONG TrqRef )
void tuneLessRobustCompensator( void *pdrcomp, LONG MotSpd, LONG TrqRef ) /* <S1A5> */

```

速度反馈滤波

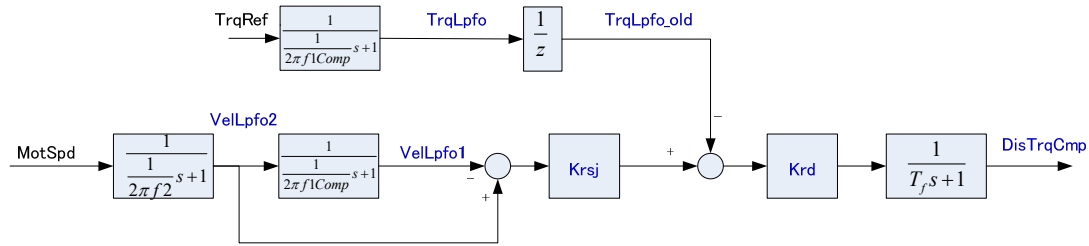
```

Drcomp->var.VelLpfo2 = MlibLpfilter1( MotSpd, Drcomp->conf.Krlpf2, Drcomp->var.VelLpfo2 );
Drcomp->var.VelLpfo1 = MlibLpfilter1( Drcomp->var.VelLpfo2, Drcomp->conf.Krlpf1,
Drcomp->var.VelLpfo1 );

```

$$x(k) = x(k-1) + \frac{T_s}{T_f + T_s} (u(k) - x(k-1))$$

速度反馈经过两个一阶低通滤波器。



用来求出摩擦补偿量，但采用的不是 $J \cdot d\omega_m / dt$ 求反馈转矩，而是采用类似PI控制中的P控制求出转矩量。为什么叫做鲁棒补偿器。

void tuneLessRobustCompensatorEx(void *pprm, LONG MotSpd, LONG TrqRef)

先不用此鲁棒计算

```

/*****
/*
/*      調整レス制御                      免調整制御                      */
/*
/*
/*****
/*
/*      概要 :調整レス用ロバスト補償器および位相補償速度オブザーバをコールする      */
/*      摘要称为免调整鲁棒补偿器和相位补偿速度观测器                      */
/*      arg      : LONG   MotSpd      : モータ速度 电机速度          [2^24/OvrSpd] */
/*      LONG   TrqRefi: フィルタ前トルク指令 滤波前扭矩指令    [2^24/MaxTrq] */
/*      LONG   TrqRef      : トルク指令      扭矩指令      [2^24/MaxTrq] */
/*      out      :
/*      g-out: LONG   AoutV.TrqFBCx : トルクFB補償 扭矩前馈补偿    [2^24/MaxTrq] */
/*      LONG   AoutV.SpdFbki      : 速度フィードバック速度反馈 [2^24/OvrSpd] */
/*      LONG   BoutV.AmonDisTrq    : 外乱トルク 扰动扭矩          [2^24/MaxTrq] */
/*      LONG   BoutV.AmonDisTrqCmp : 補償トルク 补偿扭矩          [2^24/MaxTrq] */
/*      LONG   BoutV.AmonSpdObs    : 推定速度 观测速度          [2^24/OvrSpd] */
/*
/*****
void TuningLessCtrl( TUNELESS_CTRL *TuneLessCtrl, KSGAIN MotSpd, KSGAIN TrqRefi, KSGAIN
TrqRef )      /*<S00A>*/

```

速度反馈先经过一个一阶低通滤波器，再通过鲁棒补偿器计算摩擦补偿量，也可叫扰动补偿量。滤波后的反馈速度经过相位补偿速度观测器，输出观测速度。最后是一些状态的判别。

扰动补偿量加到前馈补偿中

BaseLoops->TrqFBCx += TuneLessGetDisTrqCmp(TuneLessCtrl);