마이크로프로세서 속도제어 보고서 (LPF factor, PI Gain)

자동차IT융합학과 20183376
박선재

```
#include "MPC5604P_M26V.h"
#include "freemaster.h"
#include "init_base.h"

/********************* Macro **********************/
#define LPF(out, in, fct)  out = out-in>0 ? fct*(out-in)+ in :-fct*(in-out)+ in
#define FILFCT(wc, fct, ts)  fct =  ts/(wc+ts)
#define bound(in,lim)  ((in > (lim)) ? (lim) : ((in < -(lim)) ? -(lim) : in));
#define abs(x) ((x > 0) ? x : -x)
#define rad2rpm 9.54929
#define TWO_PI   6.28316

/********************* Variables **********************/
volatile int i = 0, temp10ms=0;
volatile int j = 0;
int Reset = 1, DSPRUN = 0, SpdCon = 0;
float IdcPre=0.0f, IdcErr=0.0f, IdcRef=0.0f, IdcFdb=0.0f, IdcOffset=0.0f;
float VdcRef=0.0f, VdcPterm=0.0f, VdcPiterm=0.0f, VdcIterm=0.0f;
float VdcFil=0.0f, VdcPre=0.0f, vRefUlim=0.0f, PWM_Scale=0.0f;
float Kpc=0.0f, Kic=0.0f;
int PWM_B=0, PWM_Peak=0; uint8_t ErrReset = 0;
long int delta_m=0, mold=0, mnew = 0;
float rpm = 0., rpmFil = 0., wrTemp = 0., wr = 0., wrFil = 0., rpmRef = 0., rpmErr
=0.;
float spdPterm = 0., spdPiterm = 0., spdIterm = 0., spdPiUlim = 3000., spdPiOut =
0.;
float kpSpd = 0., kiSpd = 0.;
float iScale=0.0f, VdcScale=0.0f, FadcScale=0.0f, FvdcHwScale=0.0f, FiHwScale=0.0f,
SpeedScale=0.0f, EncoderScale=0.0f, MotorEncPulse=4.0f;
unsigned int offsetTimer=0, OFF_SW=1, ErrCode=0;
char Err1 = 0, Err2 = 0;
float OverCurrent = 0.0f, OverSpeed = 0.0f;
float VdcFdbLpfFct=0.0f, IdcOffsetLpfFct=0.0f, rpmFilLpfFct=0.0f;

int encodercnt=0;
int speedconcnt=0;
int temp100ms = 0;
```

```
float IadcDATA = 0;
float Motor_R = 0.0f;
float Motor_L = 0.0f;
unsigned int Wcc = 0;
float absIdcFdb=0.0;
float VdcCmd = 0.0f;

/********************* Funtion **********************/
void init_Variable(void);
void init_PIN(void);
void init_ADC(void);
void init_FlexPWM0_sub1(void);
void PWMISR(void);
void ADC_Read(void);
void Errdetect(void);
void H_BridgeRun(void);
void adc_offset_cal(void);

void init_PIT(void);
void init_eTimer0(void);
void ISR_PIT_100ms(void);
void EncoderFdb(void);
void SpeedCon(void);

int main(void)
{
    initModesAndClock();
    disableWatchdog();
    enableIrq();
    initOutputClock();
    FMSTR_Init();
    init_INTC();
    init_Linflex0();
    init_PIN();
    init_ADC();
    init_FlexPWM0_sub1();
    init_Variable();
    init_PIT();
    init_eTimer0();

    INTC_InstallINTCInterruptHandler(PWMISR,183,6);
```

```
    INTC_InstallINTCInterruptHandler(ISR_PIT_100ms,59,5);
    /* Loop forever */
    for (;;)
    {
        FMSTR_Recorder();
        FMSTR_Poll();
        i++;
    }
}

void init_Variable(void)
{
    SIU.GPDO[40].B.PDO = 1; // RESET에 1 (active low임)
    SIU.GPDO[61].B.PDO = 1; // SR
    SIU.GPDO[58].B.PDO = 1; //Gate High 1을 이 핀에 내려보냄
    SIU.GPDO[59].B.PDO = 1; //Gate Low

    PWM_Peak = 1600;
    PWM_Scale = PWM_Peak/12;
    FILFCT(100. , VdcFdbLpfFct , 10000.);
    FILFCT(10. , IdcOffsetLpfFct , 100.);

    kpSpd = 0.00001;    //먼저 전류제어를 통해 속도 검출이 제대로 되는지 확인하고 실시
    kiSpd = 0.00001;
    spdPiUlim=5.0;         //전류 제한

    EncoderScale = (float)(6.28316/4.0);
    FILFCT(2.,rpmFilLpfFct, 10.);
}

//-----------------------------------------
//      MPC5604P Device Configuration
//-----------------------------------------
void init_PIN(void)
{
    SIU.PCR[62].R = 0x0600; // FlexPWM_0 B[1]->PHASE
    SIU.PCR[58].R = 0x0300;    // GateIC PWMH
    SIU.PCR[59].R = 0x0300;    // GateIC PWML
    SIU.PCR[61].R = 0x0300; // GateIC SR
    SIU.PCR[40].R = 0x0300; // GateIC Reset
    SIU.PCR[29].R = 0x0100; // GateIC Err2
```

```
    SIU.PCR[30].R = 0x0100; // GateOC Err1
    SIU.PCR[23].R = 0x2000; // ADC0 AN[0] VBAT
    SIU.PCR[34].R = 0x2000; // ADC0 AN[3] IDC

    SIU.PCR[0].R = 0x0500;
    SIU.PCR[1].R = 0x0500;
}

void init_ADC(void) // 0번 모듈 , 0번 채널 , 3번 채널
{
    ADC_0.MCR.B.ABORT = 1; //진행중인 변환 중단
    ADC_0.MCR.B.PWDN = 0; //POWER DOWN MODE OFF
    ADC_0.CTR[0].R = 0x00008208;
    ADC_0.NCMR[0].R = 0x00000009; //0번채널, 3번채널 ENABLE
    ADC_0.CDR[1].R = 0x00000000; //데이터 받아오는 레지스터 초기화
    ADC_0.CDR[3].R = 0x00000000;  //데이터 받아오는 레지스터 초기화
}

void init_FlexPWM0_sub1(void)
{
    FLEXPWM_0.OUTEN.B.PWMB_EN = 0b0010;
    FLEXPWM_0.MCTRL.B.LDOK = 0b0010;
    FLEXPWM_0.MCTRL.B.RUN = 0b0010;

    // complementary PWM pair 안함. 독립적으로 씀
    FLEXPWM_0.SUB[1].CTRL2.B.INDEP = 1;

    FLEXPWM_0.SUB[1].INIT.R = -3200;
    FLEXPWM_0.SUB[1].VAL[0].R = 0;
    FLEXPWM_0.SUB[1].VAL[1].R = 3200;
    //compare interrupt val0, val1, val2, val3 enable CMPIE REG
    FLEXPWM_0.SUB[1].INTEN.R = 0x0001;

    FLEXPWM_0.SUB[1].DISMAP.B.DISB = 0;       // PWM B Fault Mask : Turn on

    FLEXPWM_0.MCTRL.B.LDOK = 0b0010; //1번모듈 ldok
    FLEXPWM_0.OUTEN.B.PWMB_EN = 0b0010;
}

void PWMISR(void)
{
```

```
    j++;
    ADC_Read();
    H_BridgeRun();

    //PWM 1번 모듈에서 발생한 CMPI FLAG 전부 OFF
    FLEXPWM_0.SUB[1].STS.R = 0x0001;
}

void init_PIT(void)
{
    PIT.PITMCR.R = 0;

    PIT.CH[0].LDVAL.R=6400000; //100ms
    PIT.CH[0].TCTRL.B.TIE=0x1;
    PIT.CH[0].TCTRL.B.TEN=0x1;

}

void init_eTimer0(void)
{
    ETIMER_0.ENBL.R=0x0000;
    ETIMER_0.CHANNEL[0].CNTR.R=0x0000;
    ETIMER_0.CHANNEL[0].CTRL.B.CNTMODE=0b100;
    ETIMER_0.CHANNEL[0].CTRL.B.PRISRC=0b0000;
    ETIMER_0.CHANNEL[0].CTRL.B.SECSRC=0b0001;
    ETIMER_0.ENBL.R=0x0001;
}

void ISR_PIT_100ms(void)
{
    temp100ms++;
    EncoderFdb();
    SpeedCon();
    PIT.CH[0].TFLG.B.TIF = 1;
}

void EncoderFdb(void)
{
    encodercnt++;
    mold=mnew;
```

```
    mnew=ETIMER_0.CHANNEL[0].CNTR.R;
    delta_m = ((mold-mnew)<<16)>>16;
    wr=(float)(delta_m)*(EncoderScale/0.1);

    LPF(wrFil,wr,rpmFilLpfFct);

    rpmFil = (float)(wrFil*9.54929);
}

void SpeedCon(void)
{
    speedconcnt++;

    if((SpdCon==1)&&(DSPRUN==1))
    {
        rpmErr = rpmRef - rpmFil;
        spdPterm = kpSpd * rpmErr;
        spdPiterm = spdPterm + kiSpd * rpmErr + spdIterm;
        spdPiOut = bound(spdIterm, spdPiUlim);
        spdIterm += kiSpd * rpmErr;
        spdIterm = bound(spdIterm, spdPiUlim);
        IdcRef = spdPiOut;
    }
    else
    {
        rpmRef = 0.;
        rpmErr= 0.;
        spdPterm = 0.;
        spdIterm = 0.;
        spdPiterm = 0.;
        spdPiOut = 0.;
    }
}

void ADC_Read(void)// 0번 모듈 사용, 3번채널은 전류 , 0번채널은 전압
{
    ADC_0.MCR.B.NSTART = 1; // Module 0 Conversion Start
    while(ADC_0.MCR.B.NSTART) asm("nop");
    if(ADC_0.CDR[0].B.VALID == 1)
    {
```

```
        IadcDATA = (float)ADC_0.CDR[3].B.CDATA;
        IdcPre = -((float) ADC_0.CDR[3].B.CDATA - 512.0f) * (5.0f/1023.0f) * 10.0f;
        //VdcPre = (float) ADC_0.CDR[0].B.CDATA * VdcScale; //VBAT값 보정
    }
    IdcFdb = IdcPre - IdcOffset;
}

void Errdetect(void)
{
    Err1 = SIU.GPDI[30].B.PDI; //PB 14 값을 읽어오겠다
    Err2 = SIU.GPDI[29].B.PDI; //PB 13 값을 읽어오겠다
    if(Err1 && !Err2)   ErrCode = 100;        //Error : Over Temperature or Voltage
    else if(!Err1 && Err2)   ErrCode = 101;       //Error : Short Circuit detection
    else if(Err1 && Err2)   ErrCode = 111;       //Error : Under Voltage

    absIdcFdb = abs(IdcFdb);

    if((!OFF_SW) && ((abs(IdcFdb) > OverCurrent)))
    {
        ErrCode = 1;                        //Error : Over Current
    }
    if(!OFF_SW && ((rpmFil > OverSpeed) || (rpmFil < -OverSpeed)))
    {
        ErrCode = 10;                        //Error : Over Speed
    }
    //숫자를 논리값으로 볼 때는 0이아니면 무조건 참, 0이어야 false
    if(ErrCode) DSPRUN = 0;
}

void H_BridgeRun(void)
{
    if(DSPRUN)
    {
        IdcErr = IdcRef - IdcFdb;
        VdcPterm = Kpc * IdcErr;
        VdcPiterm = VdcPterm + Kic *IdcErr + VdcIterm;
        VdcRef = bound(VdcPiterm, vRefUlim);
        VdcIterm += Kic * IdcErr;
        VdcIterm = bound(VdcIterm, vRefUlim);
```

```
        PWM_B = (int16_t)(VdcRef*PWM_Scale) + PWM_Peak;

        FLEXPWM_0.SUB[1].VAL[4].R = (unsigned short)-PWM_B;
        FLEXPWM_0.SUB[1].VAL[5].R = (unsigned short) PWM_B;

        FLEXPWM_0.MCTRL.B.LDOK |= 0x02; //1번모듈 ldok
        FLEXPWM_0.OUTEN.B.PWMB_EN = 0b0010;

        SIU.GPDO[40].B.PDO = 1;   //PWM reset
        SIU.GPDO[58].B.PDO = 1; //EPWM1A(PWMH)
        SIU.GPDO[59].B.PDO = 1; //EPWM1B(PWML)
        SIU.GPDO[61].B.PDO = 1;   //PHASE

    }
    else
    {
        SIU.GPDO[40].B.PDO = 0;   //PWM reset
        SIU.GPDO[58].B.PDO = 0; //EPWM1A(PWMH)
        SIU.GPDO[59].B.PDO = 0; //EPWM1B(PWML)
        SIU.GPDO[61].B.PDO = 0;   //PHASE

        FLEXPWM_0.OUTEN.B.PWMA_EN = 0x0;
    }
}

void adc_offset_cal(void)
{
    offsetTimer++;
    if(offsetTimer > 10000)
    {
        OFF_SW = 0;
        offsetTimer = 10001;
    }
    else
    {
        DSPRUN = 0;
        OFF_SW = 1;
        ErrCode=0;
        LPF(IdcOffset, IdcPre, IdcOffsetLpfFct);
    }
}
```

❶ 비례(Proportional, P) 제어기

비례 제어기는 현재 오차 $e(t)\,(=r(t)-y(t))$에 비례한 제어 입력 값 $u(t)$를 출력한다. 따라서 오차가 크면 제어 입력 값이 커지고, 작으면 제어 입력 값이 작게 된다. 그 정도는 비례 이득(Gain) K_p에 의해 다음과 같이 결정된다.

$$u(t) = K_p\, e(t) \tag{2-19}$$

❷ 적분(Integral, I) 제어기

적분 제어기는 식(2—20)과 같이 오차 $e(t)$를 적분하여 제어 입력 값 $u(t)$를 결정한다. 이것은 현재의 제어 입력 값을 결정하는 데에 현재 오차뿐 만아니라 과거의 오차들까지도 반영된다는 것을 의미한다. 적분 제어기에서는 오차가 영이 될 때까지 제어 입력 값이 계속 변동되고 오차가 영이 되면 그 값이 유지된다.

$$u(t) = K_i \int e(t)\,dt = \frac{K_p}{T_c} \int e(t)\,dt \tag{2-20}$$

❸ 비례 적분(Proportional-Integral, PI) 제어기

비례 제어기와 적분 제어기를 결합하여 빠른 응답 특성을 보이면서 정상상태 오차를 제거할 수 있도록 하는 비례적분 제어기의 구조가 그림 2.23에 보인다.

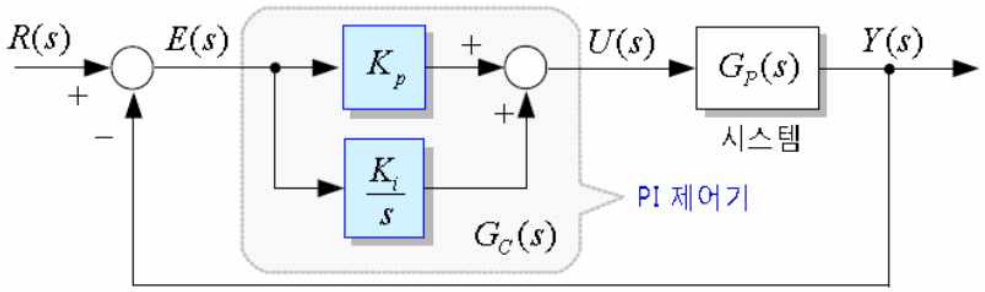
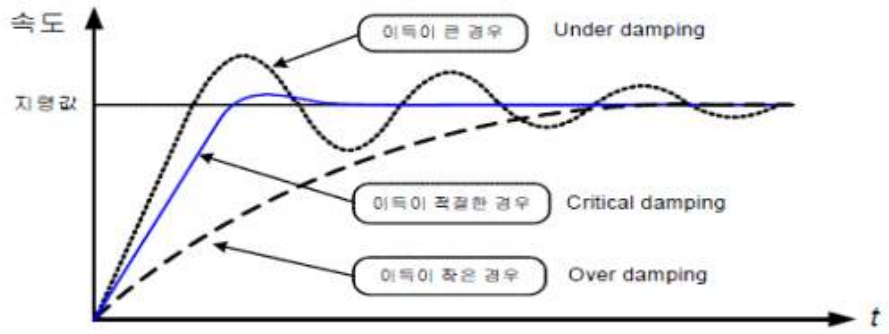


그림 2.23 비례적분 제어기

=> 비례 제어기와 적분 제어기를 함께 사용하여 빠른 응답의 특성과 정상상태의 오차를 제거함으로써 효율적인 제어기를 구성할 수 있습니다.



=> 비례적분의 이득이 큰 경우 지령값까지 도달은 빠르게 할 수 있으나, 오버슛이 발생하며 안정화까지 오랜 시간이 걸리게 됩니다. 반면에 비례적분의 이득이 작은 경우 지령값까지 도달하는데 걸리는 시간이 오랜 시간이 걸리게 되며, 오버슛이 발생하지 않으며 안정적입니다. 따라서 적절한 이득을 찾는 것이 중요합니다.

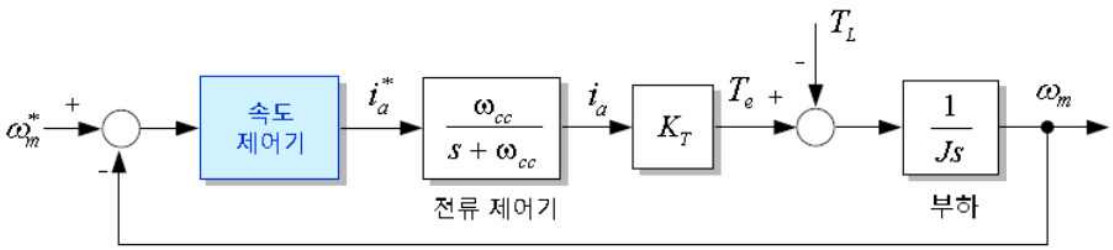


그림 2.38 속도 제어기를 갖는 구동시스템의 구조

=> 최종 속도 제어기를 갖는 구동시스템은 다음과 같이 구성됩니다.

=> 관성이 있을 시 속도제어기 이득을 튜닝한다. 출력은 토크로 발생하며, 전류 제어가 이상적이면 바로 시스템에 입력되게 되지만, 일반적으로 이상적이지 않기 때문에 전류제어 시스템을 한 번 더 거쳐서 토크를 발생하게 됩니다. 속도 제어 관점에서 전류제어기는 완벽합니다. 전류지령을 속도로 만듭니다.

- 변환 과정

No	Variable	dec	bin
1	mnew2	3	0b0000/0000/0000/0011
2	mnew2Old	250	0b0000/0000/1111/1010
3	mnew2-mnew2Old	-247	0b1111/1111/0000/1001
4	mnew2-mnew2Old << 8	2304	0b0000/1001/0000/0000
5	(mnew2-mnew2Old<<8)>>8)	9	0b0000/0000/0000/1001

※ 8bit 변수의 최대값 : 256
old값과 new값의 차이 = 256+(mnew2-mnew2Old)

- 음수 비트 변환 : 절대 값에서 2의 보수를 취함

247	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	1
보수	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	0
+1	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	1

=> 엔코더의 값은 비트의 제한이 있어 값이 넘어가면 음수의 값이 넘어가 버릴 수 있기 때문에 변수 표현형의 최대 값이 되면 다음 값은 0으로 초기화 해준다.

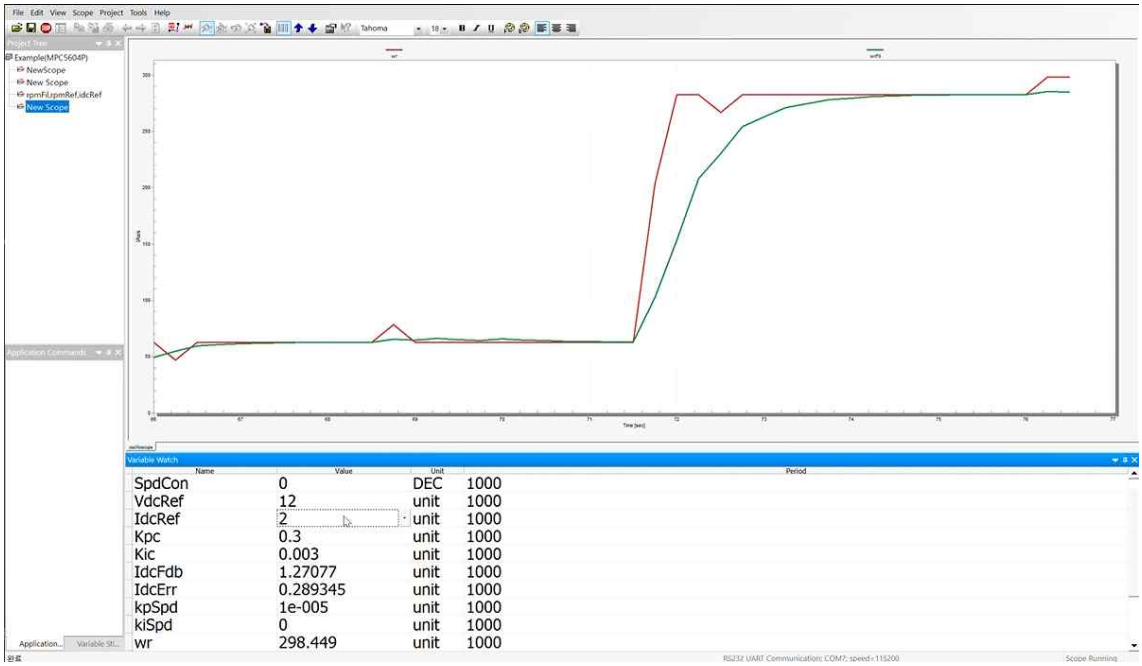
- LPF는 특정 주파수 이하의 신호를 전달하고, 그 이상의 주파수는 차단합니다. 이는 신호의 저역 통과를 의미합니다. LPF는 일정 주파수 이하의 신호를 통과시키기 위해 설계됩니다. 이를 위해 LPF는 일정 주파수 이상의 부분을 차단하거나 감쇠시키는 데 중점을 둡니다.
- ftc: LPF의 가장 중요한 매개변수 중 컷오프 주파수입니다. 이는 LPF에서 통과되는 주파수의 상한을 나타냅니다. ftc 이하의 주파수는 통과되고, ftc 이상의 주파수는 차단됩니다.
- 아날로그 LPF에서는 저항(Resistor)과 캐패시터(Capacitor) 또는 인덕터(Inductor) 등을 사용하여 신호의 주파수 특성을 결정하며, 디지털 LPF에서는 주로 디지털 신호 처리 알고리즘을 사용하여 주파수 특성을 제어합니다.
- 캐패시터는 주파수와 관계가 있습니다. 필터가 강해지면 응답이 느려지고 리플이 작아지고 파형을 안정적으로 만들어줍니다. 반대로 필터가 약해지면 응답이 빨라지지만 리플이 커지고 파형이 불안정해질 수 있습니다.

- $|V_{out}| = |V_{in}| \times \frac{1}{\sqrt{1 + w^2 R^2 C^2}}$

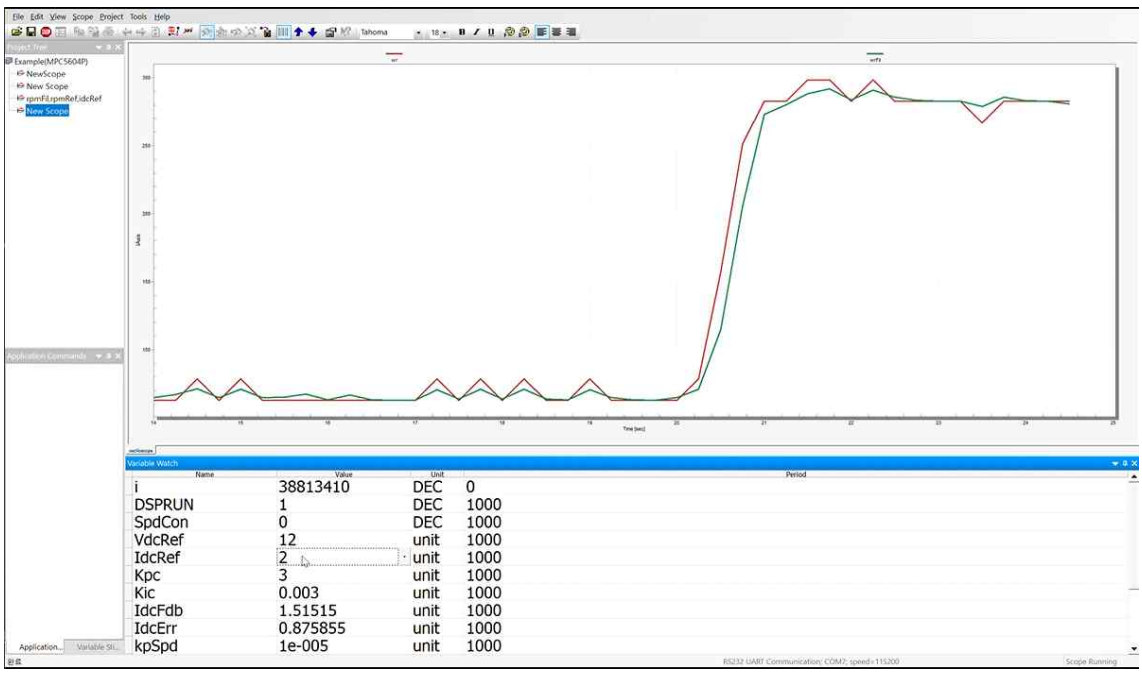
$$z_c = \frac{1}{Jwc} = \frac{1}{Sc}$$

$$r_0 = \frac{z_c}{z_R + z_c} = \frac{1}{JwR_c + 1} = \frac{1}{\alpha s + 1} v_{in}$$

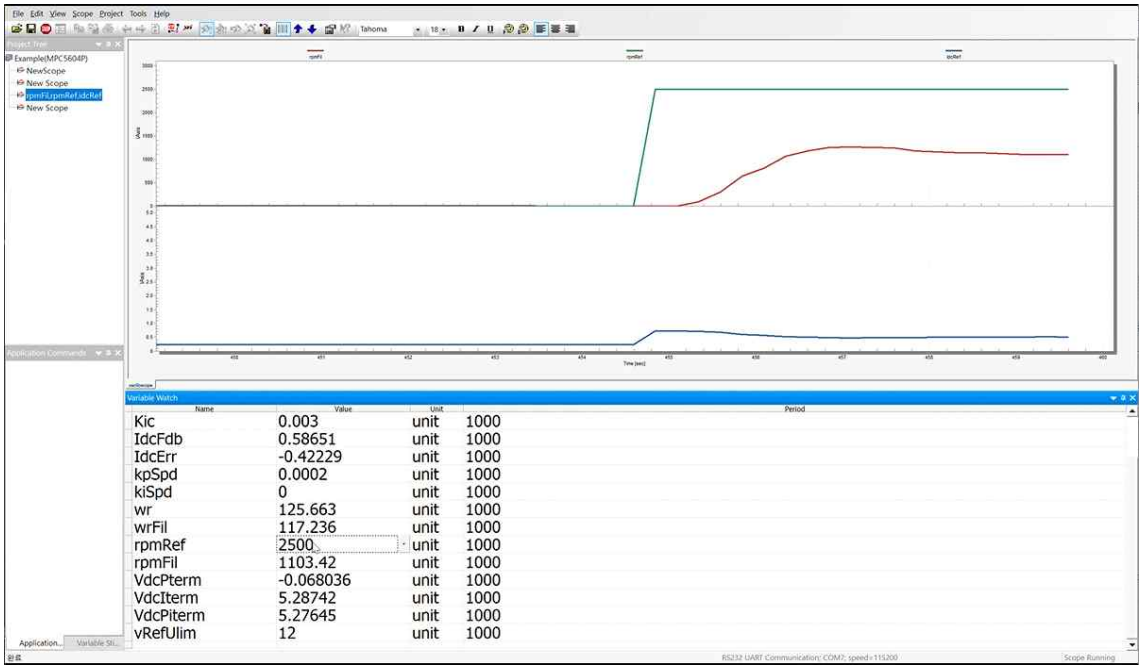
$$(z^{-1} = \frac{1}{\alpha s + 1})$$



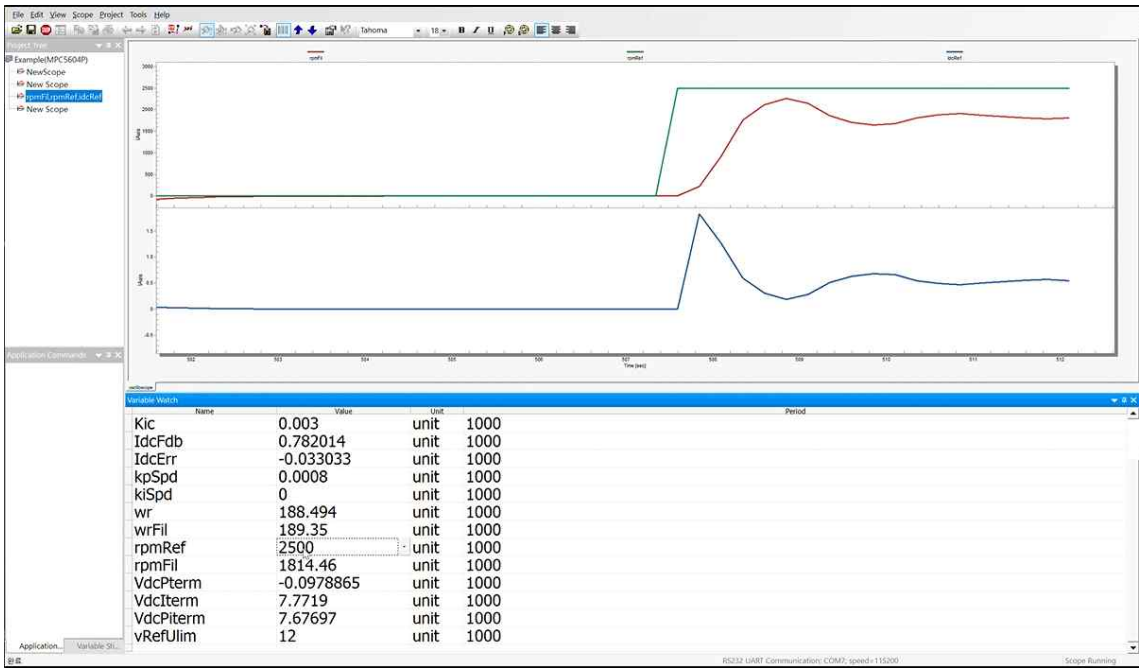
=> LPF를 약하게 했을 경우 응답은 느려지지만 리플이 작아지고 파형이 안정적인 모습



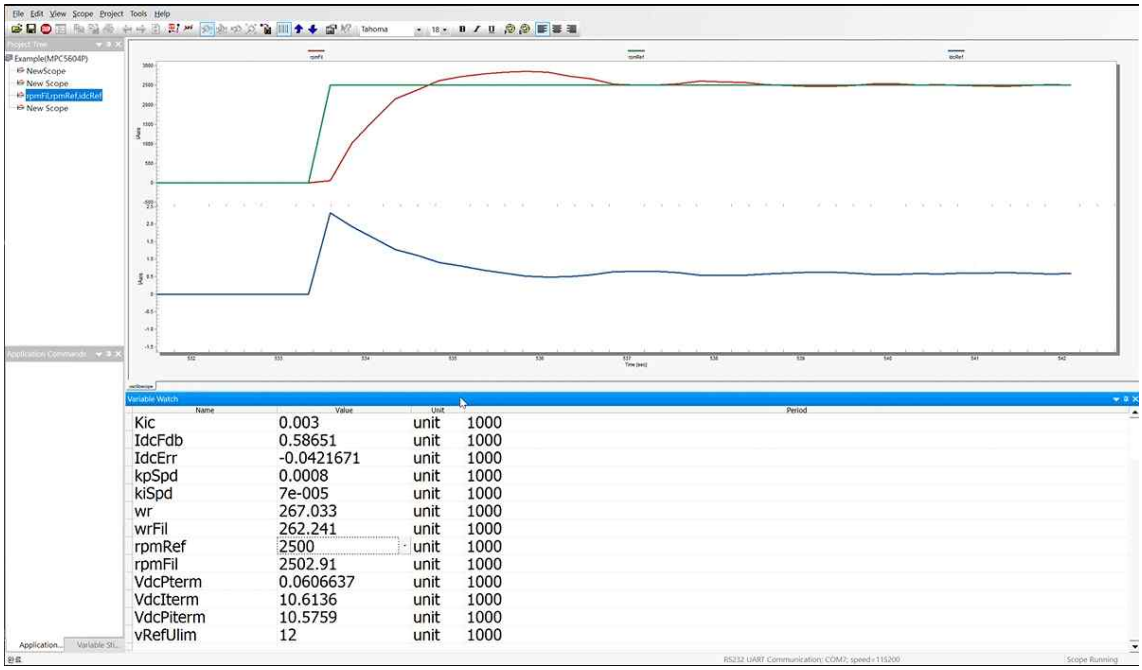
=> LPF를 강하게 했을 경우 응답은 빨라지지만 리플이 커지고 파형이 불안정적인 모습



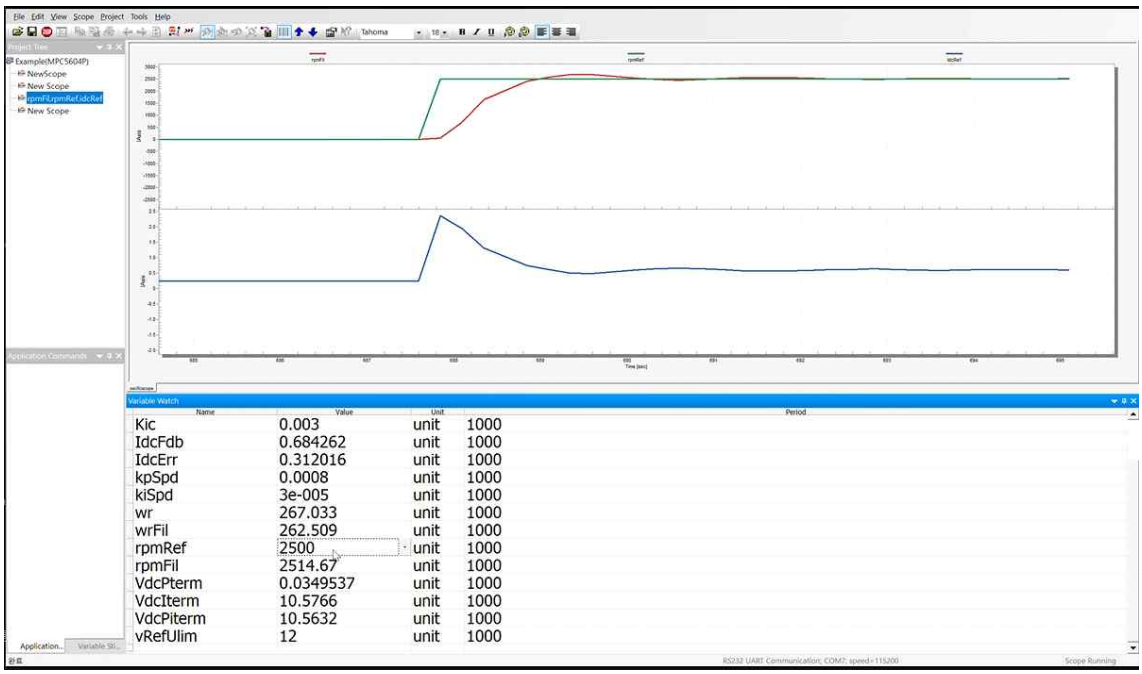
=> i gain을 0으로 주었을 경우 과거의 오차를 반영하지 않아 지령값에 최종적으로 도달하지 못함.



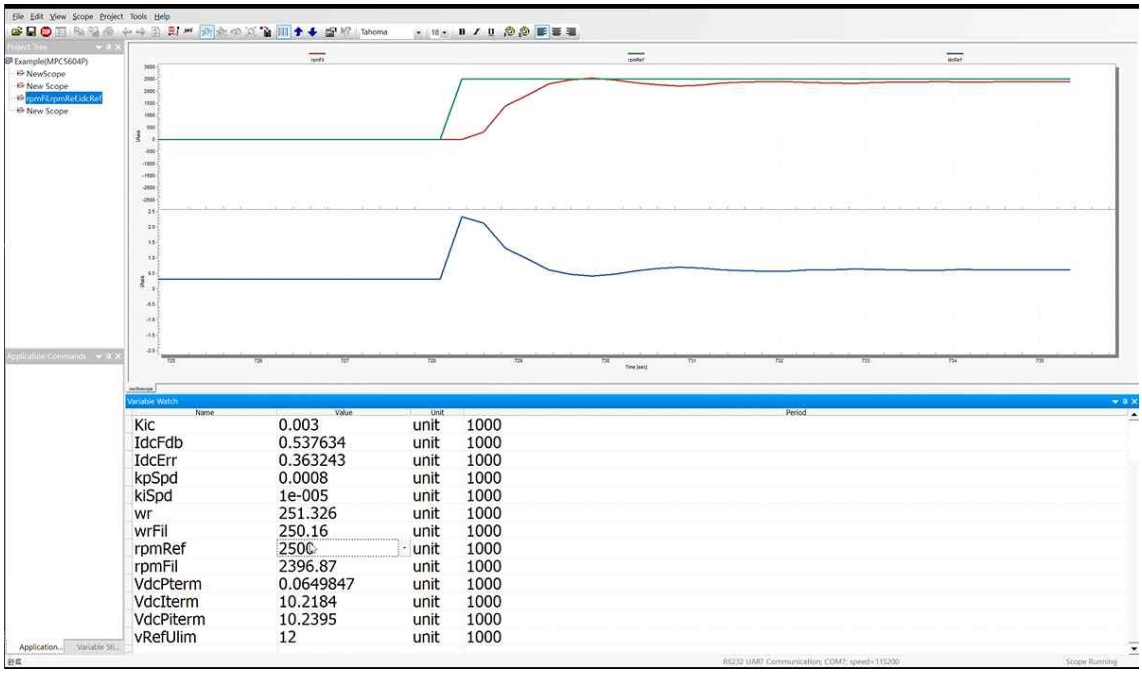
=> i gain을 0인 상태에서 p gain을 조절하여 반응성을 높임



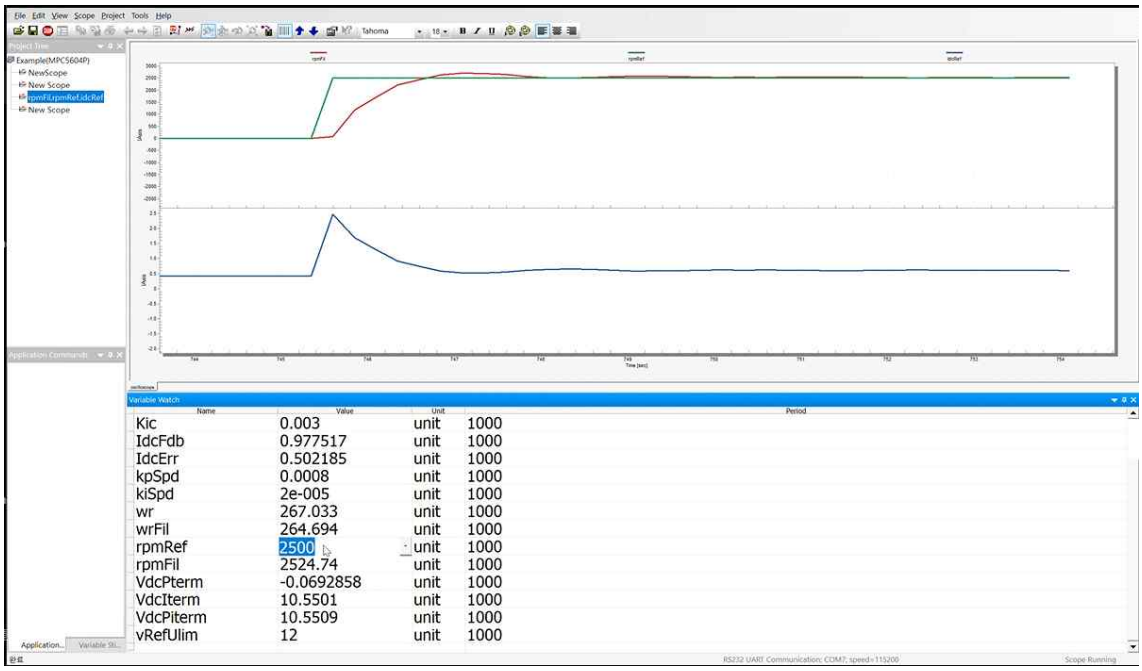
=> i gain을 미세 조정하여 지령값까지 도달하게 함. 약간의 진동성과 오버슛이 생겼으므로 i gain을 더 낮춰야 함.



=> i gain을 미세 조정하여 지령값까지 도달하게 함. 약간의 진동성과 오버슛이 위의 경우보다 많이 완화됨.



=> i gain을 미세 조정하여 지령값까지 도달하게 함. 너무 낮추게 되어 지령값까지 도달하지 못하는 결과를 확인함.



=> i gain을 미세 조정하여 지령값까지 도달하게 함. 적절한 값을 찾아 지령값깍지 비교적 빠르게 수렴하며 오버슛이 최소화 되었으며, 진동이 크게 일어나지 않고 안정적인 모습을 확인할 수 있음.