< Feedback Control Test >

앞 글에서 나름대로 혼자 생각해본 제어기를 시험해 보기 위해, 이제까지 만들어 놓았던 세 개의 프로그램 모듈들을 합쳤습니다.

1. PWM 파 뱉아내는 것
2. Timer 0을 사용해서 엔코더 읽어들이는 것
3. 간단한 PI 제어기

결과는...?

음. 역시 감감 무소식이군요. 아무 반응도 없구... 이유가 멀까 궁금해서, 하나씩 기능들을 제거해 보면서 노가다성 디버깅을 했습니당...

근데 이상한 게 발견되는군요... 저는 앞서 계획 했듯이, 한 개의 타이머로 두 가지의 타이밍을 잡기 위해 카운터 변수 i를 도입하였지요.

```
unsigned char i=0;
```

이라고 선언해준 담에,

```
SIGNAL(SIG_OVERFLOW0)
{
   어쩌구 저쩌구;
```

```
        i++;
        outp(RESET_TCNT0,TCNT0);        // 타이머 카운터 리셋
    }
```

이라고 써서 카운터를 인터럽트 걸릴 때마다 증가시켜 주구.... 메인 함수에서는

```
    void main(void)
    {
        for(;;)
        {
            if (i>5)
            {
                저쩌구 어쩌구;
                i=0;
            }
        }
    }
```

해 주었습니당.... 카운터 변수 i가 6까지 증가될 때마다 메인 함수의 루프가 한 바퀴 도는 간단한 거죠.

이게 이상없이 컴파일 되고 다운로드 잘되는데... 막상 실행시키면 먹통이 되는 겁니당... ㅜ_ㅜ;

예전에 터보씨에서 즐겨 써먹던 건데, 이상하게 AVR-GCC 에선 안먹더라구요... 요 컴파일러의 버그일 수도 있고, 아님 최적화 (Opimization) 옵션의 문제일 수도 있을거라는 생각이 듭니다. 물론 비슷한 다른 구문을 여러 가지 넣어서 시험해 보았는데, 카운터 변수 i를 도입하는 순간 먹통.... 물론 i를 빼고 그냥 무한루프 돌리면 잘 돌아갑니다. 암튼, 하다하다 안되어 이 구문을 사용하는 건 포기하고... 단순 딜레이를 주어 근사적인 샘플링 타임을 가지는 제어기를 구성할 수밖에 없었습니다. 혹시 누구 이런 증상에 대해 아시는 분 계시면 가르침을 주시길...

각설하고...

AVR-GCC 는, 아무 일도 안하는 for() 문을 한 바퀴 돌릴 때마다 5클럭을 소비한다고 합니다. 시스템 클럭이 4MHz 니까, 1250회 공회전 시키면 약 1/1000 초를 소모하는 것으로 생각됩니당. 이걸 써서 딜레이 함수를 만들고... 그냥 메인함수의 무한루프 꼬리에 달아붙였습니당... 뭐 정확한 반복시간은 못내겠지만, 어차피 정확하게 할 것도 아닌데 할 수 없죠 뭐(슬슬 걱정되기 시작함).

그래서 만들어낸 코드가 아래와 같습니다.

```
////////////////////////////////////////////////////////////////////////////////

// P-I 제어기 테스트

// 2001.12.31

// 김동호

////////////////////////////////////////////////////////////////////////////////


#include < io.h >

#include < interrupt.h >

#include < sig-avr.h >
```

```
////////////////////////////////////////////////////////////////////////////
// 각종 상수 정의
////////////////////////////////////////////////////////////////////////////
// PWM 관련
#define PWM_INIT_SET_A      0xA1    // PWM 2개 사용, 8비트 분해능
#define PWM_INIT_SET_B      0x01    // 프리스케일링 = 0분주
#define PWM_INIT_VAL_AH     0x00    // 16비트 PWM 카운터A의 상위 바이트 초기값
#define PWM_INIT_VAL_AL     0x00    // 16비트 PWM 카운터A의 하위 바이트 초기값
#define PWM_INIT_VAL_BH     0x00    // 16비트 PWM 카운터B의 상위 바이트 초기값
#define PWM_INIT_VAL_BL     0x00    // 16비트 PWM 카운터B의 하위 바이트 초기값
#define PWM_L_DIR_UP        0x04    // 좌측 모터 정방향
#define PWM_L_DIR_DN        0x08    // 좌측 모터 역방향
#define PWM_R_DIR_UP        0x02    // 우측 모터 정방향
#define PWM_R_DIR_DN        0x01    // 우측 모터 역방향
// 엔코더 관련
#define TIMER_TIMSK         0x02    // 타이머 0 오버플로우 인터럽트 인가
#define RESET_TCNT0         0x00    // 카운터 초기값 = 0
#define PRESCALE_TCCR0      0x02    // 프리스케일링 = 8분주
// P-I 제어기 관련
```

```
#define P                   100      // 비례 상수
#define I                   10       // 적분 상수
#define INTEGRAL_LIMIT      30       // 적분 리밋
#define U_LIMIT             127      // 출력값 리밋
#define E2V                 1        // 엔코더 값을 속도값으로 변환하는 계수
// 시스템 관련
#define GOOD                0        // 에러코드
#define SAMPLING_TIME       100      // 제어기의 샘플링 타임(밀리세컨드)


////////////////////////////////////////////////////////////////////////////
// 각종 변수 선언
////////////////////////////////////////////////////////////////////////////
// PWM 관련
unsigned char pwm_l=0;          // 좌측모터 PWM 출력값
unsigned char pwm_r=0;          // 우측모터 PWM 출력값
unsigned char pwm_dir=0;        // 좌우측 모터 방향값
// 엔코더 관련
unsigned char e_current_l=0, e_current_r=0;  // 최근에 읽은 값
unsigned char e_past_l=0, e_past_r=0;        // 지난번 읽은 값
```

```
char e_flag_l=0, e_flag_r=0;                // 증감 판별 값
char e_l=0, e_r=0;                          // 적분 값
// P-I 제어기 관련
char v_l=0,v_r=0;                // 양짝 바퀴의 실제속도
char v_cl=0,v_cr=0;              // 양짝 바퀴의 원하는 속도(목표속도)
char dv_l=0,dv_r=0;              // 양짝 바퀴의 실제 속도차이

char integral=0,integral_past=0; // 금번 적분 값과 과거 적분값
char u_l=0,u_r=0;                // 양짝 바퀴에 줄 전압출력(PWM 듀티비)
char u_l_past=0,u_r_past=0;
// 시스템 관련
unsigned char sensor=0;          // Port A 로 읽는 센서값

////////////////////////////////////////////////////////////////////////////
// PWM 관련 함수
////////////////////////////////////////////////////////////////////////////
// 쌍발 PWM 초기화
int pwm_init(void)
{
```

```
    // 분해능 8비트짜리 PWM 2개 준비하기
    outp(PWM_INIT_SET_A,TCCR1A);
    // A 핀을 위한 초기값 적어주기
    outp(PWM_INIT_VAL_AH,OCR1AH);    outp(PWM_INIT_VAL_AL,OCR1AL);
    // B 핀을 위한 초기값 적어주기
    outp(PWM_INIT_VAL_BH,OCR1BH);    outp(PWM_INIT_VAL_BL,OCR1BL);
    // 프리스케일링
    outp(PWM_INIT_SET_B,TCCR1B);
    return GOOD;
}


// P-I 제어기가 결정한 각모터의 방향 판별, 방향변수에 대입
int pwm_direction(void)
{
    if (u_l < 0) {pwm_dir = PWM_L_DIR_UP;}
    else if (u_l > 0) {pwm_dir = PWM_L_DIR_DN;}
    if (u_r < 0) {pwm_dir |= PWM_R_DIR_UP;}
    else if (u_r > 0) {pwm_dir |= PWM_R_DIR_DN;}
    return GOOD;
```

```
}


// P-I 제어기가 결정한 출력값을 PWM값으로 변환
int voltage2pwm(void)
{
    pwm_l = (unsigned char)(u_l>>1);
    pwm_r = (unsigned char)(u_r>>1);
    return GOOD;
}


// 쌍발 PWM 출력하기
int pwm_out(void)
{
    outp(pwm_l,OCR1AL);
    outp(pwm_r,OCR1BL);
    outp(pwm_dir,PORTC);
    return GOOD;
}
```

```
////////////////////////////////////////////////////////////////////////////
// 엔코더 관련 함수
////////////////////////////////////////////////////////////////////////////
// 타이머 초기화
int timer_init(void)
{
    timer_enable_int(TIMER_TIMSK);  // TIMSK 설정
    outp(RESET_TCNT0,TCNT0);        // 타이머/카운터 0을 일단 리셋
    outp(PRESCALE_TCCR0,TCCR0);     // 프리스케일링함... 64분주
    return GOOD;
}

// 엔코더 읽기
int e_read(void)
{
    e_current_l = 0x03 & sensor;        // 좌측 엔코더 값만 분리
    e_current_r = (0x0C & sensor) >> 2; // 우측 엔코더 값만 분리
    e_current_l ^= e_current_l>>1;      // 좌측 엔코더 코드 변환
    e_current_r ^= e_current_r>>1;      // 우측 엔코더 코드 변환
```

```
    e_flag_l = (char)(e_current_l - e_past_l);  // 좌측 엔코더 증감여부 판별
    e_flag_r = (char)(e_current_r - e_past_r);  // 우측 엔코더 증감여부 판별
    if (e_flag_l == 3) {e_flag_l = -1;} // 좌측 엔코더 예외 처리
    else if (e_flag_l == -3) {e_flag_l = 1;}
    if (e_flag_r == 3) {e_flag_r = -1;} // 우측 엔코더 예외 처리
    else if (e_flag_r == -3) {e_flag_r = 1;}
    e_l += e_flag_l;                    // 좌측 엔코더값 적분
    e_r += e_flag_r;                    // 우측 엔코더값 적분
    e_past_l = e_current_l;             // 좌측 엔코더 지난번 읽은값으로 저장
    e_past_r = e_current_r;             // 우측 엔코더 지난번 읽은값으로 저장
    return GOOD;
}

// 엔코더 읽은 값 초기화
int e_reset(void)
{
    e_l=0;
    e_r=0;
    return GOOD;
```

```
}


////////////////////////////////////////////////////////////////////////////
// P-I 제어기 관련 함수
////////////////////////////////////////////////////////////////////////////
// 엔코더 값을 속도로 변환
int encoder2velocity(void)
{
    v_l = E2V * e_l;
    v_r = E2V * e_r;
    return GOOD;
}


// I 제어기에서 튀어나온 값의 리밋 정해주기
int integral_limit(void)
{
    if (u_I>=INTEGRAL_LIMIT) {u_I=INTEGRAL_LIMIT;}
    else if (u_I<=(-1*INTEGRAL_LIMIT)) {u_I=(-1*INTEGRAL_LIMIT);}
    return GOOD;
```

```
}


// P 제어기가 결정한 출력값의 리밋 정해주기
int u_limit(void)
{
    if (u_l>=U_LIMIT) {u_l = U_LIMIT;}
    else if (u_l<=(-1*U_LIMIT)) {u_l=(-1*U_LIMIT);}
    if (u_r>=U_LIMIT) {u_r = U_LIMIT;}
    else if (u_r<=(-1*U_LIMIT)) {u_r=(-1*U_LIMIT);}
    return GOOD;
}


// P-I 제어기
int pi_controller(void)
{
    integral = integral_past + I*((v_cl-v_cr)-(v_l-v_r)); // 적분 제어기
    integral_limit();
    u_l = - (u_l_past + P*(v_cl-v_l) - integral);  // 좌측 바퀴 비례 제어기
    u_r = - (u_r_past + P*(v_cr-v_r) + integral);  // 우측 바퀴 비례 제어기
```

```
    u_limit();
    integral_past = integral;     // 계속 적분해야 되니끼니 일단 저장
    u_l_past = u_l;
    u_r_past = u_r;
    return GOOD;
}

// 속도값 초기화
int v_reset(void)
{
    v_l = 0;
    v_r = 0;
    return GOOD;
}

////////////////////////////////////////////////////////////////////////////
// 상위 제어기 관련 함수
////////////////////////////////////////////////////////////////////////////
// 요구 속도 결정
```

```
int i_love_KJS_really(void)
{
    v_cl = 0;
    v_cr = 0;
    return GOOD;
}

////////////////////////////////////////////////////////////////////////////////
// 시스템 관련 함수
////////////////////////////////////////////////////////////////////////////////
// 각 입출력 포트 설정
int port_init(void)
{
    outp(0x00,DDRA);                // Port A 입력
    outp(0xff,DDRB);                // Port B 출력
    outp(0xff,DDRC);                // Port C 출력
    outp(0xff,DDRD);                // Port D 출력
    return GOOD;
}
```

```
// 시스템 초기화
int sys_init(void)
{
    port_init();
    pwm_init();
    timer_init();
    sei();
    return GOOD;
}

// 딜레이 함수
int delay_ms(unsigned char ms)
{
    unsigned int a=0,b=0;
    for (b=0;b<ms;b++)
    {
        for (a=0;a<1250;a++) {}     // 1 ms loop at 4MHz system
    }
```

```
    return GOOD;
}

// 인터럽트 서비스 루틴
SIGNAL(SIG_OVERFLOW0)
{
    sensor = inp(PINA);             // Port A 읽기
    e_read();
    outp(RESET_TCNT0,TCNT0);        // 타이머 카운터 리셋
}

////////////////////////////////////////////////////////////////////////////
// 메인 함수
////////////////////////////////////////////////////////////////////////////
int main(void)
{
    sys_init();                     // 먼저 시스템을 초기화하면...
                                   // 백그라운드로 엔코더를 읽습니당..
    for (;;)                        // 무한루프 돌아가기 시작...
```

```
    {
        i_love_KJS_really();        // 양쪽 바퀴의 속도를 결정해주면...
        encoder2velocity();         // 일단 읽어들인 엔코더 값을 속도로 바꾸고.
        pi_controller();            // PI 제어기로 출력값을 결정하면...
        pwm_direction();            // 방향 지정 비트를 셋팅해주고,
        voltage2pwm();              // PWM 값도 셋팅해 주고 나서...
        pwm_out();                  // 마침내 모터를 돌립니당.
        e_reset();                  // 마지막으로, 엔코더 값을 0으로...
        v_reset();                  // 속도값도 0으로 돌려놓구요...안전하게.
        delay_ms(SAMPLING_TIME);    // 샘플링타임을 맞춰주기 위해서 딜레이.
    }

    return GOOD;                    // 감사합니당.  새해 복많이...
}
```

음.... 첨부 파일로 할 걸 그랬나....-_-;

기본 설정은, PWM은 프리스케일링 안했습니다. 최고 주파수로... 그리고, 엔코더 읽기도 프리스케일링 8분주로 하여 가장 고속으로 되도록 했습니다. 제어기의 샘플링 타임은 1/10 초로 했구요. 몇 번의 시험을 해 보면서 맞추어 본 것입니다.

암튼 이건, 양쪽 바퀴를 정지(속도 0)으로 유지하는 제어기입니다. 손으로 강제로 돌리려고 하면, 포켓봇이 저항합니다. 제어기의 파라미터 게인(Gain) 값을 너무 크게 주면 발통의 회전이 과해서 오버슈트(Over-shoot)가 일어나더군요. 제자리에서 왔다갔다 덜덜 떠는 현상요....

암튼, P 값을 좀 줄여주니(5-20 정도) 거의 들어맞는군요... 손으로 억지로 돌리려고 하니까 바퀴에 토크가 발생해서 낑낑 댑니다. 한 쪽 바퀴를 그래도 억지로 돌렸다 놓으니, 반대쪽 바퀴와 속도를 맞추려고 같이 꿈틀꿈틀 합니다. 일단 제어기가 동작하는건 확인됩니다.

물론 문제점이 눈에 보입니다.

일단 엔코더의 Resolution 이 좀 낮은 편이라, 저속에서 거의 제어기의 효과를 발휘하지 못하구요. 게다가 종종 엔코더 읽기에 실패하는 수도 있군요(데이타 확인한 것은 아님). 시스템이 전반적으로 불안하다는 반증이겠죠.

게다가 구동부는 상당히 열악한 성능을 보여주고 있습니다.... 마찰 땜에 PWM 듀티비 0x10 정도에선 꿈쩍도 안하네요... 끼끼~ 하는 기분나쁜 고음만 내구요... 그리고 바퀴의 편심까지!

음.... 결론적으로, 수준높은(?) 깨끗한 선형 제어를 하기엔 역시 무리라는 생각이 듭니다. 부품 가격만 수백만원 수천만원 하던걸 가지고 편하게 실험하던때가 그립습니다.... 아... 아련한 그시절... 2000만원짜리 시그널 아날라이저가 남아돌고... 독일제 천만원짜리 리니어 스케일러 쌓여있고... 헐헐...

선형 제어기의 적용은 좀 더 두고 보아야 할 것 같습니다. 시스템 리소스만 잡아먹고 이정도 불안한 성능을 보인다면, 차라리 오픈루프로 그냥 전류 인가해 줘 버리는게 더 나을 듯 싶기도 하네요.

일단 요까지만 하고, 다른 것 시도 해 보아야겠습니당...

처음에 느긋하게 시작했는데, 자꾸 마음이 급해지고... 여기저기 자꾸 불러대는 사람은 많고... 백수라 한가하단 건가????