

스테핑 라인트레이서 개발하기

류대우(davidryu@NewTC.co.kr)

1. 센서 보드 – 적외선 센서

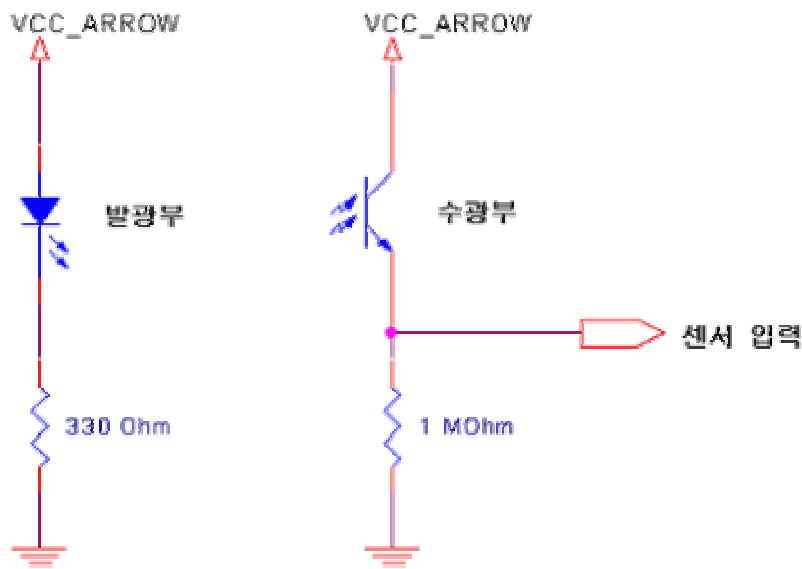
1. 적외선 센서 (Photo Sensor)

라인트레이서나 마이크로 마우스 혹은 다른 마이크로 로봇에서 센서로 사용하는 것 중 가장 많이 사용하는 것이 photo sensor입니다. 거리의 측정에도 사용되지만 원거리는 잘 사용하지 않고 근거리를 측정하고자 할 때 사용되기도 하며 물체의 유/무 등 많은 곳에서 사용되고 있습니다.

Photo sensor는 보통 적외선을 많이 이용하고 있습니다. 발광부와 수광부로 나눌 수 있습니다. 발광부는 일반 LED와 비슷하지만 가시광선을 내는 것이 아니고 적외선을 발산합니다. 수광부는 적외선이 들어온 양에 따라 아날로그 전압이 출력되게 됩니다. 이 아날로그 신호를 디지털 신호로 바꾸어 MCU에서 인식하여 사용합니다.

수광부는 빛의 양에 따라서 저항값이 변하는 가변저항으로 생각해도 됩니다.

2. Photo Sensor 회로도

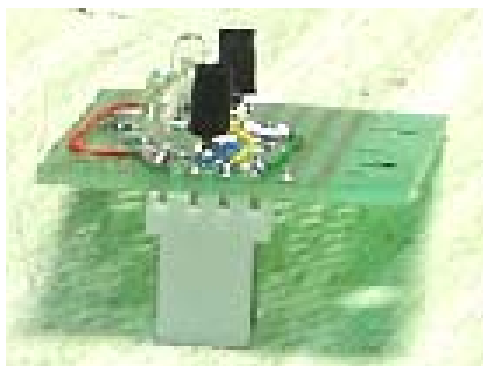
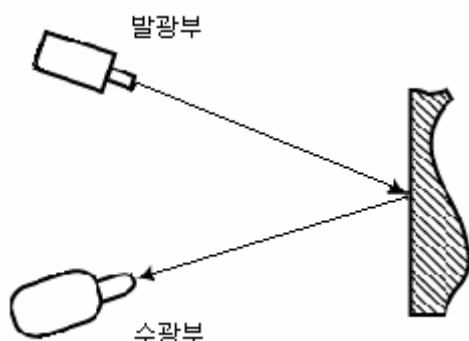


왼쪽 다이오드가 발광부 이고 오른쪽 트랜지스터처럼 생긴 것이 수광부입니다. 발광부는 일반 LED와 같이 330옴을 달아주면 적외선이 나옵니다.

수광부에는 1M~2M정도의 저항을 달아주는데 이보다 저항이 더 커져 버리면 값이 너무 커지고, 전류가 너무 적게 흐르기 때문에 적당히 달아야 합니다.

포토센서의 수광부는 일종의 TR입니다. BASE 입력 대신 적외선이 들어가면 스위치가 연결되어 Corrector와 Emitter 에 전류가 흐르고, 적외선이 없으면 스위치가 끊어져서 전류가 흐르지 않는다고 생각하시면 됩니다. 전류가 흐르면 “센서입력”이라고 명시한 Node에 전압이 걸리고 그 전압을 컨버팅 해서 적외선이 얼마나 들어왔는지 알 수 있습니다.

또는 연산 증폭기의 비교 기능을 이용하여 입력이 있음과 없음으로 바꾸어 마이크로 컨트롤러에 입력할 수 있습니다.



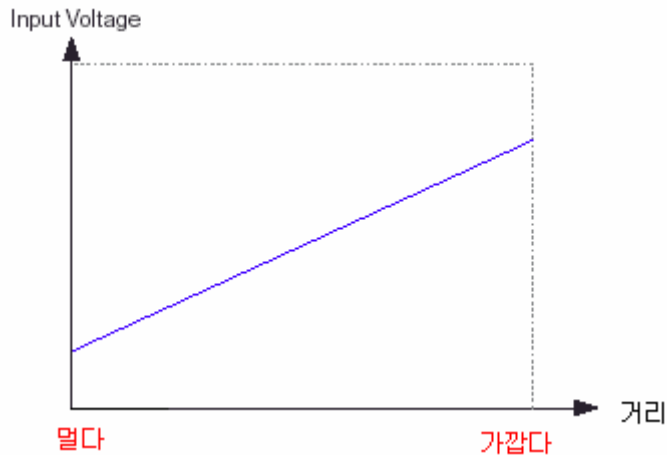
위 그림과 같이 센서의 방광부와 수광부 감지하기 위한 물체 방향을 바라보게 합니다. 정당한 거리를 기준으로 반사가 가장 많이 되도록 살짝 꺾어줍니다. 물체가 접근하면 수광부에 감지되는 적외선의 양이 많아져서 전압이 올라가고 멀어지면 감지되는 적외선의 양이 작아져 전압이 내려가게 됩니다.



위 그림과 같이 라인을 감지할 수 있도록 라인 간격을 고려하여 센서 2개의 위치를 정한다. 약 2Cm 정도 간격으로 한다.

3. Photo Sensor 신호의 입력

적외선 센서의 신호는 아래와 같이 아날로그 신호로 출력이 된다. 실제로 멀티 미터로 출력 V 를 측정하면 센서에 물체가 접근할 때 전압이 증가하는 것을 볼 수 있을 것이다. 회로 설계에 따라 반대로 물체가 접근할 때 전압이 감소 할 수도 있다.



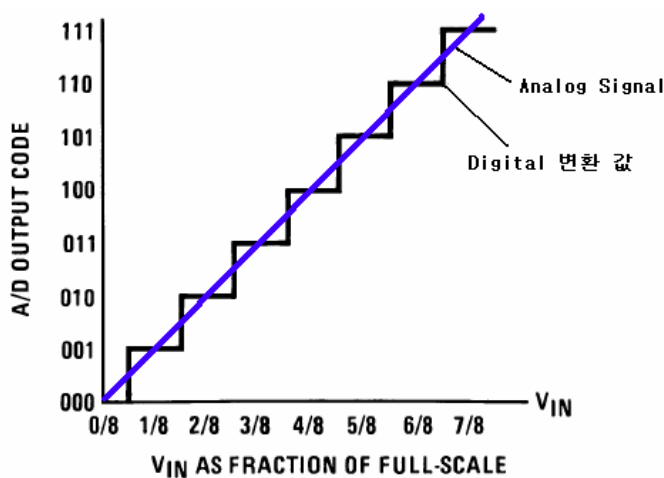
위와 같은 아날로그 신호를 MCU에서 인식하는 방법에는 OP-Amp 를 사용하거나 Analog-Digital Converter 를 사용하여 입력 받는 방법이 있다.

F/W 프로그래밍을 하기 전에 멀티미터(테스터) 또는 오실로스코프를 이용하여 센서 거리에 따라서 전압 값이 바뀌는지 확인 해본다.

4. H/W 구성 #2 (Analog-Digital Converter 를 사용하는 방법)

Analog to Digital Converter는 아날로그 신호를 디지털 신호로 바꾸어 주는 역할을 하는 장치이다. 이를 이용하여 아날로그 신호를 MCU에서 인식할 수 있는 디지털 신호로 바꿀 수 있다.

아래 그림은 아날로그 신호를 3bit ADC 를 이용하여 디지털로 변환하는 것이다.



MCU F/W 프로그래밍 (ATMega128 의 예)

1. AD 컨버터를 초기화 한다. (인터럽트 사용 여부, 공급되는 클럭, 데이터 포맷 등을 초기화한다.)

```
void adc_init(void)
{
    ADCSRA = 0x00;           //disable adc
    ADMUX = 0x00;            //select adc input 0
    ACSR = 0x80;
    ADCSRA = 0xC6;
}
```

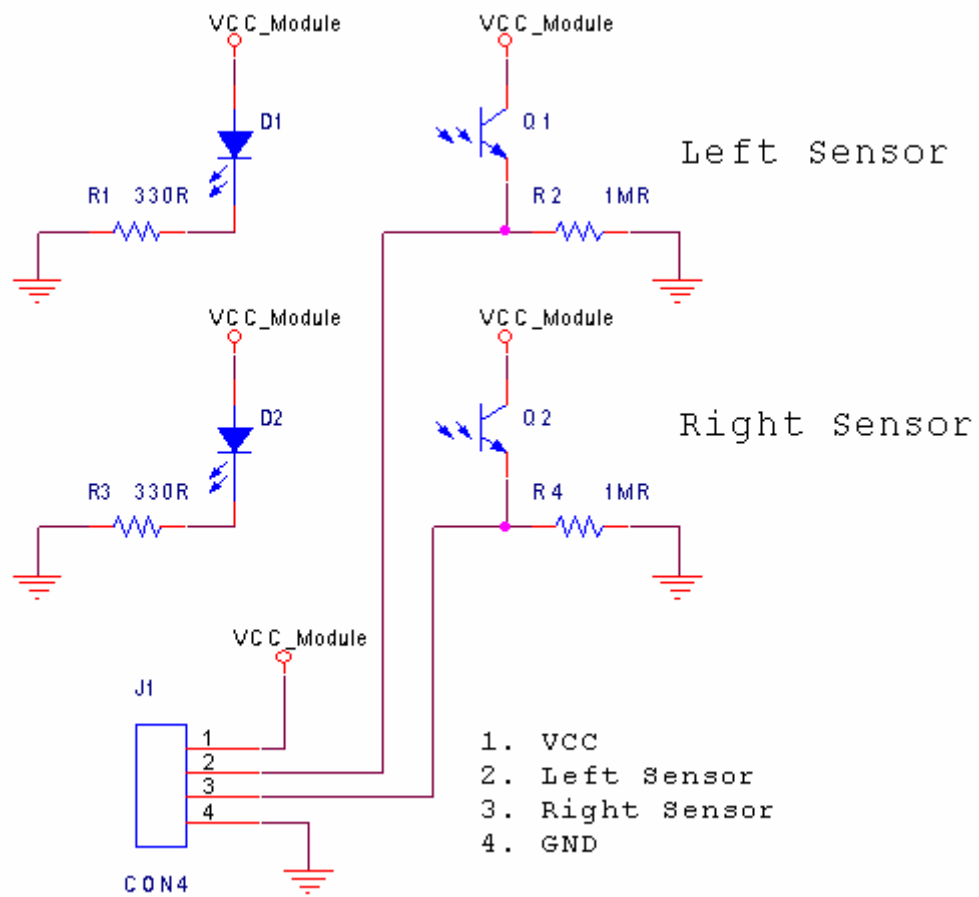
2. 원하는 채널의 컨버팅을 시작한다.

```
void startConversion(unsigned char ch)
{
    ADMUX = 0x20 | (ch & 0x0f);    // 컨버팅 채널 선택
    ADCSRA = 0xc7;                // 컨버팅 시작
}
```

3. 충분한 시간이 흐른 후 또는 컨버팅 완료된 것을 확인한 후에 컨버팅한 값을 읽는다.

```
unsigned char readConvertData(void)
{
    volatile unsigned char temp;
    while((ADCSRA & 0x10)==0);    // 컨버팅 완료까지 기다림.
    ADCSRA = ADCSRA | 0x10;
    temp = ADCH;
    return temp;
}
```

ADC 를 사용하는 센서부 회로도

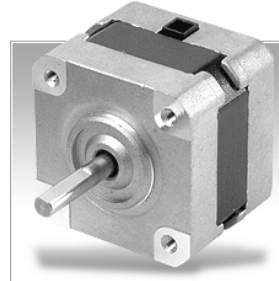


2. 스텝핑 모터 구동 (Stepping Motor Control)

① 스텝핑 모터란?

스텝핑 모터란 Step 에 의해 구동되는 모터이다. 4개의 모터상이 있어서 한 스텝 한 스텝 진행하면서 모터가 회전하는 것이다. 한 스텝 당 보통 1.8° 씩 회전한다.

스텝핑 모터는 AC servo, DC servo 모터에 비하여 값이 싸고 정확한 각도 제어에 유리하여 우리 주위에서 쉽게 접할 수 있다.



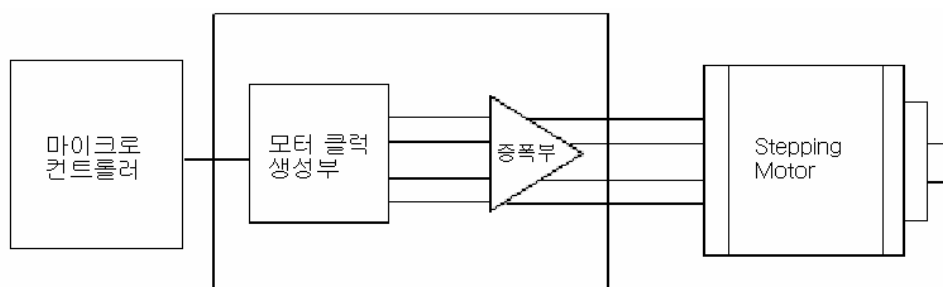
스텝핑 모터는 pulse에 의해 digital적으로 제어하는 것이 가능하므로 마이크로 컨트롤러에서 사용하기에 적합한 모터이다. 스텝핑 모터는 다른 모터와 달리 모터의 위치를 검출하기 위한 feedback없이, 정해진 각도를 회전하고, 상당히 높은 정확도로 정지할 수 있다. 그리고, system에 적용된 뒤에도 초기 성능이 만족되어있으면, 갱년 변화가 적고, 보수가 용이하며, 매우 신뢰성이 높은 system을 구성할 수 있다.

또 다른 모터에 비해 정지 시 매우 큰 유지 토크(정지 토크)가 있기 때문에 전자 브레이크 등의 유지 기구를 필요로 하지 않는다. 회전 속도에 있어서도 스텝핑 모터에 부여하는 pulse rate에 비례하므로 임의로 제어할 수 있다.

② 스텝핑 모터 구동하기

DC모터는 회로는 간단하지만 제어하는 방법이 복잡하다. 반면 스텝핑 모터는 구동회로는 상대적으로 복잡하지만 제어는 훨씬 쉽다.

스텝핑 모터를 구동하기 위해서는 L297과 SLA7024를 같이 사용한다. L297은 스텝핑 모터 구동에 필요한 상(클럭)을 만들어 주기 위한 것이고 SLA7024는 만들어진 상(클럭)의 전압과 전류를 증폭시키는 역할을 합니다.



스텝핑 모터의 구동부 구성

- ✓ 모터클럭 생성부 Stepping motor의 여자 상(Step)을 만든다.
- ✓ 증폭부 모터 입력 신호를 증폭하여 Stepping motor를 구동한다.

스테핑 모터의 동작

1. 마이크로 컨트롤러에서 모터 구동을 위한 클럭을 발생 시킨다.
2. 모터클럭 생성부에서 입력 클럭을 기준으로 스테핑 모터를 구동을 위한 Step 신호를 발생시킨다.
3. 증폭부에서는 모터클럭 생성부에서 출력된 신호를 모터를 구동할 수 있는 전압, 전류로 증폭하여 모터로 전달한다.

③ 구동방식의 종류 - 전류의 흐름에 따른 구분

1. Unipolar 구동

각각의 coil에 1개의 transistor가 접속되고 transistor를 ON하는 것으로서 각 coil에 전류를 흐르게 한다. 또 그림에서와 같이 1방향만의 coil에는 전류 (A -> Com, A' -> Com, B -> Com, B' -> Com)밖에 흐르지 않는다. 이와 같이 coil에 1방향밖에 전류가 흐르지 않기 때문에 unipolar(편극성)구동이라 부르고 있다.

2. Bipolar 구동

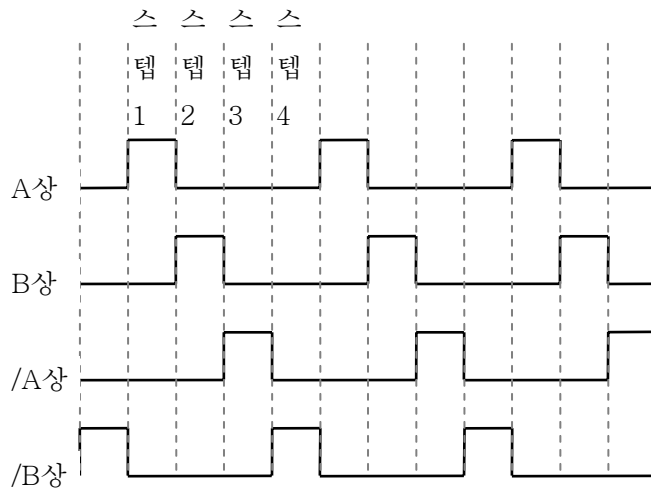
Stepping motor 의 구동으로는 전술한 것과 같은 간단한 구성으로 끝나는 unipolar구동 외에, 회로는 복잡하지만 저속 영역에서의 torque 를 개선하는 bipolar 구동이라는 방법이 있다. 이 bipolar 구동은 motor 의 coil 에 교대로 전류를 흘리도록 드라이브한다. 이 강좌에서는 bipolar 구동에 관하여는 다루지 않는다.

@ 구동방식의 종류 - 여자 방식에 따른 구분

Stepping motor 의 구동방법은 그 권선 코일에 어떤 형태로 전류를 흐르게 하는가에 따라 구별되고 있다. 이것은 모터의 종류에 따라서도 달라지지만, 가장 많이 사용되고 있는 PM형 stepping motor의 step값 90°의 경우를 예로 설명한다. 대부분의 Stepping motor의 기본적인 구조나 사고방식은 같다.

여기서 A, B, /A, /B 는 신호 입력에 따라 전자석으로 동작하게 된다. 아래 나오는 신호에서 신호 입력이 들어가면 전자석이 동작하여 축에 연결되어 있는 영구자석을 당기게 된다.

1. 4상 motor 의 1상 여자 동작



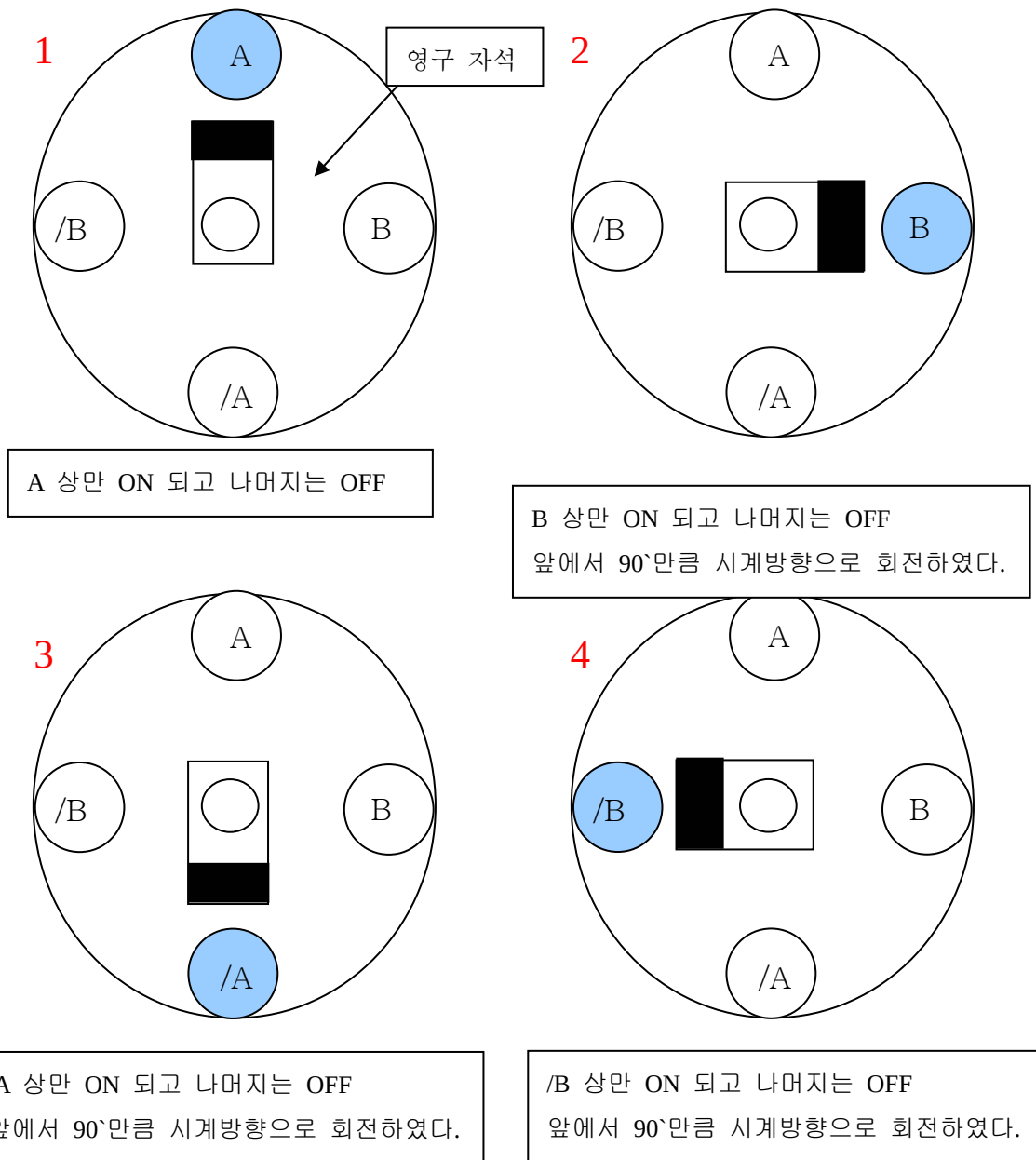
위 그림은 1상 여자 방식의 motor의 동작을 위한 신호를 나타낸다.

Step 1 A상에 연결된 TR이 On 이 되어 A->Com로 전류가 흐른다. coil에 전류가 흐름으로써 고정자의 상은 N,S극으로 여자된다. 이때 B 쪽에는 전류가 흐르고 있지 않으므로 B 쪽의 상은 비여자이지만, A쪽의 여자에 동반해서 회전자의 영구자석은 각각 N 과 S, S 와 N 이 결합해서 안정한 위치에 정지한다.

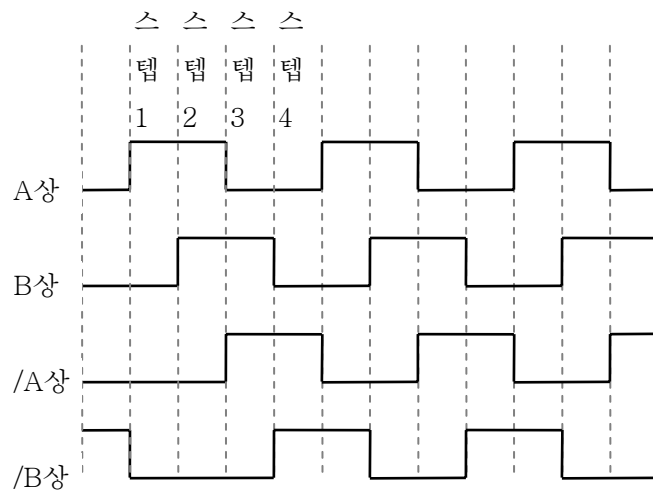
다음에 step 2로 진행하면 먼저 on 하고 있던 A상에 연결된 TR 은 off가 되고, 대신에 B상에 연결된 TR 가 on 이된다. B상에 연결된 TR이 on이 되면 B -> Com의 coil에 전류가 흘러 이번에는 90°씩 어긋나고 있는 고정자가 여자가 된다. 그리고 여자 위치가 이동한 것으로 회전자도 시계 방향으로 당겨져 90° 회전하게 된다.

같은 방법으로 step 3 과 step 4 의 동작을 함으로써 모터는 각 step 당 90°씩 진행시켜 회전시킬 수 있다.

앞에서 설명한 것을 그림으로 표현하면 아래 그림과 같다.

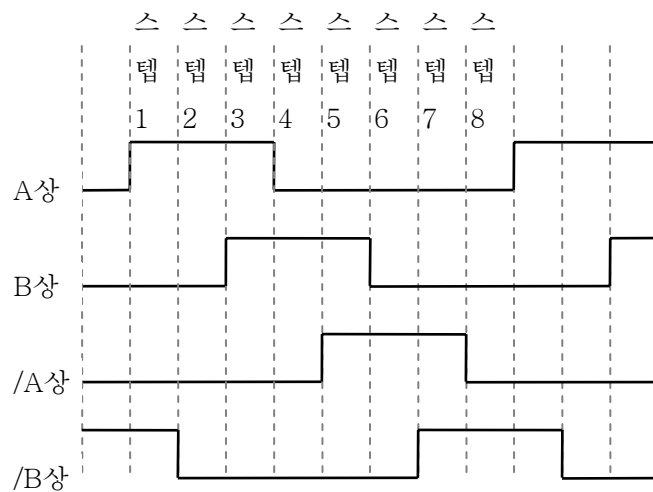


2. 4상 motor 의 2상 여자 동작



기본적인 동작은 1상 여자 방식과 동일하다 차이점은 토크를 좋게 하기 위해 2개의 코일을 동시에 동작시킨다는 것이다. 1상 여자에 비하면 전류가 많이 흐르는 단점이 있지만 정지상의 오버슈터나 언더슈터가 작고 과도 특성이 좋아진다.

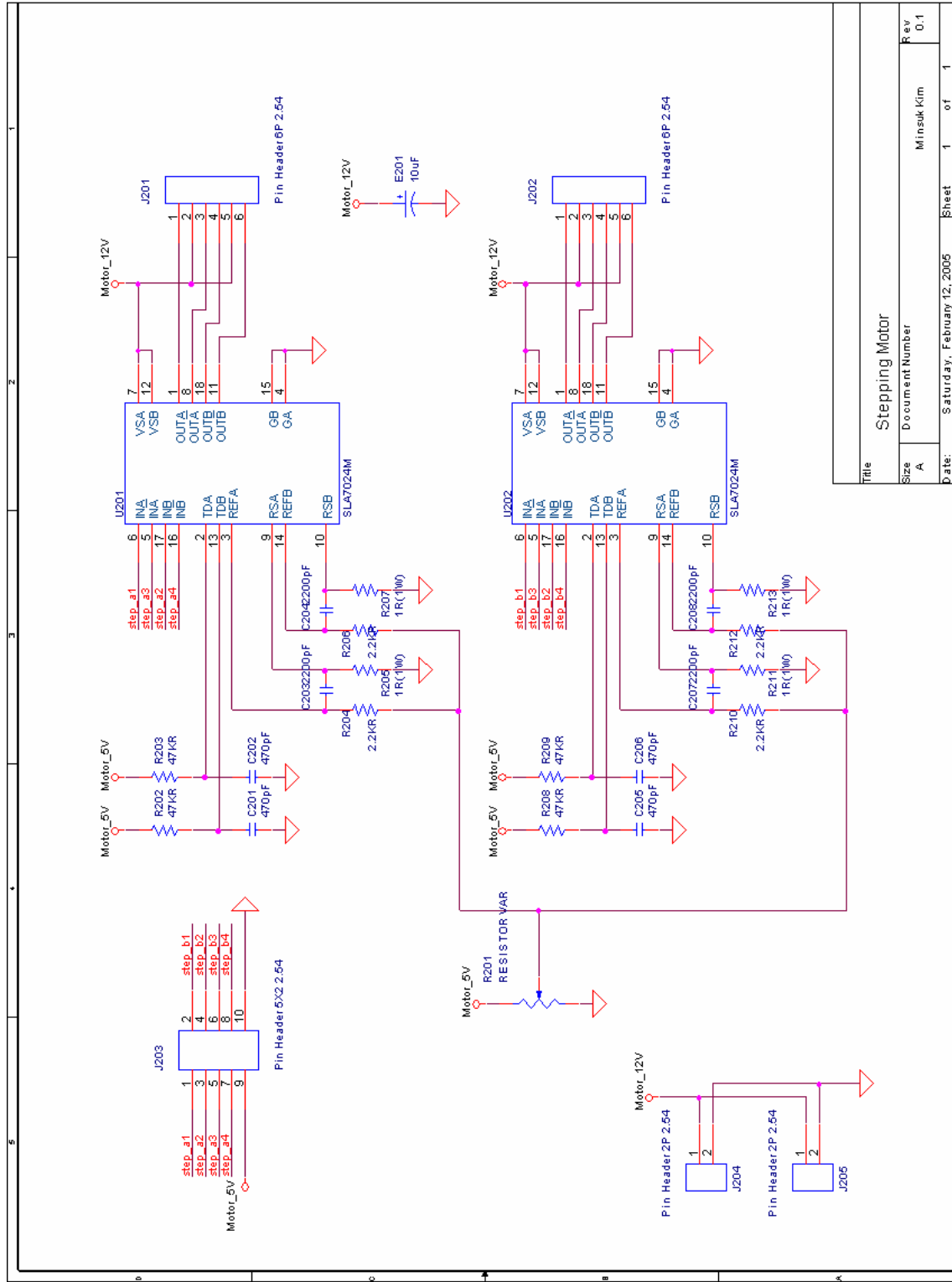
3. 4상 motor 의 1-2 상 여자 동작



위 그림의 1-2 상 여자는 전술한 1상 여자와 2상 여자가 교대로 반복하는 것이다. 따라서 회전자는 step 마다 45° 회전한다. 즉 step 각은 maker 가 표시하는 각도의 1/2 가 된다. 1-2상 여자는 1상 여자와 2상 여자의 특성을 갖고 있으므로 step rate는 배가된다. Step 의 각도가 다른 동작법에 비해 1/2 만큼 작기 때문에 3가지 방식 중 가장 성능이 좋다. 단점은 다른 여자방식과 같은 속도를 내려면 마이크로 컨트롤러 에서 클럭을 2배로 만들어야 한다는 것이다.

④ H/W 구성

모터 구동 보드 회로도



Title		
Stepping Motor		
Size	Document Number	Rev
A	Minsuk Kim	0.1
Date:	Saturday, February 12, 2005	Sheet 1 of 1

회로 설명

J203 은 모터 제어신호가 입력되는 커넥터이다. 마이크로 컨트롤러에서 생성한 모터 제어신호가 들어온다.

제어신호는 U201, U202 SLA7024(7026)으로 입력되어 이 칩에서 모터 동작을 위한 전압으로 증폭된다. 증폭된 신호는 모터 연결 커넥터인 J201과 J202 로 연결되어 모터로 전달된다.

R201 의 가변 저항은 모터의 전류를 조절해 주는 가변 저항이다. 적당히 돌려 모터의 토크를 결정할 수 있다.

나머지 저항과 캐패시터는 SLA7024 의 기본 회로이다.

예전에는 모터클럭 생성부와 증폭부 모두를 하드웨어 적으로 구성하였으니 모터클럭 생성부의 경우 마이크로 컨트롤러에서 소프트웨어적으로 구현이 가능하기 때문에 여기서는 증폭부만을 하드웨어로 구성하고 아래 MCU F/W 파트에서 모터구성을 위한 신호를 소프트웨어적으로 구현하도록 한다.

⑤ MCU F/W 프로그래밍

```
step      : 현재 모터를 위한 출력값

switch(step){
    case 0x03 : step = 0x06;  break;
    case 0x06 : step = 0x0c;  break;
    case 0x0c : step = 0x09;  break;
    case 0x09 : step = 0x03;  break;
    default   : step = 0x03;  break;
}

step      : 다음 모터 출력
```

위의 루틴을 한번 실행할 때마다 모터가 1.8° 회전한다.

이 루틴을 Timer 인터럽트 루틴에서 실행하면 모터가 타이머 속도에 맞게 돌아가게 된다. 위 루틴을 응용하여 모터를 돌려본다. 먼저 Main 함수에서 Delay 를 적당히 주고 모터를 회전시킨다.

```
void main (void)
{
    char step = 0;
    while(1){
```

```
        step = nextMotorCLK(step);           // 위의 switch case 문을 응용해서 작성
        PORTD = step;                        // PORTD 에 출력 신호가 연결 되었었다.
        Delay(100);
    }
}
```

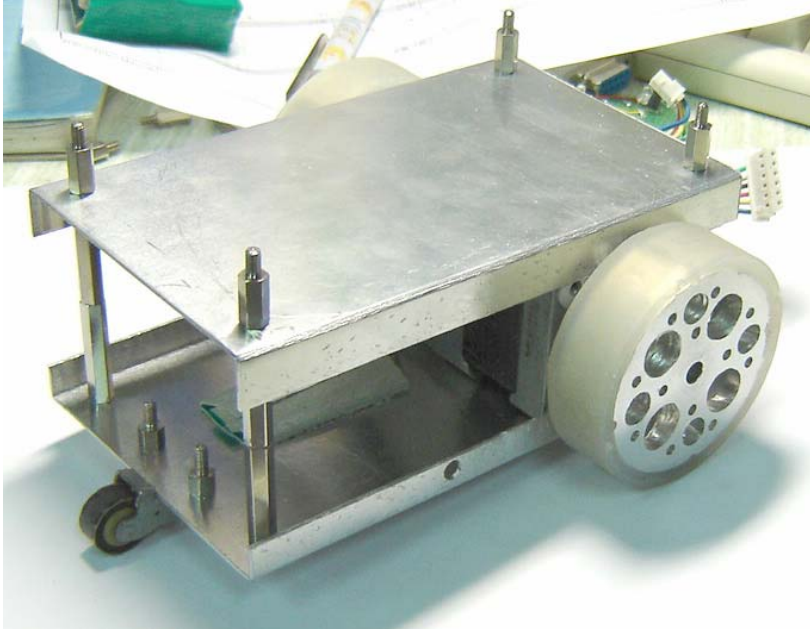
지금까지는 모터 회전을 확인하기 위한 단계이다. 모터구동 회로가 정상적으로 제작 되었는지 모터는 정상적으로 동작하는지 등을 확인 하기 위하여 작성하였다.

다음 단계로 Timer Interrupt 를 이용하여 모터를 구동 해야 한다. 타이머 인터럽트에 관련된 사항을 학습하여 테스트 해보도록 한다.

3. 기구부 제작

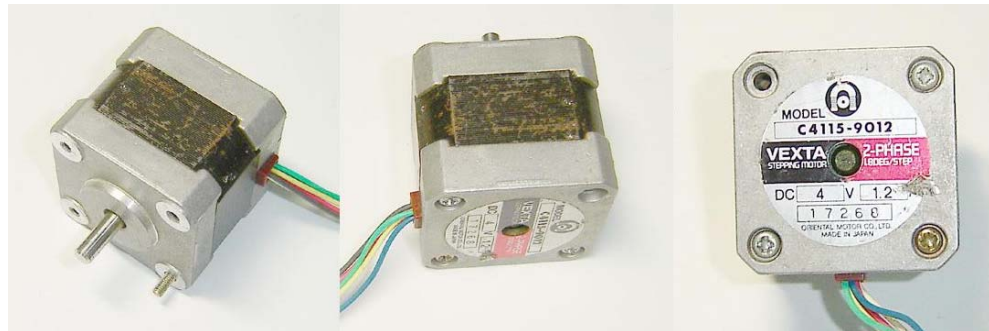
1. 기구부 제작

알루미늄판을 이용하여 라인트레이서 프레임을 구성하였다



i. 스텝핑 모터

일반적으로 4 - 5V 용 스텝핑 모터를 구매하면 적당하다. 일반적으로 스텝핑 모터에는 정격 전압의 3배 이상을 걸어 사용한다.



ii. 라인트레이서 바퀴

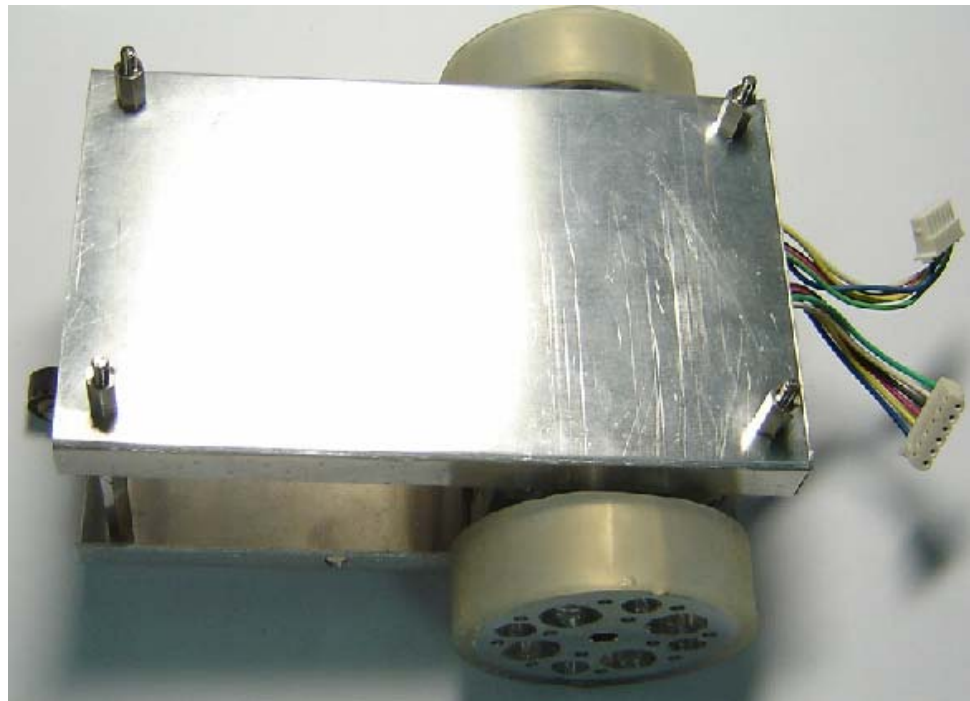
라인트레이서의 바퀴는 일반인이 쉽게 제작을 할 수 없기 때문에 전문 판매점에서 구매하였다.



iii. 라인트레이서 바디 제작하기

스테핑 모터와 바퀴를 구매하였으면 라인트레이서의 구조물에 해당하는 바디를 제작해야 한다. 바디 제작은 청계천에서 알루미늄 판을 구매하여 직접 가공함으로써 제작할 수 있다.





앞바퀴는 글루건을 이용하여 고정하였다.

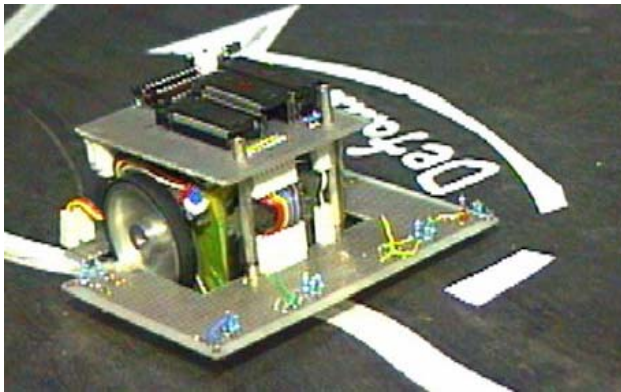


4. 프로그래밍

1. 라인트레이서란?

라인트레이서는 정해진 주행선을 따라 움직이는 자율이동로봇이다. 현재 공장자동화 부분에서 이용되고 있는 무인 반송차가 라인트레이서이다.

라인트레이서의 기본적인 원리는 주어진 주행선을 센서로 검출하여 이것에 따라 목적 위치 까지 이동하는 것이다.



라인트레이서는 크게 3부분 - 컨트롤러 부, 센서부, 모터구동부로 구성 된다.

우선 센서부의 센서를 잘 배치해야 길을 잘 인식하고 턴마크 인식할 수 있다.

모터구동부분은 잡음이 많이 생기기 때문에 잘 만들지 않으면 잡음으로 인해 다른 부분이 정상적으로 동작하지 못하는 경우가 생길 수 있다.

컨트롤러부는 기본적인 라인트레이서를 제작할 때는 상대적으로 비중이 낮으나 지능적으로 만들려면 많은 작업이 필요로 한다.

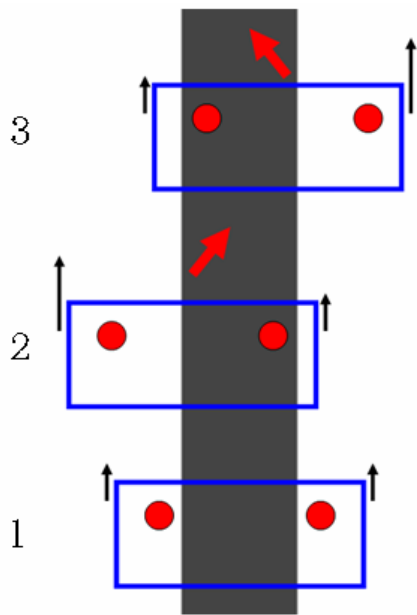
본 강좌에서는 라인트레이서를 공부하여 만들고자 하는 초보자를 위한 강좌이므로 라인트레이서로서의 동작에 목표를 두고 최대한 간단하게 동작시킬 수 있도록 한다.

2. 라인트레이서의 기본 동작 알고리즘

라인트레이서 기본 동작

센서 2쌍을 라인의 두께보다 약간 크게 위치하여 하드웨어를 제작할 경우 센서가 라인 외곽을 따라 주행 하도록 한다.

더 복잡하게 할 수도 있지만 여기서는 위와 같이 가장 간단한 알고리즘으로 알아보겠다.



1 번 상황에서 센서는 라인을 감지하지 못하였으므로 직진한다.

2 번 상황에서 오른쪽 센서가 인식 되었으므로 왼쪽 바퀴의 속도를 증가시켜 라인트레이서를 오른쪽으로 향하게 한다.

3 번 상황에서 왼쪽 센서가 인식 되었으므로 오른쪽 바퀴의 속도를 증가시켜 라인트레이서를 왼쪽으로 향하게 한다.

위의 동작을 반복하면 라인트레이서는 라인을 따라 진행하게 된다.

3. 프로그래밍 하기

기본적인 C 언어 프로그래밍과 인터럽트에 관한 공부를 먼저 해야 한다. C언어와 인터럽트에 관련해서는 관련 서적에서 자세히 나와 있으므로 이 강좌에서는 다루지 않는다.

① 타이머 인터럽트를 이용 하여 스텝핑 모터 구동하기

타이머 인터럽트를 학습하여 타이머 인터럽트를 이용하여 스텝핑 모터 구동해본다. 라인트레이서는 스텝핑 모터를 2개 구동해야 하기 때문에 타이머 인터럽트를 2개 사용하여 타이머 인터럽트 각각 바퀴 1개씩을 구동 하도록 한다.

② 센서입력 받기

센서의 입력을 받아 흰색과 검정색을 구분 할 수 있도록 한다. 시리얼 통신을 이용하여 값을 확인 하면 쉽게 할 수 있다.

③ 메인함수와 인터럽트 루틴간의 데이터 공유

메인함수에서는 속도를 변경하면 타이머 인터럽트에서 변경된 속도값을 타이머에 적용하여 스텝핑 모터의 속도를 변경할 수 있도록 한다. 그렇게 하기 위해서는 속도 값을 전역변수로 선언하여 메인함수와 인터럽트 루틴이 공유할 수 있도록 해야한다.

④ 센서 입력에 따른 스텝핑 모터 속도 변경

왼쪽 센서 입력이 있을 경우 왼쪽 스텝핑 모터의 속도를 감소시키고 오른쪽 센서 입력이 있을 경우 오른쪽 스텝핑 모터의 속도를 감소시키도록 프로그램을 작성한다.

4. 예제 프로그램

① 타이머 인터럽트를 이용 하여 스텝핑 모터 구동하기

아래 소스는 타이머 2 인터럽트의 초기화 루틴이다.

```
void timer2_init(void)
{
    TCCR2 = 0x00; //stop
    TCNT2 = 0xF1; //setup
    OCR2  = 0x0F;
    TCCR2 = 0x04; //start
}
```

아래 소스는 타이머 2 인터럽트의 인터럽트 서비스 루틴이다.

```
void timer2_ovf_isr(void)
{
    unsigned char temp;

    CLI();           //disable all interrupts
    temp = PORTD & 0x0f;

    TCNT2 = l_speed; //reload counter value
    if(l_speed == 0){
        temp = temp;
    }else{
        temp = motorOneClock(temp, 1);
    }
    PORTD = temp | (PORTD & 0xf0);
    SEI();           //re-enable interrupts
}
```

② 센서입력 받기

아래 소스는 AD Convert 의 구동 소스이다.

startConversion 함수는 해당 채널의 컨버팅을 시작하라는 함수이고 readConvertData 함수는 컨버팅한 채널의 데이터를 읽어오라는 함수이다.

```
void startConversion(unsigned char ch)
{
    ADMUX = 0x20 | (ch & 0x0f);
    ADCSRA = 0xc7;
}

unsigned char readConvertData(void)
{
    volatile unsigned char temp;
    while((ADCSRA & 0x10)==0);
    ADCSRA = ADCSRA | 0x10;
    temp = ADCH;
    temp = ADCL;
    temp = ADCH;
    return temp;
}
```

③ 메인함수 루틴

다음 소스는 메인 함수에서 수행되는 무한루프로 센서가 연결되어 있는 6번과 7번 센서를 컨버팅하여 변수에 저장시키고 이것을 임계값과 비교하여 왼쪽바퀴와 오른쪽 바퀴의 속도를 결정하도록 하였다.

```
void main(void)
{
    volatile unsigned char temp, l_sensor, r_sensor;
    volatile unsigned char step=0;

    .....

    while(1){

        l_sensor = 0;
        r_sensor = 0;
```

```

        startConversion(6);          // Left sensor Converting
        l_sensor = readConvertData();
        startConversion(7);          // Right sensor Converting
        r_sensor = readConvertData();

        PORTF = 0x03;

        if(l_sensor > THRESHOLD && r_sensor < THRESHOLD){
            l_speed = SLOW_SPEED;
            r_speed = FAST_SPEED;
            PORTF = 0x01;
        }else if(l_sensor < THRESHOLD && r_sensor > THRESHOLD){
            l_speed = FAST_SPEED;
            r_speed = SLOW_SPEED;
            PORTF = 0x02;
        }else{
            l_speed = FAST_SPEED;
            r_speed = FAST_SPEED;
        }

        printf("ch6 : %d    ch7 : %d\n\r", l_sensor, r_sensor);
    }
}

```

5. 소스 코드

```

// LineTracer Application
// Target : M128
// Crystal: 11.059Mhz
// Programmed by Minsuk Kim

#include <iom128v.h>
#include <macros.h>

#define FAST_SPEED    0x80    // 센서가 감지되었을 때 빠른 바퀴의 센서값
#define SLOW_SPEED    0x10    // 센서가 감지되었을 때 느린 바퀴의 센서값

```

```

#define THRESHOLD      17      // 센서의 라인 감지 임계값

unsigned char l_speed=0, r_speed=0; // 모터를 구동 위한 타이머 값

void port_init(void)
{
    PORTA = 0x00;
    DDRA  = 0x00;
    PORTB = 0x00;
    DDRB  = 0x00;
    PORTC = 0x00;
    DDRC  = 0x00;
    PORTD = 0x00;
    DDRD  = 0xff;      // Stepping Motor Port Left, Right
    PORTE = 0x00;
    DDRE  = 0x00;
    PORTF = 0x00;
    DDRF  = 0x03;      // LED Port PF1, PF2
    PORTG = 0x00;
    DDRG  = 0x00;
}

//UART0 initialize
// desired baud rate: 9600
// actual: baud rate:9600 (0.0%)
// char size: 8 bit
// parity: Disabled
void uart0_init(void)
{
    UCSR0B = 0x00; //disable while setting baud rate
    UCSR0A = 0x00;
    UCSR0C = 0x06;
    UBRR0L = 0x47; //set baud rate lo
    UBRR0H = 0x00; //set baud rate hi
    UCSR0B = 0x08;
}

```

```

//ADC initialize
// Conversion time: 75uS
void adc_init(void)
{
    ADCSRA = 0x00; //disable adc
    ADMUX = 0x00; //select adc input 0
    ACSR  = 0x80;
    ADCSRA = 0xC6;
}

//TIMER2 initialize - prescale:64
// WGM: Normal
// desired value: 1KHz
// actual value: 1.008KHz (0.8%)
void timer2_init(void)
{
    TCCR2 = 0x00; //stop
    TCNT2 = 0xF1; //setup
    OCR2  = 0x0F;
    TCCR2 = 0x04; //start
}

#pragma interrupt_handler timer2_ovf_isr:11
void timer2_ovf_isr(void)
{
    unsigned char temp;

    CLI(); //disable all interrupts
    temp = PORTD & 0x0f;

    TCNT2 = l_speed; //reload counter value
    if(l_speed == 0){
        temp = temp;
    }else{

```

```

        temp = motorOneClock(temp, 1);

    }
    PORTD = temp | (PORTD & 0xf0);
    SEI();                //re-enable interrupts
}

//TIMER3 initialize - prescale:64
// WGM: 0) Normal, TOP=0xFFFF
// desired value: 1KHz
// actual value: 1.008KHz (0.8%)
void timer3_init(void)
{
    TCCR3B = 0x00; //stop
    TCNT3H = 0xFF; //setup
    TCNT3L = 0xF1;
    OCR3AH = 0x00;
    OCR3AL = 0x0F;
    OCR3BH = 0x00;
    OCR3BL = 0x0F;
    OCR3CH = 0x00;
    OCR3CL = 0x0F;
    ICR3H = 0x00;
    ICR3L = 0x0F;
    TCCR3A = 0x00;
    TCCR3B = 0x04; //start Timer
}

#pragma interrupt_handler timer3_ovf_isr:30
void timer3_ovf_isr(void)
{
    unsigned char temp;

    CLI();    //disable all interrupts
             //TIMER3 has overflowed
    TCNT3H = 0xFF; //reload counter high value

```

```

    temp = (PORTD & 0xf0)>>4;

    TCNT3L = r_speed; //reload counter low value
    if(r_speed == 0){
        temp = temp;
    }else{
        temp = motorOneClock(temp, 0);
    }
    PORTD = (temp<<4) | (PORTD & 0x0f);
    SEI(); //re-enable interrupts
}

//call this routine to initialize all peripherals
void init_devices(void)
{
    //stop errant interrupts until set up
    CLI(); //disable all interrupts
    XDIV   = 0x00; //xtal divider
    XMCRA = 0x00; //external memory
    port_init();
    uart0_init();
    adc_init();
    timer2_init();
    timer3_init();

    MCUCR = 0x00;
    EICRA = 0x00; //extended ext ints
    EICRB = 0x00; //extended ext ints
    EIMSK = 0x00;
    TIMSK = 0x40; //timer interrupt sources
    ETIMSK = 0x04; //extended timer interrupt sources
    SEI(); //re-enable interrupts
    //all peripherals are now initialized
}

```

```

/* Analog Converter-----*/
void startConversion(unsigned char ch)
{
    ADMUX = 0x20 | (ch & 0x0f);
    ADCSRA = 0xc7;
}

unsigned char readConvertData(void)
{
    volatile unsigned char temp;
    while((ADCSRA & 0x10)==0);
    ADCSRA = ADCSRA | 0x10;
    temp = ADCH;
    temp = ADCL;
    temp = ADCH;
    return temp;
}

/*-----*/

void delay(int n)
{
    volatile int i,j;
    for(i=1;i<n;i++)
    {
        for(j=1;j<600;j++);
    }
}

/* Stepping Motor derive-----*/
unsigned char  motorOneClock(unsigned char step, char dir)
{
    unsigned char test;

    step = step & 0x0f;
    if(dir){
        switch(step){

```



```

while(1){

    l_sensor = 0;
    r_sensor = 0;
    startConversion(6); // Left sensor Converting
    l_sensor = readConvertData();
    startConversion(7); // Right sensor Converting
    r_sensor = readConvertData();

    PORTF = 0x03;

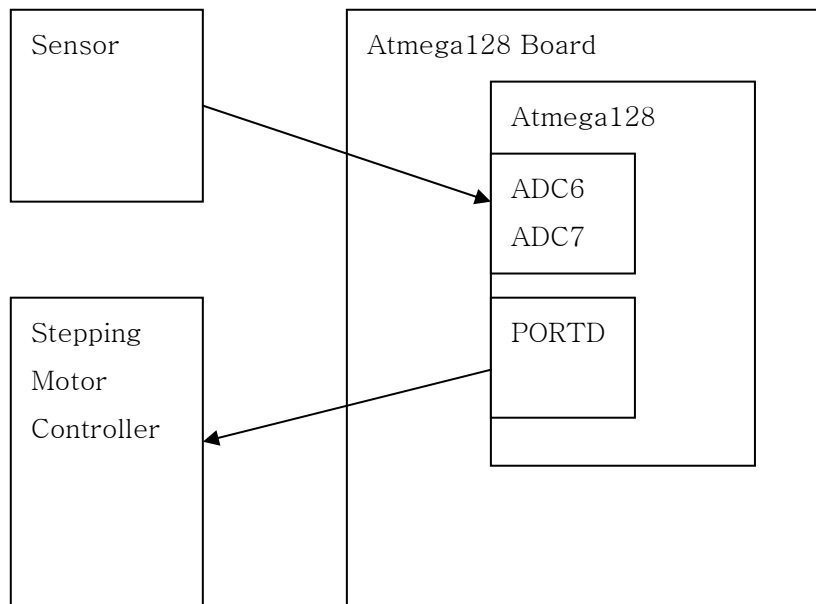
    if(l_sensor > THRESHOLD && r_sensor < THRESHOLD){
        l_speed = SLOW_SPEED;
        r_speed = FAST_SPEED;
        PORTF = 0x01;
    }else if(l_sensor < THRESHOLD && r_sensor > THRESHOLD){
        l_speed = FAST_SPEED;
        r_speed = SLOW_SPEED;
        PORTF = 0x02;
    }else{
        l_speed = FAST_SPEED;
        r_speed = FAST_SPEED;
    }

    printf("ch6 :%d    ch7 : %d\n\r", l_sensor, r_sensor);

}
}

```

라인트레이서 하드웨어 구성도



ATMega128 내부장치 사용 용도

ADC6	왼쪽 센서 입력
ADC7	오른쪽 센서 입력
PORTD0-3	왼쪽 모터구동 포트
PORTD4-7	오른쪽 모터구동 포트
타이머3	오른쪽 모터 구동
타이머2	왼쪽 모터구동
UART0	디버깅 포트