REPORT

15주차 과제

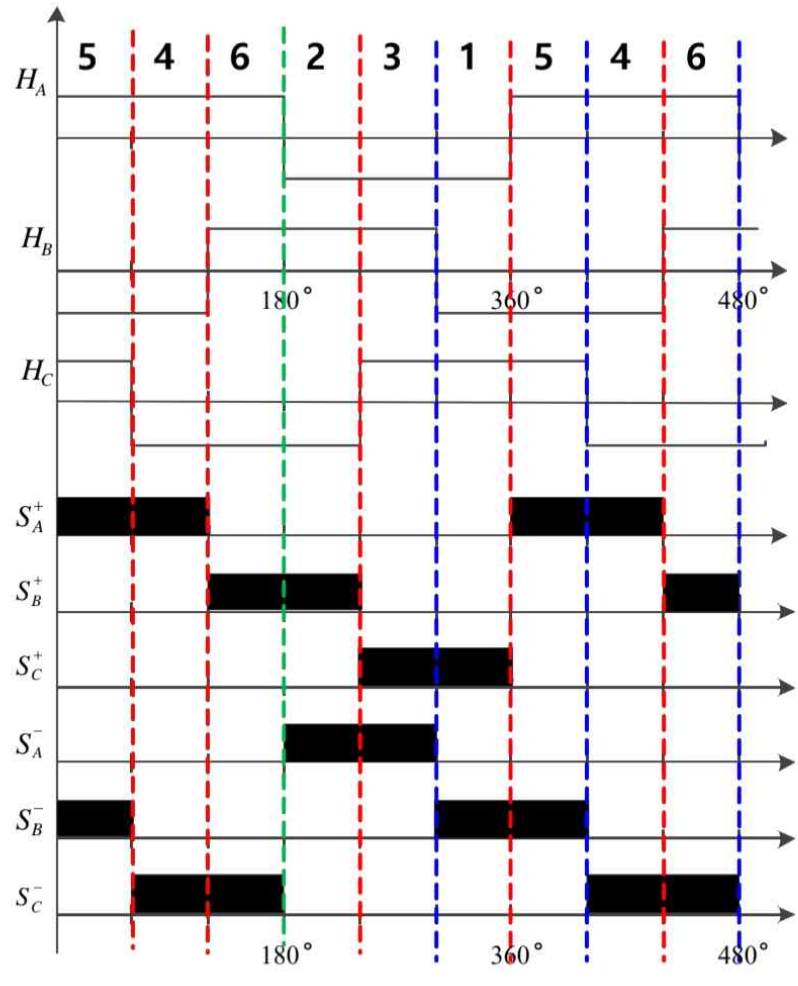
과 목 명	I	서보기기제어
담당교수	I	홍선기 교수님
학 과	I	시스템제어공학과
학 번	I	20210710
이 름	I	맹지우
제 출 일	I	2023.12.05.

BLDCM 구동 장치 제작

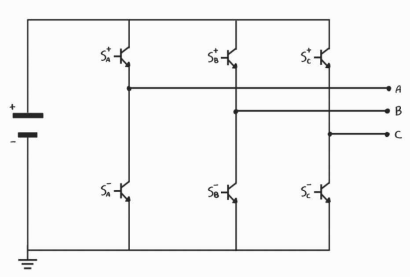
맹지우_20210710
호서대학교 시스템제어공학과
(H.P: 010-9332-6526, E-mail : 20210710@vision.hoseo.edu)

1. BLDCM 구동 원리 설명

먼저 3상 BLDC 모터를 구동시키기 위해서는 스위칭 동작 타이밍을 판단할 수 있는 회전자의 위치 정보가 필요하다. BLDC 모터의 내부에 120° 간격으로 배치된 홀센서의 신호에 따라 H_A(a상), H_B(b상), H_C(c상)의 스위치를 구동하기 위한 Gating 신호를 보이면 [그림 1]과 같다. 여기서 HA, HB, HC는 홀센서 신호이고, "S_A^+ S_B^+ S_C^+ S_A^- S_B^- S_C^-"는 각 상의 상위 스위치 및 하위 스위치를 구동하기 위한 Gating 신호이다.

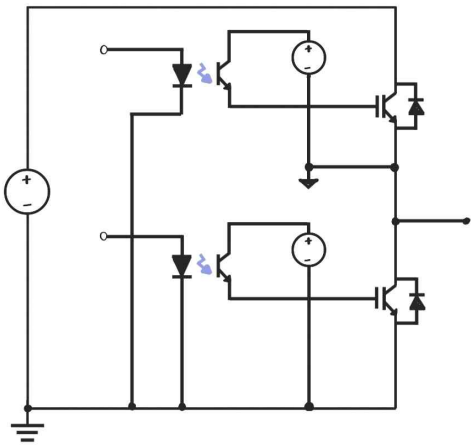


[그림 1] 홀센서 신호의 파형

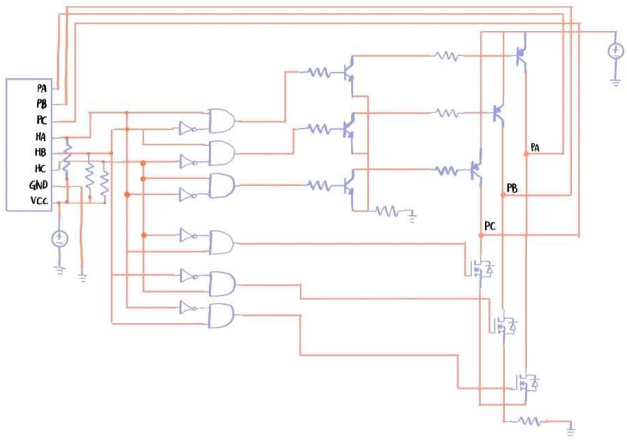


[그림 2] 인버터 회로

[그림 2]의 인버터 회로는 홀센서를 가지고 6개의 신호를 만든 다음, 나온 신호를 모터에 바로 인가시켜도 동작이 일어나지 않는다. 그 이유는 스위칭이 되지 않기 때문이다. 따라서 추가로 회로가 필요한데, 이에 대한 회로는 [그림 3]과 [그림 4]을 보면 된다.



[그림 3] 포토커플러를 사용한 회로

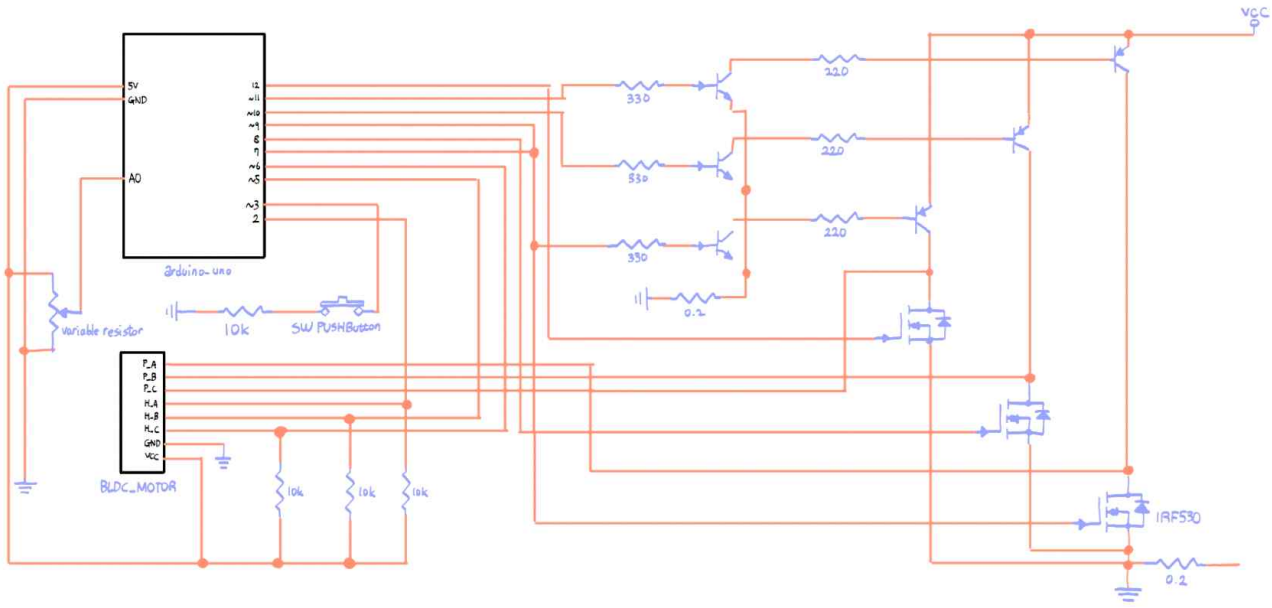


[그림 4] 분리된 전원이 없는 인버터

먼저 [그림 3]은 포토커플러를 사용한 회로이다. 포토커플러 회로에서는 전압원이 여러 개가 필요하다는 단점이 있다. 또한 서로 다른 2개의 GND가 필요한데, 같은 GND를 사용하게 되면 쇼트가 나서 스위칭이 되지 않는 문제가 생기기 때문이다. [그림 3]에서는 공통 GND를 사용하면 쇼트가 나기 때문에 스위칭이 되지 않는 문제가 있었다. 이를 해결한 방법으로는 [그림 4]와 같이 PNP형과 NPN형을 같이 사용한 달링턴 트랜지스터와 FET를 사용하였다. 회로 오른쪽 위에 총 6개의 트랜지스터가 존재한다. 이때, 왼쪽 3개가 NPN형이고 오른쪽 3개는 PNP형이다. NPN은 소용량, PNP는 고용량 트랜지스터를 달았다. 따라서 증폭 후에 GND 쪽으로 전류가 내려가 사고가 나지 않는다.

결론적으로 정리하면 BLDC 모터는 계자를 영구자석형으로 만들어 회전하도록 하고, 고정자인 전기자에는 브러시와 정류자를 대신하여 MOSFET을 스위칭 소자를 사용하여 전원을 인가한다. 직류전동기에서는 회전자가 회전하면서 브러시와 정류자가 기계적인 위치를 판별하여 전기자에 공급하는 전원을 자동적으로 적절한 타이밍에 스위칭하지만, BLDC 모터에서는 이것이 불가능하므로 외부에서 직류전원을 교류전원으로 변환하여 전동기에 공급하기 위한 인버터 회로를 가지고 있어야 하며, 홀센서로 회전자의 자극 위치를 검출하여 인버터를 구성하는 각 반도체 소자의 스위칭 동작을 수행한다.

2. 아두이노를 이용한 BLDCM 구동 원리, 회로 및 설명



[그림 5] 아두이노를 이용한 BLDC 모터 구동 회로

[그림 5]에서 알 수 있듯이, BLDC 모터의 홀센서 A, B, C에 저항을 연결하고 다른 한쪽은 Vcc(아두이노 5V)에 연결하여 풀업저항으로 사용하여 신호가 없을 때 상태를 알 수 없는 플로팅 현상을 방지했다. 그리고 이 홀센서 신호 A, B, C를 각각 아두이노의 디지털 Input 2, 5, 6번으로 연결하여 신호가 들어오면, 아두이노에 업로드된 코드의 식을 거쳐 "S_A^+ S_B^+ S_C^+ S_A^- S_B^- S_C^-"의 신호를 만들어 내보낸다. 그 후 [그림 4] 기존 인버터 회로에서와 같게 동작이 된다.

3. BLDCM PI 제어 원리 설명, 아두이노를 이용한 코드 설명

기준 속도에서 전동기의 실제 속도를 뺀 오차를 $e(t)$라고 하면 PI 제어에 대하여 아래처럼 표현한다.

$$u(t) = Kp * e(t) + Ki \int_0^t e(t)dt$$

$$u(n) = Kp * e(n) + Ki \int_0^n e(n)dt$$

여기서 K_p는 제어기의 비례 이득이고, K_i는 적분 이득을 나타내고 있다. 이 제어기의 샘플링 시간(적분 시간)을 T_s라고 하면 오차는 오차함수로서 속도오차 값인 $e_n = e(nT_s)$와 제어입력 값인 $u_n = u(nT_s)$와 같이 나타낼 수 있다. 위의 식에서 적분보다는 미분을 사용하는 것이 좀 더 편하므로, 미분하여 다음과 같이 표현해주었다. 여기서 미분항의 $\dot{e}(t)$는 $t = nT_s$에서 오차함수 $e(t)$ 곡선의 기울기가 되며, 여기서 finite-difference approximation을 적용하여 계산하면 3가지의 방법이 나온다. 하지만 2개의 방법은 사용하기 어렵기 때문에 일반적으로 가장 많이 식을 적용할 것이다.

그 식은 $\dot{e}(n) = \frac{e_n - e_{n-1}}{Ts}$ 이므로 $\dot{u}(n) = Kp * \dot{e_n} + Ki * e_n$ 에 대입하면 아래와 같이 정리할 수 있다.

$$\frac{u_n - u_{n-1}}{Ts} = Kp * \frac{e_n - e_{n-1}}{Ts} + Ki * e_n$$

위의 식을 u_n에 관한 식으로 다시 정리해보면, PI 제어기의 계산식은 다음과 같이 표현할 수 있다.

$$u_n = u_{n-1} + (Kp + Ki*Ts) * e_n - Kp * e_{n-1}$$

$$u_n = u_{n-1} + Kp(e_n - e_{n-1}) + Ki(Ts * e_n)$$

나온 PI 제어기의 계산식을 가지고, 기존의 모터 구동하는 코드에 추가해주면 된다. PI 제어를 진행할 때, T-method를 사용하여 rising 타임 간 시간을 계산하여 주파수를 계산하고 RPM을 구하는 방식을 사용하였다. 그에 대한 부분은 아래와 같다.

```
void function(){
  current_time=micros();
  period = (current_time - previous_time);
  previous_time = current_time;
  frequency = (1/period)*1000000;
  rpm = (120*frequency)/4;
  sensor_value = analogRead(A0);
  dirsed_rpm = map(sensor_value,0,1023,800,1500);
  error = dirsed_rpm - rpm;
  p = kp*error;
  pi = p+i;
  duty = duty+pi;
}
```

PI 제어의 전체 코드

```
#include "Timer.h"
Timer t;
int hall1 = 2;
int hall2 = 5;
int hall3 = 6;
int SAM = 12;
int SBM = 8;
int SCM = 7;
int SAP = 9;
int SBP = 11;
int SCP = 10;
unsigned long current_time ;
unsigned long previous_time;
float frequency;
float period;
int rpm;
int hall1_state;
int hall2_state;
int hall3_state;
int state;
int sensor_value;
int dirsed_rpm;
int error;
float kp =0.01;
float ki =0.1;
float p,i,pi;
float duty;
unsigned long a,b;
int sw_state;
int count =0;
void setup(){
  attachInterrupt(0,function,RISING);
  Serial.begin(2000000);
  pinMode(hall1,INPUT);
  pinMode(hall2,INPUT);
  pinMode(hall3,INPUT);
  pinMode(3,INPUT_PULLUP);
  for(int i =7;i<13;i++){
    pinMode(i,OUTPUT);
  }
  t.every(1,switching);
}
void loop(){
  a = millis();
  t.update();
  if(a - b >=30){
    Serial.println(rpm);
    b = a;
  }
}
void function(){
  current_time=micros();
  period = (current_time - previous_time);
  previous_time = current_time;
  frequency = (1/period)*1000000;
  rpm = (120*frequency)/4;
  sensor_value = analogRead(A0);
  dirsed_rpm = map(sensor_value,0,1023,800,1500);
  error = dirsed_rpm - rpm;
  p = kp*error;
  pi = p+i;
  duty = duty+pi;
}
void switching()
{
  i = (ki*error*0.001);
  hall1_state = digitalRead(hall1);
  hall2_state = digitalRead(hall2);
  hall3_state = digitalRead(hall3);
  state =4*hall1_state+2*hall2_state+hall3_state;
  if(state ==5){
    analogWrite(SAP,duty);
    digitalWrite(SBP,LOW);
    digitalWrite(SCP,LOW);
    digitalWrite(SAM,LOW);
    digitalWrite(SBM,HIGH);
    digitalWrite(SCM,LOW);
  }
  else if(state ==4){
    analogWrite(SAP,duty);
    digitalWrite(SBP,LOW);
    digitalWrite(SCP,LOW);
    digitalWrite(SAM,LOW);
    digitalWrite(SBM,LOW);
    digitalWrite(SCM,HIGH);
  }
  else if(state ==6){
    analogWrite(SAP,LOW);
    digitalWrite(SBP,duty);
    digitalWrite(SCP,LOW);
    digitalWrite(SAM,LOW);
    digitalWrite(SBM,LOW);
    digitalWrite(SCM,HIGH);
  }
  else if(state ==2){
    analogWrite(SAP,LOW);
    digitalWrite(SBP,duty);
    digitalWrite(SCP,LOW);
    digitalWrite(SAM,HIGH);
    digitalWrite(SBM,LOW);
    digitalWrite(SCM,LOW);
  }
  else if(state ==3){
    analogWrite(SAP,LOW);
    digitalWrite(SBP,LOW);
    digitalWrite(SCP,duty);
    digitalWrite(SAM,HIGH);
    digitalWrite(SBM,LOW);
    digitalWrite(SCM,LOW);
  }
  else if(state ==1){
    analogWrite(SAP,LOW);
    digitalWrite(SBP,LOW);
    digitalWrite(SCP,duty);
    digitalWrite(SAM,LOW);
    digitalWrite(SBM,HIGH);
    digitalWrite(SCM,LOW);
  }
}
```

4. 소감 및 결론

3학년 1학기 아두이노 실험 시간에 진행했던 라인트레이서를 접하면서 처음으로 PI 제어를 공부했었는데, 이번 2학기에 BLDC 모터를 구동시키면서 복습할 수 있는 기회가 되었던 것 같다. 실제로 이번 인버터 회로와 BLDC 모터를 구동시키기 위한 회로를 만들어보면서 논리 게이트나 트랜지스터, MOSFET 등 여러 소자를 사용하였는데, 모터가 돌아가지 않아 소자도 터지고 Power Supply에서 CC도 뜨는 등 시행착오를 겪고 회로를 수정하는 시간을 겪으면서 어떤 식으로 소자들이 작동하는지 어느 정도 이해될 수 있었다.
그리고 2학년 2학기 실험 시간에 잠깐 했었던 납땜을 이번 기회에 제대로 해보았는데, 처음에는 감이 많이 잡히지 않아서 납땜한 소자들을 도통 테스트했을 때 이어져 있는 등 실력이 많이 부족했는데 점차 어떻게 하면 잘 되는지 익힐 수 있게 되었다.