Line tracer

WITH VHDL

2021741062 로봇학부 김현준

LINETRACER 이론

내부구현과 외부구현 설명

코드 해석

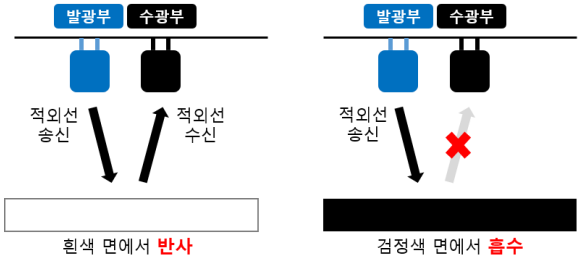
고찰 및 느낀 점

LINETRACER 이론

인식에 따른 신호처리 방법

LINETRACER는 발광부와 수광부를 이용하여 자신이 현재 검정색 면에 있는지, 흰색면에서 있는지를 확인함. 발광부에서 적외선을 송신하였을 때 검은색은 흡수, 흰색은 반사시키는 성질이 있기에 흰색면에서만 수광부에서 수신을 할 수 있게 됨.

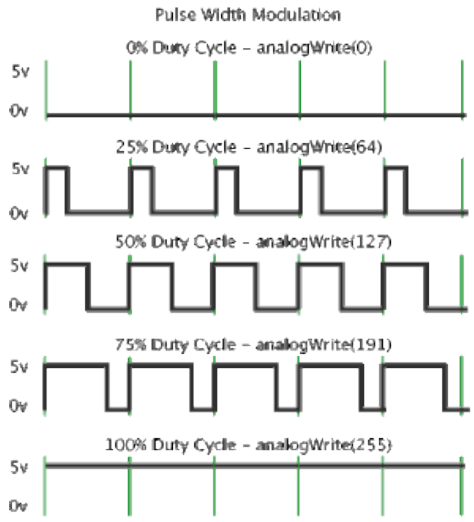
그렇기에 코드를 짤 때에도 검은색을 감지하는 것은 수광부의 상태가 '0'일 때임.



< 라인트레이서 센서 원리 >

모터 제어 방식(PWM)

PWM은 Pulse Width Modulation으로 펄스의 폭에 정보를 주는 방식임. 디지털 신호의 출력이 HIGH, LOW인 비율에 따라 출력전력을 제어할 수 있음. 주파수가 일정하기에 노이즈 필터링이 용이하다는 장점이 있음.



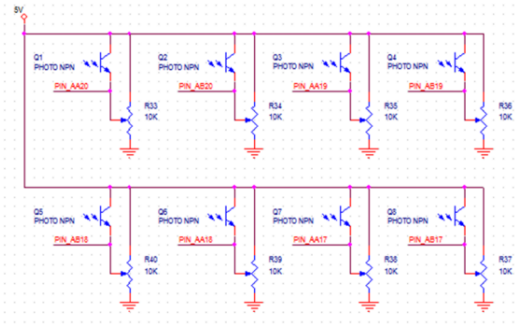
위 그림을 보자면 위에서부터 아래로 내려오며 속도가 증가하고 있음.

PWM을 코드로 구현하기 위해, 전체 범위를 정해두고, 변수를 설정하여 모터의 속도를 제어하고자 함.

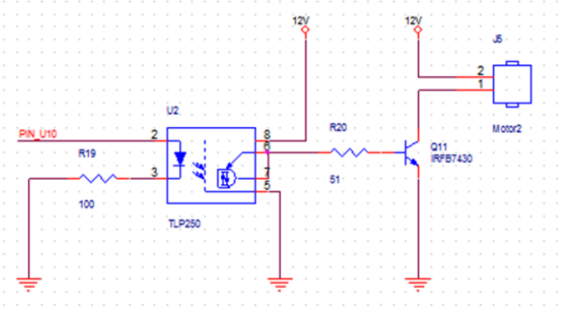
보드 구조

DE0 보드와 추가적인 라인트레이서 회로를 사용하여 로봇이 움직이게 됨.

발광부는 5V의 전원을 인가받아 적외선을 송신하며, 가변저항과 이어진 수광부에서 이를 송신하게 됨. 가변저항으로 적절한 저항값을 주며 출력 전압을 조절하고 이에 따라 로봇이 흰색과 검은색을 잘 구분할 수 있는지 체크해야함. 수광부에서 얻은 0,1 데이터는 DE0 보드에 전달됨.



TLP250 소자가 모터의 회로 부분에 사용된 것을 알 수 있음. TLP250은 게이트 드라이버로 스위치의 온오프를 제어하는데 큰 역할을 함. 보드에 연결되어 PWM 주기를 받은 만큼 스위칭 작용을 하여 모터 작동을 할 수 있게 함.



내부구현과 외부구현

내부구현으로는 DE0 보드에 있는 4개의 7 SEGMENT를 활용하여 0.01s 단위까지 표현하고자 하였으며, 9개의 스위치 중 1번 스위치의 상태가 1일 때에만 로봇이 움직이게 하여, 정확한 랩타임을 측정할 수 있도록 하였음

외부 구현의 경우 ps2 키보드를 활용하여 특정 키워드 입력 시 코드가 실행되도록 만들려고 하였으나, 내부 구현에서 그 기능을 구현하였기 때문에 외부 구현은 하지 않았음.

코드 해석

코드를 구성하기 전 원하는 로봇의 동작은 다음과 같음

1. 3,4번 센서를 활용하여 직선 주행을 하며, 중간에서 이탈하여 사이드의 센서가 감지되면 속도를 변화시키며 바로 복귀할 수 있도록 한다
2. FLAG를 새워 모든 센서가 인식되는 수를 카운트한후 마지막 선에서 정지하는 것이 아닌 다른 방법을 사용하여 정지를 한다.
3. 모든 센서가 다 인식될 시에 로봇이 방향을 순간적으로 잡지 못하여 라인에서 이탈할 가능성이 있기 때문에 모든 센서가 다 인식되어도 직진할 수 있도록 코드를 수정한다

2번과 같은 방법을 채택한 이유는, 코드에서 COUNT를 이용하여 초를 세는데 IF문이 50MHZ 정도로 굉장히 빠르게 카운트되기 때문에 정확한 수를 셀 수 없다고 생각하였으며, 이를 구현하는 것보다 다른 방법이 더 쉬울 것이라고 생각했기 때문임.

그렇기에 생각한 방법은 로봇이 1회 완주할 때 걸리는 평균시간을 T 라고 할 때, 랩타임이 T초가 넘고 모든 센서가 처음으로 인식되었을 때 멈추게 하는 방법을 채택하였음.

코드 설명

```
entity linetracer is
   port(
          -- in port
          in_bit : in bit_vector (0 to 7); -- photo diode
          clk : in std_logic; -- clk
          switch : in std_logic_vector(1 downto 0); -- switch
          -- out port
          LED : out bit_vector (0 to 7); -- 8 led
          Motor_L : out std_logic; -- left motor
          Motor_R : out std_logic; -- right motor
          seg0 : out std_logic_vector(6 downto 0);
          seg1 : out std_logic_vector(6 downto 0);
          seg2 : out std_logic_vector(7 downto 0);
          seg3 : out std_logic_vector(6 downto 0)
          );
end linetracer;
```

센서의 수광부, clock과 switch를 input으로 하며, LED, 7-SEG, MOTOR를 output으로 설정함.

```
architecture arc of linetracer is

   signal start : integer range 0 to 1 := 0; -- start when signal state '1'

   signal count : integer range 0 to 500000 := 0;
   signal carry_cnt : integer range 0 to 50000 :=0;
   signal count1 : integer range 0 to 100 := 0;
   signal count2 : integer range 0 to 100 := 0;
   signal count3 : integer range 0 to 100 := 0;
   signal count4 : integer range 0 to 100 := 0;
   signal sec_final : integer range 0 to 100 := 0;

   signal Lspeed : integer range 0 to 50000 := 0; --pwm
   signal Rspeed : integer range 0 to 50000 := 0; --pwm
```

본 코드에서 사용한 변수들은 다음과 같음.

Start 변수는 코드가 실행하거나 정지할 수 있게 제어하는 역할을 하며, switch가 state '1'을 유지할 때만 1의 값을 가짐.

count와 carry_cnt 변수는 수업시간에 배운 것처럼 초를 세기 위하여 선언되었음

count 변수들은 각각 7-segment에 활용하기 위한 변수들이며 sec_final은 로봇의 총 완주 랩타임을 저장하는 변수임.

L,R_speed는 PWM을 제어하기 위한 변수로 사용하였음.

```
if(start = 1) then
   if rising_edge(clk) then
      count <= count + 1;
      carry_cnt <= carry_cnt + 1;

      if(count = 499999) then
         sec_count1 <= sec_count1 + 1;
         if((count1 mod 10) = 9) then
            count2 <= count2 + 1;
            if((count2 mod 10) = 9) then

               count3 <= count3 + 1; -- 1sec
               sec_final <= sec_final + 1;

               if((sec_count3 mod 10) = 9) then
                  count4 <= count4 + 1;
               end if;

            end if;

         end if;

         count <= 0;

      end if;

      if(carry_cnt >= 49999) then
         carry_cnt <= 0;
      end if;

   end if;
```

위 코드는 count를 위한 코드로, count1은 0.01s, count2는 0.1s, count3은 1s, count4는 10s를 기준으로 함.

위에서 sec_final은 전체 랩타임을 잰다고 하였는데, 정확한 랩타임이 아닌 대략적인 범위를 알기 위함이기에 1s를 기준으로 체크하도록 코딩하였음.

```
   if(sec_final >= 40) then
      if(in_bit(2) = '0' and in_bit(5) = '0') then
         if(in_bit(3) = '0' and in_bit(4) = '0') then
            speed_L <= 0;
            speed_R <= 0;
         end if;
      end if;
   else
      --if(in_bit(3) = '0' or in_bit(4) = '0') then -- go straight

         if(in_bit(1) = '0' and in_bit(6) = '0') then
            speed_L <= 15000;
            speed_R <= 15000;
         elsif(in_bit(2) = '0') then -- left bias
            speed_L <= 20000;
            speed_R <= 0;
            if(in_bit(1) = '0') then
               speed_L <= 35000;
               speed_R <= 0;
            end if;
         elsif(in_bit(5) = '0') then --right bias
            speed_L <= 0;
            speed_R <= 20000;
            if(in_bit(6) = '0') then
               speed_L <= 0;
               speed_R <= 35000;
            end if;
         else -- normal state
            speed_L <= 15000;
            speed_R <= 15000;
         end if;
      --end if;
   end if;

end if; -- line 42 end if;
```

아래 코드는 모터 제어에 대한 코드로, 센서가 인식하는 부분에 따라 모터의 움직임을 바꾸고자 함.

처음에는 사용하는 8개의 센서 중 중앙센서인 3,4번의 센서가 인식되어야 앞으로 전진할 수 있게 만들었지만 주행 시에 3,4번이 모두 인식이 안될 가능성이 크기 때문에 이는 삭제하였음(주석 처리된 부분)

전체 코드의 Else문(아래부분) 안의 if문은 원하는 작동방식 1번을 해결하기 위한 코드로, 양 끝단의 센서가 인식되어도 전진할 수 있도록 구성함.

Elsif 문들은 로봇이 치우쳐졌을 때 사용하는 코드로써, 치우쳐지는 각도에 따라 세부적인 컨트롤이 가능할 수 있도록 하였음. 4번 기준으로 5번, 6번이 인식될 경우 6번이 인식될 때의 상황은 5번이 인식되는 상황보다 더 큰 각도로 치우쳐졌다고 생각할 수 있기에, 6번 센서가 인식되면 5번 센서보다 더 크게 출력을 낼 수 있도록 하였음.

0번과 7번의 경우에는 사용하지 않았는데 7번 센서가 색깔 인식을 하지 못하였기에 (12V가 인가되어야 하지만 다른 센서보다 낮은 5V가 인가됨) 0번도 사용하지 않았고 1번부터 6번의 센서만 사용하게 되었음.

위의 전체 if문(윗부분)은 FLAG를 세우지 않고 정지를 하기 위한 코드로써 평균 랩타임인 40초를 지나갔을 때 모든 센서가 검은색을 인식한다면 정지할 수 있는 코드임.

```
process(carry_cnt)

begin
   if(start = 0) then
      Motor_L <= '0';
      Motor_R <= '0';
   elsif(start =1) then

      if(carry_cnt> speed_L) then
         Motor_L <= '0';
      elsif(carry_cnt <= speed_L) then
         Motor_L <= '1';
      end if;

      if(carry_cnt> speed_R) then
         Motor_R <= '0';
      elsif(carry_cnt <= speed_R) then
         Motor_R <= '1';
      end if;

   end if;

end process;
```

Pwm 제어를 위한 코드로 모터제어 코드에서 speed_L,R에 넣은 값을 이용하여 모터의 속력을 제어할 수 있음.

고찰 및 느낀 점

모든 센서가 활용되지 못하였기에 좀 더 부정확한 결과값이 나온 것 같음.

FLAG를 사용하지 않고 더 간단한 방식을 활용하여 로봇을 정지시키고자 하였지만, 결론적으로는 최종 시연에서 로봇이 정지하지 못하였음.

코드를 수정하는 과정에서 어떠할 때는 설정해둔 시간이 되면 센서 인식에 상관없이 모터가 멈춘 적도 있었으며, 설정해둔 시간을 넘어도 정지를 하지 못하였음. 코드가 복잡하더라도 FLAG를 세우는 것이 더 확실하고 좋은 방법임을 알았음.

7805 소자가 꽤나 많은 열을 발산하여 로봇을 장시간 주행하면 소자의 열로 인해 로봇이 정상적으로 작동하지 못하였음. 7805 대신 열이 더 적은 소자를 사용했다면 길게 로봇을 구동할 수 있었을 것이라고 생각함.

PS2 키보드를 활용하진 못하였으나, 추후에 PS2 키보드를 활용해서 구동해보고 싶어 졌음.

평소에 사용하지 못할 VHDL 언어를 배우고 사용하는 것이 쉽지는 않았고, VHDL 특성상 자유로운 코딩이 힘들었기에 어려움이 있었으나 만들어진 로봇이 라인트레이싱을 잘 할때는 뿌듯함을 느낄 수 있었음.

안전하게 라인트레이싱을 하기 위하여 모터의 출력을 적게 설정하였는데 출력을 좀 높게 사용해서 랩타임을 줄였다면 더 좋았을 것이라고 생각함.

참고문헌

https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=eduino2&logNo=221048776805 – 라인트레이서 사진

https://nowrang.tistory.com/32 - TLP250 관련 정보

https://ko.wikipedia.org/wiki/%ED%8E%84%EC%8A%A4_%ED%8F%AD_%EB%B3%80%EC%A1%B0 – PWM 관련 사진

로봇학실험 11주차 강의자료