< Feedback 제어 >

본 문서는, 포켓봇을 위한 궤환제어기를 구성해 보는 내용을 담고 있습니다. 가능한 간단하고 튜닝이 손쉬운 것을 선택하고자 하는데, 역시 이를 위해서는 PID 제어기가 짱이겠죠... PID 제어기의 일반적인 사항은, 여러 웹사이트에서 손쉽게 입수할 수 있습니다만... 대개 비전공자가 이해하기엔 좀 갑갑하죠.

http://control.dgu.edu/pid.html

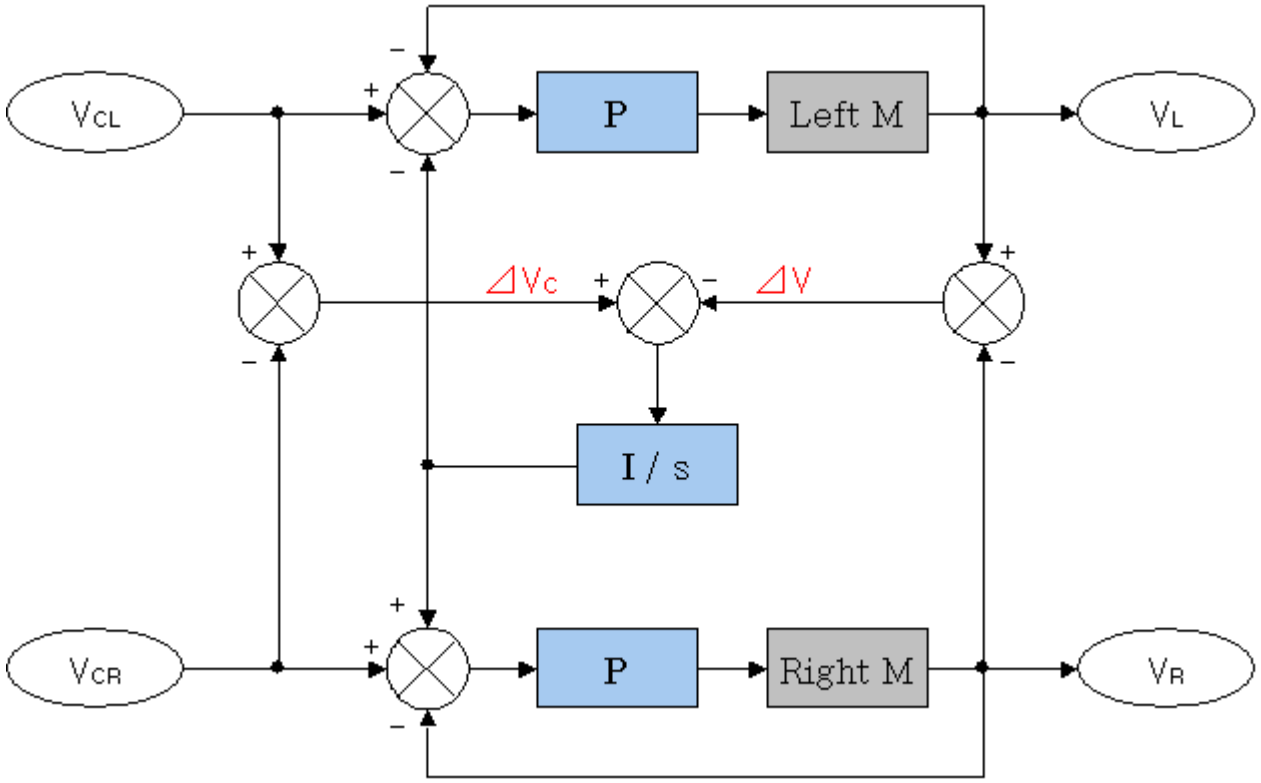
을 참조하시면 되겠습니당. 이보다 더 쉽게 설명한 것은 없을 것입니다.

뭐 비례(Proportional), 적분(Integral), 미분(Derivatives) 항들을 더해서 전체적인 효과를 본다는 것인데요... 대략 이런 원리구나 하는 정도만 알아두어도 당장 써먹는데는 지장이 없겠습니다.

1. 제어기 구성해 보기

포켓봇을 위한 제어기는 이러한 일반적인 PID 제어기에서 약간 변형된 것을 사용할 것입니다. 아이디어는 아니타 플린(Anita Flynn) 아주머니가 MIT Media Lab에서 박사과정다닐 때 지은 'Mobile Robots'라는 책에서, Rug Warrior 라는 자율이동 로봇에 사용한 것과 동일합니다.

두 개의 모터는 각각 단순한 속도에 관해 비례(P) 제어만을 받고, 동시에 적분(I) 제어는 두 바퀴의 속도차이에 관하여 이루어집니다. 미분(D)제어는 생략됩니다. 아래 다이어그램을 참조바랍니다.



Prepared P-I Controller

우선, 포켓봇의 인공지능(?)이 결정한 사항... 즉 로봇이 어디로 어떻게 움직여야 될 것인가 하는 것이 주어진다고 합시다. 그러면, 좌측과 우측의 모터가 어느정도 속도로 돌아가야 될지 결정되겠죠. 이렇게 결정된 모터의 요구 속도를 V_{CL} 및 V_{CR} 이라고 합시다.

좌측 모터와 우측 모터는, 여기에 현재의 실제 속도인 V_L 및 V_R과 각각 뺄셈을 해서 오차(Error)를 구합니다. 그 오차에 비례치 P를 곱해서 모터에 줄 전압을 결정해 주는 것이죠...

한편으로, 양측 모터의 요구 속도가 주어졌으니 그 차이값인 ΔV_C가 주어진 것으로 볼 수 있겠죠. 마찬가지로, 실제 양측 모터의 속도차인 ΔV도 구해서 그 오차를 적분 제어기에 곱합니다. 이때, 여기서도 그냥 비례치를 곱해주면 되지 왜 하필 적분하느냐? 하는 의문이 들 수 있습니다. 그 이유는 적분항의 성질 때문입니다. 즉 에러가 자꾸 누적이 되어 로봇의 방향이 엉뚱하게 틀어져 버릴 수 있으므로, 여기에는 적분항을 쓰기로 하는 것입니다. 포켓봇 처럼 양쪽 바퀴의 속도차이를 이용해서 조향하는 구조일 경우, 이것은 상당히 중요한 것이

라 할 수 있겠습니다.

물론 이것을 더욱 강인하게 제어하고자 한다면, P 제어기 부분을 PID 제어기로 대치하면 되죠... 하지만 일단은 가장 간략하게 구성해 봅니다. 만일 이정도 만으로도 성능이 충분히 나온다면 구태여 복잡하게 할 필요가 전혀 없겠죠? 특히 본 포켓봇 같은 경우는, 외란이 많은 대신 상당히 안정된(Stable) 시스템이므로... 안정도 해석등이 사실상 불필요하고, 게다가 속도까지 느린 경우니 주파수 해석도 실질적으로는 불필요하게 되겠습니당.

원래 이런식으로 시스템의 블록 다이어그램이 구성되면... 각 블록을 전달함수(Transfer Function) 또는 상태방정식(State Equation)으로 바꾸어 적어주고, 해석과 시뮬레이션을 통해 제어기의 파라미터인 P나 I 값 따위를 결정해 줍니다. 하지만 그런 과정을 생략하고 바로 코딩 과정으로 넘어가 보도록 합시다.

한 번 엔지니어링 센스를 발휘해 보도록 합시당...

2. 각 블록을 C 코드로 바꾸어 보기

다이어그램 상에서, X표 쳐진 뜅그라미 부분은... 단순 덧셈 뺄셈 부분이죠. 각 변수를 그림대로 더하고 빼주면 되겠습니다.

그리구... P 제어기 부분은... 그냥 P 값을 곱해주면 되구요. 물론 이렇게 곱해진 값은 바로 모터로 들어가는 값인 만큼, 적절한 P값을 정해주는게 중요하겠죠? 쉽당...

음... I/s 는 어떻게 바꿀까요? 고민할 필요 없습니당. '적분'이쟈나요. 그냥 적분 해 주면 되는데, 아시다시피 일정시간마다 되풀이 실행되는 '이산시간계의 디지털 제어'이므로... 이전 값에 요번 값을 그냥 계속 더해주는 것이 바로 적분입니다. 값을 쌓아나가면 되니까요. '그러면, 나중에는 값이 엄청 커지지 않나요?'하는 질문은.. 헐헐.. 음수값도 많이 나오므로, 그럴 걱정은 없죠.

참.

그런데 연산능력과 속도가 제한된 마이컴 제어에서는, DSP에서처럼 부동소숫점 연산을 마음껏 구사하기가 힘들죠? 그래서 미분항을 빼버린 이유도 있지만요(나눗셈 땜에).. 여기서는 곱셈, 덧셈, 뺄셈만 있으면 될 것이므로 걱정끝.

이상의 사항들을 모아모아.. 대강 코드를 써 봅니다.

```
#define P       10              // P 값 : 여기서는 아무거나 넣어 두었슴.
#define I       1               // I 값 : 역시 아무거나 넣어 두었슴.

int v_l, v_r;                   // 양짝 바퀴의 실제속도
int v_cl, v_cr;                 // 양짝 바퀴의 원하는 속도(목표속도)
int dv_l, dv_r;                 // 양짝 바퀴의 실제 속도차이
int ev_l, ev_r;                 // 양짝 바퀴의 목표속도와 실제속도의 오차
int integral, integral_past;    // 금번 적분 값과 과거 적분값
int u_l, u_r;                   // 양짝 바퀴에 줄 전압출력(실제로는 PWM 비율이겠죠?)

int pi_controller(void)
{
    integral = integral_past + I*((v_cl-v_cr)-(v_l-v_r));   // 적분 제어기
    u_l = P*(v_cl-v_l-integral);                            // 왼쪽 바퀴의 비례 제어기
    u_r = P*(v_cr-v_r+integral);                            // 오른쪽 바퀴의 비례 제어기
    integral_past = integral;                               // 계속 적분해야 되니끼니 저장.
    return 0;
}
```

여기서, 비례제어기 부분을 보면... integral 항에 P를 곱해주는 것으로 되어 있는데요... 여의치 않으면 괄호 바깥으로 끄집어 내어 완전히 분리해도 무방하겠죠.

이것으로 제어기 구성 끝?

아뇨... 자세히 살펴보면 아시겠지만, u_l 또는 u_r 값이 너무 크게 나와서(에러값이 너무 커서)... 변수의 한계를 넘어 버리면 오버플로우 되는 문제가 있겠죠? 그럴땐 그 값에 도달했는지 보고 막아주는 조치가 필요할 것입니다. 안그러면, 로봇이 미쳐날뛰는 경우가 생길 수 있겠죠?

그리구... 이 코드를 실제 적용하기 위해서는, 변수의 형변환도 필요할 꺼고... 출력 부분에선 PWM 신호와 DIR 신호를 분리해 주는 것도 있어야 겠군요.

그리구 나서, '튜닝'을 해야겠죠. P 값과 I 값을 결정해야 되니까요. 대략 가늠해 보구서 몇 번 반복하다보면 비교적 쉽게 잡을 수 있을 것으로 생각합니다.

3. 결 론

가장 쉬운 방법으로 피드백 제어기를 구성해 보았습니다. 실제 이 제어기를 돌려 보는 것은 다음 글로 넘깁니다. 그 이유는, 전체 프로그램의 구조를 먼저 결정해야할 필요가 있겠다고 생각하기 때문입니당.

상태 기계 (State Machine) 라는 말을 자주 쓰죠... 바로 그런 식으로 짜보려 합니다.

그럼!