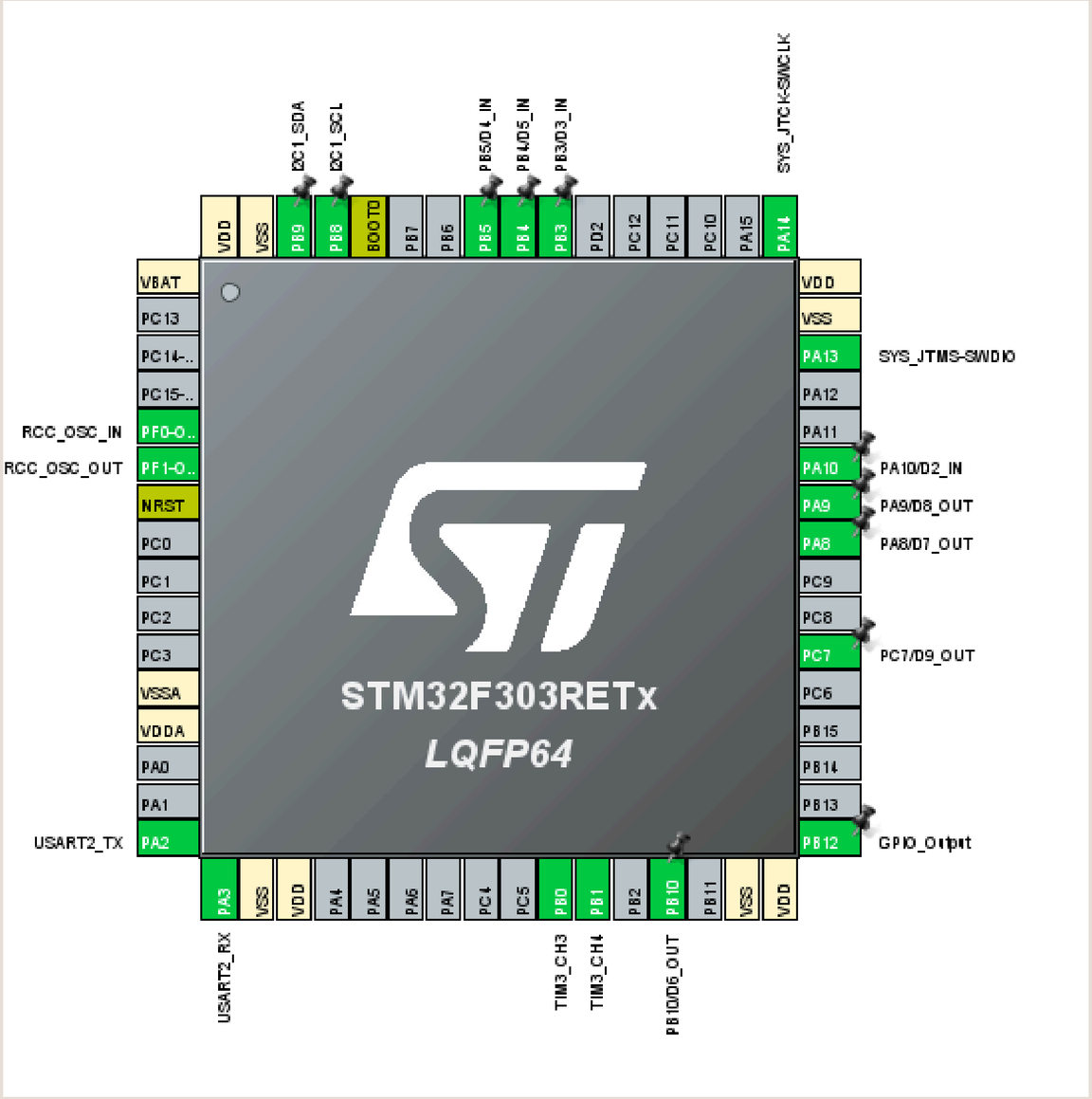
GROUP PROJECT

그룹 프로젝트 프레젠테이션

김병민 장형중
정성빈 최정우

STM32 핀 설정



IMU 센서

I2C: PB8(SCL), PB9(SDA)

USART2: PA2(TX), PA3(RX)

모터 제어를 위한 타이머 사용

TIM3: PB0(CH3), PB1(CH4)

모터 드라이버

STBY-> 모터 상태 제어: PB12

PWMA -> TIM3(CH3)

AIN1-> PA8

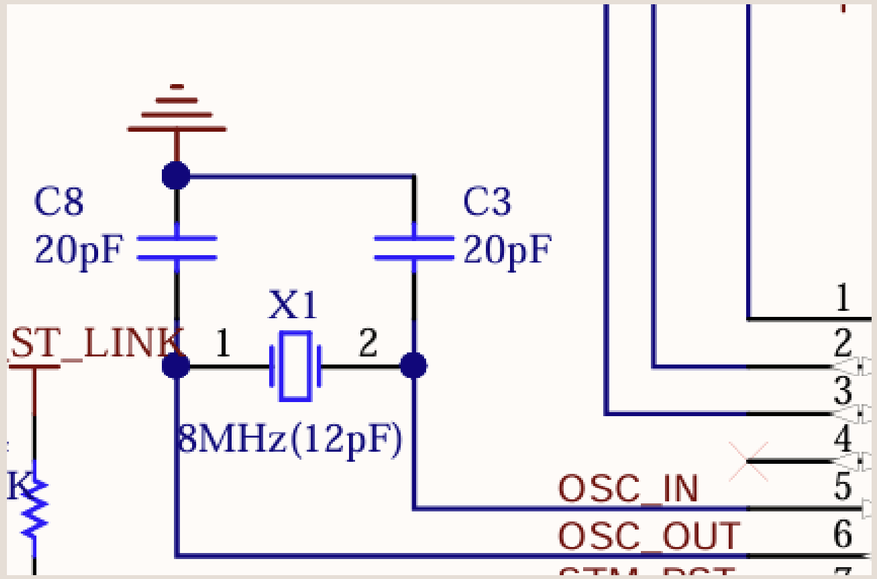
AIN2-> PB10

PWMB -> TIM3(CH4)

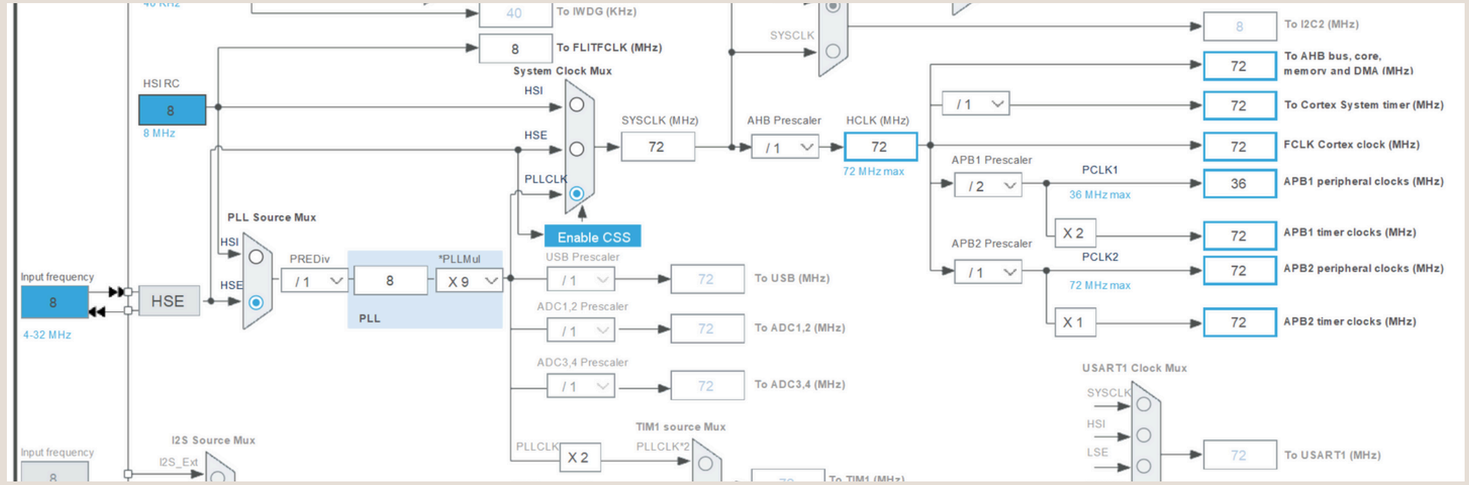
BIN1-> PA9

BIN2-> PC7

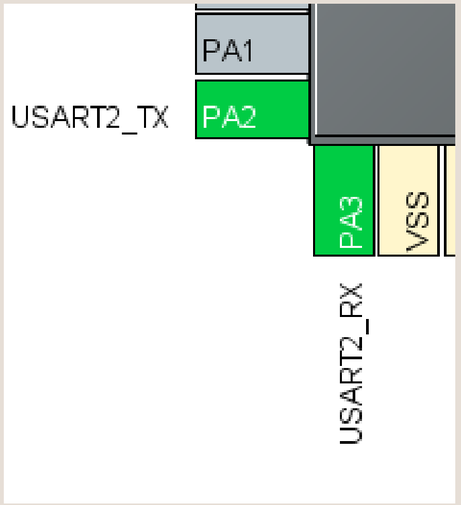
클럭 설정



클럭 속도는 데이터 시트에서 제공하는 속도와 맞지 않더라도 LED나 모터에 출력은 되지만 UART에서는 출력이 깨져서 나온다. 정확한 클럭 속도를 맞춰야 정상적으로 출력시킬 수 있다.
옆 데이터시트를 통해 속도를 8MHz로 확인하고 클럭을 설정해준다.
PLL을 최대 시스템 클럭에서 동작할 수 있도록해서 성능을 최대화 하고 안정적인 동작을 할 수 있도록 72MHz로 맞춘다.
APB1 버스는 STM32F3 시리즈에서 42MHz를 초과할 수 없고 불필요한 전력 소모를 줄이기 위해 2분주를 통해 APB1 클럭을 36MHz로 설정한다.



USART 초기 설정



데이터시트에서 USART에 대한 핀 배치도에서 PA2와 PA3에서 동작하는 것을 확인할 수 있음

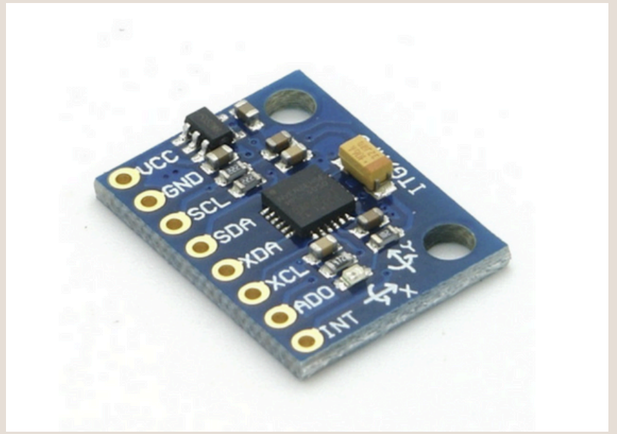
CubeIDE에서 기본 설정은 USART2 Asynchronous(비동기)로 선택하면 자동으로 PA2와 PA3이 선택된다.

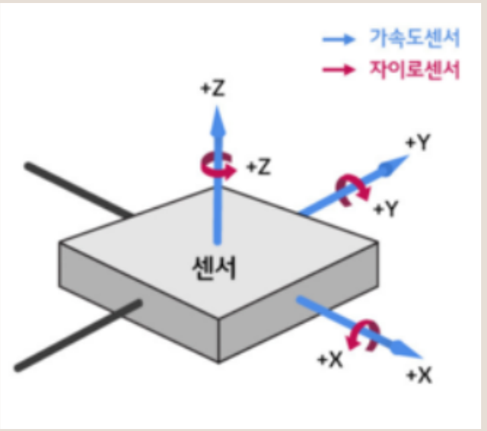
다음으로 속도를 115200으로 바꿔준다. 속도는 출력하는 터미널과 같은 속도로 맞춰야만 정상적인 출력을 할 수 있다.

SB62, SB63 (USART)	**OFF**	PA2 and PA3 on STM32 are disconnected to D1 and D0 (pin 2 and pin 1) on ARDUINO® connector CN9 and ST morpho connector CN10.
	ON	PA2 and PA3 on STM32 are connected to D1 and D0 (pin 2 and pin 1) on ARDUINO® connector CN9 and ST morpho connector CN10 as USART signals. Thus SB13 and SB14 must be OFF.
SB13, SB14 (ST-LINK-USART)	**ON**	PA2 and PA3 on STM32F103CBT6 (ST-LINK MCU) are connected to PA3 and PA2 on STM32 to have USART communication between them. Thus SB61, SB62, and SB63 must be OFF.
	OFF	PA2 and PA3 on STM32F103CBT6 (ST-LINK MCU) are disconnected to PA3 and PA2 on STM32.

MPU6050

주요 구성요소
가속도계 : 물체의 가속도 측정
자이로스코프 : 물체의 회전 속도 측정
지자기 센서 : 자기장을 이용한 측정



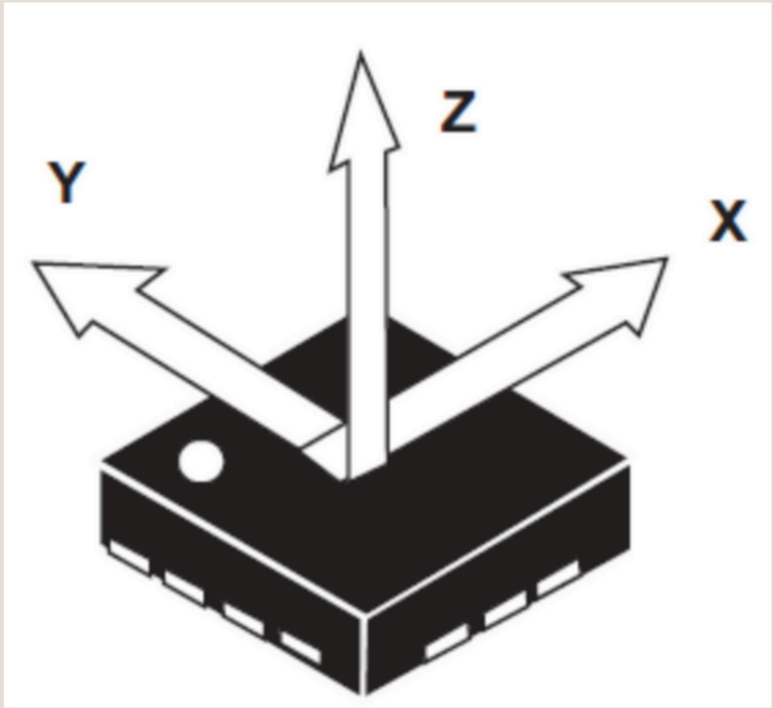


특성	내용
센서 종류	3축 가속도계 + 2축 자이로스코프 + 1축 온도
측정범위	가속도계: ±2g, ±4g, ±8g, ±16g 자이로스코프: ±250, ±500, ±1000, ±2000 dps
통신 방식	I2C
동작전압	3.3V~5V
핀 구성	8핀(VCC/GND/SCL/SDA/XDA/XCL,ADO,INT)
크기	소형(2*1.5CM)
내장 프로세서	DMP (Digital Motion Processor) 내장

- MPU-6050은 6축 자이로 가속도 센서로 하나의 칩 안에 가속도 3축, 자이로 2축, 온도 1축으로 6축 기울기 센서라고 한다.

- MPU-6050에는 지자기 센서는 포함되어 있지 않고 MPU-9050을 사용하면 3축을 추가한 9축 센서로 사용이 가능하다.

가속도



회전 각	수 식
Y축 기준 회전 각도	$AccAngle(Y) = atan(\frac{-ACCEL(X)OUT}{\sqrt{ACCEL(Y)OUT^2 + ACCEL(Z)OUT^2}}) \times (\frac{180}{\pi})$
X축 기준 회전 각도	$AccAngle(X) = atan(\frac{ACCEL(Y)OUT}{\sqrt{ACCEL(X)OUT^2 + ACCEL(Z)OUT^2}}) \times (\frac{180}{\pi})$

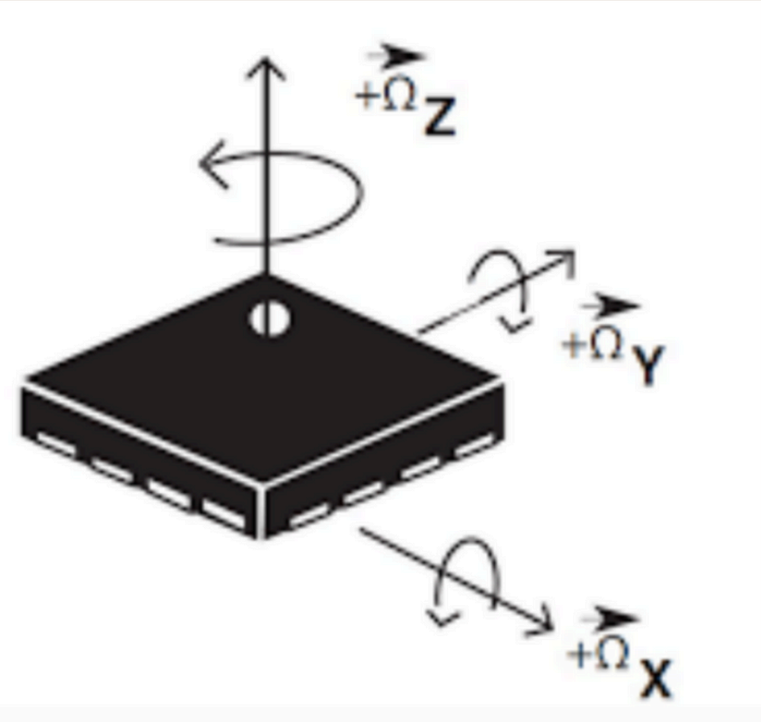
백터분해수식을 이용한 공식

가속도 : 시간에 대한 속도 변화의 비율 ->
[중력 가속도]를 이용해 가속도 측정
중력 가속도가 3축의 얼마만큼 준 영향을 측정

가속도 센서는 민감도가 매우 높아서
정적인 상태에서만 정확한 기울기를
측정 할 수 있다.

그렇기에 미세한 진동도
감지해서 떨게 되지만
긴 시간으로 평균을 계산하면
정확한 결과가 나옴

자이로



회전 각	수 식
X축 기준 회전 각도	$\mathrm{GyrAngle(X)} = \omega(X) \times \delta t$
Y축 기준 회전 각도	$\mathrm{GyrAngle(Y)} = \omega(Y) \times \delta t$
Z축 기준 회전 각도	$\mathrm{GyrAngle(Z)} = \omega(Z) \times \delta t$

순간 각속도를 측정 시간 단위로 적분 하는 공식

자이로 : 각속도를 측정, 3축의 물리량을
측정해 움직임을 감지해서 기울기를 측정한다.

순간 가속도 이므로 각도를
계산 하기 위해서는
측정 시간 단위로 적분

그렇기에 자이로 센서는
짧은 시간에 대해서 정확한 데이터를 제공
하지만, 오랜 시간이 지나면 적분에 의해
작은 오차가 쌓이게 되면서
결국 시스템이 불안정해짐

필터의 사용

필터는 센서 데이터의 노이즈를 줄이고, 더 정확한 상태 추정을 하기 위해서 사용한다.

필터를 사용하지 않고 구현을 하는 경우도 있지만 값이 빠르게 바뀔 때의 반응과 오랜기간 동작했을 때의 오차범위가 심해지기 때문에 필터를 사용하여 구현한다.
이유는 노이즈가 발생해 중심을 정확하게 잡지 못하기 때문에 정확도가 낮다.

일반적인 필터

- 이동 평균필터 : 여러 샘플을 평균내서 노이즈를 제거한다. 계산이 간단하지만 빠르게 변화하는 값에는 반응하지 못한다.
- 저역통과필터 : 고주파의 노이즈를 제거하는 것으로 유용한 신호까지 감소시킬 수 있다.

칼만 필터

칼만 필터는 노이즈가 포함된 데이터에서 최적의 추정값을 계산하는 알고리즘으로 노이즈가 있을 때에도 센서 값의 변화를 부드럽게 필터링하면서 상태를 예측한다.

- 예측과 업데이트를 반복하면서 상태 보정
- 예측하는 단계에서는 이전 상태인 Xk-1과 시스템 모델을 이용하여 현재 상태인 Xk를 예측한다.
- 업데이트 단계에서는 칼만 이득을 통해 측정값과 예측값을 최적의 비율로 결합하여 최종적으로 노이즈가 제거된 상태 Xk를 결정한다.

칼만 필터의 예측 수식

- 상태 예측 : $x-_k = A_{x_{k-1}} + B_{u_k}$

Xk = 예측된 상태

A = 시스템 모델(상태 전이 행렬)

B = 입력 모델(제어 행렬)

Uk = 제어 입력(모터 전류)

-오차 공분산 예측 : $P_k^- = AP_{k-1}A^T + Q$

Pk = 예측된 오차 공분산(시스템의 불확실성)

Q = 프로세스 노이즈 (모델의 불확실성)

현재 상태를 바탕으로 다음 값을 예측

상태 예측의 불확실성을 나타내는 오차 공분산을 예측

센서가 실시간으로 토크 zk를 측정하고 칼만 필터를 통해 현재 상태 xk를 갱신한다. 과거의 데이터와 비교하여 노이즈를 제거한 최적의 토크 값을 계산하여 정밀하게 동작한다.

칼만 필터의 업데이트 계산

칼만 이득 계산 : $K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$

H = 측정 행렬 (센서 모델)

R = 측정 노이즈 공분산 (센서의 불확실성)

Kk = 칼만 이득 (예측값과 측정값을 결합하는 가중치)

칼만 이득은 예측된 상태와 측정값을 결합할 때, 예측값과 실제 측정값 중 어느 쪽을 더 신뢰할지를 결정하는 계수

상태 업데이트 : $x_k = x_k^- + K_k(z_k - Hx_k^-)$

zk = 센서 측정값

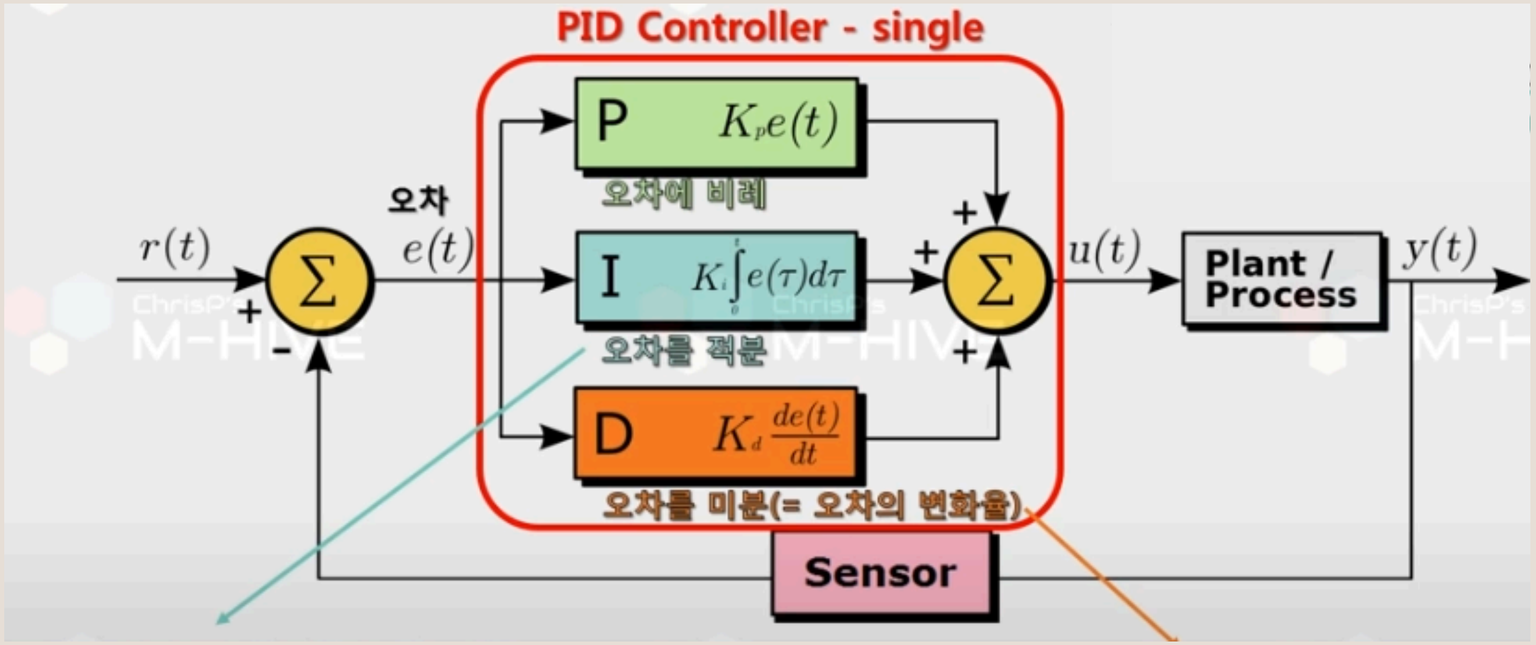
xk = 최종 보정된 상태

오차 공분산 업데이트 : $P_k = (I - K_k H)P_k^-$

상태 업데이트는 예측된 상태에 칼만 이득을 곱하고, 실제 측정값을 반영하여 상태 추정치를 수정한다.

오차 공분산 업데이트는 상태 추정의 정확도를 업데이트한다. 이 값은 상태 추정이 얼마나 정확한지를 나타내며, 새로운 측정값을 반영하여 오차 공분산을 갱신한다.

PID



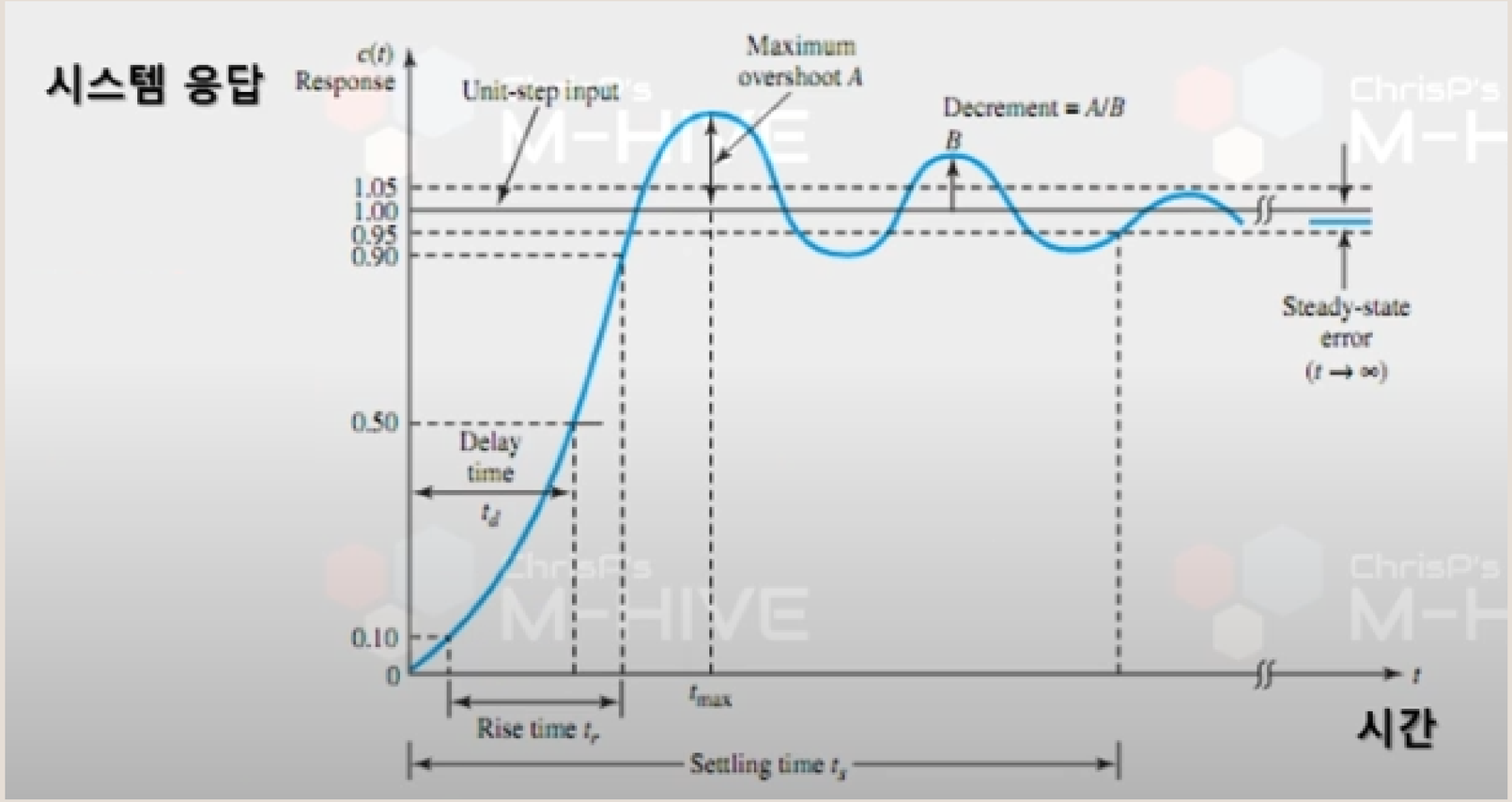
Prorportional(비례): 오차의 크기에 비례하여 제어량을 결정 -> 목표에 수렴하는 속도결정

Integral(적분): 목표에 완전히 수렴하지 못하는 미세 오차를 제거

Derivative(미분): 오차의 변화율에 따라 제동을 걸어주는 역할

목표r(t) - 현재값(sensor) = 오차e(t)로 정의

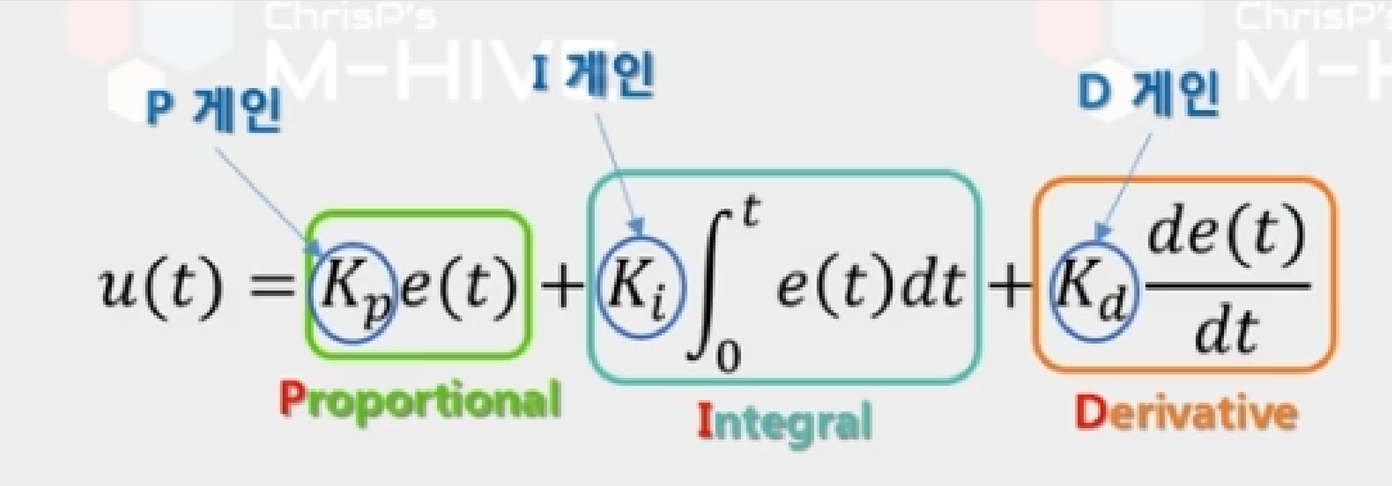
PID응답 그래프



0 부터 시작하여 1의 수렴하는 과정

시간에 따른 시스템 응답 특성 나타냄

PID



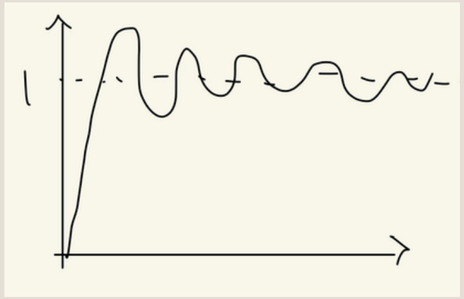
gain: 각각의 항의 계산된 결과에 얼마만큼 곱해주는 상수

각각의 계산들이 pid성능에 큰 영향을 주어 정확한 계산 필요로함

PID

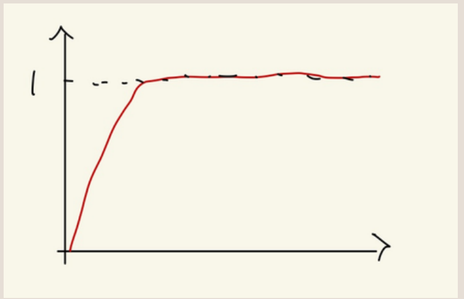
$K_p e(t)$

P term



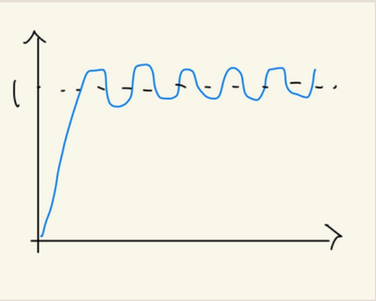
$K_i \int_0^t e(t)dt$

I term



$K_d \frac{de(t)}{dt}$

D term



오차를 줄여 목표가 빠르게 수렴하도록 하는 역할

빠른 동작위해 kp를 높일때 over shoot 발생

오차의 크기를 누적하여 제어량을 반영
미세오차를 줄여가며 목표에 점진적 수렴하게 함

상태의 변화율의 비례하여 제어량을 결정

상태가 급격히 변하는 것을 막음

코드

Main.c	메인 문 – 센서에 값을 읽어오고 모터를 제어
mpu6050.c	IMU센서 – 가속도와 자이로 값을 읽어서 칼만 필터를 사용해서 현재 각도를 측정
pid.c	pid제어 – 읽어온 각도 값을 초기 값이랑 비교해서 유지
state_machine.c	모터 방향 제어 – 현재 각도 값에 양수 음수를 읽어서 전진,후진 방향 제어

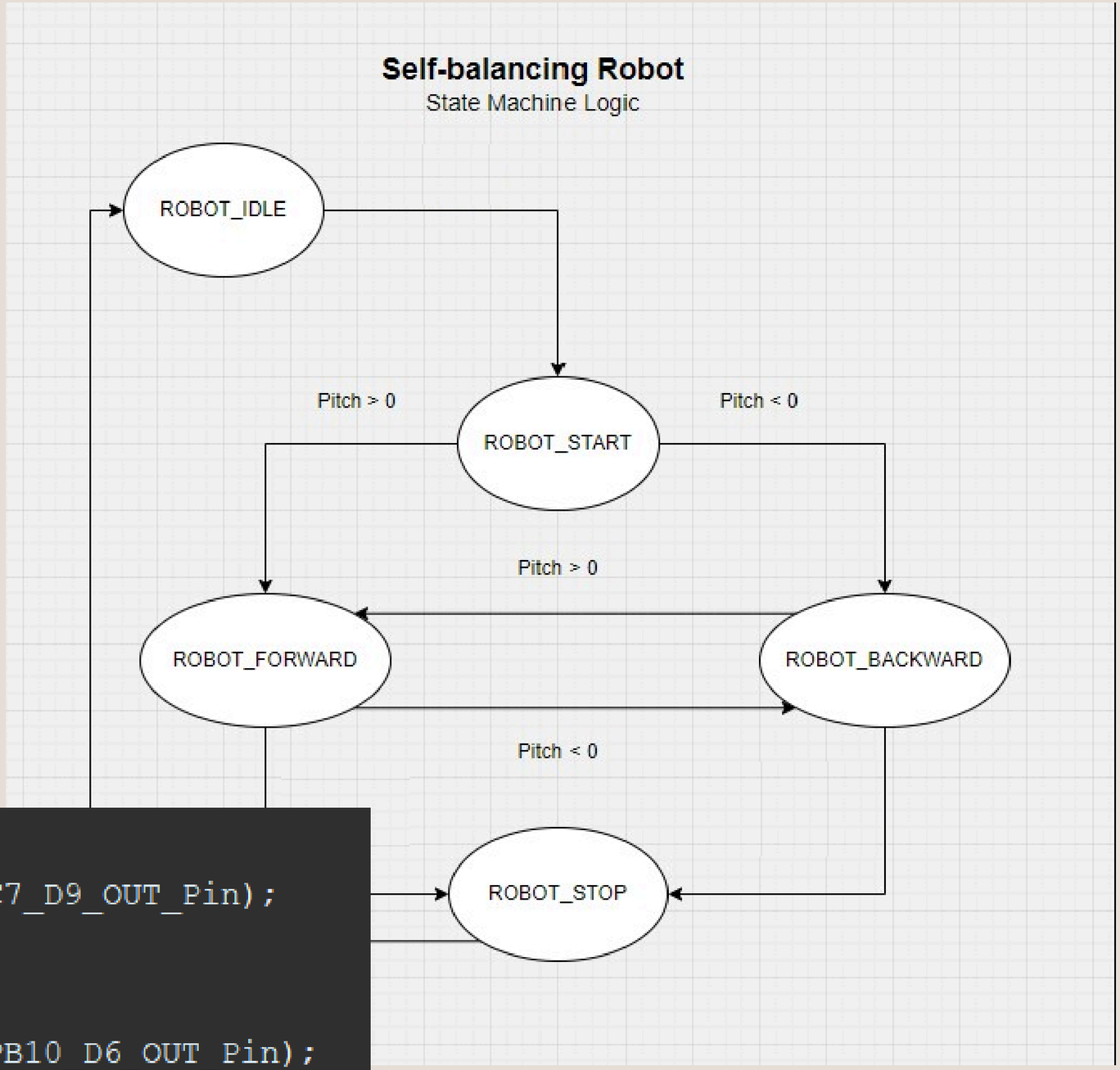
코드 설명

State_machine.c

로봇은 여러 개의 상태를 가지고 있으며, 특정 조건에 따라 상태가 변화.

로봇의 상태를 계속 피드백하여,
로봇의 기울기를 지속적으로 조정하고,
여기서의 Pitch값은, 로봇의 기울어진 정도를 뜻하며, 로봇이 정확히 직각으로 서있다면, 0도로, Robot_Stop부분에서 머무름.

상태 변화는 Pitch값을 가 값에 의해 결정됨




```
// Change direction of Left Motor
HAL_GPIO_TogglePin(GPIOA, PA9_D8_OUT_Pin);
HAL_GPIO_TogglePin(PC7_D9_OUT_GPIO_Port, PC7_D9_OUT_Pin);

// Change direction of Right Motor
HAL_GPIO_TogglePin(GPIOA, PA8_D7_OUT_Pin);
HAL_GPIO_TogglePin(PB10_D6_OUT_GPIO_Port, PB10_D6_OUT_Pin);
```

코드 설명

State_machine.c

```
Robot_State robot_idle_state
{
    __HAL_TIM_SET_COMPARE(ti
    __HAL_TIM_SET_COMPARE(ti
    return ROBOT_IDLE;
}

/* START state determines wh
   based on the pitch angle.
Robot_State robot_start_stat
{
    if((accel->pitch_angle)
    {
        return ROBOT_BACKWAR
    }
    else if((accel->pitch_an
    {
        return ROBOT_FORWARD
    }
    else
    {
        return ROBOT_START;
    }
}
```

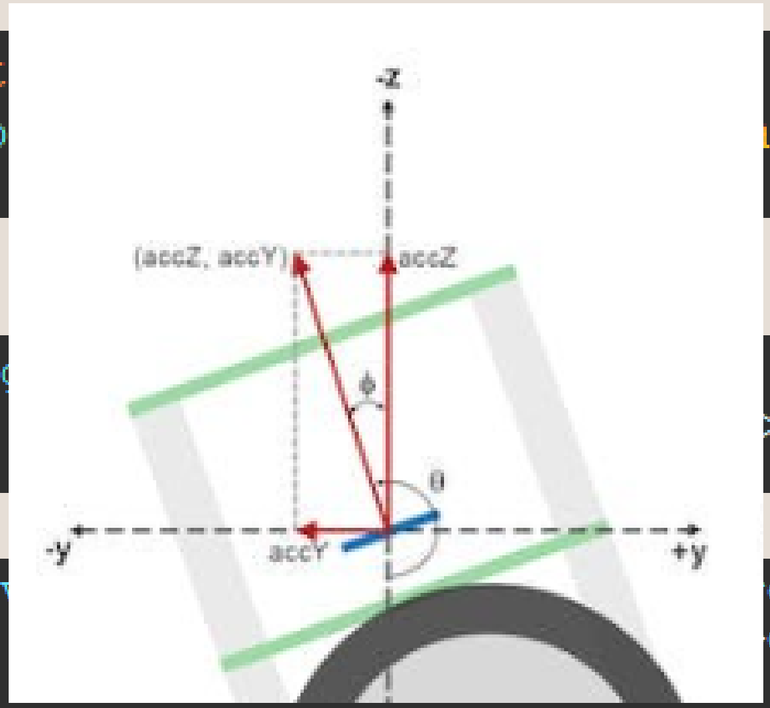
```
Robot_State robot_backward_state(accel_data * accel, int * motor_toggle)
{
    if(accel->pitch_angle < 0 && *(motor_toggle) == 1)
    {
        // Change direction of Left Motor
        HAL_GPIO_TogglePin(GPIOA, PA9_D8_OUT_Pin);
        HAL_GPIO_TogglePin(PC7_D9_OUT_GPIO_Port, PC7_D9_OUT_Pin);

        // Change direction of Right Motor
        HAL_GPIO_TogglePin(GPIOA, PA8_D7_OUT_Pin);
        HAL_GPIO_TogglePin(PB10_D6_OUT_GPIO_Port, PB10_D6_OUT_Pin);

        *(motor_toggle) = 0;
    }

    /* Transition to ROBOT_FORWARD state if pitch angle becomes positive
       This means the robot is leaning forwards now. */
    if(accel->pitch_angle > 0)
    {
        return ROBOT_FORWARD;
    }
    else
    {
        return ROBOT_BACKWARD;
    }
}
```

코드 설명 PID.c

```
// Error from current IMU reading and set pitch an
float error = fabs(setpoint - imu_reading);
```

error 값은, IMU센서로부터 읽은 각도 값과
목표로 하는 목표 값의 차이로 error값을 표현한다.

```
float proport                                    error;
controller->p                                    l;
```



P는 error값에 비례하는 출력을 생성하며, 이를 Main.c
에서 PID Controller라는 코드에서 kp(비례 게인)과 곱
하여 계산한다.

```
controller->inte
float integral =                                 >integral;
```

I는 오차의 누적 값을 더한 값을 사용하여 계산하고,
이 또한 PID Controller 에서의 ki(적분 게인)과 곱하여
적분 값을 계산한다.

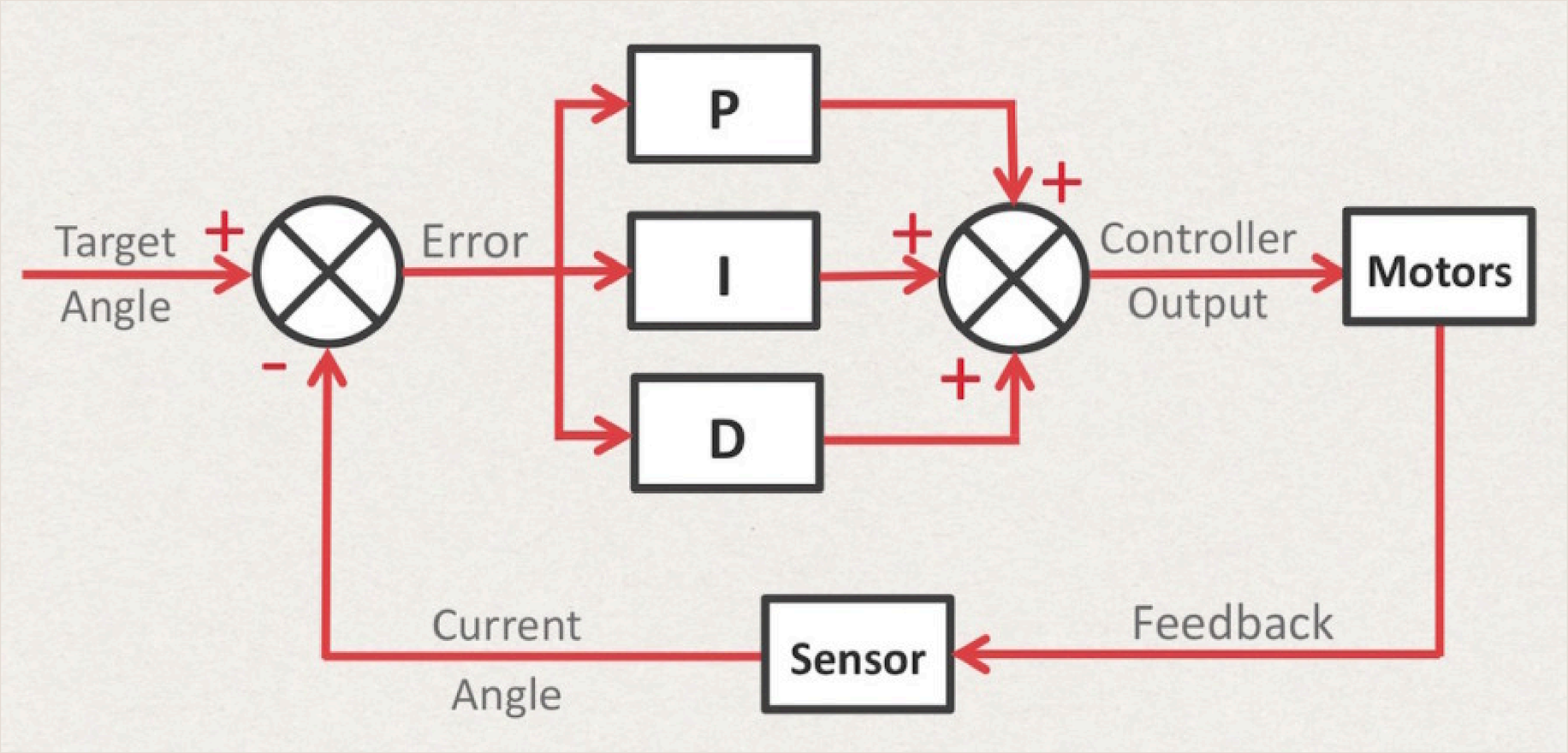
```
controller->deriv                                ev_error)/((controller->sampling_time)/1000.0f);
float derivative                                 derivative;

float motor_output = fabs(proportional + derivative);
```

D는 오차의 변화율을 사용하여 계산함.
prev_error(이전 오차)와 현재 error의 차이를 시간에 대해 나누어 변화율을 구하며,
이 또한 Controller의 kd(미분 게인)과 곱하여 미분 항을 계산함.

코드 설명 pid.c

```
float motor_output = fabs(proportional + derivative);
```

로봇에서는 적분 항을 잘못 쓰면 불안정성이 커질 수 있고,
작은 오차가 계속 누적되면 너무 큰 출력을 만들어서 로봇이 오히려 더 흔들림이 있음.



코드 설명 main.c

```
switch(state)
{
  case ROBOT_IDLE:
    state = ROBOT_START;
    break;
  case ROBOT_START:
    // Determine which state to enter next
    state = robot_start_state(&accel);
    break;
  case ROBOT_FORWARD:
    // Move the robot backward
    state = robot_forward_state(&accel, &toggle);
    break;
  case ROBOT_BACKWARD:
    // Move the robot forward
    state = robot_backward_state(&accel, &toggle);
    break;
  case ROBOT_STOP:
    // Stop all motors
    state = robot_stopped_state(&htim3);
    break;
  case ROBOT_BALANCED:
    break;
  default:
    break;
}
```

```
const float set_pitch = 6.0f; //피치
/* Set up PID Controller and gain c
PIDController controller;

controller.kp = 42.0f;
controller.ki = 30.0f;
controller.kd = 0.5f;
controller.sampling_time = 500;
PIDController_Init(&controller);
```

Zigler – Nichols

Step 1: 적분항(Ki)과 미분항(Kd)을 0으로 설정

Step 2: Kp(비례 이득)를 증가시키면서 진동 확인

Step 3: 진동의 한계 이득(Ku)과 주기(Tu) 측정

Step 4: Ziegler-Nichols 튜닝 공식 적용

감사합니다.