PID제어와 튜닝준비

An **[Unmanned aerial vehicle]** (UAV) is a Unmanned Aerial Vehicle. UAVs include both autonomous (means they can do it alone) [drones] and [remotely piloted vehicles] (RPVs). A UAV is capable of controlled, sustained level flight and is powered by a jet, reciprocating, or electric engine.

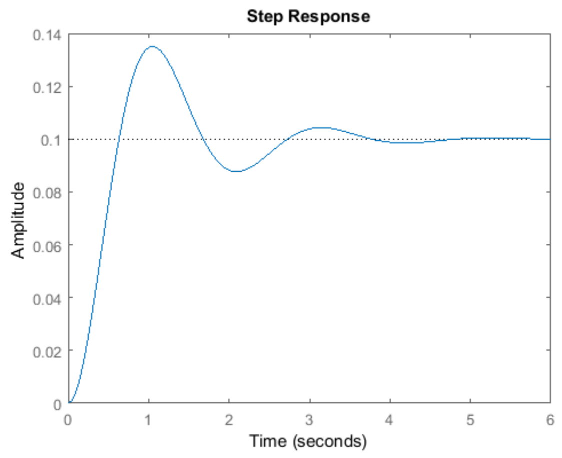
PID 제어기초

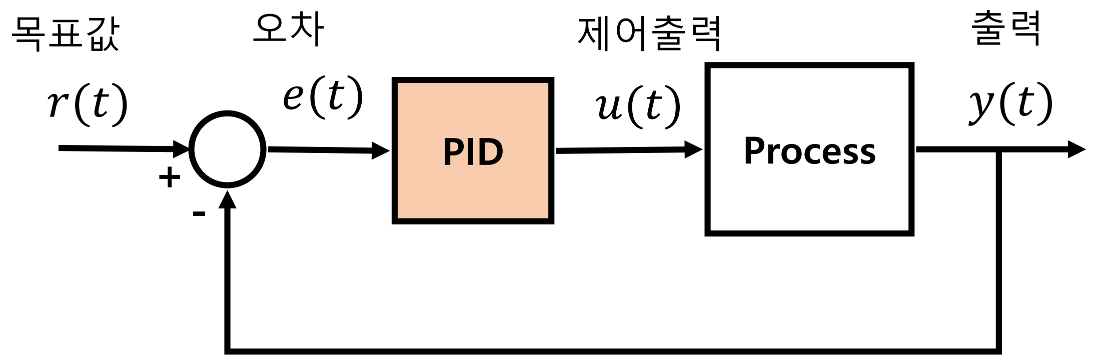
PID 제어란?

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

PID: Proportional, Integral, Derivative Control

- **비례·적분·미분제어**
  - 오차에 비례, 미분, 적분을 적용하여 원하는 목표를 달성하게 하는 제어기
- **제어목표**:
  - 프로세스의 출력이 원하는 시간 이내에 안정적으로 목표값에 도달하도록 함.

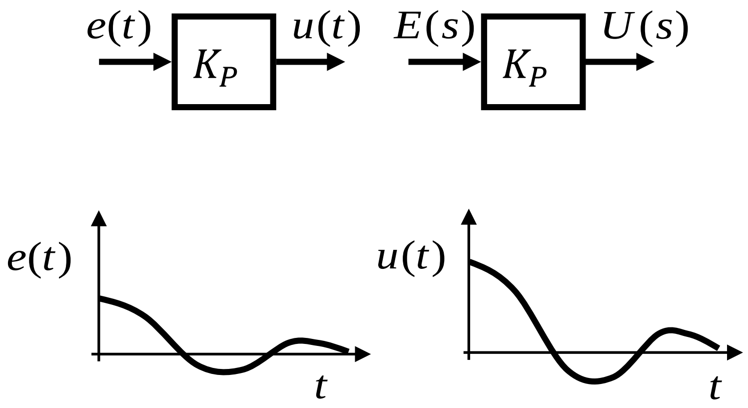




PID의 요소

■ 비례 제어(P: Proportional Control)

- 기본식
  - $u(t) = K_P e(t)$
  - K_P: 비례 게인(proportional gain)

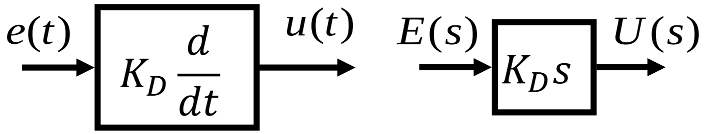


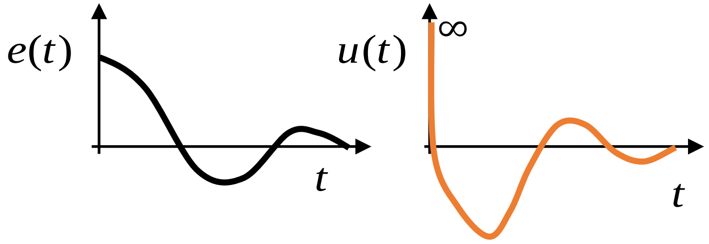
- 오차에 비례하는 출력을 내는 가장 직관적 제어기

PID의 요소

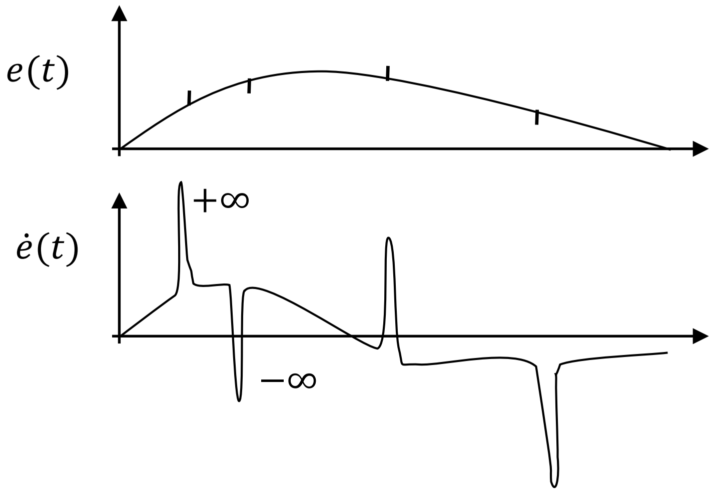
■ 미분 제어 (D: Derivative Control)

- 기본식
  - $u(t) = K_D \dfrac{de(t)}{dt}$
  - K_D : 미분게인





- 오차의 변화율에 비례하는 출력을 내는 제어기로서 댐핑을 증가시킴.
- 미분연산에 의하여 잡음에 민감하여 미분 게인을 너무 키우면 출력에 진동 발생

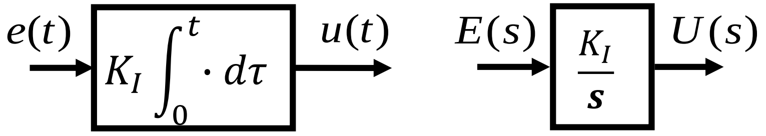


PID의 요소

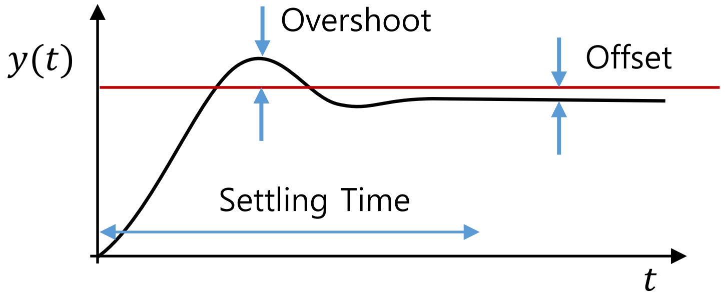
Dept. of Mechanical System Design, Seoul National University of Science and Technology.

■ 적분 제어 (I: Integral Control)

- 기본식
  - $u(t) = K_I \int_0^t e(\tau)d\tau$
  - K_I : 적분게인



- 오차의 누적값에 비례하는 값을 출력 ➔ 정상상태 오차(Offset) 감소 효과.
- 적분 게인이 너무 크면
  - 분모에 적분기를 포함하는경우 시스템이 불안해질 수 있음.
  - Overshoot가 커짐
  - 정착시간이 길어짐



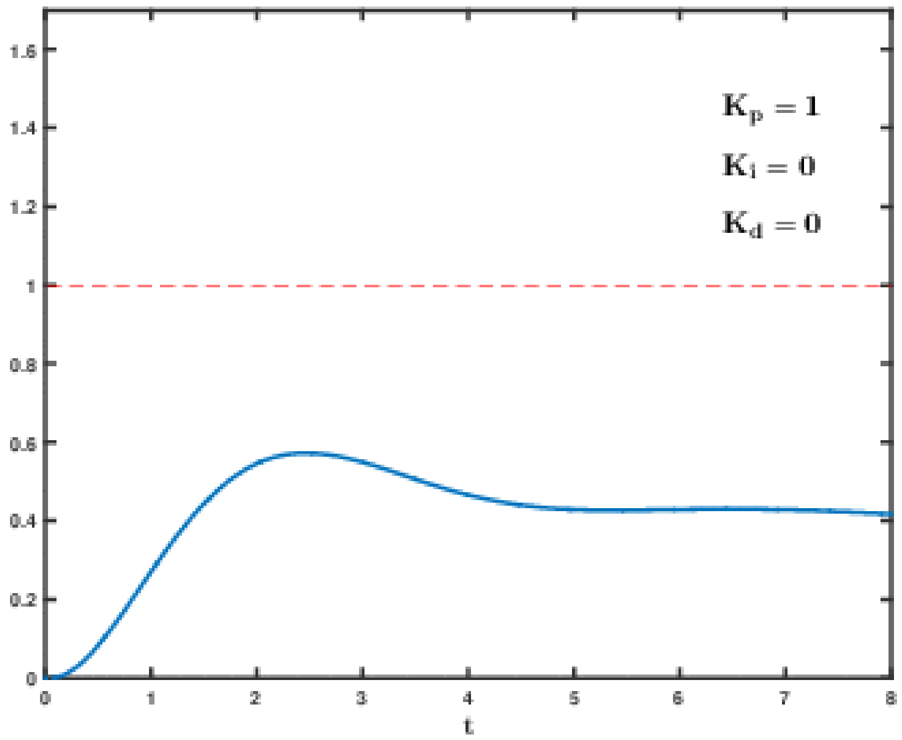
적분기의 Offset 감소효과
정상상태에 약간의 오차라도 남으면 적분 제어에 의하여 그 오차의 누적값이 계속 증가되므로 언젠가는 오차가 없어짐.

PID 게인의 역할

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

각 **게인의 역할**

- 각PID 제어 항에 대한 가중
- P - 기본제어
- I - 정상상태 오차 저감
- D - 과도기 진동 저감, 댐핑 증가.



https://commons.wikimedia.org/wiki/File:PID_Compensation_Animated.gif

PID의 요소

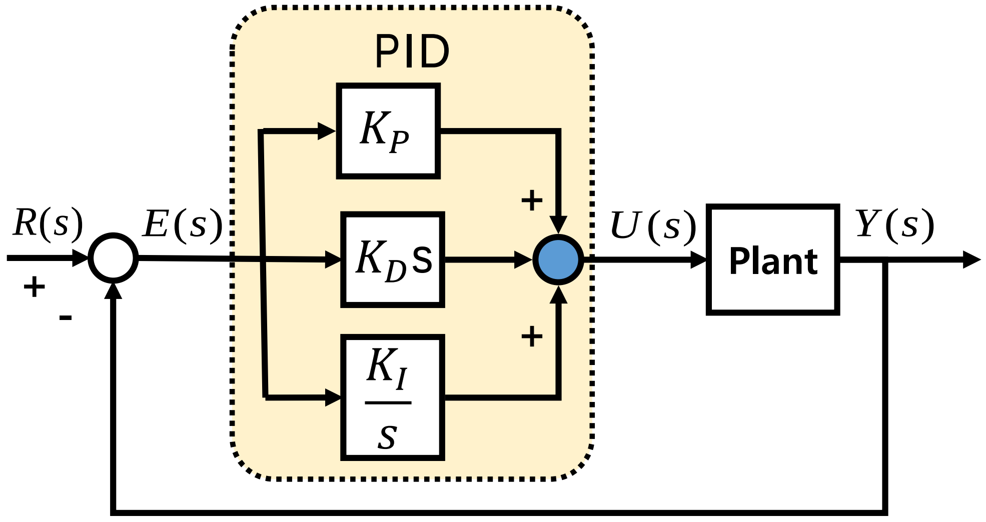
Dept. of Mechanical System Design, Seoul National University of Science and Technology.

■ 비례 미분 적분 제어 (PID Control)

- 기본식

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt} \qquad (1)$$

$$\frac{U(s)}{E(S)} = K_P + \frac{K_I}{s} + K_D s$$



- 비례, 미분, 적분 제어기를 모두 조합하여 각각의 장점을 취한 제어기
- 시행착오에 의하여 비례, 미분, 적분게인을 설정하는 것이 일반적

Digital PID Control

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

■ 디지털 비례 미분 적분 제어

- 샘플링 시간을 dt 라고 하고, 적분근사화과 오일러의 근사식을 (1)식에 적용

$$u(k) = \underbrace{K_P e(k)}_{\text{P-term}} + \underbrace{K_I \sum_{j=0}^{k} e(j)dt}_{\text{I-term}} + \underbrace{K_D \frac{e(k)-e(k-1)}{dt}}_{\text{D-term}} \qquad (2)$$

여기서, k는 샘플링 인덱스.

적분 근사화 $(dt \ll 1)$

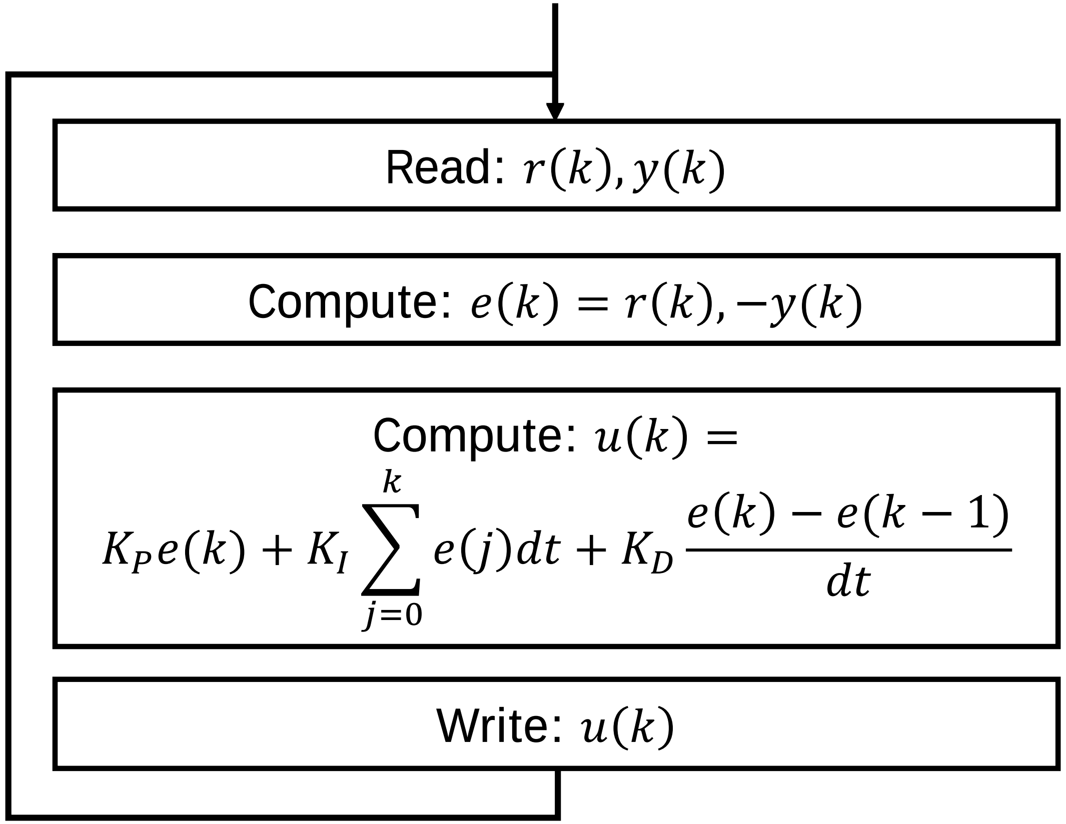
$$\int_0^t e(\tau)d\tau \cong \sum_{j=0}^{k} e(j)dt$$

오일러 근사화 $(dt \ll 1)$

$$\dot{x}(k) \cong \frac{x(k) - x(k-1)}{dt}$$

Digital PID Control

- 일반적 Digital PID의 순서도

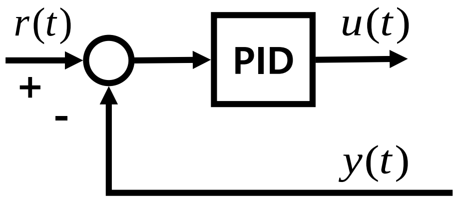


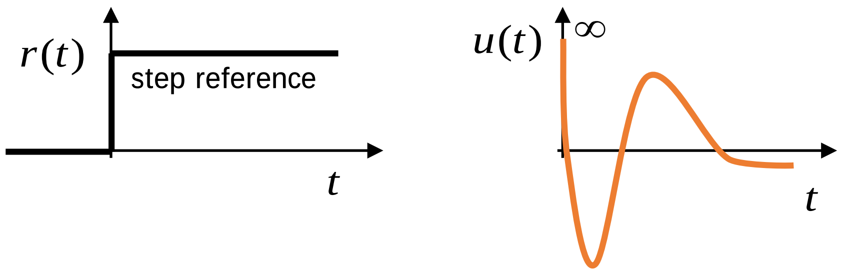
PID 제어 알고리즘 개선

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

Standard PID 문제점

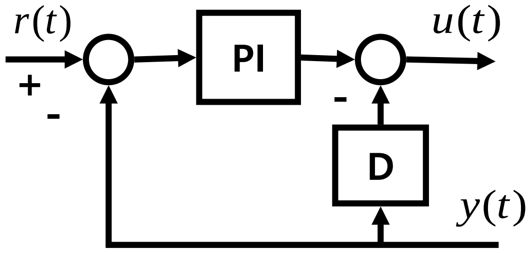
- $e(t) = r(t) - y(t)$ 이기 때문에 $r(t)$가 계단 입력인 경우 오차를 미분하면 출력 $u(t)$가 ∞
- 큰 오버슈트(overshoor) 발생





속도 feedback PID

- 미분제어기에서는 $e(t)$ 대신 $-y(t)$ 사용
- 계산식
  - $u(k) = K_P e(k) + K_I \sum_{j=0}^{k} e(j)dt - K_D \frac{y(k)-y(k-1)}{dt}$

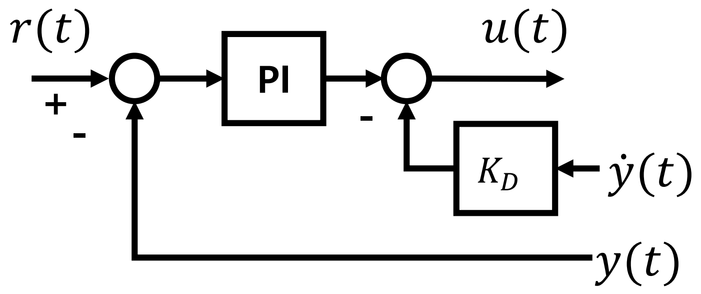


PID 제어 알고리즘 개선

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

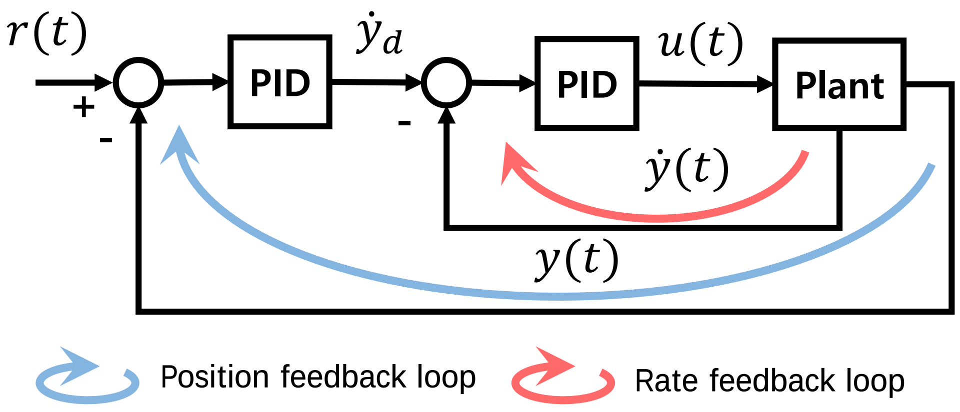
속도센서 feedback PID

- 속도 센서가 있는 경우



Dual loop PID

- 속도 센서 피드백의 속도 loop를 추가
- rate 와 position loop PID가 존재
  - rate loop는 position에 비하여 2배 이상 빨라야 좋은 성능 보장



PID 제어 알고리즘 개선

LPF가 적용된 D제어기

- 급격하게 변화하는 오차를 미분함으로써 발생되는 문제를 완화
- 1차 LPF의 전달함수

$$LPF(s) = \frac{Y(s)}{X(s)} = \frac{1}{Ts+1}$$

- 미분방정식

$$T\dot{y}(t) + y(t) = x(t)$$

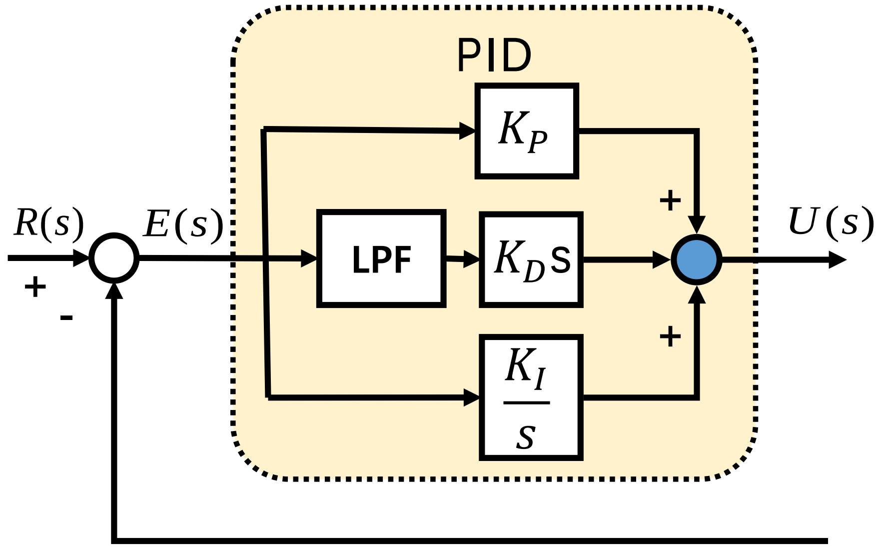
- 디지털 구현
  - 샘플시간이 dt일 때 오일러 근사식 적용

$$T\frac{y(k) - y(k-1)}{dt} + y(k) = x(k)$$

- 정리하면

$$y(k) = \alpha y(k-1) + (1-\alpha)x(k)$$

여기서, $\alpha = \frac{T}{T+dt}$ 이고



LPF에서 컷오프 주파수 f

$$f = \frac{1}{2\pi T}$$

PID 제어 알고리즘 개선

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- 미분기와 LPF를 포함한 전달함수

$$LPF(s) = \frac{Y(s)}{X(s)} = \frac{s}{Ts+1}$$

- 미분방정식

$$T\dot{y}(t) + y(t) = \dot{x}(t)$$

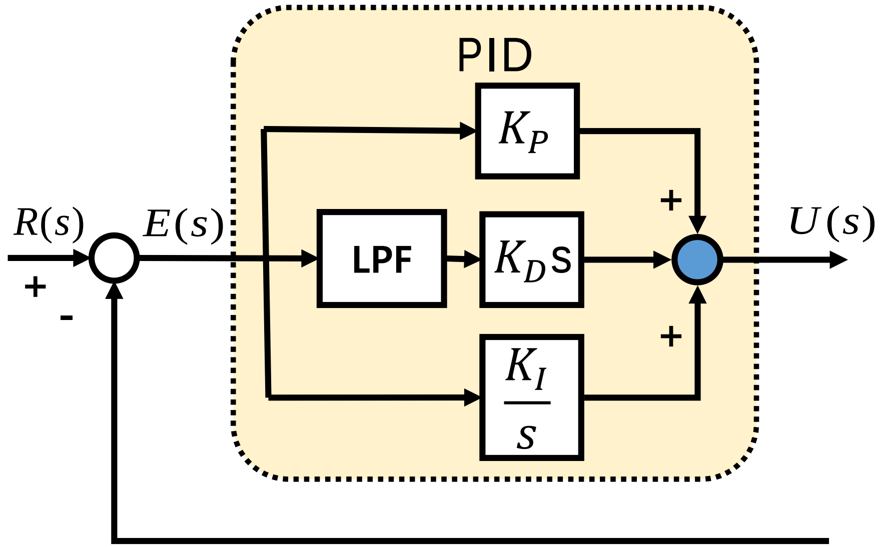
- 디지털구현
  - 샘플시간이 dt일 때 오일러 근사식 적용

$$T\frac{y(k)-y(k-1)}{dt} + y(k) = \frac{x(k)-x(k-1)}{dt}$$

- 정리하면

$$z(k) = \frac{x(k)-x(k-1)}{dt}$$

$$y(k) = y(k-1) + \frac{dt}{T+dt}(z(k)-y(k-1))$$



LPF에서 컷오프 주파수 f

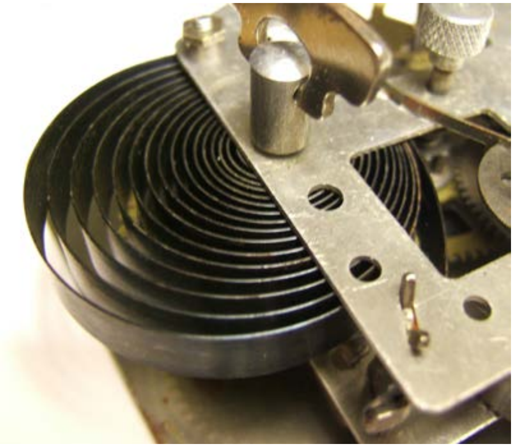
$$f = \frac{1}{2\pi T}$$

PID 제어 알고리즘 개선

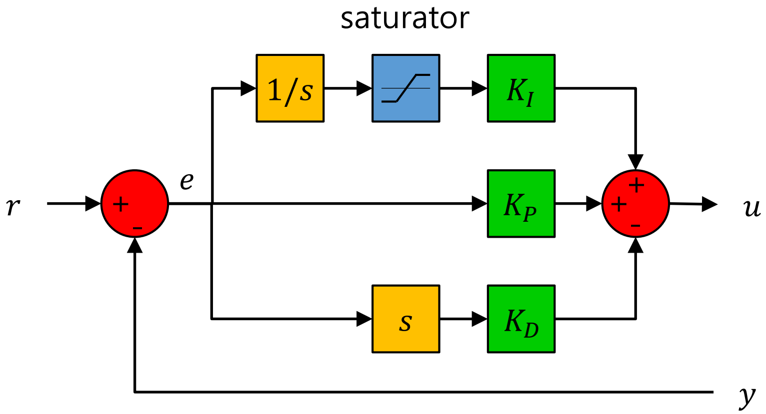
Dept. of Mechanical System Design, Seoul National University of Science and Technology.

I 제어기의 문제점

- windup 현상
  - 작동기가 제어 한계나 외력에 의하여 제어 출력의 변화가 반영이 되지 않을 때 나타남.
  - 적분 값이 매우 큰 값으로 누적되어 원인이 제거되어도
    - 매우 큰 출력이 한꺼번에 반영됨
    - 그 값이 오랫동안 제어 성능을 저하시킴
- 대책
  - Anti-windup 알고리즘 적용
  - 적분값을 상한/하한으로 포화(saturate)시킴



Wikimedia Commons: Alarm clock mainspring.JPG



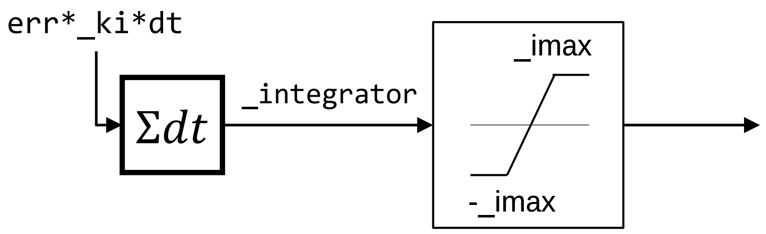
P, I - Control

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- get_p: 비례제어

```
int32_t get_p(int32_t error) {
    return (float)error * _kp;
}
```

- get_i: 적분제어




```
int32_t get_i(int32_t error, float dt)
{
    if((_ki != 0) && (dt != 0)){
        _integrator += ((float)error * _ki) * dt;
        if (_integrator < -_imax) {
            _integrator = -_imax;
        } else if (_integrator > _imax) {
            _integrator = _imax;
        }
        return _integrator;
    }
    return 0;
}
```

D - Control

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

get_d: 미분 제어

- 다음 식을 구현

$$z(k) = \frac{x(k)-x(k-1)}{dt}$$

$$y(k) = y(k-1) + \frac{dt}{T+dt}(z(k) - y(k-1))$$

_d :	$z(k)$
input :	$x(k)$
_last_input:	$x(k-1)$
_deriv:	$y(k)$
_last_deriv:	$y(k-1)$

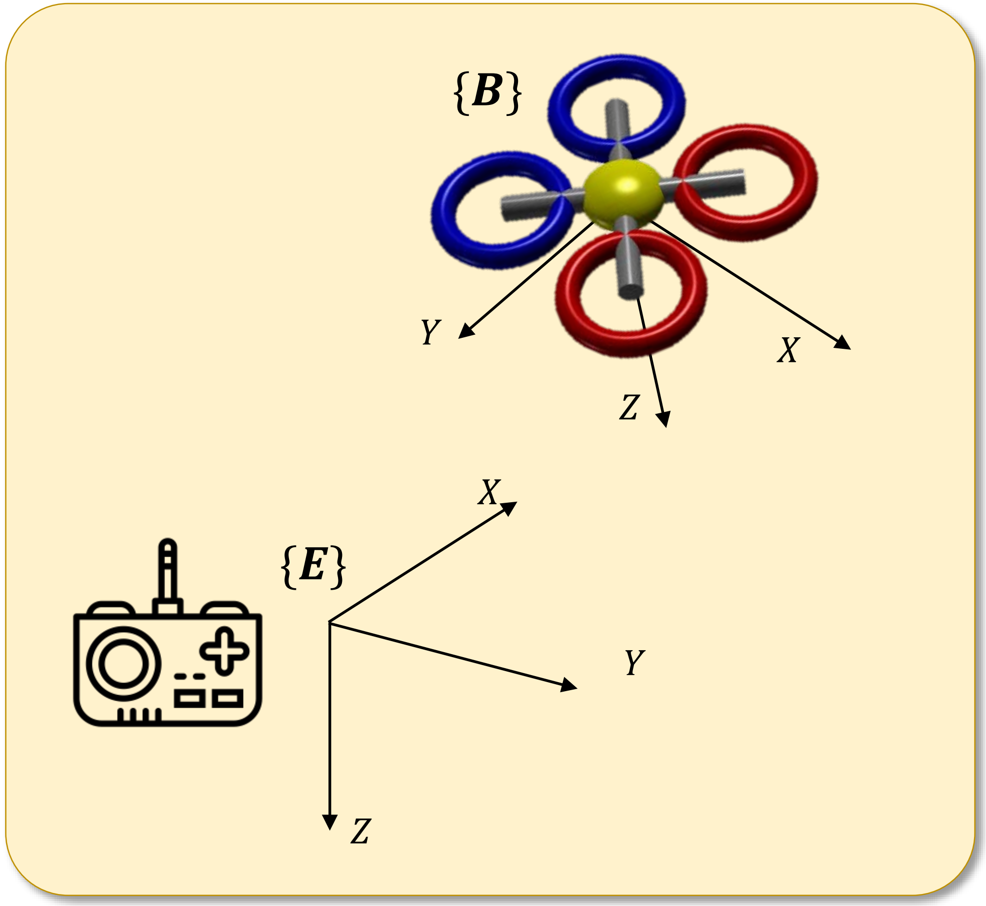
```
int32_t get_d(int32_t input, float dt){
    if ((_kd != 0) && (dt != 0)) {
         float _d = (input - _last_input) / dt;
        // discrete low pass filter
         _deriv = _last_deriv +
         (dt/(_filter + dt))*(_d-_last_deriv);
        // update state
        _last_input        = input;
        _last_deriv = _deriv;

        // add in derivative component
        return _kd * _deriv;
    }
    return 0;
}
```

PID 적용

조종의 {E}, {B}좌표계

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

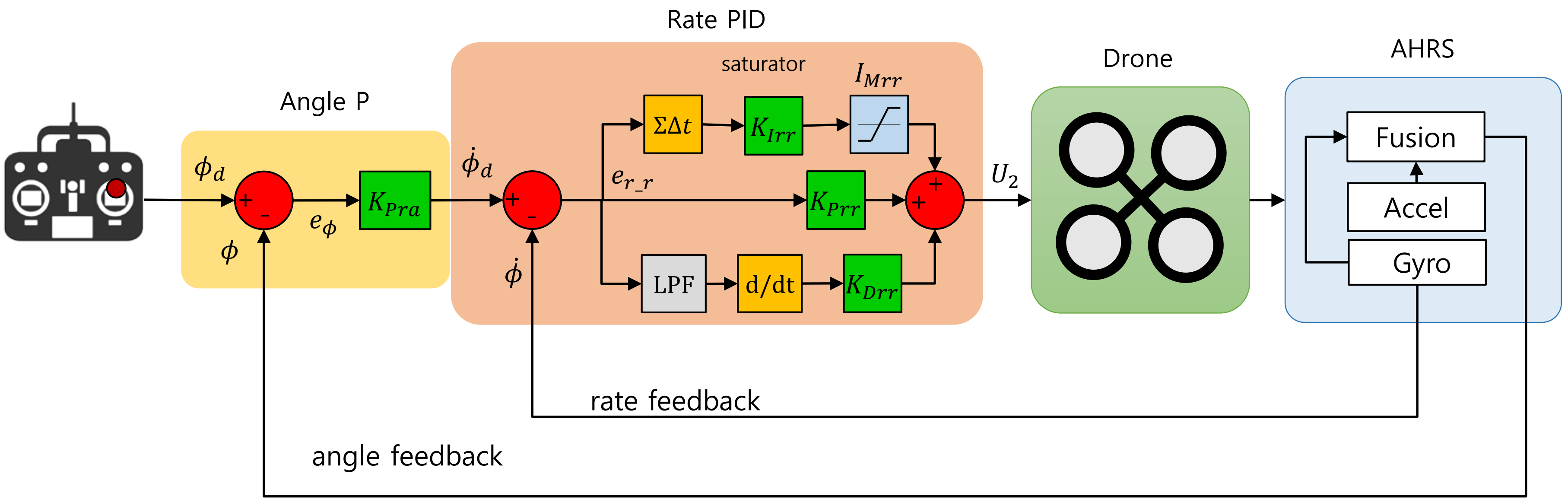
- 고도 제어
  - throttle을 올리면 {E} 좌표의 -Z축으로
- 자세제어
  - {E} 좌표를 기준으로 각도 판단



이중 루프 PID - Roll

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

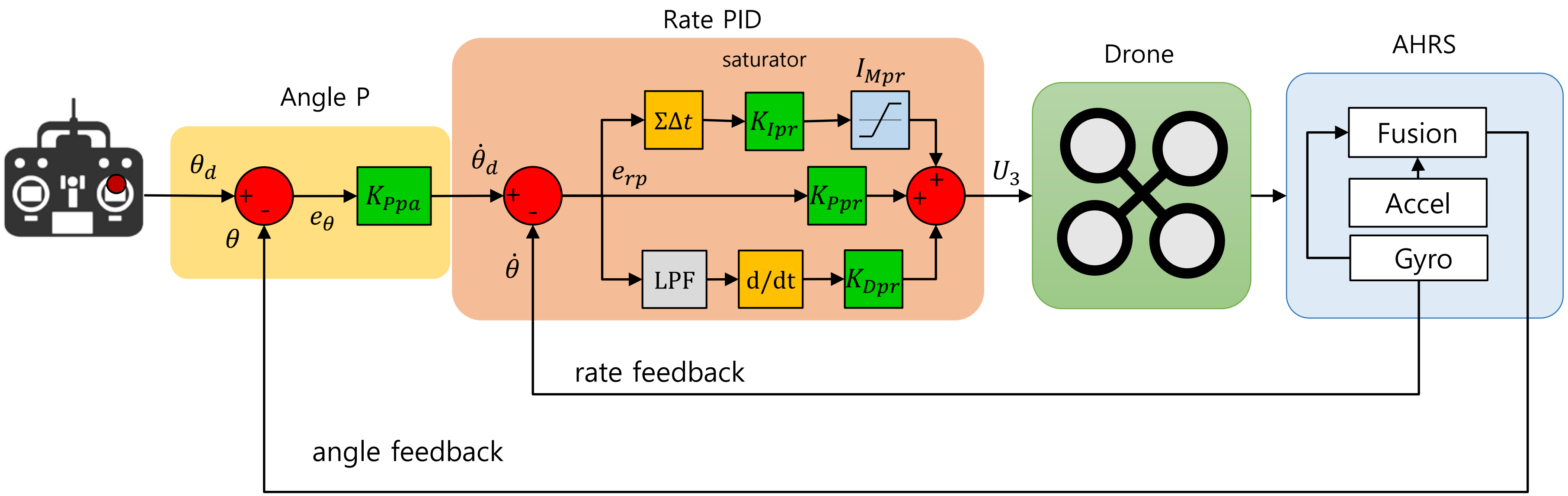
■ Roll 각도 제어



이중 루프 PID - Pitch

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

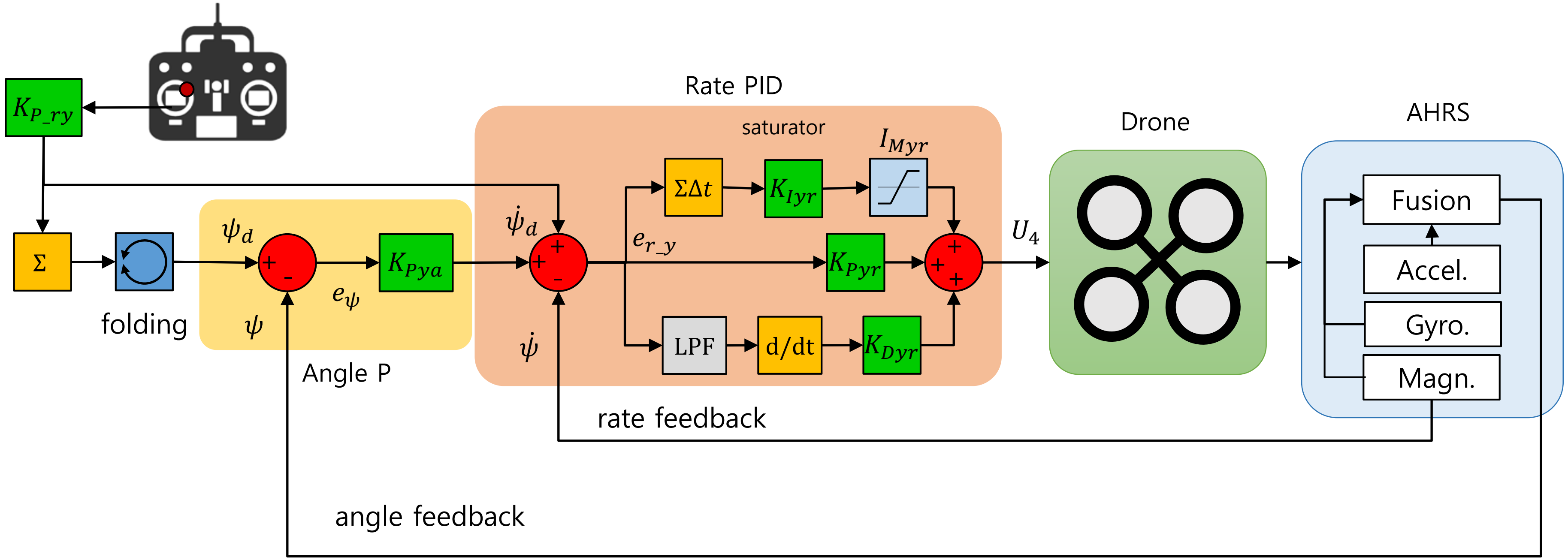
Pitch 각도 제어



이중 루프 PID - Yaw

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

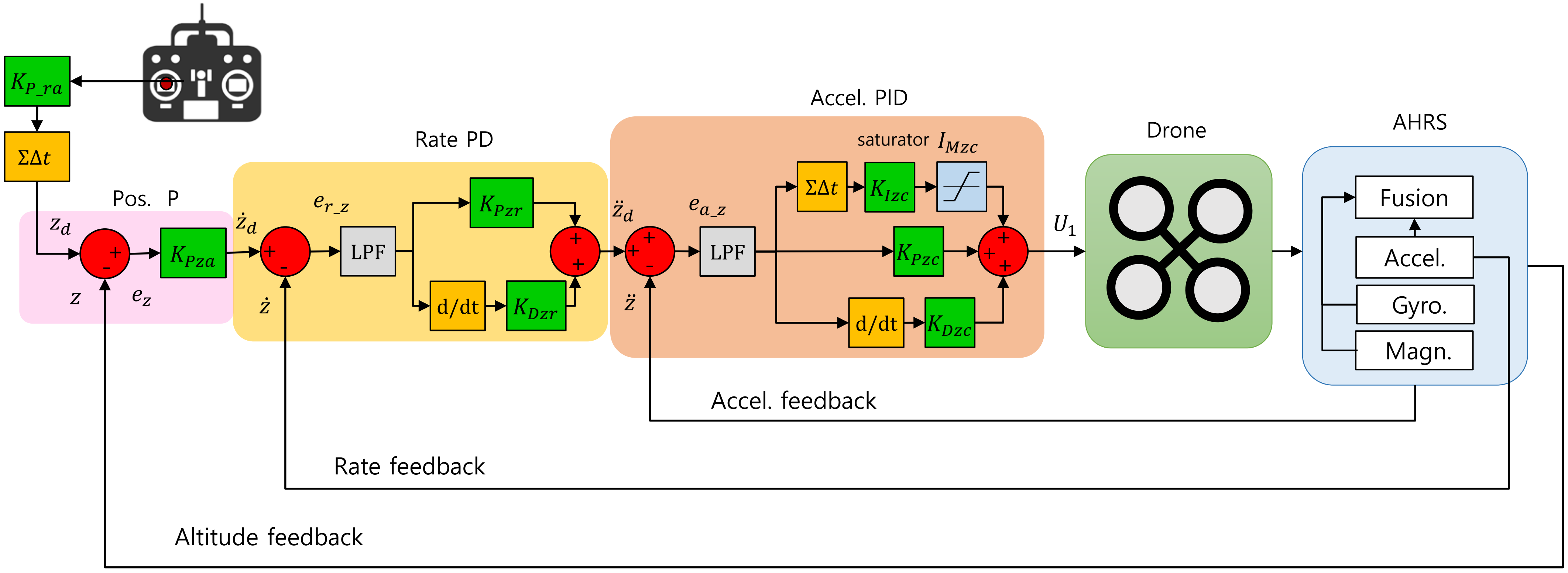
■ Yaw 제어



삼중 루프 PID - Alt

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

■ 고도 제어



주요 파라미터

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

Roll, Pitch, Yaw, Alt PID Parameter

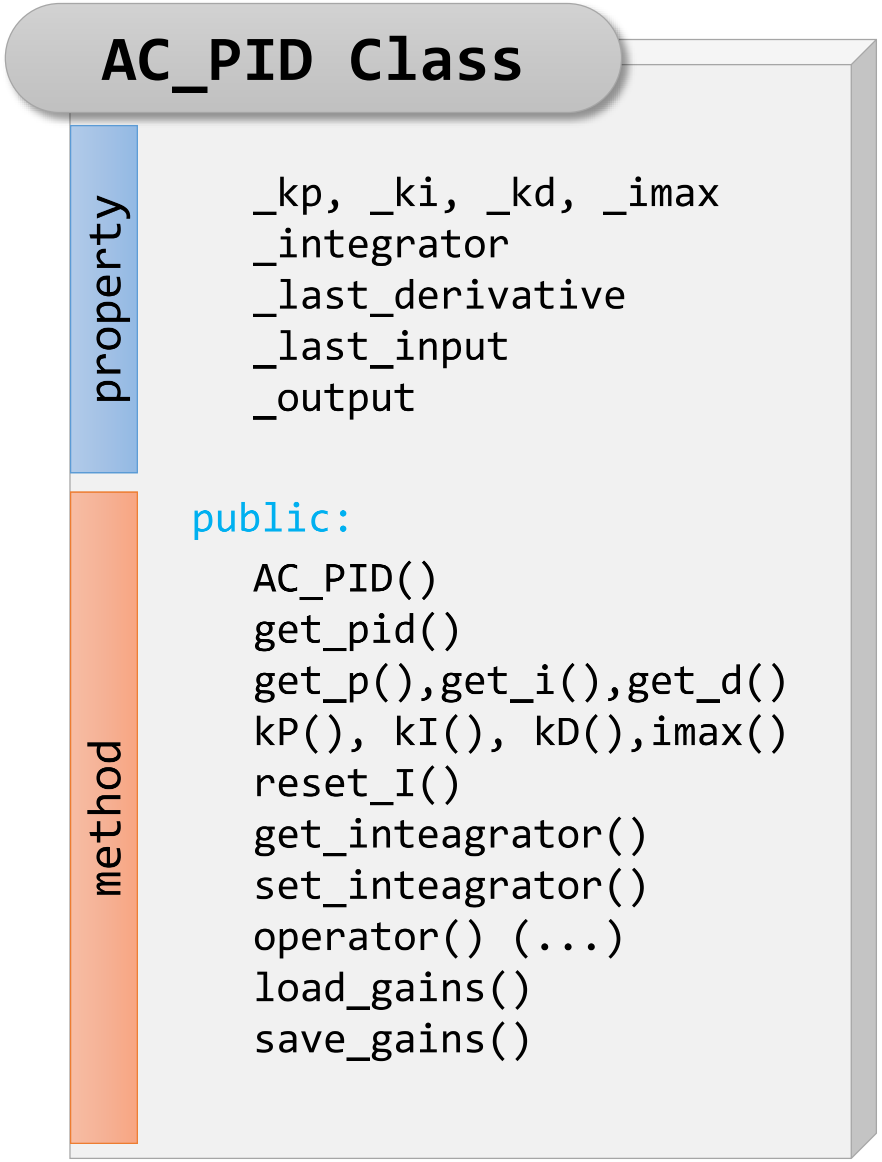
		Angle/Position loop				Rate loop				Acceleration loop				
		P	I	D	Imax	P	I	D	Imax	P	I	D	Imax	
Roll	r	K_{Pra}				K_{Prr}	K_{Irr}	K_{Drr}	I_{Mrr}					
Pitch	p	K_{Ppa}				K_{Ppr}	K_{Ipr}	K_{Dpr}	I_{Mpr}					
Yaw	y	K_{Pya}				K_{Pyr}	K_{Iyr}	K_{Dyr}	I_{Myr}					
Alt	z	K_{Pza}				K_{Pzr}		K_{Dzr}		K_{Pzc}	K_{Izc}	K_{Dzc}	I_{Mzc}	

AC_PID 클래스선언

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

AC_PID Class:

- Property
  - _kp, _ki, _kd: 각 항의 게인
  - _imax : 적분기 saturation
  - _integrator: I항 적분값
  - _last_input : 전 스텝의 오차
- Method
  - AC_PID(): 생성자
  - get_pid(): PID 계산값
  - get_p(), get_i(), get_d(): P,I,D 각 항
  - kP(),kI(),kD(),imax(): 읽기/쓰기
    - function overload
  - operator() (...):
    - Operator Overload로 게인을 한꺼번에 바꿀 수 있음.



C++ 참고

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

Function Overload

- 이름이 동일하고 매개변수가 다른 두 함수가 공존
- 예와 같이 다음 모두가 가능하다.
  - kP() : _kp 값을 반환
  - kP(v): v 값으로 _kp 변경
- 단, 매개변수의 개수 또는 타입이 달라야 함.

```
class AC_PID {
public:
...
    float kP() {
      return _kp; }
    void  kP(const float v) {
      _kp=v;}
...
};
...
AC_PID pid;
    pid.kP(1.5);
    Serial.println(pid.kP());
...
```

C++ 참고

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

Operator Overload

- 연산자를 Class에 적용하여 기존 연산 기호를 대치
- 예)
  - Complex operator + (Complex &obj2) {}
    - obj1 + obj2 가 가능하도록 + 오버로드
  - void operator ()(int r, int i){...}
    - object(a,b) 인 경우 실행

```cpp
#include<iostream>
using namespace std;
class Complex {
private:
    int real, imag;
public:
    Complex(int r=0, int i=0):real(r),imag(i){}
    Complex operator + (Complex &obj) {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }
    void operator ()(int r, int i){ real=r;imag=i;}
    void print() {cout<<real<<" + i"<<imag<<endl;}
};
int main() {
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2;  // + operator
    c3.print();
    c1(-5,7); //  () operator overloading
    c3 = c1 + c2;
    c3.print();
}
```

AC_PID Class 코드

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

■ 헤더

AC_PID.h [1/2]

```
class AC_PID {
public:
    AC_PID( const float &initial_p = 0.0, // constructor
           const float &initial_i = 0.0,
          const float &initial_d = 0.0,
             const int16_t &initial_imax = 0.0):
        _kp (initial_p), _ki (initial_i),
          _kd (initial_d),_imax(abs(initial_imax)){}
    int32_t get_pid(int32_t error, float dt);
    int32_t get_pi(int32_t error, float dt);
    int32_t get_p(int32_t error);
    int32_t get_i(int32_t error, float dt);
    int32_t get_d(int32_t error, float dt);
    void reset_I();
    void    load_gains();
    void    save_gains();
     // Overload the function call operator
    void operator() (const float p, const float i,
                     const float d, const int16_t imaxval) {
      _kp = p; _ki = i; _kd = d; _imax = abs(imaxval);
    }
```

AC_PID Class 코드

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

■ 헤더

AC_PID.h [2/2]

```
    float   kP() const   { return _kp; }
    void  kP(const float v)   { _kp=v; }
    float   kI() const   { return _ki; }
    void  kI(const float v)   { _ki=v; }
    float   kD() const   { return _kd; }
    void  kD(const float v)   { _kd=v; }
    int16_t imax() const { return _imax; }
    void imax(const int16_t v) { _imax=abs(v); }
    float   get_integrator() const    { return _integrator; }
    void set_integrator(float i)   { _integrator = i; }
private:
    float       _kp, _ki, _kd, _imax;
    float       _integrator; // integrator value
    int32_t  _last_input; // last input for derivative
    float       _last_derivative; // last derivative for low-pass filter
    float       _output;
    static const float _filter =15.9155e-3;
     // 1/(2*PI*f_cut) LPF for derivative
};
```

AC_PID Class 코드

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- C++

AC_PID.cpp [1/3]

```
#include "AC_PID.h"
int32_t AC_PID::get_p(int32_t error) {
    return (float)error * _kp;
}
int32_t AC_PID::get_i(int32_t error, float dt) {
    if((_ki != 0) && (dt != 0)){
        _integrator += ((float)error * _ki) * dt;
        if (_integrator < -_imax) {
            _integrator = -_imax;
        } else if (_integrator > _imax) {
            _integrator = _imax;
        }
        return _integrator;
    }
    return 0;
}
```

AC_PID Class 코드

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- C++

AC_PID.cpp [2/3]

```cpp
int32_t AC_PID::get_d(int32_t input, float dt){
    if ((_kd != 0) && (dt != 0)) {
        float _derivative = (input - _last_input) / dt;
        _derivative = _last_derivative +
                (dt / ( _filter + dt)) * (_derivative - _last_derivative);
        _last_input        = input;
        _last_derivative    = _derivative;
        return _kd * _derivative;
    }
    return 0;
}
int32_t AC_PID::get_pi(int32_t error, float dt){
    return get_p(error) + get_i(error, dt);
}
int32_t AC_PID::get_pid(int32_t error, float dt) {
    return get_p(error) + get_i(error, dt) + get_d(error, dt);
}
```

AC_PID Class 코드

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- C++

AC_PID.cpp [3/3]

```
void AC_PID::reset_I() {
    _integrator = 0;
    _last_input = 0;
    _last_derivative = 0;
}
void AC_PID::load_gains() {
//  to be finished
}
void AC_PID::save_gains() {
//  to be finished
}
```

아두이노 코드

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

ino

PIDClassTest.ino [1/2]

```
#include "AC_PID.h"
// default PID values
#define TEST_P 0.5
#define TEST_I 0.5
#define TEST_D 0.2
#define TEST_IMAX 20
AC_PID ratePID(TEST_P, TEST_I, TEST_D, TEST_IMAX);
int count=0;
float dt=0.01;
uint32_t prevTime=micros();
void setup() {
    Serial.begin(115200);
    angPID(0.7,0.8,0.5,10);
}
void loop() {
  makeDt(10000);
  float t= 0.01*count;
  int error_r=100*sin(t);
  int32_t pTerm_r=ratePID.get_p(error_r);
```

AC_PID Class 코드

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

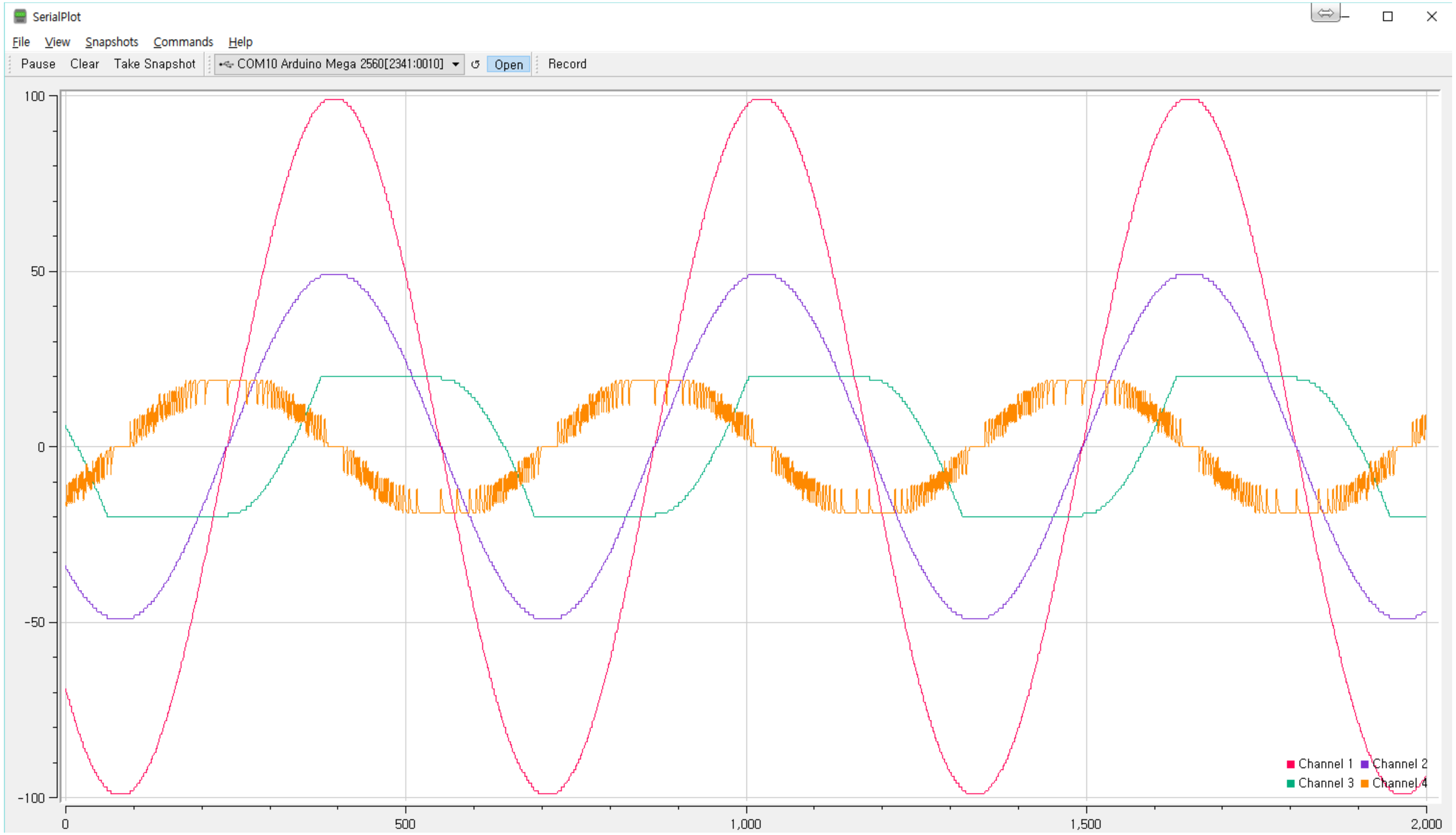
- ino

PIDClassTest.ino [2/2]

```
  int32_t iTerm_r=ratePID.get_i(error_r,dt);
  int32_t dTerm_r=ratePID.get_d(error_r,dt);
  int32_t control_r= pTerm_r+ iTerm_r+ dTerm_r;
  Serial.print(error_r);Serial.print(",");
  Serial.print(pTerm_r);Serial.print(",");
  Serial.print(iTerm_r);Serial.print(",");
  Serial.print(dTerm_r);
  Serial.print("\n");
  count++;
}
void makeDt(uint32_t p){
  uint32_t newTime;
  do {newTime = micros();}while (newTime - prevTime<p);
  prevTime = newTime;
}
```

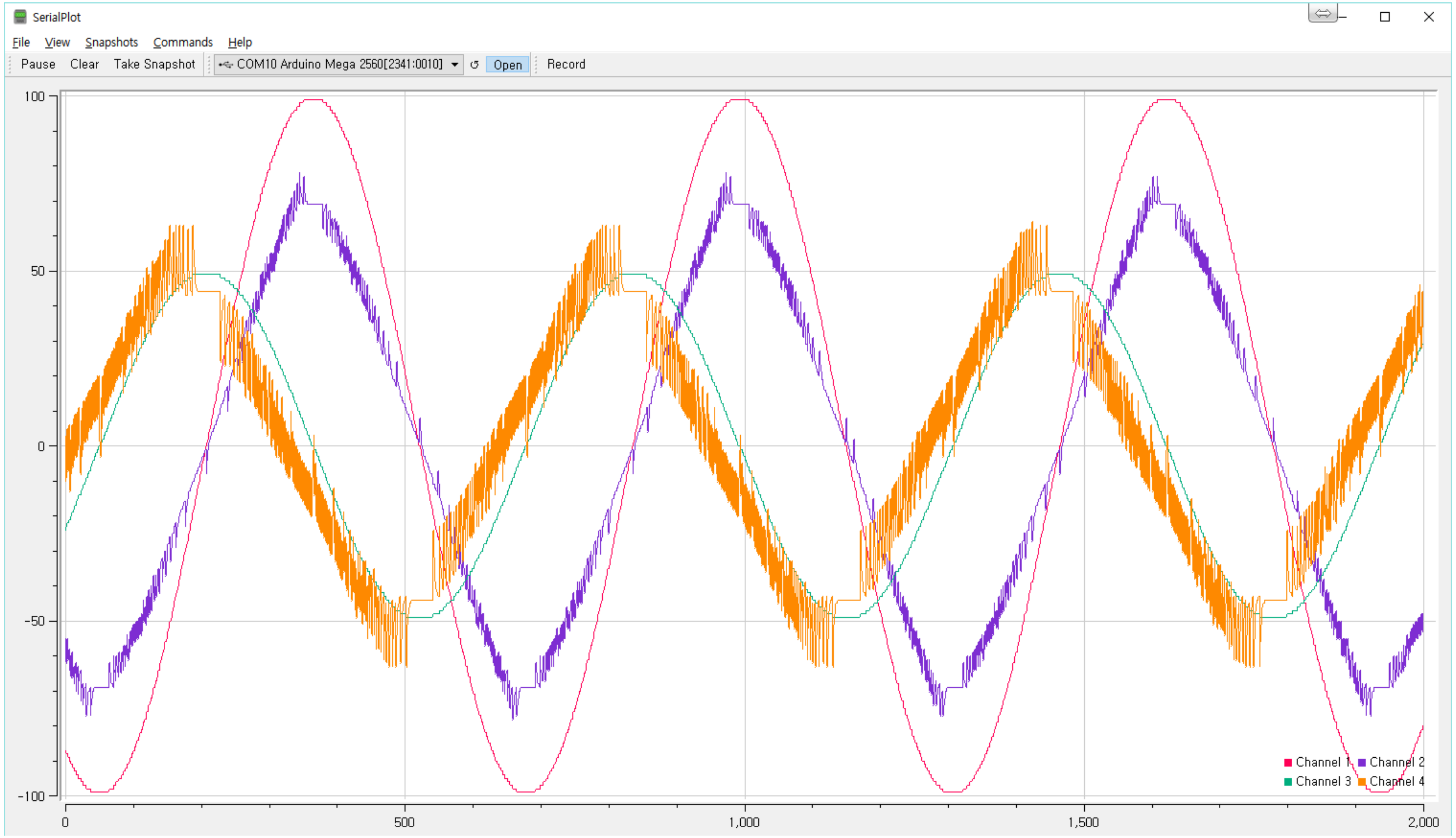
각항의 비교

Dept. of Mechanical System Design, Seoul National University of Science and Technology.



2개의 PID

Dept. of Mechanical System Design, Seoul National University of Science and Technology.



Parameters

EEPROM 사용

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

내장 EEPROM library

- #include <EEPROM.h>
- **API**
  - length()
  - Single byte 읽고 쓰기
    - read(), write(), update()
  - 구조체로 읽고 쓰기
    - get(), put()
- **[] operators**
  - 배열처럼 사용

```
uint8_t read(int idx);
void write(int idx, uint8_t val);
void update(int idx, uint8_t val);
```

```
template<typename T> T &get(int idx, T &t);
template<typename T> const T &put(int idx,
                                   const T &t);
```

```
// read the byte from address 88
uint8_t data = EEPROM[88];
// write a byte to address 76
EEPROM[76] = 97;
// increase the value in address 2 by 1
EEPROM[2]++;
```

EEPROM 사용

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- 구조체 test

```
#include <EEPROM.h>
void setup() {
  Serial.begin(115200);
  struct Format {char str[25]; int i;  float f; } ;
  Format  a = {"This is EEPROM Test!", 32767, -0.09567}, b;
  EEPROM.put(0, a); // Write all
  EEPROM.get(0, b); // Read all
  Serial.println(b.str);  //Print:
  Serial.println(b.f, 5);
  Serial.println(b.i);
    int i_new = -784;     float f_new = 782.10132;
  // write each new value to EEPROM:
  EEPROM.put(offsetof(Format, i), i_new);
  EEPROM.put(offsetof(Format, f), f_new);
  // read each new value from EEPROM:
  EEPROM.get(offsetof(Format, f), a.f);
  EEPROM.get(offsetof(Format, i), a.i);
  Serial.println("===========");   // Print:
  Serial.println(a.i);
  Serial.println(a.f, 5);
}
```

```
This is EEPROM Test!
-0.09567
32767
============
-784
782.10131
```

EEPROM 사용

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

이중 구조체 사용

- PID
- PARAM

```
#include <EEPROM.h>
void setup() {
  Serial.begin(115200);
  struct PID { float kP, kI, kD; uint16_t imax;};
  struct PARAM {
    PID angle, rate, acc;
   };
  PARAM  a = {{1.1,1.2,1.3,0},{2.1,2.2,2.3,1},{3.1,3.2,3.3,2}}, b;
  EEPROM.put(0, a); // Write all
  EEPROM.get(0, b); // Read all
  Serial.print("angle kP=");Serial.println(b.angle.kP);
  Serial.print("rate kI=");Serial.println(b.rate.kI);
  Serial.print("acc kD=");Serial.println(b.acc.kD);
}
void loop() {}
```

```
angle kP=1.10
rate kI=2.20
acc kD=3.30
```

PID gain에 적용

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

PID 삼중 구조체

- PARAM
  - CONTROL
    - PID

```
#include <EEPROM.h>
struct PID { float kP, kI, kD; uint16_t imax;};
struct CONTROL { PID angle, rate, acc; };
struct PARAM { CONTROL roll, pitch, yaw, alt; };
void setup() {
  Serial.begin(115200);
  PARAM  initial_gains = {
   // kP,    kI,   kD,    imax        roll gains
   {{0.05, 0.01, 0.003, 100},  // angle
    {0.01, 0.02, 0.03, 100},   // rate
    {0.0, 0.0, 0.0, 0}},
   // pitch gains
   {{0.05, 0.01, 0.003, 100}, // angle
    {0.01, 0.02, 0.03, 100},  // rate
    {0.0, 0.0, 0.0, 0}},
```

PID gain에 적용

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

```
    // yaw gains
    {{0.06, 0.01, 0.003, 100}, // angle
     {0.02, 0.04, 0.03, 100},  // rate
     {0.0, 0.0, 0.0, 0}},
    // alt gains
    {{0.07, 0.01, 0.003, 100}, // angle
     {0.03, 0.00, 0.03,   0},  // rate
     {0.2, 0.02, 0.03, 100}}   // acc
   };
   saveGains(initial_gains);
   PARAM gains=loadGains();
   Serial.print("kPra="); Serial.println(gains.roll.angle.kP);
   Serial.print("kIyr="); Serial.println(gains.yaw.rate.kI);
   Serial.print("kDar=");Serial.println(gains.alt.acc.kD);
   PARAM c;
   uint16_t offset=offsetof(PARAM, alt)+offsetof(CONTROL, acc)
                  +offsetof(PID, imax);
   EEPROM.get(offset, c.alt.acc.imax);
   Serial.println(c.alt.acc.imax);
 }
```

PID gain에 적용

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

주요내용

- saveGains(a)
  - a를 eeprom에 저장
- c = loadGains()
  - 읽어서 c에 저장

```
void loop() {}
void saveGains(PARAM a){
    EEPROM.put(0, a); // Write all
}
PARAM loadGains(){
  PARAM b;
    EEPROM.get(0, b); // Read all
  return b;
}
```

```
kPra=0.05
kIyr=0.04
kDar=0.03
100
```

Gyro Calibration

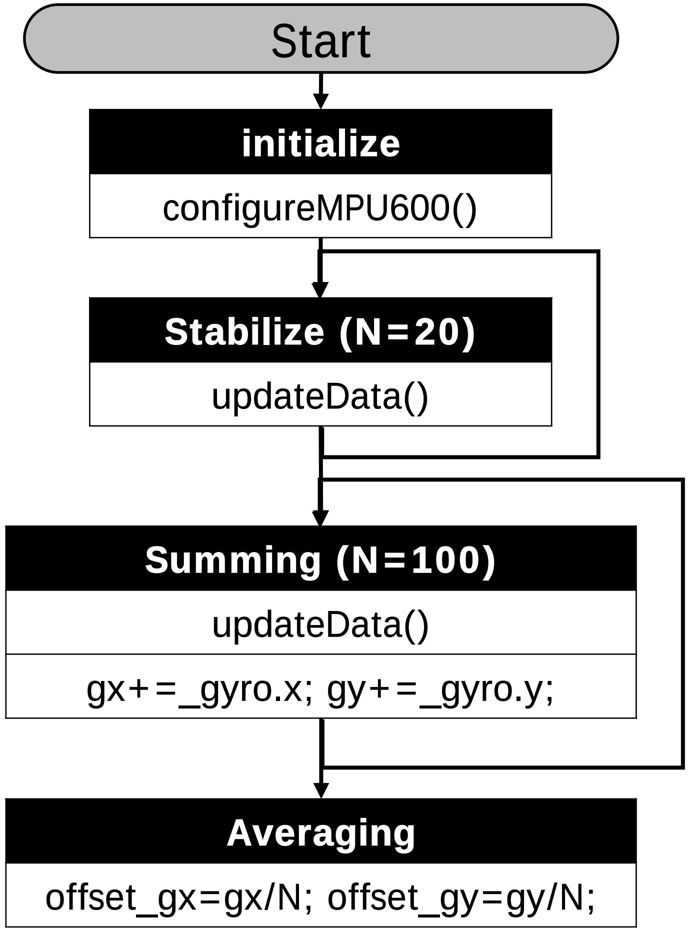
Dept. of Mechanical System Design, Seoul National University of Science and Technology.

IMU gyro의 offset

- PID 의 rate control에 roll, pitch, yaw 각속도 값이 사용됨
  - Gyro의 측정값에 작은 offset 값이 있으면
  - 정지상태에서도 각속도 값이 존재

Calibration 방법

- 장치를 초기화
- 신호의 안정화
- 100회 측정 및 누적
- 평균 구하기



Gyro Calibration 코드

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

MPU6000 Class

- calibrate()추가

MPU6k.cpp

```
void MPU6000::calibrate(){
  float gx=0,gy=0,gz=0;
  // Stabilize signal
  for (int i=0;i<20;i++){
    updateData();  delay(10);
  }
  // Measure and accumulate
  for (int i=0;i<100;i++){
    updateData();
    gx+=_gyro.x; gy+=_gyro.y; gz+=_gyro.z;
    delay(10);
  }
  // Averaging
  offset_gx=gx/100; offset_gy=gy/100;
  offset_gz=gz/100;
}
```

Gyro Calibration 코드

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

MPU6k.cpp

- updateData()
  - offset 반영

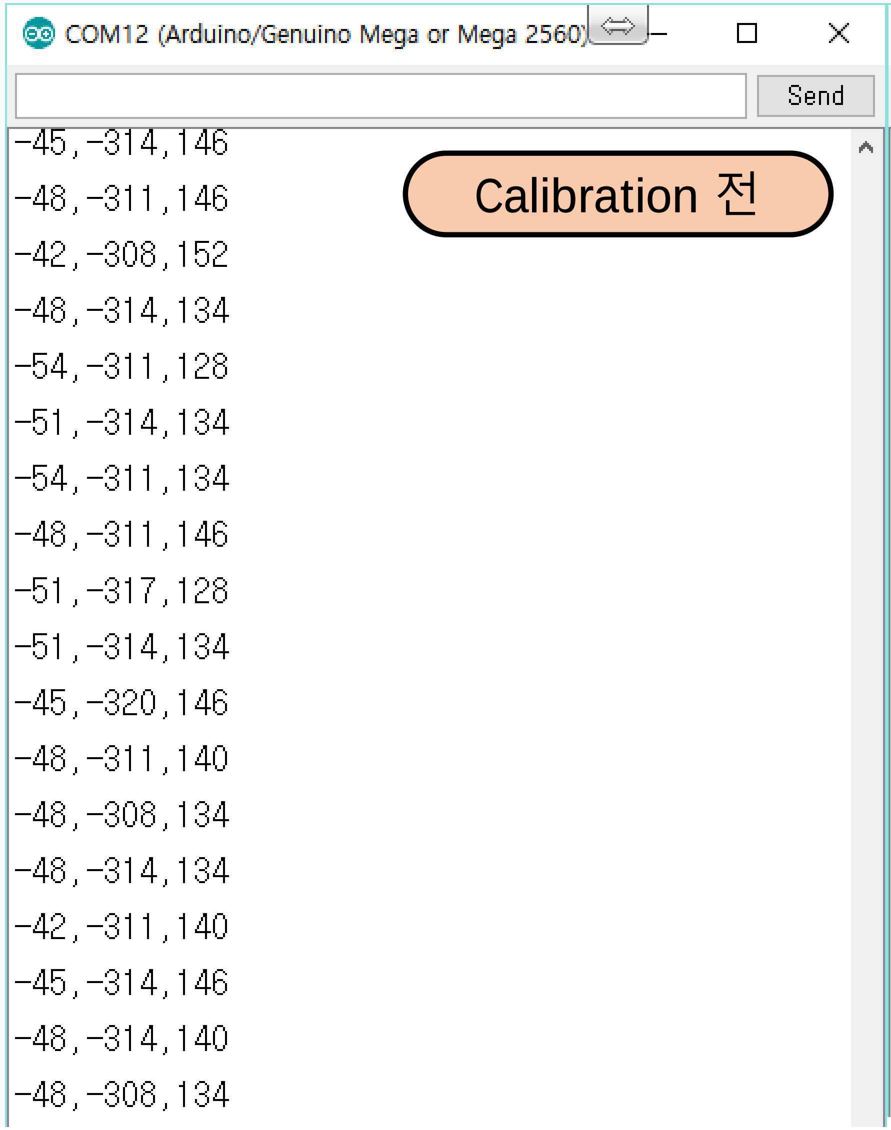
MPU6k.cpp

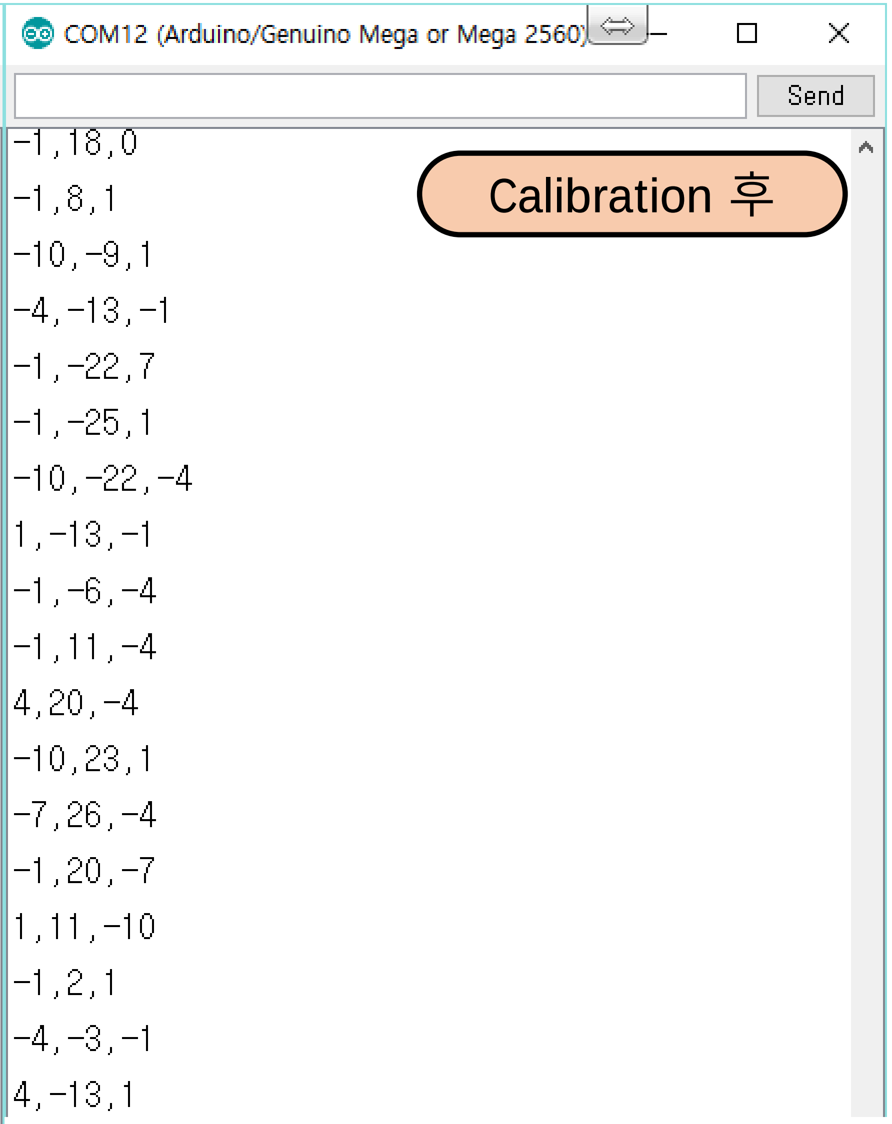
```
bool MPU6000::updateData( ) {
    ...
  sei();   //enable interrupts
  count_scale = 1.0 / count;
  _gyro.x = _gyro_scale*_gyro_data_sign[0]*sum[_gyro_data_index[0]]*count_scale;
  _gyro.y = _gyro_scale*_gyro_data_sign[1]*sum[_gyro_data_index[1]]*count_scale;
  _gyro.z = _gyro_scale*_gyro_data_sign[2]*sum[_gyro_data_index[2]]*count_scale;
  ...
  _gyro.x -= offset_gx; _gyro.y -= offset_gy;  _gyro.z -= offset_gz;
  return true;
}
```

Gyro Calibration 결과

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- offset 반영





THANK YOU

Powerpoint is a complete presentation graphic package it gives you everything you need to produce a professional-looking presentation