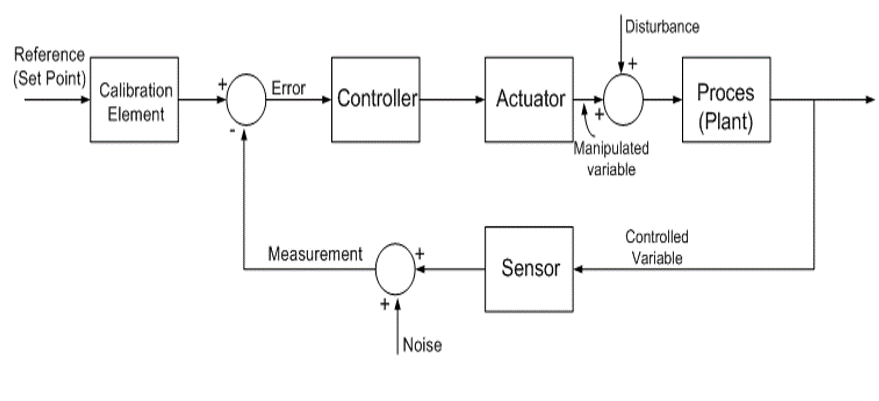
# PID 제어 시스템을 이용한 공정제어

|  |  |
| --- | --- |
| **학습 내용** | |
| [문제] | PID 제어 시스템의 이해 |
| [방법] | PID 제어 시스템의 조율 |
| [응용] | 딥러닝을 적용한 PID 제어 시스템의 조율 |
| [요약] | * PID 제어 시스템의 이해 * 기존 PID 제어 시스템의 조율 방법 * 딥러닝을 이용한 PID 제어 시스템의 조율 |

## [이론] PID 제어 시스템

### 공정제어의 개요

**공정제어**(process control)는 원하는 제품 조건을 달성하기 위해서 공정의 입력값을 조절하는 것을 말한다. 모든 공정은 정상 조건 하에서 물질 및 에너지 수지를 통해 이론적으로 설계된다. 하지만 실제 현장에서는 공정의 외부 조건이 계속 바뀌기 때문에 정상 조건을 가지는 것이 거의 불가능하다. 반응기를 예로 들면, 반응기 외부 온도는 시간에 따라 계속 변하며, 외부 습도 또한 날씨의 영향으로 계속 변한다. 이와 같이 운전상태에 변화를 일으키는 **외란**(disturbance)들로 인해 실제 공정은 정상 상태가 아닌 동적 상태에서 운행된다. 동적 상태에서 운전되는 공정에서 제어가 제대로 이루어지지 않으면 목표하는 조건에서 공정이 운행되지 못하고, 이는 제품의 품질 불량이나 수율의 감소뿐만 아니라 큰 사고를 유발할 수 있다.

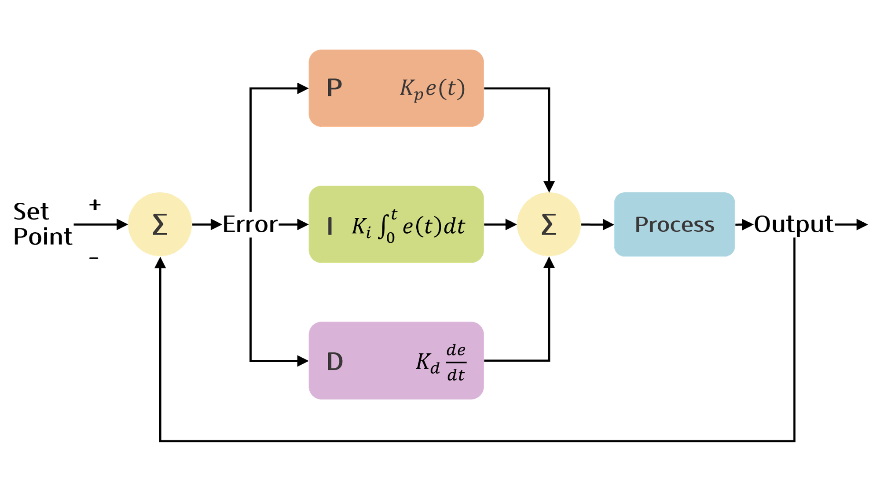


**그림 1. 피드백 제어 시스템의 블록 흐름도 예시**

공정제어 시스템은 공정(process)을 중심으로 크게 입력 변수(input variables)와 출력 변수(output variables)로 분류된다. 입력 변수에는 앞에서 언급한 외란과 **조작 변수**(manipulated variables; MV)로 나눌 수 있다. 외란은 외부 환경에 의해 변하므로 조작할 수 없는 반면, 조작 변수는 우리가 공정을 제어하기 위해 조작하는 변수로 조절할 수 있는 입력값이다. 그림 1과 같이 외란과 함께 조작 변수를 조절함으로써 공정의 출력 변수인 **제어 변수**(controllable variables; CV)가 변한다. 제어 변수는 센서에 의해 측정이 되고, 이 측정값이 목표값(set point)과 같아지도록 제어기(controller)가 새로운 조작 변수를 출력한다.

### PID 제어 시스템의 이해

**PID 제어 시스템**은 피드백 제어 시스템(feedback control system)의 하나로, 간단하고 빠르다는 장점이 있어 실제 산업에서 활발히 사용되는 제어기이다. PID 제어 시스템은 **P 제어**(proportional control; P control), **I 제어**(integral control; I control), 그리고 **D 제어**(derivative control; D control)로 구성되어 있다. P 제어와 I 제어는 단독으로도 사용되며, 경우에 따라서는 PD 제어, PI 제어 등으로 사용 가능하다.



**그림 2. PID 제어 시스템의 블록 흐름도**

**P제어**는 비례 제어라고 하며, 현 시점의 제어오차에 비례하는 제어동작을 취한다. P 제어는 계산이 간단하여 응답이 빠르다는 장점이 있으나 잔류오차 (offset)가 발생한다는 단점이 존재한다. **잔류 오차**란 목표값과 제어 변수의 차이를 해결하지 못하고 그대로 평행 유지되어 제거되지 않는 오차를 말한다. P 제어는 식 (1)과 같이 나타낼 수 있다.

식 (1)

여기서 ubias는 오차가 0일때 조작 변수가 0이 아니라 특정 조작 변수를 취하는 값을 표현하는 것으로, ubias가 0이면 오차가 0일 때 u(t)=0이다. **KC**는 비례 이득(proportional gain) 혹은 제어 이득(control gain)이라고 하고, 이는 P 제어기의 성능을 결정하는 **조율 인자**(tuning factor)이다. KC가 커지면 같은 크기의 오차에 대해서 더 큰 MV값을 출력하므로, P 제어기는 더욱 공격적으로 제어한다. PB(proportional band)는 100/KC를 의미한다.

**I 제어**는 적분 제어라고 한다. I 제어는 제어 오차가 0이 될 때까지 계속해서 제어오차를 누적하며 제어동작을 지속한다. 이로 인해 P 제어의 단점인 잔류 오차를 해결할 수 있다. 하지만 어느 정도의 제어 오차가 누적이 된 후 큰 제어 동작을 수행하기 때문에 제어 동작이 느리다는 단점이 있다. I제어는 식 (2)를 통해 나타낼 수 있다.

식 (2)

**KI**는 적분 시간(integral time) 또는 리셋 시간(reset time)이라고 하며, 단위는 ‘분’이다. KI는 I 제어의 조율 인자로, 제어오차가 일정하게 지속되는 경우 적분 동작이 매 KI마다 비례제어동작에 해당하는 제어동작을 누증시키는 것을 의미한다. 다시 말해, KI가 짧을수록 I 제어는 더 자주 제어동작을 누증하여 제어 오차가 더 빨리 쌓이므로 더 큰 제어동작을 한다. 부가적으로 KI의 역수를 리셋 속도 (reset rate; Ri)라고 한다.

마지막으로, 미분 제어를 하는 **D 제어**는 제어오차의 기울기에 비례하여 제어동작을 취한다. D 제어는 동작은 빠르나 P 제어와 같이 잔류오차가 발생한다. 또한 D 제어는 측정된 제어오차들을 통해 일정 시간 앞의 제어오차를 예측해 제어동작을 출력하므로 외란 등에 의한 노이즈에 민감하다.

식 (3)

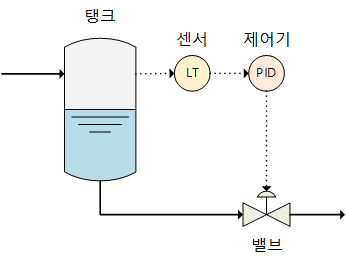
D 제어는 P 제어나 I제어와 달리 단독으로 사용하지 않으므로, 식 (3)에는 PD 제어를 나타내었다. 이때, D제어의 조율 인자인 **KD**는 미분 시간(derivative time) 또는 선행 시간(preact time)이라고 하고, 단위는 ‘분’이다. 이것의 물리적인 의미는 제어오차가 일정한 속도로 증가하는 경우, 미분동작은 초기에 비례동작보다 KD 앞선 제어동작을 출력하는 것이다.

### 인공지능과의 접목

PID 제어 시스템은 각각의 조율 인자에 따라 같은 오차에 대해서도 다른 출력을 나타낸다. 따라서 원하는 출력의 제어 시스템을 활용하기 위해서는 그에 맞게 조율하는 과정이 필요하다. 조율이 제대로 되지 않으면 같은 양의 제품을 생산하더라도 낮은 순도의 제품을 얻을 수 있고, 반응이나 분리가 제대로 되지 않아 더 많은 원료를 필요로 하게 되어 경제적인 손실을 입을 수도 있다. 따라서 최근에는 기존의 시행착오법이나 계산을 통해 조율하는 것이 아닌 인공지능을 활용하여 조율을 자동화하고자 하는 연구가 활발히 진행되고 있다. 특히, 인공지능 학습을 통해 제어 시스템의 조율이 잘 되면 공정에 소요되는 물질이나 에너지의 양이 줄어들 뿐만 아니라 공정에 소요되는 시간 또한 단축할 수 있어 큰 경제적인 효과를 불러올 것으로 기대된다.

**[문제]**

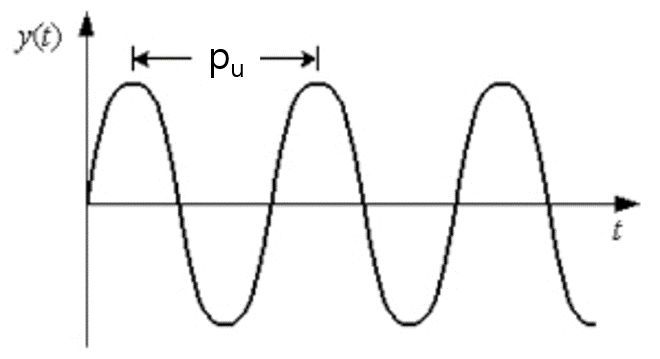
**수조에 찬 물의 수위를 PID 제어 시스템을 통해 제어하고자 한다. 관련 데이터는 ‘matplotlib.pyplot’를 불러와 이용할 때, PID 제어 시스템에 딥러닝을 적용해 PID 제어 시스템의 KP, KI, KD를 조율하라. 반복 계산 횟수는 100,000번으로 하고, 절댓값 오차를 사용한다. 초기의 조건은 KP=1.0, KI=0.5, KD=0.5이고, 오차축적 횟수는 100, 학습률(learning rate)은 0.001, 초기 오차는 10이다.**



**[방법] PID 제어 시스템의 조율**

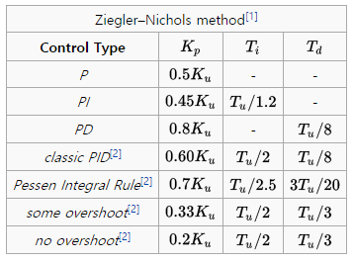
### Q1. PID 제어 시스템의 조율 방법 중 가장 대표적인 지글러-니콜스 조율(Ziegler-Nichols tuning) 방법에 대해 설명하여라.

A1. 지글러-니콜스 조율은 1942년 Ziegler와 Nichols에 의해 제안된 방법 중 하나로, 경계안정상태의 특성에 근거한 제어기 조율 방법 중 하나이다. 지글러-니콜스 조율은 먼저 I 제어와 D 제어를 끄고 P 제어기만 켠 상태 또는 KI에 무한에 가까운 수를 넣고, KD에는 0을 입력한 상태에서 진행한다. 이 상태에서 그림 3과 같이 공정 출력에 지속적인 진동 현상이 나타날 때까지 KC를 작은 값에서부터 서서히 증가시킨다.



**그림 3. 공정 출력의 지속적인 진동 현상**

KC가 작을 때에는 목표값에 수렴하다가 KC가 점점 커지면서 오차에 민감해지기 때문에 그림 3과 같은 규칙적인 진동이 관측되기 시작한다. 만약 이보다 KC가 더 커지면 공정 출력은 발산한다. 규칙적인 진동의 주기를 Pu(ultimate period)라고 하며, 이때의 KC는 KCu(ultimate controller gain)이라고 한다. 마지막으로, KCu을 표 1에 제시된 표에 대입하여 제어 시스템을 조율한다.



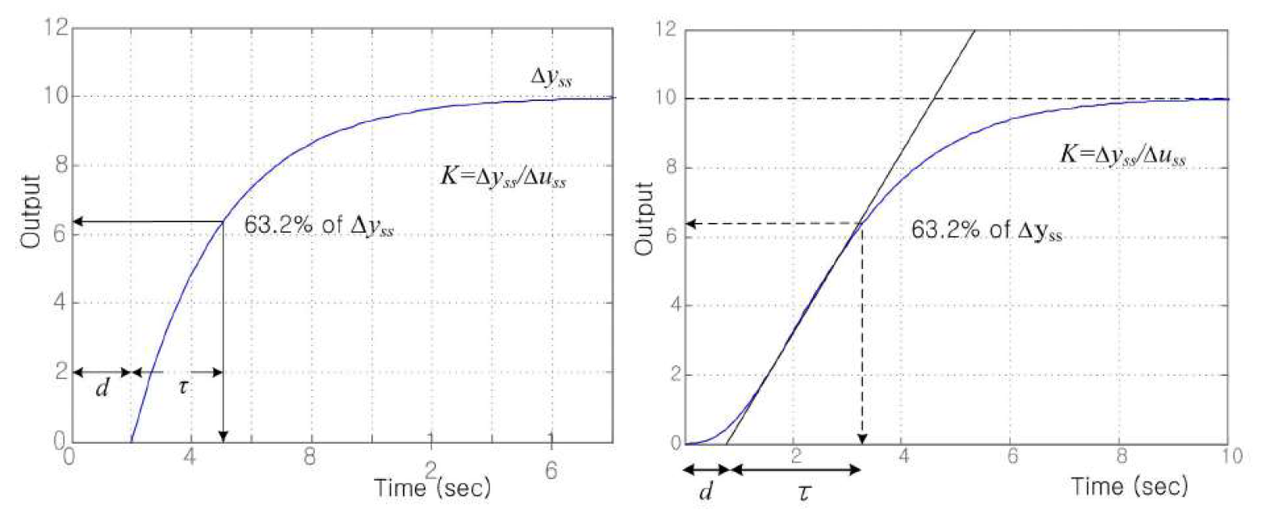
**표 1. 제어 시스템의 지글러-니콜스 조율 값**

### Q2. 또 다른 PID 제어 시스템 조율 방법인 1차 시간지연(first-order plus dead time) 모델의 전제 조건은 무엇이며, 조율하는 방법을 설명하여라.

A2. 1차 시간지연 모델은 식 (4)와 같이 나타낼 수 있다. 식 (4)는 시간 t를 라플라스 변환하여 나타낸 식이다.

식 (4)

식 (4)에서 d와 τ는 그림 4를 통해 알 수 있다. d는 지연 시간(dead time)으로, 제어기를 통해 제어출력이 나올 때까지의 시간을 의미한다. 실제 공정에서는 대부분 2차 함수와 같은 모습이 나타나는데, 이 경우 변곡점에서 접선을 그린 후 0 초부터x축과 만나는 점까지의 시간을 지연 시간으로 정의한다. τ는 시정수를 의미하며 제어 출력이 정상 상태의 제어 출력의 63.2%인 지점의 시간을 말한다.

**그림 4. (a) 1차 함수와 (b) 2차 함수의 반응응답곡선**

1차 시간지연모델의 전제 조건은 ‘0.1<d/τ<1.0’이다. d/τ<0.1일 때에는 지연 시간이 무시되는 1차 공정에 근접한다. 1차 공정의 경우 KC를 크게 할수록 제어가 잘된다. 반면, d/τ>1.0에서는 지연시간이 매우 커 일반적인 PID 제어 시스템으로는 제어성능에 한계가 생기므로 dead time compensator(ex. Smith predictor)가 필요하다.

|  |  |  |  |
| --- | --- | --- | --- |
| 제어기 | KC | KI | KD |
| P | (τ/KPd) | - | - |
| PI | 0.9(τ/KPd) | 3.33d | - |
| PID | 1.2(τ/KPd) | 2.0d | 0.5d |

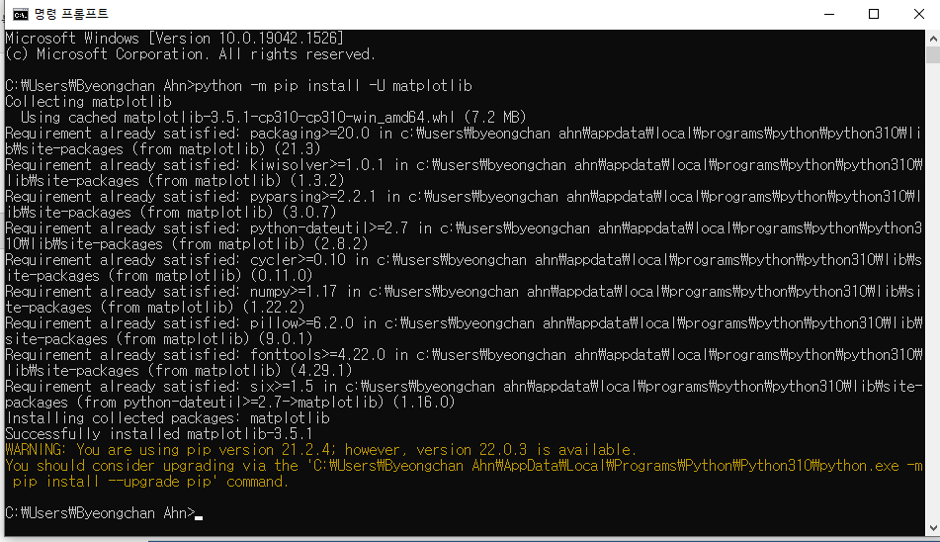
**표 2. 제어 시스템의 1차 시간지연 모델 조율 값**

마지막으로, 1차 시간지연 모델로부터 지글러-니콜스 조율에 필요한 Kcu와 Pu를 구하여 지글러-니콜스 조율을 적용한다. Kcu와 Pu를 쉽게 계산하기 위해 표2와 같이 KP, d, τ값으로도 나타낼 수 있다.

**[응용] 딥러닝을 통한 이산화탄소 배출량 예측**

### Q3. ‘matplotlib.pyplot’을 불러오고, 탱크의 수위를 PID제어기를 정의하여라.

A3. 먼저, 그림 5와 같이 윈도우 명령 프롬프트에 pip을 활용해 matplotlib을 컴퓨터에 설치한다. 그리고 아래의 절차에 따라 탱크의 수위와 PID 제어 시스템을를 정의한다.



|  |
| --- |
| import matplotlib.pyplot as plt   #데이터베이스 불러오기 |

**그림 5.** matplotlib.pyplot **설치하기**

명령 프롬프트에 ‘python -m pip install -U matplotlib’를 입력하여 matplotlib.pyplot를 설치한다.

먼저, 파이썬에 설치한 matplotlib.pyplot를 불러온다.

|  |
| --- |
| class Liquid:      def \_\_init\_\_(self, error):          self.current\_error = error          self.last\_action = 0      def take\_action(self, action):          self.current\_error += 0.1 \* action          self.last\_action = action  class PID:      err\_sum = 0      old\_err = 0      def pid(self, current, goal, kp, ki, kd):          err = goal - current          self.err\_sum += err          delta\_err = err - self.old\_err          self.old\_err = err          return kp\*err + ki\*self.err\_sum + kd\*delta\_err |

위의 코드와 같이 탱크의 수위는 목표값과 실제 탱크의 수위만큼의 차인 error를 가져와 제어동작을 취함으로써 수위를 조절한다. 이때, ‘current’는 현재 탱크의 수위, ‘goal’은 제어기의 목표값을 나타내며, kp, ki, kd는 각 제어기의 조율 인자(KP, KI, KD)를 의미한다.

### Q4. PID 제어 시스템을 조율하기 위한 함수를 정의해라

|  |
| --- |
| class Derivative:      def \_\_init\_\_(self):          self.last\_x = 0          self.last\_y = 0      def get\_gradient(self, x, y):          d = (y - self.last\_y) / (x - self.last\_x)          self.last\_x = x          self.last\_y = y          return d |

A4. PID 제어 시스템 조율에 필요한 함수는 아래의 절차대로 정의한다.

위의 코드와 같이 미분 클래스를 정의한다. 미분 클래스는 기울기를 통해 조율 인자를 최적화할 때 사용된다.

|  |
| --- |
| class Train:      kp = 1.0; ki = 0.5; kd = 0.5      goal = 0      episode\_length = 100      learning\_rate = 0.001      def \_\_init\_\_(self):          self.dp = Derivative()          self.di = Derivative()          self.dd = Derivative()          self.step = 0          self.last\_loss = 0 |

위의 코드는 제어 시스템의 조율을 위해 해당 모델을 훈련시키기 위해 훈련 클래스를 정의하는 코드이다. 가장 먼저 문제에 주어진 초기 조건들을 입력하고 변수들을 정의한다. 목표값과 실제 탱크의 수위의 차이는 0이 되는 것이 이상적이므로 goal은 0으로 설정하고, 오차축적 횟수(episode\_length)는 100, 학습률은 0.001로 설정한다.

|  |
| --- |
| def abs\_mean(self, list):          sum = 0          for i in list:              sum += abs(i)          return sum / len(list)      def loss(self):          liquid = Liquid(10)          pid = PID()          error = []          for i in range(self.episode\_length):              error.append(liquid.current\_error)              liquid.take\_action(pid.pid(liquid.current\_error, self.goal, self.kp, self.ki, self.kd))          return self.abs\_mean(error)      def optimize(self):          self.kp = self.kp - self.learning\_rate \* self.dp.get\_gradient(self.kp, self.loss())          self.ki = self.ki - self.learning\_rate \* self.di.get\_gradient(self.ki, self.loss())          self.kd = self.kd - self.learning\_rate \* self.dd.get\_gradient(self.kd, self.loss())          self.last\_loss = self.dd.last\_y          print("step={}, kp={}, ki={}, kd={}, loss={}".format(self.step, self.kp, self.ki, self.kd, self.last\_loss))          self.step += 1 |

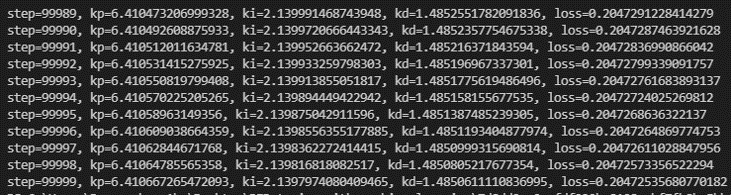
위의 코드는 훈련 클래스에서 정의된 함수들이다. ‘abs\_mean’은 문제에서 정의된 것처럼 오차를 구할 때 사용할 절댓값 평균 함수를 정의한다. ‘loss’는 for문을 통한 반복 계산을 통해 탱크의 수위를 보정하며 실제 탱크의 수위와 목표값의 차이(error)를 축적하는 함수이다. ‘loss’에서 구해진 error는 abs\_mean 함수를 통해 절댓값 평균으로 출력된다. 마지막으로, ‘optimize’는 앞서 정의한 함수들을 통해 최종적으로 조율 인자들을 최적화함으로써 PID 제어 시스템을 조율한다. 또한 이때, 뒤쪽의 그림 6과 같이 반복 계산을 하는 각 단계의 값들을 볼 수 있도록 설정했다.

### Q5. 앞서 정의한 함수들을 통해 KP, KI, KD 값을 최적화하고 이를 도식화하라.

A5. 아래와 같은 절차를 통해 조율 인자들을 최적화할 수 있다. 또한 그 결과는 그림 6과 7을 통해 각각 확인할 수 있다.

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':      loss = []      kp = []      ki = []      kd = []      train = Train()      for j in range(100000):          train.optimize()          loss.append(train.last\_loss)          kp.append(train.kp)          ki.append(train.ki)          kd.append(train.kd) |

먼저, 위의 코드와 같이 계산된 데이터를 담을 배열을 생성하고, Train의 인스턴스를 저장할 변수(train)을 생성한다. 또한 앞서 정의한 ‘optimize’함수를 사용해 조율 인자들을 최적화한다. 이때 반복 횟수는 문제에서 언급한 것처럼 100000회로 설정한다.

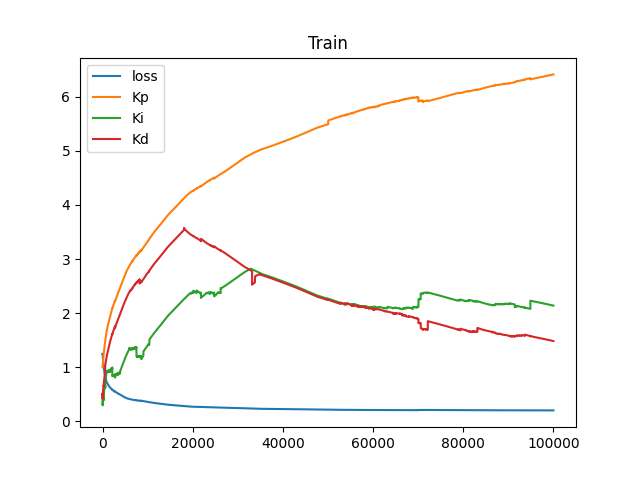


**그림 6. 조율 인자 최적화 결과 예시**

그림 6은 각 반복 횟수에 따른 최적화 결과를 나타낸다.

|  |
| --- |
| plt.plot(loss, label="loss")      plt.plot(kp, label="Kp")      plt.plot(ki, label="Ki")      plt.plot(kd, label="Kd")      plt.legend()      plt.title("Train")      plt.savefig("train.png") |

위의 코드는 그림 6에서 나타낸 최적화 과정을 도식화하는 코드이다. 위의 코드를 실행하면 그림 7과 같은 결과를 얻을 수 있다.



**그림 7. 조율 인자 도식화 코드 및 결과**

**[결론]**

**PID는 가장 기본적인 제어 시스템 중 하나로, 공정의 목표값과 실제 데이터간의 차이가 0이 되도록 조정하며 제어한다. 해당 예제에서는 딥러닝 기법을 적용하여 PID 제어 시스템을 튜닝했다. 그 결과, 반복 계산을 100,000회 반복하자 오차(loss)가 0으로 거의 수렴한 것을 확인할 수 있다. 더 많은 횟수를 반복한다면 더 정확하게 조율된 PID 제어 시스템을 얻을 수 있을 것이다.**

**[학습 결과]**

* 학습 내용

기본적인 PID 제어 시스템의 이해와 인공지능과의 접목이 필요한 이유 이해.

* 학습 결과 확인하기

대표적인 PID 제어 시스템 조율 방법 익히기.

* 학습 결과 응용하기  
  본 장의 학습내용에 기반해 실제 공정데이터를 이용해 딥러닝을 적용하여 PID 제어기를 조율하는데 응용.