智能控制项目作业—模糊控制

姓名：赵子瑞
班级：自动化钱61
学号：2160405068

项目任务

- 利用传统PID控制器实现对倒立摆的控制
- 利用PID控制器的模糊增益调整来实现倒立摆的自适应控制
- 对比试验结果并进行分析

倒立摆模型建立

简单的倒立摆可以由下图来表示。

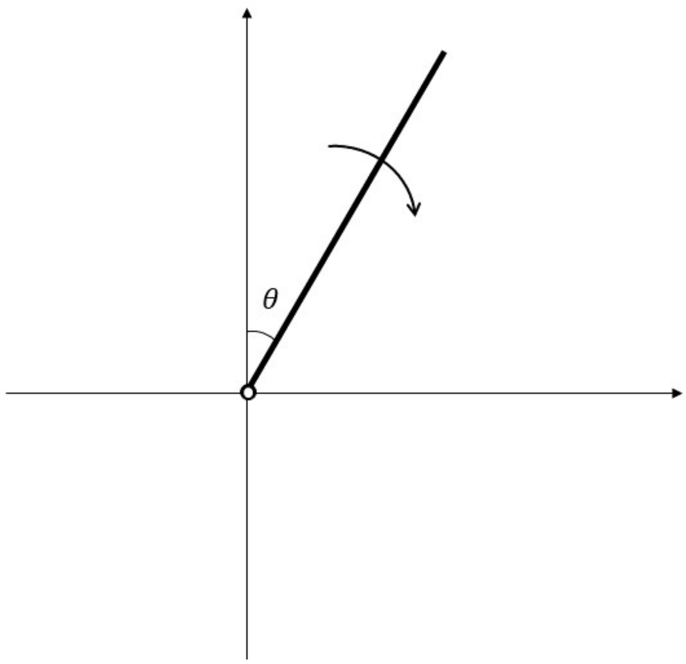

fig1. 倒立摆模型

使用前向欧拉方法计算新的角度和角速度来更新观测。 $$ \dot{\theta}k = \dot{\theta}{k-1} + \left[\frac{-3g}{2l} sin(\theta + \pi) + \frac{3T}{ml^2}\right]dt \tag{更新角速度}

\theta_k = \theta_{k-1} + \dot{\theta}_kdt \tag{更新角度} $$ 其中θ_k为第k次观测中倒立摆的角度，$\dot{\theta}_k$为第k次观测中倒立摆的角速度，T为施加在倒立摆上的扭矩，m为倒立摆的质量，l为倒立摆的摆长，g为重力加速度。

传统PID控制器

简单的传统PID控制器可以用下图来表示。

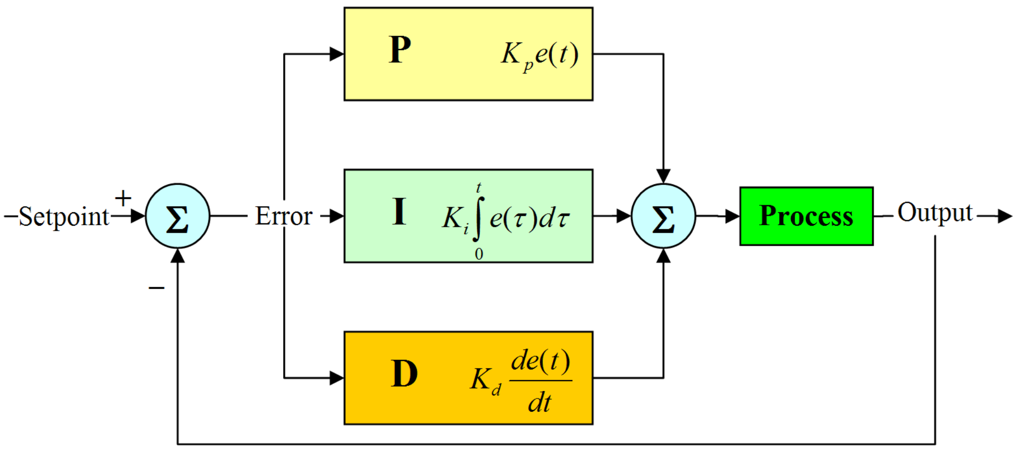

fig1. PID流程图

其中，误差可以用下面的公式表示。

$$e(t) = Setting(t) - feedback(t) \tag{误差}$$

利用计算的误差，输入到PID控制器中，得到输出按照如下公式计算。

$$O_{output} = K_p \cdot e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{PID 输出}$$

我们通过观察倒立摆的两个状态变量——角度和速度，也就是θ和$\dot\theta$来实现对倒立摆的状态的PID控制。因此误差可以表示为：

$$e(t) = \theta_{setting}(t) - \theta_{observed}(t)$$

从而建立起简单的pid模型。

利用模糊增益调整实现自适应PID控制器

模糊自适应PID控制器结构

具有模糊增益调整的PID控制器可以由下图的结构来表示。

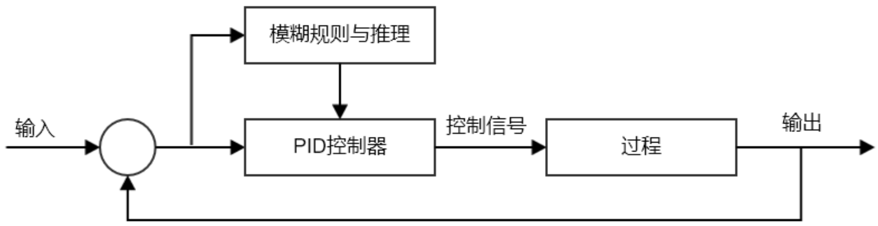

图3. 具有模糊增益调整的PID控制器

在这里我们设置PID的比例增益K_p、积分增益K_i和微分增益K_d都是可变的，同时需要一个变化范围，为$\left[K_{pmin}, K_{pmax}\right]$，$\left[K_{imin}, K_{imax}\right]$和$\left[K_{dmin}, K_{dmax}\right]$。注意，这里我并没有使用Ziegler-Nichols 调整规则，因为这样的调整规则会在这里介绍的倒立摆实验中使积分增益K_i变得非常大，因此我们采用模糊规则来对积分增益K_i进行调整，并且获得了非常好的效果。我会在实验部分给出实验数据进行对比。

模糊规则与推理过程建立

我们采用归一化参数，如下公式所示：

$$K_p' = \frac{K_p - K_{p,min}}{K_{p,max} - K_{p,min}}$$

$$K_i' = \frac{K_i - K_{i,min}}{K_{i,max} - K_{i,min}}$$

$$K_d' = \frac{K_d - K_{d,min}}{K_{d,max} - K_{d,min}}$$

参数由模糊规则决定：

$$If:\ e(K)是A_i和\Delta e(k)是B_i,\ then: K_p'是C_i, K_i'是D_i,\ K_d'是E_i$$

这里的A_i, B_i, C_i, D_i, E_i是在相应支集上的模糊集合。其隶属度如下图所示

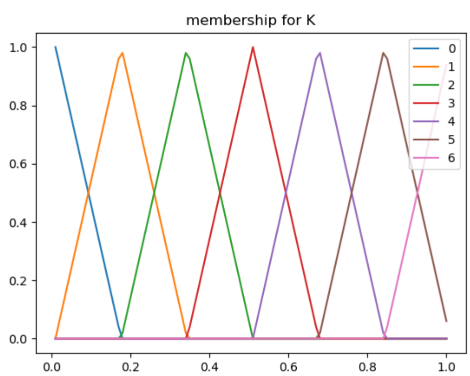

图4. 增益的模糊集合隶属度分布

$e(k)$和$\Delta e(k)$的隶属函数如下图所示。

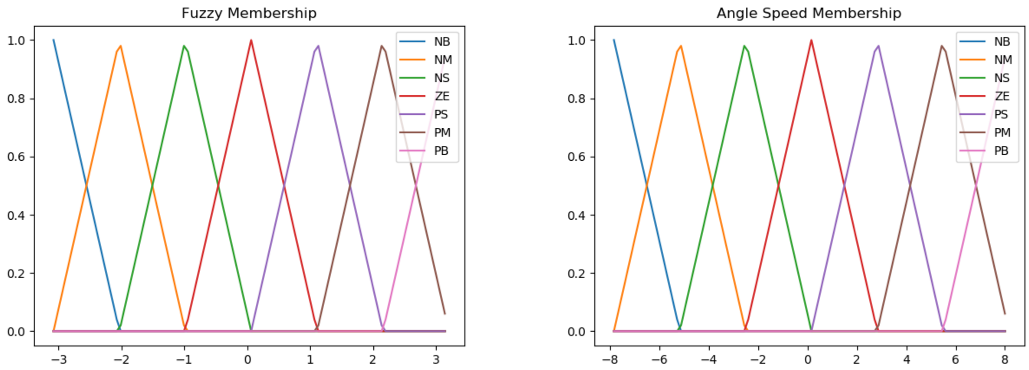

图4. 状态的模糊集合隶属度分布

我们确定了一系列准则，来保证pid控制算法的精确运行。准则可以由以下矩阵进行确定。

$$\Delta e(k) \& nb \quad nm \quad ns \quad ze \quad ps \quad pm \quad pb \;\; e(k) \;\; nb\backslash nm\backslash ns\backslash ze\backslash ps \pm \backslash pb \quad [\,3 \quad 4 \quad 5 \quad 6 \quad 5 \quad 4 \quad 3\; 2 \quad 3 \quad 4 \quad 5 \quad 4 \quad 3 \quad 2\, 1 \quad 2 \quad 3$$

$$\Delta e(k) \& nb \quad nm \quad ns \quad ze \quad ps \quad pm \quad pb \;\; e(k) \;\; nb\backslash nm\backslash ns\backslash ze\backslash ps \pm \backslash pb \quad [\,0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\; 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0\, 0 \quad 0 \quad 2$$

$$\Delta e(k) \& nb \quad nm \quad ns \quad ze \quad ps \quad pm \quad pb \;\; e(k) \;\; nb\backslash nm\backslash ns\backslash ze\backslash ps \pm \backslash pb \quad [\,3 \quad 2 \quad 1 \quad 0 \quad 1 \quad 2 \quad 3\; 4 \quad 3 \quad 2 \quad 1 \quad 2 \quad 3 \quad 4\, 5 \quad 4 \quad 3$$

这样的调整规则是基于经验获得的：

- 在$e(k)$较大、$\Delta e(k)$较小的情况下，比例环节的增益应该调大，积分和微分环节调整的较小位置；
- $e(k)$较小、$\Delta e(k)$较大的情况下，比例环节的增益应该调整较小，微分环节增益变大来缓解超调量，同时积分环节的比例调整到中等大小，来提高响应时间；
- $e(k)$较小、$\Delta e(k)$也较小的情况下，采用较大的积分增益和较小的微分、比例增益，减小系统的调节时间。

通过以上经验，我们获得了上述的矩阵，通过以上的调整规则，我们可以基于状态动态的得到K_p'，K_i'，K_d'。按照下面的公式计算PID的参数。

$$K_p = (K_{p,\max} - K_{p,\min})K_p' + K_{p,\min} \tag{K_p计算公式}$$

$$K_i = (K_{i,\max} - K_{i,\min})K_i' + K_{i,\min} \tag{K_i计算公式}$$

$$K_d = (K_{d,\max} - K_{d,\min})K_d' + K_{d,\min} \tag{K_d计算公式}$$

实验

实验环境

- 全部实验基于Python。
- 基于OpenAI的实验框架模型Gym，里面包含了很多控制模型，我们选用了里面的pendulum模型进行实验。
- Windows 10 系统环境

实验内容

我们通过python自行设计了传统PID控制系统和基于模糊推理的自适应PID控制器，进行了多次实验的对比和PID参数的调节，得到了以下实验结果。

PID控制器实验结果

我们随机设置了倒立摆的初始位置和初始速度进行了十次实验，通过PID控制器实验结果，得到了以下响应曲线。

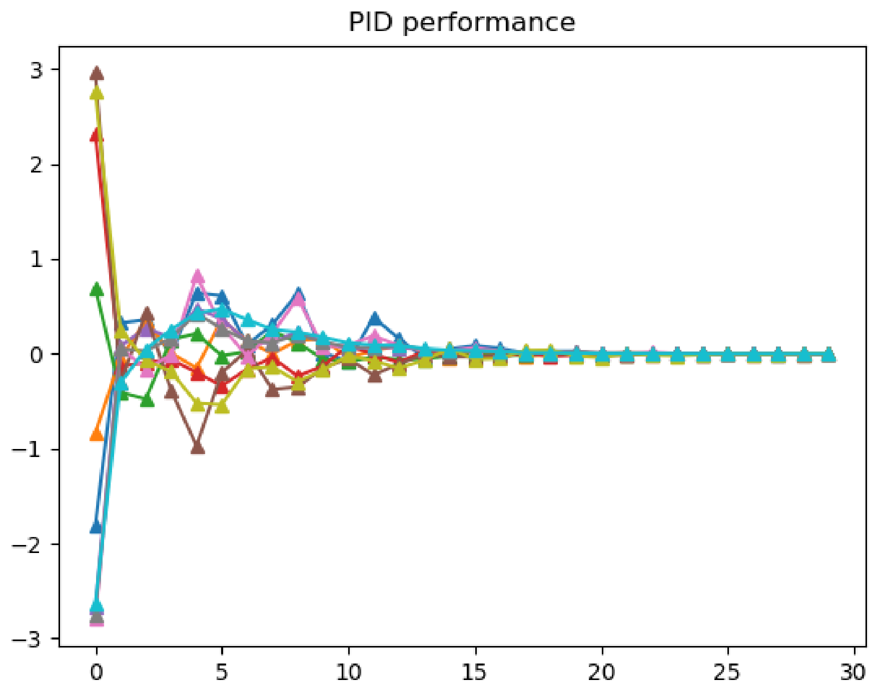

图5. 十次实验的经典PID控制结果,横坐标为时间，单位为秒，纵坐标为角度，单位是rad

基于模糊增益调整的自适应PID控制器实验结果

与经典PID控制器实验一样，我们随机设置初始状态并进行了十次实验，得到了如下响应曲线。

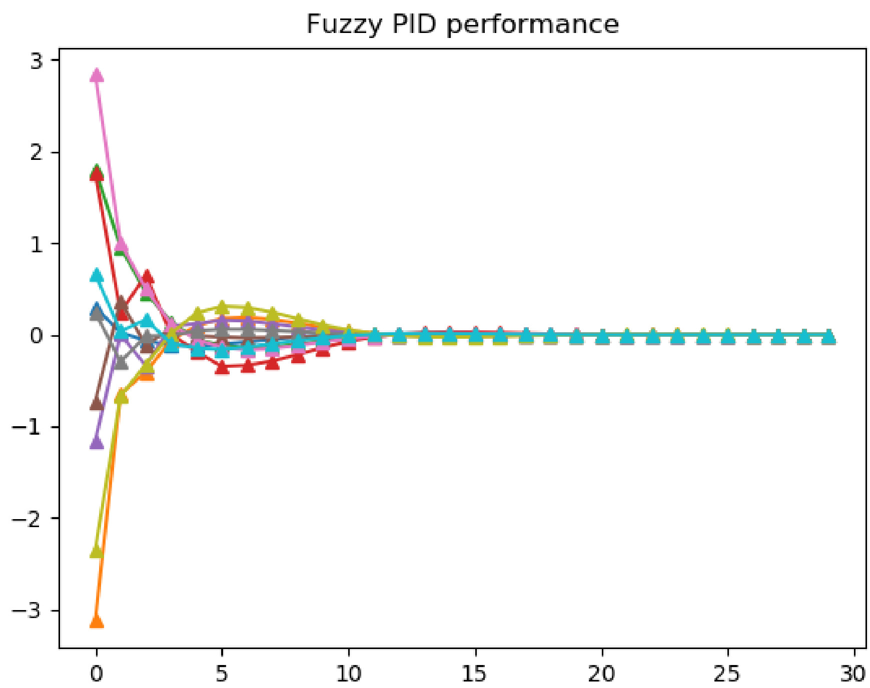

图5. 十次实验的模糊自适应PID控制结果，横坐标为时间，单位为秒，纵坐标为角度，单位是rad

实验代码

项目组织框架：

```
\src                    % 源代码
    __init__.py         % OOP初始化
    PID.py              % 经典PID控制器
    Fuzzy_PID.py        % 基于模糊增益调整的自适应PID控制器
\scripts                % 执行脚本
```

```
    fuzzy_result.py     % 自适应PID控制器脚本
    pid_result.py       % 经典PID控制器脚本
\test                   % 测试脚本
    gym_tester.py       % gym环境测试脚本
```

经典PID控制算法

```
'''
PID Controler In Python
Author: Ryan Zirui Zhao
Email: ryan_zzr@outlook.com

This is a simple PID control algorithm for python, using OOP.
'''

import time

class PID:

    def __init__(self, P = 0.2, I = 0, D = 0):
        '''
        Initialization.
        :param P: Proportional Parameter
        :param I: integral Parameter
        :param D: Derivative Parameter
        '''
        self.Kp, self.Ki, self.Kd = P, I, D
        self.sample_time = 0.0
        self.current_time = time.time()
        self.last_time = self.current_time
        self.clear()

    def clear(self):
        '''
        Clear all parameters.
        '''
        self.SetPoint = 0.0
        self.PTerm = 0.0
        self.ITerm = 0.0
        self.DTerm = 0.0
        self.last_error = 0.0

        self.int_error = 0.0
        self.windup_guard = 15.0

        self.output = 0.0

    def update(self, feedback_value):
        '''
        State Update.
        :param feedback_value: Current state value
        '''
        error = self.SetPoint - feedback_value

        self.current_time = time.time()
        delta_time = self.current_time - self.last_time
        delta_error = error - self.last_error

        if (delta_time >= self.sample_time):
            pTerm = self.Kp * error
            if (pTerm < -self.windup_guard):
                self.PTerm = -self.windup_guard
            elif (pTerm > self.windup_guard):
                self.PTerm = self.windup_guard
            else:
                self.PTerm = pTerm
            self.ITerm += self.Ki * error * delta_time
            if (self.ITerm < -self.windup_guard):
                self.ITerm = -self.windup_guard
            elif (self.ITerm > self.windup_guard):
                self.ITerm = self.windup_guard
            if delta_time > 0:
                self.DTerm = self.Kd * delta_error / delta_time
            if (self.DTerm < -self.windup_guard):
                self.DTerm = -self.windup_guard
            elif (self.DTerm > self.windup_guard):
                self.DTerm = self.windup_guard
            self.last_time = self.current_time
```

```
            self.last_error = error

            Output = self.PTerm + (self.ITerm) + (self.DTerm)
            if Output > 20:
                self.output = 20
            elif Output < -20:
                self.output = -20
            else:
                self.output = Output

    def setKp(self, Proportional_gain):
        self.Kp = Proportional_gain

    def setKi(self, Integral_gain):
        self.Ki = Integral_gain

    def setKd(self, derivative_gain):
        self.Kd = derivative_gain

    def setSampleTime(self, sample_time):
        self.sample_time = sample_time

    def setSetPoint(self, setpoint):
        self.SetPoint = setpoint
```

基于模糊增益调整的自适应PID控制器

```
import skfuzzy as sf
import time
import numpy as np
from math import pi, log

class Fuzzy_PID:

    def __init__(self, Pmax, Pmin, Imax, Imin, Dmax, Dmin):
        self.Kpmax = Pmax
        self.Kpmin = Pmin
        self.Kimax = Imax
        self.Kimin = Imin
        self.Kdmax = Dmax
        self.Kdmin = Dmin
        self.sample_time = 0.0
        self.current_time = time.time()
        self.last_time = self.current_time
        self.tfm = self.tfm_generator(-pi, pi)
        self.dtfm = self.tfm_generator(-8,8)
        self.re = self.rule()
        self.rde = self.re.T
        self.rie = self.rule_ki()
        self.a = self.rule_alpha()
        self.b = self.a.T
        self.clear()

    def tfm_generator(self, xmin, xmax):
        x = (xmax - xmin)/2

        NB = np.array([xmin, xmin, xmin+1/3*x], dtype = np.float)
        NM = np.array([xmin, xmin+1/3*x, xmin+2/3*x], dtype = np.float)
        NS = np.array([xmin+1/3*x, xmin+2/3*x, xmin+x], dtype = np.float)
        ZE = np.array([xmin+2/3*x, xmin+x, xmax - 2/3*x], dtype = np.float)
        PS = np.array([xmin+x, xmax-2/3*x, xmax-x/3], dtype = np.float)
        PM = np.array([xmax-2/3*x, xmax-x/3, xmax], dtype = np.float)
        PB = np.array([xmax - 1/3*x, xmax, xmax], dtype = np.float)

        return [NB, NM, NS, ZE, PS, PM, PB]

    def membership(self, x, tfm):
        x = np.array([x])
        return [sf.trimf(x, tfm[0]), sf.trimf(x, tfm[1]),sf.trimf(x, tfm[2]),\
            sf.trimf(x, tfm[3]),sf.trimf(x, tfm[4]),sf.trimf(x, tfm[5]),sf.trimf(x, tfm[6])]

    def rule(self):
        return np.matrix([[3,4,5,6,5,4,3],[2,3,4,5,4,3,2],[1,2,3,4,3,2,1],\
            [0,1,2,3,2,1,0],[1,2,3,4,3,2,1],[2,3,4,5,4,3,2],[3,4,5,6,5,4,3]])

    def rule_alpha(self):
        return np.matrix([[2,2,2,2,2,2,2],[3,3,2,2,2,3,3],[4,3,3,2,3,3,4],\
            [5,4,3,3,3,4,5],[4,3,3,2,3,3,4],[3,3,2,2,2,3,3],[2,2,2,2,2,2,2]])
```

```
    def rule_ki(self):
        return np.matrix([[0,0,0,0,0,0,0],[0,0,0,1,0,0,0],[0,0,2,2,2,0,0],\
            [0,2,4,2,4,2,0],[0,0,2,2,2,0,0],[0,0,0,1,0,0,0],[0,0,0,0,0,0,0]])

    def clear(self):
        self.SetPoint = 0.0
        self.PTerm = 0.0
        self.ITerm = 0.0
        self.DTerm = 0.0
        self.last_error = 0.0
        self.int_error = 0.0
        self.windup_guard = 10.0
        self.output = 0.0

    def update_K(self, error, d_error):
        self.Kp = self.re[np.argmax(self.membership(error,self.tfm)),\
            np.argmax(self.membership(d_error, self.dtfm))]/6 *(self.Kpmax-self.Kpmin)+self.Kpmin
        self.Kd = self.rde[np.argmax(self.membership(error, self.tfm)),\
            np.argmax(self.membership(d_error, self.dtfm))]/6 *(self.Kdmax-self.Kdmin)+self.Kdmin
        self.alpha = self.a[np.argmax(self.membership(error, self.tfm)),\
            np.argmax(self.membership(d_error, self.dtfm))]
        self.Ki = self.rie[np.argmax(self.membership(error, self.tfm)),\
            np.argmax(self.membership(d_error, self.dtfm))]/4 *(self.Kimax - self.Kimin)+self.Kimin

    def update(self, feedback_value, speed):
        error = self.SetPoint - feedback_value

        self.current_time = time.time()
        delta_time = self.current_time - self.last_time
        delta_error = error - self.last_error
        d_error = speed
        self.update_K(error, d_error)

        if (delta_time >= self.sample_time):
            pTerm = self.Kp * error
            if (pTerm < -self.windup_guard):
                self.PTerm = -self.windup_guard
            elif (pTerm > self.windup_guard):
                self.PTerm = self.windup_guard
            else:
                self.PTerm = pTerm
            self.ITerm += self.Ki * error * delta_time
            if (self.ITerm < -self.windup_guard):
                self.ITerm = -self.windup_guard
            elif (self.ITerm > self.windup_guard):
                self.ITerm = self.windup_guard
            if delta_time > 0:
                self.DTerm = self.Kd * delta_error / delta_time
            if (self.DTerm < -self.windup_guard):
                self.DTerm = -self.windup_guard
            elif (self.DTerm > self.windup_guard):
                self.DTerm = self.windup_guard
            self.last_time = self.current_time
            self.last_error = error

            Output = self.PTerm + (self.ITerm) + (self.DTerm)
            if Output > 15:
                self.output = 15
            elif Output < -15:
                self.output = -15
            else:
                self.output = Output

    def setKp(self, Pmax, Pmin):
        self.Kpmax = Pmax
        self.Kpmin = Pmin

    def setKd(self, Dmax, Dmin):
        self.Kdmax = Dmax
        self.Kdmin = Dmin

    def setKi(self, Imax, Imin):
        self.Kimax = Imax
        self.Kimin = Imin

    def setSampleTime(self, sample_time):
        self.sample_time = sample_time

    def setSetPoint(self, setpoint):
```

```
        self.SetPoint = setpoint
```

经典PID运行脚本

```
import skfuzzy
import time
import os
import sys
lib_path = os.path.abspath(os.path.join(sys.path[0], '..'))
sys.path.append(lib_path)
import gym
import matplotlib.pyplot as plt
from src.Fuzzy_PID import *
import math

Ctl = Fuzzy_PID(10,7,4,2,1.15, 0.75)
Ctl.setKp(10,3)
Ctl.setKi(9,0)
Ctl.setKd(0.9,0.3)
Ctl.setSampleTime(0.05)
Ctl.setSetPoint(1.1)
graph = []

env = gym.make('Pendulum-v0')
for i_episode in range(10):
    observation = env.reset()
    Ctl.clear()
    for t in range(300):
        env.render()
        feedback, thbot = env.state
        graph.append(feedback)
        Ctl.update(feedback, thbot)
        action = [Ctl.output]
        print(action)
        print(Ctl.PTerm, Ctl.ITerm,Ctl.DTerm)
        observation, reward, done, info = env.step(action)
    plt.plot(graph[::10], "^-")
    graph = []
plt.title("Fuzzy PID performance")
string = "../result/"+str(time.time())+"Fuzzy_graph.png"
plt.savefig(string)
env.close()
```

模糊自适应PID运行脚本

```
import os
import sys
lib_path = os.path.abspath(os.path.join(sys.path[0], '..'))
sys.path.append(lib_path)
import gym
import matplotlib.pyplot as plt
from src.PID import *
import math
import time

Ctl = PID()
Ctl.setKp(9)
Ctl.setKi(4)
Ctl.setKd(0.85)
Ctl.setSampleTime(0.05)
graph = []

env = gym.make('Pendulum-v0')
for i_episode in range(10):
    observation = env.reset()
    Ctl.clear()
    for t in range(300):
        env.render()
        print(observation)
        feedback, thbot = env.state
        graph.append(feedback)
        print(feedback)
        Ctl.update(feedback)
        action = [Ctl.output]
        print(action)
```

```
        print(Ctl.PTerm, Ctl.ITerm,Ctl.DTerm)
        observation, reward, done, info = env.step(action)
    plt.plot(graph[::10], "^-")
    graph = []
plt.title("PID performance")
string = "../result/"+str(time.time())+"PIDgraph.png"
plt.savefig(string)
env.close()
```