浙江大學
ZHEJIANG UNIVERSITY

智能控制技术
课程大作业

实验名称 双足机器人运动控制

姓 名 _____

学 号 _____

提交日期 January 12, 2025

指导老师 刘山

1 选题与建模

1.1 问题背景

随着机器人技术的不断进步，双足机器人在服务、救援、医疗康复等领域的应用潜力日益凸显。双足机器人因其类人结构能够适应复杂地形环境，表现出良好的灵活性和适应性，成为机器人学研究的重点方向之一。在实现双足机器人稳定行走的过程中，其复杂的动力学建模与精确的运动控制是关键技术挑战。例如，人体步态中各关节协作配合及姿态平衡的保持，需要借助复杂的算法进行实时计算和优化，以保障稳定性。

由于双足机器人系统动力学模型复杂，涉及多关节协调运动、重心控制及外部扰动等因素，实现精准建模和控制设计面临诸多挑战。为了降低模型复杂度，通常会对其进行适当简化，以便于研究其核心动态特性，从而为步态规划、姿态控制等提供支持。

双足机器人通常由髋关节、膝关节和踝关节构成，通过多个自由度的协调运动实现步态平衡与姿态控制。其单腿结构可类比为二级倒立摆模型，而整体系统可看作两个关联的二级倒立摆，通过模拟这种结构进行仿真，有助于分析其动态特性及控制策略。此外，步幅、步速与二级倒立摆模型中的小车位移和速度相对应，这种类比使得基于二级倒立摆的建模成为研究双足机器人运动特性的重要工具。

现实应用中，双足机器人的控制常采用传统的控制算法，如比例-微分控制（PD 控制）和比例-积分-微分控制（PID 控制），具有良好的安全性与可解释性。然而，随着机器人行走环境的复杂化，基于模糊控制、神经网络控制等智能控制方法逐渐受到关注。通过引入智能控制技术，能够在面对未知扰动和非线性问题时展现更强的自适应能力。

本研究将基于双足机器人及其类二级倒立摆模型进行仿真分析，结合传统控制方法和智能控制策略，探讨不同控制方案的优缺点。

1.2 问题建模

二级倒立摆结构如下图所示

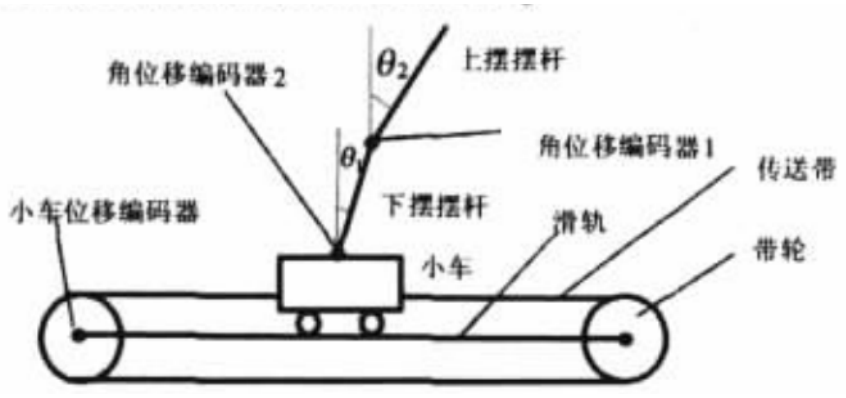


Figure 1: 二级倒立摆结构示意图

由上面问题背景的介绍可以得知，单腿结构可类比为二级倒立摆，因此其系统的动力学方程为:

$$M(\theta_1,\theta_2)\begin{bmatrix}\ddot{r}\\ \ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} = -F(\theta_1,\theta_2,\dot{\theta}_1,\dot{\theta}_2)\begin{bmatrix}\dot{r}\\ \dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} + N(\theta_1,\theta_2) + \begin{bmatrix}G_0\\ 0\\ 0\end{bmatrix} u$$

详细说明

- 加速度的计算: 通过求解线性方程组 $M \cdot \ddot{X} =$ 右端项，得到加速度向量 $\ddot{X}$。
- 非线性特性: 矩阵 $M(\theta_1,\theta_2)$ 和 $F(\theta_1,\theta_2,\dot{\theta}_1,\dot{\theta}_2)$ 包含了系统的非线性特性，因此整个状态空间方程是非线性的。

参数定义

质量矩阵 $M(\theta_1,\theta_2)$: 描述系统的惯性特性，表示各个状态变量间的加速度响应和相互耦合，由系统中各个组成部分的质量、转动惯量、几何参数等构成。

$$M(\theta_1,\theta_2) = \begin{bmatrix} M_0+M_1+M_2 & (M_1l_1+M_2L)\cos\theta_1 & M_2l_2\cos\theta_2 \\ (M_1l_1+M_2L)\cos\theta_1 & J_1+M_1l_1^2+M_2L^2 & M_2Ll_2\cos(\theta_2-\theta_1) \\ M_2l_2\cos\theta_2 & M_2Ll_2\cos(\theta_2-\theta_1) & J_2+M_2l_2^2 \end{bmatrix}$$

科氏力与离心力矩阵 $F(\theta_1,\theta_2,\dot{\theta}_1,\dot{\theta}_2)$: 描述系统中由于旋转运动产生的离心力和科氏力的影响，包含摆角及其角速度的函数，反映了系统的非线性和耦合特性。

$$F(\theta_1,\theta_2,\dot{\theta}_1,\dot{\theta}_2) = \begin{bmatrix} F_0 & -[(M_1l_1+M_2L)\sin\theta_1]\dot{\theta}_1 & -[M_2l_2\sin\theta_2]\dot{\theta}_2 \\ 0 & F_1+F_2 & -F_2-M_2Ll_2\sin(\theta_2-\theta_1)\dot{\theta}_2 \\ 0 & M_2Ll_2\sin(\theta_2-\theta_1)\dot{\theta}_1-F_2 & F_2 \end{bmatrix}$$

重力向量 $N(\theta_1,\theta_2)$: 描述系统中由于重力作用产生的力矩或力，决定了重力对系统状态的影响。

$$N(\theta_1,\theta_2) = \begin{bmatrix} 0 \\ (M_1l_1+M_2L)g\sin\theta_1 \\ M_2Lg\sin\theta_2 \end{bmatrix}$$

由于本实验是利用二级倒立摆模型去对双足机器人进行分析，二级倒立摆模型各参数值如下表所示，其中值是从论文中获得

符号	含义	取值（单位）
M_0	小车系统的等效质量	1.6 kg
M_1	下摆的质量	0.185 kg
M_2	上摆的质量	0.2 kg
F_0	系统的等效摩擦阻力系数	21.8519 N·m/s
F_1	下摆的摩擦阻力系数	0.006415 N·m/s
F_2	上摆的摩擦阻力系数	0.006717 N·m/s
L	下摆轴心到上摆轴心距离	0.483 m
l_1	下摆重心到其轴心距离	0.283 m
l_2	上摆重心到其轴心距离	0.245 m
G_0	控制力与控制电压之比	7.6889 N/V
J_1	下摆对其重心的转动惯量	0.00547 kg·m^2
J_2	上摆对其重心的转动惯量	0.00549 kg·m^2

Table 1: 系统参数说明表

1.2.1 控制目标

在二级倒立摆的模型中，输入变量 u 代表控制电压, 这个电压通过系统中的控制器（例如电动机驱动器）转换为作用在小车上的力，从而驱动小车运动。因此对于双足机器人系统来说，通过对输入变量 u 的控制，来实现对于位移以及两个关节角的控制。

在本次实验中，我们希望通过多种控制方案，控制电压 u，进而实现对于左右两足位移的控制。因此设计左右两足分别控制 X 的位移轨迹如下函数，实现在悬空状态时左右脚依次运动:

$$left(x) = \begin{cases} \sin(x), & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

$$Right(x) = \begin{cases} \sin(x + \pi), & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

2 问题分析

2.1 问题建模

系统的动力学方程为：

$$M(\theta_1,\theta_2)\begin{bmatrix}\ddot{r}\\ \ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} = -F(\theta_1,\theta_2,\dot{\theta}_1,\dot{\theta}_2)\begin{bmatrix}\dot{r}\\ \dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} + N(\theta_1,\theta_2) + \begin{bmatrix}G_0\\ 0\\ 0\end{bmatrix}u$$

我们需要将其转化为标准的状态方程形式

$$\dot{X} = f(X) + Bu$$

1. 状态变量定义

状态向量 X 定义为：

$$X = \begin{bmatrix} r & \theta_1 & \theta_2 & \dot{r} & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T$$

因此，系统的状态微分为：

$$\dot{X} = \begin{bmatrix} \dot{r} & \dot{\theta}_1 & \dot{\theta}_2 & \ddot{r} & \ddot{\theta}_1 & \ddot{\theta}_2 \end{bmatrix}^T$$

2. 计算加速度

将运动方程重写为：

$$\begin{bmatrix}\ddot{r}\\ \ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} = M^{-1}(\theta_1,\theta_2)\left(-F(\theta_1,\theta_2,\dot{\theta}_1,\dot{\theta}_2)\begin{bmatrix}\dot{r}\\ \dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} + N(\theta_1,\theta_2) + \begin{bmatrix}G_0\\ 0\\ 0\end{bmatrix}u\right)$$

3. 形成状态方程

令

$$Z = M^{-1}(\theta_1,\theta_2)\left(-F\cdot\dot{X}_{[3:5]} + N + Bu\right)$$

状态方程可写为：

$$\dot{X} = \begin{bmatrix} X_4 & X_5 & X_6 & Z_1 & Z_2 & Z_3 \end{bmatrix}^T$$

其中：

- X_4, X_5, X_6 分别为 $\dot{r}, \dot{\theta}_1, \dot{\theta}_2$；
- Z_1, Z_2, Z_3 对应 $\ddot{r}, \ddot{\theta}_1, \ddot{\theta}_2$。

因此可以得到完整的状态方程如下：

$$\dot{X} = \begin{bmatrix} X_4 \\ X_5 \\ X_6 \\ M^{-1}(\theta_1, \theta_2) \left(-F(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) \begin{bmatrix} X_4 \\ X_5 \\ X_6 \end{bmatrix} + N(\theta_1, \theta_2) + \begin{bmatrix} G_0 \\ 0 \\ 0 \end{bmatrix} u \right) \end{bmatrix}$$

2.2 特性及控制难点

1. 系统的非线性

二级倒立摆的动力学模型中存在大量非线性特征，包括：

- 三角函数项：如 $\cos(\theta_1)$、$\sin(\theta_1)$、$\cos(\theta_2)$ 等，描述了摆角的变化对系统动力学的影响。
- 乘积项：如 $M_2 L l_2 \cos(\theta_2 - \theta_1)$ 反映了上、下摆之间的相互作用。

2. 输出之间的耦合关系

摆角 θ_1 和 θ_2 之间存在耦合关系：

- 由于上下摆的物理连接，θ_1 的变化会影响 θ_2 的动态行为，反之亦然。
- 小车的水平位移 r 也会通过惯性和摆的相对位置影响两摆的角度变化。

3. 稳定性与鲁棒性挑战

- 平衡点的控制难度：二级倒立摆模型天然不稳定，系统在平衡状态下对微小扰动高度敏感，需要快速精准的反馈控制才能保持系统稳定。
- 扰动与不确定性：实际环境中可能存在外部扰动（如摩擦、风力）以及模型参数的不确定性，要求控制策略具有一定的鲁棒性。

2.3 控制方法

在本次实验中，分别利用 PID 控制、模糊控制、模糊 PID 控制、自适应神经控制、BP-PID 控制来对与该系统进行控制，实现位移量与预期值保持一致。

3 算法设计与实现

3.1 被控对象 S-Function 建模

根据上面求出的状态方程，我们可以构建 S-Function 模块代码如下所示

Listing 1: SFunction 代码

```
function [sys,x0,str,ts,simStateCompliance] = system1(t,x,u,flag)
%主函数
%主函数包含四个输出：
%               sys数组包含某个子函数返回的值
%               x0为所有状态的初始化向量
%               str是保留参数，总是一个空矩阵
%               Ts返回系统采样时间
%函数的四个输入分别为采样时间t、状态x、输入u和仿真流程控制标志变量flag
%输入参数后面还可以接续一系列的附带参数simStateCompliance
% 根据flag的值调用对应的回调函数
switch flag
    case 0
        % 初始化
        [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes;
    case 1
        % 计算连续状态导数
        sys = mdlDerivatives(t,x,u);
    case 2
        % 更新离散状态（如果有）
        sys = mdlUpdate(t,x,u);
    case 3
        % 计算输出
        sys = mdlOutputs(t,x,u);
    case 4
        % 下一个采样时间（用于可变步长仿真）
        sys = mdlGetTimeOfNextVarHit(t,x,u);
    case 9
        % 仿真结束前的清理工作
        sys = mdlTerminate(t,x,u);
    otherwise
        % 未处理的flag错误
        DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
```

```
33  end
34  % 主函数结束
35
36  %% %下面是各个子函数，即各个回调过程
37  function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes
38      %初始化回调子函数
39      %提供状态、输入、输出、采样时间数目和初始状态的值
40      %初始化阶段，标志变量flag首先被置为0，S-function首次被调用时
41      %该子函数首先被调用，且为S-function模块提供下面信息
42      %该子函数必须存在
43      sizes = simsizes;           %生成sizes数据结构，信息被包含在其中
44      sizes.NumContStates = 6;    %连续状态数，缺省为0
45      sizes.NumDiscStates = 0;    %离散状态数，缺省为0
46      sizes.NumOutputs    = 1;    %输出个数，缺省为0
47      sizes.NumInputs     = 1;    %输入个数，缺省为0
48      sizes.DirFeedthrough = 0; %是否存在直馈通道，1表示存在，0表示不存在
49      sizes.NumSampleTimes = 1; %采样时间个数，至少是一个
50      sys = simsizes(sizes);      %返回size数据结构所包含的信息
51      x0  = [0 0 0 0 0 0];          %设置初始状态
52      str = [];                   %保留变量置空
53      ts  = [0 0];                %设置采样时间
54      simStateCompliance = 'UnknownSimState';
55
56  % 计算导数回调子函数
57  function sys = mdlDerivatives(t, x, u)
58      % 计算连续状态导数
59
60      % 状态变量
61      r      = x(1);
62      theta1 = x(2);
63      theta2 = x(3);
64      r_dot  = x(4);
65      theta1_dot = x(5);
66      theta2_dot = x(6);
67
68      % 输入变量
69      u_input = u;  % 控制电压 u
70
```

```
71  % 系统参数
72  M0 = 1.6;          % 小车系统的等效质量 (kg)
73  M1 = 0.185;        % 下摆的质量 (kg)
74  M2 = 0.2;          % 上摆的质量 (kg)
75  F0 = 21.8519;      % 系统的等效摩擦阻力系数 (N*m/s)
76  F1 = 0.006415;     % 下摆的摩擦阻力系数 (N*m/s)
77  F2 = 0.006717;     % 上摆的摩擦阻力系数 (N*m/s)
78  L  = 0.483;        % 下摆轴心到上摆轴心距离 (m)
79  l1 = 0.283;        % 下摆重心到其轴心距离 (m)
80  l2 = 0.245;        % 上摆重心到其轴心距离 (m)
81  G0 = 7.6889;       % 控制力与控制电压之比 (N/V)
82  J1 = 0.00547;      % 下摆对其重心的转动惯量 (kg*m^2)
83  J2 = 0.00549;      % 上摆对其重心的转动惯量 (kg*m^2)
84  g  = 9.81;         % 重力加速度 (m/s^2)
85
86  % 构建 M 矩阵
87  M_matrix = [ ...
88      M0 + M1 + M2, (M1*l1 + M2*L)*cos(theta1), M2*l2*cos(theta2);
89      (M1*l1 + M2*L)*cos(theta1), J1 + M1*l1^2 + M2*L^2, M2*L*l2*cos(
            theta2 - theta1);
90      M2*l2*cos(theta2), M2*L*l2*cos(theta2 - theta1), J2 + M2*l2^2 ...
91      ];
92
93  % 构建 F 矩阵
94  F_matrix = [ ...
95      F0, - (M1*l1 + M2*L)*sin(theta1)*theta1_dot, - M2*l2*sin(theta2)*
            theta2_dot;
96      0, F1 + F2, -F2 - M2*L*l2*sin(theta2 - theta1)*theta2_dot;
97      0, M2*L*l2*sin(theta2 - theta1)*theta1_dot - F2, F2 ...
98      ];
99
100 % 构建 N 向量
101 N_vector = [ ...
102     0;
103     (M1*l1 + M2*L)*g*sin(theta1);
104     M2*L*g*sin(theta2) ...
105     ];
106
```

```
107     % 输入向量
108     input_vector = [G0; 0; 0] * u_input;
109 
110     % 计算加速度
111     acceleration = M_matrix \ ( -F_matrix * [r_dot; theta1_dot; theta2_dot
            ] + N_vector + input_vector );
112 
113     % 提取加速度
114     r_ddot     = acceleration(1);
115     theta1_ddot = acceleration(2);
116     theta2_ddot = acceleration(3);
117 
118     % 状态导数
119     dxdt = [ ...
120         r_dot;          % dr/dt
121         theta1_dot;     % dtheta1/dt
122         theta2_dot;     % dtheta2/dt
123         r_ddot;         % d(r_dot)/dt
124         theta1_ddot;    % d(theta1_dot)/dt
125         theta2_ddot     % d(theta2_dot)/dt
126         ];
127 
128     sys = dxdt;
129 
130 function sys=mdlUpdate(t,x,u)
131     %状态更新回调子函数
132     %给定t、x、u计算离散状态的更新
133     %每个仿真步内必然调用该子函数，不论是否有意义
134     %除了在此描述系统的离散状态方程外，还可以在此添加其他每个仿真步内都必须
            执%行的代码
135     sys = [];                   %sys表示下一个离散状态，即x(k+1)
136 
137 function sys=mdlOutputs(t,x,u)
138     %计算输出回调函数
139     %给定t,x,u计算输出，可以在此描述系统的输出方程
140     %该子函数必须存在
141     sys = [x(1)];                   %sys表示输出，即y
142 
```

```
function sys=mdlGetTimeOfNextVarHit(t,x,u)
    %计算下一个采样时间
    %仅在系统是变采样时间系统时调用
    sampleTime = 1;             %设置下一次采样时间是在1s以后
    sys = t + sampleTime;       %sys表示下一个采样时间点

function sys=mdlTerminate(t,x,u)
    %仿真结束时要调用的回调函数
    %在仿真结束时，可以在此完成仿真结束所需的必要工作
    sys = [];
```

3.2 普通 PID 控制

3.2.1 Simulink 仿真

先搭建了普通 PID 的 Simulink 模型，用 PID 控制器对左右脚的控制电压进行控制，与后面其他的智能控制方法形成对比。

在 Simulink 中搭建的仿真图如下图所示

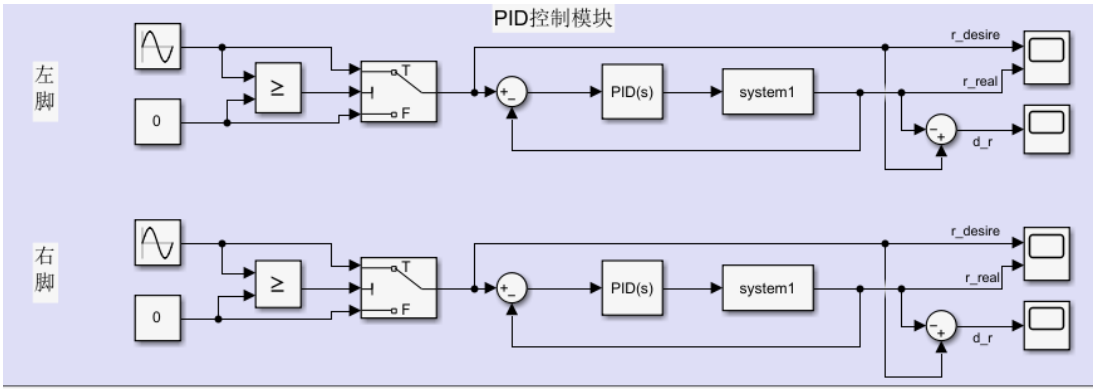


Figure 2: PID 控制

3.2.2 参数调整

在这个过程中，需要对 PID 的参数进行调整，我利用率 PID Controller 自带的**自动参数调节**（基于传递函数的调节方法）的功能，得到了如下图所示的参数

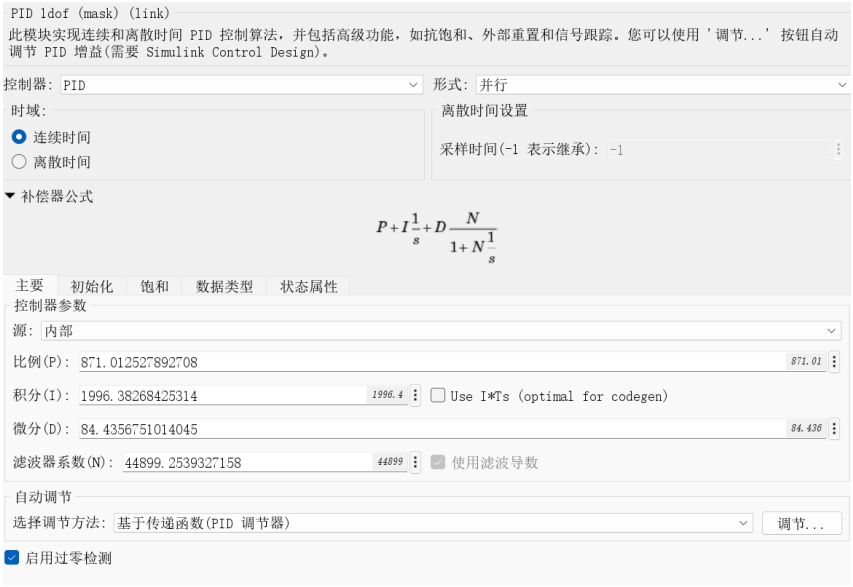


Figure 3: PID 参数

3.3 模糊控制

3.3.1 Simulink 仿真

利用 Matlab 的 Fuzzy 模块，对其进行仿真设计，具体的线路如下图所示

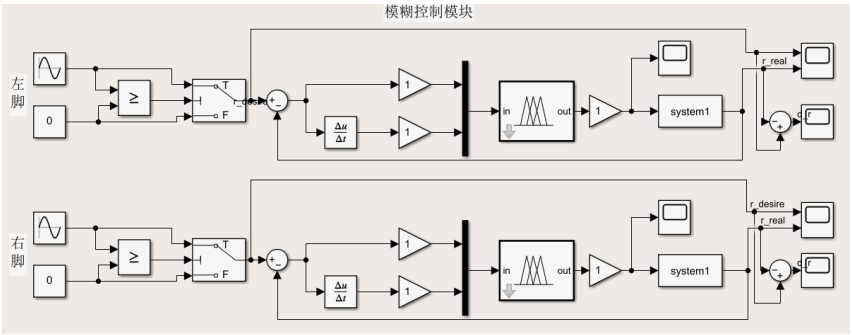
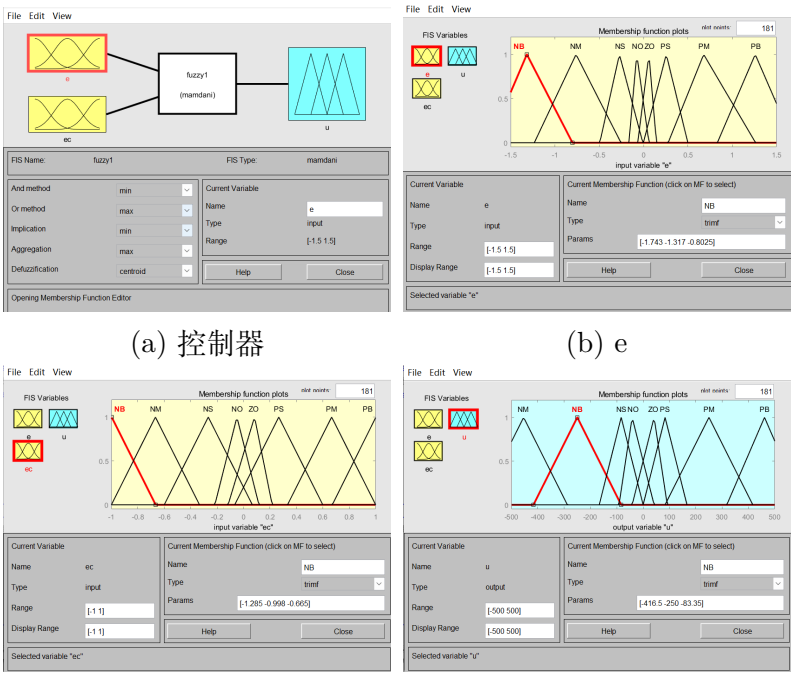


Figure 4: 模糊控制

3.3.2 隶属函数设计

将位置偏差 e、位置偏差变化率 e_c、模糊控制器的输出 u 模糊化，他们的模糊论域分别为 [-1.5，1.5]、[-1，1]、[-500，500]，隶属函数均取为三角形函数，语言变量值分别定为 NB(负大)、NM(负中)、NS(负小)、NO(负零)、Z0(正零)PS(正小)、PM(正中)、PB(正大)。因为磁悬浮系统是个控制精度要求比较高的系统，主要集中在 N0、Z0 两档附近对钢球进行精确定位控制，所以选择把输入 E、E_c, 的这两档设置的精度比较高，同理控制量输出 U 的模糊论域中心区域也设置成比较密。

在 Matlab 的模糊工具箱中建立的模糊推理系统 (FIS) 中 E、E_c、U 隶属函数如下图所示



(a) 控制器 (b) e

(c) ec (d) u

Figure 5: 隶属函数

根据建立模糊控制规则的基本思想, 并结合专家知识和 PID 控制经验进行分析、归纳，总结出磁悬浮模糊控制系统中输入 e、e_c 与输出 u 的控制规则表 根据上面的规则,

Ec \ E	NB	NM	NS	NO	ZO	PS	PM	PB
NB	NB	NB	NB	NB	PS	ZO	ZO	ZO
NM	NB	NM	NM	NB	PM	ZO	ZO	ZO
NS	NM	NM	NM	NB	PM	PS	PS	PS
NO	NM	NS	NS	NM	PM	PS	PS	PS
ZO	NS	NS	NS	NM	PB	PM	PS	PM
PS	NS	NS	NO	NM	PB	PM	PM	PM
PM	NS	NO	NO	NM	PB	PB	PM	PB
PB	NO	NO	NO	NO	PB	PB	PB	PB

Table 2: 控制规则

可以得到控制曲面如下图所示

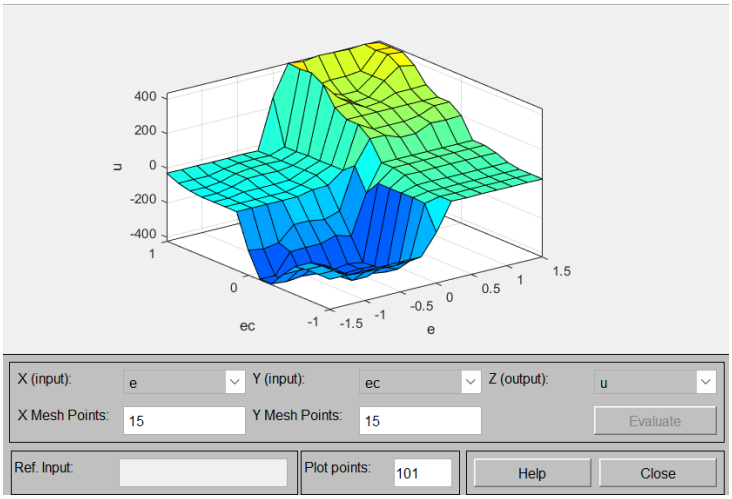


Figure 6: 控制曲面

对于解模糊来说，通过模糊推理得模糊控制输出 U，还必须通过解模糊才能得到确切的控制量，此处采用加权平均值（重心）法。

3.4 模糊 PID 控制

3.4.1 SImulink 仿真

Simulink 具体的线路如下图所示

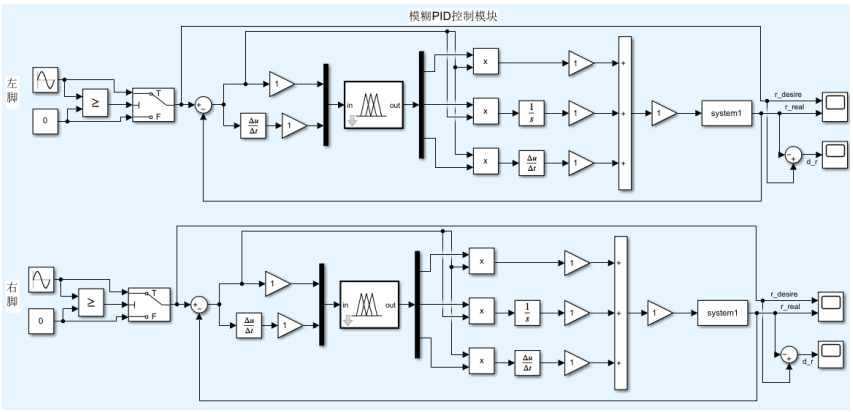


Figure 7: 模糊 PID 控制

3.4.2 隶属函数设计

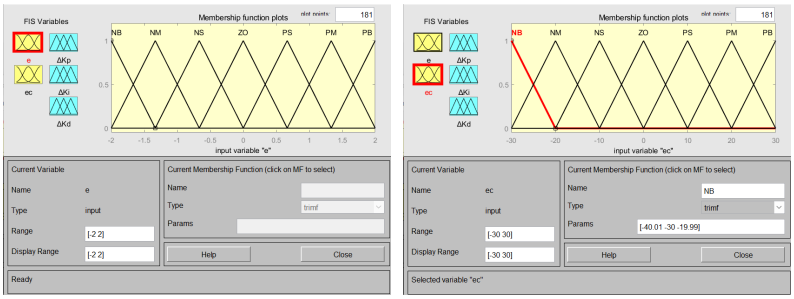
模糊 PID 在常规 PID 的基础上加设模糊参数自整定控制器使其根据系统的偏差的大小、方向、以及变化趋势等特征，通过模糊推理作出相应决策，自动的在线调整 PID 的三个参数 K_p、T_i 和 T_d，以便达到更加满意的控制效果的目的。模糊控制器的输入

为钢球位置偏差 e、钢球位置偏差变化率 e_c 输出量为 PID 参数的修正量 ΔK_p、ΔK_i、ΔK_d, 他们的隶属函数均取为三角形函数，模糊论域如下表所示，其中 ΔK_p、ΔK_i、ΔK_d 的论域参考了前面 PID 控制器调参出来的参数值，使其更加具有对照性质。

变量	e	e_c	ΔK_p	ΔK_i	ΔK_d
模糊论域	$[-2, 2]$	$[-30, 30]$	$[0, 900]$	$[-2000, 2000]$	$[0, 90]$
隶属函数	三角形				
模糊子集	[NB NM NS ZO PS PM PB]				

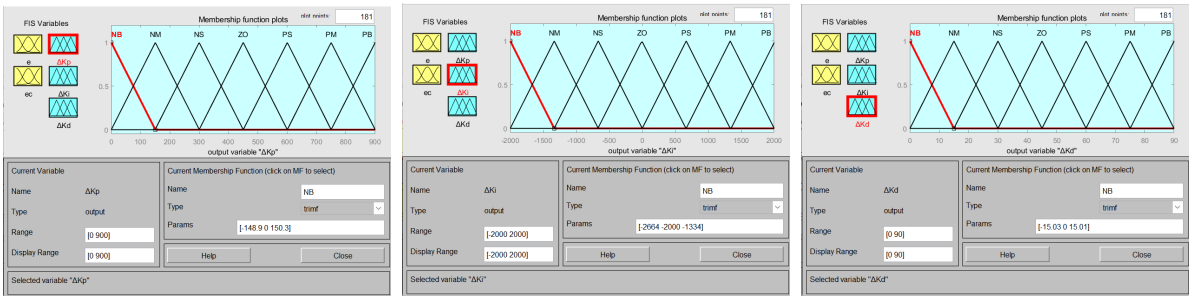
Table 3: 变量及模糊规则表

在 Matlab 的模糊工具箱中建立的模糊推理系统 (FIS) 中 E、E_c、ΔK_p、ΔK_i、ΔK_d 隶属函数如下图所示



(a) e (b) ec

Figure 8: 输入



(a) ΔK_p (b) ΔK_i (c) ΔK_d

Figure 9: 输出

他们的控制规则分别如下表所示

Ec \ E	NB	NM	NS	ZO	PS	PM	PB
NB	NB	NB	NM	ZO	ZO	ZO	PS
NM	NB	NM	NM	ZO	ZO	PS	PS
NS	NM	NM	NS	PS	PS	PS	PM
ZO	NM	NM	NS	ZO	PS	PM	PM
PS	NM	NS	NS	PS	PM	PM	PB
PM	NS	NS	ZO	PM	PM	PB	PB
PB	ZO	ZO	ZO	PM	PB	PB	PB

Table 4: ΔK_p 模糊控制规则表

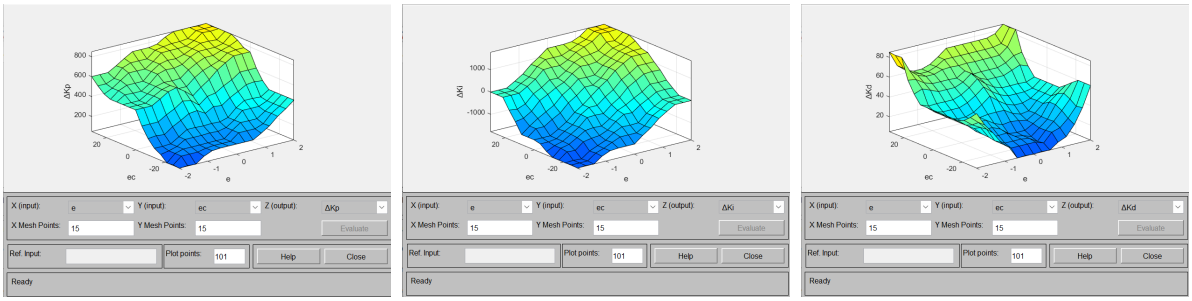
Ec \ E	NB	NM	NS	ZO	PS	PM	PB
NB	NB	NB	NB	NM	NM	ZO	ZO
NM	NB	NB	NM	NM	NS	ZO	ZO
NS	NM	NM	NS	NS	ZO	PS	PS
ZO	NM	NS	NS	ZO	PS	PS	PM
PS	NS	NS	ZO	PS	PS	PM	PM
PM	ZO	ZO	PS	PM	PM	PB	PB
PB	ZO	ZO	ZO	PM	PB	PB	PB

Table 5: ΔK_i 的模糊规则表

Ec \ E	NB	NM	NS	ZO	PS	PM	PB
NB	PS	ZO	ZO	ZO	ZO	PB	PB
NM	NS	NS	NS	NS	ZO	PS	PS
NS	NB	NM	NS	NS	ZO	PS	PS
ZO	NB	NM	NM	ZO	PS	PS	PM
PS	NB	NB	NM	NS	ZO	PS	PM
PM	NM	NS	NS	NS	ZO	PS	PM
PB	PS	PS	ZO	ZO	ZO	PB	PB

Table 6: ΔK_d 模糊规则表

可以得到控制曲面如下图所示



(a) ΔK_p (b) ΔK_i (c) ΔK_d

Figure 10: 控制面

3.5 自适应神经网络控制

3.5.1 SImulink 仿真

Simulink 具体的线路如下图所示

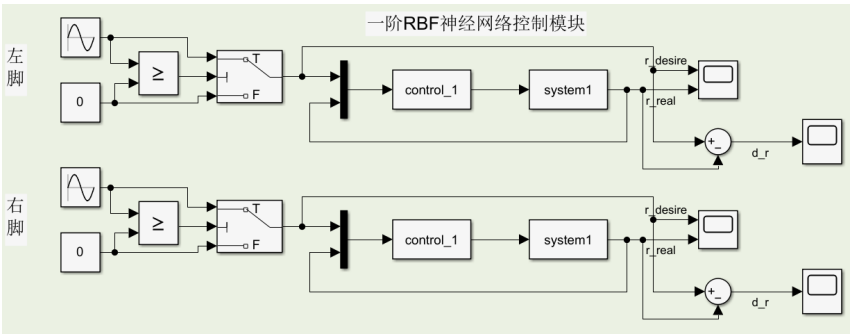


Figure 11: 自适应神经网络控制

3.5.2 神经网络结构

径向基函数（Radial Basis Function, RBF）神经网络是一种常用的前馈神经网络，特别适合于函数逼近、分类和时间序列预测等任务。RBF 网络具有良好的非线性逼近能力，能够在一个紧凑集内以任意精度逼近复杂的非线性函数。

RBF 网络通常由三个层次组成：

- **输入层**：接收输入数据，如状态向量 $[x, \dot{x}, \cdots, x^{(n-1)}]$。
- **隐藏层**：使用径向基函数作为激活函数，每个节点输出一个基函数值，以捕捉输入数据的局部特征。

常用的基函数是高斯函数：

$$h_j(x) = \exp\left(-\frac{\|x - c_j\|^2}{2b_j^2}\right)$$

其中：

- c_j 是基函数的中心；
- b_j 是基函数的宽度，决定函数的覆盖范围。

- **输出层**：对隐藏层输出进行线性加权求和，输出网络的最终结果：

$$\hat{f}(x) = \hat{W}^T h(x)$$

其中 $\hat{W}$ 是权重向量，$h(x)$ 是所有基函数的输出向量。

3.5.3 控制律与自适应律

控制律设计为：

$$u = \frac{1}{b}\left(-\hat{f}(x, \dot{x}, \cdots, x^{(n-1)}) + v - \eta \text{sgn}(s)\right)$$

其中：

- $v = y_d^{(n)} - K_1 e - K_2 \dot{e} - \cdots - K_n e^{(n-1)}$；
- s 为滑模面变量；
- η 为正的常数，用于克服逼近误差。

自适应律为：

$$\dot{\hat{W}} = \gamma h(x, \dot{x}, \cdots, x^{(n-1)}) s$$

其中：

- γ 为自适应速率；
- s 为滑模面变量；
- $h(x)$ 为 RBF 网络基函数的输出。

3.5.4 S-Function 函数

根据控制律编写对饮的 S-Function 函数如下

Listing 2: control_ 1 代码

```
function [sys,x0,str,ts,simStateCompliance] = control_1(t,x,u,flag)
% control_1.m

% 输入: u(1) = r_d (参考位置)
%       u(2) = r   (实际位置)
%
% 输出: sys = 控制输入 u(t)
%
% 连续状态: x(1..5) = W(1..5) (RBF 神经网络权重)

switch flag
    case 0
        [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes();
    case 1
        sys = mdlDerivatives(t,x,u);
    case 3
        sys = mdlOutputs(t,x,u);
    otherwise
        sys = [];
end

end % end of main function

function [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes()
    % 初始化
    sizes = simsizes;
    sizes.NumContStates = 5; % 5 个连续状态 -> 神经网络权重
    sizes.NumDiscStates = 0;
    sizes.NumOutputs    = 1; % 输出只有一个控制输入
    sizes.NumInputs     = 2; % 输入: [r_d, r]
    sizes.DirFeedthrough = 1; % 有直通
    sizes.NumSampleTimes = 1;

    sys = simsizes(sizes);
```

```
35
36      % 初始权重（可自行修改）
37      x0  = 0.1*ones(1,5);
38
39      str = [];
40      ts  = [0 0]; % 连续系统
41      simStateCompliance = 'UnknownSimState';
42
43      % 全局参数
44      global b c lambda gamma eta
45      b      = 0.1;    % RBF宽度
46      lambda = 13;     % 误差反馈增益
47      gamma = 500;     % 自适应增益
48      eta    = 3;      % 滑模开关项增益
49
50      % RBF 网络中心位置向量
51      c = [-2, -1, 0, 1, 2]; % 1 x 5
52  end
53
54  function sys = mdlDerivatives(t,x,u)
55
56      global b c lambda gamma
57
58      % 1. 误差 e = r - r_d
59      r_d = u(1);
60      r   = u(2);
61      e   = r - r_d;
62
63      % 2. 滑模面（一阶简化）s = e
64      s = e;
65
66      % 3. RBF 特征向量 h(r)
67      W  = x(1:5);
68      h  = zeros(5,1);
69      for j = 1:5
70          % c(j) 为该中心，尺寸 1 x 1
71          h(j) = exp( - (r - c(j))^2 / (2*b^2) );
72      end
```

```
73
74      % 4. 自适应律
75      dW = gamma * s * h;
76
77      sys = dW; % 返回列向量
78
79  end
80
81  function sys = mdlOutputs(t,x,u)
82      % 计算控制律
83
84      global b c lambda eta
85
86      % 提取输入信号
87      r_d = u(1);
88      r   = u(2);
89
90      % 1. 误差 e = r - r_d
91      e = r - r_d;
92
93      % 2. RBF 输出
94      W = x(1:5); % 当前的神经网络权重
95      h = zeros(5,1);
96      for j = 1:5
97          h(j) = exp( - (r - c(j))^2 / (2*b^2) );
98      end
99      fn = W'*h; % 神经网络对未知系统的估计
100
101     % 3. 控制律 (简化一阶滑模)
102     ut = -lambda*e - fn - eta*sign(e);
103
104     sys = ut; % 输出控制量
105
106 end
```

3.6 BP-PID 控制

3.6.1 SImulink 仿真

Simulink 具体的线路如下图所示

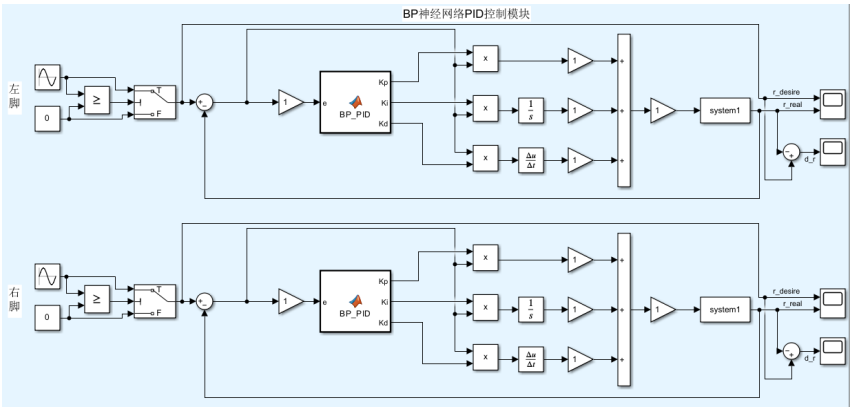


Figure 12: PID 神经网络控制

3.6.2 神经网络结构

神经网络的控制原理如下

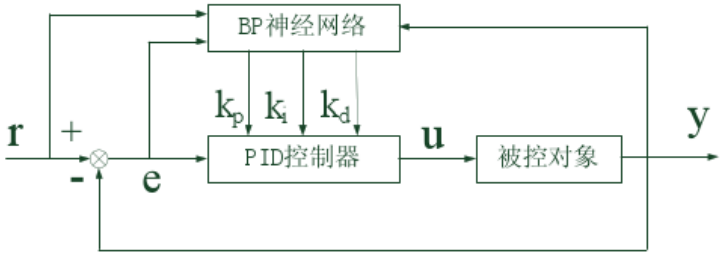


Figure 13: PID 神经网络控制原理

BP 神经网络 PID 控制是一种结合 BP（Back Propagation）神经网络与传统 PID 控制策略的自适应控制方法，通过神经网络的学习能力自动优化 PID 控制器的比例、积分和微分参数，以提升系统的动态性能和鲁棒性。

BP 神经网络的输出模型描述为：

$$y(t) = \sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x(t) + b_1\right) + b_2\right)$$

- $x(t)$：输入向量，即当前误差 e。
- $\sigma(\cdot)$：激活函数（如 Sigmoid、Tanh、ReLU 等）。
- W_1, W_2：网络的权重矩阵。

- b_1, b_2：网络的偏置项。
- $y(t)$：网络的输出，即动态调整后的 PID 参数 $\{K_p(t), K_i(t), K_d(t)\}$。

控制流程分为下面四部

(1). **前向传播**

- 输入误差 e 经过网络的隐藏层和输出层计算，输出 PID 参数 $\{K_p, K_i, K_d\}$。
- 使用激活函数 $\sigma(\cdot)$ 引入非线性特性，使网络能够对复杂系统建模。

(2). **误差计算**

- 计算神经网络输出与目标值之间的误差 Δy。
- 误差函数通常采用均方误差（MSE）：

$$E = \frac{1}{2} \sum (y_{\text{target}} - y_{\text{output}})^2$$

(3). **反向传播**

- 根据误差 Δy 通过梯度下降法更新权重和偏置。
- 计算权重和偏置的梯度：

$$\Delta W_2 = -\eta \cdot \frac{\partial E}{\partial W_2}$$

$$\Delta W_1 = -\eta \cdot \frac{\partial E}{\partial W_1}$$

- 其中，η 是学习率，控制参数更新的步幅。

(4). **参数更新**

- 更新网络权重和偏置：

$$W_1 = W_1 + \Delta W_1, \quad b_1 = b_1 + \Delta b_1$$

(5). **重复训练**

- 持续进行前向传播、误差计算、反向传播和参数更新，直到误差收敛。

根据其编写的 Matlab Function 代码如下

Listing 3: Matlab Function 代码

```
function [Kp, Ki, Kd] = BP_PID(e)
% BPNeuralPID - 使用 BP 神经网络在线调整 PID 参数的示例
```

```
3  %
4  % 输入:
5  %   e  : 当前位置误差
6  %
7  % 输出:
8  %   Kp : PID 比例增益
9  %   Ki : PID 积分增益
10 %   Kd : PID 微分增益
11 %
12 % 说明:
13 %   该函数使用一个两层的前馈 BP 神经网络 (1 个隐藏层)。
14 %   前向传播: e -> (隐藏层) -> [Kp, Ki, Kd]
15 %   反向传播: 根据训练误差调整网络权重和偏置。
16
17     %------------------------------------------------------
18     % 1. 定义输入向量
19     %------------------------------------------------------
20     inVec = e; % 单一输入: 误差 e
21
22     %------------------------------------------------------
23     % 2. 声明并初始化(若为空)网络参数: W1, b1, W2, b2
24     %   使用 persistent 来保存状态, 以便在每次函数调用之间 "记住" 网络参数
25     %------------------------------------------------------
26     persistent W1 b1 W2 b2 learning_rate
27
28     if isempty(W1)
29         %---------------网络结构超参数可自行修改---------------
30         input_size  = 1;   % 输入层神经元个数 (仅 e)
31         hidden_size = 5000; % 隐藏层神经元个数
32         output_size = 3;   % 输出层神经元个数 (Kp, Ki, Kd)
33
34         % 初始化权重和偏置 (建议较小的随机值以避免激活函数饱和)
35         W1 = rand(input_size, hidden_size) * 0.1;
36         b1 = rand(1, hidden_size) * 0.1;
37         W2 = rand(hidden_size, output_size) * 0.1;
38         b2 = rand(1, output_size) * 0.1;
39
40         learning_rate = 0.01; % 学习率, 可根据实际需求调整
```

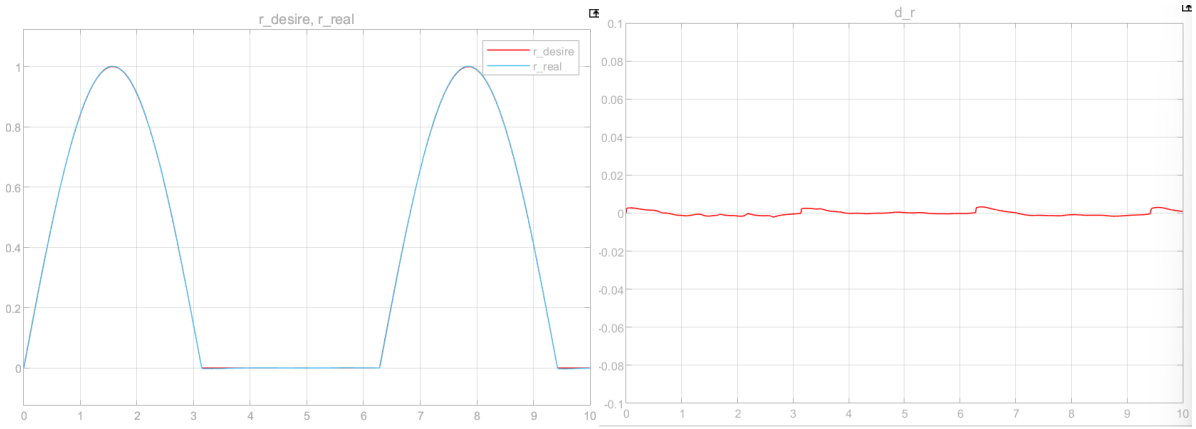
```
41      end
42
43      %-------------------------------------------------------
44      % 3. 前向传播 (Forward Pass)
45      %-------------------------------------------------------
46      % 隐藏层线性变换
47      z1 = inVec * W1 + b1;
48      % 隐藏层激活 (Sigmoid 或其他激活函数)
49      a1 = sigmoid(z1);
50
51      % 输出层线性变换 (可选择性添加激活函数)
52      z2 = a1 * W2 + b2;
53      a2 = z2; % 直接使用线性输出
54
55      % 从网络输出中获取 PID 参数
56      Kp = a2(1);
57      Ki = a2(2);
58      Kd = a2(3);
59
60      %-------------------------------------------------------
61      % 4. 计算训练误差并反向传播 (Backward Pass)
62      %-------------------------------------------------------
63      % 计算输出层误差
64      err_output = a2 - [Kp,Ki,Kd]; % 尺寸: 1 x 3
65
66      % 输出层对 z2 的偏导(线性激活, 导数为1)
67      dZ2 = err_output;
68      % 输出层权重梯度和偏置梯度
69      dW2 = a1' * dZ2;        % (hidden_size x 1) * (1 x output_size) = (
            hidden_size x output_size)
70      db2 = dZ2;              % (1 x output_size)
71
72      % 反向传播到隐藏层
73      dA1 = dZ2 * W2';        % (1 x output_size) * (output_size x
            hidden_size) = (1 x hidden_size)
74      % 隐藏层 sigmoid 激活的梯度
75      dZ1 = dA1 .* sigmoid_derivative(a1); % element-wise
76      % 隐藏层到输入层权重梯度和偏置梯度
```

```
77      dW1 = inVec' * dZ1;      % (1 x 1) * (1 x hidden_size) = (1 x
            hidden_size)
78      db1 = dZ1;               % (1 x hidden_size)
79
80      %-------------------------------------------------------
81      % 5. 更新网络权重和偏置
82      %-------------------------------------------------------
83      W2 = W2 - learning_rate * dW2;
84      b2 = b2 - learning_rate * db2;
85      W1 = W1 - learning_rate * dW1;
86      b1 = b1 - learning_rate * db1;
87
88      %-------------------------------------------------------
89      % 辅助函数: Sigmoid 及其导数
90      %-------------------------------------------------------
91      function y = sigmoid(x)
92          y = 1 ./ (1 + exp(-x));
93      end
94
95      function y = sigmoid_derivative(x)
96          y = x .* (1 - x);
97      end
98
99  end
```

4 结果比较与分析

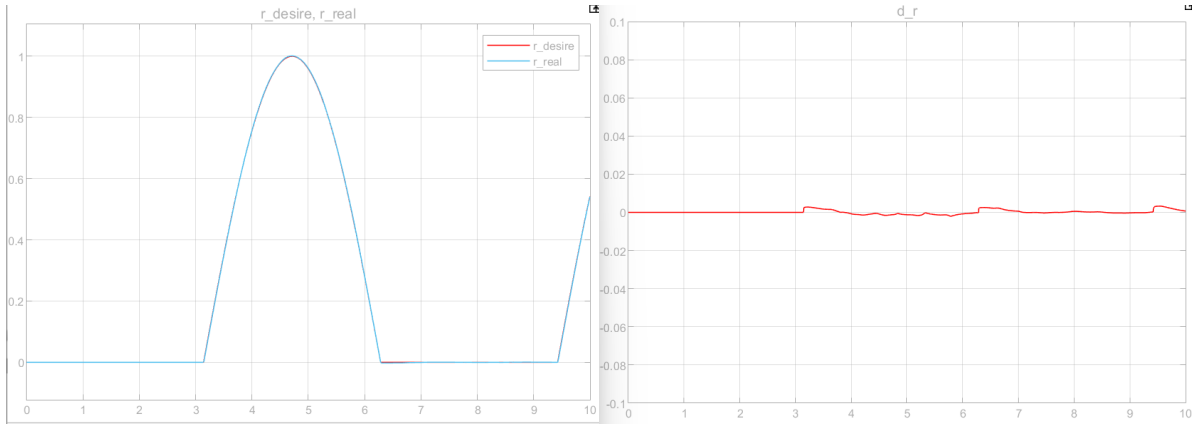
4.1 普通 PID 控制

在上面的参数的控制下，左脚与右脚得到了如下图所示的情况，r_desire 时预置位置曲线，r_real 是实际位置曲线，d_r 是两者偏差



(a) r_desire 与 r_real (b) 偏差 d_r

Figure 14: 左脚



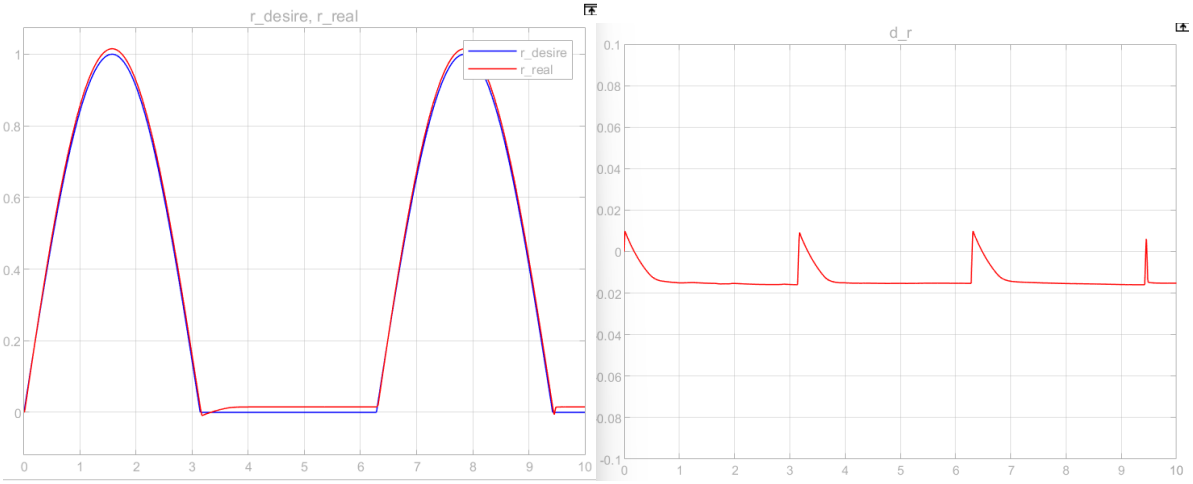
(a) r_desire 与 r_real (b) 偏差 d_r

Figure 15: 右脚

从图中可以看出，无论是左脚还是右脚，其偏差值很小，均在 [-0.01,0.01] 之内，可以忽略不计。

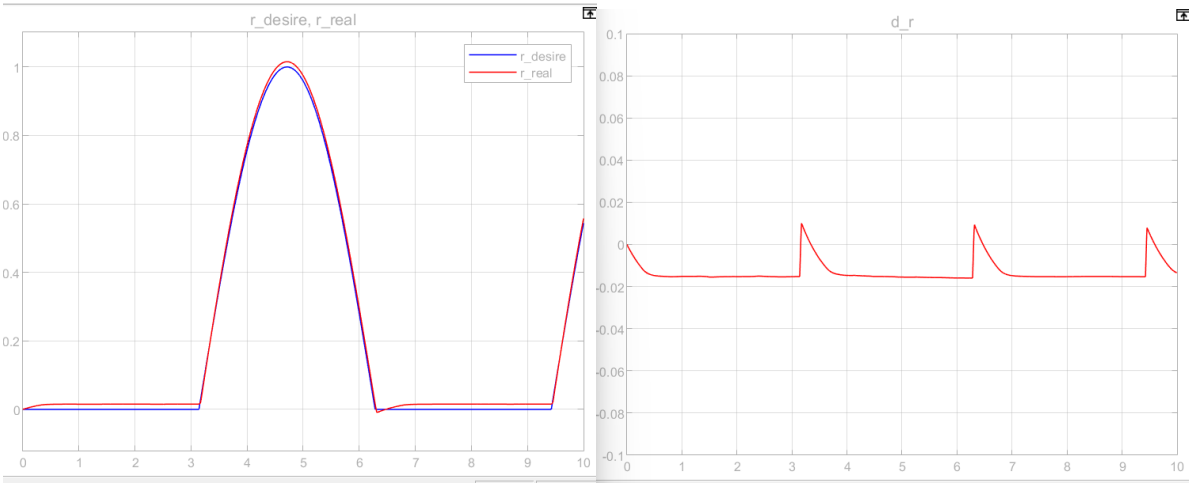
4.2 模糊控制

模糊控制的结果如下图所示



(a) r_desire 与 r_real (b) 偏差 d_r

Figure 16: 左脚



(a) r_desire 与 r_real (b) 偏差 d_r

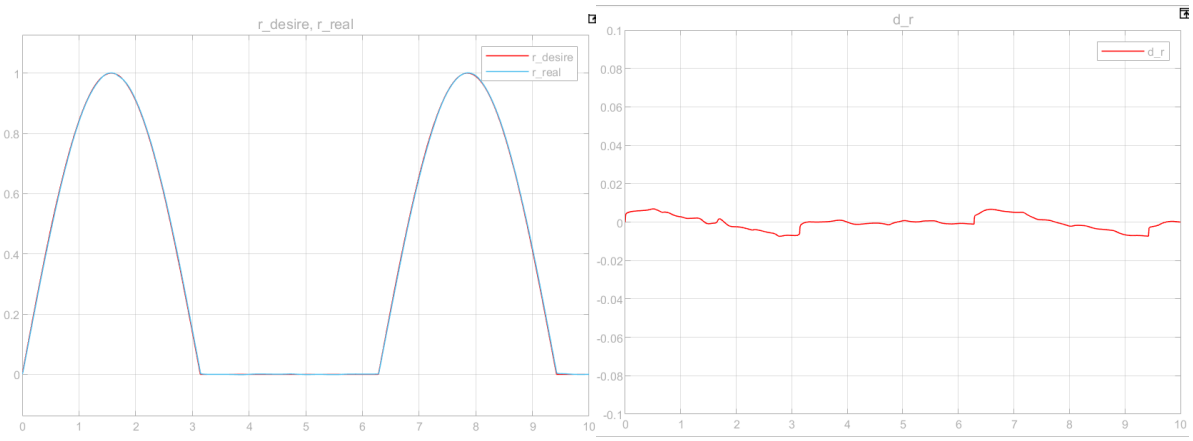
Figure 17: 右脚

从图中可以看到对于两脚，都存在着稳定的静差，且当预期值从 0 开始突变时，偏差会有突增，然后会降低回到原来的静差值左右。

这可能是由于在预期值突增时，模糊控制不能迅速的反应导致的，在很短的时间内，模糊控制的作用使得其又回归到静差值上。存在静差值是模糊控制导致的，这是无法消除的。

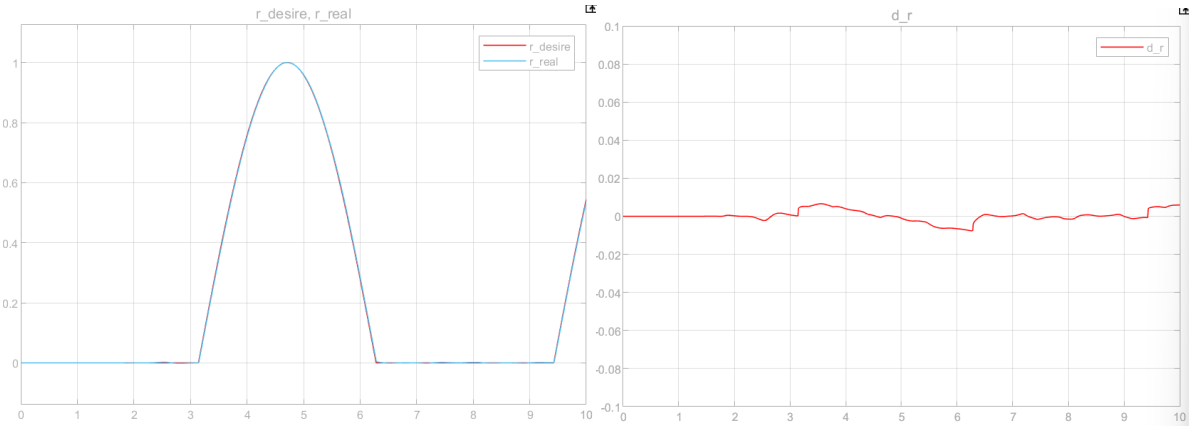
4.3 模糊 PID 控制

模糊 PID 控制的结果如下图所示



(a) r_desire 与 r_real (b) 偏差 d_r

Figure 18: 左脚



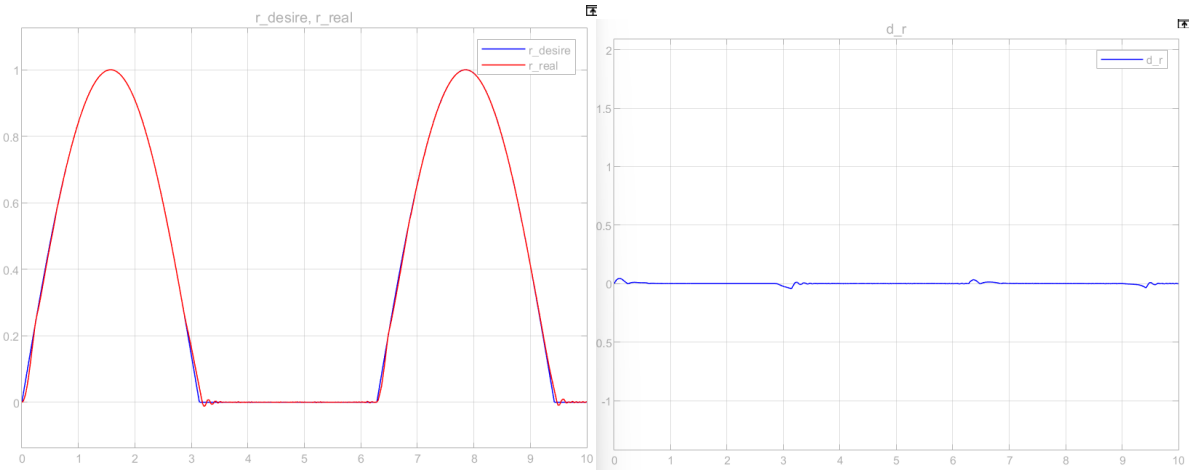
(a) r_desire 与 r_real (b) 偏差 d_r

Figure 19: 右脚

可以看到，在使用了 PID 之后，模糊控制自带的静差消失，误差较小，但是，相比普通 PID 而言，可以看到，模糊 PID 的偏差比普通 PID 的偏差略大一点，这可能是因为模糊规则或者论域设计不合适的问题。

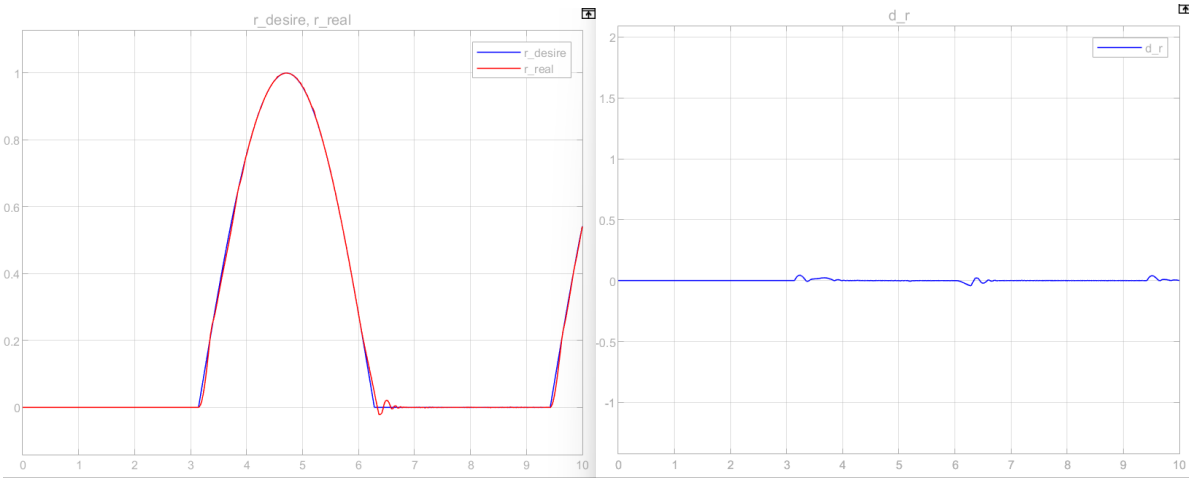
4.4 自适应神经网络控制

自适应神经网络控制的结果如下图所示



(a) r_desire 与 r_real (b) 偏差 d_r

Figure 20: 左脚



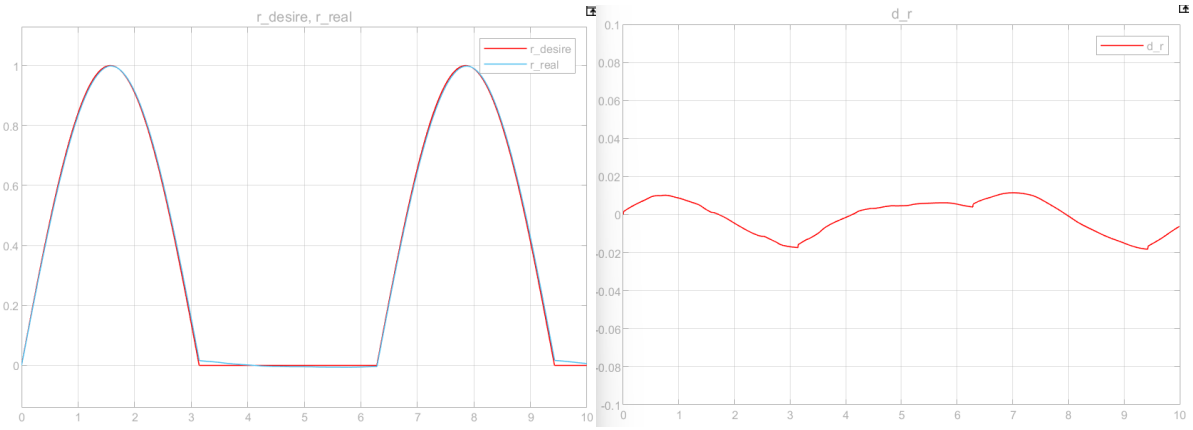
(a) r_desire 与 r_real (b) 偏差 d_r

Figure 21: 右脚

从图中可以看出自适应神经网络控制也是在预期值发生突变的时候存在一定的误差，其余时刻的值基本为 0。

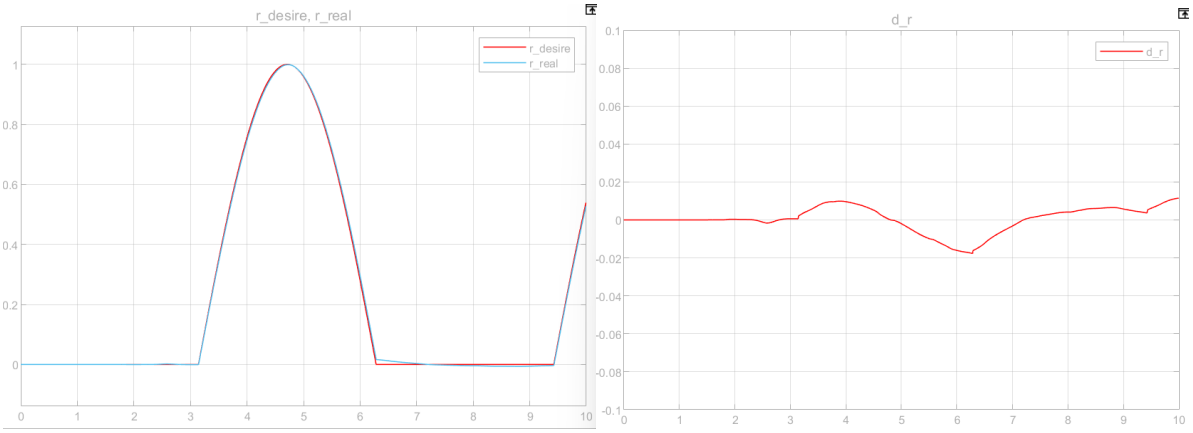
4.5 BP-PID 控制

BP-PID 控制的结果如下图所示



(a) r_desire 与 r_real (b) 偏差 d_r

Figure 22: 左脚



(a) r_desire 与 r_real (b) 偏差 d_r

Figure 23: 右脚

从图中可以看到，BP 神经控制相对于 RBF，在预期值从 0 突变时，误差比 RBF 要小。

4.6 总结

对于模糊控制而言，由于其存在静差，神经网络控制的性能优于模糊控制。同时，对于模糊 PID 控制而言，由于其 PID 的属性，因此其对于误差的控制能达到更小。

5 文件列表

文件名	功能描述
report.pdf	实验报告
README.md	文件结构说明
code	代码文件夹
system1.m	系统 S-Function 代码
control_ 1.m	RBF 自适应控制 S-Function 代码
fuzzy1.fis	模糊控制器
PIDfuzzy1.fis	模糊 PID 控制器
model.slx	Simulink 仿真文件
assert	参考文献

6 参考文献

[1] 贾荣丛, 高坤, 王划一. 双足机器人的倒立摆模型控制系统的研究 [J]. 微计算机信息,2009,25(29):193-194+183.

[2] 刘永泉, 郗安民, 刘鸿飞. 基于模糊控制的双足步行机器人控制研究 [J]. 微计算机信息,2010,26(14):15-16.

[3] 王强, 纪军红, 强文义, 等. 基于自适应模糊逻辑和神经网络的双足机器人控制研究 [J]. 高技术通讯,2001,(07):76-78+102.