Modelling and Simulation of finding PID gains using neural network with Simulink

Soumya Ranjan Das

April 9, 2024

1 Introduction

In the field of control engineering, Proportional-Integral-Derivative (PID) controllers serve as a cornerstone for regulating various systems across diverse domains. The effectiveness of a PID controller heavily relies on appropriate tuning of its parameters, commonly referred to as gains. Traditionally, the tuning process involves iterative adjustments based on system response analysis, which can be time-consuming and may not always yield optimal results, especially in complex or nonlinear systems.

In this project, I have made an approach for PID gain prediction utilizing neural networks and simulated in the Simulink environment. Simulink provides a powerful platform for modeling, simulating, and analyzing dynamic systems, offering a seamless integration of control algorithms with neural network-based predictors. By harnessing the synergies between control theory and machine learning, this methodology aims to streamline the PID tuning process, offering improved control performance and adaptability to varying system dynamics. This project has major three sections:

i) Plant

ii) Signal processing for extracting data at single point in time.

iii) Neural Network

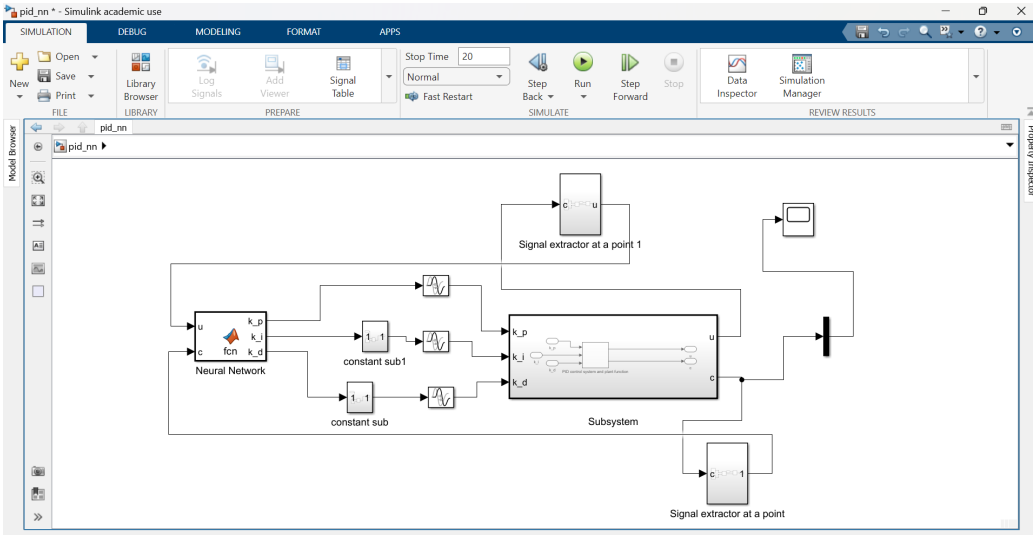


Figure 1: Block diagram in Simulink

2 Method

2.1 Plant Modelling

In real-world control applications, the precise mathematical model describing the dynamics of the system, commonly referred to as the plant function, is often elusive or difficult to obtain. Instead, we would be relying on observing system outputs, such as the control signal (u) and the corresponding process variable (c), which represents the system's behavior, such as height in a physical system.

In the Simulink environment, a model of the plant function was incorporated for simulation and verification purposes, as depicted in Figure 2. In Simulink the transfer function with degree of numerator greater than the degree of denominator is not allowed , So for laplace transform of derivative we took

$$\frac{Ns}{1+Ns}$$

where N was set in order of 1000 to make an approximation.

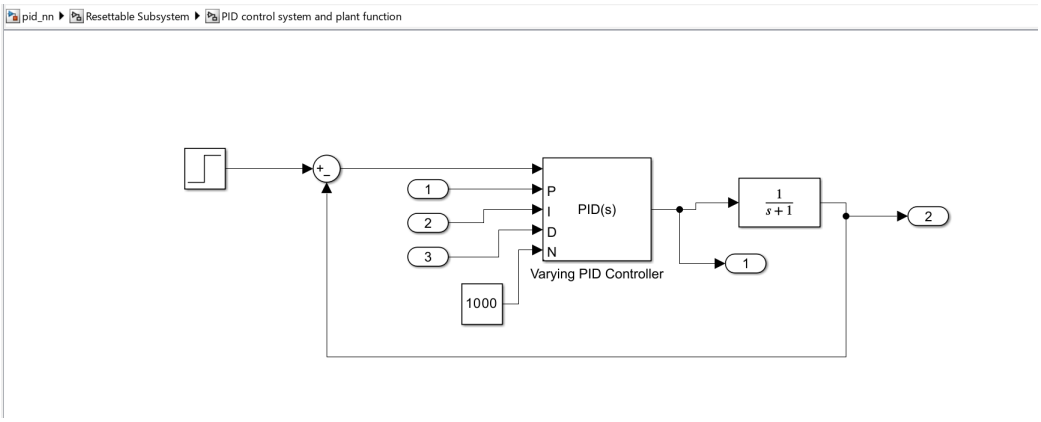


Figure 2: Plant model

2.2 Signal Processing for getting data at single timestamp

After we got the control signal and the output from the plant, the next step is to extract data at a single timestamp greater than zero and less than the saturation time for best results. The signal extractor block contains two time shifted unit step functions. The time shift is small to approximate that range to a point. The lagging step function is at the time stamp where we are interested in taking the data. The step functions are then subtracted by a subtraction block and the it was multiplied with the signal of interest of which we wanted to acquire data at a particular point. In figure 3 it shows the two signals at 1 second and 1.02 seconds subtracted to get an approximate unit pulse at 1 second which is then passed to a multiplier. The multiplier multiplies the signal with our generated approximate unit impulse. At approximately 1 second the the resulting signal follows the control signal but at other times its zero.

$$MultipliedOutput = \left\{ \begin{array}{ll} Input, & \text{if } time >= 1 \text{ and } time <= 1.02 \\ 0, & \text{else} \end{array} \right\} \quad (1)$$

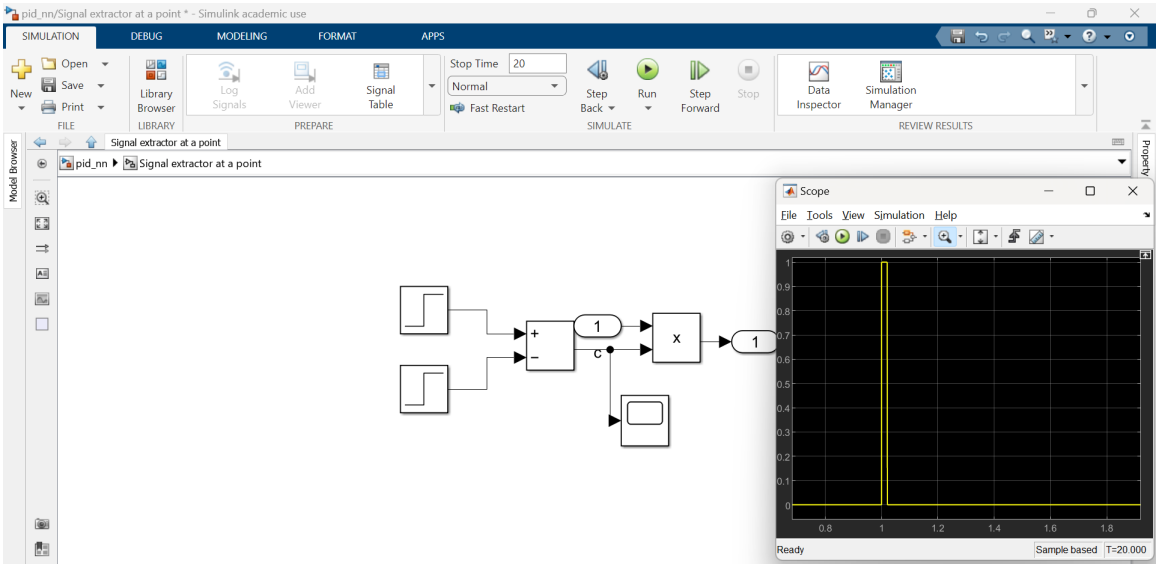


Figure 3: Signal extractor unit

2.3 Neural Network

After the extraction of the signal it is then feed to the matlab function block which contains matlab code . This contains code of a 2x3x3 neural network which uses the traditional feedforward algorithm i.e. each neuron performs a linear operation using weights and biases with input of previous layer and passes it through a sigmoid function. The application on controlling drones can be found on Yoon et al. (2016). For the detailed mathemathics one can refer to Jialiang et al. (2011).

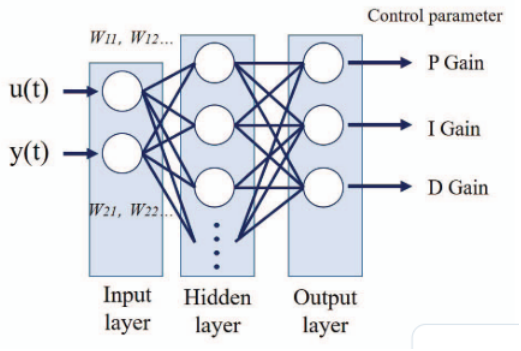


Figure 4: Neural Network Structure

$$O_j = f(net_j)$$

$$net_j = \sum_{i=1} W_{ji} O_i - \theta_j$$

$$f(x) = \frac{1}{1+e^{-x}}$$

Here θ is the bias and W is the weight matrix. The initial weights and biases was declared randomly typically each element within 0.5 to 0.6 range. This was the feedforward algorithm.

For backpropagation first the value of δ_k and δ_j was calculated where

$$\delta_k = \delta_j(\sum_k W_{kj})O_k .* (1 - O_k)$$

Here W_{kj} is the weights between hidden and output layer and O_k is the value of output neuron and .* denotes the elementwise multiplication operator .

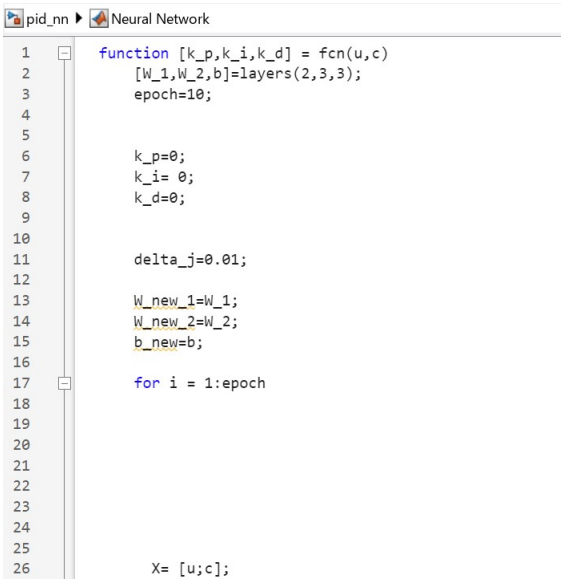
δ_j was initially set to a very small number but with every epoch it gets updated.

$$\delta_j = \delta_k(\sum_k W_{kj})O_j .* (1 - O_j)$$

Here W_{kj} is the weights between hidden and output layer and O_j is the value of hidden neurons and .* denotes the elementwise multiplication operator .

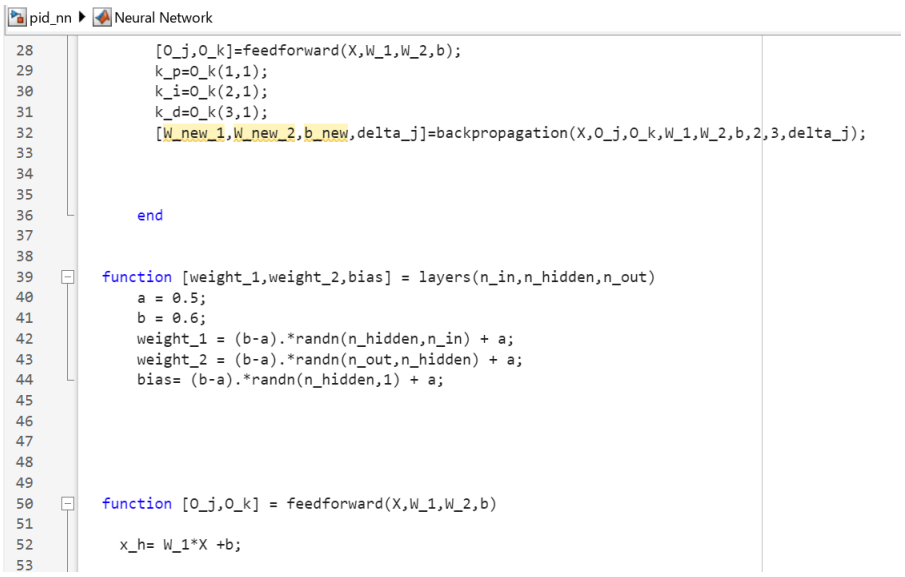
During every loop the initial value of $K_p, K_i and K_d$ is set to zero but according to the change in value of u and c it gives a prediction value of $K_p, K_i and K_d$ which is fed to the controller.

The code for neural network block is as given in figures 5 to 7.



```
pid_nn ▸ Neural Network

function [k_p,k_i,k_d] = fcn(u,c)
    [W_1,W_2,b]=layers(2,3,3);
    epoch=10;


    k_p=0;
    k_i= 0;
    k_d=0;


    delta_j=0.01;

    W_new_1=W_1;
    W_new_2=W_2;
    b_new=b;

    for i = 1:epoch









      X= [u;c];
```

Figure 5: Matlab code part 1



```
pid_nn ▸ Neural Network

      [O_j,O_k]=feedforward(X,W_1,W_2,b);
      k_p=O_k(1,1);
      k_i=O_k(2,1);
      k_d=O_k(3,1);
      [W_new_1,W_new_2,b_new,delta_j]=backpropagation(X,O_j,O_k,W_1,W_2,b,2,3,delta_j);



    end


function [weight_1,weight_2,bias] = layers(n_in,n_hidden,n_out)
    a = 0.5;
    b = 0.6;
    weight_1 = (b-a).*randn(n_hidden,n_in) + a;
    weight_2 = (b-a).*randn(n_out,n_hidden) + a;
    bias= (b-a).*randn(n_hidden,1) + a;







function [O_j,O_k] = feedforward(X,W_1,W_2,b)

  x_h= W_1*X +b;
```

Figure 6: Matlab code part 2

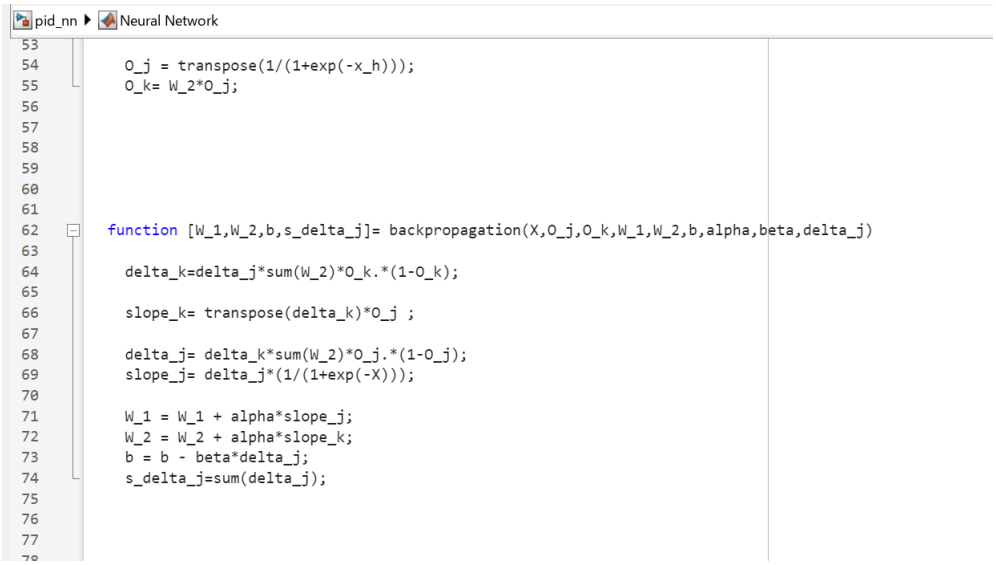


Figure 7: Matlab code part 3

3 Results

A minor further adjustment of the $K_d and K_i$ values might be required for reducing the oscillations. K_d should be increased from the predicted value and K_i must be decreased a little from the predicted value to reduce the oscillations.

The reference value was set to 5 in the control block and output or the variation of c was observed for plant function $\frac{1}{s} and \frac{1}{s^2+s+1}$. The results are shown in figure 8 and figure 9

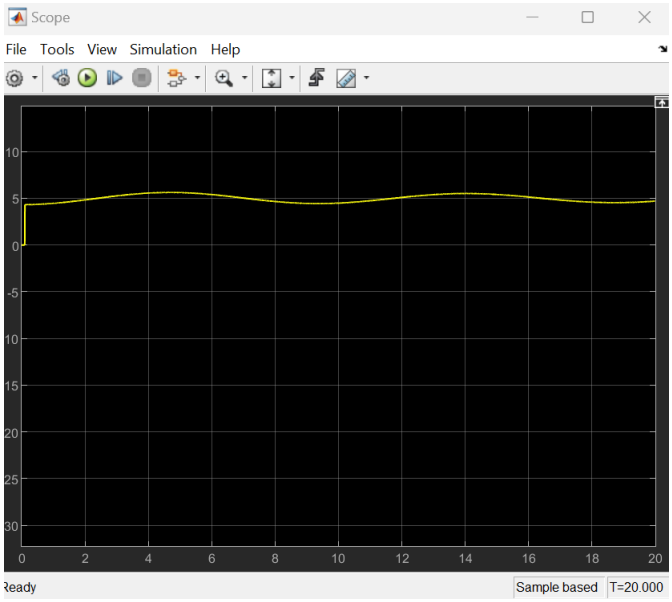


Figure 8: Result for plant function $\frac{1}{s}$

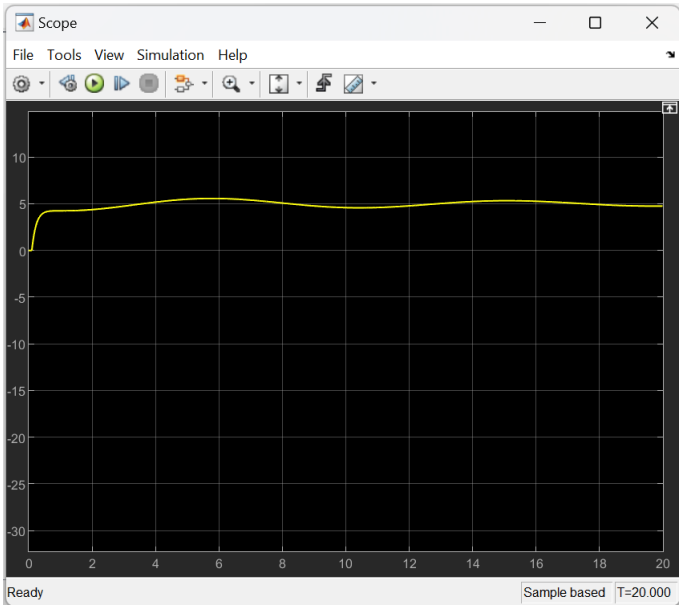


Figure 9: Result for plant function $\frac{1}{s^2+s+1}$

References

Jialiang, G., Zhimin, L., and Huaijiang, T. (2011). Research on self-tuning pid control strategy based on bp neural network. In *Proceedings of 2011 International Conference on Electronics and Optoelectronics*, volume 2, pages V2–16–V2–21.

Yoon, G.-Y., Yamamoto, A., and Lim, H.-O. (2016). Mechanism and neural network based on pid control of quadcopter. In *2016 16th International Conference on Control, Automation and Systems (ICCAS)*, pages 19–24.