Artificial Neural Network Controller

November 2, 2017

What is a neural network?

A neural network is a series of algorithms that attempts to identify underlying relationships in a set of data by using a process that mimics the way the human brain operates. Neural networks have the ability to adapt to changing input so the network produces the best possible result without the need to redesign the output criteria.

What is Back Propagation?

Backpropagation is a method used in artificial neural networks to calculate the error contribution of each neuron after a set of data (in image recognition, multiple images) is processed. Technically it calculates the gradient of loss function. It is also called backward propagation of errors, as the error is calculated at the output and distributed back through the network layers.

DESIGN OF OUR SYSTEM

The neural model reference control architecture uses two neural networks: a controller network and a plant model network, as shown in the following figure. The plant model is identified first, and then the controller is trained so that the plant output follows the reference model output. Then we run the system and begin to monitor the errors. Change in the system behavior is detected by comparing the mean and standard deviations of the errors similar to the CUSUM algorithm. Once a change in the system is detected, the system is run for a few more seconds to observe the change in the behavior and train the system emulator accordingly.

The fig(0.1) shows the block diagram of the neural network plant model and the neural network controller.

TRAINING THE EMULATOR AND CONTROLLER

First the controller and the emulator are initialized with random weights because of which the controller gives random $h(x)$ for which the system outputs $g(x)$.Using the $h(x)$ and $g(x)$ we train the system emulator using backpopogation algorithm and we check the consistency of the system emulator by observing the plant error for a short duration.

Now that the emulator is trained, it is time to train the controller. Let $f(x)$ be the reference control for the system, meaning for an input x given to the controller, the entire system when ideal gives an output $f(x)$. The controller needs to trained such that the control input produced, when sent to the system, the system output should be $h(x)$. As the system emulator behaves like the plant, it can be used instead of the system to train the controller.

The controller network is attached to the system emulator instead of the system. The error of the system with the reference input is backpropogated to the controller and the weights in the controller are adjusted keeping the weights in the emulator constant. In that way, the controller can be trained. For better understanding, refer to fig(0.3) and fig(0.4).

WHAT WE HAVE DONE SO FAR

- Initially we started learning about neural networks and methods to minimize errors like backpropagation.

- we have also gone through error detection methods like CuSUM method and ARX algorithm.
- Then we started learning matlab coding and simulation.
- After we were done with the basics we started designing a model for our project.
- Next we modeled DC motor in simulink and generated training data for training of the emulator.
- We wrote a MATLAB code for the emulator and got the emulator predictions. We had a little trouble choosing the activation layers and matrix multiplications in MATLAB as we had no prior experience.
- We evaluated the behavior of the system by using the data generated from the system previously and corrected the glitches.
- The next step was to turn the designed emulator into a function with the training data as input and network weights as output.
- Then we designed the controller and then evaluated its performance.

What next?

The next step in the project is to stitch the controller and emulator components together with the system and the error detection function so that the controller system can detect and adapt any changes in the system. Then, the controller system is ready to control nearly any the LTI system.

Advantages of using ANN as a Controller

Traditionally, designing a controller for a system needs a through study of its behavior. But using a neural network controller, the inputs and outputs of a system are enough to determine the behavior of nearly every LTI system and design a controller which can get the reference output from the system. The system parameters are not required to identify the behavior of any LTI system and mimic it. And even the controllers can be designed irrespective of the system parameters by just observing the input output characteristics.

EVALUATING THE PERFORMANCE USING DC MOTOR

- The DC motor model we are using, keeping the torque constant is given in the fig(0.2).
- First we generate a uniform random signal which is to be fed to the system as input voltage fig(0.5).
- Then we observe the system outputs and train the emulator using the input and output data obtained. In the figure(0.6), we can observe that the emulator is behaving nearly identical to the system.
- Then we train the controller model according to the reference model, which in this case is $f(x) = x$. The input given to the controller and the output from the system are shown in the fig(0.7).

CONTRIBUTION

- Raja Pradyumna(EE16BTECH11017) - Studied a book on neural networks B.Yenganarayana, designing the system, CuSUM algorithm for error detection, matlab basics, simulink basics, backpropagation, emulator codes in matlab.
- Anudeep(EE16BTECH11027) - Neural Network videos on YouTube, CuSuM algorithm for error detection, matlab basics, back propagation, emulator code in matlab.
- Aravind Reddy(EE16BTECH11004) - Neural network videos on YouTube, matlab basics, back propagation, controller code in matlab.
- Rishideep(EE16BTECH11031) - Neural network videos on Course era by Andrew NG, designing the system, matlab basics, back propagation, controller code in matlab.

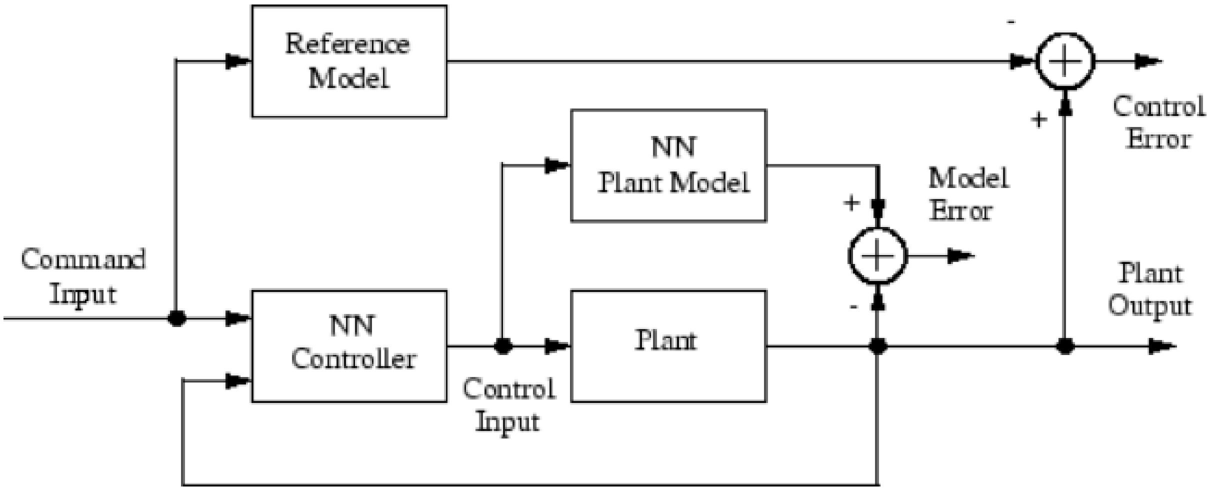


Figure 0.1: This the model of the system, system-emulator, controller model.

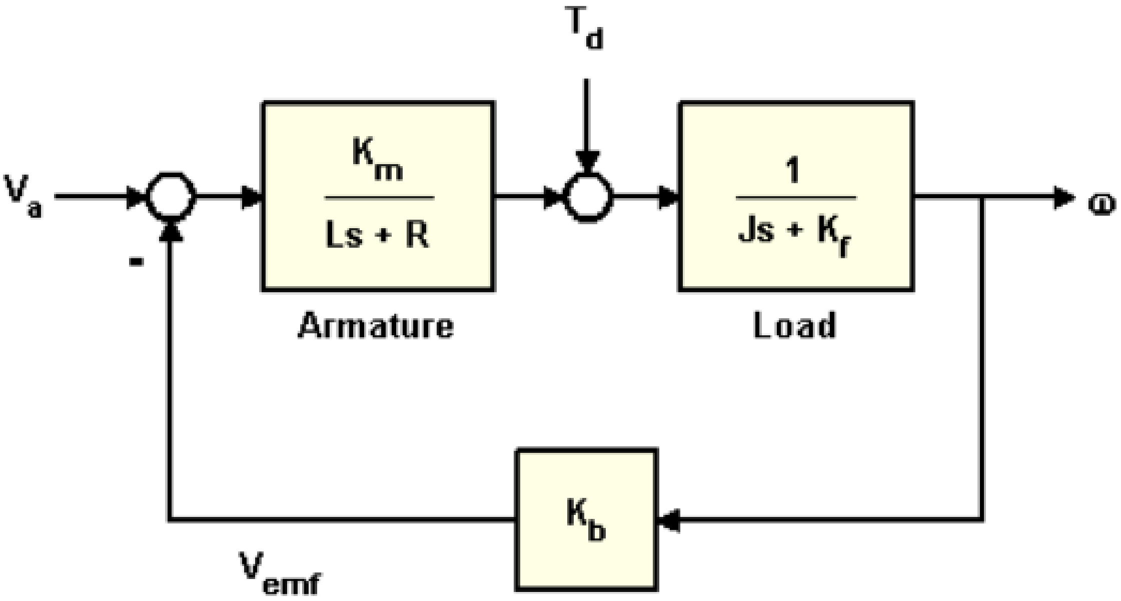
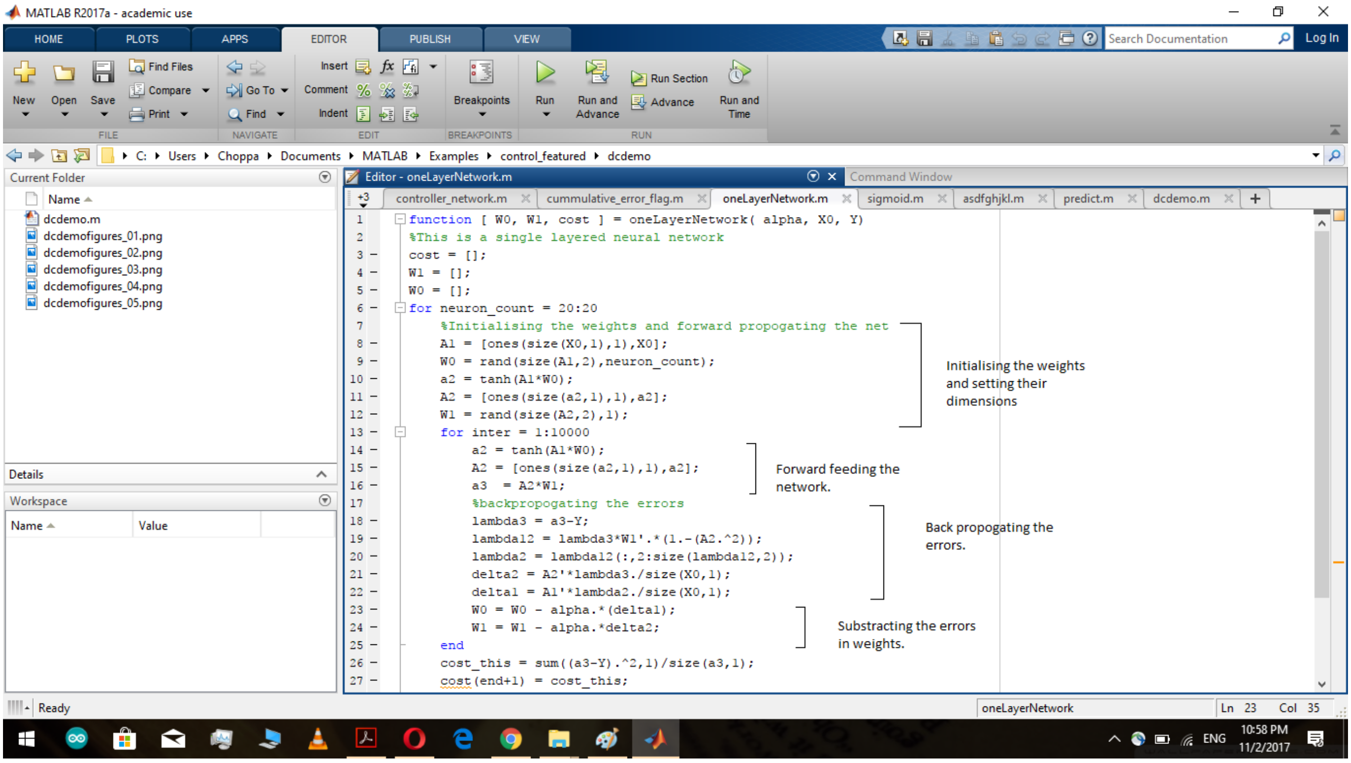


Figure 0.2: The DC motor model.



```
function [ W0, W1, cost ] = oneLayerNetwork( alpha, X0, Y)
%This is a single layered neural network
cost = [];
W1 = [];
W0 = [];
for neuron_count = 20:20
    %Initialising the weights and forward propogating the net
    A1 = [ones(size(X0,1),1),X0];
    W0 = rand(size(A1,2),neuron_count);
    a2 = tanh(A1*W0);
    A2 = [ones(size(a2,1),1),a2];
    W1 = rand(size(A2,2),1);
    for inter = 1:10000
        a2 = tanh(A1*W0);
        A2 = [ones(size(a2,1),1),a2];
        a3  = A2*W1;
        %backpropogating the errors
        lambda3 = a3-Y;
        lambda12 = lambda3*W1'.*(1.-(A2.^2));
        lambda2 = lambda12(:,2:size(lambda12,2));
        delta2 = A2'*lambda3./size(X0,1);
        delta1 = A1'*lambda2./size(X0,1);
        W0 = W0 - alpha.*(delta1);
        W1 = W1 - alpha.*delta2;
    end
    cost_this = sum((a3-Y).^2,1)/size(a3,1);
    cost(end+1) = cost_this;
```

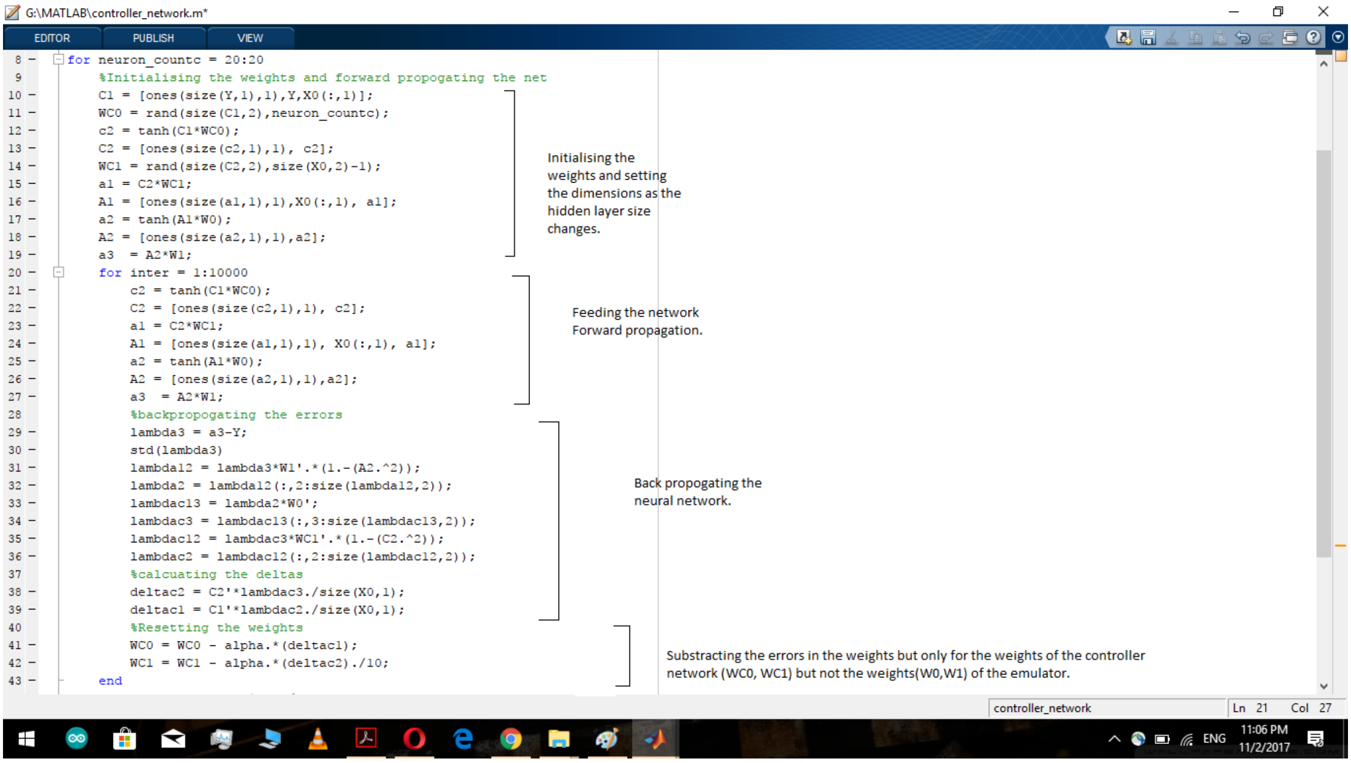
Initialising the weights and setting their dimensions

Forward feeding the network.

Back propogating the errors.

Substracting the errors in weights.

Figure 0.3: One Layer Network



```
for neuron_countc = 20:20
    %Initialising the weights and forward propogating the net
    C1 = [ones(size(Y,1),1),Y,X0(:,1)];
    WC0 = rand(size(C1,2),neuron_countc);
    c2 = tanh(C1*WC0);
    C2 = [ones(size(c2,1),1), c2];
    WC1 = rand(size(C2,2),size(X0,2)-1);
    a1 = C2*WC1;
    A1 = [ones(size(a1,1),1),X0(:,1), a1];
    a2 = tanh(A1*W0);
    A2 = [ones(size(a2,1),1),a2];
    a3  = A2*W1;
    for inter = 1:10000
        c2 = tanh(C1*WC0);
        C2 = [ones(size(c2,1),1), c2];
        a1 = C2*WC1;
        A1 = [ones(size(a1,1),1), X0(:,1), a1];
        a2 = tanh(A1*W0);
        A2 = [ones(size(a2,1),1),a2];
        a3  = A2*W1;
        %backpropogating the errors
        lambda3 = a3-Y;
        std(lambda3)
        lambda12 = lambda3*W1'.*(1.-(A2.^2));
        lambda2 = lambda12(:,2:size(lambda12,2));
        lambdac13 = lambda2*W0';
        lambdac3 = lambdac13(:,3:size(lambdac13,2));
        lambdac12 = lambdac3*WC1'.*(1.-(C2.^2));
        lambdac2 = lambdac12(:,2:size(lambdac12,2));
        %calcuating the deltas
        deltac2 = C2'*lambdac3./size(X0,1);
        deltac1 = C1'*lambdac2./size(X0,1);
        %Resetting the weights
        WC0 = WC0 - alpha.*(deltac1);
        WC1 = WC1 - alpha.*(deltac2)./10;
    end
```

Initialising the weights and setting the dimensions as the hidden layer size changes.

Feeding the network Forward propagation.

Back propogating the neural network.

Substracting the errors in the weights but only for the weights of the controller network (WC0, WC1) but not the weights(W0,W1) of the emulator.

Figure 0.4: Controller

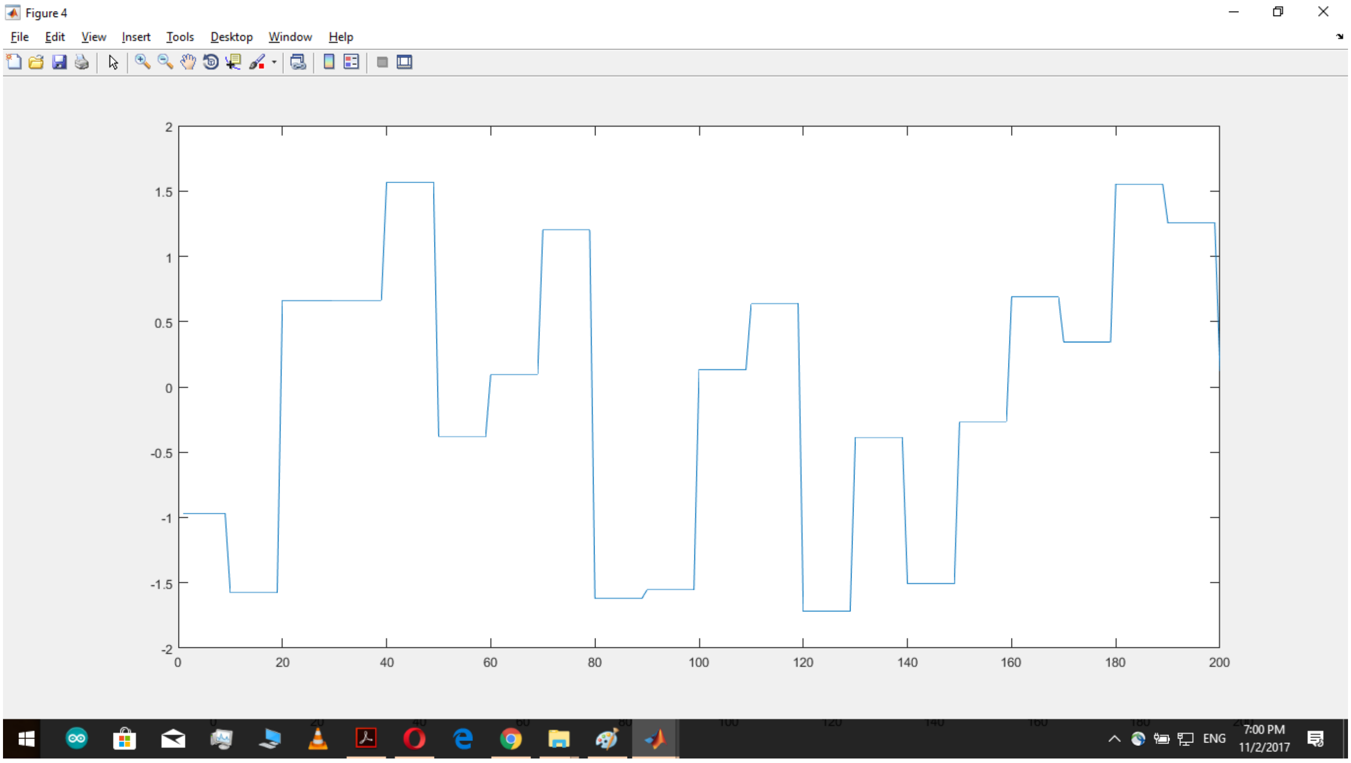


Figure 0.5: This graph shows the variation in input voltage with time.

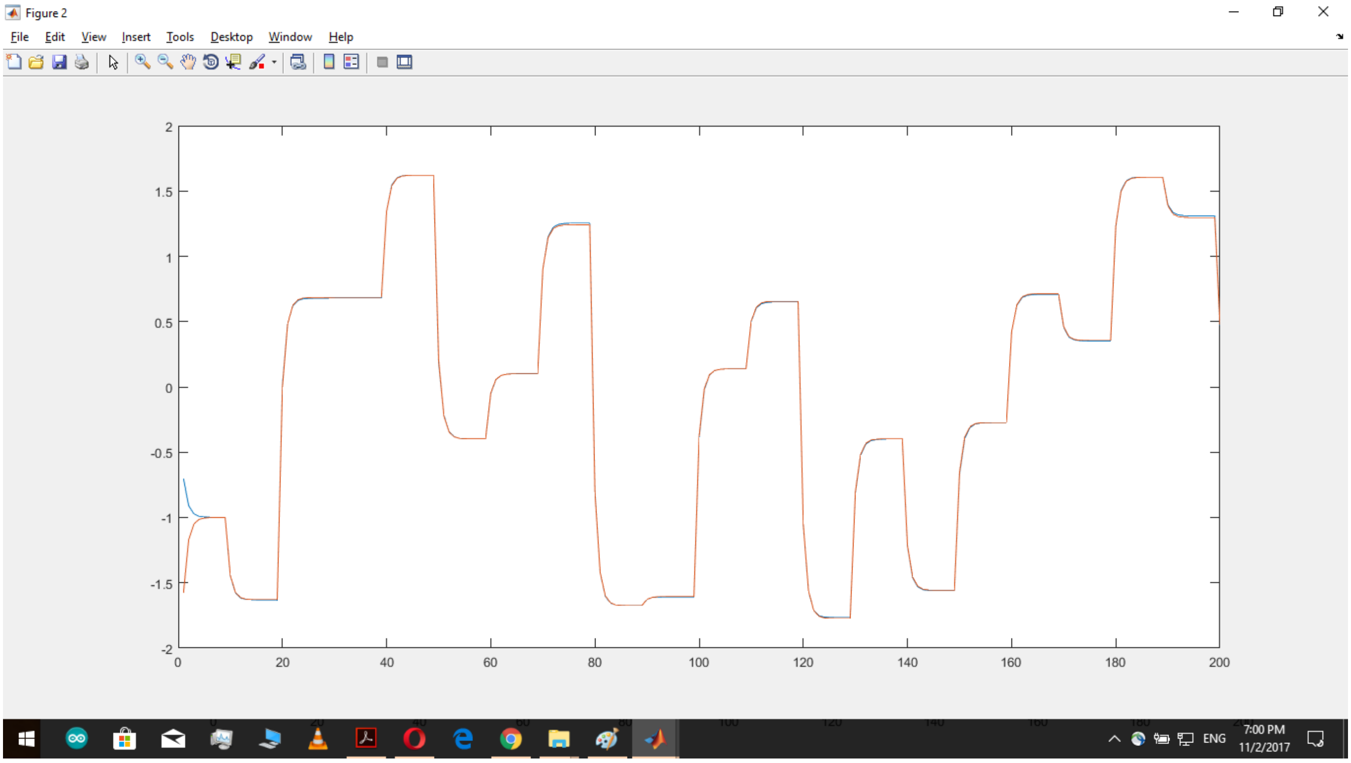


Figure 0.6: This graph shows the output signal of the system (orange) and the emulator (blue)

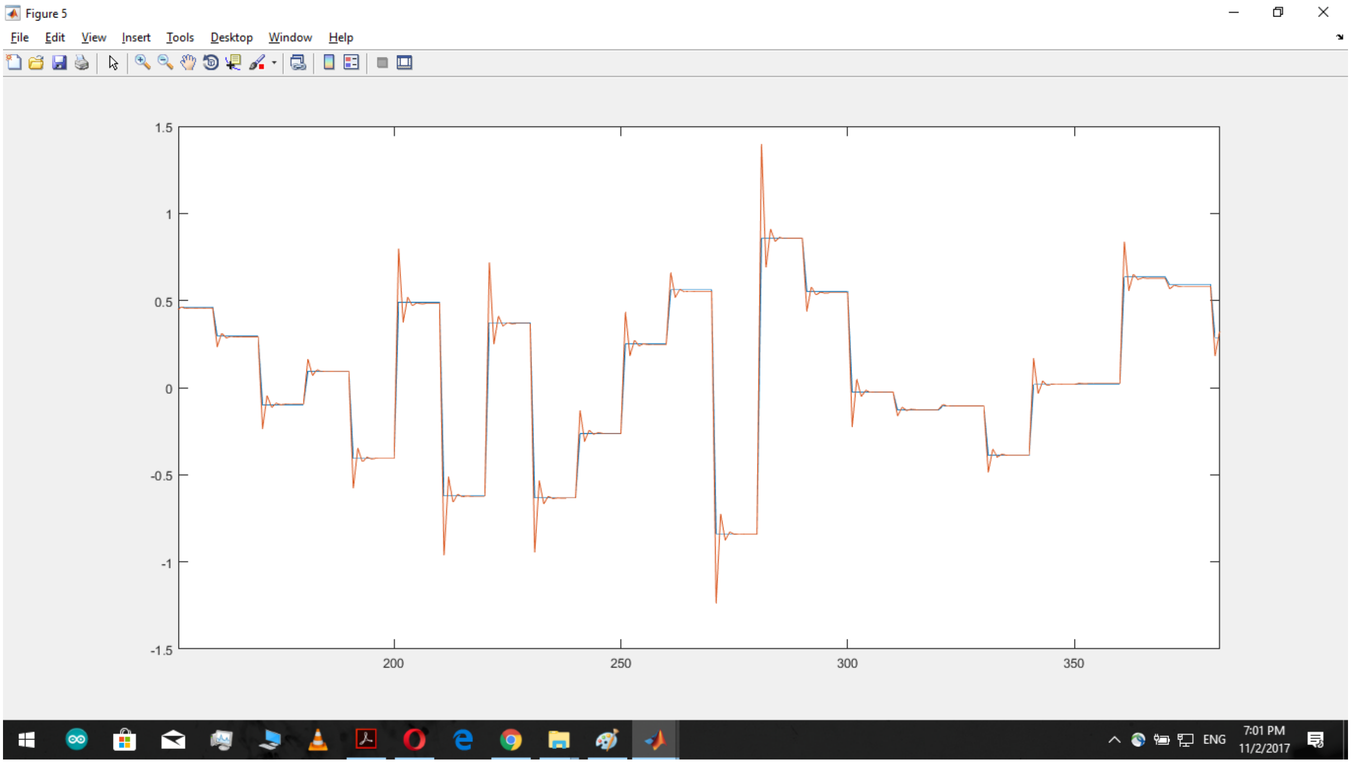


Figure 0.7: In this figure, the blue signal is the input signal given to the controller and the orange signal is the system output