

Todd Holloway  
ScottyNN  
11/8/2004

### Summary

Described in this document is the development of an adaptive controller for tracking the velocity of a vehicle. The controller consists of two components, the first being a time-delay neural network which, given recent velocities of the vehicle and a force (throttle) to be applied, predicts the change in velocity. The second component is a cost minimization function which determines the minimal costing (total force value) sequence of forces to be applied to achieve a desired velocity.

A model of vehicle velocity was implemented in Simulink (part of Matlab) to generate training data for the predictive network. For the purposes of IRRT, we probably will not have a model to obtain training data, but rather record actual vehicle data from performance of the Jeep Wrangler. Also, for simplicity of development only velocity was used, but this controller will need to be extended to include heading.

The controller was implemented in Matlab, and its performance evaluated in Matlab. Matlab's compiler, however, has the ability to generate the controller as a C shared library. The speed of such a shared library has not yet been evaluated.

The speed of the controller depends on optimizations to the cost minimization function. The network only has 19 nodes, and can be simulated in Matlab in <5ms.

### Sources

The two primary references used for this work were:

- (1) M. Lazar, O. Pastravanu, (2002). A neural predictive controller for non-linear systems. *Mathematics and Computers in Simulation*, 60 (Issue 3-5), pp. 315-324
- (2) M. Hagan, H. Demuth, O. De Jesus, "An Introduction to the Use of Neural Networks in Control Systems," *International Journal of Robust and Nonlinear Control*, Vol. 12, No. 11, September, 2002, pp. 959-985.

### Development

There were two primary goals during development, to create a neural network which models the effects of force on vehicle velocity and to create a cost minimization function to select the best inputs to the network to achieve a desired output.

#### (1)Obtain Training Data

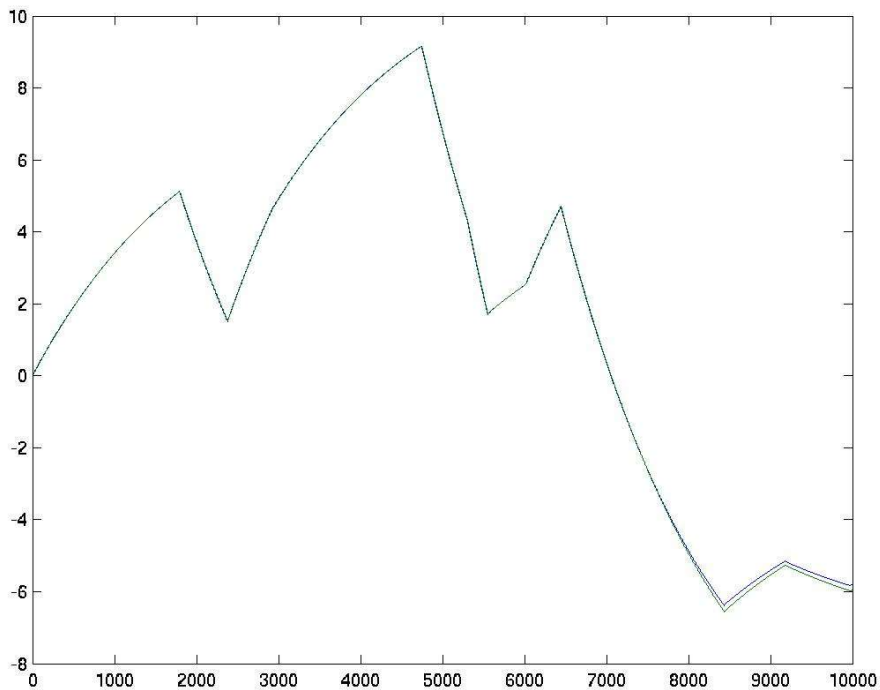
In place of data captured from the real vehicle, I developed a model of vehicle velocity in Simulink and then captured 10000 data samples in one long sequence. The data is stored as u-values (control inputs) and corresponding y-values (output and time t+1). However, the NN we will create takes two inputs, u-values and corresponding y-values at time t. So we build an input matrix as follows:

```
A = Load(...);
U = A.U';
Y = A.Y';
UI = U(2:10001)
YI = Y(1:10000)
ZI = cat(1,UI,YI);
YO = Y(2:10001);
```

## (2) Create a NN model

Because we are dealing with time sequences, I use a neural net with time delays on the inputs to capture relationships over time. I used three delays in order that acceleration/deceleration is learned from the inputs. The following code creates and trains the network.

```
net = newfftd([-1000 2000; -2 22],[0 1 2],[10 1],{'tansig' 'purelin'});
net.trainParam.epochs = 175;
net = train(net,ZI,YO);
```

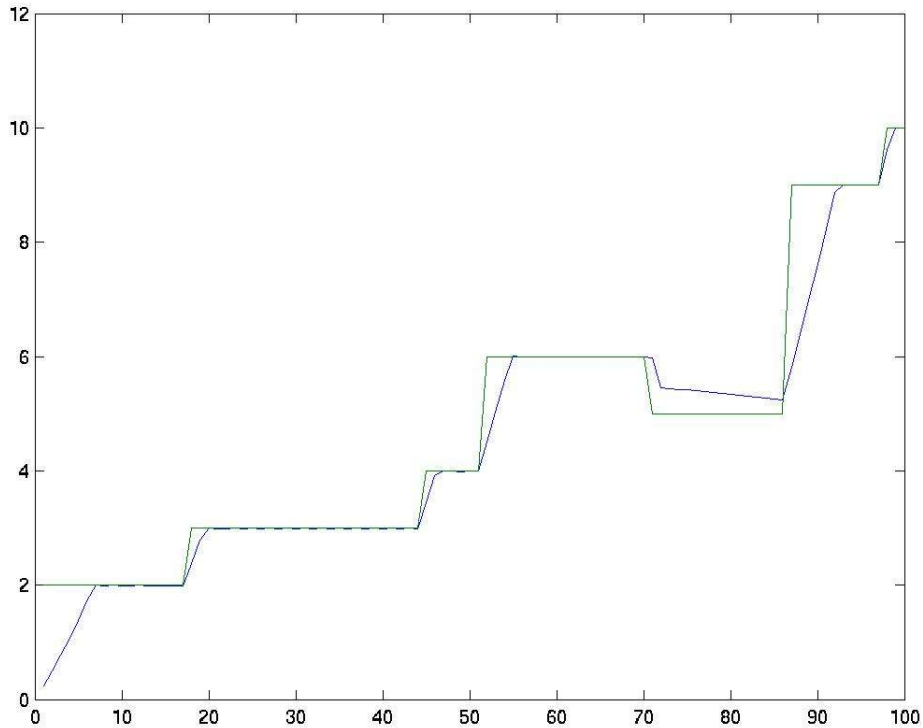


The trained neural network performs almost exactly as the original model of vehicle acceleration.

### (3) Create the Controller

Writing the cost minimization function was the most difficult part of this project, and the part that requires the most further optimization.

The code is included in external files. The primary parameters influencing the performance are the max evaluations (minimization attempts) of the prediction function and the starting values for the minimization function to use. I currently use 0 always for the starting values. This may be a good place for CBR.



As can be seen above the accuracy of the controller was quite high even with only 10000 training examples, which is quite under these circumstances.

Speed was the real issue. The neural net takes about 5ms to perform a transformation running in Matlab (within the GUI). However, right now the controller calls the network as many as 50 times to decide the next action. There are many obvious potential optimizations to speed the controller code up, but I have yet to test them out.