

Self-Tuning Speed Tracking Controller for a Portable Brake and Throttle Robot

David R. Mikesell
Ohio Northern University

Anmol Sidhu
SEA Ltd

ABSTRACT

Vehicle testing often requires accurate speed control, whether maintaining a constant speed or following a dynamic speed profile. A portable brake and throttle robot designed for this task must quickly and automatically adapt its control to each test vehicle's mass and powertrain characteristics in order for the controller to perform well on a wide variety of platforms.

This application presents two major challenges. First, each new vehicle installation requires rapid retuning of the controller. Manual tuning can be very time-consuming. Second, the "plant" is very nonlinear and asymmetric. Different actuators are employed for acceleration (engine) and braking (road loads and friction brakes).

After a broad survey of control strategies and their suitability for this particular application, neurofuzzy techniques were among the most promising. Neurofuzzy networks can not only approximate nonlinear functions accurately, but the fuzzy rule-consequent weights can be readily updated in real time.

A neurofuzzy controller was designed for the automotive speed tracking problem. Experimental testing was conducted and results were presented and analyzed.

CITATION: Mikesell, D. and Sidhu, A., "Self-Tuning Speed Tracking Controller for a Portable Brake and Throttle Robot," *SAE Int. J. Passeng. Cars - Mech. Syst.* 6(1):2013, doi:10.4271/2013-01-1434.

INTRODUCTION

Accurate speed control is an essential element in dynamic vehicle testing. Skilled human test drivers are able to achieve a certain level of accuracy and repeatability, but this level is generally lower than that of a well-tuned automated controller. In addition, the installation and use of an automated controller requires some technical ability, but no particular coordination or physical aptitude, putting quality test results within reach of a wider population.

Testing regimen frequently prescribe a time- or position-based speed profile determined *a priori*, establishing the situation as a tracking control problem. The problem is complicated by an interchangeable plant; unlike single-function controllers, this device must be re-tuned for each new vehicle, often several times in one day. For optimal performance, the portable brake and throttle robot controller designed for this task must quickly adapt to each test vehicle's mass and powertrain characteristics.

The control problem can also be characterized in other ways. In addition to being nonlinear, the plant is asymmetric;

powertrain response (acceleration) is much different than braking response (deceleration). The powertrain response to throttle inputs is roughly first-order. More precisely, assuming a gear-based transmission (not continuously variable), powertrain response follows a series of first-order trajectories which change with each gear shift. See for example the throttle step response in [Figure 1](#), showing a different time constant τ with each gear change. Braking response, by contrast, is closer to a zero-order system, generating fairly constant deceleration for a given pedal position [1].

A preliminary study investigated the suitability of various control techniques for a portable automotive speed control device [2]. Considered strategies included PID variants (auto-tuned, gain-scheduled, etc.), zero-phase error tracking controllers, and intelligent systems such as fuzzy logic and genetic algorithms. Many were ruled out due to the difficulty or limitations of automating their design or tuning process.

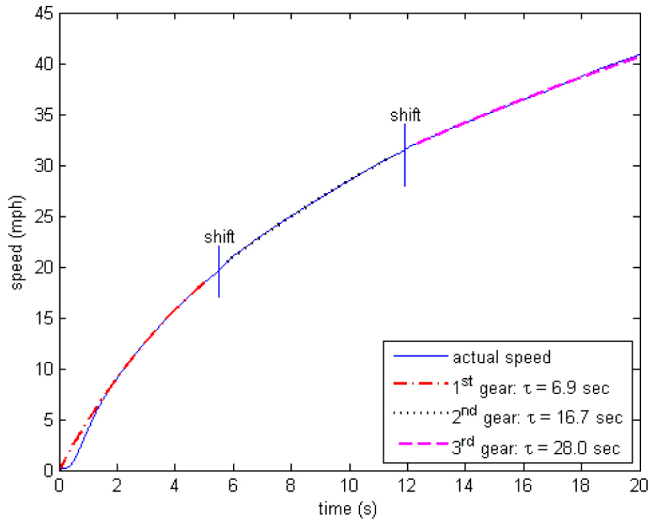


Figure 1. Throttle step response (2000 Camry CE)

Indeed, other control methods have been employed which provide good speed control but require substantial time to tune. Examples include fuzzy logic [2], PID with velocity feedforward [1], and a PID controller with a feedforward mechanism based on the static process characteristic [3]. Results of the latter method are included for comparison.

Ultimately the neurofuzzy system was chosen for development and is the focus of this study. These controllers can benefit from expert design intuition while not relying on manual tuning to achieve good control on nonlinear plants.

The neurofuzzy controller is a multi-layer neural network that is functionally equivalent to a fuzzy inference model [4, 5]. Such systems can “find new fuzzy rules, or change and tune existing ones” based on data provided, all much faster than a genetic method [6].

The following sections will describe the neurofuzzy implementation, hardware employed, experimental results, and conclusions and future work based on the results.

METHOD

Costa Branco and Dente describe the application of neuro-fuzzy techniques to an electro-hydraulic actuator [7], and their work served as a theoretical framework for this paper. First is described the neurofuzzy modeling process for the plant. This includes the learning algorithm, the choice of fuzzy inputs, the acquisition of training and testing data sets, and the computation of the neurofuzzy inverse model of the plant. Second, the extracted neurofuzzy model and its learning capabilities are used to design the controller based on the feedback-error-learning technique.

In essence, the neurofuzzy controller is a fuzzy logic system tuned by a neural network. These two elements (fuzzy system and neural network) are generally integrated in such a way that one cannot draw a sharp dividing line between them, but it is essential to have a basic understanding of fuzzy logic to comprehend the neurofuzzy controller.

Fuzzy Logic Background

A fuzzy logic system consists of three main blocks: fuzzification, an inference mechanism, and defuzzification. Fuzzification is the characterization of a given parameter in linguistic terms such as small, large, fast, or slow. Numerical parameters $x' = (x'_1, x'_2, \dots, x'_m)$ are mapped into fuzzy sets represented by membership functions. An example (of discrete sets) might be that if input $0 \leq x_1 \leq 0.5$ then x_1 is “small,” or if $0 \leq x_1 \leq 1$ then x_1 is “large.” A variety of mathematical membership functions are possible (e.g., triangular), but Gaussian functions, denoted by $\mu_{A_j^i}(x'_j)$, were selected for this study. The Gaussian function is expressed in Equation (1).

$$\mu_{A_j^i}(x'_j) = a_j^i \exp \left[-\frac{1}{2} \left(\frac{x'_j - b_j^i}{c_j^i} \right)^2 \right] \quad (1)$$

In this equation, j is the index for the m input variables; i is the index for the n_j membership functions (each input j can have a different number of functions); a_j^i defines the maximum of each Gaussian function, here normalized to 1.0 for all; b_j^i and c_j^i are the center and width of the Gaussian functions, respectively. Overlapping Gaussian curves imply that a given input will never be a member of only one function (e.g., “large”); rather, for a certain value it might simultaneously be 70% “large” and 10% “small.”

The inference mechanism is the fuzzy logic process matching fuzzified inputs to a particular output. It embodies a set of rules such as, “If the velocity is *slow* and the desired velocity is *fast*, depress the accelerator *a lot*.” Mathematically, this can be expressed as shown in equation (2).

$$\begin{aligned} R^{(l)}: & \text{ IF } (x_1 \text{ is } A_1^{(l)} \text{ and } x_2 \text{ is } A_2^{(l)} \text{ and } \dots x_m \text{ is } A_m^{(l)}) \\ & \text{ THEN } (y \text{ is } \omega^{(l)}) \end{aligned} \quad (2)$$

Here $R^{(l)}$ is the l^{th} rule out of c total rules; y is the output; $A_j^{(l)}$ are the antecedent linguistic terms (or fuzzy sets) in rule l ; and $\omega^{(l)}$ is the rule conclusion, a numerical value.

Each IF-THEN rule defines a relationship between the fuzzy conditions and the rule conclusions. Since the conditions (or memberships) are not precise, a number of rules will be “true,” or “fired,” to varying degrees. The product inference expression in (3) represents the net activation degree of each identified rule by measured condition variables.

$$\prod_{j=1}^m \mu_{A_j^{(l)}}(x'_j) \quad (3)$$

This enables the defuzzification of the output to a single weighted value. All rule contributions $\omega^{(l)}$ are combined using the centroid defuzzification formula expressed in (4). This maps input process states x'_j to the output value resulting from the inference function $Y(x')$.

$$Y(x') = \frac{\sum_{l=1}^c \left(\prod_{j=1}^m \mu_{A_j^{(l)}}(x'_j) \right) \omega^{(l)}}{\sum_{l=1}^c \left(\prod_{j=1}^m \mu_{A_j^{(l)}}(x'_j) \right)} \quad (4)$$

Neurofuzzy Algorithm

The neurofuzzy algorithm employed here, developed by Wang [8], can be summarized by Figure 2. In this example the system has two inputs, x_1 and x_2 , and one output variable y . In the first stage, the input variables are evaluated as to their degree of participation in the various fuzzy membership sets. The second stage calculates each rule $R^{(l)}$ its respective activation degree. The final stage is the inference mechanism which generates a calculated output Y by combining the weighted rule consequents according to equation (4).

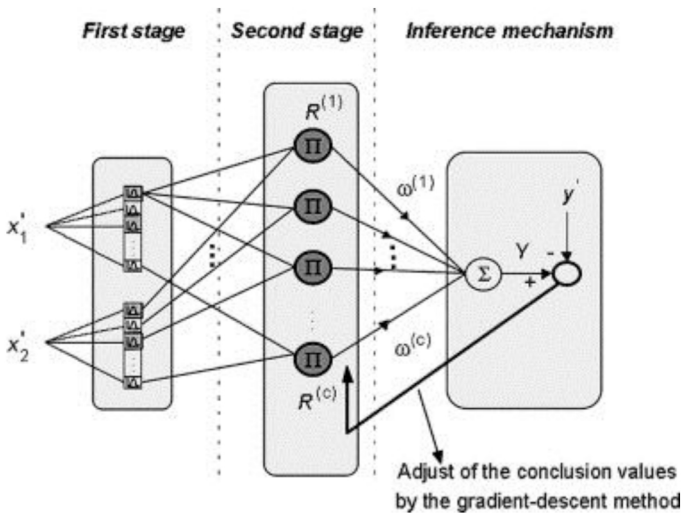


Figure 2. The neurofuzzy scheme [7]

This final stage is also where the neural-based learning takes place. The error signal between the model-inferred output Y and the respective measured value y is used by the gradient descent method to adjust each rule conclusion.

For a set of k training data elements, the mean quadratic error E is computed according to equation (5). The rule conclusions $\omega^{(l)}$ are adjusted based on equation (6), where α is the learning rate and t is the number of learning iterations executed by the algorithm.

$$E = \frac{1}{2} [Y(x'(k)) - y'(k)]^2 \quad (5)$$

$$\omega^{(l)}(t+1) = \omega^{(l)}(t) - \alpha \frac{\partial E}{\partial \omega^{(l)}} \quad (6)$$

The practical neurofuzzy implementation comprises three steps:

- Generate a set of training data from the physical system.
- Use the training data to build and refine the neurofuzzy system model.
- Complete the neurofuzzy control system by inverting the system model, then improving control via the learning mechanism.

The Experimental System

To understand the automotive speed control application of these steps, it is necessary to briefly describe the experimental setup. The system consists of the nested loop shown in Figure 3. The speed controller compares the desired and actual vehicle speeds and generates the appropriate command for brake or accelerator pedal displacement. The inner loop drives the brake and throttle actuator (Figure 4) to the position commanded by the speed controller.

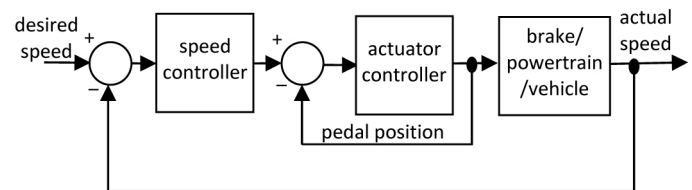


Figure 3. Experimental speed control diagram



Figure 4. SEA brake and throttle actuator prototype [9]

The actuator consists of a single motor driving a ball screw. Attached to the ball nut is a plate which, if driven to