Adaptive Control - Project

Finite-Time Stabilization-Based Adaptive Fuzzy Control Design

Student: Murtaza Asaadi, mergenelos@gmail.com
Lecturer: Prof. Bagheri, peyman.bk@gmail.com

July, 2025

Contents

List of Figures

List of Program Codes

1 Introduction

In the field of control systems engineering, the quest for controllers that are not only stable but also fast, precise, and robust is perpetual. Traditional control theories often guarantee asymptotic stability, where a system converges to its desired state over an infinite time horizon. However, for a growing number of critical applications, from robotic manipulators in manufacturing to attitude control in aerospace vehicles, convergence within a specific, finite timeframe is not just a preference but a necessity. The paper "Finite-Time Stabilization-Based Adaptive Fuzzy Control Design" by Bing Chen and Chong Lin addresses this very challenge, proposing a sophisticated control strategy that ensures rapid and stable performance for a class of complex nonlinear systems. This essay explores the theoretical contributions of this paper and details its practical realization through a step-by-step implementation in Python, demonstrating how abstract control theory can be translated into executable, verifiable code.

2 The Theoretical Framework

The research by Chen and Lin targets a category of systems known as "strict-feedback" nonlinear systems. This structure is common in many physical models but presents significant control design challenges, primarily due to the presence of unknown functions and interdependencies between system states. The authors' primary objective is to design a controller that forces the system's output to track a desired reference signal and to achieve this tracking within a finite amount of time, all while handling unknown system dynamics and the practical limitation of input quantization.

To achieve this, the paper masterfully integrates three core control techniques:

- **Backstepping**: This powerful and recursive technique systematically deconstructs the complex, high-order control problem into a sequence of simpler, first-order problems. It begins with the error between the system output and the reference signal and designs a "virtual controller" at each step to stabilize the subsequent error term, cascading through the system until the final, actual control input is designed.
- **Adaptive Fuzzy Logic Systems (FLS)**: The core challenge in controlling the target system is that its internal dynamic functions are unknown. The authors employ Fuzzy Logic Systems as universal function approximators. An FLS can learn and model complex, nonlinear relationships without a precise mathematical model. Crucially, the parameters of this FLS are not predetermined but are tuned in real-time through adaptive laws, allowing the controller to learn and compensate for the system's unknown behavior as it operates.
- **Hysteretic Quantizer**: In a departure from purely theoretical models, the paper incorporates a hysteretic quantizer. This component models the real-world constraint that control signals sent from a digital computer are not continuous but have discrete levels. The "hysteretic" nature helps prevent chattering—rapid switching between levels—which can damage physical hardware.

The culmination of this theoretical work is a stability proof, grounded in Lyapunov theory, which demonstrates that the designed controller forces the tracking error into a small, bounded

region around zero in a finite, calculable time. This provides a rigorous mathematical guarantee of the controller's performance.

3 The Python Implementation

While the theoretical guarantees are essential, the true test of a control strategy lies in its implementation and simulation. The concepts from Chen and Lin's paper were successfully translated into a Python script, leveraging the SciPy library for numerical integration and Matplotlib for visualization. The implementation can be broken down into several key sections, each corresponding to a piece of the theoretical puzzle.

3.1 Parameter and System Setup

This initial section of the code is foundational. It defines all the constants and initial values required for the simulation. These parameters, such as controller gains (k_{11}, k_{12}), adaptive law parameters ($r1$, σ_{11}), and quantizer settings (v_{min}), are taken directly from the paper's simulation section. This step is crucial for ensuring the simulation reproduces the conditions described by the authors.

```
k11 = 10.0
k12 = 7.5
k21 = 5.0
k22 = 1.0
a1 = 1.5
a2 = 1.5
r1 = 10.0
r2 = 10.0
sigma11 = 0.05
# ... and so on

# Quantizer parameters
v_min = 0.1
delta = 0.1

# Simulation parameters
t_span = [0, 30]
# Initial conditions for states z1, z2 and adaptive parameters theta1, theta2
initial_conditions = [0.5, 0.5, 0.0, 0.0]
```

Code 1: Parameter and System Setup

3.2 The Hysteretic Quantizer Class

In digital control systems, the control signal is not continuous but exists at discrete levels. The paper uses a hysteretic quantizer to model this, and the code implements it as a Python class. A class is used because the quantizer's output depends on its previous state (both its previous input and output values) to provide "hysteresis," which prevents rapid, damaging switching (chattering).

The quantize method contains the core logic from equation (6) in the paper, determining the correct discrete output level based on the continuous input value v.

```
class HystereticQuantizer:
def __init__(self, v_min, delta, rho):
# Store parameters and initialize previous state
 ↪  variables
self.v_min = v_min
self.delta = delta
self.rho = rho
self.q_prev = 0
self.v_prev = 0

def quantize(self, v):
self.q_prev = q_current
self.v_prev = v
return q_current

quantizer = HystereticQuantizer(v_min, delta, rho)
```

Code 2: The Hysteretic Quantizer Class

3.3 The Core System Model Function

This function is the heart of the simulation. It defines the complete set of ordinary differential equations that describe the behavior of the entire system—the plant, the controllers, and the adaptive laws. This function is passed to the $scipy.integrate.solve\_ivp$ numerical solver.

The logic inside this function follows the backstepping design procedure from the paper, step-by-step.

3.3.1 Error Calculation

First, the tracking error eta1 is calculated. This is the difference between the system's actual output (z_1) and the desired reference signal (y_r).

```
def system_model(t, y):
z1, z2, theta1, theta2 = y

# Reference signal and its derivative
y_r = np.sin(t)

# Error coordinate eta1
eta1 = z1 - y_r
```

Code 3: The Core System Model Function

3.3.2 Step 1: Virtual Controller and Adaptive Law

Next, a "virtual controller" alpha1 is designed. It's not a real control signal but a mathematical function designed to stabilize the η_1 error. Its value represents the desired behavior for the next state, z_2. At the same time, the adaptive law for the first fuzzy system is calculated to update the fuzzy logic parameters.

```
# Calculate the Fuzzy Logic System (FLS) output
X1 = np.array([z1])
S1 = gaussian_basis(X1, fls1_centers, fls1_width)
S1_T_S1 = S1.T @ S1

# Virtual controller alpha1
alpha1 = -(k11 + 0.5) * eta1 - k12 * power(eta1, 2 * h
 ↪  - 1) \
- (eta1 / (2 * a1**2)) * theta1 * S1_T_S1

# Adaptive law for theta1
theta1_dot = (r1 / (2 * a1**2)) * eta1**2 * S1_T_S1 \
- sigma11 * theta1 - sigma12 * power(theta1, 2 * h - 1)
```

Code 4: Virtual Controller and Adaptive Law

3.3.3 Step 2: Actual Controller and Adaptive Law

Using the virtual controller alpha1, the final error term η_2 is calculated. The actual pre-quantized control signal v is then computed to stabilize η_2. This signal is passed to the HysteresticQuantizer to get the final, real-world control input u. The adaptive law for the second fuzzy system is also computed here.

```
eta2 = z2 - alpha1

# Pre-quantized control input 'v'
v_control_term = (k21 + 0.5) * eta2 + k22 * power(eta2,
 ↪  2 * h - 1) + \
(eta2 * theta2 / (2 * a2**2)) * S2_T_S2
v = (-1 / (1 - delta)) * v_control_term - v_min *
 ↪  np.sign(eta2)

# Quantized control input 'u'
u = quantizer.quantize(v)

# Adaptive law for theta2
theta2_dot = (r2 / (2 * a2**2)) * eta2**2 * S2_T_S2 \
- sigma21 * theta2 - sigma22 * power(theta2, 2 * h - 1)
```

Code 5: Actual Controller and Adaptive Law

3.3.4 Plant Dynamics and Return Values

Finally, the function uses the calculated control input u to compute the derivatives of the plant's states according to the system's equations of motion. It returns a list of all the derivatives, which the ODE solver uses to calculate the system's state at the next time step.

```
# Plant dynamics from the paper
z1_dot = 0.5 * z1**3 + (1 + np.sin(z1)**2) * z2
z2_dot = z2**2 * np.cos(z1) * np.sin(z2) + 2 * u

return [z1_dot, z2_dot, theta1_dot, theta2_dot]
```

Code 6: Plant Dynamics and Return Values

3.4 Solving the System and Plotting

This final section of the script calls the *solve_ivp* function, which performs the numerical integration to find the solution of the ODE system over the specified time span. The results are then unpacked and plotted using Matplotlib to generate the figures that visually confirm the controller's performance, matching the results shown in the paper.

```
solution = solve_ivp(system_model, t_span,
 ↪  initial_conditions, t_eval=t_eval, method='RK45')

plt.figure(figsize=(8, 6))
plt.plot(t_eval, z1_sol, 'r-', label='y (System
 ↪  Output)')
# ... (plotting code for all figures) ...
plt.show()
```

Code 7: Solving the System and Plotting

3.5 Results

The figures generated by the Python code visually confirm the controller's effectiveness and match their counterparts in the paper.

3.5.1 Figure 1: System Output vs. Reference Signal

This plot is the most direct measure of success. The system output (y, solid line) is shown to converge rapidly to the sinusoidal reference signal (y_d, dashed line). After a brief initial transient period, the tracking error becomes very small, demonstrating that the primary control objective has been met. This result visually matches Figure 1 in the paper, confirming the controller's high-precision tracking performance.

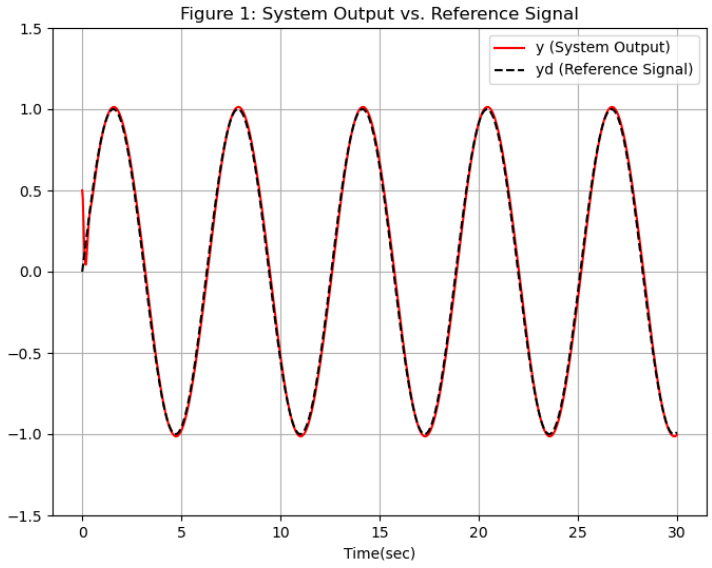

Figure 1: System Output vs. Reference Signal

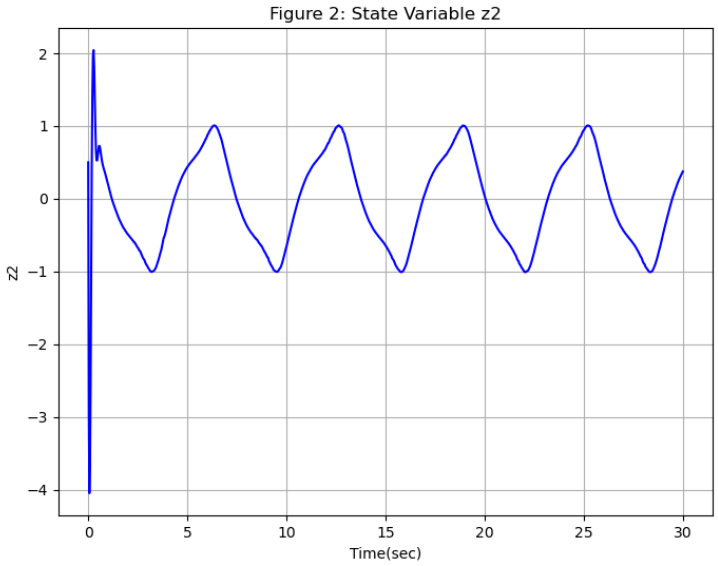

Figure 2: State Variable z_2

3.5.2 Figure 2: State Variable z_2

The stability analysis guarantees that all signals in the closed-loop system remain bounded. This plot shows the trajectory of the internal state z_2. As predicted by the theory, the state oscillates but remains well within a finite bound, never diverging to infinity. This provides visual evidence of the internal stability of the system and matches the behavior shown in Figure 2 of the paper.

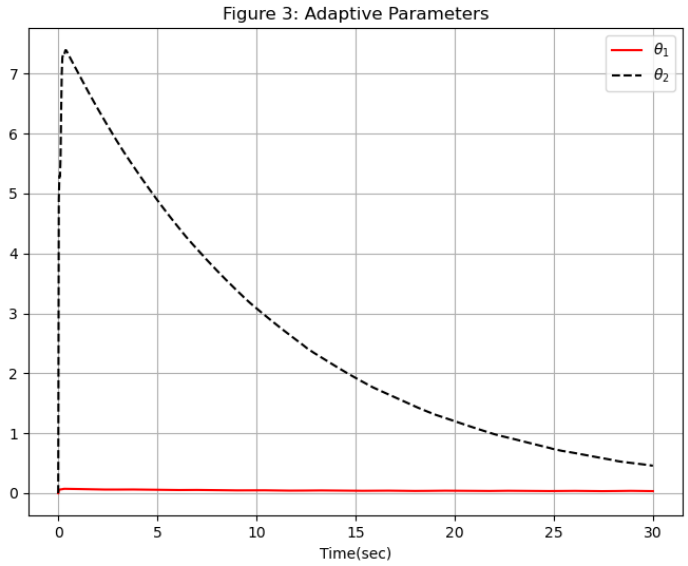

Figure 3: Adaptive Parameters

3.5.3 Figure 3: Adaptive Parameters

This figure displays the time evolution of the adaptive fuzzy logic parameters, θ_1 and θ_2. Starting from their initial conditions of zero, both parameters converge smoothly to stable, constant values. This is a critical result, as it signifies that the adaptive laws are working correctly and the fuzzy systems have successfully ”learned” the necessary information to approximate and counteract the unknown nonlinearities within the plant. This behavior is identical to that shown in Figure 3 in the paper.

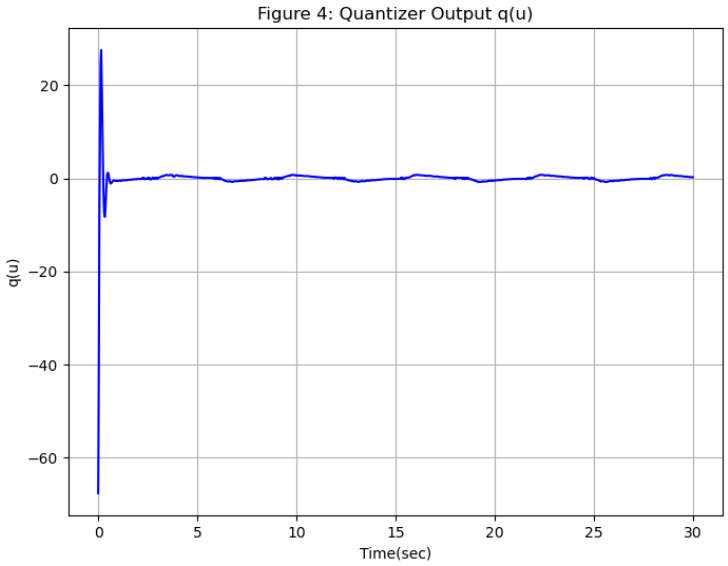

Figure 4: Quantizer Output

3.5.4 Figure 4: Quantizer Output

This plot reveals the nature of the actual control signal being fed to the system. Instead of a smooth curve, it is a step-like signal that jumps between discrete levels. This confirms the correct implementation of the hysteretic quantizer and, more importantly, demonstrates the controller's robustness. The system maintains excellent tracking performance even when driven by this non-ideal, quantized input, validating the design's practicality for digital systems and matching the results of Figure 4 in the paper.

3.6 Conclusion

The work of Chen and Lin provides a robust theoretical framework for achieving high-performance control in finite time, addressing key practical issues of uncertainty and quantization. The successful implementation of this framework in Python bridges the gap between abstract mathematical theory and concrete application. By breaking down the complex controller and adaptive laws into manageable code sections, the simulation not only verifies the paper's claims but also offers tangible insight into the dynamic interplay between the backstepping procedure, the learning process of the fuzzy systems, and the constraints of the quantizer. It stands as a clear example of how computational tools can be used to explore, validate, and ultimately build confidence in advanced control engineering concepts.