JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering

Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574

A Comparative Study of Traditional PID Tuning Techniques and AI-Based Algorithmic Approaches Utilizing the Python Control Library

Purwadi Joko Widodo[1], Heru Sukanto[2], Budi Santoso[3], Fitrian Imaduddin[4], Lullus Lambang Govinda Hidayat[5], Joko Triyono[6] , Iwan Instanto[7], Rahman Wijaya[8]

[1,2,3,4,5,6,7,8] Sebelas Maret University, Jl. Ir. Sutami Kentingan Surakarta, Indonesia

ARTICLE INFO

Article historys:

Received : 13/08/2025
Revised : 23/09/2025
Accepted : 30/10/2025

Keywords:

Artificial Intelligence; Genetic Algorithm; Particle Swarm Optimization; PID Parameter; Ziegler-Nichols; Python Control Library; Python Simulation

ABSTRACT

This study aims to compare PID parameter settings with conventional tuning methods and tune methods using AI (artificial intelligence) algorithms. This study was conducted by means of simulation using a computer program created in Python and utilizing AI libraries to solve the problem of determining PID (proportional-integral-derivative) parameters. Two AI algorithms used in this study, namely the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) methods, were compared with the conventional Ziegler-Nichols method. The study was conducted by applying the PID parameters obtained to a certain transfer function and then comparing them on several related aspects. The results of the study showed that the solution obtained using the AI method requires a longer execution time, more than 2 seconds for PSO and more than 3 seconds for GA, while ZN requires less than 1 second. However, the AI method can provide better solutions, as can be seen from the magnitude of the ITAE that occurs, where GA and PSO provide ITAE less than 1 while ZN is more than 22.

This work is licensed under a Creative Commons Attribution 4.0 International License

Corresponding Author:

Purwadi Joko Widodo
Sebelas Maret University, Jl. Ir. Sutami Surakarta, Jawa Tengah, Indonesia
Email: purwadijokow@staff.uns.ac.id.

1. INTRODUCTION

PID is one of the most widely applied methods in automation and control systems. This method can be applied to control parameters in a system, such as temperature, speed, position, pressure, and others. The PID controller minimizes the error between the setpoint value (the desired value) and the output value produced by the system. In PID, three actions are associated with the controller: proportional, integral, and derivative.

Proportional Component (P): This component is used to reduce errors by responding proportionally to the magnitude of the detected error. The larger the error, the greater the proportional response and reaction. Integral Component (I): This component is used to address errors that accumulate over time. This component is important for systems that experience small, continuous errors that cannot be addressed with the proportional component alone. Derivative Component (D): This component predicts future error trends based on the rate of error change. This component provides control to dampen rapid changes and avoid overshoot.

Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574

This method is quite simple, but determining the PID parameters, namely Kp, Ki, and Kd, which are specific to the conditions of the system in question, is quite a challenge. Mistakes in determining the Kp, Ki, and Kd parameters in a system may cause the system to become unstable or respond less than expected. For example, excessive overshoot or unacceptable stabilization time may occur[1]. Generally, PID parameter adjustment can be done through trial and error, which is, of course, time-consuming and results in less than optimal settings. Therefore, various more systematic methods such as grid search, genetic algorithms, or model-based optimization are used to help determine more precise parameters[2].

Classic methods such as Ziegler–Nichols (ZN) have long been used for PID tuning, but often result in high overshoot, long settling times, and steady-state errors that do not meet the needs of modern systems. To overcome these limitations, various artificial intelligence-based optimization methods, such as Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) have begun to be used because they are able to produce PID parameters that are closer to design specifications, namely fast, stable, and precise responses.

The existence of computers and software and their development today are very important in this regard. Python, with its advantages, is also a choice in this regard. Python is a programming language that, in the last decade, has become a very popular computer programming language among engineers and researchers because of its ease of use and the variety of libraries available for numerical data processing, control analysis, and simulation. The main advantage of using Python in computer programming is its ability to integrate various libraries to handle complex technical aspects in a simple and intuitive way [3].

Python libraries were used in PID control implementation, including NumPy, SciPy, and Control. Numpy and scipy provide tools for performing fast numerical calculations, such as matrix operations, differentiation, and integration, which are crucial in managing dynamic systems [4]. The use of the matplotlib and plotly libraries enables the graphical visualization of simulation and experimental results, providing better insight into assessing system performance. The ability to visualize the system's response to changes in PID parameters or to external disturbances helps understand the behavior of more complex systems [5]. Python with the python-control library, as open-source software, offers a more affordable and flexible solution for teaching and research in the field of engineering control, covering the implementation of PID control in Python, including its use for simulation and analysis of control systems[6 - 8].

Implementation of PID control requires careful testing and parameter tuning, often performed directly on a physical system. However, simulation using computers and software allows users to achieve time and cost savings before physically implementing the control. Python allows users to do this. The use of Python allows faster experimentation with various parameter combinations to find optimal settings for P, I, and D parameters, which is not easily achieved with conventional methods [6,9].

Various advantages offered by Python are making more and more engineers, researchers, and developers turn to Python as a primary tool in managing PID control. Many articles have been written on the use of Python in the field of control, especially PID [10-11]. This shows Python's reliable performance in control system analysis.

This article uses Python to simulate and analyze PID control parameter settings using conventional methods and advanced methods utilizing AI algorithms. Using an AI algorithm to determine PID controller parameters, there are several popular and effective algorithms. Each algorithm has its advantages and disadvantages, and the selection of the best algorithm depends on the specific application context, such as system complexity, computational time, and required accuracy.

Several AI algorithms have been used for PID parameter optimization, including:

1. Genetic Algorithm (GA). This algorithm has advantages, suitable for optimization problems with many parameters and without explicit functions, can work well on large and non-linear parameter spaces, and does not require gradients or deep function information. The disadvantage is that this algorithm requires more iterations and a longer time for convergence compared to other algorithms, as a result, it can be expensive in computation if the population size or number of generations is too large[12].

Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574

2. Particle Swarm Optimization (PSO). The PSO algorithm has several advantages, including being faster than GA for many optimization problems, being able to avoid overfitting well in large parameter spaces, and achieving faster convergence compared to GA. However, its drawback is that it is not as effective as GA for optimization with many local minimum points [13].
3. Simulated Annealing (SA). This algorithm has the advantage of being easy to implement and does not require a lot of computation. It can avoid getting stuck in local solutions and potentially find a global solution. However, its drawback is that the search process can be very slow, especially in large search spaces, and it is not always as effective as GA or PSO in terms of speed and accuracy on some problems [12].
4. Reinforcement Learning (RL) has the advantage of being able to continuously adjust based on environmental feedback, learning in real time, and adapting to changes in the system. One of the disadvantage is that this algorithm requires a lot of data for training and iteration.

GA and PSO are often used in PID parameter optimization because of their ability to explore a wide parameter space without requiring special assumptions about the form of the cost function[12].

2. RESEARCH METHOD

2.1. Research Approach

This research uses a quantitative, experimental approach based on computer simulation. The objective is to assess the performance of a PID control system with parameters determined by an artificial intelligence (AI) algorithm, compared to conventionally determined PID parameters.

2.2. Research Design

The research design is carried out in several stages, as follows:

1. System Modeling. At this stage, a mathematical model of the controlled system is developed, such as a first- or second-order linear system (e.g., a heating system, a motor positioning system, etc.). The system model is then simulated using the Python programming language.
2. PID Control Design. The next stage is implementing PID control with initial parameters (the Ziegler-Nichols method can be used as a baseline), followed by developing the PID control function to be used in the simulation.
3. AI Development and Integration. The next stage is developing an AI algorithm to optimize the PID parameters (Kp, Ki, Kd). The algorithms used in this research are the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). The objective function in the optimization process is to minimize system errors, such as:
   a. Integral of Time-weighted Absolute Error (ITAE)
   b. Overshoot, settling time, rise time
4. Simulation and Data Collection: This stage runs simulations on two scenarios: a conventional PID with Ziegler-Nichols theory (baseline) and a PID with AI-optimized parameters, and then data is recorded on the system's response to disturbances or setpoint changes.
5. System Performance Evaluation: The next stage is to analyze system performance based on response graphs and performance metrics, namely rise time, settling time, overshoot, and steady-state error. The error value in this study uses the ITAE criterion.

The flowchart of the research is illustrated in Figure 1 and Table 1 shows parameters used in simulations Figure 1 shows this research begins with system modeling in the form of a transfer function. Next, a PID control design is carried out using the conventional Ziegler–Nichols method and its performance is evaluated through simulation. To improve system performance, PID parameters are then optimized with AI-based algorithms, Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). The optimization results are simulated again to obtain system performance data which are then compared with the results of conventional methods. Based on the results of the comparative evaluation, an analysis is carried out and conclusions are drawn regarding the effectiveness of the optimization method compared to the classical approach.

2.3. Tools and Equipment

1. Programming Language: Python

JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574

2. The Python libraries used in this research are numpy, scipy, and matplotlib for numerical simulation and visualization, the control library for dynamic system models, and the Python library for AI.

2.4. Data Collection Techniques

Data is obtained from simulation results in the form of system response graphs and performance metric values for each scenario. The simulation is run several times to validate the results.

2.5. Data Analysis Techniques

Analysis is conducted quantitatively by comparing simulation results between conventional PID and PID-AI, presented in the form of comparison tables and system performance graphs.

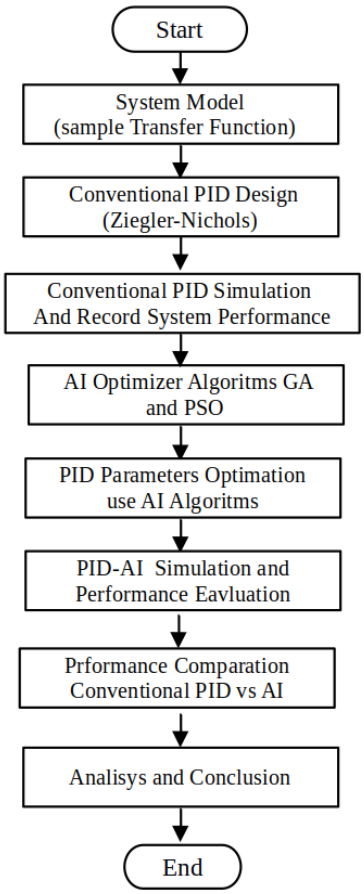

Figure 1. Flowchart of Research Method

Pseudocode used for PID + AI Simulation,

```
================================================================================================
# Step 1: System Model
def system_model(K, T, time):
    # return response of a transfer function (e.g., first order system)
    return control.TransferFunction([K], [T, 1])

# Step 2: PID Evaluation
def pid_simulation(Kp, Ki, Kd, system, t):
    # calculate system respons use PID control
    # use control.feedback and control.forced_response
    return output, error_metrics

# Step 3: Objectif Function for AI
def objective_function(params):
    Kp, Ki, Kd = params
    output, error = pid_simulation(Kp, Ki, Kd, system_model(...), t)
    # return ITAE
    return compute_ITAE(error, t)

# Step 4: Optimatiton with AI (PSO or GA)
best_params = run_ai_optimizer(objective_function)

# Step 5: Repeat Simulation with best Parameter
output_ai, error_ai = pid_simulation(*best_params, system_model(...), t)

# Step 6: Compare with Conventional PID method
output_std, error_std = pid_simulation(Kp_manual, Ki_manual, Kd_manual, system_model(...), t)
```

JURNAL ECTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574

```
# Step 7: Visualisation and Analisys
plot_comparison(output_std, output_ai)
report_performance_metrics(error_std, error_ai)
================================================================================================
```

Table 1. Parameters Simulations

Algorithm	Parameters	Values (Default in code)
GA (Genetic Algorithm)	Population Size (pop_size)	20
	Maximum Generations (gen_max)	30
	Mutation Rate (mutation_rate)	0.1
	PID Parameter Bounds (bounds)	Kp ∈ [0.5], Ki ∈ [0.5], Kd ∈ [0.2]
PSO (Particle Swarm Optimization)	Number of Particles (num_particles)	15
	Maximum Iterations (max_iter)	30
	Inertia Weight (w)	0.5
	Learning Coefficient (c1, c2)	1.5, 1.5
	PID Parameter Bounds (bounds)	Kp∈[0.5],Ki∈[0.5],Kd∈[0.2]
ZN (Ziegler–Nichols)	Calculation based on gain margin (Ku) and ultimate period (Pu)	Calculation base on formula
Aspect	Simulation Time (T_FINAL)	20 s
	Number of Time Points (T_STEPS)	500
	Time Range (T)	0 – 20 s
	Plant	$G(s)=1/(s^2+6s+5)$
	Input	Step input = 1
	Evaluation Criteria	ITAE
	Performance Analysis	Overshoot, Rise Time, Settling Time, Steady-State Error

3. RESULTS AND DISCUSSION

For observation and comparison in this study, a Python program was built using three selected methods, namely the Ziegler-Nichols method, representing the conventional method, and the GA and PSO methods, representing the AI method. The three methods produce a graph and several important data points related to the methods above. The developed Python code allows users to change the system's transfer function value, and generate PID parameter values, namely overshot, ITAE error, rise time, settling time, and the required execution time. Below is Python code for the simulation.

```
================================================================================================
# compare_pid_methods.py
import numpy as np
import matplotlib.pyplot as plt
from control import tf, feedback, step_response, margin
import random, csv, time

T_FINAL = 20  # simulation duration for step response
T_STEPS = 500  # number of time
T = np.linspace(0, T_FINAL, T_STEPS)

# === Plant ===
def get_plant():
    num = [1]
    den = [1, 6, 5]
    return tf(num, den)

# === Objecctive Function: ITAE ===
def pid_itae(params, plant):
    Kp, Ki, Kd = params
    if Kp < 0 or Ki < 0 or Kd < 0:
        print(f"Invalid PID: {params}")
        return 1e6
    try:
        C = tf([Kd, Kp, Ki], [1, 0])
        closed_loop = feedback(C * plant, 1)
        t, y = step_response(closed_loop, T)
        if np.any(np.isnan(y)) or np.any(np.isinf(y)):
            print(f"Invalid response: {params}")
            return 1e6
        e = 1 - y
     #   itae = simpson(t * np.abs(e), t)
     #   itae = simps(t * np.abs(e), t)
        itae = np.trapz(t * np.abs(e), t)
        return itae
    except Exception as e:
        print(f"Error for PID {params}: {e}")
        return 1e6

# === Syatem Respon Analisys ===
def analyze_response(pid, plant):
```

Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574


```
    Kp, Ki, Kd = pid
    C = tf([Kd, Kp, Ki], [1, 0])
    TF = feedback(C * plant, 1)
    t, y = step_response(TF, T)
    overshoot = (np.max(y) - 1.0) * 100
    ess = np.abs(1.0 - y[-1])
    rise_time = t[np.where(y >= 0.9)[0][0]] if any(y >= 0.9) else None
    settling_idx = np.where(np.abs(y - 1.0) > 0.05)[0]
    settling_time = t[settling_idx[-1]] if len(settling_idx) > 0 else t[-1]
    return overshoot, ess, rise_time, settling_time

# === GA ===
def ga_pid(plant, pop_size=20, gen_max=30, mutation_rate=0.1, bounds=((0, 5), (0, 5), (0, 2))):
    def random_individual():
        return [random.uniform(*bounds[i]) for i in range(3)]

    def crossover(p1, p2):
        alpha = random.random()
        return [(1 - alpha) * p1[i] + alpha * p2[i] for i in range(3)]

    def mutate(ind):
        i = random.randint(0, 2)
        ind[i] += random.uniform(-1, 1)
        ind[i] = max(bounds[i][0], min(bounds[i][1], ind[i]))
        return ind

    population = [random_individual() for _ in range(pop_size)]
    for _ in range(gen_max):
        scores = [pid_itae(ind, plant) for ind in population]
        ranked = sorted(zip(scores, population), key=lambda x: x[0])
        population = [x[1] for x in ranked[:pop_size//2]]

        children = []
        while len(children) < pop_size - len(population):
            p1, p2 = random.sample(population, 2)
            child = crossover(p1, p2)
            if random.random() < mutation_rate:
                child = mutate(child)
            children.append(child)

        population += children
    return ranked[0][1]

# === PSO ===
def pso_pid(plant, num_particles=15, max_iter=30, bounds=((0, 5), (0, 5), (0, 2))):
    w = 0.5
    c1 = 1.5
    c2 = 1.5
    dim = 3
    particles = np.random.uniform([b[0] for b in bounds], [b[1] for b in bounds], (num_particles, dim))
    velocities = np.zeros_like(particles)
    personal_best = particles.copy()
    personal_best_scores = np.array([pid_itae(p, plant) for p in particles])
    global_best = personal_best[np.argmin(personal_best_scores)]

    for _ in range(max_iter):
        for i in range(num_particles):
            r1, r2 = np.random.rand(dim), np.random.rand(dim)
            velocities[i] = (
                w * velocities[i]
                + c1 * r1 * (personal_best[i] - particles[i])
                + c2 * r2 * (global_best - particles[i])
            )
            particles[i] += velocities[i]
            particles[i] = np.clip(particles[i], [b[0] for b in bounds], [b[1] for b in bounds])

            score = pid_itae(particles[i], plant)
            if score < personal_best_scores[i]:
                personal_best[i] = particles[i].copy()
                personal_best_scores[i] = score

        global_best = personal_best[np.argmin(personal_best_scores)]

    return global_best.tolist()

# === Ziegler-Nichols Tuning ===
def zn_pid(plant):
    K = 1.0
    C = tf([K], [1])
    loop = C * plant
    gm, pm, wg, wp = margin(loop)
    if gm == float("inf") or wp == 0 or np.isnan(gm) or np.isnan(wp):
        print("ZN: margin gagal dihitung, fallback ke PID default")
        return [1.0, 1.0, 0.0]  # fallback

    Ku = gm
    Pu = 2 * np.pi / wp
    Kp = 0.6 * Ku
    Ki = 1.2 * Ku / Pu
    Kd = 0.075 * Ku * Pu
    return [Kp, Ki, Kd]

# === Plot Comparation ===
def plot_comparison(pids, labels, plant):
    plt.figure()
    for pid, label in zip(pids, labels):
```

JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574


```
            Kp, Ki, Kd = pid
            C = tf([Kd, Kp, Ki], [1, 0])
            TF = feedback(C * plant, 1)
            t, y = step_response(TF, T)
            plt.plot(t, y, label=label)
        plt.title("Step Response Comparison")
        plt.xlabel("Time (s)")
        plt.ylabel("Output")
        plt.grid(True)
        plt.legend()
        plt.show()

    # === Save Result as CSV ===
    def save_results_to_csv(filename, methods, pids, times, plant):
        with open(filename, mode='w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(["Method", "Kp", "Ki", "Kd", "ITAE", "Overshoot", "Steady State Error", "Rise Time", \
                             "Settling Time", "Execution Time (s)"])
            for method, pid, elapsed in zip(methods, pids, times):
                itae = pid_itae(pid, plant)
                os, ess, tr, ts = analyze_response(pid, plant)
                writer.writerow([method] + pid + [itae, os, ess, tr, ts, elapsed])

    # === Main ===
    if __name__ == "__main__":
        plant = get_plant()

        t0 = time.time()
        pid_zn = zn_pid(plant)
        t_zn = time.time() - t0

        t0 = time.time()
        pid_ga = ga_pid(plant)
        t_ga = time.time() - t0

        t0 = time.time()
        pid_pso = pso_pid(plant)
        t_pso = time.time() - t0

        methods = ["Ziegler-Nichols", "Genetic Algorithm", "Particle Swarm"]
        pids = [pid_zn, pid_ga, pid_pso]
        times = [t_zn, t_ga, t_pso]

        print("\nZiegler-Nichols PID:", pid_zn, f"(Waktu: {t_zn:.2f}s)")
        print("GA PID:", pid_ga, f"(Waktu: {t_ga:.2f}s)")
        print("PSO PID:", pid_pso, f"(Waktu: {t_pso:.2f}s)")

        for method, pid, elapsed in zip(methods, pids, times):
            os, ess, tr, ts = analyze_response(pid, plant)
            print(f"{method}: Overshoot = {os:.2f}%, ess = {ess:.4f}, Rise Time = {tr:.2f}s, Settling Time = {ts:.2f}s, \
                     Time = {elapsed:.2f}s")

        plot_comparison(pids, methods, plant)
        save_results_to_csv("pid_comparison_results.csv", methods, pids, times, plant)
        print("\nResult was saved at 'pid_comparison_results.csv'")

        ================================================================================================
```

By taking a simple transfer function as shown in Figure 2 and the system transfer function as shown in equation (1), a simulation was then carried out with the Python Program Code that was created and the results were compared. A simple example is used in the simulation to simplify the simulation and comparisons performed.

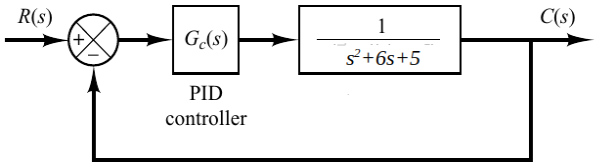

Figure 2. System block diagram used in simulations

$$FA = \frac{1}{s^2+6s+5} \tag{1}$$

Figures 3 and 4 show the simulation results of the three methods being compared to observe the system's response to the step function test signal. Figure 3 is the first simulation, and Figure 4 is the second simulation to verify the result of the first simulation.

JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574

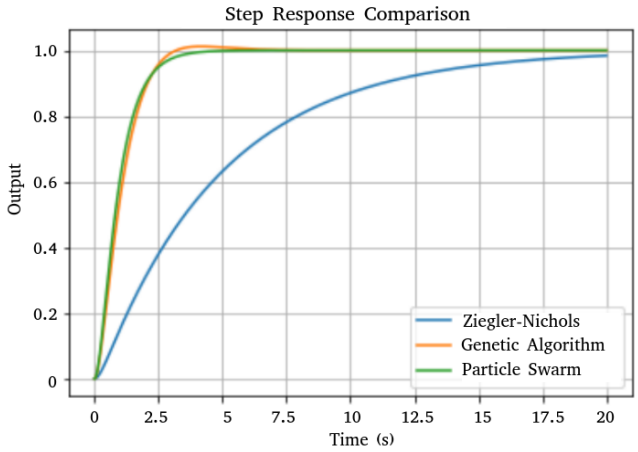

Figure 3. Simulation result 1

```
Ziegler-Nichols PID: [1.0, 1.0, 0.0] (Execution Time: 0.00s)
GA PID: [4.024979381891309, 4.887865592973621, 0.017178639320462367] (Execution Time: 3.97s)
PSO PID: [5.0, 5.0, 0.0] (Execution Time : 2.99s)
Ziegler-Nichols: Overshoot = -1.61%, ess = 0.0161, Rise Time = 11.26s, Settling Time = 14.55s, Time = 0.00s
Genetic Algorithm: Overshoot = 1.18%, ess = 0.0000, Rise Time = 2.08s, Settling Time = 2.40s, Time = 3.97s
Particle Swarm: Overshoot = -0.00%, ess = 0.0000, Rise Time = 2.04s, Settling Time = 2.48s, Time = 2.99s
```

Table 2. Numerical datas of PID parameters generated from the 1st simulation

Method	Kp	Ki	Kd	ITAE	Overshoot	Steady State Error	Rise Time	Settling Time	Execution Time (s)
Ziegler-Nichols	1.00000	1.00000	0.00000	22.08894	-1.60876	0.01609	11.26253	14.54910	0.00151
Genetic Algorithm	4.02498	4.88787	0.01718	0.92264	1.18237	0.00000	2.08417	2.40481	3.96686
Particle Swarm	5.00000	5.00000	0.00000	0.79987	0.00000	0.00000	2.04409	2.48497	2.99478

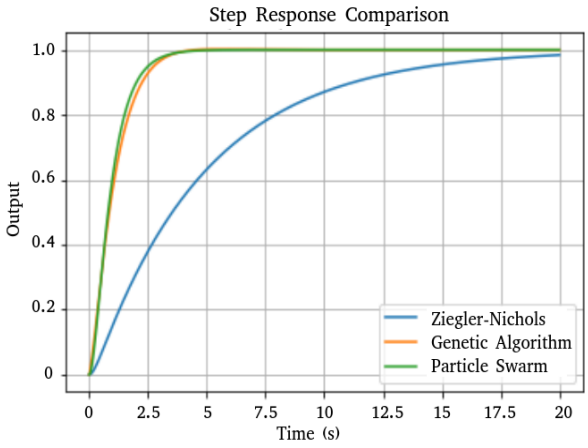

Figure 4. Simulation Result 2

```
Ziegler-Nichols PID: [1.0, 1.0, 0.0] (Execution Time: 0.00s)
GA PID: [4.587986891341883, 4.718672144798576, 0.33918189648143965] (Execution Time: 9.29s)
PSO PID: [5.0, 5.0, 0.0] (Execution Time: 6.80s)
Ziegler-Nichols: Overshoot = -1.61%, ess = 0.0161, Rise Time = 11.26s, Settling Time = 14.55s, Time = 0.00s
Genetic Algorithm: Overshoot = 0.20%, ess = 0.0000, Rise Time = 2.28s, Settling Time = 2.73s, Time = 9.29s
Particle Swarm: Overshoot = -0.00%, ess = 0.0000, Rise Time = 2.04s, Settling Time = 2.48s, Time = 6.80s
```

Table 3. Numerical datas of PID parameters generated from the 2nd simulation

Method	Kp	Ki	Kd	ITAE	Overshoot	Steady State Error	Rise Time	Settling Time	Execution Time (s)
Ziegler-Nichols	1.00000	1.00000	0.00000	22.08894	-1.60876	0.01609	11.26253	14.54910	0.00311
Genetic Algorithm	4.58799	4.71867	0.33918	0.94581	0.20224	0.00000	2.28457	2.72545	9.29095
Particle Swarm	5.00000	5.00000	0.00000	0.79987	0.00000	0.00000	2.04409	2.48497	6.80492

Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574

The evaluation was conducted by analyzing system performance against step input signals and measuring several performance parameters, such as Integral Time-weighted Absolute Error (ITAE), overshoot, rise time, settling time, and steady-state error (SSE). The results of two tests showed a consistent performance pattern.

The Ziegler–Nichols method demonstrated a very stable system response with a very small overshoot value (~1.61%). However, the response speed was relatively slow, with a rise time of 11.26 seconds and a settling time that was not reached within the 20-second simulation duration, in this case ~14.55 seconds. The high ITAE value (22.08894) also indicates that the system's accumulated error is still large. This is consistent with the characteristics of the Ziegler–Nichols method, which was initially developed for heuristic initial PID tuning of continuous-time linear systems without regard for optimal performance [2].

In contrast, Genetic Algorithm method demonstrated significantly more responsive performance. With a rise time of less than 3 seconds and a settling time of approximately under 3 seconds, this method was able to quickly adjust the system to achieve stability. Although its overshoot was higher than PSO (approximately 1.18%–0.20%), GA remained superior in terms of speed and effectiveness in reducing error, as widely reported in the control optimization literature [14]. The lower ITAE value (~0.92–0.94) compared to ZN indicates that the GA tuning results were more efficient in reducing the total error over time.

The Particle Swarm Optimization method demonstrated the most balanced performance of the three. With a smaller overshoot (approximately ~0%) and rise and settling times nearly equivalent to GA, PSO provided excellent results in the context of a tradeoff between stability and speed. PSO is known to excel in exploring and exploiting global solution spaces without requiring many explicit parameters, as demonstrated in the literature15,16]. The similar ITAE values to GA (~0.79) indicate that PSO is also capable of effectively minimizing errors.

In terms of steady-state error, both GA and PSO yielded the same final value (~0.0000), significantly lower than the ZN method. This indicates that both AI-based methods can provide more accurate and precise PID tuning, consistent with previous studies on artificial intelligence in system control[17].

Overall, intelligent optimization approaches such as GA and PSO have been shown to provide significantly superior PID tuning results compared to classical methods. GA is more suitable for systems requiring very fast response, albeit with slightly larger overshoot. Meanwhile, PSO is well-suited for systems requiring a compromise between stability and speed. On the other hand, the Ziegler–Nichols method still has practical value as an initial approach or rough reference for PID tuning, although its results are less than optimal in the context of modern dynamic performance.

The test results show that the Ziegler–Nichols (ZN) method provides the lowest performance, with high ITAE, significant overshoot, and long rise and settling times, despite its very short execution time. In contrast, artificial intelligence-based optimization methods, namely Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), consistently produce much better performance. Both methods are able to reduce ITAE by more than 95%, eliminate steady-state error, and accelerate rise and settling times by more than 80% compared to ZN. PSO shows more stable performance with zero overshoot and relatively consistent results, while GA tends to vary but still produces significant improvements.

The application of GA and PSO in control systems provides practical advantages, they are more efficient, accurate, and adaptive PID tuning compared to classical methods such as Ziegler–Nichols. Both of them are capable for optimation performance in complex systems that are nonlinear or have variable parameters, for example in industrial temperature control, electric motors, and renewable energy systems. PSO is faster in convergence, while GA is more flexible in finding optimal solutions, so both can be selected according to needs. Although they require higher computational costs, modern hardware developments allow offline and online implementations, and Python with its various libraries provides an alternative for researcher and engineers to do so.

4. CONCLUSION

Based on the simulation results, it can be concluded that tune PID parameters, can be improve use artificial intelligence algorithms, intelligent optimization-based tuning methods such as the Genetic

Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574

Algorithm (GA) and Particle Swarm Optimization (PSO) significantly improve performance compared to the classical Ziegler-Nichols (ZN) method.

The Ziegler–Nichols method demonstrates good system stability with very small overshoot values(~1.61%). ZN system response tends to be slow and less efficient in reducing the total error over time, as reflected by the high ITAE value more than 22. This makes the ZN method more suitable as an initial tuning approach or for systems that do not allow any overshoot at all. On the other hand, the Genetic Algorithm method demonstrates very fast system response, both in terms of rise time (~2s) and settling time(~2s), and very small steady-state error(~0). However, this method produces relatively higher overshoot(~0.2) than PSO value (~0). The Particle Swarm Optimization method provides the most balanced performance, with lower overshoot than GA while maintaining excellent response speed and error efficiency. ITAE values produce by PSO (~0.7)are nearly equivalent to those of GA(~0.9), demonstrating its high effectiveness in PID tuning.

Considering all evaluation parameters, the PSO method can be considered an optimal approach for systems that require a balance between speed and stability. GA, on the other hand, excels in applications that prioritize response speed. So, the Ziegler–Nichols method remains relevant as a baseline or initial reference, although it is not as efficient as the two intelligent methods in complex dynamic control. Last but not least Python with its libraries, is quite reliable for solving control problems both conventionally and by applying AI algorithms. In the future, further exploration can be carried out on the use of Python as a freeware for various experiments, especially in the field of control engineering.

Acknowledgments

LPPMP UNS through the Hibah Penguatan Kapasitas Group Riset (PKGR UNS) with Research Assignment Agreement Number 371/UN27.22/PT.01.03/2025.

REFERENCES

[1] K. Ogata, “Modern Control Engineering. in Instrumentation and Controls Series”. Prentice Hall, 2010. [Online]. Available: https://books.google.co.id/books?id=Wu5GpNAelzkC

[2] J. G. Ziegler and N. B. Nichols, “Optimum Settings for Automatic Controllers,” Journal of Fluids Engineering, vol. 64, no. 8, pp. 759–765, Nov. 1942, doi: 10.1115/1.4019264.

[3] G. van Rossum, “Python Programming Language,” in USENIX Annual Technical Conference, 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:45594778

[4] C. R. Harris et al., “Array programming with NumPy” Nature, vol. 585, no. 7825, pp. 357–362, Sept. 2020, doi: 10.1038/s41586-020-2649-2.

[5] J. D. Hunter, “Matplotlib: A 2D Graphics Environment,” Computing in Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007, doi: 10.1109/MCSE.2007.55.

[6] D. H. Kim, “Advanced Lecture for PID Controller of Nonlinear System in Python,” IJRTE, vol. 9, no. 6, pp. 20–29, Mar. 2021, doi: 10.35940/ijrte.F5375.039621.

[7] S. Fuller, B. Greiner, J. Moore, R. Murray, R. Van Paassen, and R. Yorke, “The Python Control Systems Library (python-control),” in 2021 60th IEEE Conference on Decision and Control (CDC), Austin, TX, USA: IEEE, Dec. 2021, pp. 4875–4881. doi: 10.1109/CDC45484.2021.9683368.

[8] P. Saraf, M. Gupta, and A. M. Parimi, “A Comparative Study Between a Classical and Optimal Controller for a Quadrotor,” Sept. 28, 2020, arXiv: arXiv:2009.13175. doi: 10.48550/arXiv.2009.13175.

[9] B. Smith, “Building a Simulated PID Controller in Python,” Medium. Accessed: July 31, 2025. [Online]. Available: https://medium.com/@bsmith4360/building-a-simulated-pid-controller-in-python-111b08ccae1a

Volume 12, Issue 2, October 2025, pp. 234-244
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v12i2.4574

[10] D. H. Kim and H. Alemayehu, "A study on Teaching Method of Control Engineering by Using Python Based PID," International Advanced Research Journal in Science, Engineering and Technology, vol. 7, no. 9, pp. 1–9, Sept. 2020, doi: 10.17148/IARJSET.2020.7901. R. V. Petrosian, I. A. Pilkevych, and A. R. Petrosian, "Algorithm for optimizing a PID controller model based on a digital filter using a genetic algorithm," in doors, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259115656

[11] A. A. Salem, M. A. Moustafa, and M. E. Ammar, "Tuning PID Controllers Using Artificial Intelligence Techniques.," 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:199020399

[12] A. Taeib, A. Ltaeif, and A. Chaari, "A PSO Approach for Optimum Design of Multivariable PID Controller for nonlinear systems," June 26, 2013, arXiv: arXiv:1306.6194. doi: 10.48550/arXiv.1306.6194.

[13] D. Goldberg, "Genetic Algorithm in Search, Optimization, and Machine Learning," Addison-Wesley, Reading, Massachusetts, vol. xiii, Jan. 1989.

[14] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proceedings of ICNN'95 - International Conference on Neural Networks, Nov. 1995, pp. 1942–1948 vol.4. doi: 10.1109/ICNN.1995.488968.

[15] D. Oliva, A. Ramos Michel, M. Navarro, E. H. Haro, and A. Casas, "Particle Swarm Optimization," 2023, pp. 49–71. doi: 10.5281/zenodo.7537827.