IJMECIE

International Journal of Mechanical, Electrical and Civil Engineering

E-ISSN: 3047-4531
P-ISSN: 3047-4523

Research Article

Development of a Web-Based Interactive Learning Platform for PID Control

Patricia Fernandez [1*], Ferry Hadary [2], and Seno D. Panjaitan [3]

[1] Department of Electrical Engineering, Faculty of Engineering, Universitas Tanjungpura; e-mail : d1021211001@student.untan.ac.id
[2] Department of Electrical Engineering, Faculty of Engineering, Universitas Tanjungpura; e-mail : ferry.hadary@invent.untan.ac.id
[3] Department of Electrical Engineering, Faculty of Engineering, Universitas Tanjungpura; e-mail : seno.panjaitan@ee.untan.ac.id
* Corresponding Author : Patricia Fernandez

Abstract: This study focuses on the development of an interactive web-based learning platform for Proportional-Integral-Derivative (PID) control systems, aimed at addressing the conceptual challenges faced by electrical engineering students when learning PID through conventional teaching methods. Despite its foundational role in control theory, PID remains difficult to grasp without practical visualization and hands-on experimentation. To bridge this gap, the research introduces a practical and accessible platform that enhances conceptual understanding through real-time simulations and physical interaction. The proposed system integrates key hardware components including an ESP-32 microcontroller, DC motor, rotary encoder, BTS 7960 motor driver, and I2C LCD. The platform's web interface is built using HTML, Tailwind CSS, and JavaScript, enabling intuitive user interaction. Motor response data is captured via the ESP-32 and transmitted to the web interface using the WebSocket protocol, allowing users to instantly visualize system behavior as PID parameters (Kp, Ki, Kd) are adjusted. This dynamic feedback mechanism enables students to observe changes in system characteristics such as rise time, overshoot, and settling time in real time. To evaluate the platform's feasibility, practicality, and educational effectiveness, beta testing was conducted among electrical engineering students using Likert-scale questionnaires. The results demonstrated that users were able to successfully interpret the impact of PID tuning on system performance. The average evaluation score reached 75.13%, indicating strong agreement regarding the platform's educational value and its effectiveness in enhancing learning outcomes. In conclusion, the study affirms that the developed web-based platform offers a feasible, engaging, and pedagogically effective alternative to traditional learning approaches. By combining interactive simulations with physical experimentation, the platform significantly improves students' understanding of PID control systems.

Keywords: ESP-32 Microcontroller, Interactive Learning, PID Control, Real-Time Simulation, WebSocket, Communication

Received: May 30, 2025;
Revised: June 30, 2025;
Accepted: July 12,2025;
Published : July 30, 2025
Curr. Ver.: July 30, 2025

1. Introduction

Proportional-Integral-Derivative (PID) control is a fundamental method widely used in various engineering fields, particularly in industrial automation and manufacturing systems. The PID algorithm consists of three key parameters—proportional (Kp), integral (Ki), and derivative (Kd)—that work together to optimize system performance by reducing error and improving stability, responsiveness, and accuracy [1]. Within the Electrical Engineering program at Tanjungpura University, PID control is a core component of the control systems curriculum. However, many students still struggle to grasp its conceptual and practical aspects due to conventional lecture-based teaching methods, which tend to be static and lack interactivity.

Previous studies have attempted to address this issue by introducing desktop-based simulations or simple PID training kits. For example, Åström et al. [2] developed interactive

learning modules (ILMs) for PID control using graphical interfaces. Similarly, Huba et al. [3] proposed a web-based PID tuning tool for DIPDT systems, and Mewada et al. [4] implemented autotuning techniques using ESP-32 microcontrollers. While these methods improved theoretical understanding, most lacked full integration between real-time visualization, physical experimentation, and flexible online access.

More recent advancements include the development of GUI-driven real-time PID control tools [5], IoT-based PID robotic arm simulations using ESP32 [6], and embedded WebSocket-MQTT architectures for remote PID tuning [7]. These tools aim to increase student engagement by merging embedded systems with interactive web platforms. For instance, Peters et al. [8] introduced a scalable ESP32-based platform for decentralized PID experiments, while Rofi'i et al. [9] created robotics modules for elementary students using IoT and PID control to facilitate early engineering education.

Furthermore, Márquez-Vera & Martínez-Quezada [10] demonstrated the relevance of microcontrollers like ESP32 in teaching signal processing and control to undergraduate students, and Pereira et al. [11] developed distance learning platforms utilizing ESP32 for remote control programming education. Integration of simulation, PID initialization, and parameter tuning with hardware is also emphasized by Rusli & Sabaruddin [12], highlighting Blynk-based IoT interfaces for intuitive student feedback.

The main research problem identified is the limited availability of interactive web-based platforms that provide hands-on experiences for PID learning. The proposed solution is to develop a web-based interactive learning platform utilizing the ESP-32 microcontroller to enable real-time tuning and visualization of PID system responses. This system integrates hardware and web interface components to bridge theoretical concepts with practical implementation, thereby improving students' comprehension and engagement.

This research contributes by (1) designing a hybrid simulation–experiment platform for PID learning, (2) offering a real-time WebSocket-based visualization mechanism, and (3) validating the system's effectiveness through quantitative user feedback. By combining embedded systems and educational technology, the study provides a flexible, scalable, and practical approach for enhancing PID instruction in engineering education.

2. Related Work

The development of effective PID learning methods has attracted interest across engineering education. Multiple researchers have proposed interactive tools and simulations to improve student understanding of control theory. Åström et al. [2], for example, developed three interactive learning modules (ILM-PID) focusing on core PID concepts such as loop shaping and anti-windup. These modules enhanced students' intuition through graphic-based simulations but lacked practical integration with physical devices, and were limited to continuous-time models without digital implementation.

Huba et al. [3] introduced a web-based application for PID tuning in Double Integrator Plus Dead Time (DIPDT) systems. The platform used the performance portrait method to display simulation results through an interactive graphical interface. While effective in providing visual feedback, this system was designed specifically for theoretical optimization rather than practical experimentation or integration with IoT hardware.

Another study by Wicaksono and Pramudijanto [13] applied the direct synthesis method for tuning PID controllers in DC motor speed control using PLC. Their results demonstrated significant improvements in control performance, but the hardware used (PLC) was less accessible for educational purposes compared to modern microcontrollers like the ESP-32. Similarly, Prasetyo et al. [14] implemented PID control on an AC motor using ESP-32, indicating the feasibility of using microcontrollers for real-time control, though the focus remained on industrial rather than educational settings.

Mewada et al. [4] proposed an advanced PID control system with auto-tuning based on the Ziegler–Nichols method. Their system utilized ESP-32 and a rotary encoder for closed-loop feedback, while the GUI interface was built using Python. However, the interface lacked integration into a web-based platform, limiting its portability and flexibility for broader learning environments.

Several tuning techniques have been studied in the context of educational PID implementations. Abdulameer et al. [15] provided a comparative analysis of PID tuning methods for DC motor speed control, showcasing how conventional Ziegler-Nichols approaches can be adapted for academic labs. Lestari and Hadi [16] demonstrated a multivariable nonlinear control process using a PI controller with Ziegler–Nichols tuning,

reinforcing its applicability in process control education. Ogata [17] further explains the theoretical basis of PID design and stability criteria in modern control education, serving as a foundational reference for control systems instruction. Meanwhile, Kusumah and Pradana [18] explored ESP32-based IoT interfacing in microcontroller labs, highlighting hardware-software integration for control-related coursework. Thohir et al. [19] investigated WebSocket libraries for real-time messaging in Android apps, emphasizing the relevance of such protocols in building responsive control interfaces.

Despite these advancements, gaps remain in delivering a web-based, real-time, interactive learning tool that combines both simulation and physical experimentation for PID learning. According to Hutahean [20], interactive learning is a digital approach that enhances student engagement by allowing users to actively participate with educational content and feedback, improving the learning experience beyond passive instruction.

This study addresses the identified gaps by designing a web-based interactive PID training platform using ESP-32, integrating real-time system response visualization and hands-on tuning capabilities. Unlike prior works, this platform bridges theoretical concepts with practical implementation, supports flexible remote access, and enhances student engagement through direct interaction with the control system.

3. Proposed Method

This section describes the systematic design and implementation steps for developing an interactive web-based PID controller learning kit. The method encompasses hardware design, web interface development, control algorithm programming, and testing based on Agile software development methodology.

3.1. System Design

System design involves the development of system architecture, modules, and user interfaces.

3.1.1. Hardware Architecture

The core of the system is the ESP-32 microcontroller, which handles communication, computation, and control signal generation. As illustrated in Fig. 1, the ESP-32 is powered by a battery and manages connections with motor drivers, LCD displays, and rotary encoders.

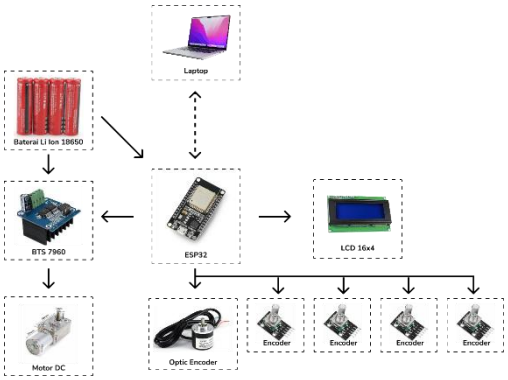

Figure 1. Hardware design

The inputs include four rotary encoders to manually adjust PID parameters (Kp, Ki, Kd) and setpoint, as well as an optical encoder to measure motor position and speed. The ESP-32 processes the input data, computes the PID output, and sends PWM signals to the BTS7960 motor driver to control the DC motor. The rotary encoder provides feedback for closed-loop control.

ESP-32 is connected to a laptop via Wi-Fi using Internet Protocol (IP) and WebSocket. It acts as both a WebServer and WebSocket server, enabling real-time, bidirectional communication with the browser. WebUI (Web User Interface) allows users to monitor system parameters and adjust PID values live, enhancing interactivity and control performance [20].

1. Casing Design: The casing is designed for stability, accurate sensor placement, and physical protection of components.

2. PCB Design: PCB layout is optimized to ensure efficient component connectivity and reduce signal interference.

3.1.2. WebUI Design

The WebUI interface provides real-time visualization of motor response, including rise time, peak time, settling time, and overshoot. Chart.js is used for rendering graphical output. Users can input Kp, Ki, Kd, and setpoint values, and visualize results instantly.

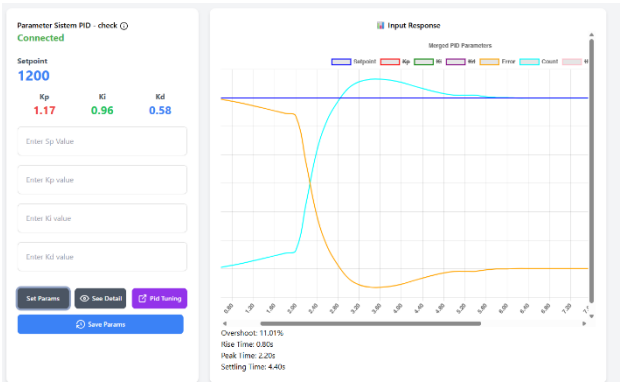

Figure 2. WebUI Interface Design

3.2. System Flowchart

The system initialization includes modules such as ESP32, encoder, driver, and network setup. If Wi-Fi is connected, a WebSocket is established. ESP32 periodically reads encoder values, computes PID control signals using the following discrete equation (1):

$$\boldsymbol{u(t) = K_p \cdot e(t) + K_i \cdot \int e(t)\,dt + K_d \cdot \frac{de(t)}{dt}} \quad (1)$$

The result is translated into PWM signals and motor direction. When recording mode is activated, encoder data is stored every 200 ms for 10 seconds and sent to WebUI for plotting and metric analysis.

3.3. Success Indicators

- The WebUI and hardware operate concurrently with synchronized outputs.
- User feedback indicates easier PID configuration than manual methods.
- Improved understanding based on questionnaire results.

3.4. Result Analysis

Results are evaluated quantitatively using system performance metrics (rise time, peak time, settling time, overshoot) and user satisfaction via questionnaire [21], [22].

From a system control perspective, four key time-domain performance indicators were measured: rise time, peak time, settling time, and overshoot. These metrics are essential to determine the effectiveness of PID tuning and real-time response of the DC motor system.

- Rise time (Tr) refers to the time it takes for the system response to increase from 10% to 90% of the final steady-state value. It indicates how quickly the system starts responding to a new setpoint.
- Peak time (Tp) is the duration required for the system to reach its first maximum (peak) value after the setpoint is applied. It reflects how soon the maximum response occurs.
- Settling time (Ts) is the time it takes for the system response to remain within a certain percentage (commonly ±2%) of the steady-state value without oscillating further. It indicates when the system becomes stable.
- Overshoot (Mp) measures how much the system exceeds the target value, usually expressed as a percentage. Excessive overshoot can indicate instability or overly aggressive tuning.

These metrics were extracted using the performance analysis algorithm implemented on the ESP32 (see *Section 3.6.5*), which captured encoder feedback in real time and performed threshold-based computation to determine critical time points. The metrics were transmitted

to the WebUI via WebSocket and displayed dynamically in tabular and graphical formats, allowing users to monitor system performance after each tuning iteration.

In addition to system metrics, the usability and effectiveness of the WebUI were evaluated through user questionnaires. Respondents, including students and faculty, indicated a significant improvement in understanding PID concepts and ease of parameter adjustment compared to conventional methods. The WebUI enabled live modification of PID values (Kp, Ki, Kd), instant visualization of motor responses, and simulation-based tuning, reducing trial-and-error and increasing control precision.

Overall, the results support the conclusion that the integration of WebUI and PID controller using ESP32 offers an interactive, accurate, and efficient environment for learning and experimentation. The system successfully demonstrated consistent performance improvements across multiple tests and user trials.

3.5. Agile Development Method

Following Trivedi [23], this project used Agile methodology:

- Requirements: Hardware/software needs defined with consultation.
- Design: UI mockups, circuit schematics, and physical casing.
- Development: Implementation of ESP32 control logic, WebUI, and database.
- Testing: Functional validation and comparison of hardware vs. simulation.
- Deployment: Integration and full system trial.
- Review: Weekly feedback and final evaluation with refinements.

3.6. Programming Configuration

This study implements control programming on the ESP32 microcontroller using Arduino IDE and C++ language. The control logic operates in a discrete domain, ensuring accurate sensor readings and control signal computations based on sampling intervals. The full implementation details and algorithmic snippets are presented below and referenced throughout the manuscript.

3.6.1. WebUI Communication

ESP32 establishes a WebSocket communication channel and transmits PID parameters using JSON format. The setup is initialized using setupWebServer() and handled with webSocketEvent() to receive commands and broadcast data in real time.

Algorithm 1. WebUI Communication Setup

```
// --- setup & inisialisasi kanal WS  ---------------------------
webSocket.begin();
webSocket.onEvent(webSocketEvent);
// --- pengiriman data realtime ke browser  ---------------------
void broadcastPIDData(const PIDData &data){
  StaticJsonDocument<256> doc;
  ...
  webSocket.broadcastTXT(out);              // kirim JSON ke semua klien
}
// --- handler pesan masuk dari browser  ----------------------
void webSocketEvent(uint8_t num, WStype_t type, uint8_t *payload, size_t len){
  ...
  xQueueSend(webCommandQueue,&cmd,pdMS_TO_TICKS(100));
}
```

3.6.2. PID Control Logic

The logic runs in hardwareTask() and pidTask(), reading encoder values and computing discrete PID values. The result is a PWM signal to drive a motor via H-bridge. The control law follows Equation (1), ensuring appropriate motor actuation.

Algorithm 2. PID Control Logic

```
count  = OpticEncoder.getCount();
degree = (count / 1200) * 360;
data_p = encoder2.getCount()*0.01;
data_i = encoder3.getCount()*0.01;
data_d = encoder4.getCount()*0.01;
```

```
error           = compensatedSetpoint - degree;
integral_error   += ((error + prev_error)/2)*0.01;
derivative_error  = (error - prev_error)/0.01;
u = data_p*error + data_i*integral_error + data_d*derivative_error;

// --- aktuasi motor (H-bridge, PWM ±255)  ----------------------
if(u >= 0){
  analogWrite(motor1Pin1,(int)u);  analogWrite(motor1Pin2,0);
}else{
  analogWrite(motor1Pin2,(int)(-u)); analogWrite(motor1Pin1,0);
}
```

3.6.3. WebUI Interface

JavaScript and Chart.js are used to visualize real-time system data. The interface also sends PID parameters through WebSocket, making the system responsive to user inputs.

Algorithm 3. WebSocket Integration in WebUI

```
//isi program html ….
<script>
ws = new WebSocket("ws://" + window.location.hostname + ":81/");
const values = {setpoint: sp, kp: kp, ki: ki, kd: kd};
ws.send(JSON.stringify({type:"setParamsAndCalculateOutput",values}));
<!-- --- terima data streaming & tambah titik ke Chart.js ---------- [pid_WebUI.h ≈ l.1750] -->
ws.onmessage = (e)=>{ const d = JSON.parse(e.data); updateMergedChart(d);
}
</script>
```

3.6.4. PID Tuning Simulation

A separate /pid_tuning endpoint enables simulation using Plotly. Ziegler-Nichols-based tuning computes PID values based on user-selected parameters, and results are visualized before being deployed to physical hardware.

Algorithm 4. PID Tuning and Visualization

```
//Program html ….
<script>
Plotly.newPlot(
        plotDiv,
        [
          {
            x: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
            y: [0, 0.3, 0.6, 0.85, 0.95, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
            type: "scatter",
            mode: "lines",
            name: "System Response",
            line: {
              color: "#4a86e8",
              width: 4,
              shape: "spline",
              smoothing: 1.3,
            },
            hovertemplate:
              "Time: %{x:.2f}s<br>Amplitude: %{y:.3f}<extra></extra>",
          },
          {
            x: [0, 10],
            y: [1, 1],
            type: "scatter",
            mode: "lines",
            name: "Set Point",
            line: { color: "#ff6b6b", dash: "dash", width: 3 },
            hovertemplate: "Set Point: %{y}<extra></extra>",
          },
        ],
```

```
      initialLayout
    );

    function calculateAndPlot() {
      plotLoading.style.display = "flex";
      setTimeout(() => {
        try {
          const coeffs = parseCoefficients();
          const num = coeffs.num;
          const den = coeffs.den;
          const setpoint = parseFloat(setpointInput.value) || 1;
          const stopTime = parseFloat(stopTimeInput.value) || 20.0;
          const responseTime = parseFloat(responseTimeSlider.value);
          const transientBehavior = parseFloat(transientBehaviorSlider.value);
          const Kp = parseFloat(kpSlider.value);
          const Ki = parseFloat(kiSlider.value);
          const Kd = parseFloat(kdSlider.value);

          updateTransferFunctionDisplay(num, den);

          let N;
          if (filterAutoMode) {
            N = calculateFilterCoefficient();
            filterCoefficientInput.value = N.toFixed(2);
            filterStatus.textContent = "Auto";
            filterStatus.style.color = "#28a745";
          } else {
            N = parseHighPrecision(filterCoefficientInput.value) || 10.0;
          }
          filterValue.textContent = N.toFixed(2);

          const simResult = simulateStepResponse(
            num,
            den,
            Kp,
            Ki,
            Kd,
            setpoint,
            N
          );
          const metrics = calculatePerformanceMetrics(
            simResult.response,
            setpoint
          );

          overshootValue.textContent = metrics.overshoot.toFixed(2) + "%";
          settlingValue.textContent = metrics.settlingTime.toFixed(2) + "s";
          riseValue.textContent = metrics.riseTime.toFixed(2) + "s";
          errorValue.textContent = metrics.error.toFixed(2) + "%";

          const setpointTrace = {
            x: [0, ...simResult.time],
            y: [0, ...simResult.time.map(() => setpoint)],
            type: "scatter",
            mode: "lines",
            name: "Set Point",
            line: { color: "#ff6b6b", dash: "dash", width: 3 },
            hovertemplate: "Set Point: %{y}<extra></extra>",
          };

          const responseTrace = {
            x: simResult.time,
            y: simResult.response,
            type: "scatter",
            mode: "lines",
            name: "System Response",
```

```
            line: {
              color: "#4a86e8",
              width: 4,
              shape: "spline",
              smoothing: 1.3,
            },
            hovertemplate:
              "Time: %{x:.2f}s<br>Amplitude: %{y:.3f}<extra></extra>",
          };

          Plotly.react(plotDiv, [responseTrace, setpointTrace], {
            ...initialLayout,
            title: `System Response | Setpoint: ${setpoint} | Stop Time: ${stopTime}s | Algorithm: ${currentAlgorithm}`,
            yaxis: {
              title: "Amplitude",
              gridcolor: "rgba(255, 255, 255, 0.1)",
              zerolinecolor: "rgba(255, 255, 255, 0.3)",
              range: [0, setpoint * 1.3],
            },
          });
        } catch (e) {
          console.error("Error in calculation:", e);
        } finally {
          plotLoading.style.display = "none";
        }
      }, 50);
    }

    responseTimeSlider.addEventListener("input", () => {
      updateSliderDisplays();
      if (!updatingSliders) {
        filterAutoMode = true;
        updatePIDSlidersFromPerformance(
          parseFloat(responseTimeSlider.value),
          parseFloat(transientBehaviorSlider.value)
        );
        calculateAndPlot();
      }
    });

    transientBehaviorSlider.addEventListener("input", () => {
      updateSliderDisplays();
      if (!updatingSliders) {
        filterAutoMode = true;
        updatePIDSlidersFromPerformance(
          parseFloat(responseTimeSlider.value),
          parseFloat(transientBehaviorSlider.value)
        );
        calculateAndPlot();
      }
    });

const url = `/pid_tuning/store_param?kp=${kp}&ki=${ki}&kd=${kd}&setpoint=${sp}`;
window.location.href = url;
```

3.6.5. Performance Metric Analysis

The system calculates performance metrics including rise time, peak time, settling time, and overshoot from encoder data. Algorithm detects key indices and updates WebUI accordingly.

Algorithm 5. Performance Metrics Calculation

```
// --- cari 10 % & 90 % (rise-time) ----------------------------
if(ascending){
  if(idx10==-1 && recordBuffer[i]>=threshold10) idx10=i;
```

```
    if(idx90==-1 && recordBuffer[i]>=threshold90){ idx90=i; break; }
  }
  // --- overshoot & peak-time ----------------------------------
  if(ascending && maxValue>finalValue){
    result.overshoot = ((maxValue-finalValue)/stepSize)*100.0;
    result.tp = peakIdx*sampleInterval;
  }
  // --- settling-time ±2 % band --------------------------------
  for(int i=bufferSize-1;i>=0;i--){
    if(recordBuffer[i]>upperBound||recordBuffer[i]<lowerBound){
      settlingIdx = i+1; break;
    }
  }
  result.ts = settlingIdx*sampleInterval;
```

4. Results and Discussion

This section presents the outcomes of the design, implementation, and testing phases. The analysis includes hardware and software outputs, system evaluation based on simulation and real-world testing, and quantitative user feedback. The findings are discussed in relation to the initial design objectives.

4.1. Hardware Design Result

The kit was developed using an acrylic top case and a PLA-based 3D-printed bottom case. The choice of materials provides durability and ease of customization (see Fig. 3 and Fig. 4).

Figure 3. Front View of the Kit (without motor and pulley)

Figure 4. Front View of the Fully Assembled Kit

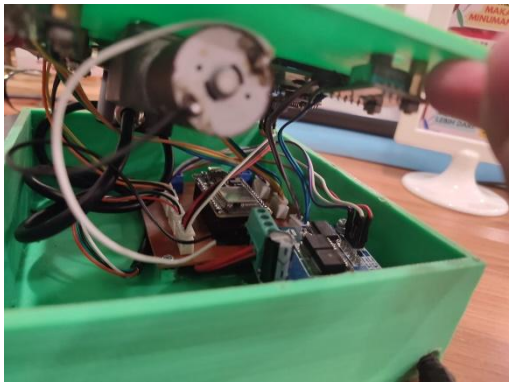

Figure 5. Internal Layout of the Kit

All components, including the DC motor, PCB, BTS7960 driver, and battery, were arranged compactly within an 18 cm × 15 cm × 7 cm enclosure to ensure system stability and optimal performance.

4.2. WebUI Implementation Res

An interactive WebUI was created to visualize system response and enable PID parameter input (Fig. 6). This interface communicates with the ESP32 through a local IP network, and its main features include real-time graph visualization, input forms for PID values, connection status, and system performance metrics.

Figure 6. WebUI Interface Showing System Response Graph

Additionally, a PID tuning page was implemented to automate parameter calibration based on real-time system data (Fig. 7). The WebUI makes the platform accessible even to users without technical programming knowledge [20].

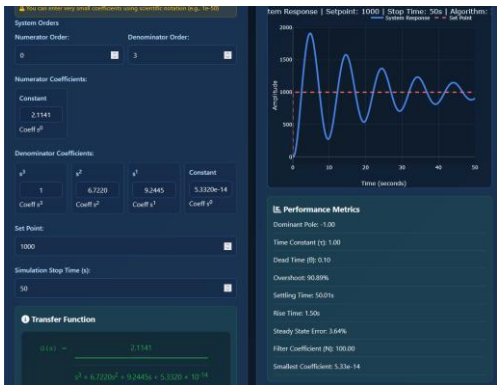

Figure 7. PID Tuning Simulation Feature on WebUI

4.3. Simulation and Kit Response Accuracy Test

Twenty test cases were conducted using various PID parameter combinations to compare simulated results with actual system responses. Table 1 shows test outcomes based on visual similarity and response metrics (rise time, peak time, settling time, overshoot).

Table 1. Test Data Comparing Simulation and Kit Responses

Test No	Setpoint	Kp	Ki	Kd	Result
1	1800	9	0.1	0.2	YES
2	1800	26	14	6	YES
3	1800	18.27	11.42	8.04	NO
4	1800	0.482	0.12	0.318	NO
5	1800	35.34	10.7	7.26	YES
6	1800	3	2	0.01	YES
7	1800	47.55	8.87	6.46	YES
8	1800	27	18	7	YES
9	1800	11.78	0.23	0.19	YES
10	1800	41.61	6.29	10.3	YES
11	1800	9	3	1	YES
12	1800	30	17	2	YES
13	1800	45	3	0.75	NO
14	1800	28	2	7.5	NO
15	1800	44.82	2	0.1	YES
16	1800	36.17	7.38	0.05	YES
17	1800	3.2986	2.41	1.863	YES
18	1800	17.57	14.21	6.58	NO
19	1800	49	16	9	YES
20	1800	5	1	1	YES

Out of 20 tests, 15 showed consistency between simulation and real kit, indicating a 75% match rate. Discrepancies in the remaining 25% were due to motor saturation, friction, and other nonlinearities not modeled in the simulation. This confirms that the transfer function approximation is valid but limited [21].

4.4. Usability and Comprehension Test

A Likert-scale questionnaire (Q1–Q12) was used to assess ease of use and understanding. Most users selected positive responses (P4, P5), indicating that the WebUI was intuitive and educational.

Table 2. Summary Statistics of User Feedback

Dimension	Mean	Std. Deviation
Ease of Use	4.35	0.64
Comprehension	4.25	0.55

Results suggest users found the system easy to operate and moderately challenging to understand in depth, consistent with expectations for a hands-on learning tool [22].

4.5. Learning Effectiveness Evaluation

Pretest and posttest evaluations were conducted to quantify learning gains. The improvement rate was calculated using Equation (2)[24]:

$$\textbf{\textit{Improvement}}\ (\%) = \frac{\textbf{\textit{Posttest}} - \textbf{\textit{Pretest}}}{\textbf{\textit{Pretest}}} \times \mathbf{100\%} \quad (2)$$

The average improvement in comprehension was 73.88%, indicating that the kit substantially enhanced understanding. Some outliers (e.g., 400% improvement) further highlight the system's potential for impactful educational outcomes.

4.6. Application Quality Evaluation

The ease of use scored highest (87%), followed by comprehension (85%) and simulation accuracy (75%). Cost efficiency (50%) was limited by hardware expenses. Replicability scored 80%, confirming the system's suitability for educational deployment with minor technical adjustments.

5. Conclusions

The development of a web-based interactive learning platform for PID control has successfully fulfilled its objectives. The WebUI and WebTuning modules were effectively implemented, allowing users to input PID parameters, monitor system performance in real time, and visualize motor responses graphically. These features enhanced user interaction and offered immediate feedback for better parameter tuning.

Experimental results demonstrated that most actual motor responses closely matched the simulated responses based on transfer function models. The observed differences, though minor, were due to real-world factors such as mechanical friction and system latency, which are not captured in simulation. Nevertheless, the alignment between theoretical and experimental outputs validates the system's reliability and accuracy.

Feedback from users via Likert-scale questionnaires indicated strong agreement regarding system usability, interactivity, and visual clarity. While ease of use received the highest ratings, comprehension scores were slightly lower, suggesting that conceptual understanding may still require support through mentoring or instructional content. The evaluation of learning improvement through pretest and posttest assessments revealed significant knowledge gains, with average understanding increasing by 73.8%. Some respondents achieved individual improvements exceeding 100%, highlighting the system's effectiveness as a learning tool.

This research contributes to control systems education by introducing a cost-effective, modular, and scalable learning tool that bridges theory with practical application. By enabling real-time experimentation, the platform enhances conceptual understanding and supports self-paced learning. It sets a foundation for similar implementations in other domains of engineering education.

Despite its effectiveness, the system has certain limitations. It currently lacks the ability to automatically recognize the order and type of the controlled plant, which could enhance adaptability. Additionally, the absence of long-term data storage features limits its use in extended experiments. To address these issues, future work should focus on integrating automated system identification, developing database functionalities, and producing supporting learning materials such as video tutorials or structured modules. With these improvements, the platform can evolve into a comprehensive and robust educational tool for broader academic application.

Author Contributions: Conceptualization: P.F. and S.D.P.; Methodology: P.F.; Software: P.F.; Validation: P.F., F.H. and S.D.P.; Formal analysis: P.F.; Investigation: P.F.; Resources: P.F.; Data curation: P.F.; Writing—original draft preparation: P.F.; Writing—review and editing: F.H.; Visualization: P.F.; Supervision: F.H.; Project administration: P.F.; Funding acquisition: S.D.P.

Funding: This research received no external funding.

Data Availability Statement: The data supporting the findings of this study are available from the corresponding author upon reasonable request. No publicly archived datasets were created or analyzed during this study due to privacy and ethical considerations.

Acknowledgments: The authors gratefully acknowledge the support from the Faculty of Engineering, Universitas Tanjungpura, particularly the Department of Electrical Engineering, as well as the guidance provided by supervisors, examiners, and academic staff. Sincere appreciation is also extended to all parties who contributed to the completion of this study.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

Abdulameer, A., Sulaiman, M., Aras, M., & Saleem, D. (2016). Tuning methods of PID controller for DC motor speed control. *Indonesian Journal of Electrical Engineering and Computer Science, 3*(2), 343. https://doi.org/10.11591/ijeecs.v3.i2.pp343-349

Ananthi, G., & Prabhakar, G. (2024). GUI-driven real-time PID control tool for educational virtual systems. *Journal of Engineering Education Transformations, 38*(2), 60–71. https://doi.org/10.16920/jeet/2024/v38i2/24190

Ardiansyah, Risnita, & Jailani, M. S. (2023). Teknik pengumpulan data dan instrumen penelitian ilmiah. *Ihsan Journal of Islamic Education, 2*, 1–9. [Online]. Available: http://ejournal.yayasanpendidikandzurriyatulquran.id/index.php/ihsan

Åström, K. J., Dormido, S., Hägglund, T., Berenguel, M., & Piguet, Y. (2008). Interactive learning modules for PID control [Lecture Notes]. *IEEE Control Systems, 28*(5), 118–134. https://doi.org/10.1109/MCS.2008.927332

de Souza Munhoz, L. G., & de Oliveira Assis, W. (2023). A web platform for learning control system based on IoT application. In *Proceedings of the ASEE Annual Conference & Exposition*. [Online]. Available: https://peer.asee.org/42549.pdf

Fadilla, Z., Ardiawan, M. K. N., Abdullah, M. E. S. K., Aiman, M. J. U., & Hasda, S. (2022). *Metodologi penelitian kuantitatif*. Yayasan Penerbit Muhammad Zaini. [Online]. Available: http://penerbitzaini.com

Hercog, D., Lerher, T., Truntić, M., & Težak, O. (2023). Design and implementation of ESP32-based IoT devices. *Sensors, 23*(15), 6739. [Online]. Available: https://www.mdpi.com/1424-8220/23/15/6739

Huba, M., Mižák, P., & Bisták, P. (2022). PID tuning for DIPDT system by web application. *IFAC-PapersOnLine, 55*(4), 201–206. https://doi.org/10.1016/j.ifacol.2022.06.033

Hutahaean, L. A., Siswandari, S., & Harini, H. (2019). Need analysis of the development of economics interactive e-module based on contextual teaching and learning for SMA. *Budapest International Research Critics Linguistic and Educational Journal, 2*(2), 343–350. https://doi.org/10.33258/birle.v2i2.309

Irhas, M., Asyiqah, S., Ilham, A., & Artikel, I. (2020). Penggunaan kontrol PID dengan berbagai metode untuk analisis pengaturan kecepatan motor DC. *Jurnal Fisika dan Terapan, 7*(1), 78–86. [Online]. Available: http://journal.uin-alauddin.ac.id/index.php/jft

Kusumah, H., & Pradana, R. A. (2019). Penerapan trainer interfacing mikrokontroler dan internet of things berbasis ESP32 pada mata kuliah interfacing. *Jurnal Cerita, 5*(2), 120–134. https://doi.org/10.33050/cerita.v5i2.237

Lestari, P. D., & Hadi, A. (2012). Desain PI controller menggunakan Ziegler Nichols tuning pada proses nonlinear multivariabel.

Márquez-Vera, M. A., & Martínez-Quezada, M. (2023). Microcontrollers programming for control and automation in undergraduate biotechnology engineering education. *Sci. African*. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772508123000406

Mewada, A., Sharma, R., & Patel, K. (2022). System for PID control and autotuning using Ziegler-Nichols method with ESP32. *International Journal of Innovative Technology and Exploring Engineering, 11*(2), 88–94.

Ogata, K. (2022). *Modern control engineering*. Prentice Hall.

Pereira, R., de Souza, C., Patino, D., & Lata, J. (2022). Platform for distance learning of microcontrollers and internet of things. *ProQuest Dissertation*. [Online]. Available: https://search.proquest.com/openview/3004342791bef23e4aa896ef111638ad

Peters, A. A., Vargas, F. J., Garrido, C., Andrade, C., & Villenas, F. (2021). Pl-toon: A low-cost experimental platform for teaching and research on decentralized cooperative control. *Sensors, 21*(6), 2072. [Online]. Available: https://www.mdpi.com/1424-8220/21/6/2072

Prasetyo, K. D., Fauziyah, M., & Adhisuwignjo, S. (2024). Kontrol PID pada pengaturan kecepatan motor AC berbasis ESP32. *Kohesi Journal of Multidisciplinary Science and Technology, 2*, 71–80.

Rofi'i, A., Afianah, N., & Kautsar, S. (2025). Development of robotics learning modules based on IoT and ESP32. *Journal of Technology and Application*. https://doi.org/10.25047/tefa.v2i3.5885

Rusli, M. H. M., Sabaruddin, M. H., M-Kayat, S., & Keat, C. S. (2024). Robust PID control of piezoelectric actuators with internet of things integration using Blynk application. In *Proceedings of the International Conference on Engineering, Technology and Education* (pp. 532–543). https://doi.org/10.2991/978-94-6463-589-8_50

Sugiyono. (2023). *Metode penelitian kuantitatif, kualitatif, dan R&D*. Alfabeta. [Online]. Available: http://www.cvalfabeta.com

Thohir, M. H. M., Pinandito, A., & Fanani, L. (2018). Perbandingan WebSocket pada komunikasi aplikasi perpesanan berbasis Android menggunakan library AndroidAsync, Java WebSocket, dan Nv WebSocket Client. [Online]. Available: http://j-ptiik.ub.ac.id

Trivedi, D. (2021). Agile methodologies. *International Journal of Computer Science & Communication, 12*(2), 91–100. [Online]. Available: https://www.researchgate.net/publication/356924683

Wicaksono, H., & Pramudijanto, J. (2004). Kontrol PID untuk pengaturan kecepatan motor DC dengan metode tuning direct synthesis. *Jurnal Teknik Elektro, 4*(1), 10–17. https://doi.org/10.9744/jte.4.1