ECEN315 CONTROL SYSTEM

Project

SMART WATER METERING USING IoT

Omar Samir Mohamed (211000372)

Mahmoud Reda Abdel Rauf (211000489)

Marwan Mohamed Zaki (211001365)

Abdullah Elkhatib (211001730)

Faculty Of Engineering Nile University

ECEN315: Fundamentals Of Control

Dr. Mohamed Saeed Darweesh

April 2024

Abstract

This study presents an innovative approach to water resource management that incorporates Internet of Things (IoT) technologies into water metering systems. Traditional water metering technologies are limited in terms of accuracy, real-time monitoring, and leak detection. The use of IoT-enabled smart water metering systems addresses these restrictions by enabling continuous data collecting, analysis, and remote monitoring. This paper provides a detailed investigation of the design, implementation, and evaluation of a smart water metering system. The system uses IoT sensors to detect water consumption in real-time, send the data to a centralized server for analysis, and provide insights into movements and irregularities. The findings show that the suggested approach is effective at improving water management practices, reducing water waste, and increasing overall water resource sustainability.

Introduction

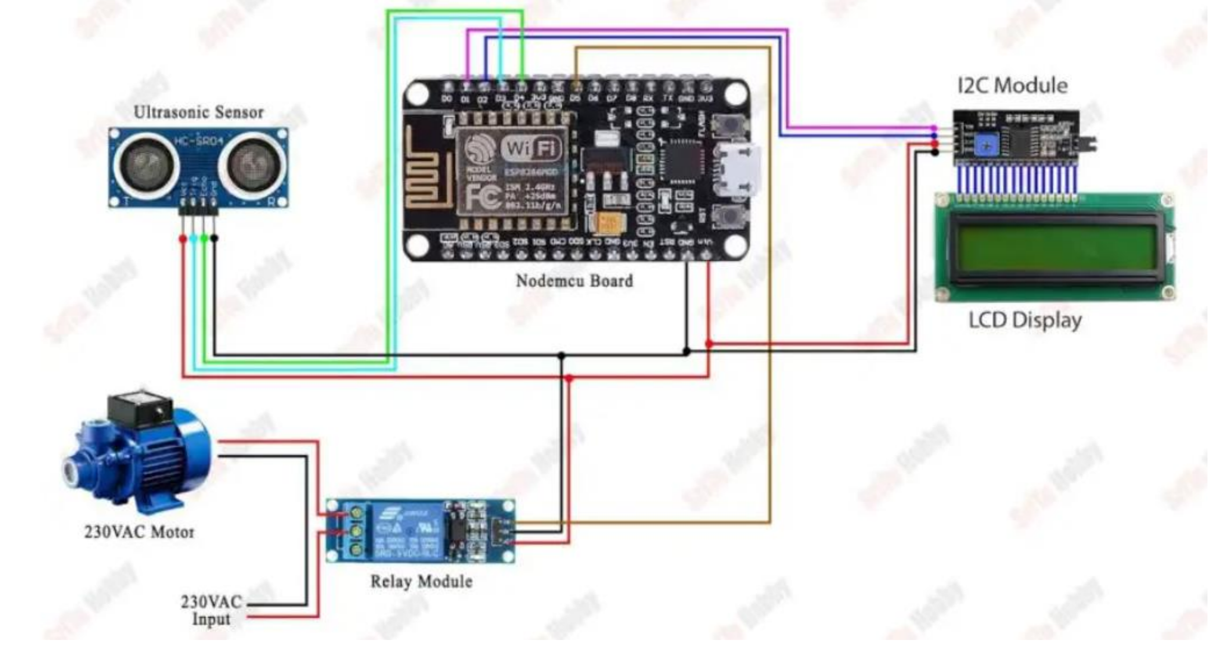
Poor water management and shortages offer significant challenges for individuals, businesses, and ecosystems worldwide. The demand for freshwater is increasing as the population grows and develops, putting further strain on the limited quantity of water resources. As a result, it is critical to implement cutting-edge solutions to address these pressing issues and ensure the long-term use of water.

Traditional water metering devices have long been used to measure commercial, industrial, and domestic water usage. nevertheless, these systems usually have several disadvantages, including low accuracy, the requirement for human understanding, and the inability to provide real-time insights into water usage patterns. Furthermore, water leaks frequently go undetected for lengthy periods, squandering a large amount of water and costing money.

To overcome these challenges, an increasing number of people are interested in developing smart water metering systems based on Internet of Things technology. These systems, which combine IoT sensors, communication networks, and data analytics capabilities, enable the optimization of water distribution networks, early leak detection, and continuous monitoring of water use.

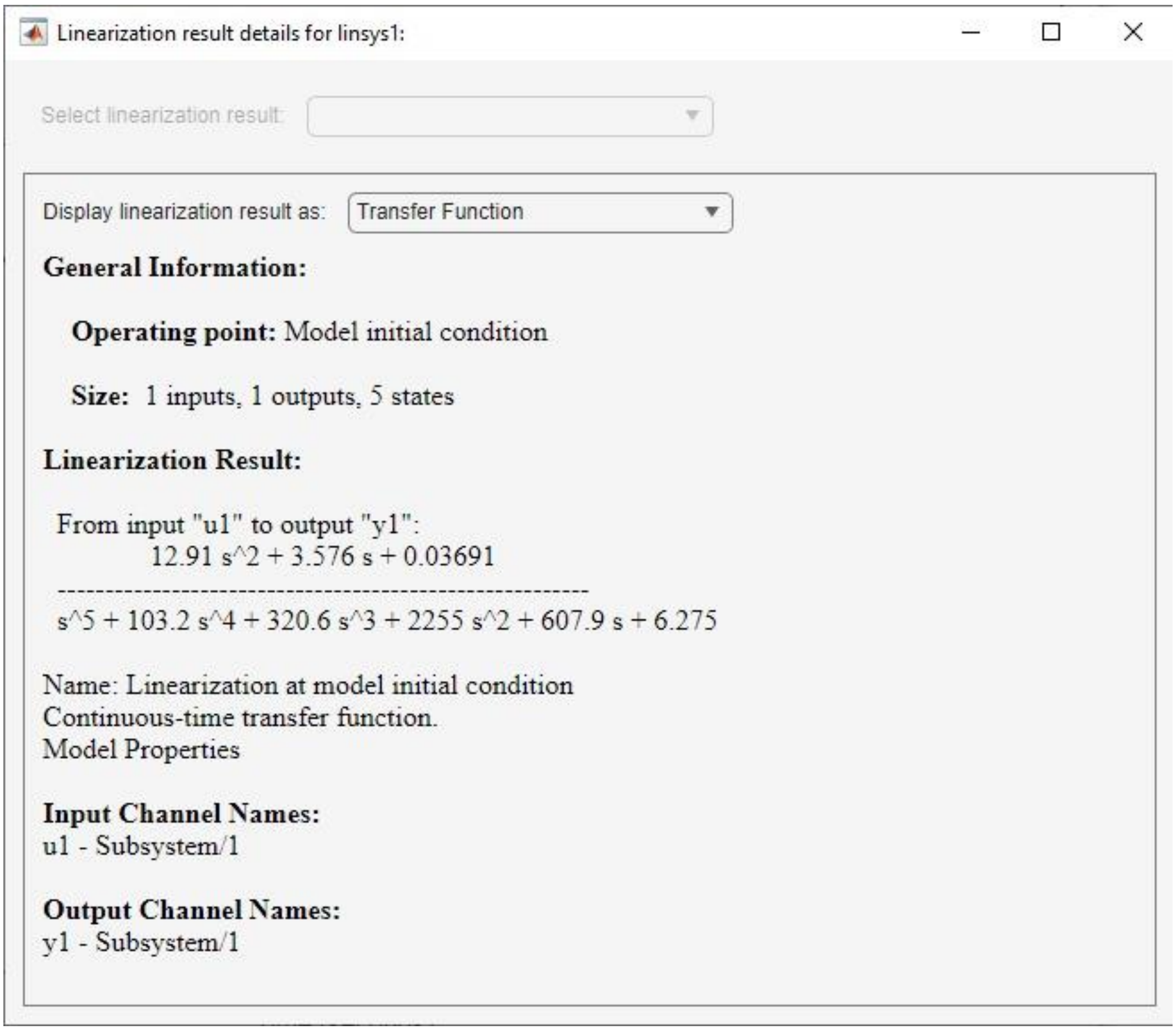
This study provides a detailed analysis of the design, implementation, and evaluation of an Internet of Things-based smart water metering system. We go over the system's primary components, including the user interfaces, data processing algorithms, communication protocols, and sensor nodes. We also run real-world deployment scenarios through the system to see how effective it is in improving resource efficiency and water management strategies.

Design
The Nodemcu board is Wi-Fi connected to the Blynk cloud while the project is running. The ultrasonic sensor also computes the distance to the water's surface. In other words, the water's level is determined. The Blynk app and LCD panel then display these values. You can also utilize the Blynk app's created button to turn on or off the relay module. In other words, a water pump connected to the relay module has the potential to be turned on and off.



The updated system transfer function after adding the controller

$$\text{T.F.}= \frac{12.91\,S^2 + 3.576\,S + 0.03691}{S^5 + 103.2\,S^4 + 320.6\,S^3 + 2255\,S^2 + 607.9\,S + 6.275}$$


Linearization result details for linsys1:
Select linearization result:

Display linearization result as: Transfer Function

General Information:

Operating point: Model initial condition

Size: 1 inputs, 1 outputs, 5 states

Linearization Result:

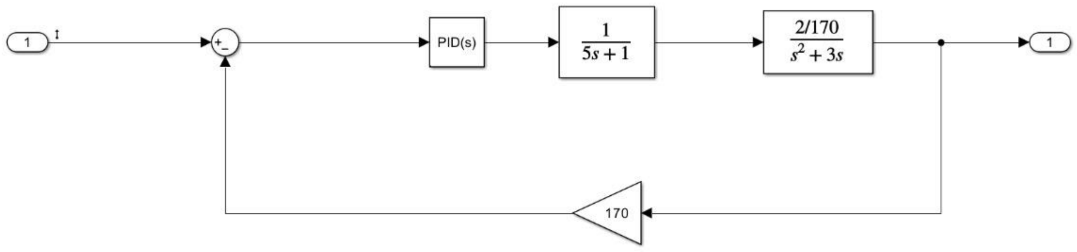
```
From input "u1" to output "y1":
        12.91 s^2 + 3.576 s + 0.03691
 ---------------------------------------------------
 s^5 + 103.2 s^4 + 320.6 s^3 + 2255 s^2 + 607.9 s + 6.275
```

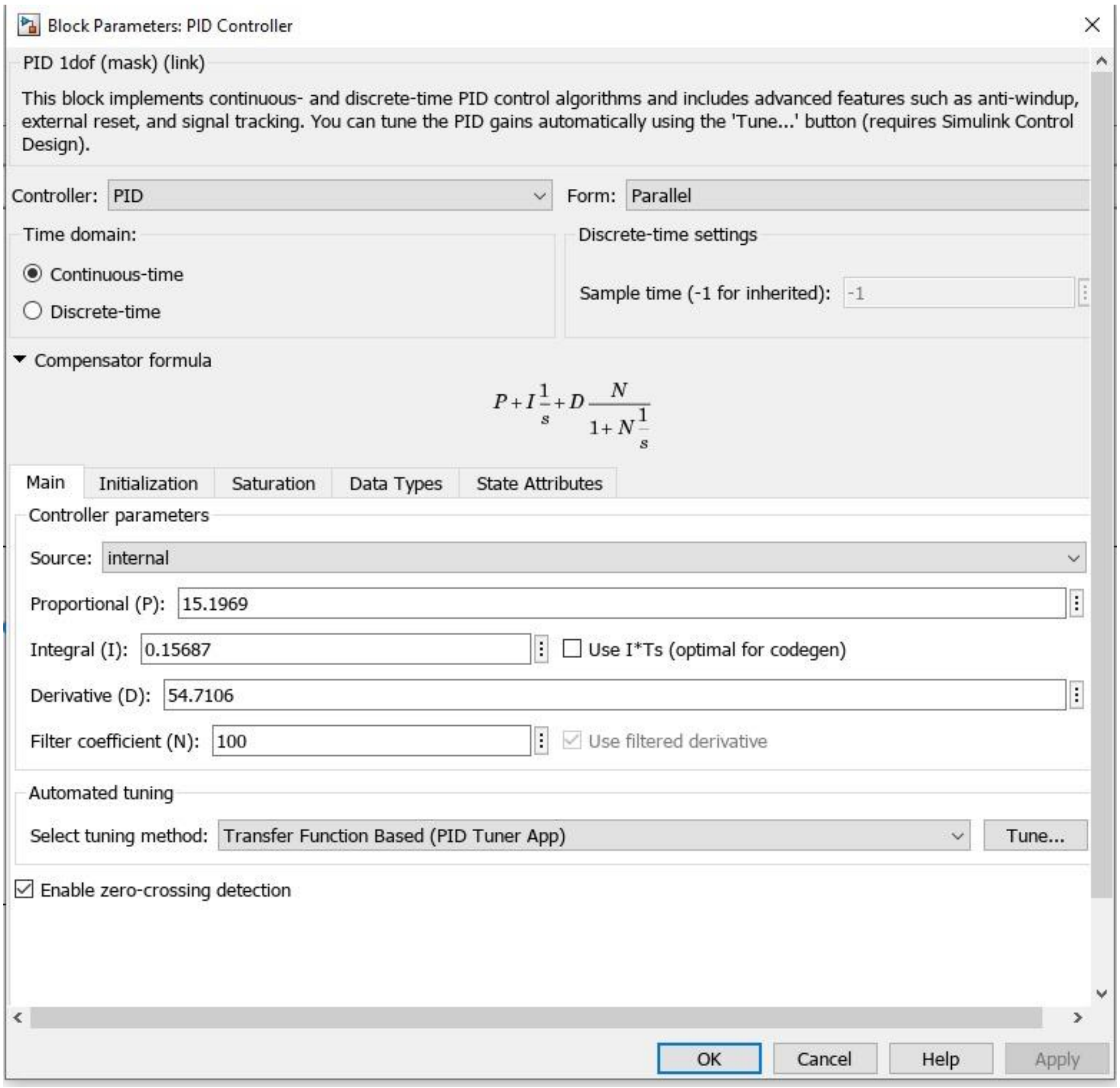
Name: Linearization at model initial condition
Continuous-time transfer function.
Model Properties

Input Channel Names:
u1 - Subsystem/1

Output Channel Names:
y1 - Subsystem/1

The Simulink model used to find the system response with figures




Block Parameters: PID Controller

PID 1dof (mask) (link)

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID

Form: Parallel

Time domain:

[✓ Continuous-time]

Discrete-time

Discrete-time settings

Sample time (-1 for inherited): -1

Compensator formula

$$P + I\frac{1}{s} + D\frac{N}{1+N\frac{1}{s}}$$

Main | Initialization | Saturation | Data Types | State Attributes

Controller parameters

Source: internal

Proportional (P): 15.1969

Integral (I): 0.15687

Use I*Ts (optimal for codegen)

Derivative (D): 54.7106

Filter coefficient (N): 100

[✓ Use filtered derivative]

Automated tuning

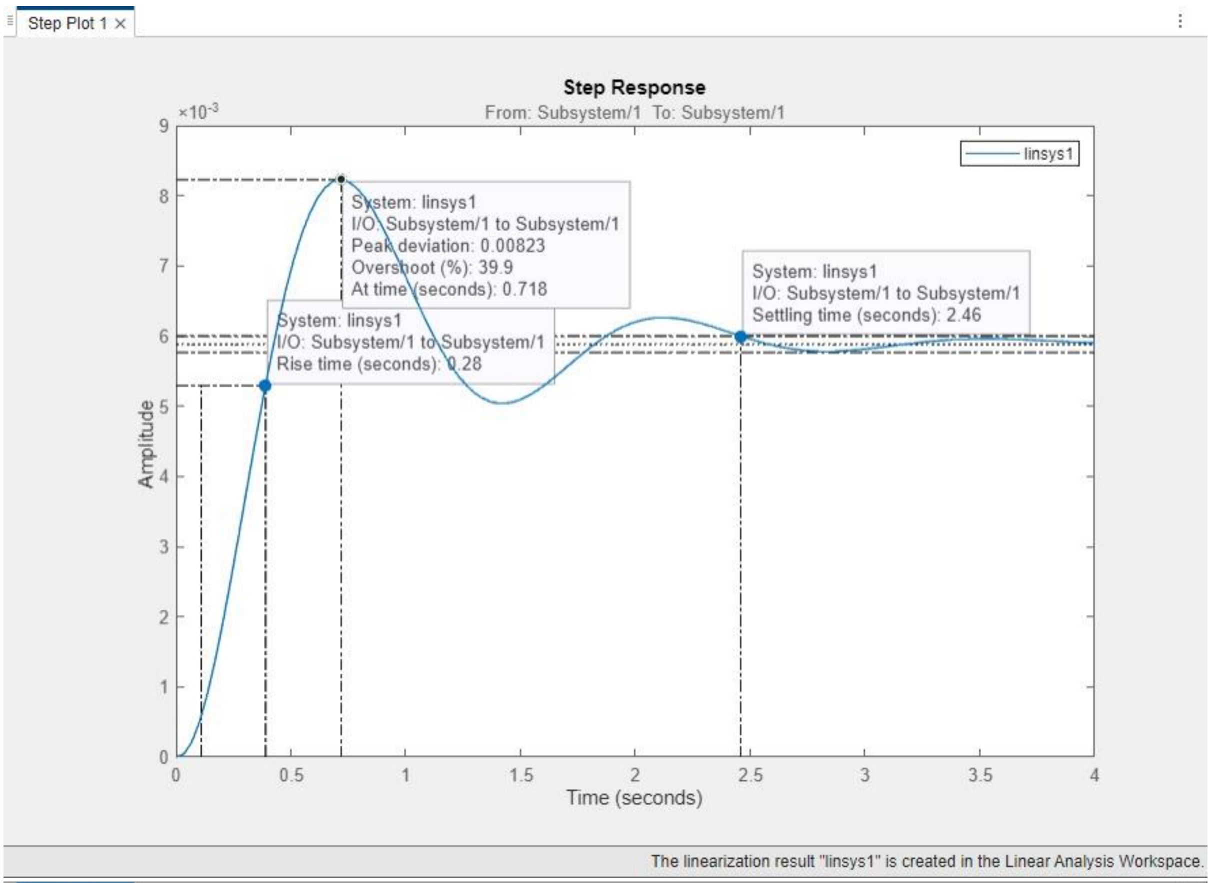
Select tuning method: Transfer Function Based (PID Tuner App)

Tune...

[✓ Enable zero-crossing detection]

OK Cancel Help Apply

Figures

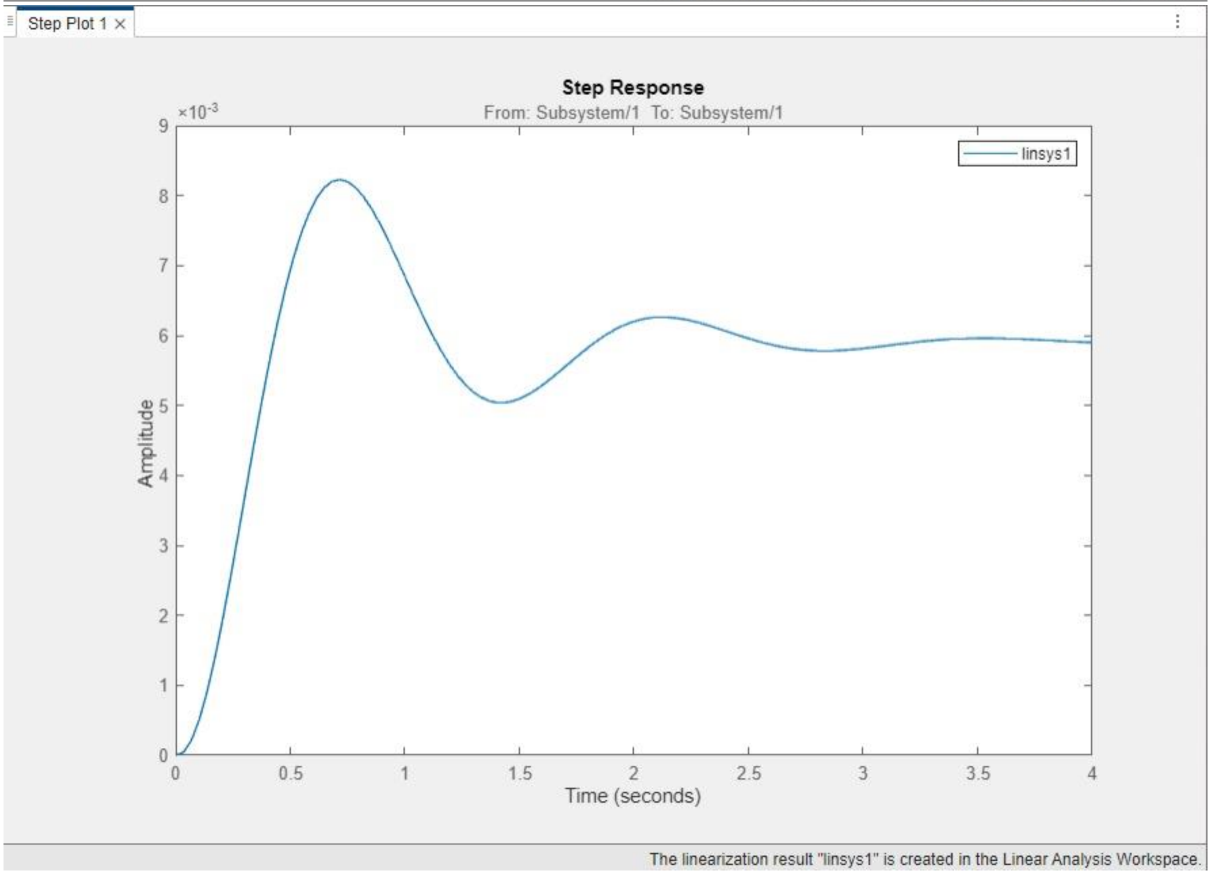


System: linsys1
I/O: Subsystem/1 to Subsystem/1
Peak deviation: 0.00823
Overshoot (%): 39.9
At time (seconds): 0.718

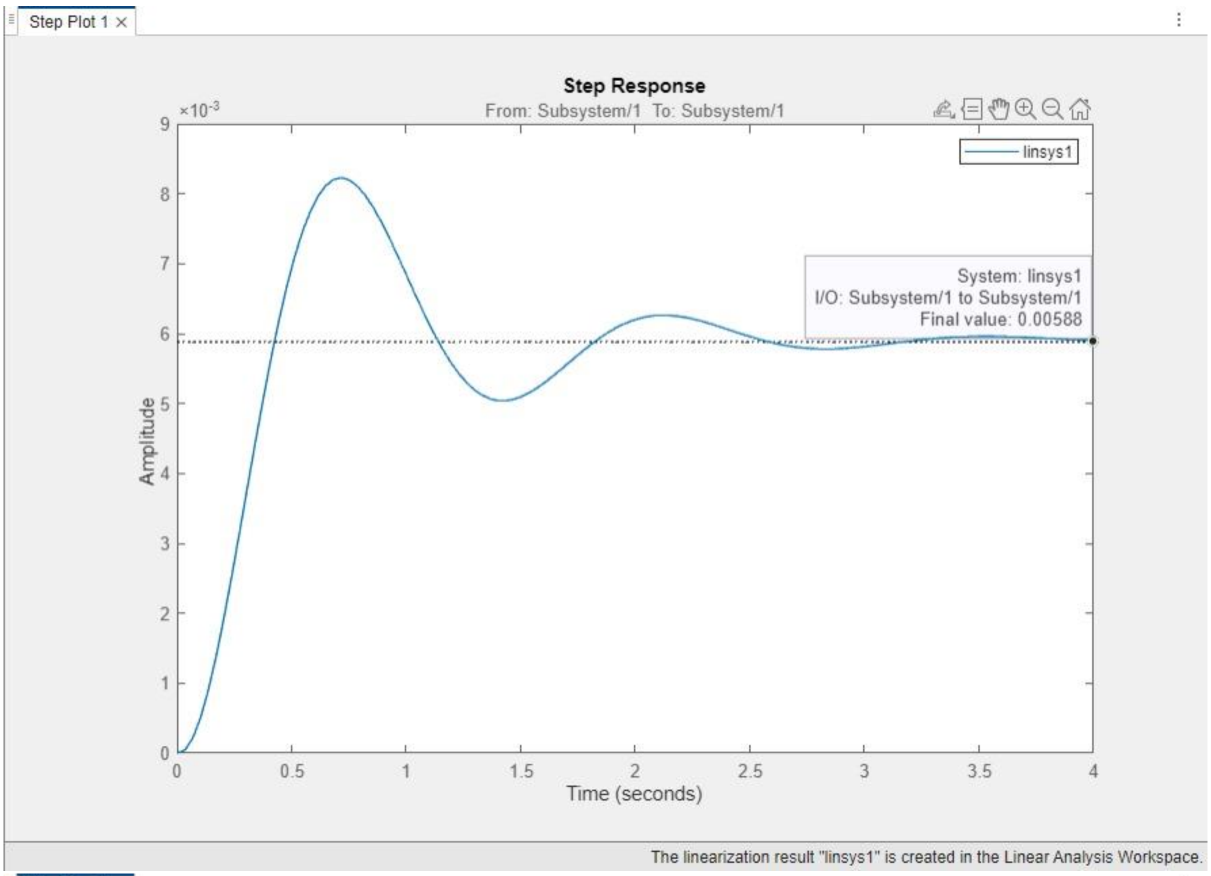
System: linsys1
I/O: Subsystem/1 to Subsystem/1
Settling time (seconds): 2.46

System: linsys1
I/O: Subsystem/1 to Subsystem/1
Rise time (seconds): 0.28

The linearization result "linsys1" is created in the Linear Analysis Workspace.

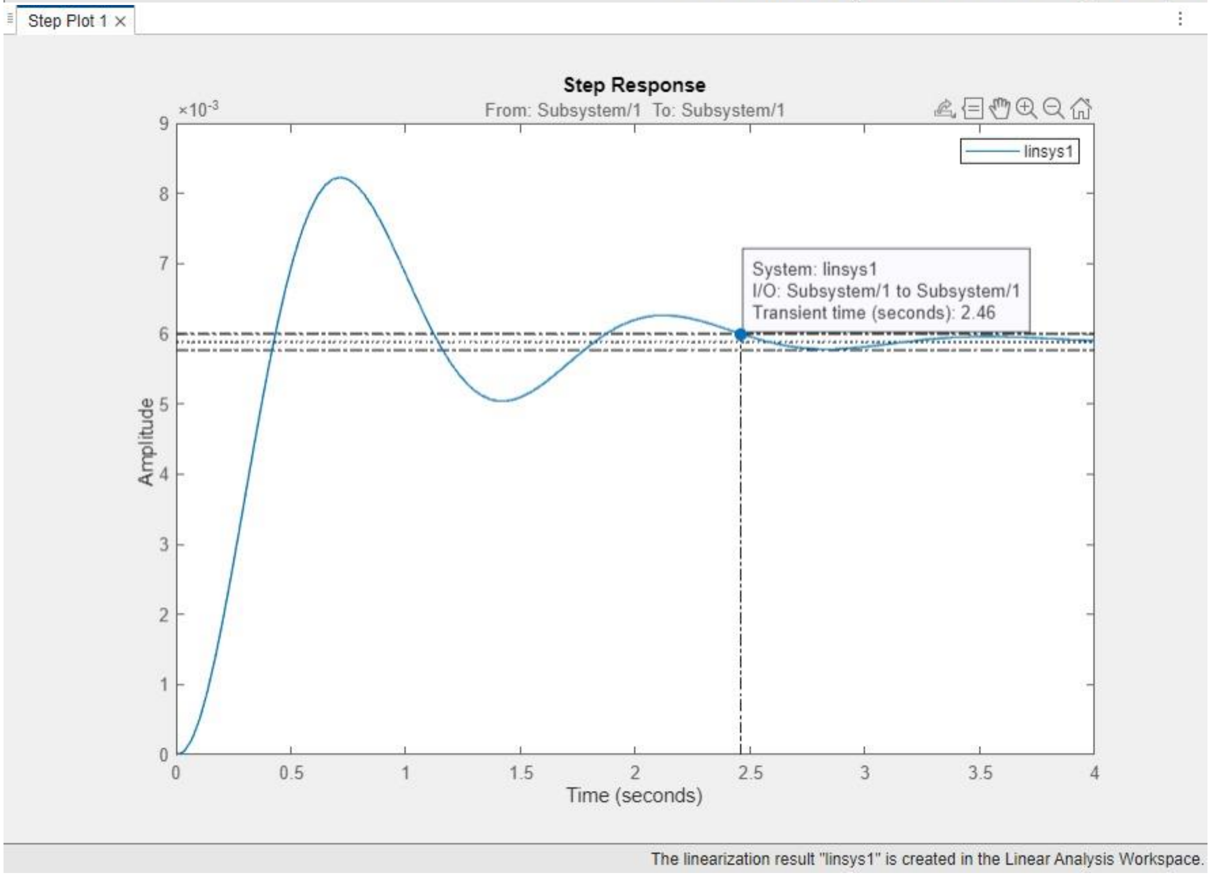


The linearization result "linsys1" is created in the Linear Analysis Workspace.


Step Plot 1

System: linsys1
I/O: Subsystem/1 to Subsystem/1
Final value: 0.00588

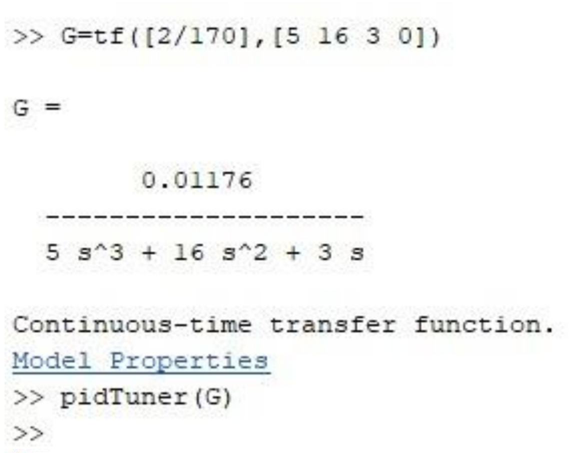
The linearization result "linsys1" is created in the Linear Analysis Workspace.


Step Plot 1

System: linsys1
I/O: Subsystem/1 to Subsystem/1
Transient time (seconds): 2.46

The linearization result "linsys1" is created in the Linear Analysis Workspace.

PID Tuner

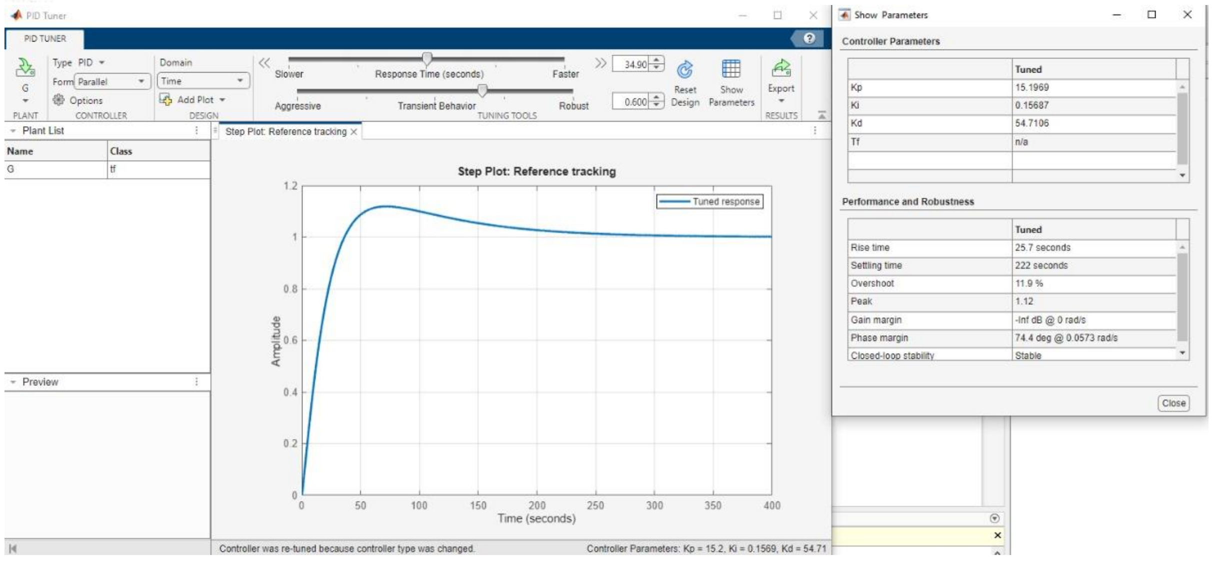


```
>> G=tf([2/170],[5 16 3 0])

G =

        0.01176
  --------------------
  5 s^3 + 16 s^2 + 3 s

Continuous-time transfer function.
Model Properties
>> pidTuner(G)
>>
```



PID Tuner — Type: PID, Form: Parallel, Domain: Time

Response Time (seconds): 34.90

Transient Behavior: 0.600

Plant List

Name	Class
G	tf

Controller was re-tuned because controller type was changed.

Controller Parameters: Kp = 15.2, Ki = 0.1569, Kd = 54.71

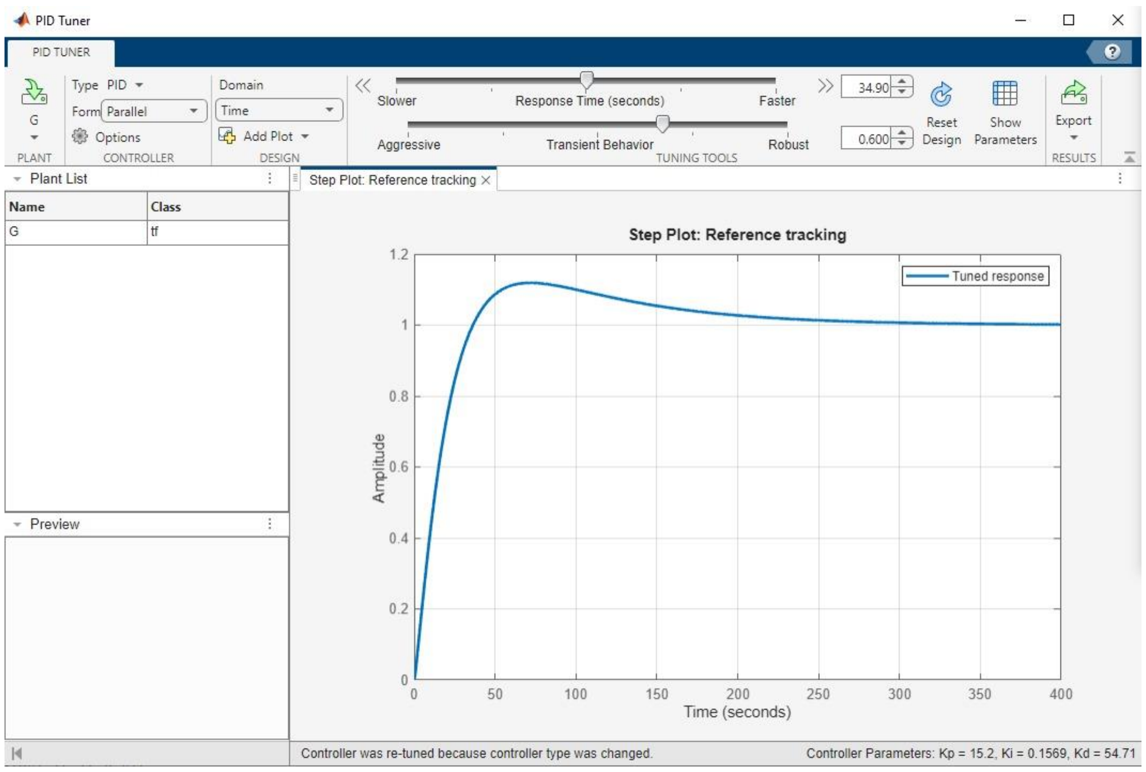
Show Parameters

Controller Parameters

	Tuned
Kp	15.1969
Ki	0.15687
Kd	54.7106
Tf	n/a

Performance and Robustness

	Tuned
Rise time	25.7 seconds
Settling time	222 seconds
Overshoot	11.9 %
Peak	1.12
Gain margin	-Inf dB @ 0 rad/s
Phase margin	74.4 deg @ 0.0573 rad/s
Closed-loop stability	Stable



PID Tuner — Type: PID, Form: Parallel, Domain: Time

Response Time (seconds): 34.90

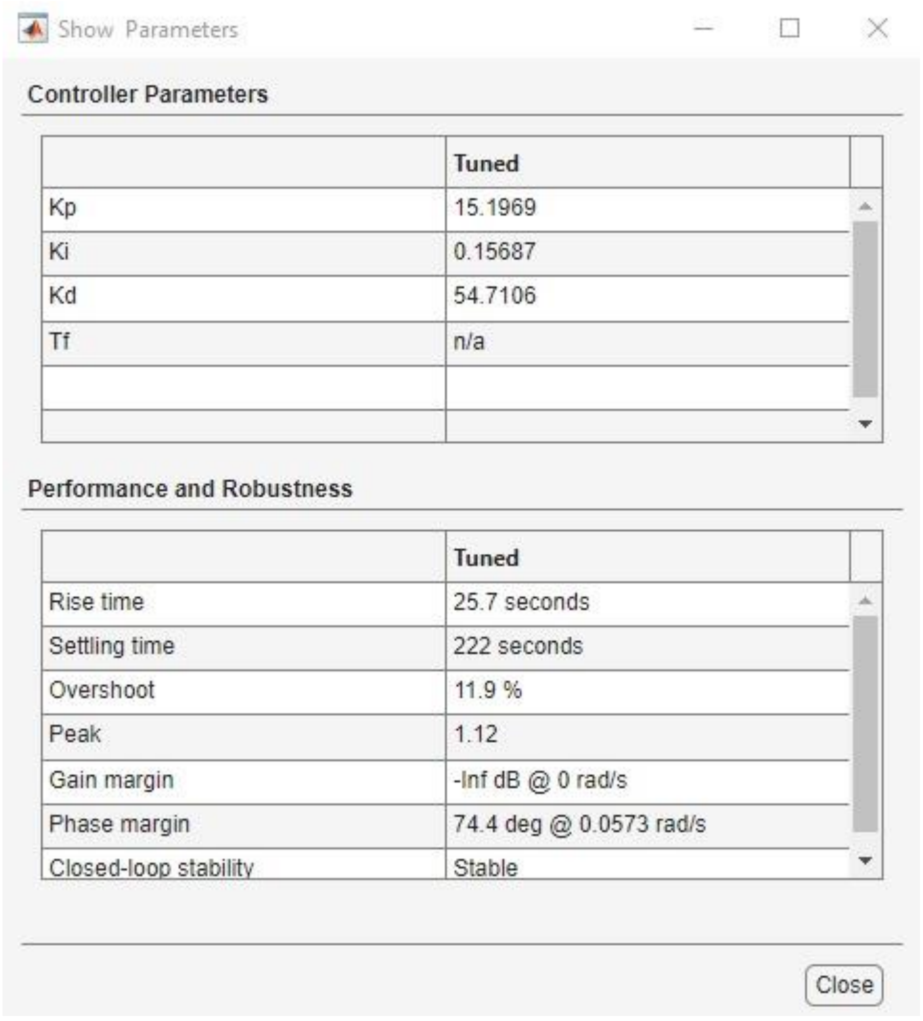
Transient Behavior: 0.600

Plant List

Name	Class
G	tf

Controller was re-tuned because controller type was changed.

Controller Parameters: Kp = 15.2, Ki = 0.1569, Kd = 54.71



Controller Parameters

	Tuned
Kp	15.1969
Ki	0.15687
Kd	54.7106
Tf	n/a

Performance and Robustness

	Tuned
Rise time	25.7 seconds
Settling time	222 seconds
Overshoot	11.9 %
Peak	1.12
Gain margin	-Inf dB @ 0 rad/s
Phase margin	74.4 deg @ 0.0573 rad/s
Closed-loop stability	Stable

Reasons for the difference in the PID Tuneer and Simulink

PID Tuner ignores nonlinear settings in the PID Controller block, which can cause the PID Tuner to give a different response from the simulation.

The Simulink model has strong nonlinearities in the plant that make the linearization invalid over the full operating range of the simulation.

Hardware implementation of the system via video

https://drive.google.com/file/d/1nU8RpYNuMyQb5AY5QEafjJqx2lyrJKsD/view?usp=drivesdk

Comparison between software response and hardware response

- **Settling time in the hardware is 29 seconds**
- **Settling time in the Simulink is 2.46 seconds**
- **Settling time in the PID Tuner is 222 seconds**

Hardware components

1- Ultrasonic Sensor (Calculates the distance to the surface of the water.)
2- Nodemcu ESP32 (Nodemcu board is connected to the Blynk cloud via WIFI.)
3- Relay Module (The water pump is connected to the relay module, it can be turned ON / OFF.)
4- LCD Display (These values are then displayed on the Blynk app and LCD.)
5- I2C Module (Connected to the LCD)
6- Connecting Wires
7- Breadboard
8- Water Tank
9- Motor 230 V AC Motor

Esp32 Code

```
#define BLYNK_TEMPLATE_ID "TMPL29MSDP0lG"
#define BLYNK_TEMPLATE_NAME "ESP 32"
#define BLYNK_AUTH_TOKEN "sN66wBPiM7fsD11NcP9bZUbkgRemnoOx"

#define BLYNK_PRINT Serial
#include <LiquidCrystal_I2C.h>
#include <Wire.h>
#include <BlynkSimpleEsp32.h>
#include <WiFi.h>
#include <WiFiClient.h>

// Blynk authentication
char auth[] = "sN66wBPiM7fsD11NcP9bZUbkgRemnoOx";
BlynkTimer timer;

// WiFi credentials
char ssid[] = "WE_493C6C";
char pass[] = "95443210";

// LCD setup
#define lcdColumns 20 // Adjusted to fit a standard 16x2 or 20x4 LCD
#define lcdRows 4
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

// Ultrasonic sensor pins
#define echoPin 2
#define trigPin 4
#define VPIN_ULTRASONIC V7

// Relay pin
#define relayPin 5
```

```
long duration, distance;

void setup() {
  Serial.begin(9600);

  // Initialize the LCD
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Smart water metering");

  // Initialize Blynk
  Blynk.begin(auth, ssid, pass, "blynk.cloud", 80);

  // Initialize ultrasonic sensor pins
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  // Initialize relay pin
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, LOW); // Ensure relay is off initially
  // Setup timer to call water level function periodically
  timer.setInterval(1000L, waterlevel); // 1 second interval
}
void waterlevel() {
  // Trigger the ultrasonic sensor
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Read the echo pin and calculate distance
  duration = pulseIn(echoPin, HIGH);
  distance = duration / 58.2;
  // Convert distance to string
  String disp = String(distance);
  // Print distance to Serial monitor
  Serial.print("Distance: ");
  Serial.print(disp);
  Serial.println(" cm");
  // Send distance to Blynk
  Blynk.virtualWrite(VPIN_ULTRASONIC, distance);
  // Update the LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Distance:");
  lcd.setCursor(0, 1);
  lcd.print(disp + " cm");
  // Check distance and control relay
  if (distance <= 2) {
    // Turn off the motor
    digitalWrite(relayPin, HIGH); // Assuming HIGH turns off the relay
    Serial.println("Motor stopped - Water level too high!");
  } else {
    // Turn on the motor
    digitalWrite(relayPin, LOW); // Assuming LOW turns on the relay
    Serial.println("Motor running");
  }
}
void loop() {
  Blynk.run();
  timer.run();
}
```

Reference

1- https://users.cs.fiu.edu/~msha/580i/2020fall_WaterLevelMonitoringSystem.pdfs
2- https://how2electronics.com/iot-based-water-level-control-monitoring-with-esp8266/
3- https://electronicscoach.com/liquid-level-control-system.html
4- https://www.elprocus.com/ac-servo-motor/#:~:text=The%20transfer%20function%20of%20the,i%2Fp%20of%20the%20system.
5- https://srituhobby.com/water-level-indicator-using-nodemcu-and-ultrasonic-sensor-step-by-step-instructions/#google_vignette
6- https://www.mathworks.com/help/slcontrol/ug/simulated-response-does-not-match-the-pid-tuner-response.html