International Journal Publishing
INFLUENCE: International Journal of Science Review
Volume 4, No. 1, 2022
https://influence-journal.com/index.php/influence/index

Research Article

System for PID Control and Autotuning PID using Ziegler-Nichols Method

Yash Mewada[1], Sarvesh Prajapati[2], Raj Hakani[3]
[1,2]UG Research Scholar, Robotics Club, Gujarat Technological University, India
[3]Assistant Professor, Gujarat Technological University, India
Email: yashmewada9618@gmail.com

Academic Editor: Nguyen Ngoc Anh

Abstract. The impact of automation had drawn our attention towards the advancements of closed-loop control systems (feedback/feedforward control systems). A PID control system is used in almost every industrial as well as commercial application. Therefore, it becomes important to efficiently tune these PID control systems. Various advancements are done in auto-tuning these parameters. However, due to the absence in the existing system, which implements the PID control as well as auto-tunes the parameter, this paper aims at constructing a system, which eliminates the hectic process of tuning the PID manually and implementing PID control with ease.

***Keywords**: PID Control, Process Control, Design, Ziegler-Nichols Tuning, DC Motor, Encoders.*

A. INTRODUCTION

With the current advancements in Control theory [1] and its subsidiaries, feedback and feedforward control systems and their design have been the topmost priority. An ordinary control system design [2] is the feedback control system which is been used in this system. A conventional feedback control system consists of feedback from various sensors connected in the peripheral which provides a more robust and customizable process. The current PID implementation system available in the market uses a tedious process for auto- tuning functionality for instance if the user wants to auto-tune the motor or load only proprietary software and hardware must be used and also there are compatibility issues. Furthermore, there are instances in which the user needs to re-tune the motor in case of the system overshoots beyond the hardcoded dead-band because of several factors [3]. For our system, we used incremental encoders as feedback and a 30A motor driver to drive a heavy load motor. This motor driver is controlled by a dual-core Bluetooth controller (ESP32).

The selection of a PID controller from its types is system depended. There are in general 6 subsidiary types of this controller. They are as below:

1. P-Controller: Proportional Controller, a linear controller with respect to the error.
2. I-Controller: Integral Controller, the output signal is directly proportional to the integral of the error.
3. D-Controller: Derivative Controller, the output signal is directly proportional to the derivative of the error.

Combination of controllers based on the above controllers.

1. PI-Controller: **P**roportional and **I**ntegral Controller, it increases the short-term response of the controller.
2. PD-Controller: **P**roportional and **D**erivative Controller, it reduces the steady-state error of the system[1].

[1]An error when the system reaches a steady-state i.e. when there are no more oscillations and damping in the system.

3. PID-Controller: **P**roportional, **I**ntegral and **D**erivative Controller, a combination of both PI and PD controller which eliminates the drawbacks of both PI and PD controller.

However, it may induce oscillations and damping if not designed carefully hence it is advisable to use a PD controller in the same scenarios. But in our case implementing a PD controller resulted in presence of offset error and instability as our system had a large gain.

B. HARDWARE

1. ESP32 [4] – Bluetooth Controller. A dual-core 32 Bit microcontroller with Bluetooth functionality. Almost all the pins in this controller are analog output and interrupt pins.
2. AMT103 [5] – Incremental Encoder. Capacitive encoder with programmable 48 – 2048 PPR. This encoder is compatible with the 2-8mm diameter of the motor shaft[2].
3. Cytron MDDS30 [6] – Motor Driver. A dual-channel 30A continuous and 80A peak current capacity smart motor driver.
4. Maxon DC Max 22S [7] – Brushed DC Motor. A Graphite brushed DC motor with a customized gearbox to increase the torque of the motor up to 100N.

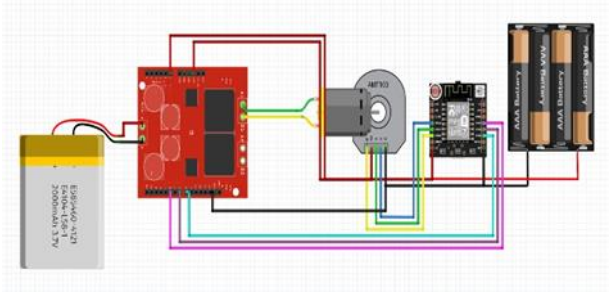

Figure 1: Circuit Connection of Proposed System

All the components are connected as shown in the figure (Figure 1). This replication of connection was done in Fritzing [8] and it should be noted that due to the unavailability of specific components in the software, we've used a different motor, motor driver and microcontroller. For power supply, we are using a 5200mah, 50C, 16.4v Lithium-Polymer battery.

C. RESULTS

Figure 2 depicts the design of a basic PID Controller required to control a DC motor. We are using Arduino IDE to program the microcontroller and using serial communication from the computer to send data to the microcontroller.

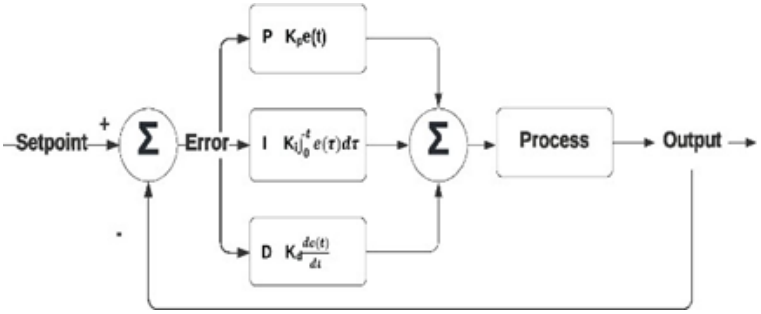

Figure 2. Block Diagram of PID Controller used in Proposed System

[2]Pulse Per Revolution is the number of pulses in one full turn of an encoder in a 90- degree phase shift.

We used Arduino PID Library [9] and Arduino Auto-Tune Library [10] so that there are no compatibility issues, and the code is accessible by everyone else.

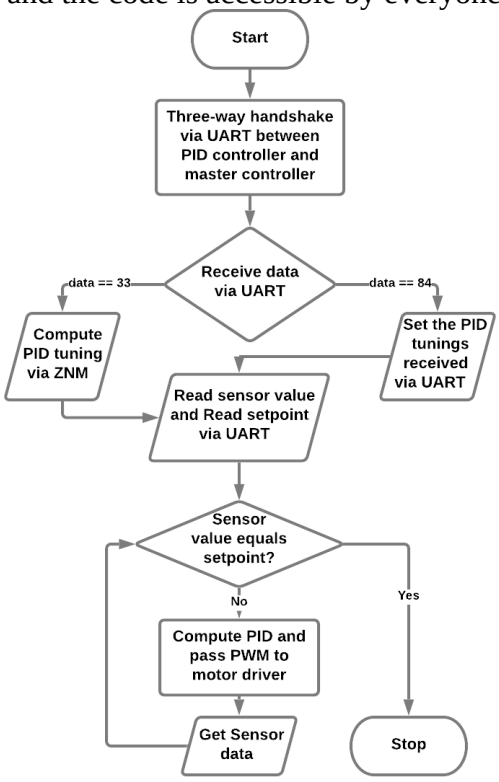

Figure 3: Flowchart of the Proposed Methodology

Initially, we start with a three-way handshake [11] to confirm that the initialization was successful. After the initialization, we send "33" as data to tell the controller to auto-tune. Note that this all is done in the GUI (refer to Figure 4) created by us to make the process easier. The source code and GUI are provided in the references section [13].

Figure 4: Graphical User Interface of the System

Once we begin to auto-tune the system using the Ziegler-Nichols [14] method, we need to provide the maximum output limit of the system, so that the controller knows what maximum output is required to produce. Auto-tuning takes 10 seconds for the process, and values of K_p, K_i, and K_d are displayed on the GUI. After auto-tuning, we enter the desired setpoint in the GUI, with which PID comes in action and the motor reaches the desired setpoint. Though this system was designed for controlling the motor, this system works for almost everything, wherever implementation of PID is needed.

For the end-user, one should connect the controller with the computer and input the required parameters. Also, the connections can be in any pair as long as they match this methodology.

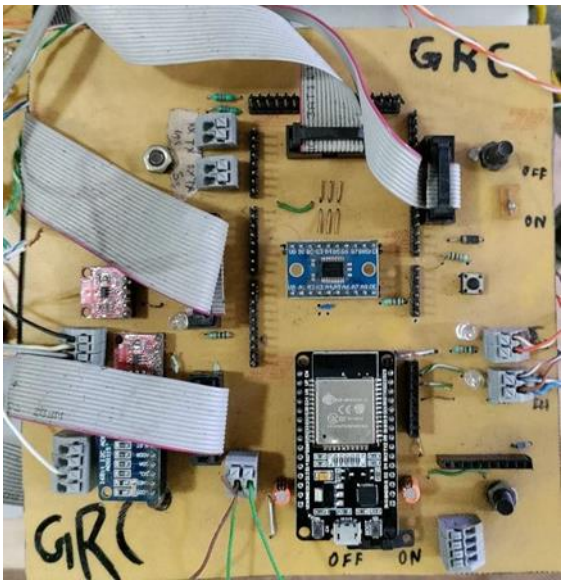

Figure 5: Hardware of the proposed methodology

Figure 5 shows the hardware used while testing the system. Although this contains some unnecessary components, they were not used by us while testing as we did not feel the need to redesign the hardware from scratch.

For the time being, we tested our logic on a pre-defined system considering the ideal conditions and constant power supply. A constant power supply is one of the limitations of our system if we neglect some of the hypothetical situations. To exemplify, if there is any power cut-off the PID parameters may get to their default state; to zero, so the user again needs to tune the motor and this may consume 10-15 seconds of the time to recalibrate the process. Summing up there is a need to recalibrate the whole system after every time turning on the power. If we consider a large time delay wherein the controller is performing other high priority tasks, it could affect the sampling time of the PID logic and will eventually result in lagging and less aggressive PID output [12].

As the testing was on a small scale and a prototyped structure we needed to retune the motor after every fluctuation in the weight to make the result more aggressive and adaptive with the range of current error.

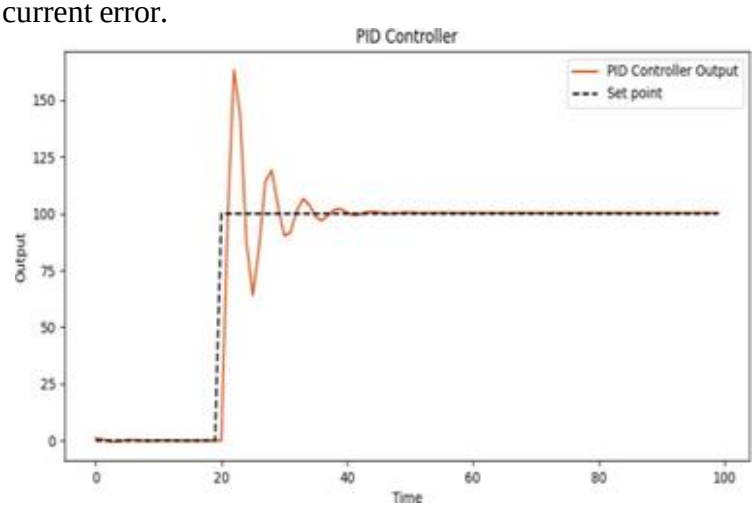

Figure 6: PID Output of the Proposed System

D. CONCLUSION

Figure 6 brings out the result of our system. As we tested our system on a heavy load there was a significant amount of peak errors and oscillations which took around 10-15 seconds to stabilise based on the load. This graph is plotted from the auto- tuned PID coefficients and by implementing adaptive tunings based on the error; if the error is greater than 50 then set aggressive K_p, K_i and K_d else use the auto- tuned ones.

We intended to design a system for autonomous mobile robots with configurable and compatibility features, all a user needs are the required hardware stated above, a python GUI and the source code. These requirements are supported by almost all the motor drivers, controllers, and encoders. The need for expensive proprietary hardware and software is also eliminated by using this system. The use of Ziegler-Nichols is what makes the implementation and controlling easy and also it provides a strong motor locking at the desired position no matter the load and momentum of the robot is. We also tried to replicate the logic of position control of DC motors proposed by the Maxon group.

To a larger extent, the use of this model can be integrated with all kinds of autonomous mobile robots. If a compact system is to be designed using this process then the implementation of inverse kinematics can be unified in a single controller which controls only the drive system of the robot. As we used ESP32 which has Bluetooth compatibility the use of a PS3 controller (Sony Dual shock 3 preferred) can increase the control and efficiency of the model significantly. Further different peripheral sensors like MPU-6050/9250 (Inertial Measurement Unit) can be used to make the feedback of this system more robust and will make sure no grey areas are left[3].

To overcome the limitation of this process the use of external/internal flash/EEPROM can be made so that the PID co-efficient is stored in the flash after every tune and the user need not retune the motor/load after every time turning on the power. All in all, there is always some room for advancements to be made in future.

REFERENCES

1. Mitra, M. (2018). Advancements in Control Systems. *International Robotics & Automation Journal*, *4*, 390-391. DOI: 10.15406/iratj.2018.04.00154
2. Goodwin, S. G. (2000). *Control System Design (p.41-61)*. Pearson.
3. Ashokbhai, A., & Vijay, S. (2014). *Performance Analysis of PID Controller and Its significance for Closed Loop System*. 1843-1847.
4. Espressif ESP32 Datasheet V3.8, October 2021 (p.35, p45-46).
5. CUI Devices, AMT10-V Modular Incremental Encoder datasheet (p. 4), April 2021
6. Cytron Technologies, SmartDrive Duo 30 MDDS30 User Manual (p. 19), April 2017
7. Maxongroup, Juergen Wagenbach. *Position Control of DC Motors*, November 2014 Fritzing Circuit Designer Software
8. B. Beauregard, T. Elenbaas. (2017). *Arduino-PID-Library (Version 1.2.1)* [Source Code].
9. Jack, Markus. (2022). *Arduino-PID- autotuner* [Source Code].
10. RFC: 793 (1981). IETF.
11. J. Astrom, T. Hagglund. (1995). *PID Controllers: Theory, design and tunning* (pp.134-199)
12. Sarvesh, Yash. (2022). *system-for-PID- control-and-auto-tune* [GitHub Source Code].
13. Sung, Su Whan and Lee, In-Beum, ACS Publications, "*Limitations and Countermeasures of PID Controllers*", August 1996.

[3]A unit that consists of gyroscope, accelerometer and angular data.