Experimental Study of Nonlinear PID Controllers in an Air Levitation System

J. Chacón [*] **H. Vargas** [**] **S. Dormido** [*] **J. Sánchez** [*]

[*] *Universidad Nacional de Educación a Distancia, Madrid 28040, Spain (e-mail: jchacon@bec.uned.es).*
[**] *Pontificia Universidad Católica de Valparaíso, Chile.*

Abstract: This paper presents an experimental study of different non-linear PI controllers using an academic platform based on an Air Levitation System. The comparison is based on common performance indexes, computed over the data extracted from the experiments. The results verify that, in addition to the PID controller in its classic form, several non-linear modifications can be considered to cope with different control needs, such as optimization on the number of control actions, or go beyond the restrictions that linearity impose on the performance that can be obtained by a PID controller.

Keywords: PID, Control Engineering, Teaching, Experimental Study, Non-linear

1. INTRODUCTION

The most known controller is, no doubt, the PID. One of the reasons of this popularity is that its principles are conceptually simple to understand: it is based on reacting to the control error, its past history and its predicted behavior. Of course, this simple idea allows for many variations that can cope with a wide range of systems and situations. Every day, many control and instruments engineers and operators have to use PID controllers (Åström and Hägglund, 2005). Ideally, a controller not only must comply its principal objective, obviously to meet the control specifications, but also it is important that the people responsible of assess the performance of the controller have a basic understanding and the ability to setup and give correct maintenance of the control loop.

Appart from being a simple controller in comparison to other most sophisticated structures, like GPC or IMC, there are some important practical details of implementation, such as the reset windup, noise filtering or bumpless transfer between manual and automatic mode, that can dramatically impact the performance of the control loop if neglected or not correctly cope with.

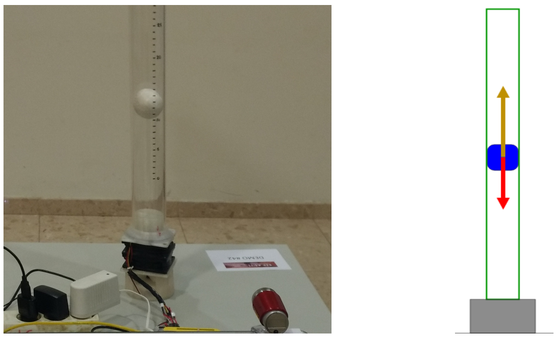
In spite of the pervasivity of PID controllers, and their ability to cope satisfactorily with many kind of processes, the PID controllers are linear systems and, as such, its linearity impose restrictions on their achievable performance. To expand the applicability of the PID control, many authors have proposed modifications that introduce different types of non-linearities (Årzén, 1999). In fact, the solution of the aforementioned practical problems, such as the antiwindup mechanism, introduce a non linear behavior. Many examples of non linear PID control exist in literature (Årzén, 1999; Lehmann and Johansson, 2012; Vasyutinskyy and Kabitzsch, 2006; Sánchez et al., 2011).

As main purpose of this paper, four different control strategies are explored in this paper, discussing how they work, providing an implementation and comparing their performance in an academic plant.

The experimental platform is based on an Air Levitation System (Chacón et al., 2017). The control objective of this system is to lift an object without mechanical support and maintain it at a desired height level. One of the interesting features of this academic platform is the flexibility to implement different control strategies, which is invaluable from a pedagogic perspective. In the platform, there is a dedicated Arduino board running the controller, and easily extensible to develop and test new control laws, without worrying about other implementation details, like the datalogging, communication protocols, etc.

In this paper, several control strategies are implemented, tested and compared in an academic plant. The development platform allows for rapid prototyping and evaluation of control laws, which can be implemented in Processing, a well-known programming language created for Arduino boards. The comparison is based on common performance indexes such as IAE or ITAE. The aim of the study is twofold: on the one hand, to evaluate in a real plant several implementations of the different PID controllers allows to have a better understanding of their capabilities, advantages and disadvantages. On the other hand, to demonstrate the value of the Air Levitation System as a pedagogical instrument to teach control engineering concepts, with a real-world approach which serves as a bridge between the control theory and the practical that appears in practice, such as erratic sensors behaviour, actuator wearout, communication delays or losses, etc.

The structure of the paper is as follows. Section 2 describes the Air Levitation System which is subject of study. Section 3 discusses the control strategies which are being compared, and provides some comments on the practical implementation of the controllers. Section 4 presents and analyzes the experimental results. Finally, Section 5 gives the conclusion and future lines of work.



(a) (b)

Fig. 1. (a) The Air Levitation System, and (b) diagram of forces.

2. THE AIR LEVITATION SYSTEM PLATFORM

The system consist of a tube with an object inside (Figure 1(a)), which must be lifted without mechanical support to a desired level (Figure 1(b). To this purpose, there is a fan forcing an air flow inside the tube to control the object movement, and an infrared sensor to measure the position. In the following lines the experimental system is described. A more detailed description can be consulted in (Chacón et al., 2017).

2.1 Model

The mathematical model of the system is studied in other previous works Timmerman and van der Weelea (1999); Escaño et al. (2006); Jernigan et al. (2009). A simplified mathematical model of the system can be obtained from a balance of forces: considering the upwards effect of the air flow, and the downwards effect of gravity (the only forces acting on the levitating object) the dynamic equation is (Chacón et al. (2017)):

$$\ddot{z} = g \cdot ((\frac{v_w - \dot{z}}{v_{eq}})^2 - 1). \tag{1}$$

where z is the vertical posisiton of the object, v_w is the wind speed, and v_{eq} is the wind speed at the equilibrium point. Given an operating point the system dynamics (not considering the fan) can be linearized and the resulting transfer function is a first order system plus an integrator: $\Delta z(s)/\Delta v(s) = a/(s(s+a))$, where $a = 2g/v_{eq}$, and (Δz, Δv) are, respectively, the increment of the position and wind speed, near the equilibrium point.

2.2 Identification

Considering that the fan can be modeled as a first order process, the transfer function between the input voltage and the wind speed is represented as: $v(s)/u(s) = k_v/(\tau s + 1)$, where v is the wind speed, u is the input voltage, k_v is the gain that relates the input voltage to the wind speed at steady state, and τ is the fan time constant that models the impossibility of the fan to change the speed instantaneously. Assuming the system is well described by the linearized model, the process transfer function is:

$$z(s) = \frac{1}{s}\frac{ak_v}{(s+a)(\tau s+1)}. \tag{2}$$

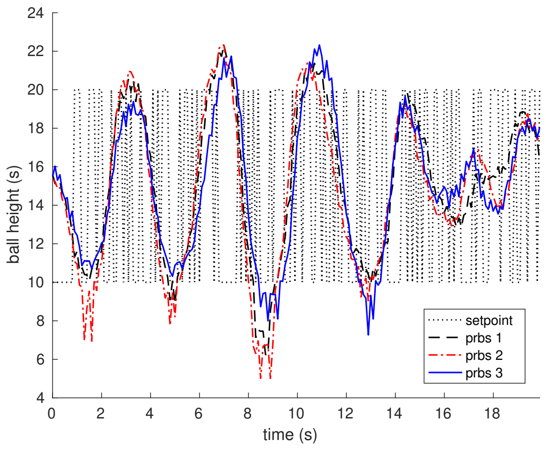


Fig. 2. An excerpt of the response to a PRBS input of amplitude 10cm applied to the setpoint.

Since the model has a pure integrator, the system is unstable in open loop, so a step input is likely to provoke the ball to either fall to the ground or to reach the upper end of the tube. To extract experimental data for identification, after some practical problems with the preliminary experiments, it was decided to identify the closed loop system. The procedure is as follows:

- The operating point is set to $h_0 = 15cm$.
- The system is identified in closed loop, using a proportional controller with unit gain ($u = r - h$).
- Setting a period of $T_s = 100ms$, the system is excited with a signal and 1024 samples are registered. Two kind of signals are applied:
  (1) Step (10cm).
  (2) *Pseudo-random bynary signal* (PRBS, $\pm$5cm).

With a proportional controller, the closed loop has a settling time of around 8 seconds, and without steady error due to the pole at the origin. These experiments were carried out as a preliminary approach to obtain an insight on the system parameters such as gain, time constant, etc. in order to design an experience that optimizes the extraction of information about the system. The amplitude (5 cm) of the PRBS signal is a tradeoff between signal/noise ratio and linearity. It must be noted that outside the range (10 cm, 30 cm), the ball approximates the ends of the tube, and both the system dynamics and the sensor presents nonlinearities, which affects the quality of the identification data. Figure 2 shows and excerpt of the PRBS signal and the corresponding system response. The experimental data was stored into the single-board computer connected to the plant. Then it was retrieved and preprocessed to generate Matlab formatted data, and Matlab with the System Identification Toolbox helped with the input signal generation and the identification of the model. These experiments provided enough information to obtain a good enough experimental model, and the parameters of the model based on physical principles were tuned to fit the experimental data. In spite of that, the model which obtained best fit to estimation data was an auto-regressive with external input (ARX), with polinomial orders of $n_a = 4$, $n_b = 4$ and $n_k = 3$:

$$A(z)y(t) = B(z)u(t) + e(t), \tag{3}$$

where:

$$A(z) = 1 - 0.714z^{-1} - 0.7218z^{-2} + 0.07565z^{-3} + 0.3468z^{-4},$$

$$B(z) = 2.053z^{-3} + 1.838z^{-4} + 1.379z^{-5} + 1.292z^{-6}.$$

Though the model was identified at an operating point $h_0 = 15$, it was also validated with experimental data for $h_0 = 10$ and $h_0 = 20$. The transfer function corresponding to the discrete process and to the disturbance can be written as:

$$G(z) = \frac{2.053z^{-3} + 1.838z^{-2} + 1.379z^{-1} + 1.292}{z^{-4} - 0.714z^{-3} - 0.7218z^{-2} + 0.07565z^{-1} + 0.3468}, \quad (4)$$

$$H(z) = \frac{1}{z^{-4} - 0.714z^{-3} - 0.7218z^{-2} + 0.07565z^{-1} + 0.3468}, \quad (5)$$

where $e(t)$ is a white noise with variance $\sigma = 0.2798$. The identified model, focused on prediction, obtained an 85% of fit to estimation data, which is reasonably considering the tradeoff decisions in the design of the system.

2.3 Implementing the controller

The controller is deployed into an Arduino Nano. The software platform provides a template to implement generic controllers, so anyone who is familiarized with Processing can easily extend the system to experiment with new control laws. The communication and other implementation details are encapsulated into the provided code. The class Controller can be extended to implement the control law. In particular, the method update is invoked with a fixed period $T = 100ms$. This period is imposed by a physical limitation of the sensor, in order to provide stable measurements. The period of the controller can be chosen to a different value, though the measurement will be updated only every T seconds.

3. CONTROL STRATEGIES

Four control strategies have been implemented and tested, namely:

- PI: A classical PI controller.
- PI-CI: A PI controller with a Clegg's integrator (Baños and Barreiro, 2012).
- PI2D: An event-based PI controller with feedforward (Sánchez et al., 2011).
- Robust Adaptive Hybrid PI (Scola et al., 2017).

The approach chosen in this paper to compare the performance of the different controllers is based on integral criteria. The following ones are commonly used to express the performance of a control system (Åström and Hägglund (2005)):

- $IE = \int_0^t e(t)dt$
- $IAE = \int_0^t |e(t)|dt$
- $ITAE = \int_0^t t|e(t)|dt$
- $ISE = \int_0^t e^2(t)dt$
- $QE = \int_0^t (e^2(t) + \rho u^2(t))dt$

To compare the performance of the different strategies, the closed loop is excited with step changes in the reference and external disturbances. The experiment are repeated several times for each implemented strategy and then the performance indexes are computed and averaged. To introduce an external disturbance, the plant provides a servo-mechanism that can modify the input air flow in a repeatable manner.

3.1 PI

The first one is a PI controller in parallel form $C(s) = (k_p + \frac{k_i}{s})e(s)$. This implementation is provided as a reference for the comparison. To cope with practical problems such as the saturation of the actuator, the controller implementation incorporate an antiwindup mechanism, based on a conditional integration. The code is shown in Figure 3(a), Listing 1.

3.2 PI-CI

The second implementation is a PI controller with a Clegg's integrator (PI-CI). The Clegg's integrator is a special kind of integrator which is based on resetting the state to zero whenever the input crosses by zero. It was shown by Clegg that the integrator with reset introduces a phase lag of $-38.1°$, as oppossed to the $-90°$ of the linear integrator. The PI-CI consist of a PI controller with a Clegg's integrator connected in parallel with the linear integrator, and a reset coefficient $\rho \in [0, 1)$ to weight the influence of each one. That means that for $\rho = 0$ the controller is equivalent to a PI controller, and $\rho = 1$ corresponds to a PI with a pure Clegg's integrator. This latter case is not recommended though: the effect of the integral term is lost and the resultant controller is not able to reject disturbances in steady state. With regard to the implementation in Arduino, the code is shown in Figure 3(b), Listing 2.

3.3 PI2D

The third implementation is an event-based PI with feedforward Sánchez et al. (2011). This controller consist of two parts: the feedforward block respond to changes in the setpoint, and generates a two-state control action that moves the output to the desired value, in absence of disturbances and assuming a perfect model of the system. The feedback block consist of a PI controller which has been modified to use a send-on-delta sampling strategy both in the proportional and integrated terms. To compute the feedforward action, in Sánchez et al. (2011) the process is modeled as a First Order plus Time Delay (FOTD), yielding the two values of the control signal. However, since the model considered here contains a pure integrator, the second value is fixed to zero, and the first one can be modified as an extra degree of freedom. The feedback block is characterized by two params, δ_P and δ_I, which are the proportional and integral event thresholds, respectively. A new event is triggered every time the difference between the current value of the signal ($e(t)$ or $I_e(t)$) and the value at the last event is greater than the threshold, i.e. $|e(t) - e(t_k)| > \delta_P$ for the proportional term and $|I_e(t) - I_e(t_k)| > \delta_I$ for the integral term.

3.4 ROBUST ADAPTIVE HYBRID PI

The last implemented controller is described in Scola et al. (2017). It is a PI controller which resets its integrator's state if under the temporal regularization and the state vector of the closed loop belong to the jump set, D. Defining $e = r - y$, $\xi = x_I - x_{eq}$, the controller reset condition is $2e\xi + \xi^2\epsilon < 0, \tau > \rho$, and the state is reset to $e^+ = e$, $x_I^+ = x_I - \alpha\epsilon$, $\xi^+ = 0, \tau+ = 0$.

Listing (1) PID controller implementation in Arduino

```
float PID::update(float y) {
  float kp = params[0], ki = params[1], kd = params[2];
  float I = state[0], e_prev = state[1];
  float e = sp - y;
  float P = kp*e;
  float I_prev = I;

  I += (ki*e) * (period/1000.0);
  float v = u0 + P + I;
  // Conditional integration
  float u = sat(v, 0.8, 1.0);
  if((u - v) * e < 0.0) {
    I = I_prev;
  }
  state[0] = I;
  state[1] = e;
  return u;
}
```

(a)

Listing (2) PICI controller implementation in Arduino

```
float PICI::update(float y) {
  float kp = params[0], ki = params[1], pr = params[2];
  float I = state[0], Ic = state[1], e_prev = state[2];
  float e = sp - y;
  float P = kp*e;
  I += (ki*e) * (period/1000.0); // I
  Ic += (ki*e) * (period/1000.0); // Ic

  float v = u0 + P + (1-pr)*I + pr*Ic;
  // Conditional integration
  float u = sat(v, 0.8, 1.0);
  if((u - v) * e >= 0.0) {
    state[0] = I;
    state[1] = Ic;
  }
  // Reset
  if(e_prev * e <= 0) {
    state[1] = 0;
  }
  state[2] = e;
  return u;
}
```

(b)

Listing (3) PI2D controller implementation in Arduino.

```
float PID::update(float y) {
  // read the params
  float kp = params[0], ki = params[1], delta = params[2], delta_I = params[3],
      uff = params[4];

  // read the state of the previous iteration
  float I = state[0], e_last = state[1], I_last = state[2], ysp = state[3], uff_last = state[4];
  float e = sp - y;

  // feedforward: Apply u_ff whenever a new setpoint arrives and remove it after entering the deadband
  if(sp != ysp) {
    uff_last = (sp > ysp) ? u0 + uff : u0 - uff;
    ysp = sp;
  } else {
    if(abs(e) < delta) {
      uff_last = u0;
    }
  }

  // P Event
  if(abs(e - e_last) > delta) {
    e_last += (e > e_last) ? delta : -delta;
  }

  // I Event
  I += e * (period/1000.0f);
  if(abs(I - I_last) > delta_I) {
    I_last += (I > I_last) ? delta_I : -delta_I;
  }
  float v = uff_last + kp*e_last + ki*I_last;

  // Conditional integration
  float u = sat(v, 0.86, 1.0);
  if((u - v) * e >= 0.0) {
    state[0] = I;
  }

  // store the state for the next iteration
  state[0] = I;
  state[1] = e_last;
  state[2] = I_last;
  state[3] = ysp;
  state[4] = uff;

  return u;
}
```

(c)

Fig. 3. Different controllers implemented in Arduino: (a) PI, (b) PI-CI, and (c) PI2D

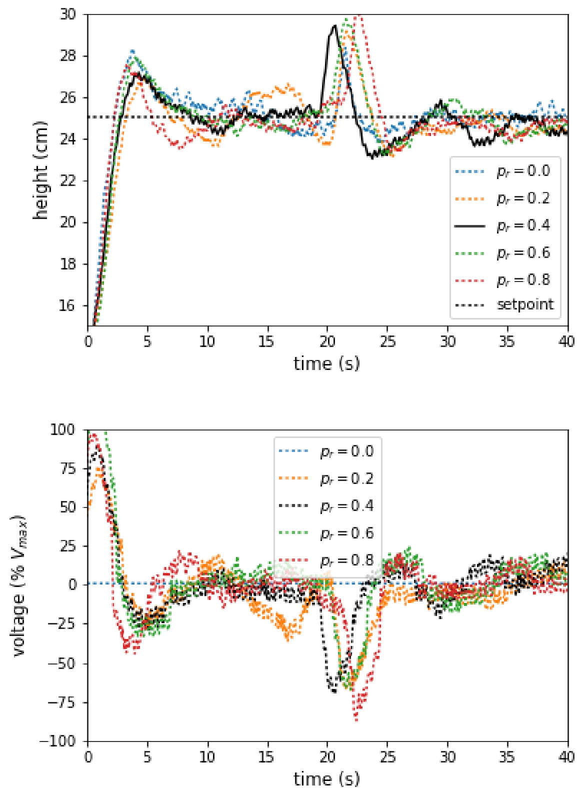
4. RESULTS

As mentioned in Section 3, the PI controller acts as reference to compare the performance of the other implementations. The controller was tuned in the model to have an overshoot of 10% and a settling time less than 5s, yielding values of the parameters $k_p = 0.006$ and $k_i = 0.002$. In practice, due to differences between the plant and the model, the gains had to be slightly decreased to $k_p = 0.005$ and $k_i = 0.001$, in order to meet the specifications. Figure 5 shows the step response for different values of k_p and k_i.

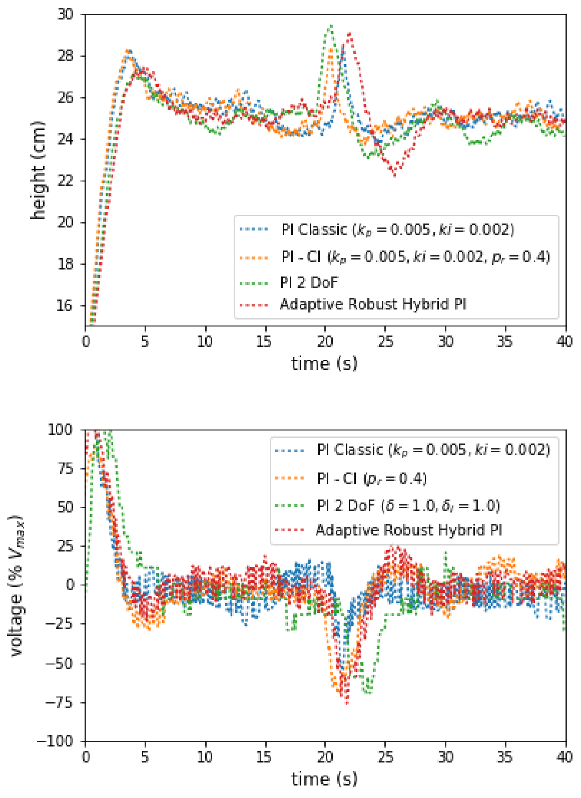
Figure 4(a) shows the step response of the PI-CI controller for different values of the reset param, $p_r \in (0.2, 0.4, 0.6, 0.8)$, compared to the response of the PI controller. As it can be seen, the reset action in the PI-CI allows to reduce the overshoot and thus a faster response to the setpoint change: the greater p_r is, the lower the overshoot obtained. For greater values, however, the ability to eliminate the steady state error is worst. This is apparent in Figure 6, where different performance indexes are computed for the experiments.

The step response of the PI2D controller is shown in Figure 4(d) for different values of the event thresholds δ and δ_I. compared to the response of the PI controller. As it can be seen, the event-based action in the PI2D allows to reduce the control effort (measured in number of control actions) with respect to the PI. For greater values, however, the control performance is affected, and at the same time the error in steady state is not zero, because the controller is not aware unless the value of the error goes beyond the threshold. This behaviour is shown quantitatively in Figure 6.

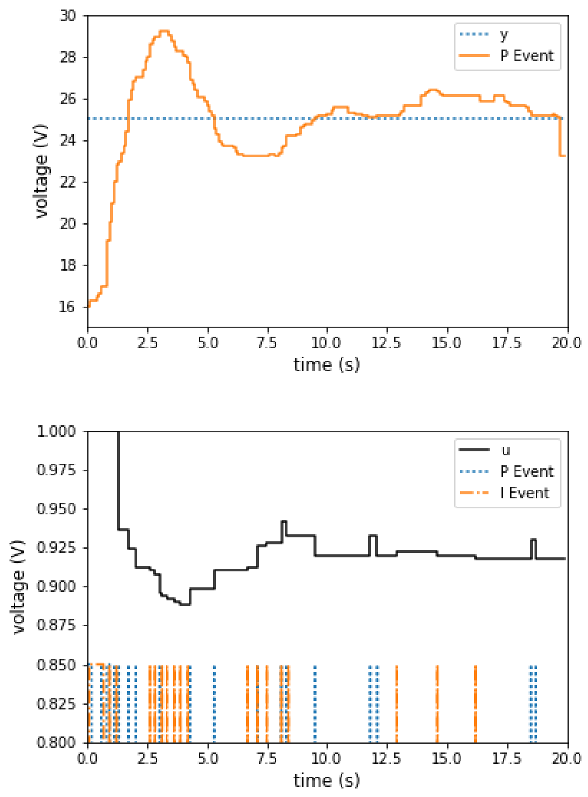
Also, with respect to the PI2D controller, Figure 4(c) shows the step response of an implementation with $k_p = 0.006$, $k_i = 0.002$, and thresholds $\delta_P = 1.0$, $\delta_I = 1.0$. The plot of the control signal shows the instants of time where the events are triggered, differentiated by their type: integral and proportional. In Figure 4(d), the effect of the thresholds δ_P, δ_I in the steady state error can be better understood.



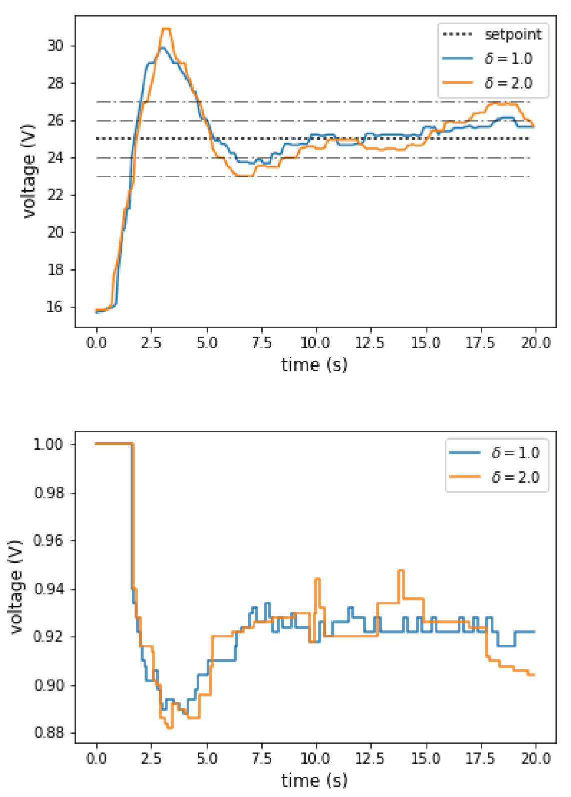
(a)



(b)



(c)



(d)

Fig. 4. (a) Step response of the PI-CI controller with $k_p = 0.005$ and $k_i = 0.002$, varying the reset action p_r between 0 and 1, (b) step response of the four controllers with $k_p = 0.005$ and $k_i = 0.002$, (c) Proportional and Integral Events, and (d) effect of δ_P and δ_I in the response.

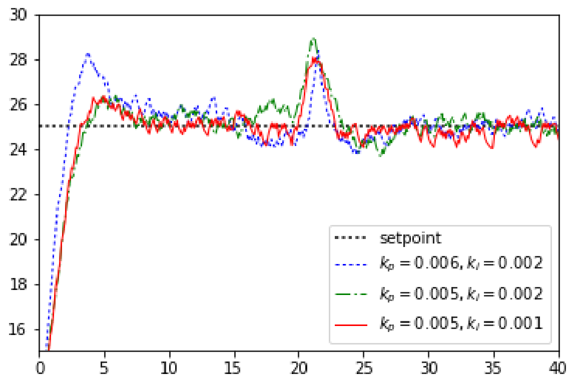


Fig. 5. Step response of the PI controller with an external disturbance, varying k_p and k_i, varying the reset action p_r between 0 and 1.

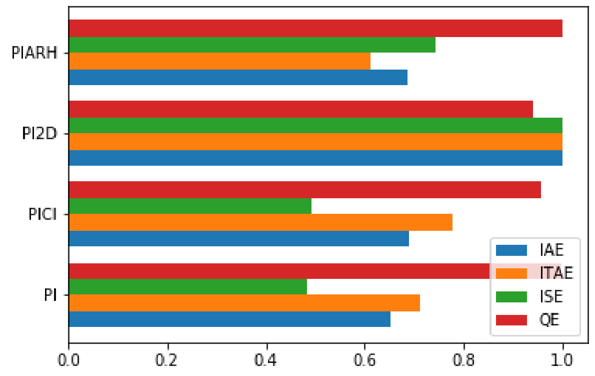


Fig. 6. Performance indexes for different configurations: Classic PI ($k_p = 0.005, k_i = 0.002$), PI-CI ($p_r = 0.4$), PI-2D ($\delta = 1.0, \delta_I = 1.0$), and ARH-PI ($\alpha = 0.8$).

The value of δ_P is straightforward: the controller ignores all that happens inside the error band $\pm\delta$, so the greater the value, the less responsive is the proportional term. Note that low values of δ are not very convenient, mostly due to two issues: sensor noise will provoke spurious triggers, and because of Zeno's effect. Since the events can be detected with a limited precision, with a lower value of the threshold the signal may cross several bands between two updates, so the controller will not work properly. Considering the characteristics of the system, values of $\delta = 1.0$ and $\delta = 2.0$ obtained good performance in the experiments. The integral event threshold δ_I has a similar behaviour, but acting on the integrated error. Due to the effect of the integrator, the problem of spurious triggering is not as important as in the proportional term, but a minimum bound for δ_I must be also considered in order to guarantee a correct detection of event times. For $\delta = 1.0$, the number of events generated was of the same order of the proportional events ($n_P = 24$, $n_I = 29$).

The comparison of all the controllers is shown in Figure 6, where the indexes mentioned in Section 3 have been computed.

5. CONCLUSION

An experimental study of four different PI controllers is discussed in this paper. Using an academic platform based on an Air Levitation System, the controller were implemented and tested under similar circumstances. The experiments were repeated several times for each tested case, yielding several datasets, in order to obtain more robust results. The controller were compared by means of several performance indexes, computed over the data extracted from the experiments, and finally the graphs and plot were generated. The results verify that, in addition to the PID controller in its classic form, several non-linear modifications can be considered to cope with different control needs, such as optimization on the number of control actions, or go beyond the restrictions that linearity impose on the performance that can be achieved by a PID.

ACKNOWLEDGEMENTS

This work has been funded by the National Plan Project DPI2017-84259-C2-2-R of the Spanish Ministry of Economy, Industry and Competitivity.

REFERENCES

Årzén, K. (1999). A Simple Event-Based PID Controller. In *14th IFAC World Congress*. Beijing, P.R. China.
Baños, A. and Barreiro, A. (2012). *Reset Control Systems*. Springer-Verlag London.
Chacón, J., Saenz, J., Torre, L.d.l., Diaz, J.M., and Esquembre, F. (2017). Design of a low-cost air levitation system for teaching control engineering. *Sensors*, 17(10).
Escaño, J.M., Ortega, M.G., and Rubio, F.R. (2006). Position control of a pneumatic levitation system. In *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation*.
Jernigan, S.R., Fahmy, Y., and Buckner, G.D. (2009). Implementing a remote laboratory experience into a joint engineering degree program: Aerodynamic levitation of a beach ball. *IEEE Transaction on Education*, 52, 205–213.
Lehmann, D. and Johansson, K. (2012). Event-Triggered PI Control Subject to Actuator Saturation. In *IFAC Conference on Advances in PID Control*.
Åström, K.J. and Hägglund, T. (2005). *Advanced PID Control*. ISA-The Instrumentation, Systems, and Automation Society.
Sánchez, J., Visioli, A., and Dormido, S. (2011). A two-degree-of-freedom pi controller based on events. *Journal of Process Control*, 21, 639–651.
Scola, I.R., Quadrios, M.M., and Leite, V.J.S. (2017). Robust Hybrid PI Controller with a Simple Adaptation in the integrator reset state. In *IFAC PapersOnLine*, volume 50.
Timmerman, P. and van der Weelea, J.P. (1999). On the rise and fall of a ball with linear or quadratic drag. *American Journal of Physics*, 67, 538–546.
Vasyutinskyy, V. and Kabitzsch, K. (2006). Implementation of PID Controller with Send-on-Delta Sampling. In *ICC'2006, International Conference on Control*. Glasgow, Scotland.