U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

# Algoritmo PID para controlo de uma Planta Industrial implementada numa sub-rotina em LabVIEW

Mestrado em Engenharia Biomédica

Unidade Curricular de Sensores, Atuadores e Controladores

Miguel Rocha

Rúben Silva

Susana Teixeira

Ano Letivo 2021/2022

# Índice

## 1. Objetivo

O presente trabalho teve como objetivo o desenvolvimento, em Arduino, de um algoritmo PID[1] de controlo de uma planta industrial numa sub-rotina em LabVIEW. As duas aplicações comunicam por RS-232 os diversos parâmetros de controlo, estudados na componente prática da Unidade Curricular de Sensores, Atuadores e Controladores.

## 2. Introdução Teórica

Os controladores PID, devido à flexibilidade e confiabilidade que proporcionam, são encontrados numa ampla gama de aplicações para o controlo de processos industriais. Aproximadamente 95% das operações em malha fechada do setor de automação industrial utilizam controladores PID [1].

O termo PID significa *proporcional-integral-derivativo* e é um tipo de dispositivo usado para controlar diferentes variáveis de processo como pressão, fluxo, temperatura e velocidade em aplicações industriais. Nestes controladores, um dispositivo de realimentação da malha de controlo é usado para regular todas as variáveis do processo, fornecendo à saída os níveis desejados. Este sistema avalia a variável de feedback usando um *Set Point* desejado para gerar um sinal de erro. Com base nisto, altera a saída do sistema. Este procedimento continuará até que o erro seja próximo de nulo, ou seja, até o valor da variável de processo se tornar equivalente ao *Set Point* [1]. Este controlador é mais estável e preciso em comparação com o controlador do tipo ON/OFF, onde o output não é estável e oscila frequentemente entre dois valores limites. Os controladores PID atingem um output estável com erro quase nulo entre a variável de processo e um ponto fixo (*Set Point*) a partir de três comportamentos de controlo que atuam em concomitância: controlo proporcional, integral e derivativo.

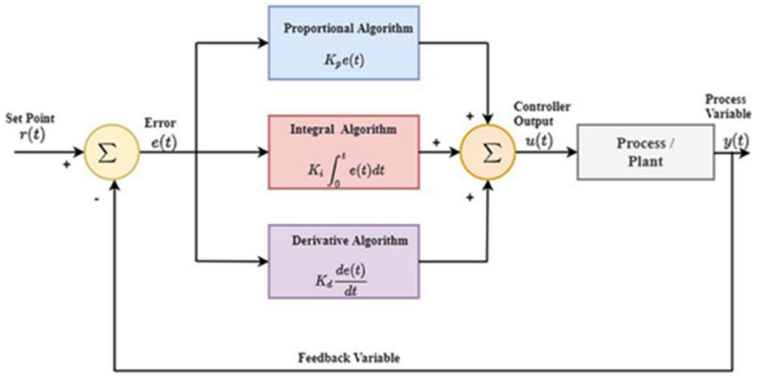


**Figura 1 –** Esquema representativo do funcionamento dos controladores PID.

O controlador proporcional ($P$) fornece uma saída proporcional ao erro naquela instância $e(t)$ [1]. Este compara o ponto desejado ou definido com o valor real ou valor do processo de feedback. O erro resultante é multiplicado por uma constante proporcional para obter a saída. Se o valor do erro for zero, a saída do controlador será zero. Este controlador requer polarização ou reinicialização manual quando singularmente utilizado. Isto ocorre porque nunca atinge a condição de estado estacionário. Fornece operação estável, mas mantém o erro de estado estacionário. A velocidade da resposta aumenta quando a constante proporcional *Kp* (constante de ganho proporcional) aumenta.

Devido à limitação do controlador P, onde existe sempre um deslocamento entre a variável de processo e o *Set Point*, é necessário o controlador *I*, que fornece a ação necessária para eliminar o erro de estado estacionário [1]. Este integra o erro durante um determinado período de tempo, até que o valor do erro atinja um valor nulo. O controlo integral diminui a sua saída quando ocorre um erro negativo. Este comportamento limita a velocidade de resposta e afeta a estabilidade do sistema. A velocidade da resposta é aumentada pelo aumento do ganho integral, $Ki$ (constante de ganho integral).

O controlador integral não tem a capacidade de prever o comportamento futuro do erro. Assim, ele reage normalmente quando o *Set point* é alterado. O controlador D supera esse problema e é capaz de antecipar o comportamento futuro

do erro [1]. A sua saída depende da taxa de variação do erro em relação ao tempo, multiplicada pela constante de ganho derivativa ($K_d$). Melhora a estabilidade do sistema compensando o atraso de fase, causado pelo controlador integral. Aumentar o ganho derivativo resulta no aumento da velocidade de resposta.

Finalmente, observa-se que, combinando esses três controladores, pode obter-se uma resposta desejada para um determinado sistema. Desta forma, o algoritmo PID pode ser definido pela seguinte expressão:

$$u(t) = K_P e(t) + K_i \int_0^t e(\tau)dr + K_d \frac{de(t)}{dt} \quad (1)$$

Tal que o erro é dado por $(SP - PV)(t)$.

Várias abordagens de controlo baseadas em PID são propostas para as aplicações biomédicas, como a infusão/regulação da pressão arterial, relaxamento muscular em pacientes cirúrgicos, controlo do ângulo articular em músculo estimulado artificialmente, transplante de fígado, controlo da glicemia, controlo da anestesia por propofol, entre outros [2].

## 3. Análise do Enunciado Proposto

No âmbito da componente prática da unidade curricular de Sensores, Atuadores e Controladores, foi proposto que se desenvolvesse um algoritmo PID para controlar uma Planta Industrial implementada num sistema de uma dada sub-rotina do LabVIEW. A comunicação entre as aplicações foi realizada por RS-232(USB).

O *LabVIEW* é um ambiente de programação gráfica, comummente utilizado em Engenharia para o desenvolvimento de sistemas automatizados de pesquisa, validação, controlo e testes de produção de sistemas [3].

Neste trabalho o utilizador deveria ser capaz de definir como *inputs* no LabVIEW, uma variável *Set Point* (ST) entre 0 a 100, uma disturbância (DIS) entre 10 a 20 e os valores dos parâmetros de controlo P, I e D.

O *LabVIEW,* por sua vez, deve retornar os valores destes *inputs* manuais ao Arduino, onde é calculado o valor da variável *Control Action* (CA) e enviado, a cada 500ms, novamente para o *LabVIEW*.

As variáveis de entrada da sub-rotina *System* são, então, a *Control Action* (CA) e o valor da disturbância (DIS). O parâmetro de saída, calculado pelo sistema a controlar, é denominado *Process Value* (PV).


**Figura 2 –** Representação do bloco da sub-rotina *System,* previamente fornecida, no LabVIEW.

## 4. Desenvolvimento do Algoritmo

Com o objetivo de desenvolver o algoritmo PID de controlo da sub-rotina fornecida, o primeiro passo do projeto consistiu em estabelecer a comunicação Arduino-LabVIEW.

### 4.1. Comunicação Arduino – LabVIEW

A comunicação entre as aplicações foi realizada por uma entrada USB, através de uma porta serial, permitindo enviar e receber dados com o comando *write* e *read*.

Como referido, é necessário que o Arduino receba, através da introdução no LabVIEW, os valores dos parâmetros ST, P, I e D. Para isso, construiu-se uma interface para o controlo do sistema que permitisse a introdução manual desses valores, como ilustrado na figura 3.

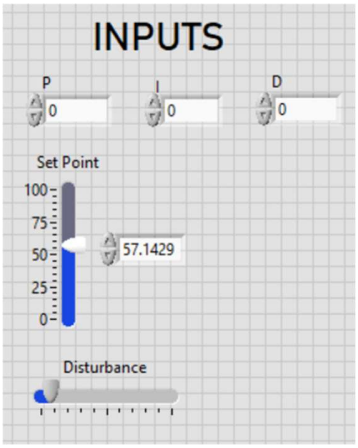


**Figura 3** – Interface construída em LabVIEW para introdução dos parâmetros de controlo necessários.

Com a pretensão de que o parâmetro introduzido, o set point (ST), que figura como *input* do sistema, fosse idealmente igual à saida da sub-rotina, o process value (PV), elaborou-se um algoritmo que possibilitasse a constante recessão e envio de dados entre as plataformas.

Como primeiro passo, elaborou-se uma estrutura "while loop" com um tempo de espera definido em 500 ms entre cada iteração, de modo a que todo o processo de comunicação ocorra "continuamente" com uma frequência de amostragem de 2 amostras/s, como pedido.

É de notar que os valores configurados no LabVIEW para os parâmetros P, I, D e SP são do tipo *double* pelo que foi necessária a sua conversão (Figura 5) para o tipo *string,* de forma a que seja possível o envio e leitura pelo Arduino.

Agruparam-se os parâmetros numa única *string* usando o modulo "*Concatenate Strings*" (Figura 4), com o delimitador ";" para facilitar, posteriormente, a extração dos valores das variáveis e fazer a sua separação no Arduino.

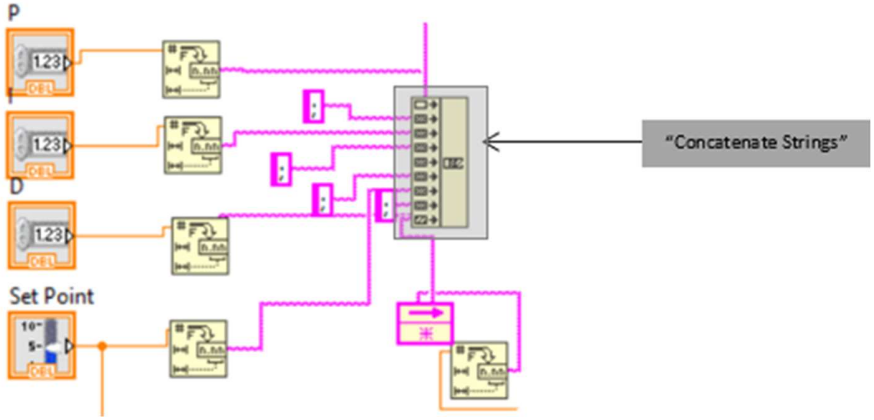


**Figura 4 –** Conversão das variáveis de *input* de *double* para *string* e concatenação – diagrama de blocos LabVIEW.

Através do comando "*write*" em LabVIEW a *string* foi enviada para o Arduino, seguida de um "tab". De notar que este "tab" é essencial para a estratégia adotada, como vamos ver mais à frente.

Nesta fase, o Arduino recebe os dados e é necessária a sua programação, de modo a manipulá-los da forma desejada para calcular o valor da variável CA. Assim:

- Configurou-se o Arduino para detetar a receção de um dado caracter através da porta serial COM através do comando: *if Serial.available() > 0*;
- O algoritmo lê, caracter a caracter, pelo comando *Serial.read()* e guarda-os numa variável do tipo *string* que é interrompida quando surge o "tab" anteriormente referido, significando o fim de um conjunto de valores introduzido.
  É de notar que esta variável tem uma forma semelhante a, por exemplo, ("0.50;0.80;0.64;50.00;45.00") correspondente a P, I, D, SP e PV, respetivamente.
- Pela função *substring* separou-se cada um dos valores, tendo como referência o delimitador ";" previamente estabelecido, e converteu-se novamente cada uma das variáveis para o tipo *float* para possibilitar a execução dos cálculos necessários.

Nesta etapa, os dados estão prontos para a obtenção do valor da variável *Control Action(CA)* no Arduino. O envio dos valores, novamente para o LabVIEW, é possível pelo comando *Serial.print()*, através da porta serial 9600. Estes são lidos pelo comando *"read"* e, posteriormente, convertidos no tipo *double.* São, ainda, retornados à interface desenvolvida, de forma a confirmar o bom funcionamento da comunicação entre as aplicações caso os valores de P, I e D sejam coincidentes com os valores instroduzidos pelo utilizador.

Como seria de esperar, para concretização do objetivo do projeto, o valor crucial é o CA, usado como parâmetro de entrada de *System* para obtenção da saída, PV, novamente enviada ao Arduino para novo cálculo de CA e assim sucessivamente e iterativamente com um período de 500ms.

Note-se que, na primeira iteração, por não existir um valor previamente calculado para o CA, não é possível obter um valor de PV. Com isto, de modo a contornar esse problema, definiu-se CA como um *random value* entre 0 e 1 para o *input* da sub-rotina.

### 4.2. Implementação do controlador PID no Arduino

Configurada a boa comunicação entre o Arduino e o LabVIEW, o próximo passo no desenvolvimento do projeto foi a implementação do controlador PID no Arduino.

Como abordado no capítulo introdutório, os controladores PID funcionam por três tipos de ganhos com funções distintas: ação proporcional ao erro (P), ação integral (I) e, ainda, ação derivativa (D).

Deste modo, é fundamental o cálculo do erro em cada instante do algoritmo (a cada 500ms). Esse erro é constantemente atualizado a cada iteração e corresponde à diferença entre o Set Point e o PV (saída do sistema).

Tendo em conta a equação 1:

- A ação proporcional é dada pela multiplicação do erro calculado com o valor de P introduzido pelo utilizador na interface do sistema de controlo.
- A ação integral corresponde à área do gráfico do erro até ao instante em que o sistema está a ser executado. Para este efeito criou-se uma variável ("*integral_error*") que guarda, sucessivamente, o valor obtido para os erros, ao longo das iterações ($integral\_erro = integral\_erro + erro \times 0.5$). É de notar que, como se trata de um sistema discreto com um período de amostragem de 500 ms, é necessário a multiplicação do valor do erro por 0,5. A integral do erro é finalmente multiplicada pelo parâmetro *I* decidido pelo utilizador.
- Para implementação da ação derivativa, recorre-se ao erro da iteração anterior para o cálculo derivativo. Para isso, criou-se uma nova variável, "*lasterror"* para guardar o valor do erro no fim de cada iteração. Com isto, em cada iteração, a derivada é dada pela substração entre o erro dessa mesma iteração e o erro anterior, dividido pelo período de amostragem: $[(lasterror - error)/0.5]$. Por sua vez, este resultado é armazenado numa variável à qual se deu o nome de *derivative_error*. Novamente, multiplicou-se a derivada do erro pelo parâmetro D definido pelo utilizador.

O valor de *CA* é, finalmente, obtido pela soma dos três controladores (Equação 1), ajustando a expressão matemática a um sistema discreto, considerando um período de amostragem de 500ms.

### 4.3. Determinação dos parâmetros ótimos do controlador

O algoritmo anteriormente descrito permite a correta comunicação entre o Arduino e LabVIEW, bem como a implementação de um controlador PID, através do Arduino. Nesta fase é relevante descobrir os parâmetros P, I e D que otimizam o controlador.

O comportamento ideal da resposta de um sistema a uma mudança de processo ou mudança do *Set Point* varia dependendo da aplicação. Dois requisitos básicos são a permanência do *PV* a algum ponto de ajuste (não havendo perturbação) e o rastreio do comando [4].

Neste último ponto, o tempo de acomodação ($t_s$) e o tempo de subida ($t_r$) foram considerados. Além disto, dependendo da aplicação, alguns dos processos não devem permitir um *overshoot*, $M_p$, da *process value* se, por exemplo, isso não for seguro. Estes critérios foram tidos em conta para a obtenção e avaliação dos parâmetros ótimos.

É importante referir ainda que, quando o sistema atinge a estabilidade, é considerada uma tolerância de erro em regime permanente de 1% relativo ao *Set Point* definido. Além disso considerou-se o tempo de subida ($t_r$) como o tempo em que o sistema demora a responder, desde o inicio do degrau até atingir o *SP*(mesmo que não atinja a estabilidade).

Existem vários métodos para ajustar os parâmetros P, I e D, pelo que foram aplicados os mais comuns.

### 4.3.1. Método Ziegler-Nichols

O método de Ziegler-Nichols consiste em definir o controlo $Ki$ e $Kd$ como 0 e aumentar o termo proporcional, $Kp$, até o sistema apresentar comportamento oscilatório de amplitude constante [5].

O termo proporcional que faz com que o sistema adquira este comportamento denomina-se ganho crítico ($K_{cr}$) e o período de oscilação é o período critico ($P_{cr}$).

Para este sistema, $K_{cr}$ tem um valor aproximado de 1,4 e $P_{cr}$ de 2,5s, quando adquire comportamento oscilatório, como se pode observar no gráfico da Figura 5.

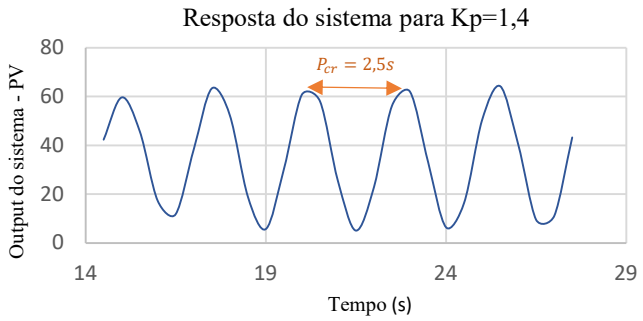


**Figura 5 –** Resposta oscilatória do sistema com amplitude constante.

Note-se que, a cada 500ms, os valores das variáveis são exportados do LabVIEW para um ficheiro .csv, permitindo a manipulação e a análise dos dados obtidos no Microsoft Excel.

Na tabela 1 estão representadas as expressões para o cálculo dos parâmetros P, I e D. Os resultados obtidos estão representados na tabela 2:

**Tabela 1** – Referência para cálculo dos parâmetros pelo método de Ziegler-Nichols [5].

| **Tipo de controlador** | $K_p$ | $T_i$ | $T_d$ |
|---|---|---|---|
| P | $0{,}5K_{cr}$ | $\infty$ | 0 |
| PI | $0{,}45K_{cr}$ | $\frac{1}{1{,}2}P_{cr}$ | 0 |
| PID | $0{,}6K_{cr}$ | $0{,}5P_{cr}$ | $0{,}125P_{cr}$ |

**Tabela 2** – Resultados obtidos para os parâmetros de controlo P, I e D através do método de Ziegler-Nichols.

| **Tipo de controlador** | $K_p$ | $K_i$ | $K_d$ |
|---|---|---|---|
| P | 0,70 | | |
| PI | 0,63 | 0,30 | |
| PID | 0,84 | 0,67 | 0,26 |

Com isto, foram realizados testes para os diversos parâmetros apresentados, como se pode observar na figura 6. Os ensaios foram efetuados tendo em conta a resposta do sistema a um degrau de 25 unidades (25-50).

Assim, para o controlador P, apesar de este apresentar uma reposta rápida ao degrau, o erro em regime permanente é muito elevado, pelo que se desconsiderou este controlador na procura da otimização do sistema.
Analisando o controlador PI, obteve-se um tempo de acomodação de cerca de 7,5s e um tempo de subida de 1.5s com um *overshoot* de aproximadamente 21,8%. Estes resultados apresentaram-se definitivamente melhores relativamente ao controlador P já que, para além da resposta ser relativamente rápida, o sistema estabilizou e o PV permaneceu no intervalo de valores toleráveis, com um erro relativo inferior a 1%. Este comportamento era espectável já que se introduziu um termo integral que possibilita o erro nulo e elimina o *offset* observado no controlador P.
Por fim, quanto ao controlador PID apesar de ter um tempo de subida muito baixo(<0.5s), o sistema demorou um tempo significativo a atingir a estabilidade com um tempo de acomodação de cerca de 51 s, o que não é de todo viável tendo em conta a performance atingida pelo controlador PI. Além disso, apresentou ainda a desvantagem de um *overshoot* de 91,6%, valor bastante considerável e em muitas aplicações, não tolerável.

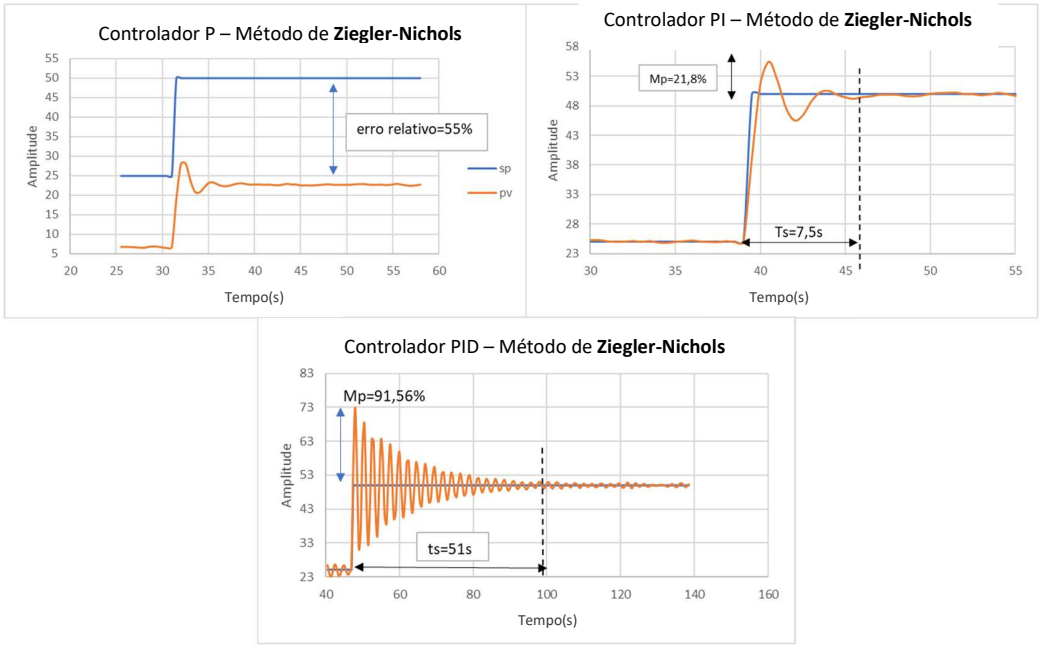


**Figura 6 –** Resposta do sistema a controladores obtidos pelo Método de Ziegler-Nichols.

### 4.3.2. Ajuste manual

O ajuste manual é considerado um dos métodos mais simples de ajuste dos parâmetros dos controladores, no entanto, por vezes, é necessário um profissional experiente para um resultado satisfatório, cumprindo com os requisitos da aplicação do controlador.

Uma estratégia para este ajuste é definir os ganhos integral ($Ki$) e derivativos ($Kd$) como 0 e aumentar progressivamente o ganho proporcional ($Kp$) até se observar oscilação do sistema[4]. Quando este comportamento é visualizável, o Kp é novamente ajustado para metade do Kp anterior e Ki é ligeiramente incrementado de modo a tentar melhorar a resposta do sistema e anular o valor do erro.

Com o objetivo de aplicar esta estratégia manual, aplicou-se, novamente, um degrau de 25 unidades ao sistema (25 a 50) e avaliou-se a resposta do sistema seguindo uma lógica semelhante da descrita anteriormente. Os resultados obtidos estão presentes na tabela 3.

**Tabela 3 –** Ajuste manual dos valores P, I e D, onde *NV* refere-se a um valor não visualizável, $t_s$ é o tempo de acomodação, $M_p$ o *overshoot* e $t_r$ o tempo de subida. De notar que quando o $M_p$ é NV e existe um $t_s$ significa que o sistema entrou em estabilidade sem nunca ultrapassar o SP com um erro relativo maior que 1%.

| $Kp$ | $Ki$ | $Kd$ | $Ts\ (s)$ | $Tr\ (s)$ | $Mp\ (\%)$ |
|---|---|---|---|---|---|
| 0,1 | 0,0 | 0,0 | NV | NV | NV |
| 0,2 | 0,0 | 0,0 | NV | NV | NV |
| 0,4 | 0,0 | 0,0 | NV | NV | NV |
| 0,8 | 0,0 | 0,0 | NV | NV | NV |
| 0,4 | 0,1 | 0,0 | 21,00 | 32,50 | NV |
| 0,4 | 0,2 | 0,0 | 11,00 | 12,50 | NV |
| 0,4 | 0,4 | 0,0 | 6,50 | 1,50 | 24,75 |
| 0,2 | 0,1 | 0,0 | 16,00 | 25,00 | NV |
| **0,2** | **0,2** | **0,0** | **7,00** | **4,00** | **2,90** |
| 0,2 | 0,4 | 0,0 | 11,50 | 2,00 | 33,40 |
| **0,4** | **0,4** | **0,1** | **4,50** | **1,50** | **7,20** |
| 0,4 | 0,4 | 0,2 | 4,50 | 1,50 | 10,20 |
| 0,2 | 0,2 | 0,1 | 7,50 | 5,00 | 2.38 |

Pela análise da tabela, é possível verificar que, quando é aplicado apenas um ganho proporcional ao erro ($Kp$), o sistema nunca atinge o erro nulo. Além disso, para valores muito baixos de $Kp$ o sistema não tem praticamente resposta e apresenta valores negativos, em contraste com o pretendido (50). De facto, isto é compreensível, já que um pequeno ganho resulta numa pequena resposta de saída a um grande erro de entrada e um controlador menos responsivo ou menos sensível. No entanto, quando se aplica um $Kp$ de 0,4 já se observa uma ligeira oscilação e a saída do sistema aproxima-se ao SP pretendido (50). Ainda assim, apenas se observa uma oscilação considerável quando $Kp$ é definido para cerca de 0,8, aproximando-se ainda mais do SP. Deste modo, e seguindo a técnica mencionada, decidiu considerar-se os 2 valores ($Kp = 0{,}4\ e\ Kp = 0{,}8$) e testar o sistema para metade de cada um dos valores de $Kp$ mencionados, adicionando um valor para $Ki$, que anteriormente assumia um valor nulo. Assim divide-se a análise para $Kp = 0{,}2$ e $Kp = 0{,}4$.

Desta forma, ao atribuir $Kp$ = 0,2 e um valor relativamente baixo como 0,1 para $Ki$ – observou-se que o sistema tende a convergir para o erro nulo. Este comportamento é espectável já que o termo integrador ($I$) elimina o erro residual, acumulando todo o erro que deveria ser ajustado até ao instante em que o sistema está a ser executado. No entanto, o sistema demora um período de tempo significativo a estabilizar. Verifica-se assim que, ao aumentar o valor de $Ki$, o sistema diminui o tempo de resposta e, consequentemente, a velocidade de resposta aumenta. Isto acontece porque o termo integral acelera o movimento do PV em direção ao ponto de ajuste, diminuindo, principalmente, o tempo de subida como é possível confirmar pela tabela 3. Um aumento demasiado significativo de $Ki$ acelera a resposta do sistema em demasia e o valor pode ultrapassar o ponto de ajuste (SP), pelo que resultará num *overshoot* ($M_p$) consideravél, como ocorre, por exemplo, quando $Ki = 0{,}4$, resultando num *overshoot de* 33,40%, o que não é aceitavél em muitas aplicações. Deste modo, uma resposta bastante interessante acontece num ponto "intermédio" quando $Kp = 0{,}2$ e $Ki = 0{,}2$, com um tempo de acomodação de 7s e um tempo de subida de 4s, apresentando um *overshoot* de apenas 2,90%, um resultado bastante apelativo.

Analisando as respostas do sistema para $Kp = 0{,}4$, conclui-se que as dinâmicas são bastantes semelhantes e que, ao aumentar o $Ki$, tanto o tempo de subida ($t_r$) como o tempo de acomodação ($t_s$), diminuem. Ainda assim, um resultado favorável deve-se ao tempo de subida de 1,5s e ao tempo de acomodação de 6,5s, obtido quando utilizamos os parâmetros $Kp = 0{,}4$ e $Ki = 0{,}4$. No entanto, possuí um *overshoot* bastante considerável de 24,75%.

Por fim, com o objetivo principal de diminuir o *overshoot* desta última resposta, aplicou-se um baixo ganho derivativo de 0,1. Para além de o *overshoot* ter diminuído de 24,75% para 7,3%, a velocidade de resposta aumentou consideravelmente, com um tempo de acomodação de 4,5s. Foram ainda testados valores maiores de $Ki$ (Tabela 3) mas, para esses casos, o sistema apresentou-se mais instável.

## 5. Discussão dos resultados

Na secção anterior utilizaram-se dois métodos para encontrar os parâmetros ótimos do controlador PID. Através do método de Ziegler-Nichols, a melhor resposta do sistema foi obtida com o controlador PI, que permitiu um tempo de acomodação de 7,5 s, um tempo de subida de 1,5 s e um *overshoot* de 21,8%. Fez-se uma tentativa de adicionar um ganho derivativo para diminuir o *overshoot* e aumentar a velocidade do sistema, mas o sistema tornou-se ainda mais instável.

Além do método de Ziegler-Nichols optou-se por um ajuste manual dos parâmetros. Depois de vários testes e observações de várias respostas, selecionaram-se dois conjuntos de parâmetros que podem ser usados com um resultado satisfatório, dependendo das suas aplicações.

- O primeiro, com $Kp = 0{,}4;\ Ki = 0{,}4\ e\ Kd = 0{,}1$ caracteriza-se principalmente por permitir uma rápida resposta e um rápido ajuste ao SP pretendido, com $t_s = 4{,}5s$ e $t_r = 1{,}5s$. A desvantagem deste controlador é possuir um *overshoot* de 7,3% que para certas aplicações pode não ser desejável.
- O segundo conjunto de valores, para os parâmetros $Kp = 0{,}2\ e\ Ki = 0{,}2$ caracteriza-se por um baixo *overshoot* (2,9%) e um tempo de resposta razoável com $t_s = 7s$ e $t_r = 4s$.

Deste modo, descartaram-se os parâmetros obtidos pelo método de Ziegler-Nichols por se revelar menos favorável em todos os aspetos comparativamente aos parâmetros ajustados manualmente.

Assim, as respostas a um degrau de 25 unidades, para o sistema com os dois controladores selecionados podem ser visualizáveis nas figuras 7 e 8. De notar que muitos outros valores intermédios para os parâmetros poderiam ser testados mas podemos considerar que se encontram nesta gama de valores.

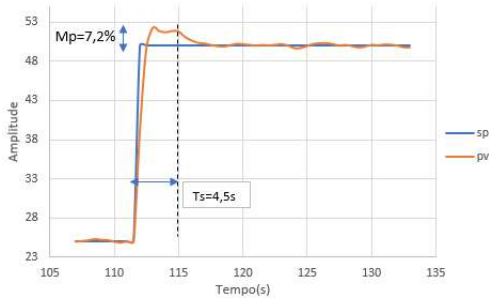


**Figura 7** – Controlador PID com com $Kp = 0{,}4;\ Ki = 0{,}4\ e\ Kd = 0{,}1$

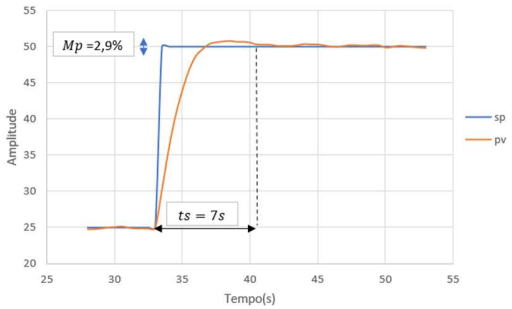


**Figura 8** – Controlador PI com $Kp = 0{,}2\ e\ Ki = 0{,}2$.

## 6. Conclusões

Com o presente trabalho foi possível a compreensão das noções básicas sobre o funcionamento e respetiva implementação de controladores PID com o auxílio das plataformas de Arduino e LabVIEW. De forma a ajustar os valores de P, I e D do sistema proposto, recorreu-se ao método de ajuste de Ziegler-Nichols e ao ajuste manual. O método de ajuste manual permitiu obter dois sistemas de controlo que estabilizam rapidamente com uma tolerância de 1%. Contudo, ambos apresentam vantagens e desvantagens um em relação ao outro. Embora o controlador PID com $Kp = 0{,}4,\ Ki = 0{,}4\ e\ Kd = 0{,}1$ estabilize mais rapidamente dentro da tolerância estabelecida ($t_s$ = 4,5s), o mesmo apresenta um *overshoot* de 7,2%, não sendo estas condições de funcionamento ideais para determinadas aplicações. Já o controlador PI, também otimizado manualmente com $Kp = 0{,}2\ e\ Ki = 0{,}2$, estabiliza mais lentamente ($t_s$ = 7s) mas de uma forma mais suave, isto é, com um *overshoot* muito menor (2,9%). Desta forma, ambos poderão ser usados a nível industrial embora com aplicações diferentes. Por outro lado, o controlador PI obtido pelo método de Ziegler-Nichols, apesar de também proporcionar resultados satisfatórios de operação, mostrou ser inferior em todos os níveis, especificamente no tempo de acomodação e no *overshoot*,

## 7. Referências Bibliográficas


[1] "PID Controller: Working, Types, Advantages & Its Applications." https://www.elprocus.com/the-working-of-a-pid-controller/ (accessed Feb. 08, 2022).

[2] R. P. Borase, · D K Maghade, · S Y Sondkar, and · S N Pawar, "A review of PID control, tuning methods and applications," *Int. J. Dyn. Control*, vol. 9, pp. 818–827, 2021, doi: 10.1007/s40435-020-00665-4.

[3] "What is LabVIEW? - NI." https://www.ni.com/pt-pt/shop/labview.html (accessed Feb. 5, 2022).

[4] "Controlador PID." https://stringfixer.com/pt/PID_control (accessed Feb. 5, 2022).

[5] A. José and R. Silva, "Controladores PID," Porto, 2021.