# Sistema de Controle de Temperatura para uma Estufa com Monitoramento via Aplicativo

**ORLEM L. D. SANTOS[1], JÓ D. S. M. JÚNIOR[1], MENDELSSON R. M. NEVES[1]**

[1]Departamento de Engenharia Elétrica – Universidade Federal de Roraima (UFRR)
Avenida Capitão Ene Garcez, 2413, Aeroporto – Boa Vista – RR – Brazil

orllem456@hotmail.com,jodossantosmelojunior@gmail.com, mendelsonrainer@gmail.com

***Abstract.*** *The temperature control can present difficulties due to either the fast heating, causing temperature peaks, or slow cooling causing a delay in response stabilization. These problems impact directly on the system response. In this paper, we present the prototype of a thermal system control that regulates the temperature inside a greenhouse. The control is feedback based with the presence of sensors, where the user can perform monitoring by mobile application, that uses a WebSocket communication protocol. There will be a clarification of mathematical modeling and parameters tuning. At last, it will be discussed and compared the design of basic PID (Proportional Integral Derivative) and Split range control techniques.*

***Resumo.*** *O controle da temperatura pode apresentar dificuldades seja devido ao rápido aquecimento, ocasionando picos de temperatura, ou a lenta refrigeração causando demora na estabilização da resposta. Esses problemas impactam diretamente na resposta do sistema. Neste artigo é apresentado o protótipo de um controle de temperatura para uma estufa. O controle será realimentado, ou seja, com o uso de sensores onde o usuário poderá efetuar o monitoramento via aplicação móvel, que utiliza para comunicação o protocolo WebSocket. Haverá esclarecimentos quanto a modelagem matemática e a determinação dos parâmetros dos controladores. Por fim, será discutido e comparado o projeto de um sistema de controle de temperatura com as técnicas de controle PID (Proporcional Integral Derivativo) básico e Split range (Faixa dividida).*

## 1. Introdução

Estufas são espaços fechados onde se eleva a temperatura do ar artificialmente. Na agricultura, seu uso se dá, principalmente, pela necessidade dos produtores de protegerem seus cultivos das grandes variações climáticas (tempestades, geadas, granizos, etc.). Além disso, auxiliam em um melhor desenvolvimento do plantio possibilitando ao produtor o plantio entre safras [Ruralnews 2012].

Alguns tipos de cultivos necessitam de um controle rigoroso de temperatura, pois sem esse há grande perda na quantidade de plantas geminadas e também má formação do fruto. A tecnologia para esses tipos de cultivos costumam ser de elevado valor comercial.

Os controladores de temperatura tem função de medir constantemente a temperatura e comparar com a temperatura desejada e corrigi-la, existem diversas técnicas de

controle aplicadas na regulação de temperatura como o *on/off*, P (proporcional), PD (Proporcional Derivativo), PI (Proporcional Integral), PID [Ogata 2010] e até outras técnicas mais avançadas como controle em cascata e controle *Split range* (Faixa dividida) [Wang et al. 2008]. A escolha entre essas técnicas dependerá, principalmente, do nível de robustez e refino desejado no projeto.

Neste presente trabalho é apresentado o projeto de um controle de temperatura de uma estufa. Serão apresentadas duas técnicas de controle uma baseada no PID básico e outra no controle *Split range*. Com o objetivo de construir um protótipo de baixo custo, os controladores serão implementados na plataforma Arduino pro mini [Pro Mini 2017] que é tecnologia de hardware livre que possibilita uma fácil integração dos componentes do hardware. Criou-se também uma interface amigável e de fácil compreensão para o usuário, que poderá monitorar e regular a temperatura da estufa via smartphone.

## 2. Protótipo do sistema para controle de temperatura

### 2.1. Estrutura do sistema

O protótipo do sistema foi desenvolvido em laboratório, conforme figura 1. O protótipo foi construído em uma caixa de papelão simulando um ambiente fechado. Os atuadores são uma lâmpada CC 12V para o aquecimento e um ventilador 12V (*cooler*) para o resfriamento. O sensor de temperatura DS18B20 [Maxim integrated Products 2015] utilizado faz a realimentação (*feedback*) para o controlador. Os controladores PID e *Split range* foram implementados na plataforma Arduino pro-mini [Pro Mini 2017], digitalmente, com uso da Arduino Software (IDE-*Integrated Development Environment*). Os dados obtidos pelo sensor e compensados pelo controladores são então enviados para o servidor (*Raspberry pi*) por intermédio do módulo *Wifi* ESP8266 [sparkfun 2017] onde são armazenados no banco de dados podendo, eventualmente, ser acessado pela aplicação móvel. O protocolo de comunicação entre os dispositivos é o *WebSocket* [Fette 2011].

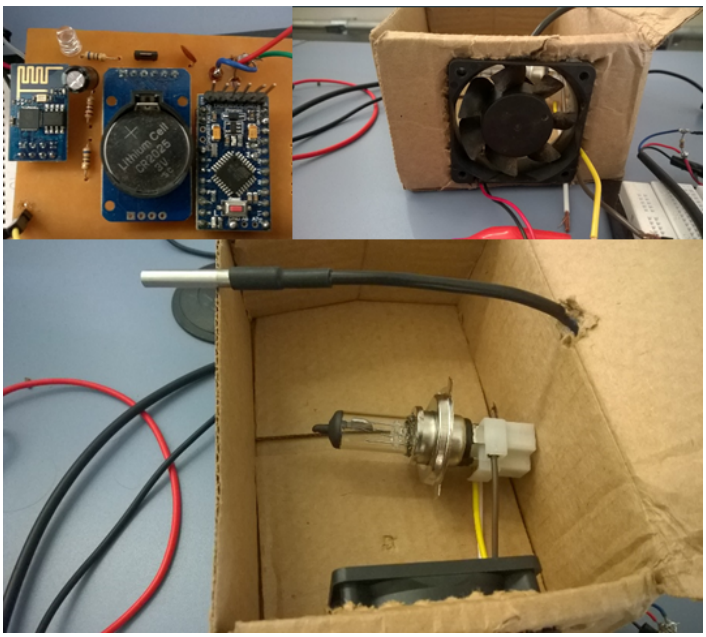

**Figure 1. Protótipo do sistema para controle de temperatura.**

A Figura 2 mostra o esquemático do hardware do sistema. Nesse esquemático pode-se observar os componentes do sistema que já foram comentados bem como outros como o regulador linear AMS317-3.3V [Advanced Monolithic Systems 2017] que tem a função de regular a tensão de 3.3V para o módulo *Wifi* ESP8266, RTC (*Real Time Clock*) ou relógio de tempo real DS3231 que responsável por fornecer o tempo (data, hora, minuto e segundo), dos dados de temperatura enviados, ao servidor e os transistores IRF3305 e BC548B utilizados, respectivamente, no controle da lâmpada CC 12V e do ventilador (*cooler*).

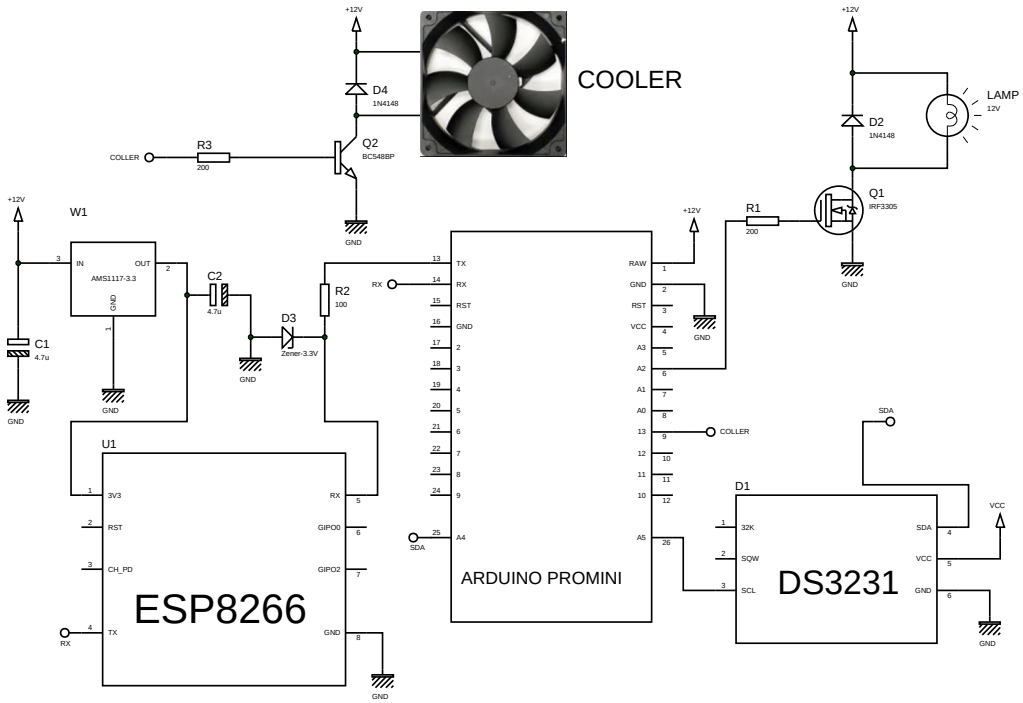


**Figure 2. Esquemático do hardware do sistema.**

A figura 3 apresenta as interações com o sistema Microcontrolador-Estufa-Lâmpada-*Cooler* para um controle de temperatura. Esse tipo de modelagem matemática é conhecida como caixa preta (*black-box*) bem como apresenta algumas vantagens em relação à modelagem física, pois geralmente esses modelos físicos não levam em consideração vários fatores práticos como a dinâmica do controlador, dinâmica do sensor e pertubações diversas.

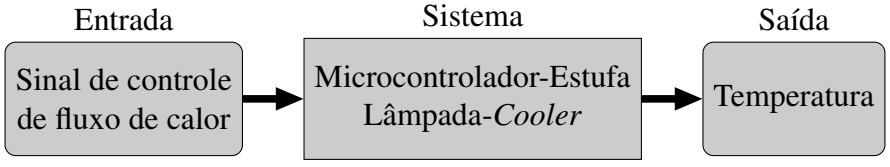


**Figure 3. Diagrama de blocos de interações com o sistema Microcontrolador-Estufa-Lâmpada-*Cooler*.**

A Figura 4 representa a planta em malha aberta do sistema na qual a entrada $\delta$ é o ciclo de trabalho (*duty cicle*) do sinal PWM (Pulse Width Modulation) ou Modulação

da Largura de Pulso, aplicado na porta do transistor *mosfet* IRF3305, que aciona uma lâmpada CC 12V .

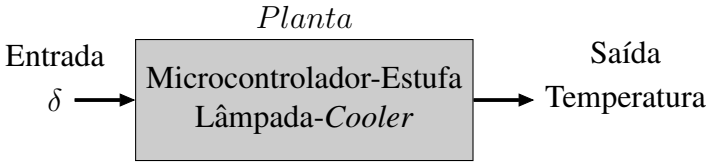


**Figure 4. Planta que relaciona a saída do sensor DS18b20 (temperatura) com a entrada $\delta$ (duty cicle).**

Utilizou-se a técnica resposta ao degrau unitário para obter a planta em malha aberta do sistema. Essa técnica se baseia em aplicar um sinal degrau unitário na entrada $\delta$ e mapear sua saída construindo um modelo matemático desse sistema.

## 2.2. Identificação do modelo matemático do sistema

Aplicando a técnica resposta ao degrau no sistema e enviando os dados para o *Matlab* via comunicação serial obtêm-se a resposta da figura 5. Nesse experimento o sinal degrau significa aplicar um PWM com o máximo $\delta$ da plataforma arduino, ou seja, $\delta = 255$.

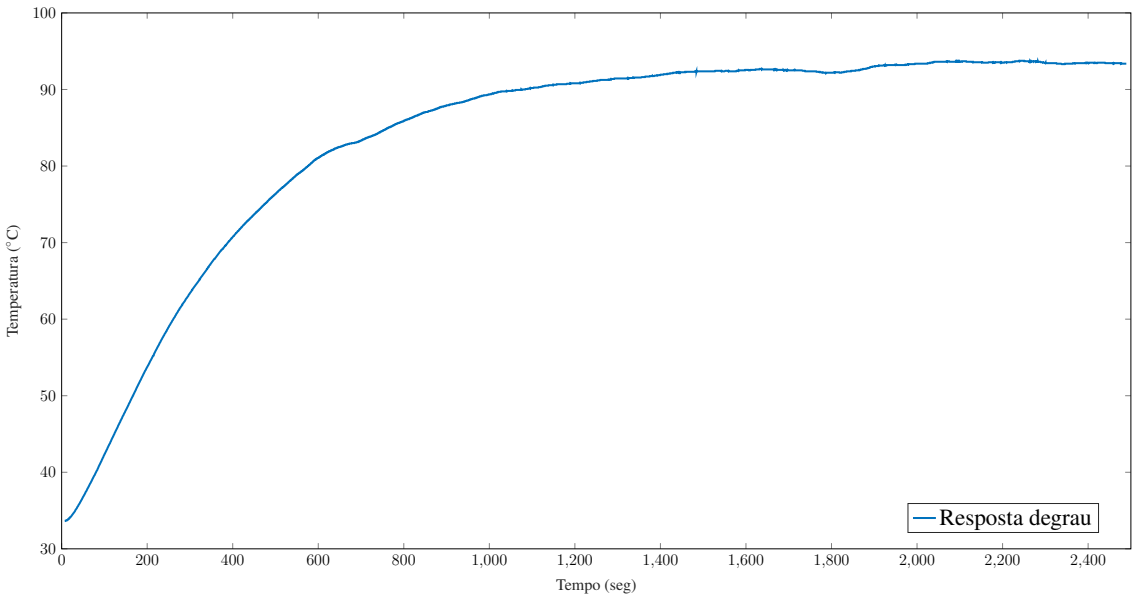


**Figure 5. Resposta ao degrau.**

Visto a curva característica da resposta ao degrau indica-se que é aproximadamente de primeira ordem, assim o ganho em malha aberta da planta conforme a resposta é:

$$G_p(s) = \frac{C(s)}{R(s)} = \frac{K}{\tau s + 1} \tag{1}$$

Onde K é o ganho CC e $\tau$ a constante de tempo do sistema. Examinando os dados obtidos na resposta ao degrau pode-se determinar $K$ e $\tau$. Devido aos ruídos sobre o sinal

que tem ponta de prova metálica do sensor de temperatura, que tende armazenar calor e demorar para perder temperatura, teremos a partir de um determinado ponto um desvio de valores. Outros motivos de desvio de valores podem ser uma variedade de efeitos externos ao sistema, como precisão do sensor e alterações em relação à temperatura ambiental.

Para simplificar a análise do modelo matemático não se executaram análises das perturbações do sistema. Analisando a figura 5, através do *Data cursor* do *Matlab*, se obtêm uma temperatura inicial de $T_\circ = 33,63°C$ em $t_0 = 7,307s$ enquanto a temperatura no estado estacionário é $93,44°C$ em 1458 s.

Similarmente como foi desenvolvido em [CTMS 2014] pode-se analisar que a constante tempo é o tempo no qual a resposta atinge $63\%$ do seu ganho CC. Sendo $K = 93,44 - 33,63 = 59,81$ então a temperatura para $63\%$ do ganho será:

$$T(63\%) = K \cdot 0,63 + T_\circ = K \cdot 0,63 + 33,63 = 71,31°C$$

Esse valor é atingido em um tempo de 409,3s. Assim obtêm-se que a constante de tempo é:

$$\tau = 409,3 - t_\circ = 409,3 - 7,331 = 401,67s$$

Portanto o modelo matemático é:

$$G_p(s) = \frac{59,81}{401,67s + 1} \tag{2}$$

A figura 6 apresenta a comparação entre modelo matemático $G_p(s)$ e a resposta degrau.

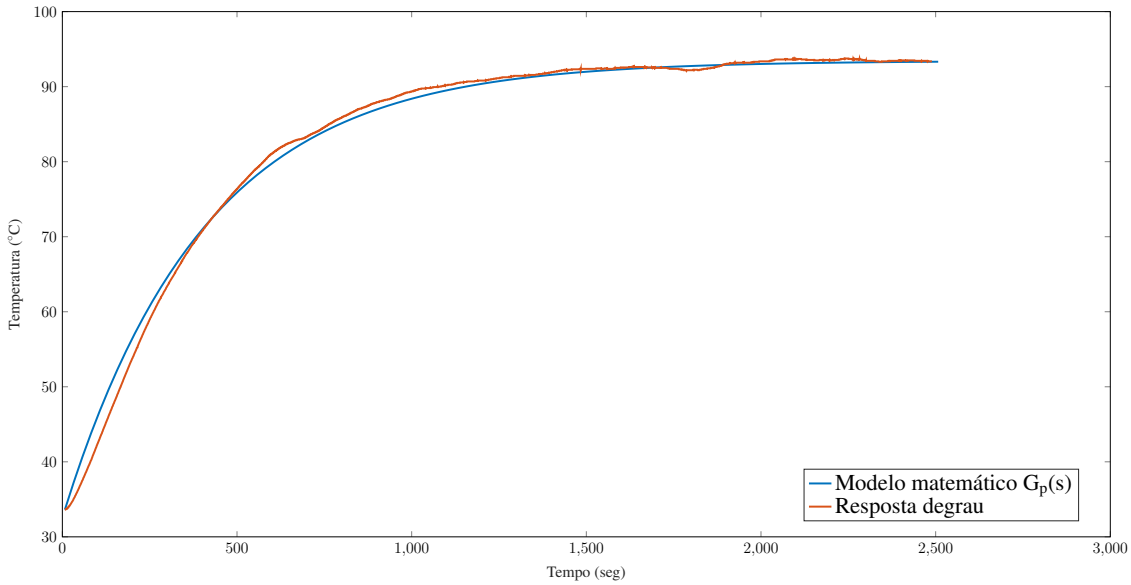


**Figure 6. Comparação entre modelo matemático $G_p(s)$ e a resposta degrau.**

## 3. Projeto do Controlador PID básico e Controlador *Split range*

### 3.1. Controlador PID básico

Para o projeto de um controlador PID, temos um controlador de ganho proporcional $K_p$, ganho integral $K_i$ e ganho derivativo $K_d$. A ação integral está diretamente ligada à precisão do sistema sendo responsável pelo erro nulo em regime permanente. O efeito desestabilizador do controlador PI é contrabalanceado pela ação derivativa que tende a aumentar a estabilidade relativa do sistema, ao mesmo tempo, que torna a resposta do sistema mais rápido devido ao seu efeito antecipatório.

O controlador PID é a soma das funções de transferência proporcional, integral e derivativa. Montando um diagrama de blocos, uma dada temperatura de referência passa pelo controlador compensador na plataforma Arduino que envia um sinal PWM de atuação da planta que tem como saída a temperatura $y(t)$ que é realimentada ao controlador. Conforme a Figura 7 [Dorf and Svoboda 2008].

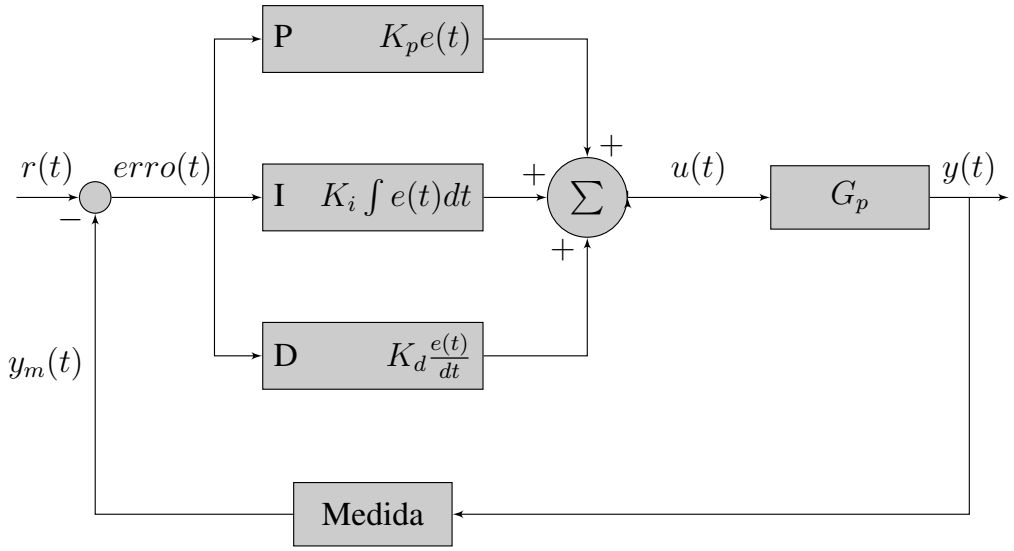


**Figure 7. Diagrama de blocos de um controlador PID.**

Nessa figura tem-se que $r(t)$ é o *Setpoint* (referência), $y_m(t)$ é a realimentação, obtida geralmente por um sensor, e as constates $K_p$, $K_i$ e $K_d$ são conhecidas como parâmetros do PID.

O processo de obtenção desses parâmetros $K_p$, $K_i$ e $K_d$ é conhecido como sintonia [Dorf and Svoboda 2008]. Esses parâmetros podem ser adquiridos a partir do modelo matemático obtido $G_p(s)$ (equação 2) na qual por intermédio dessa característica em malha aberta é obtida a função de compensação, dada pela equação 3.

$$G_c(s) = \frac{K(s+a)^2}{s} \tag{3}$$

Onde:

$$K_p = 2K \cdot a, K_i = K \cdot a^2, K_d = K$$

O sistema compensado apresenta, se sintonizado de forma eficiente, uma melhor resposta em regime permanente assim como a resposta transitória [Ogata 2010].

Esse controlador PID foi implementado, de forma digital, na plataforma Arduino por intermédio da biblioteca *Arduino PID Library* [Beauregard 2015]. A realimentação é obtida com o uso do sensor de temperatura DS18b20.

### 3.1.1. Processo de sintonia do controlador PID

Para encontrar os parâmetros do PID, em geral, é um tanto complicado por este motivo utilizou-se da ferramenta computacional *Matlab* por intermédio do algoritmo de otimização do Ogata (adaptado) [Ogata 2010] na qual tem a forma de um controlador PID equivalente, conforme à equação 3, para combinações de $K$ e $a$. O sistema compensado da estufa tem a forma do diagrama da figura 8.

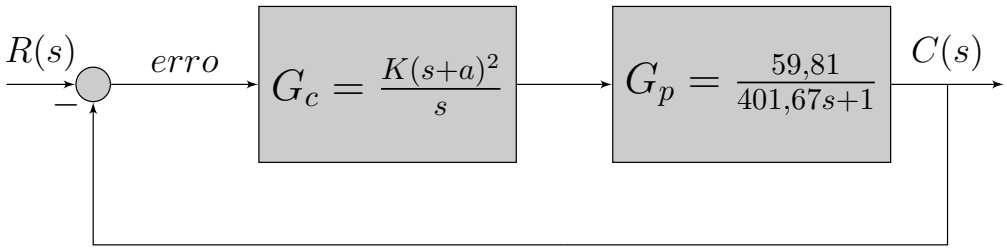


**Figure 8. Diagrama de blocos de um controlador PID da estufa.**

Adotando um critério de modo que o sistema esteja em malha fechada seja subamortecido e o sobressinal máximo da resposta ao degrau seja no máximo 10%. Para procurar $K$ e $a$ adequados damos um intervalo discreto de valores para cada um, depois por meio a resposta ao degrau unitário o algoritmo encontra uma combinação de $K$ e $a$ que satisfaça as condições do sobressinal máximo de $10\%$ ou menor [Ogata 2010].

Sendo que o intervalo de $K$ não deve ser muito grande para evitar que se utilize recursos desnecessários do sistema. Se não tiver uma solução para esses intervalos discretos deve-se aumenta-lo. Veja a figura 9 que descreve analogamente o algoritmo.

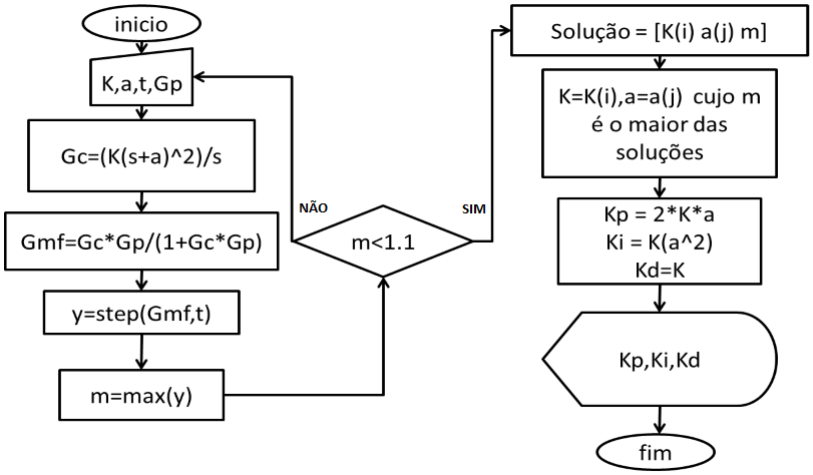


**Figure 9. Fluxograma do algoritmo de optimização utilizado na sintonia do controlador.**

Procurou-se encontrar um intervalo de K o qual tenha menor oscilação de $G_{mf}$ (ganho em malha fechada), desta forma é feita as tentativas, sendo que para este intervalo depois do sobressinal a resposta não vai para baixo da referência ou é bem próximo a referência 1 o que faz o sistema não oscilar muito para estabilizar. Também observa-se da planta ao ser ligada no máximo na variação de $1°C$ é no tempo aproximado de $97,86 - 88,71 = 10s$.

Para 1 < K < 10,0, 0 < a < 10 e 0 < t < 10 (segundos) obteve-se

$$K_p = 293,04 \ , \ K_i = 1084,2 \ \text{ e } \ K_d = 19,8$$

Esses são os parâmetros do controlador PID para o controle da temperatura que são valores de entrada da biblioteca *Arduino PID Library* [Beauregard 2015] que posteriormente será implementada como *firmware* da plataforma Arduino.

Vale ressaltar ainda que com a utilização desse algoritmo de otimização a escolha do controlador (PD, PI e PID) é feita de forma automática, pois o algoritmo já fornece as constantes, por exemplo, caso o $K_d$ encontrado fosse $0$ e $K_p$ e $K_i$ diferente de 0, o controlador seria um PI.

Verificando a estabilidade do sistema pelo critério *Routh-Hurwitz*, podemos saber se teremos polos no semiplano da direita através da inversão de sinal da primeira coluna de Routh [Nise 2013]. Ao verificar não houve polos no semiplano da direita nem no eixo $jw$ o que nos diz que o sistema é estável para os parâmetros encontrados.

O erro em regime permanente de uma entrada ao degrau é dado por

$$e(\infty) = \lim_{s\to\infty} \frac{1}{1 + G_c(s)G_p(s)} = 0$$

Assim tem-se um erro em regime permanente nulo e um sistema estável.

### 3.2. Controle *Split range* ou faixa dividida

O controle *Split range* pode ser aplicado quando existe uma variável controlada e diversos atuadores. Em um sistema de controle de temperatura de uma estufa o *Split range* é uma técnica adequada, pois esse sistema tem duas faixas (*ranges*) de atuação aquecimento e resfriamento. Dessa forma, no sistema da estufa utiliza-se a lâmpada CC 12V e o *cooler* como atuadores.

Neste protótipo essa técnica foi aplicada da seguinte maneira:

- Quando o sistema está aquecendo o *cooler* está desligado atuando apenas o controlador PID na lâmpada CC 12V.
- Quando o erro ($T_{ref} - T_{medida}$) do sistema for negativo o ventilador é ligado e permanece até quando o erro for menor que $-0.5$.

A figura 10 demonstra o funcionamento do controle *Split range*. O gráfico dessa figura mostra a forma de atuação e as faixas de operação.

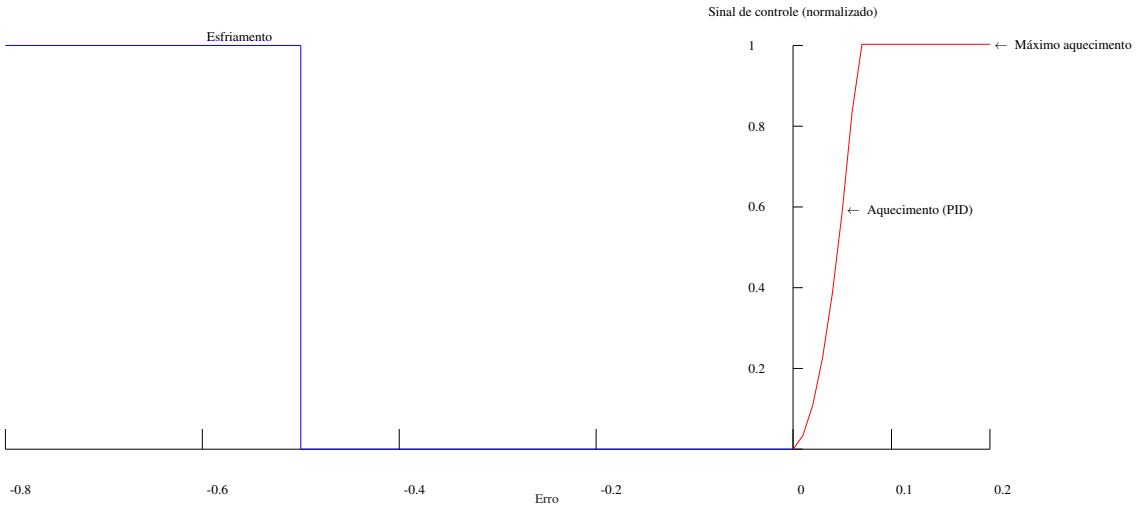


**Figure 10. Diagrama para o controlador *Split range*.**

## 4. Softwares desenvolvidos para o protótipo

### 4.1. Aplicação móvel e servidor *WebSocket*

Buscando o aperfeiçoamento e diferenciação em relação a outros projetos como [Guerra 2006, da Silva and Rosado 2012] que implementaram um sistema de controle de temperatura numa estufa. Fez-se uso da plataforma móvel para integração de dados entre um dispositivo móvel, dispositivo hardware (controlador) e um servidor *WebSocket*.

A aplicação móvel envia a temperatura de referência para aplicativo servidor que repassa para o controlador. O controlador, a cada 1min, envia a temperatura medida junto com tempo (data, hora e segundo), fornecido pelo RTC DS3231, para o aplicativo servidor que os armazena num banco de dados. Eventualmente, conforme requisição da aplicação móvel para o aplicativo servidor os dados armazenados nele são enviados para a aplicação móvel que os armazena em seu próprio banco. Na figura 11 é descrito como é feito o processo de comunicação entre as aplicações e o armazenamento de dados tanto no servidor como na aplicação móvel.

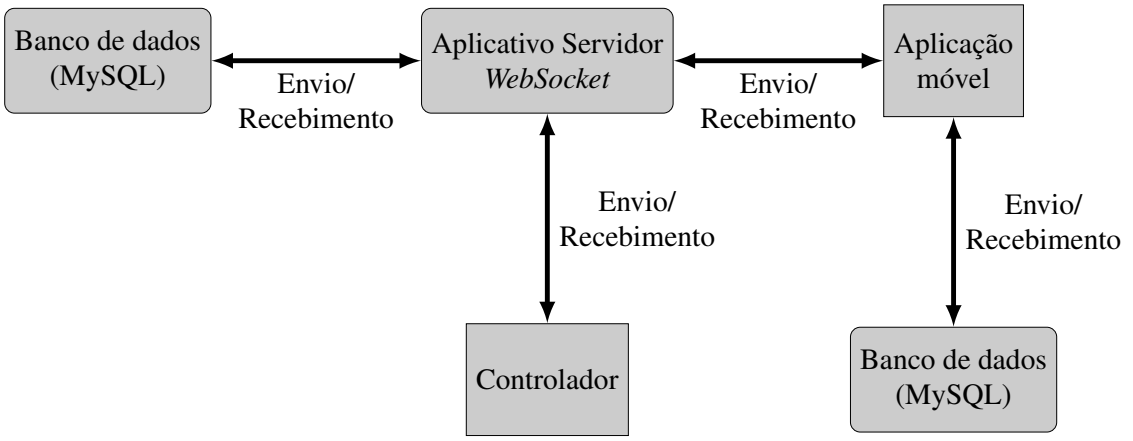


**Figure 11. Diagrama de blocos referente a comunicação entre os dispositivos envolvidos no sistema.**

### 4.2. Protocolo de comunicação

A forma de comunicação entre os dispositivos do sistema se dá pelo protocolo *WebSocket*, baseado no protocolo TCP (Protocolo de Controle de Transmissão) que permite a comunicação bidirecional entre um cliente (no caso a aplicação móvel) e um host remoto (no caso o aplicativo servidor), esse protocolo consiste em um *handshake* (aperto de mão) interpretados por servidores HTTP como um pedido de atualização abrindo somente uma conexão para várias requisições sendo esta uma vantagem em relação ao protocolo HTTP a qual faz uma requisição por conexão. Desta forma uma vez estabelecida a conexão através de *WebSocket* apenas mensagens são trocadas sem a necessidade de apresentação de cabeçalho extenso desta forma a vazão de informações do *WebSocket* é muito superior ao HTTP permitindo este ser um protocolo de baixa latência [Fette 2011].

Outra vantagem do protocolo *WebSocket* é que a troca de mensagens entre cliente e servidor há uma chave especifica (chave de máscara) que permite apenas WebSockets intermediários compatíveis desmascarem e inspecionar as mensagens, o que as protege de ataques na infraestrutura permitindo uma comunicação segura.

### 4.3. Aplicação móvel

A aplicação móvel foi desenvolvida no sistema operacional (SO) Android que, atualmente, encontrasse em diversos dispositivos principalmente nos smartphones sendo bastante difundida as suas aplicações nas quais podem ser desenvolvidas na plataforma de código aberto da Google, a IDE Android Studio, o qual fornece o Android SDK com um conjunto de API's necessárias para desenvolver as aplicações em Java.

Essa aplicação móvel conta com a biblioteca *WebSocket nv-websocket-client* [Authlete, Inc. and Neo Visionaries Inc. 2017] que já tem uma implementação nativa da biblioteca para gerenciamento de mensagens recebidas e enviadas no protocolo *WebSocket*.

### 4.4. Aplicativo servidor

O aplicativo servidor deve estabelecer a comunicação entre os clientes trocando mensagens da aplicação móvel para dispositivo hardware e vice-versa. Como o aplicativo servidor deve ser um *host* dedicado uma opção de baixo custo foi a utilização do *Raspberry pi* que é um computador onde pode se instalar um SO de distribuição Linux. Para a programação visando à utilização de POO (Programação Orientada a Objetos) no Linux já vem nativamente a linguagem leve e modular do *Python* não exigindo instalar kits de desenvolvimento de outras linguagens sem onerar memória e processamento do servidor.

Esse aplicativo necessita de um armazenamento de uma série de dados de temperatura e o tempo (data, hora, minuto e segundo) em que aconteceu, desta forma para fazer seu monitoramento (exibido em forma de gráfico na tab gráfico da figura 12) e histórico (exibido em forma de tabela) utiliza-se um banco de dados que relaciona essas informações.

O *Python* tem como interface de banco dados o *Python database API* [Lemburg 2008] suportando vários bancos de dados como, por exemplo, o *MySQL* um dos mais utilizados que tem o Workbech como uma ferramenta para planejar e editar o banco de dados para o uso do aplicativo servidor. Para o aplicativo servidor suprir suas

obrigações com uma rápida resposta é necessário a utilização de *threads*, deixando dessa forma as tarefas dedicadas, uma *thread* fica responsável pelo armazenamento dos dados no banco, outra *thread* para envio de dados armazenados no banco de dados e a *thread* principal faz a conexão de clientes e o envio/recebimento de mensagens.

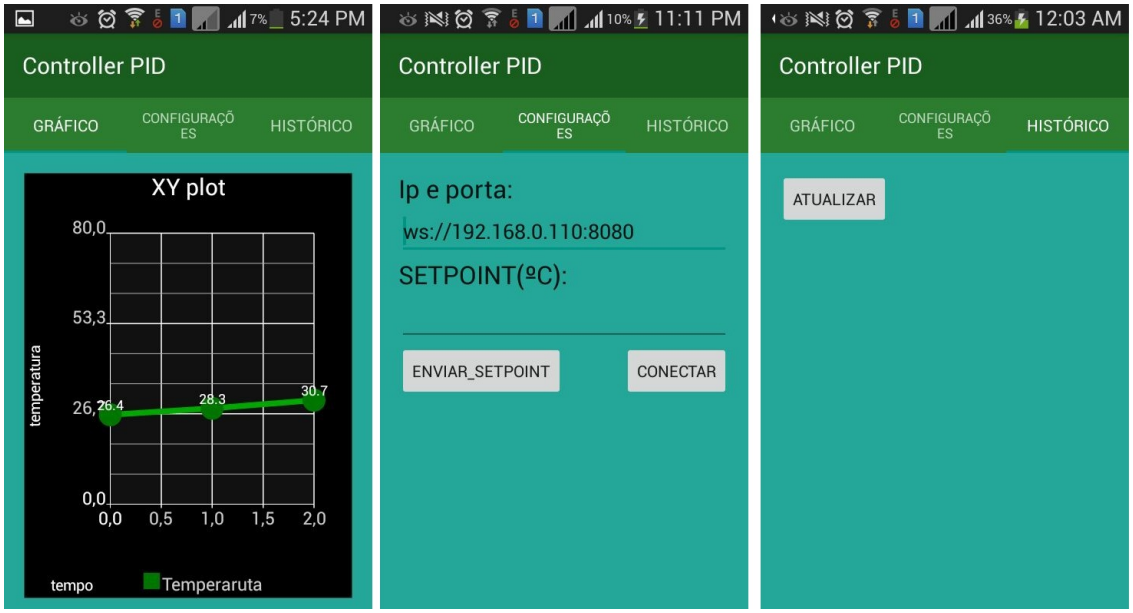


**Figure 12. Layout da aplicação móvel.**

## 5. Resultados

Nesta seção serão apresentados resultados obtidos do sistema com a utilização dos controladores descritos. Foram realizados dois experimentos:

1. No primeiro experimento é utilizado o controlador PID básico na qual o PID atua com base na realimentação recebida pelo sensor DS18b20. Nesse caso o atuador é apenas a lâmpada CC 12V.
2. No segundo experimento é utilizado o Controlador *Split range*, que possui como característica a presença de um segundo atuador nesse caso o *cooler*. Dessa forma após ultrapassado um certo valor do erro é acionado o ventilador para diminuir a temperatura.

A figura 13 mostra a resposta do sistema para o caso do controlador PID básico. O gráfico que descreve a resposta do sistema apresenta uma resposta subamortecida e apresenta um *overshoot* (sobressinal).

Nesse primeiro experimento o sistema apresentou as seguintes especificações de resposta transitória:

- Tempo de subida (*rise time*) $t_r$: $80.3368$ segundos
- Tempo de acomodação (*settling time*) $t_s$: $539.5466$ segundos
- Sobressinal (*overshoot*) $M_p$: $4.4406\%$
- Tempo de pico (*peak time*) $t_p$: $198.4684$ segundos

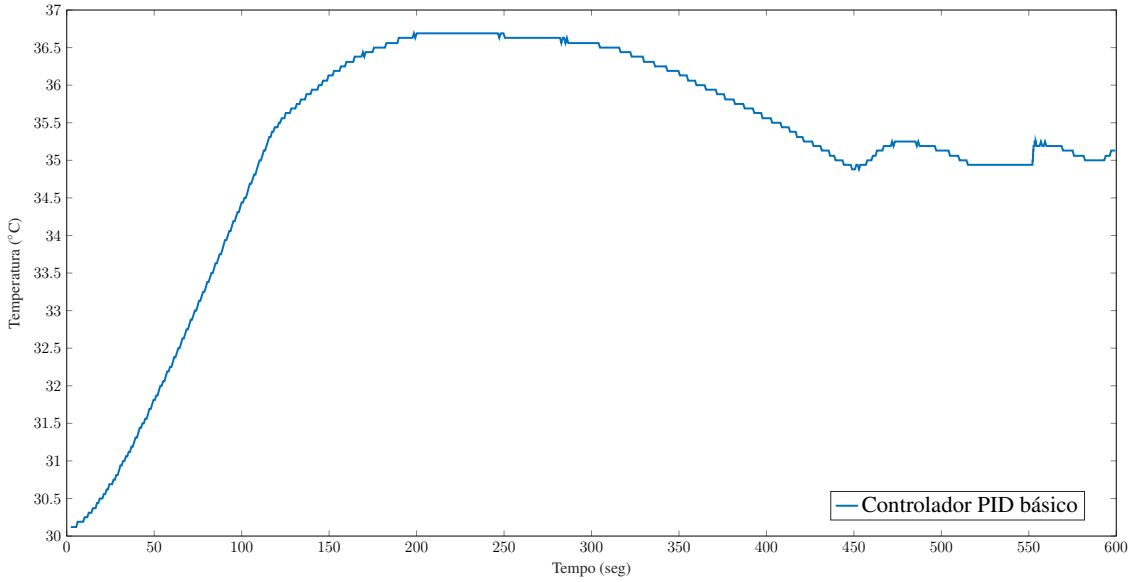


**Figure 13. Resposta do sistema com o controlador PID básico.**

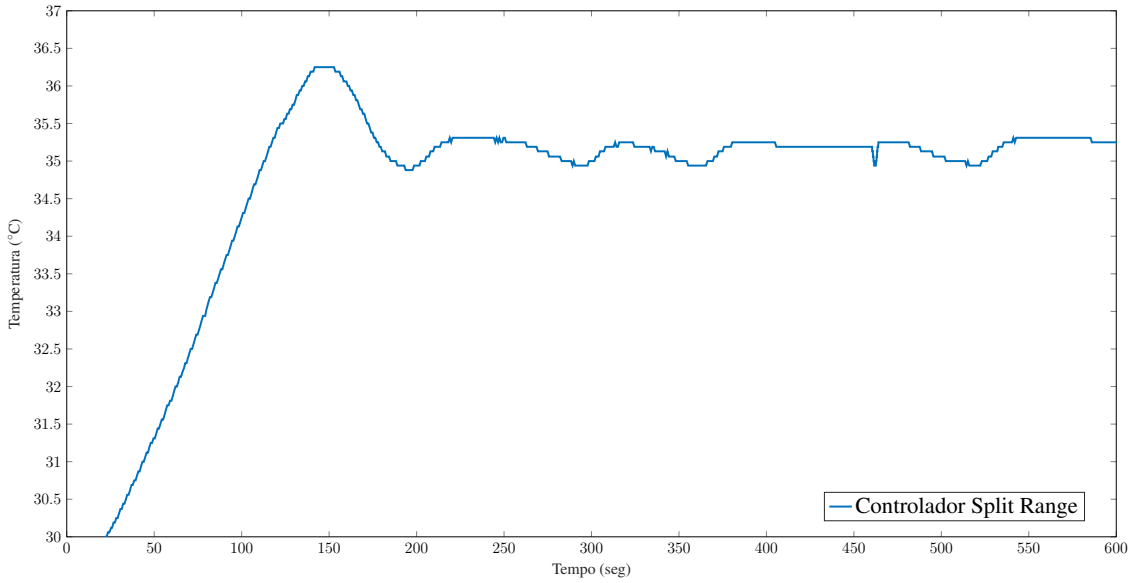


**Figure 14. Resposta do sistema com controle *Split range*.**

A figura 14 mostra a resposta do sistema para o caso do controlador *Split range*. O experimento foi realizado nas mesmas condições que o primeiro experimento. Analisando a resposta verificou-se que ainda há um pequeno sobressinal, mas nesse caso menor.

No segundo experimento o sistema apresentou as seguintes especificações de resposta transitória:

- Tempo de subida (*rise time*) $t_r$: 78.2444 segundos
- Tempo de acomodação (*settling time*) $t_s$: 585.4144 segundos
- Sobressinal (*overshoot*) $M_p$: 3.0122%
- Tempo de pico (*peak time*) $t_p$: 141.7407 segundos

Na figura 15 houve a sobreposição dos gráficos das figuras 13 e 14 na qual é possível verificar a diferença no tempo de acomodação da temperatura. Analisando o gráfico da figura 15 fica evidente que o controlador *Split range* efetua um controle mais eficiente da temperatura. Esse melhor desempenho se deve ao fato do *cooler* atuar quando o sinal de controle da lâmpada não fazer mais efeito na mudança de estado do sistema.

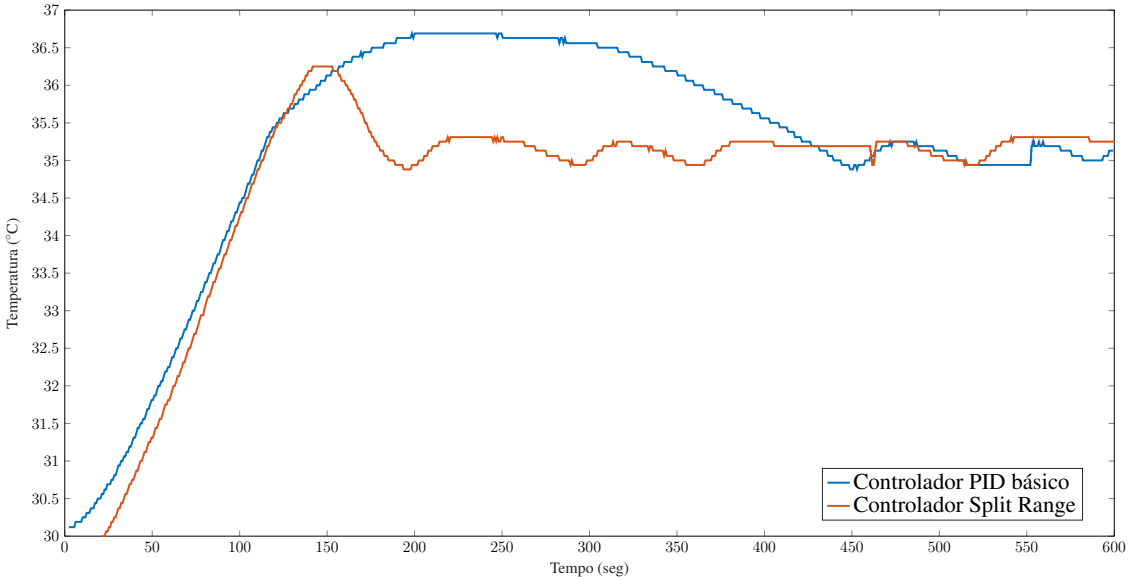


**Figure 15. Comparação das respostas com o controlador PID básico e o controlador *Split range*.**

## 6. Conclusão

Neste projeto, discutimos brevemente um esquema básico das técnicas de controle PID básico e *Split range*. Na sintonia do controlador PID foram realizados ajustes finos dos seus parâmetros ($K_P$, $K_I$ e $K_D$). Para isso foi utilizado um método de otimização computacional, a fim de buscar os parâmetros adequados conforme as especificações da resposta ao degrau. O controle *Split range* foi utilizado para resolver o problema de resfriamento do sistema quando usado apenas o atuador de aquecimento (lâmpada CC 12V).

Os controladores tanto o PID básico e o *Split range* se apresentaram eficientes no controle de temperatura, não apresentando um erro na resposta tão significativo na aplicação a qual se destina. O controlador *Split range* obteve um desempenho melhor no controle de temperatura.

Por se tratar apenas de um protótipo ainda requer aperfeiçoamento tanto no hardware quanto no próprio PID, melhor sintonia. Ambas melhorias trariam uma melhor precisão no controle da temperatura.

O software do dispositivo móvel se mostrou efetivo e prático na comunicação usuário e máquina, criando uma interface amigável para esse, cumprindo o quesito de monitoramento proposto no artigo. O protocolo *WebSocket* apresentou grandes vantagens na aplicação de monitoramento que, em geral, tem a necessidade de tempo de resposta curto. Essas vantagens foram uma baixa latência (quase em tempo real), velocidade de transferência bidirecional e a segurança do protocolo.

Portanto o projeto se mostra útil a aplicação a qual se destina, podendo ser ampliado para melhor atendimento das necessidades pertinentes a este tipo de sistema.

## References

Advanced Monolithic Systems (2017). Ams117 1a low dropout voltage regulator. http://www.advanced-monolithic.com/pdf/ds1117.pdf.

Authlete, Inc. and Neo Visionaries Inc. (2017). High-quality websocket client implementation in java. https://github.com/TakahikoKawasaki/nv-websocket-client. Acessado: 26-06-2017.

Beauregard, B. (2015). Arduino pid library. *Arduino Playground*.

CTMS (2014). Ctms - control tutorials for matlab and simulink. activity 4: Temperature control of a light bulb. http://ctms.engin.umich.edu/CTMS/index.php?aux=Activities_Lightbulb. Acessado: 2015-12-20.

da Silva, C. E. F. and Rosado, V. O. G. (2012). Sintonia de um controlador pid no aquecimento de uma câmara térmica. *Journal of Exact Sciences*, 18(1).

da Silva, J. G. (2014). O controlador proporcional-integral-derivativo. http://www.ece.ufrgs.br/jmgomes/pid/Apostila/apostila/node30.html. Acessado: 2015-12-20.

Dorf, R. and Svoboda, J. (2008). *Introduction to Electric Circuits*. LTC, 7 th edition.

Fette, I. (2011). The websocket protocol.

Guerra, L. (2006). Uso de compensador pid no controle da taxa de variação de temperatura em um forno elétrico a resistência. *Universidade Federal do Rio de Janeiro*, page 44.

Lemburg, M.-A. (2008). Python database api specification v2. 0. *Python Enhancement Proposal*, 249.

Maxim integrated Products (2015). Programmable resolution 1-wire digital thermometer. https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

Nise, N. S. (2013). *Control Systems Engineering*. LTC, 6 th edition.

Ogata, K. (2010). *Modern Control Engineering*. Prentice Hall, 5 th edition.

Pro Mini, A. (2017). Arduino pro mini. https://www.arduino.cc/en/Main/ArduinoBoardProMini. Acessado: 24-05-2017.

Ruralnews (2012). Estufas - producao agricola em ambientes controlados. Avaliable: [http://www.ruralnews.com.br/visualiza.php?id=202]. Acessado: 2015-12-20.

sparkfun (2017). Wifi module - esp8266. https://www.sparkfun.com/products/13678. Acessado: 25-05-2017.

Wang, Q.-G., Ye, Z., Cai, W.-J., and Hang, C.-C. (2008). *PID control for multivariable processes*, volume 373. Springer Science & Business Media.