

CENTRO UNIVERSITÁRIO FEI
GUILHERME BRAGA DE PAULA

**SLIDING MODE CONTROL E PID NO CONTROLE DE TEMPERATURA DE
UM FLUIDO COM PARAMETRIZAÇÃO VIA IOT**

São Bernardo do Campo

2020

GUILHERME BRAGA DE PAULA

**SLIDING MODE CONTROL E PID NO CONTROLE DE TEMPERATURADE
DE UM FLUIDO COM PARAMETRIZAÇÃO VIA IOT**

Relatório Final de Iniciação Científica
apresentado ao Centro Universitário FEI,
como parte dos requisitos do Programa
PIBIC-FEI. Orientado pelo Prof. Fabrizio
Leonardi.

São Bernardo do Campo

2020

RESUMO

A implementação do controle de temperatura de fluidos pode ser feita de diversas maneiras, por exemplo utilizando o controlador linear PID, devido o seu fácil manuseio e por existir muitas formas de sintonizá-lo. Por ser um sistema relativamente simples, ele pode não se adequar bem a perturbações e erros de modelagem, além da possibilidade de apresentar falta de desempenho por não conseguir acompanhar a referência. O controlador por modos deslizantes (Sliding Mode Control – SMC) é um controlador não linear robusto que se apresenta superior no controle de sistemas descontínuos. O projeto possuiu como objetivo inicial analisar e comparar os controladores PID e SMC para o controle de temperatura de um fluido em repouso e com suas parametrizações sendo feitas remotamente por meio de um sistema baseado em Internet das Coisas (IoT). A planta do projeto seria implementada através de um reservatório metálico com aquecimento do fluido sendo feito de maneira indireta por uma resistência elétrica. No decorrer do projeto, alguns fatores fizeram com que parte da metodologia do projeto fosse alterada e consequentemente, o objetivo inicial não foi alcançado. Existem diversos serviços para a implementação de simulações com plantas emuladas, chamados sistema de controle *hardware-in-the-loop* e esses serviços possuem custos elevados e assim, acaba por não ser acessível para a maioria dos usuários. Desse modo, o projeto possui como objetivo apresentar uma plataforma de baixo custo e uma metodologia para implementação de sistemas *hardware-in-the-loop* por meio das plataformas de prototipagem eletrônica Arduino UNO e ESP32 e com suas parametrizações sendo feitas remotamente por meio de um sistema baseado em Internet das Coisas (IoT). A razão da parametrização remota é proporcionar o reprojeto do controlador e *download* de seus parâmetros em locais de difícil acesso. Para demonstrar a metodologia, foi implementado um controlador PID num ESP32 e uma planta dinâmica de segunda ordem emulada no Arduino UNO, interligando os microcontroladores por meio dos conversores A/D e D/A das plataformas. A metodologia proposta é baseada na implementação em tempo real das equações diferenciais ordinárias do controlador e da planta por meio de simulação dos integradores da sua representação no espaço de estados e fazendo uso de interrupções por hardware.

Palavras-chave: *Hardware-in-the-loop*. Controle PID. Controle por Modos Deslizantes. Internet das Coisas.

LISTA DE FIGURAS

| | |
|--|----|
| Figura 1: Controle em Malha Fechada | 11 |
| Figura 2: Classificação dos métodos de simulação em relação à velocidade e exemplos de aplicação | 13 |
| Figura 3: Classificação da simulação em tempo real | 14 |
| Figura 4: Simulação hardware-in-the-loop e possíveis estruturas híbridas | 15 |
| Figura 5: Sistema básico com Controle PID | 16 |
| Figura 6: Implementação do Anti-Windup via Tracking | 19 |
| Figura 7: Efeito da ação do Anti-Windup do sistema | 19 |
| Figura 8: Sistema Massa-Mola Amortecido | 21 |
| Figura 9: Resposta ao degrau sem compensador | 22 |
| Figura 10: Resposta ao degrau com Compensador Proporcional | 23 |
| Figura 11: Resposta ao degrau com Compensador PD | 23 |
| Figura 12: Resposta ao degrau com Compensador PI | 24 |
| Figura 13: Resposta ao degrau com Compensador PID | 25 |
| Figura 14: Diagrama de Fase da Lei de Controle | 27 |
| Figura 15: Diagrama de Fase do Sistema de Estrutura Variável | 28 |
| Figura 16: Diagrama de Fase para valores de γ altos | 29 |
| Figura 17: Planos de fase interceptando a superfície L de condições iniciais diferentes | 30 |
| Figura 18: Diagrama de fase de um movimento deslizante | 30 |
| Figura 19: Ação de Controle | 31 |
| Figura 20: Exemplo SMC | 32 |
| Figura 21: Resposta do Sistema com SMC | 33 |
| Figura 22: Comunicação MQTT | 37 |
| Figura 23: Pinagem ESP32 | 38 |
| Figura 24: Pinagem Arduino UNO | 40 |
| Figura 25: Passo 1 – Instalação ESP32 na IDE Arduino | 43 |
| Figura 26: Passo 2 – Instalação ESP32 na IDE Arduino | 43 |
| Figura 27: Passo 3 – Instalação ESP32 na IDE Arduino | 44 |
| Figura 28: Passo 4 – Instalação ESP32 na IDE Arduino | 44 |
| Figura 29: Aspectos físicos do ESP32 | 45 |
| Figura 30: Seleção do Microcontrolador na IDE | 45 |
| Figura 31: Configurações ESP32 (Software) | 46 |

| | |
|---|----|
| Figura 32: Configuração ESP32 (Hardware)..... | 46 |
| Figura 33: PWM utilizado com o MS15 | 47 |
| Figura 34: Declaração de Variáveis..... | 47 |
| Figura 35: Função Setup() | 48 |
| Figura 36: Função <i>loop()</i> | 48 |
| Figura 37: PWM com Duty Cycle de 100% | 49 |
| Figura 38: PWM com Duty Cycle de 15% | 49 |
| Figura 39: Timer – Definição das variáveis e função de alarme | 50 |
| Figura 40: Timer – Declaração dos parâmetros..... | 51 |
| Figura 41: Timer – Lógica do Botão | 51 |
| Figura 42: Exemplo de Implementação do Timer | 52 |
| Figura 43: Disco com código de Gray | 52 |
| Figura 44: Simulação Código de Gray | 53 |
| Figura 45: WebClient HiveMQ | 55 |
| Figura 46: WebClient HiveMQ | 55 |
| Figura 47: Página inicial MQTT Dash | 56 |
| Figura 48: Configurando nova aplicação (1)..... | 56 |
| Figura 49: Configurando nova aplicação (2)..... | 57 |
| Figura 50: Funções da aplicação | 57 |
| Figura 51: Configuração nova função | 58 |
| Figura 52: Teste da Aplicação | 58 |
| Figura 53: Gráfico Referência x Planta - Simulink | 60 |
| Figura 54: Sinais de Esforço de Controle - Antes e pós conversões | 60 |
| Figura 55: Layout Sistema HIL - Placa de Ensaio | 61 |
| Figura 56: Construção do diagrama de blocos canônico | 62 |
| Figura 57: Diagrama de blocos na forma canônica observável da equação (3.1) | 63 |
| Figura 58: Diagrama de blocos da equação (3.2) | 64 |
| Figura 59: Fluxograma da síntese da EDO da planta no Arduino Uno..... | 65 |
| Figura 60: Diagrama de blocos da EDO do controlador PID..... | 66 |
| Figura 61: Layout MQTT Dash..... | 67 |
| Figura 62: Resultado 1 - Resposta simulada (esquerda) e real (direita)..... | 68 |
| Figura 63: Resultado 2 - Resposta simulada (esquerda) e real (direita)..... | 69 |

LISTA DE QUADROS

| | |
|---|----|
| Quadro 1: Efeito do aumento das ações de controle | 20 |
| Quadro 2: Lista de Materiais | 42 |
| Quadro 3: Brokers gratuitos | 54 |

SUMÁRIO

| | | |
|----------------|---|-----------|
| 1 | INTRODUÇÃO | 9 |
| 1.1 | OBJETIVO..... | 9 |
| 1.2 | JUSTIFICATIVA..... | 9 |
| 2 | REVISÃO BIBLIOGRÁFICA | 11 |
| 2.1 | CONTROLE EM MALHA FECHADA | 11 |
| 2.2 | SISTEMA DE CONTROLE HARDWARE-IN-THE-LOOP (HIL) | 12 |
| 2.3 | CONTROLE PID | 16 |
| 2.3.1 | Controle Proporcional | 16 |
| 2.3.2 | Controle Integral | 17 |
| <i>2.3.2.1</i> | <i>Wind-up.....</i> | <i>17</i> |
| <i>2.3.2.2</i> | <i>Anti Wind-up</i> | <i>18</i> |
| 2.3.3 | Controle Derivativo | 20 |
| 2.3.4 | Exemplo de Aplicação | 20 |
| <i>2.3.4.1</i> | <i>Controlador Proporcional.....</i> | <i>22</i> |
| <i>2.3.4.2</i> | <i>Controlador Proporcional+Derivativo.....</i> | <i>23</i> |
| <i>2.3.4.3</i> | <i>Controlador Proporcional+Integral</i> | <i>24</i> |
| <i>2.3.4.4</i> | <i>Controlador PID.....</i> | <i>24</i> |
| 2.4 | CONTROLE SMC | 25 |
| 2.5 | INTERNET DAS COISAS | 33 |
| 2.5.1 | Protocolo MQTT | 36 |
| 2.5.2 | Microcontrolador ESP32..... | 37 |
| <i>2.5.2.1</i> | <i>Sinal PWM.....</i> | <i>38</i> |
| <i>2.5.2.2</i> | <i>Timer.....</i> | <i>39</i> |
| 2.5.3 | Arduino UNO..... | 39 |
| 3 | METODOLOGIA | 41 |
| 3.1 | MATERIAIS | 42 |

| | | |
|-------|---|----|
| 3.2 | TESTES INICIAIS | 42 |
| 3.2.1 | Configuração IDE Arduino para ESP32 | 42 |
| 3.2.2 | Implementação do PWM | 47 |
| 3.2.3 | Implementação do Timer | 49 |
| 3.2.4 | Conexão MQTT | 53 |
| 3.3 | PROCEDIMENTO EXPERIMENTAL | 58 |
| 3.3.1 | Implementação do sistema HIL no Simulink | 58 |
| 3.3.2 | Montagem dos dispositivos na Protoboard | 61 |
| 3.3.2 | Representação por diagrama de blocos | 62 |
| 3.3.3 | Implementação das equações diferenciais | 64 |
| 3.3.4 | Parametrização IoT | 66 |
| 4 | RESULTADOS | 68 |
| 5 | Conclusão | 70 |
| | REFERÊNCIAS | 72 |
| | Anexo A – Código Teste MQTT | 75 |
| | Anexo B – Sistema HIL no simulink | 79 |
| | Anexo C – Código final do controlador | 80 |
| | Anexo D – Código final da planta | 87 |

1 INTRODUÇÃO

1.1 OBJETIVO

Existem diversos tipos de controladores para realizar o controle de um sistema e muitos necessitam, eventualmente, de reprojeto. O projeto **buscava** a investigação comparativa entre os controladores PID e SMC, analisando seus métodos de sintonia e os resultados adquiridos no controle de temperatura de um fluido em repouso. **Mas, em consequência do período de quarentena devido à pandemia gerada pelo COVID-19 no ano de 2020, o projeto visa propor uma plataforma de baixo custo e uma metodologia para implementação de sistemas *hardware-in-the-loop* por meio das plataformas de prototipagem eletrônica Arduino UNO e ESP32. A parametrização do controlador será feita de maneira remota utilizando a Internet das Coisas e, para tal, será utilizado o microcontrolador ESP32, em conjunto com o protocolo de comunicação MQTT (Message Queuing Telemetry Transport).**

1.2 JUSTIFICATIVA

O projeto de controladores de malha fechada depende das características da planta e do desempenho esperado e, para implementá-lo, existem diversas opções de técnicas de controle para ser escolhida. O controlador PID é o mais utilizado em aplicações práticas, mas pode implicar em limitações de desempenho por ser um controlador linear de tempo contínuo. Controladores desse tipo costumam conferir respostas assintóticas em malha fechada, fazendo com que a saída só atinja o valor final depois de um tempo infinito. **Existem diversos recursos de alto desempenho disponíveis no mercado para implementação de simulações com plantas emuladas, sendo atribuído a esses recursos o nome de “sistema de controle *hardware-in-the-loop*”. Entretanto, esse tipo de produto muitas vezes apresenta custos elevados, tornando-os inacessíveis para a maioria dos usuários. Pelo exposto, o projeto irá apresentar uma plataforma de baixo custo e uma metodologia que implemente um sistema de controle *hardware-in-the-loop*.**

Como as condições da planta podem mudar pela carga térmica, é razoável que o controlador seja parametrizável em função dessa carga. Isso implica em reprojeto do controlador para a nova condição e alterar os parâmetros do compensador no microcontrolador ESP32. Para facilitar esse procedimento, pretende-se que os parâmetros

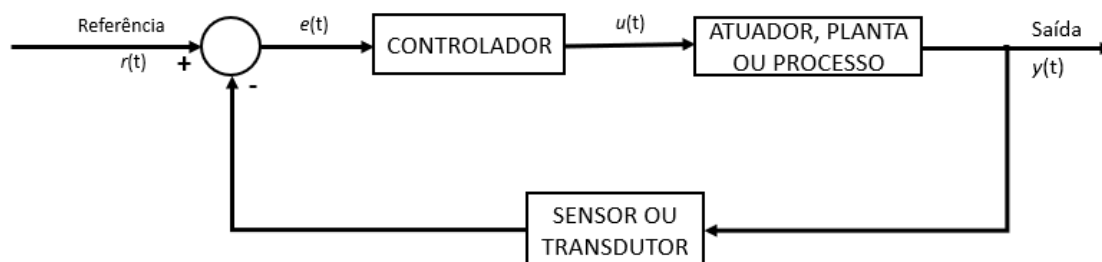
do controlador sejam enviados por um computador remoto ou smartphone via conectividade do ESP32 por meio de um broker num esquema típico de IoT.

2 REVISÃO BIBLIOGRÁFICA

2.1 CONTROLE EM MALHA FECHADA

O controle em malha fechada atua comparando a resposta do sistema com a referência de entrada, como mostra a figura a seguir.

Figura 1: Controle em Malha Fechada



Fonte: MAYA; LEONARDI, 2011, p. 8

O ramo que inicia na entrada e termina na saída possui o controlador que através do erro $e(t)$ que gera o sinal necessário para que o atuador ou planta produza uma saída que acompanhe a referência. Assim, através do ramo de realimentação o sensor fornecerá o sinal modificado para uma grandeza necessária que sirva de comparação para a referência, produzindo assim um novo erro.

Ao projetar o controlador, é necessário levar em consideração os limites físicos do atuador e fazê-lo de modo a limitar o sinal de controle $u(t)$ a não passar de um limite determinado pelo atuador.

Um sinal mais elevado que o limite do atuador não produzirá nenhum dano, pois o próprio atuador limitará o valor nele. Entretanto, o sinal na saída do controlador não corresponderá ao sinal no atuador e isso pode ser um problema para a lei de controle que poderá produzir respostas incompatíveis com o desempenho esperado. Para evitar essa situação, o sinal de saída do controlador também pode ser limitado pelo próprio controlador por meio de uma estratégia denominada de anti-windup quando há uma ação integral no controlador, como será detalhado mais adiante. (MAYA; LEONARDI, 2011)

2.2 SISTEMA DE CONTROLE HARDWARE-IN-THE-LOOP (HIL)

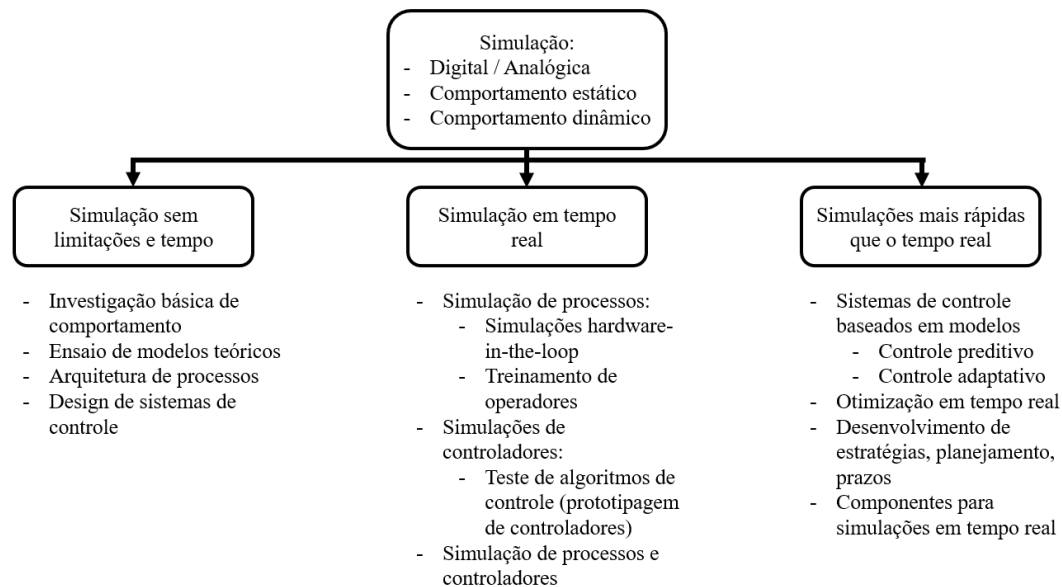
O desenvolvimento de um sistema de controle demanda certa complexidade, sendo necessário a criação de modelos computacionais para validação de rotinas e possíveis erros de modelagem. Esse processo é influenciado pelo tempo disponível de desenvolvimento, sendo este o tempo máximo para que o produto final esteja disponível no mercado e é também influenciado pelo constante acréscimo da qualidade, confiabilidade do sistema e requisitos de segurança.

Em relação às técnicas de simulação utilizadas para a criação dos modelos computacionais, elas podem ser subdivididas em:

- Simulações sem limitação de tempo;
- Simulações em tempo real;
- Simulações que são mais rápidas que o tempo real.

A figura 2 apresenta alguns tipos de simulação e exemplos de aplicação. A simulação em tempo real ocorre quando um sistema é simulado de forma que os sinais de entrada e saída dos elementos que o compõe possuem a mesma base de tempo e, se comparados com os sinais do sistema real, devem possuir valores iguais. Esse requisito pode vir a ser um problema computacional na presença de processos que possuem dinâmicas rápidas comparados com os algoritmos requeridos e com a rapidez dos cálculos. (ISERMANN; SCHAFFNIT; SINSEL, 1999)

Figura 2: Classificação dos métodos de simulação em relação à velocidade e exemplos de aplicação

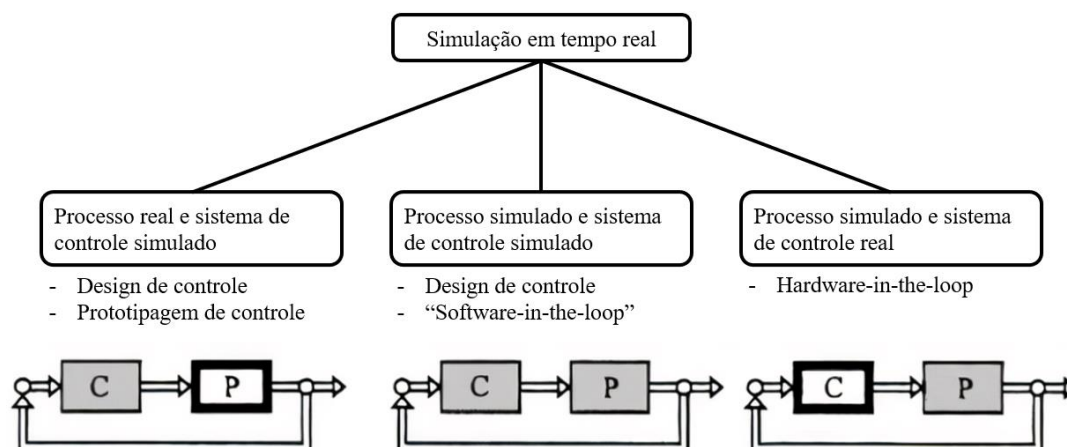


Fonte: Autor, “adaptado de” ISERMANN; SCHAFFNIT; SINSEL, 1999

A Figura 3 mostra diferentes tipos de simulações e métodos com relação a simulação em tempo real, subdivisão em que a técnica *hardware-in-the-loop* se encontra. A simulação em tempo real se baseia em simular parte do sistema e interligá-lo com a parte real. Há três cenários que devem ser conhecidos antes de implementar a simulação (ISERMANN; SCHAFFNIT; SINSEL, 1999):

- A planta real é conectada a um controlador simulado por meio de hardware;
- A planta simulada é conectada a um controlador real;
- A planta simulada é conectada a um controlador simulado em tempo real.

Figura 3: Classificação da simulação em tempo real



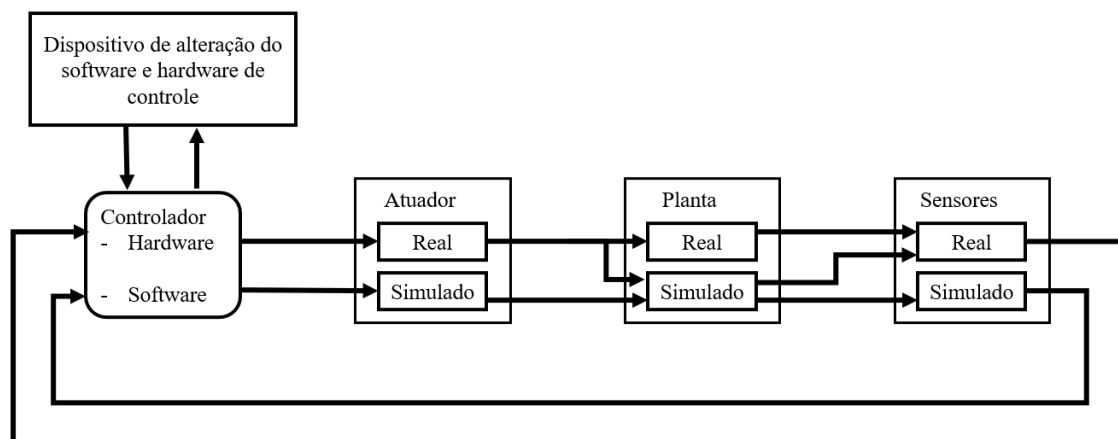
Fonte: Autor, “adaptado de” ISERMANN; SCHAFFNIT; SINSEL, 1999

Neste projeto, a simulação hardware-in-the-loop será utilizada, pois, mesmo que a planta (processo) e o controlador sejam emulados, eles estarão em dispositivos diferentes, sendo necessário uma conexão “real” entre os dois elementos que, no projeto, serão o ESP32 para o controlador e o Arduino UNO para a planta. A ligação entre os dois sistemas é física e modelando a planta com a maior precisão possível, o controlador possuirá comportamento igual ao que possuiria caso fosse utilizada uma planta real. Assim, será possível prever o comportamento que o sistema real apresentaria observando apenas o comportamento da simulação.

Os sistemas hardware-in-the-loop (HIL) são caracterizados por componentes reais e componentes emulados operando juntos dentro de um mesmo sistema. O sistema controlado constituído pelos atuadores, planta e sensores podem ser totalmente ou parcialmente simulados ou emulados, como mostra a Figura 4. O controlador, diferentemente do sistema controlado, é totalmente emulado em um hardware único que possui entradas e saídas para a planta. Assim, o sistema HIL pode ser resumido como uma técnica para o desenvolvimento de controladores embarcados, utilizando de softwares que ofereçam controle de baixo nível em hardwares que emulam o controlador e o sistema a ser controlado. Para que seja possível redefinir algumas funções do software de controle, é necessária a implementação de uma via de comunicação com o hardware do controlador. Vale discutir a diferença entre simulação e emulação. Enquanto a primeira busca modelar um elemento de maneira precisa, sendo necessário levar em conta as propriedades do elemento com o maior detalhamento possível, a segunda tem por objetivo

imitar o comportamento deste elemento, sendo possível utilizar essa emulação no lugar do elemento real. (ISERMANN; SCHAFFNIT; SINSEL, 1999)

Figura 4: Simulação hardware-in-the-loop e possíveis estruturas híbridas



Fonte: Autor, “adaptado de” ISERMANN; SCHAFFNIT; SINSEL, 1999

Algumas vantagens do sistema HIL são:

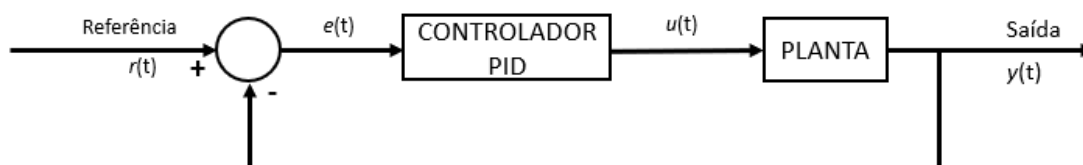
- Modelagem e teste de software do sistema sem a necessidade de implementá-lo realmente;
- Testes do controlador que no sistema real operaria em condições de ambiente extremas se tornam possíveis de serem realizados dentro da oficina;
- Testes de desempenho do sistema em condições de ambiente extremas;
- Observação dos efeitos das falhas e defeitos dos elementos que compõem o sistema;
- Repetibilidade de experimentações;
- Fácil implementação de interfaces homem-máquina;
- Redução de custo e tempo de desenvolvimento.

Os sistemas HIL possuem diversas aplicações, como testes de sistemas de voo (RONALDO BARROS DOS SANTOS et al., 2011) e de performance de veículos como aquele disponível no software CarSim[®]. Ainda existem outros softwares que permitem a modelagem e simulação de qualquer sistema, como o SIMULINK[®] e o LabView[®].

2.3 CONTROLE PID

O controlador PID é um compensador com ação proporcional, integral e derivativa. A figura abaixo mostra um sistema básico desse controle com o compensador PID. O que o difere dos demais controladores é a junção da estabilidade da configuração PD (Proporcional + Derivativo) com a eliminação do erro estacionário (offset) que a configuração PI (Proporcional + Integral) proporciona. Esse sistema possui a desvantagem da necessidade do ajuste de três parâmetros: os ganhos proporcional, integral e o derivativo, chamados respectivamente de K_P , K_I e K_D . Essa desvantagem resulta na dificuldade de sintonia desses valores. Existem algumas técnicas de sintonia para cada parcela do controlador. (BARRETO, [s.d.])

Figura 5: Sistema básico com Controle PID



Fonte: Autor “adaptado de” MAYA; LEONARDI, 2011, p. 8

2.3.1 Controle Proporcional

A ação proporcional reflete exatamente o erro, desconsiderando o tempo. O incremento do ganho proporcional K_p irá aumentar proporcionalmente o sinal de controle para um mesmo nível de erro. Na medida em que o sistema trabalha diretamente com o erro dado, a reação do sistema será mais rápida, consequentemente podendo aumentar o pico (overshoot) e podendo aumentar a oscilação de resposta do sistema. O erro de estado estacionário (offset), quando existir, é um erro impossível de ser eliminado apenas com o controle proporcional, mas pode ser reduzido conforme o aumento de K_p . (BARRETO, [s.d.])

2.3.2 Controle Integral

A ação integral trabalha, tecnicamente, com o “passado” do erro. Ao integrar o erro, essa parcela consegue verificar como o erro se comportou e irá aumentar ou reduzir o sinal de controle. Ou seja, o termo integrador soma os valores instantâneos do erro do início até o instante atual.

O controlador integral raramente possui aplicação sozinho, mas se utilizado em conjunto com o controlador proporcional, possui algumas vantagens. A maior vantagem do controlador PI em relação ao controlador P é a capacidade de eliminar o offset. Outra vantagem é a possibilidade de ajuste da velocidade de resposta, ou seja, da constante de tempo do sistema. Mas o controlador PI possui desvantagens, como eventualmente acarretar uma grande oscilação até o sistema se estabilizar por causa do “windup” da ação integral enquanto o atuador está saturado.

2.3.2.1 *Wind-up*

Wind-up é o efeito ocasionado quando a parcela integradora possui valor muito alto e não consegue acompanhar o erro de maneira rápida, pois o atuador entra em saturação e se mantém nela, independente do erro, até que a ação integradora diminua.

Por exemplo, há um drone que possui um controlador PI para determinar a altura desejada que o drone deve planar. A variável controlada é a altura do drone, a variável manipulada é a velocidade das hélices. Sendo o nível do solo igual a zero e desejando uma altura de 50 metros do nível do solo, o erro atual é de 50 metros. Então, as hélices do drone são ativadas e, por motivos de teste, o drone é preso ao chão. Nessa situação, conforme o tempo passa, o erro será constantemente 50 metros, fazendo com que a ação da parcela integradora aumente cada vez mais, chegando a valores muito altos. Agora, o drone é solto do chão e ele sobe rapidamente, passando pela marca de 50 metros, mas não para no nível desejado, pois a ação integral está elevada, mantendo a rotação das hélices. Apenas quando o drone está bem acima do nível desejado, é que a parcela integral conseguiu diminuir o suficiente para fazer o drone descer. Esse processo exige tempo, causando grande oscilação na resposta do sistema.

O alto valor do termo integral normalmente faz com que o sistema demore a estabilizar, ou seja, o atuador do sistema não irá danificar-se, pois em testes reais, os

motores do drone possuem um limite de velocidade e quando o sinal de controle chega a esse limite ou é maior que ele, significa que o sistema chegou a saturação. (USP, 2013)

2.3.2.2 Anti Wind-up

As estratégias para eliminar o efeito de wind-up da ação integral são chamadas de “anti wind-up”. Algumas das técnicas mais usadas são “clamping”, “back calculation” e “tracking”.

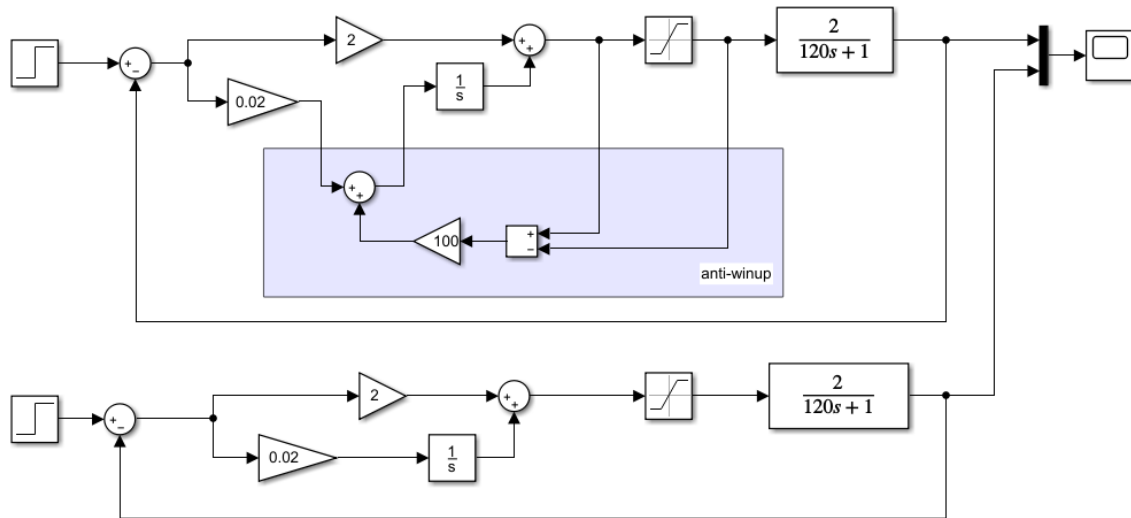
O funcionamento do clamping é simples, é implementado um circuito que congela o efeito da parcela integradora quando o sinal de controle chega a saturação e a saída do controlador e a entrada do sistema possuem o mesmo sinal. O circuito religa a ação integral quando o sinal quando uma das condições não forem atendidas.

A back calculation é uma forma matemática de lidar com o wind-up. Seu funcionamento se baseia em recalculer o termo integral quando a saída do atuador satura. O cálculo é geralmente feito dinamicamente, utilizando uma constante de tempo.

O tracking consiste em fazer a saída do controlador acompanhar um sinal de referência gerado pela diferença entre os sinais de entrada e saída do atuador. Cria-se, portanto, uma malha de controle temporária que força o valor da saída do controlador se manter no nível máximo do atuador enquanto este estiver saturado. Essa malha temporária atua na entrada da ação de controle integral e é desligada quando o atuador deixa de ficar saturado. (USP, 2013)

A figura a seguir ilustra a implementação do anti-windup via tracking.

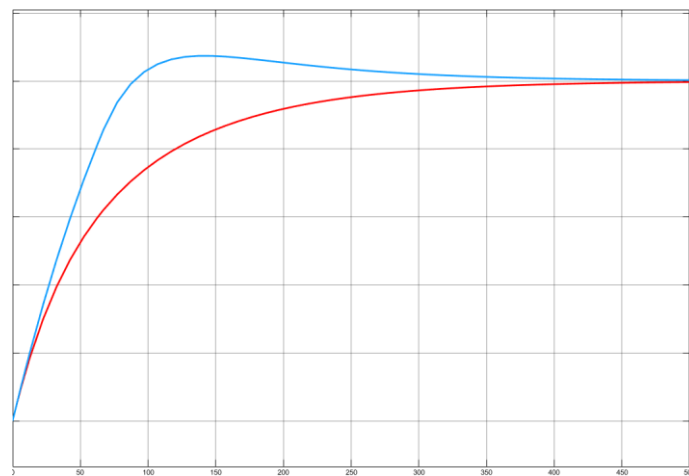
Figura 6: Implementação do Anti-Windup via Tracking



Fonte: Autor

O subsistema em destaque impõe o tracking da saída do controlador em relação ao sinal da saída do atuador quando há saturação $\{-1,1\}$. O resultado da simulação está ilustrado nos gráficos da figura a seguir. A resposta em azul corresponde ao sistema sem anti-windup que, por isso apresenta sobressinal na resposta, enquanto a saída do sistema com compensação de windup (vermelho) não apresenta sobressinal e tem um tempo de acomodação similar.

Figura 7: Efeito da ação do Anti-Windup do sistema



Fonte: Autor

2.3.3 Controle Derivativo

O controle derivativo raramente possui aplicação sozinho, apenas em conjunto com outra ação. O controle PD é muito eficiente em relação a correção, pois por possuir a parcela derivativa, tem a capacidade de “ver” o futuro, ou seja, o controle não trabalha com o erro em si, mas com a variação do erro. Adicionando a parcela derivativa, o offset, se existir, não será removido, mas a estabilidade é melhorada.

A aplicação da derivada, em teoria pode funcionar, mas na realidade, pode haver um grande problema, chamada ruído. Esses ruídos sempre aparecem em qualquer sistema e possuem muitas razões para ocorrer, como defeitos nos aparelhos, método de implementação, condições do ambiente, e principalmente pelo sensor.

O ruído pode causar grandes problemas para o PID, devido ao termo derivativo. A parcela derivativa amplifica ruídos de altas frequências, pois quanto maior a frequência, maior será a amplitude da derivada.

Uma solução para o problema é colocar um filtro passa baixa (low pass filter) em série com o termo derivativo. Isso faz com que ruídos de baixas frequências passem junto com o sinal de erro, pois estas não interferem no sinal, mas ruídos de alta frequência são atenuados (filtrados), resultando em um sinal de acordo com o esperado, atenuando o problema. (MESSNER; HILL; TAYLOR, [s.d.])

O quadro a seguir resume os efeitos típicos de cada ação de controle PID.

Quadro 1: Efeito do aumento das ações de controle

| Efeitos | Tempo de Subida | Sobressinal | Tempo de Acomodamento | Erro de Estado Estacionário |
|----------------|------------------------|--------------------|------------------------------|------------------------------------|
| Kp | Atenuação | Aumento | Pouca variação | Atenuação |
| Ki | Atenuação | Aumento | Aumento | Atenuação |
| Kd | Pouca variação | Atenuação | Atenuação | Sem efeito |

Fonte: Autor “adaptado de” MESSNER; HILL; TAYLOR, [s.d.]

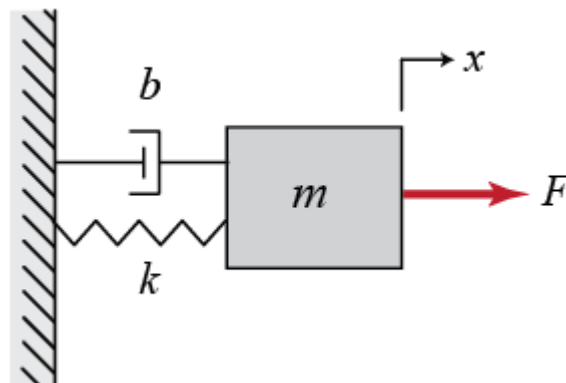
2.3.4 Exemplo de Aplicação

O software MATLAB em conjunto com o Simulink é uma ótima opção para a simulação de sistemas de controle e, para visualizar os efeitos de cada tipo de controlador, os exemplos a seguir foram retirados de um website com foco em tutoriais sobre o uso

dos softwares citados acima no desenvolvimento de compensadores. (MESSNER; HILL; TAYLOR, [s.d.]

O sistema massa-mola-amortecedor abaixo foi utilizado para simular seu comportamento no MATLAB.

Figura 8: Sistema Massa-Mola Amortecido



Fonte: MESSNER; HILL; TAYLOR, [s.d.]

A equação do movimento é dada por:

$$m\ddot{x} + b\dot{x} + kx = F$$

Aplicando a transformada de Laplace:

$$ms^2X(s) + bsX(s) + kX(s) = F(s)$$

A função de transferência entre a força aplicada (F) e o deslocamento (x) resulta em:

$$\frac{X(s)}{F(s)} = \frac{1}{ms^2 + bs + k}$$

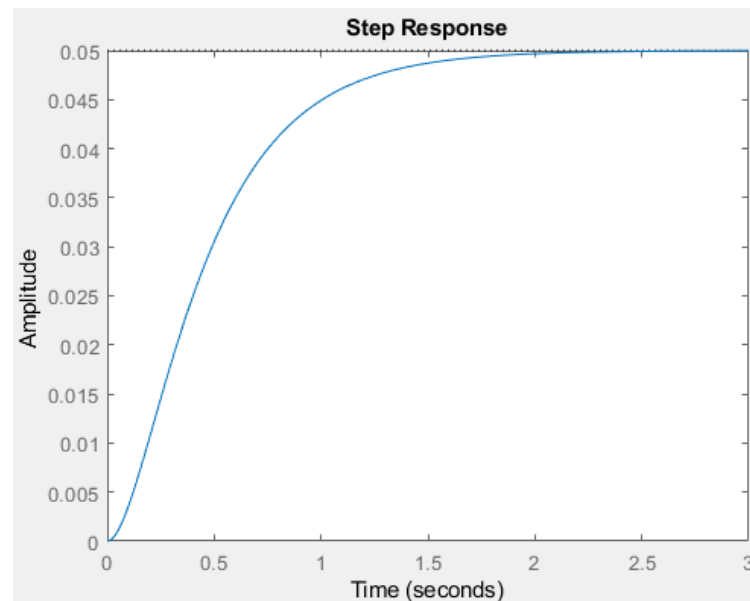
Considerando:

- $m = 1 \text{ kg}$
- $b = 10 \text{ N s/m}$
- $k = 20 \text{ N/m}$
- $F = 1 \text{ N}$

O objetivo é parametrizar, ou seja, sintonizar os tipos de controladores para um tempo de subida pequeno, um overshoot mínimo ou nenhum overshoot e anular o offset.

A resposta ao degrau do sistema descrito acima, sem nenhum tipo de controle, está representada no gráfico abaixo.

Figura 9: Resposta ao degrau sem compensador



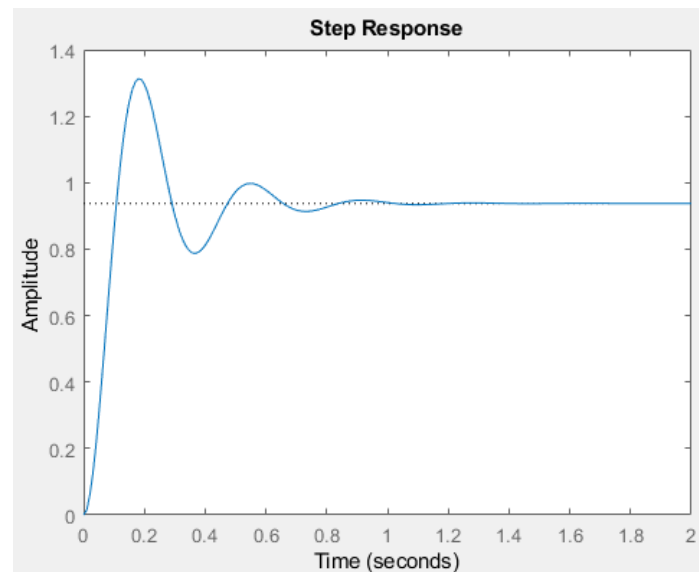
Fonte: Autor

O ganho em malha aberta do sistema é 0.05, logo este é o valor final de resposta ao degrau unitário. O erro de estado estacionário está em 0.95, o tempo de subida é cerca de 1 segundo e o tempo de acomodação está em 1.5 segundos. O controlador PID pode melhorar essas características, mas antes, será mostrado os efeitos de cada controlador. (MESSNER; HILL; TAYLOR, [s.d.])

2.3.4.1 Controlador Proporcional

Os efeitos esperados do controlador proporcional são a redução do tempo de subida e a diminuição do sobressinal, além da redução do erro de estado estacionário. A figura abaixo mostra a resposta obtida como o controle proporcional.

Figura 10: Resposta ao degrau com Compensador Proporcional



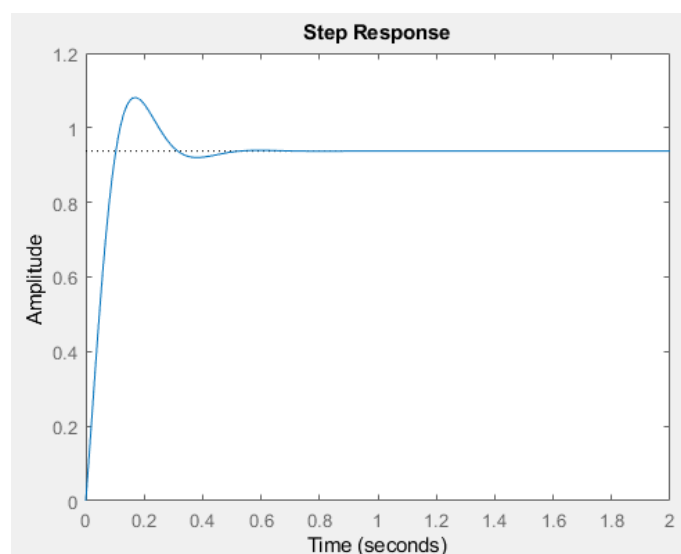
Fonte: Autor

A partir da Figura 10, observa-se que o erro de estado estacionário diminuiu bastante, houve um sobressinal e o tempo de acomodação diminuiu uma pequena parcela.

2.3.4.2 Controlador Proporcional+Derivativo

Os efeitos esperados do Controlador P+D são a redução do sobressinal e a redução do tempo de acomodação. A resposta do sistema ao degrau de entrada é mostrada abaixo.

Figura 11: Resposta ao degrau com Copensador PD



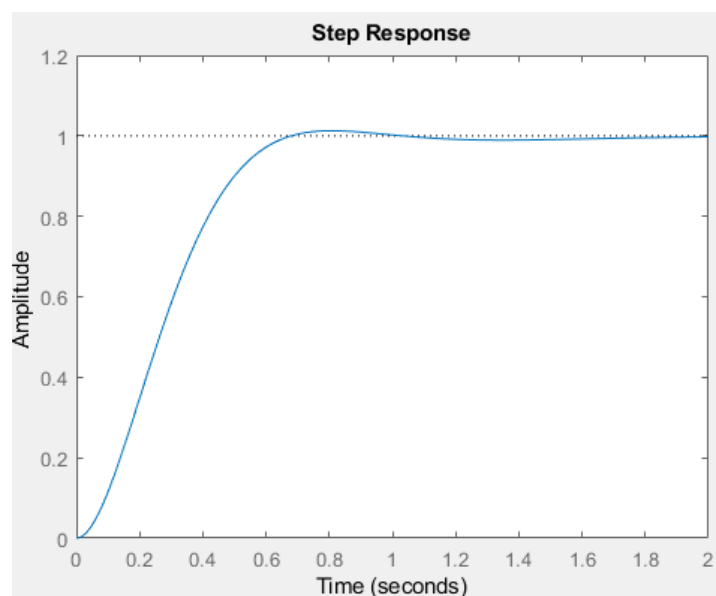
Fonte: Autor

A partir do gráfico, é possível observar a redução do sobressinal e a redução do tempo de acomodação, assim como esperado.

2.3.4.3 Controlador Proporcional+Integral

Efeitos esperados são a diminuição do tempo de subida, o aumento do sobressinal e do tempo de acomodação e o erro de estado estacionário zerado.

Figura 12: Resposta ao degrau com Compensador PI



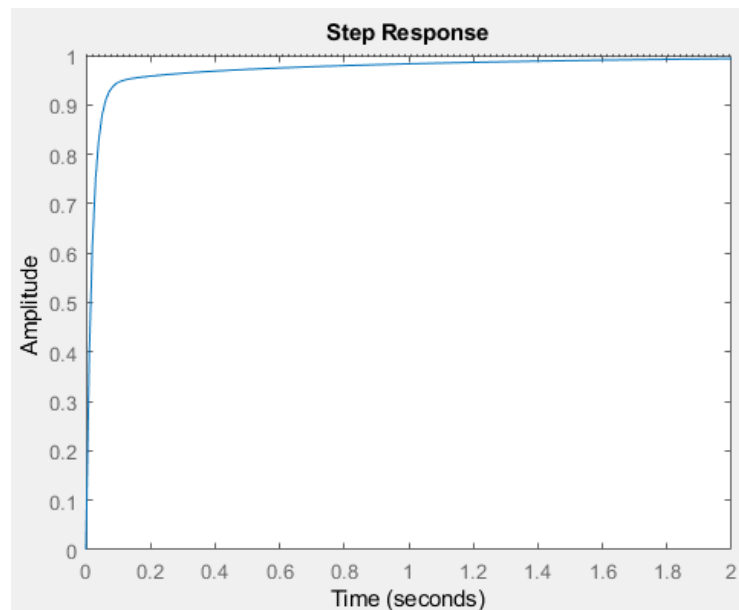
Fonte: Autor

O K_P foi reduzido pois ambos K_P e K_I reduzem o tempo de subida e aumentam o sobressinal. No gráfico, é possível observar que em relação a reposta sem controle, o tempo de subida diminuiu, o sobressinal e o tempo de acomodação aumentou e o erro de estado estacionário foi eliminado.

2.3.4.4 Controlador PID

A sintonização do PID pode ser feita através de iterações ou utilizando o comando “pidTuner” que o MATLAB oferece. Através do programa, é possível mudar a resposta do sinal variando as constantes no domínio da frequência e do tempo. O exemplo mostrado é simples, logo não é necessário utilizar o “pidTuner”.

Figura 13: Resposta ao degrau com Compensador PID



Fonte: Autor

Assim, através da sintonização dos parâmetros do PID, obtém-se uma resposta sem sobressinal, tempo de subida baixo, tempo de acomodação reduzido e erro de estado estacionário eliminado.

2.4 CONTROLE SMC

O controle por modos deslizantes (Sliding Mode Control – SMC) evoluíram de um trabalho na Rússia no começo da década de 1960. É uma técnica que aborda os distúrbios e incertezas que uma planta a ser controlada e utiliza uma lei de controle que “desliza” seus estados sobre trajetórias se mantendo dentro de uma região desejada no espaço de estados. Essa região em que o estado desliza é chamada de “superfície de escorregamento” ou “superfície de deslizamento”. Com esse controlador é possível incluir explicitamente incertezas do modelo no seu projeto. Sua lei de controle não linear permite resolver problemas que os compensadores lineares não conseguem. A parametrização da lei de controle é simples, diferente de um controlador linear como o PID.

A principal característica dos modos deslizantes é a chamada propriedade de invariância, ou seja, a capacidade do sistema em não variar seu desempenho ao ser submetido a incertezas ou distúrbios uma vez que a superfície de deslizamento seja

alcançada. No modo deslizante convencional a trajetória do sistema se resume a uma superfície de deslizamento $s(x) = 0$. O conceito foi generalizado por Levant (1993) com a introdução dos modos deslizantes de ordem superior (High Order Sliding Modes – HOSM), onde há mais superfícies de escorregamentos e pode ser descrita como $s(x) = \dot{s}(x) = \dots = s^{(r-1)}(x) = 0$, sendo r a ordem de deslizamento.

Resumidamente, os modos deslizantes são uma classe de sistemas onde a lei de controle muda durante o processo de controle de acordo com algumas leis definidas que dependem do estado do sistema. Essa classe de controladores é denominada por estrutura variável.

O exemplo abaixo mostra a construção do sistema de estrutura variável, para isso, considere o duplo integrador (2.1) e o efeito da lei de controle linear (2.2) abaixo.

$$\ddot{y}(t) = u(t) \quad (2.1)$$

$$u(t) = -ky(t) \quad (2.2)$$

Sendo k um escalar positivo. Um modo de analisar a equação (2.2) em malha fechada é por um diagrama no plano de fase entre a velocidade e a posição. Substituindo a lei de controle na equação (2.1) e multiplicando ambos os lados por \dot{y} , temos que:

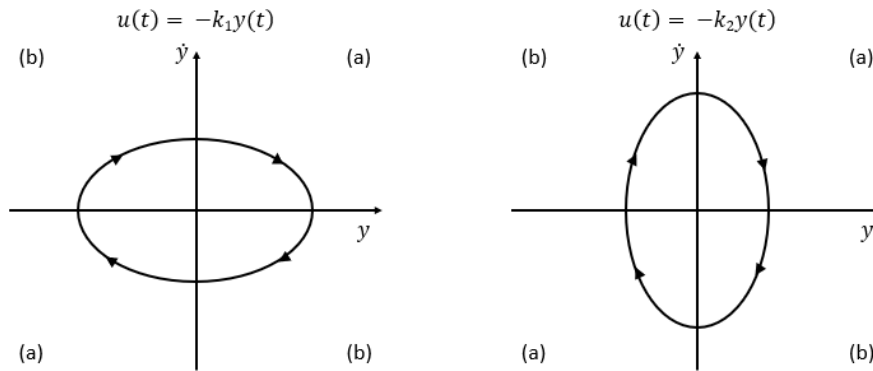
$$\dot{y}\ddot{y} = -k\dot{y}y \quad (2.3)$$

Integrando a equação, obtém-se a seguinte relação entre velocidade e posição:

$$\dot{y}^2 + ky^2 = c \quad (2.4)$$

Onde c é a constante de integração positiva derivada das condições iniciais do sistema. Supondo $k = 1$, a equação (2.4) representa um círculo com centro na origem e raio \sqrt{c} , ou seja, o gráfico de \dot{y} por y é uma elipse que depende das condições iniciais como mostrado na Figura 14. Note-se que a lei de controle dada acima não é interessante do ponto de vista de controle, pois as variáveis y e \dot{y} não se movem para a origem.

Figura 14: Diagrama de Fase da Lei de Controle



Fonte: EDWARDS; SPURGEON, 1998, p. 2

Agora, desconsiderando a lei de controle anterior e tomando a estrutura abaixo como a nova lei de controle:

$$u(t) = \begin{cases} -k_1 y(t) & \text{se } y\dot{y} < 0 \\ -k_2 y(t) & \text{se } y\dot{y} > 0 \end{cases} \quad (2.5)$$

Sendo $0 < k_1 < 1 < k_2$. O plano (y, \dot{y}) é dividido pela lei de controle em quatro quadrantes separados por eixos como mostrado na Figura 14. Com a comutação das leis de controle em função do quadrante, o sistema torna-se estável. A lei de controle $u = -k_2 y$ irá agir nos quadrantes do plano nomeados de (a). Nessa região, a distância da origem até os pontos do diagrama de fase diminui ao longo da trajetória do sistema. Da mesma forma, na região (b) quando a lei de controle $u = -k_1 y$ está ativa, a distância da origem até os pontos do diagrama também diminui. Assim, o sistema acaba por mostrar que está indo de encontro a origem, ou seja, o sistema de controle de estrutura variável (Variable Structure Control System – VSCS) se tornou estável. Entretanto, ele inda não é chamado de controle SMC, pois não é perceptível a superfície de escorregamento, a qual será definida a seguir.

O diagrama de fase para o sistema de malha fechada sob a lei de controle u é obtido emendando as regiões apropriadas dos dois diagramas de fase da Figura 14, como mostra a Figura 15. Considere a função (2.6) que pode ser interpretada como uma métrica de energia

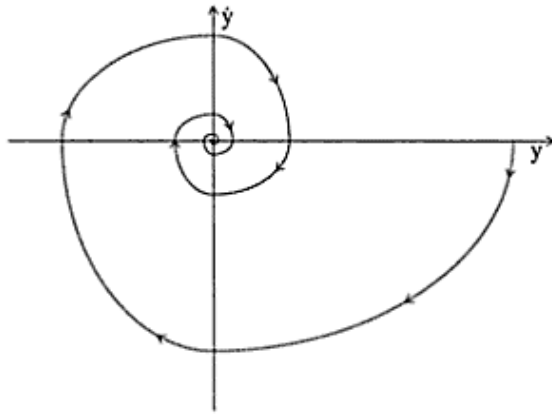
$$V(y, \dot{y}) = y^2 + \dot{y}^2 \quad (2.6)$$

A partir do teorema de Pitágoras, a função representa a distância do ponto (y, \dot{y}) até a origem no plano de fase. A derivada em função do tempo de $V(y, \dot{y})$ ao longo da trajetória de malha fechada é dada por:

$$\begin{aligned}\dot{V} &= 2\dot{y}y + 2\ddot{y}\dot{y} \\ &= 2\dot{y}(y + u) = \begin{cases} 2y\dot{y}(1 - k_1) & \text{se } y\dot{y} < 0 \\ 2y\dot{y}(1 - k_2) & \text{se } y\dot{y} > 0 \end{cases} \end{aligned}$$

Desse modo, o seguinte diagrama de fase é obtido:

Figura 15: Diagrama de Fase do Sistema de Estrutura Variável



Fonte: EDWARDS; SPURGEON, 1998, p. 3

Considerando agora uma lei de controle do seguinte tipo:

$$u(t) = \begin{cases} -1 & \text{se } s(y, \dot{y}) > 0 \\ 1 & \text{se } s(y, \dot{y}) < 0 \end{cases} \quad (2.7)$$

Onde a função de chaveamento é dada por:

$$s(y, \dot{y}) = my + \dot{y} \quad (2.8)$$

Onde m é um positivo escalar. A lei de chaveamento impõe a superfície de escorregamento que objetivos estados devem percorrer. A equação (2.7) pode ser descrita como:

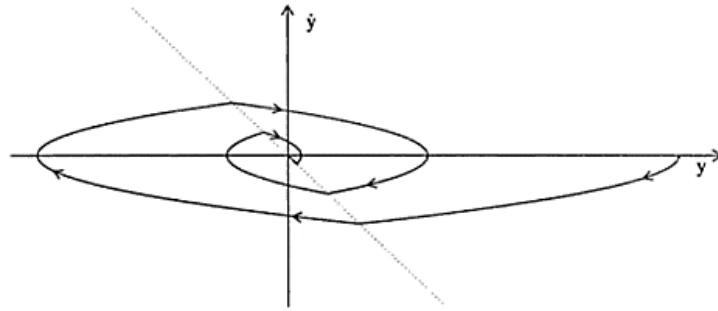
$$u(t) = -\text{sgn}(s(t)) \quad (2.9)$$

Onde $\text{sgn}(\cdot)$ é chamado de *signum*, a função sinal. Essa função possui como propriedade:

$$s \text{sgn}(s) = |s| \quad (2.10)$$

A expressão dada por (2.7) é usada para controlar o duplo integrador (2.1) e o plano de fase é mostrado na Figura 16.

Figura 16: Diagrama de Fase para valores de \dot{y} altos



Fonte: EDWARDS; SPURGEON, 1998, p. 4

A linha pontilhada representa um conjunto de pontos para que $s(y, \dot{y}) = 0$. A prova de que ocorre tal estabilidade é mostrada abaixo.

Seja a seguinte Função de Lyapunov:

$$V = \frac{1}{2} s^2 \quad (2.11)$$

$$\dot{V} = 2 \left(\frac{1}{2} s \right) (\dot{s}) = s \dot{s} \quad (2.12)$$

Para obter estabilidade assintótica, \dot{V} deve ser negativo, então:

$$\dot{V} = s \dot{s} < 0 \quad (2.13)$$

$$\dot{s} = m \dot{y} + \ddot{y} \quad (2.14)$$

Logo:

$$s \dot{s} = s(m \dot{y} - \text{sgn}(s)) < |s|(m|\dot{y}| - 1) < 0 \quad (2.15)$$

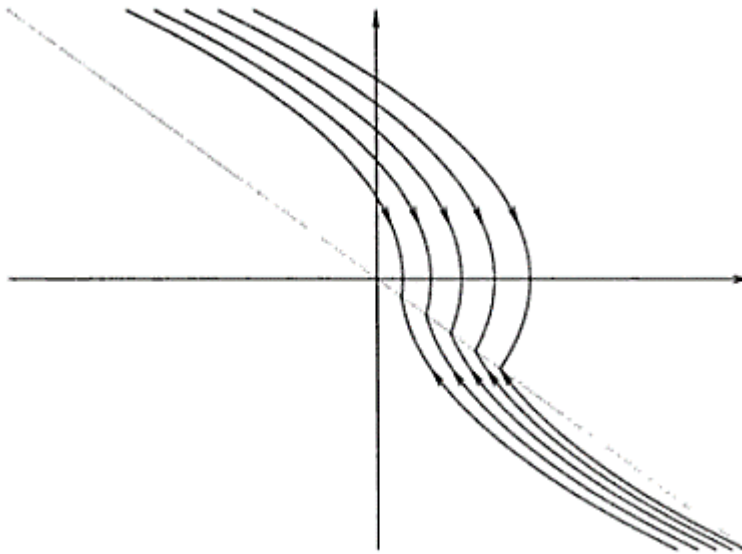
Ou equivalentemente:

$$\lim_{s \rightarrow 0^+} \dot{s} < 0 \quad \text{e} \quad \lim_{s \rightarrow 0^-} \dot{s} > 0 \quad (2.16)$$

Consequentemente, quando $m|\dot{y}| < 1$ as trajetórias do sistema em cada lado da linha apontam para ela mesma.

Isso pode ser visto na Figura 17, que mostra diferentes planos de fase interceptando a superfície L de condições iniciais diferentes.

Figura 17: Planos de fase interceptando a superfície L de condições iniciais diferentes



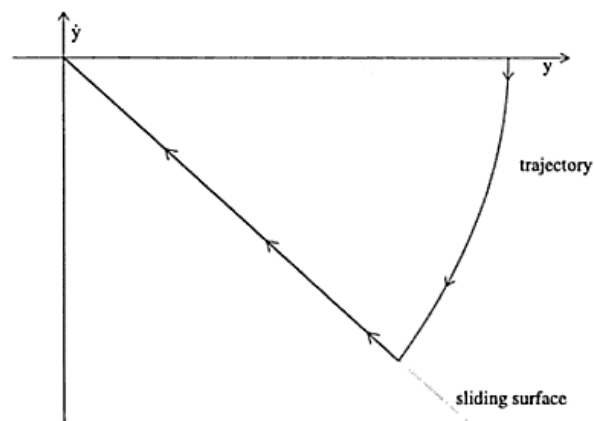
Fonte: EDWARDS; SPURGEON, 1998, p. 4

As trajetórias de estado ao se prenderem na superfície de deslizamento ($s = 0$), ocorre um chaveamento de alta frequência chamado de “chattering” e a dinâmica resultante pode ser descrita como:

$$\dot{y}(t) = -my(t) \quad (2.17)$$

Isso representa uma dinâmica de primeira ordem e as trajetórias irão deslizar ao longo da linha L_s para a origem (Figura 18).

Figura 18: Diagrama de fase de um movimento deslizante



Fonte: EDWARDS; SPURGEON, 1998, p. 5

Esse comportamento dinâmico é descrito como um movimento deslizante ideal e a linha L_s é chamada de superfície de deslizamento. Durante o movimento deslizante, o sistema se comporta como um sistema de ordem reduzida que é aparentemente independente do controle e possui a seguinte condição:

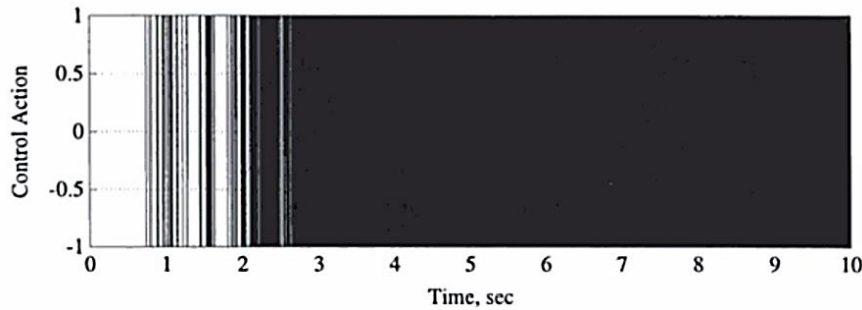
$$s\dot{s} < 0 \quad (2.18)$$

Essas condições são referidas como condições de acessibilidade. Deste modo, em termos de modelagem do SMC, a escolha da função de chaveamento, representada nessa situação pelo parâmetro m , controla o desempenho da resposta enquanto que a lei de controle é designada para garantir que a condição de alcançabilidade (2.14) seja satisfeita. Ou seja, a condição de alcançabilidade está satisfeita somente no domínio do plano de fase em que:

$$m|\dot{y}| < 1 \quad (2.19)$$

A figura a seguir mostra a ação de lei de controle quando a trajetória encontra a superfície de deslizamento. É possível observar que o chaveamento em alta frequência, ou seja, o chattering começa depois que a trajetória encontra a superfície.

Figura 19: Ação de Controle



Fonte: EDWARDS; SPURGEON, 1998, p. 6

Suponha que em um instante t_s a superfície de chaveamento é alcançada e um movimento de deslizamento ideal começa. Resulta que a função de chaveamento satisfaz $s(t) = 0$ para todos $t > t_s$, que por sua vez implica que $\dot{s}(t) = 0$ para todos $t \geq t_s$.

Assim:

$$s(t) = my(t) + \dot{y}(t) = 0 \quad (2.20)$$

$$\dot{s} = m\dot{y} + \ddot{y} = 0 \quad (2.21)$$

Utilizando (2.1) em (2.21):

$$\dot{s}(t) = m\dot{y}(t) + u(t) \quad (2.22)$$

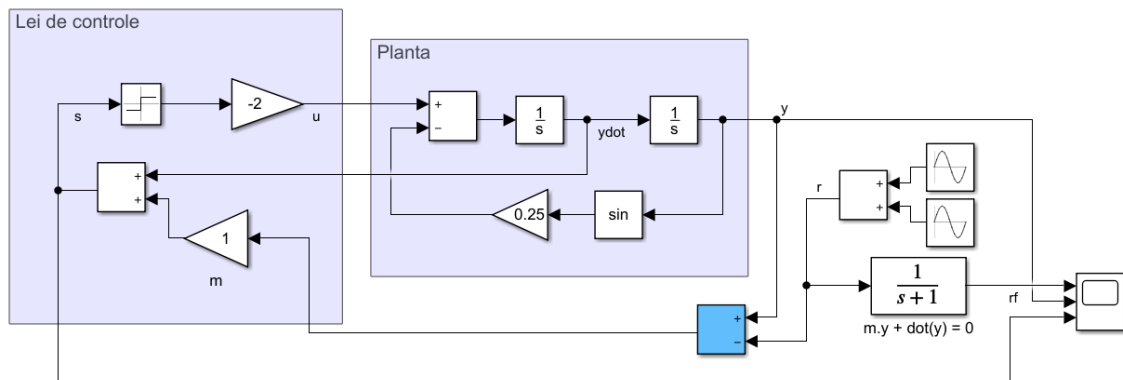
Isso implica que:

$$u(t) = -m\dot{y}(t) \quad (t \geq t_s) \quad (2.23)$$

Essa lei de controle é referida como a ação de controle equivalente. Isso não é o sinal de controle que é propriamente aplicado na planta, mas pode ser visto como um sinal de controle médio aplicado. (EDWARDS; SPURGEON, 1998)

O diagrama de blocos da figura abaixo ilustra como o problema original SMC pode ser adaptado para o acompanhamento do sinal de referência por meio da adição de um somador à saída de planta.

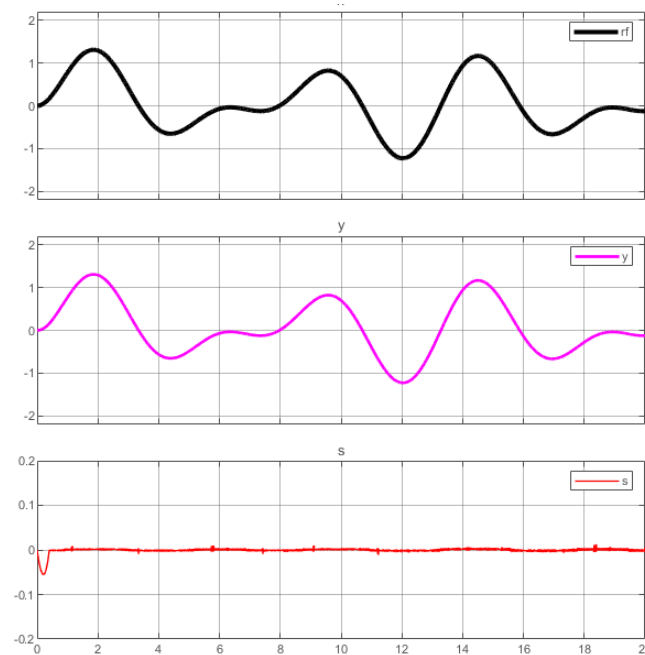
Figura 20: Exemplo SMC



Fonte: Autor

Nesse exemplo a planta é não linear e a superfície de escorregamento corresponde a um sistema de primeira ordem cuja função de transferência associada tem um ganho unitário e uma constante de tempo também unitária. O resultado da simulação está ilustrado pelas curvas da figura a seguir. Note que mesmo com uma referência $r(t)$ composta por dois sinais senoidais, a malha fechada responde (y) do mesmo modo que uma função de transferência (rf) de igual dinâmica da superfície de escorregamento. Isso pode ser comprovado pelo sinal s que rapidamente é anulado e se mantém assim durante toda a simulação

Figura 21: Resposta do Sistema com SMC



Fonte: Autor

2.5 INTERNET DAS COISAS

Internet das Coisas, também chamada de IoT (Internet of Things) é a definição dada para a tecnologia que conecta pessoas ou máquinas através de uma rede sem fio, utilizando programas, aparelhos eletrônicos e sensores que possibilitam a troca de informações entre os elementos conectados à rede, ou seja, é uma rede de “coisas” físicas conectadas a uma rede e que possuem a habilidade de coletar e trocar dados. (GURU99, [s.d.])

A ideia de conectar dispositivos data de 1970, desde então, com o avanço de áreas como a microeletrônica, sensoriamento e comunicação, a ideia foi adquirindo força quando em 1990 John Romkey conseguiu ligar e desligar uma torradeira via internet, desafio feito pelo presidente da INTEROP (feira anual de tecnologia da informação organizada pela empresa britânica UBM). Em 1999, Kevin Ashton propôs o termo “Internet of Things” no seu trabalho na P&G, termo que foi muito aceito. Após várias conferências pelo mundo inteiro, em 2008 o termo foi mundialmente reconhecido. (GURU99, [s.d.]; MANCINI, 2018)

O objetivo maioritário do IoT é garantir maior facilidade ao gerenciar dispositivos, trocar dados, tornando fácil integrar sistemas autônomos, tornando nossas vidas melhores

e até mais eficazes. O IoT não se limita a apenas conectar dispositivos. Essa tecnologia permite, principalmente, otimizar gastos em tempo real, de acordo com dados adquiridos automaticamente. (GURU99, [s.d.]; MANCINI, 2018)

O IoT começa com a interface com o usuário, que podem ser os celulares, computadores ou painéis de controle. Os principais componentes que formam o IoT são:

- Sensores: dispositivos que coletam dados, podem ser sensores de calor, presença, pressão, imagem, luz, entre vários outros que existem, todos podem ser conectados com dispositivos controladores;
- Controladores: os controladores são os dispositivos que fazem a conexão entre a internet e os atuadores. São eles que coletam e enviam as informações dos sensores, podem tratar os dados ou não, geralmente a ferramenta que processa os dados é o software que administra a rede;
- Conectividade: a conexão entre todos os elementos pode ser feita de maneiras diversas, através de WI-FI, Bluetooth, satélite;
- Processamento de dados: geralmente é um software que processa todos os dados que chegam à rede e trata eles, seja lendo uma temperatura e ajustando se necessário ou até mesmo identificando pessoas ou objetos através de uma imagem.
- Interface de usuário: as informações necessárias devem ser exibidas para o usuário, preferencialmente de forma limpa e fácil de visualizar, principalmente em sistemas domésticos. A interface possibilita não só a visualização do que está ocorrendo no sistema, como também a possibilidade do usuário tomar ações, caso seja necessário.

A partir desses elementos, a IoT está pronta para ser configurada e usada, sendo necessário reparos apenas em casos de problemas mecânicos ou atualização de softwares. (GURU99, [s.d.])

Há muitas áreas que a IoT pode ser usada, as mais conhecidas são as “Casas Inteligentes” e “Transportes Inteligentes”. A aplicação em domicílio é amplamente estudada, pois facilita a vida das pessoas, principalmente das pessoas portadoras de deficiência. Uma vez que é possível controlar iluminação, aparelhos, janelas, todos através de um tablet, a vida se torna muito mais cômoda. (CLARK, 2016; RANGER, 2020)

A IoT pode também ser aplicada aos transportes, para conversar com automóveis próximos, fazer autoavaliações e mandar essas avaliações para a montadora, possibilita desligar e ligar faróis através do celular e, no caso de carros autônomos, é possível chamar eles para sua localização pela chave ou celular. (CLARK, 2016)

Esses foram exemplos de aplicações que uma pessoa comum pode vir a possuir acesso, mas há também a IoT industrial, onde todos os maquinários, linhas de produção e produtos são conectados na nuvem, a qual realiza diversas avaliações para gerenciar a fábrica ou indústria. As máquinas podem possuir IoT próprias, uma vez que podem se conectar com a máquina ao lado para informar quando alguma peça ou produto está pronto. (RANGER, 2020)

Essa tecnologia não é perfeita, portanto há alguns desafios que estão sendo pesquisados para serem superados. É possível citar a falta de testes do sistema, softwares pesados para o tratamento de dados além da possibilidade de haver grandes volumes de dados e o sistema não dar conta. A necessidade de possuir uma fonte de energia para funcionar é um problema que qualquer sistema possui, mas que pode ser amenizado com o uso de energia solar ou eólica.

Há alguns problemas cruciais com o IoT, problemas que devem ser cautelosamente revistos e remediados para que não aconteça, por exemplo, a segurança dos dados. Muitas vezes, em Smart Homes, pode haver microfones e câmeras coletando tudo o que é dito e feito na casa sempre. Esses dados podem ser roubados, o sistema pode ser invadido e os hackers podem mexer com os dispositivos, como um forno. Para isso, é necessário utilizar profissionais e programas que cuidam da segurança do sistema, pois a privacidade do usuário deve ser sempre resguardada. No ambiente industrial, o vazamento de dados e infiltração no sistema pode causar danos enormes aos funcionários, uma vez que pode haver risco de desregulação dos parâmetros do maquinário e causar acidentes. (GURU99, [s.d.]; RANGER, 2020)

Assim com toda tecnologia, o IoT possui suas vantagens e desvantagens. Algumas das vantagens são:

- Otimização do trabalho: a otimização é uma das características mais marcantes do IoT, pois possibilita a análise de vários dados de uma só vez, sem a necessidade de analisar caso a caso, como por exemplo, as condições de cada máquina de um sistema;
- Aquisição de dados melhorada: o IoT consegue coletar todos os dados necessários e organizá-los de maneira rápida e eficaz;

- Economia: como os dados são adquiridos em tempo real, a nuvem consegue tratar essas informações e imediatamente envia uma resposta com as ordens sobre o que deve ser feito, como manejar recursos, acionar atuadores, entre outras funções. Assim, o sistema consegue detectar problemas e resolvê-los no mesmo instante. (RANGER, 2020)

As desvantagens são, como citado anteriormente, a possibilidade de haver falhas na segurança da rede, ocasionando em invasões e roubo de dados. Sua manutenção não é fácil, uma vez que atualmente cada dispositivo possui um sistema proprietário, o que torna a implementação do IoT uma tarefa complexa e difícil. (GURU99, [s.d.])

Atualmente, a IoT está sendo discutida por todo o mundo por se tratar de uma tecnologia extremamente nova e que ainda não foi alcançada totalmente. Mas, é possível imaginar como esse termo irá afetar nosso futuro, por exemplo, imagine que você tenha um compromisso marcado na agenda do seu celular e, utilizando a IoT, o seu carro se comunique com o celular e calcule a melhor rota para o destino. Ou então, quando algum equipamento esteja com seus recursos comprometidos, ele mesmo peça recursos novos ou então agende uma manutenção necessária. Há muitas possibilidades do uso do IoT e certamente irá afetar nossas vidas. (MORGAN, 2014)

2.5.1 Protocolo MQTT

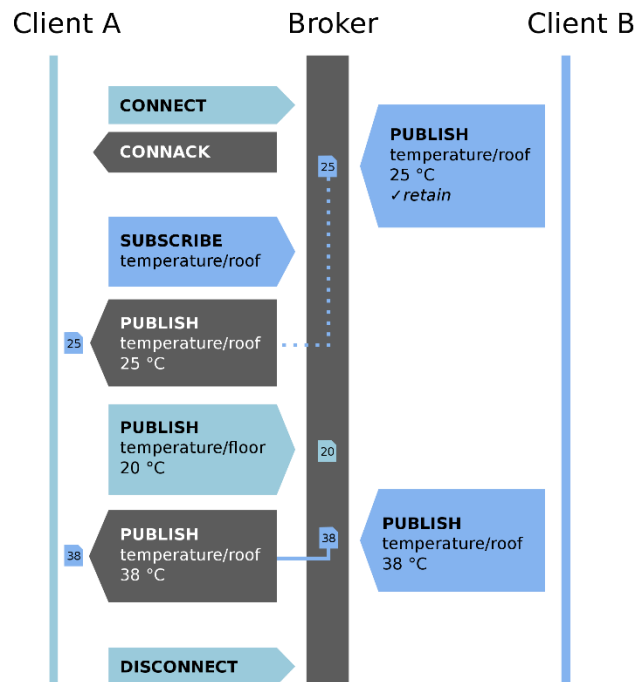
A comunicação entre qualquer máquina conectada à internet deve ser normalizada e regulamentada para que assim todas as máquinas se comuniquem da mesma forma. Para isso, foram criados os protocolos. (L., 2018)

Este projeto utilizará o protocolo MQTT (Message Queuing Telemetry Transport), um protocolo de comunicação que atua na presença de um cliente e um servidor, onde o primeiro pode enviar e receber dados e o segundo faz administrar os dados a serem recebidos e enviados. O protocolo foi criado pela IBM devido a necessidade de um protocolo simples e leve para fazer a comunicação entre máquinas onde os dados a serem transmitidos não possuíssem tamanho elevados, por exemplo dados de pequenos sensores. Devido a sua rápida taxa de transmissão para informações leves e baixo consumo de memória, o MQTT se integrou bem com as aplicações de IoT.

O protocolo funciona com o sistema Publish-Subscribe. A comunicação é feita de maneira hierárquica, onde a informação é organizada em tópicos e apenas os clientes que

estão inscritos em determinado tópico poderão receber as informações deste. O diagrama abaixo representa a comunicação MQTT.

Figura 22: Comunicação MQTT



Fonte: NERI; LOMBA; BULHÕES, [s.d.]

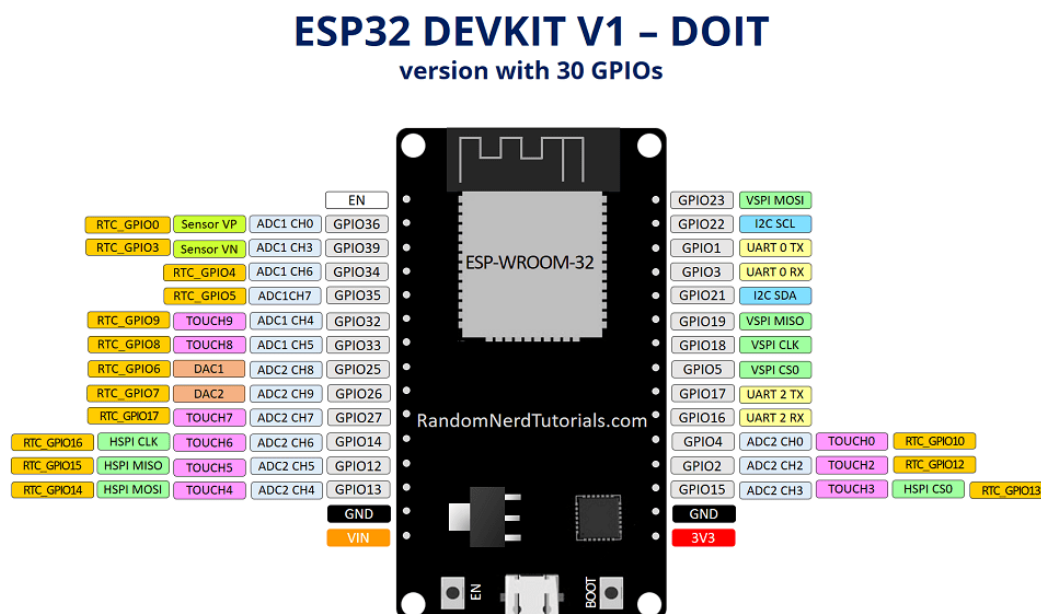
A partir da Figura 22, é possível observar que o servidor é chamado de broker, ele é quem faz o tratamento dos dados e os envia para os clientes que estão inscritos no tópico onde a mensagem foi publicada. No exemplo, o Cliente A é o “Subscriber” e o Cliente B é o “Publisher”. Para este projeto, será utilizado o broker gratuito HiveMQ, a partir dele é possível observar todo o fluxo de dados do servidor. (NERI; LOMBA; BULHÕES, [s.d.])

2.5.2 Microcontrolador ESP32

O microcontrolador utilizado no projeto **para realizar a função de controlador** será o ESP32 modelo Devikit v1, criado pela DOIT. Atualmente o dispositivo é amplamente conhecido devido ao seu baixo custo, alto poder de processamento e baixo consumo de energia. Possui conectividade Wi-Fi e Bluetooth embutido, assim como sensores capacitivos (touch) e um sensor Hall. A maioria de seus pinos podem ser usados como GPIO's (pinos de uso geral). O microcontrolador conta 16 canais de saída PWM, com

resolução de até 16bits e clock de 80MHz e quatro timers de uso geral. (ESPRESSIF SYSTEMS, 2020)

Figura 23: Pinagem ESP32



Fonte: SANTOS S.; SANTOS R., [s.d.]

O projeto irá usá-lo para a implementação dos controladores, comunicação MQTT e aquisição dos dados de temperatura da planta.

2.5.2.1 Sinal PWM

A modulação por largura de pulso (Pulse Width Modulation – PWM) é uma técnica utilizada para controlar dispositivos analógicos através de sinais digitais. O PWM funciona aplicando pulsos no sinal, deixando-o em nível lógico alto por um tempo e a uma frequência pré-determinada, assim, a tensão média que o dispositivo apresentará em sua saída PWM irá mudar de acordo com essas configurações. (HEATH, 2017)

A frequência é configurada dependendo do funcionamento desejado. Por exemplo, para controlar a luminosidade de um LED, ao utilizar uma frequência muito baixa, será possível observar os momentos em que o sinal está em nível lógico alto ou baixo, simplesmente devido a frequência estar baixa e consequentemente o período estar longo. Para obter uma luminosidade onde o LED aparente estar completamente aceso, a frequência deve ser aumentada.

O nível de luminosidade em si é controlado através do Duty Cycle que, como explicado anteriormente, define o tempo de permanência em nível lógico alto do sinal. Esse parâmetro é um valor calculado de acordo com a resolução do sinal. Caso o sinal possua uma resolução de 10 bits, o Duty Cycle será um valor numérico que varia de 0 a 1023.

O ESP32 possui dezesseis canais para utilizar o PWM, ou seja, são dezesseis sinais PWM independentes para configuração. A resolução pode variar de 1 a 16 bits, mas em contrapartida, a frequência máxima diminui quando maior a resolução. Devido ao microcontrolador funcionar a um clock de 80MHz, a relação entre frequência máxima do PWM e resolução é dada por:

$$Fm = \frac{80 * 10^6}{2^n}$$

Onde Fm representa a frequência máxima e n o número de bits da resolução. Sendo assim, o ESP32 apresenta um alcance de frequência variando de aproximadamente 1220Hz para uma resolução de 16 bits a 40MHz para uma resolução de um bit. (MORAIS, 2017)

A função “ledc” é uma das formas de implementar o PWM no ESP32, mas quando é interessante trabalhar apenas com os tempos de nível lógico alto e baixo, o melhor é utilizar os timers disponíveis no microcontrolador.

2.5.2.2 Timer

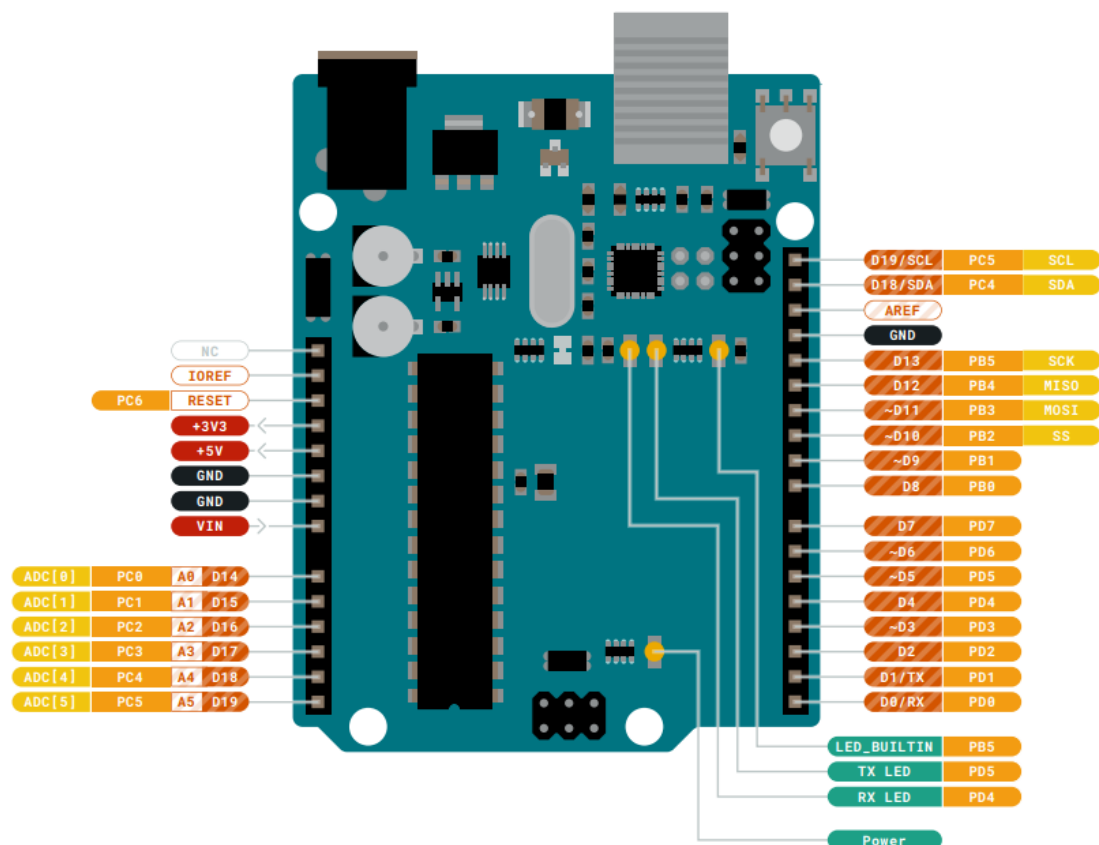
O microcontrolador possui quatro timers de propósitos gerais. São baseados em contadores de 64 bits e um prescaler, um “reduzidor” de 16 bits, ou seja, o prescaler pode assumir valores entre 2 a 65536. O timer funciona com base no clock de 80MHz do ESP32, sendo possível incrementar ou decrementar o timer. Não é necessário configurar o timer mais de uma vez, pois quando a contagem chega ao máximo, o próprio timer dispara o sinal de alarme e ele é resetado, retomando o ciclo de contagem. (ESPRESSIF SYSTEMS, 2019)

2.5.3 Arduino UNO

A plataforma de prototipagem que realizará a emulação da planta do projeto é o Arduino UNO, um projeto open-source criado pela empresa Arduino. Mais conhecido

que o ESP32, o Arduino UNO possui bom desempenho de processamento, baixo consumo de energia e custo ainda menor que o outro controlador utilizado no projeto. Não possui conectividade WiFi ou bluetooth de fábrica, mas existem os chamados “shields” que nada mais são que dispositivos criados para serem acoplados nas portas do microcontrolador. Ele possui 14 pinos digitais que podem ser usados como entradas e saídas, onde 6 deles podem ser usados para gerar saída PWM. Além disso, possui 6 entradas analógicas e opera com uma tensão de referência de 0 a 5V com um clock de processamento de 16 MHz. A placa possui conversores A/D com resolução de 10 bits, ou seja, o sinal analógico lido é mapeado para um valor entre 0 a 1023 bits e sua saída PWM possui resolução de 8 bits, ou seja, podem ser definidos valores de escrita (Duty Cycle) de 0 a 255, equivalentes ao alcance de tensão que o microcontrolador consegue escrever em sua saída, sendo de 0 a 5V. O microcontrolador possui ao todo, três timers disponíveis para uso: dois possuem resolução de 8 bits e um possui resolução de 16 bits. Os timers de 8 bits geralmente são utilizados para funções internas do dispositivo, portanto, o timer de 16 bits está livre para ser usado. (MICROCHIP TECHNOLOGY INC., 2018)

Figura 24: Pinagem Arduino UNO



Fonte: ARDUINO, [s.d.]

As funções de leitura e escrita PWM e timers também existem para o Arduino, sendo necessário instalar a biblioteca “Timer1” no IDE do Arduino UNO, para que seja possível utilizar o timer de hardware de 16 bits que a placa possui.

3 METODOLOGIA

Inicialmente, a revisão bibliográfica foi levantada para maior entendimento e planejamento das etapas do procedimento experimental. Em seguida, foram feitos testes das funções do ESP32 e do Arduino UNO, para que houvesse maior familiarização com os dispositivos e observar suas peculiaridades.

Testes iniciais em malha aberta foram emulados para validação de rotinas de atuação na variável manipulada utilizando uma lógica PWM. Também foram feitos testes de conectividade MQTT do ESP32 com o broker. Os resultados observados sugeriram que será possível atuar na planta e fazer a supervisão pelo computador.

A metodologia empregada é formada por três etapas onde, inicialmente, as equações diferenciais ordinárias (EDO) do controlador e da planta são representadas no espaço de estados por meio de diagramas de blocos na forma canônica observável, garantindo assim que cada EDO seja formada por um número de integradores em relação a sua ordem, independente do controlador ou da planta serem lineares ou não lineares. Em seguida, as equações diferenciais devem ser modeladas para que seja possível implementá-las nos microcontroladores e, por fim, é realizada a parametrização via *IoT*.

3.1 MATERIAIS

Os materiais utilizados foram:

Quadro 2: Lista de Materiais

| Item | Descrição |
|------|--|
| 1 | Microcontrolador ESP32 |
| 2 | Microcontrolador Arduino UNO |
| 3 | Jumpers |
| 4 | Protoboard de 1660 Pontos |
| 5 | Fonte ajustável para protoboard |
| 6 | Fonte 9V DC chaveada de parede |
| 7 | Potenciômetro Linear 2k Ω (B2K) |
| 8 | Capacitores 1 μ F e 10 μ F |
| 9 | Resistores 10k Ω e 330 Ω |
| 10 | Cabo de dados USB A/B |
| 11 | Cabo de dados Micro-USB |
| 12 | Chave tátil (Push Botton) |

Fonte: Autor

3.2 TESTES INICIAIS

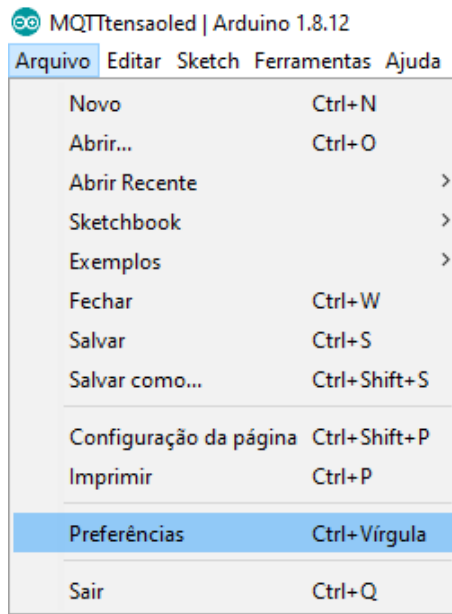
3.2.1 Configuração IDE Arduino para ESP32

O microcontrolador ESP32 não possui um software próprio para o desenvolvimento de aplicações. Entretanto, é possível adicionar o dispositivo em IDE's famosas como a Arduino, CodeBlocks ou PlatformIO. No projeto, será utilizada a IDE Arduino por ser mais fácil e intuitiva em relação as outras.

A instalação da placa ESP32 na IDE é simples. Após a instalação da IDE, o programa foi aberto, sendo necessário acessar a opção “Arquivo” e em seguida “Preferências” (Figura 25). Na janela que se abriu, o campo “URLs Adicionais para Gerenciadores de Placas:” foi preenchido pelo site que fornece as bibliotecas ligadas ao ESP32 (Figura 26). Esses passos foram apenas para adicionar o site fonte para que a IDE possa localizar a placa para download. O download foi feito acessando “Ferramentas”, em seguida “Placa: Arduino Uno” e clicando na opção “Gerenciador de Placas” (Figura

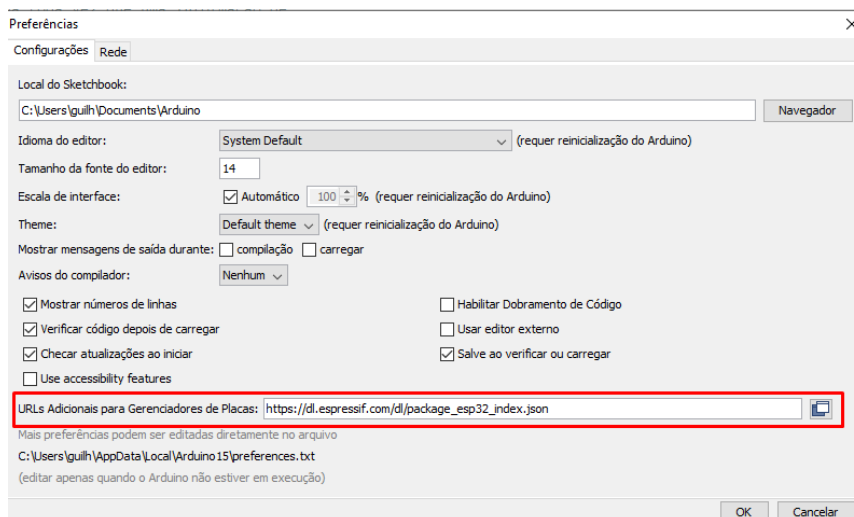
27). O programa atualizou as placas padrões existentes e o campo na parte superior da janela foi habilitado para pesquisa e, procurando por ESP32, foi instalado o único resultado existente (Figura 28).

Figura 25: Passo 1 – Instalação ESP32 na IDE Arduino



Fonte: Autor

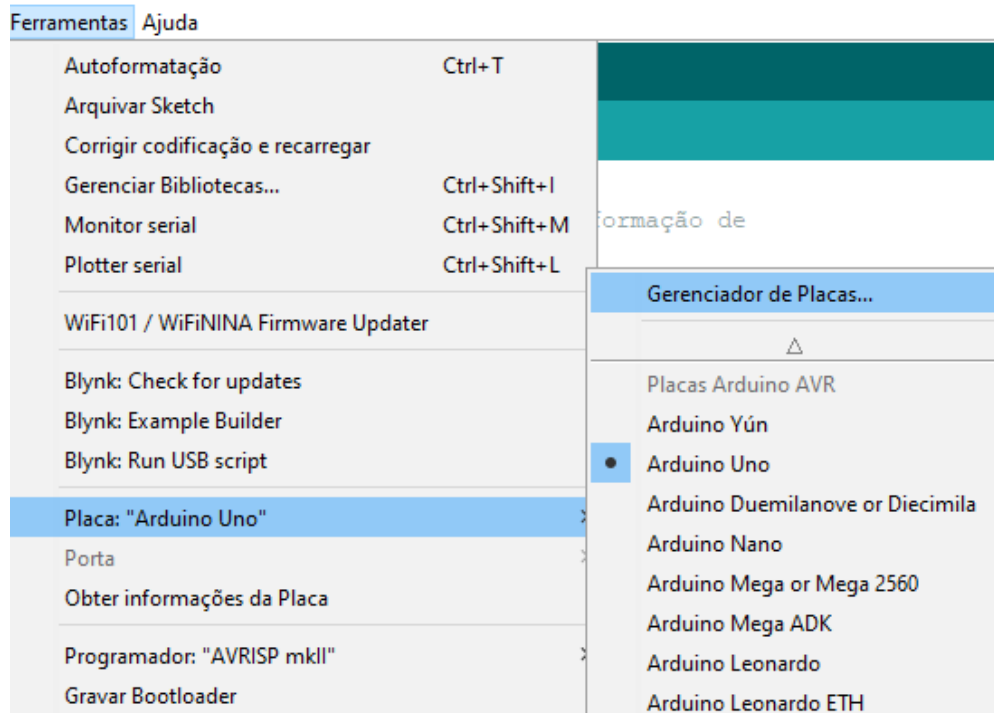
Figura 26: Passo 2 – Instalação ESP32 na IDE Arduino



Fonte: Autor

Figura 27: Passo 3 – Instalação ESP32 na IDE Arduino

Arduino 1.8.12



Fonte: Autor

Figura 28: Passo 4 – Instalação ESP32 na IDE Arduino

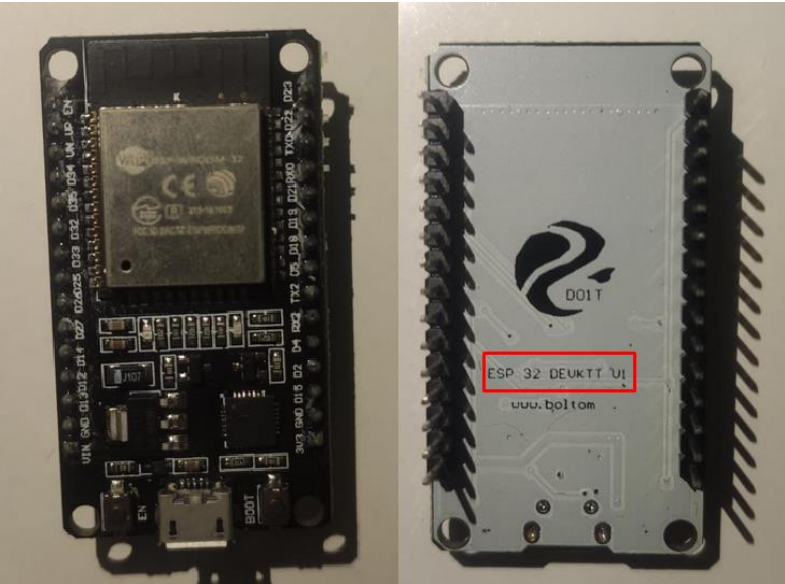


Fonte: Autor

Após essas configurações iniciais, o microcontrolador ESP32 aparecerá como opção a ser selecionada na opção “Ferramentas”, “Placa: “, sendo necessário procurar entre as diversas versões do ESP32 a compatível com o microcontrolador utilizado

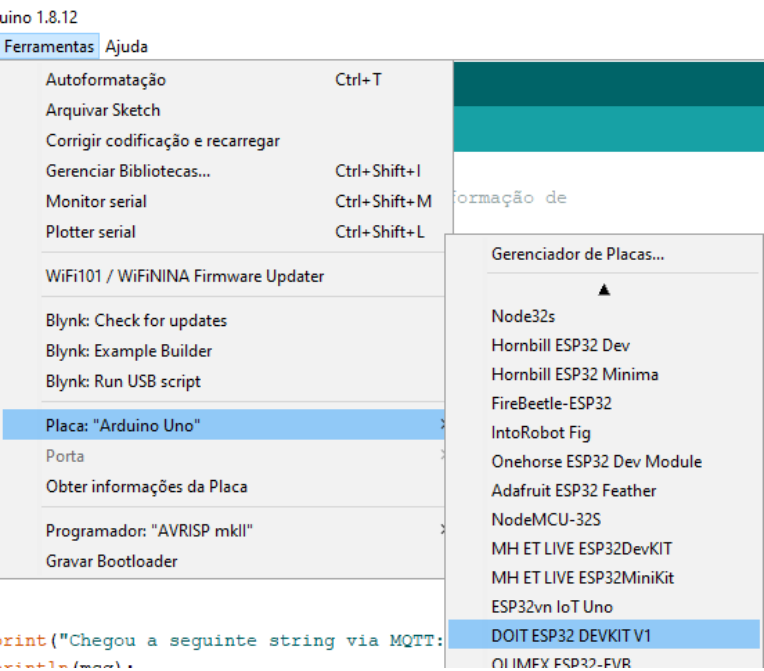
(Figura 29). A versão da placa geralmente está escrita atrás da placa (Figura 30). No projeto, será utilizada a versão DOIT ESP32 DEVKIT V1, com as configurações padrões da placa (Figura 31).

Figura 29: Aspectos físicos do ESP32



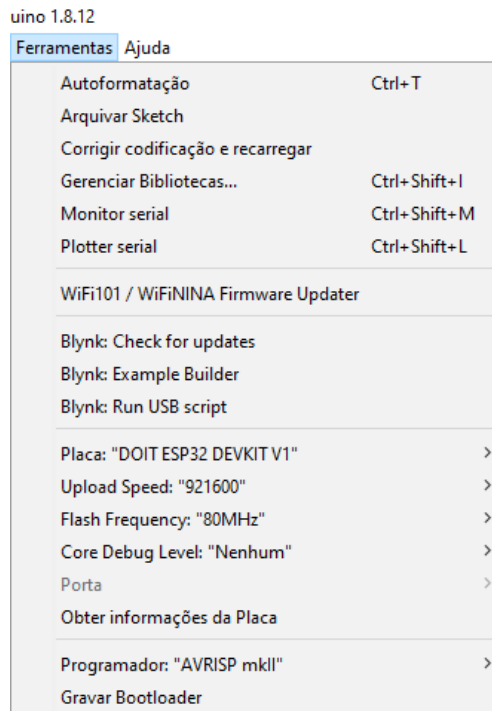
Fonte: Autor

Figura 30: Seleção do Microcontrolador na IDE



Fonte: Autor

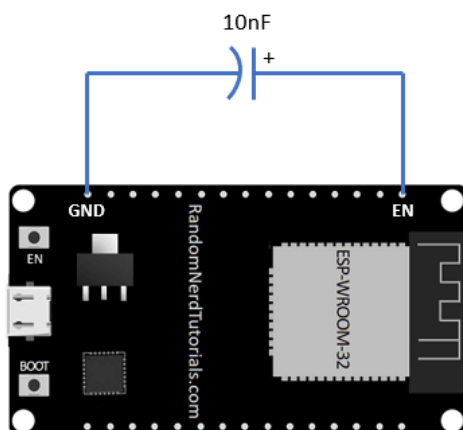
Figura 31: Configurações ESP32 (Software)



Fonte: Autor

Alguns modelos do ESP32 não entram no modo de “upload” do código automaticamente, necessitando então pressionar o botão “BOOT” até que o código seja baixado. Para que o download do código na placa seja feito de maneira automática, é necessário conectar um capacitor de 10nF entre os pinos “EN” (Enable) e “GND” (Ground – Terra), conforme figura abaixo. Assim, a IDE Arduino e o ESP32 estarão configurados para programação.

Figura 32: Configuração ESP32 (Hardware)

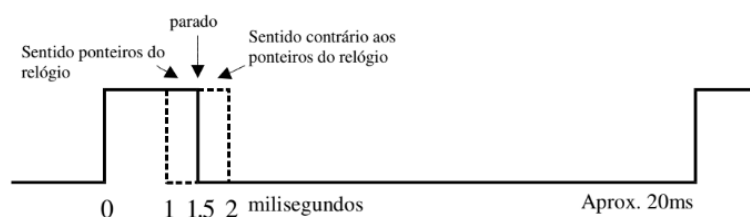


Fonte: Autor

3.2.2 Implementação do PWM

A saída PWM do ESP32 foi posta em prática no acionamento em malha aberta do Módulo Motor DC MS15 (Figura 33) projetada pela LJ Systems. Esse dispositivo é formado por um servomotor e alguns periféricos para torná-lo em uma placa didática. De acordo com as especificações do módulo, o motor deve funcionar de acordo com um sinal PWM representado no esquema abaixo.

Figura 33: PWM utilizado com o MS15



Fonte: LJ SYSTEMS, [s.d.], p. 181

Utilizando a função “*ledc*” do ESP32 e um potenciômetro linear de $2k\Omega$, foi possível controlar via PWM o motor. Para isso, foi necessário fazer as ligações do Kit LJ com a alimentação e o microcontrolador. O terminal da saída PWM da placa foi configurado para o GPIO2, canal 0 e com frequência 50Hz de acordo com o período de 20ms mostrado na Figura 33. A leitura do potenciômetro foi feita no GPIO34.

A Figura 34 apresenta a declaração das variáveis usadas no exemplo. A variável *PotPin* define qual GPIO será lido e, como dito anteriormente, foi o GPIO34. As variáveis devem ser inicializadas com algum valor, pois dependendo da aplicação, há o risco da variável assumir valores incoerentes ao longo da execução do programa. Vale ressaltar que variáveis declaradas fora de qualquer função são chamadas de variáveis globais, pois podem ser usadas no código inteiro. As variáveis declaradas dentro das funções são as variáveis locais e podem ser usadas apenas dentro das funções.

Figura 34: Declaração de Variáveis

```
int dutyCycle = 0;
const int potPin = 34;
int potValue = 0;
```

Fonte: Autor

A figura abaixo mostra a função *Setup()* da IDE Arduino, usada para descrever as ações que serão feitas apenas uma vez no código. A estrutura é simples, primeiro inicializa-se a porta serial (USB) para leitura, definindo sua velocidade através da função *Serial.begin()*. Em seguida, o terminal GPIO2 é definido como “OUTPUT” (Saída), para a saída PWM. A função *ledcAttachPin()* atribui ao GPIO2 o canal 0, assim como a função *ledcSetup()* atribui ao canal 0, a frequência de 50Hz com 12 bits de resolução. Nota-se que a frequência máxima para a resolução de 12 bits é aproximadamente 19,5KHz, portanto, a frequência utilizada está dentro do alcance do microcontrolador.

Figura 35: Função *Setup()*

```
void setup() {
  Serial.begin(115200);
  delay(10);
  Serial.println("Programa baixado");

  pinMode(2, OUTPUT);
  ledcAttachPin(2, 0);
  ledcSetup(0, 50, 12);
}
```

Fonte: Autor

A função *loop()* descreve o código que será rodado infinitamente até que ocorra alguma interrupção. A partir dela, a leitura da tensão do potenciômetro será feita através da função *analogRead()* com resolução de 12 bits por padrão, ou seja, o valor da leitura pode variar de 0 a 4095. Assim, utilizando a função *ledcWrite()*, o valor lido é atribuído como o valor do Duty Cycle do canal 0 do PWM, conforme figura abaixo.

Figura 36: Função *loop()*

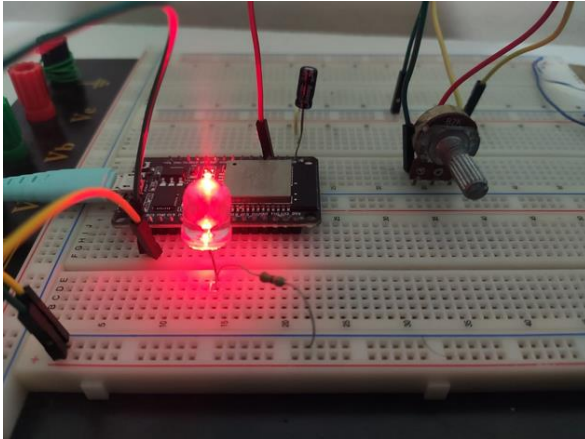
```
void loop() {
  potValue = analogRead(potPin);
  ledcWrite(0, potValue);
  delay(10);
}
```

Fonte: Autor

Devido ao sinal PWM possuir resolução de 12 bits e a leitura do potenciômetro possuir resolução de mesmo valor, o curso do potenciômetro percorrerá completamente

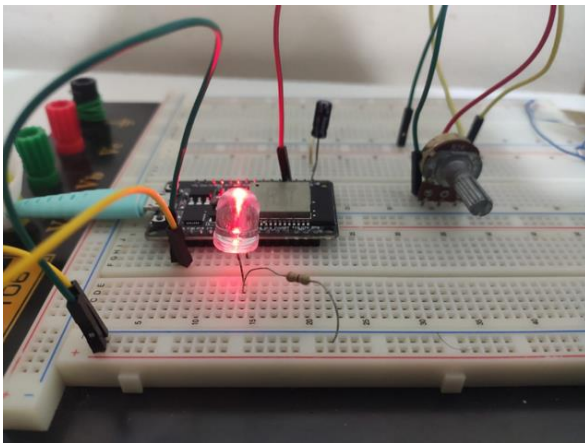
o período do sinal PWM. Após a execução do código, foi constatado que de fato, o motor se comporta da maneira descrita na Figura 33.

Figura 37: PWM com Duty Cycle de 100%



Fonte: Autor

Figura 38: PWM com Duty Cycle de 15%



Fonte: Autor

3.2.3 Implementação do Timer

O timer foi implementado com um prescaler no valor de 80, ou seja, o sinal de clock do ESP32 será dividido por 80, resultando em uma contagem de 10^6 us, ou seja, 1 segundo. A configuração no software é inicializada definindo uma variável do tipo timer, no caso, será chamada de *tp*. Em seguida, na função *Setup()*, serão configurados os parâmetros da variável *tp* utilizando a função *timerBegin()*. A partir dela, é selecionado a identificação do timer do ESP32, sendo de 0 a 3 para cada timer do total de quatro. Em

seguida o prescaler é declarado e o parâmetro booleano final possui a função de incrementar ao ser definida como *true* ou decrementar ao ser definida como *false*.

A próxima etapa é a declaração da função *timerAttachInterrupt()*, o primeiro parâmetro é variável *tp*, o segundo deve ser o endereço da função que será chamada quando o contador chegar a 1 segundo e o terceiro parâmetro define se o timer irá incrementar na borda (edge) do sinal do clock ou em uma amplitude (level) específica do sinal, sendo *true* para selecionar o modo da borda ou *false* para selecionar o modo da amplitude.

Em seguida, deve-se configurar a função *timerAlarmWrite()* para definir em que valor do contador o timer será acionado. A função possui três parâmetros: a variável timer, *tp* no caso; o valor de disparo da interrupção, sendo no valor de 10^6 neste exemplo e a opção de resetar o timer automaticamente, definida para *true*, para resetar ou *false* para não resetar.

Por fim, a função *timerAlarmEnable()* ativa a variável timer *tp* e a função *timerWrite()* é adicional, ela apenas reseta o temporizador, ou seja, inicia o temporizador em 0. (SANTOS N., [s.d.])

A função *ppsa()* que o timer irá mostrar na saída serial o número de vezes que um botão configurado anteriormente foi pressionado. A função *loop()* irá rodar uma contagem por software enquanto detecta a mudança de estado lógico do botão e incrementa uma variável.

Ou seja, é possível trabalhar tanto com os timers por hardware do ESP32 quanto com o timer por software utilizando a função *millis()* e ambos ao mesmo tempo. As figuras abaixo mostram passo a passo da implementação do programa descrito.

Figura 39: Timer – Definição das variáveis e função de alarme

```
const int pinButton = 4;
float contagem = 0;
int est = 0;
int est_a = 0;
float pps = 0;
hw_timer_t *tp = NULL;

void ppsa() {
    pps = (contagem*1000)/millis();
    Serial.println("Apertos por segundo: " + String(pps));
    timerWrite(timer, 0);
}
```

Fonte: Autor

Figura 40: Timer – Declaração dos parâmetros

```
void setup() {
    Serial.begin(115200);
    pinMode(pinButton, INPUT);
    pinMode(internLed, OUTPUT);

    tp = timerBegin(0, 80, true);
    timerAttachInterrupt(tp, &ppsa, true);
    timerAlarmWrite(tp, 1000000, true);
    timerAlarmEnable(tp);
    timerWrite(timer, 0);
}
```

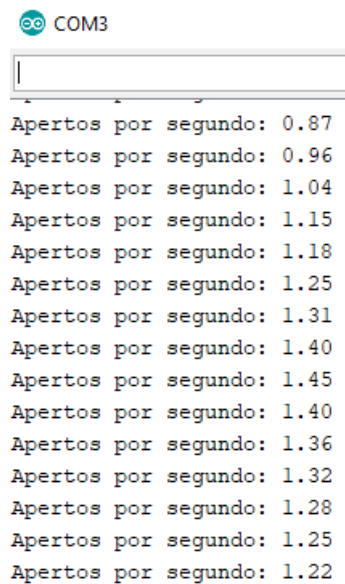
Fonte: Autor

Figura 41: Timer – Lógica do Botão

```
void loop() {
    long tme = millis(); //tempo inicial do loop
    est = digitalRead(pinButton);
    if (est != est_a) {
        if (est == HIGH) {
            contagem++;
        }
        else {
            contagem = contagem;
        }
        delay(10);
    }
    est_a = est;
}
```

Fonte: Autor

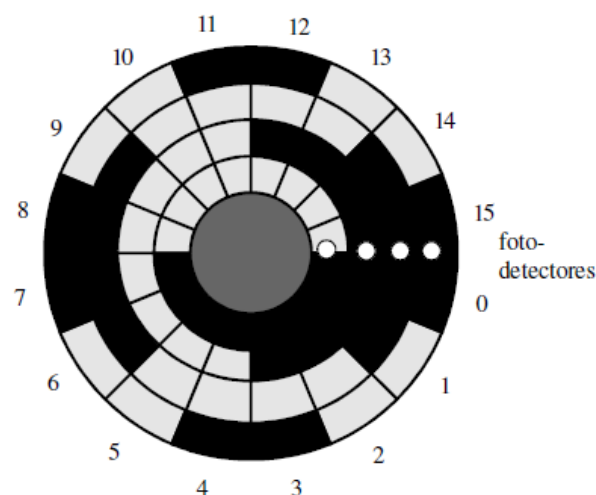
Figura 42: Exemplo de Implementação do Timer



Fonte: Autor

O Kit LJ possui diversos periféricos, um deles é a leitura de posição e, consequentemente, rotação sendo possível ser feita a partir de um disco com código de Gray, onde há quatro fotodetectores alinhados com o disco. O disco possui dezesseis marcas, ou seja, dezesseis combinações binárias para a identificação das posições.

Figura 43: Disco com código de Gray



Fonte: LJ SYSTEMS, [s.d.]

O código seria implementado para fazer a leitura das posições e a partir delas, obter as velocidades em RPM do motor, mas, devido ao afastamento das estruturas da

FEI e dos equipamentos, o funcionamento físico do disco foi substituído pelo uso de dois botões para implementar quatro estados, sendo apenas três necessariamente lidos.

O funcionamento se baseia em, ao pressionar um botão, será simulada a rotação horária do disco considerada positiva. Assim, quanto mais vezes o botão for pressionado, a contagem da “velocidade” irá aumentar. Ao soltá-lo, a contagem diminuirá conforme o tempo passa e, ao pressionar o segundo botão, será simulada a rotação anti-horária do motor, ou seja, a contagem irá decrementar até se tornar negativa. Os botões sem estarem pressionados representam o motor parado, ou seja, ficou parado em um estado, no caso, o estado “00” pois os botões estão em nível lógico baixo. Essa implementação foi feita utilizando o timer do ESP32, baseado no exemplo anterior. Note que os botões que simulam os sentidos de rotação do disco foram acionados duas vezes, em sequência e é possível observar a contagem incrementando e decrementando.

Figura 44: Simulação Código de Gray

```
Apertos por segundo: 0.00
Contagem: 1.00
Contagem: 2.00
Apertos por segundo: 0.06
Apertos por segundo: 0.00
Apertos por segundo: 0.00
Apertos por segundo: 0.00
Apertos por segundo: 0.00
Apertos por segundo: 0.00
Contagem: -1.00
Contagem: -2.00
Apertos por segundo: -0.04
```

Fonte: Autor

3.2.4 Conexão MQTT

A conectividade MQTT do ESP32 com o broker foi implementada utilizando das bibliotecas chamadas “PubSubClient” e “Wifi”. Através dela, foi possível definir os tópicos que o ESP32 como cliente foi inscrito e em qual tópico o microcontrolador iria receber dados ou enviar dados.

O teste a seguir possui como objetivo inscrever o ESP32 em um tópico onde ele irá enviar como dado a tensão lida de um potenciômetro e irá receber a ordem de liga e desliga de um LED.

O código completo do teste está comentado e em anexo (Anexo A), entretanto, vale ressaltar algumas informações importantes sobre a construção do código. Primeiramente, as bibliotecas “WiFi” e “PubSubClient” são inseridas, os pinos de entrada e saída da placa que serão utilizados são definidos e as variáveis globais são declaradas. Em seguida, todos os tópicos necessários, conexão WiFi, conexão com o Broker, funções de conexão e leitura e escrita são descritos. Algumas funções possuem nomes intuitivos e é possível entender imediatamente o que elas fazem, mas a função mais importante é a chamada “mqtt_callback”, pois é a partir dela que os dados recebidos são tratados. Essa função deve ser definida com alguns parâmetros de entrada, sendo o parâmetro “topic” do tipo ponteiro para uma informação do tipo *char* que guarda o nome do tópico em que a mensagem foi recebida, o parâmetro “payload” também do tipo ponteiro, mas para uma informação do tipo *byte* que guarda a mensagem enviada. O terceiro parâmetro é utilizado para definir o tamanho em *bytes* de uma variável do tipo *unsigned int* pois como a mensagem é recebida em *bytes*, é necessário definir esse número de *bytes* para que dentro da função “callback”, cada caractere da mensagem recebida seja transformado de *byte* para *char* e em seguida uma informação do tipo *String* seja formada e armazenada em uma variável para futuro uso.

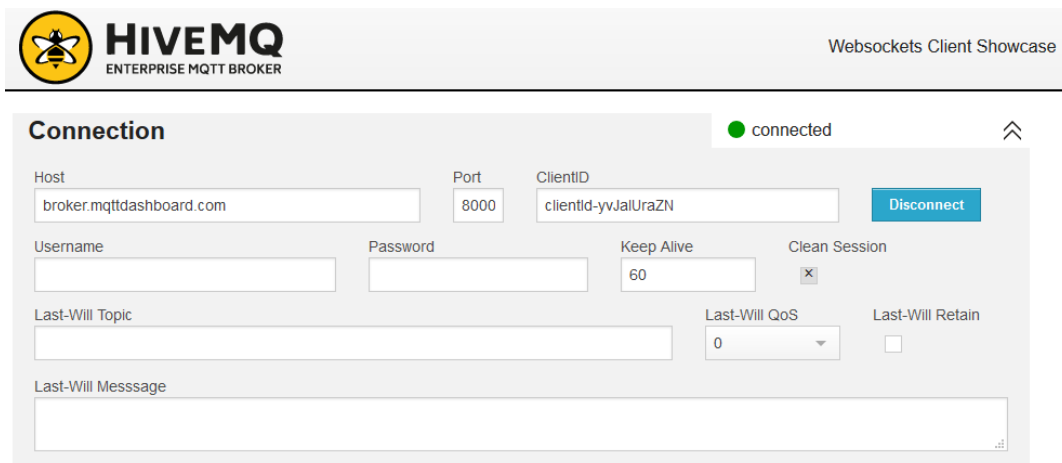
Há vários brokers gratuitos disponíveis para testes. Eles não possuem alta taxa de transmissão, mas, para pequenas aplicações, são eficientes. O quadro abaixo mostra alguns dos brokers disponíveis para teste. No projeto, foi usado o broker HiveMQ, pois além de disponibilizar a função gratuitamente, ainda dispõe de um site para conectar-se como cliente e então visualizar o fluxo de dados do sistema.

Quadro 3: Brokers gratuitos

| Broker | Endereço | Porta |
|-----------|-------------------------|----------------|
| HiveMQ | broker.hivemq.com | 1883 |
| HiveMQ | broker.mqtdashboard.com | 1883 |
| Mosquitto | test.mosquitto.org | 1883/8883/8884 |
| Mosquitto | iot.eclipse.org | 1883/8883 |

Fonte: Autor

Figura 45: WebClient HiveMQ



HIVEMQ
ENTERPRISE MQTT BROKER

Websockets Client Showcase

Connection connected

Host: Port: ClientID: Disconnect

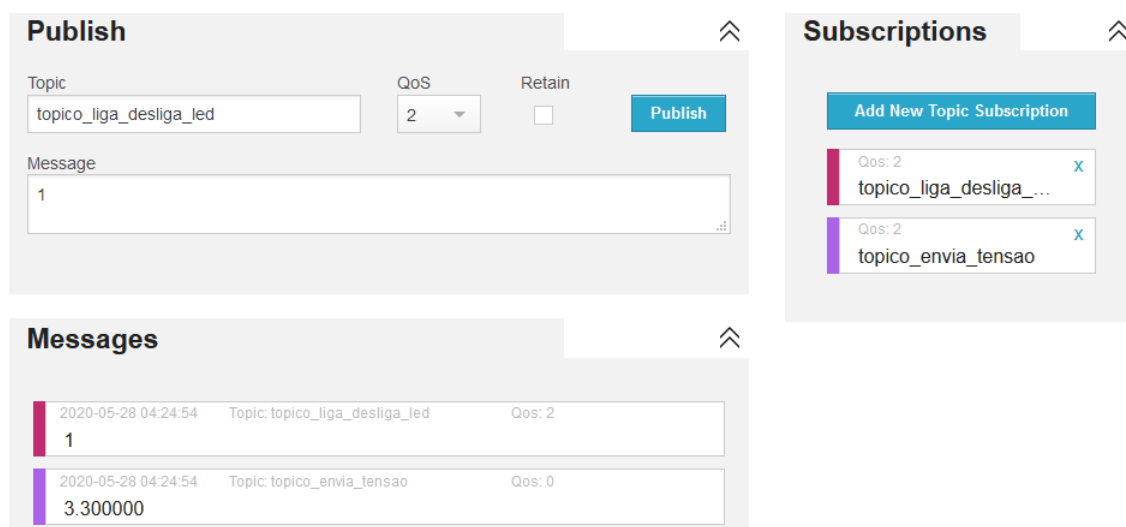
Username: Password: Keep Alive: Clean Session: ☒

Last-Will Topic: Last-Will QoS: Last-Will Retain: ☐

Last-Will Message:

Fonte: Autor

Figura 46: WebClient HiveMQ



Publish

Topic: QoS: Retain: ☐ Publish

Message:

Subscriptions

Add New Topic Subscription

- Qos: 2 X
topico_liga_desliga_...
- Qos: 2 X
topico_envia_tensao

Messages

| | | |
|---------------------|--------------------------------|--------|
| 2020-05-28 04:24:54 | Topic: topico_liga_desliga_led | Qos: 2 |
| 1 | | |
| 2020-05-28 04:24:54 | Topic: topico_envia_tensao | Qos: 0 |
| 3.300000 | | |

Fonte: Autor

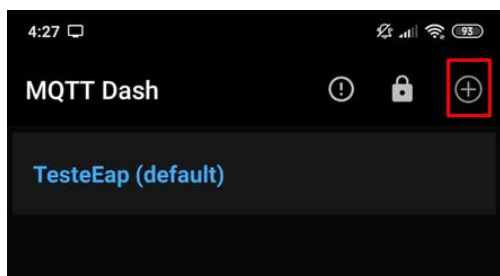
A partir das imagens acima, observa-se que o layout do site é simples, mas possui as funções essenciais do MQTT que são a opção de inscrever o cliente no tópico (Subscribe), publicar mensagens (Publish) e ainda há um histórico de mensagens.

A conexão com o broker é feita apenas se o ESP32 estiver conectado com alguma rede WiFi. Portanto, para redes locais, é possível criar um broker particular e utilizá-lo, mas em contrapartida, o broker poderá ser acessado apenas na rede local. Por isso, para obter um broker particular conectado à internet é necessária a assinatura de serviços de terceiros ou utilizar métodos de encaminhamento de porta.

Além do site disponibilizado pela HiveMQ, há o aplicativo de celular chamado MQTT Dash onde é possível configurar botões, campos de texto e imagens e, a partir deles, configurar um cliente customizável para a aplicação. No teste feito, foram utilizados o WebClient e o MQTT Dash.

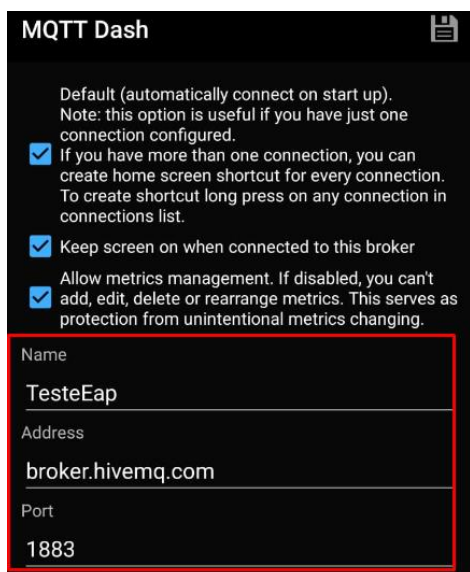
A configuração do aplicativo deve ser feita pressionando o sinal de “+” na tela, em seguida é dado um nome para a aplicação e inserido o endereço e a porta do broker utilizado, na maioria dos brokers gratuitos a porta é 1883, mas pode ser definido outros valores no caso de criar um broker MQTT. É necessário também definir um ID do cliente, sendo possível inserir qualquer nome, desde que seja o único conectado ao broker. Os campos *Username* e *Password* podem ser deixados em branco, mas caso seja necessária uma proteção maior, basta criar um login inserindo o nome de usuário e a senha nos campos, respectivamente.

Figura 47: Página inicial MQTT Dash



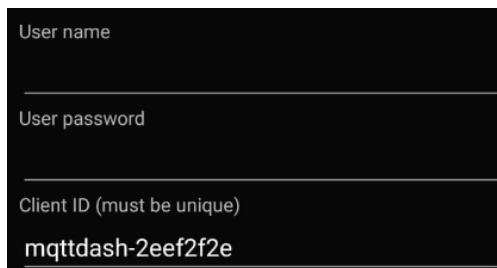
Fonte: Autor

Figura 48: Configurando nova aplicação (1)



Fonte: Autor

Figura 49: Configurando nova aplicação (2)



User name

User password

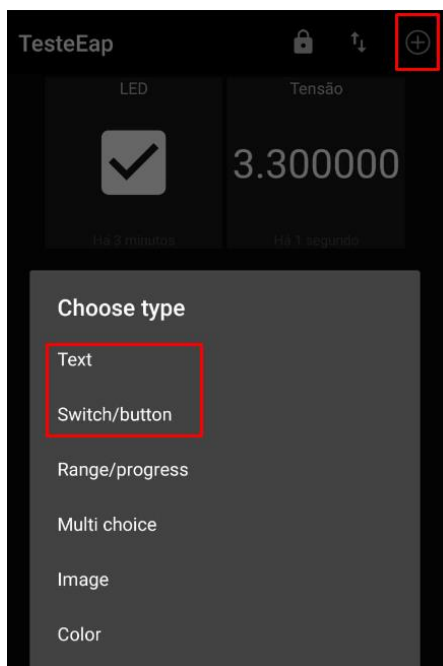
Client ID (must be unique)

mqttdash-2eef2f2e

Fonte: Autor

Terminada a configuração, basta clicar no símbolo de disquete no canto superior direito e a aplicação está criada. Para adicionar as funções de envio e leitura de dados, é necessário clicar na aplicação e novamente, clicar no símbolo de “+”. No exemplo, foi usado a opção *Button* e *Text* para ligar e desligar o LED e receber a informação de tensão do potenciômetro, respectivamente.

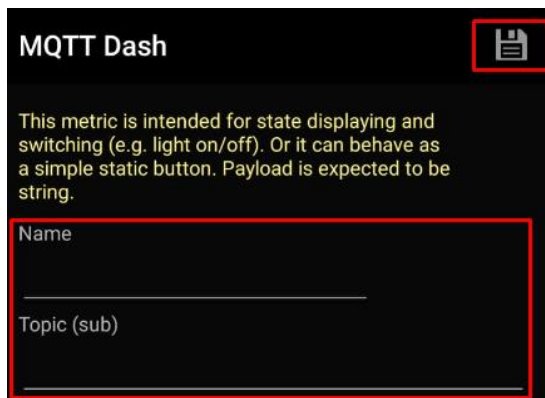
Figura 50: Funções da aplicação



Fonte: Autor

Para configurar as funções, o processo é semelhante em todas. Basta atribuir um nome qualquer à função, inscrevê-la no tópico corresponde, sendo de envio de dados para o botão e recebimento de dados para o texto. Em seguida, basta salvar a função e a aplicação está configurada para uso.

Figura 51: Configuração nova função



Fonte: Autor

Figura 52: Teste da Aplicação



Fonte: Autor

3.3 PROCEDIMENTO EXPERIMENTAL

3.3.1 Implementação do sistema HIL no Simulink

Algumas questões devem ser consideradas no momento da implementação do sistema HIL. Os microcontroladores possuem características únicas e inerentes ao seu fabricante. O ESP32, por exemplo, opera com uma tensão de referência de 0 a 3,3V, com resoluções de escrita e leitura de sinais variando entre 9 a 12 bits enquanto o Arduino UNO trabalha com tensões de 0 a 5V, resolução de leitura de sinal analógico de 10 bits e resolução de escrita de sinal PWM de 8 bits.

Outro fator importante a ser observado é que os sinais da lei de controle podem apresentar valores negativos mesmo que os sinais de entrada e saída dos conversores sejam valores somente positivos. Esse problema foi resolvido modificando o referencial de tensão dos sinais, associando o intervalo de 0 a 1.65V a valores negativos, enquanto

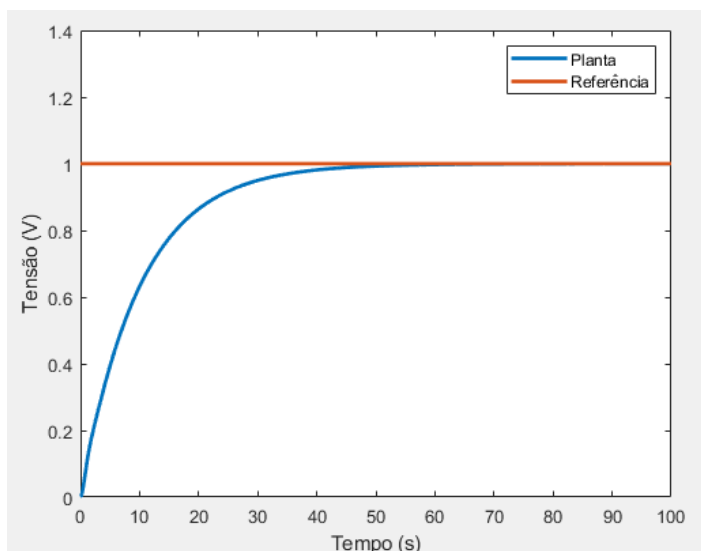
os valores positivos ficam entre 1.65 e 3.3V. Para minimizar a perda de precisão que essa alteração na referência provoca, os sinais de entrada do ESP32 foram multiplicados por uma potência de dois para que assim fosse possível trabalhar com valores numéricos com uma range maior e não haver perdas por arredondamento. No Arduino UNO, foi utilizada a mesma estratégia, chamando de Compressão a ação de transformar o sinal multiplicado pela potência de dois em um valor dentro da faixa da resolução de saída dos microcontroladores e de Descompressão a ação de multiplicar os valores de leitura pela potência de dois.

A ação integral do controlador PID foi tratada para eliminar o efeito da ação do windup, sendo necessário estipular limites máximos e mínimos que a ação integral poderia alcançar com base um parâmetro adicional denominado “Fator B” cujo valor pode ser definido pelo usuário através da conexão MQTT. Outro parâmetro adicional foi criado para modificar o sinal de controle do PID caso necessário, denominado de “Fator A”.

O *software* Simulink foi usado como base antes da construção do sistema em uma protoboard. A partir dele, funções de ambos os controladores foram emulados, como os conversores A/D e D/A, manipulação das grandezas dos valores lidos pelos microcontroladores devido a diferença dos referenciais e validação das estratégias adotadas. O diagrama resultante se encontra no Anexo B.

A emulação realizada no Simulink consistiu em uma planta com constantes K_g , K_I e K_D de valores 1, 3 e 5, respectivamente. Através da técnica de cancelamento de polos, as constantes K_P , K_I e K_D do controlador PID foram configuradas para os valores 0.3, 0.5 e 0.1, respectivamente. O gráfico ilustrado na Figura 53 relaciona a referência de 1V com a saída da planta ao longo de 100 segundos. Vale ressaltar que na montagem, o valor de referência será controlado por um potenciômetro ligado à alimentação do ESP32 e, portanto, seu range de tensão será de 0 a 3.3V. Como o referencial do microcontrolador foi trocado para 1.65V para poder trabalhar com valores negativos da ação de controle, a tensão lida do potenciômetro foi dividida para que não ocasionasse erros de operação. Portanto, no Simulink, essa ação foi realizada adicionando um ganho de 0.5 ao valor lido da referência.

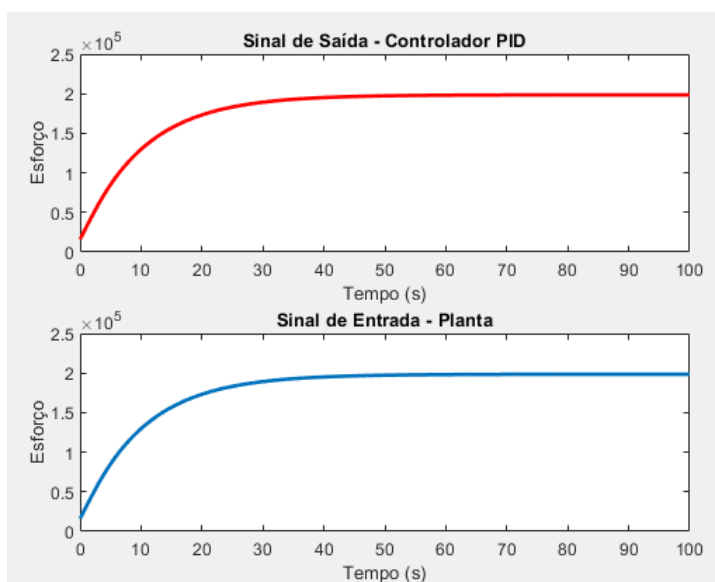
Figura 53: Gráfico Referência x Planta - Simulink



Fonte: Autor

Outro teste foi feito para comparar o sinal de esforço de controle que sai do controlador PID e o sinal que chega na planta para validar as conversões feitas entre esses dois pontos. O gráfico da Figura 54 mostra que as conversões estão corretas.

Figura 54: Sinais de Esforço de Controle - Antes e pós conversões



Fonte: Autor

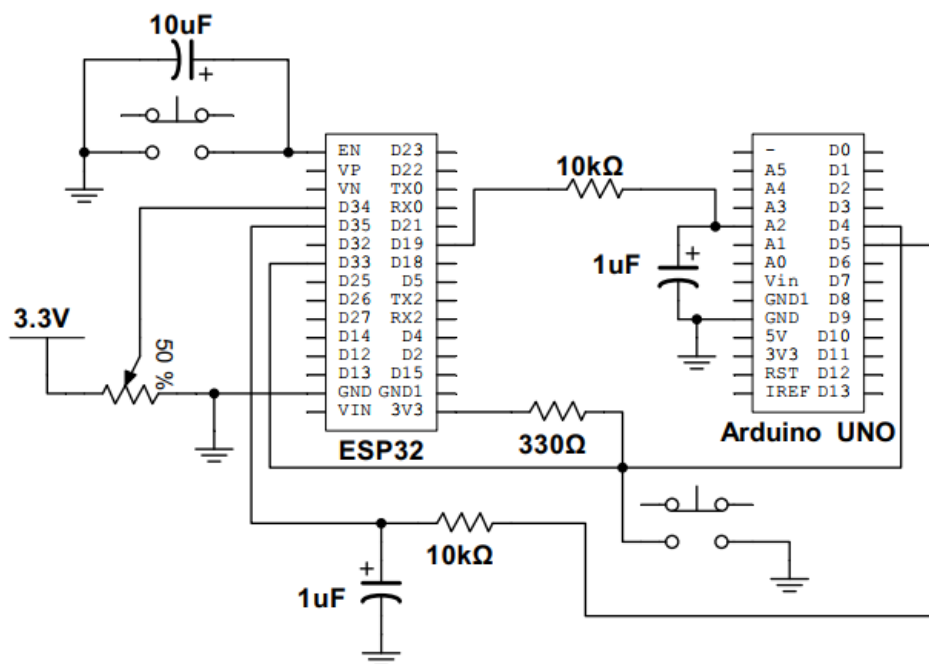
3.3.2 Montagem dos dispositivos na Protoboard

Os microcontroladores Arduino UNO e ESP32 não precisam, para poderem ser operados sozinhos, de uma montagem placa de ensaio. Mas, devido à quantidade de elementos eletrônicos adicionados, foi utilizada uma protoboard para comportar os capacitores e resistores usados para a conversão de PWM em sinal analógico, sinal este que é possível de ser lido pelos microcontroladores.

No Arduino UNO, existem técnicas utilizando interrupções para leitura de PWM, mas estas utilizam dos timers internos para serem executadas e foi preferível utilizar um filtro Passa-Baixa RC para fazer a conversão do PWM. Utilizando um resistor de $10\text{k}\Omega$ e um capacitor de $0.1\mu\text{F}$, foi possível filtrar o sinal PWM com um tempo de acomodação e tensão de ripple razoáveis. Um push button foi utilizado conectado a uma porta de cada dispositivo, para que os dois iniciassem ao mesmo tempo, quando fosse necessário. Outro botão foi posicionado na porta ENABLE do ESP32 para que fosse possível reiniciá-lo caso necessário.

A Figura 55 mostra o diagrama do circuito após a montagem, com as respectivas portas de cada microcontrolador utilizadas. As representações dos dispositivos não são exatas, são apenas um guia para as conexões.

Figura 55: Layout Sistema HIL - Placa de Ensaio



3.3.2 Representação por diagrama de blocos

Para ilustrar o processo de representar uma EDO por meio de um diagrama de blocos com integradores, considere o exemplo a seguir onde a entrada é $u(t)$ e a saída é $z(t)$

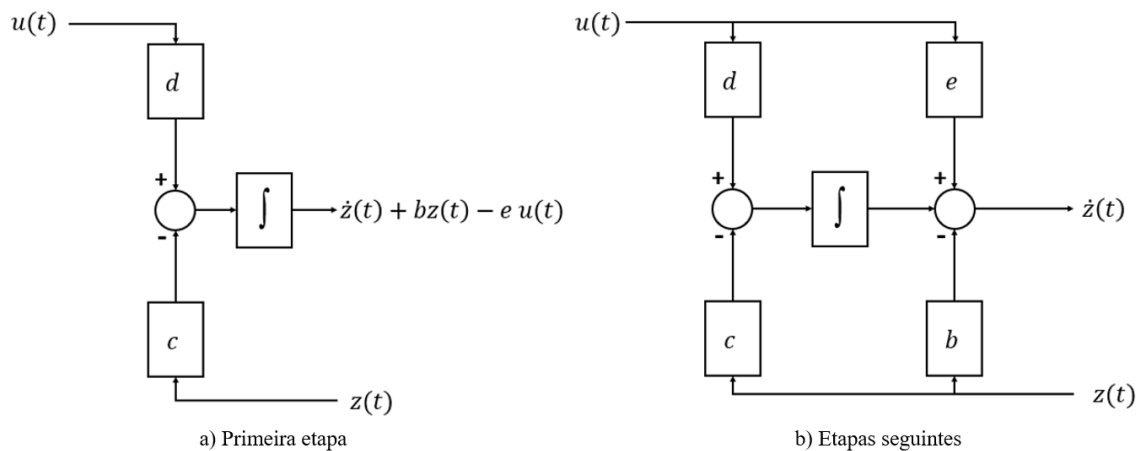
$$\ddot{z}(t) + b \dot{z}(t) + c z(t) = d u(t) + e \dot{u}(t) \quad (3.1)$$

A construção do diagrama de blocos na forma canônica observável necessita que as variáveis de entrada e saída de menor zero, ou seja, sem derivada, sejam representadas da seguinte maneira

$$d u(t) - c z(t) = \ddot{z}(t) + b \dot{z}(t) - e \dot{u}(t)$$

A figura a seguir mostra a representação por blocos dos dois membros da equação (3.2) onde no lado esquerdo (Figura 56a) o resultado é integrado e no lado direito (Figura 56b) os termos de menor ordem são removidos por adição de parcelas de sinais contrários.

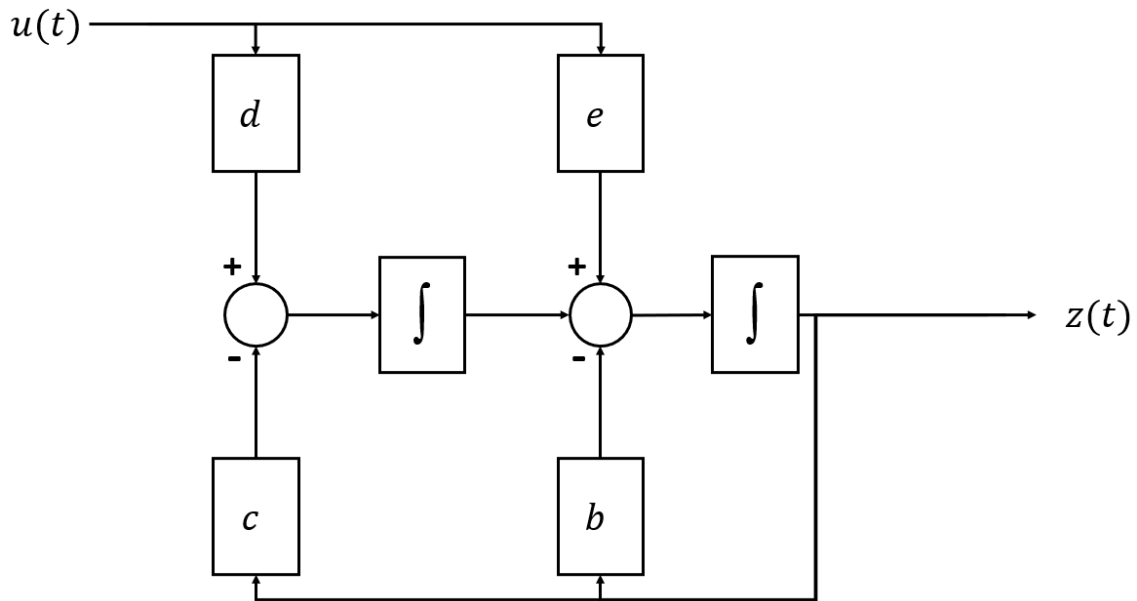
Figura 56: Construção do diagrama de blocos canônico



Fonte: RODRIGUES; PAULA; LEONARDI, 2020

Assim, o sinal $\dot{z}(t)$ é integrado resultando no diagrama na forma canônica observável da Figura 57.

Figura 57: Diagrama de blocos na forma canônica observável da equação (3.1)



Fonte: RODRIGUES; PAULA; LEONARDI, 2020

Observe que embora exista uma derivada do sinal de entrada, o diagrama resultante é composto por apenas dois integradores, coincidindo com a ordem da derivada máxima de $z(t)$ e nenhum diferenciador, além de somadores e blocos de ganhos.

Existem outras formas de conversão de uma EDO num diagrama de blocos com integradores. A seguir, será ilustrado a forma utilizada neste projeto para representar a EDO da planta utilizada, descrita por uma função de transferência de segunda ordem com ganhos K_g , K_1 e K_2 de valores 1, 3 e 5, respectivamente.

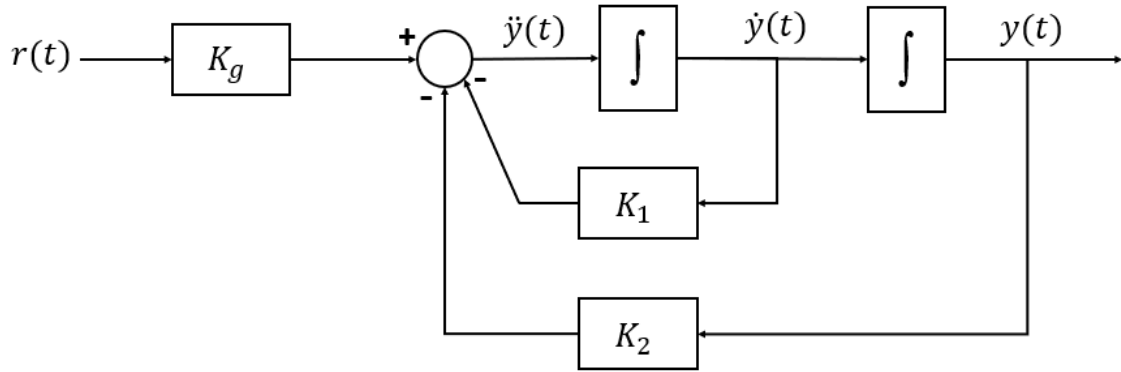
$$\frac{Y(s)}{R(s)} = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} = \frac{K_g}{s^2 + K_1 s + K_2}$$

Em seguida, transforma-se a função de transferência numa EDO

$$\ddot{y}(t) = K_g u(t) - K_1 \dot{y}(t) - K_2 y(t) \quad (3.2)$$

A Figura 58 representa o membro direito da equação (3.2) por um diagrama de blocos composto por integradores, ganhos e soma. O sinal resultante da soma é integrado duas vezes, obtendo-se $y(t)$ como saída.

Figura 58: Diagrama de blocos da equação (3.2)



Fonte: RODRIGUES; PAULA; LEONARDI, 2020

A próxima etapa da metodologia demonstra a implementação do diagrama de blocos com integradores numa plataforma de prototipagem eletrônica como o Arduino UNO ou ESP32.

3.3.3 Implementação das equações diferenciais

A implementação da EDO nas plataformas de prototipagem Arduino UNO ou ESP32 requer que os integradores sejam convertidos do domínio do tempo contínuo para o tempo discreto. Existem diversas técnicas para realizar essa conversão, como a aproximação de ordem zero (ZOH), aproximação bilinear (Tustin), etc. A escolha de qual técnica é a mais apropriada depende da aplicação desejada, assunto que não é o foco deste trabalho. Portanto, para simplificar o processo, a aproximação foi realizada baseando-se na definição de derivada de uma função.

$$\frac{dy(t)}{dt} = \lim_{T \rightarrow 0} \left(\frac{y(t+T) - y(t)}{T} \right) \cong \frac{y(t+T) - y(t)}{T}$$

Reescrevendo a aproximação para $t = kT$ para $k = \{0, 1, 2, \dots\}$, é possível mapear cada integrador $\frac{1}{s}$ do diagrama de blocos da EDO correspondente, da seguinte forma

$$s \rightarrow \frac{y(k+1) - y(k)}{T} \text{ ou } \frac{1}{s} \rightarrow \frac{T}{y(k+1) - y(k)}$$

A partir dessas associações, cada integrador i pode ser descrito por uma equação de diferenças do tipo em que u_i é a i -ésima entrada e y_i é a i -ésima saída.

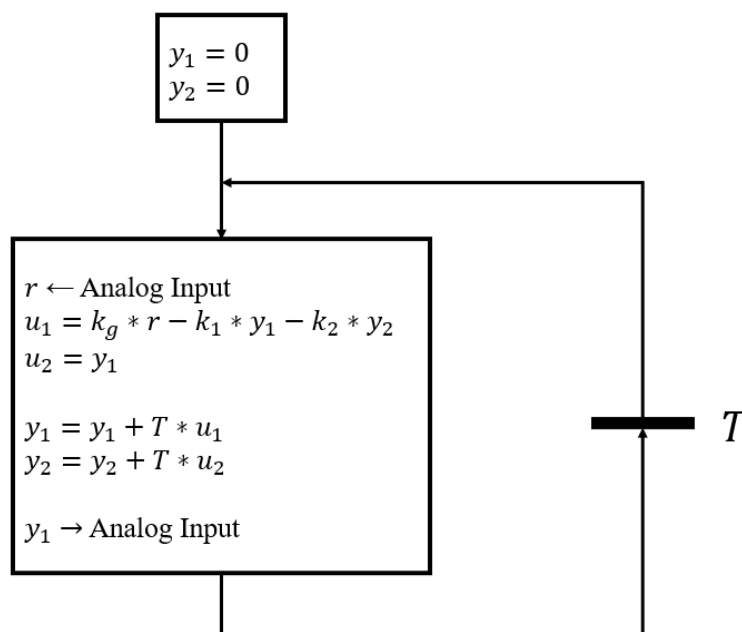
$$y_i(k+1) = y_i(k) + T u_i(k)$$

O restante do diagrama de blocos da EDO correspondente acrescenta apenas operações algébricas de simples codificação.

A variável T representa a duração do período entre kT e $(k + 1)T$. Isso significa que a cada rotina de integração, o algoritmo precisa ler o valor de entrada u_i e produzir um valor de saída y_i , em intervalos fixos de tempo T . Os microcontroladores possuem em sua arquitetura temporizadores de hardware com resolução de $1\mu s$ que são utilizados para realizar as interrupções que eles fornecem. No caso da planta da Figura 58, a rotina de integração é controlada por interrupções de $1ms$, tempo suficientemente pequeno para que a aproximação do integrador na forma discreta no Arduino UNO forneça valores precisos.

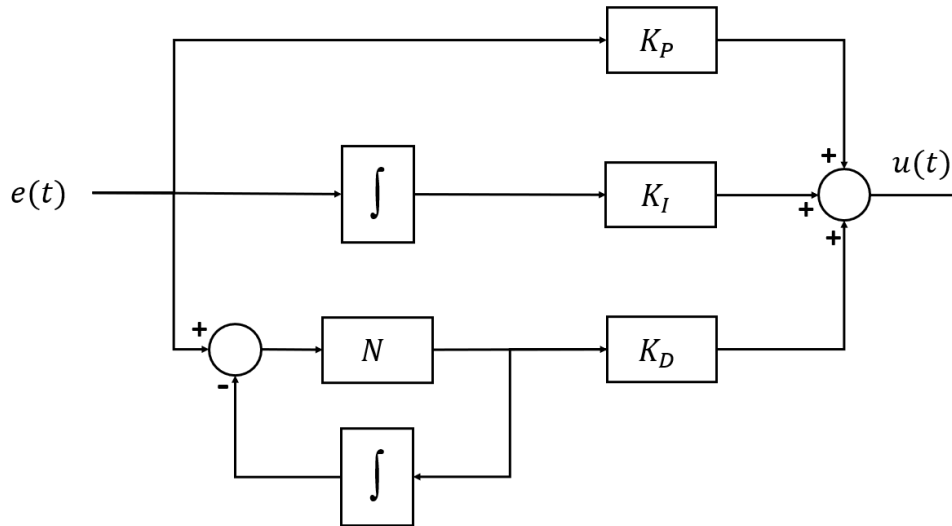
O algoritmo da planta está ilustrado no fluxograma da Figura 59. A condições iniciais dos integradores devem ser impostas fora da rotina que irá se repetir. A rotina de leitura da entrada e cálculo e escrita da saída será executada infinitamente a cada período de tempo T controlado pelas interrupções. A entrada r é obtida através da leitura feita por um conversor A/D para que assim a expressão algébrica do primeiro somador seja codificada. Em seguida, os dois integradores têm seus valores atualizados para que seja fornecida uma saída, enviada para o conversor D/A. Assim que o valor de saída é enviado, a rotina aguarda a próxima interrupção para que o ciclo se repita.

Figura 59: Fluxograma da síntese da EDO da planta no Arduino Uno



O controlador PID foi implementado pelo mesmo método da planta, mas na plataforma de prototipagem ESP32. O diagrama da Figura 60 ilustra como o PID pode ser descrito por meio de um diagrama de blocos com integradores, sem nenhum diferenciador.

Figura 60: Diagrama de blocos da EDO do controlador PID



Fonte: RODRIGUES; PAULA; LEONARDI, 2020

A função de transferência representada pelo diagrama de blocos da Figura 60 possui um ganho N cuja função é filtrar o termo derivativo, sendo normalmente um número de valor alto escolhido por simulação.

$$\frac{U(s)}{Y(s)} = k_p + \frac{k_i}{s} + \frac{s}{\frac{1}{N}s + 1} k_d \cong k_p + \frac{k_i}{s} + s k_d$$

O sistema HIL utilizou uma versão modificada da equação anterior, levando em conta a ação anti-windup do integrador. Ele então foi implementado no ESP32 para os valores das constantes K_p , K_i e K_d obtidos num projeto por cancelamento de polos e zeros.

3.3.4 Parametrização IoT

A implementação da parametrização IoT foi realizada do mesmo modo mostrado nos testes iniciais, utilizando o broker público HiveMQ e o aplicativo MQTT Dash. As credenciais de rede foram configuradas e os tópicos foram criados. Ao todo, foram seis

tópicos criados, cinco para cada parâmetro do controlador e um para o ESP32 enviar a resposta lida da planta. Assim, na função de callback, quando uma mensagem é recebida pelo ESP32, a rotina identifica o tópico de origem utilizando a informação que a variável “topic” possui e, assim que o tópico foi identificado, realiza a troca do parâmetro respectivo. Essa rotina é composta basicamente por funções if-else, que ao identificar o tópico, converte a mensagem para um valor do tipo *Double*, mesmo tipo de variável utilizada nos parâmetros.

A imagem a seguir ilustra um possível layout simples e funcional do cliente criado pelo aplicativo MQTT Dash.

Figura 61: Layout MQTT Dash

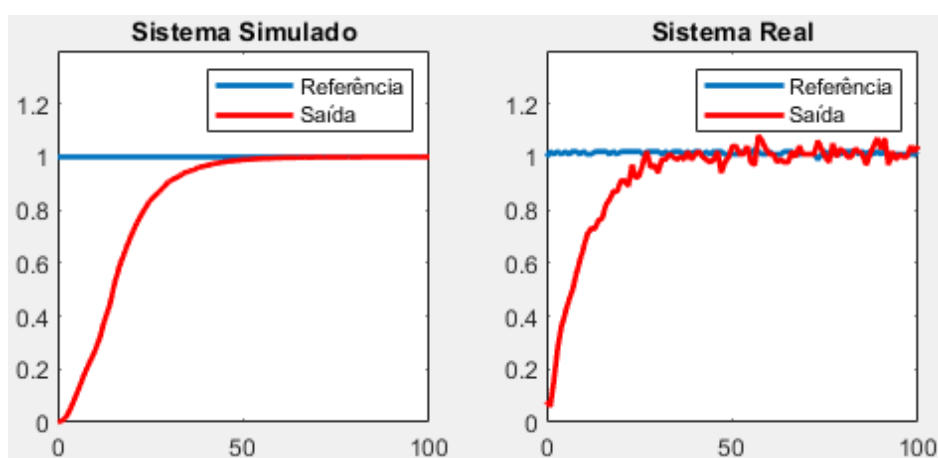


Fonte: Autor

4 RESULTADOS

O controle da planta emulada foi realizado para uma referência de 1 V, constantes K_p , K_i e K_d do controlador PID de 0,3, 0,5 e 0,1, respectivamente e constantes K_g , K_1 e K_2 da planta iguais a 1, 3 e 5, respectivamente. As constantes do controlador foram escolhidas através da técnica algébrica de cancelamento de polos (Maya & Leonardi, 2011). A Figura 62 mostra a resposta simulada e a resposta real do sistema.

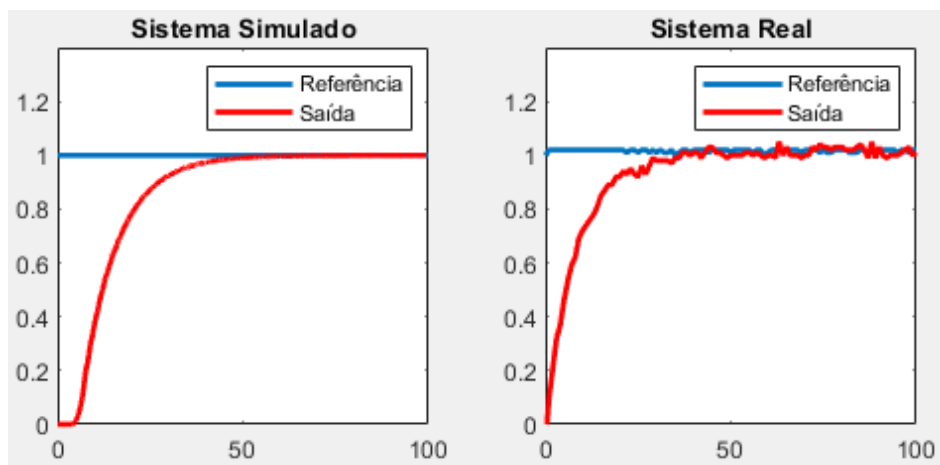
Figura 62: Resultado 1 - Resposta simulada (esquerda) e real (direita)



Fonte: RODRIGUES; PAULA; LEONARDI, 2020

A Figura 63 mostra outra simulação, desta vez com constantes K_p , K_i e K_d do controlador PID de 1, 0,3 e 0,1, respectivamente e constantes K_g , K_1 e K_2 da planta iguais a 1, 10 e 3, respectivamente.

Figura 63: Resultado 2 - Resposta simulada (esquerda) e real (direita)



Fonte: RODRIGUES; PAULA; LEONARDI, 2020

O código final do controlador e da planta se encontram, respectivamente, nos anexos C e D. Os arquivos também estão disponíveis em um repositório no site GitHub, através deste [link](#). (RODRIGUES; PAULA, 2020)

5 CONCLUSÃO

Este projeto apresentou uma nova forma de implementação de sistemas hardware-in-the-loop utilizando uma plataforma de baixo custo por meio das plataformas de prototipagem eletrônica Arduino UNO e ESP32, com parametrização via IoT. A metodologia proposta consiste em implementar na implementação em tempo real das equações diferenciais por meio de simulação dos integradores da sua representação no espaço de estados com o controle do período das rotinas sendo feito pelas interrupções por hardware de cada microcontrolador.

O código resultante possui fácil remodelagem, pois quando a ordem da equação diferencial aumenta, conseqüentemente o número de integradores irá aumentar proporcionalmente a ordem da equação. Essa relação entre ordem da equação e número de integradores resulta na facilidade e viabilidade da implementação das funções de transferência da planta ou controlador em microcontroladores de baixo custo.

Após a implementação das EDO's da planta e do controlador em suas respectivas plataformas de prototipagem eletrônica, o desempenho em malha fechada em comparação com as simulações numéricas no Simulink, se mostraram satisfatórias e dentro do esperado. Os bons resultados obtidos para essa planta e controlador em particular, sugerem que a metodologia é válida e pode ser aplicada para outras plantas ou controladores que possam ser apresentados no espaço de estados.

O microcontrolador ESP32 se mostrou estável na maioria dos testes, mas, algumas vezes, demonstrou instabilidade ao baixar os códigos para a placa. Fora esses contratempos, o dispositivo se mostrou eficiente e capaz de fazer muito além do que foi feito até o momento. Os timers do microcontrolador se mostraram precisos e sua conectividade WiFi é estável e de fácil configuração, assim como suas saídas PWM. Assim como o ESP32, o Arduino UNO demonstrou ótimo desempenho para a leitura e escrita de sinais PWM.

O projeto possui como base de parametrização do controlador os dispositivos remotos conectados via IoT. Os testes com a conexão utilizando o protocolo MQTT se mostraram satisfatórios pois não houve grandes problemas na implementação das funções e ou falta de recursos para viabilizar o uso. A parametrização IoT apresentou uma conexão razoavelmente rápida, com pouquíssimo delay, permitindo a troca de parâmetros do controlador em tempo real e a distância, sem a necessidade de ambos o cliente e o microcontrolador estarem em uma rede local. Portanto, o atraso apresentado é

consequência do broker utilizado que devido a ser um broker público, apresenta delay maior em relação a um broker privado. Em resumo, o microcontrolador trabalhou bem realizando as funções de controle e comunicação MQTT, sendo, portanto, uma alternativa viável para projetos que requerem essa interação remota.

Como proposta de continuidade deste trabalho, propõe-se a implementação do controle por modos deslizantes ou outros tipos de modelos não lineares utilizando a mesma metodologia, avaliando seus desempenhos e os limites da arquitetura e framework. Em especial, a comparação entre o controle por modos deslizantes e o controle PID seria de grande interesse para cumprir o objetivo inicial deste projeto.

REFERÊNCIAS

- ARDUINO. **Arduino Uno Rev3 | Arduino Official Store**. Disponível em: <<https://store.arduino.cc/usa/arduino-uno-rev3>>. Acesso em: 18 dez. 2020.
- BARRETO, C. G. **Tudo o que você precisa saber sobre controle PID - Automacao & Cartoons**. Disponível em: <<https://automacaoecartoons.com/2019/01/23/controle-pid/>>. Acesso em: 14 maio. 2020.
- CLARK, J. **What is the Internet of Things, and how does it work?** Disponível em: <<https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/>>. Acesso em: 23 maio. 2020.
- EDWARDS, C.; SPURGEON, S. K. **Sliding Mode Control: Theory and Applications**. 1. ed. [s.l.] CRC Press, 1998.
- ESPRESSIF SYSTEMS. **ESP32 Series**, 2020. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>
- ESPRESSIF SYSTEMS. **ESP32: Technical Reference**. 2019.
- GURU99. **Internet of Things (IoT) Tutorial for Beginners: Introduction, Basics, Applications**. Disponível em: <<https://www.guru99.com/iot-tutorial.html>>. Acesso em: 23 maio. 2020.
- HEATH, J. **PWM: Pulse Width Modulation: What is it and how does it work?** Disponível em: <<https://www.analogictips.com/pulse-width-modulation-pwm/>>. Acesso em: 27 maio. 2020.
- ISERMANN, R. K.; SCHAFFNIT, J.; SINSEL, S. Hardware-in-the-loop simulation for the design and testing of engine-control systems. **Control Engineering Practice**, v. 7, n. 5, p. 643–653, 1999.
- L., A. **Protocolos de Rede: O Que São, Como Funcionam e Tipos**. Disponível em: <<https://www.weblink.com.br/blog/tecnologia/conheca-os-principais-protocolos-de-internet/>>. Acesso em: 24 maio. 2020.
- MANCINI, M. M. M. **Internet das Coisas: História, Conceitos, Aplicações e Desafios**. São Paulo: PMI São Paulo, 2018.
- MAYA, P. A.; LEONARDI, F. **Controle Essencial**. 1ª ed. São Paulo: Pearson Education do Brasil, 2011.
- MESSNER, B.; HILL, R.; TAYLOR, J. **Control Tutorials for MATLAB and Simulink - Introduction: PID Controller Design**. Disponível em: <<http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction§ion=Contro>

IPID>. Acesso em: 22 maio. 2020.

MICROCHIP TECHNOLOGY INC. **megaAVR Data Sheet**. [s.l: s.n.]. Disponível em: <<http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf>>.

MORAIS, J. **Controle de potência via PWM - ESP32 | Portal Vida de Silício**.

Disponível em: <<https://portal.vidadesilicio.com.br/controle-de-potencia-via-pwm-esp32/>>. Acesso em: 27 maio. 2020.

MORGAN, J. **A Simple Explanation Of “The Internet Of Things”**. Disponível em: <<https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#cfd31171d091>>. Acesso em: 23 maio. 2020.

NERI, R.; LOMBA, M.; BULHÕES, G. **Protocolo MQTT**. Disponível em: <<https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/>>. Acesso em: 24 maio. 2020.

RANGER, S. **What is the IoT? Everything you need to know about the Internet of Things right now**. Disponível em: <<https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/>>.

RODRIGUES, B. L.; PAULA, G. B. DE. **GitHub - GuilhermeBDP/Sistema-HIL---Controle-PID**. Disponível em: <<https://github.com/GuilhermeBDP/Sistema-HIL---Controle-PID>>. Acesso em: 18 dez. 2020.

RODRIGUES, B. L.; PAULA, G. B. DE; LEONARDI, F. **CONTROLE DE UMA PLANTA EMULADA COM PARAMETRIZAÇÃO VIA IOT. XXCII Congresso Nacional de Estudantes de Engenharia Mecânica**, 2020.

RONALDO BARROS DOS SANTOS, S. et al. **MODELING OF A HARDWARE-IN-THE-LOOP SIMULATOR FOR UAV AUTOPILOT CONTROLLERS**. Natal: [s.n.]. Disponível em: <[http://www.ele.ita.br/~neusa/MODELING OF A HARDWARE-IN-THE-LOOP SIMULATOR FOR UAV AUTOPILOT CONTROLLERS.pdf](http://www.ele.ita.br/~neusa/MODELING_OF_A_HARDWARE-IN-THE-LOOP_SIMULATOR_FOR_UAV_AUTOPILOT_CONTROLLERS.pdf)>. Acesso em: 25 out. 2020.

SANTOS, N. **ESP32 Arduino: Timer interrupts – techtutorialsx**. Disponível em: <<https://techtutorialsx.com/2017/10/07/esp32-arduino-timer-interrupts/>>. Acesso em: 28 maio. 2020.

SANTOS, S.; SANTOS, R. **ESP32 Pinout Reference: Which GPIO pins should you use? | Random Nerd Tutorials**. Disponível em: <<https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>>. Acesso em: 27 maio. 2020.

SYSTEMS, L. T. **Controle Analógico e Digital de Motor: Manual de Experimentos 207-00**, [s.d.].

USP, E. P. DA. **Laboratório de Controle**, 2013. Disponível em:

<https://www.academia.edu/36395680/PTC2512_Apostila_2014a?email_work_card=thumbnail>

ANEXO A – CÓDIGO MQTT

```

#include <WiFi.h>
#include <PubSubClient.h>

//Definições do LED
#define PIN_LED      25

//Definições do Potenciômetro
#define POT_PIN      34
int potValue = 0;

//Definições do MQTT - NOME DA VARIÁVEL "nome do tópico"
#define TOPICO_SUBSCRIBE_LED      "topico_liga_desliga_led"
#define TOPICO_PUBLISH_VOLTS      "topico_envia_tensao"
#define ID_MQTT      "esp32_mqtt_ic"

//Definição das conexões
const char* SSID = "Nome da Rede"; //nome da rede WI-FI a se conectar
const char* PASSWORD = "Senha da Rede"; // Senha da rede WI-FI a se conectar
const char* BROKER_MQTT = "broker.hivemq.com"; //URL do broker MQTT
int BROKER_PORT = 1883; // Porta do broker MQTT

//Variáveis e objetos globais
WiFiClient espClient; // Cria o objeto espClient
WiFiServer server(80); // Porta padrão de conexão WiFi
PubSubClient MQTT(espClient); //Cria o cliente MQTT

//Declaração das funções
float faz_leitura_tensao(void);
void initWiFi(void);
void initMQTT(void);
void mqtt_callback(char* topic, byte* payload, unsigned int length);
void reconnectMQTT(void);
void reconnectWiFi(void);
void VerificaConexoesWiFiMQTT(void);

// faz leitura da tensao do potenciometro e retorna o valor
float faz_leitura_tensao(void)
{
    potValue = analogRead(POT_PIN);
    float tensao = (potValue*3.3)/4095;
    return tensao;
}

//Inicia a conexão com o Wifi
void initWiFi(void)
{
    delay(10);
    Serial.println("-----Conexao WI-FI-----");
    Serial.print("Conectando-se na rede: ");
    Serial.println(SSID);
    Serial.println("Aguarde");
}

```

```

        reconnectWiFi();
    }

//Inicializa a conexão com o Broker MQTT
void initMQTT(void)
{
    MQTT.setServer(BROKER_MQTT, BROKER_PORT);    //informa qual broker
e porta deve ser conectado
    MQTT.setCallback(mqtt_callback);              //atribui função de
callback
}

//Função que lê a informação dos tópicos subscritos
void mqtt_callback(char* topic, byte* payload, unsigned int length)
{
    String msg;

    /* obtém a string do payload recebido */
    for(int i = 0; i < length; i++)
    {
        char c = (char)payload[i];
        msg += c;
    }

    Serial.print("Chegou a seguinte string via MQTT: ");
    Serial.println(msg);

    /* toma ação dependendo da string recebida */
    if (msg.equals("1"))
    {
        digitalWrite(PIN_LED, HIGH);
        Serial.print("LED aceso mediante comando MQTT");
    }

    if (msg.equals("0"))
    {
        digitalWrite(PIN_LED, LOW);
        Serial.print("LED apagado mediante comando MQTT");
    }
}

//Reconecta ao Broker em caso de queda ou falha na conexão
void reconnectMQTT(void)
{
    while (!MQTT.connected())
    {
        Serial.print("* Tentando se conectar ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if (MQTT.connect(ID_MQTT))
        {
            Serial.println("Conectado com sucesso ao broker MQTT!");
            MQTT.subscribe(TOPICO_SUBSCRIBE_LED);
        }
        else
        {
            Serial.println("Falha ao reconectar no broker.");
            Serial.println("Havera nova tentatica de conexao em 2s");
            delay(2000);
        }
    }
}

```

```

    }
}

// Verifica situação da conexão do Wifi e do Broker
void VerificaConexoesWiFiMQTT(void)
{
    if (!MQTT.connected())
        reconnectMQTT(); //se não há conexão com o Broker, a conexão é
refeita

    reconnectWiFi(); //se não há conexão com o WiFi, a conexão é
refeita
}

// Reconecta ao Wifi
void reconnectWiFi(void)
{
    //se já está conectado a rede WI-FI, nada é feito.
    //Caso contrário, são efetuadas tentativas de conexão
    if (WiFi.status() == WL_CONNECTED)
        return;

    WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(100);
        Serial.print(".");
    }

    Serial.println();
    Serial.print("Conectado com sucesso na rede ");
    Serial.print(SSID);
    Serial.println("IP obtido: ");
    Serial.println(WiFi.localIP());
}

void setup()
{
    Serial.begin(115200);

    //Definição do pino do LED
    pinMode(PIN_LED, OUTPUT);
    digitalWrite(PIN_LED, LOW);

    //Inicia conexão WiFi
    initWiFi();

    //Inicia conexão MQTT
    initMQTT();
}

void loop()
{
    char tensao_str[10] = {0};

    /* garante funcionamento das conexões WiFi e ao broker MQTT */

```

```
VerificaConexoesWiFiMQTT();

/* Compose as strings a serem enviadas pro dashboard (campos texto)
*/
sprintf(tensao_str,"%f", faz_leitura_tensao());

/* Envia as strings ao dashboard MQTT */
MQTT.publish(TOPICO_PUBLISH_VOLTS, tensao_str);

/* keep-alive da comunicação com broker MQTT */
MQTT.loop();

/* Refaz o ciclo após 2 segundos */
delay(2000);
}
```

ANEXO B – SISTEMA HIL NO SIMULINK

Diagrama de blocos geral

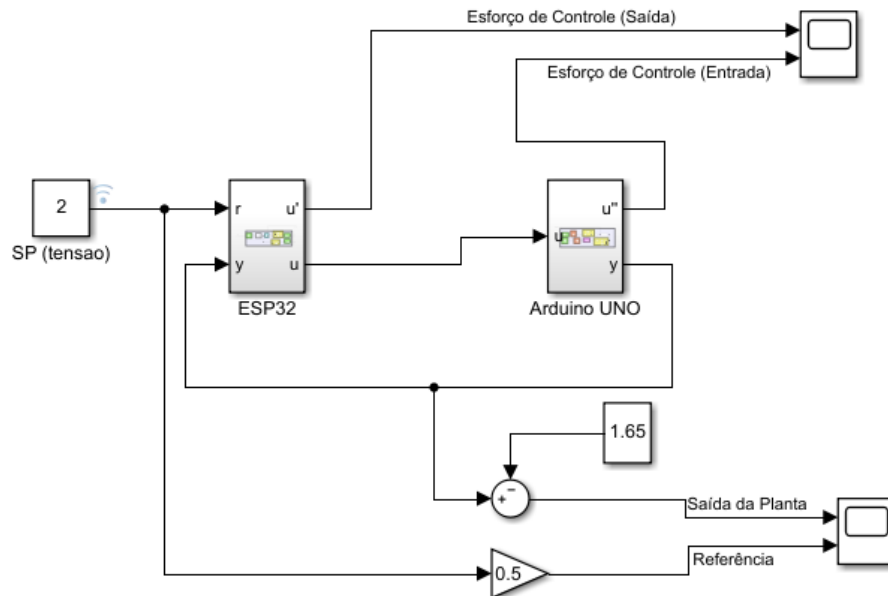


Diagrama de blocos – Subsistema ESP32

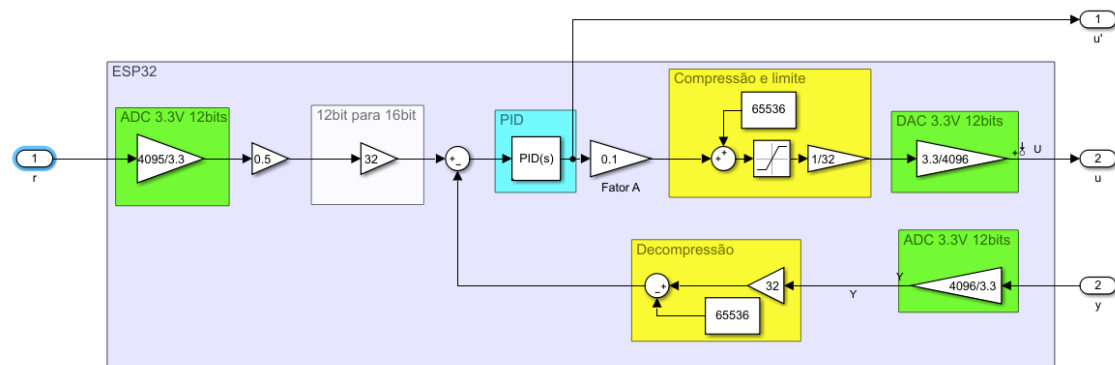
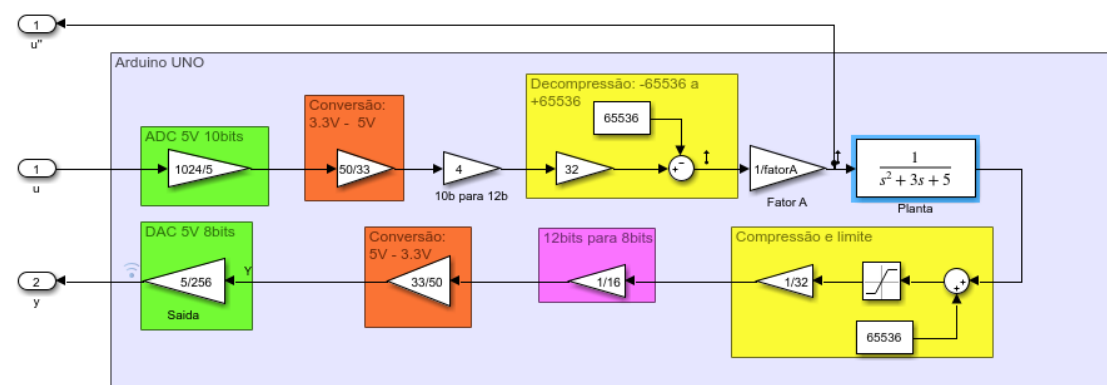


Diagrama de Blocos – Subsistema Arduino UNO



ANEXO C – CÓDIGO FINAL DO CONTROLADOR

```

#include <WiFi.h>
#include <PubSubClient.h>

// Fatores recebidos via mqtt
double kpr, kir, kdr, far, fbr;

// WIFI E MQTT
// Definições do LED
#define PIN_LED 25
// Definições do MQTT
#define TOPICO_SUBSCRIBE_KP "topico_ler_kp"
#define TOPICO_SUBSCRIBE_KI "topico_ler_ki"
#define TOPICO_SUBSCRIBE_KD "topico_ler_kd"
#define TOPICO_SUBSCRIBE_FA "topico_ler_fa"
#define TOPICO_SUBSCRIBE_FB "topico_ler_fb"
#define TOPICO_PUBLISH_RESP "topico_envia_resp"
#define ID_MQTT "esp32_mqtt_GUIBDP"

// Definição das conexões
const char* SSID = "Nome da Rede"; // Nome da rede WI-FI a se conectar
const char* PASSWORD = "Senha da rede"; // Senha da rede WI-FI a se
conectar
const char* BROKER_MQTT = "broker.hivemq.com"; //URL do broker MQTT
int BROKER_PORT = 1883; // Porta do broker MQTT

// Variáveis e objetos globais
WiFiClient espClient; // Cria o objeto espClient
WiFiServer server(80);

String header;

PubSubClient MQTT(espClient); // Cria o cliente MQTT

// Declaração das funções
void initWiFi(void);
void initMQTT(void);
void mqtt_callback(char* topic, byte* payload, unsigned int length);
void reconnectMQTT(void);
void reconnectWiFi(void);
void VerificaConexoesWiFIEMQTT(void);

// Inicia a conexão com o Wifi
void initWiFi(void)
{
    delay(10);
    Serial.println("-----Conexao WI-FI-----");
    Serial.print("Conectando-se na rede: ");
    Serial.println(SSID);
    Serial.println("Aguarde");

    reconnectWiFi();
}

// Inicializa a conexão com o Broker MQTT
void initMQTT(void)
{
    MQTT.setServer(BROKER_MQTT, BROKER_PORT); // Informa qual broker
e porta deve ser conectado

```



```

    MQTT.setCallback(mqtt_callback);           // Atribui função de
callback
}

// Função que lê a informação dos tópicos subscritos
void mqtt_callback(char* topic, byte* payload, unsigned int length)
{
    String msg;
    String topico;

    /* obtem a string do payload recebido */
    for(int i = 0; i < length; i++)
    {
        char c = (char)payload[i];
        msg += c;
    }

    topico = String(topic); // Converte em String a informação da
variável ponteiro

    Serial.println("Chegou a seguinte string via MQTT: ");
    Serial.print("Tópico: ");
    Serial.println(topico);

    // Parâmetro Kp
    if(topico=="topico_ler_kp"){
        Serial.print("Kp trocado para: ");
        int i = 0;
        String valor;
        while(msg[i]!=NULL){
            char c = (char)msg[i];
            valor += c;
            i++;
        }
        kpr = valor.toDouble();
        Serial.println(kpr);
    }

    // Parâmetro Ki
    else if(topico=="topico_ler_ki"){
        Serial.print("Ki trocado para: ");
        int i = 0;
        String valor;
        while(msg[i]!=NULL){
            char c = (char)msg[i];
            valor += c;
            i++;
        }
        kir = valor.toDouble();
        Serial.println(kir);
    }

    // Parâmetro Kd
    else if(topico=="topico_ler_kd"){
        Serial.print("Kd trocado para: ");
        int i = 0;
        String valor;
        while(msg[i]!=NULL){
            char c = (char)msg[i];

```

```

        valor += c;
        i++;
    }
    kdr = valor.toDouble();
    Serial.println(kdr);
}

// Parâmetro Fator A
else if(topico=="topico_ler_fa"){
    Serial.print("Fator A trocado para: ");
    int i = 0;
    String valor;
    while(msg[i]!=NULL){
        char c = (char)msg[i];
        valor += c;
        i++;
    }
    far = valor.toDouble();
    Serial.println(far);
}

// Parâmetro Fator B
else if(topico=="topico_ler_fb"){
    Serial.print("Fator B trocado para: ");
    int i = 0;
    String valor;
    while(msg[i]!=NULL){
        char c = (char)msg[i];
        valor += c;
        i++;
    }
    fbr = valor.toDouble();
    Serial.println(fbr);
}

// Caso qualquer mensagem chegue que não for de nenhum dos tópicos
acima
else{
    Serial.println("Mensagem recebida mas não utilizada");
}

}

// Reconecta ao Broker em caso de queda ou falha na conexão
void reconnectMQTT(void)
{
    while (!MQTT.connected())
    {
        Serial.print("* Tentando se conectar ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if (MQTT.connect(ID_MQTT))
        {
            Serial.println("Conectado com sucesso ao broker MQTT!");
            MQTT.subscribe(TOPICO_SUBSCRIBE_KP);
            MQTT.subscribe(TOPICO_SUBSCRIBE_KI);
            MQTT.subscribe(TOPICO_SUBSCRIBE_KD);
            MQTT.subscribe(TOPICO_SUBSCRIBE_FA);
            MQTT.subscribe(TOPICO_SUBSCRIBE_FB);
        }
        else

```

```

        {
            Serial.println("Falha ao reconectar no broker.");
            Serial.println("Havera nova tentatica de conexao em 2s");
            delay(2000);
        }
    }
}

// Verifica situação da conexão do Wifi e do Broker
void VerificaConexoesWiFIEMQTT(void)
{
    if (!MQTT.connected())
        reconnectMQTT(); // Se não houver conexão com o Broker, a
        conexão é refeita

        reconnectWiFi(); // Se não houver conexão com o WiFI, a conexão é
        refeita
    }

// Reconecta ao Wifi
void reconnectWiFi(void)
{
    // Se o ESP32 já estiver conectado a rede WI-FI, nada é feito.
    // Caso contrário, são efetuadas tentativas de conexão
    if (WiFi.status() == WL_CONNECTED)
        return;

    WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(100);
        Serial.print(".");
    }

    Serial.println();
    Serial.print("Conectado com sucesso na rede ");
    Serial.print(SSID);
    Serial.println("IP obtido: ");
    Serial.println(WiFi.localIP());
}

// LED INTERNO
#define pinLED 2 // Definição do pino em que o LED interno está
conectado
#define nivelLED LOW // Estado inicial do LED

// DECLARAÇÃO DOS PINOS
#define ardPin 35 // Sinal que chega do arduino(planta)
#define setPoint 34 // Sinal de referencia (potenciometro)
#define pidOut 19 // Sinal que sai do esp(pid) e entra no
arduino(planta)

// CONFIGURAÇÃO DO pidOut (sinal de saída do pid)
#define pidOutFrequency 1000 // Frequencia do sinal pwm de saída
#define canal 0 // Canal que foi utilizado (0 - 15)
#define resolution 12 // Resolução do sinal (1 - 16bits)

#define botao 33
bool estado = HIGH;

```

```

// DECLARAÇÃO DE VARIÁVEIS EXTRAS
double potValue = 0;
double dutyCycle = 0;
double ardValue = 0;

// VARIÁVEIS PID
/* saída: saída do PID, err: erro, pValue: sinal proporcional,
   dValue: sinal derivativo, iValue: sinal integrador, errAnt: erro
   anterior
   outCorrigida: saída a ser comprimida, fatorA: fator de limitação da
   saída,
   fatorB: fator de limite da ação integral, limite: limite da parcela
   integral*/
double saída, err, pValue, dValue, iValue, errAnt, outCorrigida,
fatorA, fatorB, limite;
double kp, ki, kd;
double T, kg;
float tempo = 0;

// Função que o temporizador irá chamar, para reiniciar o ESP32
void pid() {

    // Verificação de mudança dos parâmetros
    if(kp != kpr){
        kp = kpr;
    }
    if(kd != kdr){
        kd = kdr;
    }
    if(ki != kir){
        ki = kir;
    }
    if(fatorA != far){
        fatorA = far;
    }
    if(fatorB != fbr){
        fatorB = fbr;
    }

    // Início do controle

    potValue = analogRead(setPoint)*0.5;
    ardValue = analogRead(ardPin);
    potValue = potValue * 32;
    ardValue = (ardValue * 32)-65536; // Descompressão
    err = potValue - ardValue;
    pValue = err * kp; // Parcela proporcional
    iValue = (iValue + T * (ki * err) * kg); // Parcela integral
    limite = pow(2,fatorB);
    if (iValue > limite){ // Limite da parcela integral
        iValue = limite;
    }
    else if (iValue < -limite){
        iValue = -limite;
    }
    dValue = kd * (err - errAnt) / T; // Parcela derivada
    errAnt = err; // Salva o erro atual
    saída = pValue + iValue + dValue; // Esforço de controle bruto
    outCorrigida = saída*fatorA + 65536;
}

```

```

    if (outCorrigida > 131072){
        outCorrigida = 131072;
    }
    if (outCorrigida < 0){
        outCorrigida = 0;
    }
    outCorrigida = outCorrigida / 32; // Compressão
    errAnt = err; // Atualiza o erro anterior
    ledcWrite(canal, outCorrigida); // Escrita do sinal PWM
}

// TIMER
hw_timer_t * timer = NULL; // Instância do timer para fazer o controle
do temporizador (interrupção por tempo)

void startTimer(){ // Função para iniciar o timer
    /*
        hw_timer_t * timerBegin(num, divider, countUp)
        num: é a ordem do temporizador. Podemos ter quatro temporizadores,
        então a ordem pode ser [0,1,2,3].
        divider: É um prescaler (reduz a frequência por fator). Para fazer
        um agendador de um segundo,
        usaremos o divider como 80 (clock principal do ESP32 é 80MHz).
        Cada instante será  $T = 1/(80) = 1\mu s$ 
        countUp: True o contador será progressivo
    */

    timer = timerBegin(0, 80, true); // timerID 0, div 80
    // timer, callback, interrupção de borda
    timerAttachInterrupt(timer, &pid, true);
    // timer, tempo (us), repetição
    timerAlarmWrite(timer, 1000, true);
    timerAlarmEnable(timer); // Habilita a interrupção
    timerWrite(timer, 0);
}

void setup() {
    Serial.begin(9600);
    // Inicia conexão WiFi
    initWiFi();

    // Inicia conexão MQTT
    initMQTT();

    Serial.println("\nPara iniciar o programa, aperte o botão.");
    pinMode(botao, INPUT);

    while (estado != LOW){ // Botão para iniciar o programa
        Serial.println("Aperte o botão.");
        delay(500);
        estado = digitalRead(botao);
    }

    Serial.println("/--INICIANDO PROGRAMA--/");
    pinMode(pinLED, OUTPUT);
    digitalWrite(pinLED, !nivelLED);
    pinMode(pidOut, OUTPUT);

    // CONFIGURAÇÃO PWM
    ledcSetup(canal, pidOutFrequency, resolution);

```

```

    ledcAttachPin(pidOut, canal);

    startTimer(); // Inicialização do timer

    // CONFIGURACAO INICIAL DO PID
    kp = 1;
    ki = 0.3;
    kd = 0.1;
    T = 0.001;
    kg = 1;
    fatorA = 0.1;
    fatorB = 19;
    potValue = analogRead(setPoint);
    ardValue = analogRead(ardPin);
    kpr = kp;
    kir = ki;
    kdr = kd;
    far = fatorA;
    fbr = fatorB;

    Serial.println("/--PROGRAMA INICIADO--/");
}

void loop(){
    /* Garante funcionamento das conexões WiFi e ao broker MQTT */
    VerificaConexoesWiFiEMQTT();

    // Escrita na porta Serial de alguns valores do controlador
    Serial.print(" Saída (V): ");
    Serial.println((outCorrigida)*3.3/4096);
    Serial.print(" PID (esforço): "); Serial.print(outCorrigida);
    Serial.print(" Erro: "); Serial.print(err);
    Serial.print(" Ação Proporcional: "); Serial.print(pValue);
    Serial.print(" Ação Integral: "); Serial.print(iValue);
    Serial.print(" Ação Derivativa: "); Serial.print(dValue);
    Serial.println("");

    // Envio da resposta da planta para o Broker
    float resp2 = (ardValue/32)*3.3/4096;
    char resposta[10] = {0};
    sprintf(resposta, "%.2f", resp2);
    MQTT.publish(TOPICO_PUBLISH_RESP, resposta);
    /* keep-alive da comunicação com broker MQTT */
    MQTT.loop();
    delay(1000);
}

```

ANEXO D – CÓDIGO FINAL DA PLANTA

```
//DECLARAÇÃO DOS PINOS DE SINAL
#define plantIn A3 // Entrada do arduino(planta) que vem do esp32(pid)
CASO O SINAL FOR Analogico
#define plantOut 5 // Saída do arduino(planta) que entra no esp32(pid)
passando pelo filtro RC
#define botao 4

//LED INTERNO
#define pinLED 13 // Declaração do pino do LED interno
#define nivelLED LOW // Estado inicial do LED interno
bool estado = HIGH;

// Intervalo do timer
#define tempoTimer 1000

//BIBLIOTECAS
#include <TimerOne.h>

//FUNCOES
void planta();

//VARIÁVEIS DA PLANTA
/*
 * y: primeiro integrador, y2: segundo integrador; T: constante de
tempo
 * kg: ganho da planta, k1 e k2: parâmetros da planta,
 * fatorA: fator de limite da ação de controle, ucorrigida: esforço de
controle corrigido.
 * y2corrigida: saída do segundo integrador corrigida
 */

double y, y2, T, u, kg, k1, k2, sum, fatorA, ucorrigida, y2corrigida;
int aux;

void setup(){
  bool btcontrol = true; // Liga-desliga função de controle por botao
  Serial.begin(9600);

  if(btcontrol){
    Serial.println("Para iniciar o programa, aperte o botão.");
    pinMode(botao, INPUT);
    while (estado != LOW){
      Serial.println("Aperte o botao.");
      delay(500);
      estado = digitalRead(botao);
    }
  }

  pinMode(plantOut, OUTPUT);

  Serial.println("/--INICIANDO PROGRAMA--/");
  pinMode(pinLED, OUTPUT);
  digitalWrite(pinLED, !nivelLED);

  //INICIO DO TIMER
  Timer1.initialize(tempoTimer); // Inicializa o timer com o tempo
estipulado
```

```

    Timer1.attachInterrupt(planta); // Define a função "planta" como a
função a ser executada
                                     // quando a interrupção for
disparada

    //CONFIGURAÇÃO INICIAL DA ENTRADA
    u = 0;

    //CONFIGURAÇÃO INICIAL DA PLANTA
    y = 0;
    y2 = 0;
    k1 = 10;
    k2 = 3;
    T = 0.001;
    kg = 1;
    fatorA = 0.1;
    sum = 0;
    aux = 0;

    Serial.println("/--PROGRAMA INICIADO--/");
}

void loop(){
    // Escrita de alguns parâmetros da planta na porta Serial
    Serial.print("y2: "); Serial.print(y2);
    Serial.print(" Entrada: "); Serial.print(u);
    Serial.print(" Saída: "); Serial.println(y2corrigida);
}

void planta() {
    u = analogRead(plantIn);
    ucorrigida = u * 1.515151 * 4;
    ucorrigida = (ucorrigida * 32) - 65536; // Descompressão
    ucorrigida = ucorrigida * (1/fatorA);
    //-----
    sum = ucorrigida - (k1 * y) - (k2 * y2);
    y = y + T * sum * kg;
    y2 = y2 + T * y * kg;
    //-----
    y2corrigida = y2 + 65536;
    if (y2corrigida > 131072){
        y2corrigida = 131072;
    }
    if (y2corrigida < 0){
        y2corrigida = 0;
    }
    y2corrigida = y2corrigida*0.03125*0.25*0.25*0.66; // Compressão e
conversão
    if (y2corrigida > 169){
        y2corrigida = 169;
    }
    if (y2corrigida < 0){
        y2corrigida = 0;
    }
    analogWrite(plantOut, y2corrigida);
}

```