
UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE TECNOLOGIA
Departamento de Engenharia Elétrica

MATHEUS ANDRÉ MARQUES DA SILVA

# DESENVOLVIMENTO DE UM CONTROLADOR PID DE ORDEM FRACIONÁRIA EM C PARA MICROCONTROLADORES DO TIPO MSP430

CURITIBA
2019

MATHEUS ANDRÉ MARQUES DA SILVA

# DESENVOLVIMENTO DE UM CONTROLADOR PID DE ORDEM FRACIONÁRIA EM C PARA MICROCONTROLADORES DO TIPO MSP430

TCC apresentada ao curso de Engenharia Elétrica, Setor de Tecnologia, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica..

Orientador (a): Prof. Gideo Villar Leandro

CURITIBA
2019

# RESUMO

O interesse na utilização de controladores de ordem fracionária vem aumentando de forma significativa nos últimos anos de forma que se atinja uma maior robustez na performance de sistemas de controle. Para isso, muitos acadêmicos têm proposto modelos matemáticos complexos que provam a eficiência e usabilidade desses controladores em sistemas quaisquer. Neste trabalho será discutida a praticidade da utilização desse tipo de controlador implementando-o dentro de um microcontrolador do tipo MSP430. Utilizando conceitos de discretização de sistemas contínuos em tempo discreto, e comunicação serial entre o MatLab e o microcontrolador, foi construído uma arquitetura do tipo hardware-in-loop para sistemas dinâmicos discretos, onde o MSP430 realiza o cálculo do sinal de controle e retorna o valor do MatLab, onde se é calculado o erro novamente a partir de uma planta bem definida. Ainda neste trabalho são feitas comparações entre simulações de controladores discretos com a saída do Microcontrolador de modo a notar as diferenças que a aproximação discreta para lógica de programação e as limitações da comunicação serial podem causar no sistema como um todo.

Palavras-chave: Controlador PID de Ordem fracionária; Microcontrolador; MSP430; Controle Digital

# SUMÁRIO

# 1. INTRODUÇÃO

Com o avanço da tecnologia em sistemas elétricos e mecânicos, principalmente, e a necessidade das indústrias de implementar essas novas tecnologias o mais depressa e da forma mais automática possível, é preciso que os sistemas de controle desses processos acompanhem o avanço tecnológico. Para isso, novas formas de controle, além do controle tradicional criado em meados do século passado se tornam essenciais para que os processos se tornem mais eficientes como um todo. Um dos modelos que vem sendo cada vez mais estudado por cientistas e engenheiros no mundo todo de forma a garantir a robustez e a confiabilidade nos sistemas dinâmicos é o controlador PID de ordem fracionária.

Este tipo de controlador, introduzido em 1994 pelo pesquisador Igor Podlubny, se utiliza de técnicas de cálculo fracionário em cima do já conhecido controlador PID convencional. Segundo (PODLUBNY; 1994), o cálculo fracionário aplicado a sistemas dinâmicos de controle traz visibilidade de efeitos que são praticamente negligenciados em sistemas convencionais de controle como memória e hereditariedade de sinal. Com isso, em diferentes sistemas de aplicação de controle dinâmico, o modelo de ordem fracionária dá respostas mais assertivas e adequadas do que sistemas de ordem inteira. (MICHARET, 2006) mostrou que controladores de ordem fracionária superam em desempenho controladores de ordem inteira devido ao fato de que são generalizações desses controladores, sendo então os sistemas de ordem inteira casos particulares de sistemas de ordem fracionária.

## 1.1. Problematização e Contexto

Desde que foi proposto, o controlador PID de ordem fracionária tem sido estudado de forma contínua por vários pesquisadores, e sido proposto com diferentes modelagens com diferentes métodos de sintonização. A literatura atual vem se enriquecendo nos últimos anos com desenvolvimentos teóricos e desdobramentos matemáticos da teoria de sistemas de ordem fracionária de forma a confirmar sua

eficiência em diferentes tipos de aplicações. Porém, uma abordagem mais prática tem sido pouco explorada. Dessa forma, existe a necessidade de verificar o comportamento real de um controlador desse tipo, bem como a identificação de possíveis limitações do protótipo e cenários onde ele se comporta como o esperado.

# 2. OBJETIVOS

## 2.1. Objetivo Geral

O objetivo geral desse projeto é a implementação de um controlador PID de ordem fracionária em um microcontrolador do tipo MSP430 para um sinal qualquer em uma planta qualquer.

## 2.2. Objetivos Específicos

- Simular em MatLab o comportamento de um controlador FOPID comparando-o a um PID convencional
- Utilizar métodos de aproximação de operadores derivativos e integrativos de ordem não fracionária para operadores de ordem inteira
- Utilizar métodos de discretização de sistemas no domínio S para o domínio Z
- Realizar comunicação entre MSP430 e MatLab para simulação do controlador em arquitetura de *"hardware-in-loop"*

## 2.3. Justificativa

Com a necessidade descrita na seção 1.1, a justificativa do presente trabalho se deve ao diferencial que ele terá em relação à literatura disponível na área de controladores de ordem fracionária. O levantamento de vantagens, desvantagens, limitações e ganhos de uma aplicação do controlador PID de ordem fracionária em um microcontrolador específico devem gerar uma base de conhecimento importante para futuros desenvolvedores e pesquisadores da área de teoria de sistemas de controle e microcontroladores.

# 3. Controle de Processos

O controle automático de processos tem como função garantir a operabilidade, segurança e atendimento dos requerimentos de desempenho de sistemas dinâmicos. Estes sistemas são desenvolvidos com aplicações das mais diferentes formas: sistemas de temperatura, sondas espaciais, sistemas robóticos, além de qualquer tipo de controle de grandezas dimensionais, como pressão, temperatura, umidade, iluminação, etc...

Controlar, por definição (OGATA, 2011), significa medir uma variável controlada do sistema e manipular o sinal de entrada de forma a corrigir ou limitar efeitos indesejáveis no sistema. A partir disso, de acordo com o tipo de efeito que se deseja introduzir ou retirar do sistema, diferentes abordagens de controle podem ser utilizadas. Neste capítulo serão discutidos controladores PID, sua variante PID de ordem fracionária, e suas características.

## 3.1. Controlador PID

Controladores do tipo PID (Proporcional Integral Derivativo) são certamente os mais utilizados hoje em dia na indústria. Isso se deve ao fato de ser o mais simples de se construir, além de apresentar um bom desempenho em casos onde se desconhece o sistema dinâmico que representa a planta.

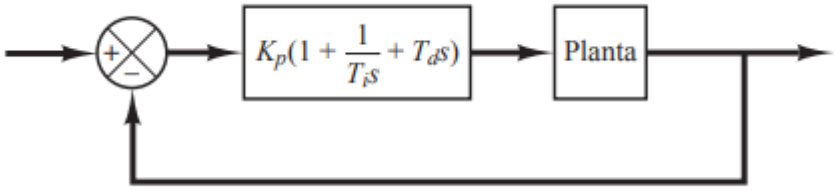


**Figura 1 – Sistema de malha fechada de controle PID de uma Planta**

Onde temos um controlador definido por:

$$C(s) = K_p + \frac{K_I}{S} + K_d s \quad (1)$$

A usabilidade do PID comum pode ser explicada pelo conhecimento do que cada um de seus 3 parâmetros interferem no sistema. Isso se torna útil para aplicação em plantas que não são conhecidas pelo projetista ou em processos onde não há a necessidade de uma otimização da resposta, mas sim apenas o interesse em algum dos parâmetros de resposta.

| | $K_p$ | $K_i$ | $K_d$ |
|---|---|---|---|
| **Tempo de subida** | Diminui | Diminui | Muda pouco |
| **Sobressinal** | Aumenta | Aumenta | Diminui |
| **Tempo acomodação** | Muda pouco | Aumenta | Diminui |
| **Erro no equilíbrio** | Diminui | Elimina | Nenhum |

**Figura 2 - Efeitos que o aumento das constantes do controlador PID têm sobre a resposta do sistema em malha fechada (SERAPIÃO, 2011)**

### 3.2. Controlador PID de Ordem Fracionária

O controlador PID de Ordem Fracionária é uma generalização do PID convencional. Nele tem-se um integrador de ordem λ e um derivador de ordem μ, onde λ e μ podem ser quaisquer números reais.

Este tipo de controlador apresenta uma melhor resposta (PRODLUBNY, 1997) para controle de sistemas de ordem inteira muito alta e sistemas cuja planta era também de ordem fracionária.

$$C_{PI^{\lambda}D^{\mu}}(s) = K_p + \frac{K_I}{s^{\lambda}} + K_d s^{\mu} \quad (2)$$

**Figura 2 - Controlador PID de ordem fracionária**

Pela figura 4, pode-se notar como o PID convencional está contemplado no controlador FOPID:

- Para λ = 0 e μ = 0: Controlador Proporcional Convencional
- Para λ = 1 e μ = 0: Controlador Proporcional Integrador Convencional
- Para λ = 0 e μ = 1: Controlador Proporcional Derivativo Convencional
- Para λ = 1 e μ = 1: Controlador Proporcional Integrador Derivativo Convencional

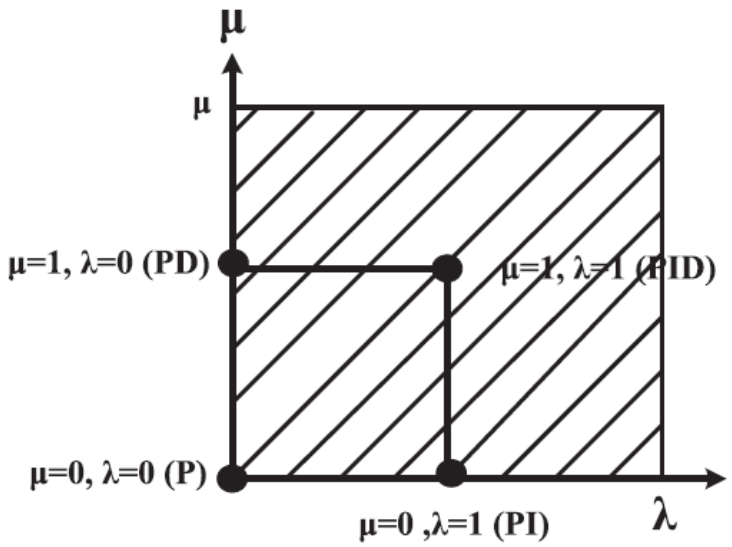


**Figura 3 – Plano de possibilidades de um controlador FOPID para $K_P$, $K_I$, e $K_D$ fixos.**

Outra vantagem do controlador PID de ordem fracionária é a menor sensibilidade a mudanças nos parâmetros do sistema controlado (XUE, 2006). Isso se deve aos dois graus extras de liberdade (λ e μ além de $K_p$, $K_i$, e $K_d$) que este controlador tem em relação ao PID convencional. (MATIGNON, 1998) demonstrou que o FOPID tem uma

maior região de estabilidade em relação ao PID convencional, conforme abaixo:

$$|\arg(\mathrm{eig}(\boldsymbol{A}))| > q\frac{\pi}{2} \tag{3}$$

Onde 0 < q < 1, para q sendo a ordem do sistema e eig(**A**) são os autovalores da matriz que determina o sistema, ou os pólos da função de transferência do sistema em questão. Com isso, a região de estabilidade não se limita ao lado esquerdo do plano real-imaginário e o sistema pode ser estável com pólos à direita do plano real-imaginário.

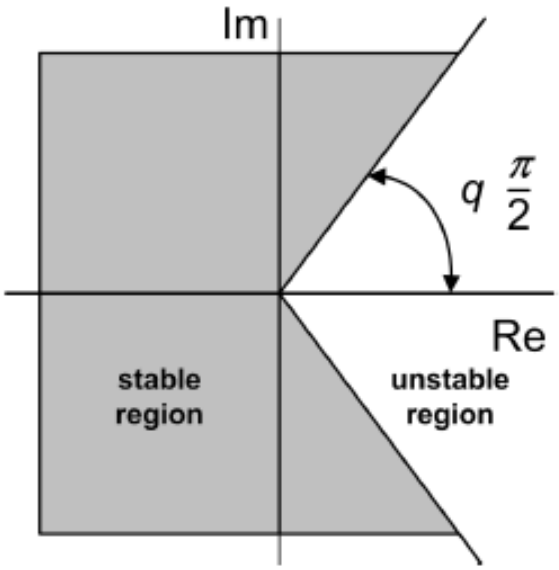


**Figura 3 - Região de estabilidade de um PID de ordem fracionária**

### 3.2.1. Métodos de Aproximação Discreta

Para que se possa reproduzir um sistema de ordem fracionária em um ambiente discreto como um microcontrolador ou ferramentas de processamento como o MatLab é essencial que haja uma discretização do sistema de interesse (PETRÁŠ, 2003). Isso se deve ao fato de que em geral os sistemas de ordem fracionária não têm solução analítica direta (FAIEGHI, 2011), sendo assim necessária uma solução numérica para cada caso.

Em geral, temos dois métodos de aproximação de um sistema de ordem fracionária qualquer; o método direto, onde se aplica uma transformação do componente $s^r$ para um domínio $z^r$ e depois aproxima-se o polinômio de ordem fracionária $G(z^r)$ por uma expansão em série; e o método indireto, onde se faz uma primeira aproximação do sistema de ordem fracionária $s^r$ para $s^n$ e depois aplica-se uma transformação para o domínio Z, já em ordem inteira. Esta seção mostrará como cada uma se dá, e as vantagens e desvantagens de se aplicar cada uma delas (AL ALAOUI, 2009).

#### 3.2.1.1. Método Direto

No método direto, se aplica o sistema de ordem fracionária diretamente em uma função geradora de aproximação inteira. Essas funções geradoras são séries de potência e são escolhidas pelo projetista de acordo com o que a resposta que se julga mais adequada para o problema em questão.

Neste método, tem-se uma primeira discretização simples do sistema contínuo (domínio S) de ordem fracionária para um sistema discreto (domínio Z) de ordem fracionária através de uma transformação do tipo s = G(z) simples, para a partir daí aproximar por expansão em série de potência num sistema de ordem inteira.

A transformação do domínio S para domínio Z pode ser feita a partir de métodos bem conhecidos como transformação de Euler, de Tustin ou por *Backward Difference*:

$$s \cong \frac{z-1}{T} \tag{4}$$

**Método de transformação de Euler**

$$s \cong \frac{z-1}{zT} \tag{5}$$

**Método de transformação *Backward Difference***

$$s \cong \frac{2}{T}\frac{z-1}{z+1} \tag{6}$$

**Método de transformação de Tustin**

A partir da função G(z) utilizada para substituir s no sistema de ordem fracionária, se deve aplicar também o expoente real r na função. Dessa forma, para uma transformação de Tustin, teríamos uma função $[G(z)]^r$ na forma:

$$s^r \cong \left[\frac{2}{T}\frac{z-1}{z+1}\right]^r = \frac{2^r}{T^r}\left[\frac{z-1}{z+1}\right]^r \tag{7}$$

Com uma função $G_{fim}(z)$ dependendo de uma expansão de um polinômio de z (para eq. 7 esse polinômio é z-1 e z+1) de ordem r, sendo r um número real, a aproximação final do sistema de ordem fracionária para um sistema de ordem inteira será dada por uma expansão em série. O método de expansão dependerá do que mais for apropriado para o tipo de resposta que se espera ((PETRÁŠ, 2003) expansão de Taylor-McLaurin; expansão em séries de potência; expansão em frações contínuas; etc..).

Independentemente do tipo de expansão que se escolha, em geral, a digitalização de um sistema de ordem fracionária para uma forma programável é complicada e custosa em termos de processamento. Para sistemas cuja banda é muito menor que 1/T ou muito maior que 1/T, essas aproximações tendem a ter métodos mais simples, porém se tornam mais complexos para uma generalização do método de

aproximação.

### 3.2.1.2. Método Indireto

No método indireto, a lógica em relação ao método direto é invertida. Primeiramente ocorre a aproximação do sistema de ordem fracionária no domínio S para um sistema de ordem inteira, ainda no domínio S, para depois ser feita a conversão em domínio Z.

A aproximação da parcela $s^r$ para $f(s^n)$ (parcela de ordem fracionária para uma parcela de ordem inteira) pode ser feita através de uma transformação direta por aproximação de Oustaloop, dada uma banda de interesse [$\omega_1$; $\omega_N$]. O parâmetro k deve ser ajustado para que a o termo tenha ganho 1 em 1rad/s (OUSTALOOP, 1991).

$$s^q = k\prod_{n=1}^{N}\frac{1+\frac{s}{\omega_{zn}}}{1+\frac{s}{\omega_{pn}}} \qquad q>0 \tag{8}$$

Para q sendo um número real, a aplicação em (8) retornará uma função transferência de ordem inteira. Ex: Para $s^{0,1}$, com banda de interesse [0.01;100] rad/s e N=3, temos:

$$s^{0,1} \cong \frac{s^3 + 2800s^2 + 46744s + 4642}{1585s^2 + 4437s + 7357} \tag{9}$$

Com uma função H(s) bem definida e de ordem inteira, pode-se aplicar qualquer das transformações de domínio S para domínio Z (por exemplo uma das equações 4, 5 ou 6) para que se tenha a forma final do sistema discretizado de ordem inteira. Por exemplo, a partir da aproximação de ordem inteira da equação 9, podemos aplicar transformação de Tustin, onde teremos a seguinte função transferência no domínio Z.

$$C(z) = \frac{2.832z^3 - 0.2674z^2 - 2.763z + 0.249}{z^3 - 0.7166z^2 - 0.9597z + 0.7569} \quad (10)$$

### 3.3. Função de Transferência em tempo discreto

Para que se consiga implementar um controlador digital, a partir da função transferência discreta em domínio Z do sistema em questão, é necessária uma transformação para o domínio do tempo. Dessa forma, o sinal de entrada do sistema que é dado em tempo discreto pode ser tratado como tal.

Uma função de transferência H(z), cuja ordem do numerador é M e do denominador é N, e as constantes dos polinômios são representadas por $b_k$ e $a_k$, tem a seguinte forma:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M=1} b_k z^{-k}}{\sum_{j=0}^{N-1} a_k z^{-j}} \quad (11)$$

Com isso, aplicando a transformada inversa de Z de deslocamento nos termos, e a propriedade da divisão, teremos uma saída discretizada dada por:

$$y[n] = \sum_{k=0}^{M-1} b[k]x[n-k] - \sum_{j=0}^{N-1} a[j]x[n-j] \quad (12)$$

A equação 12 mostra que, tendo uma entrada bem definida em tempo discreto (x), com uma função de transferência bem definida, obtemos a saída do sistema utilizando álgebra simples. Esta equação é implementável em qualquer linguagem de programação, sendo assim a que será usada para este trabalho.

# 4. Microcontroladores

Microcontroladores são circuitos integrados com capacidade de processamento e memória que desempenham determinada tarefa dentro de um sistema. Em geral, microcontroladores tem periféricos de entrada e saída que auxiliam na comunicação com sistemas externos como sensores e atuadores. Neste trabalho será usado o microcontrolador MSP430.

## 4.1. MSP430

O MSP430 é um microcontrolador do tipo RISC (*Reduced Instruction Set Computer)*, ou seja, pertence a uma classe de microcontroladores que favorece o desenvolvimento a partir de estruturas de comando e instruções mais simples. Em geral é utilizado para pequenas aplicações como sistemas embarcados de baixa potência.

Embora seja idealizado para uso em pequenas aplicações, o MSP430 conta com uma variedade de possibilidades de configuração como osciladores internos, temporizador para PWM, conversores analógico-digitais, entre outros. Neste trabalho, usaremos a configuração de comunicação Serial UART. O microcontrolador utilizado será o MSP430g2553x

**Figure 4 - Microcontrolador MSP430**

Para embarcar uma aplicação em um MSP430, é necessário um ambiente de desenvolvimento que se comunique com o microcontrolador. As formas de se programar em MSP430 são variadas. Diversas IDEs e depuradores são disponíveis no mercado para usos pessoais e comerciais. Neste trabalho, será usado o **Energia IDE**.

## 4.2. Energia IDE

O Energia IDE é um ambiente de desenvolvimento *open*-source para microcontroladores baseados na configuração do MSP430 da Texas Intruments. Alguns dos microcontroladores mais utilizados no mercado (microcontroladores das famílias MSP430, MSP432, TM4C (Tiva µC)) são suportadores pelo Energia.

O ganho do Energia em relação a outros ambientes é sua facilidade de programação e depuração, dado que a plataforma é baseada no framework utilizado para programação em Arduino.


**Figure 5 - Tela de programação do Energia - similar à do Arduino**

# 5. Metodologia e resultados

Neste capítulo serão discutidas as premissas do protótipo construído; as suposições feitas durante a construção do modelo; a metodologia utilizada para chegar no protótipo final; e a apresentação dos resultados.

## 5.1. Modelagem de Malha Fechada

Para que se possa ter um controlador FOPID colocado dentro do MSP430, é necessário escolher uma arquitetura de comunicação de dados, bem como um modelo de como testar o protótipo depois de pronto.

O controlador será submetido a uma malha fechada, para uma planta qualquer, para um sinal qualquer, no modelo tradicional de malha fechada, como o abaixo:

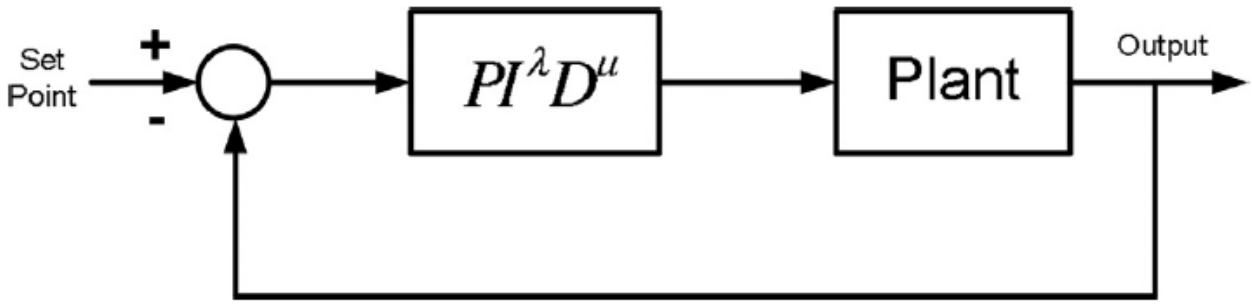


**Figura 6 - Modelo de malha fechada para o controlador FOPID**

Por muitas vezes não existirem soluções analíticas para sistemas desse tipo (FAIEGHI, 2011), é preciso definir a planta e o controlador FOPID que será utilizada para aplicação do sinal de controle e assim verificar o comportamento numérico do que foi implementado.

A planta e o controlador utilizados foram escolhidos aleatoriamente, por tentativa e erro, observando a saída do sistema. A planta e o controlador FOPID escolhidos para análises e testes do protótipo são os seguintes:

$$P(s) = \frac{0.06608}{5s + 0.0013} \tag{13}$$

Planta

$$C(s) = 5.585 + \frac{0.1}{s^{0.8}} + 2.5s^{0.7} \tag{14}$$

Controlador FOPID

### 5.1.1. Hardware-in-loop

A execução do controle em malha fechada foi realizada utilizando o conceito de hardware-in-loop. Com a aplicação deste conceito, é possível separar os papéis desempenhados por cada sistema dentro da execução de cada um dos blocos.

Para que o MSP430 execute apenas a função de um controlador FOPID, foi utilizado o conceito de hardware-in-loop, onde o MatLab realiza os outros papéis dentro do sistema completo em malha fechada. O desenho de como será executado o processo se dará da seguinte forma:

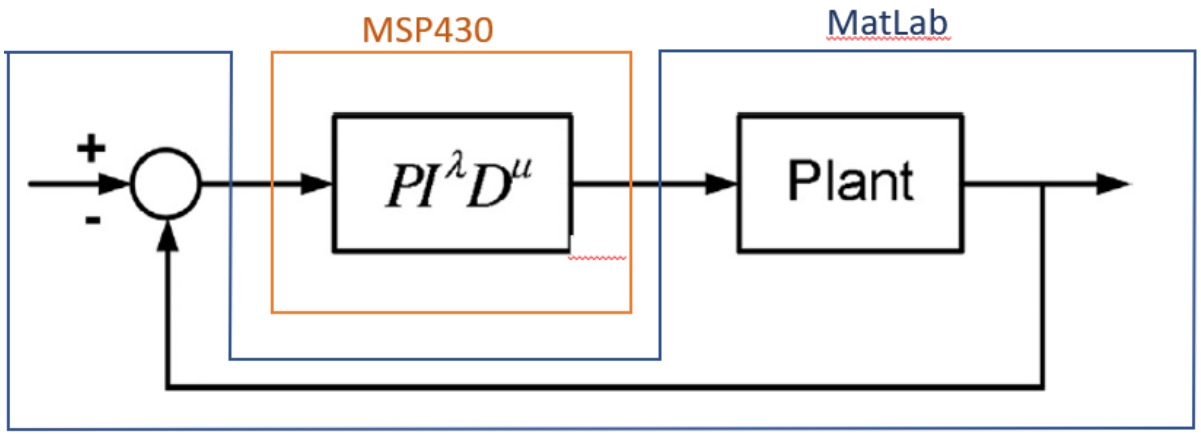


**Figura 7 - Arquitetura da solução proposta**

Neste modelo, o MSP430 ficará exclusivamente com o papel de receber a informação do erro calculado a partir da referência de entrada e transmitir o sinal de controle de volta para dentro do MatLab.

Essa troca de dados é feita a partir de comandos de leitura e escrita tanto do lado do MSP430 como do lado do MatLab. A escrita do lado do MatLab é realizada de forma direta, através do comando:

```
fprintf(s, '%f', sinalErro);
```

Onde é passado o parâmetro de sinal de erro especificando seu tipo *float.* A recepção desta informação do lado do MSP430 é feita após verificação de que há informação na porta Serial, e recebendo a informação convertendo o dado para *float*, como abaixo:

```
if(Serial.available())
  erro = Serial.parseFloat();
```

Após o controlador FOPID realizar os cálculos dentro do MSP430, a transmissão do sinal de controle para o MatLab é um pouco menos direto do que a recepção. Pelo fato de a escrita na porta serial a partir do MSP430 só aceitar dados em formatos de bytes, e assim não sendo possível escrever um valor *float* como feito no MatLab, é

necessária uma conversão de *byte array* em *float*. A escrita do MSP430 do sinal calculado no canal de comunicação ocorre da seguinte forma:

```
byte *b = (byte *) &sinalControle;
  Serial.write(b,4);
```

O MatLab recebe a informação da seguinte forma:

```
h1 = fread(s,4)
```

A variável h1 recebe um vetor de números inteiros, de tamanho 4, passa por uma transformação para hexadecimal, e somente então, se torna o número do tipo *float* que o MSP430 transmitiu.

## 5.2. Discretização do modelo

Como discutido anteriormente, o MSP430 terá apenas o bloco do controlador FOPID implementado, portanto, a necessidade de discretização do modelo ocorre apenas para a função transferência do controlador. Todo o resto da malha, calculado dentro do próprio MatLab, será calculado em tempo contínuo (domínio S), sendo desnecessária a discretização da planta.

Como discutido anteriormente, a transferência do MSP430 para o MatLab é de 4 bytes, havendo assim pouco ganho em realizar aproximações onde os índices da função transferência discreta tenham mais de 3 digitos significativos.

Considerando um tempo de amostragem de 0,1 segundos, para determinar qual ordem de aproximação utilizar para o protótipo foi realizada a aproximação de Euler no controlador para diferentes ordens. A aproximação de ordem 2 já traz pouca melhora em termos de dígitos significativos, dada a limitação da transmissão do MSP430. A aproximação de segunda ordem do controlador FOPID escolhido para o sistema é a seguinte:

$$C_{n=2}(z) = \frac{1.831 - 1.491z^{-1} + 0.06085z^{-2} + 0.0036z^{-3}}{0.1995 - 0.1596z^{-1} - 0.01596z^{-2}} \quad (15)$$

Com isso, a ordem de aproximação para o controlador será de 1. A aproximação utilizada para o protótipo é a seguinte:

$$C_{n=1}(z) = \frac{1.831 - 1.491z^{-1} + 0.224z^{-2}}{0.1995 - 0.1596z^{-1}} \quad (16)$$

### 5.3. Sistema embarcado no MSP430

O código embarcado no MSP430 basicamente recebe as informações de erro, como discutido na seção 5.2.1, calcula o sinal de controle para estas informações recebidas e joga essa informação na porta serial.

O cálculo do sinal de controle dentro do microcontrolador é feito com o input do último sinal de erro, e dos dois predecessores, por se tratar de uma aproximação para deslocamento de tempo máximo de 2 amostras (aplicação do sinal em tempo discreto da equação 12 no controlador FOPID discretizado da equação 16). A descrição completa de como é realizado o cálculo do sinal de controle discreto dentro do MSP está disposto no Apêndice A.

### 5.4. Código de execução MatLab

O trecho do protótipo executado dentro do MatLab, que vai desde a entrada do sinal de controle na planta até o cálculo do erro em relação a referência de entrada, é construído baseado na função *lsim()*, que simula a resposta no tempo para sistemas dinâmicos para entradas qualquer.

As entradas da função são: a função transferência da planta em domínio S (equação 14); o vetor com todo o histórico de sinal de controle; o vetor de tempo no qual o sistema será simulado sobre.

## 5.5. Simulações e Resultados

Para que se houvesse possibilidade de comparação do que foi construído no MSP430 com o que seria um FOPID real, foram realizadas algumas simulações do sistema de malha fechada no domínio S utilizando o pacote de cálculo fracionário do Simulink. Os controladores utilizados nessa seção são os discutidos anteriormente: controlador FOPID contínuo da equação 14 e os controladores discretizados pelo MatLab, além do controlador discreto da equação 16 (para ordem de aproximação igual a 2).

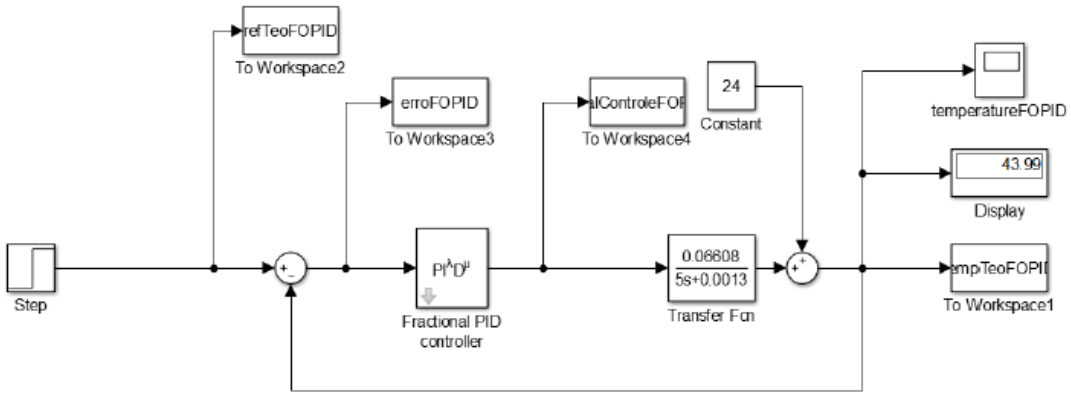


**Figura 8 - Esquemático de simulação de um FOPID contínuo**

Também é possível simular, ainda dentro do MatLab, o controlador já discretizado, variando as técnicas de discretização e a ordem da discretização. O esquemático com o controlador discreto é o abaixo:

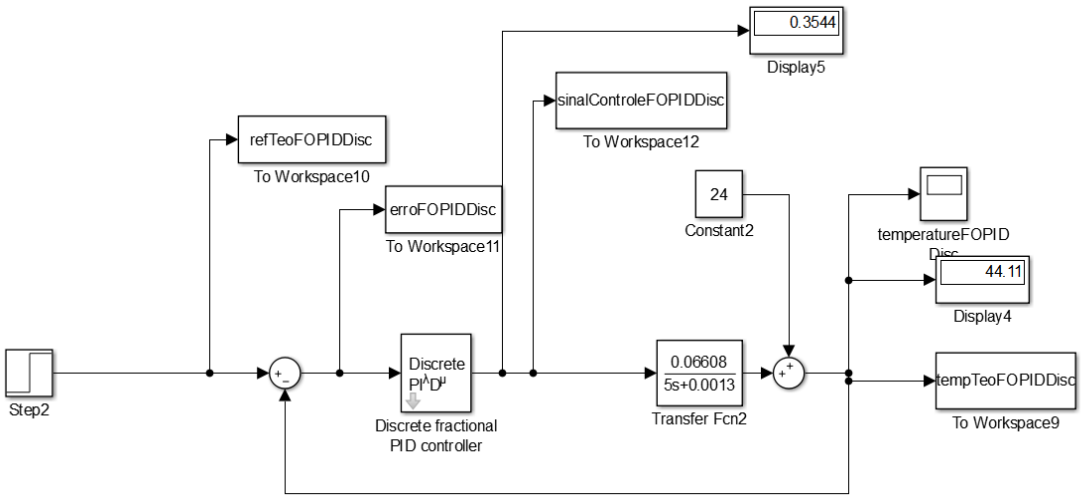


**Figura 9 - Esquemático da malha fechada com controlador discreto dentro do simulink**

### 5.5.1. FOPID no MatLab

Para comparar o controlador FOPID contínuo com o controlador FOPID discreto, ainda dentro do MatLab, foram utilizados os seguintes parâmetros.

- SetPoint = 44
- Valor inicial = 24
- Tempo de amostragem = 0.1 segundos (mesmo utilizado no MSP430)
- Tempo de simulação = 400 segundos

#### 5.5.1.1. FOPID Discreto x FOPID Contínuo no MatLab

A primeira comparação é de um FOPID contínuo em relação a um controlador discreto de alta ordem (6) e discretização do tipo Bilinear. Já é possível notar desvios que a discretização causa no sistema, mesmo com alta ordem de aproximação.

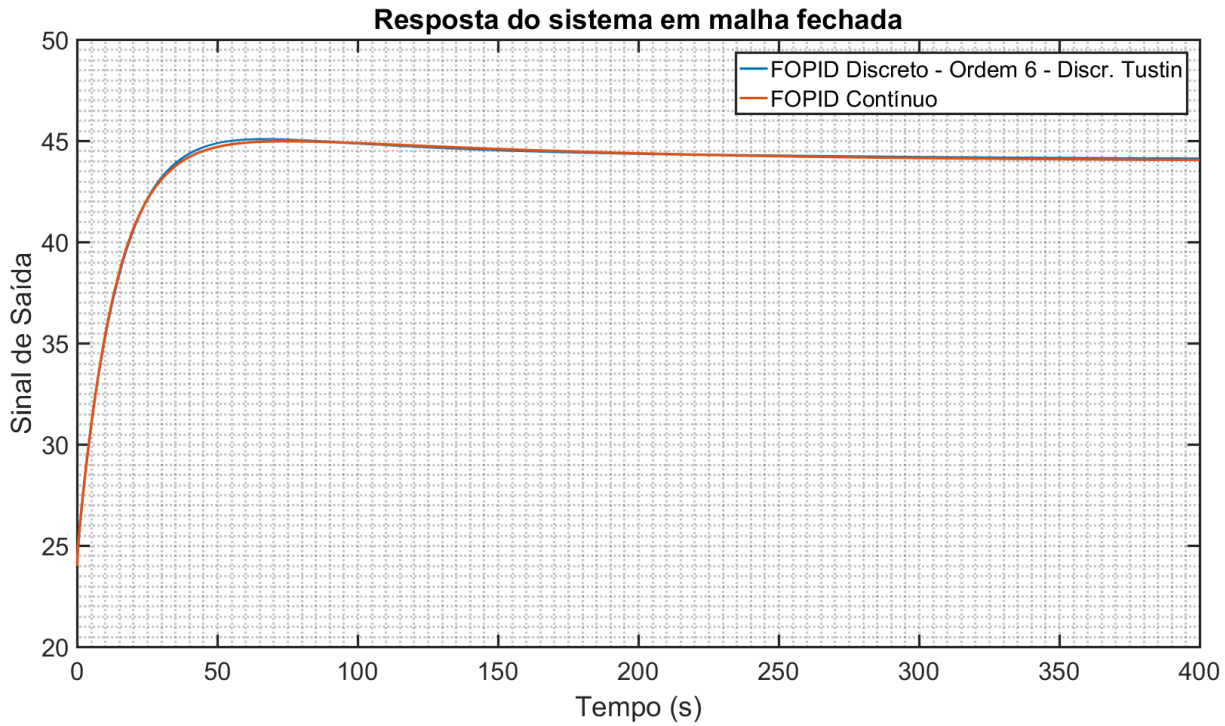


**Figura 10 - Comparação da resposta de um sistema com FOPID contínuo e um FOPID discretizado pelo método Bilinear de ordem 6**

O sinal de controle na saída do controlador demonstra mais claramente a diferença de um sistema em tempo contínuo para um sistema em tempo discreto.

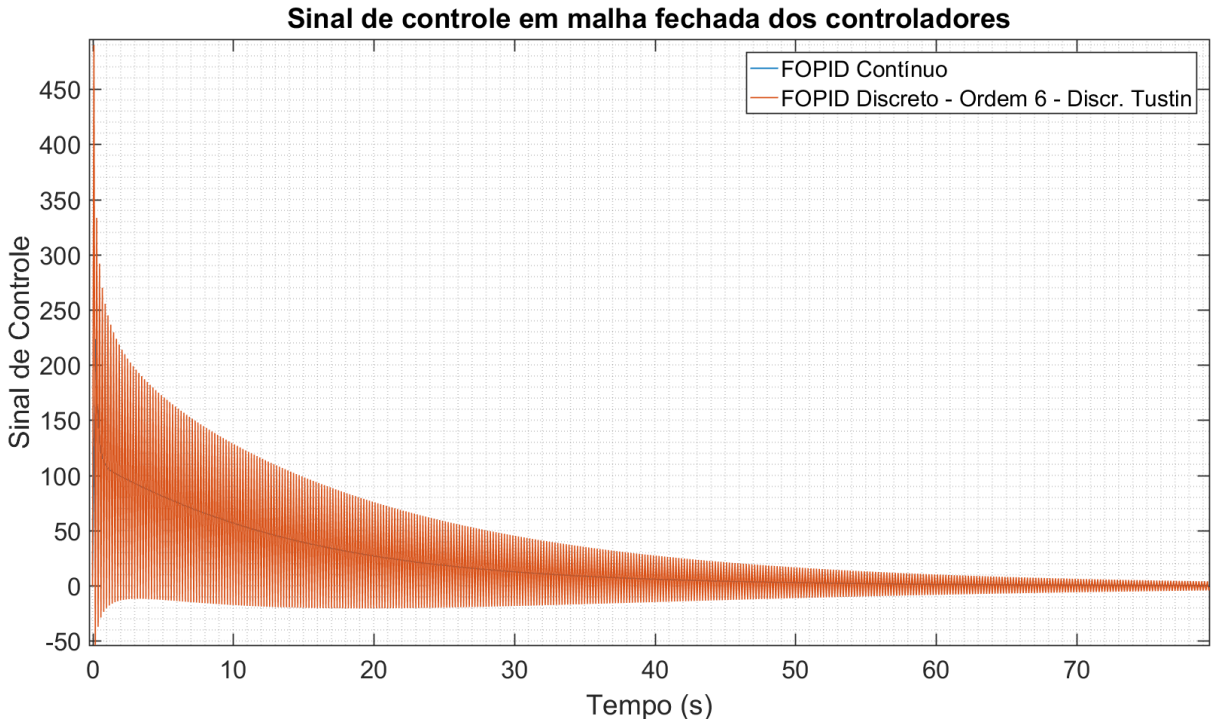


**Figura 11 - Comparação do sinal de controle de um controlador FOPID contínuo e um FOPID discretizado pelo método Bilinear de ordem 6**

### 5.5.1.2 FOPID Discreto x FOPID Discreto no MatLab

Comparado o controlador contínuo com o controlador discreto de alta ordem, e já notado que a diferença já é bem grande, foi simulada uma comparação entre o FOPID Discreto de ordem 6 discretizado pelo método bilinear com um FOPID Discreto de ordem 2, também discretizado pelo método bilinear, que em teoria deve ser mais próximo do controlador construído dentro do MSP430.

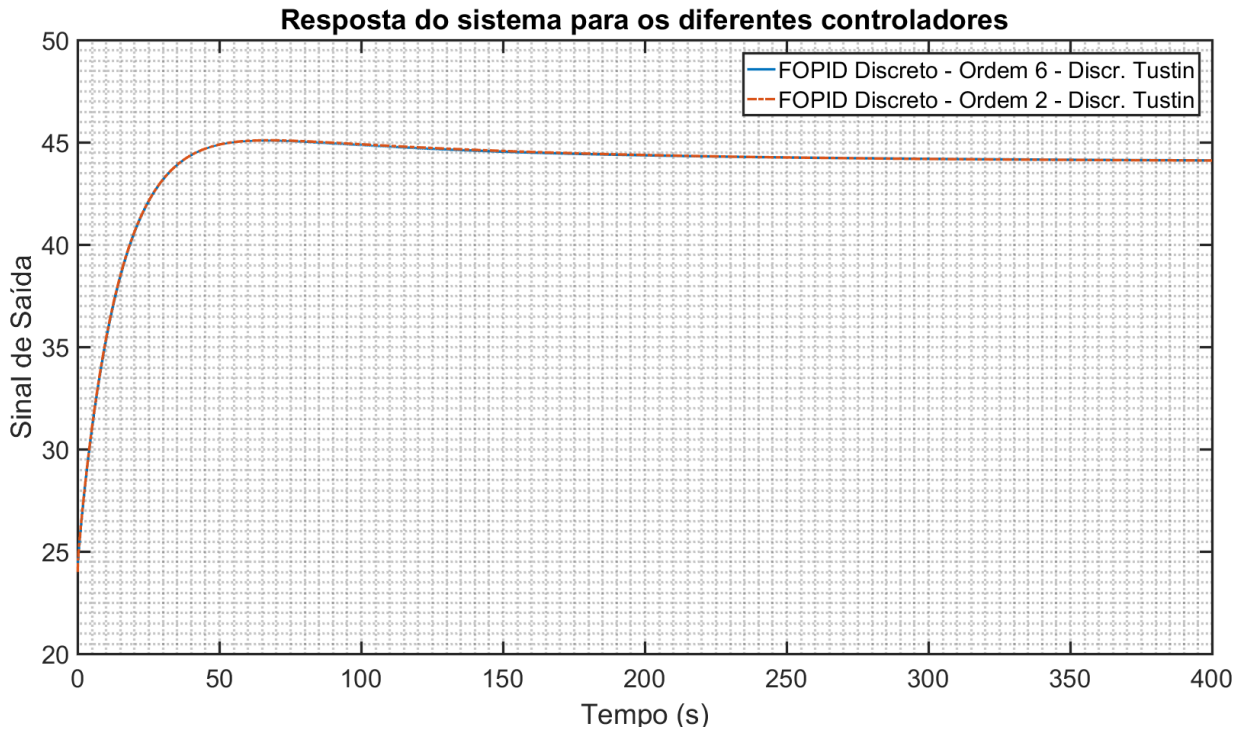


**Figura 12 -Comparação da resposta de um sistema com FOPID discreto de ordem 2 e um FOPID discreto de ordem 6**

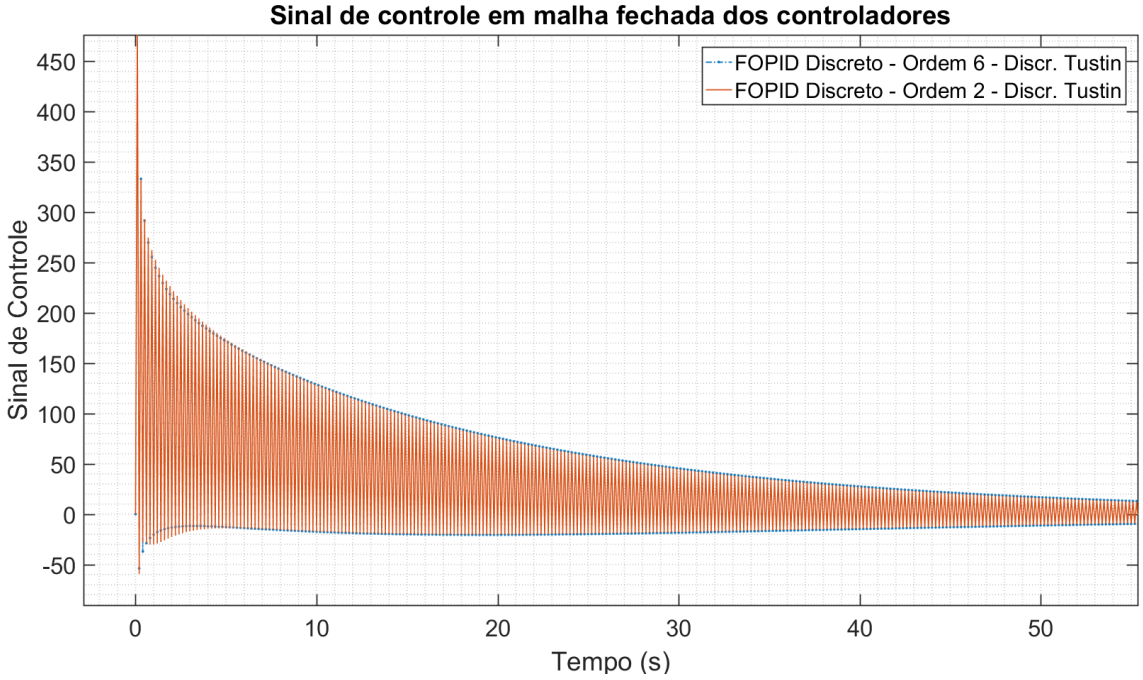


**Figura 13 - Comparação do sinal de controle de um controlador FOPID discreto de ordem 2 e um FOPID discreto de ordem 6**

Com as comparações entre gráficos discretos de diferentes ordens e contínuo contra discreto, pode-se perceber que a maior perda está na discretização a partir do controlador contínuo, já que o controlador discreto de ordem 2 e de ordem 6 estão bem mais próximos que o controlador discreto de ordem 6 está do controlador contínuo.

#### 5.5.1.3 FOPID Discreto MSP430

A saída do controlador FOPID discreto do MSP430 foi dada em tempo não real, devido ao tempo de processamento da função lsim() do MatLab, que varia de acordo com o momento em que se retira a amostra e o tamanho do vetor de entrada. A primeira comparação do resultado do MSP430 é com o seu par, o controlador discreto de ordem 2 do Simulink.

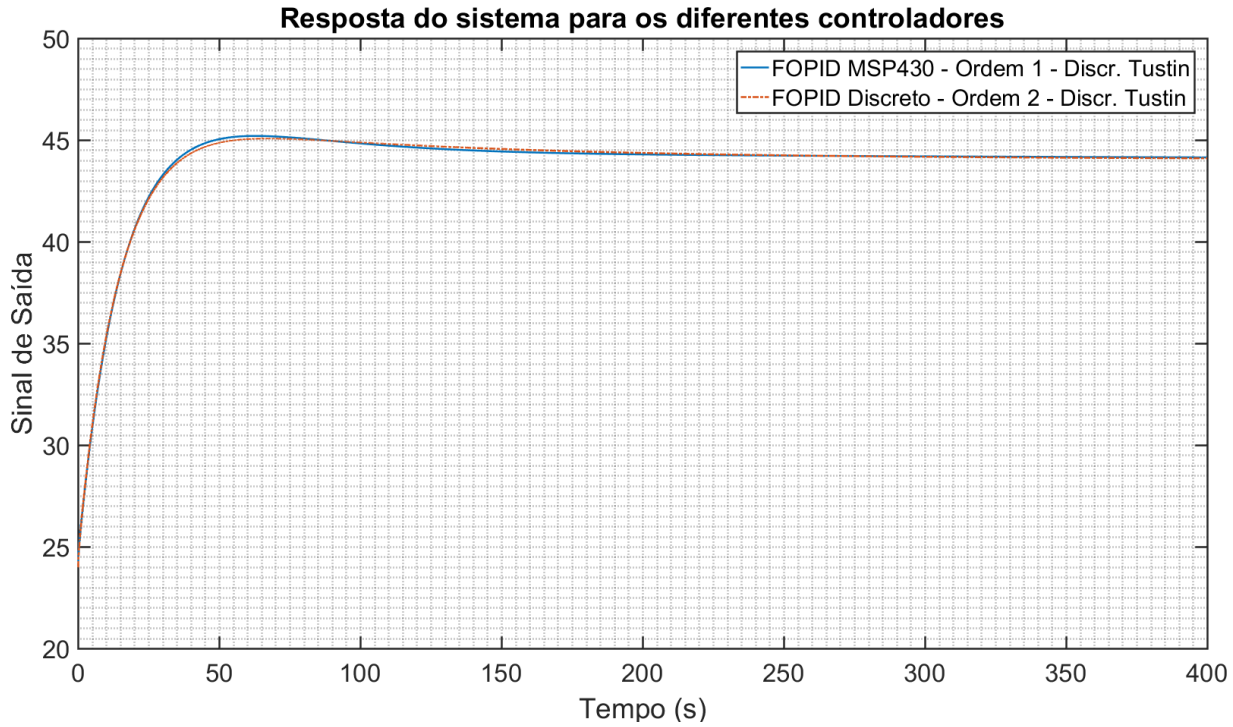


**Figura 14 - Comparação da resposta de um sistema com FOPID discreto de ordem 2 e o FOPID discreto de ordem 1 do MSP430**

A maior diferença entre os dois se dá na subida, onde se obtém uma diferença de até 1% em cima do sinal de entrada, enquanto que na comparação entre o discreto

de ordem 6 e o discreto de ordem 2 no MatLab essa diferença era de apenas 0.2%.

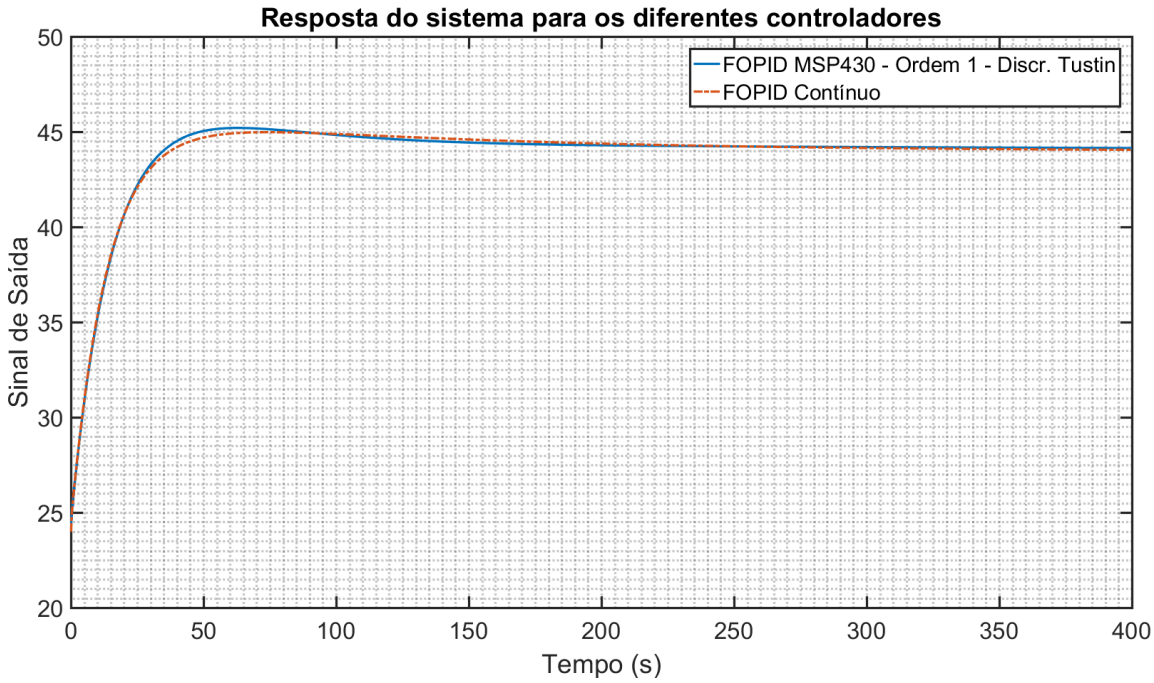


**Figura 15 - Comparação entre o controlador discreto do MSP430 e o controlador contínuo do MatLab**

A diferença entre a saída do sistema com o controlador do MSP430 e o controlador contínuo do MatLab é maior do que todas as diferenças entre as comparações feitas até o momento, como esperado. A maior diferença no sinal de saída é de 1.7% em cima do sinal de entrada.

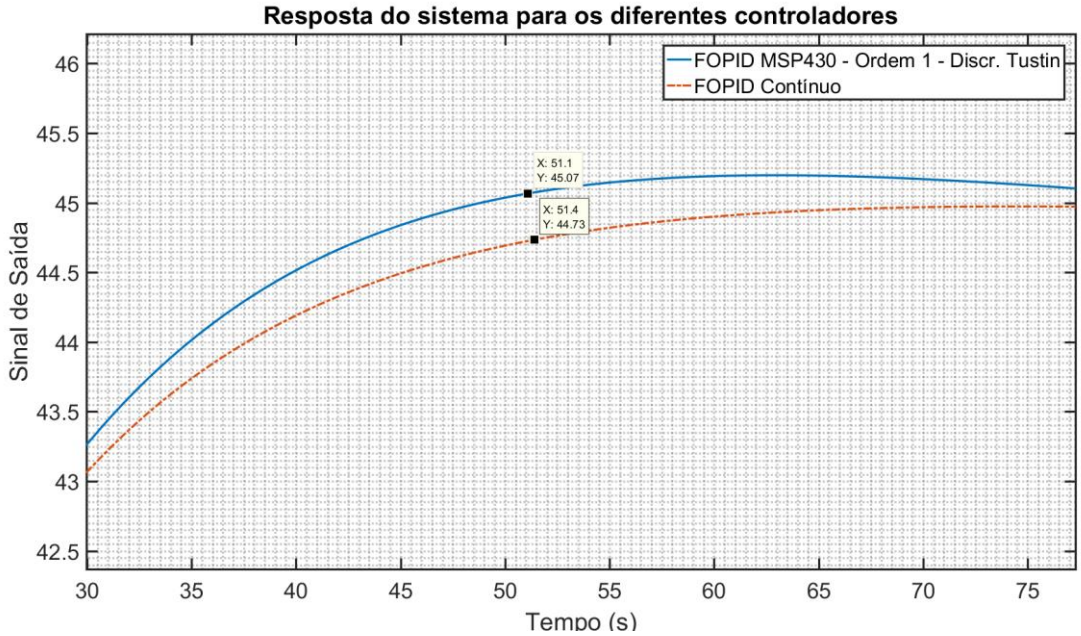


**Figura 16 - Diferença entre o controlador MSP430 e o controlador contínuo do MatLab**

# 6. Conclusão

Neste trabalho foram apresentados conceitos de controle de sistemas dinâmicos, controle digital, microcontroladores, comunicação entre dispositivos aliando todos esses conhecimentos a fim de se atingir um objetivo bem definido.

Os objetivos gerais e específicos traçados foram completamente atingidos, sendo possível realizar a comparação final entre o controlador que o MSP430 é capaz de gerar e um controlador contínuo simulado pelo MatLab.

Para trabalhos futuros, é possível aproveitar o conhecimento aqui deixado para explorar novas formas de discretização para microcontroladores, bem como fazer uma maior análise quanto à desempenho do controlador implementado em outros microcontroladores e às ferramentas utilizadas, além de poder ser colocado em prática em sistemas reais, dado que neste trabalho foram feitas comparações apenas com simulações.

# 7. Referências


- •Shah, P. Agashe, S. **Review of fractional PID controller.** Symbiosis International University, Pune, India. 2016
- •Valério, D. Sá da Costa, J. **Time-domain implementation of fractional order controllers.** IEE Proc.-Control Theory Appl., Vol. 152, No. 5, September 2005
- Tepljakov, A. Belikov, J. Petlenkov, E. **FOMCON: a MATLAB toolbox for fractional-order system identification and control.** 2014
- AL-ALAOUI, M.A.**Simulation and Discretization of Fractional Order Systems.** DepartmentofElectricalandComputerEngineering.AmericanUniversityofBeirut, Líbano. 2009
- DORČÁK, L. PETRÁŠ, I. et al. **ComparisonoftheMethodsfor DiscreteApproximation**ofth**efractional-orderoperator.** Department of Informatics and Process Control, BERG Faculty. 2003
- PODLUBNY, I. **Fractional-Order Systems and Fractional-Order Controllers.** 1994
- MICHARET, A. et al **Fractional-order Systems and Controls: Fundamentals and Applications**, 2006
- XUE, D. **Fractional order PID control of a DC-motor with elastic shaft: a case study.** 2006
- MATIGNON, D. **Stability properties for generalized fractional differential systems.** , 1998
- FAIEGHI, M. **Discrete Fractional Calculus: Applications in Control and Image Processing,** 2011
- Documentação Energide IDE. <Disponível em: <www.tortaparatodos.com.br>. Acesso em: 12 out. 2019>

## APÊNDICE A -

```
byte * b;
int Ts = 1000;
float erro = 0;
float sinalControle = 0;
float input[1] = {0};
int i = 0;

float* fopid(float* b, float* a, float* x);

void setup() {
  Serial.begin(9600);

}

void loop(  ) {

  if(Serial.available())
     erro = Serial.parseFloat();

  float bc[3] = {1.831, -1.491, 0.224};
  float ac[2] = {0.1995, -0.1596};

  input[0] = erro;

  float *resp = fopid(bc,ac,input);

  sinalControle = resp[0];
```

```
   byte *b = (byte *) &sinalControle;
   Serial.write(b,4);

  delay(Ts);
}


float* fopid(float* b, float* a, float* x) {


      float a1[sizeof(a)];
      for (int i = 0; i < sizeof(a); i++) {
        float roundupA = a[i]*1000;
        float roundupB = a[0]*1000;
        a1[i] = roundupA/roundupB;
      }

      float b1[sizeof(b)];
      for (int i = 0; i < sizeof(b); i++) {
        float roundupA = b[i]*1000;
        float roundupB = a[0]*1000;
        b1[i] = roundupA/roundupB;
      }


      int sx = sizeof(x);
      float output [sx];
      output[0] = b1[0]*x[0];
      for (int i = 1; i < sx; i++) {
```

```
        output[i] = 0.0;
        for (int j = 0; j <= i; j++) {
          int k = i-j;
          if (j > 0) {
            if ((k < sizeof(b1)) && (j < sizeof(x))) {
              output[i] += b1[k]*x[j];

            }
            if ((k < sizeof(filter)) && (j < sizeof(a1))) {
              output[i] -= a1[j]*output[k];
            }
          } else {
            if ((k < sizeof(b1)) && (j < sizeof(x))) {
              filter[i] += (b1[k]*x[j]);
            }
          }
        }
      }
      return output;
}
```