Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Pato Branco

# Trabalho de Controle Digital
# Implementação de um filtro digital em microcontrolador

Nome: Marcos Henrique Gomes Vieira Saito
RA: 2112906

# 1. Introdução

## 1.1. Filtros Digitais

São ferramentas essenciais no processamento de sinais, usados para manipular ou alterar as características de um sinal de entrada. Esses filtros podem ser utilizados para atenuar ruídos, extrair sinais de interesse ou modificar as características de frequência do sinal, diferente dos filtros analógicos, que operam diretamente sobre sinais contínuos, os filtros digitais operam sobre sinais discretos, processando amostras em vez de uma forma de onda contínua.

## 1.2. Implementação

A implementação prática de um filtro digital em um microcontrolador pode ser ilustrada pelo uso de um filtro passa-baixa para remover ruído de alta frequência de um sinal de entrada. O código para o filtro é programado no microcontrolador, onde cada amostra do sinal é processada em tempo real para produzir a saída filtrada.

O filtro digital é implementado utilizando uma equação de diferenças, que é uma forma discreta da função de transferência do filtro. Por exemplo, um filtro IIR de segunda ordem pode ser representado como:

$$y[n] = a1 \cdot y[n-1] + a2 \cdot y[n-2] + b0 \cdot x[n] + b1 \cdot x[n-1] + b2 \cdot x[n-2]$$

Onde [n] é a saída do filtro na amostra n, e x[n] é a entrada na amostra n. Os coeficientes a1, a2, b0, b1, e b2 determinam a resposta do filtro.

# 2. Objetivo

Como definido no escopo do projeto ele se trata de filtrar um sinal de entrada que possui uma frequência de f=0,04 Hz e remover o ruído a 0,8 Hz presente no sinal, mantendo as componentes na frequência de interesse. A implementação deve ser feita em um microcontrolador.

## 3. Cálculo da função de transferência

### 3.1. Dados os parâmetros

Para o projeto foram pré definidos os valores:

- Ganho nas baixas frequências (ganho DC) H0= 1
- Frequência de corte fc= 0.04 Hz
- Fator de qualidade Q = 1

E a função de transferência:

$$F(s) = Ho.\frac{woc^2}{s^2+\frac{woc}{Q}.s+woc^2}$$

### 3.2. Função de transferência do filtro passa-baixa

Para a obter a F(s) em tempo contínuo começaremos definindo suas componentes, aplicando os parâmetros definidos no tópico acima.

A frequência angular de corte dada pela equação:

$$Woc = 2 * \pi * fc$$
$$Woc = 2 * \pi * 0.04$$
$$Woc = 0.2513\, rad/s$$

**Sendo assim a função de transferência fica**

$$F(s) = Ho.\frac{0.2513^2}{s^2+\frac{0.2513}{1}.s+0.2513^2}$$
$$F(s) = Ho.\frac{0.0632}{s^2+0.2513.s+0.0632}$$
$$F(s) = \frac{0.0632}{s^2+0.2513.s+0.0632}$$

## 4. Discretização da função de transferência

Neste trabalho, vamos utilizar a Transformação de Tustin (trapezoidal) para discretizar a função de transferência encontrada no passo anterior, visando sua implementação em um microcontrolador.

Determinação do Período de Amostragem

Para calcular o período de amostragem Ts, utilizamos a fórmula:

$$Ts = \frac{1}{100 \cdot 0{,}8} = 0.0125\ s$$

A Transformação de Tustin, também conhecida como aproximação trapezoidal, é usada para converter a função de transferência do domínio s para o domínio z. A fórmula da Transformação de Tustin é:

$$s = \frac{2}{Ts} \cdot \frac{Z-1}{Z+1}$$

Substituindo o valor de Ts na equação, temos:

$$s = \frac{2}{0.0125} \cdot \frac{Z-1}{Z+1}$$

Realizando a simplificação, obtemos:

$$s = 0.0250 \cdot \frac{Z-1}{Z+1}$$

Essa transformação nos permite discretizar a função de transferência contínua, logo reescrevendo a equação da seção 2.3 e aplicando os parâmetros encontrados temos:

$$F(s) = \frac{1}{\frac{1}{Ho \cdot woc^2} \cdot s^2 + \frac{1}{Ho \cdot woc \cdot Q} \cdot s + \frac{1}{Ho}}$$

A função de transferência discreta :

$$F(z) = \frac{1}{\frac{1}{Ho \cdot woc^2} \cdot \left(0.0250 \cdot \frac{Z-1}{Z+1}\right)^2 + \frac{1}{Ho \cdot woc \cdot Q} \cdot \left(0.0250 \cdot \frac{Z-1}{Z+1}\right) + \frac{1}{Ho}}$$

$$F(z) = \frac{1}{\frac{1}{Ho \cdot woc^2} \cdot s^2 + \frac{1}{Ho \cdot woc \cdot Q} \cdot s + \frac{1}{Ho}}$$

A função F(z) pode ser reorganizada para a forma padrão:

$$F(z) = \frac{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-1}}{a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-1}}$$

**Coeficientes da Função de Transferência**

Os coeficientes são determinados a partir da função de transferência contínua e da Transformação de Tustin:

Estes coeficientes $a_0$, $a_1$, $a_2$, $b_0$, $b_1$, $b_2$ são usados na implementação do filtro digital no microcontrolador, permitindo a discretização da função de transferência contínua.

$$a_0 = 1$$

$$a_1 = \frac{-8.Q + 2.woc^2.Ts^2.Q}{4.Q + woc^2.Ts^2.Q + 2.woc.Ts} = -1,99685$$

$$a_2 = \frac{-2 \cdot \omega oc \cdot Ts \cdot 4 \cdot Q + woc^2 \cdot Ts^2 \cdot Q}{4.Q + woc^2.Ts^2.Q + 2.woc.Ts} = 0,99686$$

$$b_0 = b_0 = \frac{woc^2 \cdot Ts^2 \cdot Q \cdot Ho}{4.Q + woc^2.Ts^2.Q + 2.woc.Ts} = 0.0000024635$$

$$b_1 = \frac{2 \cdot woc^2 \cdot Ts^2 \cdot Q \cdot Ho}{4.Q + woc^2.Ts^2.Q + 2.woc.Ts} = 0.0000049271$$

## 5. Código em python

O código em Python foi inicialmente implementado, porém enfrentamos problemas com os coeficientes do filtro ou com a implementação da equação de diferenças, resultando em uma curva inadequada, como mostrado na primeira imagem (Imagem 1). Após algumas correções e melhorias no aspecto temporal, obtivemos a saída desejada conforme mostrado na segunda imagem (Imagem 2). Comparando com os requisitos solicitados, a saída agora atende plenamente às

expectativas.

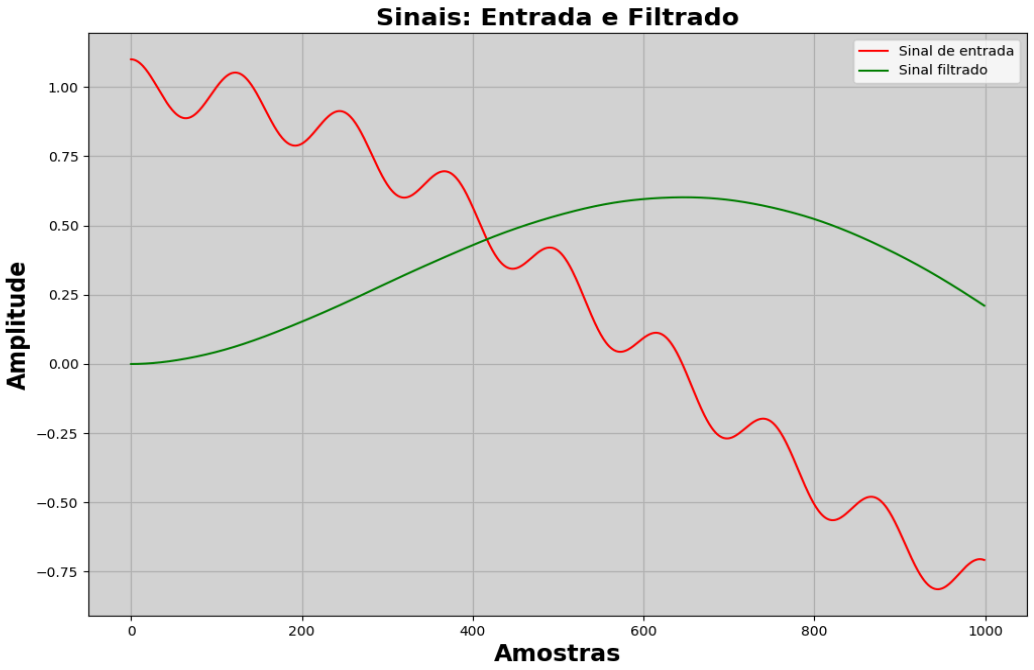


Imagem 1. Saída com erro.

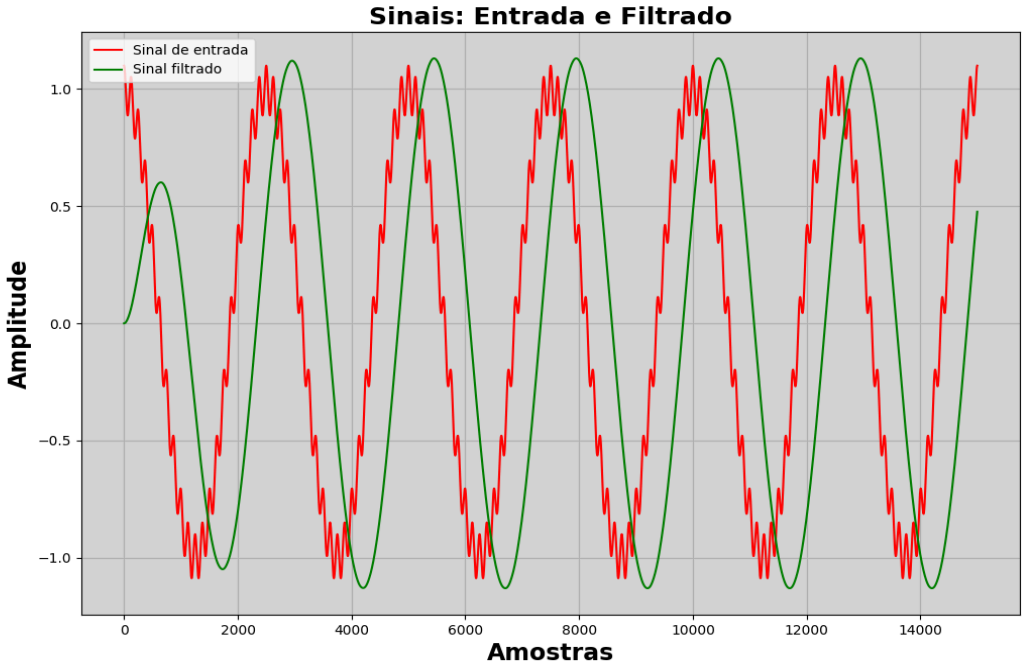


Imagem 2. Saída ajustada

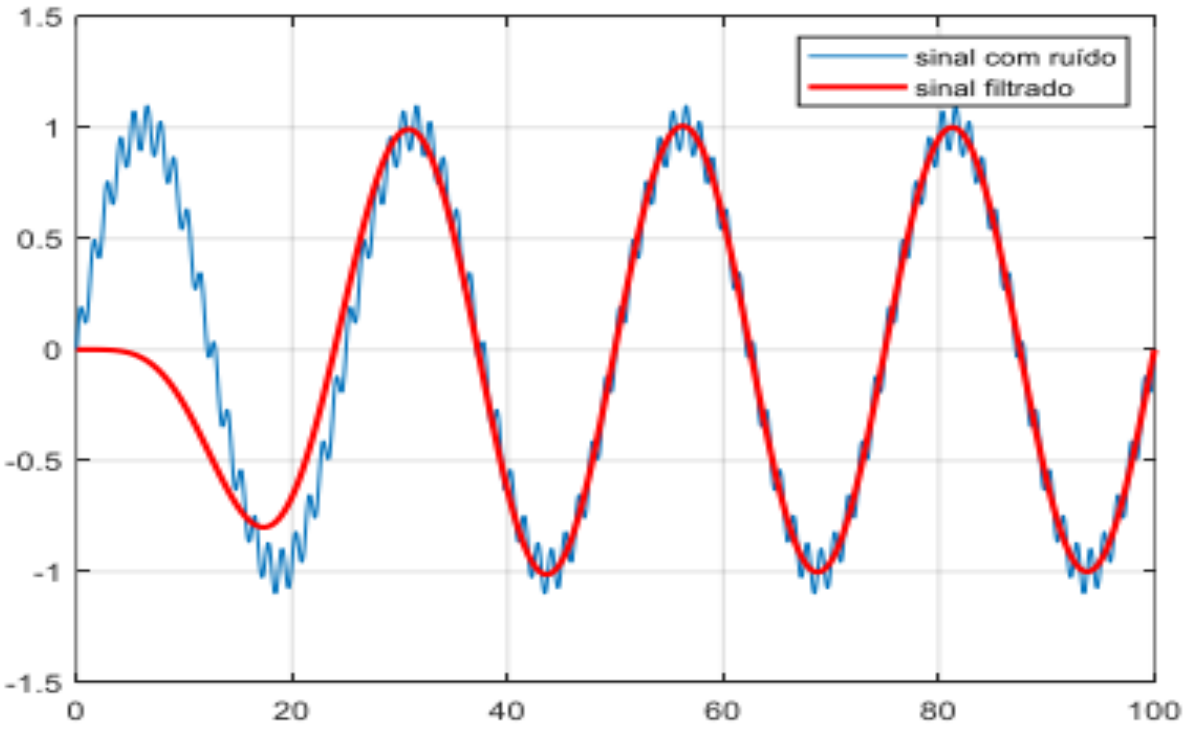


Imagem 3. Saída esperada

Python

```python
import numpy as np
import matplotlib.pyplot as plt

def filtro(x):
   a1 = 1.99685
   a2 = 0.99686
   b = 0.00000246352
   y = np.zeros(len(x))
   y[0] = 0
   y[1] = 0
   for n in range(2, len(x)):
       y[n] = a1 * y[n-1] - a2 * y[n-2] + b * (2 * x[n-1] + x[n-2] + x[n])
   return y

def gerar_sinal(freq_sinal, freq_ruido, duracao):
   num_pontos = int(duracao * 100)  # Número de pontos desejados no vetor t
   t = np.linspace(0, duracao, num_pontos, endpoint=False)
   x = np.cos(2 * np.pi * freq_sinal * t) + 0.1 * np.cos(2 * np.pi * freq_ruido *
t)
   return x

# Gerar sinal de entrada
x = gerar_sinal(0.04, 0.8, 150)

# Aplicar filtro duas vezes
filtrado = filtro(x)
novo = filtro(filtrado)

# Plotar os sinais
plt.figure(figsize=(12,8), facecolor='white')
plt.plot(x, label="Sinal de entrada", color='red')
plt.plot(filtrado, label="Sinal filtrado", color='green')
plt.gca().set_facecolor('lightgrey')  # Cor de fundo do gráfico

plt.xlabel('Amostras', fontsize=17, fontweight='bold')
plt.ylabel('Amplitude',  fontsize=17, fontweight='bold')
plt.title('Sinais: Entrada e Filtrado',  fontsize=18, fontweight='bold')
plt.legend()
plt.grid()
plt.show()
```

## 6. Implementação no STM CUBE IDE e STM MONITOR

C/C++

```
// Incluir as bibliotecas necessárias para o STM32
#include "stm32f1xx_hal.h"
#include <stdio.h>
#include <math.h>
// Definir os parâmetros do filtro
#define A1 1.99685
#define A2 0.99686
#define B 0.00000246352
// Função para aplicar o filtro
void filtro(float* x, float* y, int len) {
   y[0] = 0;
   y[1] = 0;
   for (int n = 2; n < len; n++) {
       y[n] = A1 * y[n-1] - A2 * y[n-2] + B * (2 * x[n-1] + x[n-2] +
x[n]);
   }
}
// Função para gerar o sinal de entrada
void gerar_sinal(float* x, float freq_sinal, float freq_ruido, float
duracao, int num_pontos) {
   float t = 0;
   float dt = duracao / num_pontos;
   for (int i = 0; i < num_pontos; i++) {
       x[i] = cos(2 * M_PI * freq_sinal * t) + 0.1 * cos(2 * M_PI *
freq_ruido * t);
       t += dt;
   }
}
int main(void) {

   float freq_sinal = 0.04;
   float freq_ruido = 0.8;
   float duracao = 1.5;
   int num_pontos = 150;
   float x[num_pontos];
   float filtrado[num_pontos];
   gerar_sinal(x, freq_sinal, freq_ruido, duracao, num_pontos);

   filtro(x, filtrado, num_pontos);
```

```
    for (int i = 0; i < num_pontos; i++) {
        printf("%f %f %f\n", x[i], filtrado[i], filtrado[i]); // Exemplo de
formatação para CubeMonitor // @suppress("Float formatting support")
    }
    // Loop principal do programa
    while (1) {
        // Pode adicionar outras operações aqui se necessário
    }
}
```