Estágio Verão

Autor:

Luís Miguel Pereira Silva

Responsáveis:

Sara Cerqueira e Gilberto Martins

Julho 2023

Plano

Introdução

1 Objetivos do projeto

O objetivo geral do projeto consistiu em criar uma solução funcional e otimizada para aquisição, filtragem e transmissão de dados de um sensor, garantindo a integridade e confiabilidade das informações durante o processo de comunicação entre o Arduino e a STM e entre a STM e a porta série - USB.

- **Comunicação** Implementar um protocolo de comunicação eficiente e seguro, permitindo a transferência dos dados do sensor de forma confiável.
- **Filtragem em software dos dados no Arduino:** Desenvolver um algoritmo de filtragem de dados no próprio Arduino, possibilitando o processamento das informações antes da transmissão.
- **Otimização da transmissão de dados:** Realizar otimizações no processo de transmissão de dados entre os dispositivos, visando reduzir o tempo de resposta e minimizar perdas ou erros durante a comunicação.

2 Contextualização

2.1 Informações pessoais

Meu nome é Luís Miguel Pereira Silva, sou estudante de Engenharia Física na Universidade do Minho. Durante os meses de junho e julho de 2023, tive a oportunidade de realizar um estágio de Verão na empresa BirdLab - Biomedical Robotic Devices.

2.2 Motivação para o projeto

A minha principal motivação residia na oportunidade de aplicar na prática os conhecimentos adquiridos ao longo do meu percurso académico, este estágio permitiu me especialmente nas áreas de sistemas e computação e processamento de sinal. Além disso, a possibilidade de aprender a trabalhar com dispositivos Arduino e STM despertou em mim uma grande curiosidade, abrindo caminho para o fascinante mundo da automação.

Descrição das atividades desenvolvidas

3 Introdução ao Arduino

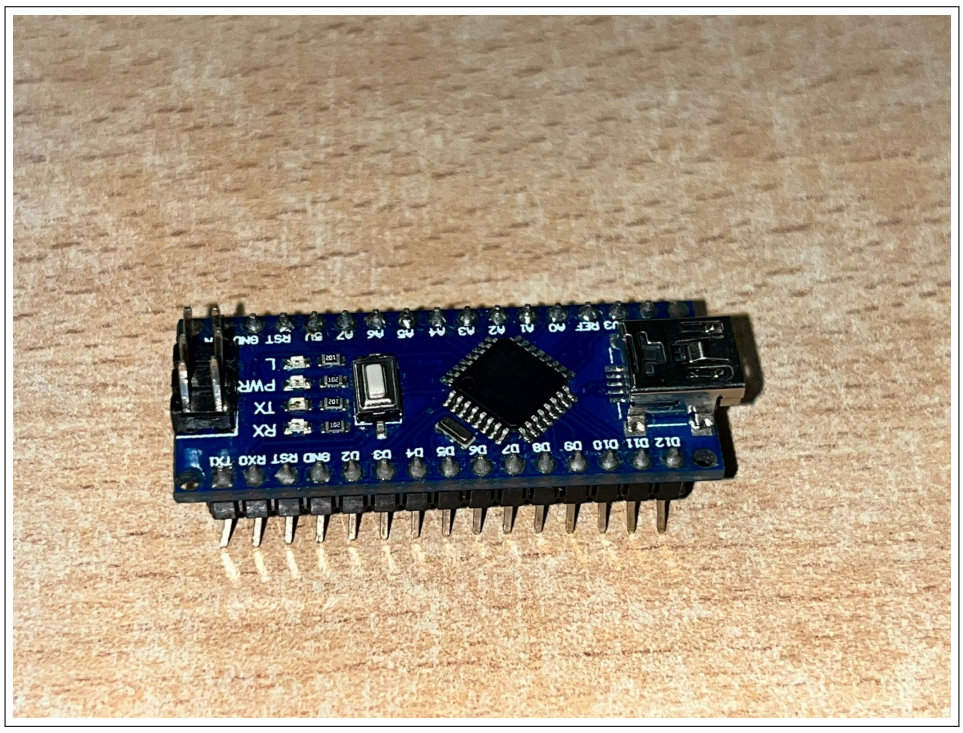
3.1 Configurar o ambiente de desenvolvimento

O estágio na BirdLab teve início com a configuração do ambiente de desenvolvimento, que envolveu a instalação de softwares e bibliotecas essenciais para a programação em Arduino. Os seguintes elementos foram instalados:

- **Arduino IDE:** A Arduino IDE é uma aplicação com uma interface de programação amigável e recursos essenciais para compilar e enviar o código para o microcontrolador do Arduino.
- **Coolterm:** Coolterm é uma aplicação que permite enviar e receber dados serialmente, sendo útil para depuração e monitoramento das comunicações. Esse programa foi escolhido por possibilitar a análise dos dados recebidos em formato hexadecimal, auxiliando na depuração e monitoramento do sistema. Além disso, é que o Coolterm não apresenta limitações quanto à escolha da taxa de baudrate, proporcionando flexibilidade na configuração da comunicação entre placas.
- **TimerOne:** A biblioteca TimerOne foi incorporada ao ambiente Arduino e forneceu funcionalidades para configurar e manipular timers internos do microcontrolador do Arduino. Isso possibilitou a execução de tarefas periódicas com base em interrupções temporizadas.
- **stdio:** A biblioteca stdio foi utilizada para operações de entrada e saída padrão, inclui funções como printf() e scanf().
- **math:** A biblioteca math contém funções matemáticas para realizar operações mais complexas, inclui funções como sqrt(), sin(), cos(), log(), entre outras.
- **stdint:** A biblioteca stdint define tipos inteiros com tamanhos de bits específicos, independentemente da arquitetura do microcontrolador, fornecendo tipos como uint8..t, uint16..t, entre outros.
- **string:** A biblioteca string oferece funções para manipulação de strings, como copiar, concatenar, comparar e dividir strings.

3.2 Inicialização

Após a configuração do ambiente de desenvolvimento, recebi um Arduino Nano que foi fundamental para o meu aprendizado e o desenvolvimento do projeto. Com entusiasmo, explorei o mundo da programação embarcada através da plataforma Arduino IDE, compreendi os conceitos essenciais da linguagem de programação e a interação da placa com o código escrito e realizei experiências práticas, como leitura de portas analógicas e controlo de portas digitais, ampliando minha compreensão sobre os pinos. Esta jornada inicial foi crucial para o sucesso das etapas seguintes do estágio.



4 Introdução à STM

4.1 Configurar o ambiente de desenvolvimento

Assim como ocorreu com o Arduino, para a STM iniciei com a configuração do ambiente de desenvolvimento dos softwares Keil e STM32CubeMX que foram fundamentais para a programação da placa STM32F407G-DISC1.

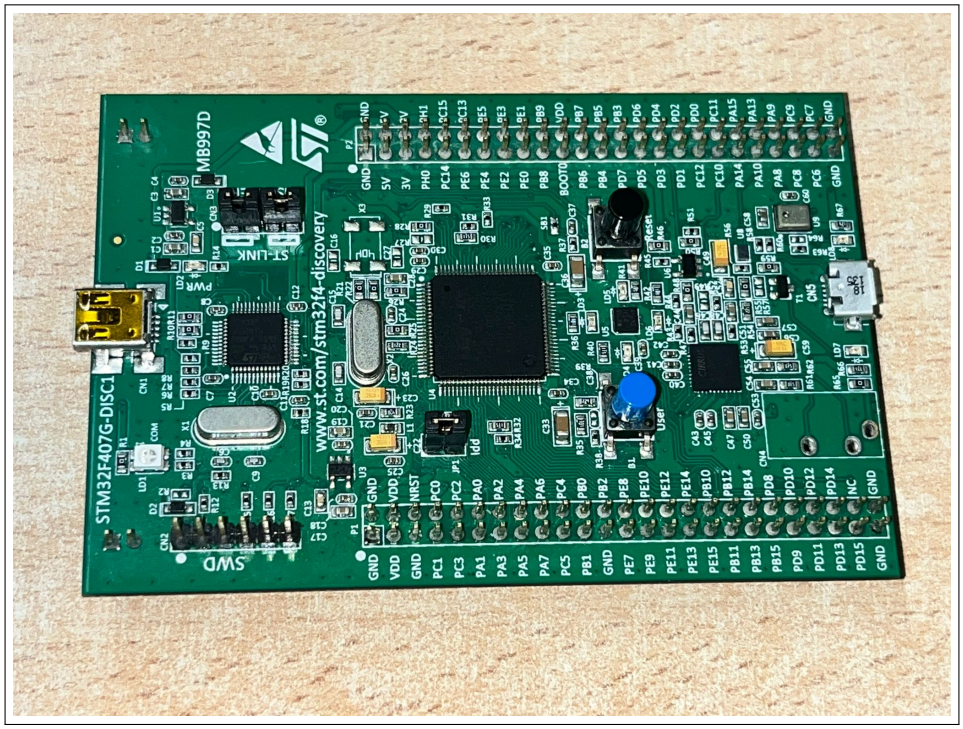
- **Keil:** O Keil é uma aplicação de desenvolvimento integrado (IDE) que oferece uma interface intuitiva e recursos essenciais para a programação de microcontroladores STM32. Com o Keil, foi possível escrever, compilar e depurar

o código para a placa STM32F407G-DISC1. Nesta aplicação, foi necessário alterar umas definições (Project > Options for Target) em Target, alterar a versão do compilador ARM para a 6ª Versão e em C/C++ alterar a otimização para -00.

- **STM32CubeMX:** O STM32CubeMX é uma aplicação que auxilia na configuração e geração de código inicial para as placas STM32, possibilitando a configuração dos periféricos, bem como a geração de código base para o projeto.
- **Bibliotecas:** As bibliotecas necessárias para programar a STM32 foram as mesmas utilizadas anteriormente para o Arduino.

4.2 Inicialização

Posteriormente, recebi a placa STM32F407G-DISC1 e a partir do Keil e do STM32CubeMX explorei a arquitetura da STM32, aprofundando a compreensão sobre os periféricos.



5 Introdução à comunicação Arduino - STM

5.1 Envio de dados do Arduino

Para enviar dados do Arduino para a placa STM, inicialmente usou-se o serial.print para enviar elementos codificados em ASCII pela porta série, com as seguintes instruções:

- **1ºCiclo:** Percorre todos os elementos buffer e após passar o 2º ciclo faz Serial.print de um elemento para separar elementos (estilo um ';').
- **2ºCiclo:** Percorre todos os carácteres ASCII e faz o Serial.print de cada carácter.

5.2 Receção de dados da STM

Para receber dados do Arduino na placa STM, foi necessário usar o HAL..USART..Recive para receber um conjunto de bytes num intervalo de tempo pela porta série, com os seguintes argumentos:

- **HAL..UART..Receive(E, (uint8..t*)B , N, T)** Este comando lê os bytes os N (numero) bytes do endereço E (endereço da porta recetora) e coloca no apontador do B (para o buffer), os bytes durante o T (tempo de receção). Este numero de bytes, N, terá de ser calculado tendo em conta o numero de elementos do buffer enviado, Z, e o numero de carácteres ASCII enviados por elemento ,Y.

$$N = Y * Z$$

5.3 Baudrate

A baudrate corresponde ao número de vezes que um sinal em um canal de comunicação muda o seu estado, ou seja, significa que pode variar de 0 para 1 ou de 1 para 0 até X vezes por segundo. Considerando o caso extremo, todos os bits sequenciais diferentes:

$$Bitrate = Baudrate$$

Para os restantes casos,

$$Bitrate = Baudrate * N$$

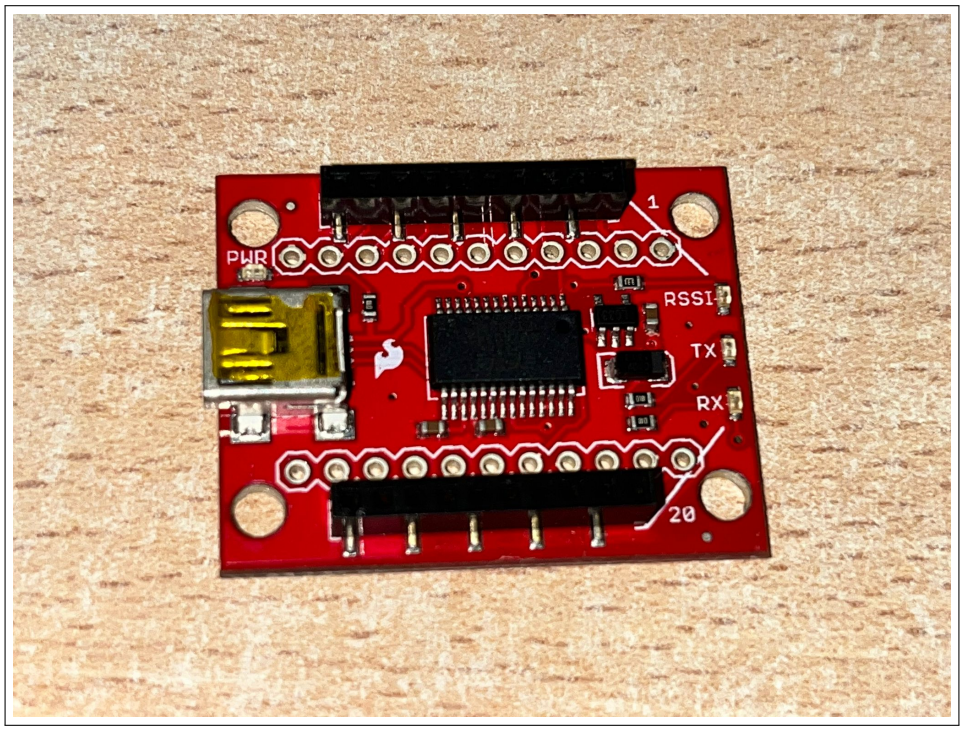
onde N corresponde ao numero de bits por baud. Assim, verifica-se que a baudrate afetará a velocidade de envio de dados.

5.4 Primeira comunicação

Na primeira comunicação como dito anteriormente, foram enviados carácteres ASCII, ou seja em cada elemento do buffer for enviados um digito inteiro (1 byte) + um ponto (1 byte) + quatro casas decimais (4 bytes) + um ponto e virgula (1 byte). Na STM, cada elemento ASCII é colocado numa célula de um buffer com 8 vezes o tamanho do buffer do Arduino.

6 Transmissão da informação na STM

Os dados recebidos na STM para ser transmitidos, foi me disponibilizando um conversor série - USB.



6.1 Envio de dados da STM

Para enviar dados na placa STM para o conversor série - USB, foi necessário usar o HAL..UART..Transmit para enviar um conjunto de bytes num intervalo de tempo pela porta série, com os seguintes argumentos:

- **HAL..UART..Transmit(E, (uint8..t*)B , N, T)** Este comando envia os N (numero) bytes, começando no apontador do B (buffer) para o endereço E (endereço da porta emissora) durante o T (tempo de receção).

A visualização dos dados transmitidos pode ser feita usando o Coolterm, conectado-o à entrada USB, que está ligada a porta série-USB.

7 Comunicação Arduino - STM

7.1 Envio de dados do Arduino

Na primeira comunicação foram enviados 8 bytes e sabendo que podemos enviar um float em binário que no IA-32 representa-se por FP32 do IEE 754, 4 bytes, assim verifica-se a ineficiência desta codificação de informação... Numa segunda comunicação é necessário um comando para envio de bytes do Arduino para a placa STM, deste modo, pode-se usar o serial.write para enviar bytes pela porta série, com os seguintes argumentos:

- **Serial.write((byte*) B, N)** Este comando acede à posição do apontador do B (buffer) e envia os N (numero) bytes pertencente ao B (buffer).
- **Serial.write((uint8..t)BYTE)** Este comando envia apenas um BYTE.

7.2 Receção de dados da STM

Para receber dados do Arduino na placa STM, foi necessário usar como anteriormente o USART..Recive, recebendo um conjunto de bytes num intervalo de tempo pela porta série, com os seguintes argumentos:

- **HAL..UART..Receive(E, (uint8..t*)B , N, T)** Este comando lê os bytes os N (numero) bytes do endereço E (endereço da porta recetora) e coloca no apontador do B (buffer), os bytes durante o T (tempo de receção).

7.3 Comunicação

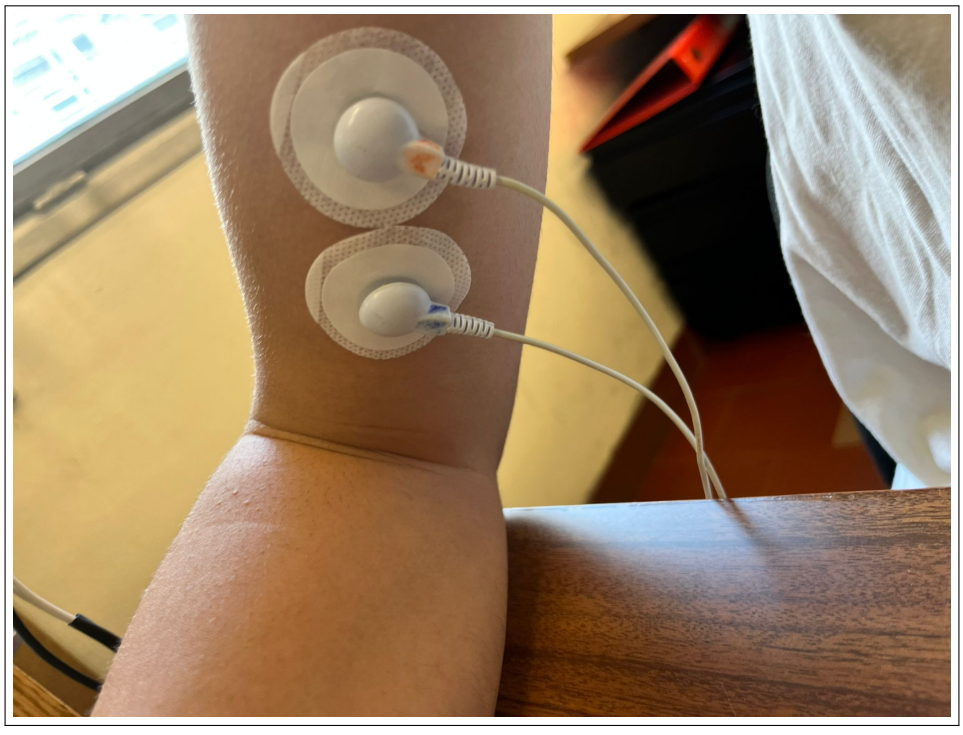
Na segunda comunicação, poderíamos converter o float numa notação IEEE 754 FP32 com 4 bytes, mas podemos usar também a notação IEEE 754 FP16 com 2 bytes diminuindo a precisão e a gama, mas aumentando a velocidade de envio. Na STM, cada FP16 é colocado num elemento de um buffer e posteriormente é convertido para float e colocado no respetivo elemento de outro buffer. Assim não só são enviados menos bytes para os elementos do buffer do Arduino como o buffer que recebe os dados na STM será 8 vezes menor.

8 Sensor

Até ao momento, apenas a comunicação foi testada com dados controlados no buffer (por exemplo um buffer todo a zero e alguns elementos diferentes de zero). Assim, para um melhor entendimento do sistema global, foi me disponibilizado um sensor que mede uma eletromiografia. Os fios que saem do sensor, têm a seguinte configuração,

Configuração sensor		
Cor	Saída	Potencial
Vermelho	+V	3,3 V
Preto	GND	0 V
Verde	Vcc	5 V
Branco	-V	-3,3 V

após ligar a alimentação do sensor como indicado acima e conectar as referências do Arduino e do sensor (GND com GND), liga-se o sinal de saída a uma entrada analógica do Arduino. Para finalizar conecta se o cabo com os elétrodos já colados à pele da seguinte forma:



9 Sistema sem filtro

Agora com dados recolhidos pelo sensor no pino analógico é necessário efetuar-se a leitura. Por isso, usa-se o seguinte comando:

- **(analogRead(AX)*5)/1023** Onde o analogRead() lê o valor da porta analógica X que corresponde a um valor que varia entre 0 e 1023 e como o valor do potencial da porta analógica varia entre 5 e 0 Volts, precisamos o valor lido por 5/1024 (1024, porque são 1024 números representáveis).

Nota: Pelo facto do potencial na porta analógica estar restrito à gama entre 0 e 5 Volts é colocado um offset nos dados enviados pelo sensor. Os dados sem offset variam entre 1 e -1. Assim, no loop são lidos conjuntos de valores (retirando o offset) e colocados ordenadamente no buffer, no mesmo momento para a comunicação com a STM os floats tem de ser convertidos num valores binários de 2 bytes e colocados noutro buffer igualmente ordenados. Com a representação FP16 do IEE 754 possui uma resolução variável dependendo do quanto afastado do zero, achei preferível usar uma representação, onde a gama de valores é -1 a 1, tendo assim a melhor resolução constante em toda a gama com 2 bytes.

- **Um bit sinal**
- **Quinze bits para a mantissa** Com 15 bits temos $2^{15} - 1$ valores representáveis, assim como a gama de valores sem sinal está entre 0 e 1, multiplicando o float pelo numero de valores representáveis resulta um inteiro que passado para binário puro tem apenas 2 bytes (exemplo, $1 * (2^{15} - 1) = 32767 = 0x7FFF$).

Quando há uma interrupção, são enviado os todos os elementos do buffer com a representação binária. Na STM, os dados são recebidos convertidos da representação acima para float e posteriormente transmitidos para a porta serie - USB. O respetivo fluxograma em anexo (18).

10 Filtragem

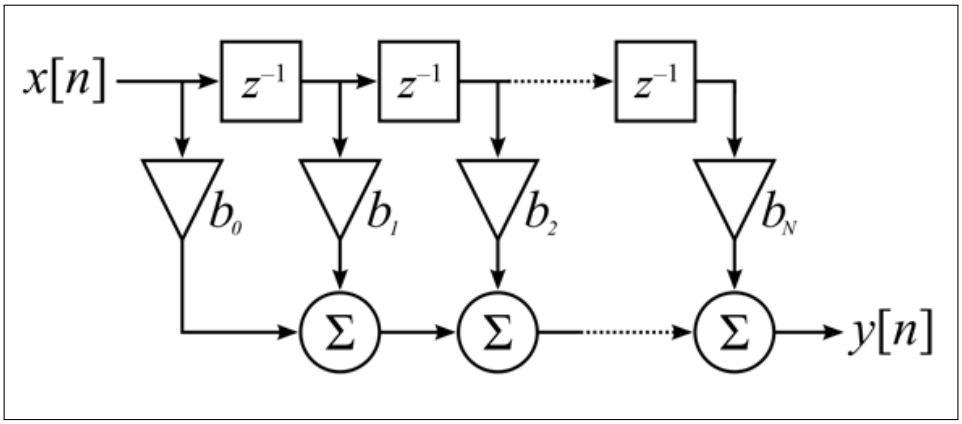
10.1 Biblioteca

Inicialmente, procurei uma biblioteca para aplicar o filtro aos dados brutos, mas vendo a documentação da biblioteca EmotiBit vi que tinha algumas limitações e não teria o entendimento total de como seria aplicação do filtro.

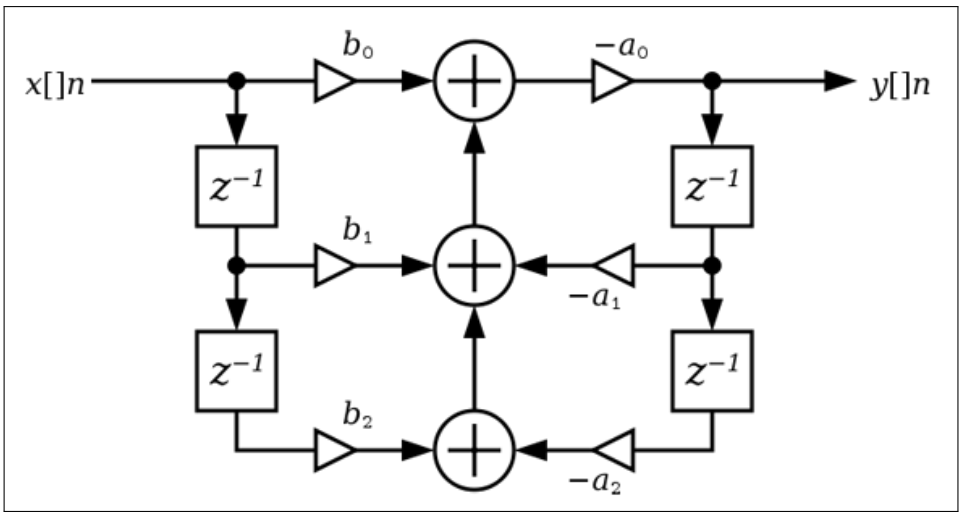
10.2 Entendimento da aplicação de filtros

Para puder aplicar o filtro, com base na teoria, temos dois tipos de filtro:

- **FIR - Filtro de Resposta ao Impulso Finito** $y[n] = b[0] * x[n] + b[1] * x[n-1] + b[2] * x[n-2] + ... + b[N] * x[n-N]$



- **IIR - Filtro de Resposta ao Impulso Infinito** $y[n] = (b[0] * x[n] + b[1] * x[n-1] + b[2] * x[n-2] + ... + b[N] * x[n-N] - a[1] * y[n-1] + a[2] * y[n-2] + ...a[N] * y[n-N])/a[0]$



Assim para pode-se aplicar o filtro fica fácil, apenas é preciso de calcular os coeficientes do a e b.

10.3 Projeção do Filtro

Com a ajuda do MATLAB na aplicação Filter Design, consegui projetar diversos tipos de filtros FIR e IIR. Diante da necessidade de filtrar sinais de eletromiografia, foi escolhido um filtro passa-banda Butterworth de 12ª ordem, que essencialmente é a combinação de um filtro passa-baixo de 6ª ordem e um filtro passa-alto de 6ª ordem. A eletromiografia apresenta características importantes na faixa de frequências de 0 a 500 Hz. No entanto, devido ao movimento do corpo e possíveis interferências, é necessário cortar algumas frequências. Para alcançar esse objetivo, optou-se por aplicar um filtro passa-banda com largura de banda entre 20 Hz e 450 Hz (com frequência de amostragem de 1200 Hz, que tem de ser superior a duas vezes a frequência de corte, teorema de Nyquist) a fim de restringir o sinal apenas à faixa de interesse. Estes valores foram considerados tendo em conta os filtros da delsys, que têm uma largura de banda entre 20 ± 5 Hz e 450 ± 50 Hz.

10.4 Obtenção dos coeficientes

Após projetar o filtro passa-banda Butterworth de 12ª ordem utilizando a aplicação Filter Design do MATLAB, é possível exportar as variáveis necessárias para implementar o filtro. Dentro da aplicação Filter Design, vá em "File > Export...", e pode-se extrair as variáveis "SOS Matrix" (SOS) e "Scale Values" (G), para obter os coeficientes do filtro basta utilizar seguinte o comando no MATLAB:

- **[b, a] = sos2tf(SOS, G)**

Este comando calcula os coeficientes do filtro e vai armazená-los nas variáveis "b" e "a". É importante notar que, os coeficientes tem uma quantidade reduzida de dígitos decimais. Para aumentar a precisão dos coeficientes, é possível realizar o cálculo novamente no Jupyter. No Jupyter, usando a biblioteca *signal* do *scipy*. Podemos assim usar o seguinte comando para obter os coeficientes do filtro passa-banda:

- **b, a = signal.iirfilter(n, [fci, fcs], fs=fs, btype='bandpass')**

Onde , "n" representa metade da ordem do filtro, "fci" e "fcs" representam as frequências de corte inferior e superior, respetivamente, e "fs" é a frequência de amostragem. O argumento "btype='bandpass'" especifica que é projetado um filtro passa-banda.

10.5 Aplicação filtro

Observando o código anteriormente feito, temos duas frequências de amostragem definidas, uma referente à leitura do valor analógico e outra definida pelo período

de interrupção. Assim não é possível aplicar o filtro aos dados recolhidos, por isso é necessário recorrer à alteração do código, a solução proposta consiste em criar dois buffers, um para armazenar os dados brutos e outro para armazenar os dados filtrados. A cada interrupção, um elemento é alterado nos buffers e o índice é incrementado. Quando o buffer atinge o seu limite, uma flag é ativada para indicar que os dados estão prontos para serem enviados. No loop principal, é verificado se a flag está ativada e, caso afirmativo, os dados são enviados. Com esta abordagem, é possível garantir uma frequência de amostragem consistente, permitindo a aplicação adequada do filtro nos dados capturados. O respetivo fluxograma em anexo (19).

11 Testes e validação do sistema

Ao logo de todo o estágio foram sendo feitos testes aos sistemas criados, de modo a chegar a esta verão final. Os testes dividiram se na parte da comunicação e na filtragem. Na comunicação, os testes a seguir indicados foram validos durante toda esta parte.

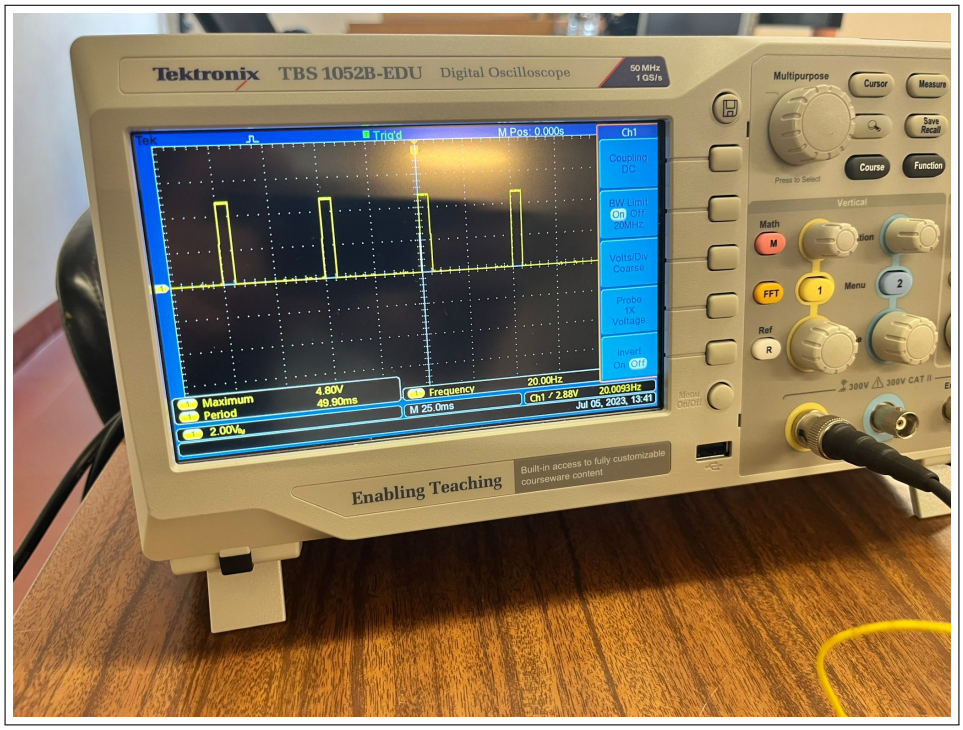
11.1 Sistema de comunicação

Obviamente, neste sistema, não podemos usar os dados do sensor porque não temos controlo e, assim, não podemos confiar nos dados transmitidos. Para testar pode-se enviar um buffer de zeros e com valores diferentes de zero em algumas posições, testando a fiabilidade das posições e dados transmitidos. Também, podemos enviar um buffer inicializado com múltiplos de 0.0001, dependendo da posição, onde é incrementado 0.0001 a cada interrupção, garantindo assim a validade dos dados transmitidos e alterados a cada interrupção. Com estes algoritmos, foi possível verificar que o comando HAL..USART..RECEIVE usa um byte de inicialização (é possível que isso aconteça devido à baudrate ou tempo de interrupção) e por isso no código fornecido tem um byte a mais sendo enviado em cada envio. Na STM, aumenta assim um elemento dispensável no buffer que recebe os dados, mas garantindo a validade. Desta forma, também para determinar a baudrate limite, fixa-se o tempo de interrupção num valor alto de modo a não afetar os dados. Posteriormente, vamos aumentando a baudrate até haver aleatoriedade nos dados (usa-se o ultimo valor sem aleatoriedade), isto é feito tanto para o Arduino - STM como para a STM porta série - USB (no caso da baudrate, STM não tem tempo de interrupção). No caso Arduino - STM, com a baudrate limite definida (249600), aumenta-se o tempo de interrupção até obter uma frequência de amostragem viável para a aplicação do filtro, tendo cuidado com o tempo necessário para o sistema responder ás instruções. Este ultimo pode ser visto com o Digital Lab em portas

digitais com os seguintes comandos:

- **Arduino** digitalWrite(D,!digitalRead(D)), onde D corresponde ao pino digital.
- **STM** HAL..GPIO..TogglePin(GPIOX, GPIO..PIN..N) , onde o pino correspondente é o XN.

Assim colocando dois destes comandos com uma instrução no meio, podemos ver a frequência de emissões e receções, ligando o osciloscópio ao pino indicado.

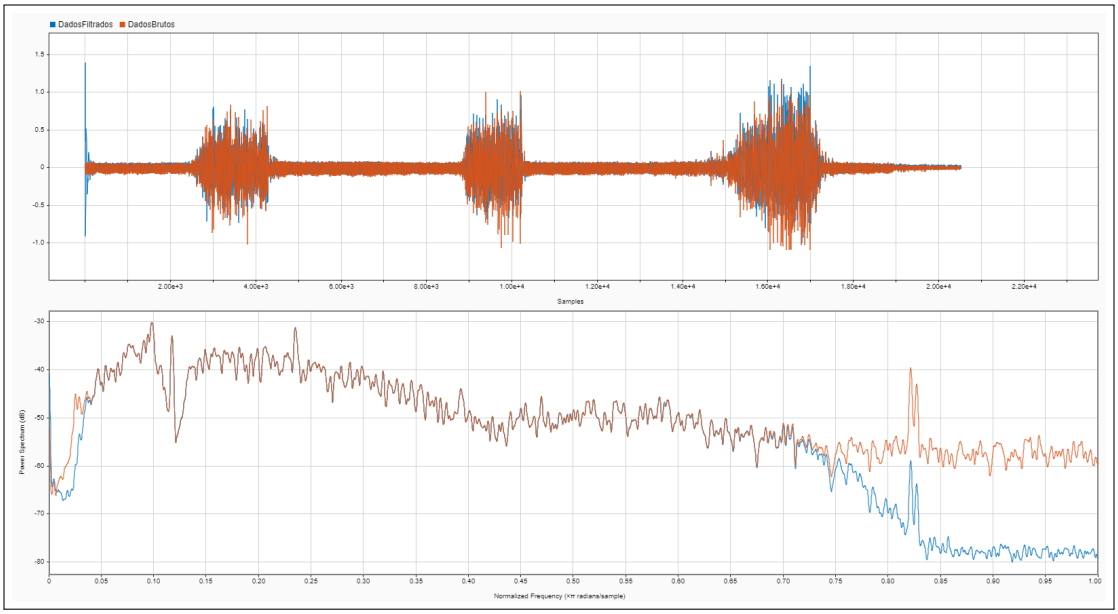


Com a visualização do osciloscópio num exemplo da imagem acima, podemos tirar varias conclusões, como o período de uma ou um conjunto de instruções e também o tempo que demora a instrução ou instruções a serem efetuadas. Este procedimento é bastante importante, por principalmente definir as frequências de emissão ou receção.

11.2 Sistema de filtragem

Para validar o filtro aplicado, procedemos à análise espectral de um conjunto de dados brutos e filtrados. Para isso, utiliza-se um "serial print" para imprimir os valores analógicos e filtrados a cada interrupção e, com a ajuda do CoolTerm,

guarda-se os dados num bloco de notas. Posteriormente, no MATLAB, utiliza-se a aplicação Signal Analyser para observar o espectro de potência dos dados recolhidos. Desta forma, podemos verificar se a filtragem está a ser feita de forma adequada.



Outra abordagem para garantir a fidelidade do sistema de filtragem é através da inserção de um sinal de frequência controlada pelo Digital Lab e, em seguida, observar a resposta do filtro no SerialPlot. Com estas análises, consegue-se assegurar que o filtro está a atuar corretamente e que os dados estão a ser processados de forma fiável e precisa.

Aprendizagens e desafios

12 Aprendizagens técnicas

Durante o estágio na BirdLab - Biomedical Robotic Devices, adquiri diversos conhecimentos técnicos relevantes. Alguns dos principais incluem:

12.1 Protocolos de comunicação:

Aprendi a implementar protocolos eficientes de comunicação entre dispositivos, e entendi a importância de garantir a integridade dos dados durante a transferência.

12.2 Filtragem de dados:

Desenvolvi habilidades na aplicação de filtros em software, permitindo o processamento das informações.

12.3 Otimização de transmissão:

Aprendi a otimizar o processo de transmissão de dados, procurando reduzir o tempo de resposta e minimizar possíveis perdas ou erros durante a comunicação.

12.4 Análise espectral:

Adquiri conhecimentos sobre a análise espectral de sinais, utilizando o MATLAB para verificar a eficácia do filtro aplicado e garantir a fidelidade dos dados processados.

13 Aprendizados profissionais

Ao longo do estágio, pude vivenciar o ambiente de trabalho numa empresa de tecnologia e engenharia. Alguns dos principais aprendizados profissionais incluem:

13.1 Resolução de problemas:

Desenvolvi habilidades na identificação e resolução de desafios que surgiram ao longo do projeto.

14 Desafios superados

Durante o estágio, enfrentei alguns desafios que foram superados com dedicação e esforço. Alguns dos principais desafios foram:

14.1 Inicialização aos microcontroladores:

Um dos principais desafios, no início do estágio foi a familiarização com microcontroladores, que era algo novo para mim. Contudo, aprendi a configurar e programar um microcontrolador, e familiarizei-me com os periféricos disponíveis.

14.2 Implementação do protocolo de comunicação:

Criar um protocolo de comunicação eficiente exigiu um estudo aprofundado das especificações técnicas e a resolução de problemas relacionados à emissão e receção de dados.

14.3 Otimização da transmissão:

Encontrar a configuração ideal para a transmissão de dados foi um desafio, pois era necessário equilibrar a tanto a frequência de amostragem como a baudrate com a confiabilidade dos dados.

14.4 Filtragem e processamento de dados:

Principalmente pelas limitações das bibliotecas, desenvolver algoritmos de filtragem adequados e ajustar os parâmetros para garantir a precisão e a eficiência do processamento dos dados coletados.

Resultados e conclusões

Após o processo de desenvolvimento e implementação, os resultados alcançados neste estágio foram satisfatórios e atenderam aos objetivos propostos. Foram criadas soluções funcionais para a aquisição, filtragem e transmissão de dados.

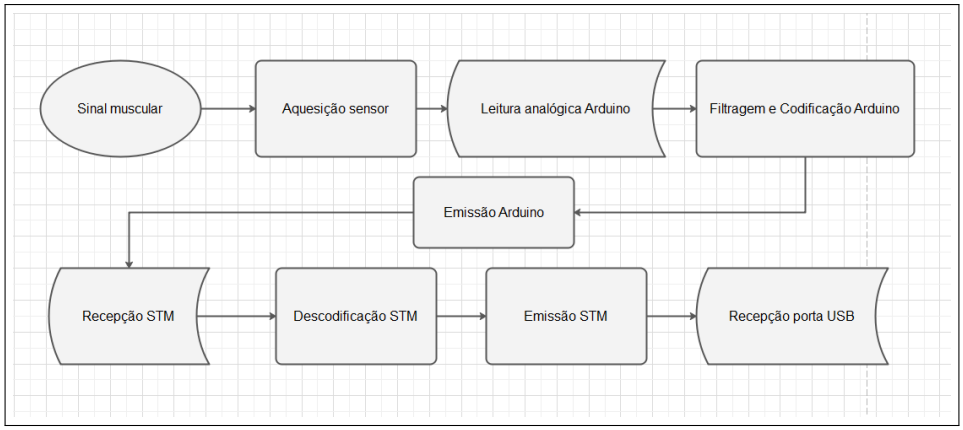
15 Comunicação

A comunicação entre os dispositivos é eficiente e confiável. A baudrate do Arduino para a comunicação com a STM foi configurada em 249600, enquanto a baudrate para a comunicação com a STM e a porta série - USB foi ajustada para 43600. Estas configurações foram escolhidas de forma a garantir a velocidade adequada de transmissão, sendo que as baudrates não puderam ser definidas com valores superiores.

16 Filtragem

A frequência de aquisição do Arduino foi definida em 1200 Hz, o que possibilitou uma amostragem adequada dos dados do sensor. A implementação dos filtros em software no Arduino foi realizada de forma ajustável, permitindo a alteração dos coeficientes "a" e "b" do filtro conforme necessário. A análise espectral dos dados validou a eficácia do filtro aplicado, mostrando que os dados são processados de forma adequada e que o filtro está a atuar de acordo com o esperado.

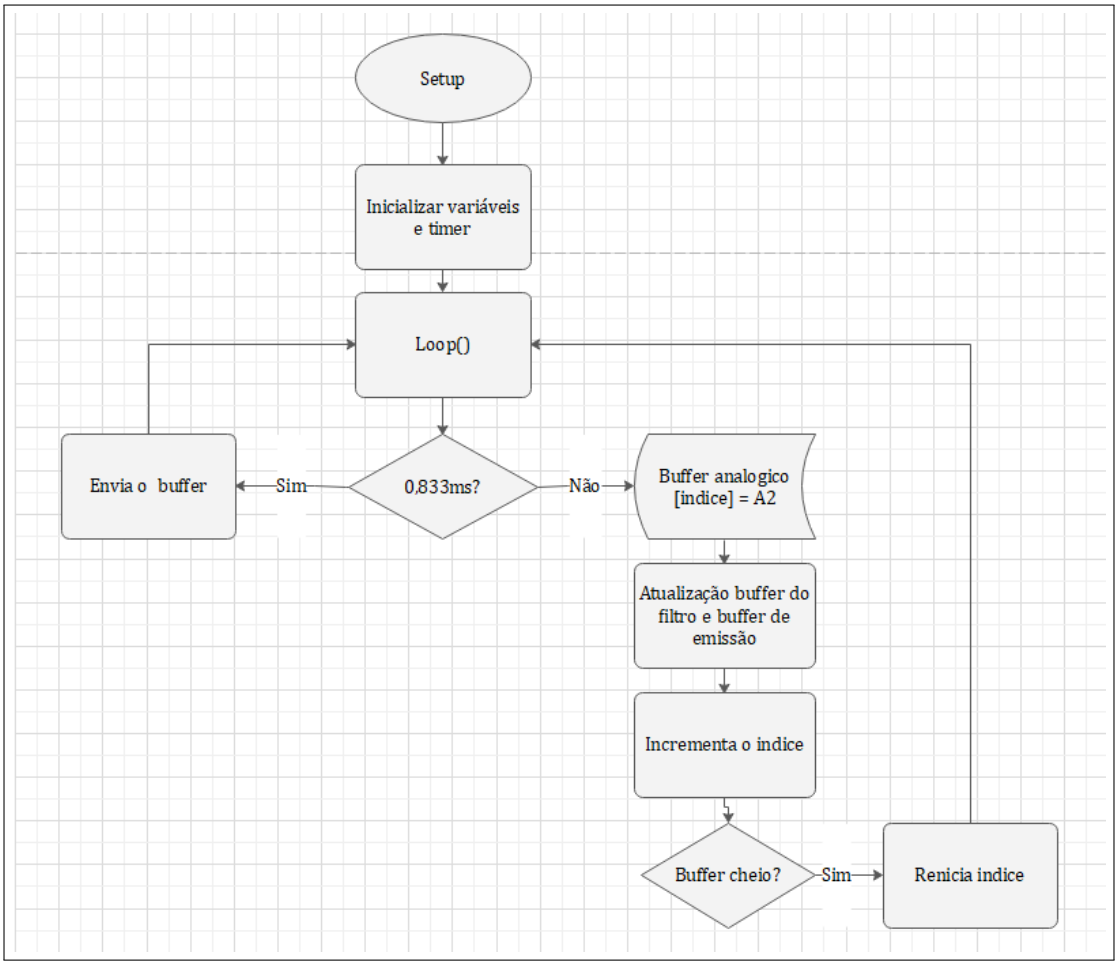
17 Sistema



Anexos

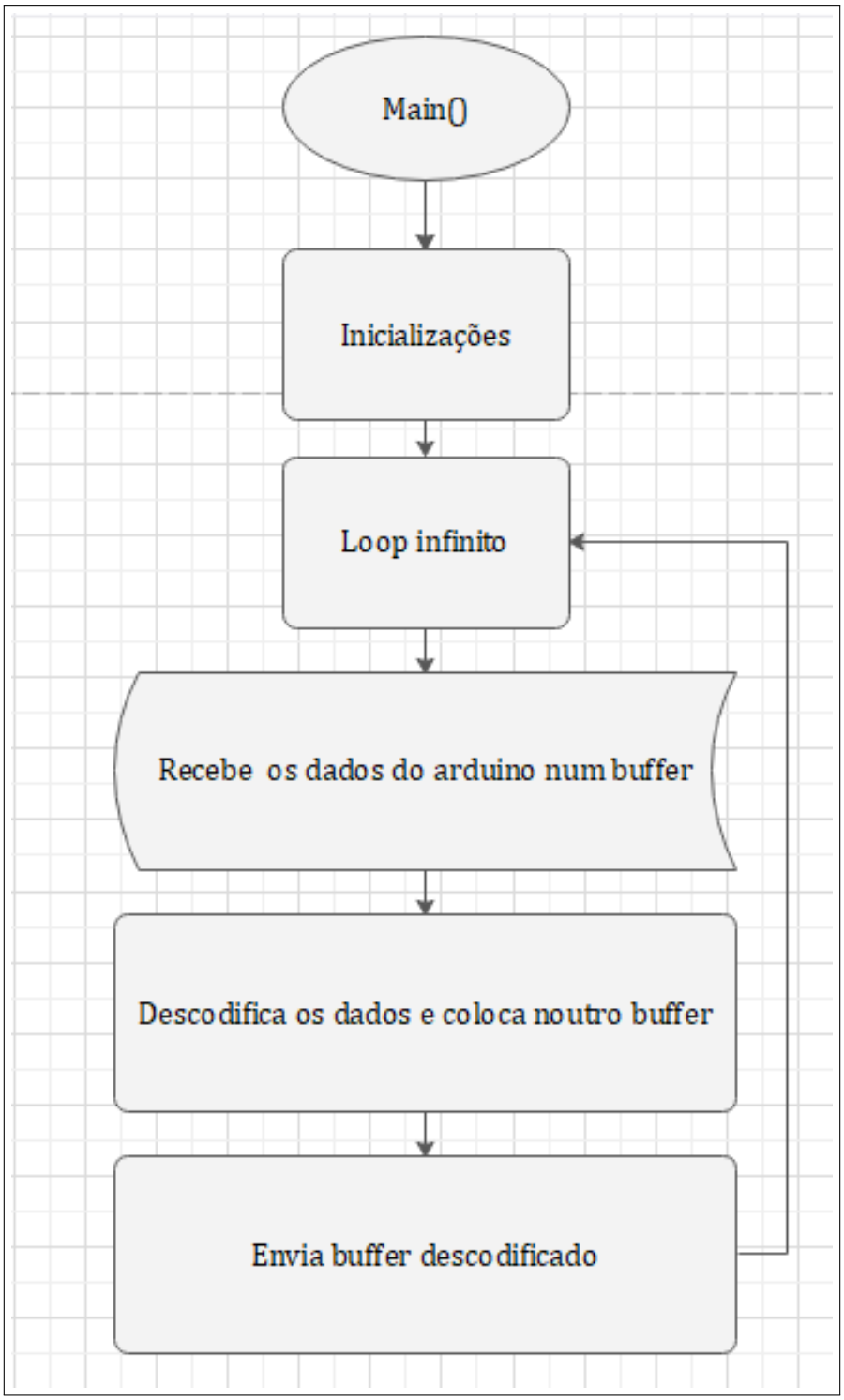
18 Sistema inicial

18.1 Fluxograma Arduino



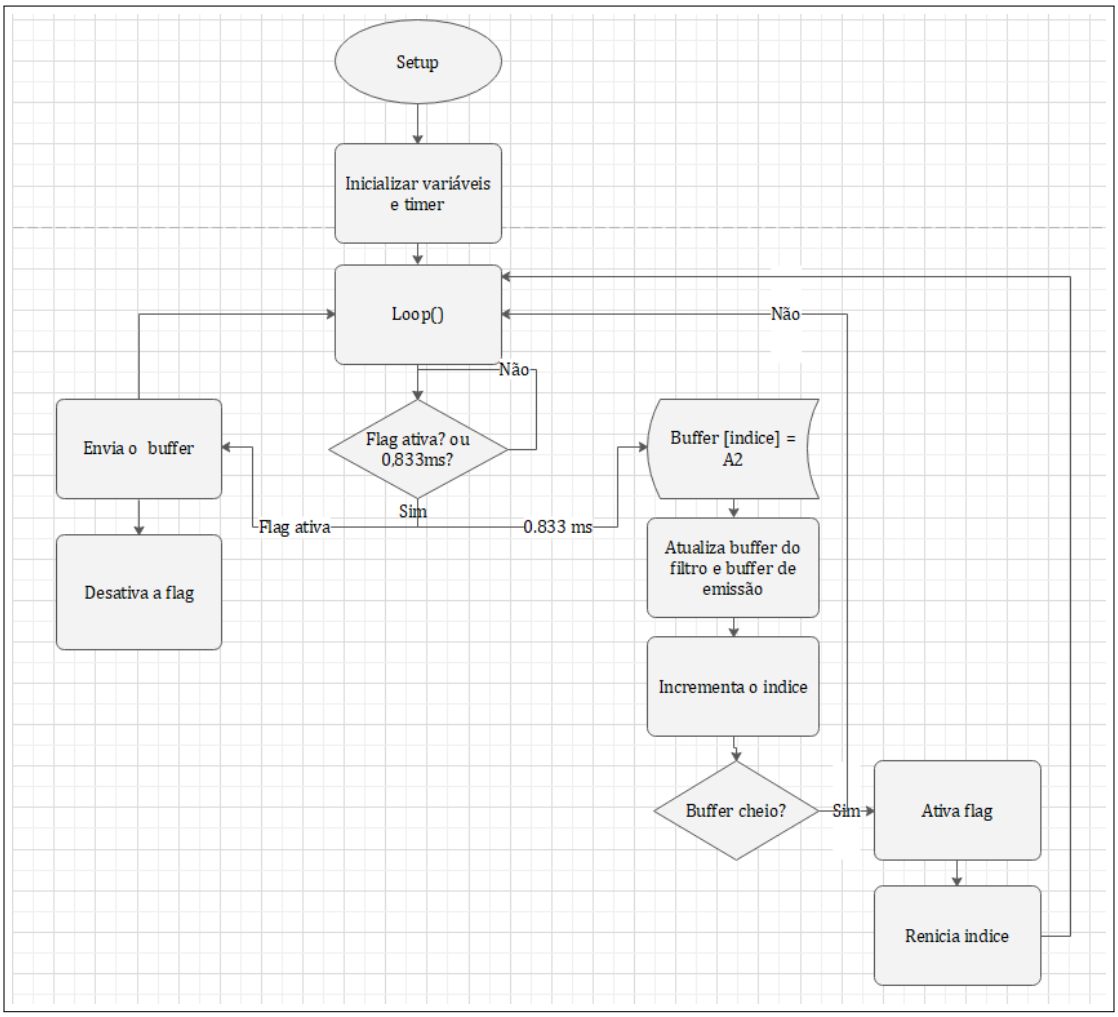
Nota: Observando o fluxograma mostra que não temos uma frequência de amostragem definida para aquisição dos dados do sensor mas sim de envio, que está mal...

18.2 Fluxograma da STM



19 Sistema final

19.1 Fluxograma Arduino



19.2 Fluxograma da STM

Como não foi alterado o que é enviado, a única diferença é que os dados estão filtrados, tanto o código como o fluxograma são iguais.

Contents