Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Visualização de dados e controle de um ventilador oscilatório de alta frequência

Felipe da Costa Malaquias

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia de Computação

Orientador
Prof. Dr. Adson Ferreira da Rocha

Brasília
2019

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Visualização de dados e controle de um ventilador oscilatório de alta frequência

Felipe da Costa Malaquias

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia de Computação

Prof. Dr. Adson Ferreira da Rocha (Orientador)
Universidade de Brasília

Prof. Dr. Wilson Henrique Veneziano
Universidade de Brasília

MSc. Filipe Emídio Tôrres
Universidade de Brasília

Brasília, 08 de Julho de 2019

Brasília/DF, Julho de 2019

Ficha Catalográfica

Malaquias, Felipe da Costa

Visualização dos dados e controle de um ventilador oscilatório de alta frequência.

54p., 210 × 297 mm (CIC/UnB, Graduação - Engenharia da Computação, 2019)

Trabalho de conclusão de curso em engenharia da computação
Universidade de Brasília, Campus Darcy Ribeiro – CIC/UnB

1. Ventilação	2. Ventiilação oscilatória de alta frequência
3. *Raspberry Pi*	4. *Arduino*
I. CIC/UnB	II. Adson Ferreira da Rocha

Referência

Malaquias, Felipe da Costa (2019). Visualização dos dados e controle de um ventilador oscilatório de alta frequência. Trabalho de conclusão de curso em engenharia da computação, Universidade de Brasília, Campus Darcy Ribeiro, DF, 54p.

Dedicatória

Dedico esse trabalho aos meus pais Arlete Batista da Costa e Cleber Malaquias Onofre, a minha irmã Elisa da Costa Malaquias e aos meus amigos e parentes que estiveram comigo durante essa jornada.

Agradecimentos

Agradeço aos professores da Universidade de Brasília por me guiarem nessa difícil jornada e agradeço especialmente ao professor Adson Ferreira da Rocha por me orientar neste trabalho.

Resumo

Ventiladores são aparelhos médicos de extrema importância, pois permitem que pacientes que tiveram a capacidade respiratória comprometida possam continuar respirando [1]. Estudos mostram que o uso de ventilação convencional, que tenta replicar a maneira como humanos respiram, pode causar danos ao pulmão do usuário, especialmente em recém nascidos e por isso outros métodos adequados devem ser utilizados [2]. Um desses métodos é a ventilação oscilatória de alta frequência. Este projeto é a continuação da implementação de um protótipo de ventilador oscilatório de alta frequência baseado no modelo de ventilador *3100A* da *SensorMedics*®. O foco principal deste projeto é na área de software, pois foi desenvolvida a interface que auxilia no controle da respiração do paciente. Para isso, foi utilizada a placa *Raspberry PI* em conjunto com outros componentes eletrônicos. O sistema desenvolvido neste trabalho permitiu que o usuário possa controlar a frequência e a amplitude de operação do ventilador além de permitir a visualização de diversos dados obtidos a partir de sensores instalados no protótipo.

Palavras-chave: respiração, ventilação, ventilador de alta frequência, *Raspberry PI*, equipamentos médicos.

Abstract

Mechanical ventilators are medical equipments of extreme importance since they assist patients who have had their breathing capacity compromised [1]. Studies have shown that conventional ventilation, the one that tries to imitate human breathing, may be linked to pulmonary injury especially when it comes to newborn babies therefore, in these cases, other types of ventilation are recommended [2]. One of these methods is the high frequency oscillatory ventilation. This project resumes the implementation of a prototype based on the model 3100A produced by SensorMedics®. The main focus of the project was the development of a graphical user interface that helps controlling the patient breathing. To do this, a Raspberry Pi is was used as the main board, together with a few other electronic components. The system developed in this project was able to allow the user to control the frequency and amplitude of operation and also allowed data visualization.

Keywords: breathing, ventilation, high frequency oscillatory ventilation, Raspberry PI, medical equipment.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACK *Acknowledgement.*

ARM *Advanced RISC Machines.*

CPU *Central Processing Unit.*

DSI *Display Serial Interface.*

GPIO *General Purpose Input/Output.*

GUI *Graphical User Interface.*

HFOV *High-Frequency Oscillatory Ventilation.*

I2C *Inter-Integrated Circuit.*

IDE *Integrated Development Environment.*

ISO *International Organization for Standardization.*

MVC *Model View Controller.*

SCL *Serial Clock.*

SD *Secure Digital.*

SDA *Serial Data.*

UART *Universal Asynchronous Receiver/Transmitter.*

USB *Universal Serial Bus.*

VOAF Ventilação Oscilatória de Alta Frequência.

1 Introdução

Ventilação (ou respiração) é o processo pelo qual seres vivos realizam trocas gasosas com o meio ambiente inalando oxigênio e exalando dióxido de carbono. Por se tratar de um ato essencial para a manutenção da vida no qual interrupções, mesmo que breves, podem levar a graves consequências, é de grande interesse da medicina o desenvolvimento de métodos artificiais de ventilação tendo em vista as inúmeras situações em que pacientes podem ter suas capacidades respiratórias comprometidas.

1.1 Definição do Problema Científico

O projeto consiste na continuação do desenvolvimento do protótipo de um ventilador de alta frequência baseado no modelo de ventilador *3100A High-Frequency Oscillatory Ventilation (HFOV)*. Esse modelo é produzido pela empresa *Vyaire Medical* que é especializada em produtos voltados para monitoramento, diagnóstico e tratamento de doenças respiratórias.

O protótipo já possui as peças necessárias para o funcionamento básico. Isto é, a ventilação funciona, porém sem a leitura dos sensores e sem a possibilidade de controle da frequência e amplitude por meios eletrônicos. Neste projeto, o foco será fazer a leitura dos sensores, e controlar as configurações do ventilador de maneira eletrônica e integrada.

1.2 Objetivos

O principal objetivo desse projeto é projetar e implementar o *display* que vai mostrar informações e realizar o controle do fluxo de ar no protótipo do ventilador oscilatório de alta frequência por meio do controle da frequência e amplitude. Para isso, alguns ajustes podem precisar ser feitos nos componentes atuais. A implementação tanto do *software* quanto do *hardware* devem seguir as boas práticas de desenvolvimento e as telas devem visar oferecer uma boa experiencia ao usuário.

1.2.1 Objetivos Específicos

Para se alcançar o objetivo geral foram definidos os seguintes passos:

- Configuração e montagem das placas e componentes eletrônicos utilizados no projeto;
- Desenvolvimento de um protótipo para que se possa obter familiaridade com as ferramentas utilizadas;
- Projetar a arquitetura geral do software e as principais classes que serão utilizadas nos módulos;
- Integrar os componentes desenvolvidos com o ventilador;
- Fazer ajustes no código para sanar possíveis erros gerados no processo de integração;
- Testar o funcionamento dos componentes desenvolvidos e recolher dados dos sensores para fazer as análises.

1.3 Justificativa

O desenvolvimento do protótipo não possui fins comerciais tendo em vista que aparelhos utilizados na área da saúde devem passar por diversos testes de qualidade extremamente rigorosos. No entanto, o desenvolvimento do protótipo pode levar a descobertas de melhorias relevantes e inovadoras para máquinas de ventilação e essas inovações podem ser aplicadas por empresas e profissionais da área.

1.4 Estrutura do Trabalho

Este trabalho está estruturado em 5 partes. O Capítulo 1 é a introdução onde o principal problema é apresentado, os objetivos do projeto são expostos e a estrutura geral do texto é descrita.

Em seguida, no Capítulo 2 será revisada a bibliografia de modo a expor alguns conceitos importantes para que assim se possa ter um melhor entendimento dos assuntos abordados nas próximas partes.

No Capítulo 3, o processo de desenvolvimento do *software* e do *hardware* utilizado para resolver o problema em questão serão detalhadamente descritos com auxílio de imagens e partes do código usado. Essa etapa é seguida dos resultados e discussão no Capítulo 4 onde os resultados obtidos com o projeto são analisados e é verificado se os objetivos foram de fato cumpridos.

Por fim, no Capítulo 5, o texto é encerrado com as conclusões obtidas a partir da análise dos resultados e algumas sugestões de trabalhos futuros para que o protótipo possa continuar a ser aperfeiçoado.

2 Fundamentação Teórica e Estado da Arte

Este capítulo irá abordar os assuntos mais importantes para o entendimento deste trabalho.

2.1 Repiração

A respiração tem como objetivo promover a troca de gases entre a atmosfera e o corpo humano, absorvendo oxigênio por meio dos alvéolos pulmonares e expelindo dióxido de carbono [3]. As células do corpo precisam de oxigênio para produzir a adenosina trifosfato, molécula responsável pela produção de energia livre na célula [4].

Em humanos, o ciclo respiratório consiste na repetição intercalada da inalação e da exalação sendo esses processos responsáveis por trazer o ar até os pulmões e retirar o dióxido de carbono do sistema respectivamente [3]. O número de ciclos respiratórios por minuto medidos em uma pessoa que se encontra em estado de repouso é chamado de frequência respiratória, sendo essa medida considerada parte dos quatro principais sinais vitais [5].

2.1.1 Sistema Respiratório

Segundo Valerie Scanlon e Tina Sanders, o sistema respiratório pode ser dividido em duas partes principais, são elas o trato respiratório superior e o trato respiratório inferior [4].

Os órgãos pertencentes a parte superior são:

- Nariz e cavidade nasal: porta de entrada do ar, é feito de osso e cartilagem. Pequenos fios e mucosa no interior das narinas são responsáveis pela filtragem e umidificação do ar [4];
- Faringe: canal mucoso que faz parte tanto do aparelho respiratório quanto do digestório. Mecanismos internos realizam a separação entre os alimentos e o ar sigam para

que esses caminhos diferentes, sendo transportados para seus respectivos sistemas encarregados [4];

- Laringe: liga a faringe à traqueia. Possui mucosa e cílios responsáveis por varrer a poeira e os micro-organismos para fora do aparelho respiratório. É neste órgão onde ficam as cordas vocais [4].

Os órgãos que integram a parte inferior são:

- Traqueia: se estende da laringe até os brônquios, que são as partes que se ligam ao pulmão. Chegando no pulmão, os brônquios se dividem várias vezes formando o que é chamado de árvore brônquica [4];
- Pulmão: estruturas similares a um balão, ficam localizados em ambos os lados do coração e são protegidos pela caixa torácica. Cada pulmão possui milhões de alvéolos [4];
- Alvéolos: unidades funcionais do pulmão que ficam no final da ramificação dos brônquios. São envoltos por uma fina membrana repleta de capilares sanguíneos. É neles onde ocorrem as trocas gasosas por meio da difusão do ar com o sangue nos capilares [4].

As membranas pleurais e o diafragma também fazem parte do sistema respiratório e tem como função respectivamente, proteger os pulmões dos atritos com a caixa torácica e promover o processo da respiração por meio do contínuo ciclo de contração e relaxamento [3]. A Figura 2.1 ilustra uma foto do sistema respiratório mostrando como os órgãos discutidos se conectam.

2.1.2 Mecânica da Respiração

Os impulsos respiratórios são gerados pela medula e pela ponte do sistema nervoso [4]. Esses impulsos ativam os músculos responsáveis pela ventilação sendo o principal deles o diafragma que na inspiração se contrai fazendo um movimento de descida junto aos pulmões permitindo a entrada do ar e na expiração relaxa deixando que a própria elasticidade dos órgãos comprima os pulmões expelindo o ar [3]. Uma representação desse processo pode ser visualizado na Figura 2.2. Durante esse processo, os músculos intercostais externos e internos participam expandindo e comprimindo a caixa torácica respectivamente [4].

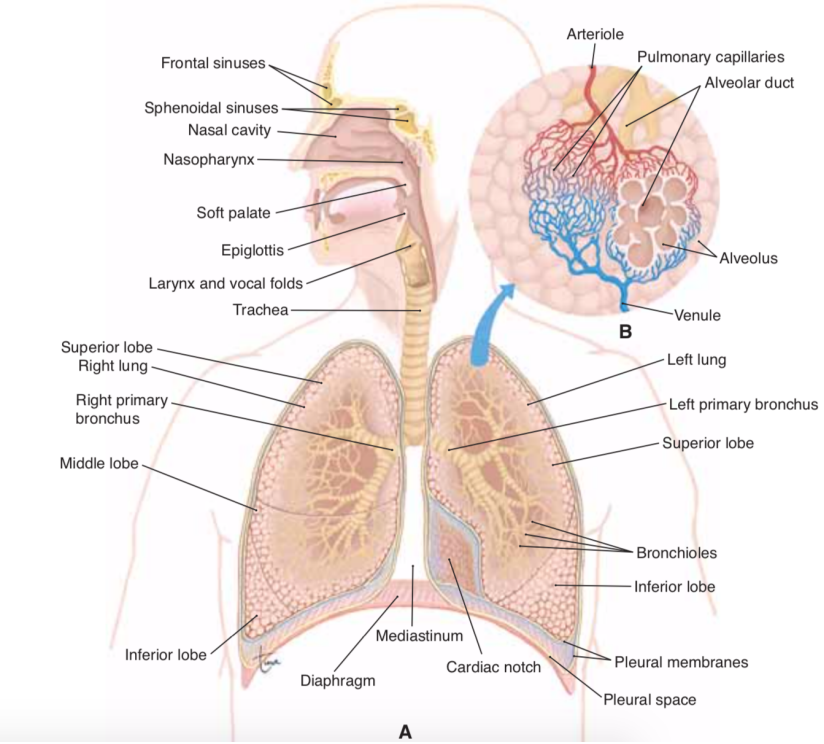

Figura 2.1: Imagem do sistema respiratório com legendas[4].

Vale ressaltar que na inspiração, a expansão do diafragma e dos pulmões faz com que a pressão intrapulmonar fique menor que a pressão atmosférica. Essa diferença faz com que o ar entre nos pulmões até que as duas pressões se igualem. Ao exalar, o volume do pulmão diminui expulsando parte do ar para que as pressões internas e externas se mantenham iguais.

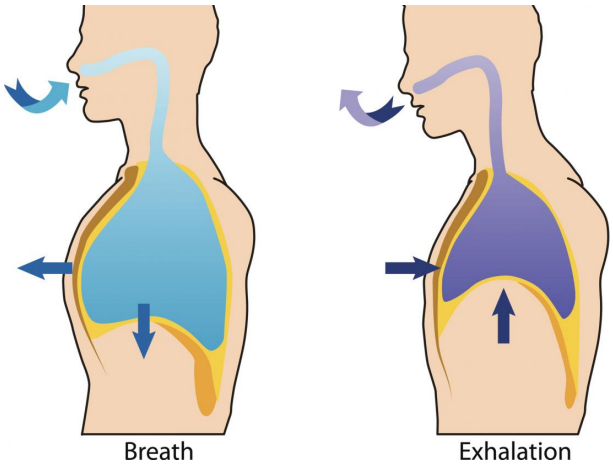

Figura 2.2: Representação da inalação e da exalação[6].

2.1.3 Volume do Pulmão

O volume máximo do pulmão varia de pessoa para pessoa baseado em fatores como idade e tamanho. Os tipos de volumes pulmonares, com exemplos baseados em um homem adulto [3], são:

- Volume corrente: Quantidade de ar envolvido na respiração normal. É de aproximadamente 500 mililitros;
- Volume de reserva inspiratório: Volume que o pulmão atinge quando se inspira com esforço total após a inspiração normal. É de aproximadamente 3 litros;
- Volume de reserva expiratório: Volume que pode ser expirado com esforço total após a expiração normal. É de aproximadamente 1,1 litros;
- Volume residual: Volume que fica no pulmão após uma expiração com força total. É de aproximadamente 1,3 litros.

O volume máximo de ar que pode passar pelo pulmão em um ciclo é dado pela soma desses quatro valores. A Figura 2.3 ilustra a relação entre esses volumes.

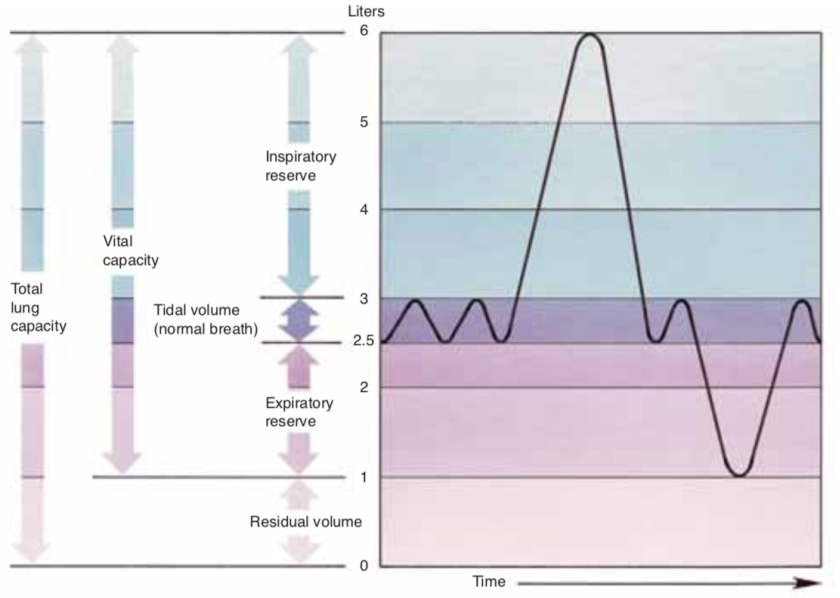

Figura 2.3: Gráfico representando os volumes respiratórios e como se relacionam[4].

2.2 Ventilação Mecânica

Quando uma pessoa se encontra com a capacidade respiratória comprometida, medidas urgentes devem ser tomadas para que o processo de trocas gasosas não seja interrompido

por muito tempo. Ventiladores mecânicos são máquinas que auxiliam o processo da respiração parcial ou integralmente. O acompanhamento médico durante o uso do ventilador é essencial tendo em vista que o aparelho interage com o sistema respiratório do paciente e portanto deve ser monitorado e ajustado para cada caso [1].

O ventilador pode funcionar por meio de tubos dentro da traqueia (métodos invasivos) ou por meio de máscaras na face ou no nariz (métodos não-invasivo)[7]. Também pode ser classificado pela maneira como o ar é colocado nos pulmões sendo a ventilação por pressão positiva aquela em que a pressão externa exercida pelo aparelho empurra o ar pelas vias aéreas e a ventilação por pressão negativa aquela em que se reduz a pressão nos pulmões para que o ar seja sugado [2].

O uso do ventilador deve servir apenas como apoio para o tratamento da doença que está causando a incapacidade respiratória. Exemplos de doenças em que o uso do ventilador é recomendado são a hipóxia (baixa concentração de oxigênio no sangue), a hipoventilação (quando não há ar suficiente para trocas gasosas satisfatórias no pulmão) e até mesmo quando o paciente precisa fazer muito esforço para respirar devido à alguma obstrução [2].

Outro motivo pelo qual a atenção médica durante o tratamento se faz necessária é o grande número de complicações que podem ser causadas pelo uso dessas máquinas. No caso de métodos invasivos de ventilação, os tubos podem causar ferimentos e rupturas na faringe, laringe ou traqueia por exemplo. A ventilação também pode causar barotrauma pulmonar, lesão pulmonar e atrofia do diafragma por falta de uso [2] [1] além de várias outras complicações. Portanto, é recomendado que a condição que está causando a incapacidade respiratória seja revertida o mais rápido possível para que uso de aparelhos ventilatórios seja o mais breve possível reduzindo os riscos associados.

2.3 Ventilador Oscilatório de Alta Frequência

Enquanto ventiladores convencionais tentam simular a respiração humana tanto na frequência dos ciclos quanto no volume corrente, um outro método chamado de ventilação oscilatória de alta frequência utiliza baixos níveis de volume corrente e uma frequência respiratória maior que a normal fazendo com que a pressão nas vias aéreas varie rapidamente em torno de uma média [2].

Uma peculiaridade desse tipo de ventilação é que diferentemente de outros tipos de ventilação em alta frequência, a parte da exalação acontece de maneira ativa, ou seja, a máquina auxilia a saída de ar do pulmão na parte negativa do ciclo [2].

Um estudo realizado em 2002 [8], comparou 500 recém nascidos que pesavam entre 600 e 1200 gramas em 26 unidades de tratamento intensivo diferentes. O objetivo era

saber se os neonatos que foram tratados com ventiladores oscilatórios de alta frequência teriam mais chances de conseguir respirar sem a necessidade de oxigênio suplementar após o tratamento. O estudo descobriu que 56% dos recém nascidos tratados com VOAF conseguiram passar a respirar sem a necessidade de oxigênio suplementar, enquanto que no outro método, apenas 47% conseguiram. Apesar da diferença ser pequena, ela mostra que existem benefícios para esse método de ventilação.

Em adultos, o uso de ventiladores convencionais pode estar associado à piora em quadros de lesão pulmonar [9] e portanto, medidas protetivas precisavam ser buscadas. De acordo com um estudo realizado por Jerry A. Krishman [10], o uso de ventiladores de alta frequência, especialmente os do tipo oscilatório, podem apresentar benéficos no tratamento desses pacientes porém ainda é preciso buscar mais evidências [10].

2.3.1 Ventilador *SensorMedics 3100A*

O ventilador oscilatório de alta frequência modelo *3100A* é produzido pela empresa *Vyaire Medical* e de acordo com o *website* da empresa, é recomendado para o uso em neonatos que tiveram capacidade respiratória comprometida.

De acordo com o manual [11], o *3100A* consiste de oito subsistemas, são eles:

- Equipamento de alimentação elétrica: converte voltagem AC para DC e alimenta os outros subsistemas;
- Controles e alarmes eletrônicos: faz o controle do sistema de oscilação e recebe dados do ventilador para alertar sobre possíveis erros;
- Sistema de monitoramento de pressão nas vias aéreas: mostra valores relativos à pressão das vias aéreas;
- Subsistema de oscilação: sistema que causa as oscilações através de ondas quadradas que controlam um pistão;
- Circuito do paciente: parte onde o ar passa e é tratado para chegar até o paciente;
- Sistema de lógica e controle da parte pneumática: esse subsistema realiza o controle de variáveis como a pressão média e limite das vias aéreas e o fluxo de ar;
- Umidificador: o umidificador deve ser feito especialmente para neonatais e ter um fluxo de até 40 litros por minuto;
- Misturador de gases respiratórios: permite controlar a quantidade de oxigênio no ar.

Desses subsistemas, o fornecimento dos últimos dois são de responsabilidade do usuário. Uma foto do modelo em questão sem o umidificador e sem o misturador de gases respiratórios pode ser vista na Figura 2.4.

Figura 2.4: Foto de um ventilador modelo *3100A*[12].

O modelo é capaz de ventilar pacientes de até 30 kg e possui um volume corrente de aproximadamente 1,5~3,0 cc/kg [13]. As configurações iniciais de frequência variam entre valores de 6 e 15 Hz baseado no peso do bebê e no tempo de gestação. Neonatos prematuros com menos de 2,5 kg por exemplo devem ser submetidos à uma frequência inicial de 15 Hz enquanto que a configuração de 6 Hz deve ser usada em crianças já crescidas e com mais de 10 kg [13].

2.4 Comunicação Serial

A comunicação serial, diferentemente da paralela, funciona enviando um *bit* para cada sinal de *clock* por meio de um canal de comunicação [14]. É uma das formas mais simples de comunicação entre dispositivos [15]. A comunicação funciona em *full duplex* ou seja, o envio e recebimento de dados pode acontecer simultaneamente. Como apenas um *bit* é enviado por vez, diversos problemas são evitados como por exemplo o *cross-talk* (que é quando o sinal de um fio interfere no sinal de outro que está próximo) e o número de fios é menor.

A transmissão de dados pode acontecer de forma síncrona ou assíncrona sendo a primeira quando os dispositivos conectados estão sincronizados por um *clock* em comum e a segunda quando a sincronia vem nos próprios dados (por exemplo na forma de sinais *start*/*stop*) e não nos *clocks* [16]. O *baud rate* especifica a velocidade com que os *bits* são

enviados na linha serial sendo que o inverso desta medida representa o tempo necessário para se enviar um único *bit*.

2.5 Protocolo de Comunicação *I2C*

O *I2C* é um protocolo de comunicação muito utilizado para comunicação de baixa velocidade entre dispositivos como circuitos integrados e microcontroladores ou comunicação *intra-board* entre circuitos [17]. Criado pela *NXP Semiconductors* em 1982, o protocolo contou com diversas melhorias introduzidas ao longo do tempo, inclusive por outras empresas, tornando-o assim, mais confiável e interoperável.

Para que a comunicação *I2C* seja possível, todos os dispositivos envolvidos devem ter suporte ao protocolo que pode ser implementado de diversas formas como, por exemplo, em *hardware*, *software* ou com circuitos integrados externos. No barramento *I2C*, os equipamentos conectados se dividem em dois tipos: mestre e escravo. O mestre é responsável por coordenar a comunicação entre ele e os múltiplos escravos suportados pelo sistema [17].

A comunicação no barramento se dá por dois fios chamados de SDA (*Serial Data*) e SCL (*Serial Clock*) que são responsáveis, respectivamente, pela transmissão dos dados em si e pelo controle do *clock*. Todos os dispositivos conectados ao barramento possuem um endereço lógico, sendo assim, o número máximo de nós no barramento se dá pelo número de *bits* utilizado no endereçamento que pode ser 7, 10 ou 16 *bits*. Um outro limitante do número de nós em um barramento *I2C* é a carga capacitiva que de acordo com o manual oficial do protocolo não deve ultrapassar o valor de 400pF.

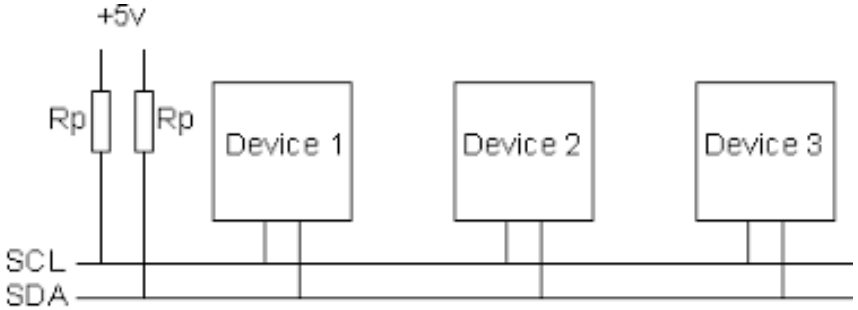

Figura 2.5: Representação de um barramento *I2C* com três dispositivos[18].

Conforme se observa na Figura 2.5, os fios SCL e SDA estão ligados à voltagem por dois resistores (Rp). Esses resistores atuam como resistores *pull-up*, ou seja, forçam um sinal padrão (*HIGH*) nos fios quando eles não estiverem sendo utilizados.

A Figura 2.6 demonstra como se dá a transferência de dados por meio do protocolo. Primeiramente, o fio SDA envia um sinal de *START* indicando o início da comunicação.

Os dados são transferidos de forma que a cada sinal de *clock* um *bit* é lido do SDA e a cada 8 *bits* transferidos um sinal de *Acknowledgement* (ACK) deve ser enviado, indicando o recebimento da informação. Após o fim da troca de mensagens, um sinal de *STOP* deve indicar o fim da comunicação. Vale ressaltar que antes de enviar ou receber a informação em si, o mestre deve sempre dizer com qual dispositivo no barramento ele está se comunicando.

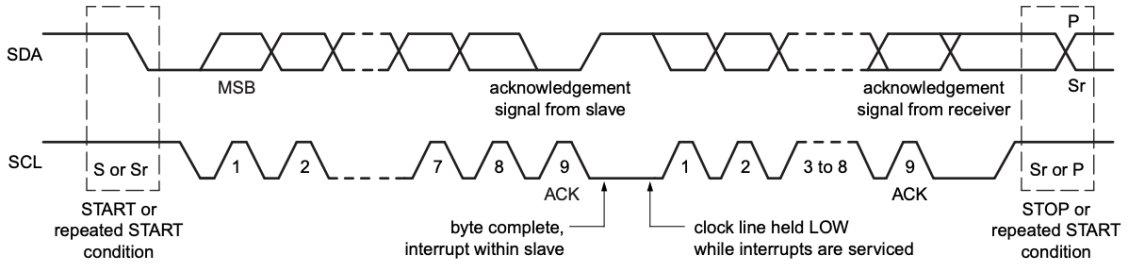

Figura 2.6: Imagem mostrando um exemplo de trânsferencia de dados por meio do protocolo *I2C*[19].

Apesar de ser um protocolo amplamente utilizado e com diversos benefícios, o *I2C* possui alguns problemas e limitações. O principal problema está relacionado à distância dos dispositivos já que as ligações mais longas acabam aumentando a capacitância e limitando a velocidade do barramento ou até mesmo impossibilitando o uso do protocolo.

2.6 Linguagem de Programação - *Python*

Python é uma linguagem de programação interpretada e de alto nível desenvolvido em 1991 por Guido Von Rossum. A linguagem é dinamicamente tipada e suporta diversos tipos de paradigmas de programação como orientação a objetos e programação funcional. O *Raspbian* oferece suporte ao *python* por meio de bibliotecas que facilitam a manipulação do hardware da *Raspberry Pi* [20].

2.7 *Arduino*

O *Arduino* é uma placa que possui um microcontrolador e diversos pinos que funcionam como saída e entrada de dados. O projeto é *open source* e é liderado pela empresa de mesmo nome [21]. Desde a sua criação, diversas versões diferentes foram criadas para atender o maior público possível. A grande maioria dessas versões, conta com microcontroladores *AVR ATMel* [21] e cristais osciladores de 16 MHz.

Devido à sua facilidade de uso e flexibilidade, o *Arduino* se tornou um dos microcontroladores mais utilizados entre hobistas, ganhando uma posição de destaque no desenvolvimento de protótipos que requerem microcontroladores, principalmente no meio acadêmico.

2.8 *Raspberry Pi*

O *Raspberry Pi* é um computador de placa única lançado em 2012 pela *Fundação Raspberry Pi* no Reino Unido. A placa possui todos os componentes de um computador (*System on a Chip*) e conta com portas USB e HDMI permitindo assim o uso de telas e dispositivos de *input*. Desde o lançamento, o *Raspberry* passou por diversas melhorias sendo que a mais nova versão, modelo 3 B+, conta com um processador ARM de 1,4 GHz 64 *bits* com quatro *cores*, *wireless* integrado e 1 GB de memória RAM [22]. O baixo custo da placa e a facilidade de uso fez com que ela se tornasse um grande sucesso mundial, chegando ao posto de computador britânico mais vendido [23].

O sistema operacional recomendado pelo fabricante é o *Raspbian* que pode ser obtido no próprio *website* da organização e instalado a partir de um cartão SD. O sistema é *open source*, baseado no *Debian* e otimizado para o *hardware* da placa. Ele já vem de fábrica com suporte para diversas linguagens como C e *Python* e conta com o gerenciador de pacotes APT [24].

2.9 Biblioteca de Interface Gráfica *PyQtGraph*

Biblioteca de interface gráfica multi-plataforma e código aberto com foco em aplicações de engenharia e ciência. Internamente, utiliza principalmente o *framework Qt* e o pacote *NumPy* que são respectivamente uma plataforma de desenvolvimento de interface gráfica e uma biblioteca de processamento numérico [25]. Diferentemente do *NumPy*, o *Qt* não é nativo do *python* e portanto precisa do auxílio do pacote *PyQt5* que atua como intermediário entre o *framework* e o *PyQtGraph*.

Quando comparado com outras bibliotecas gráficas focadas no uso científico, como o *matplotlib* por exemplo, o PyQtGraph apresenta maior facilidade de integração com elementos de interface de usuário e maior velocidade de execução [26] sendo portanto a escolha mais adequada para este projeto tendo em vista a necessidade de gráficos em tempo real e *input* do usuáro.

O *framework Qt* que gerencia as funcionalidades de GUI da aplicação funciona da seguinte maneira: a janela é representada por uma instância da classe *QMainWindow* e ela pode conter diversos elementos como botões, *labels* e gráficos. Todos os elementos que

podem ser adicionados dentro de uma janela são chamados de *widgets*. Uma janela pode conter quantos *widgets* sejam necessários sendo a limitação dada apenas pelo tamanho da mesma. Por fim, a organização desses elementos se dá por meio de um *layout* que controla a posição dos elementos dentro da janela. O PyQtGraph se integra nessa estrutura fornecendo seus elementos como *widgets* para que dessa forma, eles possam ser usados pela janela do *Qt*.

A instalação básica da biblioteca conta com um aplicação de exemplo que pode ser visualizada naFigura 2.7. Nela, percebe-se que há suporte para diversos tipos de gráficos e funcionalidades, como ajuste nas escalas, seleção de regiões importantes e *zoom*. A funcionalidade mais importante para este projeto mostrada no exemplo é a da atualização em tempo real pois é ela que vai permitir o desenvolvimento dos gráficos necessários.

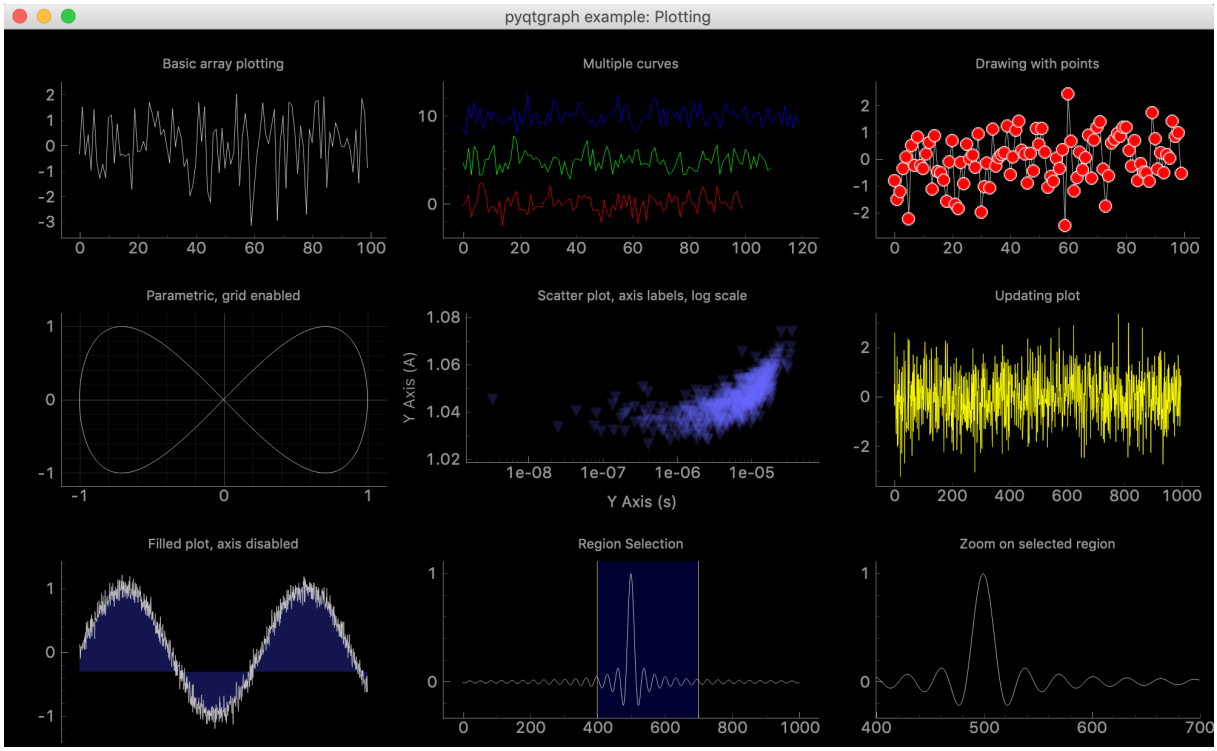

Figura 2.7: Exemplo de aplicação do *PyQtGraph*.

3 Metodologia

Na metodologia, todo o processo de implementação do sistema proposto será detalhadamente explicado.

3.1 Instalação do *Raspbian*

Para instalar o sistema operacional *Raspbian* foi necessário um cartão SD, um teclado e um monitor conectados ao *Raspberry Pi*. A documentação oficial recomenda que a instalação seja feita por meio do *software* NOOBS (*New Out Of the Box Software*) que pode ser baixado no site oficial. Feito o *download*, a imagem ISO do instalador foi passada para o cartão SD e a instalação do *Raspbian* foi feita.

3.2 Montagem da Tela *Touchscreen*

Para que os gráficos e dados possam ser mostrados foi escolhida a tela oficial do *Raspberry Pi*. Por ser *touchscreen*, o aparelho já permite a entrada de dados do usuário sem a necessidade de outros dispositivos. A tela tem 7 polegadas, resolução de 800 x 480 pixels e tem suporte para detecção *multi-touch* até o limite de 10 toques. Os *drivers* de suporte para a tela já vem inclusos nas versões mais novas do *Raspbian OS* e portanto não há necessidade de nenhuma configuração adicional em *software* [22]. Uma foto da tela e dos componentes que compõe o kit pode ser vista na Figura 3.1.

Os componentes do kit são:

- Placa adaptadora que lida com a conversão da tensão de alimentação e com o tratamento do sinal *touch* de entrada;
- Parafusos de apoio para fixar a placa adaptadora e a *Raspberry Pi* na tela;
- *Jumpers* fêmea-fêmea para conectar os pinos de alimentação do *Raspberry* na placa adaptadora;
- Cabo *flat* responsável por fazer a troca de dados entre a placa adaptadora e o *Raspberry Pi*. Ele deve ser conectado na porta DSI.

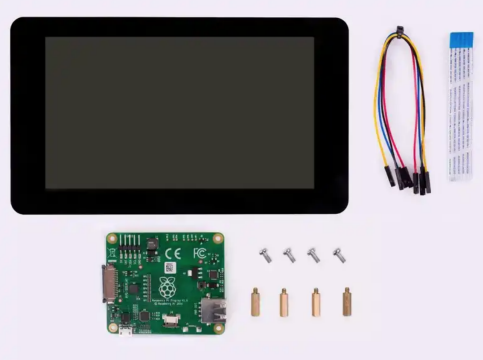

Figura 3.1: Foto do kit que acompanha a tela[22].

A Figura 3.2 mostra a montagem da tela e a conexão dela com o *Raspberry Pi*. O número 1 mostra os *jumpers* que são utilizados para a alimentação da tela, no 2 encontra-se os cabo *flat* usado para troca de dados entre o *Raspberry Pi* e a placa adaptadora. Por fim, o 3 mostra o cabo utilizado pela placa para enviar os dados para a placa.

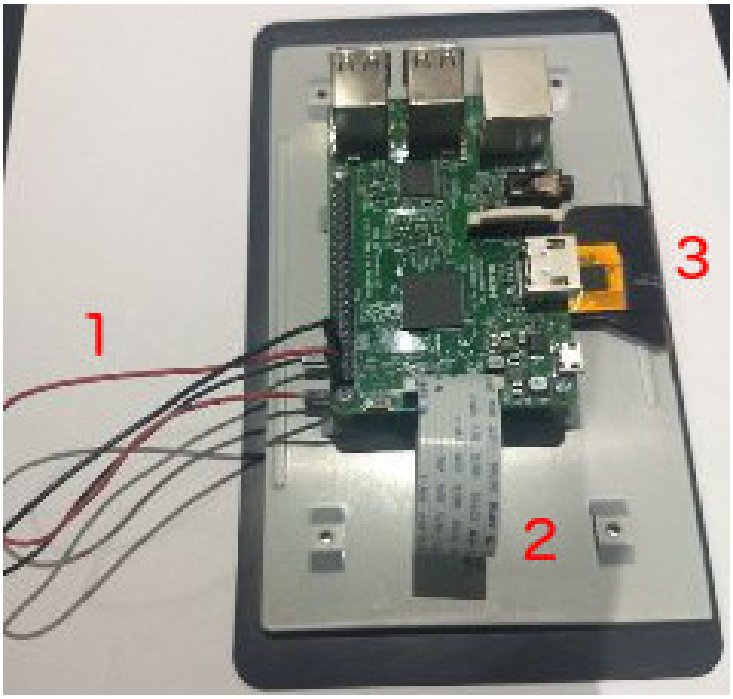

Figura 3.2: Foto da tela após a montagem.

3.3 Configurando o *Raspberry Pi* para comunicação *I2C*

Na configuração de fábrica, o *Raspberry PI* vem com a interface *I2C* desabilitada e portanto o primeiro passo deve ser habilitá-la. Para isso, primeiramente deve-se retirar o *I2C* da *blacklist* no arquivo */etc/modprobe.d/raspi-blacklist.conf*, em seguida é preciso adicionar a linha *i2c-dev* ao arquivo */etc/modules* e instalar o pacote *i2c-tools*. Após a instalação a interface deve ser habilitada a partir do terminal, na tela mostrada na Figura 3.3 que é acessada pelo comando *sudo raspi-config*.

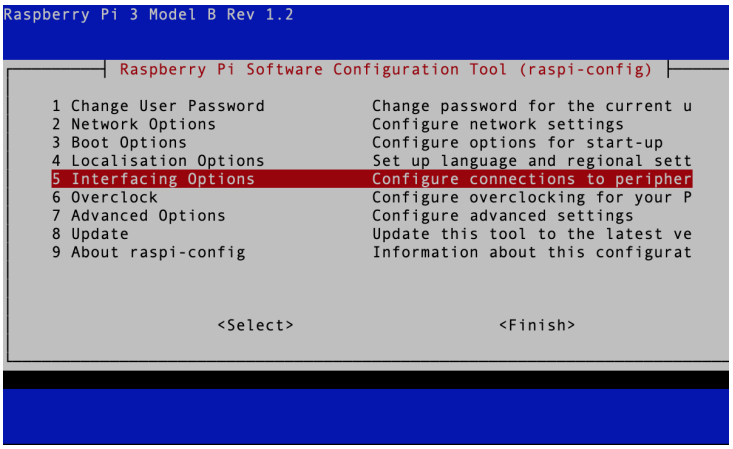

Figura 3.3: Tela de configuração do *Raspberry Pi* acessada pelo terminal.

Para testar se a interface foi habilitada, um teste foi realizado com uma placa *Arduino UNO*. Os pinos GPIO 2 e 3 do *Raspberry Pi* (SDA e SCL respectivamente) foram conectados aos pinos A4 e A5 do *Arduino*. Após configurar o *Arduino* para atuar no endereço 0x04, utilizou-se o comando *i2cdetect -y 1* no *Raspberry* que retornou uma tabela com os dispositivos conectados ao barramento conforme a Figura 3.4.

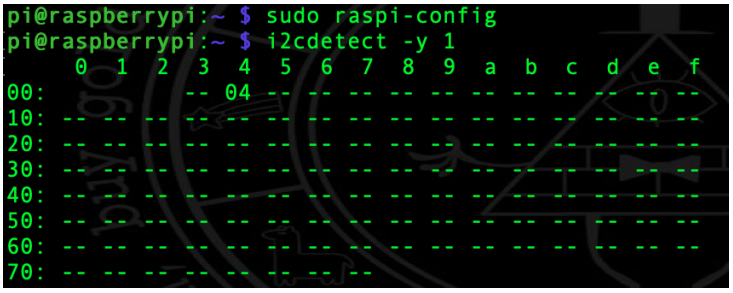

Figura 3.4: Leitura de endereços *I2C* após a conexão entre o *Raspberry Pi* e o *Arduino*.

Vale ressaltar que enquanto o *Raspberry Pi* roda a 3,3 V, o *Arduino* funciona a uma voltagem de 5 V e essa diferença poderia causar problemas na comunicação entre os

dispositivos caso não fosse utilizado um conversor de nível. No entanto, diferentemente do *Arduino*, os pinos de comunicação do *Raspberry Pi* possuem resistores *pull-up* de 1k8 ohms ligados ao trilho de voltagem de 3,3 V e portanto a comunicação entre os dois não causa problemas.

3.4 Desenvolvimento do *Software*

Para Sommervile, o conjunto de atividades e resultados associados que produzem um produto de *software* é chamado de processo de *software* [27]. O processo de software pode ser dividido em atividades sendo e as atividades consideradas fundamentais são:

- Especificação: onde os desenvolvedores definem o *software* a ser produzido;
- Desenvolvimento: o *software* é projetado e programado;
- Validação: parte onde se verifica se as especificações foram cumpridas;
- Evolução: o *software* é modificado para se adaptar à mudanças.

3.4.1 Modelo de Processo de Software

O modelo é uma descrição simplificada de como essas atividades, os produtos de software e as pessoas envolvidas devem interagir para se atingir o objetivo desejado [27]. Existem diversos modelos que devem ser escolhidos tendo em mente o tipo de *software* que se pretende desenvolver.

No desenvolvimento do *software* deste trabalho será utilizado o modelo em cascata. Nesse modelo, as atividades são divididas em estágios de forma que só se pode passar para o próximo estágio quando o anterior estiver concluído. A Figura 3.5 mostra as etapas do modelo e como elas se relacionam. De acordo com Sommervile [27], as etapas são:

- Definição de requisitos: definição dos serviços, restrições e objetivos do sistema;
- Projeto de sistemas e do software: estabelecimento da arquitetura geral do sistema identificando os componentes necessários e suas relações;
- Implementação e teste de unidades: o *software* é desenvolvido e suas unidades são testadas;
- Integração e teste de sistemas: as unidades do sistema são integradas e testadas;
- Operação e manutenção: com o sistema em operação, a manutenção se dá pela correção de erros que não foram detectados nas fases anteriores.

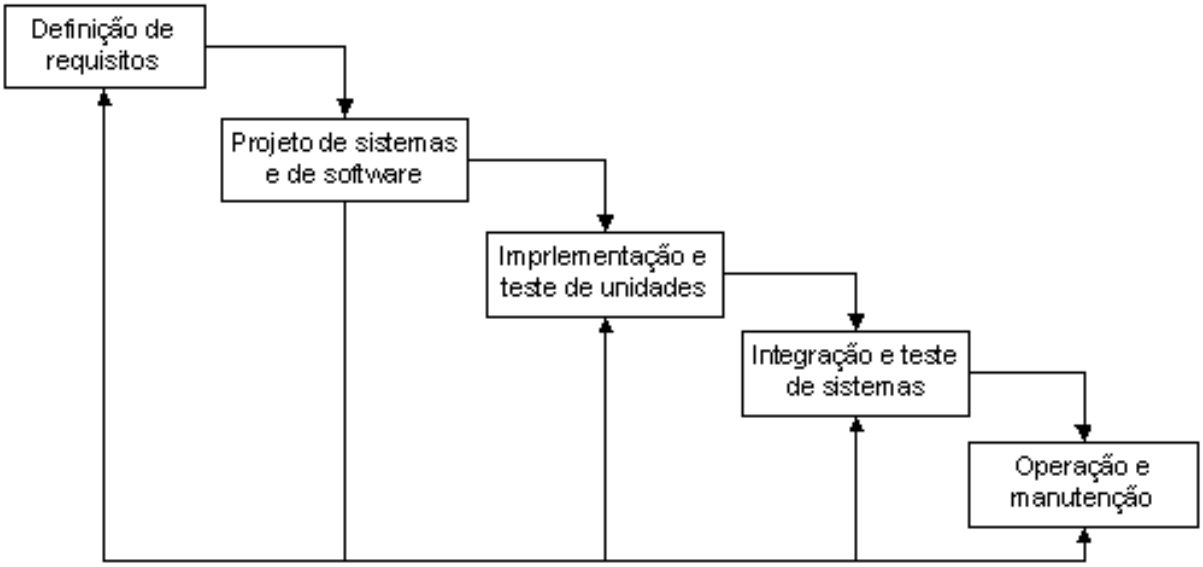

Figura 3.5: Representação do modelo cascata[27].

A escolha desse modelo se deu pelo fato de que os requisitos do sistema eram muito bem definidos no início do projeto e não iriam mudar durante a fase de desenvolvimento.

3.4.2 Ferramentas Utilizadas

Para o desenvolvimento do *software* que irá ser executado no Raspberry Pi, foi escolhida a linguagem *Python 3* pois ela é suportada pelo sistema operacional além de possuir as bibliotecas necessárias para a implementação do sistema proposto. Os programas desenvolvidos em *Python 3*, são executados por um interpretador que roda no terminal a partir do comando *python3*. Foram usadas as bibliotecas *pyqtgraph*, *smbus* e *serial*, sendo a primeira para a programação relativa a interface gráfica e as duas últimas para comunicação entre dispositivos.

O *software* desenvolvido para as placas *Arduino* usadas no projeto foi programado com a linguagem *C* na IDE oficial fornecida pelo site da empresa que produz a placa. O código desenvolvido é passado para o compilador *avr-g++* que é compilado e passado para a memória interna do *Arduino*.

3.4.3 Arquitetura

A arquitetura do *software* será uma adaptação do padrão MVC, isto é, dividida em três módulos chamados de modelo, visão e controle que serão responsáveis respectivamente por: manipular os dados, cuidar da interface gráfica e interação com o usuário, e por

fim, realizar a comunicação entre os dispositivos do sistema (*Raspberry Pi*, *Arduinos* e sensores).

O diagrama representado na Figura 3.6 mostra como os módulos do programa devem interagir entre eles e os agentes externos. A ideia da arquitetura é que o usuário por meio da interação com o módulo da visão possa alterar o estado do modelo que vai atuar como um meio termo entre os dois outros módulos. A estrutura do ventilador vai fornecer dados para o modelo utilizando o módulo de controle como interface e vice e versa. Pelo *design* da arquitetura, espera-se que os módulos de visão e controle nunca troquem informação diretamente já que isso implicaria no tratamento e manipulação de dados fora do modelo.

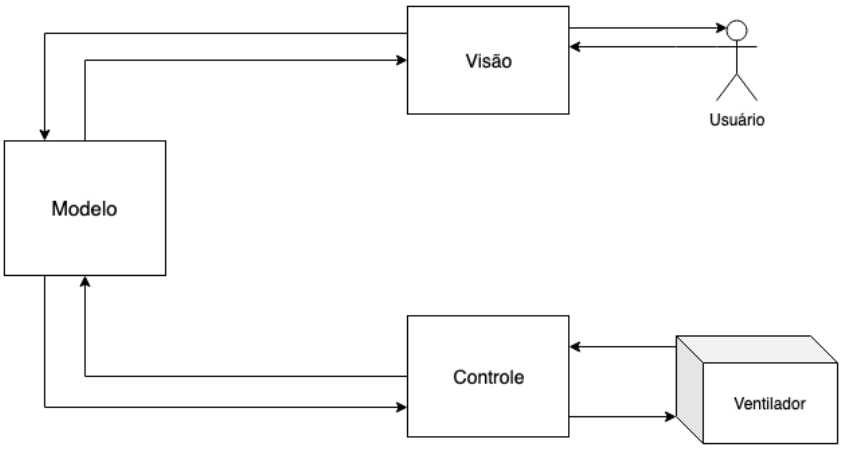

Figura 3.6: Diagrama da arquitetura do sistema.

As atualizações da tela e das leituras foram configuradas para acontecer a cada 20 milissegundos por meio da classe *QtTimer* do pacote *QtCore*. Dentro desta classe, podemos passar uma função que será chamada durante a execução do programa em um intervalo definido. A função de *update* será chamada do módulo de visão e será propagada para os diversos módulos atualizando todo o estado da máquina.

3.4.4 Módulo - Visão

A interface gráfica é a parte do *software* que interage diretamente com o usuário e por isso precisa ser pensada e desenvolvida para oferecer a melhor experiência. A GUI do protótipo do ventilador de alta frequência precisa de 3 áreas principais, são elas:

- Gráficos: nessa área ficam os gráficos sendo atualizados em tempo real com as leituras dos sensores do aparelho;
- Informações: aqui ficam as informações do sistema como quais sensores estão conectados e alertas dos possíveis erros que podem ocorrer no ventilador;

- Controle: parte responsável pelo controle da ventilação. Valores de frequência e amplitude do sinal por exemplo são controlados nessa área.

O design da interface gráfica foi feito com auxílio da ferramenta MockFlow disponível online no sítio https://www.mockflow.com/app/wireframe [28]. O protótipo desenvolvido pode ser visto na Figura 3.7.

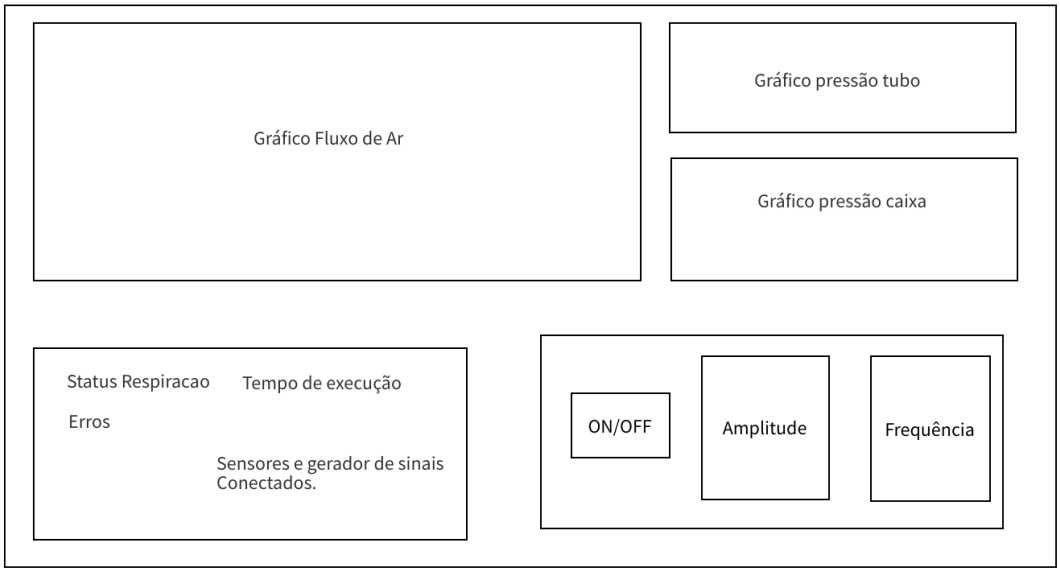

Figura 3.7: Protótipo da tela desenvolvido no *MockFLow*.

Pode-se observar que a divisão das áreas se deu da seguinte forma: gráficos ocupando a parte superior inteira, informações na parte inferior esquerda e controle na parte inferior direita.

A partir do protótipo, foram projetados os principais componentes que irão compor o módulo de visão e a relação entre eles e o diagrama de classes resultante pode ser visto na Figura 3.8. Cada área da interface gráfica possui uma classe correspondente e todas herdam a classe *QGridLayout* do Qt. O *QGridLayout* permite que os elementos do *layout* sejam organizados em uma grade e que seja determinada quantas linhas e colunas da grade cada elemento vai ocupar. Cada uma das classes tem a responsabilidade de organizar seus elementos dentro do seu *layout* e a organização na tela fica por conta da classe *MainLayout* que também herda a classe *QGridLayout*.

Para atualizar a tela e o estado da máquina a classe *MainLayout* possui a função *update* que chama as funções de mesmo nome em cada uma das áreas da interface e dos outros módulos. Vale ressaltar que a atualização da parte de informações só acontece uma vez a cada cem chamadas da função já que ela realiza comunicações por meio serial

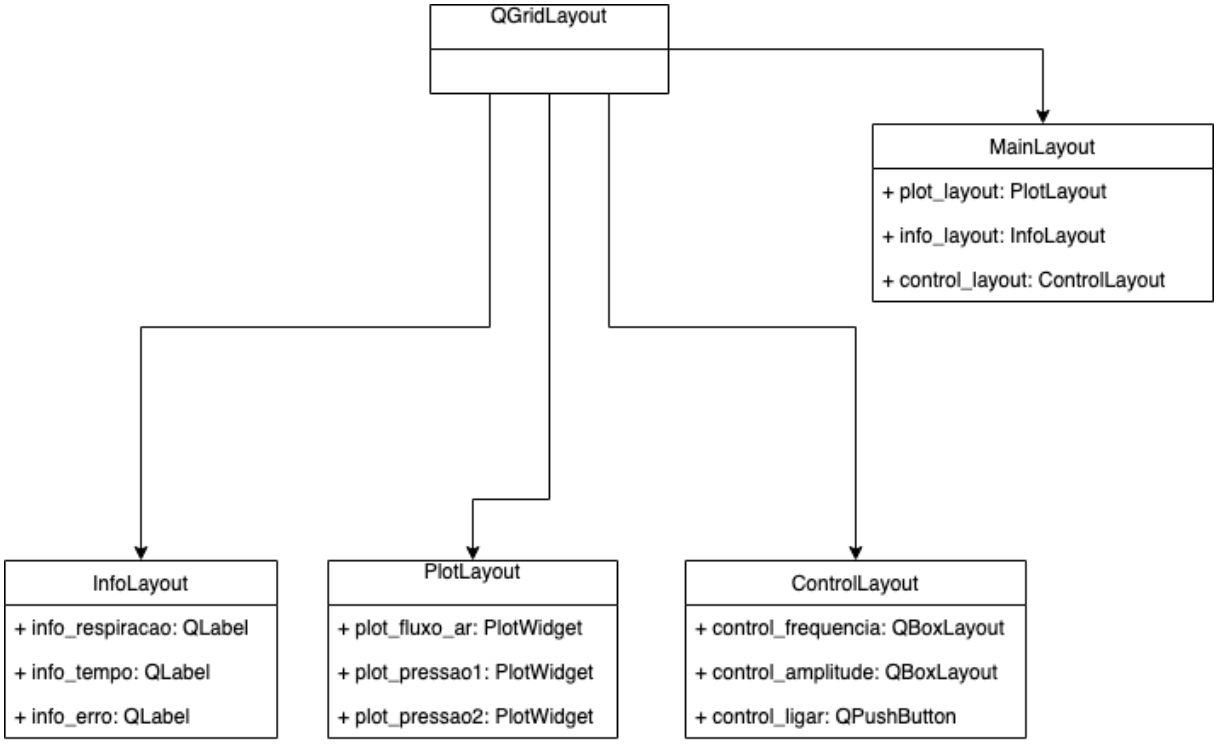

Figura 3.8: Diagrama de classes do *layout* proposto.

e *I2C* entre os dispositivos e isso leva um certo tempo. Caso essa atualização acontecesse em todas as chamadas, as outras partes do programa seriam prejudicadas.

O *Qt* oferece a opção de configuração de estilos como cor de fundo, fontes e bordas por meio de arquivos do tipo *QSS* que podem ser desenvolvidos separadamente e incluídos no programa dinamicamente permitindo que a interface tenha varias opções de temas.

A Figura 3.9 mostra uma captura de tela da interface gráfica desenvolvida.

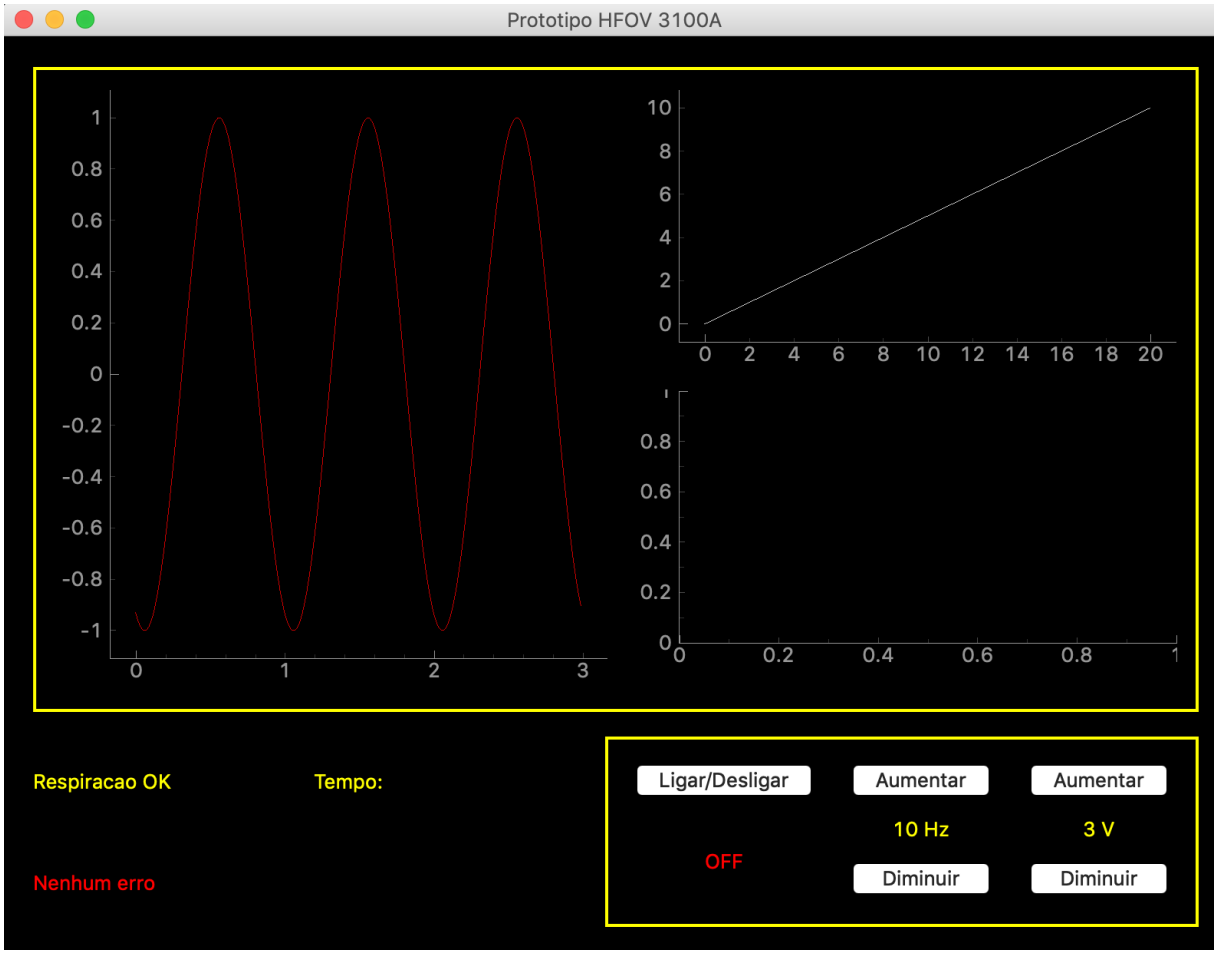

Figura 3.9: Captura de tela da interface gráfica desenvolvida.

3.4.5 Módulo - Controle

No módulo de controle, acontece toda a comunicação, desde a configuração até a troca de dados entre os componentes. São quatro os componentes externos que irão se comunicar com o programa e todos eles são *Arduinos*.

O *Arduino* possui algumas limitações que causaram problemas durante o desenvolvimento sendo a principal delas o fato de que a biblioteca *Wire.h* usada na comunicação *I2C* não suporta barramentos com mais de um mestre. Por isso, a comunicação entre os *Arduinos* do sensor de ar e do gerador de sinais precisou ser feita por meio de comunicação Serial por meio do cabo *USB* e dos pinos UART devido ao fato de que estes *Arduinos* já estavam atuando como mestres em outros barramentos *I2C*.

As conexões se deram da seguinte forma:

- Sensor de fluxo ar: a leitura do sensor é feita por um *Arduino Uno* por meio de uma conexão *I2C*. Após a leitura o *Arduino* repassa essa informação para o *Raspberry Pi*.

A leitura não pode ser feita diretamente pois há incompatibilidade entre o software do *I2C* utilizado na biblioteca *smbus* no *Raspberry* e o sensor;

- Sensor de pressão: a leitura é feita por uma porta analógica do *arduino*. Como os pinos *I2C* ficam livres, a transferência de dados entre o *Arduino* e o *Raspberry pi* podem acontecer por meio desse protocolo;
- Gerador de sinais: o gerador de sinais utilizava um *Arduino Pro Mini* que precisou ser substituído por um *UNO* devido às dificuldades de se utilizar a comunicação serial.

A comunicação com dispositivos que utilizam esses protocolos são extremamente complexas de serem implementadas do zero, no entanto, o sistema operacional *Raspbian* já vem com bibliotecas de *Python* que auxiliam na comunicação serial e *I2C*, são elas respectivamente: *serial* e *smbus*.

A biblioteca *serial* fornece a classe *Serial* que deve ser instanciada recebendo como parâmetros o nome da porta e o *baudrate* que para ambas as conexões será 9600 símbolos por segundo. As portas USB no *Raspberry Pi* são representadas por arquivos que possuem a nomenclatura no formato */dev/ttyACMX* onde o X é um número que representa a ordem em que os dispositivos são conectados, ou seja, o primeiro dispositivo fica na porta */dev/ttyACM0*, o segundo na */dev/ttyACM1* e assim em diante. No código, o gerador de sinais ficou na porta 0 e o sensor de fluxo de ar na porta 1 e portanto eles devem ser conectados nessa ordem.

A comunicação se dá pelos métodos *readline()* e *write* sendo que a leitura do primeiro e os parâmetros de escrita do segundo são em bytes puros, ou seja, não podem ser de nenhum dos tipos nativos do *python*. Para isso, o valor deve ser convertido para *string* e a conversão deve ser feita com os métodos *decode('utf-8')* para a leitura e *encode()* para a escrita.

O código a seguir mostra a classe base para realização da comunicação serial. Os métodos presentes na classe podem ser usados para enviar e receber dados e também podem ser adaptados para usos específicos. No caso do gerador de sinais por exemplo, o método que envia os dados foi modificado para que o valor enviado seja ajustado de acordo com as regras estabelecidas pelo *Arduino*.

```
1 import serial
2
3 class SerialComm():
4     """
5     Classe base para comunicacao serial.
6     Recebe como parametros a porta e o baud rate que por padrao
7     sao /dev/ttyACM0 (primeira conexao) e 9600 bps
```

```
8       """
9       def __init__(self, portName="/dev/ttyACM0", baudrate=9600):
10          self.portName = portName
11          self.baudrate = baudrate
12          self.ser = None
13          self.connect()
14
15      def is_connected(self):
16          """
17          Retorna verdadeiro caso a conexao esteja ativa.
18          """
19          if self.ser:
20              return True
21          return False
22
23      def connect(self):
24          """
25          Tenta realizar a conex o e trata a exception no
26          caso de erro
27          """
28          try:
29              self.ser = serial.Serial(self.portName, baudrate=self.baudrate)
30          except serial.serialutil.SerialException:
31              self.ser = None
32
33      def read_value(self):
34          """
35          Faz a leitura do valor e converte para utf-8. No caso de
36          exception atualiza a conex o.
37          """
38          if self.ser:
39              try:
40                  value = float(self.ser.readline().decode("utf-8"))
41                  return value
42              except serial.serialutil.SerialException as e:
43                  self.ser = None
44          return 0
45
46      def set_value(self, new_value):
47          """
48          Envia o valor recebido no parametro new_value.
49          """
50          if self.ser:
51              try:
52                  self.ser.write(str(new_value))
```

```
53                  return True
54              except serial.serialutil.SerialException:
55                  self.ser = None
56          return False
```

No *Arduino* que faz a leitura do sensor de fluxo, foi adicionada a função *Serial.print* no final do *loop* principal. No gerador de pressão, três *bytes* são enviados pelo *Raspberry Pi* sendo que o primeiro representa qual parâmetro deve ser mudado (1 para frequência e 2 para amplitude) e os dois últimos representam o valor. Os números 300 e 400 representam os comandos para ligar e desligar o aparelho respectivamente.

Já para o *I2C*, a biblioteca *smbus* fornece a abstração necessária para o uso do protocolo. Primeiramente é preciso criar uma instância do barramento que então pode ser usado com as funções de leitura e escrita. Os sensores de pressão utilizados se comunicam com o *Arduino* por meio das portas analógicas e o *Arduino* por sua vez se conecta ao *Raspberry* por meio dos pinos A4, A5 (SDA e SCL respectivamente) e terra.

O código abaixo mostra como funciona a classe base utilizada na comunicação por *I2C*. Nela, existem métodos para leitura e escrita de dados e métodos que lidam com a situação da conexão. Nenhuma configuração especial precisa ser feita para se conectar dispositivos pelo *I2C* e portanto o método *connect* apenas testa se a conexão está ativa e atualiza a variável. O objeto que representa o barramento deve ser declarado fora da classe pois ele é único para todas as instâncias.

```
1  import smbus
2  bus = smbus.SMBus(1)
3
4  class I2CComm():
5      """
6      Classe base para comunicacao I2C.
7      Recebe como parametro o endereco do dispositivo
8      que o objeto vai se comunicar
9      """
10     def __init__(self, addr):
11         self.addr = addr
12         self.connected = False
13         self.connect()
14
15     def is_connected(self):
16         """
17          Retorna verdadeira caso a conexao esteja ativa
18         """
19         con = True
20         try:
21             bus.read_byte(self.addr)
```

```
22          except Exception as e:
23              con = False
24          self.connected = con
25          return con
26
27      def connect(self):
28          """
29          Tenta realizar a conex o e retorna o resultado
30          """
31          self.is_connected()
32          return self.connected
33
34      def read_value(self):
35          """
36          Faz a leitura do valor no barramento
37          """
38          if self.connected:
39              value = bus.read_byte(self.addr)
40              return value
41          return 0
42
43      def set_value(self, new_value):
44          """
45          Escreve o valor do parametro new_value no barramento
46          """
47          if self.connected:
48              bus.write_byte(new_value)
49              return True
50          return False
```

No código que rodava no *Arduino* foi necessário incluir a funcionalidade do *I2C* que é feita por meio da biblioteca *Wire*. Após definir o endereço do dispositivo no barramento (0x04 e 0x05) a função *onRequest*, que é chamada sempre que o mestre requisita dados, recebeu como parâmetro a funcionalidade de enviar a última leitura do sensor.

3.4.6 Módulo - Modelo

O modelo é a representação do ventilador em código. Ele armazena todos os dados dos sensores além dos valores de frequência e amplitude nos quais a máquina está operando. Para isso, foi criada a classe *EstadoMaquina* que deve ser instanciada uma única vez no programa, já que duas ou mais instâncias configurariam um caso de inconsistência.

Também é função do modelo tratar os dados e verificar possíveis erros na operação do ventilador. O tratamento de dados se resume a conversão e ajustes para facilitar a

visualização e a manipulação pelos outros componentes. Os erros detectáveis são quando o fluxo de ar é interrompido e quando a leitura da pressão age de maneira indevida. Ao detectar algum erro o módulo passa um sinal para o módulo de visão que fica encarregado de mostrar o erro para o usuário.

O código do construtor da classe principal do modelo se chama *EstadoMaquina*. A variável externa *status_created* garante que apenas uma instância da classe seja criada. Todos os outros métodos da classe são utilizados pelos outros módulos para alterar as variáveis declaras no construtor.

```
1  # Forma de garantir que apenas um objeto Status seja criado
2  status_created = False
3
4  class EstadoMaquina():
5      """
6      Guarda todas as variaveis de estado da maquina.
7      Recebe amplitude e frequencia iniciais.
8      Deve ser um Singleton.
9      """
10     def __init__(self, frequencia, amplitude, ligado=True):
11         global status_created
12         if status_created:
13             raise Exception("Tentativa de criacao de multiplos Status")
14         status_created = True
15         self._frequencia = frequencia
16         self._amplitude = amplitude
17         self._ligado = ligado
18         self.erro = None
19         self.erro_list = {"fluxo_ar_interrompido": "Fluxo de ar foi
               interrompido",
20                           "problema_pressao_tubo": "A pressao no tubo
                                 apresenta problemas",
21                           "problema_pressao_caixa": "A pressao na caixa
                                 apresenta problemas"}
22         self.time = 0
23
24         self.sensors = connections.SensorComponents()
25         self.gerador = connections.Gerador()
```

3.5 Montagem

Na fase de montagem, ocorreu a integração do sistema desenvolvido neste trabalho com a estrutura já existente do protótipo do ventilador.

A estrutura do ventilador mostrada na Figura 3.10 conta com as seguintes peças:

- Fonte de alimentação: uma fonte que é ligada na tomada e fornece uma tensão de 12 V para alimentar o controle da caixa;
- Caixa de sustentação: uma caixa de som foi aberta e o *speaker* foi colocado em uma caixa de madeira com a saída isolada para que não ocorra o vazamento de ar. O amplificador que controla o alto falante recebe em sua entrada uma tensão de 12 V para alimentação e o sinal do gerador de função;
- Gerador de função: o gerador de função é um circuito que gera uma onda senoidal com frequência e amplitude ajustáveis. Ele deve ser alimentado pela tomada e possui duas saídas (*VCC* e *Ground*) que devem ser ligadas no controlador da caixa de som;
- Tubulação de ar: os tubos são ligados na saída da caixa de som para levar o ar até o pulmão artificial. A tubulação é toda isolada e possui aberturas para que os sensores possam ser acoplados. Uma das aberturas da tubulação possui uma válvula de pressão que permite controlar quanto ar efetivamente chega no pulmão;
- Compressor de ar: o compressor de ar se liga na tubulação para fornecer o ar que será utilizado na respiração;
- Pulmão artificial: simula um pulmão humano e deve ser conectado na saída da tubulação de ar para testes;
- Sensores de pressão e fluxo de ar: Os sensores são utilizados para obtenção dos dados da respiração fornecendo assim um melhor controle ao operador da máquina. Os sensores são utilizados para obtenção dos dados da respiração fornecendo assim um melhor controle ao operador da máquina. O sensor de fluxo de ar fica conectado à saída da tubulação e os sensores de pressão são ligados na saída da caixa e entre o pulmão e o sensor de fluxo de ar.

O funcionamento se dá da seguinte forma: o gerador de função gera uma onda senoidal que é usada pelo amplificador do alto falante para que este emita ondas sonoras. Essas ondas, juntamente com o ar fornecido pelo compressor, criam um fluxo de ar que chega até o pulmão artificial, simulando a respiração em alta frequência.

Infelizmente, não foi possível conectar o sensor de pressão que deveria ficar entre a saída do fluxo de ar e o pulmão artificial tendo em vista que o tubo utilizado para fazer a conexão não foi encontrado e as tentativas realizadas com outros tipos de tubos acabaram resultando em vazamento de ar, o que compromete o funcionamento do aparelho.

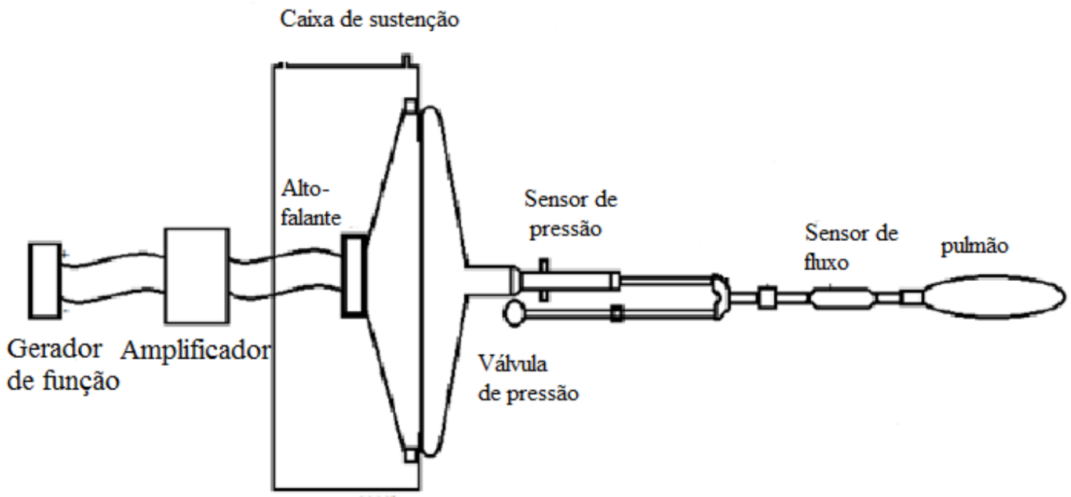

Figura 3.10: Estrutura já existente do protótipo do ventilador[29].

A Figura 3.11 mostra uma foto do gerador de funções junto do ventilador. O circuito marcado pelo número 1 é o circuito responsável por gerar o sinal e transformá-lo em um sinal analógico. O circuito do número 2 é uma fonte simétrica de 5V que alimenta os componentes do gerador e o número 3 mostra o amplificador que recebe o sinal e o repassa para o alto falante.

O sensor de fluxo pode ser visto na Figura 3.13. No número 1, o sensor conecta o pulmão artificial marcado pelo número 2 ao final da tubulação na outra ponta. A Figura 3.12 mostra o circuito do sensor de pressão na placa marcada pelo número 2 e o esfigmomanômetro usado para calibração da leitura está marcado pelo número 1. A entrada do compressor de ar pode ser vista na na marcação do número 3.

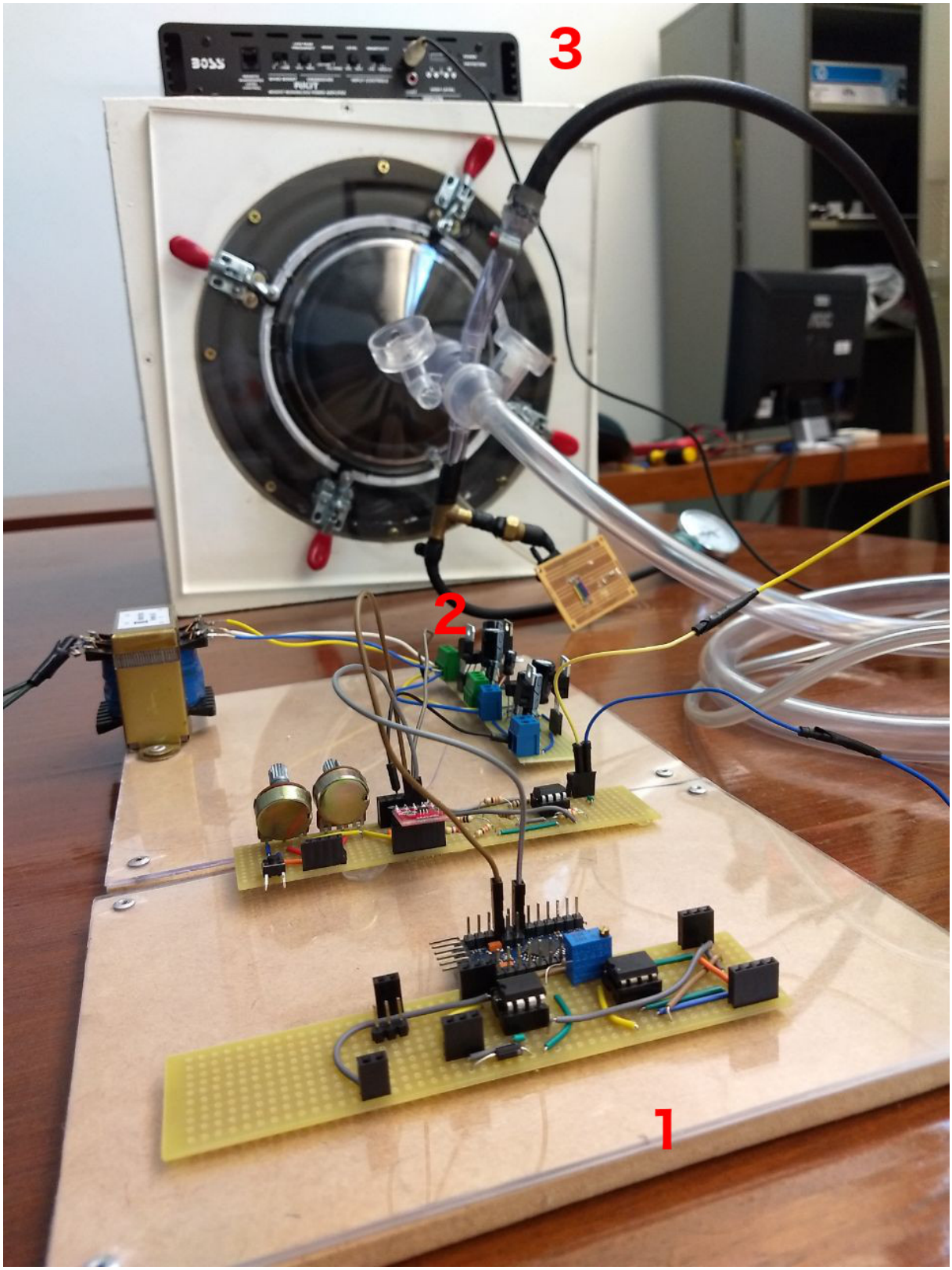

Figura 3.11: Foto do gerador de funções conectado ao ventilador.

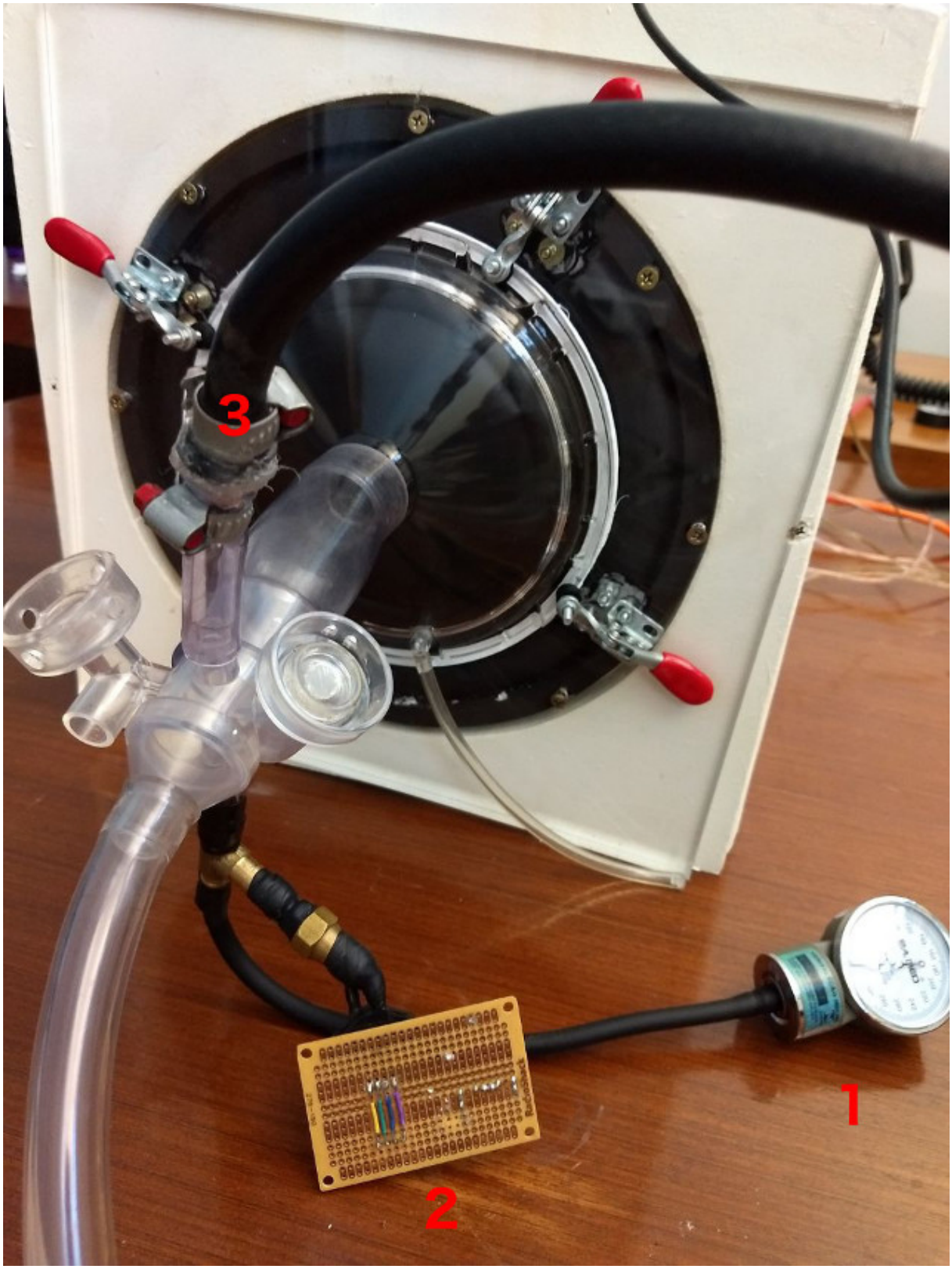

Figura 3.12: Foto do sensor de pressão conectado ao ventilador.

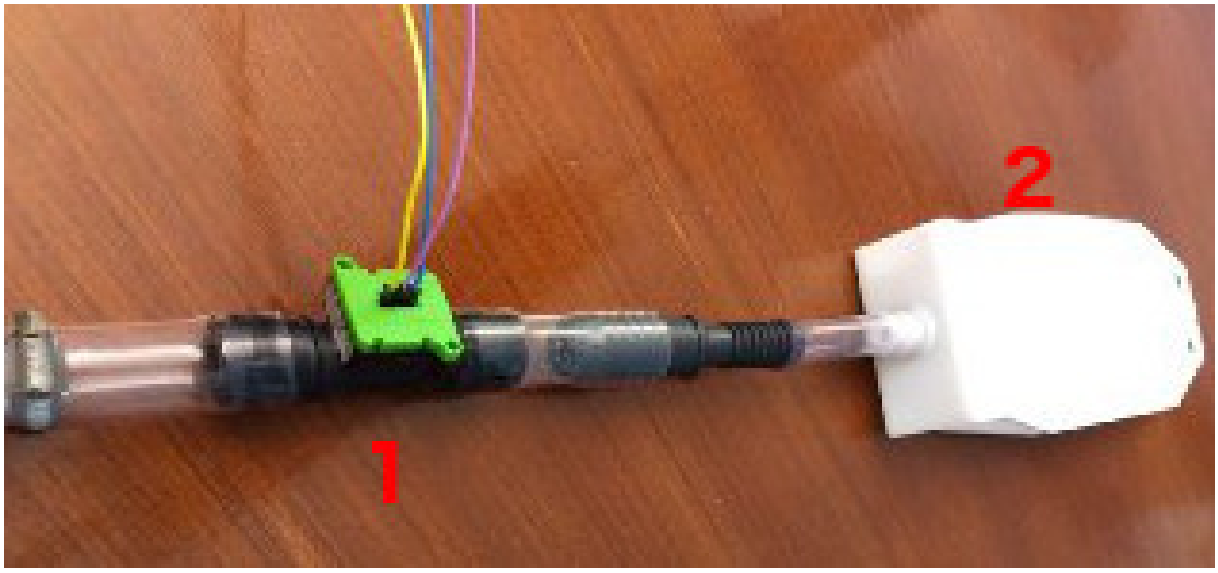

Figura 3.13: Foto do sensor de fluxo conectado ao ventilador.

Para integrar o *Raspberry Pi* e a tela com o protótipo já montado, foi preciso conectar os cabos usados para comunicação *I2C* e serial com os *Arduinos* do ventilador. O esquemático da Figura 3.14 mostra como se deu as conexões, os protocolos utilizados e a direção do tráfego de dados indicada pelo sentido das setas.

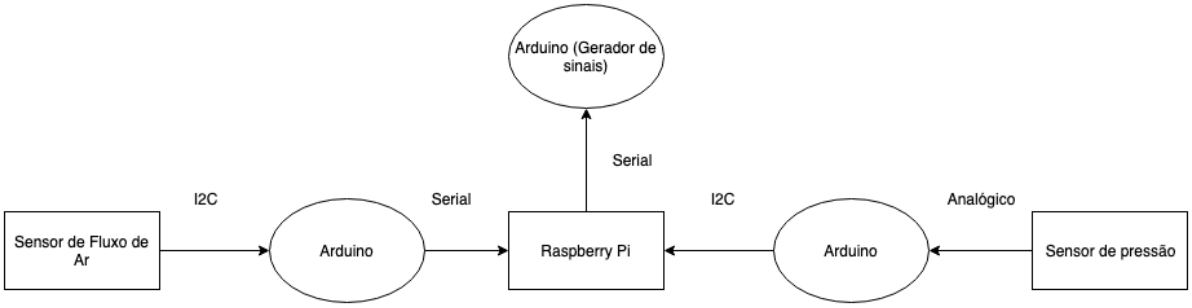

Figura 3.14: Esquematico das conexões e protocolos utilizados entre os dispositivos.

Uma foto da montagem final do ventilador pode ser vista na Figura 3.15.

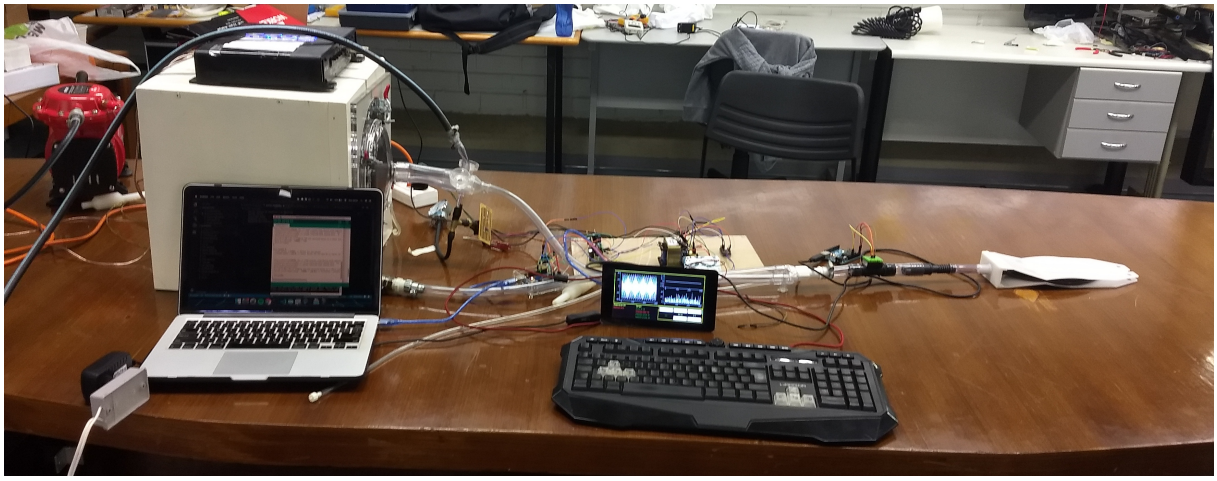

Figura 3.15: Foto da montagem final do ventilador.

4 Resultados e Discussão

Primeiramente o aparelho foi ligado em seu estado normal com os sensores conectados, com exceção de um sensor de pressão por motivos explicados na Seção 3.5. A frequência e a amplitude foram ajustadas para 3Hz e 4V e foi feita a captura da tela com as leituras.

Como é possível observar na Figura 4.1, as leituras dos sensores de fluxo de ar (esquerda) e pressão na saída da caixa (inferior direito) estão sendo feitas e mostradas corretamente, sendo que as unidades são litros por minuto (l/m) e centímetros de água (cmH2O) respectivamente. No eixo horizontal temos uma escala que vai de 0 a 300, indicando o número de amostras, onde cada 100 unidades representam aproximadamente 2 segundos. O sinal do gráfico que seria utilizado para leitura do outro sensor de pressão está constantemente em zero devido a falta deste sensor.

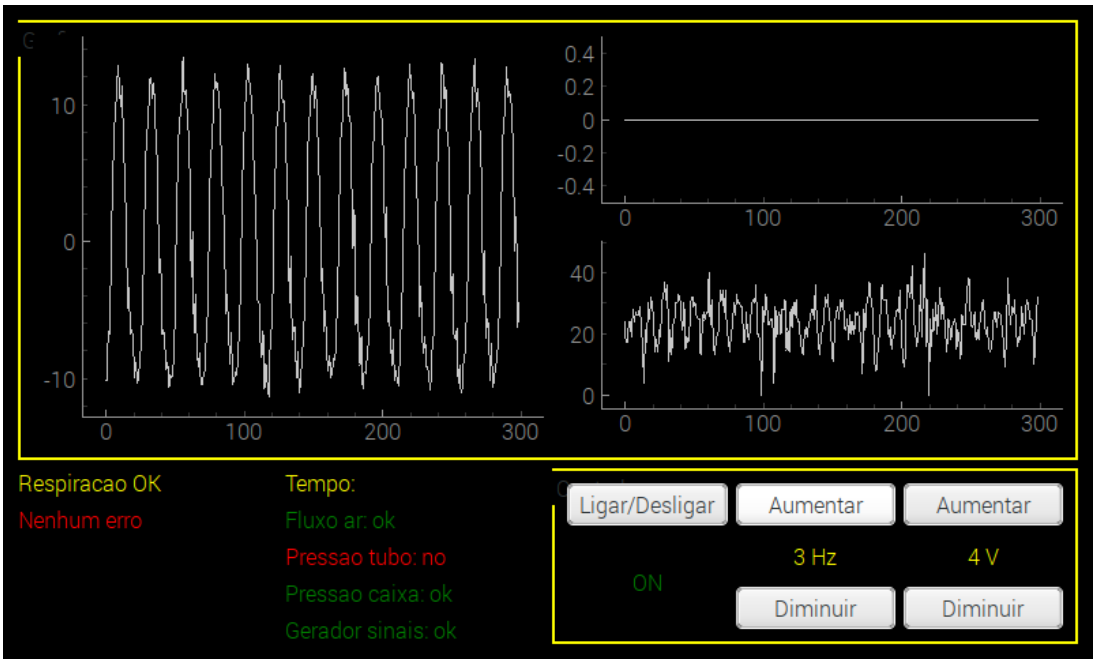

Figura 4.1: Captura de tela do ventilador funcionando com frequência de 3 Hz e amplitude de 4 V..

Com o aumento da frequência de funcionamento para valores acima de 12 Hz, as

leituras do sensor de fluxo de ar passaram a apresentar problemas. O elevado número de picos e vales juntos em um pequeno espaço acabou prejudicando a visualização. A Figura 4.2 mostra como o gráfico do fluxo se comporta com frequências altas. Uma das soluções seria diminuir dinamicamente o intervalo de valores do eixo X conforme a frequência utilizada no entanto, não foi possível implementar tal funcionalidade para os gráficos em tempo real. O tamanho da tela também impossibilitou o aumento do tamanho do gráfico.

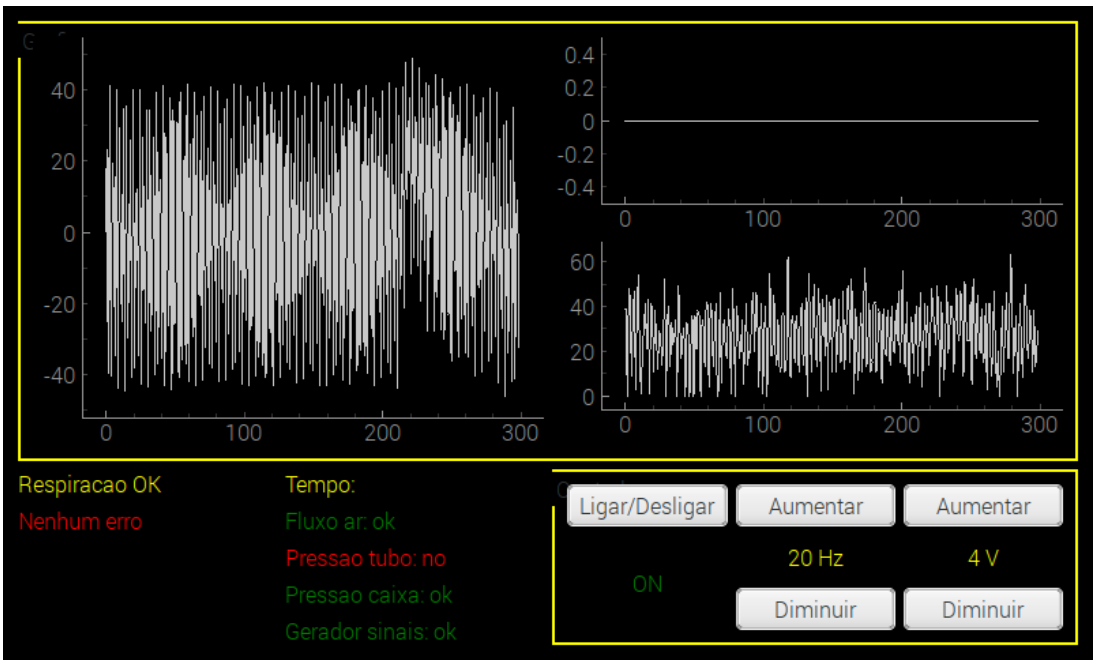

Figura 4.2: Captura de tela do ventilador funcionando com frequência de 20 Hz e amplitude de 4 V..

Para fins de comparação, utilizando a ferramenta *Serial plotter* da IDE do *Arduino*, as leituras do sensor de fluxo foram feitas diretamente, sem passar pelo *Raspberry Pi*. O resultado pode ser observado na Figura 4.3.

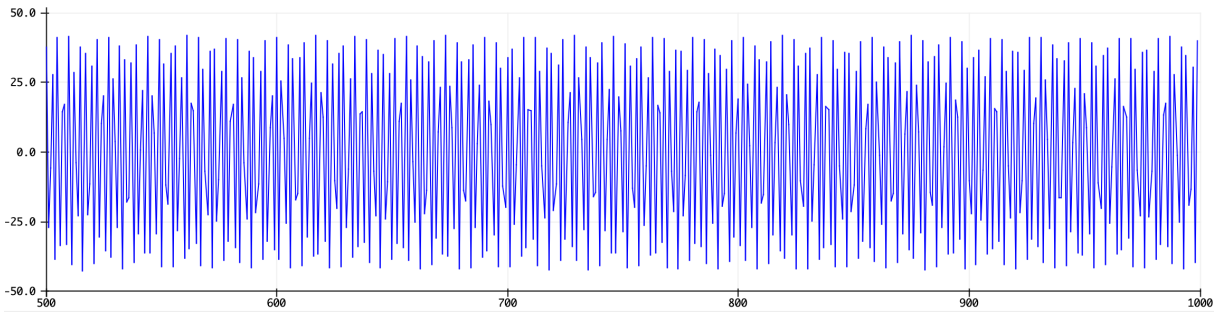

Figura 4.3: Leitura do sensor de ar feita diretamente pelo *Arduino*..

É possível ver que os gráficos das Figuras 4.2 e 4.3 se assemelham e que portanto não se trata de um erro no *software*.

O botão Ligar/Desligar do programa atua apenas no gerador de sinais, fazendo com que ele não escreva nenhum valor enquanto a máquina esteja desligada. Ao desligar o aparelho pela interface gráfica observou-se o resultado mostrado na Figura 4.4. No gráfico de fluxo (esquerda) entre as marcas de 0 e 100 amostras percebe-se a abrupta diminuição do fluxo de ar, que foi o ponto em que a máquina foi desligada. O sensor de pressão não apresentou grandes diferenças do estado ligado e desligado, isso porque o compressor de ar, que fica ligado logo em cima da saída que vai no sensor de pressão, não foi desligado já que se trata de outro sistema independente e portanto a pressão do ar não sofreu grandes alterações. Há também influência do ruído nos fios do gerador de sinais que ativam o *speaker* causando uma pequena movimentação do ar, como pode-se observar pelo sinal ruidoso no gráfico do sensor de fluxo após o desligamento do aparelho.

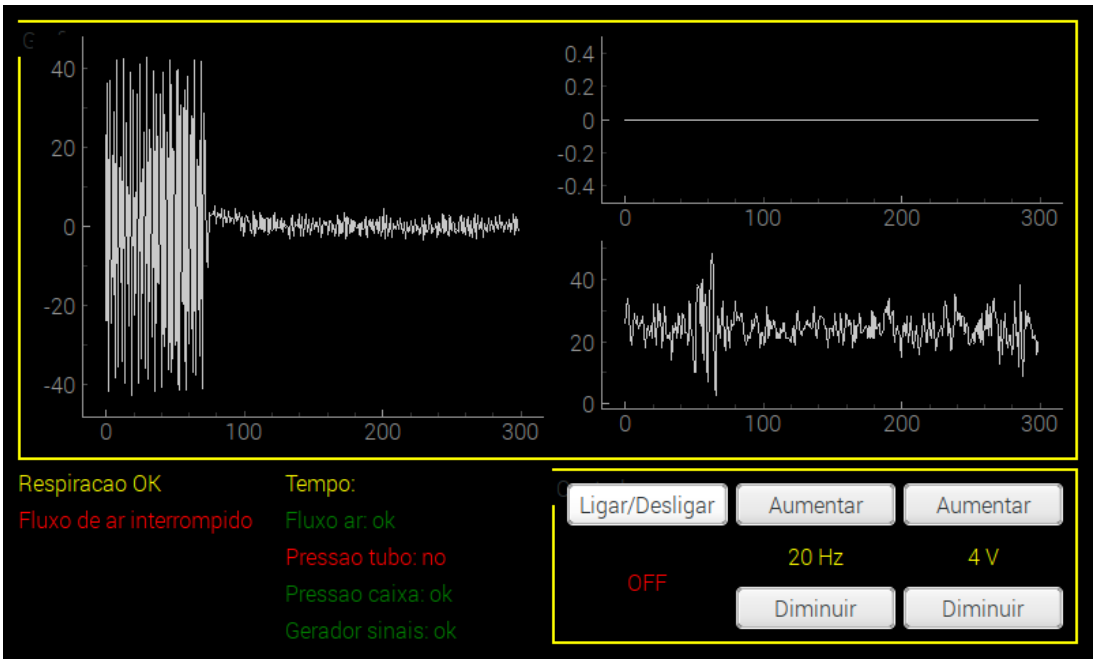

Figura 4.4: Captura de tela após o desligamento do aparelho..

O programa também verifica automaticamente quais dispositivos estão conectados durante a execução do programa. A captura de tela mostrada na Figura 4.5 foi feita após a retirada de todos os dispositivos externos. Percebe-se na parte de informações que o programa mostra todas as conexões inativas. Ao reconectar os componentes com o programa em execução, apenas as conexões que utilizam o *I2C* são atualizadas automaticamente. Isso acontece porque realizar testes de conectividade com portas seriais consome uma

quantidade considerável de tempo de CPU, o que acaba deixando todo o programa lento e atrapalhando os gráficos de tempo real.

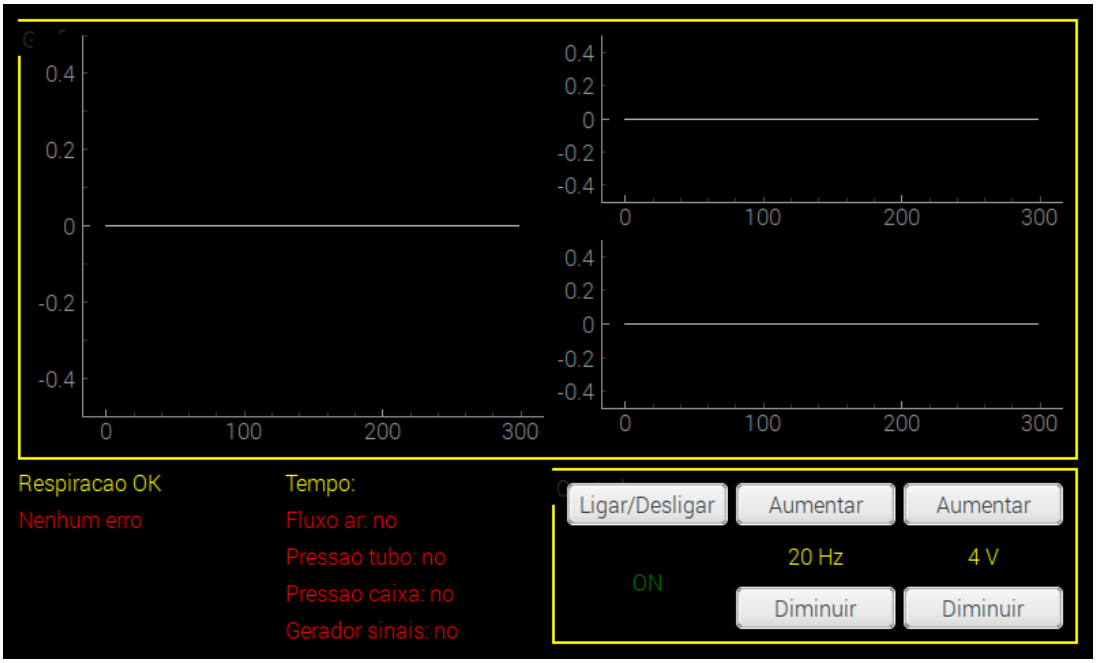

Figura 4.5: Captura de tela após a retirada de todos os dispositivos..

4.1 Medições de Fluxo

Na análise de fluxo foi observada como a mudança de frequência e amplitude afetam o fluxo e qual seria a melhor combinação desses parâmetros para que se obtenha o maior fluxo possível. Para obter os dados necessários para fazer essas medidas, utilizou-se um *script* que faz as medidas de maneira automatizada. A análise foi feita com auxílio da ferramenta *Excel*.

Para analisar o efeito da frequência no fluxo médio, foi escolhida uma amplitude fixa de 3 V e foram analisados os dados para uma faixa de frequência entre 8 e 15 Hz. Essa faixa foi escolhida pois é a mais utilizada em ventiladores deste tipo. A média do fluxo foi obtida a partir da divisão entre a soma dos valores obtidos e o número de amostras.

Dos dados mostrados na Tabela 4.1, conclui-se que o aumento da frequência quase sempre acarreta no aumento do fluxo médio, sendo que o maior fluxo aconteceu para uma frequência de 13 Hz e o menor fluxo para a frequência de 8 Hz. Mudanças na calibragem da válvula de escape de ar, nas conexões da tubulação e até mesmo no pulmão artificial utilizado podem acarretar em resultados diferentes.

Tabela 4.1: Tabela mostrando os fluxos médios para uma amplitude de 3V e frequências na faixa de 8 a 15Hz.

Frequência (Hz)	Fluxo positivo (l/min)	Fluxo negativo (l/min)	Média de fluxo (l/min)
8	11,12	11,43	11,28
9	11,91	12,19	12,05
10	11,78	12,18	11,98
11	12,43	13,34	12,89
12	13,22	13,97	13,59
13	13,78	14,07	13,92
14	13,03	14,22	13,62
15	12,92	14,16	13,54

Na Tabela 4.2, foram listadas as frequências que resultam no maior fluxo para uma dada amplitude entre os valores de 1 e 5 V. Os dados mostram que os maiores fluxos são obtidos entre as frequências de 12 e 15 Hz. Observa-se também que o aumento da amplitude também afeta o fluxo de ar.

Tabela 4.2: Tabela mostrando os maiores fluxos para diferentes amplitudes.

Amplitude (V)	Frequência (Hz)	Fluxo médio (l/min)
1	15	8,20
2	12	10,24
3	13	13,92
4	14	14,86
5	12	17,24

Nos gráficos mostrados na Figura 4.6, fica claro como os parâmetros afetam o fluxo de ar. Percebe-se que os aumentos de frequência quase sempre são acompanhados de aumentos de fluxo não muito grandes quando comparados às variações de fluxo causadas pelo aumento da amplitude.

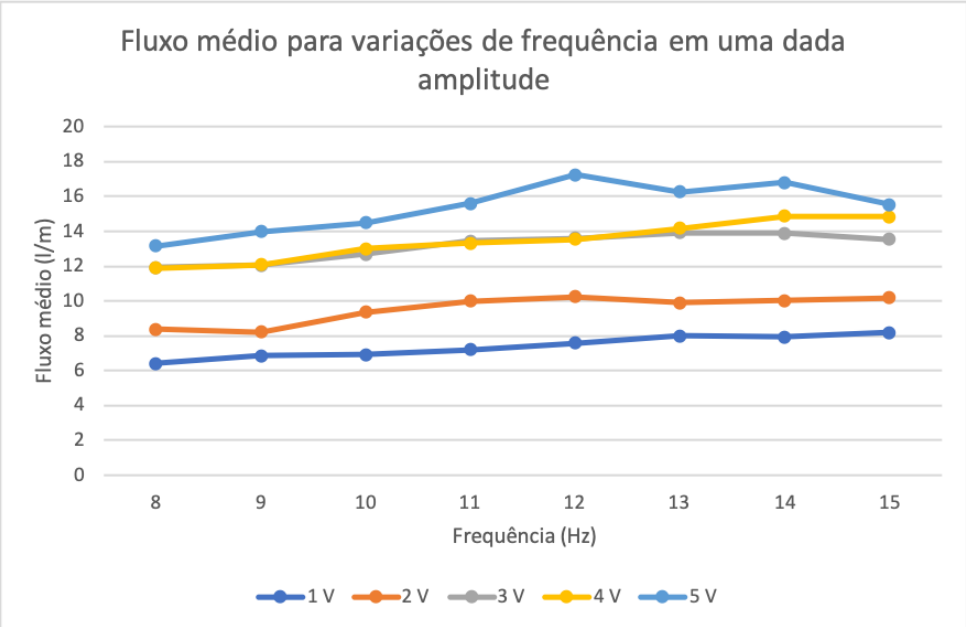

Figura 4.6: Gráficos de fluxo de ar médio para variações de frequência e amplitude..

4.2 Medições da Pressão

Primeiramente, analisou-se a relação entre a pressão e o fluxo durante o funcionamento do aparelho. Para isso, o gráfico a seguir foi feito com auxílio da biblioteca *matplotlib*. Os dados mostrados são para a leitura dos dados com a frequência em 9 Hz e a amplitude em 1 V.

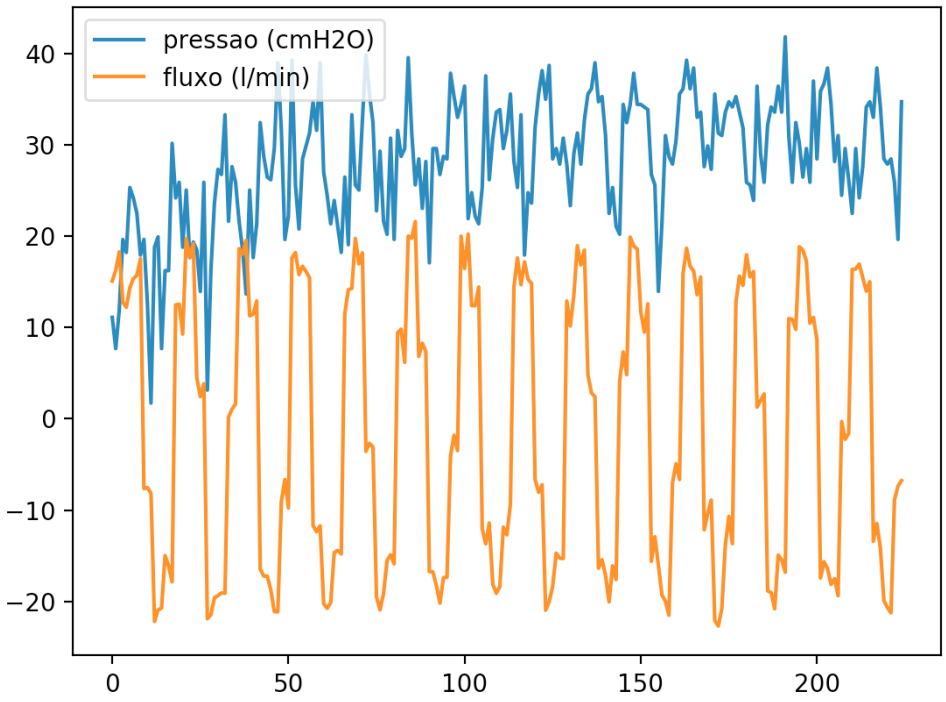

Figura 4.7: Gráfico da relação entre a pressão e o fluxo de ar..

É possível observar na Figura 4.7 que a maioria dos picos e vales dos dois gráficos se encontram nos mesmos pontos, o que era esperado. As diferenças são explicadas pelo fato de que durante a coleta de dados, os sensores de pressão e fluxo estavam sendo lidos em placas diferentes que rodavam códigos distintos o que resultou em um número diferente de amostras sendo 1702 coletadas para o primeiro e 576 para o segundo. O ajuste da diferença se deu pela utilização de 3 amostras de fluxo para cada amostra de pressão

Os valores de pressão foram medidos para as mesmas configurações de frequência e amplitude da análise de fluxo feita na Seção 4.1. Desses valores foi extraída a média de pressão em cmH2O para cada variação dos parâmetros. A Figura 4.8 mostra como a pressão na saída da caixa se comportou no experimento.

Observando os gráficos da Figura 4.8, percebe-se que não existe um padrão de alteração da pressão quando os parâmetros variam, tendo em vista que todos os valores ficaram variando na faixa de 30 e 33 cmH2O de maneira aleatória. Uma possível explicação para isso, é o fato de que a entrada de ar comprimido fica localizada logo acima de onde o sensor faz a leitura e devido à força do compressor, a pressão não se altera muito com os sinais do *speaker*.

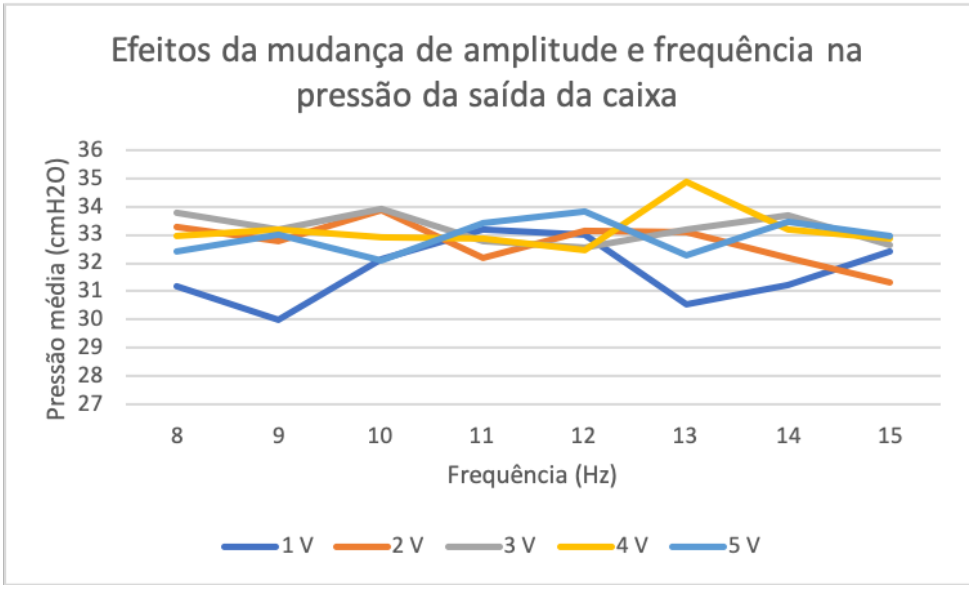

Figura 4.8: Variação da pressão média para diversos valores de amplitude e frequência..

4.3 Comentários Finais

Os testes do *software* desenvolvido mostraram que ele foi capaz de controlar eletronicamente a frequência e a amplitude de operação do ventilador. Os gráfico em tempo real funcionaram para frequências abaixo de 12Hz e apresentaram problemas de visualização para valores maiores. As informações de dispositivos conectados funcionou perfeitamente para dispositivos conectados por *I2C* mas apresentou problemas pontuais para os que utilizam do protocolo *serial*.

A análise de dados mostrou que quanto maior a amplitude maior é o fluxo de ar que chega até o pulmão artificial e que aumentos da frequência quase sempre vem acompanhados de aumento no fluxo. Esse era o resultado esperado, diferentemente dos resultados obtidos a partir dos dados do sensor de pressão que mostraram uma pressão média praticamente constante para as alterações de frequência e amplitude.

5 Conclusão e trabalhos futuros

A proposta deste trabalho de conclusão de curso foi a continuação do desenvolvimento do protótipo de um ventilador oscilatório de alta frequência com foco na visualização dos dados obtidos e na implementação do controle eletrônico.

Durante a implementação do trabalho, foi feito um estudo aprofundado sobre microcontroladores, protocolos de comunicação, visualização de dados em tempo real, respiração humana e métodos de ventilação, propiciando uma oportunidade para que os conhecimentos adquiridos durante o curso sejam aplicados em um cenário real.

Foi possível observar nos resultados do projeto que apesar de alguns contratempos, o *software* cumpriu com a proposta do trabalho. Os resultados obtidos nos sensores mostraram que de acordo com o esperado, o uso de frequências e amplitudes mais altas resultam em um maior fluxo de ar. Os resultados obtidos com a leitura do sensor de pressão, apesar de inesperados, também foram satisfatórios e a relação entre a pressão e o fluxo, mesmo com a diferença da taxa de amostragem, mostrou como se dá a dinâmica dessas medidas.

A escolha da placa *Raspberry Pi* para o sistema de processamento principal acabou mostrando algumas desvantagens que não eram esperadas no início do projeto. Se por um lado, o suporte de um sistema operacional permitiu a utilização de bibliotecas de interface gráfica que facilitaram o desenvolvimento, a lentidão causada pelo *overhead* do SO atrapalhou as funcionalidades desenvolvidas de modo que nem todas puderam ser incluídas no produto final. A escolha da tela principal também poderia ter sido melhor já que o tamanho da tela foi um fator limitante.

Durante a implementação deste projeto, as maiores dificuldades encontradas foram durante a fase de integração do sistema desenvolvido com as partes já existentes do ventilador. Diversas conexões não funcionaram de acordo com o esperado e a montagem do ventilador foi desafiadora pois algumas peças não serviam mais e precisaram ser trocadas. Felizmente, a montagem e a integração foram bem sucedidas.

Por fim, conclui-se que mesmo com alguns erros durante os experimento e a implementação, o protótipo do ventilador encontra-se em um estado muito mais avançado do que ele estava antes do início deste projeto.

5.1 Cronograma

Este trabalho foi desenvolvido ao longo de um ano nas matérias de trabalho de conclusão de curso 1 e 2. A divisão de tarefas por mês se deu da seguinte forma:

- Agosto até outubro: Leitura de trabalhos anteriores e familiarização com a estrutura já existente do protótipo.
- Outubro até novembro: Compra de materiais necessários e leitura de documentações das bibliotecas que seriam utilizadas.
- Fevereiro até abril: Desenvolvimento do *software* e escrita da introdução e do referencial teórico.
- Maio até junho: Montagem do ventilador e integração com o sistema desenvolvido neste trabalho.
- Junho: Finalização da escrita desta monografia.
- Julho: Apresentação final.

5.2 Trabalhos Futuros

Após a conclusão deste projeto, o ventilador já se encontra funcionando com controle eletrônico de amplitude e frequência além de leitores com visualização em tempo real para todos os sensores. Os próximos projetos, além de implementar novas funcionalidades, devem buscar resolver alguns dos problemas que por falta de tempo e recursos não foram solucionados neste trabalho. São eles:

- Procurar uma forma de alimentar todos os componentes do ventilador a partir da mesma fonte;
- Organizar os componentes de ventilador que atualmente ficam espalhados;
- Vedar os vazamentos de ar ainda presentes em alguns tubos e na caixa de som;
- Comprar ou produzir tubos para integrar os sensores que ficaram faltando;
- Melhorar o *hardware* ou o *software* dos dispositivos que fazem os gráficos em tempo real para que o número de funcionalidades fique menos limitado.

Por fim, seguem algumas sugestões de novas funcionalidades.

- Calibragem automática de frequência e amplitude para que o software encontre automaticamente o maior fluxo de ar possível;

- Controle eletrônico da válvula de escape de ar.

Referências

[1] Poor, Hooman: *Basics of Mechanical Ventilation*. Springer, 2018. vi, vii, 8

[2] Hasan, Ashfaq: *Understanding Mechanical Ventilation*. Springer, 2010. vi, vii, 8

[3] Hall, Guyton : *Tratado de fisiologia médica*. Elsevier, 2011. 4, 5, 7

[4] Valerie C. Scanlon, Tina Sanders: *Essentials of anatomy and Physiology*. Davis Company, 2007. 4, 5, 6, 7

[5] Henry M. Seidel, Rosalyn W. Stewart: *Mosby's Guide to Physical Examination*. Mosby, 2010. 4

[6] *Medical News Today website*. http://www.medicalnewstoday.com. 6

[7] Guy W Soo Hoo, MD: *Noninvasive ventilation*. emedicine.medscape.com, 2018. 8

[8] SHERRY E. COURTNEY, M.D., DAVID J. DURAND M.D. JEANETTE M. ASSELIN R.R.T. M.S. MARK L. HUDAK M.D. JUDY L. ASCHNER M.D. e M.D CRAIG T. SHOEMAKER: *High-frequency oscillatory ventilation versus conventional mechanical ventilation for very-low-birth-weight infants*. N Engl J Med, Vol. 347, No. 9, 2002. 8

[9] Dreyfuss D, Saumon G: *State of the art: ventilator-induced lung injury; lessons from experimental studies*. Am J Respir Crit Care Med, 1998. 9

[10] Jerry A. Krishnan, MD; e MD Roy G. Brower: *High-frequency ventilation for acute lung injury and ards*. Chest Journal, 2000. 9

[11] *3100A® High frequency oscillatory ventilator Operator's manual*. https://www.vyaire.com/documents/guides/user-guides/$RC_3100A - HFOV_UG_EN.pdf$. 9

[12] *Vyaire website*. http://www.vyaire.com. 10

[13] Jonathan M. Klein, MD: *Management strategies with high frequency oscillatory ventilation (hfov) in neonates using the sensormedics 3100a high frequency oscillatory ventilator*. Iowa Neonatology Handbook, 2001. 10

[14] Axelson, Jan: *Serial Port Complete: COM Ports, USB Virtual Com Ports, and Ports for Embedded Systems (Complete Guides)*. Lakeview Research, 2007. 10

[15] Nelson, Mark: *Serial Communications Developer's Guide*. Wiley, 1999. 10

[16] Cowley, John: *Communications and Networking: An Introduction*. Springer, 2013. 10

[17] Jean-Marc Irazabal, Steve Blozis: *I2C Manual*. https://www.nxp.com/docs/en/application-note/AN10216.pdf. 11

[18] *Robot electronics website*. http://www.robot-electronics.co.uk. 11

[19] *Embetronicx website*. http://www.embetronicx.com. 12

[20] *Python Software Foundation*. https://www.python.org/. 12

[21] *Arduino Website*. https://www.arduino.cc/en/Guide. 12

[22] *Raspberry Pi Website*. https://www.raspberrypi.org/. 13, 15, 16

[23] Guardian, The: *Raspberry pi becomes best selling british computer*. 13

[24] *Raspbian Website*. https://www.raspbian.org/. 13

[25] *Documentation for PyQtGraph*. http://www.pyqtgraph.org/documentation/. 13

[26] *PyQtGraph Website*. http://www.pyqtgraph.org/. 13

[27] Sommervile, Ian: *Engenharia de software*. Pearson, 2012. 18, 19

[28] *MockFlow Website*. https://www.mockflow.com/app/Wireframe. 21

[29] Ferreira, Jefferson Adiniz Borges: *Projeto de ventilador oscilatorio pulmonar de alta frequencia*. Trabalho de Conclusão de Curso - UnB/FGA, 2016. 30