Monitoramento da Quantidade de Carbono no Ar - Carby

Aline Rosa dos Santos Rocha[1], 16/0023076, Sofia Consolmagno Fontes[2], 16/0018234
[1,2]Programa de Engenharia Eletrônica, Faculdade Gama - Universidade de Brasília, Brasil

***Resumo*—O artigo em questão descreve o desenvolvimento de um módulo eletrônico em conjunto com a *Raspberry Pi*, capaz de monitorar a quantidade de gás carbônico (CO_2) no ar, o qual será elaborado para a disciplina de Sistemas Operacionais Embarcados. A proposta surgiu a partir da dificuldade de clínicas médicas e odontológicas verificarem as trocas de ar realizadas na sala de espera durante o período de pandemia do vírus SARS-CoV-2. Por fim, para avaliar o projeto será realizado um protótipo, o qual passará por testes de viabilidade em um consultório odológico de Brasília.**

***Index Terms*—COVID-19, Qualidade do Ar Interior, CO_2, *Raspberry Pi* e Sistema Embarcado**

I. Introdução

Em dezembro de 2019 na China foi diagnosticado o primeiro paciente infectado pelo vírus SARS-CoV-2, causador da doença COVID-19, a qual atualmente constitui uma Emergência de Saúde Pública de Importância Internacional [1]. Por conseguinte, a pandemia se espalhou rapidamente, sendo o primeiro caso confirmado no Brasil no dia 25 de fevereiro de 2020 [2].

A transmissão do vírus se dá pela vias aéreas, por meio de gotículas respiratórias expelidas durante a fala, tosse ou espirros. Dessa forma, é fundamental seguir as principais medidas orientadas pelas autoridades sanitárias, as quais são: isolamento físico ou domiciliar, assepsia, cuidados individuais, utilização de máscaras e respiradores do tipo N95 e em ambientes fechados deve-se realizar a transferência e substituição do ar possivelmente contaminado do interior pelo ar exterior [3].

Consequentemente, a maior parte da transmissão do vírus SARS-CoV-2 ocorre em ambientes fechados, principalmente pela inalação de partículas transportadas pelo ar que contêm o coronavírus. Além do mais, o dióxido de carbono (CO_2) existe naturalmente na atmosfera, é uma molécula produzida pelo corpo humano através da respiração [4]. Logo, para ambientes fechados os níveis de CO_2 podem ser utilizados para aferir se o ambiente está sendo preenchido por exalações potencialmente infecciosas [5].

Normalmente, o nível de CO_2 no ambiente é estável e tem-se uma variação desse nível a partir da exalação humana, assim é possível estimar se há uma quantidade suficiente de ar fresco entrando no espaço [6]. Por conseguinte, em ambientes externos os níveis de CO_2 são, em média, de menos de 400 partes por milhão (ppm), e em um ambiente interno bem ventilado terá mais de 400 ppm, dessa forma, um número maior que 800 ppm indica que o ambiente precisaria de mais ventilação [7]. Segundo a Resolução da ANVISA nº 9/2003, o valor máximo recomendável para um ambientes climatizados de uso público e coletivo é de 1000 ppm [8].

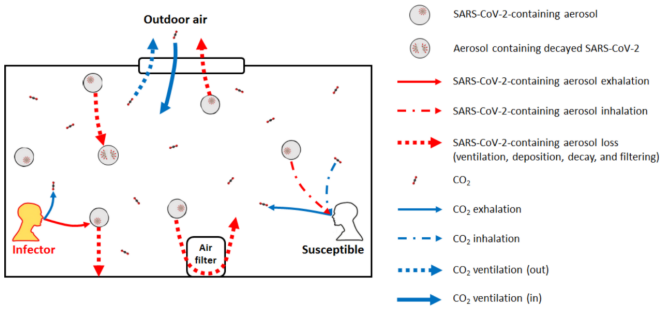

Fig. 1. Esquema da ilustração da expiração, inalação e outros processos de perda de SARS-CoV-2 contendo aerossóis em um ambiente interno. Fonte: Exhaled CO2 as COVID-19 infection risk proxy for different indoor environments and activities [5]

II. Justificativa

Após um ano de pandemia e sem medidas de isolamento social efetivas, é inevitável que haja circulação de pessoas e possibilidades de aglomeração, além de que existem pessoas assintomáticas não diagnosticados mas que continuam dispersando o vírus. Com tais características, os ambientes fechados como clínicas e hospitais que apresentam uma grande fluxo de pessoas diariamente, sofrem com os altos riscos de contaminação tanto para os pacientes quanto para a equipe de saúde [9]. Assim, clínicas odontológicas podem provocar a infecção cruzada devido ao uso de instrumentos que produzem aerossóis, gotículas e secreções de saliva e sangue [10].

Dessa forma, com a finalidade de diminuir a dispersão da COVID-19 este projeto consiste no desenvolvimento de um módulo eletrônico capaz de detectar quantas trocas de ar deverão ser realizadas por hora em uma clínica odontológica, para obter uma maior segurança do profissional da saúde e dos pacientes. As variáveis que compõem as realizações das trocas, são: quantidade de pessoas, medidas do espaço físico, concentração de CO_2, temperatura e umidade.

É vantajoso que haja esse monitoramento em clínicas tendo em vista que em certos períodos do dia o ambiente pode ter maior concentração de pessoas e assim, apenas a ventilação por meio de exaustores, ventiladores e ar-condicionado não sejam suficientes para manter a concentração de CO2 está em níveis seguros. Bem como que nas estações do ano mais frias é comum manter menos ventilação em ambientes fechados, o que favorece no aumento da concentração de CO_2.

III. Objetivo

Proporcionar aos funcionários e clientes de uma clínica odontológica segurança contra a propagação do vírus dentro do ambiente fechado. Para isso, tem-se por objetivo projetar e prototipar um equipamento que atue na sala de espera de uma clínica odontológica, o qual pode ser acionado pelo comando de voz do usuário para monitorar a qualidade do ar por meio de sensores de CO_2, temperatura e umidade. Além do mais, esse processamento deve ser realizado pela Single-Board Computer (SBC) Raspberry Pi 3B, e será possível ao usuário observar um gráfico da concentração de CO_2 nas últimas 24 horas e deverá emitir alertas por comando de voz quando a concentração de CO_2 for superior a 1000 ppm e, assim, houver a necessidade das trocas de ar.

IV. Metodologia

Com a finalidade de de acompanhar e analisar o desenvolvimento do projeto dividiu-se os marcos em quatro pontos de controle, conforme descritos abaixo:

- **PC1:** proposta do projeto (justificativa, objetivos, requisitos, benefícios, revisão bibliográfica);
- **PC2:** protótipo funcional do projeto, utilizando as ferramentas mais básicas da placa de desenvolvimento, bibliotecas prontas etc;
- **PC3:** refinamento do protótipo, acrescentando recursos básicos de sistema (múltiplos processos e threads, pipes, sinais, semáforos, MUTEX etc.);
- **PC4:** refinamento do protótipo, acrescentando recursos de Linux em tempo real.

Além do mais, para facilitar o desenvolvimento do protótipo, o projeto será dividido em quatro áreas de trabalho: módulos de aquisição, controle, alimentação e estrutura, e Software.

A. Módulo de aquisição

A partir do módulo de aquisição que é composto por sensores de CO_2, temperatura e umidade é possível obter os dados do ambiente fechado e retornar para o servidor.

B. Controle

A área de controle será o foco principal do projeto e contará com a *Raspberry Pi 3B* para realizar toda comunicação entre os módulos e o usuário a partir da tela, do microfone e da caixa de som.

C. Alimentação e Estrutura

Esta área é responsável pela elaboração do circuito de alimentação do protótipo. Além do mais, outro enfoque dessa área é o correto posicionamento dos sensores na estrutura para possibilitar uma melhor medição. Assim, o protótipo será construído visando uma interface amigável e intuitiva para o usuário.

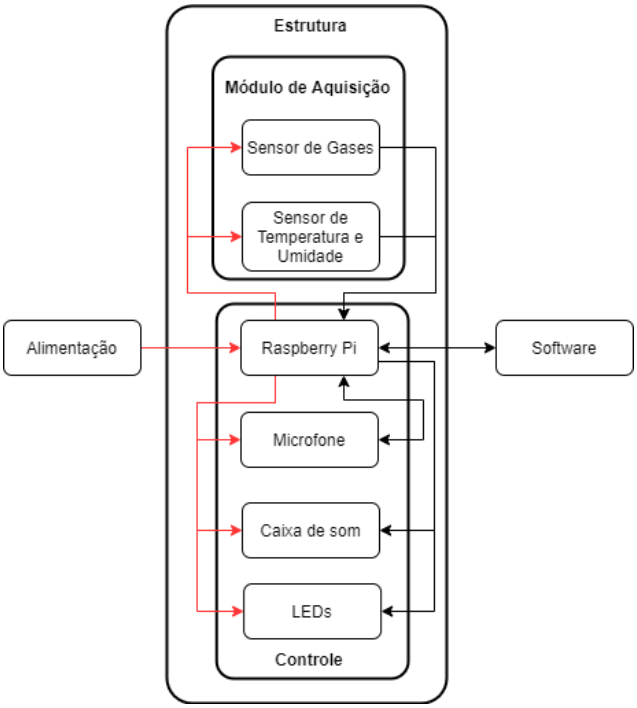

Fig. 2. Divisão das áreas de trabalho do protótipo

D. Software

O software indica ao usuário por meio de avisos quando é necessário realizar as troca de ar do ambiente e, ainda, será possível observar um gráfico da concentração de CO_2 nas últimas 24 horas.

V. Requisitos

Dado que o projeto será desenvolvido para um fim específico e com um público alvo bem definido, os requisitos devem ser definidos de forma clara e objetiva para que o módulo eletrônico supra e alcance as condições e as capacidades em que ele foi projetado.

A. Requisitos de Materiais e Custos

O projeto deve ser viável economicamente para o escopo da disciplina e restrições da universidade. Assim, uma análise de custos mais detalhada será feita em fases mais avançadas do projeto.

B. Requisitos Técnicos

1) **Hardware**
O hardware deverá adquirir os dados de temperatura, umidade e concentração de CO_2 corretamente, além de que é necessário conter pelo menos um conversor A/D para que haja uma entrada digital vinda do sensor de gás analógico para a *Raspberry*. O projeto do hardware deverá ser acessível para o usuário, construído visando uma interface amigável e intuitiva, mas que também forneça a ele ferramentas e funcionalidades que supram os objetivos que o sistema se propõe. O protótipo resultante do projeto deve ser robusto, portátil e funcional.

2) **Software**
Os comandos de voz de entrada, interpretação e os comandos de voz de saída serão processados com o auxílio do *Google Assistent SDK* com a configuração da biblioteca de voz pelo *ActionPackage*. O software disponibilizará via web, através do comando *curl*, um gráfico com o nível da concentração de $CO2$, a umidade e a temperatura nas últimas 24 horas, bem como a média dessas grandezas, tutorial de como usar o módulo Carby e dúvidas frequentes. Sendo assim, é fundamental a integração entre a *Raspberry Pi* e o armazenamento dos dados.

VI. Benefícios

O projeto se mostra importante e favorável na travessia deste momento de Emergência de Saúde Pública de Importância Internacional, uma vez que, irá beneficiar diretamente secretárias, dentistas, técnicos, pacientes e demais pessoas que frequentam a clínica odontológica, além de que, por meio do monitoramento é possível garantir uma maior segurança. Outrossim, também beneficia indiretamente a sociedade, já que tem-se a produção e conhecimento científico e experimental do monitoramento de vírus, principalmente o SARS-CoV-2, em ambientes fechados.

O grande diferencial do projeto proposto e pensando no conforto e segurança dos usuários, a ativação por comando de voz possibilita que o dentista, por exemplo, que for ativar o início da verificação da qualidade do ar do ambiente não precise parar sua atividade em desenvolvimento e tocar no dispositivo.

Além disso, há um compromisso custo $versus$ benefício. O protótipo completo é estimado no valor de $R\$490,00$, sendo que o microcomputador é o que adiciona maior custo. Com uma maior liberdade de custos, podem ser adicionadas mais funcionalidades que interessem aos usuários como verificação da quantidade de pessoas no ambiente por meio de sensores de presenças, automação da ventilação do ambiente ou adicionar módulos ao protótipo para que seja verificada a qualidade do ar em diferentes ambientes por meio da recepção dos dados através de comunicação Wi-Fi pelo microcomputador, operando como um servidor.

VII. Revisão Bibliográfica

É de extrema importância conhecer os produtos já existentes no mercado para avaliar os pontos fortes e fracos, bem como as funcionalidade existentes. Consequentemente, essa análise pode ser utilizada como referencia para criar estrategias para o desenvolvimento do próprio produto.

Assim, após uma pesquisa de mercado encontrou-se alguns detectores de gás carbônico para ambientes internos que custam na faixa de R$ 800,00, os quais apresentavam na tela os dados de temperatura e umidade, concentração de CO_2, data e hora em tempo real, gráfico de tendência, configuração de alarme, registro de dados de medição de intervalo de tempo, possui bateria de lítio recarregável por meio do USB externo.

Fig. 3. Detector de Gás Carbônico Portátil

O periódico *Sustainability* publicou um artigo em 2020 de um sistema de monitoramento multi-paramétrico e simultâneo da qualidade do ar. Sendo assim, esse sistema monitora a maioria dos gases poluentes, o que é crítico e essencial, além de que os desafios enfrentados por equipamentos convencionais existentes na medição de múltiplas concentrações de poluentes em tempo real incluem alto custo, capacidade de implantação limitada e detectabilidade de apenas alguns poluentes [11]. Ele apresenta, então, um sistema módulo sensor abrangente de monitoramento da qualidade do ar interno usando um *Raspberry Pi* de baixo custo. O sistema personalizado mede 10 ambientes internos, condições de temperatura, umidade relativa, material particulado , incluindo poluentes: NO_2, SO_2, CO_2, O_3 e compostos orgânicos voláteis.

Ainda, a *startup* Omni-eletrônica anunciou em 2020 um sistema de monitoramento da presença do coronavírus em ambientes internos, desenvolvido a partir da parceria com o Hospital das Clínicas (HC) da Faculdade de Medicina da Universidade de São Paulo e apoiado pelo Programa FAPESP. A empresa oferece o SPIRI, serviço de monitoramento da qualidade do ar, por meio de uma assinatura. Um aparelho instalado no local com vários sensores integrados, os quais enviam as informações para a central, que gera laudos on-line em tempo real. Além do mais, os técnicos podem instruir o cliente sobre a melhor forma de aumentar a circulação do ar quando ela não está adequada [12].

VIII. Desenvolvimento

A. Descrição de Hadware

Este projeto fará uso dos seguintes componentes para a solução de Hardware:

- Raspberry Pi 3 Model B+ com cartão micro SD. **Preço:** $R\$300,00$;
- Sensor de gases MQ-135. **Preço:** $R\$19,90$;
- Sensor de temperatura e umidade DHT11. **Preço:** $R\$12,90$;

- Conversor Analógico para Digital MCP3008. **Preço:** $R\$34,90$;
- Caixa acrílica.
- Adaptador de áudio USB plugável com 3,5 mm. **Preço:** $R\$20,00$;
- Dispositivo de entrada: Microfone. **Preço:** $R\$14,50$;
- Dispositivo de saída: Caixa de som com conector USB ou P2 3.5 mm. **Preço:** $R\$25,90$;

*1) **Conexão Módulo de Aquisição - Raspberry**:* A conexão do Módulo de Aquisição com a *Raspberry* é feita com o uso de uma protoboard. O sensor de umidade e temperatura, DHT11, é um sensor digital e a *Raspberry* faz sua leitura de forma direta. Já o sensor de gases MQ-135, é um sensor analógico e, como a *Raspberry* não tem pinos de entrada analógica, faz-se necessário o uso de um conversor AD (MCP3008). Além disso, o sensor MQ-135 é resistivo e precisa de calibração. A figura 4 mostra o diagrama de blocos do módulo de aquisição.

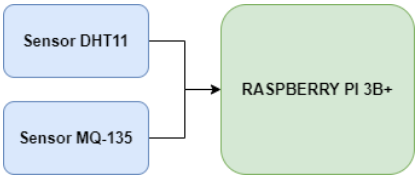

Fig. 4. Diagrama de blocos do Módulo de Aquisição.

Para a calibração do MQ-135, a figura 5 mostra a característica típica de sensibilidade do sensor para os gases de amônia, dióxido de carbono, benzeno, óxido nítrico, fumaça e álcool. Sendo assim, usa-se como parâmetro: temperatura de 20°, umidade de 65% e uma resistência de $20K\Omega$ [13].

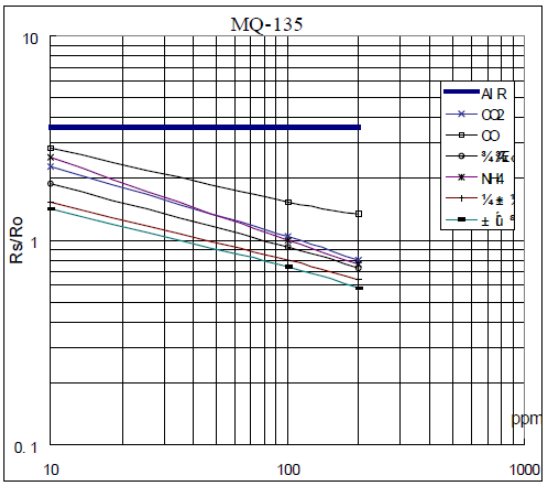

Fig. 5. Características de sensibilidade do sensor de gases MQ-135

Fundamentando no gráfico da figura 5, para realizar a calibração e da leitura do sensor é necessário marcar dois pontos da linha de CO_2, aplicar o log e posteriormente realizar o cálculo do coeficiente angular da reta [14], conforme a equação 1, com os pontos $x_1 = 100$ e $x_2 = 200$:

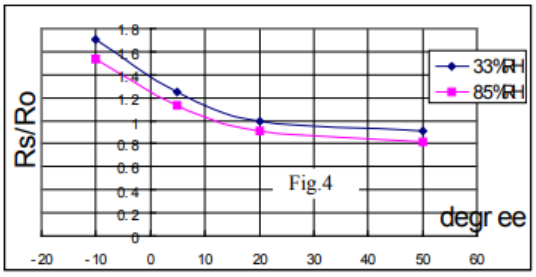

Fig. 6. Típica dependência do sensor MQ-135 da temperatura e umidade.

$$\frac{Log_{10}(0.8) - log_{10}(1.01)}{log_{10}(200) - log_{10}(100)} = -0.336 \quad (1)$$

Além do mais, como no gráfico de referência usa-se um resistor de $20K\Omega$ e no *datsheet* é também sugerido, esse valor de resistência foi utilizado no potenciômetro do módulo do sensor.

Para a correta calibração foi necessário deixar o sensor funcionado de 12h a 24h interruptas para realizar o "Burn-it" (tempo de queima) ou preheat (tempo de aquecimento). Outrossim, conforme é sugerido no datasheet deve-se determinar o R_o a partir da concentração de 100 ppm de NH_3. Entretanto, como a dupla não possuía esse gás a determinação do R_o foi realizada a partir da medição de ppm de CO_2 em um ambiente aberto até que a concentração se aproximasse de 400 ppm.

O esquemático das conexões do Módulo de Aquisição é apresentado na figura 7. É possível observar a ligação dos sensores de umidade e gases à Raspberry. O sensor DHT11 é alimentado com 5V e ligado ao GPIO 4 para transmissão dos dados de leitura. O sensor MQ-135 também é alimentado com 5V, sua saída de dados entra no canal 0 do conversor AD e é enviado para os pinos de comunicação SPI da Raspberry.

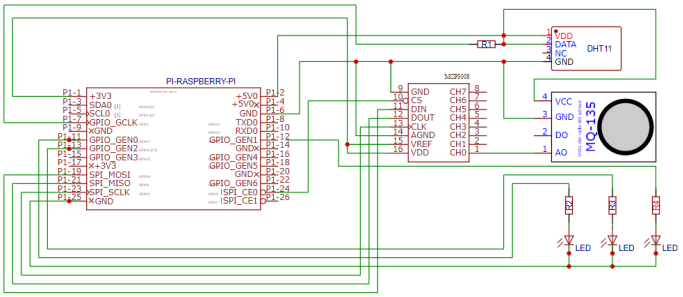

Fig. 7. Esquemático Módulo de Aquisição e Leds

*2) **Conexão Módulo de Controle - Raspberry**:* O Módulo de Controle se refere aos periféricos que são controlador pela Raspberry. Quando a Raspberry obtiver os dados de concentração de CO2 do sensor MQ135 e tiver a relação de concentração Boa - Média - Ruim, os LEDs serão controlados para que o verde acenda indicando concentração Boa, o amarelo acenda indicando concentração Média e o

vermelho acenda indicando concentração Ruim. O microfone envia comandos que serão usados com o assistente de voz que, por sua vez, os responde através da caixa de som. O diagrama de blocos da figura 8 ilustra tal funcionamento.

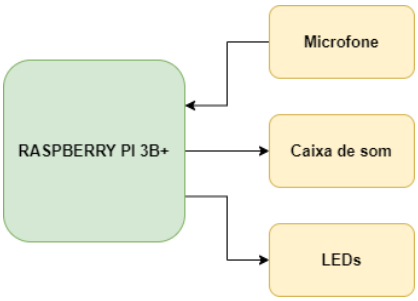

Fig. 8. Diagrama de blocos do Módulo de Controle.

Os LEDs são ligados aos pinos GPIO 17, 18 e 23. Como a entrada do microfone é analógica, foi preciso adicionar um adaptador USB de áudio em que o microfone foi plugado. A caixa de som é alimentada em um entrada USB e sua entrada P2 na saída P2 da Rasp.

B. Descrição de Software

À nível de software, a programação da *Raspberry* foi para que primeiramente os dados de umidade, temperatura e concentração de CO2 fossem adquiridos. É verificado se eles foram de fato adquiridos e, a partir disso, são enviados para uma tabela no *Google Sheets*. Dependendo da concentração, um LED verde, amarelo ou vermelho é aceso. A tabela no *Google Sheets* é compartilhada com o *Google Assistant* para comunicação com o usuário e com o *Excel*. No Excel uma interface é apresentada ao usuário com gráficos dos dados coletados nas últimas 24 horas, bem como a média deles no mesmo período. O fluxograma da figura 9 ilustra o funcionamento do sistema.

1) ***Comunicação sensor de umidade e temperatura com a Raspberry****:* Para a leitura dos dados do sensor DHT11 foi desenvolvido um script em *C++* que usa a biblioteca *pigpio* para setar o pino GPIO4 e fazer a leitura digital. Para facilitar na aquisição dos dados foram divididos em bits mais e menos significativos tanto para os dados de temperatura quanto para umidade. Posteriormente esses bits são agrupados e separados por ”,” para serem printados na tela, enviados para o arquivo *database* que é *.csv* e para a planilha na aplicação *Google Sheets*.

2) ***Comunicação sensor de gases com a Raspberry****:* Como a Raspberry não faz leitura de sensores analógicos, o conversor AD MCP3008 de oito canais é por onde os dados do sensor MQ-135 chegam para o processamento. Foi julgado como não necessário ter dois sensores de gases, já que ambos estariam em um mesmo módulo e não haveria variação de concentração suficiente que justificasse o uso. Usando a biblioteca *pigpio* para comunicação SPI, é feito um script em C++ que lê os dados do conversor, faz a calibração do sensor e calcula a curva da figura 5 com base na calibração realizada do R_0 e

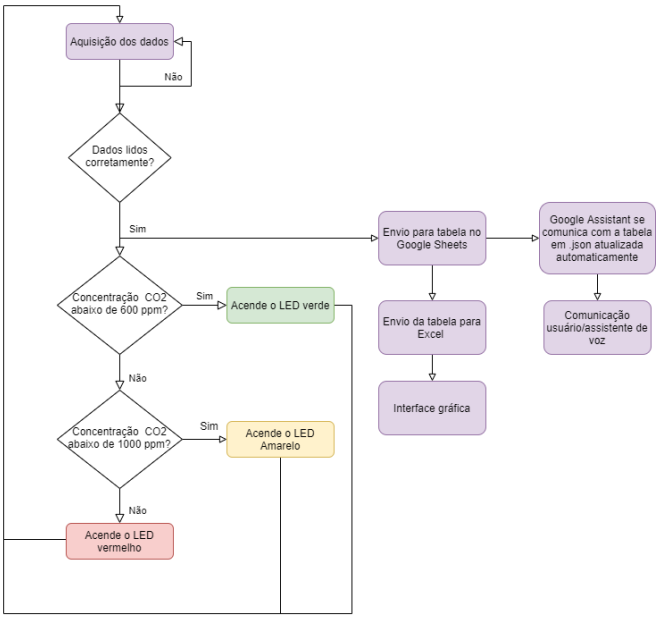

Fig. 9. Fluxograma do funcionamento do sistema.

dos dados adquiridos por meio da equação 1. Sendo assim, retorna a concentração em partes por milhão do gás CO_2.

3) ***Integração sensor de gases e sensor temperatura e umidade com a Raspberry****:* As leituras individuais dos sensores foram testadas e mostraram correto funcionamento. Para a integração dessas leituras, em um arquivo principal *.cpp* ambos os sensores são lidos. Isso permite que os dados sejam enviados juntos para o arquivo *database* que é *.csv* e para a mesma planilha na aplicação *Google Sheets* através do comando *curl*.

Quando a concentração de CO_2 está até 600 ppm, um LED verde é aceso indicando que a concentração está em um nível baixo. Entre 600 ppm e 1000 ppm um LED amarelo é aceso indicando que a concentração está em um nível médio e que se deve ter atenção. Acima de 1000 ppm, um LED vermelho é acesso indicando ao usuário a necessidade de se fazer trocas de ar no ambiente. Enquanto a troca de ar não tiver sido concluída e a concentração não estiver abaixo de 1000 ppm, o LED vermelho não desliga. As novas leituras do sensor de gases atualizam o liga/desliga dos LEDs e o usuário pode observar se a troca de ar foi suficiente para reduzir o nível de CO_2 no ambiente.

4) ***Gráfico dos dados de monitoramento - Google Sheets****:* Nos respectivos scripts do sensor de concentração de CO_2 e do sensor de temperatura e umidade, são enviados os dados obtidos para um formulário do *Google*, com respectivos campos de temperatura e umidade, concentração de CO_2 e qualidade do ar. Inicialmente, tinha-se como fundamento enviar todos os dados adquiridos pelos sensores para um arquivo *.csv* e posteriormente a partir desses dados enviar para o *Google Sheets*. A requisição é automatizada com o comando *curl*, entretanto foi perceptível que juntando os dois arquivos retirava-se o problema de sincronização, então não

foi necessário a utilização de um arquivo *.csv* para enviar os dados.

Os dados enviados para os formulários são armazenados em tabela no *Google Sheets* e são enviados para o *Excel*, uma vez que, este possui mais ferramentas e facilita a integração dos dados para uma melhor interface para os usuários. Para esse processo os dados são refinados e separa-se a coluna de data e hora e os dados adquiridos são formatados como números por meio da ferramenta *Power Query*.

Dentro da aplicação do *Excel* é montada uma apresentação com gráficos dos dados de umidade, temperatura e concentração de gás CO_2, bem como a média desses dados. Na mesma interface é apresentado ao usuário um espaço com os passos para habilitar seu módulo Carby e dúvidas frequentes.

5) ***Assistente de voz - Google Assistant SDK****:* A empresa *Google* oferece gratuitamente um serviço de assistente de voz integrado com a internet e a Raspberry para desenvolvedores que não forem usar o serviço comercialmente. Tal serviço também inclui a possibilidade de adicionar funcionalidades para aplicações específicas. Como se deseja que o usuário do dispositivo Carby seja capaz de, por meio do comando de voz, habilitar a leitura dos sensores, ouvir a concentração de CO2, a umidade e a temperatura, ser alertado quando for necessário fazer a troca de ar do ambiente e a concentração de CO_2 estiver em níveis adequados, essas funcionalidades poderão ser adicionadas com o Assistente da Google através do *Actions on Google Console*.

A comunicação entre o usuário, o Google Assistant e o Google Sheets é estabelecida por ações de conversação customizadas que seguem o fluxograma da figura 10. Para dada entrada vinda do usuário, o assistente da Google faz a combinação do parâmetro de entrada com os comandos customizados e, então, é chamada a ação correspondente. A ação faz a conexão com o serviço Google Sheets e obtém os dados requiridos. Os dados são retornados para o Google Assistant que, por sua vez, envia para o usuário por comando de voz.

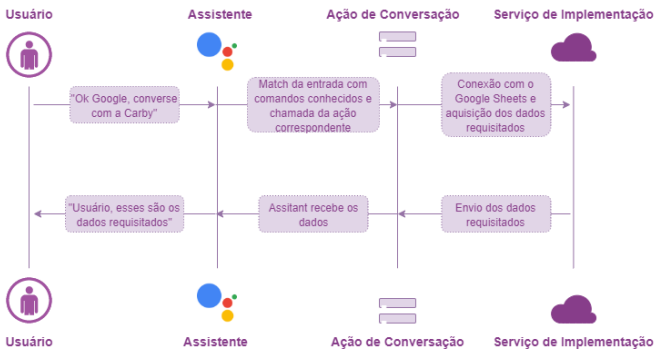

Fig. 10. Fluxograma das ações customizadas do Google Assitant

Como primeiro passo, foi realizado um cadastro no *Google* para montar a assistente, fazer o registro, ativar a API, adquirir o modelo, o ID e realizar o *download* da aplicação. Em seguida foram instaladas as *APIs* do *Google* por meio de um ambiente virtual (*venv*) e depois foi necessário fazer autorização do *Google Assistant* na Raspberry. Por meio do comando **–lang pt-BR** as funcionalidades da assistente foram transferidas para o idioma Português-Brasil e suas funcionalidades foram testadas.

Para que a Assistente transmita informações a respeito das grandezas medidas pelo módulo de aquisição, são customizadas ações de conversação, por meio da aplicação do Google *Actions Console*, é parametrizado as variáveis temperatura, umidade e concentração de CO2. Usando a aplicação *Sheet DB* na planilha do *Google Sheets*, que contém os dados de leitura dos sensores, é possível alterar a planilha pra um arquivo Json API que atualiza conforme são obtidos novos dados dos sensores. Além do mais é criado um arquivo *.js* na API do *Google Assistant* e cria-se uma função para receber os dados da planilha usando o cliente HTTP Axios a partir do link do arquivo Json API. Então, as funções *GetTemperature, GetUmidity, GetCO e GetData* são categorizadas para que, se o usuário informar que deseja saber o valor de alguma grandeza medida, ele seja informado da última medição feita, que corresponde à última linha da planilha.

Por meio do*Actions Console do Google Assistant*, criou-se um Projeto de Ação onde toda a lógica das ações de conversação foram implementadas. Logo, foram elaboradas as seguintes pastas:

1) ***Actions***: A invocação de uma ação é definida por um único *intent* que é correspondido quando os usuários solicitam a ação. Para o projeto a intenção inicial (*main invocation*) foi definida como ”Ok Google, falar com Carby” no arquivo *action.intent.MAIN*. A segunda e a terceira invocações são invocações de link direto que permitem especificar frases adicionais que permitem aos usuários solicitar uma funcionalidade específica. A figura 11 ilustra o fluxo.

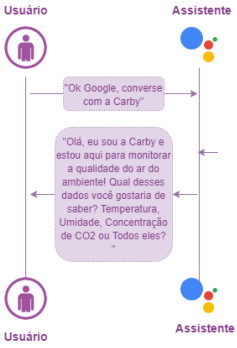

Fig. 11. *Fluxograma ”Actions”.*

2) ***Custom***:
   - ***Intents***: Por meio das intents é possível estender a capacidade do Assistant de entender as solicitações do usuário que são específicas para a Carby. Dessa forma, foram definidas as frases de treinamento personalizadas

dentro de uma intent, que por sua vez gerou um modelo de linguagem de intent. Logo para o projeto foram definidas as intenções negativa e positiva com suas respectivas frases de treinamento conforme a figura 12.

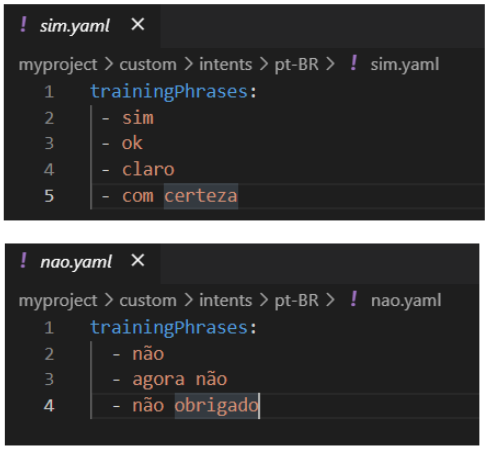

Fig. 12. *Intents* de Sim e Não

- ***Types***: Pode-se anotar frases de treinamento com *Types* para criar *slots*. Quando os usuários dizem algo que corresponde a um sinônimo no *slot* conforme a figura 13, o mecanismo *Natural Language Understanding* (NLU) o extrai como um parâmetro digitado, para que você possa processá-lo em uma cena.

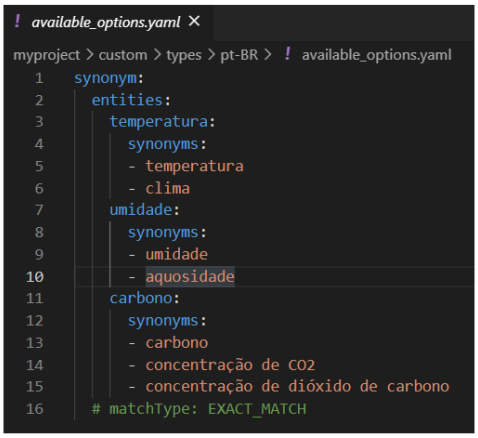

Fig. 13. *Type* opções válidas

- ***Scenes***: Em combinação com as intenções, as cenas são executadas em um *loop* até atender aos critérios de transição definidos pelo usuário. Isso permite que você crie fluxos de lógica de controle com muito mais eficiência em uma única cena. No projeto foram criadas quatro cenas:
  - *Start*: Após a inovação ”falar com Carby” , essa cena tem a função de selecionar o dado (Temperatura, Umidade e Concentração de CO2) escolhido e enviar a resposta obtida pelos sensores a partir da integrar a aquisição dos dados pelo *Webhook* com a API do *Google Assistant*. A figura 14 ilustra o fluxo.

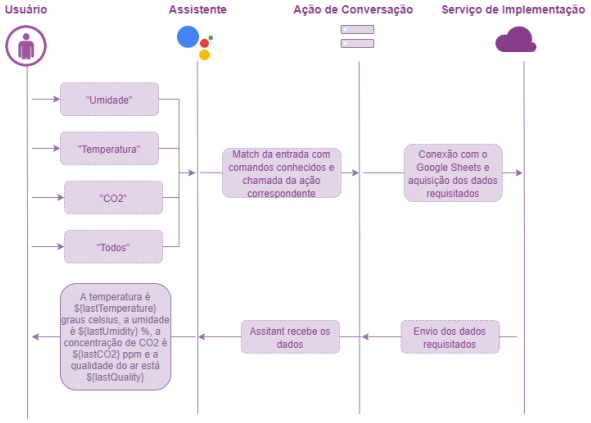

Fig. 14. *Fluxograma ”Start”.*

  - *Integração*: tem a função de perguntar ao usuário se ele quer saber algum outro dado (Temperatura, Umidade, Concentração de CO2 ou todos).
  - *Dados*: Após o questionamento se o usuário quiser saber de outro dado o usuário deve responder ”sim” ou ”não” e a partir disso é direcionado para a cena de ”Start” caso a resposta for ”sim”, e para cena ”Fim” caso a resposta for ”não”.A figura 15 ilustra o fluxo das cenas Integração e Dados.

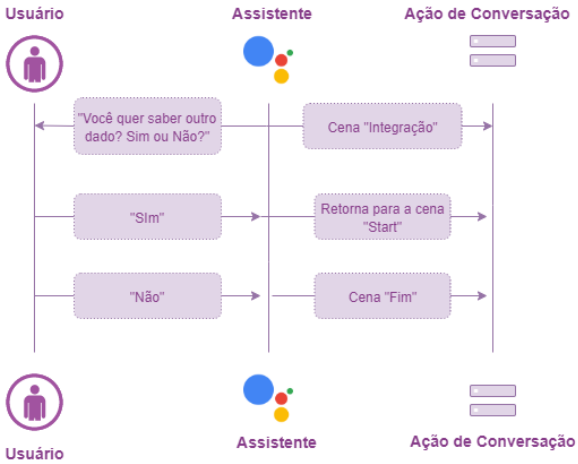

Fig. 15. *Fluxograma ”Integração” e ”Dados”.*

  - *Fim*: Encerramento da conversa. A figura 16 ilustra o fluxo da cena Fim.
  - ***Aviso***: Aviso quando a variável *Qualidade* da planilha estiver com a string ”Ruim”. A figura 17 ilustra o fluxo.

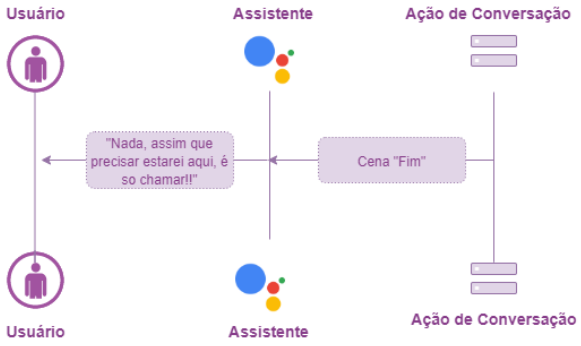

Fig. 16. *Fluxograma "Fim".*

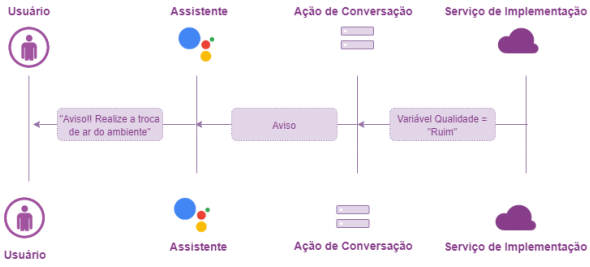

Fig. 17. *Fluxograma "Aviso".*

3) ***Settings***: As configurações locais do *Google Actions* fornecem os dados de voz, localização, nome, linguagem, região e o ID do projeto.

Posteriormente, com o *Google Assistant SDk* já instalado na *Raspberry* foi necessário apenas colocar o seguinte comando para instalar na *Raspberry* as funções e ações criadas: "google-oauthlib-tool –scope https://www.googleapis.com/auth/assistant-sdk-prototype –save –headless –client-secrets /path/to/client secret client-id.json"

6) ***Automação****:* Para o pleno funcionamento do dispositivo Carby, deseja-se que quando o dispositivo for ligado à energia a Raspberry inicie as leituras dos sensores, bem como o envio dos dados para a planilha no Google Sheets e se conecte ao Google Assistant de forma automática.

A automatização foi feita a partir da criação de três scripts *SHELL* com os executáveis da leitura dos sensores de concentração de CO_2, temperatura e umidade, o segundo para para a ativação do Google Assistant e o terceiro que simula o usuário apertar a tecla "Enter" já que utiliza-se o comando *"Push to talk"*. Sendo assim, o laço *for* compara o número de PID daquele processo com o obtido anteriormente. Se forem diferentes, o que indica que a Rasp foi reiniciada, os comandos são executados novamente. Ao acessar o arquivo *crontab* da Raspberry pelo terminal com o comando "crontab -e", é possível edita-lo. Foram adicionados dois comandos, um para cada arquivo *.sh*, que executa o script SHELL a cada minuto. Assim, quando o dispositivo é ligado os comandos são executados e, a cada minuto, é verificado se ainda se trata do mesmo processo. Sendo um processo diferente, os comandos são executados novamente.

IX. Resultados

1) ***Comunicação sensor de umidade e temperatura com a Raspberry****:* A figura 18 mostra o resultado do teste da comunicação com o sensor DHT11. Os dados são enviados para um arquivo *database.csv* e para um formulário do Google.

Fig. 18. Arquivo data base do sensor DHT11.

2) ***Comunicação sensor de gases com a Raspberry****:* A figura 19 mostra o resultado do teste de comunicação do sensor MQ-135 para diferentes concentrações de CO2. Assim, os testes foram realizados em um ambiente aberto, e tem-se a comparação dos valores obtidos no caso 1, no qual o sensor fica distante da respiração de uma pessoa, resultando em uma baixa concentração de CO_2. Enquanto que no caso 2, assim que o sensor é aproximado de uma pessoa respirando, é possível visualizar o aumento na concentração de CO_2. Com o afastamento da pessoa, tem-se o caso 3 com a gradual queda da concentração.

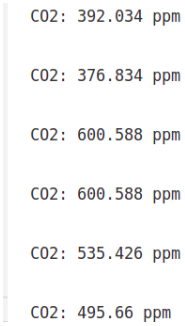

Fig. 19. Leitura do sensor MQ-135 com variação da concentração de CO2.

3) ***Integração sensor de gases, sensor temperatura e umidade com a Raspberry, envio dos dados para o Google Sheets e acionamento dos LEDs****:* Com a leitura dos sensores sendo feitas em um mesmo arquivo, todos os dados obtidos podem ser enviados, através do comando *curl* para uma mesma planilha do Google Sheets com uma coluna para cada grandeza conforme demostrado na figura 20.

Os LEDs verde, amarelo e vermelho são devidamente acionados conforme muda a qualidade do ar no ambiente. Um comando de voz é emitido alertando para a necessidade de trocas de ar no ambiente quando o LED vermelho é aceso. Quando a concentração de CO_2 for abaixo de 600 ppm, o LED verde é acionado. Quando a concentração de CO_2 for

Carby

Arquivo Editar Ver Inserir Formatar Dados Ferramentas Formulário Complementos Ajuda

	A	B	C	D	E
1	Timestamp	Temperatura	Umidade	CO2	Qualidade
108	14/05/2021 16:59:08	26,6	71	1159	Ruim
109	14/05/2021 17:00:09	26,8	45	362	Boa
110	14/05/2021 17:01:10	27,3	62	362	Boa
111	14/05/2021 17:02:11	27,9	42	416	Boa
112	14/05/2021 17:03:11	28,2	45	508	Boa
113	14/05/2021 17:04:12	28,5	45	580	Boa
114	14/05/2021 17:05:13	28,8	46	619	Media
115	14/05/2021 17:06:14	29	53	445	Boa

Fig. 20. Tabela no Google Sheets com os dados de umidade, temperatura e concentração de CO_2 e a qualidade do ar.

entre 600 ppm e 1000 ppm, o LED amarelo é acionado. Por último, quando a concentração de CO2 for maior que 1000 ppm, o LED vermelho foi acionado, como mostra a figura 21.

Fig. 21. Módulo Carby com os LEDs verde, amarelo e verde acesos.

*4) **Assistente de voz - Google Assistant SDK**:* O *Google Assistant* foi instalado na Raspberry com o sistema operacional *Ubuntu 20.04*. No sistema operacional Raspbian há erros de compilação e não é possível concluir a instalação de suas APIs. No sistema Ubuntu, o Google Assistant foi integrado com o microfone e o alto-falante e foi possível haver comunicação no idioma Português.

```
INFO:root:Transcript of user request: "qual é a temperatura e a previsão do tempo  de amanhã".
INFO:root:End of audio request detected.
INFO:root:Stopping recording.
INFO:root:Transcript of user request: "qual é a temperatura e a previsão do tempo de ananhã".
INFO:root:Playing assistant response.
```

Fig. 22. Terminal rodando o *Google Assistant*

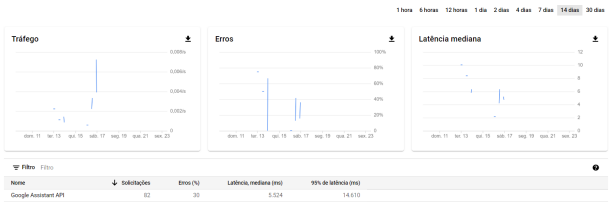

Fig. 23. Painel da API *Google Assistant*

Na figura 22, o *Google Assistant* está rodando no terminal. Dessa forma, é possível visualizar que é reconhecido o comando de voz, uma vez que, tem-se a escrita do que foi solicitado pelo comando de voz, e quase instantaneamente é enviada a resposta. A figura 23, se refere às requisições que são enviadas ao *Google*, provando seu bom funcionamento.

Para realizar o teste da implementação das ações foi realizado um *deploy* para a plataforma *Actions Console* e assim foi realizado o diálogo conforme a figura 24:

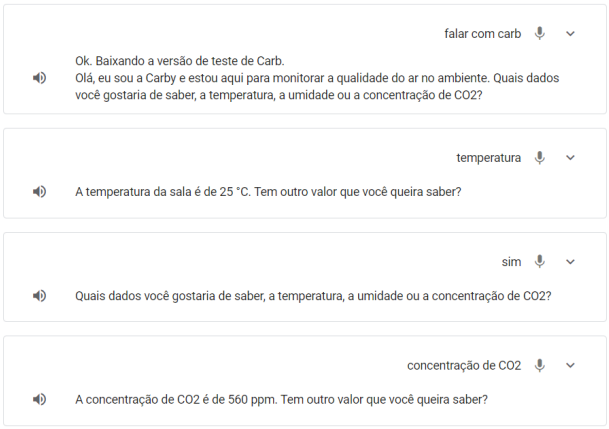

Fig. 24. Simulador do *Actions Console*

*5) **Gráfico dos dados de monitoramento - Excel**:* Foi realizada uma interface para o usuário verificar a concentração de CO_2, temperatura e umidade ao longo do dia conforme apresentado na figura 25. Para o próximo ponto de controle, a base de dados, os gráficos e os *labels* superiores serão atualizados de forma automática.

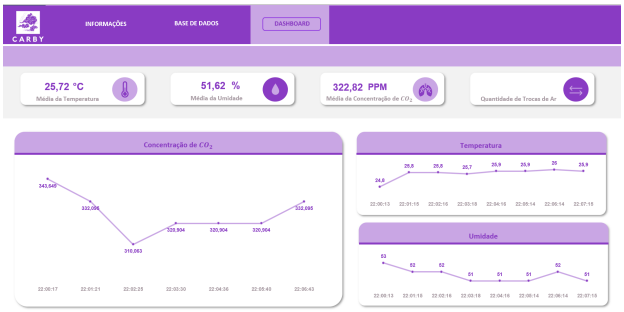

Fig. 25. Interface com o usuário

*6) **Teste do Módulo Carby - Clínica Odontológica**:* Os testes foram realizados em uma clínica odontológica em sua sala de espera. Segundo [15], é bom que o módulo de monitoramento da qualidade do ar seja alocado distante de janelas, em uma altura de 1,2m do chão e à 1,5m de paredes. Assim, o módulo foi posicionado em uma bancada à 1 metro do chão e 1,5 metros de uma parede, o mais distante possível de janelas e em contato intermediário com fluxo de pessoas. A figura 27 mostra o protótipo físico do Módulo Carby finalizado e a figura 26 o Módulo Carby na recepção de um consultório odontológico.

Como experimento, o módulo foi deixado em funcionamento durante todo o dia e parte da noite. Conforme os dados eram adquiridos, os LEDs iam se intercalando de acordo com a concentração medida. O assistente de voz respondeu prontamente quando chamado e alertou para casos de concentração alta de CO2.

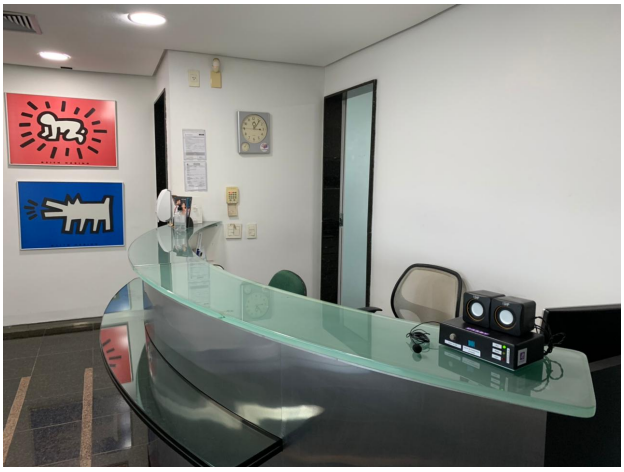

Fig. 26. Módulo Carby em um consultório odontológico.

X. Considerações Finais

Conforme proposto inicialmente pela dupla, o Módulo Carby de monitoramento da qualidade de carbono no ar, de fato, faz aquisição de dados referentes a umidade, temperatura e umidade através dos sensores DHT11 e MQ-135, disponibiliza uma interface gráfica com a concentração de tais grandezas nas últimas 24 horas, bem como suas médias e disponibiliza um assistente de voz para que as requisições e alertas sejam feitos por ele. Todo esse processamento é feito pela Raspberry Pi 3b+.

Fig. 27. Módulo Carby.

Referências

[1] Organização Pan-Americana da saúde, "Folha informativa - covid-19 (doença causada pelo novo coronavírus)," 2020, [Online; accessed 22-julho-2020]. [Online]. Available: https://www.paho.org/bra

[2] V. Aquino and N. Monteiro, "Brasil confirma o primeiro caso da doença," 2020, [Online; accessed 23-julho-2020]. [Online]. Available: saude.gov.br/noticias/agencia-saude/46435-brasil-confirma-primeiro-caso-de-novo-coronavirus

[3] Ministério da Saúde, "Sobre a doença," 2020, [Online; accessed 23-fevereiro-2021]. [Online]. Available: https://coronavirus.saude.gov.br/

[4] ANSES, "Carbon dioxide (co2) in indoor air," 2016, [Online; accessed 25-fevereiro-2021]. [Online]. Available: https://www.anses.fr/en/content/carbon-dioxide-co2-indoor-air

[5] Z. Peng and J. L. Jimenez, "Exhaled co2 as covid-19 infection risk proxy for different indoor environments and activities," *medRxiv*, 2020.

[6] R. K. Bhagat, M. D. Wykes, S. B. Dalziel, and P. Linden, "Effects of ventilation on the indoor spread of covid-19," *Journal of Fluid Mechanics*, vol. 903, 2020.

[7] S. N. R. De Araújo, S. A. R. Farias, D. S. Cruz, and R. Farias, "Concentração de dióxido de carbono em salas de aula da ufcg, climatizadas artificialmente," 2018.

[8] ANVISA, *Resolução - RE nº9, de 16 de janeiro de 2003*.

[9] C. Wang, L. Miao, Z. Wang, Y. Xiong, Y. Jiao, and H. Liu, "Emergency management in dental clinic during the coronavirus disease 2019 (covid-19) epidemic in beijing," *International dental journal*, 2021.

[10] S. N. Isha, A. Ahmad, R. Kabir, and E. H. Apu, "Dental clinic architecture prevents covid-19-like infectious diseases," *HERD: Health Environments Research & Design Journal*, vol. 13, no. 4, pp. 240–241, 2020.

[11] V. G. He Zhang, Ravi Srinivasan, "Low cost, multi-pollutant sensing system using raspberry pi for indoor air quality monitoring," *Sustainability*, 2020.

[12] FAPESP, "Sistema monitora presença do novo coronavírus no ar," 2020, [Online; accessed 25-fevereiro-2021]. [Online]. Available: https://pesquisaparainovacao.fapesp.br/1526/boletim

[13] *Technical Data MQ-135 Gas Sensor*.

[14] "Configure and read out the raspberry pi gas sensor (mq-x)," 2017, [Online; accessed 23-março-2021]. [Online]. Available: https://tutorials-raspberrypi.com/configure-and-read-out-the-raspberry-pi-gas-sensor-mq-x/

[15] R. S. He Zhang and V. Ganesan, "Low cost, multi-pollutant sensing system using raspberry pi for indoor air quality monitoring," *Sustentability*, 2021.

Apêndice

a) Comunicação sensor DHT11, sensor MQ-135 e LEDs com a Raspberry:

```
#include <errno.h>
#include <pigpio.h>
#include <fstream>
#include <limits.h>
#include <fcntl.h>
#include <getopt.h>
#include <linux/spi/spidev.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>
#include <iostream>
#include <vector>
#include <map>
#include <math.h>
#include <algorithm>
#include <iostream>
#include <iomanip>

using namespace std;
#define MAX_ADC_CH 8

#define DHTPIN 4
#define PINO_Vm 23
```

```
#define PINO_A 18
#define PINO_Vd 17

string x, y;
string formid =
↪  "1_q2E8SroEc5OOQh54l85XWK4
mJrh9U7zLE_mN11dT_I";
static unsigned cleanupPin = UINT_MAX;
static bool verbose = false;

int read_dht11(unsigned pin) {
    gpioSetMode(pin, PI_OUTPUT);
    gpioDelay(19 * 1000);
    gpioSetMode(pin, PI_INPUT);
    return 0;
}

static void cleanup(void) {
    if (verbose) {
      fprintf(stderr,"...
      ↪  cleanup()\n");
    }
    if (cleanupPin != UINT_MAX) {
      gpioSetPullUpDown( cleanupPin,
      ↪  PI_PUD_OFF);
    }
    gpioTerminate();
}

enum pulse_state { PS_IDLE = 0,
↪  PS_PREAMBLE_STARTED, PS_DIGITS };

static void pulse_reader(int gpio, int
↪  level, uint32_t tick) {
    ofstream database;
    database.open ("database.csv",
    ↪  std::ofstream::out |
    ↪  std::ofstream::app);
    static uint32_t lastTick = 0;
    static enum pulse_state state =
    ↪  PS_IDLE;
    static uint64_t accum = 0;
    static int count = 0;
    uint32_t len = tick - lastTick;
    lastTick = tick;

    switch (state) {
      case PS_IDLE:
        if (level == 1 && len > 70 &&
        ↪  len < 95) {
          state = PS_PREAMBLE_STARTED;
        }
        else {
          state = PS_IDLE;
        }
        break;
      case PS_PREAMBLE_STARTED:
    if (level == 0 && len > 70 && len <
    ↪  95) {
      state = PS_DIGITS;
      accum = 0;
      count= 0;
    } else state = PS_IDLE;
      break;
      case PS_DIGITS:
    if (level == 1 && len >= 35 && len
    ↪  <= 65);
    else if (level == 0 && len >= 15 &&
    ↪  len <= 35) {
      accum <<= 1;
      count++;
    }
    else if (level == 0 && len >= 60 &&
    ↪  len <= 80) {
      accum = (accum << 1) + 1;
      count++;
    }
    else {
      state = PS_IDLE;
    }

    if (count == 40) {
      state = PS_IDLE;

      uint8_t parity = (accum & 0xff);
      uint8_t tempLow = ((accum>>8) &
      ↪  0xff);
      uint8_t tempHigh = ((accum>>16)
      ↪  & 0xff);
      uint8_t humLow = ((accum>>24) &
      ↪  0xff);
      uint8_t humHigh = ((accum>>32) &
      ↪  0xff);

      uint8_t sum = tempLow + tempHigh
      ↪  + humLow + humHigh;
      bool valid = (parity == sum);

      if (valid) {
        printf("{\"Temperatura\":
        ↪  %d,%d, \"Umidade\":
        ↪  %d,%d}\n", tempHigh,
        ↪  tempLow, humHigh, humLow);
        string temp =
        ↪  to_string(tempHigh) + ','
        ↪  + to_string(tempLow);
        string humi =
        ↪  to_string(humHigh) + ',' +
        ↪  to_string(humLow);
        x = temp;
        y = humi;
      }
```

```
    }
    break;
  }
  if (verbose) {
    printf("pulse %c %4uµS state = %d digits = %d\n", (level == 0 ? 'H' : (level == 1 ? 'L' : 'W')), len, state, count);
  }
}

int selectedChannels[MAX_ADC_CH];
int channels[MAX_ADC_CH];
char spidev_path[] = "/dev/spidev0.0";
const int blocksDefault = 1;
const int channelDefault = 0;
const int samplesDefault = 100;
const int freqDefault = 0;
const int clockRateDefault = 3600000;
const int coldSamples = 1000;

vector<double> CO2Curve {2.0, 0.004, -0.34};
class MQ {
 public:
  int MQ_PIN = 0;
  int RL_VALUE = 20;
  double RO_CLEAN_AIR_FACTOR = 3.8;
  int val;

  int CALIBRATION_SAMPLE_TIMES = 50;
  int CALIBRATION_SAMPLE_INTERVAL = 500;

  int READ_SAMPLE_INTERVAL = 50;
  int READ_SAMPLE_TIMES = 5;
  int GAS_CO2 = 0;
  int Ro;
  int analogPin;
  MQ() : Ro(185), analogPin(0) {}
  MQ(int _ro, int _analogPin) {
    Ro = _ro;
    MQ_PIN = _analogPin;
    cout << "Calibrating..." << '\n';
    Ro = MQCalibration(MQ_PIN);
    cout << "Calibration is done..." << '\n';
    cout.precision(3);
    cout << "Ro= " << Ro << '\n';
  }
  double MQResistanceCalculation(int raw_adc) {
    return double(RL_VALUE * (1023.0 - raw_adc) / double(raw_adc));
  }

  double MQCalibration(int mq_pin) {
    double value = 0.0;
    for (int i = 0; i < CALIBRATION_SAMPLE_TIMES; i++) {
      value += MQResistanceCalculation(val);

      sleep(CALIBRATION_SAMPLE_INTERVAL / 1000.0);
    }
    value /= CALIBRATION_SAMPLE_TIMES;
    value /= RO_CLEAN_AIR_FACTOR;
    return value;
  }

  double MQRead(int mq_pin) {
    double rs = 0.0;
    for (int i = 0; i < READ_SAMPLE_TIMES; i++) {
      rs += MQResistanceCalculation(val);
sleep(READ_SAMPLE_INTERVAL / 1000.0);
    }
    rs /= READ_SAMPLE_TIMES;
    return rs;
  }

  double MQGetGasPercentage(double rs_ro_ratio, int gas_id) {
    if (gas_id == GAS_CO2) {
      return MQGetPercentage(rs_ro_ratio, CO2Curve);
    }
    return 0;
  }

  double MQGetPercentage(double rs_ro_ratio, const vector<double>& pcurve) {
    return (pow(10, (((log(rs_ro_ratio) - pcurve[1]) / pcurve[2]) + pcurve[0])));
  }

  map<string, double> MQPercentage() {
    map<string, double> val;
    double read = MQRead(MQ_PIN);
    val["GAS_CO2"] = MQGetGasPercentage(read / Ro, GAS_CO2);
    return val;
  }
};
```

```
int main(int argc, char *argv[]) {
    unsigned pin = DHTPIN;
    if (gpioInitialise() ==
     ↪  PI_INIT_FAILED) {
      fprintf(stderr, "failed to
       ↪  initialize GPIO\n");
      exit(EXIT_FAILURE);
    }
    atexit(cleanup);
    gpioSetMode(pin, PI_INPUT);
    gpioSetPullUpDown(pin, PI_PUD_UP);
    gpioWrite(pin, 0);
    cleanupPin = pin;
    gpioSetWatchdog(pin, 50);
    MQ *mq = new MQ();
    while (true) {
        int i, j;
        int ch_len = 0;
        int vSamples = samplesDefault;
        double vFreq = freqDefault;
        int vClockRate =
         ↪  clockRateDefault;
        int vBlocks = blocksDefault;

        if (ch_len == 0) {
            ch_len = 1;
            channels[0] =
             ↪  channelDefault;
        }
        int microDelay = 0;
        if (vFreq != 0) {
            microDelay = 1000000 /
             ↪  vFreq;
        }
        int count = 0;
        int fd = 0;
        int val;
        struct timeval start;
        int *data;
        data = (int*)malloc(ch_len *
         ↪  vSamples * sizeof(int));
        struct spi_ioc_transfer *tr =
         ↪  0;
        unsigned char *tx = 0;
        unsigned char *rx = 0;
        tr = (struct spi_ioc_transfer
         ↪  *)malloc(ch_len * vBlocks
         ↪  * sizeof(struct
         ↪  spi_ioc_transfer));
        if (!tr) {
            perror("malloc");
            goto loop_done;
        }
        tx = (unsigned char
         ↪  *)malloc(ch_len * vBlocks
         ↪  * 4);
        if (!tx) {
            perror("malloc");
            goto loop_done;
        }
        rx = (unsigned char
         ↪  *)malloc(ch_len * vBlocks
         ↪  * 4);
        if (!rx) {
            perror("malloc");
            goto loop_done;
        }
        memset(tr, 0, ch_len * vBlocks
         ↪  * sizeof(struct
         ↪  spi_ioc_transfer));
        memset(tx, 0, ch_len *
         ↪  vBlocks);
        memset(rx, 0, ch_len *
         ↪  vBlocks);
        for (i = 0; i < vBlocks; i++)
         ↪  {
            for (j = 0; j < ch_len;
             ↪  j++) {
                tx[(i * ch_len + j) *
                 ↪  4] = 0x60 |
                 ↪  (channels[j] <<
                 ↪  2);
                tr[i * ch_len +
                 ↪  j].tx_buf =
                 ↪  (unsigned
                 ↪  long)&tx[(i *
                 ↪  ch_len + j) * 4];
                tr[i * ch_len +
                 ↪  j].rx_buf =
                 ↪  (unsigned
                 ↪  long)&rx[(i *
                 ↪  ch_len + j) * 4];
                tr[i * ch_len + j].len
                 ↪  = 3;
                tr[i * ch_len +
                 ↪  j].speed_hz =
                 ↪  vClockRate;
                tr[i * ch_len +
                 ↪  j].cs_change = 1;
            }
        }
        tr[ch_len * vBlocks -
         ↪  1].cs_change = 0;
        fd = open(spidev_path,
         ↪  O_RDWR);
        if (fd < 0) {
            perror("open()");
            printf("%s\n",
             ↪  spidev_path);
```

```
            goto loop_done;
        }
        while (count < coldSamples) {
            if (ioctl(fd,
             ↪  SPI_IOC_MESSAGE(ch_len
             ↪  * vBlocks), tr) < 0) {
                perror("ioctl");
                goto loop_done;
            }
            count += ch_len * vBlocks;
        }
        count = 0;
        if (gettimeofday(&start, NULL)
         ↪  < 0) {
            perror("gettimeofday:
             ↪  start");
            return 1;
        }
        while (count < ch_len *
         ↪  vSamples) {
            if (ioctl(fd,
             ↪  SPI_IOC_MESSAGE(ch_len
             ↪  * vBlocks), tr) < 0) {
                perror("ioctl");
                goto loop_done;
            }
            for (i = 0, j = 0; i <
             ↪  ch_len * vBlocks; i++,
             ↪  j += 4) {
                val = (rx[j + 1] << 2)
                 ↪  + (rx[j + 2] >>
                 ↪  6);
                mq->val = val;
                data[count + i] = val;
            }
            count += ch_len * vBlocks;
            if (microDelay > 0) {
                usleep(microDelay);
            }
        }
    loop_done:
      map<string, double> perc =
       ↪  mq->MQPercentage();
      cout << "CO2: " <<
       ↪  perc["GAS_CO2"] << " ppm\n";
      string quality = "";
      if (perc["GAS_CO2"] < 600.0) {
        cout << "A qualidade do ar
         ↪  está boa" << '\n';
        gpioSetMode(PINO_Vm, 0);
        gpioSetMode(PINO_Vd, PI_ALT0);
        gpioSetMode(PINO_A, 0);
        quality = "Boa";
      } else if (perc["GAS_CO2"] >=
       ↪  600.0 && perc["GAS_CO2"] <=
       ↪  1000.0) {
        cout << "A qualidade do ar
         ↪  está média" << '\n';
        gpioSetMode(PINO_Vm, 0);
        gpioSetMode(PINO_Vd, 0);
        gpioSetMode(PINO_A, PI_ALT0);
        quality = "Media";
      } else if (perc["GAS_CO2"] >
       ↪  1000.0) {
        cout << "A qualidade do ar
         ↪  está ruim" << '\n';
        gpioSetMode(PINO_Vm, PI_ALT0);
        gpioSetMode(PINO_Vd, 0);
        gpioSetMode(PINO_A, 0);
        quality = "Ruim";
      }
      string CO2 =
       ↪  to_string(perc["GAS_CO2"]);
      replace(CO2.begin(), CO2.end(),
       ↪  '.', ',');
      string CO2_dininutivo = "";
      for(char c:CO2){
        if(c == ','){
          break;
        }
        CO2_dininutivo += c;
      }
      gpioSetAlertFunc(pin,
       ↪  pulse_reader);
      read_dht11(pin);
      if (x != "" && y != "") {
       string command = "curl
        ↪  https://docs.google.com/
       forms/d/"+ formid +
        ↪  "/formResponse -d ifq -d
        ↪  \"entry.1076682711= +" + x
        ↪  + "\" -d
        ↪  \"entry.1505376468= + " + y
        ↪  + "\" -d
        ↪  \"entry.1031950862= + " +
        ↪  CO2_dininutivo + "\" -d
        ↪  \"entry.41984470= + " +
        ↪  quality + "\" -d
        ↪  submit=Submit;";
        system(command.c_str());
        sleep(60.0);
      }
      if (fd)
        close(fd);
      if (rx)
        free(rx);
      if (tx)
        free(tx);
      if (tr)
        free(tr);
    }
    return 0;
```

```
}
```

b) Script SHELL executável da leitura dos sensores e ativação dos LEDs:

```
#!/bin/bash

PID=`cat /home/ubuntu/dados.pid`

if ! ps -p $PID > /dev/null
then
  rm /home/ubuntu/dados.pid
  sudo ./mcp3008hwspi & echo $!
  >>/home/ubuntu/dados.pid
fi
```

c) Script SHELL executável do Google Assistant:

```
#!/bin/bash

PID=`cat /home/ubuntu/assistant.pid`

if ! ps -p $PID > /dev/null
then
  rm /home/ubuntu/assistant.pid
  screen -S assistant
  source env/bin/activate
  googlesamples-assistant-pushtotalk
  --project-id
  \'raspberriassistant-600e5\'
  --device-model-id
  'raspberriassistant-600e5-
  carby-j63dqh' --lang pt-BR & echo $!
  >>/home/ubuntu/assistant.pid
fi
```

d) Script SHELL executável para pressionar enter:

```
#!/bin/bash

PID=`cat /home/ubuntu/assistant.pid`

if ps -p $PID > /dev/null
then
  for i in {1..58}
  do
    screen -S assistant -p 0 -X echo
    sleep 1
  done
fi
```

e) Script para acionamento do Google Assistant:

```
const { conversation } =
  require('@assistant/conversation');
const functions =
  require('firebase-functions');
const axios = require('axios');

const {
  dialogflow,
  Image,
} = require('actions-on-google');

// Create an app instance
const app = conversation();

app.handle('Start', conv => {
  conv.add('Olá, eu sou a Carby e
  estou aqui para monitorar a
  qualidade do ar do ambiente!
  Qual desses dados você gostaria
  de saber? Temperatura, Umidade,
  Concentração de CO2 ou Todos
  eles?');
});

app.handle('Finish', conv => {
  conv.add('Nada, assim que precisar
  estarei aqui, só chamar!!!')
});


function getSpreadSheetData() {
  return
  axios.get('https://sheetdb.io/api
  /v1/9g3wvetd2bx25');
}

app.handle('GetTemperature', conv => {
  return
  getSpreadSheetData().then(my_data
  => {
    const lastTemperature =
    my_data.data[my_data.data.length
    - 1].Temperatura;
    conv.add(`A temperatura é de
    ${lastTemperature} graus`);
    conv.add('Você gostaria de saber
    mais algum dado? Sim ou
    não?');
  });
});

app.handle('GetUmidity', conv => {
```

```
  return
   ↪ getSpreadSheetData().then(my_data
   ↪ => {
    const lastUmidity =
     ↪ my_data.data[my_data.data.length
     ↪ - 1].Umidade;
    conv.add(`A umidade está em
     ↪ ${lastUmidity} %`);
    conv.add('Você gostaria de saber
     ↪ mais algum dado? Sim ou
     ↪ não?');
  });
});

app.handle('GetCO', conv => {
  return
   ↪ getSpreadSheetData().then(my_data
   ↪ => {
    const lastCO2 =
     ↪ my_data.data[my_data.data.length
     ↪ - 1].CO2;
    conv.add(`A concentração de CO2 é
     ↪ de ${lastCO2} PPM`);
    conv.add('Você gostaria de saber
     ↪ mais algum dado? Sim ou
     ↪ não?');
  });
});

app.handle('GetData', conv => {
  return
   ↪ getSpreadSheetData().then(my_data
   ↪ => {
    const lastTemperature =
     ↪ my_data.data[my_data.data.length
     ↪ - 1].Temperatura;
    const lastUmidity =
     ↪ my_data.data[my_data.data.length
     ↪ - 1].Umidade;
    const lastCO2 =
     ↪ my_data.data[my_data.data.length
     ↪ - 1].CO2;
    const lastQuality =
     ↪ my_data.data[my_data.data.length
     ↪ - 1].Qualidade;
    conv.add(`A temperatura é
     ↪ ${lastTemperature} graus
     ↪ celsius, a umidade é
     ↪ ${lastUmidity} %, a
     ↪ concentração de CO2 é
     ↪ ${lastCO2} ppm e a qualidade
     ↪ do ar está ${lastQuality}`);
    if (lastQuality === 'Ruim') {
      conv.add('Aviso!!!!, realize a
       ↪ troca de ar do ambiente.');
    }
  });
});


exports.ActionsOnGoogleFulfillment =
 ↪ functions.https.onRequest(app);
```