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 400 partes por milhão (ppm), e em um ambiente interno bem ventilado terá aproximadamente 800 ppm, dessa forma, um número maior do que esse indica que o ambiente precisa de mais ventilação [7].

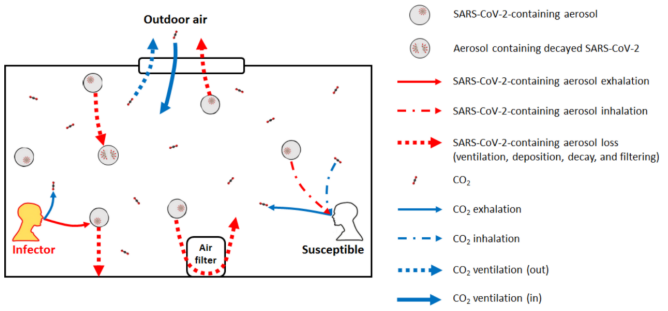

Fig. 1. Esquema da ilustração da expiração, inalação e outros processos de perda de SARS-CoV-2 contendo aerossóis em um ambiente interno. Fonte: Exhaled CO2 as COVID-19 infection risk proxy for different indoor environments and activities [5]

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 [8]. Assim, clínicas odontológicas podem provocar a infecção cruzada devido ao uso de instrumentos que produzem aerossóis, gotículas e secreções de saliva e sangue [9].

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 exautores, 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 quando 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.

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.

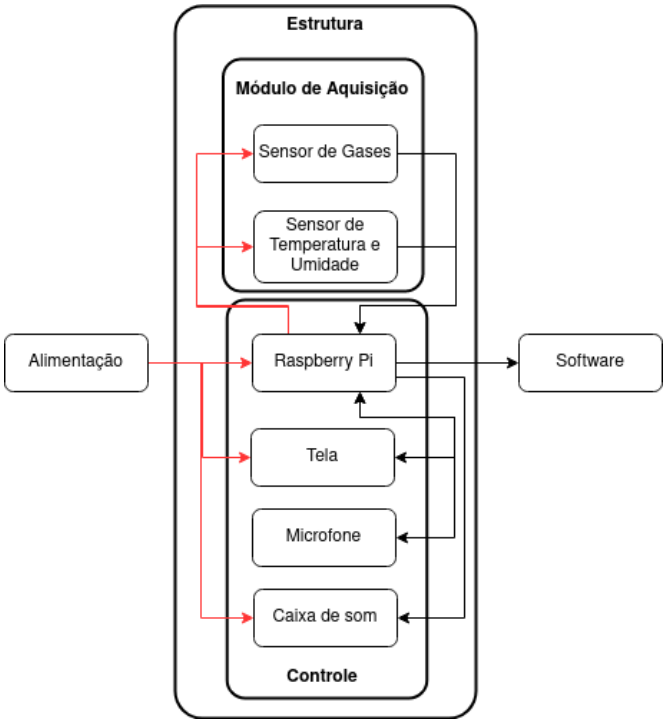

Fig. 2. Divisão das áreas de trabalho do protótipo

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 DSK* 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 $CO2$ nas últimas 24 horas, sendo assim, é fundamental a integração entre a *Raspberry Pi* e o armazenamento dos dados. Com o decorrer do projeto pode-se adicionar algumas funcionalidades no software, tais como a criação de um servidor, que gerencie diversos módulos de aquisição.

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.

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 [10]. 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.

Fig. 3. Detector de Gás Carbônico Portátil

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 [11].

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:** $R\$34,90$;
- Display Oled.**Preço:** $R\$37,90$;
- Caixa acrílica ou caixa de impressão 3D.
- 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**:* O sensor de temperatura e umidade, DHT11, é um sensor digital, e consequentemente os seus dados foram adquiridos a partir da conexão do sensor na *Raspberry* conforme representado na figura 4.

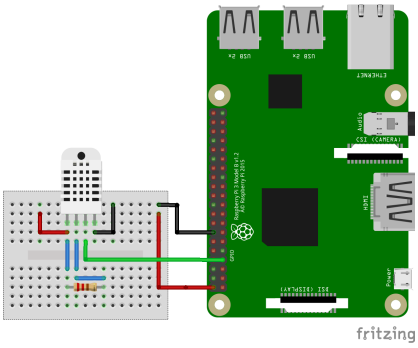

Fig. 4. Conexão sensor DHT11 com Rasberry Pi 3b+

Já o sensor de gás MQ-135, é um sensor analógico e precisa de calibração. Como a *Raspberry* não tem pinos de entrada analógica, faz-se necessário o uso de um conversor AD (MCP3008), conforme a figura 5.

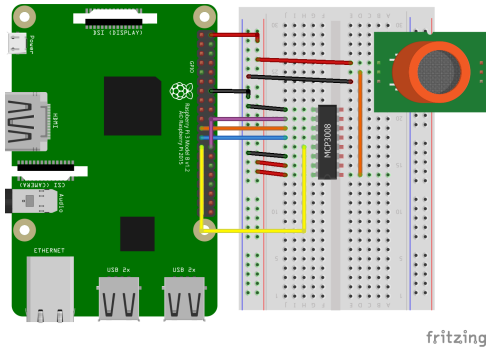

Fig. 5. Conexão sensor MQ135 com Rasberry Pi 3b+

Para a calibração do MQ-135, a figura 6 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 $20K\Omega$ [12].

Fundamentando no gráfico da figura 6, 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 [13], conforme a equação 1, com os pontos $x_1 = 100$ e $x_2 = 200$:

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

Além do mais, como no gráfico de referência usa-se um resistor de $20K\Omega$ e no *datsheet* é também é sugerido, esse

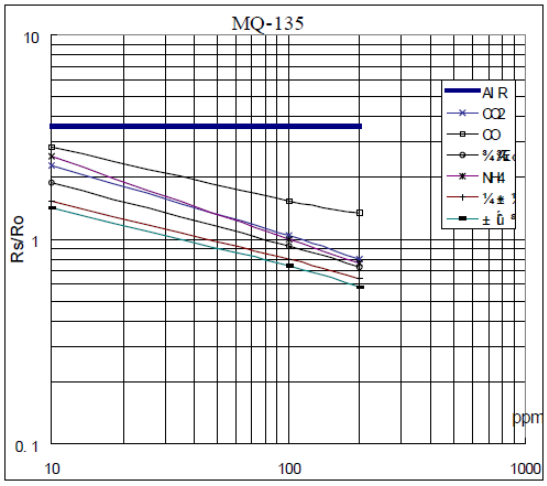

Fig. 6. Características de sensibilidade do sensor de gases MQ-135

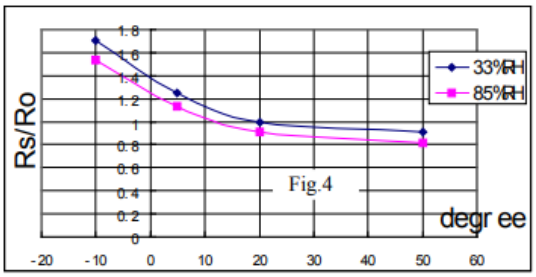

Fig. 7. Típica dependência do sensor MQ-135 da temperatura e umidade.

valor de resistência foi utilizado no potenciômetro do módulo do sensor.

Além do mais, 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.

2) Conexão Tela OLED - Raspberry Pi: Para realizar a conexão da *Raspberry* com o *Display* OLED, é necessário conectar os pinos de VCC em 3.3V (pino 3), o GND (pino 14), SDA (pino 3) e SCL (pino 5), conforme ilustrado na figura 8.

A implementação da tela OLED em linguagem C será implementada como uma funcionalidade extra conforme as disponibilidades de tempo. Isso porque o Assistente de voz e a planilha já são comunicações suficientes para envio de informações ao usuário.

B. Descrição de Software

Para o ponto de controle 3, a equipe focou em 3 pontos principais: transferir os códigos dos sensores de concentração de CO_2, temperatura e umidade da linguagem *Python* para C

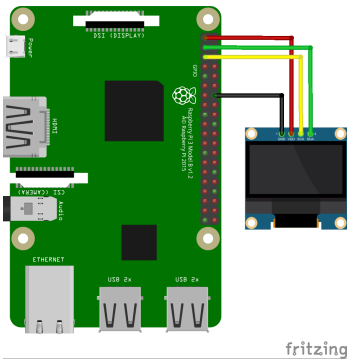

Fig. 8. Conexão *display* OLED com Rasberry Pi 3b+

e C++, enviar os dados dos sensores para o *Google Sheets* e ativar o *Google Assitant*.

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 6 com base na calibração realizada do R_0 e dos dados adquiridos por meio da equação 1. Sendo assim, retorna a concentração em partes por milhão do gás CO2.

3) ***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 dois formulários do *Google*, um deles específico para para temperatura e umidade e outro para concentração de CO_2. 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 ocorram problemas de sincronização, uma vez que, os códigos dos sensores são executados de forma manualmente e separados, então, quando for realizada uma *thread* ou processo pai-filho para automatizar a execução dos dois sensores imagina-se que não terão problemas de sincronização.

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.

4) ***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 ser alertado quando a troca de ar tiver sido concluída e a concentração de CO2 estiver em níveis adequados, essas funcionalidades poderão ser adicionadas com o Assistente da Google.

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 a utorização 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.

IX. Resultados

1) ***Comunicação sensor de umidade e temperatura com a Raspberry****:* A figura 9 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.

sensor_dht11.cpp | database.csv

database.csv

```
1   Temperatura;Umidade
2   27,0;42,0
3   27,1;42,0
4   27,0;42,0
5   27,1;41,0
6   27,4;41,0
7   27,1;41,0
8   27,1;41,0
9   27,1;41,0
10  27,3;41,0
11  27,1;41,0
12  27,1;42,0
```

Fig. 9. Arquivo data base do sensor DHT11

2) ***Comunicação sensor de gases com a Raspberry****:* A figura 10 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.

```
CO2: 392.034 ppm

CO2: 376.834 ppm

CO2: 600.588 ppm

CO2: 600.588 ppm

CO2: 535.426 ppm

CO2: 495.66 ppm
```

Fig. 10. Leitura do sensor MQ-135 com variação da concentração de CO2

3) ***Envio dos dados para o Google Sheets****:* Os dados de temperatura, umidade e concentração ficam armazenados no *Google Sheets* conforme as figuras 11 e 12.

Carby_teste

Arquivo Editar Ver Inserir Formatar Dados Ferramentas Formulário

C11

	A	B	C	D
1	Carimbo de data/hora	Temperatura	Umidade	
2	22/04/2021 22:00:13	24,8	53	
3	22/04/2021 22:01:15	25,8	52	
4	22/04/2021 22:02:16	25,8	52	
5	22/04/2021 22:03:18	25,7	51	
6	22/04/2021 22:04:16	25,9	51	
7	22/04/2021 22:05:14	25,9	51	
8	22/04/2021 22:06:14	26	52	
9	22/04/2021 22:07:15	25,9	51	

Fig. 11. Planilha com os dados de Temperatura e umidade

Carby_teste

Arquivo Editar Ver Inserir Formatar Dados Ferramentas Formulário

E17

	A	B	C
1	Carimbo de data/hora	Concentração de CO2	
2	22/04/2021 22:00:17	343,649532	
3	22/04/2021 22:01:21	332,09586	
4	22/04/2021 22:02:25	310,063882	
5	22/04/2021 22:03:30	320,904182	
6	22/04/2021 22:04:36	320,904182	
7	22/04/2021 22:05:40	320,904182	
8	22/04/2021 22:06:43	332,09586	

Fig. 12. Planilha com os dados da Concentração de CO_2

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 amanha".
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. 13. Terminal rodando *Google Assistant*

Fig. 14. Painel da API *Google Assistant*

Na figura 13 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 14 se refere às requisições que são enviadas ao Google, provando seu bom funcionamento.

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 15. Para o próximo ponto de controle, a base de dados, os gráficos e os *labels* superiores serão atualizados de forma automática.

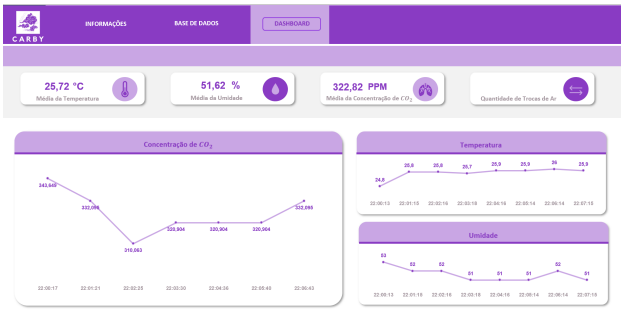

Fig. 15. Interface com o usuário

X. Considerações Finais

Nota-se que o embasamento teórico do projeto está consolidado e mostra-se como um ponto de partida para o desenvolvimento do projeto. As leituras dos sensores estão na linguagem C, o Google Assistant está instalado e em funcionamento na Raspberry e o gráfico com os dados da concentração de CO2 está disponível. Portanto, para o seguinte ponto de controle os códigos serão adaptados e alterados para estarem integrados com *threads*, bem como os dados de umidade, temperatura e concentração de CO2 sendo enviados juntos para uma mesma tabela, automatizada a atualização da planilha e o Google Assistant personalizado para a aplicação.

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] 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.

[9] 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.

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

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

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

[13] R. P. Tutorials, "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/

Apêndice

a) Comunicação sensor DHT11:

```
#include <pigpio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <fstream>
#include <limits.h>
#include <string>
#define DHTPIN 4
using namespace std;
bool ok = true;

string formid ="15IJEaBsBONEIt4QTXtla
 ↪  NEV0Q7UbLJjDFw5itk2_eoY";

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; //
     ↪  not handling rollover, you
     ↪  will get a bad read on that
     ↪  one
    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);

       string command = "curl
        ↪  https://docs.google.com/
        ↪  forms/d/" + formid +
        ↪  "/formResponse -d ifq -d
        ↪  \"entry.1115749519= +" + temp
        ↪  + "\" -d \"entry.460396820= +
        ↪  " + humi + "\" -d
        ↪  submit=Submit;";
          system(command.c_str());
          database << temp << ';' <<
           ↪  humi << '\n';
        }
      }
      break;
    }
}

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);
    while (1) {
      gpioSetAlertFunc( pin,
       ↪  pulse_reader);
      read_dht11(pin);
      gpioDelay(60000000);
    }
    exit(2);
    return 0;
}
```

b) Comunicação sensor MQ-135:

```
#include <errno.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;
string formid =
"19G0yK3KeaRI4AkYW-SpCNy2DJooWY
 ↪  -eOLfsvJWB4frQ";

#define MAX_ADC_CH 8
int selectedChannels[MAX_ADC_CH];
int channels[MAX_ADC_CH];
char spidev_path[] = "/dev/spidev0.0";
const int blocksDefault = 1;
```

```cpp
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(67), 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[]) {
    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;
                    //cout << "Val == " <<
                     ↪  val << '\n';
                    map<string, double>
                     ↪  perc =
                     ↪  mq->MQPercentage();
                    cout << "CO2: " <<
                     ↪  perc["GAS_CO2"] <<
                     ↪  " ppm\n";
                    string CO2 =to_string
                     ↪  (perc["GAS_CO2"]);
                    std::replace(
                     ↪  CO2.begin(),
                     ↪  CO2.end(), '.',
                     ↪  ',');
                    string command = "curl
                     ↪  https://docs.google.
                     ↪  com/forms/d/" +
                     ↪  formid +
                     ↪  "/formResponse -d
                     ↪  ifq -d
                     ↪  \"entry.916980903=
                     ↪  +" + CO2 + "\" -d
                     ↪  submit=Submit;";
                    system
                     ↪  (command.c_str());
                    sleep(60.0);
                    data[count + i] = val;
                }
                count += ch_len * vBlocks;
                if (microDelay > 0) {
                    usleep(microDelay);
                }
            }
        loop_done:
            if (fd)
                close(fd);
            if (rx)
                free(rx);
            if (tx)
                free(tx);
            if (tr)
                free(tr);
        }
        return 0;
    }
```