

Monitoramento da Quantidade de Carbono no Ar - Carby

Aline Rosa dos Santos Rocha¹, 16/0023076, Sofia Consolmagno Fontes², 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 odontológico 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 contém 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 precisaria de mais ventilação [7]. Segundo a Resolução ANVISA nº 9/2003, o valor máximo

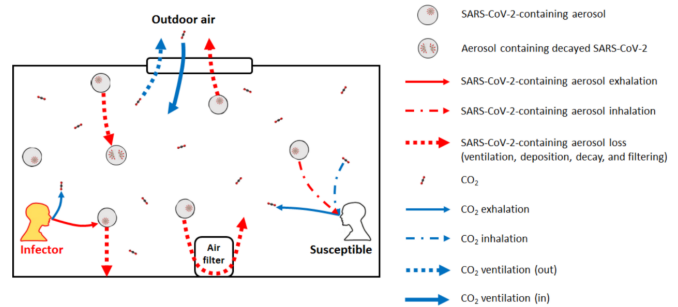


Fig. 1. Esquema da ilustração da expiração, inalação e outros processos de perda de SARS-CoV-2 contendo aerossóis em um ambiente interno. Fonte: Exhaled CO_2 as COVID-19 infection risk proxy for different indoor environments and activities [5]

recomendável para um ambientes climatizados de uso público e coletivo é de 1000 ppm [8].

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 CO_2 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 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.

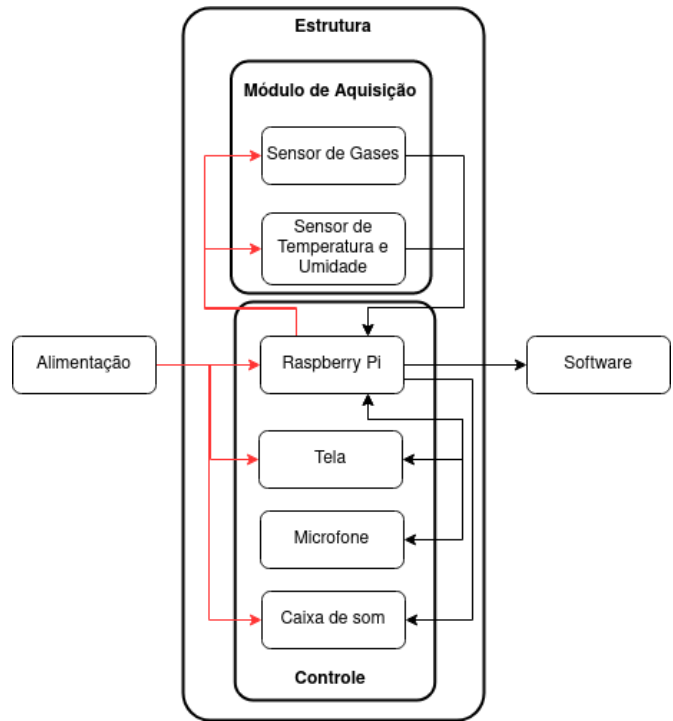


Fig. 2. Divisão das áreas de trabalho do protótipo

D. Software

O software indica ao usuário por meio de avisos quando é necessário realizar a 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 Assistant 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 CO_2 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 funcionalidades existentes. Consequentemente, essa análise pode ser utilizada como referência para criar estratégias 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 Hardware

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.

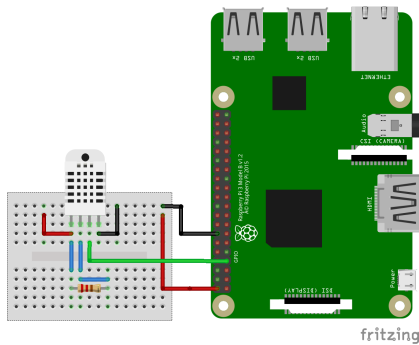


Fig. 4. Conexão sensor DHT11 com Raspberry Pi 3b+

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.

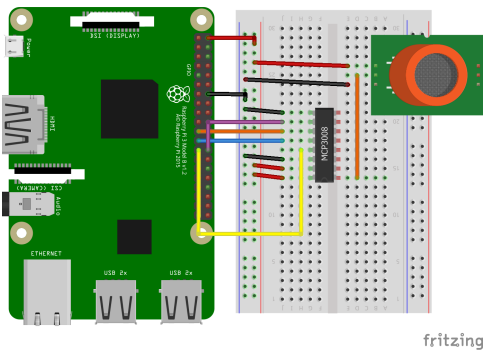


Fig. 5. Conexão sensor MQ135 com Raspberry Pi 3b+

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 20KΩ [13].

Fundamentando no gráfico da figura 6, para realizar a calibração e da leitura do sensor é necessário marcar dois

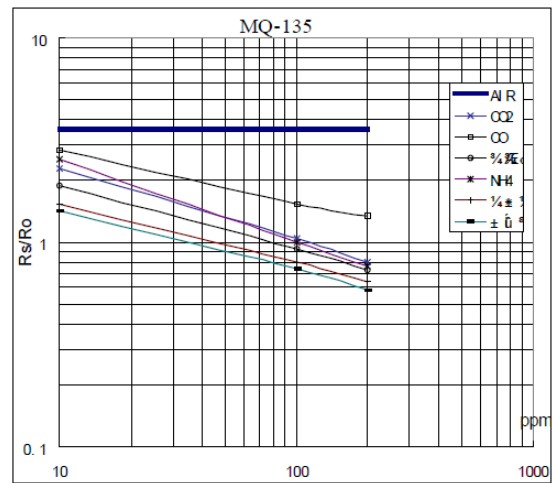


Fig. 6. Características de sensibilidade do sensor de gases MQ-135

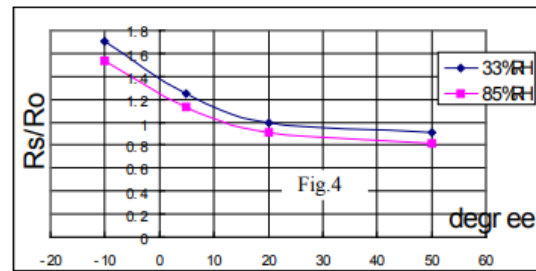


Fig. 7. Típica dependência do sensor MQ-135 da temperatura e umidade.

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 $x_2 = 200$:

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

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

Além do mais, para a correta calibração foi necessário deixar o sensor funcionando 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 4) 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

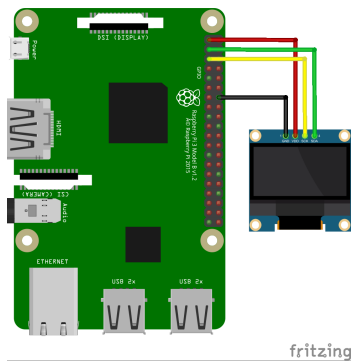


Fig. 8. Conexão display OLED com Raspberry Pi 3b+

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

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 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*.

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 dois formulários do Google, um deles específico 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 ocorreram 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 haverão 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.

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 CO_2 , a umidade e a temperatura, ser alertado quando for necessário fazer a troca de ar do ambiente, ser alertado quando a troca de ar tiver sido concluída e a concentração de CO_2 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 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 isso, são customizadas ações de conversação, por meio da aplicação do Google *Dialogflow*, é parametrizado as variáveis temperatura, umidade e concentração de CO_2 . 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, na função *welcome* já existente, é categorizada para que, quando o usuário envia uma temperatura presente na lista de valores da planilha, o código compara com o parâmetro e assim é possível adquirir os outros valores de umidade e concentração para a temperatura digitada. Sendo assim, para o projeto final, esses dados do arquivo JSON serão integrados no *Actions Console do Google*

Em seguida, deseja-se que o *Google Assistant* seja capaz de estabelecer uma comunicação de forma que o usuário possa requisitar por comando de voz os dados dos sensores. 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 9. 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 requeridos. Os dados são retornados para o Google Assistant que, por sua vez, envia para o usuário por comando de voz. As ações estão feitas e só precisam de uma integração com o Google Assistant na Raspberry.

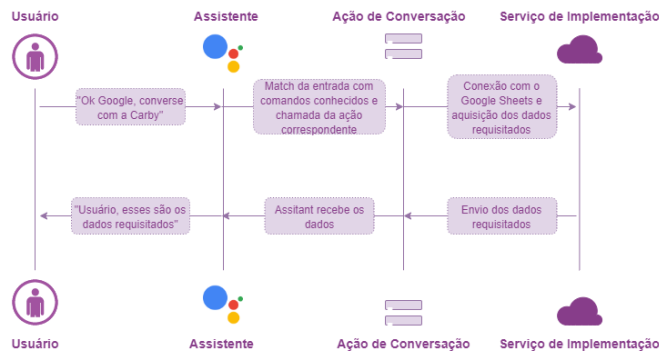


Fig. 9. Fluxograma das ações customizadas do Google Assistant

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.
- 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 10.

```
! sim.yaml X
myproject > custom > intents > pt-BR > ! sim.yaml
1 trainingPhrases:
2   - sim
3   - ok
4   - claro
5   - com certeza

! nao.yaml X
myproject > custom > intents > pt-BR > ! nao.yaml
1 trainingPhrases:
2   - não
3   - agora não
4   - não obrigado
```

Fig. 10. 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 11, o mecanismo *Natural Language Understanding (NLU)* o extrai como um parâmetro digitado, para que você possa processá-lo em uma cena.

```
! available_options.yaml X
myproject > custom > types > pt-BR > ! available_options.yaml
1 synonym:
2   entities:
3     temperatura:
4       synonyms:
5         - temperatura
6         - clima
7     umidade:
8       synonyms:
9         - umidade
10        - aquosidade
11     carbono:
12       synonyms:
13         - carbono
14         - concentração de CO2
15         - concentração de dióxido de carbono
16 # matchType: EXACT_MATCH
```

Fig. 11. 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

duas cenas, a cena de *Start* e a cena *Fortune*. A cena *Start* tem a função de perguntar ao usuário se ele quer saber algum outro dado (Temperatura, Umidade e Concentração de CO₂). Já a cena *Fortune* 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 *Dialogflow* com a API do *Google Assistant*.

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

IX. RESULTADOS

1) **Comunicação sensor de umidade e temperatura com a Raspberry:** A figura 12 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.



Line	Temperatura	Umidade	Concentração de CO ₂
1	27,0	42,0	0
2	27,1	42,0	0
3	27,0	42,0	0
4	27,1	41,0	0
5	27,4	41,0	0
6	27,1	41,0	0
7	27,1	41,0	0
8	27,1	41,0	0
9	27,3	41,0	0
10	27,1	41,0	0
11	27,1	42,0	0
12	27,1	42,0	0

Fig. 12. Arquivo data base do sensor DHT11.

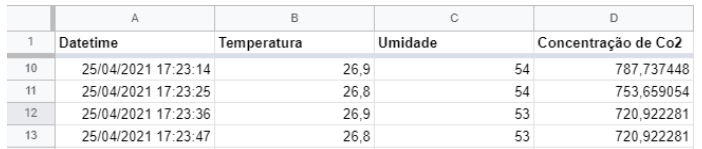
2) **Comunicação sensor de gases com a Raspberry:** A figura 13 mostra o resultado do teste de comunicação do sensor MQ-135 para diferentes concentrações de CO₂. Assim, os testes foram realizados em um ambiente aberto, e tem-se a comparação dos valores obtidos no caso 1, no qual o sensor fica distante da respiração de uma pessoa, resultando em uma baixa concentração de CO₂. Enquanto que no caso 2, assim que o sensor é aproximado de uma pessoa respirando, é possível visualizar o aumento na concentração de CO₂. 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. 13. Leitura do sensor MQ-135 com variação da concentração de CO₂.

3) **Integração sensor de gases, sensor temperatura e umidade com a Raspberry e envio dos dados para o Google Sheets:** 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.

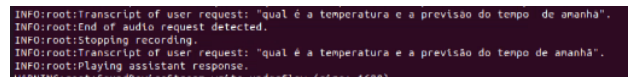
Os dados de temperatura, umidade e concentração ficam armazenados no *Google Sheets* conforme a figura 14.



	A	B	C	D
1	Datetime	Temperatura	Umidade	Concentração de Co2
10	25/04/2021 17:23:14	26,9	54	787,737448
11	25/04/2021 17:23:25	26,8	54	753,659054
12	25/04/2021 17:23:36	26,9	53	720,922281
13	25/04/2021 17:23:47	26,8	53	720,922281

Fig. 14. Tabela no Google Sheets com os dados de umidade, temperatura e concentração de CO₂.

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 ananias".
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 ananias".
INFO:root:Playing assistant response.
```

Fig. 15. Terminal rodando o *Google Assistant*



Fig. 16. Painel da API *Google Assistant*

Na figura 15, 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 16, 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 17:

5) **Gráfico dos dados de monitoramento - Excel:** Foi realizada uma interface para o usuário verificar a concentração de CO₂, temperatura e umidade ao longo do dia conforme apresentado na figura 18. Para o próximo ponto de controle, a base de dados, os gráficos e os *labels* superiores serão atualizados de forma automática.

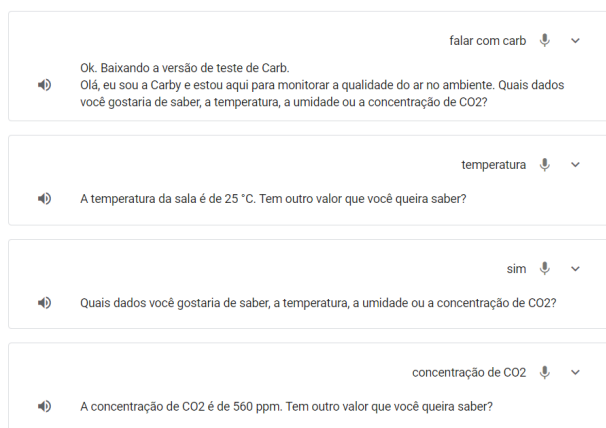


Fig. 17. Simulador do Actions Console

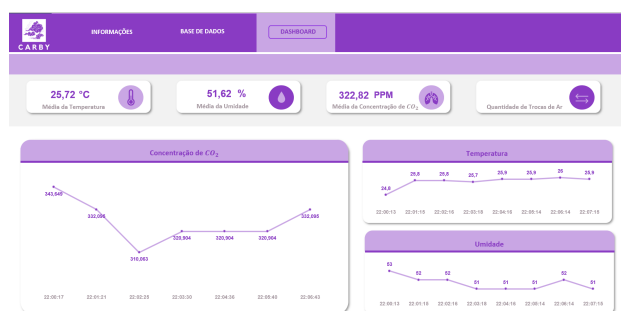


Fig. 18. Interface com o usuário

X. CONSIDERAÇÕES FINAIS

As leituras dos sensores estão na linguagem C e integradas em um único código. O Google Assistant está instalado e em funcionamento na Raspberry, bem como suas ações de conversação customizadas. O gráfico com os dados da concentração de CO2 está disponível. Portanto, para a entrega final do projeto as ações de conversação que estão na API do Google serão devidamente conectadas com a tabela do Google Sheets e integradas com o Google Assistant e o módulo físico será concluído.

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 ufmg, 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/>

APÊNDICE

a) Comunicação sensor DHT11 e sensor MQ-135 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

string x, y;

string formid =
"1_q2E8SroEc500Qh54185
XWK4mJrh9U7zLE_mN11dT_I";
```



```

35 static unsigned cleanupPin = UINT_MAX;
36 static bool verbose = false;
37
38 int read_dht11(unsigned pin) {
39     gpioSetMode(pin, PI_OUTPUT);
40     gpioDelay(19 * 1000);
41     gpioSetMode(pin, PI_INPUT);
42     return 0;
43 }
44
45 static void cleanup(void) {
46     if (verbose) {
47         fprintf(stderr, "...
48             ↳ cleanup()\n");
49     }
50     if (cleanupPin != UINT_MAX) {
51         gpioSetPullUpDown(cleanupPin,
52             ↳ PI_PUD_OFF);
53     }
54     gpioTerminate();
55 }
56
57 enum pulse_state { PS_IDLE = 0,
58     ↳ PS_PREAMBLE_STARTED, PS_DIGITS };
59
60 static void pulse_reader(int gpio, int
61     ↳ level, uint32_t tick) {
62     ofstream database;
63     database.open ("database.csv",
64         ↳ std::ofstream::out |
65         ↳ std::ofstream::app);
66     static uint32_t lastTick = 0;
67     static enum pulse_state state =
68         ↳ PS_IDLE;
69     static uint64_t accum = 0;
70     static int count = 0;
71     uint32_t len = tick - lastTick;
72     lastTick = tick;
73
74     switch (state) {
75     case PS_IDLE:
76         if (level == 1 && len > 70 &&
77             ↳ len < 95) {
78             state = PS_PREAMBLE_STARTED;
79         }
80         else {
81             state = PS_IDLE;
82         }
83         break;
84     case PS_PREAMBLE_STARTED:
85         if (level == 0 && len > 70 && len <
86             ↳ 95) {
87             state = PS_DIGITS;
88             accum = 0;
89             count = 0;
90         } else state = PS_IDLE;
91     }

```

```

82 break;
83     case PS_DIGITS:
84         if (level == 1 && len >= 35 && len
85             ↳ <= 65);
86         else if (level == 0 && len >= 15 &&
87             ↳ len <= 35) {
88             accum <<= 1;
89             count++;
90         }
91         else if (level == 0 && len >= 60 &&
92             ↳ len <= 80) {
93             accum = (accum << 1) + 1;
94             count++;
95         }
96         else {
97             state = PS_IDLE;
98         }
99
100         if (count == 40) {
101             state = PS_IDLE;
102
103             uint8_t parity = (accum & 0xff);
104             uint8_t tempLow = ((accum>>8) &
105                 ↳ 0xff);
106             uint8_t tempHigh = ((accum>>16)
107                 ↳ & 0xff);
108             uint8_t humLow = ((accum>>24) &
109                 ↳ 0xff);
110             uint8_t humHigh = ((accum>>32) &
111                 ↳ 0xff);
112
113             uint8_t sum = tempLow + tempHigh
114                 ↳ + humLow + humHigh;
115             bool valid = (parity == sum);
116
117             if (valid) {
118                 printf("{\"Temperatura\":
119                     ↳ %d,%d, \"Umidade\":
120                     ↳ %d,%d}\n", tempHigh,
121                     ↳ tempLow, humHigh, humLow);
122                 string temp =
123                     ↳ to_string(tempHigh) + ',' +
124                     ↳ + to_string(tempLow);
125                 string humi =
126                     ↳ to_string(humHigh) + ',' +
127                     ↳ + to_string(humLow);
128                 x = temp;
129                 y = humi;
130                 database << temp << ';' << humi
131                     ↳ << '\n';
132             }
133         }
134         break;
135     }
136     if (verbose) {

```

```

121     printf("pulse %c %4uµS state = %d
122         ↳ digits = %d\n", (level == 0 ?
123         ↳ 'H' : (level == 1 ? 'L' :
124         ↳ 'W')), len, state, count);
125 }
126 }
127
128 int selectedChannels[MAX_ADC_CH];
129 int channels[MAX_ADC_CH];
130 char spidev_path[] = "/dev/spidev0.0";
131 const int blocksDefault = 1;
132 const int channelDefault = 0;
133 const int samplesDefault = 100;
134 const int freqDefault = 0;
135 const int clockRateDefault = 3600000;
136 const int coldSamples = 1000;
137
138 vector<double> CO2Curve {2.0, 0.004,
139     ↳ -0.34};
140
141 class MQ {
142 public:
143     int MQ_PIN = 0;
144     int RL_VALUE = 20;
145     double RO_CLEAN_AIR_FACTOR = 3.8;
146     int val;
147
148     int CALIBRATION_SAMPLE_TIMES = 50;
149     int CALIBRATION_SAMPLE_INTERVAL =
150         ↳ 500;
151
152     int READ_SAMPLE_INTERVAL = 50;
153     int READ_SAMPLE_TIMES = 5;
154     int GAS_CO2 = 0;
155     int Ro;
156     int analogPin;
157     MQ() : Ro(120), analogPin(0) {}
158     MQ(int _ro, int _analogPin) {
159         Ro = _ro;
160         MQ_PIN = _analogPin;
161         cout << "Calibrating..." << '\n';
162         Ro = MQCalibration(MQ_PIN);
163         cout << "Calibration is done..."
164             ↳ << '\n';
165         cout.precision(3);
166         cout << "Ro= " << Ro << '\n';
167     }
168
169     double MQResistanceCalculation(int
170         ↳ raw_adc) {
171         return double(RL_VALUE * (1023.0 -
172             ↳ raw_adc) / double(raw_adc));
173     }
174
175     double MQCalibration(int mq_pin) {
176         double value = 0.0;

```

```

167     for (int i = 0; i <
168         ↳ CALIBRATION_SAMPLE_TIMES; i++)
169     {
170         value +=
171             ↳ MQResistanceCalculation(val);
172         sleep(CALIBRATION_SAMPLE_
173             ↳ INTERVAL/1000.0);
174     }
175     value /= CALIBRATION_SAMPLE_TIMES;
176     value /= RO_CLEAN_AIR_FACTOR;
177     return value;
178 }
179
180 double MQRead(int mq_pin) {
181     double rs = 0.0;
182     for (int i = 0; i <
183         ↳ READ_SAMPLE_TIMES; i++) {
184         rs +=
185             ↳ MQResistanceCalculation(val);
186         sleep(READ_SAMPLE_INTERVAL /
187             ↳ 1000.0);
188     }
189     rs /= READ_SAMPLE_TIMES;
190     return rs;
191 }
192
193 double MQGetGasPercentage(double
194     ↳ rs_ro_ratio, int gas_id) {
195     if (gas_id == GAS_CO2) {
196         return
197             ↳ MQGetPercentage(rs_ro_ratio,
198                 ↳ CO2Curve);
199     }
200     return 0;
201 }
202
203 double MQGetPercentage(double
204     ↳ rs_ro_ratio, const
205     ↳ vector<double>& pcurve) {
206     return (pow(10,
207         ↳ (((log(rs_ro_ratio) -
208             ↳ pcurve[1]) / pcurve[2]) +
209             ↳ pcurve[0])));
210 }
211
212 map<string, double> MQPercentage() {
213     map<string, double> val;
214     double read = MQRead(MQ_PIN);
215     val["GAS_CO2"] =
216         ↳ MQGetGasPercentage(read / Ro,
217             ↳ GAS_CO2);
218     return val;
219 }
220 };
221
222 int main(int argc, char *argv[]) {

```

```

207 unsigned pin = DHTPIN;
208 if (gpioInitialise() ==
    ↳ PI_INIT_FAILED) {
209     fprintf(stderr, "failed to
    ↳ initialize GPIO\n");
210     exit(EXIT_FAILURE);
211 }
212 atexit(cleanup);
213 gpioSetMode(pin, PI_INPUT);
214 gpioSetPullUpDown(pin, PI_PUD_UP);
215 gpioWrite(pin, 0);
216 cleanupPin = pin;
217 gpioSetWatchdog(pin, 50);
218 MQ *mq = new MQ();
219 while (true) {
220     int i, j;
221     int ch_len = 0;
222     int vSamples = samplesDefault;
223     double vFreq = freqDefault;
224     int vClockRate =
    ↳ clockRateDefault;
225     int vBlocks = blocksDefault;
226
227     if (ch_len == 0) {
228         ch_len = 1;
229         channels[0] =
    ↳ channelDefault;
230     }
231     int microDelay = 0;
232     if (vFreq != 0) {
233         microDelay = 1000000 /
    ↳ vFreq;
234     }
235     int count = 0;
236     int fd = 0;
237     int val;
238     struct timeval start;
239     int *data;
240     data = (int*)malloc(ch_len *
    ↳ vSamples * sizeof(int));
241     struct spi_ioc_transfer *tr =
    ↳ 0;
242     unsigned char *tx = 0;
243     unsigned char *rx = 0;
244     tr = (struct spi_ioc_transfer
    ↳ *)malloc(ch_len * vBlocks
    ↳ * sizeof(struct
    ↳ spi_ioc_transfer));
245     if (!tr) {
246         perror("malloc");
247         goto loop_done;
248     }
249     tx = (unsigned char
    ↳ *)malloc(ch_len * vBlocks
    ↳ * 4);
250     if (!tx) {

```

```

251         perror("malloc");
252         goto loop_done;
253     }
254     rx = (unsigned char
    ↳ *)malloc(ch_len * vBlocks
    ↳ * 4);
255     if (!rx) {
256         perror("malloc");
257         goto loop_done;
258     }
259     memset(tr, 0, ch_len * vBlocks
    ↳ * sizeof(struct
    ↳ spi_ioc_transfer));
260     memset(tx, 0, ch_len *
    ↳ vBlocks);
261     memset(rx, 0, ch_len *
    ↳ vBlocks);
262     for (i = 0; i < vBlocks; i++)
    ↳ {
263         for (j = 0; j < ch_len;
    ↳ j++) {
264             tx[(i * ch_len + j) *
    ↳ 4] = 0x60 |
    ↳ (channels[j] <<
    ↳ 2);
265             tr[i * ch_len +
    ↳ j].tx_buf =
    ↳ (unsigned
    ↳ long)&tx[(i *
    ↳ ch_len + j) * 4];
266             tr[i * ch_len +
    ↳ j].rx_buf =
    ↳ (unsigned
    ↳ long)&rx[(i *
    ↳ ch_len + j) * 4];
267             tr[i * ch_len + j].len
    ↳ = 3;
268             tr[i * ch_len +
    ↳ j].speed_hz =
    ↳ vClockRate;
269             tr[i * ch_len +
    ↳ j].cs_change = 1;
270         }
271     }
272     tr[ch_len * vBlocks -
    ↳ 1].cs_change = 0;
273     fd = open(spidev_path,
    ↳ O_RDWR);
274     if (fd < 0) {
275         perror("open()");
276         printf("%s\n",
    ↳ spidev_path);
277         goto loop_done;
278     }
279     while (count < coldSamples) {

```

```

280         if (ioctl(fd,
315             ↪ SPI_IOC_MESSAGE(ch_len
281             ↪ * vBlocks), tr) < 0) {
282             perror("ioctl");
283             goto loop_done;
284         }
285         count += ch_len * vBlocks;
286     }
287     count = 0;
288     if (gettimeofday(&start, NULL)
316     ↪ < 0) {
317         perror("gettimeofday:
318             ↪ start");
319         return 1;
290     }
291     while (count < ch_len *
320     ↪ vSamples) {
321         if (ioctl(fd,
322             ↪ SPI_IOC_MESSAGE(ch_len
323             ↪ * vBlocks), tr) < 0) {
324             perror("ioctl");
325             goto loop_done;
293         }
294         for (i = 0, j = 0; i <
326             ↪ ch_len * vBlocks; i++,
327             ↪ j += 4) {
297             val = (rx[j + 1] << 2)
328             ↪ + (rx[j + 2] >>
329             ↪ 6);
298             mq->val = val;
299             data[count + i] = val;
300         }
301         count += ch_len * vBlocks;
302         if (microDelay > 0) {
303             usleep(microDelay);
304         }
305     }
306     loop_done:
307     map<string, double> perc =
308     ↪ mq->MQPercentage();
309     cout << "CO2: " <<
310     ↪ perc["GAS_CO2"] << " ppm\n";
311     string CO2 =
312     ↪ to_string(perc["GAS_CO2"]);
313     replace(CO2.begin(), CO2.end(),
314     ↪ '.', ',');
315     gpioSetAlertFunc(pin,
316     ↪ pulse_reader);
317     read_dht11(pin);
318     if (x != "" && y != "") {
319         string command = "curl
320         ↪ https://docs.google.com/

```

```

forms/d/" + formid +
↪ "/formResponse -d ifq -d
↪ "\"entry.1076682711= +\" + x
↪ + "\" -d
↪ "\"entry.1505376468= + \" +
↪ y + "\" -d
↪ "\"entry.1031950862= \" +
↪ CO2 + "\" -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;
}

```