Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Programa de Graduação em Engenharia de Instrumentação, Automação e Robótica

CAPUZ INTELIGENTE USANDO PROTOCOLO DE COMUNICAÇÃO PROPRIETÁRIO

LUANA DE OLIVEIRA SOSTIZZO

Santo André, Junho de 2021

LUANA DE OLIVEIRA SOSTIZZO

CAPUZ INTELIGENTE USANDO PROTOCOLO DE COMUNICAÇÃO PROPRIETÁRIO

Monografia apresentada ao Curso de Engenharia de Instrumentação, Automação e Robótica da Universidade Federal do ABC, como parte dos requisitos para obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Prof. Drº Mario Alexandre Gazziro

Santo André – SP

2021

Sistema de Bibliotecas da Universidade Federal do ABC
Elaborada pelo Sistema de Geração de Ficha Catalográfica da UFABC
com os dados fornecidos pelo(a) autor(a).

Sostizzo, Luana de Oliveira
Capuz inteligente usando protocolo de comunicação proprietário / Luana de Oliveira Sostizzo. — 2021.

57 fls. : il.

Orientador: Mario Alexandre Gazziro

Trabalho de Conclusão de Curso — Universidade Federal do ABC, Bacharelado em Engenharia de Instrumentação, Automação e Robótica, Santo André, 2021.

1. Aplicativo mobile. 2. EPI. 3. Microcontroladores. 4. Sensores. 5. Sistemas embarcados. I. Gazziro, Mario Alexandre. II. Bacharelado em Engenharia de Instrumentação, Automação e Robótica, 2021. III. Título.

LUANA DE OLIVEIRA SOSTIZZO

CAPUZ INTELIGENTE USANDO PROTOCOLO DE COMUNICAÇÃO PROPRIETÁRIO

Trabalho aprovado em: 12/07/2021

Banca Examinadora:

Mario Alexandre Gazziro
Orientador

Filipe Ieda Fazanaro
Avaliador

Santo André – SP

2021

AGRADECIMENTOS

Agradeço ao meu orientador Mario Gazziro pela direção da definição do tema do projeto, pela oportunidade e pelo apoio durante todo o processo de construção do mesmo. Agradeço também ao André Mozeto por toda experiência trocada no projeto no que diz respeito aos materiais utilizados para a construção do capuz utilizado no projeto, assim como suas considerações a respeito dos ambientes e locais para quais o capuz melhor atenderia a necessidade de proteção. Por todo auxílio e desenvolvimento no âmbito de codificação, também agradeço ao colega Vinicius Azeka.

“É precisamente na fronteira do conhecimento que a imaginação tem seu papel mais importante; o que ontem foi apenas um sonho, amanhã poderá se tornar realidade.” (Marcelo Gleiser).

RESUMO

Dada a elevada infectividade de seu agente etiológico, o coronavírus SARS-CoV-2, aliada à ausência de imunidade prévia na população humana e à inexistência de vacina, fazem com que o crescimento do número de casos seja exponencial, se não forem tomadas medidas para deter sua transmissão. Tendo esse aspecto em mente, o presente projeto visou elaborar um equipamento de proteção individual (EPI), o qual deve trazer todas as garantias de proteção biológica para as pessoas comuns, sem a necessidade de treinamentos extensivos para esse tipo de equipamento. Com a proposta de desenvolver um equipamento guiado por aplicativo de monitoramento em *smartphone* (a fim de orientar o usuário sobre as trocas de componentes perecíveis, garantindo sua segurança e proteção biológica em meios potencialmente contaminados), foram construídas as eletrônicas embarcadas desse equipamento, assim como seus *softwares* de controle, incluindo um aplicativo para *smartphone*. Dessa forma, foi elaborado com sucesso um equipamento para monitorar e auxiliar indivíduos leigos à utilização de um dispositivo EPI avançado.

Palavras-chave: aplicativo *mobile*; EPI; microcontroladores; sensores; sistemas embarcados.

ABSTRACT

Given the high infectivity of its etiological agent, the SARS-CoV-2 coronavirus, combined with the absence of previous immunity in the human population and the lack of vaccine, cause the growth in the number of cases to be exponential, if no measures are taken to stop your transmission. Keeping this aspect in mind, this project aimed to develop a personal protective equipment (PPE), which should bring all the guarantees of biological protection for common people, without the need for extensive training for this type of equipment. With the proposal to develop a device guided by a smartphone monitoring application (in order to guide the user on the exchange of perishable components, ensuring their safety and biological protection in potentially contaminated places), the embedded electronics of this equipment were built, as well as their control, including a smartphone app. Thus, a device was successfully developed to monitor and help lay individuals to use an advanced PPE device.

Keywords: Embedded systems; microcontrollers; mobile app; PPE; sensors.

LISTA DE SIGLAS

EPI	-	Equipamento de Proteção Individual
IDE	-	*Integrated Development Environment*
MCU	-	*Microcontroller Central Unit*
BT	-	*Bluetooth*
ER	-	Expressão Regular (do inglês, *Regular Expression*)
CO_2	-	Dióxido de carbono
NBR	-	Norma brasileira
SMD	-	*Surface Mounted Device*
SRAM	-	*Static Random-Access Memory*
EEPROM	-	*Electrically-Erasable Programmable Read-Only Memory*
BAUDS	-	*Baud Rate*
QR	-	*Quick Response*
PFF	-	Peças Faciais Filtrantes
PPM	-	Parte por milhão

LISTA DE TABELAS

Tabela I – Listagem de custo dos componentes da eletrônica embarcada……..27

Tabela II – Símbolos interpretados pela biblioteca *CompactRegex* e suas funções…..30

Tabela III – Símbolos interpretados pela biblioteca *RE* e suas funções……...32

Tabela A-I – Tempos de carga e descarga da bateria…….41

Tabela B-I – Resultado do conversor com entrada de 5 volts……...42

Tabela B-II – Resultado do conversor com entrada de 12 volts.……43

LISTA DE FIGURAS

Figura 1 – Exemplo de traje EPI de pressão positiva para o corpo todo……..16

Figura 2 – Produto *BioVYZR – Personal Air-Purifying Shield* da *startup VYZR Technologies*……..…….18

Figura 3 – Diagrama de blocos do “Capuz Inteligente” *Hoodi*, apresentando todos os componentes do sistema, incluso a eletrônica embarcada e aplicativo de monitoramento customizado para *smartphone*……..21

Figura 4 – Dispositivos físicos do “Capuz Inteligente”. (a) Capuz descartável em tecido. (b) Eletrônica embarcada e filtro. (c) Estrutura de suporte. (d) *Smartphone* de monitoramento. (e) Interação entre eletrônica embarcada e *smartphone v*ia *Bluetooth..*22

Figura 5 – Capuz descartável em tecido laminado, com visor de poliéster cristal e costura em nylon convencional……23

Figura 6 – Esquemático da eletrônica embarcada do equipamento……..26

Figura 7 – Componentes da eletrônica embarcada……..27

Figura 8 – Detalhamento dos componentes do módulo de monitoramento, e sua interconexão com o microcontrolador de controle……28

Figura 9 – Protocolo elaborado para intercâmbio de dados entre a eletrônica embarcada e o aplicativo do *smartphone*, usando expressões regulares…….29

Figura 10 – Teste, no PC, da biblioteca *CompactRegex* através do IDE do Arduino….31

Figura 11 – Teste online de expressões regulares……...32

Figura 12 – Interface do aplicativo apresentando os níveis de bateria, dióxido de carbono e as validades do filtro e do capuz……...33

Figura 13: Codificação dos códigos *QR* com preâmbulos e números de série para identificação única dos itens de consumo no equipamento……..34

Figura 14: Montagem final do módulo de monitoramento……..35

Figura 15: Demonstração do monitor de CO2……..36

Figura 16: Demonstração da leitura de um código *QR…….36*

Figura A-1: Tempos de carga e descarga da bateria……...41

Figura B-1: Conversor utilizando 5 volts de entrada……...42

Figura B-2: Conversor utilizando 12 volts de entrada…….43

SUMÁRIO

1 Introdução……14
1.1 Objetivos……15
1.2 Justificativa……15
1.3 Organização do trabalho……15
2 Tecnologias Relacionadas……16
2.1 Trajes EPI……16
2.1.1 Projetos relacionados no mercado……17
2.2 Sistemas embarcados……18
2.3 Linguagem de programação: C e *Python*……18
2.4 Expressões regulares……19
2.5 Quick Response (QR) Code……20
3 Metodologia e Implementação……21
3.1 Visão geral……21
3.1.1 Capuz em tecido……22
3.1.2 Filtro biológico……23
3.1.3 Bateria……23
3.1.4 Módulo de potência……24
3.1.4.1Turbina……24
3.1.4.2 Conversor DC/DC……24
3.1.5 Módulo de monitoramento……24
3.1.5.1 Microcontrolador Arduino……25
3.1.5.2 Sensor de CO_2 ……25
3.1.5.3 Módulo de Bluetooth……25
3.1.6 Aplicativo *mobile*……25
3.2 Eletrônica embarcada……25
3.2.1 Módulo de potência……28
3.2.2 Módulo de monitoramento……28
3.3 Protocolo proprietário de comunicação……29

3.3.1 Controle do protocolo no Arduino……..…….30
3.3.2 Controle do protocolo no aplicativo do smartphone…….……..……...32
3.4 Detalhamento do aplicativo *mobile*…….……..……..33
3.4.1 Funcionalidades…….……..…….……..……...33
3.4.2 Substituição do filtro e capuz com QR Code…….……..…..….34
4 Resultados…….……..…….……...…..35
5 Conclusões…….……..…….…….37
6 Trabalhos futuros…….……..…….……...…..38
Apêndice A – Testes sobre autonomia da bateria…….……...…..…41
Apêndice B – Eficiência do conversor DC/DC…….…….……42
Apêndice C – Códigos-fonte…….……..…….……..…….44

1 Introdução

A COVID-19, doença identificada pela primeira vez em Wuhan, China, em dezembro de 2019, propagou-se rapidamente e tornou-se uma pandemia em pouco mais de dois meses. A elevada infectividade de seu agente etiológico, o coronavírus denominado SARS-CoV-2, aliada à ausência de imunidade prévia na população humana e à inexistência de vacina, fazem com que o crescimento do número de casos seja exponencial, se não forem tomadas medidas para deter sua transmissão (KUCHARSKI, 2021).

Já ficou claro que no auge de nossa ciência e tecnologia, e com todo esforço mental e econômico envolvido, criar vacinas tem se mostrado uma tarefa hercúlea para a humanidade, sendo que mesmo hoje em dia os vírus ainda podem causar destruição de vidas e de economias. Certamente muita coisa deverá ser refletida sobre as decisões globais e locais a respeito das maneiras como se deu o enfrentamento da pandemia atual, muitas vezes pautado em ações efetivas, outras em ações paliativas, seguindo a filosofia de que “não é a melhor solução que existe, mas é a melhor que temos no momento” (GAZZIRO, 2021).

Nesse quesito, a adoção imediata do uso de máscaras contribuiu para a frenagem da expansão do vírus, porém, não o parou por completo por oferecer um benefício protetor incerto. Segundo Garcia (2020), o uso da máscara pode dar falsa sensação de proteção e fazer com que as pessoas relaxem a adesão a outras medidas reconhecidamente efetivas, como a lavagem das mãos.

Tendo esse aspecto em mente, foi dado início a um projeto a fim de elaborar um Equipamento de Proteção Individual (EPI), denominado “Capuz Inteligente”, o qual tem por objetivo trazer todas as garantias de proteção biológica (normalmente usufruídas apenas pelos biomédicos profissionais) para as pessoas comuns, sem a necessidade de treinamentos extensivos, e com o máximo de conforto e segurança.

Para tal, devem ser utilizados os chamados sistemas de “pressão positiva” (pressão interna maior que a externa), já bastante comuns em trajes EPI para o corpo inteiro, e devidamente regulados pela norma NBR 14749 (ABNT, 2001). Eles têm sido

uma realidade segura nos meios profissionais, tanto para produção de fármacos que geram subprodutos tóxicos, como para manipulação de agentes biológicos nocivos.

1.1 Objetivos

Desenvolver um equipamento de proteção individual para uso por pessoas sem treinamento biomédico, guiadas por aplicativo de monitoramento em *smartphone*, a fim de orientar o usuário sobre as trocas de componentes perecíveis, garantindo sua segurança e proteção biológica em meios potencialmente contaminados.

1.2 Justificativa

Tornar efetivo um maior nível de proteção à contaminação às pessoas leigas e sem treinamento em proteção biológica, preferencialmente voltado aos indivíduos nos grupos de risco em idade avançada ou que apresentam comorbidades, de forma a tornar suas tarefas diárias, como realizar compras no mercado local ou visitar a farmácia nas proximidades, muito mais seguras do ponto de vista de proteção quanto à contaminação pelo vírus da SARS-CoV-2.

1.3 Organização do trabalho

O estudo está dividido em cinco grandes tópicos. O primeiro se trata de uma breve descrição das tecnologias utilizadas para a construção do EPI que envolvem assuntos relacionados principalmente a sistemas embarcados, linguagens de programação e expressões regulares. O segundo tópico foca na apresentação da metodologia e da implementação do projeto, trazendo uma visão mais detalhada sobre como se deu a divisão e construção dos principais módulos constituintes do EPI. Já o terceiro e o quarto tópico trazem os resultados obtidos na construção do aparato assim como as conclusões obtidas do mesmo, e por fim, o quinto tópico expõe alguns pontos identificados para futuras melhorias.

2 Tecnologias Relacionadas

2.1 Trajes EPI

Segundo a norma do Ministério do Trabalho e Emprego (MTE), Equipamento de Proteção Individual (EPI) é todo dispositivo ou produto, de uso individual utilizado pelo trabalhador, destinado à proteção de riscos suscetíveis de ameaçar a segurança e a saúde no trabalho (NR6, 2001).

Tais dispositivos podem proteger todo o corpo, na forma de trajes completos, ou então apenas partes do corpo, como por exemplo, o uso de capuzes EPI para proteção apenas da cabeça do usuário.

A Figura 1 apresenta um traje completo de proteção EPI, cujo princípio de funcionamento é baseado em pressão positiva, os quais dependem da geração externa de ar comprimido, o qual deve ser ainda filtrado e injetado dentro do traje com um fluxo médio de 170 litros por minuto, de acordo com a norma NBR14749. A necessidade de injeção externa de ar comprimido limita a mobilidade física do indivíduo apenas ao entorno da área de trabalho, e não seria aplicável, nessa forma, aos objetivos do presente trabalho.

Figura 1: Exemplo de traje EPI de pressão positiva para o corpo todo. Fonte: Connex.

2.1.1 Projetos relacionados no mercado

O *BioVYZR Personal Air-Purifying Shield* da *startup* sediada em Toronto *VYZR Technologies* é um produto que utiliza conceitos parecidos aos que serão abordados nesse trabalho e está apresentado na Figura 2. O *BioVYZR* possui um sistema de purificação de ar integrado que filtra o ar que entra e o ar exalado pelo usuário. Também cria um ambiente interno de pressão positiva que impede que o ar externo chegue ao usuário. Além disso, o sistema contém dois ventiladores que duram até 8 horas quando usados com um banco de energia de 10.000 mAh.

Figura 2: Produto *BioVYZR – Personal Air-Purifying Shield* da *startup VYZR Technologies.* Fonte: https://www.vyzrtech.com/products/biovyzr.

2.2 Sistemas embarcados

Nas últimas décadas, houve uma rápida evolução da eletrônica, dos computadores pessoais e dos equipamentos eletrônicos. Nesse cenário, os sistemas computacionais embarcados tornaram-se cada vez mais importantes e presentes no dia a dia. De acordo com Garcia (2021), eles estão presentes em diversos equipamentos do cotidiano, como por exemplo, eletrodomésticos (cafeteira elétrica, micro-ondas, geladeira, máquina de lavar), sistemas de armamentos militares (sistemas bélicos) e sistemas industriais (sistemas de controle, robótica). À luz de Patterson et al. (2013), os sistemas embarcados são considerados como computadores dedicados. Um exemplo de computador dedicado amplamente utilizado em aplicações didáticas são os microcontroladores (MCU) Arduinos.

Como ainda explicado por Garcia (2021), sistemas embarcados também são caracterizados como sistemas reativos, pois a aplicação executada é dependente do ambiente. Portanto, o microcontrolador pode ser visto como uma caixa-preta, em que as entradas são fornecidas ao sistema, processadas e, por fim, geram uma ação, caracterizadas pelas saídas. Nesse cenário, entradas e saídas representam elementos como sensores, atuadores, dispositivos de comunicação, dispositivos de interface gráfica, entre outros.

2.3 Linguagens de programação C e *Python*

De acordo com Monteiro (2021), linguagem de programação é uma forma de escrita formal que especifica um conjunto de instruções e regras usadas para gerar um programa (*software*). Um *software* pode ser desenvolvido para rodar em um computador, dispositivo móvel ou em qualquer equipamento que permita sua execução.

Assim, conforme mencionado por Roveda (2019), a linguagem de programação permite que um programador crie programas a partir de um conjunto de ordens, ações

consecutivas, dados e algoritmos. Esse conjunto faz o controle do comportamento físico e lógico de uma máquina.

Existem diversas linguagens, pois também existem diversas formas de transmitir um mesmo comando para alcançar um mesmo objetivo. A linguagem C foi originalmente projetada e implementada no sistema operacional UNIX, por Dennis Ritchie em 1972. Segundo Kernighan e Ritchie (1978), o C é uma linguagem de programação de propósito geral que apresenta economia de expressão, avançados fluxos de controles e de estruturas de dados, além de um rico conjunto de operadores.

Já a linguagem de programação *Python* foi desenvolvida no início da década de 90 pelo programador e matemático holandês Guido Van Rossum, oferecendo recursos como tipagem dinâmica forte, orientação a objetos, multi-paradigmas (programação funcional e imperativa), além de recursos poderosos em biblioteca padrão via módulos e *frameworks* (MELO, 2020). Possui aplicações em diversas áreas como implementações para inteligência artificial, ciência de dados e aplicativos *mobile*.

2.4 Expressões regulares

Conforme brevemente explicado por Ullman (1986) e Trevisan (2004), uma expressão regular (ER ou *RegEx*) é um método que utiliza alguns caracteres com significado especial para especificar um padrão de texto. O suporte às ERs pode ser realizado em programas fontes escritos com linguagens de programação como C*, Perl, Java, Python* ou *php*.

Com isso, Jargas (1999) explica que as ERs se tornam composições de símbolos, caracteres com funções especiais, que, agrupados entre si, formam uma expressão. Essa expressão é interpretada como uma regra, comumente chamada de mascaramento, que indicará sucesso se uma entrada de dados qualquer obedecer exatamente a todas as suas condições.

Um exemplo didático é a implementação de uma ER para validação de uma entrada de DDD (Discagem Direta a Distância) de número de telefone nos padrões ‘(NN)’ ou ‘NN’, sendo N um valor numérico de 0 a 9. Para isso, a máscara correspondente é a ^(\([0-9]{2}\)|[0-9]{2})$, onde os caracteres ‘^’ e ‘$’ representam,

respectivamente, o início e o fim da ER. O bloco em vermelho abaixo valida um DDD no formato com parênteses – exemplo: (11) – já o bloco de cor azul, valida um DDD no formato sem parênteses, ambos verificando o uso de exatamente dois valores numéricos (sendo essa quantidade, 2, definido pelo número entre chaves).

^ (\([0-9]{2} \) | [0-9]{2}) $

2.5 *Quick Response (QR) Code*

O *QR Code* é um tipo de código de barras em matriz, ou seja, bidimensional, criado pela empresa japonesa *Denso-Wave*, em 1994. Ele é constituído de uma série de códigos e caracteres codificados em uma imagem quadrada, dispondo de uma alta capacidade para armazenar dados que pode ser facilmente escaneado usando a maioria dos celulares com câmera. Esse código é convertido em texto (interativo), um endereço URL, um número de telefone, uma localização georreferenciada, um e-mail, um contato ou um SMS (MACORATTI).

De acordo com Silva (2021), é amplamente usado em diversos países, seja em campanhas publicitárias, cartões de visitas, terminais turísticos, jornais, revistas, rótulos de produtos, programas de TV e em uma infinidade de outros meios.

3 Metodologia e Implementação

3.1 Visão geral

Denominado de *Hoodi* (do inglês *HOOD*, capuz, acrescido da letra I, *intelligent*), é um projeto desenvolvido com o intuito de construir um EPI para uso de pessoas comuns, sem treinamento em sistemas desse tipo.

O *Hoodi* foi nomeado como “capuz inteligente” porque possui funções de monitoramento que se utilizam de dados advindos de sensores, os quais são apresentados numa interface de aplicativo customizado, que por sua vez é executado em um *smartphone.* Sua finalidade é prover auxílio a seus usuários quanto às ações efetivas para sua proteção biológica. Isso é feito através da troca de informações em tempo real entre a eletrônica embarcada no capuz e o *smartphone* de monitoramento, através de comunicação sem fio usando a rede *Bluetooth* (BT).

O conjunto de blocos do sistema que constitui o capuz inteligente *Hoodi* é integrado por: um filtro biológico, um capuz (em tecido), um módulo de potência, um módulo de monitoramento e um aplicativo *mobile* (*app*), conforme exibido na Figura 3.

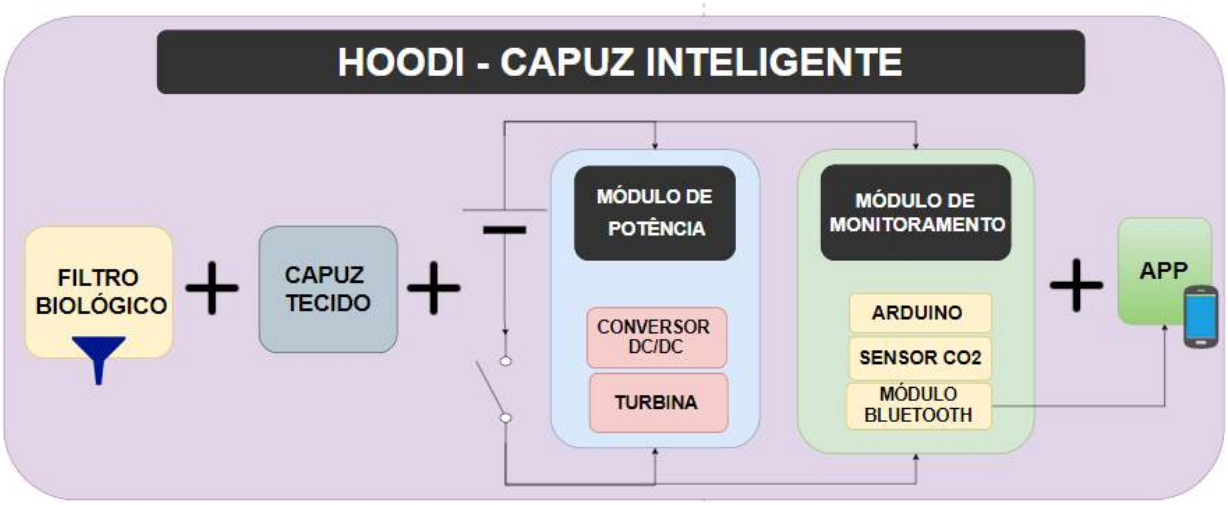

Figura 3: Diagrama de blocos do “Capuz Inteligente” *Hoodi*, apresentando todos os componentes do sistema, incluso a eletrônica embarcada e aplicativo de monitoramento customizado para *smartphone*. Fonte: autoria própria.

Já os dispositivos físicos que constituem o capuz inteligente são representados na Figura 4, a seguir.

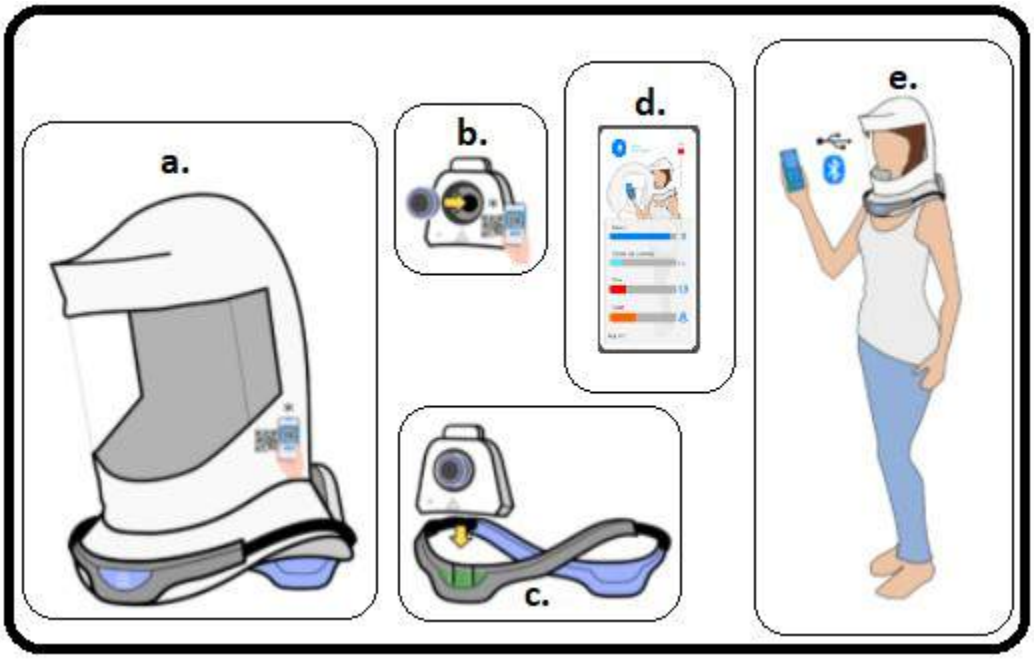

Figura 4: Dispositivos físicos do “Capuz Inteligente”. (a) Capuz descartável em tecido. (b) Eletrônica embarcada e filtro. (c) Estrutura de suporte. (d) *Smartphone* de monitoramento. (e) Interação entre eletrônica embarcada e *smartphone* via *Bluetooth*.
Fonte: André Mozeto, adaptado.

3.1.1 Capuz em tecido

Apresentado nas Figuras 4a e Figura 5, trata-se de um produto de linha da empresa Connex, parceira no desenvolvimento desse projeto, tradicionalmente utilizado em trajes EPI. O capuz é fabricado a partir de TNT laminado, com visor de poliéster cristal e costura em nylon convencional.

Como o objetivo do projeto não é a utilização do capuz em ambientes altamente tóxicos, como indústrias de tintas, por exemplo, o capuz não precisa ser descartado com um único uso, podendo ser higienizado e reutilizado. Porém, mesmo para nossa aplicação mais simples, sua vida útil está estimada em torno de 80 horas de uso, ou 5000 minutos de operação contínua devido a aspectos dos materiais utilizados para a construção do mesmo.

Figura 5: Capuz descartável em tecido laminado, com visor de poliéster cristal e costura em nylon convencional. Fonte: autoria própria.

3.1.2 Filtro biológico

Apresentado na Figura 4b, e representado pelo disco azul na sub-figura, são filtros biológicos que atendem ao padrão PFF3 (Peças faciais filtrantes, nível 3), os quais possuem vida útil muito reduzida, dependendo do nível de contaminação do ambiente, e necessitam ser substituídos com regularidade, para não comprometer a contaminação do usuário. Esses filtros devem ser substituídos a cada 1000 minutos.

3.1.3 Bateria

Alojada dentro do compartimento representado na Figura 4b, fica a bateria recarregável do sistema, a qual deve prover uma autonomia de pelo menos 8 horas de funcionamento ininterruptas, compatível com a duração do turno de trabalho. O recarregador da bateria fica externo ao equipamento, de modo que o módulo da Figura 4b deve ser desacoplado totalmente do equipamento para realizar sua recarga. Os testes com a capacidade de autonomia da bateria estão relatados no Apêndice A.

3.1.4 Módulo de potência

Também alojado dentro do compartimento representado na Figura 4b, é constituído por uma turbina elétrica e seu respectivo conversor DC/DC.

3.1.4.1 Turbina

Trata-se do elemento principal do capuz inteligente, responsável por injetar o ar externo, devidamente filtrado, dentro do capuz, criando então uma pressão positiva, expelindo assim partículas e gotículas do meio ao redor, evitando que essas possam adentrar no espaço interno do capuz trazendo uma potencial carga virótica danosa. As turbinas elétricas necessitam, em geral, de tensão acima de 20 volts (corrente contínua) para operarem devido à alta taxa de rotação dos motores elétricos internos, que demandam esse consumo de tensão.

3.1.4.2 Conversor DC/DC

Uma vez que o sistema é alimentado por bateria, cujo padrão de tensão costuma se encontrar abaixo de 20 volts, e que a turbina necessita de uma maior tensão de operação, faz-se necessário o uso de um conversor DC/DC (corrente contínua para corrente contínua), o qual amplia o nível de tensão da bateria, ao custo de decrementar a sua corrente máxima de saída, inferindo ainda uma pequena perda de eficiência no processo.

Os testes com o conversor estão relatados no Apêndice B, em que são apresentadas as eficiências de desempenho para operação com duas tensões de entrada, respectivamente 5 e 12 volts (sendo 5 volts a menor tensão de entrada possível para garantir uma saída de 24 volts no conversor e atender demanda de consumo da turbina elétrica).

3.1.5 Módulo de monitoramento

Também alojado dentro do compartimento representado na Figura 4b, é constituído por um microcontrolador Arduino, um sensor de CO_2 e um módulo de comunicação *Bluetooth*.

3.1.5.1 Microcontrolador Arduino

Para controlar o módulo de monitoramento, foi escolhido um microcontrolador da família Arduino, modelo Nano, devido à sua facilidade de programação, ao seu tamanho compacto, ao seu baixo consumo de energia e pela diversidade de bibliotecas de *software* disponíveis e exemplos de programação.

3.1.5.2 Sensor de CO_2

Uma vez que, ao utilizar o equipamento, o usuário está sendo privado de usufruir diretamente do oxigênio ao seu redor (ele o faz através do ar filtrado injetado dentro do capuz pela turbina elétrica), faz-se necessário um monitoramento adequado dos níveis de CO_2 dentro do capuz, de forma a gerar um alerta ao usuário caso esse nível ultrapasse o valor recomendado para segurança no trabalho, atualmente definido em 5000 PPM (parte por milhão) de CO_2. O modelo de sensor utilizado foi CCS811 da empresa *SparkFun*.

3.1.5.3 Módulo de *Bluetooth*

A fim de tornar a comunicação entre a eletrônica embarcada e o *smartphone* mais confortável ao usuário, optou-se pela comunicação sem fios, sendo adotado o protocolo *Bluetooth*. Em sendo assim, foi escolhido o módulo modelo HM10, o qual apresenta tanto compatibilidade com ambientes Android (TM) como IOS (TM).

3.1.6 Aplicativo *mobile*

Por fim, a última parte do sistema trata-se do aplicativo *mobile*, o qual é responsável pelo monitoramento do equipamento e também atualização dos registros internos referentes à troca do filtro e à troca do capuz de tecido descartável.

O detalhamento do funcionamento desse aplicativo é realizado na seção 3.4 subsequente.

3.2 Eletrônica embarcada

O esquemático de toda eletrônica embarcada no equipamento é apresentado na Figura 6 a seguir. Note que o módulo de potência (à direita, composto pelo conversor DC/DC XL6009 e pela turbina 24VDC) e o módulo de monitoramento (à esquerda, composto pelo Arduino Nano, Sensor CO_2, módulo *Bluetooth* HM-10, *buzzer* e led) são completamente independentes um do outro, compartilhando apenas a mesma bateria como fonte de energia. Uma chave liga-desliga no terminal positivo da bateria serve para ligar ou desligar ambos os módulos simultaneamente.

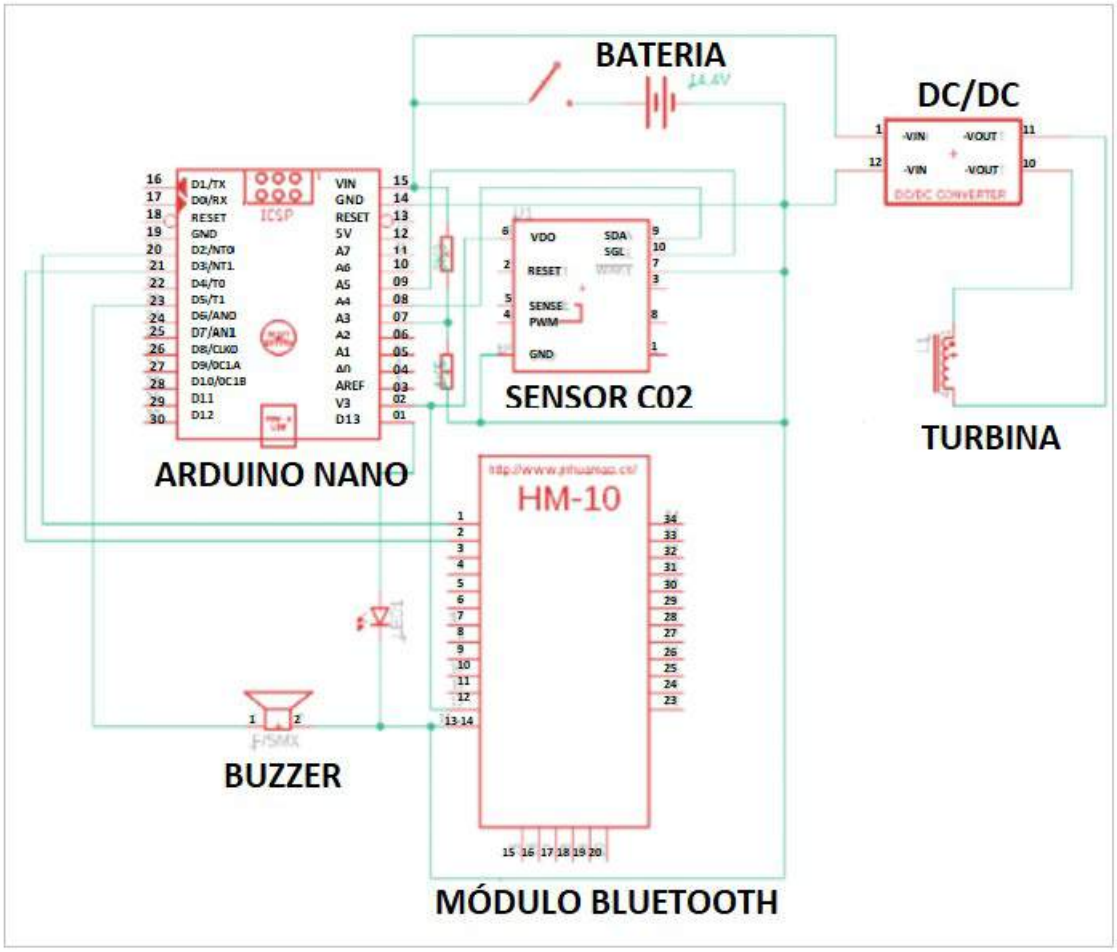

Figura 6: Esquemático da eletrônica embarcada do equipamento. Fonte: autoria própria.

A Figura 7 e a Tabela I apresentam respectivamente os componentes da eletrônica embarcada, e seu respectivo custo, abaixo de R$ 500,00. A área destacada em vermelho na Figura 7 representa os componentes específicos do módulo de monitoramento. Os demais componentes pertencem ao módulo de potência.

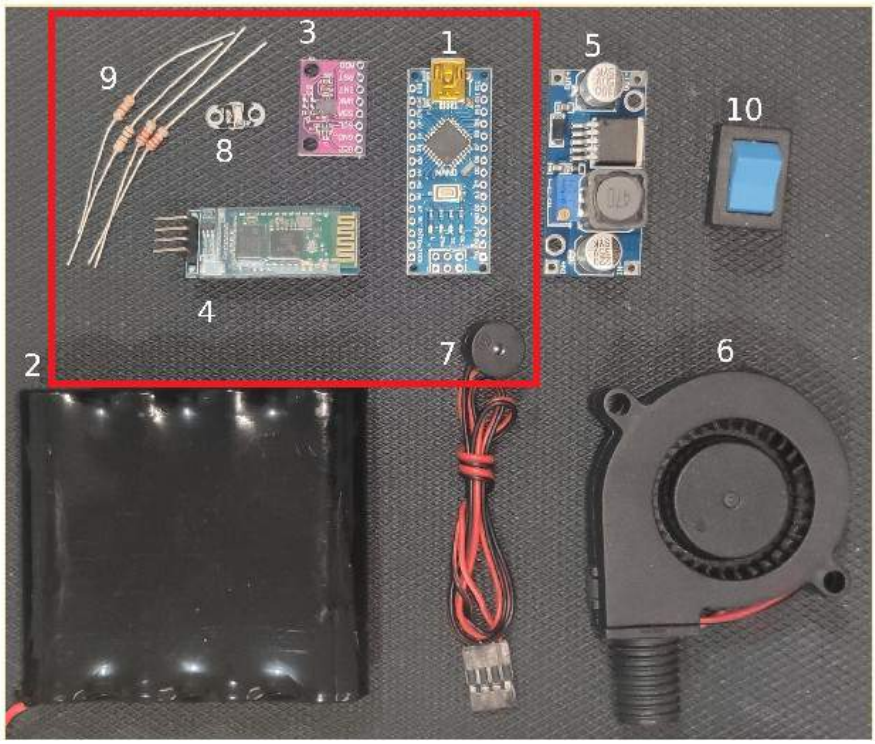

Figura 7: Componentes da eletrônica embarcada. Fonte: autoria própria.

Tabela I: Listagem de custo dos componentes da eletrônica embarcada.

Item	Componente	Qtd.	Preço(R$)
1	Arduino Nano 3.0 Atmega328 328 Ch340	1	39,95
2	Bateria Recarregável 14.4V Li-Ion	1	199,01
3	Sensor Ccs811 Dióxido de Carbono CO_2	1	104,99
4	Módulo *Bluetooth* HM-10	1	49,99
5	Conversos DC/DC Lm2596 3a	1	13,51
6	Turbina Radial 5015 Hotend – 24VDC	1	24,01
7	Buzzer Mini 5V para Arduino	1	15,01
8, 9 e 10	Led SMD, resistores e chave *on-off*	1	5,01
			Total R$ 451,48

Fonte: autoria própria.

3.2.1 Módulo de potência

Composto pela turbina elétrica, conversor DC/DC e pela bateria, esse módulo não depende de nenhuma programação e funciona independente do módulo de monitoramento (bastando ligar a chave liga-desliga para iniciar o funcionamento da turbina). As únicas considerações sobre ele estão relacionadas aos testes de eficiência do conversor DC/DC frente à demanda de consumo de potência acoplada do conversor. Como esses testes são realizados empiricamente em bancada, ajustando um potenciômetro no conversor, optamos por detalhar esse procedimento no Apêndice B, para não quebrar o fluxo de apresentação da eletrônica embarcada nessa seção.

3.2.2 Módulo de monitoramento

O módulo de monitoramento, apresentado na Figura 8, é responsável por: ler o sensor (CO_2), monitorar o nível de bateria, gerar os alertas (sonoros e visuais) e se comunicar via *Bluetooth* com o *smartphone,* através de um protocolo proprietário.

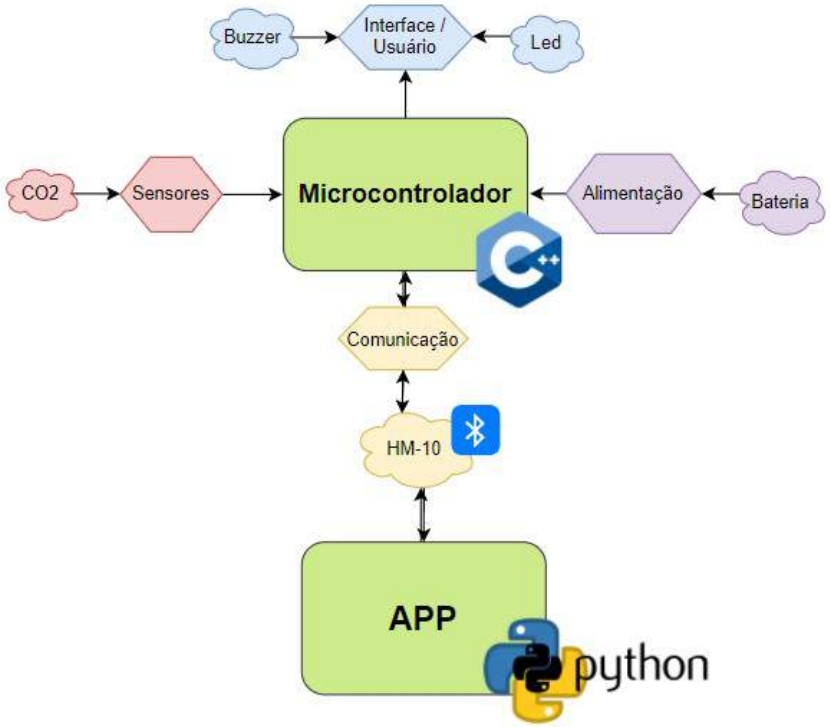

Figura 8: Detalhamento dos componentes do módulo de monitoramento e sua interconexão com o microcontrolador de controle. Fonte: autoria própria.

3.3 Protocolo proprietário de comunicação

A fim de garantir uma comunicação sem erros entre a eletrônica embarcada e o aplicativo de *smartphone*, foi estabelecida a necessidade de criação de um protocolo de comunicação baseado na interpretação de uma gramática regular, definida através do processamento de expressões regulares.

Dado que as informações intercambiadas entre a eletrônica embarcada e o aplicativo de monitoramento no *smartphone* são basicamente o envio do *status* do sensor de CO_2, o nível da bateria (por parte da eletrônica) e o recebimento dos tempos de validade do filtro e do capuz (por parte do *smartphone*), foi estabelecido o protocolo apresentado na Figura 9. Esse protocolo estabelece o formato dos dados enviados e recebidos por ambas as partes (em formato de cadeia de caracteres padrão ASCII), e de acordo com as funções interpretadas pela biblioteca de expressão regular.

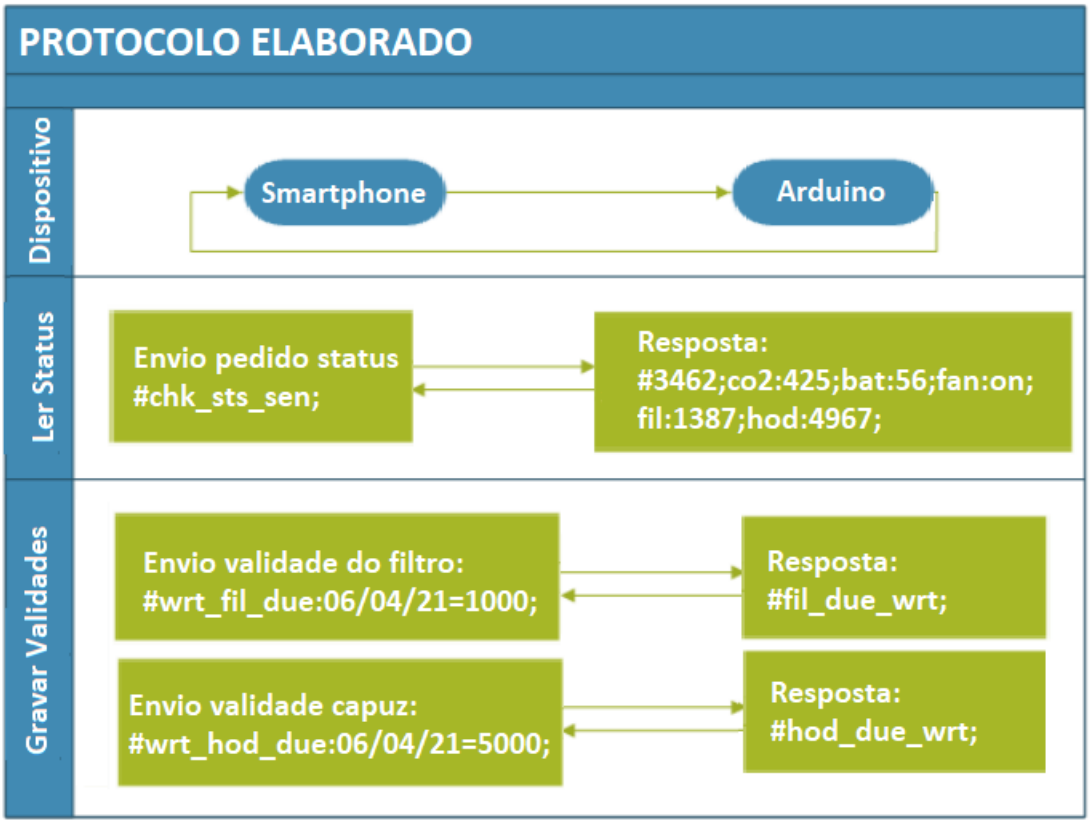

Figura 9: Protocolo elaborado para intercâmbio de dados entre a eletrônica embarcada e o aplicativo do *smartphone* usando expressões regulares. Fonte: autoria própria.

3.3.1 Controle do protocolo no Arduino

Para implementação do protocolo de comunicação no Arduino, foi utilizada uma biblioteca aberta de expressão regular (ER) para Arduino desenvolvida por Griffiths (2019), chamada *CompactRegex*, a qual se mostrou muito eficiente quanto ao uso mínimo de recursos do microcontrolador, tanto em termos de memória ROM quanto memória RAM.

Conforme comentado no capítulo 2.4, as ERs podem ser utilizadas como mascaramento, ou seja, formas padronizadas para validação de dados. A tabela II apresenta as funções implementadas pela biblioteca utilizada.

Tabela II: Símbolos interpretados pela biblioteca *CompactRegex* e suas funções.

Símbolos	Significado
^	indicador de caractere de início (à esquerda)
\\w+	sequência qualquer de alfanuméricos
?	nenhuma ou pelo menos uma ocorrência da sentença anterior
\\d	ocorrência de um único dígito entre 0 e 9
*	ocorrência obrigatória de pelo menos 1 ou mais sentenças
$	indicador de caractere terminador (à direita)

Fonte: Griffiths (2019).

Dessa forma, no lado da eletrônica embarcada, o microcontrolador Arduino deve então processar como válidos os seguintes universos de cadeias de caracteres (sendo o símbolo `X` qualquer dígito entre 0 e 9):

```
#chk_sts_sen;
#wrt_fil_due:XX/XX/XX=XXXX;
#wrt_hod_due:XX/XX/XX=XXXX;
```

Quaisquer sequências de cadeias de caracteres que não estejam nesse formato devem ser ignoradas, filtrando então (de forma digital) quaisquer eventuais ruídos gerados na comunicação e impedindo o recebimento de dados errôneos.

A fim de interpretar as cadeias de caracteres apresentadas, foi criada uma gramática regular usando a expressão:

```
^#\\w+[:]?[0-1]?[\\d\\d?/\\d\\d?/\\d\\d]*[=]?[\\d\\d\\d\\d]*;$
```

Essa expressão foi devidamente testada, conforme teste de comunicação exibido na Figura 10, com entradas válidas (a. e b.) e inválida (c.), e a listagem do código-fonte utilizada para realização do teste é apresentada na Listagem ‘A’ a seguir.

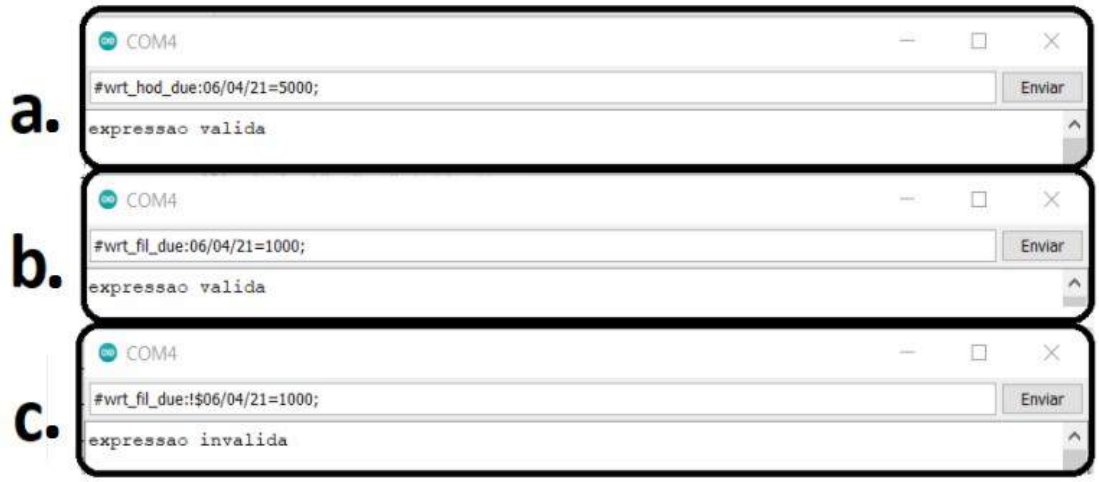

Figura 10: Teste, no PC, da biblioteca *CompactRegex* através do IDE do Arduino.

Listagem A: Código-fonte do *software* de teste, utilizando a biblioteca *CompactRegex*.

```
#include "Regex.h" //cabecalho da biblioteca CompactRegex
char buffer_serial[30], regex[60];
int index=0, res=-1;
void setup() { Serial.begin(9600); }
void loop() {
 if(Serial.available()) {
 char ch=Serial.read();
 if (ch!='\n') buffer_serial[index++]=ch;
 else {
  buffer_serial[index]=0;
  index=0;
  regex[60]="^#\\w+[:]?[0-1]?[\\d\\d?/\\d\\d?/\\d\\d]*[=]?[\\d\\d\\d\\d]*;$";
  res=regex.match(buffer_serial); //funcao que testa o buffer com base na ER
  if(res>=0) Serial.println("expressao valida");
  else Serial.println("expressao invalida");
  Serial.flush();
  }
 }
}
```

3.3.2 Controle do protocolo no aplicativo do *smartphone*

Para processamento das expressões regulares, foi utilizada a biblioteca RE disponível de forma nativa na linguagem de programação *Python* (versão 3.7.9 ou superior), cujos símbolos estão representados na Tabela III.

Tabela III: Símbolos interpretados pela biblioteca RE e suas funções.

Símbolos	Significado
\w+	sequência de alfanuméricos
\d+	sequência de dígitos entre 0 e 9

Fonte: Melo (2021).

No lado do *smartphone*, o aplicativo deve então processar como válidas as seguintes cadeias de caracteres (sendo o símbolo `X` qualquer dígito de 0 a 9):

```
#XXXXX;co2:XXX;bat:XX;fan:on;fil:XXXX;hod:XXXX;
```

A gramática regular implementada no *smartphone* deve garantir a chegada das informações de: CO_2, nível de bateria, validade do capuz e validade do filtro. A expressão regular que define essa gramática é apresentada a seguir.

```
#\d+;co2:\d+;bat:\d+;fan:\w+;fil:\d+;hod:\d+;
```

Conforme mostra a Figura 11, essa expressão, também foi testada em um portal online (https://tools.lymas.com.br/regexp_br.php) e cuja Listagem B apresenta o código.

Figura 11: Teste online de expressões regulares. Fonte: testador de expressão regular.

Listagem B: Código-fonte do *software* de teste, utilizando a biblioteca *RE* no *Python*.

```
import re
def validacao(mensagem):
    padrao = r"#\d+;co2:\d+;bat:\d+;fan:\w+;fil:\d+;hod:\d+;"
    return bool(re.match(padrao, mensagem))
```

3.4 Detalhamento do aplicativo *mobile*

3.4.1 Funcionalidades

O aplicativo deve orientar o indivíduo quanto às necessidades de trocas do filtro biológico e do capuz, um fator extremamente importante para a efetiva proteção do usuário. Num caso eventual dos indivíduos se esquecerem de substituir os filtros, o equipamento emitirá avisos sonoros até que a troca seja comprovada.

Para comprovação da troca de um filtro ou capuz vencidos (fora da validade), é necessário que o usuário utilize o *software* de monitoramento no celular para escanear um código de barras (*QR Code*), que se encontrará interno à embalagem de um filtro ou gravado no capuz novo, a fim de desativar o alarme sonoro do equipamento (ativado automaticamente quando a validade do filtro ou capuz se esgota), garantindo assim, que o conjunto só venha a ser utilizado em total condição de segurança ao indivíduo. Além da funcionalidade de troca do filtro biológico e do capuz mencionada anteriormente, há também as informações de bateria restante do capuz e quantidade de dióxido de carbono presente em seu interior. A Figura 12 demonstra a interface da tela principal do aplicativo com as funcionalidades mencionadas previamente.

Figura 12: Interface do aplicativo apresentando os níveis de bateria, dióxido de carbono e as validades do filtro e do capuz. Fonte: Vinicius Azeka/André Mozeto.

3.4.2 Substituição do filtro e capuz com *QR Code*

Quando um filtro ou capuz descartável é substituído, é necessário informar ao equipamento sobre tal substituição, utilizando para isso a função específica no aplicativo desenvolvido para *smartphone*, a qual deve ler o código de barras *QR* gravado na embalagem do filtro ou impresso no próprio capuz.

Foram elaborados códigos diferentes para as validades do filtro e do capuz, sendo que ambos possuem 11 dígitos, porém compostos de preâmbulos diferentes de 4 dígitos, os quais diferenciam o tipo de produto (filtro ou capuz), seguidos pelo número de série (com 7 dígitos) do produto em questão. A Figura 13 apresenta esse padrão, associado aos trechos que implementam essa funcionalidade no código do aplicativo.

Figura 13: Codificação dos códigos *QR* com preâmbulos e números de série para identificação única dos itens de consumo no equipamento. Fonte: autoria própria.

4 Resultados

Foram construídas as eletrônicas embarcadas do equipamento, tanto o módulo de monitoramento (Figura 14), quanto o módulo de potência (apêndices A e B), assim como seus *softwares* de controle (apêndice C), incluindo um aplicativo para *smartphone* elaborado para sistema operacional Android (TM), apresentado nas Figuras 15 e 16.

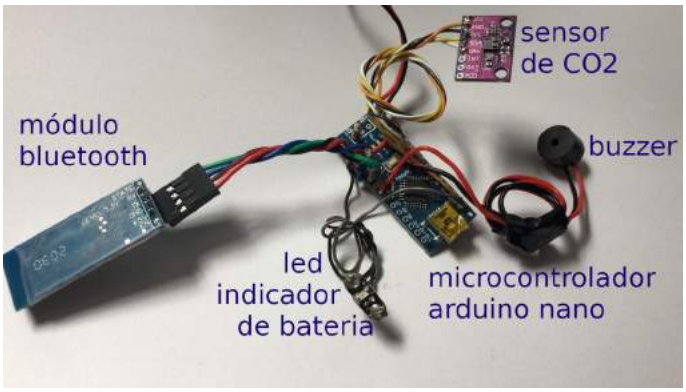

Figura 14: Montagem final do módulo de monitoramento. Fonte: autoria própria.

Foram testados com sucesso o monitoramento do nível de CO_2 transmitido pelo equipamento e monitorado pelo aplicativo do *smartphone*, conforme apresentado na Figura 15.

Caso o nível de CO_2 se torne crítico, ou seja, acima de 5000 PPM (parte por milhão), o equipamento emite um alerta sonoro, de forma independente do aplicativo do *smartphone*, de maneira a manter a segurança mesmo sem esse dispositivo.

Com relação à funcionalidade de troca dos filtros e capuzes, foram realizados testes com sucesso dessa substituição, como apresentado na Figura 16, em que um filtro hipotético com número de série 1234567 foi substituído através da leitura da imagem de um código *QR* gerado com o devido preâmbulo identificador do tipo de troca, sendo 5500 o valor pré-estabelecido para categorizar os filtros.

A imagem do filtro com a string 1234567 foi gerada em um portal gerador de *QR codes* com base em entrada textual da cadeia de caracteres a ser convertida.

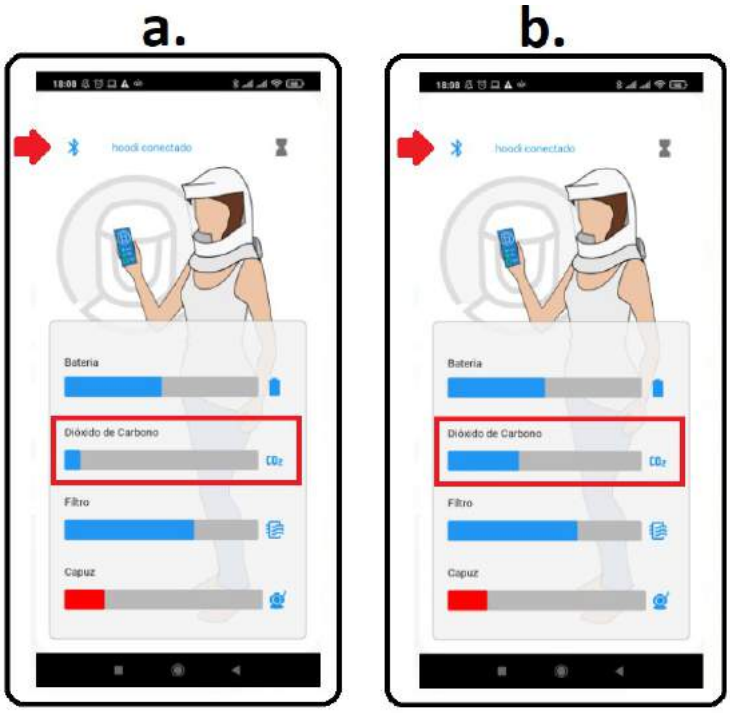

Figura 15: Demonstração do monitor de CO_2. Fonte: autoria própria.

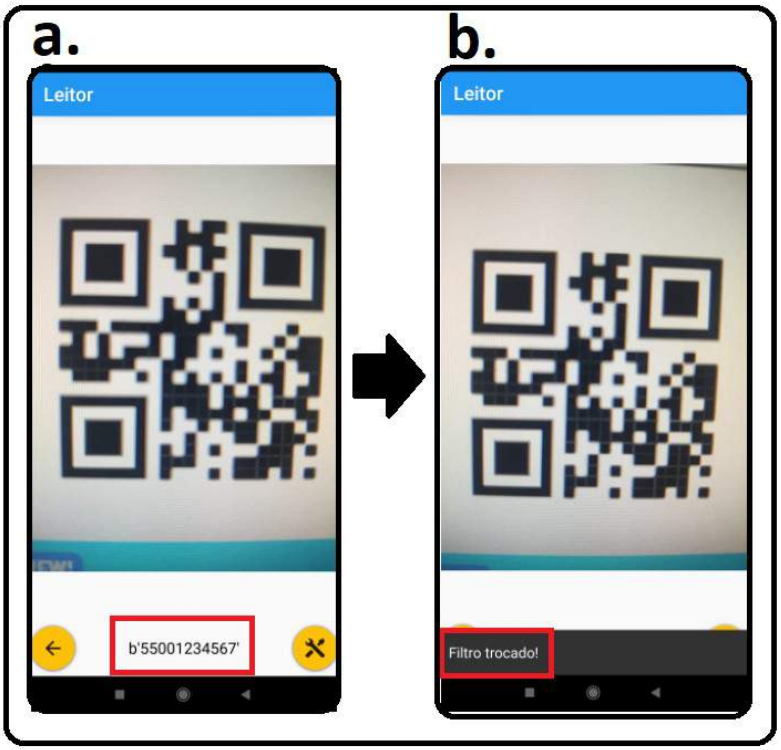

Figura 16: Demonstração da leitura de um código *QR*. Fonte: autoria própria.

5 Conclusões

Foi elaborado com sucesso um equipamento para monitorar e auxiliar indivíduos leigos à utilização de um dispositivo EPI avançado. Embora testes com o novo sistema ainda necessitem ser realizados com uma grande população de usuários, os componentes principais para atendimento aos requisitos necessários estão todos contidos nesse projeto, o que deve ocasionar baixa necessidade de ajustes futuros, mesmo após processada uma grande massa de testes com populações de indivíduos.

O equipamento possui tecnologia nacional e um preço de custo abaixo de 500 reais que garante trazer todas as garantias de proteção biológica (normalmente usufruídas apenas pelos biomédicos profissionais) para as pessoas comuns sem treinamentos.

Ao se comparar o presente projeto com soluções do mercado que se baseiam nas mesmas tecnologias, como o produto *BioVYZR – Personal Air-Purify*ing Shield da Vyzr Technologies (comentado brevemente no capítulo 2.1.1) que custa em torno de $379,00 (convertido para valores atuais em R$ 1970,00), o HOODI oferece um valor muito mais em conta e acessível estimado em R$ 900,00.

6 Trabalhos futuros

- Melhorar a eficiência do sistema de bateria e carregador, ampliando o tempo de autonomia do equipamento de 5 horas para 9 horas de operação, atingindo a especificação recomendada;

- Implementar sistema de controle de disputa de recursos críticos (como por exemplo a EEPROM interna) através do uso de semáforos, evitando conflito entre rotinas de gravação da memória EEPROM dentro do *loop* principal com rotinas de gravação das mesmas regiões de memória EEPROM, porém executadas dentro das rotinas de interrupção de *timer*, garantindo a atomicidade da operação.

Referências bibliográficas

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 14749, Equipamento de proteção respiratória - Respirador de adução de ar - Respirador de linha de ar comprimido com capuz., p. 22. 2001

GARCIA, F. D. Introdução aos sistemas embarcados e microcontroladores. Disponível em: https://www.embarcados.com.br/sistemas-embarcados-e-microcontroladores/ Acesso em: 16/03/21.

GARCIA, L.P. Uso de máscara facial para limitar a transmissão da COVID-19 - Epidemiol. Serv. Saúde vol.29 no.2 - 2020 – Brasília.

GAZZIRO, et al. Tornando os equipamentos de proteção individual (EPI) mais acessíveis. Revista Pesquisa ABC, No. 29, p.30, 2021.

GRIFFITHS, M. Arduino Regular Expressions. Disponível em: https://practicalarduinoc.blogspot.com/2019/10/arduino-regular-expressions.html. Acesso em: 27/06/2021.

JARGAS, A. Expressoes Regulares. Disponível em: https://aurelio.net/regex. Acesso em: 14 abril 2021.

KERNICHAN, B. W.; RITCHIE, D. M. The C Programming Language. Prentice Hall, 2º edição. 1978.

KUCHARSKI AJ, Russel TW, Diamond C, Liu Y, Edmunds J, Funk S, et al. Early dynamics of transmission and control of COVID-19: a mathematical modelling study. Disponível em: https://doi.org/10.1016/S1473-3099(20)30144-4. Acesso em: 21/06/21.

MACORATTI, J. C. Gerando, Salvando e Lendo QRCode Disponível em: http://www.macoratti.net/14/10/vbn_qrcd1.htm. Acessado em: 26/06/2021.

MELO, D. O que é Python? Disponível em: https://tecnoblog.net/405640/o-que-e-python-guia-para-iniciantes/. Acesso em: 24/04/2021.

MONTEIRO, L. P. O que é linguagem de programação. Disponível em: https://universidadedatecnologia.com.br/**o-que-e-linguagem-de-programacao/** Acesso em: 12/04/21.

NR06. NORMA REGULAMENTADORA 06 Equipamento de proteção individual - Ministério do Trabalho e Emprego - Artigo 6.1, p. 01. 2001

PATTERSON, D. A.; HENNESSY, J. L. Computer Organization and Design: The Hardware/Software Interface 5th Edition. Morgan Kaufmann-Elsevier. 2013.

ROVEDA, A. Linguagem de programação. Disponível em: https://kenzie.com.br/blog/linguagem-de-programacao/. Acesso em: 13/04/ 2021.

SILVA, G. QR Code: Entenda como funciona e aprenda como fazer Disponível em: https://canaltech.com.br/internet/qr-code-saiba-como-funciona-e-aprenda-como-fazer/ Acesso em: 27/06/2021.

TREVISAN, T. S. Usando Expressões Regulares na Linguagem C. Disponível em: **https://thobias.org/doc/er_c.html**. Acesso em: 27/06/2021.

ULLMAN, J. D. et. al. Compilers: Principles, Techniques, and Tools. AbeBooks, 1º edição. 1986.

Apêndice A – Testes sobre autonomia da bateria

A Tabela A-I e a Figura A-1 mostram o comportamento da bateria do equipamento durante sua descarga (quando do equipamento em utilização), assim como sua carga (com o equipamento desligado e a bateria acoplada ao carregador).

Constatamos uma autonomia de apenas 5 horas e meia, e um tempo de carregamento de 3 horas. Após 5 horas, o nível da bateria fica abaixo de 5 volts, e o conversor DC/DC não consegue mais elevar essa tensão para os 24 volts da turbina.

Tabela A-I: Tempos de carga e descarga da bateria.

Tempo (h)	Carga (V)	Descarga (V)
00:00:00	7,3	16,3
01:30:00	9,3	-
03:00:00	16,3	-
04:00:00	-	14,3
05:30:00	-	5
06:45:00	-	2,3

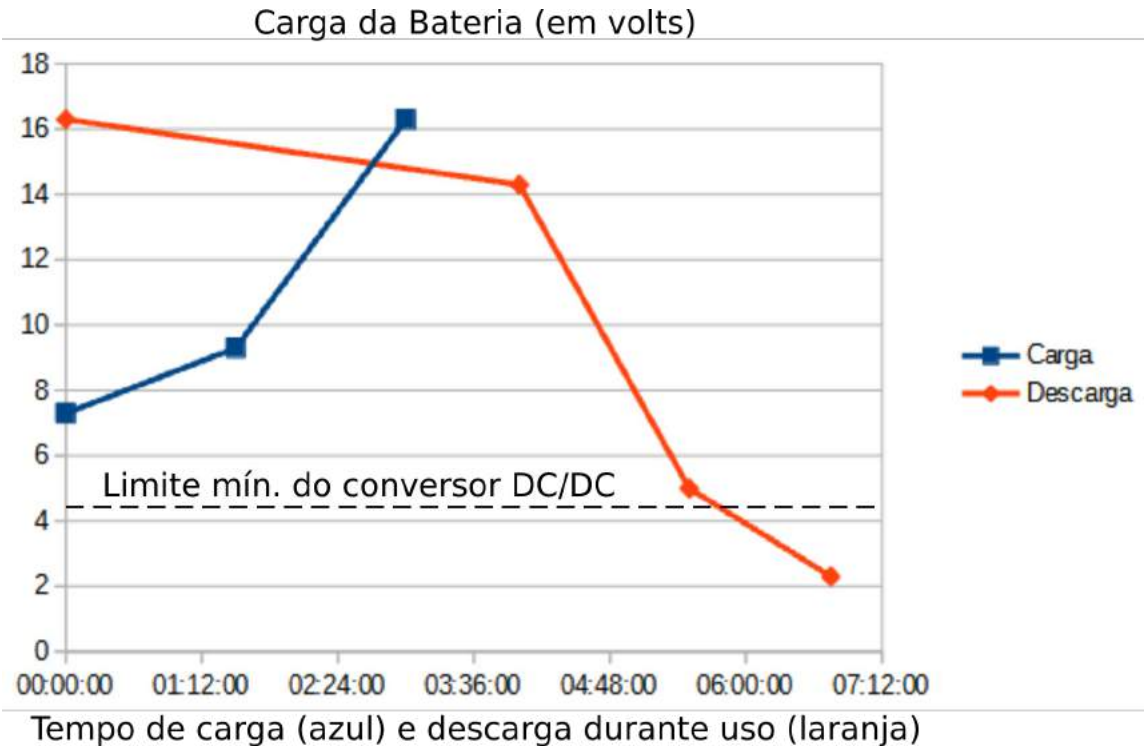

Figura A-1: Tempos de carga e descarga da bateria. Fonte: autoria Própria.

Apêndice B – Eficiência do conversor DC/DC

Com base nos testes de eficiência pesquisados (apresentados nas Figuras B-1 e B-2) com o mesmo modelo de conversor DC/DC utilizado no projeto, foi verificado que o uso de uma bateria com tensão de (no mínimo) 12 volts seria uma escolha ideal em função da melhor eficiência da conversão da tensão, visando uma maior autonomia de bateria do equipamento.

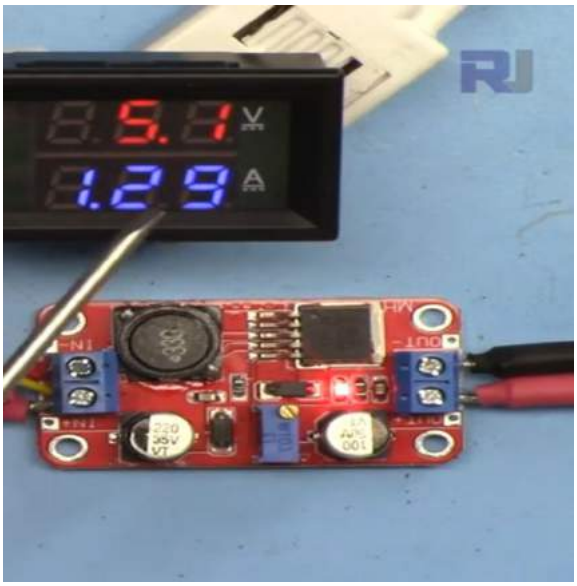

Figura B-1: Conversor utilizando 5 volts de entrada. Fonte: Robojax.com.

Tabela B-I: Resultado do conversor com entrada de 5 volts.

Eficiência	**74,3 %**
Tensão de entrada (tensão da bateria)	5V
Corrente de entrada	2,54A
Potência de entrada	12,7W
Tensão de saída (tensão da turbina)	24V
Corrente de saída	0,4A
Potência de saída	9,44W

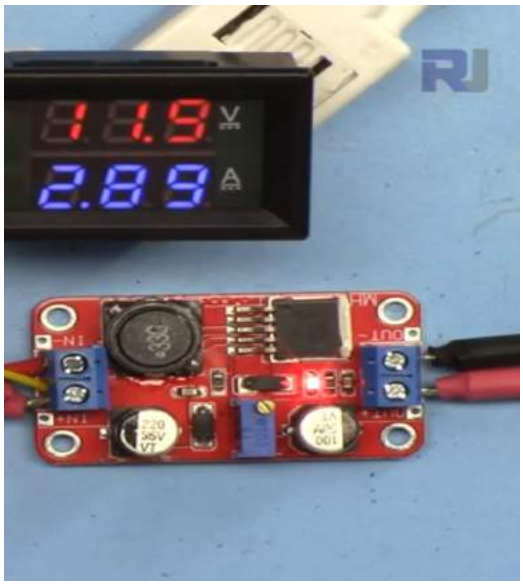

Figura B-2: Conversor utilizando 12 volts de entrada. Fonte: Robojax.com.

Tabela B-II: Resultado do conversor com entrada de 12 volts.

Eficiência	**90,0 %**
Tensão de entrada (tensão da bateria)	12V
Corrente de entrada	2,66A
Potência de entrada	31,9W
Tensão de saída (tensão da turbina)	24V
Corrente de saída	1,2A
Potência de saída	28,7W

As tabelas B-1 e B-2 apresentam respectivamente as eficiências de 74,3% e 90,0% para a eficiência do conversor DC/DC, deixando claro que tensões acima de 12 volts devem ser utilizadas pela bateria, para evitar perda de eficiência no processo de conversão, utilizando ao máximo a durabilidade da carga da bateria.

Apêndice C – Códigos-fonte

Código-fonte do Arduino

```
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>
#include "SparkFunCCS811.h"
#include <EEPROM.h>
#include <TimerOne.h>

#define TIMER_US 100000
#define TICK_COUNTS 600
SoftwareSerial mySerial(2, 3); // RX, TX
#define CCS811_ADDR 0x5A
#define BUZZER 5
#define NIVEL_BATERIA A3
#define LED_BATERIA_BAIXA 13
CCS811 mySensor(CCS811_ADDR);

char buffer_serial[30];
int index = 0;
String comando, valor;
int CO2;
int BAT;
int FIL;
int HOD;
byte hiFIL;
byte loFIL;
byte hiHOD;
byte loHOD;
volatile long tick_count = TICK_COUNTS;
volatile bool in_long_isr = false;

void chk_new_eeprom()
{
  hiFIL = EEPROM.read(0);
  loFIL = EEPROM.read(1);
  hiHOD = EEPROM.read(2);
  loHOD = EEPROM.read(3);
  FIL = word(hiFIL, loFIL);
  HOD = word(hiHOD, loHOD);
  if ((FIL==65535) || (HOD==65535))
  {
    EEPROM.write(0, 0);
    EEPROM.write(1, 0);
    EEPROM.write(2, 0);
    EEPROM.write(3, 0);
  }
}

void setup()
{
  pinMode(LED_BATERIA_BAIXA, OUTPUT);
  pinMode(BUZZER, OUTPUT);
  digitalWrite(LED_BATERIA_BAIXA, LOW);
  musica_inicio();
  Timer1.initialize(TIMER_US);
  Timer1.attachInterrupt( timerIsr );
  chk_new_eeprom();
  mySerial.begin(9600);
  Wire.begin();
  if (mySensor.begin() == false)
  {
    mySerial.print("#err:1");
    while (1);
  }
}
```

```
void loop()
{
  int com=0;
  String tempFILstr;
  String tempHODstr;
  int val = analogRead(A3);
  BAT = map(val, 0, 1023, 0, 100);
  if (BAT<=45) digitalWrite(LED_BATERIA_BAIXA, HIGH);

  if(mySerial.available())
  {
    char ch = mySerial.read();
    if (ch != '\n')
    {
      buffer_serial[index++] = ch;
    }
    else
    {
      buffer_serial[index] = 0;
      index = 0;
      validate_serial_buffer();
      mySerial.flush();
      com=processCommand();

      switch (com)
      {
        case 1:
          mySerial.print("#");
          mySerial.print(millis());
          mySerial.print(";co2:");
          mySerial.print(String(CO2));
          mySerial.print(";bat:");
          mySerial.print(String(BAT));
          mySerial.print(";fan:on;fil:");
          hiFIL = EEPROM.read(0);
          loFIL = EEPROM.read(1);
          FIL = word(hiFIL, loFIL);
          if (FIL<0) FIL=0;
          mySerial.print(String(FIL));
          mySerial.print(";hod:");
          hiHOD = EEPROM.read(2);
          loHOD = EEPROM.read(3);
          HOD = word(hiHOD, loHOD);
          if (HOD<0) HOD=0;
          mySerial.print(String(HOD));
          mySerial.println(";");
          break;
        case 2:
          // comando(s)
          tempFILstr = getValue(valor, '=', 1);
          FIL=tempFILstr.toInt();
          if (FIL>1500) { FIL=1500; }
          hiFIL = highByte(FIL);
          loFIL = lowByte(FIL);
          EEPROM.write(0, hiFIL);
          EEPROM.write(1, loFIL);
          hiFIL = EEPROM.read(0);
          loFIL = EEPROM.read(1);
          FIL = word(hiFIL, loFIL);
          mySerial.println("#fil_due_wrt;");
          break;
        case 3:
          // comando(s)
          tempHODstr = getValue(valor, '=', 1);
          HOD=tempHODstr.toInt();
          if (HOD>5000) { HOD=5000; }
          hiHOD = highByte(HOD);
          loHOD = lowByte(HOD);
          EEPROM.write(2, hiHOD);
          EEPROM.write(3, loHOD);
          hiHOD = EEPROM.read(2);
```

```
          loHOD = EEPROM.read(3);
          HOD = word(hiHOD, loHOD);
          mySerial.println("#hod_due_wrt;");
          break;
        default:
          break;
      }
    }
  }

  if (mySensor.dataAvailable())
  {
    String TempCO2STR;
    mySensor.readAlgorithmResults();
    TempCO2STR=mySensor.getCO2();
    CO2=TempCO2STR.toInt();
    if (CO2>4500) alarme();
    if (CO2>5000) CO2=5000;
  }
}

void validate_serial_buffer()
{
  int mc = 0;
  char regex[60] = "^#\\w+[:]?[0-1]?[\\d\\d?/\\d\\d?/\\d\\d]*[=]?[\\d\\d\\d\\d]*;$";

  int res = testRegex(regex, buffer_serial, &mc);

  if(res>=0)
  {
    comando = getValue(buffer_serial, ':', 0);
    valor = getValue(buffer_serial, ':', 1);
    if (valor.length()>0) valor[valor.length()-1]=0;
    else comando[comando.length()-1]=0;
  }
}

int testRegex(char* regx, char* text, int* mc) {
  Regex regex(regx);
  int res = regex.match(text);
  *mc = regex.getMatchLength();
  return res;
}

String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

int processCommand() {
  int com;
  if (comando=="#chk_sts_sen") com=1;
  else
    if (comando=="#wrt_fil_due") com=2;
    else
      if (comando=="#wrt_hod_due") com=3;
      else com=0;
  comando[0]=0;
  return com;
}
```

```
// --------------------------
// timerIsr() 100 milli second interrupt ISR()
// Called every time the hardware timer 1 times out.
// --------------------------
void timerIsr()
{
    if (!(--tick_count))
    {
      tick_count = TICK_COUNTS;
      tick_60s_isr();
    }
}

// --------------------------
// tick_60s_isr() 60 second routine
// Called every time the count gets to 60S
// --------------------------
void tick_60s_isr()
{
    if (in_long_isr)
    {
      return;
    }

    in_long_isr = true;

    volatile long i;

    hiFIL = EEPROM.read(0);
    loFIL = EEPROM.read(1);
    FIL = word(hiFIL, loFIL);
    if (FIL>0)
    {
      FIL=FIL-1;
      hiFIL = highByte(FIL);
      loFIL = lowByte(FIL);
      EEPROM.write(0, hiFIL);
      EEPROM.write(1, loFIL);
    }
    else
    {
      alarme();
    }

    hiHOD = EEPROM.read(2);
    loHOD = EEPROM.read(3);
    HOD = word(hiHOD, loHOD);
    if (HOD>0)
    {
      HOD=HOD-1;
      hiHOD = highByte(HOD);
      loHOD = lowByte(HOD);
      EEPROM.write(2, hiHOD);
      EEPROM.write(3, loHOD);
    }
    else
    {
      alarme();
    }

    interrupts();

    for (i = 0; i < 400000; i++)
    {
    }

    noInterrupts();
    in_long_isr = false;
}
```

```
void alarme()
{
    analogWrite(BUZZER, 200);
    delay(100);
    analogWrite(BUZZER, 25);
    delay(100);
    analogWrite(BUZZER, 200);
    delay(100);
    analogWrite(BUZZER, 25);
    delay(100);
    analogWrite(BUZZER, 200);
    delay(100);
    analogWrite(BUZZER, 25);
    delay(100);
    analogWrite(BUZZER, 200);
    delay(100);
    analogWrite(BUZZER, 25);
    delay(100);
    analogWrite(BUZZER, 0);
}

void musica_inicio()
{
    // Aciona o BUZZER na frequencia relativa ao Dó em Hz
    tone(BUZZER,261);
    // Espera um tempo para Desativar
    delay(200);
    //Desativa o BUZZER
    noTone(BUZZER);
    // Aciona o BUZZER na frequencia relativa ao Ré em Hz
    tone(BUZZER,293);
    delay(200);
    noTone(BUZZER);
    // Aciona o BUZZER na frequencia relativa ao Mi em Hz
    tone(BUZZER,329);
    delay(200);
    noTone(BUZZER);
    // Aciona o BUZZER na frequencia relativa ao Fá em Hz
    tone(BUZZER,349);
    delay(200);
    noTone(BUZZER);
    // Aciona o BUZZER na frequencia relativa ao Sol em Hz
    tone(BUZZER,392);
    delay(200);
    noTone(BUZZER);
}
```

Código-fonte do aplicativo *mobile*

```
from kivymd.app import MDApp
from kivy.lang.builder import Builder
from kivy.uix.screenmanager import ScreenManager, Screen
from screen_nav2 import mainscreen
import re
import time
from plyer import notification
from kivy.clock import Clock
from kivymd.uix.snackbar import Snackbar
from jnius import autoclass
from datetime import date

def validacao(mensagem):
 padrao = r"#\d+;co2:\d+;bat:\d+;fan:\w+;fil:\d+;hod:\d+;"
 return bool(re.match(padrao, mensagem))

def QRvalido(mensagem):
 padrao = r"b'(5500|8800)\d{7}'"
 return bool(re.match(padrao, mensagem))
```

```
BluetoothAdapter = autoclass('android.bluetooth.BluetoothAdapter')
BluetoothDevice = autoclass('android.bluetooth.BluetoothDevice')
BluetoothSocket = autoclass('android.bluetooth.BluetoothSocket')
UUID = autoclass('java.util.UUID')
CharBuilder = autoclass('java.lang.Character')

def get_socket_stream(name):
 paired_devices = BluetoothAdapter.getDefaultAdapter().getBondedDevices().toArray()
 socket = None
 for device in paired_devices:
  if device.getName() == name:
   socket = device.createRfcommSocketToServiceRecord(
    UUID.fromString("00001101-0000-1000-8000-00805F9B34FB"))
   recv_stream = socket.getInputStream()
   send_stream = socket.getOutputStream()
   break
 socket.connect()
 return recv_stream, send_stream


class TelaPrincipal(Screen):
 def poweron(self):
  if self.ids.lblbluetooth.text == "Dispositivo desconectado":
   self.ids.btnpower.theme_text_color = "Custom"
   self.ids.btnpower.text_color = ConnexApp().theme_cls.primary_color
   self.ids.icnbluetooth.theme_text_color = "Custom"
   self.ids.icnbluetooth.text_color = ConnexApp().theme_cls.primary_color
   self.ids.lblbluetooth.theme_text_color = "Custom"
   self.ids.lblbluetooth.text_color = ConnexApp().theme_cls.primary_color
   self.ids.lblbluetooth.text = "Dispositivo conectado"
  else:
   self.ids.btnpower.theme_text_color = "Secondary"
   self.ids.icnbluetooth.theme_text_color = "Secondary"
   self.ids.lblbluetooth.theme_text_color = "Secondary"
   self.ids.lblbluetooth.text = "Dispositivo desconectado"

class TelaScanner(Screen):
 pass

class MyScreenManager(ScreenManager):
 pass

class ConnexApp(MDApp):
 def passatempo(self, dt):

  if self.root.get_screen('telaprincipal').ids.lblbluetooth.text == "Dispositivo conectado":
   i = []
   pre = bytearray(i)
   cmd = '#chk_sts_sen;\n'.encode('UTF-8')
   pre.extend(cmd)
   self.send_stream.write(pre)
   self.send_stream.flush()

  read = ""

  while self.recv_stream.available() != 0:
   r = self.recv_stream.read()
   theChar = CharBuilder.toChars(r)  # gives a tuple of which the first element is a character
   read += theChar[0]

  if read != "" and validacao(read):
   CO2 = int(read.split(";")[1].split(":")[1])
   BAT = int(read.split(";")[2].split(":")[1])
   FIL = int(read.split(";")[4].split(":")[1])
   HOD = int(read.split(";")[5].split(":")[1])

   if CO2 != int(self.root.get_screen('telaprincipal').ids.pbco2.value) \
     or BAT != int(self.root.get_screen('telaprincipal').ids.pbbateria.value) \
     or FIL != int(self.root.get_screen('telaprincipal').ids.pbfiltro.value) \
     or HOD != int(self.root.get_screen('telaprincipal').ids.pbcapuz.value):
    self.root.get_screen('telaprincipal').ids.pbco2.value = CO2
```

```
    self.root.get_screen('telaprincipal').ids.pbbateria.value = BAT
    self.root.get_screen('telaprincipal').ids.pbfiltro.value = FIL
    self.root.get_screen('telaprincipal').ids.pbcapuz.value = HOD

    if self.root.get_screen('telaprincipal').ids.pbfiltro.value < 120:
     if self.root.get_screen('telaprincipal').ids.pbfiltro.value % 10 == 0:
      notification.notify('Filtro',
           'Seu filtro está com ' + str(self.root.get_screen(
            'telaprincipal').ids.pbfiltro.value) + 'minutos restantes.')
    if self.root.get_screen('telaprincipal').ids.pbcapuz.value < 120:
     if self.root.get_screen('telaprincipal').ids.pbcapuz.value % 10 == 0:
      notification.notify('Capuz',
           'Seu capuz está com ' + str(self.root.get_screen(
            'telaprincipal').ids.pbcapuz.value) + 'minutos restantes.')
    if self.root.get_screen('telaprincipal').ids.pbbateria.value < 50:
     if self.root.get_screen('telaprincipal').ids.pbbateria.value % 10 == 0:
      notification.notify('Bateria', 'Sua bateria está com ' + str(
       self.root.get_screen('telaprincipal').ids.pbbateria.value) + '%')


 def abrirleitor(self):
  self.root.get_screen('telaprincipal').ids.lblbluetooth.text = "Dispositivo desconectado"
  if self.root.get_screen('telaprincipal').ids.pbfiltro.value > 25 and self.root.get_screen(
    'telaprincipal').ids.pbcapuz.value > 25:
   self.root.current = 'telascanner'
   i = []
   pre = bytearray(i)
   cmd = '#chg_lvl:1;\n'.encode('UTF-8')
   pre.extend(cmd)
   self.send_stream.write(pre)
   self.send_stream.flush()
   Snackbar(text="Você ainda não necessita de trocas").show()
  else:
   self.root.current = 'telascanner'
   i = []
   pre = bytearray(i)
   cmd = '#chg_lvl:1;\n'.encode('UTF-8')
   pre.extend(cmd)
   self.send_stream.write(pre)
   self.send_stream.flush()

 def voltartela(self):
  self.root.current = 'telaprincipal'
  i = []
  pre = bytearray(i)
  cmd = '#chg_lvl:0;\n'.encode('UTF-8')
  pre.extend(cmd)
  self.send_stream.write(pre)
  self.send_stream.flush()
  self.root.get_screen('telaprincipal').ids.lblbluetooth.text = "Dispositivo conectado"

 def build(self):
  # self.theme_cls.secondary_palette = "Green"
  # self.theme_cls.theme_style = "Dark"
  self.recv_stream, self.send_stream = get_socket_stream('HC-06')
  screen = Builder.load_string(mainscreen)
  Clock.schedule_interval(self.passatempo, 0.1)

  return screen

 def leitor(self):
  capture = self.root.get_screen('telascanner').ids.qrlabel.text
  if QRvalido(capture):
   if "5500" in str(capture)[0:6]:
    read = ""
    tentativas = 0
    while read != "#fil_due_wrt;" or tentativas < 10:
     i = []
     pre = bytearray(i)
            cmd  =  str('#wrt_fil_due:'  +  str(date.today().strftime("%d/%m/%Y"))  +  '=1500;\
n').encode('UTF-8')
```

```
     pre.extend(cmd)
     self.send_stream.write(pre)
     self.send_stream.flush()
     time.sleep(0.5)
     read = ""
     while self.recv_stream.available() != 0:
      r = self.recv_stream.read()
      theChar = CharBuilder.toChars(r)  # gives a tuple of which the first element is a character
      read += theChar[0]
     time.sleep(0.5)
     tentativas = tentativas + 1
    Snackbar(text="Filtro trocado!").show()

   if "8800" in str(capture)[0:6]:
    read = ""
    while read != "#hod_due_wrt;":
     i = []
     pre = bytearray(i)
 cmd = str('#wrt_hod_due:' + str(date.today().strftime("%d/%m/%Y")) + '=5000;\n').encode('UTF-8')
     pre.extend(cmd)
     self.send_stream.write(pre)
     self.send_stream.flush()
     time.sleep(0.5)
     read = ""
     while self.recv_stream.available() != 0:
      r = self.recv_stream.read()
      theChar = CharBuilder.toChars(r)  # gives a tuple of which the first element is a character
      read += theChar[0]
     time.sleep(0.5)

    Snackbar(text="Capuz trocado!").show()

  else:

   Snackbar(text="QR inválido!").show()

if __name__ == '__main__':
 ConnexApp().run();
```

Biblioteca de expressão regular para Arduino

Arquivo regex.h

```
#ifndef Regex_h
#define Regex_h
#include "Vector.h"
#include <Arduino.h>

class Regex
{
  struct RE{
    int8_t cType;
    union {
      char c;
      char* pc;
    };
  };
  void compile(char*);
  void zapClasses();
  int matchPattern(char*, int);
  int matchPatternToken(int, char*);
  bool isAlphaNum(char);
  bool matchRange(char, char*);
  bool isMeta(char);
  bool matchMeta(char, char);
  bool matchType(char, RE);
  bool matchClass(char, char*);
  bool matchNextToken(int, char*);
```

```
  int matchAsterisk(int, char*);
  int matchPlus(int, char*);
  int matchQuestionMark(int, char*);
  Vector<RE> repSet;
  int matchLength;
  public:
    Regex(char*);
    ~Regex();
    int match(char* text, int startPos = 0);
    int getMatchLength();
};
#endif
```

Arquivo vector.h

```
#ifndef Vector_h
#define Vector_h
#include <Arduino.h>
template<typename T>
class Vector {
public:
  Vector();
  ~Vector();
  void push_back(const T);
  size_t size();
  T &operator[](size_t n) {
    if (n < top) {
      return vStart[n];
    }
  };
private:
  T* vStart;
  int top;
  size_t tSize, mSize;
  void init();
  void resize(int);
  static const int defSize = 8; // tweak as required
};

// constructors and Destructor
template<typename T>
Vector<T>::Vector() {
  mSize = defSize;
  init();
}
template<typename T>
Vector<T>::~Vector() {
  free(vStart);
}

// public methods
template<typename T>
size_t Vector<T>::size() {
  return top;
}
template<typename T>
void Vector<T>::push_back(const T t) {
  if (top == mSize) {
    int inc = mSize + (log(mSize) / log(2));
    resize(inc);
    if (mSize < inc) {
      return;
    }
  }
  memcpy((vStart + top++), &t, tSize); // increments top after memcpy()
}

// private methods follow
template<typename T>
```

```
void Vector<T>::resize(int newSize) {
  void* newStart = realloc(vStart, tSize * newSize);
  if (newStart) {
    vStart = (T*)newStart;
    mSize = newSize;
  }
}
template<typename T>
void Vector<T>::init() {
  tSize = sizeof(T);
  vStart = (T*)malloc(tSize * mSize);
  if (!vStart) {
    mSize = 0;
  }
  top = 0;
 }
#endif
```

Arquivo regex.cpp

```
#include "Regex.h"
#include "Vector.h"
#include <Arduino.h>
#define CHARCLASS_MAX 20

enum Cons : int8_t {BEGIN, END, QUESTIONMARK, ASTERISK, PLUS, DOT, CHAR, CHARCLASS, CHARCLASSNOT,
DIGIT, NOTDIGIT, ALPHA, NOTALPHA, WHITESPACE, NOTWHITESPACE};

Regex::Regex(char* regexp) {
  compile(regexp);
}
Regex::~Regex() {
  zapClasses(); // recover heap space used for classes
}
int Regex::match(char* text, int startPos) {
  if(repSet.size()) {
    if(repSet[0].cType == BEGIN) {
      return (matchPattern(text, 1) == 0) ? 0 : -1;
    } else {
      int mPos = matchPattern(text + startPos, 0);
      return (mPos > -1) ? mPos + startPos : mPos;
    }
  }
  return -1;
}

int Regex::getMatchLength() {
  return matchLength;
}

//  private methods
int Regex::matchPattern(char* text, int patternPos) {
  int matchStart = 0, matchPos = 0;
  int j = repSet.size() - 1;
  while (text[0] != '\0') {
    char* workPos = text;
    matchLength = 0;
    matchStart = matchPos;
    for (int i = patternPos; i <= j; i++) {
      int charsMatched = matchPatternToken(i, workPos);
      if (charsMatched == -1) { break; }
      workPos += charsMatched;
      matchLength += charsMatched;
      if (i == j) {
        // match found
        return matchStart;
      }
      if (workPos[0] == '\0') {
        // run out of text but what tokens do we still have?
        if ((i - j) > 1) { return -1; }
```

```
      }
    }
    matchPos++;
    text++; // restart text search at next character
  }
  matchLength = 0;
  return -1;
}
int Regex::matchPatternToken(int tokenPos, char* text) {
  int mcs = -1;
  int repSetSize = repSet.size();
  if (tokenPos == repSetSize) {
    return -1;
  }
  RE token = repSet[tokenPos];
  if (token.cType == ASTERISK) {
    mcs = matchAsterisk(tokenPos - 1, text);
  }
  else if (token.cType == PLUS) {
    int mCount = matchPlus(tokenPos - 1, text);
    if (mCount > 0) { mcs = mCount; }
  }
  else if (token.cType == QUESTIONMARK) {
    mcs = matchQuestionMark(tokenPos - 1, text);
  }
  else if (token.cType == END) {
    if (text[0] == '\0') {
      mcs = 0;
    }
  }
  else {
    // first look forward one token
    if (tokenPos + 1 < repSetSize) {
      int8_t typ = repSet[tokenPos + 1].cType;
      if (typ == ASTERISK || typ == PLUS || typ == QUESTIONMARK) {
        return 0;
      }
    }
    if (matchType(text[0], token)) {
      mcs = 1;
    }
  }
  return mcs;
}

bool Regex::isAlphaNum(char c) {
  return isAlphaNumeric(c) || c == '_';
}
bool Regex::matchRange(char c, char* r) {
  if (r[0] != '\0' && r[1] == '-' && r[2] != '\0') {
    return (c >= r[0]) && (c <= r[2]);
  }
  return false;
}
bool Regex::isMeta(char c) {
  return (strchr("dDwWsS+", c) != NULL);
}
bool Regex::matchMeta(char c, char m) {
  switch(m) {
    case 'd':
      return isDigit(c);
    case 'D':
      return !isDigit(c);
    case 's':
      return isSpace(c);
    case 'S':
      return !isSpace(c);
    case 'w':
      return isAlphaNum(c);
    case 'W':
      return !isAlphaNum(c);
```

```
    default:
      return c == m;
  }
}
bool Regex::matchType(char c, RE token) {
  switch(token.cType) {
    case DOT:
      return (c != '\n' && c != '\0');
    case DIGIT:
      return isDigit(c);
    case NOTDIGIT:
      return !isDigit(c);
    case ALPHA:
      return isAlphaNum(c);
    case NOTALPHA:
      return !isAlphaNum(c);
    case WHITESPACE:
      return isSpace(c);
    case NOTWHITESPACE:
      return !isSpace(c);
    case CHARCLASS:
      return matchClass(c, token.pc);
    case CHARCLASSNOT:
      return !matchClass(c, token.pc);
    default:
      return c == token.c;
  }
}
bool Regex::matchNextToken(int tokenPos, char* text) {
  if(tokenPos < repSet.size()) {
    return matchType(text[0], repSet[tokenPos]);
  }
  return false;
}
bool Regex::matchClass(char c, char* clDef) {
  // class may contain one or more range or
  // a list of chars to match against that list might include '-'
  bool range = (clDef[1] == '-' && clDef[2] != '\0');
  if (range) {
    while (range) {
      if (matchRange(c, clDef)) { return true; }
      clDef += 3;
      range = (clDef[1] == '-' && clDef[2] != '\0');
    }
    return false;
  }
  while (clDef[0] != '\0') {
    if (clDef[0] == '\\' && clDef[1] != '\0') {
      if (isMeta(clDef[1])) {
        clDef++;
        if (matchMeta(c, clDef[0])) {
          return true;
        }
      }
    }
    else {
      if (c == clDef[0]) { return true; }
    }
    clDef++;
  }
  return false;
}
// match zero or more chars to previous token
// stopped by lookahead match
int Regex::matchAsterisk(int tokenPos, char* text) {
  int retVal = 0;
  if (matchNextToken(tokenPos + 2, text)) {
    return retVal;
  }
  RE token = repSet[tokenPos];
  while (matchType(text[0], token)) {
```

```
    retVal++;
    text++;
    if (text[0] == '\0' || (matchNextToken(tokenPos + 2, text))) { break; }
  }
  return retVal;
}
// match one or more char to previous token
// stopped by lookahead match
int Regex::matchPlus(int tokenPos, char* text) {
  int retVal = 0;
  RE token = repSet[tokenPos];
  while (matchType(text[0], token)) {
    retVal++;
    text++;
    if (text[0] == '\0' || (matchNextToken(tokenPos + 2, text))) { break; }
  }
  return retVal;
}
// match zero or one char to previous token
// even this blocked by lookahead match I think
int Regex::matchQuestionMark(int tokenPos, char* text) {
  int retVal = 0;
  if (matchNextToken(tokenPos + 2, text)) {
    return retVal;
  }
  RE token = repSet[tokenPos];
  if (matchType(text[0], token)) { retVal++; }
  return retVal;
}

// compile() tokenises the regular expression
void Regex::compile(char* regexp) {
  int cPos = 0;
  RE tre;
  while (regexp[cPos] != '\0') {
    tre.pc = NULL;
    switch (regexp[cPos]) {
    case '^':
      tre.cType = BEGIN;
      break;
    case '$':
      tre.cType = END;
      break;
    case '.':
      tre.cType = DOT;
      break;
    case '*':
      tre.cType = ASTERISK;
      break;
    case '?':
      tre.cType = QUESTIONMARK;
      break;
    case '+':
      tre.cType = PLUS;
      break;
    case '\\':
      if (regexp[cPos + 1]) {
        cPos++;
        switch (regexp[cPos]) {
        case 'd':
          tre.cType = DIGIT;
          break;
        case 'D':
          tre.cType = NOTDIGIT;
          break;
        case 'w':
          tre.cType = ALPHA;
          break;
        case 'W':
          tre.cType = NOTALPHA;
          break;
```

```
        case 's':
          tre.cType = WHITESPACE;
          break;
        case 'S':
          tre.cType = NOTWHITESPACE;
          break;
        default:
          tre.cType = CHAR;
          tre.c = regexp[cPos];
        }
      }
      break;
    case '[':
    {
      if (regexp[cPos + 1] == '^') {
        tre.cType = CHARCLASSNOT;
        cPos++;
      }
      else {
        tre.cType = CHARCLASS;
      }
      char buff[CHARCLASS_MAX];
      int bPos = 0;
      cPos++;
      while (regexp[cPos] != ']' && regexp[cPos] != '\0' && bPos < CHARCLASS_MAX - 1) {
        buff[bPos] = regexp[cPos];
        bPos++;
        cPos++;
      }
      buff[bPos] = '\0';
      tre.pc = (char*)malloc(++bPos);
      memcpy(tre.pc, buff, bPos);
      break;
    }
    default:
      tre.cType = CHAR;
      tre.c = regexp[cPos];
    }
    repSet.push_back(tre);
    cPos++;
  }
}

void Regex::zapClasses() {
  for (int i = 0, j = repSet.size(); i < j; i++) {
    if (repSet[i].cType == CHARCLASS || repSet[i].cType == CHARCLASSNOT) {
      free(repSet[i].pc);
    }
  }
}
```