UNIVERSIDADE FEDERAL DE PERNAMBUCO

Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Eletrônica e Sistemas

Graduação em Engenharia Eletrônica

Daniel Costa Sampaio

Sistema de detecção de queda utilizando a plataforma Toit

Recife

2023

Daniel Costa Sampaio

Sistema de detecção de queda utilizando a plataforma Toit

Trabalho de Conclusão apresentado ao Curso de Graduação em Engenharia Eletrônica, do Departamento de Eletrônica e Sistemas, da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em Engenharia Eletrônica.

Orientador: Prof. Guilherme Nunes Melo, D.Sc.

Recife
2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Sampaio, Daniel Costa.
Sistema de detecção de queda utilizando a plataforma Toit / Daniel Costa Sampaio. - Recife, 2023.
47 : il.

Orientador(a): Guilherme Nunes Melo
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Eletrônica - Bacharelado, 2023.
Inclui referências, apêndices.

1. Sistemas Embarcados . 2. Firmware. 3. Software. 4. Desenvolvimento. I. Melo, Guilherme Nunes . (Orientação). II. Título.

620 CDD (22.ed.)

Daniel Costa Sampaio

Sistema de detecção de queda utilizando a plataforma Toit

Trabalho de Conclusão apresentado ao Curso de Graduação em Engenharia Eletrônica, do Departamento de Eletrônica e Sistemas, da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em Engenharia Eletrônica.

Aprovado em: 17/04/2023

Banca Examinadora

Documento assinado digitalmente
gov.br
GUILHERME NUNES MELO
Data: 11/05/2023 14:03:35-0300
Verifique em https://validar.iti.gov.br

Prof. Guilherme Nunes Melo, D.Sc.
Universidade Federal de Pernambuco

Documento assinado digitalmente
gov.br
JOAO MARCELO XAVIER NATARIO TEIXEIR
Data: 11/05/2023 12:45:20-0300
Verifique em https://validar.iti.gov.br

Prof. João Marcelo Xavier Natário Teixeira , D.Sc.
Universidade Federal de Pernambuco

Agradecimentos

Primeiramente gostaria de agradecer à minha família pelo apoio e compreensão durante todo o processo de elaboração deste trabalho. Eles foram minha fonte de inspiração e força para continuar lutando pelos meus objetivos.

Gostaria de agradecer ao meu orientador, Guilherme Nunes Melo, pela orientação e apoio incansável durante todo o desenvolvimento deste trabalho. O seu conhecimento e experiência foram fundamentais para o sucesso deste projeto.

Agradeço também aos meus colegas de turma, Paulo Guedes, Thiago Cavalcanti e Gabriel Dutra, pelas discussões e ideias que foram fundamentais para o desenvolvimento do meu trabalho.

Por fim, agradeço a todos aqueles que de alguma forma contribuíram direta ou indiretamente para a conclusão deste trabalho.

Resumo do Trabalho de Conclusão de Curso apresentado ao Departamento de Eletrônica e Sistemas, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Eletrônica(Eng.)

Sistema de detecção de queda utilizando a plataforma Toit

Daniel Costa Sampaio

Atualmente, a sociedade tem se tornado cada vez mais dependente de sistemas embarcados. Tais sistemas podem representar desde simples aparelhos domésticos até aeronaves. Sistemas embarcados diferem bastante da maioria dos sistemas de aplicação *Desktop*, pois devem ser altamente otimizados para o ciclo de vida, devem atender restrições temporais e de consumo de energia, e devem ainda tratar de limitações de recursos, tais como tamanho e peso. No contexto de criação de *firmwares* para sistemas embarcados, as instruções de códigos são restritas ao *hardware* usado, existem *bugs* de *firmware* escondidos e longos ciclos de compilação, o que leva a uma má experiência de desenvolvimento. Esse trabalho visa explorar a nova tecnologia Toit, que tem como objetivo otimizar o processo de desenvolvimento de firmwares embarcados, trazendo mais semelhanças com o mercado de desenvolvimento de softwares. Esse novo enfoque promete diferencias, melhorias e novas tendências para o desenvolvimento de sistemas embarcados, e é apresentado ao longo do trabalho em conjunto com o desenvolvimento do sistema de detecção de quedas utilizando um sensor de acelerômetro. Essa abordagem inovadora pode ter grande impacto no campo dos sistemas embarcados, tornando o processo de desenvolvimento mais eficiente, acessível e adaptado às necessidades do mercado atual.

Palavras-chave: sistemas embarcados; desenvolvimento; Toit; *firmware*; *software*

.

Abstract of Course Conclusion Work, presented to Departament of Eletronic and Systems, as a partial fulfillment of the requirements for the degree of Bachelor of Electronic Engineering(Eng.)

Fall detection system using the Toit platform

Daniel Costa Sampaio

Currently, society has become increasingly dependent on embedded systems. These systems can range from simple household appliances to aircraft. Embedded systems differ significantly from most desktop application systems, as they must be highly optimized for lifecycle, meet time and power consumption constraints, and still deal with resource limitations such as size and weight. In the context of creating firmware for embedded systems, code instructions are restricted to the hardware used, there are hidden firmware bugs, and long compilation cycles, leading to a poor development experience. This work aims to explore the new technology Toit, which aims to optimize the process of developing embedded firmware, bringing more similarities to the software development market. This new approach promises differences, improvements, and new trends for the development of embedded systems, and is presented throughout the work in conjunction with the development of the fall detection system using an accelerometer sensor. This innovative approach can have a great impact in the field of embedded systems, making the development process more efficient, accessible, and adapted to the needs of the current market.

Keywords: embedded systems; development; Toit; *firmware*; *software* .

Sumário

Lista de Ilustrações

Lista de Abreviações

IoT	*Internet Of Things*
OTA	*Over The Air*
IDE	*Integrated Development Environment*
ESPIDF	*Espressif IoT Development Framework*
ESP32	*Espressif Systems Single-Chip System*
AHT10	*Sensor digital de temperatura e umidade*
OLED	*Organic Light Emitting Diode*
u8g2	*Universally 8bit Graphics Library 2*
SSD13	*Solomon Systech Display Controller*
AHT20	*Sensor digital de temperatura e umidade*
Wi-Fi	*Wireless Fidelity*
TSMC	*Taiwan Semiconductor Manufacturing Company*
RF	*Radio Frequency*
MQ135	*Sensor de gás para amônia, Álcool, Benzeno e fumaça*
SnO_2	*Dióxido de estanho*
Ni-Cr	*Níquel-Cromo*
CO	*Monóxido de carbono*
CO_2	*Dióxido de carbono*
CLI	*Command Line Interface*
APP	*Application*
FreeRTOS	*Free Real-Time Operating System*

Capítulo 1

Introdução

1.1 Motivação

A ideia que está por trás da evolução tecnológica é criar invenções que tornem a vida da sociedade mais fácil. De certa forma, a busca pelo conhecimento só faz sentido quando seu objetivo é melhorar a sociedade. Com o passar do tempo, a necessidade de evolução tecnológica tem se tornado cada vez mais recorrente, chegando ao ponto em que em uma única geração podemos ver mais de uma revolução tecnológica, como observado pelo inventor, empresário e futurista Ray Kurzweil: "O ritmo da tecnologia está acelerando. O que levou séculos para se desenvolver leva agora apenas alguns anos, e o que levou anos agora leva apenas meses". Ray Kurzweil é cofundador e diretor de engenharia da Google e autor de vários livros sobre tecnologia e futurismo.

O setor de sistemas embarcados está vivendo uma verdadeira revolução nos dias de hoje, graças à tendência crescente de robotização e automação nos processos produtivos e de serviços. Essa revolução é conhecida como Indústria 4.0 ou quarta revolução industrial, e engloba uma série de tecnologias voltadas para a automação e troca de dados, como sistemas ciber-físicos, internet das coisas e computação em nuvem.

Com a busca cada vez maior pelo uso da Indústria 4.0 e com a tendência de dispositivos conectados à rede, é comum o surgimento de novas tecnologias que visam

facilitar e agilizar o desenvolvimento de sistemas embarcados. Essas tecnologias são fundamentais para aproveitar as oportunidades e superar os desafios trazidos pela revolução tecnológica.

No contexto do desenvolvimento de sistemas embarcados, saber como criar um sistema de detecção de quedas pode ser muito vantajoso para um desenvolvedor. Com a crescente necessidade de dispositivos conectados à rede e a busca pela automação dos processos produtivos, é cada vez mais importante desenvolver sistemas que possam garantir a segurança das pessoas e evitar acidentes. Além disso, a habilidade de criar um sistema de detecção de quedas pode ser uma competência valiosa para o currículo de um desenvolvedor, pois essa tecnologia pode ser aplicada em diversos contextos, como em sistemas de monitoramento de idosos ou em equipamentos de proteção individual para trabalhadores.

Tendo isso em mente, um dos cenários de maior desvantagens de desenvolver *softwares* para sistemas embarcados, é o fato de que o sistema embarcados geralmente operam em tempo real e em ambientes com recursos limitados, o que requer um conhecimento especializado para otimizar o desempenho e a eficiência do software. Além disso, a fase de testes costuma ser longa e envolver uma ampla variedade de cenários, a fim de garantir o funcionamento adequado do sistema. É importante ressaltar que, uma vez que o dispositivo está na fase de testes, não há uma forma eficiente para armazenar ou revisar informações sobre os erros ocorridos. Por exemplo, se um erro no software do dispositivo só ocorre após 24 horas de uso, o desenvolvedor precisa manter o dispositivo conectado ao computador via porta USB para obter as informações necessárias para identificar a causa raiz do problema, o que torna o ciclo de desenvolvimento maior e mais desafiador.

Considerando a necessidade de escolher um *framework* que apresente soluções eficazes para o desenvolvimento do sistema de detecção de quedas, foi escolhido o *Toit* que vem ganhando destaque no desenvolvimento de sistemas embarcados, uma vez que oferece recursos como a possibilidade de acesso remoto ao monitor serial do dispositivo. Essa funcionalidade possibilita uma visualização mais prática dos

códigos de erro gerados pelo dispositivo, acelerando o processo de identificação e correção de falhas e tornando mais ágil a fase de testes além de outras vantagens durante o ciclo de desenvolvimento do produto que irão ser abordadas ao longo do trabalho.

1.2 Objetivo Geral

Desenvolver um sistema de detecção de queda, analisando o funcionamento da *Toit*, como uma opção de ambiente de desenvolvimento.

1.2.1 Objetivos Específicos

- Mostrar diferenciais da *Toit* que agregam ao processo de desenvolvimento;
- Configurar dependências da *Toit* para desenvolvimento de aplicações;
- Comparar a *Toit* com *Arduino IDE*;
- Implementar um sistema de detecção de queda em *Toit Language*.

1.3 Organização do TCC

O conteúdo deste TCC está dividido em 4 capítulos. As referências encontram-se nas páginas finais. A seguir, um resumo dos capítulos seguintes do TCC.

Capítulo 2 - Fundamentação Teórica. Contém a fundamentação teórica de conceitos utilizados para o entendimento e composição do projeto.

Capítulo 3- Desenvolvimento. Contém a análise dos benefícios que a *Toit* traz ao desenvolvedor, além de apresentar o desenvolvimento do projeto exemplo e mostrar os resultados obtidos.

Capítulo 4 - Considerações Finais. Expõe as considerações finais referentes ao trabalho desenvolvido, além de propor trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 IoT

A Internet das Coisas (IoT) é um conceito cada vez mais presente em nosso cotidiano, e tem sido amplamente estudado na área da tecnologia da informação. Segundo o artigo "*Internet of Things is a revolutionary approach for future technology enhancement: a review*" (Kumar e Zymbler, 2019), IoT é definido como "um paradigma emergente que permite a comunicação entre dispositivos eletrônicos e sensores através da Internet, a fim de facilitar as nossas vidas".

A implementação de IoT tem sido possível devido ao desenvolvimento de tecnologias de sensores, dispositivos de baixo custo e conectividade cada vez mais acessível. O IoT visa facilitar a comunicação e a troca de informações para permitir novas formas de interação entre as coisas e as pessoas (Cirani et al., 2014).

Além disso, a implementação de IoT tem gerado novas oportunidades para a indústria, como a melhoria da eficiência operacional e a criação de novos modelos de negócios. De acordo com o estudo "*A decade of research on patterns and architectures for IoT security*" (Rajmohan e Ferry, 2022), A maioria das infra-estruturas críticas apontadas na Diretiva da União Europeia sobre segurança de redes e sistemas de informação, tais como para energia, água, transporte e saúde, são ou serão baseadas em IoT. Por exemplo, as cidades inteligentes estão integrando sensores IoT com análise para racionalizar os gastos e melhorar a eficiência da infra-estrutura.

Porém, a implementação de IoT também traz desafios, como questões de segurança e privacidade. Com a crescente quantidade de dispositivos IoT conectados à internet, a segurança tem se tornado uma preocupação cada vez mais importante. Os dispositivos IoT são frequentemente alvos de ataques cibernéticos, como roubo de dados, ataques de negação de serviço e invasões de privacidade, que podem resultar em perda de receita, danos à reputação e perda de confiança dos clientes (Kandasamy e Achuthan, 2020).

Em resumo, IoT é uma realidade cada vez mais presente em nossas vidas e tem sido objeto de muitos estudos na área de tecnologia da informação. A implementação de IoT tem sido possível graças ao avanço das tecnologias de sensores, dispositivos de baixo custo e conectividade cada vez mais acessíveis. Além disso, a adoção de IoT tem gerado novas possibilidades para a indústria, mas também traz desafios, como problemas de segurança e privacidade.

2.2 Sistemas Embarcados

Os sistemas embarcados são sistemas computacionais que são incorporados em dispositivos ou equipamentos para realizar uma tarefa específica. Segundo o livro "Embedded Systems Design" (Heath, 2003), "um sistema embarcado é um sistema baseado em microprocessador que é construído para controlar uma função, ou intervalo de funções, e não é projetado para ser programado pelo usuário final da mesma forma que um computador pessoal".

A implementação de sistemas embarcados exige a combinação de conhecimento de *hardware* e *software*, o que leva a um aumento na funcionalidade e complexidade dos sistemas embarcados. Devido a isso, o desenvolvimento eficiente de produtos de sistemas embarcados tem se tornado cada vez mais desafiador. A combinação de diferentes tecnologias e a necessidade de atender a requisitos específicos dos sistemas embarcados torna cada vez mais difícil o desenvolvimento eficiente de produtos de sistemas embarcados (Kaisti e Mujunen, 2013).

Os sistemas embarcados apresentam desafios devido às limitações de recursos,

como memória e capacidade de processamento. Além disso, a segurança é uma preocupação crescente nesses sistemas, devido ao aumento de ameaças cibernéticas. Com o avanço da tecnologia, os sistemas embarcados estão se tornando cada vez mais complexos e variados, exigindo maior atenção em relação à confiabilidade, segurança e proteção (Guessi e Maldonado, 2012).

Em resumo, os sistemas embarcados são sistemas computacionais que são incorporados em dispositivos ou equipamentos para realizar uma tarefa específica, e são utilizados em uma ampla variedade de aplicações. A implementação de sistemas embarcados exige conhecimento de *hardware* e *software* e os desenvolvedores devem ser capazes de gerenciar restrições de recursos e implementar medidas de segurança para garantir a confiabilidade e segurança do sistema.

2.3 Microcontrolador

Os microcontroladores são dispositivos integrados que combinam um microprocessador, memória e periféricos em um único chip. Eles são amplamente utilizados em aplicações como automação industrial, automóveis, eletrodomésticos, dispositivos médicos e dispositivos de consumo (Matheus, 2020).

A história dos microcontroladores começa com a criação do circuito integrado, em 1958. A evolução dos circuitos integrados permitiu a incorporação de mais componentes em um único chip, o que levou ao desenvolvimento de microprocessadores. Os primeiros microprocessadores eram geralmente utilizados em computadores pessoais, mas logo foram adaptados para outras aplicações, como automação industrial e sistemas embarcados (Wikipedia, 2021).

A criação dos microcontroladores foi uma evolução natural do desenvolvimento dos microprocessadores. Diferentemente dos microprocessadores, os microcontroladores possuem todos os componentes necessários para o funcionamento de um sistema embarcado em um único chip, incluindo memória, periféricos e um processador. Isso tornou possível o desenvolvimento de sistemas embarcados mais compactos e eficientes (Wikipedia, 2021).

Um dos principais benefícios dos microcontroladores é a capacidade de personalização. Eles podem ser programados para realizar tarefas específicas, tornando-os ideais para aplicações específicas. Além disso, devido ao seu tamanho compacto e baixo consumo de energia, os microcontroladores são amplamente utilizados em aplicações de IoT (Matheus, 2020).

2.4 *Firmware*

O *firmware* é um código que é armazenado em dispositivos eletrônicos, como microcontroladores, dispositivos de armazenamento, dispositivos de rede e dispositivos de entrada/saída. Ele é responsável por controlar o *hardware* e fornecer funcionalidades ao sistema. O *firmware* é específico para cada dispositivo e é escrito de acordo com as especificações do desenvolvedor(Tim, 2023).

Uma das principais características do *firmware* é que ele é imutável, ou seja, ele não pode ser alterado pelo usuário final. Isso o difere de *software*, que pode ser atualizado e modificado pelo usuário. Isso permite que o *firmware* tenha um controle mais preciso sobre o *hardware* e garanta a estabilidade do sistema.

O *firmware* também é responsável por garantir a segurança do sistema. Ele é projetado para evitar acessos não autorizados e garantir a confidencialidade e integridade dos dados. Isso é especialmente importante em sistemas críticos, como sistemas de automação industrial, sistemas de transporte e sistemas de saúde (Wikipedia, 2020b).

2.5 *Toit*

Toit é uma plataforma de desenvolvimento de IoT que permite instanciar *containers* em microcontroladores. O *Firmware* da *Toit* simula uma máquina virtual em cima do sistema operacional do microcontrolador (Figura 2.1). Isso faz com que os algoritmos criados sejam independentes, e isolados entre si, levando a um ambiente mais controlável com foco em desenvolvimento de *software* (Toit, 2021e).

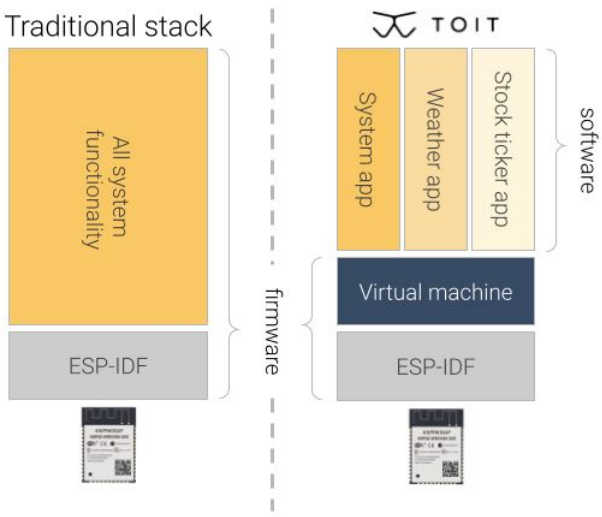

Figura 2.1: *Toit* Stack Fonte: toit.io

Através da plataforma e de ferramentas desenvolvidas para aplicações *Toit* (Figura 2.2) pode-se realizar atualizações na camada da assim intitulada pela *Toit* camada de *Software* (Figura 2.1) do seu microcontrolador, tornando obsoleto se conectar via cabo ao microcontrolador e ter que realizar o *upload* de toda a imagem do *Firmware* cada vez que o código é alterado.

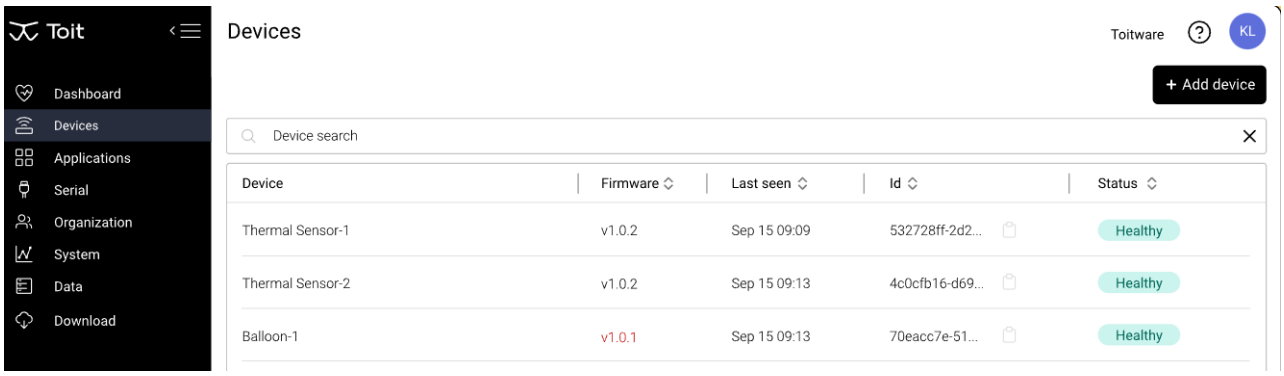

Figura 2.2: Plataforma Fonte: toit.io

Para tudo isso funcionar, a *Toit* criou sua própria linguagem pensada para IoT que é a *Toit language*, uma linguagem de programação de alto nível com uma sintaxe semelhante à de *python* (Toit, 2021a). E também o pacote de ferramentas *Jaguar* que permite o *upload* ao vivo ao microcontrolador podendo assim atualizar seu código em menos de dois segundos via Wi-Fi (Toit, 2021d).

2.6 Updates-*Over the Air* (OTA)

OTA (Atualização Over the Air) é uma tecnologia que permite a atualização de *software* em dispositivos sem a necessidade de conectar fisicamente o dispositivo a um computador. Isso é possível através da transmissão de dados via rede sem fio, como Wi-Fi ou celular. A atualização OTA é muito importante para garantir a segurança e a funcionalidade dos dispositivos, pois permite corrigir *bugs* e adicionar novas funcionalidades sem a necessidade de acesso físico ao dispositivo (Otavio, 2019).

Além disso, a atualização OTA é uma ferramenta valiosa para fabricantes de dispositivos, pois permite fornecer suporte e melhorias contínuas para seus produtos, mesmo depois de vendidos. Isso ajuda a garantir a satisfação dos clientes e aumenta a confiança na marca (Erica, 2022).

A implementação de atualização OTA também é importante para dispositivos IoT, pois esses dispositivos podem ser atualizados remotamente e sem a necessidade de intervenção humana. Isso é especialmente importante para dispositivos críticos, como equipamentos industriais e de segurança, pois garante a continuidade do funcionamento e a segurança desses dispositivos(Otavio, 2019).

No entanto, a implementação de atualização OTA também traz desafios, como questões de segurança e privacidade. É importante garantir que as atualizações sejam seguras e autênticas, para evitar a instalação de *software* malicioso. Além disso, é necessário garantir a privacidade dos usuários, evitando a coleta indevida de dados. A implementação de mecanismos de criptografia e autenticação é essencial para garantir a segurança e privacidade dos dispositivos atualizados via OTA (Erica, 2022).

Capítulo 3

Desenvolvimento

3.1 *Toit*

A *Toit* nasce com a proposta de revolucionar a forma de desenvolver sistemas embarcados, trazendo uma perspectiva de desenvolvimento muito mais ligada a *softwares* do que *firmwares*. Essa nova abordagem de desenvolvimento só se faz possível com a implementação de um sistema operacional no microcontrolador que é responsável por fazer a conexão através de *Cloud APIs* da *Toit* (Toit, 2021e).

Outro aspecto que propicia o desenvolvimento de dispositivos através da *Toit* é o rumo no qual o mercado de sistema embarcados está seguindo, com a crescente tendência que os dispositivos se tornem cada vez mais propícios à se conectar a rede de internet (Emyle e Mateus, 2022). Sendo assim a *Toit* vem com a ideia de trazer *serviceability* para os dispositivos embarcados, que é a capacidade de instalar, configurar, monitorar, identificar exceções ou falhas, depurar ou isolar falhas para análise de causa raiz e fornecer manutenção de *software* em busca de resolver um problema e restaurar o produto em serviço(Toit, 2021b).

3.2 Diferenciais da *Toit*

A *Toit* traz uma abordagem inovadora para o ciclo de desenvolvimento de sistemas embarcados, apostando em trazer uma semelhança maior com o ciclo de desenvolvimento de *softwares*. Enquanto as formas tradicionais de desenvolver sistemas embarcados tem como fundamentos (Florian, 2022):

- Desenvolvimento de *firmwares* onde não há uma separação bem definida entre sistema operacional, *drivers* e a aplicação;
- Linguagens de programação de baixo nivel (c, c++, *assembly*);
- Longo tempo de *upload* do código em comparação ao desenvolvimento de *softaware*;
- Erros de aplicação, geralmente resultam em *crashing* de todo o programa;
- Precisa estar conectado ao dispositivo por serial, para realizar o *upload* de novos códigos.

A *Toit* traz como diferencial os seguintes aspectos:

- Separação entre as camadas de *drivers*, sistema operacional e aplicação;
- Linguagem de programação de alto nível;
- Desenvolvimento remoto.

3.2.1 Separação entre as camadas de *drivers*, sistema operacional e aplicação

A *Toit* se destaca por sua arquitetura de separação dos *drivers* e sistema operacional em relação à aplicação, o que permite uma redução significativa no tempo de compilação e gravação do código no microcontrolador. Isso é possível pois os *drivers* já estão pré-gravados no microcontrolador, não sendo necessário uma nova gravação toda vez que o código da aplicação é alterado.

Para comprovar essa afirmação, foram realizadas comparações com outras plataformas de desenvolvimento, como o *Arduino IDE* (Arduino, 2023) e a plataforma *Platformio* (Espressif, 2023). O teste utilizado foi o código *Blink*, que faz piscar o led do *Devkit ESP32 WROOM 32* (Figura 3.7) e foi realizado três vezes em cada ambiente de programação. Os resultados mostraram (Figura 3.1) que a *Toit* levou em média 3,34 segundos para fazer o *upload* do código, o que representa uma redução de aproximadamente 80% no tempo de compilação e gravação em comparação com o *Arduino IDE* e 87% em comparação com o *Platformio*.

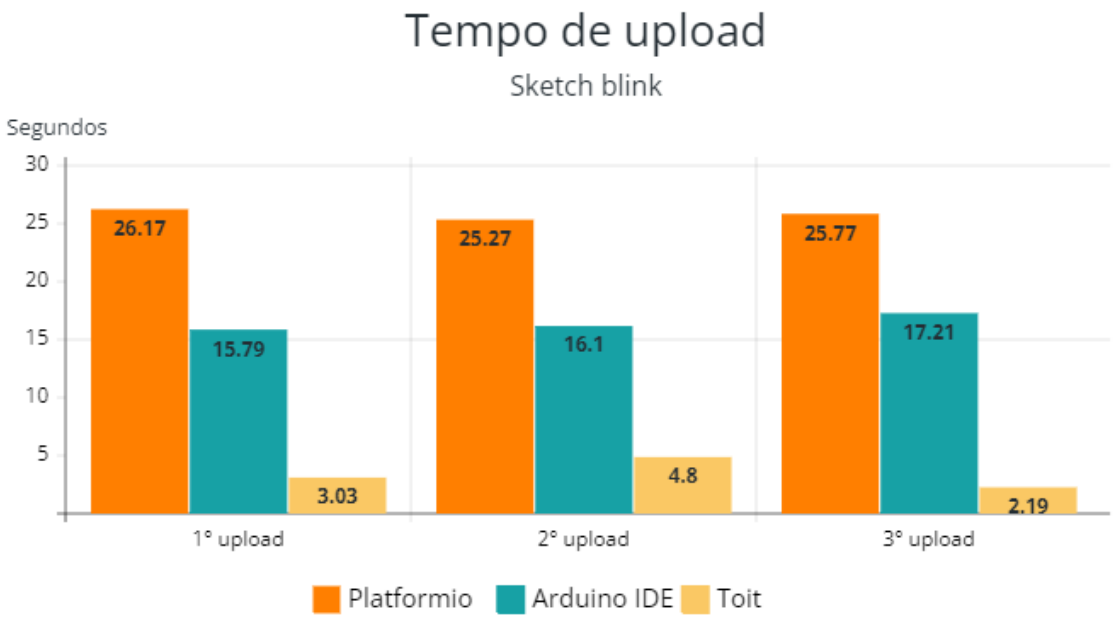

Figura 3.1: Tempo de *upload* para ESP32 entre os *framework Arduino IDE*, *ESPIDF* e *Toit*
Fonte: Autor

Esse ganho de desempenho é atribuído, em partes, à separação das camadas de aplicação e *drivers* e sistema operacional, mas também à utilização da ferramenta *Jaguar*, um pequeno aplicativo *Toit* que permite a atualização e reinicialização do código do microcontrolador via WiFi, Utilizando-se da máquina virtual *Toit* para permitir que você atualize e reinicie o código do microcontrolador.

3.2.2 Linguagem de programação de alto nível

Toit é uma linguagem de programação de alto nível orientada a objetos de código aberto, pensada para a Internet das Coisas (Toit, 2022a). Nela está presente um *Gar-*

bage Collector (Toit, 2022c) que é uma forma de realizar o gerenciamento automático de memória, Ou seja, sua principal função é descartar os espaços de memória alocados, que não são mais usados pela aplicação (Wikipedia, 2023). Nas linguagens C/C++ esse processo é feito pelo desenvolvedor.

Para se ter uma melhor análise comparativa, foi desenvolvido um exemplo que envolve periféricos de entrada e de saída, que é um sistema recorrente no âmbito de sistemas embarcados. O periférico de entrada escolhido foi um sensor de temperatura e umidade (ASAIR, 2018) e o periférico de saída foi um display OLED (SOLOMON, 2009).

Realizar esse experimento foi desafiador devido à necessidade de trabalhar com duas linguagens de programação diferentes, cada uma com suas próprias bibliotecas e peculiaridades. Além disso, houve a necessidade de desenvolver o código para dois ambientes de desenvolvimento distintos, o que exigiu um esforço adicional para garantir que as funcionalidades estivessem disponíveis em ambos os ambientes.

Embora o código tenha sido relativamente curto, com 29 e 32 linhas de código para Arduino IDE e Toit, respectivamente, o processo de desenvolvimento exigiu um bom conhecimento das linguagens de programação e das bibliotecas envolvidas. Além disso, foi necessário lidar com possíveis erros e falhas durante a implementação, o que pode ter adicionado ainda mais complexidade ao processo.

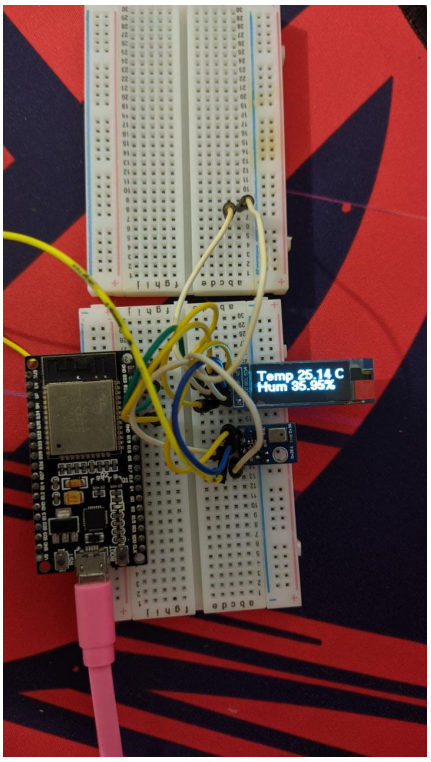

Figura 3.2: Display oled de temperatura e umidade fonte: Autor

O algoritmo de programação desenvolvido adiquire por meio do protoclo I2C os dados do sensor e exibe no display como mostra a Figura 3.2. Por fins de praticidade, somente foi desenvolvido essa aplicação para *Arduino IDE* e *Toit*, onde para o *Arduino IDE* o código gerado na linguagem *C/C++*, usando as bibliotecas AHT10 (Limor, 2020) e u8g2 (Olikraus, 2021). Para a *Toit* o código gerado na linguagem *Toit language*, usando as bibliotecas SSD13 (Floitsch, 2021) e AHT20 (Davidlao2k, 2022).

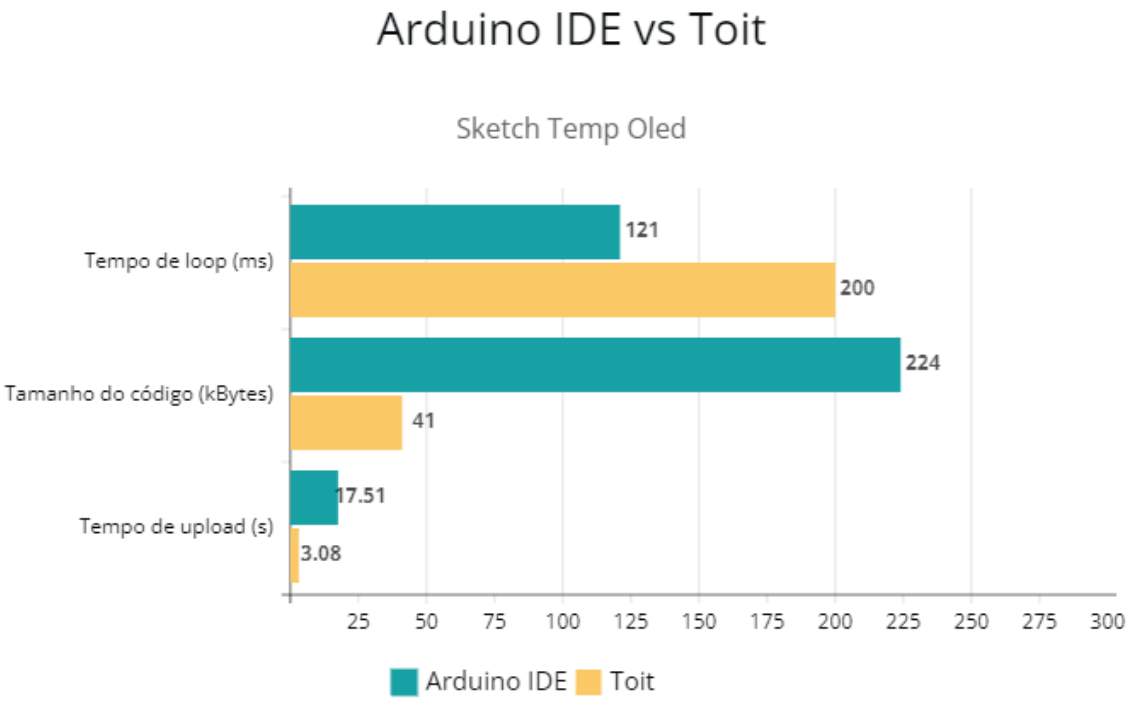

Figura 3.3: Arduino IDE vs Toit Fonte: Autor

Como pode ser visto na Figura 3.3 a diferença entre o tempo de *upload* se mantém próxima da comparação da Figura 3.1 enquanto que a diferença do tamanho do código compilado é mais de 5 vezes maior no Arduino IDE, porém o tempo que leva para o a função de *loop()* ser lida é mais lento no *Toit*.

3.2.3 Desenvolvimento remoto

A *Toit* possui uma característica inovadora que permite aos desenvolvedores criar *software* de forma remota em dispositivos(Toit, 2022b). Isso oferece uma ampla gama de possibilidades para os desenvolvedores, pois eles podem desenvolver e testar o *software* ao mesmo tempo, mesmo que o dispositivo esteja em um ambiente desfavorável. Dessa forma, é possível criar um ciclo de desenvolvimento mais próximo da realidade do dispositivo, trazendo dinâmica e agilidade ao processo de criação de *software*.

Para mostrar essa funcionalidade foi desenvolvido um projeto de medidor da qualidade do ar, que detecta os niveis de concentração de CO, CO_2, tolueno, álcool e acetona. O desenvolvimento desse projeto foi desafiador devido à necessidade de

adaptação de bibliotecas e reescrita de código na linguagem *Toit*, além da necessidade de criar um ambiente controlado para realização de testes. A característica inovadora da linguagem *Toit* de permitir o desenvolvimento remoto em dispositivos, embora traga dinamicidade ao processo, também apresentou desafios adicionais em relação ao desenvolvimento e teste do software.

Tendo isso em mente, foi realizada uma adaptação da biblioteca já existente do sensor de gás MQ135 para o Arduíno, chamada *MQUnifiedsensor.h* (Phoenix1747, 2021). A fim de utilizar essa biblioteca no projeto, foi necessário reescrevê-la na linguagem *Toit* que pode ser vista no Apêndice A.2.

Por ser um sistema que necessita de um ambiente controlado para realizar testes, foi utilizado um recipiente semi hermiticamente fechado para ser o ambiente de teste de validação de funcionamento do sistema. Tendo isso em mente esses foram os resultados apresentados no monitor serial da plataforma *Toit* Figuras 3.4, 3.5, 3.6.

Vale ressaltar que o circuito do sistema não estava diretamente ligado ao computador, o que traz pra esse sistema uma liberdade de poder testar em qualquer ambiente, no qual, seja controlado pelo desenvolvedor.

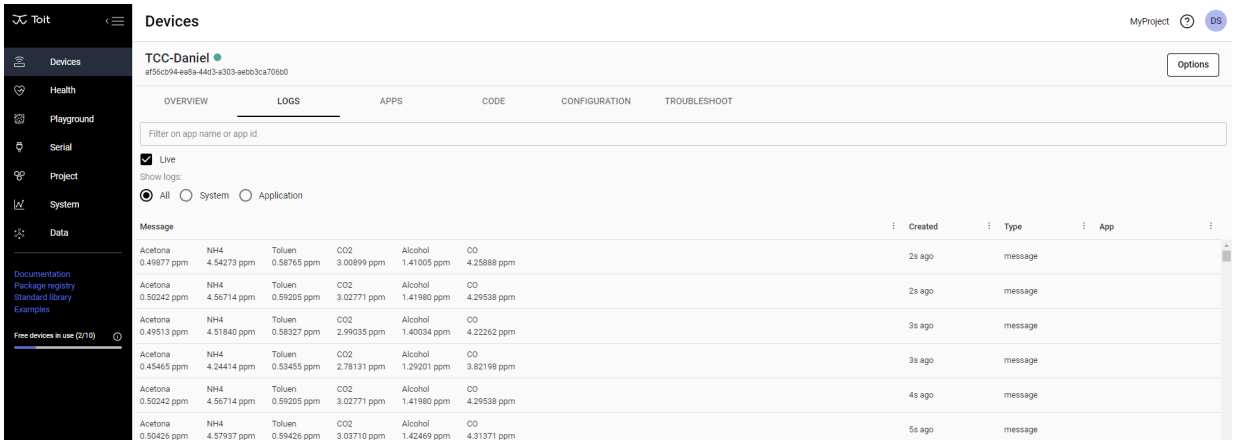

Figura 3.4: APP - Ler Sensor de Gas Fonte: Autor

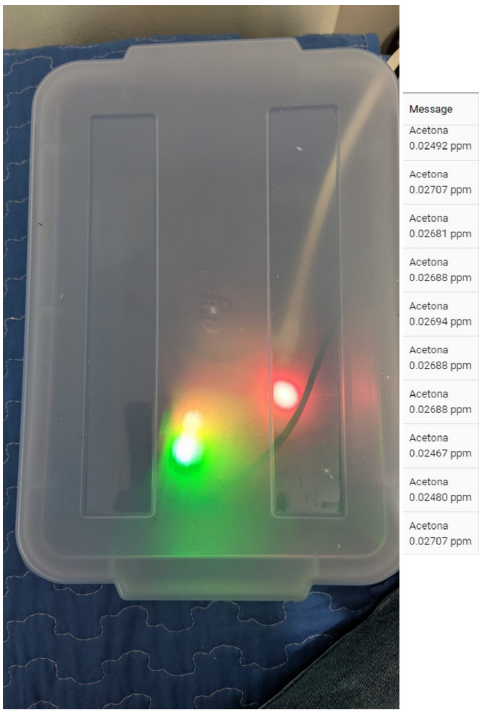

Figura 3.5: 1° cenário com o recipiente sem algodão embebido de acetona e o monitor serial Fonte: Autor

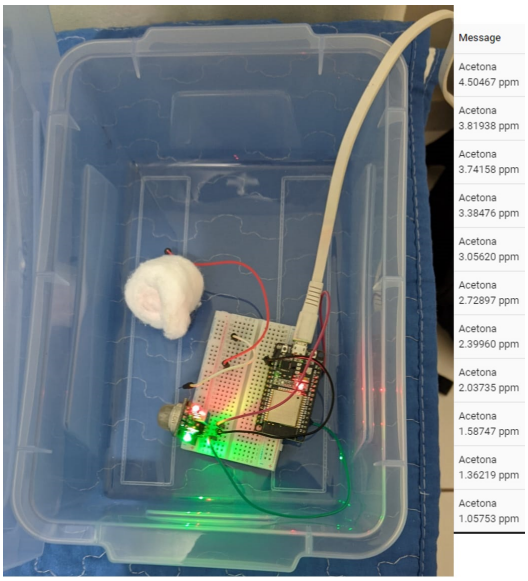

Figura 3.6: 2° cenário recipiente com algodão embebido de acetona e o monitor serial Fonte: Autor

3.3 Sistema de detecção de quedas

3.3.1 Motivação

Um sistema de detecção de quedas, tem como base a análise de ocorrência de um evento indesejado que é a queda. Esse sistema tem diversas aplicações, como é o caso dos HD (*Hard Drive*) que usam sensores para recolher as ponteiras que faziam contato com o disco, para evitar que danificasse o produto, caso o dispositivo detectasse que estava em queda livre (Wikipedia, 2020a).

Além disso é importante destacar a ampla variedades de eventos em que esse sistema pode ser útil. Em outras palavras trata-se de uma tecnologia que pode trazer melhorias para diferentes áreas.

3.3.2 Hardware utilizados

ESP32

O ESP32 é um único chip combinado de Wi-Fi e Bluetooth de 2,4 GHz projetado com o TSMC de 40 nm de ultrabaixa potência tecnologia (Espressif, 2016). Ele foi projetado para alcançar o melhor desempenho de potência e RF, mostrando robustez, versatilidade e confiabilidade em uma ampla variedade de aplicações e cenários de energia. Na placa utilizada tem-se o Devkit ESP32 WROOM 32 (Figura 3.7) com antena embutida, uma interface usb-serial e um regulador de tensão para 3,3V.

Figura 3.7: ESP32 Fonte: cdn.awsli.com.br

MPU 6050

O MPU6050 é um dispositivo que combina um acelerômetro de 3 eixos, um giroscópio de 3 eixos e um sensor de temperatura em um único chip MEMS (sistema microeletromecânico). Ele é usado para medir a orientação, a velocidade angular e a aceleração de objetos em movimento (InvenSense, 2013).

Em termos de física, o acelerômetro mede a aceleração linear, ou seja, a mudança na velocidade ao longo do tempo, em três eixos - X, Y e Z. Ele funciona detectando a força da gravidade em cada um dos três eixos, que é equivalente à aceleração devido à gravidade na Terra (9,81 m/s^2), e a diferença dessa força em relação à força devido à aceleração. Isso permite que o acelerômetro determine a aceleração em relação a cada um dos três eixos(João, 1999).

Para obter as medidas, o MPU6050 usa um processador de sinal digital (DSP) para converter os sinais analógicos dos sensores em valores digitais, que podem ser lidos por um microcontrolador através de uma interface I2C ou SPI. Ele também inclui recursos de filtro e compensação para reduzir o ruído nas medições(InvenSense, 2013).

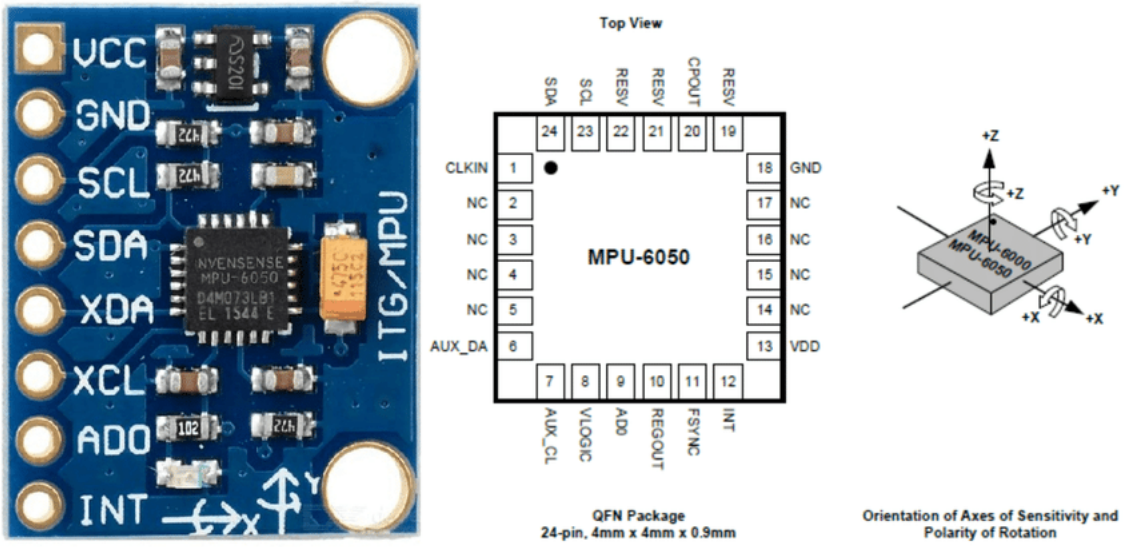

Figura 3.8: MPU6050 Fonte: *researchgate.net*

3.3.3 Organizando ambiente *Toit*

Para configurar o ambiente Toit é necessário fazer um cadastro no site *toit.io* e lá realizar um *upload* do *Toit firmware* no seu microcontrolador através da interface serial, nesse *firmware* contém toda a abstração de uma máquina virtual que instanciará *containers* no seu microcontrolador. Após o processo de inserção do *firmware*, pode-se energizar seu microcontrolador em qualquer lugar que tenha o sinal WiFi, que já estará preparado para receber atualizações de *firmware* customizados *Over The Air* (Toit, 2021c).

Tem duas opções para realizar um upload do seu *firmware* personalizado, a 1° opção é através da plataforma (*toit.io*) na seção *device*, na aba código (Figura 3.9), que é um editor *web* sem muitos recursos (Toit, 2021c).

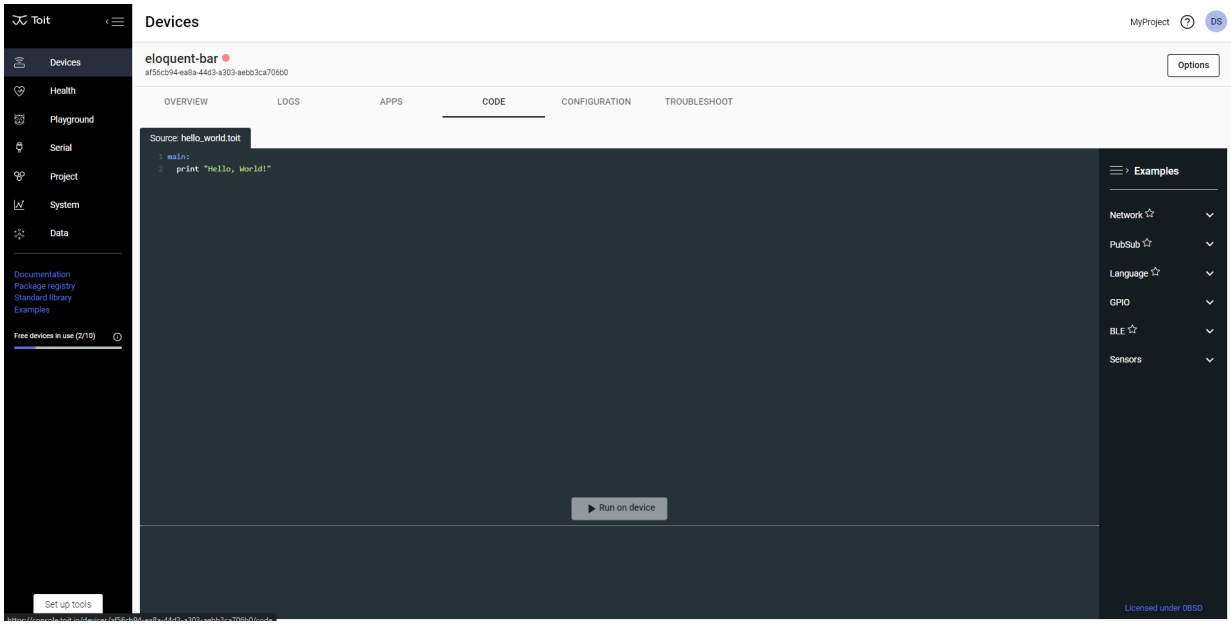

Figura 3.9: Editor de texto dentro da plataforma Toit Fonte: Autor

A 2° opção é baixar o editor de texto *Visual Studio Code* e instalar a extensão do *Toit* (Figura 3.10) e já terá acesso ao *Toit CLI (Comand line interface)*, capaz de fazer *upload* dos códigos para o microcontrolador através de uma linha de comando (Toit, 2021c).

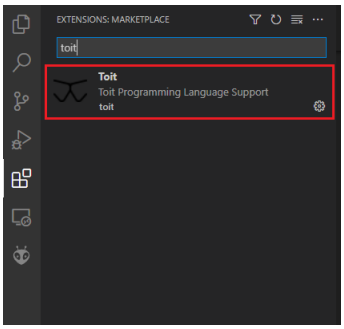

Figura 3.10: Extensão do Toit no Visual Studio Code Fonte: Autor

3.4 Implementação

Para fins de experimentação, foi realizado a construção de um protótipo utilizando uma *protoboard*. Os pinos VCC, GND, SCL e SDA do MPU6050 foram conectados a placa ESP32, como evidenciado pela Figura 3.11. Este processo foi realizado com o objetivo de testar a funcionalidade do dispositivo em um ambiente controlado, permitindo uma avaliação mais precisa e detalhada das suas capacidades.

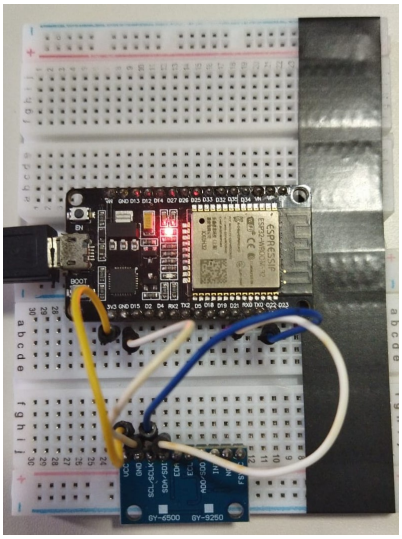

Figura 3.11: Protótipo do sistema de detecção de quedas

Para detectar um evento de queda do dispositivo, é necessário usar os valores da aceleração do corpo nos eixos: X, Y e Z. Fazendo o calculo da aceleração resultante, através da formula

$$|a| = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{3.1}$$

O MPU6050 mede a aceleração em cada eixo em unidades de g (gravidade), onde 1g é igual à aceleração da gravidade. Portanto no caso que o corpo esteja em queda livre, a aceleração total do corpo será aproximadamente 0, pois ele estará sobre efeito somente da força da gravidade.

$$|a| = \sqrt{a_x^2 + a_y^2 + a_z^2} \approx 0 \tag{3.2}$$

Para determinar qual valor será a condição para detectar a queda do objeto, foi plotado o valor da aceleração total e realizado a queda do objeto, e o gráfico gerado foi o da Figura 3.12, que como pode-se ver, o limite para detectar a queda está próximo de 0,2 do valor da aceleração.

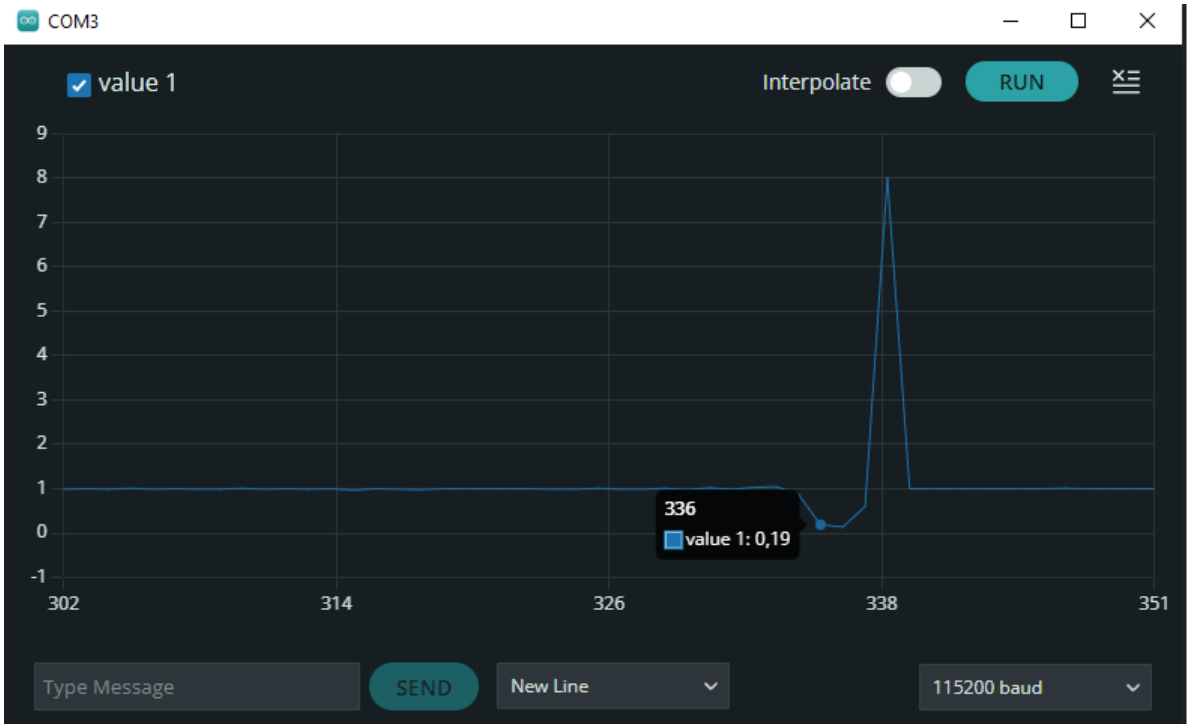

Figura 3.12: Aceleração total do objeto mediante a uma queda

3.4.1 APP - Ler Sensor MPU6050

Os *APP* assim chamados pelo *Toit*, são onde o código é executado como um ou mais aplicativos, dentro da máquina virtual Toit. A máquina virtual cria um ambiente de área restrita para seu código. Os *APP* podem ser comparados as *Task* do sistema operacional FreeRTOS, que são um pequeno algoritmo independente das outras tarefas do sistema, porém os *APP* tem uma vantagem que são independentes do sistema como um todo, tornando um erro associado ao *APP* travando somente o *APP* e não interferindo no funcionamento do resto do sistema (Toit, 2021f).

Para o código do sistema de detecção de quedas, foi realizado leituras dos valores de aceleração nos eixos x,y e z, e transformados para aceleração total do corpo segundo a fórmula 3.2, quando essa aceleração é menor que 0.2, uma mensagem é exibida no monitor serial, informando que o objeto está caindo. Quando a aceleração volta a crescer é exibido no monitor serial que o objeto caiu. E por fim quando a aceleração volta a ser próximo 1g é exibido no monitor serial que o objeto está em repouso como pode ser visto no apêndice A.1.

3.5 Resultados

Como resultado temos o funcionamento dos 3 modos de operação, quando dispositivo está caindo, quando ele terminou de cair, e quando ele está em repouso como mostra a Figura 3.13.

Os testes de funcionamento do sistema realizados evidenciaram que o sensor MPU6050 é capaz de detectar quedas com precisão, desde que a altura de queda livre seja superior a 25cm. Para quedas abaixo desse limite, foi observado que o sensor não consegue detectá-las de maneira confiável. Essa informação é importante para o entendimento das limitações do sistema de detecção de quedas e pode auxiliar na definição de critérios de uso adequados.

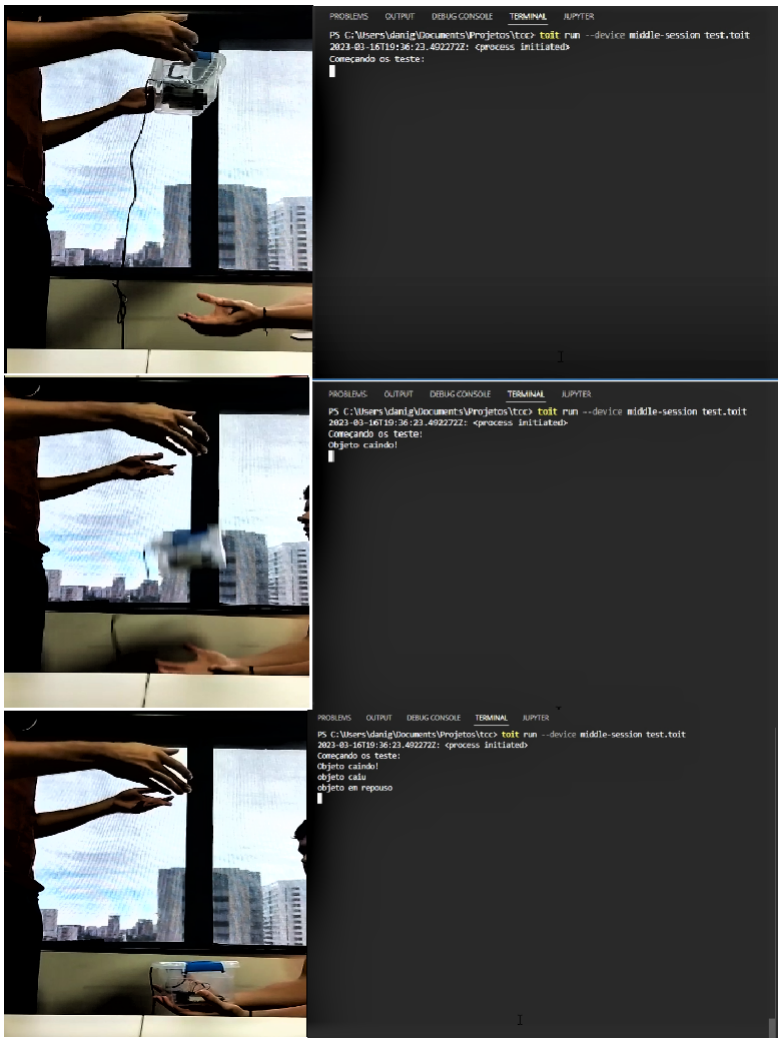

Figura 3.13: Teste do funcionamento do detector de quedas

Capítulo 4

Considerações Finais

4.1 Conclusão

Em suma, ao longo desse trabalho, foi possível observar que o sistema detector de quedas está longe de ser um sistema complexo de se implementar, mesmo se mostrando ser um sistema muito versátil e útil para produtos, que usa de um giroscópio para verificar se houve ou não o evento da queda. O fato à se observar porém, é a tecnologia utilizada para desenvolver o sistema cujo a aplicação é a detecção de quedas. Sendo desenvolvido com a Toit através de maquinas virtuais embarcadas no microcontrolador, traz ao dispositivo uma ampla gama de modificações e aperfeiçoamentos de seu *firmware* caso haja necessidade, trazendo mais *serviceabillity* ao produto, que é ter a capacidade de instalar, monitorar e configurar o produto.

Com os resultados anteriormente comentados do projeto teste, tem-se uma melhor ideia dos ganhos que o *Toit* pode fornecer à *stack* de desenvolvimento de *softwares* para sistemas embarcados, contudo para um projeto que se queira uma maior liberdade de controle dos parâmetros de funcionamento do sistema, a *Toit* não se demonstra ser uma boa escolha. Pois por usar uma linguagem que abstrai aspectos da linguagem c/c++, por exemplo a tipificação ou alocação de memória, tira o poder de gerenciamento desses recursos do programador. Além disso, até a conclusão desse trabalho a *Toit* só tem suporte para microcontroladores da família *ESP32* e mais especificamente somente os ESP32-D0WDQ6 e ESP32-D0WD-V*, tornando

muito restrito o uso dessa tecnologia.

De toda forma, para sistemas embarcados que exijam um controle rigoroso do ambiente em que planeja-se coloca-lo, é bastante conveniente o fato de se ter acesso ao monitor serial de forma remota e ter todos os dispositivos concentrados em um único ambiente de desenvolvimento. Como um bom exemplo dessa conveniência foi o projeto proposto, onde para se ter uma melhor análise de funcionamento do MPU6050, faz-se necessário que o sistema tivesse mais liberdade de movimentação, para que seja possível realizar a queda e aferir os valores limites de acontecimento do evento.

Então pode-se concluir que há espaço para *Toit* dentro do cenário de desenvolvimento de sistemas embarcados, como uma boa visão de onde o mercado de embarcados deve seguir evoluindo, trazendo ao programador uma maior versatilidade de poder desenvolver longe do sistema embarcado físico, permitindo uma maior dinâmica de desenvolvimento que não se limita a ter ou estar perto do *hardware* para poder desenvolver.

4.2 Dificuldades Encontradas

O *Toit* é um *framework* em ascensão no desenvolvimento de sistemas embarcados, no entanto, apesar de suas vantagens, há algumas dificuldades encontradas ao trabalhar com essa tecnologia.

Uma das principais dificuldades enfrentadas é a falta de documentação ampla e detalhada. Como o *Toit* ainda é uma tecnologia relativamente nova, muitas informações ainda não foram documentadas ou não estão disponíveis publicamente, o que pode dificultar a compreensão de certos aspectos do *framework*.

Outra dificuldade é a curva de aprendizado para se trabalhar com a linguagem de programação do *Toit*, que por mais que seja uma linguagem de alto nível, ainda continua sendo uma linguagem nova e diferente, onde os desenvolvedores precisam dedicar tempo para aprender a sintaxe e as particularidades antes de conseguirem usá-la efetivamente em projetos.

Além disso, o *Toit* ainda tem um número limitado de bibliotecas disponíveis. Isso pode ser uma dificuldade na hora de implementar certas funcionalidades ou recursos específicos em um projeto, exigindo que os desenvolvedores criem essas bibliotecas por conta própria.

Outra dificuldade é a falta de suporte da comunidade de desenvolvedores em relação a *Toit*. Como o *framework* ainda é relativamente novo, não há uma grande comunidade de desenvolvedores trabalhando com a tecnologia, o que pode dificultar a obtenção de suporte e resolução de problemas.

Apesar dessas dificuldades, o Toit ainda tem um grande potencial e é uma opção viável para o desenvolvimento de sistemas embarcados. Com o tempo, espera-se que essas dificuldades sejam minimizadas à medida que a tecnologia amadureça e mais desenvolvedores comecem a trabalhar com ela.

4.3 Trabalhos futuros

Como sugestão para futuros trabalhos, a verificação do comportamento da plataforma com um código maior, que chegue perto dos limites de memória programável da ESP32, para avaliar como se sairiam os recursos da plataforma. Outra sugestão também seria usar outras IDEs como exemplo, e até mesmo comparar com desenvolvimento de sistemas embarcados usando *Circuit python* e *IoT AppStore* que tem abordagem diferentes.

Referências

Arduino (2023). Arduino integrated development environment (ide) v1. https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics. Acesso em: 10/04/2023.

ASAIR (2018). Aht10 datasheet. https://server4.eca.ir/eshop/AHT10/Aosong_AHT10_en_draft_0c.pdf. Acesso em: 10/04/2023.

Cirani, S., Davoli, L., Ferrari, G., Léone, R., Medagliani, P., Picone, M., e Veltri, L. (2014). A scalable and self-configuring architecture for service discovery in the internet of things. *IEEE internet of things journal*, 1(5):508–521.

Davidlao2k (2022). Aht20. https://github.com/davidlao2k/aht20-driver. Acesso em: 10/04/2023.

Emyle, F.; Marcus, M. e Mateus, M. (2022). Perspectivas de mercado para sistemas embarcados e iot. https://www.virtus.ufcg.edu.br/perspectivas-de-mercado-para-sistemas-embarcados-e-iot/. Acesso em: 10/04/2023.

Erica, M. (2022). Ota update (over-the-air update). https://www.techtarget.com/searchmobilecomputing/definition/OTA-update-over-the-air-update. Acesso em: 10/04/2023.

Espressif (2016). Esp32 datasheet. https://img.filipeflop.com/files/download/Datasheet_ESP8266_esp32_en.pdf. Acesso em: 10/04/2023.

Espressif (2023). platformio/framework-espidf. https://registry.platformio.org/tools/platformio/framework-espidf. Acesso em: 10/04/2023.

Floitsch (2021). ssd1306. https://github.com/toitware/toit-ssd1306. Acesso em: 10/04/2023.

Florian, L. (2022). Toit will bring your iot projects up to speed. https://blog.codecentric.de/toit-will-bring-your-iot-projects-up-to-speed. Acesso em: 10/04/2023.

Guessi, M.; Yumi, E. O. F. e Maldonado, J. C. (2012). Architectural description of embedded systems: a systematic review. *Proceedings of the 3rd international ACM SIGSOFT symposium on Architecting Critical Systems*.

Heath, S. (2003). *Embedded Systems Design*. Newnes, Linacre House, Jordan Hill, Oxford OX2 8DP 200 Wheeler Road, Burlington MA 01803, 2^a edição.

InvenSense (2013). Mpu6050 datasheet. https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. Acesso em: 10/04/2023.

João, A. (1999). Princípio de funcionamento. http://www.eletrica.ufpr.br/edu/Sensores/1999/joao/funcionamento.htm. Acesso em: 10/04/2023.

Kaisti, M.; Rantala, V. e Mujunen, T. (2013). Agile methods for embedded systems development - a literature review and a mapping study. *J Embedded Systems*.

Kandasamy, K.; Srinivas, S. e Achuthan, K. (2020). Iot cyber risk: a holistic analysis of cyber risk assessment frameworks, risk vectors, and risk ranking process. *EURASIP J. on Info. Security*.

Kumar, S.; Tiwari, P. e Zymbler, M. (2019). Internet of things is a revolutionary approach for future technology enhancement: a review.

Limor, F. (2020). Aht10 arduino. https://adafruit.github.io/Adafruit_AHTX0/html/_adafruit___a_h_t_x0_8h.html. Acesso em: 10/04/2023.

Matheus, C. (2020). O que É um microcontrolador? https://edu.ieee.org/br-ufcgras/o-que-e-um-microcontrolador/. Acesso em: 10/04/2023.

Olikraus (2021). u8g2 arduino lib. https://github.com/olikraus/u8g2. Acesso em: 10/04/2023.

Otavio, S. (2019). Introdução a atualização ota (over-the-air). https://embarcados.com.br/atualizacao-ota-over-the-air-introducao/. Acesso em: 10/04/2023.

Phoenix1747 (2021). Mq135 gas sensor.
https://github.com/Phoenix1747/MQ135. Acesso em: 10/04/2023.

Rajmohan, T.; Nguyen, P. e Ferry, N. (2022). A decade of research on patterns and architectures for iot security. *Cybersecurity*.

SOLOMON (2009). Ssd1306 datasheet.
https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf. Acesso em: 10/04/2023.

Tim, F. (2023). What is firmware?
https://www.lifewire.com/what-is-firmware-2625881. Acesso em: 10/04/2023.

Toit (2021a). Device software high-level language.
https://toit.io/product/device-software. Acesso em: 10/04/2023.

Toit (2021b). Faq toit. https://toit.io/developers/faq. Acesso em: 10/04/2023.

Toit (2021c). Get started with toit. https://docs.toit.io/getstarted. Acesso em: 10/04/2023.

Toit (2021d). Jaguar: Live reloading for your esp32.
https://github.com/toitlang/jaguar. Acesso em: 10/04/2023.

Toit (2021e). Toit documentation. https://docs.toit.io/. Acesso em: 10/04/2023.

Toit (2021f). Toit platform. https://docs.toit.io/platform. Acesso em: 10/04/2023.

Toit (2022a). https://toitlang.org/. Acesso em: 10/04/2023.

Toit (2022b). https://docs.toit.io/platform/console. Acesso em: 10/04/2023.

Toit (2022c). Toit language basics. https://docs.toit.io/language. Acesso em: 10/04/2023.

Wikipedia (2020a). Active hard drive protection.
https://en.wikipedia.org/wiki/Active_hard-drive_protection. Acesso em: 10/04/2023.

Wikipedia (2020b). Firmware. https://pt.wikipedia.org/wiki/Firmware. Acesso em: 10/04/2023.

Wikipedia (2021). Microcontrolador.
`https://pt.wikipedia.org/wiki/Microcontrolador`. Acesso em: 10/04/2023.

Wikipedia (2023). Garbage collection.
`https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)`.
Acesso em: 10/04/2023.

Apêndice A

Apêndices

A.1 Código do projeto de detecção de quedas

```
import gpio
import i2c
import mpu6886
import fixed_point show FixedPoint
import math

flag := true
main:
  // I2C init
  bus := i2c.Bus
    --sda=gpio.Pin 21
    --scl=gpio.Pin 22

  // MPU6886 init.
  device := bus.device mpu6886.I2C_ADDRESS
  driver := mpu6886.Driver device
  Acc_past := 1.0
```

```
while true:
  // Read MPU6886.
  temp := driver.temperature
  acceleration := driver.acceleration
  gyro := driver.gyro
  // Print the read data.
  Acc :=( (math.pow acceleration.x 2) + (math.pow acceleration.y 2) + (

  if Acc < 0.2 and flag:
    print "Objeto caindo!"
    flag = false
  if Acc >1.1:
    print "objeto caiu"

  if Acc >=0.9 and Acc <= 1.1 and not flag:
    print "objeto em repouso"
    flag = true
  sleep --ms=50
```

A.2 Código para analise da qualidade do ar usando o MQ135

```
import gpio
import math
import gpio.adc show Adc
LED ::= 2
MQ135 ::= 34
```

```
retry_interval ::= 20
retries ::= 2
volt_resolution/float ::= 3.3
_ADC_Bit_Resolution ::= 12
A_CO ::= 605.18
B_CO ::= -3.937
A_Alcohol ::= 77.255
B_Alcohol ::= -3.18
A_CO2 ::= 110.47
B_CO2 ::= -2.862
A_Toluen ::= 44.947
B_Toluen ::= -3.445
A_NH4 ::= 102.2
B_NH4 ::= -2.473
A_Aceton ::= 34.668
B_Aceton ::= -3.369
_R0/float :=?
_sensor_volt/float := 0.0
_RL/float := 10.0
RatioMQ135CleanAir := 3.6


getVoltage MQ/Adc -> float:
  voltage/float :=?
  avg/float := 0.0
  adc/float := 0.0

  for i := 0; i < retries; i++:
    adc = MQ.get
```

```
    avg += adc
    sleep --ms=retry_interval


  voltage = ((avg/retries) * volt_resolution)/  ((math.pow 2 _ADC_Bit_Re
  _sensor_volt = voltage
  //MQ.close
  return voltage

calibrate ratioInCleanAir/float ->float:
  R0/float := 0.0
  RS_air/float := 0.0

  RS_air = volt_resolution*_RL
  RS_air = RS_air/_sensor_volt
  RS_air = RS_air-_RL

  if RS_air < 0.0:
    RS_air = 0.0

  R0 = RS_air/ratioInCleanAir

  if R0 <0.0:
    R0 = 0.0

  return R0

readSensor _A/float _B/float -> float:
  PPM/float :=?
```

```
  ratio/float := 0.0
  RS_Calc/float := 0.0



  RS_Calc = volt_resolution*_RL
  RS_Calc = RS_Calc/_sensor_volt
  RS_Calc = RS_Calc-_RL

  ratio = RS_Calc / _R0
  PPM = _A * (math.pow ratio _B)
  return PPM




main:
  //led := gpio.Pin LED --output
  //gas := Adc (gpio.Pin MQ135)
  MQ := Adc (gpio.Pin MQ135)
  calcR0/float := 0.0
  for i := 0; i < 10; i++:
    _sensor_volt = getVoltage MQ
    calcR0 += calibrate RatioMQ135CleanAir
    print "."

  _R0 = calcR0/10
  print "done"
```

```
while true:
  //print "blink"
  // led.set 1
  //print "Valor do gasssssss   : "
  _sensor_volt = getVoltage MQ // checar pra ver se essa fun   o    va
  CO :=  readSensor A_CO B_CO
  Alcohol :=  readSensor A_Alcohol B_Alcohol
  CO2 :=  readSensor A_CO2 B_CO2
  Toluen :=  readSensor A_Toluen B_Toluen
  NH4 :=  readSensor A_NH4 B_NH4
  Aceton :=  readSensor A_Aceton B_Aceton
  print "Acetona \t\t NH4 \t\t\t Toluen \t\t\t CO2 \t\t\t Alcohol \t\t
  //gas.close

  sleep --ms=500
  // led.set 0
  // sleep --ms=500
```