UNIVERSIDADE FEDERAL DE JUIZ DE FORA

INSTITUTO DE CIÊNCIAS EXATAS

CIÊNCIA DA COMPUTAÇÃO

Bryan Carolino Muniz Barbosa

Desenvolvimento de Bibliotecas em Python para Mini computadores de placa única com distribuições Linux OpenWrt

Juiz de Fora

2022

Bryan Carolino Muniz Barbosa

Desenvolvimento de Bibliotecas em Python para Mini computadores de placa única com distribuições Linux OpenWrt

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Profª. DSc. Bárbara de Melo Quintela

Juiz de Fora
2022

Bryan Carolino Muniz Barbosa

Desenvolvimento de Bibliotecas em Python para Mini computadores de placa única com distribuições Linux OpenWrt

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Aprovada em (dia) de (mês) de (ano)

BANCA EXAMINADORA

_____

Prof. DSc Bárbara de Melo Quintela - Orientadora
Universidade Federal de Juiz de Fora

_____

Prof. PhD Mário Ribeiro Dantas
Universidade Federal de Juiz de Fora

_____

Prof. DSc Luciano Jerez Chaves
Universidade Federal de Juiz de Fora

Bryan Carolino Muniz Barbosa
Universidade Federal de Juiz de Fora

Para H.K.,
minha estrela, meu perfeito
silêncio.

AGRADECIMENTOS

Ao Deus Único, que estendeu o céu, fundou a terra, e formou o espírito do homem dentro dele.

Ao meu pai, por me ensinar sobre eletricidade, mesmo com os choques.

À minha mãe, que nunca me deixou desistir, mesmo depois dos choques.

À minha noiva e melhor amiga, Heloísa Kohatsu, que exaustivamente leu e releu este trabalho atentamente ao meu lado e foi meu apoio e auxílio durante todo o processo.

À Tori Muniz, por todos os doces que me foram combustível (e novos quilos) durante as noites lendo *datasheets*.

À sempre acessível Professora Bárbara Quintela, que prontamente se dispôs a me orientar, pela educação e pela lição sobre me apaixonar por temas da pesquisa.

Ao Professor Steve Garan, pela oportunidade e por todo aprendizado no projeto.

À Alan Turing, o pai da Ciência da Computação.

RESUMO

Visando possibilitar o desenvolvimento de um dispositivo *Internet of Medical Things* (*IoMT*, Internet das Coisas Médicas, em Português) para cumprir a função de hipotálamo artificial, o presente trabalho foca no desenvolvimento e na documentação de bibliotecas Python para o mini computador Vocore 2, com vistas a utilizá-lo como central de processamento e controle de tal dispositivo. As bibliotecas desenvolvidas visam prestar funções como: controle das portas digitais para o envio de corrente elétrica (*OUTPUT*) e a detecção do recebimento de corrente elétrica (*INPUT*); comandar drivers controladores de motores de passo por meio das portas digitais; estabelecer comunicação com outros dispositivos por meio do protocolo I^2C (Inter-Integrated Circuit); mensurar e pré-processar sinais advindos de sensores em comunicação com o mini computador por meio do protocolo I^2C. Trabalhando em conjunto, as bibliotecas tornam possível o desenvolvimento de um ambiente para mensuração e processamento de sinais biológicos vindos de um organismo e o controle de atuadores mecânicos com o propósito de injetar automaticamente determinados hormônios selecionados previamente.

Palavras-chave: Internet das Coisas, Mini Computador, Python, Biblioteca, Biomedicina.

ABSTRACT

Aiming to develop an Internet of Medical Things (IoMT) device to fulfill the function of an artificial hypothalamus, the present work focuses on the development and documentation of Python libraries for the mini computer, Vocore 2, with the goal of using the device as a process and control center of said device. The libraries that were developed provide the following functions: controlling the digital doors that send electric current (OUTPUT) and detect its receival (INPUT); commanding the stepper motors' controlling drivers through the digital doors; establishing the communication with other devices via the I^2C protocol (Inter-Integrated Circuit); measuring and pre-processing the signals from communication sensors with SBC through I^2C. Working in consonance, the libraries make way for the development of an environment to measure and process biological signals that come from a given organism, and also to control mechanical actuators with the goal of automatically injecting previously selected hormones.

Keywords: Internet of Things, Single-Board Computer, Python, Library, Biomedicine.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CREA	Center for Research and Education on Aging
ECG	Eletrocardiograma
GPIO	General Purpose Input/Output
I2C	Inter-Integrated Circuit
IoMT	Internet of Medical Things
IoT	Internet of Things
LED	Light Emissor Diode
RBP	Raspberry Pi
SBC	Single-Board Computer

SUMÁRIO

1. INTRODUÇÃO

1.1. MOTIVAÇÃO

A presente pesquisa desenrola-se a partir de um projeto em processo de desenvolvimento coordenado pelo pesquisador Steve Garan no *Center for Research and Education on Aging* (*CREA*), da *University of California, Berkeley,* em colaboração com professores e alunos da Universidade Federal de Juiz de Fora. Este projeto visa estudar mais a fundo o envelhecimento a partir do monitoramento e controle de sinais biológicos e concentrações hormonais no corpo, com vistas a, em um futuro próximo, gerar soluções no retardamento dos efeitos do envelhecimento. Assim, a priori, serão feitos testes e monitoramentos biológicos em ratos para, a posteriori, migrar para humanos.

A fim de que tal monitoramento e controle ocorram, surgiu a necessidade do desenvolvimento de um dispositivo do tipo "*Internet of Medical Things"* (*IoMT*) sob medida que, com acurácia, pudesse medir sinais biológicos, processar algoritmos contendo modelos preditivos e para tomada de decisão, e controlar atuadores mecânicos em tarefas como a injeção de hormônios.

Para a criação de tal dispositivo, foi necessária a seleção de um computador que conseguisse atender requisitos quanto a tamanho, peso, consumo de energia e processamento e, por isso, o *Single-Board Computer* (*SBC*) Vocore 2 foi escolhido.

Contudo, o Vocore 2, devido a ser um dispositivo relativamente novo e pertencente a uma organização pouco conhecida, possui, ainda, comunidade incipiente, pouco material documentado e, também, poucas implementações de bibliotecas de controle, assim, faz-se necessária a criação de bibliotecas de controle e comunicação que sirvam de base para toda a implementação do dispositivo da pesquisa. A partir dessa necessidade de desenvolvimento, surge a necessidade do presente trabalho.

1.2. JUSTIFICATIVA

Um dos objetivos do projeto do CREA é desenvolver um dispositivo que expanda as possibilidades de compreensão da relação de sinais biológicos, níveis hormonais e qualidade/expectativa de vida no decorrer dos anos e de possíveis intervenções para o seu aperfeiçoamento. Tem-se a intenção de iniciar por testes em ratos para adquirir extensa compreensão acerca do ciclo circadiano, podendo depois haver o transporte desses

conhecimentos para humanos. Todavia, o conjunto de reações metabólicas circadianas ocorrem, por definição, no decorrer de cerca de um dia e, por isso, um estudo realmente acurado e abrangente acerca do tema demandaria controle e monitoramento constantes de níveis hormonais e sinais biológicos.

Pesquisas com ratos são comuns, contudo demanda-se a intervenção externa na vivência do animal para tarefas como a injeção de substâncias e, até mesmo a sedação do animal para exames como eletrocardiograma (ECG). Assim, os métodos atualmente utilizados, ou tornam-se limitados para não se tornarem invasivos ou tornam-se invasivos, gerando interferências no ciclo circadiano normal do animal (como em VERWEY et al., 2013). Por isso, estes métodos são limitados em ao menos um dos quatro pontos: (a) são invasivos; (b) exigem a intervenção frequente de um agente externo; (c) medem sinais biológicos somente em um determinado instante de tempo específico e (d) necessitam da retirada do animal do estado de vigília para aferições complexas como ECG (HO et al., 2012). Partindo disso, surge a necessidade do desenvolvimento de um dispositivo minimamente invasivo para monitoramento do ritmo circadiano do animal a todo instante.

Tal dispositivo, ainda não existente, se capaz de realizar a medição de sinais biológicos do animal de maneira a minimizar abordagens invasivas, processar modelos matemáticos para predição e auxílio na tomada de decisão, controlar micro atuadores mecânicos para injeção de hormônios, e trocar dados pela rede, mantendo-se dentro de restrições de tamanho e peso, poderia representar uma verdadeira revolução no estudo do envelhecimento. Para a criação de um dispositivo tal qual descrito, dentre as primeiras decisões tomadas, está a escolha da placa que atuará como central de processamento e controle. Tal escolha é vital para o cumprimento do objetivo principal do projeto, uma vez que é em tal placa onde todo o processamento estará centralizado.

No que diz respeito ao desenvolvimento de protótipos de produtos eletrônicos e computacionais, principalmente em ambiente acadêmico, duas organizações e seus dispositivos despontam em relação às outras: Arduino e *Raspberry*. As duas, estando no mercado já há mais de uma década, têm soluções bem consolidadas no que diz respeito a microcontroladores e *SBCs*. Contudo, seja por falta de poder de processamento satisfatório, seja por tamanho, peso ou consumo de energia que impossibilitariam o transporte do dispositivo completo por um rato sem a geração de estresse físico, nenhuma das placas das organizações citadas conseguiu atender a todas as restrições.

Por isso, reunindo o benefício de outras placas, a escolhida foi o *SBC* pouco conhecido Vocore 2, da empresa Vonger, por ter sido a única a atender todos os

pré-requisitos. A placa, apesar de não possuir uma grande comunidade de suporte, possui poder de processamento adequado, ocupa pouco espaço, é extremamente leve e consome pouca energia, reduzindo a necessidade de uma bateria volumosa e pesada para manter-se funcionando por várias horas. Entretanto, justamente por se tratar de um dispositivo relativamente novo e de comunidade de adeptos reduzida, encontrar implementações de códigos para controle do Vocore 2 e interfaces de comunicação com outros dispositivos tornou-se um desafio.

Apesar de uma ótima solução do ponto de vista físico, energético e de poder de computação, o Vocore 2 é carente em implementações para questões simples como o controle de pinos GPIO e comunicação com sensores, sendo que suas interfaces sequer possuem literatura ou documentação clara a respeito. A partir disso, neste trabalho tratamos do desenvolvimento e da caracterização de bibliotecas desenvolvidas para o Vocore 2 com o objetivo de tornar possível o controle e a comunicação com dispositivos externos com vistas ao desenvolvimento de um protótipo capaz de aferir sinais biológicos e injetar hormônios por meio de atuadores mecânicos em ratos.

As bibliotecas em desenvolvimento atuam em, basicamente, duas frentes: (a) controle de pinos digitais *Input*/*Output* (*I*/*O*) e (b) interface com dispositivos compatíveis com o protocolo I^2C. Quanto à primeira, por meio de pinos digitais *I*/*O* torna-se possível comandar e ativar componentes como motores de passo, que em nosso projeto receberam também sua respectiva biblioteca derivada da biblioteca de controle de pinos digitais.

Já acerca do protocolo I^2C (*Inter-Integrated Circuit*), trata-se de um protocolo de comunicação entre dispositivos eletrônicos criado pela empresa Philips nos anos 1980, que possui a função de, através de apenas dois fios, estabelecer comunicação bidirecional entre dois dispositivos. O protocolo possui um irmão/análogo de nome *SMBus*, que, na maioria dos casos, atua como um subconjunto do próprio protocolo I^2C (THE LINUX KERNEL, 2022). Os dois são compatíveis entre si, na maior parte das aplicações, para velocidades de até 100kHz. Em conjunto, os dois foram utilizados na criação de interfaces entre sensores (lado I^2C) e o *SBC* escolhido (lado *SMBus*). Devido ao fato de cada sensor possuir suas peculiaridades no estabelecimento de comunicação, foram desenvolvidas bibliotecas individuais para cada conexão e extração de dados de cada dispositivo. Assim, a partir dessas duas frentes, surgem todas as bibliotecas necessárias para a criação de um dispositivo *IoMT* controlado pelo *SBC* Vocore 2.

1.3. OBJETIVOS

1.3.1. Objetivo Geral

O objetivo principal do presente trabalho reside no desenvolvimento de bibliotecas com vistas a tornar possível a criação de um dispositivo *IoMT* minimamente invasivo, de baixo tamanho e peso e de pouco consumo energético, considerando o Vocore 2 como unidade centralizada de processamento de modelos matemáticos, aferição de sinais biológicos, controle de atuadores mecânicos e envio de dados pela rede.

1.3.2. Objetivos Específicos

Implementar as seguintes bibliotecas que fazem interface com outros dispositivos e controle de atuadores mecânicos em Python tornando o acesso às funções do dispositivo mais alto nível:

- **Vocore 2 GPIO** - para controlar os pinos *I/O*;
- **Vocore 2 Stepper Motor** - realizar interface para controle de *drivers* para motores de passo;
- **Vocore 2 MAX6675** - ler os sinais de termopares modelo MAX6675;
- **Vocore 2 ADS1115** - interface de comunicação com o conversor Analógico-Digital ADS1115 via protocolo I^2C;
- **Vocore 2 MPU-6050** - interface de comunicação com o acelerômetro e giroscópio MPU-6050 via protocolo I^2C;
- **Vocore 2 BH-1750** - interface de comunicação com o sensor de luminosidade BH-1750 via protocolo I^2C;
- **Vocore 2 TMP117** - interface de comunicação com o sensor de temperatura de alta acurácia TMP117 via protocolo I^2C.

Dessarte, uma pormenorização das ferramentas e procedimentos será analisada em seu viés histórico e conceitual na fundamentação teórica, descrita no Capítulo dois. Em seguida, no Capítulo três, abordaremos a metodologia utilizada no desenvolvimento de cada biblioteca para o Vocore 2. No Capítulo quatro, serão evidenciados e discutidos os resultados obtidos e, por fim, no Capítulo cinco constarão as considerações finais, com imagens e redirecionamentos para vídeos documentando tudo o que está descrito neste trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Joyia et al. (2017) e Al-Turjman et al. (2019) fizeram uma retrospectiva acerca dos progressos de *IoT* e, consequentemente, de *IoMT*, na ciência e de seus avanços e conquistas para a comunidade da saúde. A partir disso, é possível apurar que, embora tenham havido muitos estudos com variadas aplicações importantes para a área da saúde, ainda se trata de uma área que passou a ganhar tração recentemente (JOYIA et al., 2017; AL-TURJMAN et al., 2019).

De acordo com Joyia e colaboradores (2017), ainda há desafios para a área da *IoMT* destravar, a saber: gerenciamento de diversidade de dispositivos; escala, volume de dados e performance; flexibilidade e evolução de aplicações; necessidade de expertise médica; capacidade de CPU; disponibilidade de recursos; relação de modelagem entre medidas adquiridas e doenças; implementação de software de esquemas médicos analíticos; previsibilidade do sistema, entre outros.

Além disso, em busca de artigos que pudessem auxiliar no desenvolvimento do dispositivo em questão, a maior parte dos estudos acadêmicos parece estar centrada na função de segurança do dispositivo, como é trazido por Koutras et al. (2020), em sua pesquisa sobre segurança em *IoMT*. Outros pesquisadores também debruçam-se nas diversas formas de criar um algoritmo seguro para aplicação, como via *blockchain* (GARG et al., 2020) ou *machine learning* (HAMEED et al., 2021).

Portanto, mostra-se necessário e relevante o desenvolvimento do presente trabalho, a fim de alcançar a meta descrita, e que será dissecada melhor adiante, bem como para que outros pesquisadores também tenham acesso a novas descobertas no campo da *IoMT*.

O dispositivo *IoMT* em desenvolvimento utiliza um mini computador de placa única (*Single-Board Computer* - *SBC*, em inglês). De acordo com Isikdag (2015), o termo *SBC* é usado para definir computadores que consistem em uma única placa de circuito contendo a memória e o processador. A maioria dos *SBC*s tem interfaces *I/O* (*Input/Output*), em que diferentes tipos de sensores e atuadores podem ser conectados. Esses computadores geralmente não têm *slots* para expansão como os computadores normais, e utilizam processadores de baixo custo. O tamanho desses dispositivos varia de uma caixa de fósforos a uma carta de baralho (ISIKDAG, 2015).

O dispositivo para o qual as bibliotecas foram desenvolvidas trata-se do Vocore 2, um *SBC* de hardware aberto. Todavia, o exemplo mais conhecido de dispositivos com interfaces *I/O* programáveis e *SBCs* são os das organizações Arduino e *Raspberry*. Suas placas possuem

extensa comunidade de adeptos e são comuns de serem utilizadas em projetos acadêmicos. A seguir serão descritos os microcontroladores e *SBC*s mais conhecidos.

2.1. CONCEITOS

2.1.1. Sistema Operacional

Acerca do Sistema Operacional do dispositivo, trata-se de uma distribuição Linux, que é um kernel de sistema operacional do tipo Unix, livre e de código aberto, monolítico, modular e multitarefa (MAUERER, 2008). A distribuição usada é a OpenWrt, que consiste em um Sistema Operacional Embarcado baseado em Linux projetado especialmente para roteadores, mas com enorme potencial para aplicações IoT (JIN, 2012).

2.1.2. Linguagem Python

A linguagem de programação escolhida para o desenvolvimento de tais bibliotecas foi Python, uma linguagem interpretada, que, como prevê Rossum (1998), vem incrementando sua performance a cada atualização. A escolha de Python se deu tanto devido a onda crescente da utilização da linguagem em sistemas embarcados e em aplicações gerais em *IoT*, como também pela sua estrutura pouco verborrágica e de curva de aprendizado curta, possibilitando a integração com facilidade entre pesquisadores de campos além da Computação e da Engenharia Eletrônica, e da área da saúde e das Ciências Biológicas.

Essa linguagem é caracterizada por seu alto nível de interpretação, com uma semântica dinâmica de digitação e encadernação. De acordo com Rossum (1998), é de 5 a 10 vezes mais fácil programar em Python do que em C/C++, e de 3 a 5 que em Java. Ademais, é extremamente incorporável a outras aplicações, suportando inclusive programação orientada a objetos, bem como o seu uso em todo tipo de plataformas. Por conta disso, traduzir Python para outras linguagens gera resultados com muita facilidade e rapidez (ROSSUM, 1998).

2.1.3 Biblioteca

De acordo com o dicionário Oxford University Press (2019), uma biblioteca consiste em uma coleção de programas e pacotes acessíveis para uso comum dentro de determinado ambiente, sem que os itens individuais necessariamente estejam relacionados. É frequente

encontrar em uma biblioteca compiladores, programas utilitários, pacotes de operações matemáticas, entre outros (OXFORD University Press, 2019). O seu uso está muito relacionado com a praticidade e flexibilidade de poder instalar uma mesma biblioteca em algoritmos diferentes, diminuindo pela metade o tempo e o esforço de fazer o mesmo trabalho implementando tais programas e pacotes várias vezes.

2.1.4. Portas GPIO

Portas, ou pinos, GPIO tipicamente referem-se a pinos de sinal digital em um placa embarcada em que o comportamento, incluindo o modo do pino (entrada ou saída) e o nível lógico do sinal (alto ou baixo), pode ser programado pelo usuário em tempo de execução (ORACLE, 2014).

Ao configurar-se os pinos como saída (*OUTPUT*), é possível enviar sinal digital de nível baixo (0-1 Volts) ou alto (2-5 Volts, de acordo com o dispositivo), sendo esses sinais normalmente representados por 0 e 1, respectivamente. Esse sinal pode ser utilizado para tarefas simples como fornecer energia a um *LED* (*Light Emissor Diode* - Diodo Emissor de Luz, em português), até aplicações mais complexas, como realizar o controle de um motor de passo através de uma alternância definida entre estado lógico alto e baixo, assim como é feito neste trabalho. Da mesma maneira, da configuração de um pino como entrada (*INPUT*), tem-se a possibilidade de detectar o recebimento de sinal digital, podendo, por conseguinte, utilizar componentes como chaves interruptoras (THE LINUX KERNEL ARCHIVES, 2022).

2.1.5. Protocolo I^2C

Inter-Integrated Circuit (I^2C ou *IIC* - Inter-Circuito Integrado, em português) é um barramento serial de comunicação entre um ou mais dispositivos *masters* (em português, mestres) e um ou mais dispositivos *slaves* (em português, escravos), criado pela *NXP Semiconductors*, originalmente, uma divisão de semicondutores da empresa Philips, com o objetivo de conectar periféricos a sistemas embarcados (HILL, 2021).

De acordo com TronicsBench (2022), o conceito de *masters* e *slaves* é dado da seguinte maneira: Um dispositivo *master* é o dispositivo encarregado do barramento no presente momento. Este dispositivo controla o *clock* e gera sinais de *Start* e *Stop*. Já *slaves* apenas recebem os comandos e agem no controle e nos comandos do envio de dados (TRONICSBENCH, 2022).

Utilizando somente duas conexões, assim como mostrado na Figura 7 abaixo, é possível conectar um ou mais dispositivos *masters* a até 128 dispositivos *slaves*. Cada chip que opera como *slave* tem, por padrão, um conjunto de endereços específicos que o farão responder às chamadas de um dos *masters*.

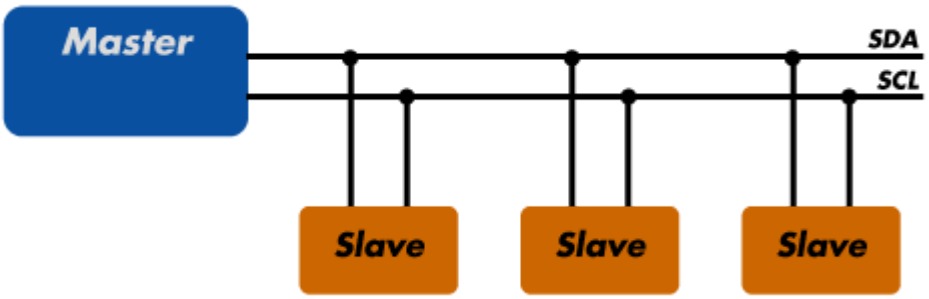

Figura 6: Representação simples de um barramento I^2C.

No protocolo I^2C as velocidades de operação podem ser: 100kHz, 400kHz e 3,4MHz. Nem todos os os chips suportam todas as velocidades; no entanto, segundo TronicsBench (2022), 100kHz é a velocidade mais comumente suportada.

2.1.5.1. *SMBus*

Derivado do protocolo I^2C, o *SMBus* é um protocolo que se mantém compatível com o seu precursor para velocidades de até 100kHz. Seu uso neste trabalho se deu devido a sua implementação em software preparada e de funcionamento análogo ao funcionamento de outros *SBCs* mais comumente adotados. Desse modo, dispositivos, tais quais os sensores utilizados, que possuem de fábrica compatibilidade I^2C, puderam ser conectados ao Vocore 2 utilizando-se de uma implementação em *software* com comandos *SMBus* compatíveis com I^2C (THE LINUX KERNEL, 2022).

2.1.6. Motor de Passo

A Advanced Micro Systems (AMS) define os motores de passo como motores que, ao contrário dos comumente utilizados motores de corrente contínua, funcionam por meio alternância controlada eletronicamente de campos magnéticos que leva a movimentação controlada e direcionada de um rotor interno (AMS).

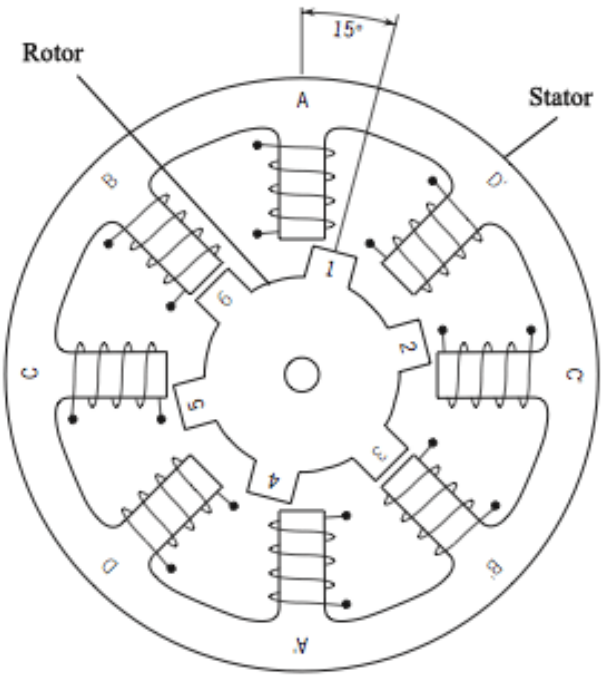

Figura 7: Esquema interno de um motor de passo

Por meio da alternância das fases das bobinas, como ilustrado na Figura 8, os “dentes” do rotor são atraídos para um eletroimã vizinho ao que estão presentemente estacionados. Tal atração gera movimento de rotação no rotor, que é chamado de passo (ORIENTAL MOTOR, 2017).

Motores de passo têm, dentre as suas especificações técnicas, a quantidade de passos necessários para que o rotor execute uma revolução, chamada de passos por revolução. Geralmente, com vistas a manter a fluidez e a acurácia na movimentação e reduzir os riscos de queima de portas ou do circuito de onde os comandos estão sendo enviados por inteiro, utiliza-se um circuito controlador de motores de passo chamado de *driver*. Por meio de *drivers*, além de ser possível ter mais fluidez e acurácia no controle do motor, é possível utilizar motores que demandam tensões e correntes por fase maiores do que um pino GPIO normalmente pode fornecer.

Os *drivers* utilizados neste trabalho operam em *Legacy Mode*, isto é, o modo mais simples de operação do *driver*, que consiste no controle da direção e dos passos por meio de alterações no estado lógico do sinal digital enviado. Neste trabalho, não foram utilizados outros modos de operação complexos disponíveis, como a gravação permanente da implementação na memória *PROM* (Programmable Read-Only Memory - Memória Programável de Somente Leitura, português) do dispositivo ou modo dinâmico, que consiste no envio de comandos em tempo de execução via protocolo *UART* (*Universal Asynchronous*

Receiver-Transmitter - Transmissor/Receptor Universal Síncrono e Assíncrono, em português) (FABLAB KANNAI, 2022).

2.1.7. Acelerômetro

Um acelerômetro é um dispositivo que mede a vibração ou aceleração do movimento de uma estrutura. A força causada pela vibração ou variação no movimento (aceleração) leva uma massa a pressionar um material piezoelétrico, produzindo, assim, variação na tensão elétrica proporcional à força exercida. Tendo em vista que esta variação é proporcional à força, considerando que esta é um produto entre massa e aceleração e, semelhantemente, que a massa é constante, temos que a variação na tensão elétrica será proporcional à aceleração (OMEGA, 2022).

2.1.8. Giroscópio

De acordo com Passaro e colaboradores (2017), os giroscópios são dispositivos montados em uma estrutura capaz de detectar rotação. Da mesma maneira que os acelerômetros, giroscópios, como o utilizado neste trabalho, dependem de um material piezoelétrico, e por meio dele conseguem mensurar a velocidade angular (EPSON, 2022).

2.1.9. Termopar

Um termopar é um dispositivo que funciona a partir do chamado “Efeito Seebeck”, de acordo com Scervini (2009). Tal fenômeno se dá pela seguinte conceituação: dados dois pedaços de metal unidos em uma de suas pontas, ao se submeter a ponta unida a uma temperatura diferente da temperatura das pontas não unidas, verificar-se-á uma diferença de potencial (tensão elétrica) entre as pontas não unidas que pode ser transliterada matematicamente, tendo em vista as propriedades inerentes de cada material, em um valor específico de temperatura. Por consequência, o efeito ocorre para qualquer par de metais; no entanto, é comum que se utilizem pares de metais específicos que já possuem um valor de diferença de potencial esperada tabelado de acordo com a progressão da temperatura ao qual o dispositivo foi submetido (SCERVINI, 2009).

Os pares de metais/ligas metálicas comumente utilizados em termopares são vários, todavia o tipo escolhido para compor o termopar utilizado é o K, um conjunto de cromel e

alumel, operado continuamente em faixas de temperatura de 0 a 1100ºC (TMS EUROPE, 2015), ligado a um módulo MAX6675, que trabalha na conversão de sinais analógicos advindos do termopar em sinais digitais. A resolução do módulo é de 0,25ºC e a sua acurácia reside em ±5ºC, o que o levou, posteriormente, a ser retirado da pesquisa, sendo substituído pelo sensor TMP117 (Seção 2.2.7).

2.2. DISPOSITIVOS

2.2.1. Arduino

A organização Arduino (Figura 2) traz placas de microcontroladores e ecossistemas de programação que têm como objetivo eliminar ou diminuir a dificuldade e o tempo desnecessário muitas vezes gasto com o desenvolvimento de circuitos eletrônicos para o pleno funcionamento do microcontrolador e, também, com o desenvolvimento de código em baixo nível. Para problemas como esses, as placas Arduino tornam-se uma solução elegante, pois, no que diz respeito à eletrônica, são, praticamente, dispositivos "*plug-and-play*", isto é, prontos por meio de uma configuração rápida, posto que são projetados para que o usuário rapidamente acople dispositivos e componentes eletrônicos (KUSHNER, 2011).

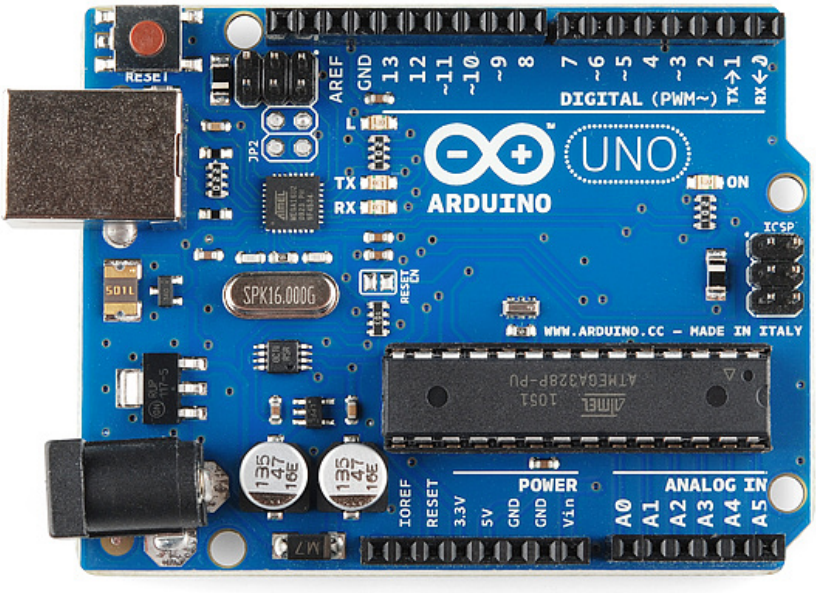

Figura 1: Arduino Uno, uma das placas mais populares.

Já no que diz respeito à programação, Souza e colaboradores (2011) descrevem que as placas oferecem ambiente de programação e conjunto de comandos próprios baseados em C++. Com comandos básicos prontos e interfaces de controle de dispositivos de terceiros

implementados por uma grande comunidade de código aberto adepta das placas, a programação em placas Arduino é extremamente facilitada e de curva de aprendizado reduzida (SOUZA et al., 2011). Contudo, tais dispositivos têm em si a limitação do poder de processamento. Apesar de toda a facilidade na utilização, por se tratarem de microcontroladores e não de minicomputadores, placas Arduino tendem a ser limitadas quanto ao desempenho.

No período das decisões iniciais do projeto, a placa Arduino de tamanho compatível e maior poder processamento, o Arduino Nano 33 BLE, possuía um chip nRF52840 32-bit ARM Cortex-M4 com clock de 64 Mhz (ARDUINO DOCS, 2022), o que, no futuro, poderia vir a ser um problema para um dispositivo que, em funcionamento, estaria lidando constantemente e em paralelo com o monitoramento e armazenamento de sinais biológicos, controle acurado de atuadores através de *encoders*, processamento de modelos matemáticos relativamente complexos e conexão frequente com a rede para envio de dados.

2.2.2. Raspberry Pi

Há, também, a organização *Raspberry*, responsável pela linha de *SBCs* e microcontroladores *Raspberry Pi* (*RBP*). Os *RBP* seguem o mesmo princípio dos microcontroladores Arduino do ponto de vista eletrônico, apresentando placas com uma estrutura pronta para realizar a conexão com outros dispositivos e componentes eletrônicos e dar início ao desenvolvimento. Somado a isso, com respeito especificamente às placas *SBCs*, os *RBP* contam com um poder de processamento consideravelmente grande, bem como um ambiente de programação com mais de uma possibilidade no caminho de desenvolvimento, por contar tanto com Python (através de MicroPython) quanto com C/C++ como linguagens suportadas por padrão (RASPBERRY PI, 2022).

A organização *Raspberry* conta com uma grande comunidade, tornando fácil a resolução de problemas durante o desenvolvimento nas placas. Porém, o tamanho e o peso das placas não se saiu bem na comparação com a placa escolhida (Vocore 2) durante a escolha do equipamento. O modelo *Raspberry Pi Pico*, que possui o microcontrolador RP2040, conta com as dimensões 51.3mm x 21.0mm x 3.9mm (Comprimento x Largura x Altura) e 3g, perdendo para o Vocore 2, que possui 25.6mm x 25.6mm x 3.0mm e 3g (RASPBERRY PI LTDA, 2022).

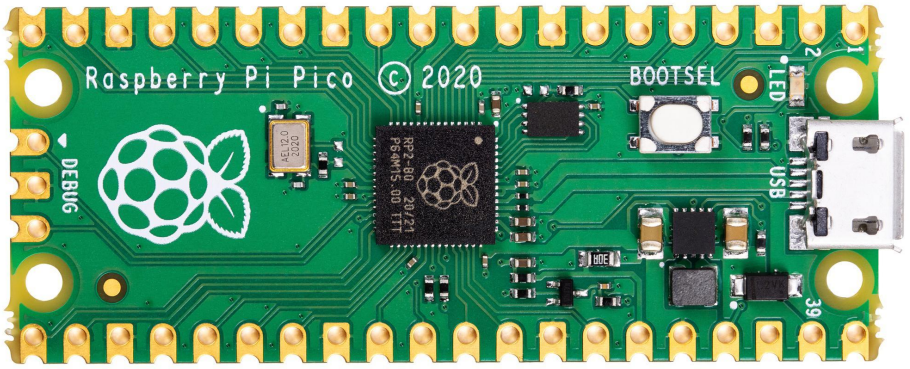

Figura 2: *Raspberry Pi Pico*

Além disso, de acordo com Thothadri (2021), a diferença no poder de processamento também deixa a escolha clara: o *Raspberry Pi Pico* possui um processador Dual ARM Cortex-M0+ com clock de 133MHz, e o Vocore 2, um Mediatek MT7628 MIPS 24K com clock de 580 MHz, pertencente a uma família de processadores focada e otimizada para dispositivos que trabalham em rede. Elevando a disputa, o *Raspberry Pi Zero 2 W* é um *SBC* que possui processador Cortex-A53 quad-core com arquitetura ARM e 1GHz de clock; entretanto, nem o tamanho nem o peso se adequam aos parâmetros com 65.0mm x 30.0mm x 5.3mm (Comprimento x Largura x Altura) e 10g (GAMESS, HERNANDEZ, 2022).

2.2.3. Outras Placas

Vale mencionar ainda as placas *BeagleBoard* e os microcontroladores *ESP32*, tendo o primeiro grupo não sido utilizado por limitações quanto ao tamanho maior do que o ideal e o segundo devido ao poder de processamento limitado do processador de 240 MHz (COLEY, 2014; EXPLORE EMBEDDED, 2017).

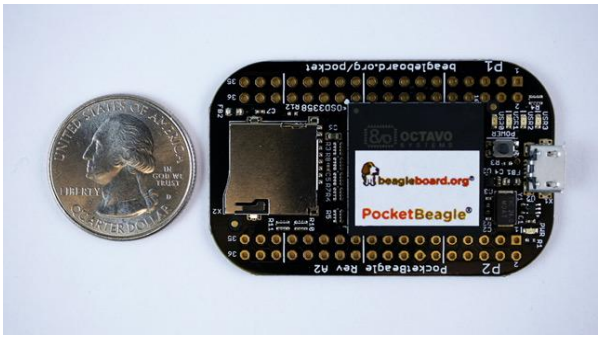

Figura 3: *PocketBeagle*, a menor placa *BeagleBoard*, com 56mm x 35mm x 5mm (Comprimento x Largura x Altura)

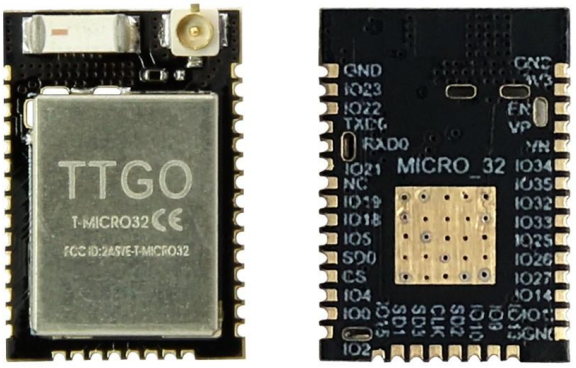

Figura 4: TTGO Micro-32 V2.0, o menor módulo encontrado portador do microcontrolador ESP32, com 13mm x 18,9mm x 5mm (Comprimento x Largura x Altura)

2.2.4. VOCORE 2

O Vocore 2 é um *SBC* que fornece interessantes possibilidades para desenvolvimento de dispositivos Internet das Coisas (em inglês, *Internet of Things/IoT*) tendo em vista a suas especificações técnicas, sendo algumas delas: (a) poder de processamento satisfatório devido a um processador de 580 MHz; (b) tamanho e peso diminutos (25.6mm x 25.6mm x 3.0mm/3g); (c) chip e antena para conexão wireless 802.11n embutidos na placa, que possibilitam nativamente tanto conectar o dispositivo a rede quanto utilizá-lo como um pequeno roteador, além da possibilidade de integração e criação de interfaces como *Bluetooth* e outras conexões sem fio; (d) interface e portas GPIO para conexão com outros dispositivos; (e) interface I^2C; (f) baixo consumo de energia em operação (230 mA); (g) sistema operacional OpenWrt de fábrica, uma distribuição Linux direcionada para aplicações de rede e *IoT* (VOCORE STUDIO, 2022).

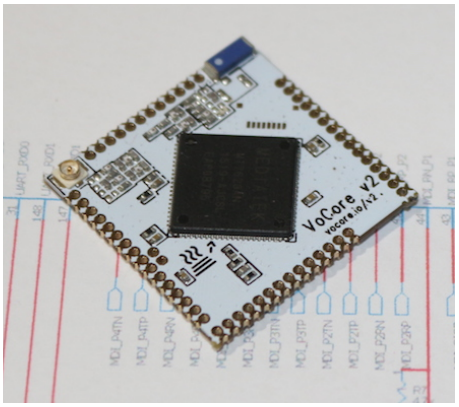

Figura 5: Vocore 2 (VOCORE STUDIO, 2022)

O Vocore 2 foi selecionado porque, apesar de seu tamanho diminuto e seu peso quase desprezível (ver Figura 6), conta uma quantidade considerável de poder de processamento (com uma CPU MT7628 de 580 Mhz e uma Memória Ram 128 MB), bem como interfaces de controle *I/O* e de comunicação como WIFI, USB, UART, *I2C*, SPI e um armazenamento de 16MB com possibilidade de expansão para 2TB por meio de um cartão SD (VOCORE STUDIO, 2022).

2.2.5. Acelerômetro/Giroscópio MPU-6050

O chip MPU-6050 possui embutidos em sua estrutura um acelerômetro de 3 eixos (x, y, z), um giroscópio de 3 eixos (x, y, z) e sensor de temperatura (não utilizado neste trabalho). As faixas de precisão do acelerômetro e do giroscópio do chip podem ser programadas pelo usuário em ±2, 4, 8 e 16g e ±250, 500, 1000, 2000°/s, respectivamente. Sua interface de comunicação se dá por meio do barramento I^2C (INVENSENSE, 2013).

2.2.6. Sensor de luminosidade BH-1750

O sensor gera medições de luminosidade em uma faixa de 1 a 65535 lux (lúmen/metro quadrado) que são traduzidos em sinal digital de 16 bits. Sua interface de comunicação se dá por meio do barramento I^2C (ROHM, 2010). A medição da luminosidade torna-se importante para o acompanhamento da influência da luz do ambiente nos níveis hormonais no decorrer do ciclo circadiano e, dentre estes, principalmente a melatonina, hormônio fortemente associado com o ciclo de sono.

2.2.7. Sensor de temperatura TMP117

O chip TMP117 possui um sensor de alta acurácia capaz de medir dentro da faixa de -20ºC a 50ºC com acurácia de ±0,1ºC. O alto nível de acurácia das medições do dispositivo fazem do seu *hardware* a cumprir os requisitos de medições com rastreabilidade *NIST*, isto é, as medidas feitas pelo dispositivo podem ser metrologicamente rastreadas de acordo com a compreensão prática da definição da unidade de medida (temperatura) para o *NIST* (*National Institute for Standards and Technology* - Instituto Nacional de Padrões e Tecnologia, em português). Além disso, o dispositivo também é capaz de receber programação interna para

emissão de alertas e sua interface de comunicação se dá por meio do barramento I^2C (TEXAS INSTRUMENTS, 2021).

2.1.8. Conversor analógico-digital ADS1115

O ADS1115 é um dispositivo de quatro portas capaz de receber sinais analógicos (sinais de tensão elétrica variada dentro de um certo intervalo), e, diante de precisão limitada, de traduzi-los em valores equivalentes dentro de outro intervalo injetivo. A amplitude desse intervalo se dá em função da precisão da porta, dada em bits. No caso do ADS1115, cada uma das quatro portas possui amplitude de 16 bits, o que possibilita aos valores recebidos serem traduzidos em qualquer inteiro no intervalo fechado [0, 65535]. Sua interface de comunicação se dá por meio do barramento I^2C (TEXAS INSTRUMENTS, 2009).

2.1.9. Motor de passo NEMA 17 VZS1734-028-0404

O motor de passo utilizado para realização dos testes é um motor bipolar, isto é, dois grupos de bobina (A e B) com as seguintes especificações: *holding torque* de 2,8 Kgf.cm; *detent torque* de 120 gf.cm; inércia do rotor de 34 gf.cm²; corrente por fase de 0,4A; tensão de fase de 12V; resistência de 30Ohm ±10% por fase; indutância de 35mH ±20% por fase; 4 fios; conexão do tipo bipolar; ângulo de passo de 1,8° ±5%; dimensões de 42,3 x 42,3x 33,5 mm; e peso de 0,25Kg.

3 METODOLOGIA

Para uso do Vocore 2 como unidade centralizada de controle de todas as funções do projeto, tornou-se necessário o desenvolvimento de bibliotecas para controlar as funções do dispositivo. As bibliotecas desenvolvidas foram: Vocore 2 GPIO; Vocore 2 Stepper Motor; Vocore 2 MAX6675; Vocore 2 ADS1115; Vocore 2 MPU6050; Vocore 2 BH-1750 e Vocore TMP117, cuja implementação será descrita nas próximas subseções.

3.1. VOCORE 2 GPIO

A biblioteca "Vocore 2 GPIO[1]" foi desenvolvida com o objetivo de tornar fácil o controle de portas GPIO no Vocore 2. Através dela, é possível setar o estado das portas como *INPUT* (entrada de dados/corrente elétrica) ou *OUTPUT* (saída de dados/corrente elétrica), e ler (em caso de *INPUT*) o estado da porta (0, para nível lógico baixo sendo recebido, ou 1, para nível lógico alto sendo recebido) e escrever (em caso de *OUTPUT*) sinal na porta (0, para nível alto baixo sendo enviado, ou 1, para nível lógico alto sendo enviado).

Visando trazer uma compreensão rápida da biblioteca, buscou-se conceber a biblioteca com comandos parecidos ao máximo com os utilizados no *framework* Arduino.

Quanto ao gerenciamento de estados das portas e sinais enviados e recebidos, foi seguido o princípio "*Everything is a file*" dos sistemas Unix, em que alterar o estado de uma porta ou alterar o nível lógico dessa porta ocorre pela simples subscrição de um arquivo.

Assim, de maneira análoga ao *framework* Arduino, a configuração de um pino ocorre por meio do processo: 1. setagem do modo do pino (*INPUT* ou *OUTPUT*); e 2. leitura ou escrita de sinal digital no pino (alto ou baixo). Na biblioteca desenvolvida, o uso se dá de maneira semelhante, com adição de um comando para a criação de um arquivo correspondente a porta, que foi omitido dentro da função de setagem do modo da porta.

3.2. VOCORE 2 STEPPER MOTOR

A biblioteca "Vocore 2 Stepper Motor[2]" foi desenvolvida com o objetivo de controlar motores de passo através de *drivers* em *Legacy Mode* (ver Seção 2.10). Esse controle acontece pela alteração no estado lógico das portas GPIO com uso da biblioteca "Vocore 2 GPIO".

---

[1] Disponível em <https://github.com/BryanCMBarbosa/Vocore2-GPIO> (BARBOSA, 2022)
[2] Disponível em <https://github.com/BryanCMBarbosa/Vocore2-Stepper-Motor> (BARBOSA, 2022)

Em uma visão geral, um controle básico efetuado por um *driver* em *Legacy Mode* ocorre por meio de setar a direção da rotação, através do estado lógico do pino *DIR* e do envio de pulsos em estado lógico alto ao pino *STEP*, em que cada pulso correspondente a um comando para um passo no rotor. Dessarte, a quantidade de rotações por minuto do motor pode ser regulada meramente pela alteração no tempo entre os pulsos enviados. Há a opção, ainda, de ativar o modo *microstepping*, no qual, em vez da relação de 1:1 entre pulsos e passos, teremos uma relação de $1:2^n$, em que n, nos *drivers* testados, varia no intervalo inteiro [2, 8], possibilitando o movimento do motor em passos ainda menores, aumentando o número de passos por rotação.

3.3. VOCORE 2 ADS1115

A biblioteca "Vocore 2 ADS1115[3]" supre a falta de entradas analógicas no Vocore 2. A partir do protocolo de comunicação I^2C, os dados de quatro entradas analógicas presentes no conversor analógico-digital são enviados ao *SBC* com uma amplitude de 16 bits.

O funcionamento se dá por meio de: 1. escrita no registrador de configuração de uma tupla contendo endereço das portas das quais será calculada a diferença de potencial; 2. espera de 500 milissegundos para aferição do valor; e 3. leitura do valor aferido.

3.4. VOCORE 2 MPU6050

O sensor MPU-6050 tem em si as funções de acelerômetro e giroscópio, permitindo que o chip seja capaz de mensurar aceleração e inclinação em três dimensões, totalizando a geração concomitante de 6 eixos de dados acerca do ambiente no qual o dispositivo está imerso. A biblioteca atua com objetivo de tornar possível a conexão e a leitura de dados do sensor.

A implementação da biblioteca[4] se dá por: 1. setagem dos registradores de configuração do dispositivo de acordo com o desejado, segundo a especificação "*Register Map and Descriptions*", documento fornecido pelo fabricante do dispositivo (INVENSENSE, 2013); e 2. a partir disso, o MPU-6050 opera em modo contínuo na leitura de valores de aceleração e de velocidade angular relativos ao próprio chip no ambiente e seu posicionamento, bastando ler o par de registradores correspondentes ao eixo de valor requerido (Exemplo: Eixo X-Acelerômetro, Eixo Z-Giroscópio, etc.).

---

[3] Disponível em <https://github.com/BryanCMBarbosa/Vocore2-ADS1115> (BARBOSA, 2022)
[4] Disponível em <https://github.com/BryanCMBarbosa/Vocore2-MPU6050> (BARBOSA, 2022)

3.5. VOCORE 2 BH-1750

O BH-1750 é um sensor de luminosidade. O sensor consegue detectar diferentes níveis luminosidade retornando-os na unidade de medida lux (lúmen por metro quadrado). Assim como as outras bibliotecas, nesta[5] também atua na conexão e na aquisição de leituras do sensor por parte do Vocore 2.

O sensor possui funcionamento simples: a partir da sua ligação ao fornecimento de energia e da conexão correta com os pinos do barramento I^2C, já é possível enviar requisições de leitura, bastando apenas ler o registrador correspondente ao modo (Exemplo: Modo de leitura contínua, alta resolução 0.5lx: registrador 0x11.).

3.6. VOCORE 2 TMP117

O desenvolvimento de uma biblioteca[6] para o sensor de temperatura TMP117 surgiu a partir da constatação da falta de acurácia do termopar MAX6675. O termopar possui acurácia de ±5ºC, o que não é ideal para a mensuração de temperatura corporal. Por conta disso, o sensor TMP117, com acurácia de ±0,1ºC e comunicação via I^2C, foi adicionado ao projeto para a substituição.

A leitura dos valores da temperatura do TMP117 ocorre de maneira simples, tal qual a do sensor BH-1750, no qual basta escrever uma requisição com o valor 0x0220 no registrador 0x00, aguardar 500 milissegundos e ler a resposta do registrador 0x00.

4 RESULTADOS

Nesta seção serão abordados os resultados obtidos a partir da implementação das bibliotecas em Python para o Vocore 2.

4.1. VALIDAÇÃO DA BIBLIOTECA VOCORE 2 GPIO

A testagem[7] da biblioteca Vocore 2 GPIO ocorreu por meio da implementação de um código simples de detecção de sinal lógico alto por meio do pressionamento de um resistor *pull-down*, como mostrado na Figura 8 a seguir:

---

[5] Disponível em <https://github.com/BryanCMBarbosa/Vocore2-BH1750> (BARBOSA, 2022)
[6] Disponível em <https://github.com/BryanCMBarbosa/Vocore2-TMP117> (BARBOSA, 2022)
[7] Disponível em <https://youtu.be/nNGnrGSDweU> (BARBOSA, 2022)

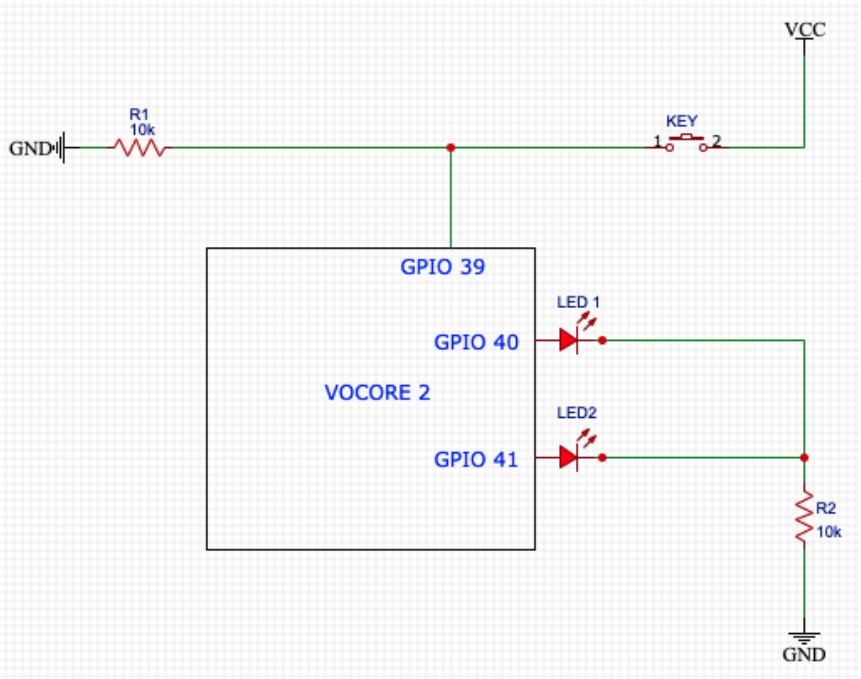

Figura 8: Circuito de teste da biblioteca VOCORE 2 GPIO

Na Figura 8, o resistor *pull-down* trabalha como um *switch* que, em estado normal (sem pressionamento), mantém interrompida a passagem de corrente elétrica até a porta GPIO 39 do Vocore 2. O pressionamento do resistor permite a passagem de corrente elétrica para que esta corrente seja detectada pela porta 39, que está configurada como *INPUT*. A detecção da passagem de sinal elétrico faz com que o código execute ação de alternar entre os dois *LED*s ligados às portas 40 e 41, configurados como *OUTPUT*.

O código de teste foi implementado da seguinte maneira:

```
1.  from vocoreGPIO import *
2.
3.  pinMode(39, "INPUT")
4.  pinMode(40, "OUTPUT")
5.  pinMode(41, "OUTPUT")
6.
7.  digitalWrite(40, "HIGH")
8.  digitalWrite(41, "LOW")
9.
10. while(1):
11.         if digitalRead(39):
12.                 delay(300)
13.                 digitalWrite(40, not digitalRead(40))
14.                 digitalWrite(41, not digitalRead(41))
```

Foi implementado um pequeno trecho de código na linha 12, logo após a detecção de sinal, que simboliza um retardamento de 150 milissegundos antes do prosseguimento na execução do código. Esse retardamento trabalha com o objetivo de dar tempo para que haja encerramento da trepidação mecânica do resistor *pull-down*, que ocorre quando este é

pressionado. Esta trepidação pode ser detectada como vários pressionamentos sucessivos e, por isso, a espera pelo seu encerramento é uma solução simples e de fácil implementação para o problema.

Nos testes, a biblioteca portou-se conforme o desejado.

4.2. VALIDAÇÃO DA BIBLIOTECA VOCORE 2 STEPPER MOTOR

Para que a biblioteca fosse testada, o Vocore 2 foi ligado a um *driver* modelo DRV8825 e o *driver* ligado ao motor de passo Nema 17 modelo VZS1734-028-0404, com 2,8 Kgf.cm, 200 passos por revolução, 0,4A de corrente por fase e 12V, conforme a Figura 9.

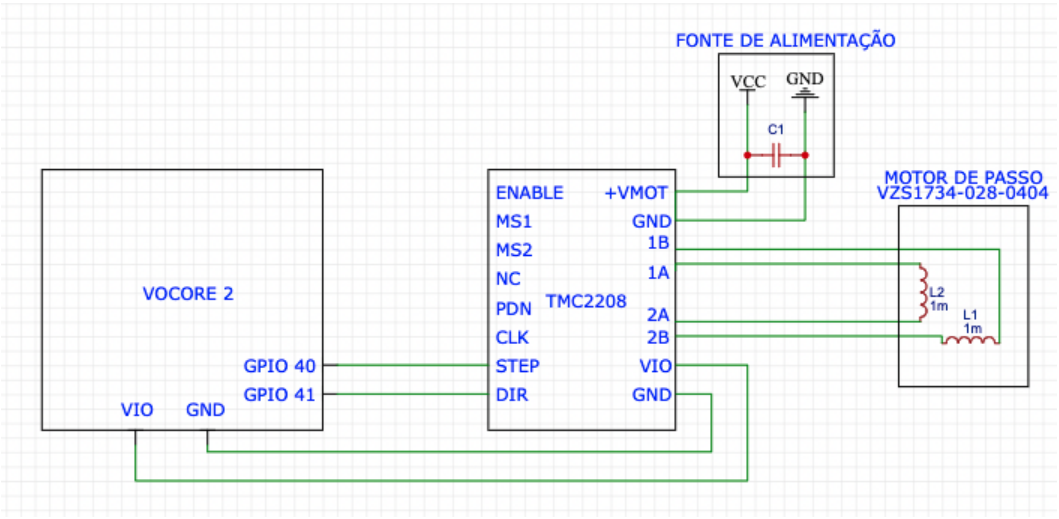

Figura 9: Circuito do driver modelo DRV8825 ligado ao motor de passo Nema 17 VZS1734-028-0404.

A testagem[8] ocorreu por meio de enviar comandos para que o motor de passo executasse ângulos específicos com certa exatidão. O cálculo do número de passos necessários ocorreu a partir da especificação própria do motor de 200 passos por revolução. Dessa maneira, cada passo equivale a um ângulo de 1,8º. Foram executados XX comandos de ângulos escolhidos aleatoriamente, seguidos da medição em relação ao ponto anterior no qual o eixo do motor se apresentava. A partir disso, todos os comandos mostrados a seguir foram executados de maneira satisfatória pelo motor:

```
1. from vocoreStepperMotor import *
2.
3. #Define the pins attached to the driver:
4. direction_pin = 40
5. step_pin = 41
6.
7. #Define the direction
8. direction = 1
```

[8] Disponível em <https://youtu.be/4HxdrzttOts> (BARBOSA, 2022)

```
9.
10. #Define the number of steps for revolution according to the Stepper Motor
    specs
11. steps_per_revolution = 200
12.
13. #Create an object Stepper Motor
14. sm = stepper(direction_pin, direction, step_pin, steps_per_revolution)
15.
16. #Now, our Stepper Motor is ready:
17. delay(2000) #Wait for 2 seconds
18. sm.turn_xdegrees(180) #Sends a command to the Stepper Motor to turn 180
    degrees
19. delay(2000) #Wait for 2 seconds
20. sm.turn_xdegrees(180) #Sends a command to the Stepper Motor to turn 180
    degreesdelay(2000) #Wait for 2 seconds
21. sm.turn_xdegrees(360) #Sends a command to the Stepper Motor to turn a full
    revolution
```

4.3. VALIDAÇÃO DA BIBLIOTECA VOCORE 2 ADS1115

Tendo em vista que o ADS1115 por si só não possui valores a serem retornados, carecendo de um dispositivo gerador de sinais analógicos para que estes sejam enviados ao Vocore 2, o teste da biblioteca[9] ocorreu por meio da ligação de um potenciômetro de escala linear de 10KΩ às portas analógicas. Mediante a variação do eixo deste, houve verificação da existência de leituras coerentes no Vocore 2 paralelamente a um Voltímetro e um Arduino Uno, conforme o circuito esquematizado na Figura 10.

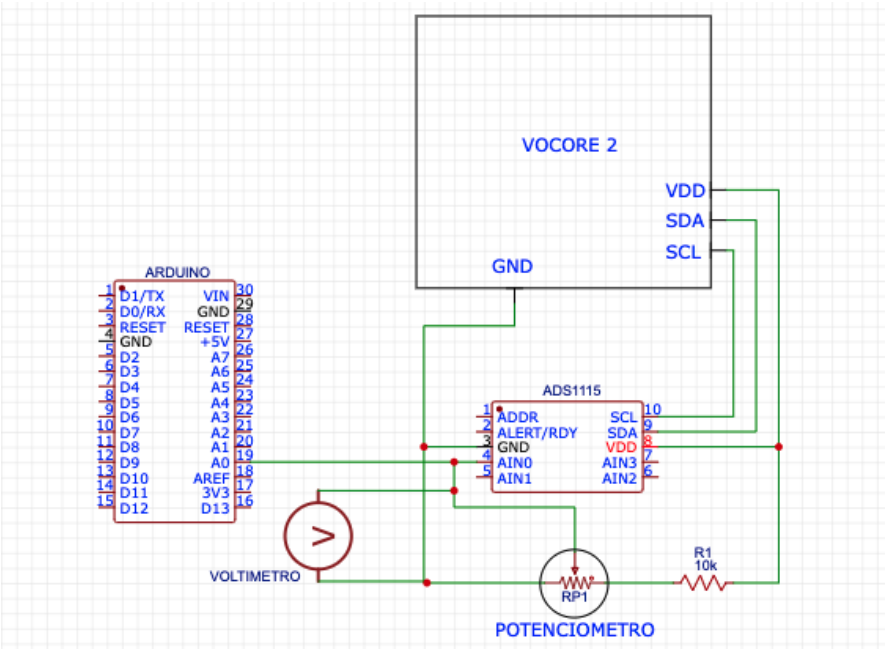

Figura 10: Circuito do de ligação do potenciômetro a um Voltímetro, um Arduino UNO e ao ADS1115, que, por sua vez, está conectado ao Vocore 2.

[9] Disponível em <https://youtu.be/-LGog3VDHAM> (BARBOSA, 2022)

Os resultados ocorreram conforme o esperado com a variação retornada compatível com as rotações realizadas no eixo do potenciômetro. Cabe ressaltar, ainda, que, no Vocore 2, através do conversor analógico-digital ADS1115, que possui resolução muito maior que a resolução das portas analógicas de um Arduino Uno (16 bits do primeiro contra 10 bits do segundo), a sensibilidade da rotação no eixo do potenciômetro foi muito maior, resultado coerente com a diferença na amplitude de valores entre os dois, sendo esta o intervalo inteiro [0, 65535] para o ADS1115 e [0, 127] para o Arduino Uno. O código fonte do teste realizado encontra-se a seguir:

```
1. from vocore2ads1115 import *
2. from time import sleep
3.
4. adc = ads1115()
5.
6. while(1):
7.     print("Potentiometer value:", adc.read_pin_converted(0))
8.     sleep(0.1)
```

4.4. VALIDAÇÃO DA BIBLIOTECA VOCORE 2 MPU6050

A biblioteca para o dispositivo MPU-6050 pôde ser validada[10] a partir do seguinte código e da verificação da compatibilidade dos resultados segundo as regras de conversão dos valores especificadas no *datasheet* do dispositivo. O código de teste se deu conforme mostrado a seguir:

```
9.  from vocore2mpu6050 import *
10. from time import sleep
11.
12. acc_gyro = mpu6050()
13.
14. while(1):
15.     ax = acc_gyro.read_rescaled_data("ax")
16.     ay = acc_gyro.read_rescaled_data("ay")
17.     az = acc_gyro.read_rescaled_data("az")
18.     gx = acc_gyro.read_rescaled_data("gx")
19.     gy = acc_gyro.read_rescaled_data("gy")
20.     gz = acc_gyro.read_rescaled_data("gz")
21.     print("Accelerometer\nX:", ax, "Y:", ay, "Z:", az)
22.     print("Gyroscope\nX:", gx, "Y:", gy, "Z:", gz)
23.     print("::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::")
24.     sleep(0.7)
```

[10] Disponível em <https://youtu.be/7DlBx80uvyo> (BARBOSA, 2022)

4.5. VALIDAÇÃO DA BIBLIOTECA VOCORE 2 TMP117

A validação da biblioteca Vocore 2 TMP117 não pôde ser realizada devido ao fato do sensor TMP117 não estar acessível no momento da finalização deste trabalho, estando em processo de importação, por não ser um dispositivo presente no mercado nacional.

5 CONSIDERAÇÕES FINAIS

O presente trabalho atendeu aos objetivos propostos ao prover uma base de *software* para a implementação de todo o sistema que controlará o *hardware* do projeto em desenvolvimento. Os resultados que puderam ser validados foram satisfatórios e, por meio do que foi obtido, temos o leque de recursos passíveis de implementação: (a) controlar reservatórios e a injeção de hormônios, graças a biblioteca de controle de motores de passo "Vocore 2 Stepper Motor"; (b) obter dados de movimentação/agitação do animal, por meio da biblioteca "Vocore 2 MPU6050"; (c) aferir temperatura corporal do animal, por meio da biblioteca "Vocore 2 TMP117"; (d) acoplar sensores infravermelho e desenvolver *encoders* lineares e medidores de bateria por meio do conversor analógico-digital ADS1115 e de sua respectiva biblioteca "Vocore 2 ADS1115"; e (d) medir a luminosidade do ambiente, por meio da biblioteca "Vocore 2 BH-1750".

O presente trabalho, ainda que como anteriormente dito, satisfatório, possui limitações inerentes advindas das escolhas tomadas no decorrer do projeto. Dentre estas limitações, pode-se citar: (a) a escolha da linguagem Python, possui em si o ônus de se tratar de uma linguagem interpretada (ROSSUM, 1998), o que leva a execução do código a não ter desempenho tão eficiente quanto seria em C/C++, ou até mesmo em variantes como MicroPython e Cython; (b) O Vocore 2, ainda que satisfatório no desempenho, ainda é limitado em seu poder de processamento (VOCORE STUDIO, 2022), o que pode vir a ser um problema futuro mediante a evolução da complexidade dos modelos matemáticos; (c) alguns dos dispositivos escolhidos, tal qual o próprio Vocore 2 e o sensor de temperatura TMP117, não estão disponíveis no mercado nacional, o que é um fator dificultante para a reprodução do projeto, de seus resultados e da sua progressão em solo nacional.

Por outro lado, haverá continuidade do projeto e progressão na parte de *hardware* para criação do dispositivo. Dentre implementações futuras para o projeto no ramo da computação física, há: (a) o desenvolvimento de *encoders* lineares para verificação da posição do atuador mecânico linear que estará acoplado aos motores de passo; (b) o desenvolvimento de um medidor de bateria; (c) implementação de códigos que, por meio do acoplamento de sensores infravermelho às portas analógicas do conversor ADS1115, possibilitará a conversão das leituras em aferições de oxigenação sanguínea, eletrocardiograma (ECG), frequência cardíaca e outros sinais biológicos; e (d) a implementação de um sistema que opere todas as funções de *hardware* implementadas em paralelo à computação de modelos e à conexão com a rede.

Por fim, é importante destacar que todo o material gerado no presente trabalho, desde as bibliotecas, até este documento em si, será disponibilizado ao público com vistas a fomentar a pesquisa na área.

REFERÊNCIAS

AL-TURJMAN, F.; NAWAZ, M. H.; ULUSAR, U. D. Intelligence in the Internet of Medical Things era: A systematic review of current and future trends. **Computer Communications**, 2019. Disponível em: doi: https://doi.org/10.1016/j.comcom.2019.12.030 Acesso em 09 de maio, 2022.

AMS. **Advanced Micro Systems**. Stepper Motor System Basics. Disponível em: <http://stepcontrol.com/pdf/step101.pdf> Acesso em 31 de julho, 2022.

ARDUINO DOCS. **Arduino Nano 33 Ble**. 2022. Disponível em: <https://docs.arduino.cc/hardware/nano-33-ble> Acesso em 31 de julho, 2022.

COLEY, G. BeagleBone Black system reference manual. **Beagleboard.org**, 2014. Disponível em: <https://cdn-shop.adafruit.com/product-files/1996/BBB_SRM.pdf> Acesso em 31 de julho, 2022.

EPSON. **Epson Exceed Your Vision**. Circuit design information, 2022. Disponível em: <https://www5.epsondevice.com/en/information/technical_info/gyro/> Acesso em 01 de agosto, 2022.

EXPLORE EMBEDDED. **Overview of ESP32 features - what do they pratically mean**, 2017. Disponível em: <https://www.exploreembedded.com/wiki/Overview_of_ESP32_features._What_do_they_practically_mean%3F> Acesso em 31 de julho, 2022.

FABLAB KANNAI. **Stepper motor driver TMC2208**, 2022. Disponível em: <https://fabacademy.org/2022/labs/kannai/Instruction/tips/stepper_TMC2208/> Acesso em 31 de julho, 2022.

GAMESS, E.; HERNANDEZ, S. Performance evaluation of different Raspberry Pi models for a broad spectrum of interests. **International Journal of Advanced Computer Science and Applications**, v. 13, n. 2, 2022, p. 819-829. Disponível em:

<https://d1wqtxts1xzle7.cloudfront.net/85279250/Paper_95-Performance_Evaluation_of_Different_Raspberry_Pi_Models-with-cover-page-v2.pdf?Expires=1659486774&Signature=B1bUOyC3l5~ZIeTKliudepiaZtHDK5kRvkfvO72ze33iiL0M1-UHLMnBhdiOTu76JCSyHXq254tsxoxo5Bo1yDw8LEC9W8bQkKodIAA6kLBKw2ci4yiyKrKEAw5vWS8UCNE1rhfrZKcyLg~vtTHQGUEXIWXYHQRZXYIqekx7B3fr-6exVNdrRHze1MgQUYnPVxBr1wtHCBtcGIZ5r7obTPSiCDEI0VwOZ4M95X5vvL1FVppR47ie2AJts0qcvA3U3l6VJmDqOZg~0u0b-e2IT3-AEk1pseQjluHRr0XrnqPJZ9ppDJnIL3ggEdurmMCCgEiF8bS0ubZ7RGdXEXP8dg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA> Acesso em 31 de julho, 2022.

GARG, N.; WAZID, M.; DAS, A. K.; SINGH, D. P.; RODRIGUES, J. J. P. C.; PARK, Y. BAKMP-IoMT: design of blockchain enabled authenticated key management protocol for internet of medical things deployment. **IEEE Access**, v. 8, p. 95956-95977, 2020. Disponível em: <https://ieeexplore.ieee.org/ielx7/6287639/8948470/09097179.pdf?tp=&arnumber=9097179&isnumber=8948470&ref=aHR0cHM6Ly9zY2hvbGFyLmdvb2dsZS5jb20uYnIv> Acesso em 08 de junho, 2022.

HAMEED SS, HASSAN WH, ABDUL LATIFF L, GHABBAN F. 2021. A systematic review of security and privacy issues in the internet of medical things; the role of machine learning approaches. PeerJ Computer Science 7:e414. Disponível em: https://doi.org/10.7717/peerj-cs.414 Acesso em 08 de junho, 2022.

HILL, G. C. Serial communications and I2C. In: **Microprocessor Based System Design**. E444 Syllabus, California State University Long Beach, 2021. Disponível em: <https://home.csulb.edu/~hill/ee346/Lectures/20%20C++%20ATmega%20I2C%20Serial%20Comm.pdf> Acesso em 31 de julho, 2022.

HO, D.; ZHAO, X.; GAO, S.; HONG, C.; VATNER, D. E.; VATNER, S. F. Heart rate and electrocardiography monitoring in mice. **Curr. Protoc. Mouse Biol.**, v. 1, 2012, p. 123-139. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3130311/#:~:text=heart%20rate%20variability.-,Non%2Dinvasive%20ECG%20system,the%20analysis%20of%20heart%20rates> Acesso em 09 de maio, 2022.

INVENSENSE. **MPU-6000 and MPU-6050 product specification revision 3.4**, 2013. Disponível em: <https://s3-sa-east-1.amazonaws.com/robocore-lojavirtual/974/MPU-6000-Datasheet1.pdf> Acesso em 31 de julho, 2022.

INVENSENSE. **MPU-6000 and MPU-6050 register map and descriptions revision 4.2**, 2013. Disponível em: <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf> Acesso em 31 de julho, 2022.

ISIKDAG, U. Internet of Things: Single-Board Computers. In: **Enhanced Building Information Models**, SpringerBriefs in Computer Science, p. 43-53, 2015.

JIN, T. **OpenWrt Development Guide**. Wireless Networks Lab, CCIS, NEU, 2012. Disponível em: <http://www.liwangmeng.com/wp-content/uploads/2016/01/OpenWrt_Dev_Tutorial.pdf> Acesso em 31 de julho, 2022.

JOYIA, G. J.; LIAQAT, R. M.; FAROOQ, A.; REHMAN, S. Internet of Medical Things (IOMT): Applications, Benefits and Future Challenges in Healthcare Domain. **Journal of Communications**, v. 12, n. 4, p. 240-247, 2017.

KOUTRAS, D.; STERGIOPOULOS, G.; DASAKLIS, T.; KOTZANIKOLAOU, P.; GLYNOS, D.; DOULIGERIS, C. Security in IoMT communications: a survey. **Sensors**, v. 20, n. 17, p. 1-49, 2020. Disponível em: <https://mdpi-res.com/d_attachment/sensors/sensors-20-04828/article_deploy/sensors-20-04828-v2.pdf?version=1598606141> Acesso em 8 de junho, 2022.

KUSHNER, D. **The making of Arduino**. IEEE Spectrum, 2011. Disponível em: <https://web.eecs.umich.edu/~prabal/teaching/resources/eecs582/kushner11arduino.pdf> Acesso em 31 de julho, 2022.

MAUERER, W. **Professional Linux Kernel Architecture**. Wiley Publishing, Inc., Indianapolis, 2008.

OMEGA. **Omega Engineering**, 2003-2022. Accelerometer: What is it and how it works. Disponível em: <https://www.omega.com/en-us/resources/accelerometers> Acesso em 01 de agosto, 2022.

ORACLE. General Purpose Input/Output. In: **Oracle Java ME Embedded Developer's Guide**, Release 8, 2014. Disponível em: <https://docs.oracle.com/javame/8.0/me-dev-guide/gpio.htm> Acesso em 31 de julho, 2022.

ORIENTAL MOTOR. **Oriental Motor do Brasil Ltda.** Motor de Passo Overview, 2017. Disponível em: <https://www.orientalmotor.com.br/motores-de-passo/technology/stepper-motor-overview.html> Acesso em 31 de julho, 2022.

PASSARO, V. M. N.; CUCCOVILLO, A.; VAIANI, L.; CARLO, M. de; CAMPANELLA, C. E. Gyroscope technology and applications: a review in the industrial perspective. **Sensors**, v. 17, p. 1-22, 2017. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5677445/pdf/sensors-17-02284.pdf> Acesso em 01 de agosto, 2022.

PROGRAM LIBRARY. In: OXFORD University Press. **Encyclopedia.com**, 2019. Disponível em: <https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/program-library> Acesso em 09 de maio, 2022.

RASPBERRY PI. **Raspberry Pi documentation**, 2012-2022. Microcontrollers. Disponível em: <https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html> Acesso em 31 de julho, 2022.

RASPBERRY PI LTDA. **Raspberry Pi Pico datasheet,** 2022. Disponível em: <https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf> Acesso em 31 de julho, 2022.

ROHM. **Digital 16bit serial output type ambient light sensor IC**, 2010. Disponível em: <https://www.filipeflop.com/img/files/download/Datasheet-bh1750fvi.pdf> Acesso em 31 de julho, 2022.

ROSSUM, G. Glue it all together with Python. **OMG-DARPA-MCC Workshop on Compositional Software Architecture,** California, 1998. Disponível em: <https://www.python.org/doc/essays/omg-darpa-mcc-position/> Acesso em 16 de julho, 2022.

SCERVINI, M. **Thermoelectric materials for thermocouples.** University of Cambridge, 2009. Disponível em: <https://www.msm.cam.ac.uk/utc/thermocouple/pages/ThermocouplesOperatingPrinciples.html> Acesso em 31 de julho, 2022.

SOUZA, A. R. de; PAIXÃO, A. C.; UZÊDA, D.D.; DIAS, M. A.; DUARTE, S.; AMORIM, H. S. de. A placa Arduino: uma opção de baixo custo para experiências de física assistidas pelo PC. **Revista Brasileira de Ensino de Física**, v. 33, n. 1, 2011. Disponível em: <https://www.scielo.br/j/rbef/a/FWYNZZqJJgkchRqBQcLbYyh/?lang=pt#> Acesso em 01 de agosto, 2022.

TEXAS INSTRUMENTS. **Ultra-small, low-power, 16-bit analog-to-digital converter with internal reference**, 2009. Disponível em: <https://www.filipeflop.com/img/files/download/Datasheet_ADC_ads1115.pdf> Acesso em 31 de julho, 2022.

TEXAS INSTRUMENTS. **TMP-117 high-accuracy, low-power, digital temperature sensor with SMBus - and I2C-compatible interface**, 2021. Disponível em: <https://www.ti.com/lit/ds/symlink/tmp117.pdf?ts=1659577301300&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTMP117%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Dasc-null-null-gpn_en-cpc-pf-google-soas%2526utm_content%253Dtmp117%2526ds_k%253DTMP117%2526dcm%253Dyes%2526gclid%253DCj0KCQjwuaiXBhCCARIsAKZLt3mFf8eHqq1tl8SXmHgfMY1zq-RnbA5MQ-yzVdMXOrpEhUX9JGLLjmcaApflEALw_wcB%2526gclsrc%253Daw.ds> Acesso em 01 de agosto, 2022.

THE LINUX KERNEL. Introduction to I2C and SMBus. In: **The Linux Kernel documentation.** 2022. Disponível em: <https://docs.kernel.org/i2c/summary.html> Acesso em 16 de julho, 2022.

THE LINUX KERNEL ARCHIVES. **GPIO Documentation**, 2022. Disponível em: <https://www.kernel.org/doc/Documentation/gpio/gpio.txt> Acesso em 31 de julho, 2022.

THOTHADRI, M. An analysis on clock speeds in Raspberry Pi Pico and Arduino Uno microcontrollers. **American Journal of Engineering and Technology Management**, v. 6, n. 3, 2021, p. 41-46. Disponível em: <https://www.researchgate.net/profile/Madhavan-Thothadri/publication/353149807_An_Analysis_on_Clock_Speeds_in_Raspberry_Pi_Pico_and_Arduino_Uno_Microcontrollers/links/60e9beec0fbf460db8fa6617/An-Analysis-on-Clock-Speeds-in-Raspberry-Pi-Pico-and-Arduino-Uno-Microcontrollers.pdf> Acesso em 31 de julho, 2022.

TMR EUROPE. **Thermocouple colour codes & tolerances**, 2015. Disponível em: <https://tmseurope.co.uk/applications/thermocouple-rtd-colour-codes-tolerances> Acesso em 31 de julho, 2022.

TRONICSBENCH. **Tronicsbench.com**, 2005-2022. I2C Tutorial: All you need to know about I2C. Disponível em: <https://www.best-microcontroller-projects.com/i2c-tutorial.html> Acesso em 31 de julho, 2022.

VERWEY; M.; ROBINSON, B.; AMIR, S. Recording and analysis of circadian rhythms in running-wheel activity in rodents. **Journal of Visualized Experiments: JoVE**, v. 71, 2013. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3582575/> Acesso em 09 de maio.

VOCORE STUDIO. **Vocore**, 2013-2022. VoCore is open hardware and runs Linux(OpenWrt). It has 128MB DDR, WIFI, USB, UART, I2C, SPI, 20+ GPIOs but only one inch square(25.8mm). Disponível em: <https://vocore.io>. Acesso em 09 de maio, 2022.

APÊNDICE A - Vocore 2 GPIO

```
1.  import time
2.
3.  def pinMode(num_pin, mode): #Defines pin mode as OUTPUT or
4.      config = file('/sys/class/gpio/export', 'w') #exporting the
    GPIO%num_pin
5.      config.write(str(num_pin))
6.      config.close()
7.
8.      pin_path = "/sys/class/gpio/gpio%d/" % num_pin #directory where all the
    files for controlling the pin <num_pin> are stored
9.      file_direc = file(pin_path + "direction", "w") #Opens the file to write
    data. If we write "out", the pin goes to OUTPUT mode, if we write "in",
    the pin goes to INPUT mode.
10.     value = "out"
11.
12.     '''
13.     If mode is setted as True, "true", "1", "output", "out", 1 or 1.0, the
    pin will be setted as output.
14.     Else, pin will be setted as input.
15.     '''
16.     if type(mode) is bool: #Any of this parameters leads to put the pin as
    OUTPUT
17.         if mode:
18.             value = "out"
19.         else:
20.             value = "in"
21.
22.     elif type(mode) is str:
23.         mode = mode.lower()
24.         if mode  == "true" or mode == "1" or mode == "output" or mode ==
    "out":
25.             value = "out"
26.         else:
27.             value = "in"
28.
29.     elif type(mode) is int or type(mode) is float:
30.         if mode == 1 or mode == 1.0:
31.             value = "out"
32.         else:
33.             value = "in"
34.
35.     else:
36.         value = "in"
37.
38.     file_direc.write(value)
39.     file_direc.close()
40.
41.
42. def digitalWrite(num_pin, state):
43.     pin_path = "/sys/class/gpio/gpio%d/" % num_pin
44.     file_direc = file(pin_path + "direction", "r") #Gets the info if the
    pin is in OUTPUT mode. We can only use digitalWrite on pins in OUTPUT
    mode.
45.     pin_mode = file_direc.read()
46.     file_direc.close()
47.
48.     '''
49.     If state is setted as True, "true", "1", "high", "on", 1 or 1.0, the
    pin will be setted in high state.
50.     Else, the pin will be setted in low state.
51.     '''
52.     if pin_mode == "out" or pin_mode == "out\n" or pin_mode == "out \n":
53.         value = "0"
54.         file_value = file(pin_path + "value", "w") #Opens the file that
```

```
    determinates if the pin is setted as ON or OFF.
55.
56.         if type(state) is bool:
57.             if state:
58.                 value = "1"
59.             else:
60.                 value = "0"
61.
62.         elif type(state) is str:
63.             state = state.lower()
64.             if state == "true" or state == "1" or state == "high" or state
    == "on":
65.                 value = "1"
66.             else:
67.                 value = "0"
68.
69.         elif type(state) is int or type(state) is float:
70.             if state == 1 or state == 1.0:
71.                 value = "1"
72.             else:
73.                 value = "0"
74.
75.         else:
76.             value = "0"
77.
78.         file_value.write(value)
79.         file_value.close()
80.
81.     else:
82.         print("Pin {} is not in OUTPUT mode.".format(num_pin))
83.
84.
85. def digitalRead(num_pin):
86.     pin_path = "/sys/class/gpio/gpio%d/" % num_pin
87.     file_value = file(pin_path + "value", "r") #Gets info about the pin
    state
88.     state = file_value.read()
89.     file_value.close()
90.
91.     if state == "1" or state == "1\n" or state == "1 \n": #If the pin is
    setted as ON or is receiving a reading, the return will be 1
92.         return 1
93.
94.     return 0
95.
96.
97. def delay(t): #Delay in Milliseconds
98.     time.sleep((t/1000))
99. def delayMicroseconds(t): #Delay in Microseconds
100.        time.sleep((t/1000000))
```

APÊNDICE B - Vocore 2 Stepper Motor

```
1.  from vocoreGPIO import *
2.
3.  class stepper:
4.      def __init__(self, step_pin, steps_per_revolution, direction_pin,
    direction = 1, enable_pin = -1, enable_on=False, pulse_width=5,
    driver="drv8825", micro_stepping=1, micro_stepping_pins=()):
5.          self.direction_pin = direction_pin
6.          self.direction = direction
7.          self.step_pin = step_pin
8.          self.steps_per_revolution = steps_per_revolution
9.          self.enable_pin = enable_pin
10.         self.enable_on = enable_on
11.         self.pulse_width = pulse_width
12.         self.driver = driver
13.         self.micro_stepping = micro_stepping
14.         self.micro_stepping_pins = micro_stepping_pins
15.         self.micro_stepping_on = False
16.
17.         self.setup_pins()
18.
19.     def setup_pins(self): #Initial pin setup for mode and state for
    direction, step and enable
20.         pinMode(self.direction_pin, True)
21.         pinMode(self.step_pin, True)
22.
23.         if self.micro_stepping != 1 and len(self.micro_stepping_pins) != 0:
24.             for i in self.micro_stepping_pins: #Sets each of the micro
    stepping pins as OUTPUT
25.                 pinMode(i, True)
26.
27.         digitalWrite(self.direction_pin, self.direction)
28.         digitalWrite(self.step_pin, False)
29.
30.         if self.enable_pin != -1:
31.             pinMode(self.enable_pin, True)
32.             digitalWrite(self.enable_pin, self.enable_on)
33.
34.     def turn_off_microstepping(self): #Turns off the pins attached to
    microstepping ports on driver
35.         if self.micro_stepping_on:
36.             for pin in self.micro_stepping_pins:#Sets each of the micro
    stepping pins signal as LOW
37.                 digitalWrite(pin, False)
38.
39.             self.micro_stepping_on = False
40.
41.     def setup_micro_stepping(self): #Sets up the microstepping
    configuration according to the microstepping fraction and the driver model
    setted
42.         if self.driver.lower() == "drv8825" or self.driver.lower() ==
    "drv8824":
43.             self.setup_drv8825_drv8824()
44.         elif self.driver.lower() == "a4988":
45.             self.setup_a4988()
46.         elif self.driver.lower() == "tmc2208":
47.             self.setup_tmc2208()
48.
49.         self.micro_stepping_on = True
50.
51.     def setup_drv8825_drv8824(self): #Microstepping configuration for
    DRV8825 and DRV8824
52.         if self.micro_stepping == 2:
53.             digitalWrite(self.micro_stepping_pins[0], True)
54.             digitalWrite(self.micro_stepping_pins[1], False)
```

```
55.             digitalWrite(self.micro_stepping_pins[2], False)
56.         elif self.micro_stepping == 4:
57.             digitalWrite(self.micro_stepping_pins[0], False)
58.             digitalWrite(self.micro_stepping_pins[1], True)
59.             digitalWrite(self.micro_stepping_pins[2], False)
60.         elif self.micro_stepping == 8:
61.             digitalWrite(self.micro_stepping_pins[0], True)
62.             digitalWrite(self.micro_stepping_pins[1], True)
63.             digitalWrite(self.micro_stepping_pins[2], False)
64.         elif self.micro_stepping == 16:
65.             digitalWrite(self.micro_stepping_pins[0], False)
66.             digitalWrite(self.micro_stepping_pins[1], False)
67.             digitalWrite(self.micro_stepping_pins[2], True)
68.         elif self.micro_stepping == 32:
69.             digitalWrite(self.micro_stepping_pins[0], True)
70.             digitalWrite(self.micro_stepping_pins[1], False)
71.             digitalWrite(self.micro_stepping_pins[2], True)
72.         else:
73.             digitalWrite(self.micro_stepping_pins[0], False)
74.             digitalWrite(self.micro_stepping_pins[1], False)
75.             digitalWrite(self.micro_stepping_pins[2], False)
76.
77.     def setup_a4988(self): #Microstepping configuration for A4988
78.         if self.micro_stepping == 2:
79.             digitalWrite(self.micro_stepping_pins[0], True)
80.             digitalWrite(self.micro_stepping_pins[1], False)
81.             digitalWrite(self.micro_stepping_pins[2], False)
82.         elif self.micro_stepping == 4:
83.             digitalWrite(self.micro_stepping_pins[0], False)
84.             digitalWrite(self.micro_stepping_pins[1], True)
85.             digitalWrite(self.micro_stepping_pins[2], False)
86.         elif self.micro_stepping == 8:
87.             digitalWrite(self.micro_stepping_pins[0], True)
88.             digitalWrite(self.micro_stepping_pins[1], True)
89.             digitalWrite(self.micro_stepping_pins[2], False)
90.         elif self.micro_stepping == 16:
91.             digitalWrite(self.micro_stepping_pins[0], True)
92.             digitalWrite(self.micro_stepping_pins[1], True)
93.             digitalWrite(self.micro_stepping_pins[2], True)
94.         else:
95.             digitalWrite(self.micro_stepping_pins[0], False)
96.             digitalWrite(self.micro_stepping_pins[1], False)
97.             digitalWrite(self.micro_stepping_pins[2], False)
98.
99.     def setup_tmc2208(self): #Microstepping configuration for TMC2208
100.         if self.micro_stepping == 2:
101.             digitalWrite(self.micro_stepping_pins[0], True)
102.             digitalWrite(self.micro_stepping_pins[1], False)
103.         elif self.micro_stepping == 4:
104.             digitalWrite(self.micro_stepping_pins[0], False)
105.             digitalWrite(self.micro_stepping_pins[1], True)
106.         elif self.micro_stepping == 16:
107.             digitalWrite(self.micro_stepping_pins[0], True)
108.             digitalWrite(self.micro_stepping_pins[1], True)
109.         else:  #The default for TMC2208 is microstepping at 1/8
110.             digitalWrite(self.micro_stepping_pins[0], False)
111.             digitalWrite(self.micro_stepping_pins[1], False)
112.             self.micro_stepping = 8
113.
114.     def step(self, number_of_steps=1): #Sends a command to the motor to
    rotate a number of steps passed through parameter (pre-setted as 1)
115.         if self.enable_on:
116.             print("Turn off enable before sending step commands.")
117.         else:
118.             digitalWrite(self.direction_pin, self.direction)
119.             if self.micro_stepping != 1 and self.micro_stepping_on:
120.                 self.setup_micro_stepping()
```

```
121.
122.                 for i in range(number_of_steps):
123.                     digitalWrite(self.step_pin, True)
124.                     delay(self.pulse_width)
125.                     digitalWrite(self.step_pin, False)
126.                     delay(self.pulse_width)
127.
128.                 self.turn_off_microstepping()
129.
130.     def turn_xdegrees(self, degrees=360): #Sends a command to the motor to
    rotate an specific angle
131.         steps = int(((self.steps_per_revolution * self.micro_stepping) *
    degrees) / 360)
132.         self.step(steps)
133.
134.     def toggle_direction(self): #Toggle the direction setted for rotation
135.         digitalWrite(self.direction_pin, not
    digitalRead(self.direction_pin))
136.
137.     def toggle_enable(self): #Toggle the enable pin state
138.         if self.enable_pin != -1:
139.             digitalWrite(self.enable_pin, not
    digitalWrite(self.enable_pin))
140.
141.         if digitalRead(self.enable_pin) == "1":
142.             self.enable_on = True
143.         else:
144.             self.enable_on = False
145.         else:
146.             print("Enable pin not defined.")
```

APÊNDICE C - Vocore 2 MAX6675

A biblioteca Vocore 2 MAX6675[11] tem o objetivo de disponibilizar ao usuário a aferição de temperatura por meio de um termopar modelo MAX6675. O funcionamento da biblioteca se dá por meio de: 1. o envio de uma requisição (feita pela setagem do pino "*CS*" como estado lógico baixo); 2. espera de 10 milissegundos para realização da aferição pelo dispositivo; 3. recebimento da leitura de forma seriada, bit a bit e, simultaneamente, conversão do valor por meio operadores *bitwise*.

```
1.  from vocoreGPIO import *
2.
3.  class max6675:
4.      '''
5.      *Inicialize a MAX6675 sensor*
6.      SCLK The Vocore 2 pin connected to Clock
7.      CS The Vocore 2 pin connected to Chip Select
8.      MISO The Vocore 2 pin connected to Data Out
9.      '''
10.     def __init__(self, SCLK, CS, MISO):
11.         self.sclk = SCLK
12.         self.cs = CS
13.         self.miso = MISO
14.
15.         pinMode(self.cs, 'OUTPUT')
16.         pinMode(self.sclk, 'OUTPUT')
17.         pinMode(self.miso, 'INPUT')
18.
19.         digitalWrite(self.cs, 'HIGH')
20.
21.
22.     '''
23.     *Read the Celsius temperature*
24.     =Returns Temperature in C or NaN on failure!=
25.     '''
26.     def read_celsius(self):
27.         digitalWrite(self.cs, 'LOW')
28.         delayMicroseconds(10)
29.
30.         v = self.spiread()
31.         v <<= 8
32.         v |= self.spiread()
33.
34.         digitalWrite(self.cs, 'HIGH')
35.
36.         if(v & 0x4):
37.             # uh oh, no thermocouple attached!
38.             return float("NaN")
39.
40.         v >>= 3
41.
42.         return v * 0.25
43.
44.
45.     '''
46.     *Read the Fahrenheit temperature*
47.     =Returns Temperature in F or NaN on failure!=
48.     '''
49.     def read_fahrenheit(self):
```

[11] Disponível em <https://github.com/BryanCMBarbosa/Vocore2-max6675> (BARBOSA, 2022)

```
50.         return self.read_celsius() * 9.0 / 5.0 + 32
51.
52.
53.     '''
54.     *Read the Kelvin temperature*
55.     =Returns Temperature in K or NaN on failure!=
56.     '''
57.     def read_kelvin(self):
58.         return self.read_celsius() + 273.15
59.
60.
61.     def spiread(self):
62.         d = 0
63.
64.         for i in range(7, -1, -1):
65.             digitalWrite(self.sclk, 'LOW')
66.             delayMicroseconds(10)
67.             if(digitalRead(self.miso)):
68.                 # set the bit to 0 no matter what
69.                 d |= (1 << i)
70.
71.             digitalWrite(self.sclk, 'HIGH')
72.             delayMicroseconds(10)
73.
74.         return d
```

APÊNDICE D - Vocore 2 ADS1115

```
1. import smbus
2. import time
3.
4.
5. class ads1115:
6.     def __init__(self, address = 0x48):
7.           self.bus = smbus.SMBus(0)
8.           self.pins_addresses = [[0xC4,0x83], [0xD4,0x83], [0xE4,0x83],
   [0xF4,0x83]]
9.
10.    def read_pin_raw(self, pin):
11.        self.bus.write_i2c_block_data(0x48, 0x01, self.pins_addresses[pin])
12.        time.sleep(0.5)
13.        response = self.bus.read_i2c_block_data(0x48, 0x00, 2)
14.        return response
15.
16.    def read_pin_converted(self, pin):
17.        response = self.read_pin_raw(pin)
18.        raw_data = response[0] * 256 + response[1]
19.
20.        if raw_data > 32767:
21.            raw_data -= 65535
22.
23.        return raw_data
```

APÊNDICE E - Vocore 2 MPU6050

```
1.  import smbus
2.  from time import sleep
3.
4.  class mpu6050():
5.      def __init__(self, address = 0x68):
6.          self.bus = smbus.SMBus(0)
7.
8.          self.device_address = address
9.
10.         self.reading_regist = {"ax": 0x3B, "ay": 0x3D, "az": 0x3F, "gx":
    0x43, "gy": 0x45, "gz": 0x47}
11.         self.config_regist = {"smp_rt_rgt": 0x19, "pwr_mg": 0x6B, "cfg":
    0x1A, "g_cfg": 0x1B, "itrpt_enab_rgt": 0x38}
12.
13.         self.bus.write_byte_data(self.device_address,
    self.config_regist["smp_rt_rgt"], 7)
14.         self.bus.write_byte_data(self.device_address,
    self.config_regist["pwr_mg"], 1)
15.         self.bus.write_byte_data(self.device_address,
    self.config_regist["cfg"], 0)
16.         self.bus.write_byte_data(self.device_address,
    self.config_regist["g_cfg"], 24)
17.         self.bus.write_byte_data(self.device_address,
    self.config_regist["itrpt_enab_rgt"], 1)
18.
19.     def read_raw_data(self, regist_id):
20.         regist_id = regist_id.lower()
21.         high_regist_reading = self.bus.read_byte_data(self.device_address,
    self.reading_regist[regist_id])
22.         low_regist_reading = self.bus.read_byte_data(self.device_address,
    self.reading_regist[regist_id]+1)
23.
24.         reading = ((high_regist_reading << 8) | low_regist_reading)
25.
26.         if(reading > 32768):
27.                 reading -= 65536
28.         return reading
29.
30.     def read_rescaled_data(self, regist_id):
31.         if 'a' in regist_id:
32.             return self.read_raw_data(regist_id) / 16384.0
33.         else:
34.             return self.read_raw_data(regist_id) / 131.0
```

APÊNDICE F - Vocore 2 BH1750

```
1.  import smbus
2.
3.  class bh1750():
4.   def __init__(self, address = 0x23): #can use 0x5c
5.     self.bus = smbus.SMBus(0)
6.     self.device_address = address
7.     self.read_ids = {"o_low": 0x23, "o_high_1": 0x21, "o_high_0.5":0x20,
    "c_low": 0x13, "c_high_1": 0x10, "c_high_0.5": 0x11}
8.
9.   def read_raw_data(self, mode):
10.    data = self.bus.read_i2c_block_data(self.device_address,
    self.read_ids[mode])
11.
12.    return data
13.
14.  def read_converted_data(self, mode):
15.    response = self.read_raw_data(mode)
16.    return ((response[1] + (256 * response[0])) / 1.2)
```

APÊNDICE G - Vocore 2 TMP117

```
1.  import smbus
2.  import time
3.
4.  class tmp117():
5.      def __init__(self, address = 0x48):
6.          self.bus = smbus.SMBus(0)
7.          self.device.address = address
8.          self.r = [0x0220]
9.
10.     def read_raw_data(self):
11.         self.bus.write_i2c_block_data(0x48, 0x01, self.r)
12.         time.sleep(0.5)
13.         response = self.bus.read_i2c_block_data(0x48, 0x00, 2)
14.         return response
15.
16.     def read_converted_data(self):
17.         reading = self.read_raw_data()
18.         converted_reading = (reading[0]<<8 | reading[1])
19.         return converted_reading
20.
21.     def read_celsius_temp(self):
22.         return self.read_converted_data() * 0.0078125
23.
24.     def read_fahrenheit_temp(self):
25.         return ((self.read_converted_data() * 0.0078125) * 1.8) + 32
26.
27.     def read_kelvin_temp(self):
28.         return (self.read_converted_data() * 0.0078125) + 273.15
```