DENSÍMETRO PARA MONITORAR PRODUÇÃO ARTESANAL DE CERVEJA: COLETA E ANÁLISE DE DADOS

PATRICK VITAS REGUERA BEAL

TRABALHO DE CONCLUSÃO DE CURSO EM ENGENHARIA DE COMPUTAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DENSIMETER FOR MEASURING BEER FERMENTATION: DATA COLLECTION AND ANALYSIS

DENSÍMETRO PARA MONITORAR PRODUÇÃO ARTESANAL DE CERVEJA: COLETA E ANÁLISE DE DADOS

PATRICK VITAS REGUERA BEAL

ORIENTADOR: RICARDO ZELENOVSKY

TRABALHO DE CONCLUSÃO DE CURSO EM
ENGENHARIA DE COMPUTAÇÃO

PUBLICAÇÃO: PPGEA.TD-001/11

BRASÍLIA/DF: MAIO - 2021

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DENSÍMETRO PARA MONITORAR PRODUÇÃO ARTESANAL DE CERVEJA: COLETA E ANÁLISE DE DADOS

PATRICK VITAS REGUERA BEAL

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE GRADUANDO EM ENGENHARIA DE COMPUTAÇÃO.

APROVADA POR:

_____

Prof. Dr. Ricardo Zelenovsky – ENE/Universidade de Brasília
Orientador

_____

Prof. Dr. Edson Mintsu Hung – ENE/Universidade de Brasília
Membro Interno

_____

Prof. Dr. Eduardo Peixoto Fernandes da Silva – ENE/Universidade de Brasília
Membro Interno

BRASÍLIA, 26 DE MAIO DE 2021.

FICHA CATALOGRÁFICA

PATRICK VITAS REGUERA BEAL
Densímetro para monitorar produção artesanal de cerveja: coleta e análise de dados [Distrito Federal] 2021.
xii, 63p., 210 x 297 mm (ENE/Universidade de Brasília/UnB, Graduando em Engenharia de Computação, Engenharia de Computação, 2021).
Trabalho de Conclusão de Curso – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Arduino	2. Densímetro
3. Fermentação	4. Cerveja
I. ENE/Universidade de Brasília/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

BEAL, P. (2021). Densímetro para monitorar produção artesanal de cerveja: coleta e análise de dados. Trabalho de Conclusão de Curso em Engenharia de Computação, Publicação PPGEA.TD-001/11, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 63p.

CESSÃO DE DIREITOS

AUTOR: Patrick Vitas Reguera Beal
TÍTULO: Densímetro para monitorar produção artesanal de cerveja: coleta e análise de dados.
GRAU: Graduando em Engenharia de Computação ANO: 2021

Patrick Vitas Reguera Beal

Departamento de Engenharia Elétrica (ENE) - Universidade de Brasília
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedico este trabalho aos meus pais,
Fernanda e Patrick

AGRADECIMENTOS

Agradeço à minha família, por sempre ter me dado todo suporte possível, emocional e financeiro, para que conseguisse chegar onde cheguei.

Aos meus cachorros Madonna e Bobi, companheiros fiéis de longa data, que se foram no meio desta caminhada, mas que sempre estarão presentes em mim.

Aos meus amigos que fiz durante minha jornada na UnB, André Garrido, João Marcelo, João Victor, Lucas Campos e Rafael Chianca, que tornaram minha graduação mais agradável e motivadora.

Ao meu orientador Ricardo Zelenovsky, por ter sido um excelente professor e orientador, que empregou seu papel de forma empenhosa e admirável.

Aos bons professores que tive o prazer de conhecer na graduação, que olham os alunos como seres humanos.

Às pessoas que conheci durante a graduação, companheiras de curso ou não, que estiveram ao meu lado, me apoiando de diversas maneiras.

RESUMO

Título: Densímetro para monitorar produção artesanal de cerveja: coleta e análise de dados
Autor: Patrick Vitas Reguera Beal
Orientador: Ricardo Zelenovsky
Graduação em Engenharia de Computação
Brasília, 26 de maio de 2021

Este trabalho realiza o aprimoramento de um sistema embarcado digital com Arduino, um densímetro capaz de realizar medições de densidade em barris de fermentação de cervejas. O densímetro continha problemas com eficiência energética, alta divergência estrutural entre suas iterações, nenhum controle dinâmico sobre centro de gravidade e comunicação via rádio pouco desenvolvida para transferência dos dados da fermentação.

A partir dos conhecimentos adquiridos com a versão anterior, foram implementados novos módulos como relógio digital, que possibilitou ligar o densímetro em momentos determinados para mantê-lo desligado quando não estiver realizando medições, melhorando drasticamente a eficiência energética do projeto.

Outro módulo importante implementado foi de memória *flash* de capacidade superior, 4 MB, que foi organizada em forma de registros, gerando maior organização dos dados para múltiplas fermentações. Também foram realizadas modificações e refatorações no programa de testes já existente, incorporando as novas funcionalidades do densímetro.

Um módulo de vibrador foi adicionado ao densímetro para tentar disseminas as bolhas que grudam em seu corpo durante as fermentações, porém com ensaios iniciais ele não se mostrou eficaz em remover bolhas de refrigerante.

Neste projeto, a melhoria principal foi a implementação de um software feito em linguagem Python com interface gráfica, capaz de apresentar medidas e cálculos importantes em formato gráficos interativos, para a análise de fermentação.

Palavras-chave: Arduino, Densímetro, Fermentação, Cerveja.

ABSTRACT

Title: Densimeter for measuring beer fermentation: data collection and analysis
Author: Patrick Vitas Reguera Beal
Supervisor: Ricardo Zelenovsky
Graduate Program in UnB
Brasília, May 26th, 2021

This work is an improvement over a digital embedded system, a densimeter using Arduino to feed beer fermentation analysis with data from it's tilt angle when floating. The initial version had problems with energy management, structural differences between iterations, bad control over the densimeter's center of mass and primitive radio comunication for transfering data from the barrels.

New modules were implemented to solve those issues, such as a real time clock used to turn back on the system before taking fermentation data samples. A small vibrator were added to treat bubbles hanging on the surface of the device, but initial experiments showed no efficacy.

Changes were also made in the test program, including new functionalities from the new peripherals and code refactoring were also part of this project.

The main improvement made was the new python dashboard, which displays measurements and calculations over interactive graphs for beer fermentation analysis.

Keywords: Arduino, Densimeter, Brewing, Beer.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 HISTÓRICO

A fabricação de cerveja é um processo bastante conhecido e estudado há muitos milênios pelos seres humanos, que cada vez mais tentam aprimorá-lo para obter novas receitas da bebida. Em torno de 6000 a.C. (YOUNG, 2021), os Sumérios utilizavam da fermentação da cevada em água, formando pedaços condensados de matéria orgânica que depois eram postos a secar. Quando secos, misturavam eles em água novamente para obter um extrato de microrganismos fermentadores.

As técnicas e a bebida em si popularizaram-se para a Europa por meio do Oriente Médio. Relatos, feitos por historiadores locais como Plínio e Tacitus (MCDONALD, 2008), do século I a.C, mostram que a cerveja era consumida por povos diversos como saxões e tribos germânicas. Portanto a fabricação de cerveja já tinha alcançado uma população significativamente maior, fazendo com que a bebida fosse diversamente modificada para atender aos paladares curiosos dos Europeus, que não poupavam esforços com os instrumentos e conhecimentos da época para aprimorar a bebida.

No período da Idade Média, a fabricação de cervejas foi classificada pelos governos locais como artesanato, o que ajudou na sua popularização juntamente com teor alcoólico mais baixo do que bebidas como o vinho, bebida também amplamente consumida, e o seu custo de fabricação ser significativamente mais baixo. Eram comumente colocadas ervas como gengibre e lúpulo, para conceder diversificação no aroma e sabor. Com isso, a produção de cervejas ganhou força suficiente para terem mosteiros próprios (MORADO, 2009).

Porém, seguindo a história para a Revolução Industrial, a produção de cerveja atingiu a sua maior escala, visto que era um produto altamente demandado por uma população em constante crescimento. Isso conferiu menos diversidade e mais acessibilidade, já que as empresas miravam no maior público alvo possível, optando por aromatizações e sabores sem muito requinte, que agradassem a todos.

A história da cerveja também está presente nos nomes comumente usados no ramo, como a palavra "malte", oriunda do inglês *malt*, que por sua vez tem origem anglo-saxã. Etimologicamente, a raiz da palavra "*mel-*" indica "maciez", que condiz com a técnica utilizada de deixar os grãos de cevada germinados antes de fermentá-los, conferindo-os essa maciez, citada anteriormente.

Já mais recentemente, a fabricação de cerveja tem voltado às suas raízes, de forma que a popularização da chamada "Cerveja Artesanal" é algo que ocorre há diversos anos. A cerveja

retoma assim a sua "tradicionalidade", com cada vez mais pequenos produtores simpatizantes do produto produzindo-o em suas casas.

1.2 MOTIVAÇÃO

O projeto em questão nasceu da problemática que atualmente atinge as fabricação de cervejas: a necessidade de constantemente monitorar a taxa de fermentação nos barris em produção. O método usado atualmente consiste na abertura destes barris e o uso de densímetros específicos para realizar as medições e constatações necessárias.

Um dos densímetros utilizados para realizar estas medições são conhecidos como varetas de vidro. Inicialmente uma amostra é retirada da cerveja em fermentação e analisada sua temperatura. Depois é colocada a vareta de vidro de forma que ela indique, após um período de estabilização de sua flutuação, a gradação no densímetro de acordo com sua altura flutuante. Posteriormente um cálculo com a temperatura é feito afim de considerá-la no processo, tendo assim uma indicação menos tendenciosa.

O refratômetro também é um densímetro disseminado no ecossistema cervejeiro. Ele funciona por meio da análise da refração da luz permeada pelo líquido, indicando assim sua densidade. Obtendo sua densidade, um cálculo matemático consegue inferir o atual nível de fermentação da amostra coletada.

O problema está na quantidade de variáveis externas que entram neste procedimento. Ao abrir os barris, o produto está sujeito a contaminação externa, bem como variações de temperatura que podem prejudicar o resultado final da fermentação, já que o cálculo preciso está intimamente ligado a ela.

Outro problema que surge no processo tradicional de fabricação está na análise de forma constante do líquido a fermentar, propagando então os problemas listados anteriormente para cada examinação realizada na cerveja. Uma terceira variável problema é o erro instrumental dos densímetros utilizados.

1.3 PROPOSTA DE SOLUÇÃO

Figura 1.1 – Densímetro flutuando sobre a água

Partindo de um projeto já iniciado por PRIOTO (2018), que consiste em um sistema embarcado que realiza medições automáticas e periódicas do ângulo de inclinação em relação a vertical, utilizando o acelerômetro MPU6050, podíamos observar problemas como duração baixa de bateria, dificuldade em controlar o centro de gravidade do densímetro e o tamanho pequeno da memória *flash*.

Estes problemas conferiam pouca versatilidade ao densímetro, visto que a bateria não durava uma fermentação completa, e tampouco havia espaço organizado e suficiente para diversas fermentações, e as curvas de inclinação extraídas não eram consistentes.

A solução proposta neste trabalho incorpora dados de tensão de bateria e temperatura, bem como o cálculo das posições angulares e comunicação serial com o computador para a análise dos dados, além de estruturação da nova, e maior, memória em registros e a transferência deles via porta USB para um computador.

A principal melhoria proposta neste trabalho, no entanto, é a elaboração de um *Dashbo-*

ard interativo, em que o usuário poderá selecionar qual arquivo de fermentação quer analisar, com gráficos de variações angular, temperatura, tensão da bateria e os três eixos X, Y e Z do acelerômetro.

Em um trabalho elaborado por Poletti (2021), dispõe-se de outras melhorias como a implementação da rotina liga/desliga do sistema, a comunicação via rádio e uso de um relógio RTC DS3231 para possibilitar o acionamento e desligamento em intervalos pré-definidos. Para atingir a estabilização do densímetro no líquido, foi acoplado um motor com contrapeso formando um lastro, delimitado por dois sensores do tipo Hall.

Estas melhorias conferem maior controle e análise dos sucessivos ensaios realizados, aumentando também a flexibilidade de uso do densímetro, visto que sua memória *flash* é estruturada por diretórios, garantindo individualidade e persistência das medições.

1.4 PRODUTOS SIMILARES

1.4.1 iSpindle

O projeto, de origem alemã e já disponível comercialmente, utiliza alguns componentes iguais aos usados neste trabalho, como microcontrolador Arduino e acelerômetro MPU 6050. O iSpindle também se utiliza da inclinação do densímetro para estimar a densidade do líquido em que está flutuando.

Porém, recentemente um dos cálculos realizados para estimar o ângulo de inclinação (*tilt*) do iSpindle foi modificado, e se assemelha bastante aos realizados neste projeto.

Figura 1.2 – Densímetro digital iSpindle (Disponível em https://www.ispindel.de)

Como descrito em seu site e projeto GitHub, o iSpindle utiliza módulo Wi-Fi para transferência dos dados, e possui seu próprio software para as devidas interpretações.

Porém, há também de considerar que não necessariamente se terá sinal Wi-Fi em todos os contextos do densímetro, já que considerando que uma fermentação completa pode durar de 2 a 4 semanas, as pessoas tendem a deixar os barris fermentando em locais ermos, daí a garantia de sinal Wi-Fi nestes locais é duvidosa e individual.

Outro fator deste densímetro é que ele somente realiza uma medição instantânea, em comparação as medições periódicas que o densímetro deste trabalho executa.

1.4.2 Tilt, Tilt Pro & Tilt Repeater

(a) Tilt (b) Tilt Pro (c) Tilt Repeater

Figura 1.3 – Disponível em https://tilthydrometer.com/

O Tilt é um hidrômetro comercial de análise contínua indicado para fabricantes amadores de cerveja. Estes hidrômetros são medidores de densidade de líquidos que utilizam o princípio da flutuabilidade, analisando a força que o líquido exerce sobre o dispositivo em sua parte submersa, causando assim uma diferença de pressão entre as partes do hidrômetro.

O produto é calibrado de fábrica com um valor específico de gravidade em seus sensores, utilizados na aferição da densidade do líquido em que estiver flutuando. Também conta com sensor de temperatura, dado importante na fabricação de cervejas, e Bluetooth 4.0+ para transferência dos dados da análise para dispositivos iOS ou Android.

Pelo que se pode analisar, esta é a sua desvantagem em relação ao densímetro deste projeto: não existe qualquer persistência local dos dados, necessitando então que a coleta seja manual dos dados, mesmo que via Bluetooth. A captação dos dados via Bluetooth é feita por meio de um aplicativo da própria empresa.

O Tilt Pro é uma versão maior e mais robusta do Tilt, com duração de bateria de 3 a 5 anos de acordo com o fabricante. Não foi dito quantas, mas são usadas pilhas AA de lítio

da marca Energizer®. A empresa também afirma superior estabilidade em ocasiões de alto Krausen (espuma superficial), projetando então em leituras mais estáveis. Também se afirma que a antena do Tilt Pro é superior, com 20% maior alcance em relação ao Tilt, e maior precisão, um decimal extra, de leituras da temperatura e gravidade.

O Tilt Repeater é indicado para estender o alcance de um Tilt pré-existente. A empresa afirma que este produto é indicado para análise e persistência continua dos dados, atualizando e emitindo os dados a cada 2 minutos. Possui bateria interna recarregável via USB com vida útil de 6 meses.

Os produtos Tilt possuem frete internacional, e cada um custa, respectivamente 135, 250 e 60 dólares para o Tilt, Tilt Pro e Tilt Repeater.

1.5 METODOLOGIA

A metodologia deste trabalho baseou-se em sucessivas experimentações para o aprimoramento contínuo do densímetro, referente as captações do seu ângulo de inclinação, duração da bateria, persistência dos dados e ajuste fino do centro de massa.

Em suma, o projeto continha os seguintes problemas:

- Alta divergência estrutural entre iterações do densímetro
- Baixa eficiência energética
- Nenhum controle dinâmico sobre centro de gravidade
- Comunicação via rádio pouco desenvolvida
- Acúmulo de bolhas durante fermentações
- Falta de estruturação na persistência dos dados

Neste trabalho, houve a troca completa da placa em que o densímetro era montado, justificada pois havia muitos fios e inconsistências entre cada iteração feita, bem como o encaixe do densímetro em sua cápsula ser folgado ou apertado, de acordo com a iteração. Então, no trabalho de Rotta (2021), foi projetada e desenhada uma placa de circuito impresso para o densímetro e, utilizando ela neste projeto, reduziu-se drasticamente a variação entre suas iterações, bem como a quantidade de fios.

A partir deste novo circuito, foram feitos novamente sucessivos testes com os componentes do densímetro. Estes testes pertencem ao programa de testes, que conta com 21 modos

de operação, cada um elaborado para testar componentes e comportamentos, bem como simulações de funcionamento com escala de tempo reduzido.

Os devidos ajustes foram feitos à medida que as experimentações ocorreram, e o objetivo é evoluir e aprimorar cada vez mais todos estes parâmetros do densímetro, a ponto que ele esteja apto a servir seu propósito de análise de fermentações de cerveja, de forma precisa.

1.6 RESUMO DOS CAPÍTULOS

O capítulo 1 contextualiza o leitor sobre o histórico da fabricação e consumo de cerveja, detalhando sua origem e sucessivas mudanças com o passar dos séculos em sua fabricação e aspecto. Com este contexto, o capítulo segue com a motivação do projeto, inspirado em dificuldades existentes nos métodos atualmente utilizados pelos amadores da fabricação artesanal de cerveja.

Continuando, o capítulo apresenta a proposta de solução a estes problemas, mostrando o que o projeto tem a agregar na fabricação de cervejas, e certos problemas que ele resolve em comparação aos métodos atuais. Em seguida, produtos similares a este desenvolvido no projeto são brevemente detalhados, levantando suas vantagens e desvantagens. Por fim, é detalhada a metodologia do projeto, evidenciando os problemas encontrados e as melhorias feitas ao decorrer deste e de outros trabalhos.

O capítulo 2 descreve o funcionamento do *hardware* e programa Arduino do densímetro. Cada módulo trabalhado neste projeto é aprofundado em seu funcionamento, princípios e como foi utilizado, evidenciando as respectivas necessidades do projeto. Também é comentado outros componentes presentes e trabalhados neste densímetro em um projeto paralelo, que também possuem suma importância para a sua evolução.

Por fim, o capítulo 2 evidencia os atuais modos do programa de testes existentes, explicando brevemente a motivação por trás de seus desenvolvimentos. Especificamente sobre os modos 16, 17, 18 e 19, referentes a ensaios de fermentações simuladas, configurações do densímetro e calibrações do acelerômetro, foi realizado um detalhamento profundo, visto que são de suma importância para o entendimento deste trabalho.

O capítulo 3 descreve o funcionamento do *dashboard*. Inicialmente é explicada a motivação por trás de seu desenvolvimento, bem como o detalhamento de como ele é acoplado ao projeto do densímetro. Em seguida, o trabalho entra em detalhes sobre as tecnologias usadas para atingir as funcionalidades necessárias ao *dashboard*, e a motivação da escolha por Python como linguagem de programação.

Continuando, é feita uma descrição de como o programa se comunica com o densímetro, evidenciando o estabelecimento da conexão e abstrações implementadas ao projeto para a

não necessidade do usuário ter conhecimentos profundos sobre endereçamento serial, como provavelmente ocorrerá em versões futuras.

Em seguida, o capítulo entra em análise sobre o tratamento e captação dos dados que chegam ao *buffer* serial, visto que a interpretação deles é feita no *dashboard*, enquanto o Arduino transmite somente dados crus. O *dashboard* também realiza as contas matemáticas envolvidas para estimar o ângulo de inclinação do densímetro.

Por fim, a interface gráfica é detalhada, evidenciando a adequação da linguagem de programação Python escolhida para o projeto e o que ela agrega em escalabilidade e manutenção de futuras versões.

O capítulo 4 demonstra ensaios realizados durante o desenvolvimento, como captação de medidas a cada segundo, movimentos de rotação do densímetro, testes com sensor de temperatura e uma análise de fermentação real, com duração de duas semanas. Também é detalhado o ensaio de vida útil da bateria.

O capítulo 5 termina o trabalho com conclusões resultantes, bem como propostas de evoluções deste projeto para trabalhos futuros.

2 DETALHAMENTO: DENSÍMETRO E COMPONENTES

2.1 DENSÍMETRO

Como visto anteriormente, a versão anterior do densímetro possuía pontos a melhorar, como baixa eficiência energética, divergências estruturais, comunicação via rádio, dentre outros. Com as melhorias citadas implementadas neste trabalho e em outro paralelo, obtivemos o seguinte protótipo:

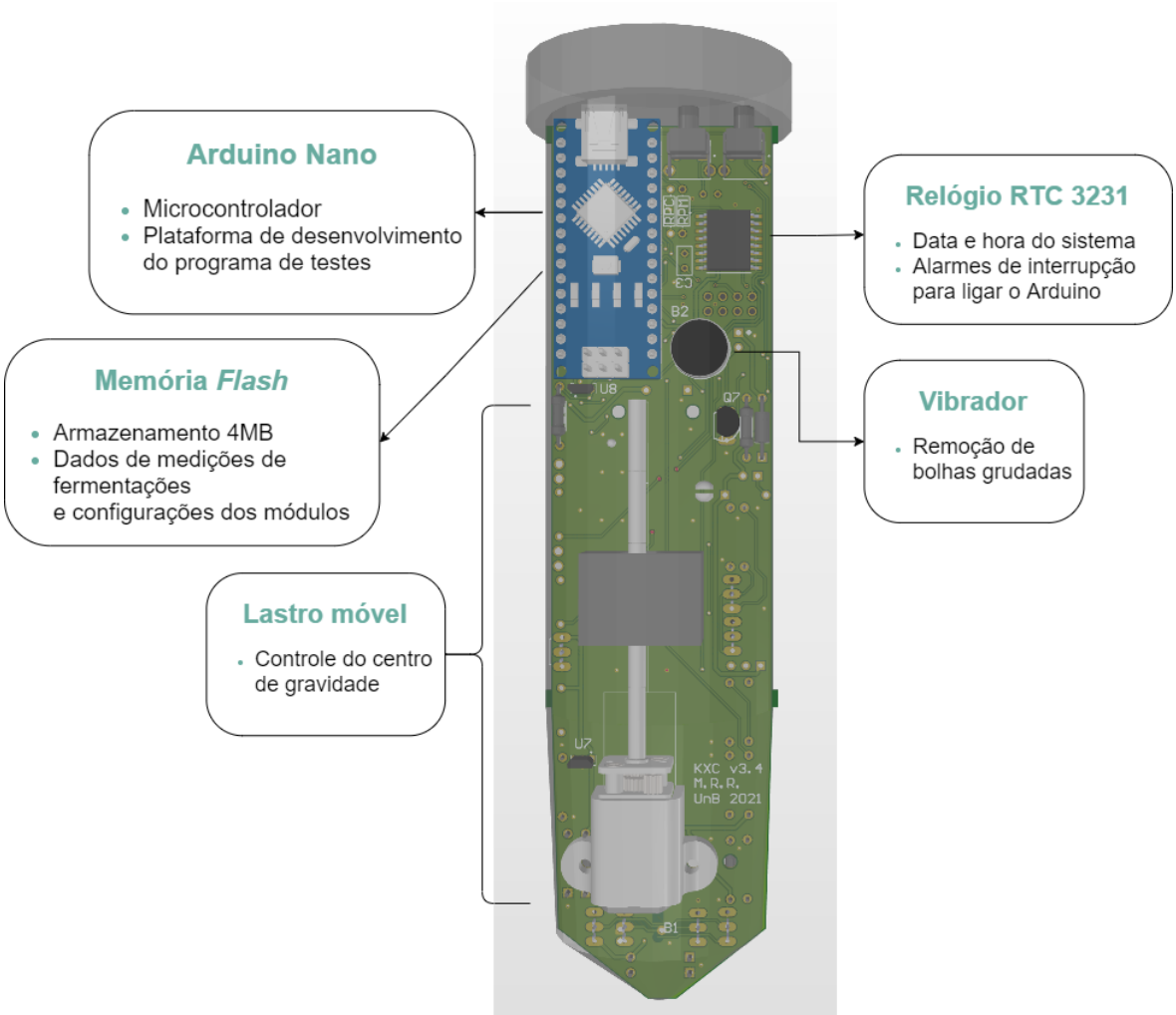


Figura 2.1 – Protótipo: visão frontal

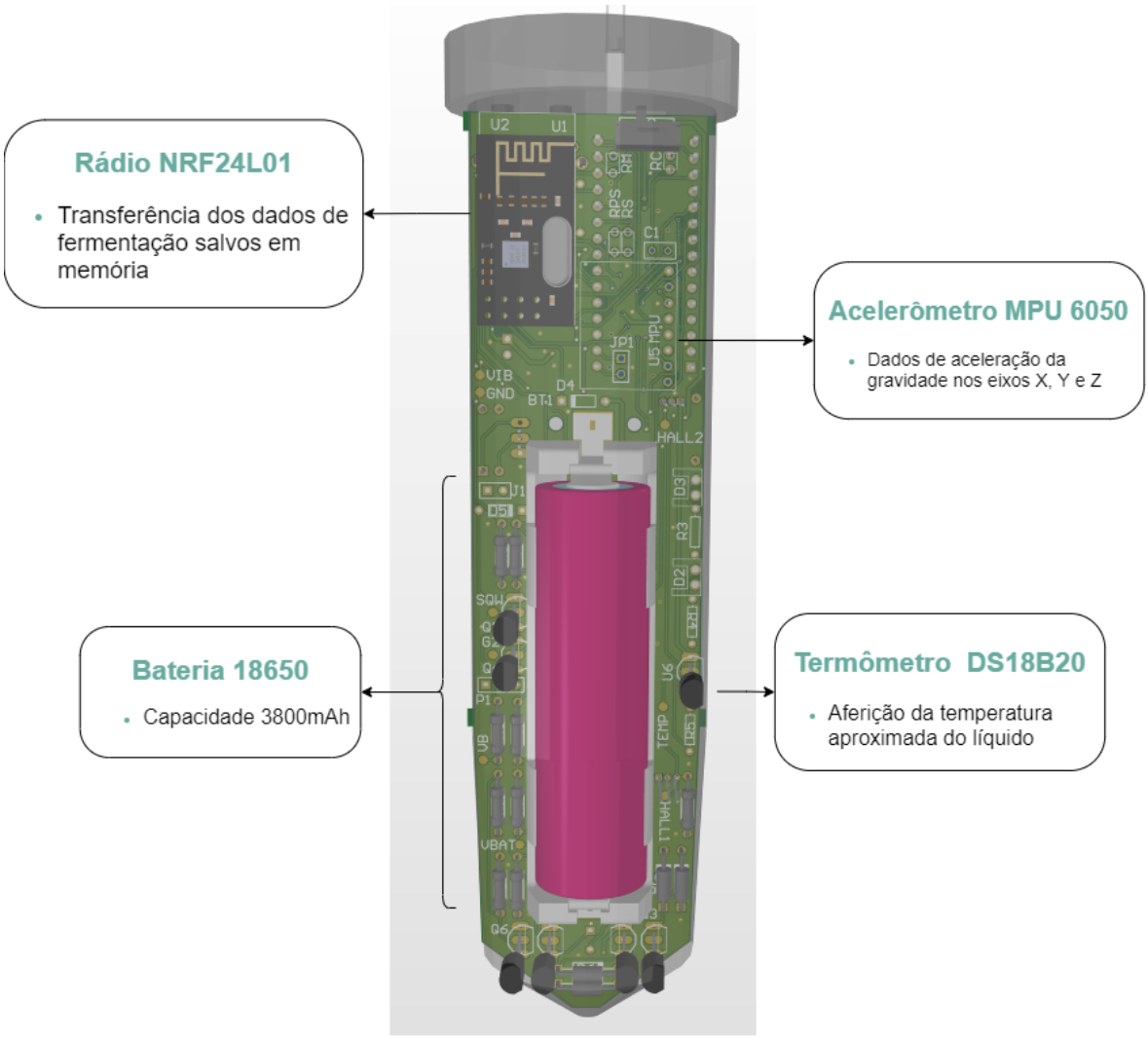


Figura 2.2 – Protótipo: visão posterior

O protótipo possui dimensões aproximadas de 16cm de altura e 2,5cm de diâmetro, em formato cilíndrico. Na primeira versão deste densímetro, Prioto (2018) deu o apelido de "cachacinha" a ele, daí surgiu-se o nome da versão atual, abreviada para KXC.

Primeiramente, procurou-se aumentar a gama de dados coletados dos sensores durante os ensaios, em relação às versões anteriores do densímetro. O acelerômetro MPU 6050 provê dados em seus registradores dos três eixos do acelerômetro e giroscópio. Com estes dados, foi possível calcular o produto vetorial entre dois vetores, um deles sendo uma referência vertical à linha do líquido, tendo assim a variação angular entre eles, denominada *tilt*.

Com isso, foi necessário estabelecer que a referência de variação angular seria o eixo vertical a linha d'água. O modo 18 de operação, um dos modos que serão descritos posteriormente neste capítulo, demonstra como é captada esta referência. Definindo então este ângulo *tilt*, tem-se uma hipótese que relaciona ele a densidade do líquido em que esta flutuando, por meio da fórmula 2.5 (BARRETO, 2019).

Foi implementado um sistema de lastro ao densímetro, com o objetivo de alterar seu centro de massa e estabilizar sua angulação durante as medições. Com ensaios de BARRETO (2019), percebeu-se que inicialmente o ângulo de inclinação em água pura deve ser próximo a 25 graus, devido a não linearidade da distribuição de massa do densímetro, o que resultava em curvas de inclinações não consistentes e pouco lineares. O modo de operação 19 realiza este posicionamento.

Assim, a medições começam e, de hora em hora, o sistema liga de forma automática e salva na memória flash do densímetro dados do acelerômetro, tensão da bateria e temperatura. O modo 21 realiza esta rotina de fermentação real.

O sistema de liga/desliga é uma melhoria realizada em um trabalho paralelo a este, em que é empregado o módulo de relógio digital RTC DS3231 para sinalizar o horário, por meio de alarme, e ligar o Arduino. Posteriormente a cada medição, o Arduino se auto desliga, o que favorece a vida útil da bateria em relação a versões passadas deste densímetro.

Vale ressaltar que, para motivos de testagem rápida, o modo 16 foi amplamente utilizado, pois simula uma fermentação real como o modo 21, porém com medidas sendo feitas a cada segundo.

Também foi acrescido um módulo vibrador, pois em ensaios passados, notou-se o acúmulo de bolhas no corpo do densímetro, o que poderia influenciar sua posição no líquido. O intuito é que ele seja acionado imediatamente antes de se realizar uma medição dos sensores, em uma tentativa de diminuir a quantidade de bolhas coladas no densímetro.

Quando o usuário coletar as informações salvas na memória flash, por meio do rádio acoplado também em um módulo externo, os dados serão transmitidos dele a um computador via conexão USB.

O computador utilizado para receber os dados os analisará por meio de um software desenvolvido neste trabalho, o KXC *Dashboard*. O *dashboard* é uma plataforma interativa, com interface, que apresenta quatro gráficos informativos de tensão da bateria, temperatura, variação angular e as três coordenadas X, Y e Z do acelerômetro. Foi desenvolvido utilizando linguagem de programação Python e o *framework* PyQT para elaborar a interface gráfica.

2.2 ARDUINO

Microcontroladores são instrumentos de pequena escala contendo um processador, que, juntamente com uma memória e barramentos de dispositivos I/O, são acoplados em um Circuito Integrado (C.I.). Eles entraram para o mercado consumidor em 1971, sendo o TMS1000, da empresa do ramo *Texas Instruments*, o primeiro modelo disponível. O uso de microcontroladores é vital para o setor de automação, uso em ferramentas, veículos, ele-

trodomésticos e, no contexto deste projeto, sistemas embarcados.

O microcontrolador escolhido para o projeto foi o Arduino. Com a sua alta disseminação, facilidade de programação, compatibilidade com módulos extras como sensores e código e hardware aberto, realizar o projeto com o Arduino traz vantagens ideais para o sistema embarcado. O modelo específico de Arduino escolhido foi o Nano, por motivos como tamanho extremamente reduzido comparado ao seu irmão Arduino Uno, baixo custo e possuírem o mesmo processador Atmel Atmega328P, portanto mesma funcionalidade.

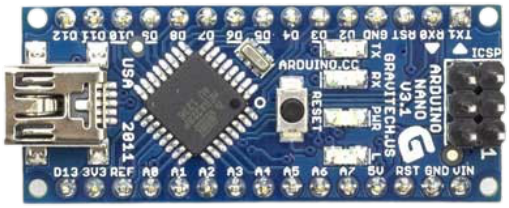


Figura 2.3 – Arduino Nano. (Disponível em https://multilogica-shop.com)

O esquema elétrico dos circuitos presentes no densímetro foram descritos em Barreto (2019) e em Poletti (2021). As conexões do Arduino com os módulos e sensores, além de *leds* indicativos estão apresentados abaixo:

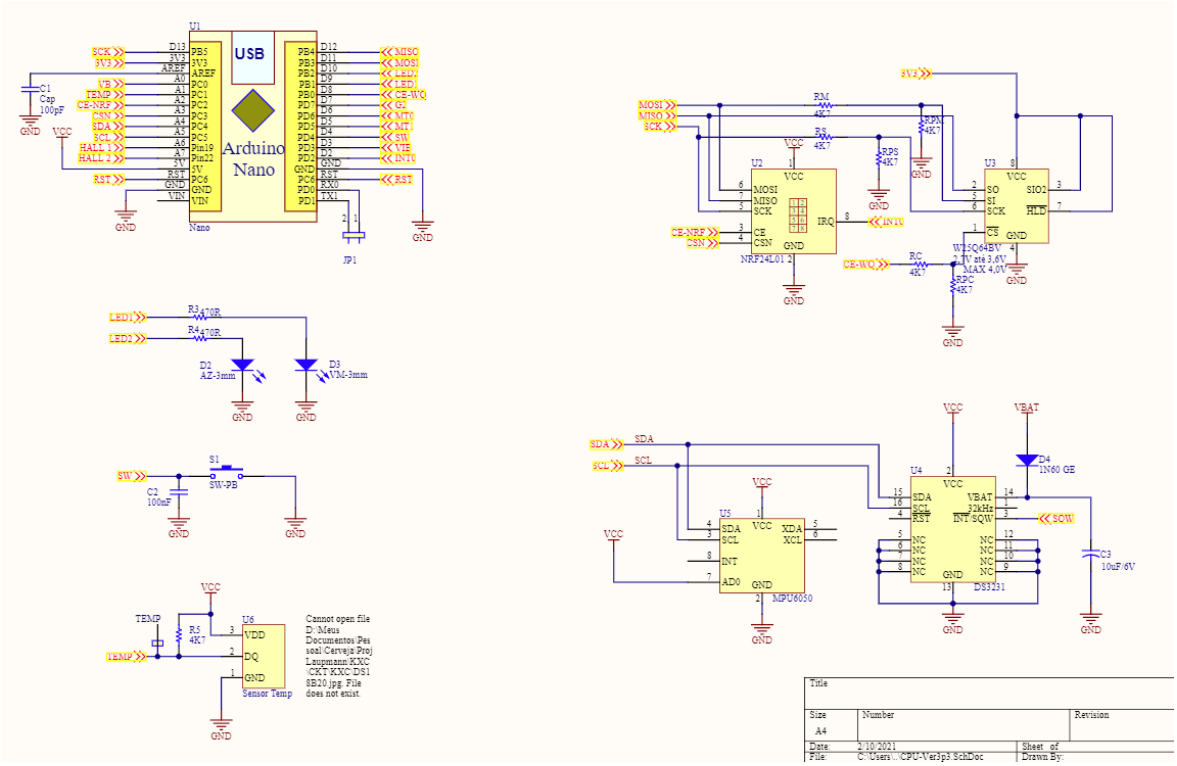


Figura 2.4 – Esquema elétrico do densímetro

O fato do Arduino ser largamente usado no setor de automação aumenta o número de bibliotecas disponíveis para sua linguagem de programação, que é basicamente C/C++, visto que muitas soluções autômatas envolvem as mesmas categorias de módulos e sensores.

Porém, bibliotecas de terceiros não foram usadas neste projeto, visto que os envolvidos em seu desenvolvimento no passado encontraram dificuldades relacionadas a persistência dos dados coletados, controles de funcionalidades importantes como liga/desliga e memória disponível.

As especificações do Arduino Nano, dentre outras, são:

- Memória Flash: 32 KB, sendo 2 KB usados pelo bootloader
- Memória EEPROM: 1KB
- Memória SRAM: 2KB
- Tensão de alimentação (recomendada): 7 – 12V
- *Clock* máximo da CPU: 16 MHz
- Pinos digitais (I/O): 22, sendo 6 PWM
- Pinos analógicos: 8
- Corrente DC por pino (I/O): 40 mA
- Conversor A/D de 10 bits
- Porta Serial síncrona e assíncrona
- Interface I2C
- Interface SPI
- Temporizador: um de 16 bits e dois de 8 bit

Portanto, o desenvolvimento de funções para manipulação da memória flash, acesso de registradores de controles, estabelecimento de protocolos de comunicação como I2C, comunicação via rádio, dentre outras funcionalidades vitais, foi necessário para o controle do Arduino e sua interface com os módulos usados.

Com isso, o desenvolvimento das funcionalidades requeridas ficou consideravelmente mais custoso, mas com a vantagem de se ter total controle do programa, o que de certa forma foi benéfico para depurações de *bugs* e estados de execução dos modos de operação, citados logo mais.

2.3 ACELERÔMETRO

Acelerômetros são dispositivos utilizados para obter valores de aceleração da gravidade nos três eixos X, Y e Z. Com ele, é possível aferir múltiplos fatores como vibrações, choques e taxas de inclinação de um sistema, sendo portanto amplamente usado em estudos e na indústria, e vital para o projeto.

O acelerômetro usado foi o MPU6050, da empresa *Invense*. Consiste, em um único chip, um acelerômetro e um giroscópio tipo MEMS. São 3 eixos para o acelerômetro e 3 eixos para o giroscópio, sendo ao todo 6 graus de liberdade (6DOF). As escalas providas são, respectivamente, +/- 250 gr/s, +/- 500 gr/s, +/- 1000 gr/s e +/- 2000 gr/s (gr/s = graus/segundo) para o giroscópio e +/- 2 g, +/- 4 g, +/- 8 g e +/- 16 g para o acelerômetro.

O chip em questão é encontrado soldado a uma placa modelo GY-521, e também possui seu próprio sensor de temperatura, utilizado para fazer correções de medidas da MPU em razão da temperatura. Porém, retiramos o chip desta placa e soldamos diretamente a PCB, e para a temperatura utilizamos um sensor de alta precisão isolado no circuito, modelo DS18B20 , da *Dallas*. As medidas do giroscópio foram desprezadas, já que o interesse do projeto é somente saber sua inclinação (*tilt*). Dados do giroscópio poderiam ser usados para verificar se o sistema se encontra parado.

Previamente a seu uso, o acelerômetro precisa ser calibrado. Basicamente é realizada uma sucessão de medidas de curto intervalo. Esta calibração é feita em modos de operação do programa de testes, detalhado ao decorrer deste trabalho.

Para obtermos uma variação angular do densímetro, necessidade vital para a correta aferição da densidade do líquido, precisamos ter necessariamente dois vetores. Ou seja, precisamos de uma posição de referência e as sucessivas posições seguintes do densímetro, gerando assim sucessivas variações angulares.

Acelerômetros são dispositivos que medem a aceleração da gravidade nos três eixos, X, Y e Z. Idealmente, um objeto parado está submetido à aceleração da gravidade. Analisando

o acelerômetro MPU6050, presente na figura 2.5, na vertical, seu eixo Z terá o valor da aceleração da gravidade, 9,8 m/s, e os eixos X e Y terão valores zerados.

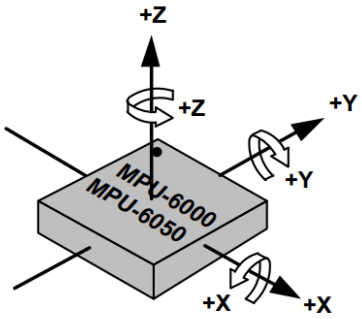


Figura 2.5 – Referências X, Y e Z do MPU-6050 (*Datasheet* MPU-6050)

De acordo com a posição em que o MPU foi soldado à placa, o eixo de maior interesse será o Y, visto que o densímetro na posição vertical posiciona o acelerômetro paralelamente a este eixo, conforme está encaixado no circuito.

Essa referência é composta por três valores para as coordenadas X, Y e Z, inseridas manualmente no código, como será explicado ao decorrer deste trabalho. Com a referência sendo o eixo vertical, garantimos também que possíveis rotações do densímetro não interfiram em sua variação angular, denominada *tilt*.

Enquanto a fermentação ocorre, a densidade do líquido é alterada, e com isso ocorrem variações angulares referentes a sua posição. Obtendo então duas posições distintas, o cálculo do ângulo resultante entre elas é feito utilizando trigonometria.

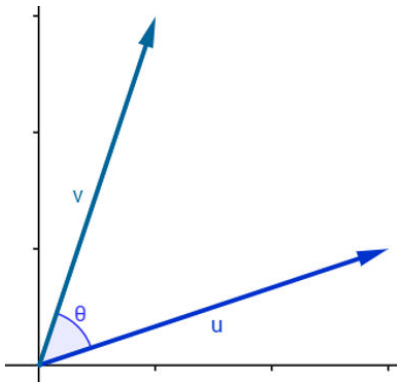


Figura 2.6 – Ângulo θ entre u e v

Considerando u e v, temos que o produto interno entre eles é dado por:

$$< u, v >= |u|.|v|. \cos\theta \tag{2.1}$$

em que $|u|$, $|v|$ representam os módulos dos vetores:

$$|u| = \sqrt{x_u^2 + y_u^2 + z_u^2}, \tag{2.2}$$

$$|v| = \sqrt{x_v^2 + y_v^2 + z_v^2} \tag{2.3}$$

Portanto, o ângulo entre ambos é definidor por:

$$\theta = \arccos\left(\frac{< u, v >}{|u|.|v|}\right).\frac{180}{\pi}, \tag{2.4}$$

em graus.

Para estimarmos a densidade do líquido em que o sistema está submerso, podemos usar esta variação angular obtida com o passar do tempo nas sucessivas medições, de hora em hora. Fatores como peso, volume e centro de gravidade do dispositivo foram adaptados e considerados, como demonstrado em versões anteriores deste projeto (BARRETO, 2019), resultando na seguinte fórmula:

$$\theta = \arctan\sqrt{\frac{1}{\frac{24.CTE.X'_g}{d^2.d_{fluido}} - \frac{12.CTE^2}{d^2.d_{fluido}^2} - 2}}, \tag{2.5}$$

$$CTE = h.d_L + (c - h).d_T, \tag{2.6}$$

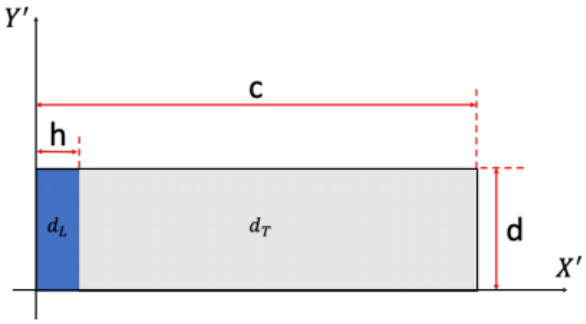


Figura 2.7 – Centros de gravidade d_L (região onde o lastro se encontrava) e d_T (restante do tubo com seus componentes), com seus respectivos comprimentos h e c (BARRETO, 2019)

onde θ é o ângulo de inclinação do densímetro em relação a superfície da água, que obtemos com dados do acelerômetro, e d_{fluido} é a densidade do líquido em que ele está flutuando. Vale ressaltar que esta fórmula necessita de ensaios para sua comprovação, bem como adaptações ao novo sistema de lastros executado no trabalho de Poletti (2021), portanto segue como pontos de melhorias para futuros trabalhos.

Para realizarmos a interface do acelerômetro com o Arduino, utilizaremos comunicação serial I2C. Trata-se de um tipo de transferência de dados entre entidades computacionais por meio de um barramento. Muitas tecnologias do dia a dia utilizam a comunicação serial como a arquitetura Ethernet presente na conexão entre redes e conexões de periféricos via porta USB.

O Arduino provê a interface I2C de comunicação serial, conhecida também como Mestre Escravo. Ela é composta por dois fios, SDA (*Serial Data*) e SCL (*Serial Clock*), tensão V_{dd} de alimentação 3.3V e resistores de *pull up* para garantir que as linhas estejam em nível alto quando não acionadas. A interface trabalha em modo *Half-Duplex*, o que significa que tanto o mestre quando escravos podem receber ou enviar informações ao barramento, em instantes diferentes. Porém, para não haver conflito de uso, o mestre orquestra esta comunicação, estabelecendo as regras da transferência de dados.

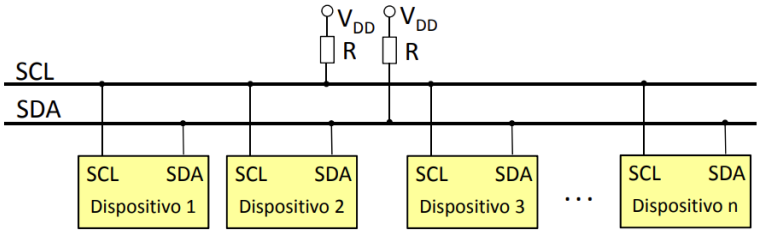


Figura 2.8 – Barramento I2C

A comunicação é iniciada com um sinal de *START* enviado pelo mestre, quando a linha SDA transiciona de alto para baixo com a linha SCL em nível alto, seguido de 7 bits de endereço, usado para determinar qual dispositivo será envolvido na transação, e um bit de controle R/W, dizendo se a operação será de leitura (*Write*) ou escrita (*Read*).

A transação do endereço, assim como toda transação de dados, tem sinalizada seu término por meio do sinal de *STOP*, quando a linha SDA transiciona de nível baixo para alto enquanto a linha SCL está em nível alto. A troca de dados, portanto, ocorre neste padrão de sinal de *START*, seguido dos dados e um sinal *ACK*, que confirma o recebimento por parte do dispositivo endereçado, assim sucessivamente.

Portanto, por meio do acelerômetro, o programa no Arduino capta os valores de aceleração de X, Y e Z e os salva em memória *flash*, para que, futuramente, por meio dos modos de operação, possa transmiti-los via Rádio a um segundo densímetro, denominado "Fofoca", responsável por captar os dados e passa-los ao computador via comunicação serial com a porta USB.

2.4 MEMÓRIAS

Para o armazenamento dos dados, foi implementada uma estrutura de diretórios na memória EEPROM do Arduino e na memória *flash*, cada diretório definido em 16 *bytes* de tamanho para guardar informações importantes do densímetro, conforme listado nas tabelas a seguir. Esta divisão possibilitou o armazenamento de dados referentes a diferentes fermentações, organizando-os de forma a serem facilmente acessados no futuro.

A memória EEPROM do Arduino é utilizada para guardar informações únicas do densímetro, bem como dados de calibração do acelerômetro. Por fim, o resto de espaço é utilizado para o mapeamento dos diretórios. A comunicação com a EEPROM é feita via procedimentos próprios extraídos do manual da CPU, e a memória capacidade de armazenamento de 1KB.

A EEPROM possui uma área reservada de configuração de 64 bytes, segmentada em quatro páginas de 16 bytes cada. Com isso, a área destinada para mapeamento dos diretórios são 60 páginas, totalizando então 60 diretórios, ou seja, até 60 fermentações indexadas na EEPROM.

Tabela 2.1 – EEPROM - Área de Configuração

Nº Registro	**Nome**	**Faixa de bytes**	**Nº de bytes**	**Descrição**
1	EE_IDT	0, 1, 2, 3	4	(ASCII) Identificação única do densímetro
2	EE_VER	4, 5	2	(ASCII) Versão do programa
3	EE_VOLTAR	6, 7	2	(ASCII) Ao ligar, voltar ao último Modo
4	EE_MODO	8, 9	2	(bin) Último Modo de operação
5	EE_PROX_ADR	10, 11, 12, 13	4	(bin) Próximo endereço da Flash
6	EE_TAM_FLASH	14, 15	2	(bin) Tamanho da memória Flash em KB

Para o projeto, é vital que consigamos armazenar os dados coletados durante o processo de fermentação, que normalmente dura algumas semanas. Com isso, utilizamos uma memória *Flash*, modelo W25Q32BV, com 4MB de capacidade. A comunicação é do tipo serial SPI

(*Serial Peripheral Interface*), que também utiliza a arquitetura Mestre Escravo e orientação *full duplex*, em que o Mestre orquestra a conexão.

Figura 2.9 – Memória Flash modelo W25Q32 (Disponível em https://www.magazineluiza.com.br/)

A memória *flash* pode ser alimentada com no máximo 3,6V, porém o V_{cc} do densímetro pode variar até 5V, dependendo da carga da pilha de alimentação. Para adequarmos a tensão na memória quando o Arduino é alimentado via USB, foi aplicado um divisor de tensão e após ensaios, foi observado que a memória funciona bem com tensões baixas até 2,6V.

Esta memória foi segmentada em 64 blocos de 64KB, com registros de 16 bytes de tamanho, totalizando 4096 registros por diretório de fermentação. Isso permite o armazenamento dos dados de cerca de 170 dias de operação por bloco, tomando um registro ocupado a cada hora. A estrutura dos registros de cada bloco, ou página, é demonstrada a diante.

O Registro 0 guarda informações de identificação do densímetro, que são introduzidas por meio de modos de operação do programa de teste. São guardados dados de nome do densímetro, versão utilizada do programa Arduino, escalas do acelerômetro e giroscópio, também definidas por um modo de operação específico, e uma *flag* indicando se o sistema passou pelo *self-test*, modo que realiza a checagem de funções básicas do densímetro.

Tabela 2.2 – Registro 0 - Identificação

Nº Registro	Nome	Faixa de bytes	Nº de bytes	Descrição
1	REG0_CONT	0,1	2	Contador em BCD
2	REG0_IDT	2, 3, 4 ,5	4	(ASCII) Identificação única do densímetro
3	REG0_VER	6, 7	2	(ASCII) Versão do programa (0, 1, ...,99)
4	REG0_ACEL_ESC	8	1	(ASCII) Escala acelerômetro (0,1,2,3)
5	REG0_GIRO_ESC	9	1	(ASCII) Escala Giroscópio (0,1,2,3)
6	REG0_ST	10, 11	2	Flag Self-test (SS/NN)
7	REG0_VAZIO	12, 13, 14, 15	4	Vazio

O Registro 1 é utilizado para parâmetros de Data e Hora do início das fermentações. Este registro é atualizado utilizando valores do relógio do densímetro, um módulo RTC 3231, devidamente detalhado em Poletti (2021).

Tabela 2.3 – Registro 1 - Data e Hora

Nº Registro	Nome	Faixa de bytes	Nº de bytes	Descrição
1	REG1_CONT	0,1	2	Contador em BCD
2	REG1_DATA	2 - 9	8	(ASCII) Data do início da fermentação
3	REG1_HORA	0,1	6	(ASCII) Hora do início da fermentação

O Registro 2 é responsável por guardar os *offsets* resultantes da calibração da MPU6050, obtidas em um modo de operação do programa. São importantes para conferências posteriores às medições realizadas, aumentando sua precisão.

Tabela 2.4 – Registro 2 - Offsets da MPU6050

Nº Registro	Nome	Faixa de bytes	Nº de bytes	Descrição
1	REG2_CONT	0,1	2	Contador em BCD
2	REG2_OFF_AX	2, 3	2	(bin) Offset de ax
3	REG2_OFF_AY	4, 5	2	(bin) Offset de ay
4	REG2_OFF_AZ	6, 7	2	(bin) Offset de az
5	REG2_OFF_GX	8, 9	2	(bin) Offset de gx
6	REG2_OFF_GY	10, 11	2	(bin) Offset de gy
7	REG2_OFF_GZ	12, 13	2	(bin) Offset de gz
8	REG2_VAZIO	14, 15	2	Vazio

O Registro 3 é responsável por guardar informações de calibração do densímetro na água, para usar como referência em análises posteriores. Os dados guardados são as coordenadas X, Y e Z do acelerômetro e giroscópio na posição vertical, em referência a linha d'água e na posição de 25 graus, atingida com o uso do motor com lastro. O resto da memória, até o momento, está sem uso, e foi prevista a opção para se fundir dois diretórios, duplicando a quantidade de dados a ser guardada.

Tabela 2.5 – Registro 3 - Calibração na água

Nº Registro	Nome	Faixa de bytes	Nº de bytes	Descrição
1	REG3_CONT	0,1	2	Contador em BCD
2	REG3_AX_VERT	2, 3	2	(bin) ax com KXC na vertical
3	REG3_AY_VERT	4, 5	2	(bin) ay com KXC na vertical
4	REG3_AZ_VERT	6, 7	2	(bin) az com KXC na vertical
5	REG3_AY_CAL	8, 9	2	(bin) ax com KXC Calibrado (25 graus)
6	REG3_AY_CAL	10, 11	2	(bin) ay com KXC Calibrado (25 graus)
7	REG3_AZ_CAL	12, 13	2	(bin) az com KXC Calibrado (25 graus)
8	REG3_VAZIO	14, 15	2	Vazio

Do Registro 4 em diante, são guardados os dados extraídos a cada medição do densímetro. Neste modelo, temos dados do acelerômetro nos eixos X, Y e Z, bem como a temperatura

em graus Celsius com o último dígito representando sua parte decimal, tensão da bateria em centésimos de Volts e grau de agitação.

Tabela 2.6 – Registro 4 - Dados de fermentação

Nº Registro	Nome	Faixa de bytes	Nº de bytes	Descrição
1	REG_CONT	0,1	2	Contador em BCD
2	REG_AX	2,3	2	(bin) ax
3	REG_AY	4,5	2	(bin) ay
4	REG_AZ	6,7	2	(bin) az
5	REG_TP	8,9	2	(bin) Temperatura
6	REG_BT	10,11	2	(bin) Tensão da bateria
7	REG_AG	12	1	(bin) Agito
8	REG_VAZIO	13,14,15	3	Vazio

2.5 TERMÔMETRO

Como descrito no início do projeto, a temperatura do líquido durante seu processo de fermentação é determinante, sendo importante, portanto, o seu monitoramento. Para isso, o densímetro conta com um sensor de temperatura de alta precisão, modelo DS18B20. Este termômetro provê a temperatura interna do densímetro, que provavelmente está próxima da temperatura do líquido externo, durante o processo de fermentação. Também conta com divisor resistivo, com resistores de 1% de precisão.

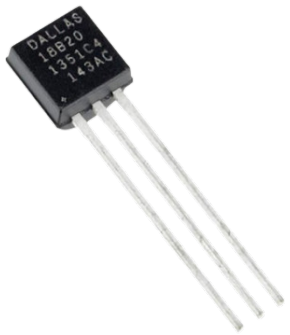


Figura 2.10 – Termômetro DS18B20

A comunicação do Arduino com o termômetro é feita via protocolo 1-Wire. O protocolo 1-Wire se assemelha com o I2C, visto que ambas são da categoria de comunicação Mestre Escravo, em que o dispositivo Mestre dita as regras de transações de dados entre o barramento.

Como o nome sugere, a arquitetura do 1-Wire é composta por um único fio para transferência de dados e um referência terra comum para todos envolvidos na comunicação. O dispositivo que deseja iniciar a comunicação sinaliza mudando a linha para baixo, ao acionar seu transistor de saída. Já quando os dispositivos estão inativos, a linha está em nível alto, e neste estado é possível que os dispositivos continuem monitorando o barramento.

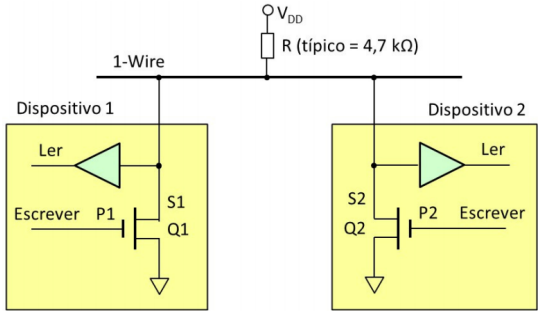


Figura 2.11 – Conexão via 1-Wire (Zelenovsky, 2017)

O termômetro DS18B20 está conectado na porta analógica A1 do Arduino por não haver outras portas digitais disponíveis, porém utilizamos este pino de forma digital sem necessidade de adaptações. Para obtermos a temperatura, consultamos o registrador de temperatura do DS18B20, que possui dois *bytes*, e está representado em sistema binário.

2.6 BATERIA

Por se tratar de um sistema autônomo, a alimentação do densímetro é feita por duas formas: através dos 5V da porta USB ou através de uma bateria, quando está em operação. Em sua versão anterior, o densímetro era alimentado por quatro pilhas do tipo AAA, porém neste projeto, a alimentação também foi trocada. Para o funcionamento do dispositivo, utilizamos uma pilha recarregável modelo 18650, com capacidade de 3800mAh e aproximadamente 4,5V de tensão quando totalmente carregada.

Na figura 2.12, temos o circuito de alimentação do densímetro. Seu projeto e detalhes estão melhores descritos em Poletti (2021). Basicamente, foi projetado um circuito para implementação do modo liga/desliga, bem como se adapte seguramente tanto quando ligado via USB, quanto pela bateria. Componentes utilizados foram diodos, transistores tipo N e P, botões e *switches*.

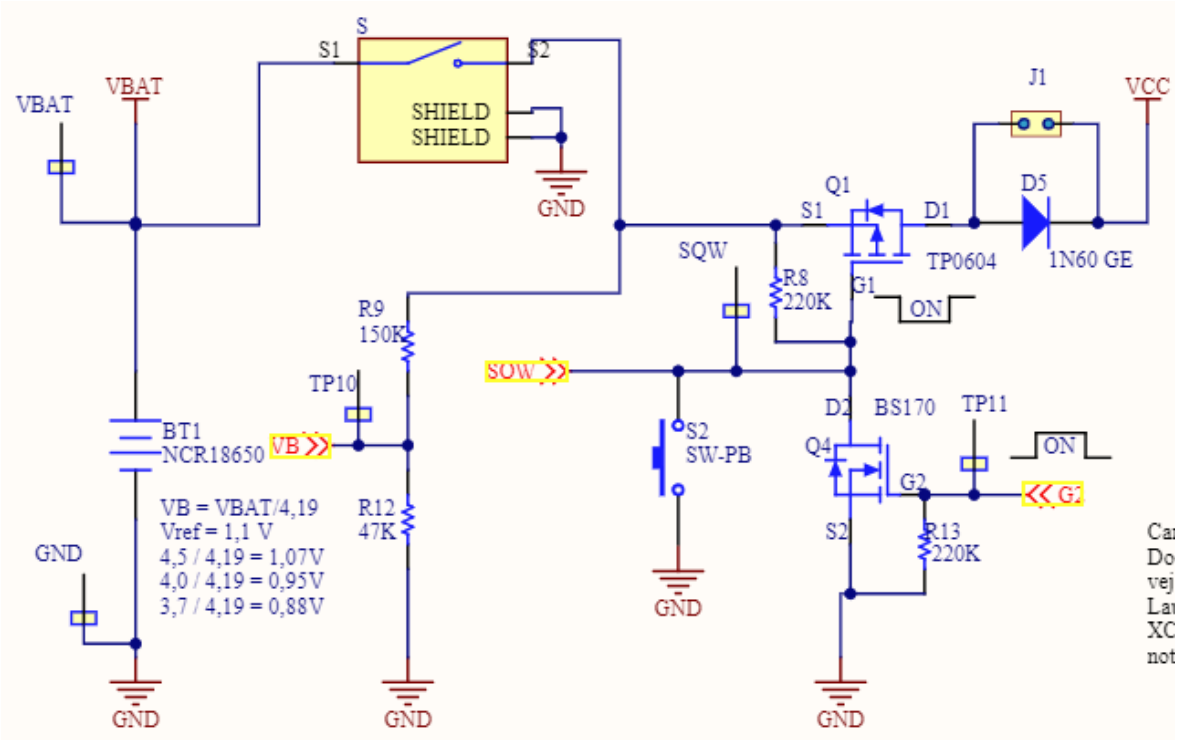
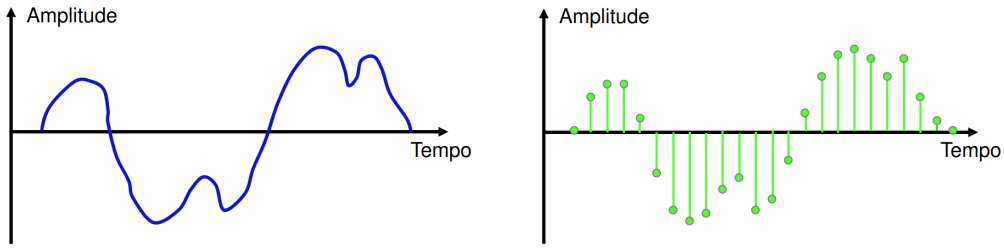


Figura 2.12 – Circuito de alimentação

Um dado importante de ser armazenado durante o funcionamento do densímetro é a tensão da bateria, pois assim conseguimos estimar a duração de tempo funcional do sistema, bem como perceber algumas inconsistências nos dados em razão de má alimentação. Os sinais disponíveis da bateria são de natureza analógica, então necessitamos de uma conversão analógico para digital para conseguirmos interpretar corretamente seus dados de tensão.

Sinais analógicos são contínuos no tempo, e portanto não são interpretáveis neste formato por um processador digital, como os de microcontroladores. Portanto, é necessária a discretização e quantização deste sinal, diminuindo assim a faixa de frequência em que o sinal trafega.



(a) Sinal Analógico (b) Sinal Discretizado

Figura 2.13 – Discretização de um sinal analógico (Zelenovsky, 2017)

A partir da discretização do sinal analógico, podemos então quantizar os pontos obtidos para reconstruir o sinal em sua forma digital. Percebe-se que quanto maior a taxa de

amostragem, ou seja, quanto mais amostras forem feitas em um determinado período, mais fidedigno será o sinal em sua forma digital.

Três elementos são monitorados via conversão A/D (ADC): divisor resistivo, indicando a tensão da bateria, e dois sensores Hall do lastro móvel. Para monitorar a tensão da bateria, é necessário usar a referência interna da CPU do Arduino, de 1,1 V. De acordo com a entrada a ser monitorada, é preciso também alterar a referência do ADC.

Foi utilizada uma interrupção a cada 20ms, ou 50Hz, realizada pelo *timer*, que alterna entre as diferentes entradas analógicas e garante a temporização exigida pelo ADC.

Na primeira interrupção, ocorre a configuração do primeiro sensor Hall. Durante as próximas seis interrupções são adquiridos dados da porta e calculada a média dos quatro últimos valores obtidos, tendo sempre as duas primeiras medidas desprezadas para maior precisão nas medidas, pois o Arduino necessita de um tempo para a mudança de sua referência interna. O programa constantemente checa a variável global para saber qual próximo dispositivo analógico a ser monitorado.

Seguindo, na sétima interrupção é configurada a porta do segundo sensor Hall, e novamente calculada a média. Por fim, na interrupção nº 14, é configurada a referência de 1,1 V para medir a tensão do divisor resistivo da bateria, coletados seus dados e feito a média. Ao chegar na interrupção de nº 20, temos neste momento outras quatro interrupções, sem atribuições definidas no momento, e o contador é zerado, fazendo o ciclo de medições dos dispositivos analógicos recomeçar.

A persistência desses dados é feito em variáveis globais do programa, que são atualizadas portanto com essas interrupções descritas, podendo ser consultadas a qualquer momento.

Figura 2.14 – Ciclo de monitoramento analógico

2.7 VIBRADOR

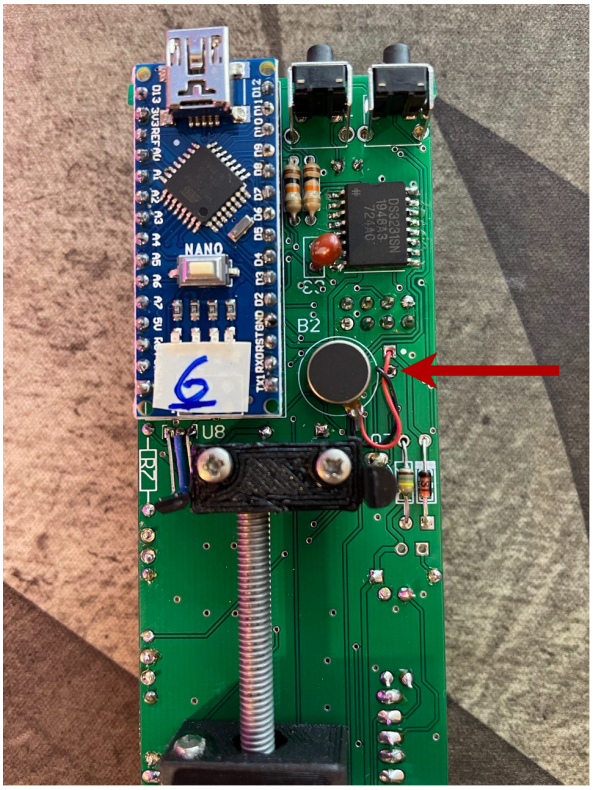


Figura 2.15 – Módulo vibrador

O dispositivo vibrador foi uma melhoria acrescentada neste projeto. Em versões passadas, por meio de ensaios foi possível perceber o acúmulo de bolhas no corpo do densímetro. Podemos considerar a hipótese de que, dependendo do volume de bolhas, a flutuabilidade do densímetro é impactada, e, consequentemente, as medições de grau de inclinação serão imprecisas.

Para contornar esta variável ambiental das medições, foi implementado um módulo vibrador colado no circuito impresso do densímetro. O objetivo é que, previamente a cada medição realizada pelo densímetro, o vibrador seja acionado por alguns segundos para diminuir a quantidade de bolhas coladas em seu corpo.

Foram realizados ensaios que não demonstraram eficácia deste vibrador quando flutuando em refrigerante **(link)**, porém sugere-se novos ensaios com bolhas da fabricação de cerveja.

2.8 OUTROS COMPONENTES

Como mencionado, um projeto paralelo foi realizado que faz diversas melhorias e introduz novas funcionalidades a este densímetro. Foi implementada uma melhoria para a estabilidade do dispositivo enquanto estiver flutuando na cerveja. Em suas versões iniciais, foram usados contrapesos de chumbo colados em diversos pontos do corpo do densímetro. De fato foi alcançada uma estabilidade superior com esta solução, mas logo viu-se a necessidade de poder fazer ajustes rápidos com a constante evolução do dispositivo.

Com isso, foi acoplado ao densímetro um sistema de lastro. Um motor com um bloco de chumbo colado a ele pode trafegar em um eixo de aço. O objetivo do lastro é alterar o centro de gravidade do sistema, podendo se mover conforme programado e dando assim mais controle de posicionamento e calibração. O lastro utiliza dois sensores do tipo Hall para a delimitação do início e fim do eixo trafegável.

Outro componente acoplado ao densímetro foi um relógio digital, modelo RTC 3231, vital para a eficiência energética do sistema. Com o relógio, foi possível disparar alarmes, por meio de interrupção, para ligar o densímetro. Ele pode ser configurado utilizando comunicação serial I2C, e pode ter hora, datas e múltiplos alarmes definidos pelo usuário.

Também no quesito eficiência energética, em suas versões anteriores, o densímetro era colocado em seu modo *low power*, em que é reduzido o *clock* do Arduino. Porém, para estender ainda mais a duração da bateria, foi implementada uma rotina de ligar e desligar o sistema, garantindo que ele funcione somente poucos minutos em cada aferição, sendo depois desligado até o alarme do relógio acordá-lo novamente.

A comunicação com o módulo exterior ao local de fermentação é feita via rádio, modelo NRF24L01+, da empresa *Nordic*. O protocolo de comunicação é o SPI e envia os dados requisitados em pacotes, no momento, definidos em 32 bytes de tamanho. Caso o pacote seja menor que 32 bytes, o resto do espaço é preenchido com zeros para totalizar o pacote.

2.9 MODOS DE TESTE

O programa desenvolvido para o densímetro consiste em uma evolução de uma série de testes dos módulos acrescentados e já existentes, e suas respectivas configurações ao Arduino. O objetivo, em evoluções futuras, é que crie-se um programa separado para usá-lo em ambiente real de fermentação, quando o projeto amadurecer e consolidar suas funcionalidades com base em ensaios e otimizações.

Os modos foram desenvolvidos e são executados por meio da IDE (*Integrated Development Environment*, Ambiente de Desenvolvimento Integrado) do Arduino, o Arduino IDE.

Ela possui a funcionalidade de Monitor Serial, que permite selecionar uma porta serial com conexão ativa no computador, bastando então selecionar o endereço referente ao densímetro e abrir o monitor.

Com o monitor serial aberto, o Arduino detecta o início da conexão serial e envia um texto perguntando qual número de modo deseja ser executado. Daí, basta digitar o número correspondente no campo de entrada de texto do monitor com o código da operação desejada.

Na tabela 2.7, temos listados os modos de operação habilitados nesta versão do densímetro. Tratam-se de modos de testes de persistência de dados, de funcionalidades e configurações. Para as evoluções realizadas neste projeto, os modos 16, 17, 18 e 19 foram os mais importantes, e portanto serão explanados a diante.

Tabela 2.7 – Modos de operação

Modo	**Descrição**	**Modo**	**Descrição**
1	Teste Leds	12	Rádio, Tx contínuo
2	Teste chave SW	13	Rádio, Rx contínuo
3	Teste DS18B0	14	Rádio, Fofoca
4	Teste RTC	15	Rádio, Xereta
5	Liga/Desliga a cada minuto	16	Ensaio de uso
6	Teste MPU	17	Configuração
7	Teste Mem. Flash W25Q32	18	Teste posição vertical
8	Teste EPROM	19	Teste posição 25 graus
9	Teste Vibrador	20	Teste de Funcionalidade Lastro
10	Teste Motor do Lastro	21	Acompanhar Fermentação
11	Ver. Carga da Bateria		

2.9.1 Modo 16

O modo 16 foi desenvolvido para simular o funcionamento do densímetro por completo, analisando dados do acelerômetro, temperatura e tensão da bateria. Os dados lidos são então transmitidos a cada segundo via comunicação serial e mostrado no monitor, conforme a figura 2.16. O formato de saída foi decidido entre os envolvidos no processo, sendo composto por informações do modo escolhido e diretório correspondente ao ensaio.

```
==> Modo?
16) Ensaio
Criar Dir = 4
Erase_64K = 00040000
01234567891011121314
Pronto
0004 -02270 +00462 +13752 +00250 +00000 +085
0005 -02258 +00458 +13786 +00250 +00000 +085
0006 -02404 +00484 +13716 +00250 +00000 +085
0007 -01694 -14200 +02798 +00250 +00000 +085
0008 -06140 -08490 -13620 +00251 +00000 +085
0009 -12538 -00578 +06854 +00251 +00000 +085
0010 +06956 +01346 +13358 +00251 +00000 +085
0011 +16014 +06686 +01612 +00251 +00000 +085
```

Figura 2.16 – Monitor serial em Modo 16

Os parâmetros das medições começam a partir do registro "0004", representando o registro 4 da tabela 2.6, indicado pelo primeiro elemento de cada linha, pois os registros 0, 1, 2 e 3 representam a configuração do densímetro, conforme elucidado nas tabelas 2.2, 2.3, 2.4 e 2.5.

Em seguida, vem três parâmetros, com sinal, que correspondem as medidas dos eixos X, Y e Z do acelerômetro. O quinto elemento é a temperatura do módulo de termômetro, em ºC, sendo o último dígito a parte decimal da medida. O sexto elemento é a tensão da bateria, que não estava conectada neste ensaio, seguido do grau de agitação.

2.9.2 Modo 17

O modo 17 foi criado para a configuração do densímetro. Ao entrar no modo, será esperado o comando específico da configuração desejada, conforme a tabela 2.8. Como podemos ver pelo primeiro comando, após executar qualquer uma das instruções, o usuário pode voltar a qualquer momento ao menu anterior para escolher outros modos. As instruções possuem tratamento de letras maiúsculas e minúsculas, sendo qualquer um dos dois corretamente interpretados pelo programa.

Tabela 2.8 – Modo 17

Comando	Ação
X	Retorna ao menu anterior
C	Interpreta e imprime toda configuração
D	Imprime EEPROM em Hexadecimal
L	Lista diretórios ocupados
P + 'num'	Imprime dados do diretório especificado
N + 'nome'	Definir nome do densímetro. Trunca em 4 letras
F	Registra tamanho da memória *flash*
V + 'ver'	Registra versão do código. Trunca em 2 números
T	Realiza *Self-Test*
O	Cálculo dos *offsets* do acelerômetro a partir da vertical
E + 'dir'	Deleta o diretório especificado. 'E + 99' apaga toda memória

2.9.3 Modo 18

O modo 18 foi elaborado para registrar uma referência vertical. Esta referência é obtida colocando o densímetro totalmente vertical, paralelo ao eixo da gravidade.

Ao executar esse modo, o programa do Arduino faz a média aritmética de 256 medidas para os eixos X, Y e Z do acelerômetro e giroscópio, e grava elas em memória, de acordo com as estruturas de diretório da tabela 2.5.

2.9.4 Modo 19

O modo 19 utiliza da referência vertical gravada no modo 18 para posicionar o densímetro a 25 graus desta referência. Este ângulo foi escolhido baseado nos ensaios de Barreto (2019), que mostraram bons resultados de variação angular de fermentações com o densímetro nesta posição. Para isso ele move a mesa do lastro, alterando seu centro de gravidade e, portanto, sua inclinação. Este sistema é melhor detalhado em um trabalho paralelo a este.

Vale ressaltar que o ângulo referência ideal pode mudar para a versão do densímetro utilizada neste projeto, devido as sucessivas mudanças estruturais listadas, recomendando-se então novas investigações.

Também é possível enviar a letra 'a' seguido de um número representando o ângulo em que se deseja manter o densímetro de sua referência, fazendo o lastro movimentar de acordo com o objetivo.

3 DETALHAMENTO: *DASHBOARD*

3.1 MOTIVAÇÃO E FUNCIONAMENTO

O projeto deste densímetro passou por diversas evoluções e ajustes. Com este trabalho, foram realizadas diversas refatorações de código do programa teste, bem como incorporadas funcionalidades como lastro programável e relógio. Porém, acredito que a evolução prática mais impactante para o futuro do projeto seja a implementação do KXC *Dashboard*, feita neste projeto.

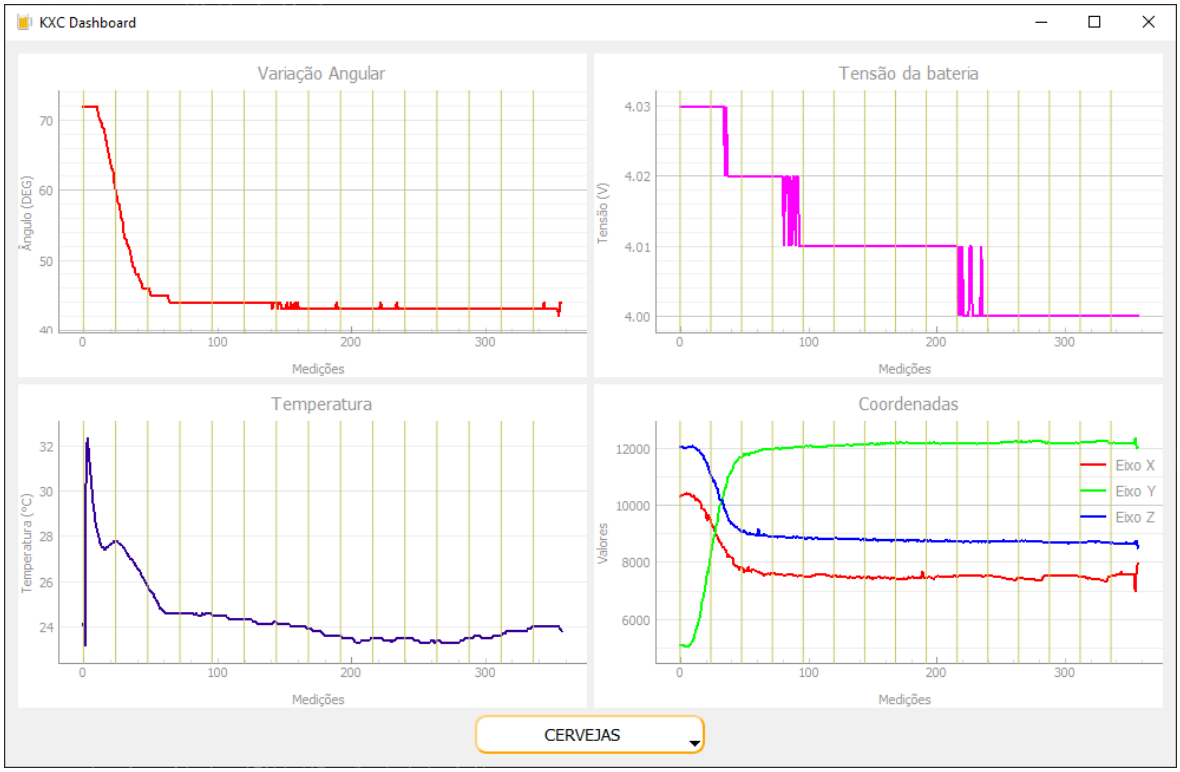


Figura 3.1 – KXC *Dashboard*

Em suas versões anteriores, o densímetro sempre dependia da Arduino IDE para ser executado, já que assim temos acesso ao monitor serial e conseguimos interagir com o programa de testes. Neste projeto, foi implementada um programa com interface gráfica que possibilita uma interação muito mais natural e menos exigente tecnicamente do usuário.

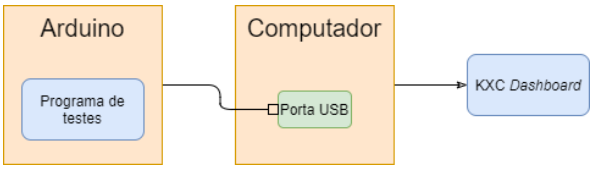


Figura 3.2 – Interface Arduino e KXC *Dashboard*

A partir dos diretórios preenchidos com fermentações na memória *flash* do densímetro, funcionalidade explicada no capítulo anterior, o KXC *Dashboard* permite transferir esses diretórios para um arquivo texto, a título de documentação.

Em seguida, o usuário pode visualizar os dados mais importantes da fermentação, de forma mais amigável e compreensível por meio de gráficos interativos, em forma contraposta ao que tínhamos na figura 2.16. O *dashboard* também é uma grande evolução no sentido do densímetro atingir seu público alvo no futuro: fabricantes de cerveja.

O sistema é composto por quatro gráficos, contendo variações angulares, de temperatura, de tensão de bateria e das coordenadas do acelerômetro, e um botão do tipo menu, que quando clicado, lista todos os diretórios ocupados do Arduino, com as respectivas data e hora de criação, em caso de poder diferenciar as fermentações com múltiplos diretórios de mesmo nome. Ao clicar no botão e em seguida no diretório desejado, o *dashboard* carrega as respectivas informações.

Figura 3.3 – Botão Menu

3.2 TECNOLOGIAS

Como esta é a primeira versão do KXC *Dashboard*, foi necessário decidir quais tecnologias seriam usadas para desenvolver este software. Baseando em linguagens de programação de alto nível, que possibilitam o rápido desenvolvimento e evolução com relativamente pouco volume de código, a linguagem escolhida que também atenderia os requisitos do projeto foi Python, versão 3.9.

Python é uma linguagem de alto nível interpretada, isto é, possui um interpretador que implementa uma máquina virtual, cuja "linguagem de máquina" é a linguagem de programa-

ção de alto nível. O interpretador lê as declarações nessa linguagem mais ou menos um de cada vez, executando-as à medida que avança (SCOTT, 2017).

Isso possibilita certas flexibilizações ao programador, o que aumenta o rendimento do código e melhora sua depuração, mas tira certo controle sobre o programa. Contudo, analisando o contexto do *dashboard*, este fator alto nível é importante para futuras evoluções do projeto, e não precisamos ter o nível de controle de memória, por exemplo, que teríamos com linguagens como C, de baixo nível.

Linguagens de baixo nível são linguagens compiladas, em que o código escrito pelo programador é diretamente transformado para linguagem de máquina que o sistema operacional em questão carrega em memória para execução direta pelo processador (SCOTT, 2017). São extremamente rápidas e complexas, indicadas para sistemas embarcados, em que cada recurso disponível precisa ser utilizado da forma mais eficiente. O programa sendo executado no Arduino, em linguagem C++ adaptada, faz uso destes controles de recursos, mas não há consenso na sua comunidade sobre sua classificação entre linguagem de baixo e médio nível.

A escolha do Python também foi determinada pela quantidade de bibliotecas disponíveis e estáveis. Bibliotecas, em linguagens de programação, são conjuntos de instruções elaboradas para atingir funcionalidades específicas dentro de um software.

O KXC *Dashboard* utiliza, dentre outras, duas bibliotecas principais para atingir suas funcionalidades: *pySerial* e *PyQt5*. A primeira é uma biblioteca oficial do Python, e foi utilizada para habilitar a comunicação serial com dispositivos conectados nas portas USB do computador, provendo métodos importantes como ler e escrever na porta serial.

A biblioteca *PyQt5* foi utilizada para desenvolver a interface gráfica. Trata-se de uma adaptação para linguagem Python do Qt, um *framework* de desenvolvimento de interfaces gráficas para a linguagem C++ amplamente utilizado no mercado, pertencido a empresa Trolltech.

Com aplicações em sistemas embarcados de grandes empresas como Tesla e Mercedes-Benz, o Qt possui alta performance e confiabilidade, e também possui uma documentação bastante completa, o que possibilitou o rápido avanço das funcionalidades previstas neste trabalho. Portanto, a escolha do *PyQt5* foi bem embasada.

O programa do KXC *Dashboard* é dividido em dois arquivos Python: um para realizar a comunicação serial e tratamento dos dados que chegam do densímetro (*engine.py*), outro contendo programação e estilização da interface gráfica (*dashboard.py*). O primeiro será referido como "*engine*", o segundo como "*dashboard*" ao decorrer deste trabalho.

3.3 COMUNICAÇÃO SERIAL

Para o *dashboard* se comunicar com o densímetro, o usuário precisa conectar o Arduino no computador, em qualquer porta do tipo USB. Quando analisamos o Arduino IDE, para estabelecer a conexão serial o usuário precisa ter conhecimento do nome interno da porta USB em que foi conectado (ex: COM1, COM2, COM3, ...).

Como uma melhoria a este procedimento, visto que possivelmente o usuário do densímetro não terá conhecimentos sobre endereçamentos de barramento USB, foi elaborada uma rotina no programa do *dashboard* que realiza a pesquisa em todas as portas com conexão ativa, analisando o campo *"description"* de cada uma dessas portas. O batimento para descobrir a porta correta foi pelo nome do gerenciador de conexão USB do Arduino utilizado: USB-SERIAL CH340.

```
COM1 - Porta de comunicação (COM1)
COM4 - USB-SERIAL CH340 (COM4)
Arduino encontrado!
```

Figura 3.4 – Listagem de conexões USB

Esta rotina é executada assim que o *dashboard* é iniciado, garantindo assim que o programa não prossiga caso não tenha conectado o densímetro ao computador. Neste caso, é enviado ao console "Arduino não encontrado!", e o programa fecha. A execução do *dashboard* também instancia a classe *SerialIO*, referente ao arquivo *engine*. As definições de classe e instância serão abordados com detalhes a seguir.

Com a conexão estabelecida, foi notada a necessidade de colocar o programa em *hold* por dois segundos, visto que o Arduino reinicia toda vez que estabelece uma conexão serial. Este fator pode ser notado também quando utilizamos o Monitor Serial da Arduino IDE: o monitor fica em estado bloqueado por dois segundos, podendo então, depois deste intervalo, se comunicar com o Arduino via serial. Para isso, foi utlizado o método *sleep*, da biblioteca nativa *time* do Python.

3.4 CAPTAÇÃO E TRATAMENTO DOS DADOS

Com a conexão estabelecida entre o densímetro e o KXC *Dashboard*, a *engine*, que tem sua classe instanciada quando executamos o *dashboard*, executa em seu construtor um método para extrair todos os diretórios ocupados do densímetro, contendo as fermentações feitas.

Para isso, ao relembrarmos os modos de operação do programa do Arduino, listados na

tabela 2.7, foi necessário enviar via serial o número 17, referente ao Modo 17, entrando então no menu de configurações, vide tabela 2.8.

De acordo com a tabela, enviamos então o caractere "L", modo que lista todos os diretórios de fermentação ocupados do densímetro. Ao receber estes dados, o programa dinamicamente persiste os dados em um arquivo local *directories.txt*. De fato, é realizado um *dump* de memória. Cada linha do arquivo é composta pelo número do diretório, seguido de seu nome, data de criação e hora de criação.

Tendo então os dados de diretórios persistidos, é executado um método elaborado de voltar ao menu anterior, bastando enviar "X" na porta serial, conforme listado na tabela 2.8. Em seguida, com o programa portanto esperando o modo desejado, a interface gráfica é executada, e uma janela é exposta ao usuário com um botão menu da figura 3.3, montado dinamicamente por um método no módulo de interface.

Para cada opção dentro do botão menu, é atribuída a mesma chamada de função, porém com o parâmetro variado de acordo com o nome do diretório selecionado. Este parâmetro é uma classe, denominada *Beer*, que, em linguagens orientadas a objeto como Python pode ser, representa uma entidade com um ou mais atributos e/ou métodos.

```
1 class Beer(object):
2     index = 0
3     name = ''
4 
5     def __init__(self, index, name):
6         self.index = index
7         self.name = name
```

Listing 1 – Classe *Beer*

Os atributos escolhidos para a classe *Beer* são *index*, que representa o índice do diretório, obtido do primeiro elemento de cada linha do *directories.txt*, e *name*, que representa o nome do diretório, bem como seus metadados de data e hora de criação.

Para instanciarmos, isto é, criarmos um objeto com valores específicos, a classe *Beer*, nota-se o uso do método reservado `__init__`. No paradigma orientado a objeto, esse método se chama construtor, e é responsável por fazer as atribuições dos parâmetros que são inseridos no momento da instanciação da classe, bem como executar algum método. O mesmo ocorre com a instanciação da classe *SerialIO*, que ocorre quando executamos o *dashboard*.

```
1 class Dashboard(QWidget):
2     def __init__(self, *args, **kwargs):
3         super(Dashboard, self).__init__(*args, **kwargs)
```

```
4           self.engine = SerialIO()
5           self.layout_init()
```

Listing 2 – Classe *Dashboard* e seu construtor, instanciando a Classe *SerialIO*

É portanto desta forma que, quando executamos o *Dashboard*, instanciamos não só sua classe, mas também a classe *SerialIO* (linha 4), referente a *engine* do KXC *dashboard*. Em seu construtor está a configuração da conexão serial, explicada anteriormente, sendo então a primeira ação do *Dashboard* quando executado.

```
1  class SerialIO():
2      def __init__(self):
3          self.setup()
4          self.list_dir_mode()
5
6      def setup(self):
7          ports = list(serial.tools.list_ports.comports())
8          self.arduino_data = None
9          # Procura pelo Arduino conectado:
10         for p in ports:
11             if 'USB-SERIAL CH340' in p.description:
12                 try:
13                     self.arduino_data = serial.Serial(p.name, 9600, timeout=1)
14                     print('Arduino encontrado!')
15                 except Exception:
16                     print('Porta Serial Arduino ocupada!')
17                     exit()
```

Listing 3 – Classe *SerialIO* e seu construtor, executando o método *setup* e *list_dir_mode*, que persiste os diretórios

Assim, ao construir o botão menu, a *engine* também vai instanciando a classe *Beer*, atribuindo os respectivos dados para cada diretório, e assim associando dinamicamente à mesma função as instâncias de *Beer* montadas para cada opção do botão menu. Esta associação foi feita utilizando expressões *lambda*, conceito poderoso presente em muitas linguagens de programação, que permite a criação de funções e parâmetros dinamicamente sem a necessidade de declará-los de forma convencional.

Com a implementação da expressão *lambda* nas atribuições dos diferentes parâmetros para a mesma função, o KXC *Dashboard* se torna significativamente mais robusto e escalável. Isso ocorre pois não é necessário saber previamente quantos diretórios existem no densímetro para realizar as respectivas atribuições.

Quando o usuário seleciona a opção desejada no botão menu, o *dashboard* executa a função, que é única, mas com a respectiva instância *Beer* como parâmetro, para entrar nova-

mente no modo 17. Esta função pertence a *engine*, sendo acessada por meio da instância da classe *SerialIO*.

Para coletarmos os dados da fermentação selecionada, esta função envia o caractere "P" no barramento serial, seguido de um espaço e o campo *index* da respectiva *Beer*. Com isso, o densímetro envia ao *dashboard* os dados da fermentação. Eles estão salvos em memória no mesmo formato que aparece no monitor serial no modo 16 de operação, conforme a figura 2.16.

Como o densímetro nos envia não somente os dados do diretório especificado, mas também seus metadados, foi necessária a implementação, no programa do Arduino, de caracteres especiais agindo como delimitadores. Estes delimitadores permitem que o programa os identifique e capte somente os dados relativos as medições, para suas corretas interpretações e manipulações.

Em mais uma etapa de comunicação serial foi decidido persistir os dados, para assim termos um *dump* espelho para depuração de erros encontrados. Os dados são lidos do *buffer* serial e salvos dinamicamente em um arquivo texto com nome da propriedade *name* da instância *Beer* escolhida.

Após a persistência dos dados, o *dashboard* executa outro método da *engine*, desta vez para ler do arquivo os dados da fermentação. Esse método interpreta cada linha de medida conforme foi detalhado anteriormente com a figura 2.16, sendo uma sequencia de sete valores: número do registro, leituras dos eixos do acelerômetro, temperatura, tensão da bateria. O grau de agitação foi desprezado por necessidade de aprimoramentos futuros em seu cálculo.

Esta interpretação foi a chamada de métodos específicos para extrair cada dado. Os métodos filtram cada linha fazendo as respectivas atribuições pelo seu índice, sendo cada índice um dado entre espaços. Cada dado é concatenado à sua respectiva lista, tendo ao final uma lista de dados para cada atributo, totalizando quatro listas.

O retorno deste método da *engine* é no formato de tupla. Trata-se de uma estrutura de dados de dois ou mais elementos formando uma lista imutável. O *dashboard* então cria quatro listas vazias, e atribui a elas o resultado desta chamada. Com isso, o módulo de interface possui a lista separada de cada parâmetro das medições realizadas.

3.5 CÁLCULOS

Especificamente sobre a lista de ângulos, a variação angular de cada medida precisa ser calculada para de fato obtermos esta lista. Durante o desenvolvimento do projeto, inicialmente este cálculo era feito dinamicamente, durante a leitura do *buffer* serial, para persistir em arquivo o ângulo e não as coordenadas do acelerômetro.

Contudo, após discussões e possíveis necessidades de avaliar os dados vindos do acelerômetro, optou-se por salvá-los cru, e seu cálculo ser realizado a posteriori. Esta decisão viabilizou a construção do quarto gráfico, que corresponde às coordenadas X, Y e Z do acelerômetro.

A variação angular é calculada utilizando a equação 2.4. Quanto ao ângulo de referência, foram feitos diversos ensaios para analisar a aceleração da gravidade em nossa localidade, Brasília/DF. Os ensaios consistiram em deixar o densímetro em diversas orientações e analisar os valores resultantes de aceleração em cada eixo. Por fim, a referência foi tomada como o cálculo do módulo destes valores, tendo assim o valor mais próximo da realidade.

O valor do módulo da aceleração da gravidade atingido foi de aproximadamente 16704, considerando que o acelerômetro está configurado para escala +/-2g. Aplicando esta escala, obtemos aproximadamente 1,02g de aceleração da gravidade no local dos ensaios, e considerando que foram realizados em Brasília/DF, observamos uma leve defasagem da medida real, que é < 1g, devido a altitude da cidade.

Porém, considerando que esta defasagem da referência é propagada também para as medições, provavelmente não teremos discrepâncias significativas da realidade com os valores de variação angular, medida mais importante para o densímetro. Vale ressaltar que essa observação necessita de mais ensaios para ser comprovada.

Outra forma possível de se tomar o valor da referência vertical é colocar o densímetro nesta posição e realizar sucessivas medições, gravando a que tiver maior valor no eixo vertical, que para o densímetro é o eixo Y. Ensaios posteriores devem demonstrar qual método é o mais preciso.

Portanto, o vetor referência é definido em código como $V_{ref} = (0, 16704, 0)$. Para fazer os cálculos matemáticos, foi utilizada a biblioteca nativa *math*.

3.6 INTERFACE GRÁFICA

O *PyQt5* provê diversas formas de construir interfaces gráficas. Neste projeto, utilizamos o conceito de *Widgets*, representado pela classe de mesmo nome. Desta classe, utilizamos diversos métodos para construir a moldura da janela da interface, botões e *layouts* de grade para ajustarmos os posicionamentos.

O interpretador do Python possui uma variável reservada chamada `__name__`, que representa o nome do módulo. Quando executamos um arquivo diretamente, essa variável é atribuída o nome de `__main__`. Com isso, conseguimos fazer uma simples checagem condicional para executar rotinas desejadas assim que o programa é executado. No caso do *dashboard*, utilizamos essa condicional para instanciar a classe de interface e o método

show() para abri-la.

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    w = Dashboard()
    w.show()
    sys.exit(app.exec_())
```

Listing 4 – Inicialização da interface

Como mostramos no código 2, o construtor da classe *Dashboard* além de instanciar a classe $SerialIO()$, executa o método $layout\_init()$, que inicializa a janela principal e constrói o botão menu. Parâmetros como tamanho mínimo da tela, *layout* de grade, posicionamento dos elementos são configurados neste método.

Em seguida, quando o botão menu aparece ao usuário, ele é posicionado ao centro da tela, enquanto nenhuma consulta é feita. Ao usuário clicar na fermentação desejada, o programa faz todos os procedimentos explicados para montar as listas de parâmetros.

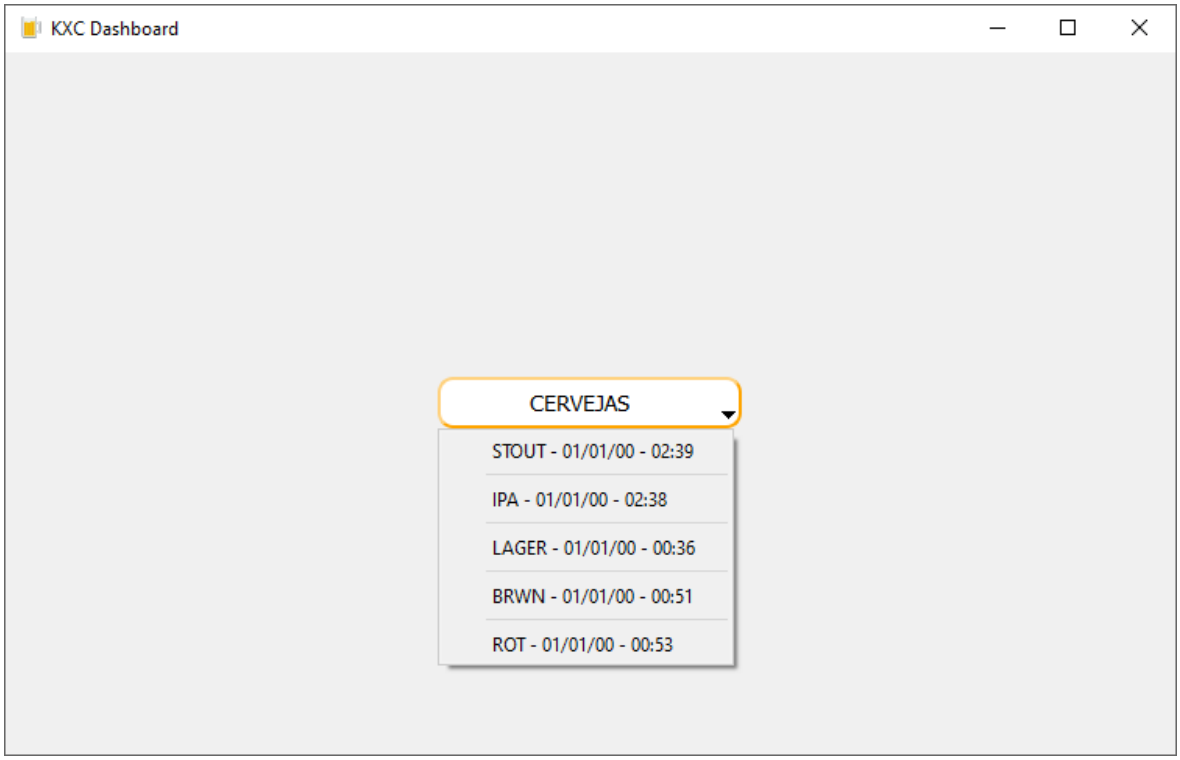


Figura 3.5 – Estado inicial do *Dashboard* com botão menu aberto

Com as respectivas listas no módulo *dashboard*, o último passo é a construção dos quatro gráficos: variação de ângulo, de tensão, de temperatura e leituras nos eixos aX, aY e aZ do acelerômetro.

Criando um método $build\_graphs()$, utilizamos o $PyQtGraph$, que é uma biblioteca pertencente ao $PyQt$, para construir os gráficos a partir das quatro listas montadas anteriormente. Primeiro, instanciamos um objeto do tipo $PlotWidget$ em uma variável chamada $angles\_widget$, e cada gráfico posterior é representado por sua respectiva instância $PlotWidget$. A partir disso, vamos configurando atributos de título de gráfico, legendas para os eixos, cores dos traçados e aspectos visuais.

```
def build_graphs(self):
    angles_widget = pg.PlotWidget()
    angles_widget.setTitle('Variação Angular')
    y = self.angle_list
    x = list(range(len(y)))  # y time points
    angles_widget.setBackground('w')
    pen = pg.mkPen(color=(255, 0, 0), width=2)
    angles_widget.setLabel('left', 'Ângulo (DEG)')
    angles_widget.setLabel('bottom', 'Medições')
    angles_widget.showGrid(x=True, y=True)
    angles_widget.plot(x, y, pen=pen)
```

Listing 5 – Construção do gráfico de Variação Angular

Tendo a lista de cada atributo, utilizamos uma função nativa $len(lista)$ para obtermos seu tamanho. Em seguida, utilizamos a função $range(num)$, que retorna uma lista de inteiros inciando em 0 até o valor passado como parâmetro.Com isso, temos então uma lista de pontos no tempo do Eixo X, representado em todos os gráficos com a legenda "Medições".

Durante o desenvolvimento, foi verificada a necessidade de marcar linhas verticais a cada 24 medições. Como em seu futuro o densímetro realizará uma medida por hora, teremos então 24 medidas por dia, e estas linhas verticais servirão para melhor compreensão do comportamento da cerveja e densímetro dia a dia.

O código 5 demonstra a construção para um dos gráficos, e ela é semelhante para os outros gráficos, com excessão do gráfico de coordenadas. O formato das coordenadas na sua lista é uma lista de vetores, em que cada vetor contêm as três coordenadas. Extraíamos cada uma e a concatenamos para sua respectiva lista:

```
x_list = list()
y_list = list()
z_list = list()
y = self.coord_list
for coord in y:
    x_list.append(coord[0])
    y_list.append(coord[1])
    z_list.append(coord[2])
```

Listing 6 – Extração das coordenadas do acelerômetro

Com as três listas de coordenadas agora separadas, chamamos o método $Plot$ três vezes, em cada uma atribuindo como parâmetro a respectiva lista. Foi adotado o padrão de cores comumente usado para as coordenadas X, Y e Z em construção de gráficos (RGB).

Com isso, o botão menu é movido pela $grid$ para parte central inferior da janela principal e os quatro $PlotWidgets$ são adicionados ao $layout$ de grade e por fim mostrados, como mostra a figura 3.1.

Uma grande vantagem vista em utilizar o $PyQt5$ é que seus gráficos são responsivos, isto é, se adaptam a redimensionamentos de tela, mantendo o gráfico legível em qualquer tamanho permitido inicialmente. Outra vantagem é que o usuário pode interagir com os gráficos, ampliando ou reduzindo a escala e, no caso do gráfico das coordenadas, mover a legenda dos traçados.

4 ENSAIOS

4.1 ROTAÇÃO DO DENSÍMETRO

Como é esperado, o densímetro pode rotacionar em seu eixo enquanto estiver flutuando. Este ensaio demonstra a necessidade de utilizar uma referência vertical livre de acelerações nos eixos X e Z, para que prováveis movimentos de rotação não interfiram no cálculo de seu ângulo *tilt*, demonstrado na equação 2.4.

Foi utilizado o mesmo conjunto de dados, porém com duas referências diferentes para o cálculo variação angular *tilt*: densímetro na vertical, outro a aproximadamente 25º. O ensaio foi realizado segurando o densímetro a 25º da linha vertical e o rotacionando, minimizando ao máximo variações de *tilt*.

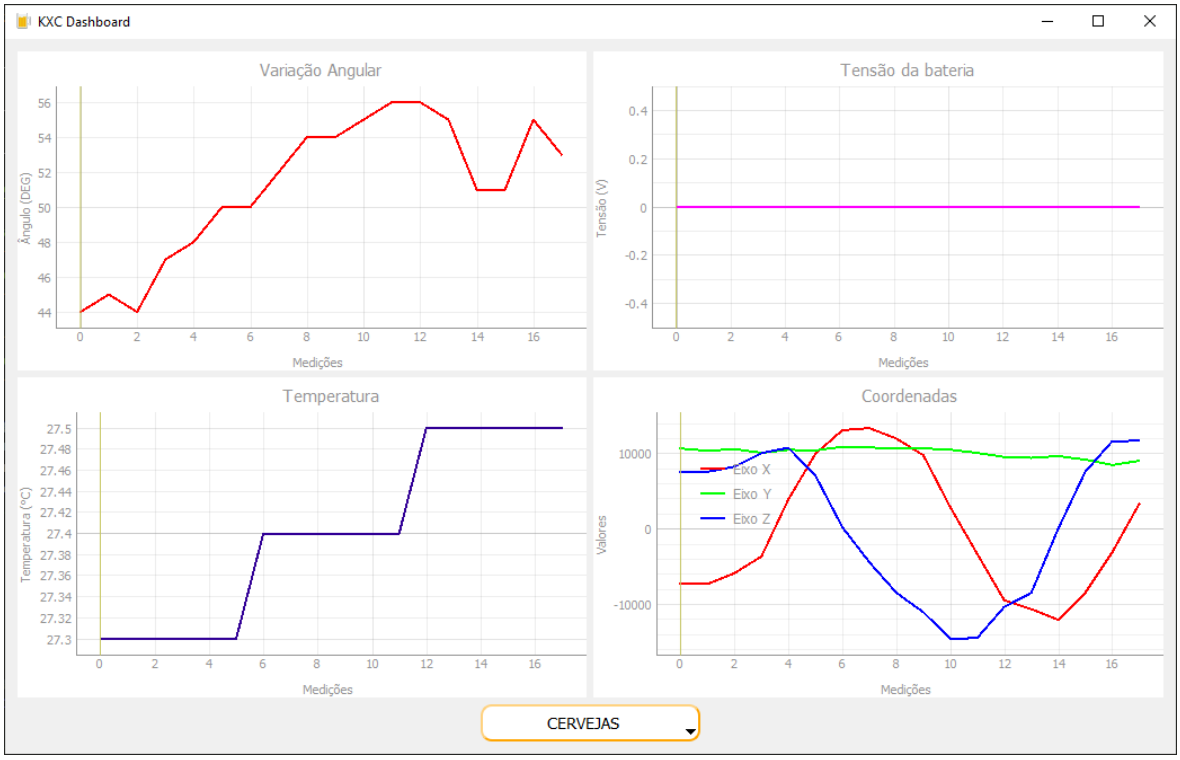


Figura 4.1 – Referência vertical: eixos X e Z com aceleração zero.

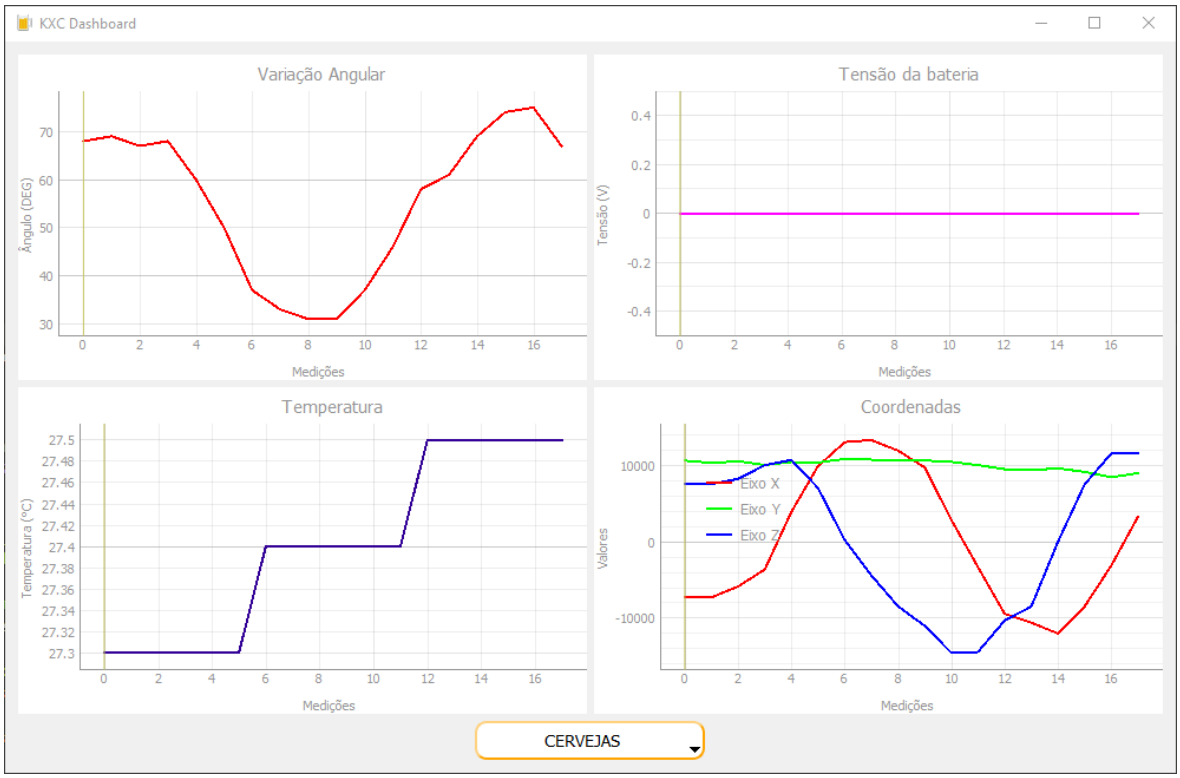


Figura 4.2 – Referência 25º: eixos X e Z com aceleração diferente de zero.

Podemos notar a diferença que a referência faz analisando a variação angular. Enquanto a figura 4.1 nos mostra uma variação máxima de aproximadamente 12º, a figura 4.2 mostra um valor de 45º, valor totalmente discrepante da realidade. A tensão da bateria se manteve em 0 V pois este ensaio foi realizado com alimentação via USB, sem nenhuma bateria conectada.

Vale ressaltar que, devido a impossibilidade de manter o densímetro perfeitamente a 25º enquanto é rotacionado, foi registrada essa variação de 12º com a referência vertical, mas que matematicamente deveria ser zero, de acordo com a equação 2.4.

4.2 SIMULAÇÃO DE FERMENTAÇÃO COM VARIAÇÃO DE TEMPERATURA

Neste ensaio, foi simulado com uma medição por segundo o movimento esperado do densímetro durante uma fermentação. Além disso, o termômetro foi segurado com os dedos para registrarmos aumentos de temperatura. O objetivo foi observar o correto aumento de temperatura e o decaimento do ângulo *tilt*.

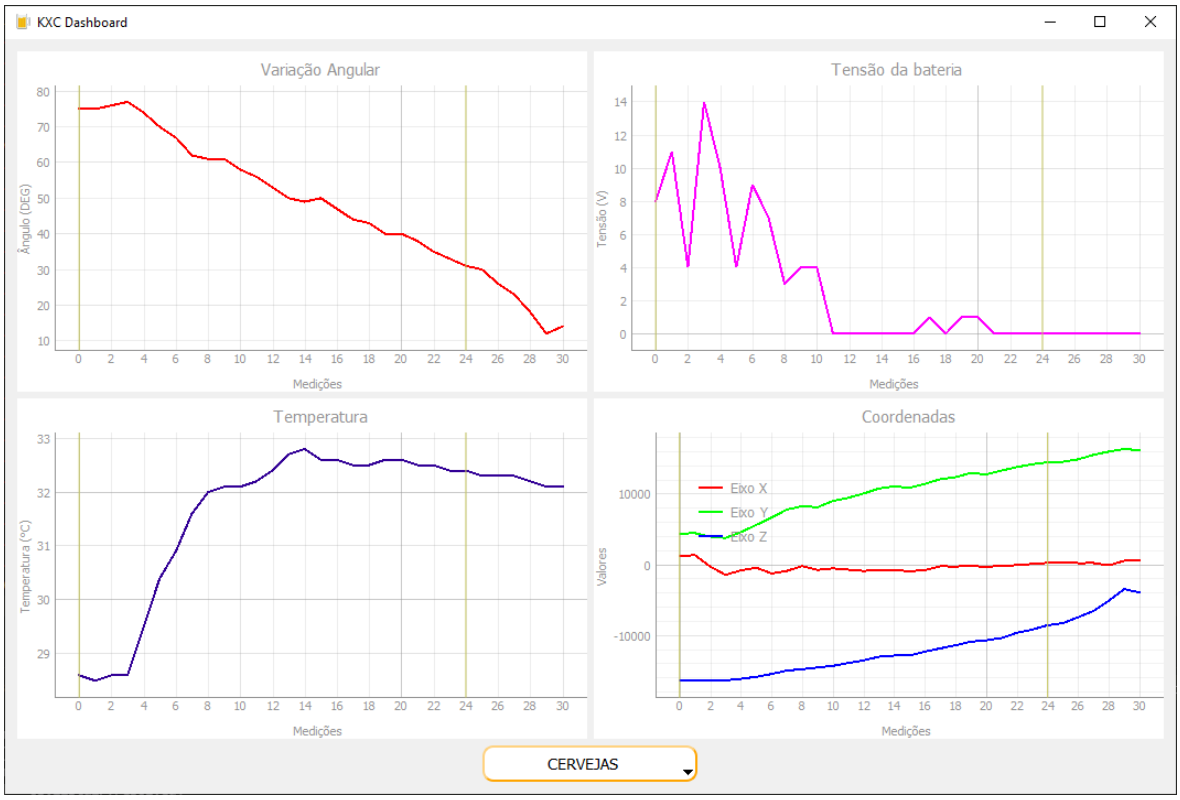


Figura 4.3 – Fermentação simulada

O comportamento esperado foi observado, e variações na tensão da bateria foram registradas pois durante o ensaio os terminais de alimentação estavam expostos e foram tocados.

4.3 MONITORAMENTO DE FERMENTAÇÃO DE CERVEJA

Este ensaio, conduzido pelo orientador Zelenovsky, teve duração de duas semanas, com cada medida sendo realizada de hora em hora. A densidade inicial do líquido foi aferida com um densímetro de vareta, que marcou 1042 g/cm^3. Ao final deste ensaio, o líquido continha uma densidade de 1010 g/cm^3.

Podemos perceber dados importantes como a grande variação angular nos primeiros dois dias, bem como de temperatura, devido ao mosto estar ainda quente (32ºC) quando o densímetro foi colocado. O gráfico da tensão da bateria demonstra a grande eficiência energética adquirida nesta versão no densímetro, perdendo 30 mV de tensão em duas semanas de uso, representando cerca de 3,75% de consumo, considerando os parâmetros de tensão e vida útil do ensaio da bateria a seguir.

Por fim, vale ressaltar a alta compreensibilidade destes gráficos obtidos, mostrando que, mesmo em aplicações reais, as escolhas de design e formas de elucidação destes dados foram certeiras.

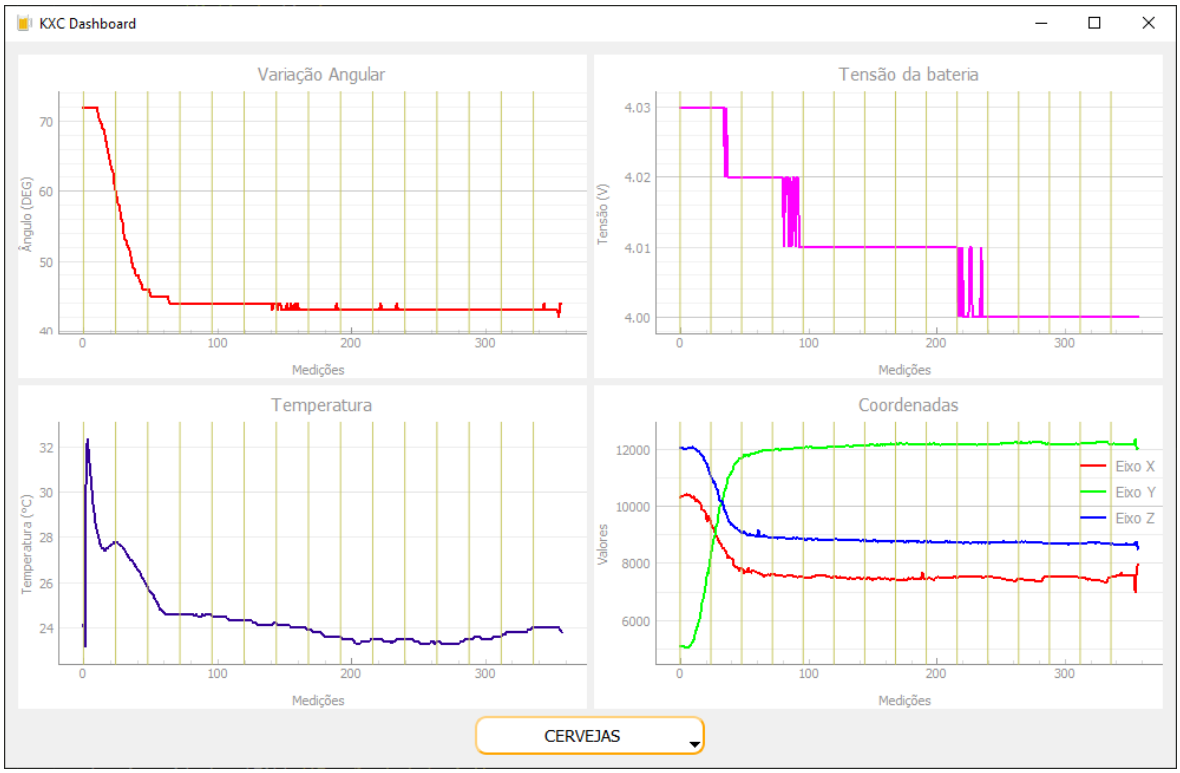


Figura 4.4 – Fermentação de cerveja

Podemos notar algumas semelhanças e diferenças com os testes conduzidos em um trabalho passado, a começar pelo teste 14 (BARRETO, 2019):

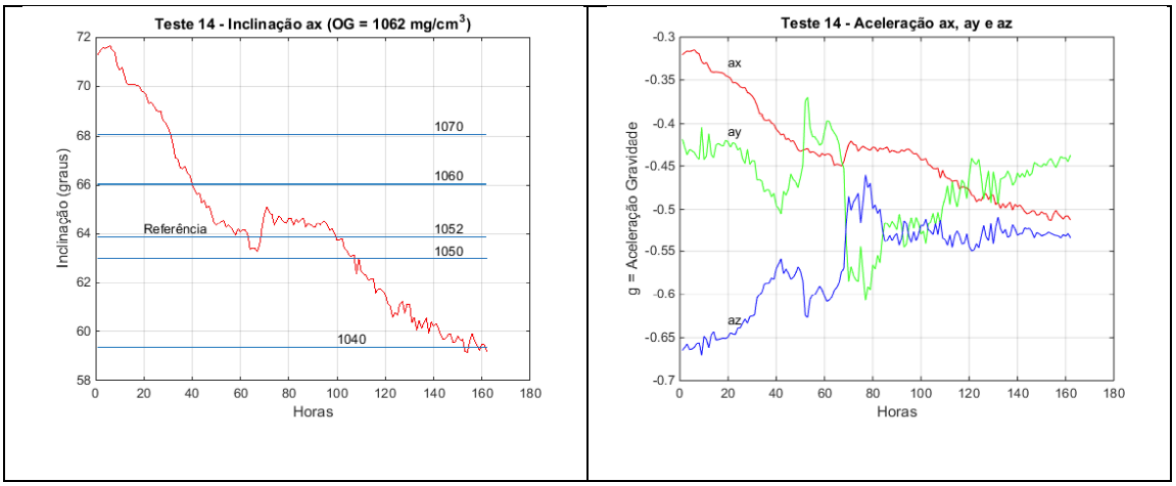


Figura 4.5 – Teste 14 de fermentação (BARRETO, 2019)

Este teste mostra um leve aumento da densidade do líquido após sucessiva diminuição, fato que pode ser atribuído a presença de bolhas no densímetro deste ensaio. Em relação ao ensaio 4.4, isso não ocorre, sendo então uma boa hipótese a boa utilidade do vibrador.

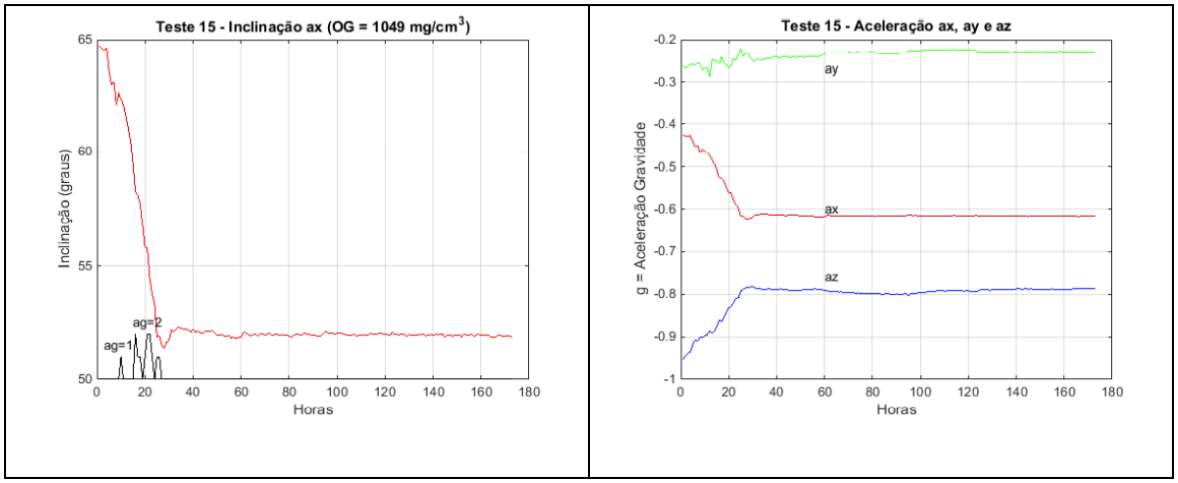


Figura 4.6 – Teste 15 de fermentação (BARRETO, 2019)

O teste 15 também mostrou o comportamento de aumento de densidade após sucessivo decaimento, que pode ser atribuído às bolhas. A semelhança com nosso ensaio ocorre na estabilização da densidade após certo período.

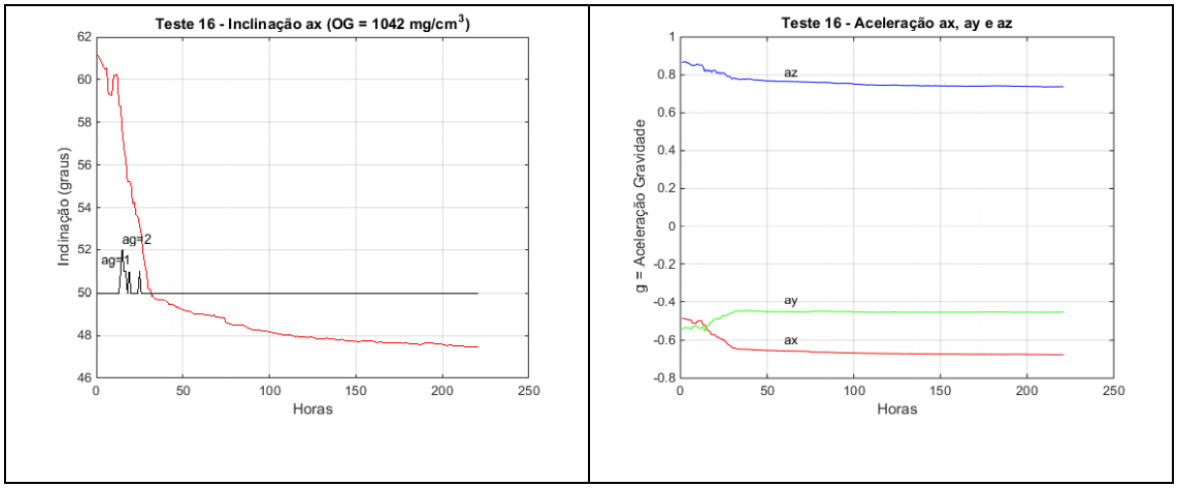


Figura 4.7 – Teste 16 de fermentação (BARRETO, 2019)

O teste 16 mostra bom comportamento da densidade, com rápido decaimento seguido de tendência de estabilização, um pouco semelhante ao nosso ensaio, da figura 4.4.

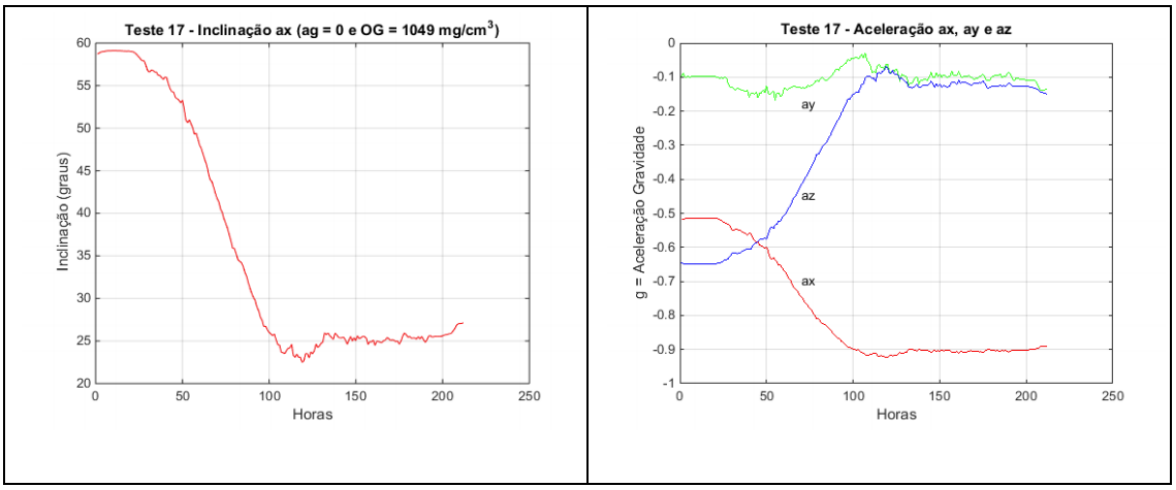


Figura 4.8 – Teste 17 de fermentação (BARRETO, 2019)

O teste 17 observamos o mesmo comportamento dos dois primeiros feitos em Barreto (2017): densidade aumentando após rápido decaimento.

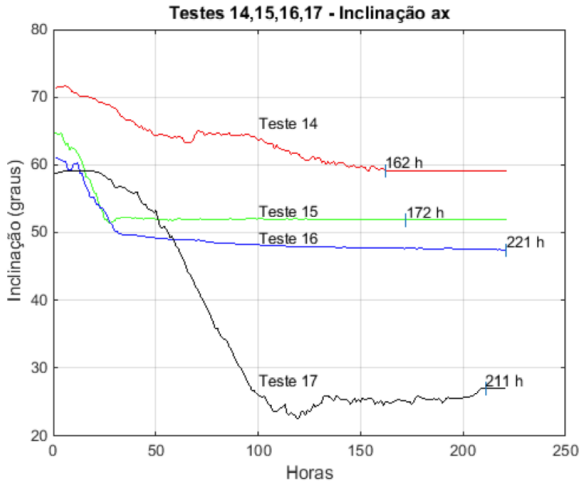


Figura 4.9 – Comparação dos testes de fermentação (BARRETO, 2019)

Com essa comparação, nota-se grande variabilidade nos testes conduzidos, nenhum deles apresentando ao mesmo tempo variação angular somente decrescente seguida de estabilização (indicando fim da fermentação), como foi notado no ensaio deste projeto (figura 4.4), que é o comportamento esperado durante uma fermentação.

Ainda é preciso novos ensaios para estimar a relação entre inclinação e densidade, dada pela equação 2.5 (BARRETO, 2019). Serão necessários vários ensaios para se descobrir a melhor posição dos lastros e depois levantar a curva inclinação versus densidade. É de se notar também que a fermentação estabilizou no quarto dia. Essa é uma informação importante pois pode indicar ao usuário o instante de finalizar a etapa de fermentação.

4.4 BATERIA

Para estimarmos a duração da bateria, foi realizado o ensaio no modo liga/desliga de modo ininterrupto, mantendo o Arduino ligado por 45s e desligado por 15s a cada minuto. A tensão inicial da bateria foi medida em V_{bat} = 4,12 V. O teste foi interrompido quando a bateria atingiu a tensão de 3,27 V, após duração aproximada de 14 dias.

Vale ressaltar que o ensaio foi conduzido em fases iniciais do projeto, em que o circuito era composto apenas pelo Arduino Nano e uma placa comercial com o RTC DS3231. Esta placa tem um pequeno *led* que fica aceso enquanto ela está energizada.

Quando ligado, o densímetro registrou um consumo de 16,5mA. Já desligado, ainda havia pequeno consumo de corrente, 2,2mA, devido a presença do *led* citado.

Enquanto o ensaio esteve em execução, usando um voltímetro, a tensão da bateria era medida duas vezes por dia. As tensões medidas estão na tabela a seguir.

Tabela 4.1 – Teste liga/desliga

Medidas	**Dia**	**Dia**	**Hora**	**VBAT**
1	Terça	10/11/2020	23:00	4,12
2	Quarta	11/11/2020	07:00	4,07
3	Quarta	11/11/2020	23:00	4,00
4	Quinta	12/11/2020	07:00	3,95
5	Quinta	12/11/2020	22:00	3,87
6	Sexta	13/11/2020	07:00	3,82
7	Sexta	13/11/2020	22:00	3,78
8	Sábado	14/11/2020	07:00	3,73
9	Sábado	14/11/2020	22:00	3,66
10	Domingo	15/11/2020	07:00	3,59
11	Domingo	15/11/2020	22:00	3,52
12	Segunda	16/11/2020	07:00	3,45
13	Terça	17/11/2020	07:00	3,27

A figura 4.10 se pode ver a exatidão das medidas de tensão feitas pelo Arduino Nano. Há um pequeno erro de *offset*, que pode ser facilmente corrigido.

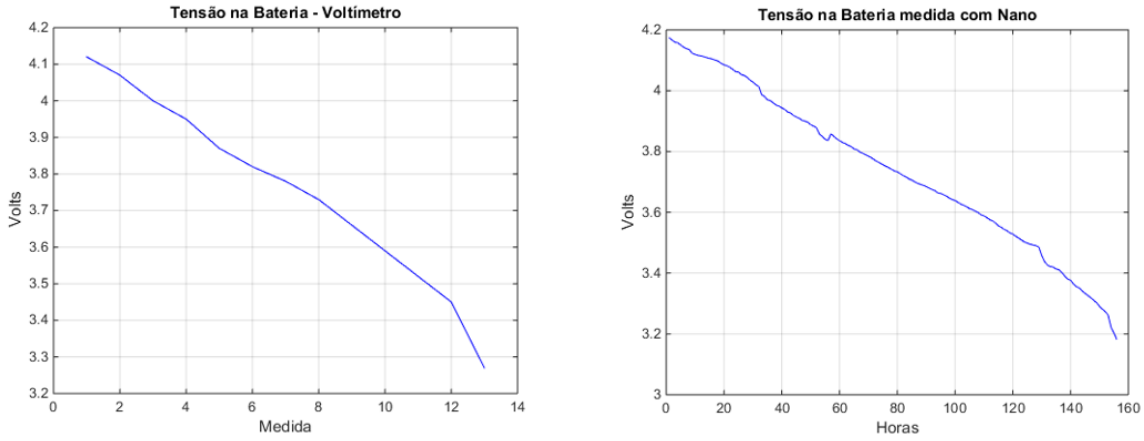


Figura 4.10 – Ensaios de duração de bateria

Em cada medida realizada, foi incrementado um contador salvo em memória flash. Ao final do ensaio, o valor encontrado foi de 9313, ou seja, o ensaio ocorreu 9313 vezes. Considerando que a cada minuto o densímetro ficou ligado por 45 segundos, podemos estimar a duração da bateria. Partimos da hipótese que o densímetro vai ligar a cada hora e que a medição pode ser feita em um minuto (é mais rápida em ensaios reais). Com isso, obtivemos o seguinte rendimento da bateria, em dias:

$$\frac{9313 \times 0.75}{24} = 291 ininterruptos \tag{4.1}$$

Então, seguindo este padrão de consumo, o densímetro conseguiria funcionar durante 291 dias, sem recarga. Na verdade, deve ser um pouco menos, visto que o ensaio foi feito com poucos componentes, quando ainda estava se decidindo sobre o circuito a ser usado.

5 CONCLUSÃO E PROPOSTAS DE EVOLUÇÃO

A criação de um sistema embarcado, capaz de medir a densidade do líquido em que está imerso, propõe diversas soluções para os problemas inicialmente citados com métodos convencionais desta análise. As sucessivas melhorias neste projeto o estão levando em boa direção, com cada vez mais métricas validadas e aprimoramentos realizados, sejam em aspectos pré-existentes do densímetro ou implementação de novas funcionalidades.

Além de melhorias no densímetro, deste projeto começa um novo ramo para futuras melhorias e superiores interpretações dos dados da fermentação, com a criação de um software auxiliar. Este software permite análises refinadas do comportamento do densímetro e seus componentes eletrônicos acoplados.

Diversos ensaios e considerações foram feitas já pensando em seu uso de forma real, como a possível criação de um executável para o KXC *Dashboard*, bem como a procura e conexão automática com o densímetro quando conectado a um computador via USB.

Como deste projeto sai a primeira versão do KXC *Dashboard*, ele foi desenvolvido pensando em escalabilidade, como a construção e captação dos dados de diretório sendo feita de forma dinâmica e automatizada.

Sugere-se que no futuro não mude os caracteres especiais enviados durante o *dump* da memória, usados para delimitar começo e fim dos dados da fermentação, visto que são padrões específicos inseridos em código. O mesmo ocorre para o formato em que os dados são enviados, linha a linha, em relação a quantidade e/ou ordem dos atributos.

Este trabalho também sugere evoluções visuais da interface do KXC *Dashboard*, com o acoplamento de novos dados, ou mesmo a aplicação e verificação da fórmula 2.5, visto que ela em si ainda não foi testada para aferição da densidade de cervejas.

A conexão e reconhecimento do densímetro via porta USB é também importante ser aprimorada, visto que não necessariamente todos os densímetros terão o mesmo nome para o seu controlador USB. Sugere-se então que, ao tentar fazer uma conexão com um dispositivo serial, que o *Dashboard* possa fazer um diálogo com o densímetro usando um conjunto padronizado de perguntas e respostas para determinar o endereçamento correto.

Outra sugestão seria o cálculo prévio do módulo da gravidade antes de iniciar a fermentação, considerando que a aceleração gravitacional varia de acordo com o local da medição. Também é aconselhado o desenvolvimento do programa de uso real, separado do programa de testes que é foi usado durante estes trabalhos sobre o densímetro.

É preciso também de mais ensaios par verificar o melhor local para os lastros e testes para levantar a curva de inclinação versus densidade.

REFERÊNCIAS BIBLIOGRÁFICAS

1 YOUNG, T. W. *Beer, Definition, history, types, brewing process and facts*. 2021. Disponível em: <https://www.britannica.com/topic/beer>.

2 ORAL History Panel on the Development and Promotion of the Intel 8048 Microcontroller. 2021. <https://web.archive.org/web/20120619154428/http://archive.computerhistory.org/resources/access/text/Oral_History/102658328.05.01.acc.pdf>.

3 A Cerveja: Sindicerv - Sindicato Nacional da Indústria da Cerveja. 2021. Disponível em: <https://www.sindicerv.com.br/>.

4 ZELENOVSKY, R.; A, A. M. *Arduino - Guia Avançado Para Projetos*. 2019.

5 SCOTT, M. L. *Programming Language Pragmatics*. 2016.

6 FERREIRA, R. *Inovação na fabricação de cervejas especiais na região de Belo Horizonte. Perspectivas em Ciência da Informação*. 2011.

7 PRIOTO, H. L. *Proposta de Densímetro com Acelerômetro. Dissertação (Graduação em Engenharia Elétrica) – Faculdade de Tecnologia, Universidade de Brasília*. 2018.

8 VASCONCELOS, B. B. de A. F. *PROJETO DE DENSÍMETRO DIGITAL PARA A PRODUÇÃO ARTESANAL DE CERVEJA. Dissertação (Graduação em Engenharia Elétrica) – Faculdade de Tecnologia, Universidade de Brasília*. 2019.

9 ISPINDEL. Repositósio GitHub. 2021. Disponível em: <https://github.com/universam1/iSpindel>.

10 QT for Python Documentation. 2021. Disponível em: <https://doc.qt.io/qtforpython-6/>.

11 PYTHON Documentation. 2021. Disponível em: <https://docs.python.org/3/>.

12 MPU 6050 Documentation and Datasheet. 2021. Disponível em: <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>.

13 READING AND WRITING 1-WIRE® DEVICES THROUGH SERIAL INTERFACES. 2021. Disponível em: <https://www.maximintegrated.com/en/design/technical-documents/app-notes/7/74.html>.

APPENDIX

A APÊNDICE

A.1 CÓDIGO KXC *DASHBOARD*

Disponível em https://github.com/patrickvrb/kxc-dashboard/.

A.1.1 dashboard.py

```python
import sys

import pyqtgraph as pg
from PyQt5 import QtGui
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import (QAction, QApplication, QGridLayout, QMenu,
                             QPushButton, QWidget)

from engine import SerialIO


class Dashboard(QWidget):

    def __init__(self, *args, **kwargs):
        super(Dashboard, self).__init__(*args, **kwargs)
        self.engine = SerialIO()
        self.layout_init()

    def layout_init(self):
        self.list_button_init()
        self.setWindowTitle('KXC Dashboard')
        self.setWindowIcon(QtGui.QIcon('assets/beer.ico'))
        self.setMinimumSize(750, 450)
        self.grid = QGridLayout()
        self.setLayout(self.grid)
        self.grid.addWidget(self.drop_button, 1, 1,
                            Qt.AlignmentFlag.AlignCenter)
        self.show()

    def list_button_init(self):
        self.drop_button = QPushButton('CERVEJAS', self)
        # self.drop_button.resize(100, 32)
        # self.drop_button.move(200, 200)
        menu = QMenu(self)
        dir_list = self.engine.get_directories()
```

```
36          for dir in dir_list:
37              action = QAction(dir.name, self)
38              action.triggered.connect(
39                  lambda _, beer=dir: self.fetch_data(beer))
40              menu.addAction(action)
41              menu.addSeparator()
42          self.drop_button.setMenu(menu)
43          self.drop_button.setStyleSheet(self.get_button_stylesheet())
44          return
45
46      def fetch_data(self, beer):
47          self.engine.ard_dump_mode(beer)
48          self.tension_list, self.temp_list, self.angle_list, self.coord_list = self.eng
49              beer)
50
51          self.build_graphs()
52
53      def build_graphs(self):
54          angles_widget = pg.PlotWidget()
55          angles_widget.setTitle('Variação Angular')
56          y = self.angle_list
57          x = list(range(len(y)))  # y time points
58          angles_widget.setBackground('w')
59          pen = pg.mkPen(color=(255, 0, 0), width=2)
60          angles_widget.setLabel('left', 'Ângulo (DEG)')
61          angles_widget.setLabel('bottom', 'Medições')
62          angles_widget.showGrid(x=True, y=True)
63          angles_widget.plot(x, y, pen=pen)
64
65          tension_widget = pg.PlotWidget()
66          tension_widget.setTitle('Tensão da bateria')
67          y = self.tension_list
68          x = list(range(len(y)))  # y time points
69          tension_widget.setBackground('w')
70          pen = pg.mkPen(color=(255, 0, 255), width=2)
71          tension_widget.setLabel('left', 'Tensão (V)')
72          tension_widget.setLabel('bottom', 'Medições')
73          tension_widget.showGrid(x=True, y=True)
74          tension_widget.plot(x, y, pen=pen)
75
76          temp_widget = pg.PlotWidget()
77          temp_widget.setTitle('Temperatura')
78          y = self.temp_list
79          x = list(range(len(y)))  # y time points
80          temp_widget.setBackground('w')
81          pen = pg.mkPen(color=(50, 0, 150), width=2)
82          temp_widget.setLabel('left', 'Temperatura (°C)')
83          temp_widget.setLabel('bottom', 'Medições')
```

```
84          temp_widget.showGrid(x=True, y=True)
85          temp_widget.plot(x, y, pen=pen)
86
87          coord_widget = pg.PlotWidget()
88          coord_widget.setTitle('Coordenadas')
89          x_list = list()
90          y_list = list()
91          z_list = list()
92          y = self.coord_list
93          for coord in y:
94              x_list.append(coord[0])
95              y_list.append(coord[1])
96              z_list.append(coord[2])
97
98          x = list(range(len(y)))  # y time points
99          coord_widget.setBackground('w')
100         coord_widget.setLabel('left', 'Valores')
101         coord_widget.setLabel('bottom', 'Medições')
102         coord_widget.showGrid(x=True, y=True)
103         coord_widget.addLegend()
104         coord_widget.plot(x, x_list, name="Eixo X", pen=pg.mkPen(
105             color=(255, 0, 0), width=2))
106         coord_widget.plot(x, y_list, name="Eixo Y", pen=pg.mkPen(
107             color=(0, 255, 0), width=2))
108         coord_widget.plot(x, z_list, name="Eixo Z", pen=pg.mkPen(
109             color=(0, 0, 255), width=2))
110         for idx, _ in enumerate(y):
111             if idx % 24 == 0:
112                 angles_widget.addItem(pg.InfiniteLine(
113                     pos=idx), ignoreBounds=True)
114                 temp_widget.addItem(pg.InfiniteLine(
115                     pos=idx), ignoreBounds=True)
116                 tension_widget.addItem(pg.InfiniteLine(
117                     pos=idx), ignoreBounds=True)
118                 coord_widget.addItem(pg.InfiniteLine(
119                     pos=idx), ignoreBounds=True)
120
121         self.grid.addWidget(angles_widget, 0, 0)
122         self.grid.addWidget(tension_widget, 0, 1)
123         self.grid.addWidget(temp_widget, 1, 0)
124         self.grid.addWidget(coord_widget, 1, 1)
125         self.grid.addWidget(self.drop_button, 2, 0, 1, 2,
126                             Qt.AlignmentFlag.AlignCenter)
127
128         return
129
130     def update_plot_data(self, data):
131
```

```
132             self.x = self.x[1:]  # Remove the first y element.
133             # Add a new value 1 higher than the last.
134             self.x.append(self.x[-1] + 1)
135 
136             self.y = self.y[1:]  # Remove the first
137             self.y.append(data)  # Add the new value.
138 
139             self.data_line.setData(self.x, self.y)  # Update the data.
140 
141     def get_button_stylesheet(self):
142         return "background-color: white;border-style: outset;border-width: 2px;border-
143 
144 
145 if __name__ == "__main__":
146     app = QApplication(sys.argv)
147     w = Dashboard()
148     w.show()
149     sys.exit(app.exec_())
```

A.1.2 engine.py

```
1  from math import acos, pi, sqrt
2  from time import sleep
3  
4  import serial
5  import serial.tools.list_ports
6  
7  
8  class SerialIO():
9      def __init__(self):
10         self.setup()
11         self.list_dir_mode()
12 
13     def setup(self):
14         ports = list(serial.tools.list_ports.comports())
15         self.arduino_data = None
16         # Procura pelo Arduino conectado:
17         for p in ports:
18             if 'USB-SERIAL CH340' in p.description:
19                 try:
20                     self.arduino_data = serial.Serial(p.name, 9600, timeout=1)
21                     print('Arduino encontrado!')
22                 except Exception:
23                     print('Porta Serial Arduino ocupada!')
24                     exit()
25 
```

```
26          if not self.arduino_data:
27              print('Arduino não encontrado!')
28              exit()
29
30          sleep(2)
31
32      def get_integer_value(self, str_value):
33          if str_value.startswith('+'):
34              return int(str_value[1:])
35          else:
36              return int(str_value)
37
38      def get_x_y_z_dump(self, input_str):
39          # Retira o primeiro e os quatro últimos elementos da linha
40          var_list = input_str.split(' ')[1:][:-3]
41          if len(var_list) == 0:
42              return 0, 0, 0
43          x, y, z = [self.get_integer_value(value) for value in var_list]
44          return x, y, z
45
46      def get_temp_dump(self, input_str):
47          temp = self.get_integer_value(input_str.split(' ')[4]) / 10
48          if temp < 0 or temp > 50:
49              return self.prev_temp
50          else:
51              self.prev_temp = temp
52              return temp
53
54      def get_bat_tension_dump(self, input_str):
55          return self.get_integer_value(input_str.split(' ')[5])
56
57      def real_time_read(self):
58          reference_vector = None
59          current_vector = None
60          while True:
61              normalized_buffer = self.arduino_data.read_until().decode('utf-8')
62              if not normalized_buffer.startswith('==>'):
63                  x, y, z = self.get_x_y_z(normalized_buffer)
64                  current_vector = [x, y, z]
65                  if reference_vector is None:
66                      reference_vector = current_vector
67                  if current_vector and reference_vector:
68                      angulo = self.angle_calc(reference_vector, current_vector)
69                      print(round(angulo, 0))
70
71      def angle_calc(self, coord_vertical, vector):
72          try:
73              produto_interno = sum([vector[i] * coord_vertical[i]
```

```
                                    for i in range(3)])
            u = sqrt(sum([vector[i] * vector[i] for i in range(3)]))
            v = sqrt(sum([coord_vertical[i] * coord_vertical[i]
                          for i in range(3)]))
            angulo = acos(produto_interno/(u*v)) * 180/pi
        except Exception:
            angulo = 0
        return round(angulo, 0)

    def ard_dump_mode(self, beer):
        # Forçando modo 17 (Config Mode)
        self.arduino_data.write('17\n'.encode('utf-8'))
        self.arduino_data.flushOutput()
        # Printando dados do primeiro diretório
        self.arduino_data.write(
            ('p ' + str(beer.index) + '\n').encode('utf-8'))
        self.arduino_data.flushOutput()
        self.save_dump(beer.name.split(' ')[0])

    def save_dump(self, beer_name):
        with open('dumps/' + beer_name + '_dump.txt', 'w') as f:
            while True:
                buffer = self.serial_read()
                if '#[f' in buffer:
                    while True:
                        buffer = self.serial_read()
                        if not 'f]#' in buffer:
                            f.write(buffer + '\n')
                        else:
                            break
                    self.voltar_menu_serial()
                    return

    def list_dir_mode(self):
        # Forçando modo 17 (Config Mode)
        self.arduino_data.write('17\n'.encode('utf-8'))
        self.arduino_data.flushOutput()

        # Printando todos diretórios ocupados
        self.arduino_data.write('l\n'.encode('utf-8'))
        self.arduino_data.flushOutput()

        self.save_directories()

    def save_directories(self):
        with open('dumps/directories.txt', 'w') as f:
            while True:
                # Ler linha a linha,  Decodificação para incluir \n, \r
```

```
                buffer = self.serial_read()
                if 'Dir:' in buffer:  # Primeira medida do dump
                    while buffer:
                        buffer = self.serial_read()
                        if buffer:
                            f.write(buffer + '\n')
                    self.voltar_menu_serial()
                    return

    def get_directories(self):
        dir_list = list()
        with open('dumps/directories.txt', 'r') as f:
            for idx, line in enumerate(f, start=0):
                beer = Beer(idx, line.split()[
                            1] + ' - ' + line.split()[2] + ' - ' + line.split()[3])
                dir_list.append(beer)
        return dir_list

    def serial_read(self):
        try:
            buffer = self.arduino_data.readline().decode('ISO-8859-1')
        except Exception:
            buffer = None
        return buffer.rstrip()

    def get_measures_lists(self, beer):
        tension_list = list()
        temp_list = list()
        angle_list = list()
        coord_list = list()
        buffer = ''
        with open('dumps/'+beer.name.split(' ')[0] + '_dump.txt', 'r') as f:
            while buffer[:4] != '0004':
                buffer = f.readline()

            # Vetor referencia fixado na vertical [0, 16704, 0]
            coord_vertical = [0, 16704, 0]
            while True:
                try:
                    tension = self.get_bat_tension_dump(buffer)
                    tension_list.append(tension)

                    temp = self.get_temp_dump(buffer)
                    temp_list.append(temp)

                    vector = self.get_x_y_z_dump(buffer)
                    angle_list.append(self.angle_calc(coord_vertical, vector))

```

```
170                         coord_list.append(self.get_x_y_z_dump(buffer))
171
172                         buffer = f.readline()
173                         if buffer[:4] == 'FFFF':
174                             break
175                     except Exception:
176                         break
177             return tension_list, temp_list, angle_list, coord_list
178
179         def voltar_menu_serial(self):
180             self.arduino_data.write('x\n'.encode('utf-8'))
181             self.arduino_data.flushOutput()
182             return
183
184
185 class Beer(object):
186     index = 0
187     name = ''
188
189     def __init__(self, index, name):
190         self.index = index
191         self.name = name
192
193
194 if __name__ == "__main__":
195     SerialIO()
```