Centro Federal de Educação Tecnológica de Minas Gerais
Campus Divinópolis
Graduação em Engenharia Mecatrônica

Felipe Paulo Tavares de Oliveira
Vítor César Tavares

Desenvolvimento de um sistema para monitoramento da eficiência alimentar de gado criado em regime de confinamento

Divinópolis
2018

Felipe Paulo Tavares de Oliveira
Vítor César Tavares

Desenvolvimento de um sistema para monitoramento da eficiência alimentar de gado criado em regime de confinamento

Monografia de Trabalho de Conclusão de Curso apresentada ao Colegiado de Graduação em Engenharia Mecatrônica como parte dos requisitos exigidos para a obtenção do título de Engenheiro Mecatrônico.
Áreas de integração: Eletrônica, Mecânica e Computação.

Orientador: Prof. Doutor Luiz Cláudio Oliveira

Divinópolis
2018

Felipe Paulo Tavares de Oliveira
Vítor César Tavares

Desenvolvimento de um sistema para monitoramento da eficiência alimentar de gado criado em regime de confinamento

Monografia de Trabalho de Conclusão de Curso apresentada ao Colegiado de Graduação em Engenharia Mecatrônica como parte dos requisitos exigidos para a obtenção do título de Engenheiro Mecatrônico.
Áreas de integração: Eletrônica, Mecânica e Computação.

Comissão Avaliadora:

Prof. Dr. Lucio Flavio Santos Patricio
CEFET/MG *Campus* V

Prof. Dr. Luiz Cláudio Oliveira
CEFET/MG *Campus* V

Prof. M. Márcio Alves de Aguiar
CEFET/MG *Campus* V

Divinópolis
2018

Centro Federal de Educação Tecnológica de Minas Gerais CEFET-MG
Campus Divinópolis
Curso de Engenharia Mecatrônica

Monografia intitulada "Desenvolvimento de um sistema para monitoramento da eficiência alimentar de gado criado em regime de confinamento", de Felipe Paulo Tavares de Oliveira e Vítor César Tavares, graduandos em Engenharia Mecatrônica, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Luiz Cláudio Oliveira - CEFET-MG / Campus Divinópolis

Prof. Dr. Lucio Flavio Santos Patricio - CEFET-MG / Campus Divinópolis

Prof. M. Márcio Alves de Aguiar - CEFET-MG / Campus Divinópolis

Prof. Dr. Lucio Flavio Santos Patricio
Coordenador do Curso de Engenharia Mecatrônica
CEFET-MG / Campus Divinópolis

Divinópolis - Dezembro de 2018

Dedico a todos àqueles que amo.

Dedico este trabalho à minha mãe, aos meus familiares e amigos que estiveram comigo durante o percurso.

Agradecimentos

Agradeço,

À minha minha mãe, meu pai e meus irmãos pelo apoio e suporte incondicional nas minhas decisões durante a graduação e na vida.

À Stefany Faria que aguentou todas as minhas reclamações, desabafos, alegrias e tristezas durante o curso e com certeza o fará, ao meu lado, durante toda a minha vida.

Aos meus amigos que a todo momento estiveram presentes e tornaram esta caminhada mais fácil.

Agradecimentos

Agradeço,

À minha mãe, Adriana, que me ensinou valores e me indicou qual o melhor caminho. Aos meus avós, tios e tias que me deram abrigo, apoio e carinho. À minha irmã, Sara, que me mostrou que uma personalidade diferente não implica em um amor menor.

Aos meus amigos, que durante o ensino médio e a graduação contribuíram para a minha formação humana, me tornando o que sou hoje.

Ao meu orientador, Luiz Cláudio, que nos defendeu e guiou durante a elaboração deste trabalho.

A todos aqueles que de alguma forma impactaram na minha vida, pois querendo ou não, fizeram ou fazem parte de quem eu sou.

My thirst for knowledge cannot be quenched.

Rubick, the Grand Magus

I have several theories I'd like to put into practice. Should have taken up science, Invoker, magic's long term prospects are dim.

Boush, the Tinker

Resumo

O projeto proposto visa o desenvolvimento de um sistema para monitorar a eficiência alimentar de gado confinado, englobando três das grandes áreas da engenharia: eletrônica, mecânica e automação. Tem-se como intenção, proporcionar ao criador de gado para corte uma ferramenta que auxilie a tomada de decisão a respeito das ações a serem aplicadas no rebanho confinado. Com a perspectiva de crescimento do mercado de boi confinado em todo o país, problemas inerentes à essa cultura tendem a aumentar. Um dos problemas enfrentados pelos produtores brasileiros é a dificuldade do pecuarista, principalmente os pequenos e médios produtores que dispõe de poucos recursos, em saber quando o animal confinado deve ser abatido, ou se o mesmo dará lucro após o abate. A informação da quantidade de alimento ingerido por um animal, e qual sua eficiência para metabolizar isto em massa, é essencial para uma tomada de decisão. O acesso a esses dados é essencial para que sejam traçadas estratégias de redução de gastos, aumento de lucros e melhora na genética do gado. O modelo atual de confinamento, não acompanha em tempo real o peso e a rotina de cada animal separadamente, o que diminui a rentabilidade do rebanho como um todo. Com o avanço tecnológico atual, e o conceito de Internet das Coisas, é possível mudar a forma como se obtém informações sobre uma criação. Sendo assim, o sistema proposto será projetado para atender ao crescimento do mercado de gado confinado e também proporcionar ao produtor maior controle sobre seu rebanho. O sistema será composto por elementos sensores, para realizar a pesagem, microcontroladores e circuitos eletrônicos que serão projetados para tratar os sinais provenientes da medição de peso e do consumo alimentar de cada animal, além de armazenar essas informações, já tratadas, em um banco de dados. Outro componente importante do projeto será o aplicativo *web*, que poderá ser acessado de qualquer local com internet, levando informação e proporcionando ao usuário ferramentas para acompanhar o desenvolvimento do seu rebanho, de maneira individual ou geral, podendo traçar padrões de alimentação e ganho de peso, além de detectar doenças que acarretam na perda de massa ou apetite do animal.

Palavras-chave: Automação; IoT; Eficiência alimentar.

Abstract

The proposed project aims the development of a system to monitor the feed efficiency of confined livestock, encompassing three of the major areas in engineering: electronics, mechanics and automation. The intention is to provide the farmer with a tool to help him to make decisions regarding the actions towards the confined herd. With the prospects of growth of the confined cattle market across the country, this culture's inherent problems tend to increase. One of the problems faced by Brazilian producers is the difficulty of the cattle rancher, especially the small and medium-sized producers with limited resources, to know when the confined animal should be slaughtered, or whether it will make a profit after slaughter. Information on the amount of food ingested by an animal, and how efficiently it is to metabolize it in bulk, is essential for decision making. Access to these data is essential for strategies to reduce costs, increase profits and improve livestock genetics. The current model of confinement does not accompany in real time the weight and the routine of each animal separately, which reduces the profitability of the herd as a whole. With the current technological advance, and the concept of Internet of Things, it is possible to change the way one can acquire information about a herd. Therefore, the proposed system will be designed to meet the growth of the confined livestock market and also provide the producer with greater control about his flock. The system will be composed of weighing sensors, microcontrollers and electronic circuits that will be designed to treat the signals coming from the measurement of weight and food consumption of each animal, as well as storing this information, already treated, in a database. Another component of the project will be the web application, which can be accessed from any location through the Internet, taking information and providing the user with tools to track the development of the herd, individually or generally, being able to trace feeding patterns and weight gains to detect diseases that lead to loss of body mass or appetite.

Key-words: Automation. IoT. Feed Efficiency.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos e Notação

ADC	Analog-to-Digital converter
ARM	Acorn RISC Machine
BLE	Bluetooth Low Energy
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
CSS	Cross Site Scripting
EMBRAPA	Empresa Brasileira de Pesquisa Agropecuária
FEA	Análise de Elementos Finitos
GPRS	General Packet Radio Services
HCI	Interface Controller Host
HTML	Hypertext Markup Language
I2C	Inter-Integrated Circuit
IOS	iPhone Operating System
MVC	Model View-Controller
PIB	Produto Interno Bruto
POO	Programação Orientada a Objeto
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Compute
RTOS	Real Time Operating Sy
RTC	Real Time Counter
SAR	Successive Approximation Register
SOAP	Simple Object Access Protocol
SPI	Serial Peripheral Interface
SQL	Structured Query Language
UART	Universal Asynchronous Receiver/Transmitter
UFMG	Universidade Federal de Minas Gerais
UFU	Universidade Federal de Uberlândia
URI	Uniform Resource Identifier
XML	Extensible Markup Language
WSDL	Web Services Description Language

Capítulo 1

Introdução

A pecuária, no Brasil, representou, em 2015, 6% do PIB brasileiro ou 30% do PIB do agronegócio e, nos últimos 5 anos teve uma evolução de 45% neste índice [3]. O relatório também afirma que o número de bois abatidos passou de aproximadamente 10 milhões de cabeças em 1997 para quase 27 milhões em 2015 e esse substancial aumento relativo não estava relacionado à dimensão de área de pasto, o que, em algumas regiões, até diminuiu ao longo dos anos. Isso ocorreu por avanços tecnológicos como aprimoramento da alimentação do bovino e o desenvolvimento do sistema de confinamento e semi-confinamento que proporcionou o aumento de bois convivendo no mesmo espaço disponível. [3]

Sistema esse em que o gado confinado está sujeito a uma série de doenças e problemas metabólicos provenientes de falhas no manejo e de adaptação do animal à dieta. Algumas dessas doenças e distúrbios possuem como sintomas a diminuição do peso e do apetite do animal. No sistema atual, essa perda só pode ser notada visualmente, o que acarreta um maior tempo para a identificação das doenças. [4]. Também é comum o produtor ter dificuldades de monitorar a evolução do peso do boi, portanto está sujeito a riscos de manter um animal que não aumentará seu peso e consequentemente irá gerar prejuízos. [5] Visando no aumento da produtividade da pecuária e a solução de problemas como detecção de doenças e o acompanhamento do peso dos animais, existem esforços relacionados ao aprimoramento da alimentação, do melhoramento genético e tecnologias relacionadas ao ambiente e ao comportamento do animal.[6]

Atualmente é inviável para o produtor rural a pesagem constante de seu rebanho, visto que a frequente manipulação pode acarretar estresse e perda de peso [5], visto que, apenas 10% dos produtores possuem, em sua propriedade, uma balança que, normalmente, é utilizada apenas para conferir o peso informado pelo frigorífico que compra o animal [7].

Uma destas tecnologias é o monitoramento do peso utilizando balanças automatizadas, pois é um fator apontado como facilitador para a otimização de lucros na pecuária, pois o produtor terá ciência se a sua criação ainda está engordando ou com peso estacionário,

além de informações sobre o comportamento. [8]

1.1 Definição do Problema

Este trabalho estuda as limitações atualmente existentes no processo de pesagem do gado criado em regime de confinamento bem como a falta de informação que os produtores têm para otimizar seus lucros.

1.2 Motivação

O tema do trabalho surge de demandas relacionadas ao agronegócio no Brasil. Um dos problemas encontrados foi o desconhecimento por parte do produtor de como escolher o melhor momento para o abate de seus animais e a dificuldade deste em manter um banco de dados com o peso dos bois durante a vida. Estas limitações ocorrem devido ao alto investimento em balanças digitais de um sistema que armazene os dados coletados automaticamente e devido à dificuldade em medir a eficiência alimentar do boi confinado, medida esta que relaciona o ganho de peso com quantidade de comida ingerida.

1.3 Objetivos do Trabalho

1.3.1 Objetivo Geral

Desenvolver um conjunto de dispositivos para o monitoramento e registro individual do peso dos animais para o cálculo contínuo da eficiência alimentar de bovinos em regime de confinamento e apresentar estes dados através de um aplicativo de celular para futuras tomadas de decisões.

1.3.2 Objetivos Específicos

- Especificar e projetar uma balança de grande porte para até duas toneladas
- Analisar e projetar uma estrutura robusta para a balança de grande porte
- Projetar e construir uma balança de pequeno porte para ração, de até cinquenta quilos
- Projetar e construir a estrutura para a balança de pequeno porte
- Projetar e implementar circuitos condicionadores de sinais para as balanças de pequeno e grande porte

- Desenvolver um sistema de identificação individual para cada animal
- Implementar uma rede de comunicação entre os dispositivos
- Programar um firmware para o circuito de leitura das variáveis das balanças, identificação e comunicação em rede
- Implementar um banco de dados na nuvem para armazenamento dos dados provenientes do monitoramento contínuo e individual
- Elaborar um aplicativo para dispositivos móveis da plataforma Android

1.4 Estado da Arte

Atualmente existem poucos trabalhos acadêmicos sendo desenvolvidos com o objetivo de projetar equipamentos para o monitoramento de peso em criação de bovinos, e as pesquisas são, em boa parte, realizadas por empresas privadas. Um dos produtos mais relevantes no mercado atualmente é o sistema de monitoramento denominado de plataforma de pecuária de precisão criado pela BOSCH [9], que são posteriormente tratados e exibidos em forma de gráficos que indicam a evolução do peso dos animais de maneira individual ou agrupada. A plataforma em questão não armazena dados referentes a quantidade de alimento ingerida por cada animal, portanto não é possível mensurar a eficiência alimentar dos animais, dado relevante para identificar o melhor momento para o abate.

No mercado, são encontradas outros sistemas de pesagem automatizado, as empresas DeLaval[10] e Toledo Brasil [11], possuem modelos comerciais. Porém, estes sistemas não eliminam o contato humano com o rebanho, sendo necessário o deslocamento dos animais até os postos de pesagem, tornando inviável o acompanhamento contínuo do peso de cada animal.

No âmbito acadêmico a EMBRAPA realizou, em parceria com a empresa Intergado e as universidades federeis de Viçosa (UFV) e de Minas Gerais (UFMG), o desenvolvimento de uma balança que armazena os dados de peso e ingestão alimentar de bovinos, para que se desenvolvessem pesquisas relacionadas a dietas, identificação de doenças e melhoramento genético [12]. Porém, o sistema não disponibiliza os dados coletados em uma rede de fácil acesso, sendo inviável o acesso as informações através da internet. Outra característica negativa do sistema é a disposição da balança de pesagem, que não permite um fluxo grande de animais quando comparado ao o sistema desenvolvido pela BOSCH, aumentando de maneira significativa sua aplicação em grandes criações.

Em 2007, um trabalho feito por NADALIN, utilizou de sensores piezo-elétricos para mensurar a força peso das pisadas dos bois para estimar a evolução de seu peso. O trabalho não apresentava exatamente o peso do animal por causa de problemas em condicionamento de sinal, porém, foi efetivo em mostrar a evolução do peso em protótipos. [13]

Para demonstrar que o assunto tem poucas pesquisas a tabela 1 foi elaborada com o objetivo de informar a quantidade de resultados relevantes, de 2013 até 2018, relacionados ao escopo do presente trabalho, retornados para cada palavra-chave informada na Biblioteca Digital Brasileira de Teses e Dissertações e no repositório Alice da EMBRAPA.

Tabela 1.1: Resultado Pesquisas

Palavra-Chave	Resultados relevantes	Fonte
desenvolvimento balança automática	0	http://bdtd.ibict.br/vufind/Search/Results?lookfor=desenvolvimento+balan%C3%A7a+autom%C3%A1tica&type=AllFields&limit=20&sort=relevance
desenvolvimento monitoramento bovino	0	http://bdtd.ibict.br/vufind/Search/Results?lookfor=desenvolvimento+monitoramento+bovino&type=AllFields&limit=20&sort=relevance
balança automática	0	https://www.alice.cnptia.embrapa.br/alice/simple-search?query=balan%C3%A7a+autom%C3%A1tica
desenvolvimento monitoramento pecuária	0	https://www.alice.cnptia.embrapa.br/alice/simple-search?query=desenvolvimento+monitoramento+pecu%C3%A1ria
desenvolvimento eficiência alimentar	0	https://www.alice.cnptia.embrapa.br/alice/simple-search?location=%2F&query=desenvolvimento+efici%C3%AAncia+alimentar&rpp=10&sort_by=score&order=desc

1.5 Organização do Documento

O presente trabalho é dividido em quatro capítulos. São eles: Introdução, Fundamentos, Desenvolvimento e Considerações finais.

O primeiro capítulo compreende a contextualização, as pesquisas e os desenvolvimentos mais recentes sobre o tema, a problematização, os objetivos de projeto e também a estratégia escolhida para a sua realização.

No capítulo dois, são retratados os principais conceitos envolvidos neste trabalho e também é apresentado um breve histórico sobre a pecuária no Brasil, e sua evolução tecnológica.

No terceiro capítulo são evidenciados os principais requisitos utilizados como base para desenvolvimento do projeto e alguns componentes escolhidos para a confecção do produto final. As justificativas do uso destes materiais também são encontradas neste capítulo do documento. Partes do código em desenvolvimento para a utilização no produto final também são mostrados.

Por fim, são apresentadas as conclusões obtidas a partir do desenvolvimento do trabalho até o momento, assim como um acompanhamento da evolução através de um comparativo com o cronograma planejado.

Capítulo 2

Fundamentos

Neste capítulo será abordado a a evolução das pesquisas relacionado a ganho de produtividade na pecuária, então será apresentado a divisão para o desenvolvimento do trabalho proposto. Por fim, uma fundamentação teórica com os assuntos necessário para o bom entendimento do projeto será desenvolvido.

2.1 Metodologia

Para que o trabalho seja desenvolvido, é necessário que se tenha uma metodologia a ser seguida, permitindo que as tarefas sejam realizadas de maneira a se complementarem na montagem final do projeto.

Para que as balanças funcionem de maneira efetiva, é necessário que os sinais provenientes de sua medição possam ser interpretados por um micro controlador, para isso, serão projetados elementos transdutores, para que as balanças sejam facilmente calibráveis.

Para que seja possível a identificação de cada animal, será utilizada uma tecnologia de identificação individual. O sistema de identificação será implementado levando em consideração fontes de alimentação energética, alcance dos sensores e outros aspectos técnicos.

Existe uma grande variedade de microcontroladores disponíveis no mercado, variando da quantidade de portas à capacidade de armazenamento e processamento de dados. Será determinado o microcontrolador que mais se adequará ao projeto, e a partir da determinação do mesmo, serão desenvolvidas rotinas de leitura de sensores e transmissão de dados, sendo construído assim o *firmware* que além de gerenciar a rede, efetuará a gravação dos dados em um banco de dados.

Os elementos eletrônicos necessitam de uma proteção robusta, que além de protege-los de intempéries deverá facilitar a comunicação entre os mesmos.

Para que todo o sistema seja interligado, é necessária a determinação de uma rede de comunicação, para que seja determinada essa rede, serão estudados os protocolos de

comunicação que mais se adequam ao sistema e também os padrões físicos que mais facilitam a implementação da rede.

Para que o usuário tenha acesso a toda a informação coletada pelo sistema, será desenvolvido um aplicativo *web*, que permite o acesso as funcionalidades do sistema de qualquer plataforma com acesso à internet.

2.2 Revisão Bibliográfica

Eficiência alimentar é um importante aspecto para o aumento da produtividade bovina, para tanto, foram desenvolvidas técnicas que visam medir esta variável de forma indireta.

Segundo Arthur, existem mais de 40 formas de se mensurar a eficiência alimentar de bovinos, porém, a maneira mais utilizada é o cálculo de conversão alimentar, obtido através da relação entre a quantidade de alimento ingerida e o peso ganho em um determinado período de tempo. O autor aponta a existência de falhas neste método de avaliação, visto que são considerados apenas os parâmetros de peso e idade do animal. [14]

Ferreira, entretanto, diz que para bovinos em regime de confinamento é necessária a constante avaliação individual da conversão alimentar dos animais, pois, desta forma, é possível avaliar o custo unitário individual, levando a um maior controle da produtividade. [15]

Abreu ratifica Ferreira e complementa dizendo que animais com baixos niveis de conversão alimentar podem inviabilizar o método de criação em confinamento. Portanto, recomenda-se o acompanhamento constante do peso dos animais, possibilitando a obtenção de dados para o cálculo desta variável. [16]

2.3 Fundamentação Teórica

2.3.1 Elemento sensor

Para realizar a medição de qualquer grandeza física, é necessário que se tenha um elemento sensor, que possa efetivamente quantificar a grandeza que se deseja medir. Para que sejam avaliados valores de peso, um sistema de aquisição deve ser capaz de mensurar quanta força foi aplicada. Para isso, são utilizadas células de carga. A figura 2.1 mostra uma célula de carga.

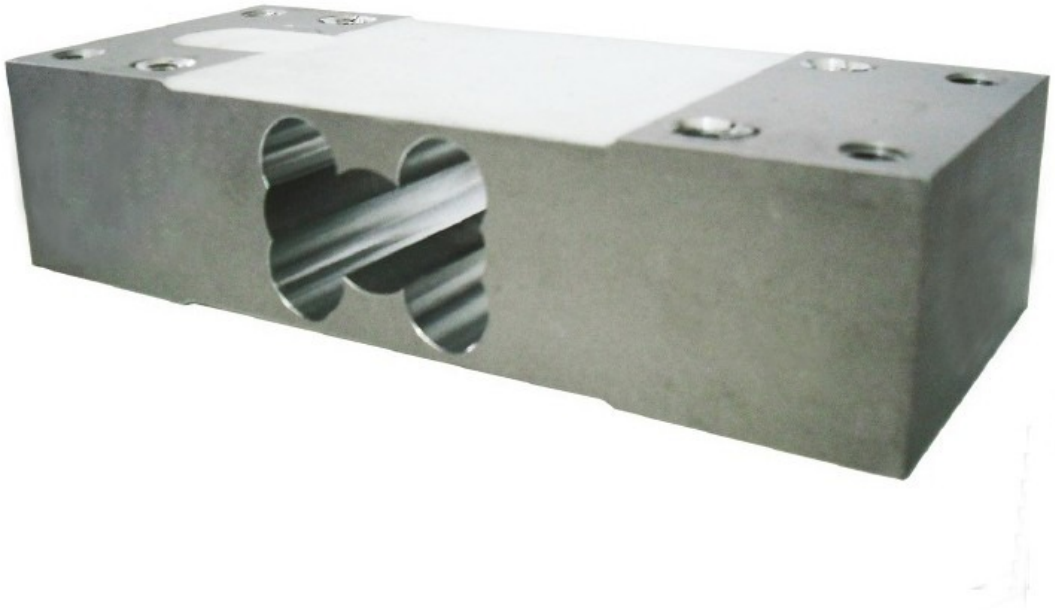

Figura 2.1: Célula de carga. Fonte: http://brasilcalibracao.com.br

Célula de carga é um elemento transdutor de força. Existem vários tipos de célula de carga, que se distinguem uns dos outros com base em seu princípio de funcionamento. As mais comuns são com extensômetro de folha, de corda vibrante e de carga hidráulica[17].

Células de carga podem ser fabricadas em aço inox, carbono ou alumínio. O funcionamento das células de carga se dá a partir da deformação de seu corpo, essa deformação modifica o valor de resistência do extensômetro a ela acoplado, e essa variação de resistência pode ser medida e tratada por um circuito eletrônico, identificando o valor medido[18].

A figura2.2 mostra o esquemático de uma célula de carga, através deste esquemático é possível visualizar o funcionamento do elemento, que se dá através da deformação do furo central da peça.

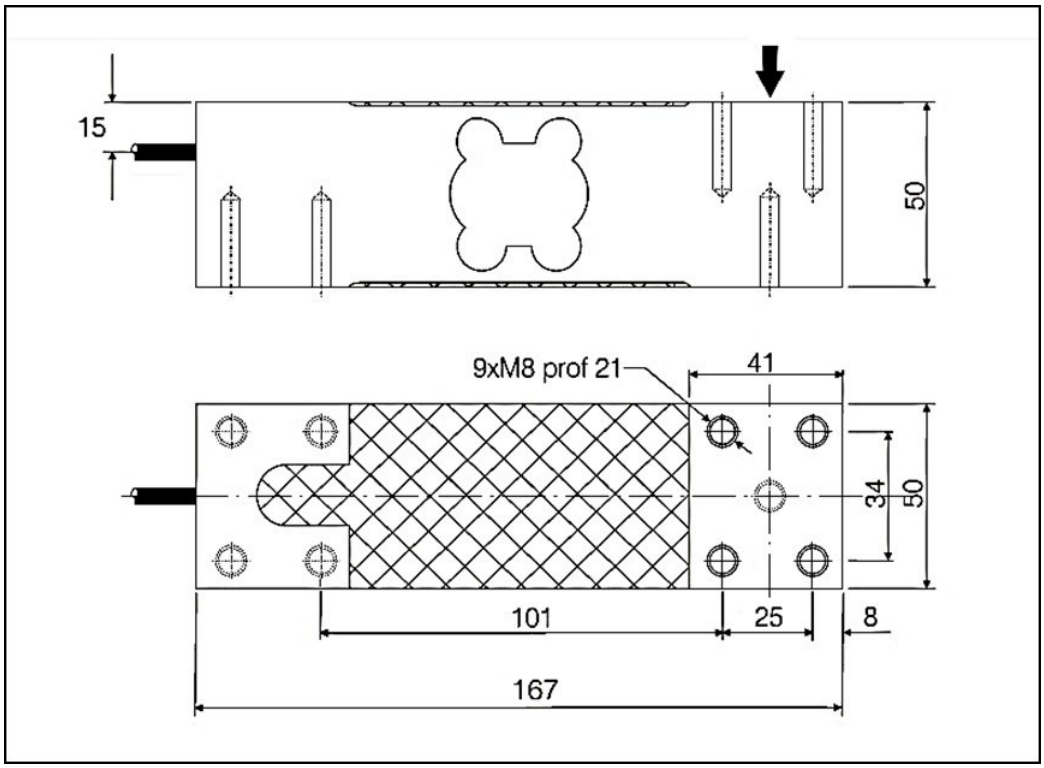

Figura 2.2: Desenho técnico. Fonte: http://www.iwm-brasil.com.br/

2.3.2 Circuitos eletrônicos

Condicionadores de Sinais

Medir grandezas físicas é um grande desafio no campo da eletrônica, pois raramente a medição dessas grandezas se dará de forma fácil de ser interpretada pelo sistema central de processamento. Para que os dados físicos possam ser interpretados, é necessária a realização de um condicionamento, que converte os valores medidos em dados que facilitam o processamento do microcontrolador[19].

Hoje em dia, existem no mercado vários sensores e transdutores que convertem as mais variadas grandezas físicas em sinais elétricos. Porém, mesmo utilizando medidores comerciais, existem problemas quanto à linearidade das medições, amplitude do sinal elétrico gerado e seu tipo, que pode ser corrente ou tensão.

Para que um sinal analógico possa ser interpretado por um circuito digital, é necessária que seja realizada uma conversão, transformando uma grandeza contínua, que pode ser corrente ou tensão, em uma medida discreta, ou não contínua. Realizar este procedimento gera um problema, a precisão da grandeza medida é comprometida, pois infinitos pontos (sinal analógico) são convertidos em um número finito de valores (sinal digital). Muito utilizados, os ADCs ou conversores A/D (Conversor Analógico Digital) são os responsáveis por receber o sinal analógico e transforma-lo em um sinal digital. Conversores A/D podem ser implementados das mais diversas formas, existindo várias arquiteturas diferentes para

a sua montagem.

Cada arquitetura apresenta vantagens e desvantagens, possuindo menor ou maior consumo de potência, que variam de acordo com as características requisitadas, como velocidade e precisão da conversão. Neste trabalho, não serão explicadas todas as topologias existentes no mercado atual de eletrônicos, mas, para efeito de conhecimento, algumas das arquiteturas mais difundidas nos dias atuais são as arquiteturas *Flash*, *pipeline*, aproximações sucessivas (SAR - *Sucessive Aproximation Registers*), integrador e sigma-delta[20].

A figura 2.3 demonstra a relação entre velocidade, resolução e consumo de potência de algumas arquiteturas. No gráfico apresentado, é possível perceber que não existe um conversor ideal, existe aquele que é mais adequado a uma aplicação, levando em consideração os requisitos de projeto [21].

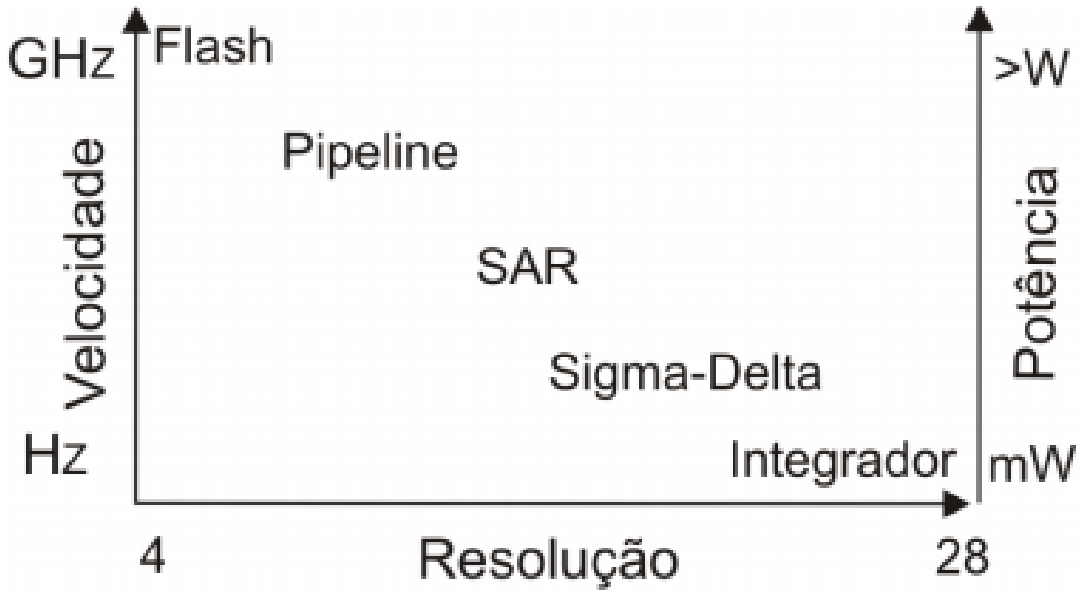

Figura 2.3: Relação entre parâmetros ADC. Fonte: https://www.embarcados.com.br

Além das características de velocidade, precisão e consumo energético, outro fator que deve ser levado em consideração na escolha de um ADC é a quantidade de componentes necessários para a sua implementação. Entre os elementos constituintes de um conversor A/D, os de maior impacto no custo e complexidade de construção são os conversores D/A, e os comparadores lógicos, que não estão presentes em todos os tipos [22].

A topologia de do conversor A/D por aproximações sucessivas é apresentada na figura 2.4. A topologia SAR, funciona comparando N vezes, sucessivamente, o valor de entrada com o valor da tensão de realimentação. A primeira comparação determina se o sinal é maior ou menor que a metade da tensão de entrada do conversor. A próxima comparação determina em qual quarto da variação se encontra a tensão de entrada e assim

sucessivamente, estreitando a faixa de comparação por um fator de dois[23].

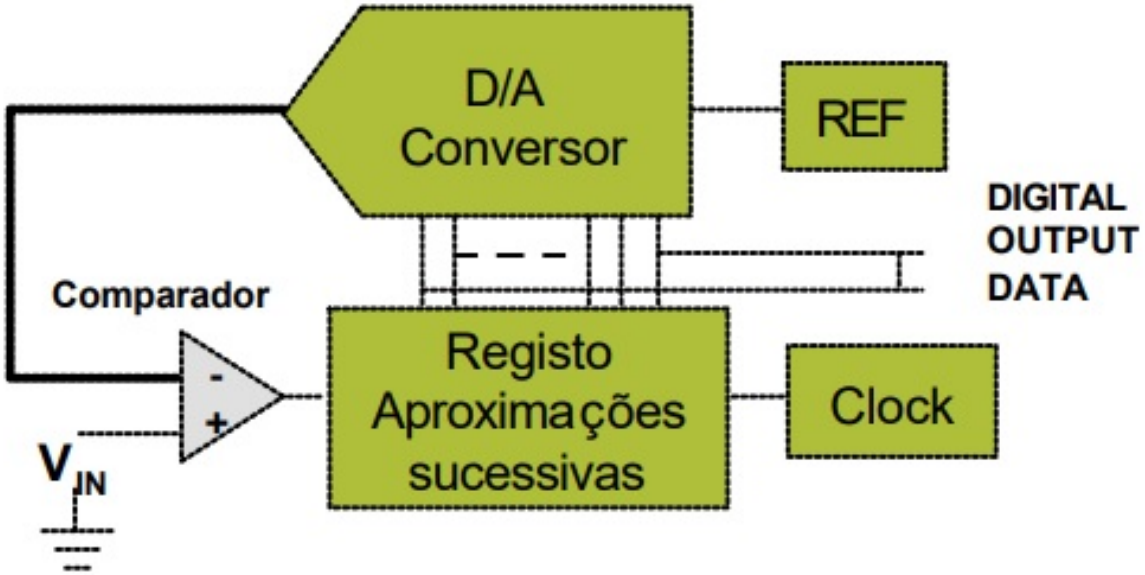

Figura 2.4: Conversor ADC SAR. Fonte: https://www.embarcados.com.br

Esta arquitetura exige um aumento exponencial de comparadores para obter uma maior resolução de medida. A figura2.5 mostra um esquemático da topologia.

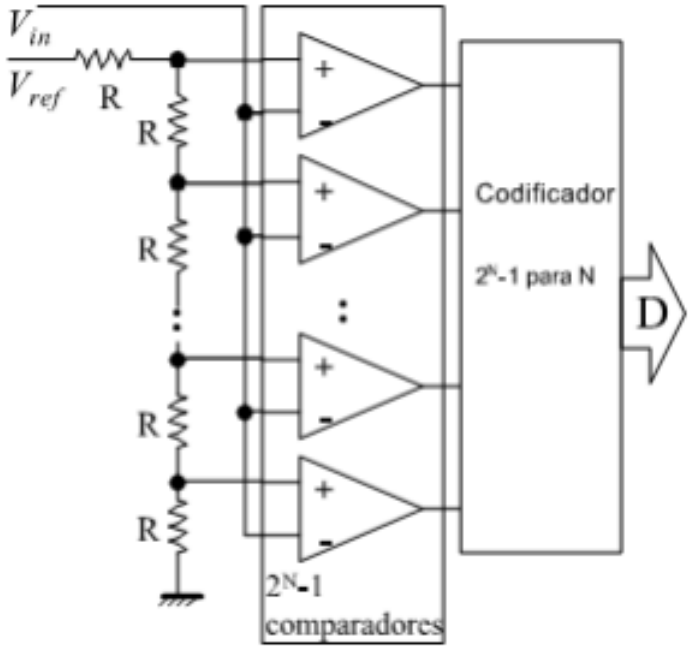

Figura 2.5: Arquitetura *flash*. Fonte: https://devzone.nordicsemi.com//

Filtros

Ao medir grandezas físicas, estamos sempre sujeitos a interferências de ruídos. Ruídos são distúrbios que alteram o sinal de medição, podem ter sua fonte em diversos fatores, desde a dinâmica do processo a ser mensurado ao ambiente em que está inserido, sofrendo variações de campos elétricos e magnéticos.

Medições possuem frequências características, normalmente de valor muito diferente dos ruídos captados junto ao sinal, que normalmente são de alta frequência. Frequências características de sinais, suas naturezas e teorias, não são o tema abordado neste trabalho, e por isso, não serão tratadas de maneira aprofundada.

A característica de possuir ordem de grandeza muito diferente à do sinal de interesse proporciona uma solução muito efetiva no tratamento dos ruídos, que é a atenuação da magnitude do sinal indesejado.

Para atenuar ruídos, são utilizados filtros, que tem como função alterar o sinal de entrada, atenuando o que é indesejado e não alterando o que se deseja medir, ou seja, um filtro visa manter todas as características do sinal desejado, retirando apenas o ruído[24].

Filtros podem ser classificados de várias maneiras. Alguns tipos comuns são:

- Passa-Baixa, o filtro atenua sinais de alta frequência e proporciona ganho unitário em baixas frequências.
- Passa-Alta, o filtro atenua sinais de baixa frequência e proporciona ganho unitário em altas frequências.
- Passa-Faixa, o filtro atenua sinais em uma determinada faixa e proporciona ganho unitário nas demais frequências.

Filtro passivo é aquele implementado apenas com elementos passivos, como resistores, indutores e capacitores. São utilizados pela facilidade de implementação dos circuitos e baixo custo dos componentes empregados em sua construção. Podem ser implementados em série com o sistema ou em paralelo (ou *shunt*). As duas formas possuem vantagens e desvantagens, a principal diferença entre elas é a potência que flui pelo filtro, que é maior na aplicação em série.

Filtro ativo é aquele implementado com elementos não passivos, que são constituídos por semicondutores, capazes de injetar sinais no processo, atenuando os ruídos presentes no sistema. Muito utilizados atualmente, possuem uma maior capacidade de compensação quando comparados aos filtros passivos. Também podem ser implementados em série ou paralelo com a carga.

Filtro híbrido é a combinação de filtros ativos e passivos. São uma outra forma utilizada na correção de sinais. Possuem as qualidades de ambas as implementações, ao custo

de se aumentar a complexidade do circuito a ser implementado [25].

As figuras 2.6 e 2.7 mostram duas topologias possíveis para um filtro passa baixa, sendo o primeiro ativo e o segundo passivo.

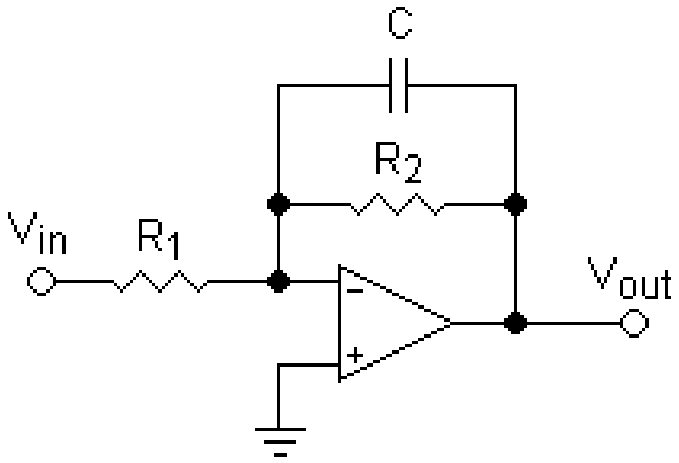

Figura 2.6: Filtro Passa-Baixa ativo. Fonte: https://sintoniafinadotcom.com

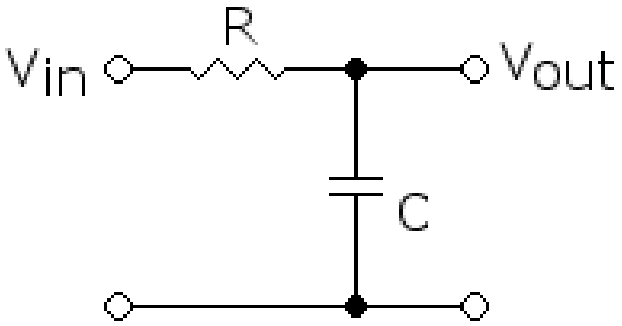

Figura 2.7: Filtro Passa-Baixa passivo. Fonte: https://sintoniafinadotcom.com

Como última forma de classificação, podemos separar filtros em dois grandes grupos, os filtros analógicos e os filtros digitais.

Filtros analógicos são aqueles que trabalham com o sinal contínuo, implementados de maneira física, de forma que não existe amostragem do sinal[26]. A figura2.8 mostra uma placa com um filtro analógico.

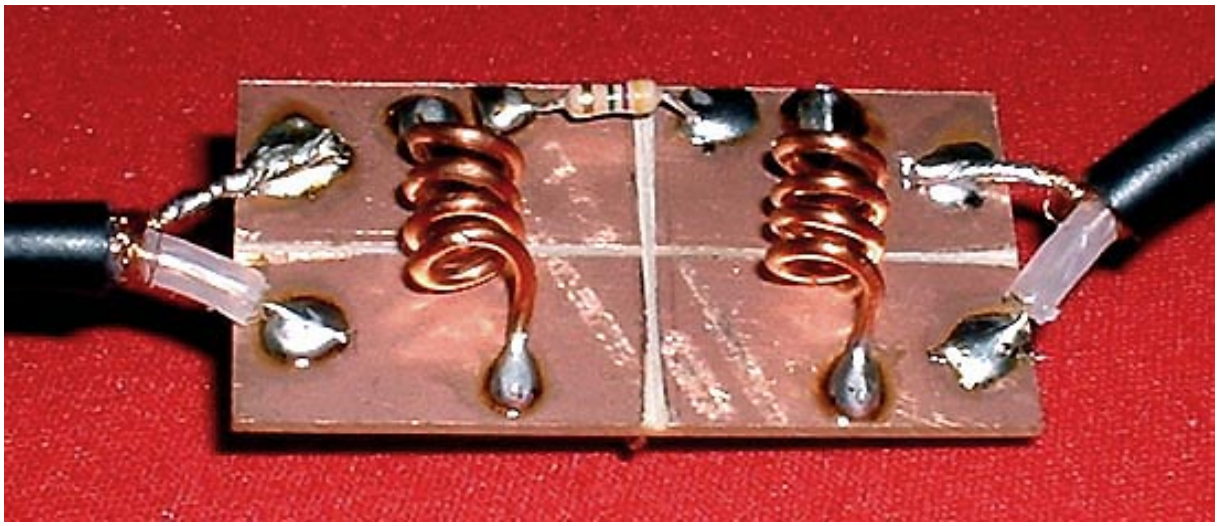

Figura 2.8: Filtro analógico RL. Fonte: http://www.qsl.net/g4wpw/l

Filtros digitais são aqueles que trabalham com sinais discretos, onde um sinal analógico, é amostrado e convertido em sinal digital, permitindo um processamento da informação em elementos digitais, como microprocessadores e microcontroladores[27].

Filtros digitais são implementados através de uma equação a diferenças feita no código de um microprocessador, por exemplo. Uma das abordagens para se projetar um filtro digital é discretizando a função de transferência obtida por técnicas para filtros analógicos[28]. A definição dos parâmetros do filtro será feita após a realização de testes que possibilitarão a compreensão das características dos sinais de interesse.

2.3.3 Microcontroladores

Para a criação e o gerenciamento das redes de comunicação, podem ser utilizados vários dispositivos, neste trabalho, iremos utilizar microcontroladores para realizar esta tarefa.

Microcontrolador é um circuito integrado constituído por uma CPU, memória e periféricos programáveis de entrada e saída [29].

Devido ao seu tamanho reduzido e pouco consumo energético, microcontroladores são muito utilizados em sistemas embarcados, que necessitam de grande autonomia de funcionamento.

Microprocessadores são construídos baseados em uma arquitetura, que é uma definição de como é construído o hardware daquele dispositivo. Algumas arquiteturas muito utilizadas atualmente são as arquiteturas CISC, RISC, ARM.

Os periféricos constituem, juntamente com a CPU e memória, o Hardware de um microcontrolador. Os periféricos são dispositivos de entrada/saída que auxiliam o funci-

onamento do sistema como um todo. Também são periféricos conversores e circuitos de comunicação com outros periféricos ou mesmo outros dispositivos.

Para programar um microcontrolador, é necessário que seja utilizada uma linguagem de programação, que agrupará funções e métodos que permitirão ao dispositivo ler e interpretar as instruções fornecidas pelo programador.

Existem atualmente diversas linguagens de programação, que podem ser divididas em Programação Orientada a Objeto e Programação Estruturada.

Programação orientada a objetos (POO) é um modelo de programação de software que se baseia na interação e composição de objetos. Normalmente, são baseadas em classes que agrupam características comuns em objetos e possuem como principal característica distintiva da programação estruturada os conceitos de Herança e Polimorfismo[30].

A programação estruturada é um modelo de programação linear, onde funções são definidas apenas para modularizar e facilitar a compreensão de códigos, não sendo necessárias a implementações de classes que agrupem informações. Podem ser feitas structs, que são estruturas de dados semelhantes a classes, com a finalidade de definir tipos de variáveis a serem utilizadas no código[31].

Para a programação de microcontroladores, é mais utilizada a abordagem de programação estruturada, mais especificamente a linguagem C, que permite grande controle de registradores e endereços de memória, através de ponteiros e deslocadores de bits.

2.3.4 Redes de comunicação

Introdução Bluetooth 4.0

O *Bluetooth Low Energy* (BLE), ou *Bluetooth* 4.0 foi concebido com o intuito de melhorar eficiência energética em aplicações móveis. O BLE não veio para acabar com o *Bluetooth* Clássico (BR/EDR), ele serve como uma nova opção para a comunicação em rede.

O *Bluetooth* é recomendado para aplicações que precisam fazer grandes transferências de dados, como falar ao telefone, podendo alcançar altas taxas de transmissão de dados, ao custo de um maior consumo energético e maior preço para aquisição de componentes.

O BLE, tem como objetivo aplicações que trocam pequenas quantidades de dados, havendo periodicidade ou não na transferência desses dados.

O *Bluetooth* possui uma limitação de 7 dispositivos por rede, tornando-se mais comum a topologia *peer-to-peer*. O BLE suporta uma rede com muito mais dispositivos, devido ao maior número de bits para endereçamento disponíveis no protocolo, dessa forma, ele permite as topologias *peer-to-peer*, estrela e rede *mesh*. Por isso, o BLE possui mais atrativos para aplicações em ambientes industriais e comerciais[32].

Topologia da rede

Dispositivos BLE podem assumir duas diferentes funções, dispositivos centrais ou dispositivos periféricos.

Dispositivos centrais são geralmente aparelhos com mais capacidade de processamento e memória, como computadores e celulares. Dispositivos periféricos comumente possuem baixa potência, como sensores, e se conectam ao dispositivo central.

Dispositivos BLE podem enviar dois tipos de dados, *Advertising Packets* e *Scan Response Data*.

Os *Advertising Packets* são constantemente enviados por um dispositivo periférico para serem recebidos por outros dispositivos, quando uns dispositivos recebem esses pacotes, eles podem solicitar dados adicionais através de *Scan Response Data*.

Um dispositivo BLE pode comunicar com dispositivos próximos de duas maneiras, *Broadcasting* e *Connections*[33].

Broadcasting é quando o dispositivo envia dados para todos os equipamentos que o escutam. Quando o BLE está operando em *Broadcasting*, são definidos dois papeis, *Broadcaster* e *Observer*. O *Broadcaster* envia *Advertising Packets* não conectáveis periodicamente para qualquer dispositivo que queira recebe-los. Quando o *Observer* recebe um *advertising packet*, ele poderá solicitar, enviando um *scan response data*, solicitando conexão ou dados.

Apenas quando o BLE está operando em regime de *Broadcasting* é possível que um dispositivo possa transmitir dados para mais de um outro dispositivo. A figura 2.9 apresenta um esquema da topologia *Broadcasting*.

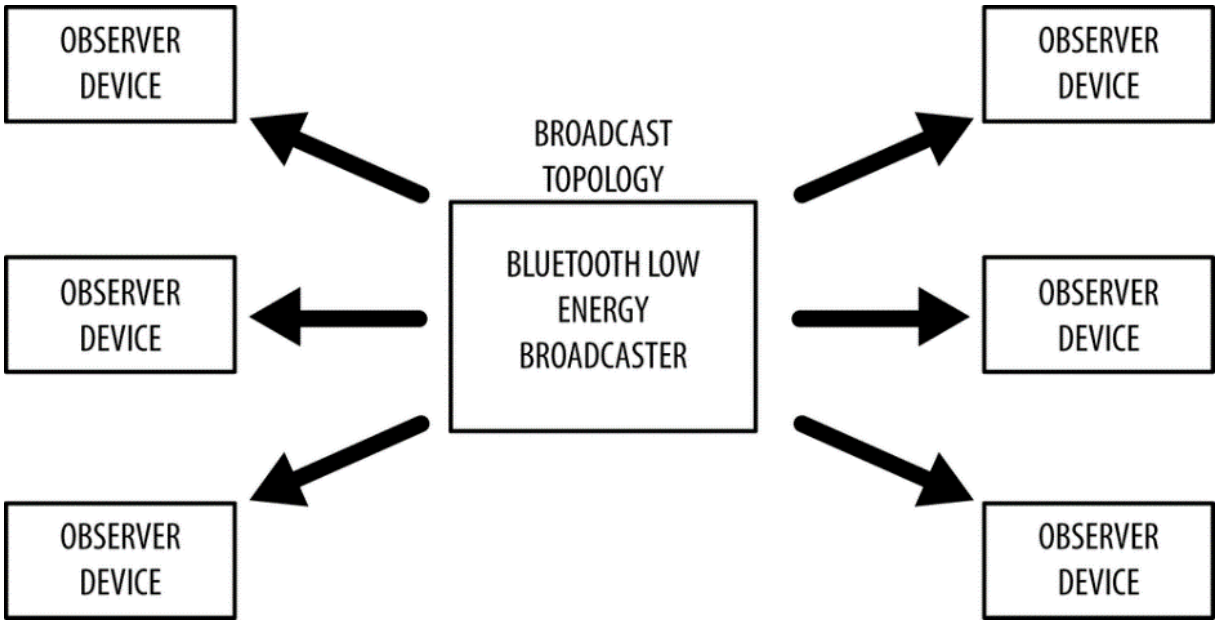

Figura 2.9: Topologia BLE em *Broadcasting*. Fonte: https://www.bluetooth.com

Uma *Connection* é uma troca de dados de maneira periódica e continua entre dois dispositivos. O mestre (dispositivo central) verifica periodicamente por *conectable advertising packets*, para que, quando solicitado, inicie uma conexão. Após estabelecida a conexão, o dispositivo central gerencia a troca de informações. O escravo (dispositivo periférico), envia *advertising packets* periodicamente e aceita conexões de entrada, quando a conexão for estabelecida, o escravo troca dados regularmente com o mestre.

A vantagem de se utilizar dispositivos BLE nesta topologia é a possibilidade de dois dispositivos iniciarem um *connection event*, trocarem dados, e logo após entra em suspensão até o próximo *connection event*. A figura 2.10 apresenta um esquema da topologia *Connection* [33].

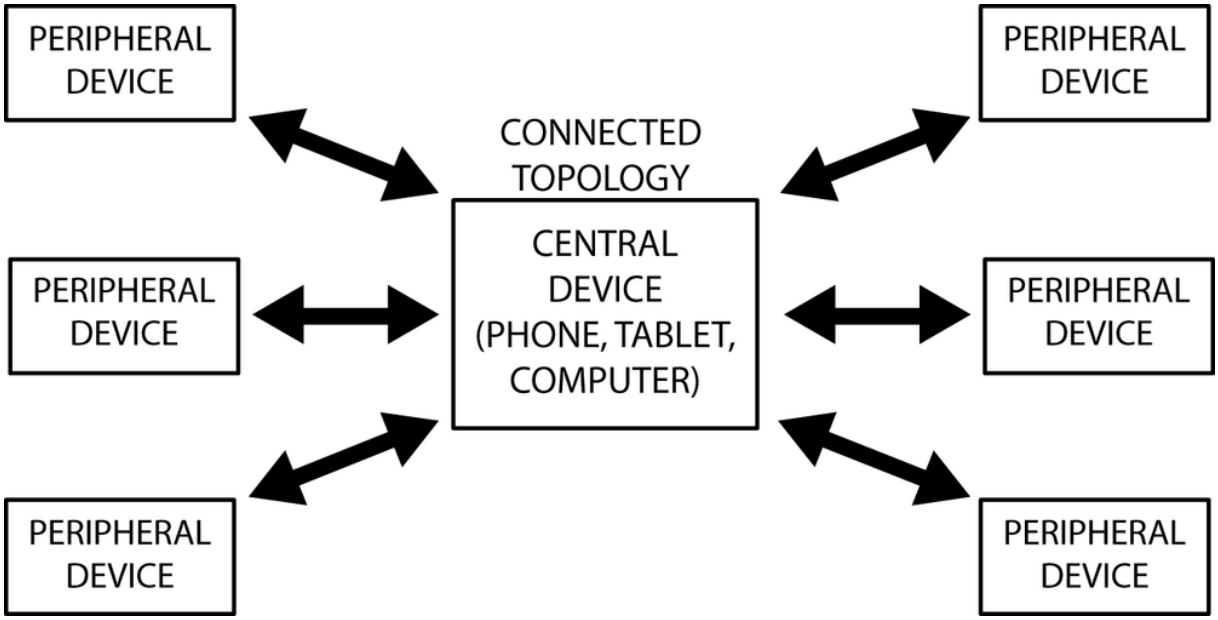

Figura 2.10: Topologia BLE em *Connection*. Fonte: https://www.bluetooth.com

Blocos de construção do BLE

O BLE é organizado em três grandes blocos de construção: *Application, Host e Controller*. O bloco *Application* é o aplicativo que faz a interface com o *stack* do protocolo *Bluetooth*. O bloco *Controller* é responsável pela comunicação entre as camadas inferiores da arquitetura. O *Host* é quem se comunica com o módulo BLE, utilizando o HCI (*Interface Controller Host*), essa interface torna possível a comunicação de vários hosts com o controlador.

A Aplicação é executada pelo microcontrolador que interage com um dispositivo de conectividade que possuí *Host* e *Controller*[34].

2.3.5 Aplicações WEB

Com a vasta gama de plataformas utilizadas hoje em dia, um desafio para a evolução dos softwares é estar presente em todos eles. [35]

As aplicações nativas são softwares desenhados para uma arquitetura em específico e tem a limitação de não ser portátil, por exemplo, um programa projetado para Android não é possível ser executado em um sistema operacional do Windows ou da Apple. Desta forma, as aplicações web surgem como uma alternativa aos aplicativos nativos como um software capaz de rodar de em todas os sistemas que possam acessar a internet.[36]

Estas aplicações têm a vantagens de ser programado somente uma vez, ser escalável e rodar em todos os sistemas, mas as desvantagens também são inúmeras e é necessário levá-las em consideração na hora de escolher a forma que o aplicativo será desenvolvido. Desvantagens estas são a limitação da performance, ou seja, os botões, telas e outros componentes não respondem de forma instantânea como em uma aplicação nativa, portanto a experiência do usuário fica comprometida. Outra desvantagem é não ser possível utilizar recursos de celulares como câmeras, acelerômetros, controle de processos entre outros. [37]

Para superar a limitação de performance e aprimorar a experiência do usuário foi desenvolvido uma técnica chamada Single Page Aplication. Técnica esta que permite que toda aplicação seja executada sem a necessidade de atualizar a página que foi carregada, desta forma, o usuário tem a experiência de um software mais fluido e mais rápido. A figura 2.11 apresenta uma aplicação web sendo executada em diferentes plataformas, como notebook, celular com a plataforma IOS e tablet com Android, o sistema se adapta ao tamanho da tela para melhorar a experiência do usuário com a aplicação. [38]

Figura 2.11: Aplicação *web* multi plataformas. Fonte: http://ww.sencha.com

2.3.6 Padrões de software

Criado na década de 70, o padrão de projeto foi utilizado na arquitetura por Christopher Alexander. A ideia surgiu depois de perceber que os produtos criados na época não satisfaziam os usuários e eram de má qualidade. Após, Christopher criou mais de 200 padrões de projetos bem documentados que reforçavam a comunicação de integrantes das equipes de forma direta ou indireta e que elevavam a qualidade do projeto. [39]

Utilizando o mesmo conceito, padrões de projetos são ferramentas utilizadas para programadores que resolvem problemas recorrentes de desenvolvimento de software, dentro de um contexto próprio de aplicações. Estes padrões são descritos na literatura de forma estruturada e específica e geralmente seguem a seguinte regra: Nome do padrão, o problema que este resolve, o contexto em que está envolvido e outros padrões relacionados. [40]

Dentro deste conceito existe o padrão MVC. Bastante utilizado em aplicações móveis, este padrão divide a aplicação em três camadas, sendo o Model, Modelo, responsável por dar características aos objetos que compõe o software, o View, camada que é responsável pela interação com o usuário, a inteface gráfica e o controller, camada responsável por atribuir a ação que deve ser tomada a partir de cada interação e atualizar os dados pertinentes na view. [41] O MVC tem o objetivo de evitar a reescrita de códigos, pois para

programas que tem a mesma função, mas são exibidos em locais diferentes, por exemplo, um aplicativo Android e um web, podem compartilhar do mesmo modelo e controlador, alterando somente a visão. A figura 2.12 mostra um exemplo de uma aplicação simples que ignora o controller, mas mostra um mesmo modelo sendo utilizado por três modos diferentes, uma tabela, um gráfico de barras e um gráfico de pizza. [1]

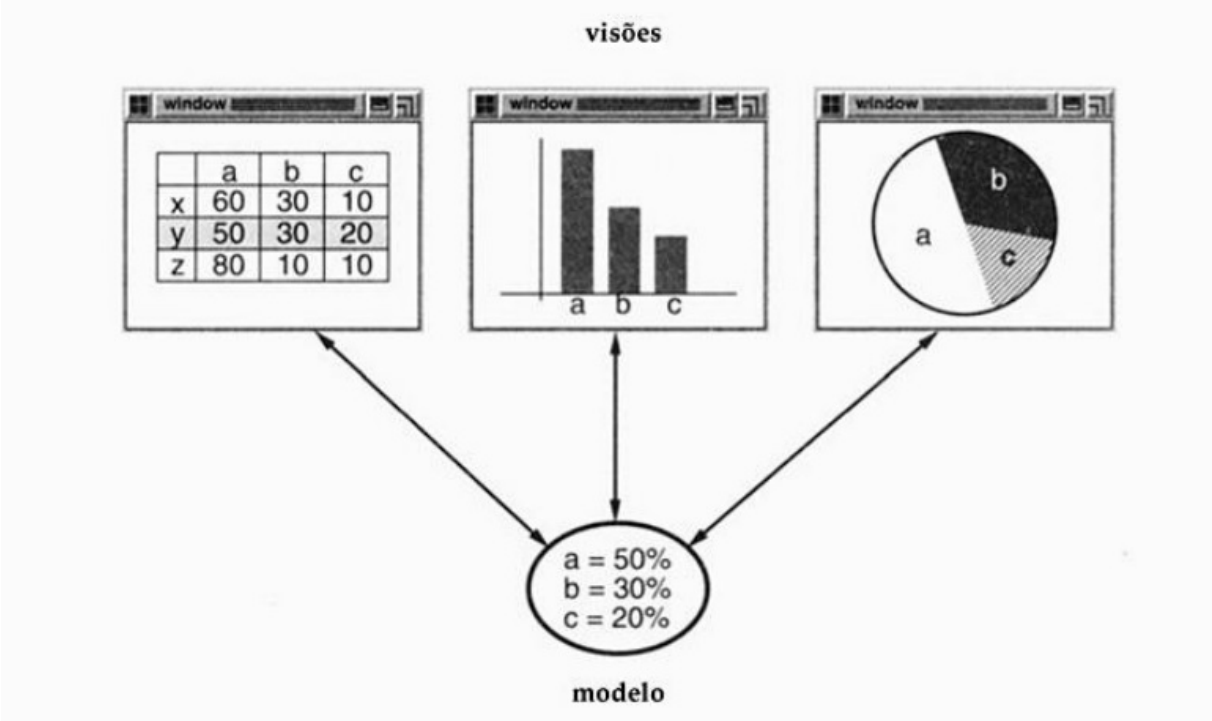

Figura 2.12: Exemplo de MVC. Fonte: Padrões de Projetos: Soluções Reutilizáveis [1]

Banco de dados

Banco de dados são estruturas projetadas para armazenar informações de forma compartimentalizada, de forma que a consultas a estas informações sejam feitas rapidamente e a medida que este banco ganha dados o tempo destas consultas não sejam prejudicadas.[42] Estas são divididas em tabelas, colunas e linhas onde são inseridos os valores em cada célula, a figura 2.13 demostra a construção da mesma, onde a tabela contém linhas, chamadas de registros, dividas por colunas chamadas de campos em que a célula é um valor de dados. [43]

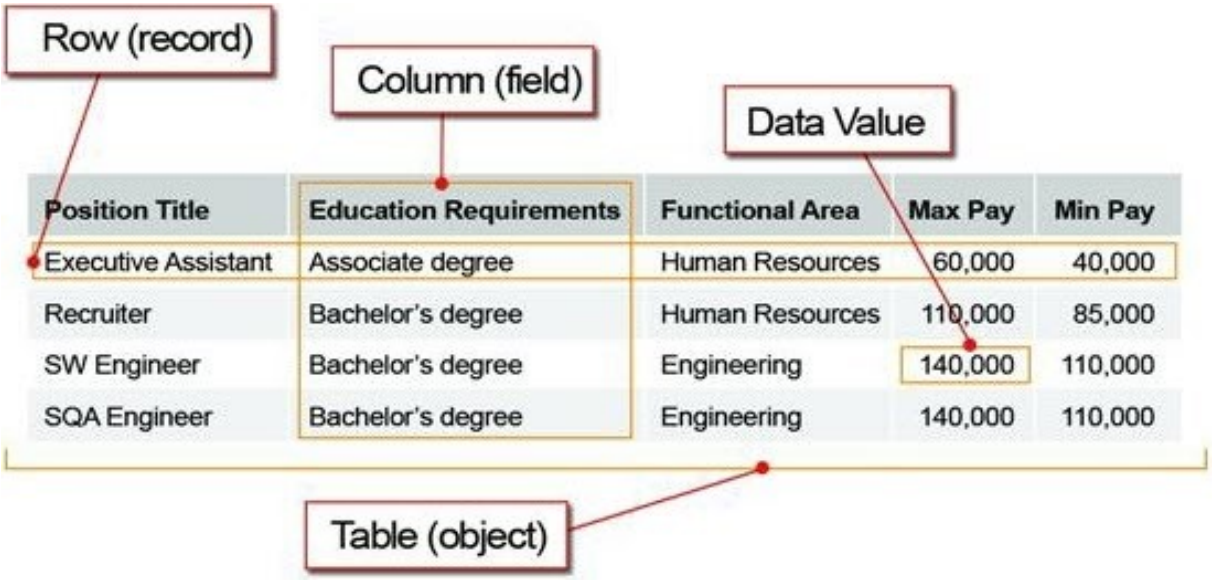

Figura 2.13: Estrutura de um banco de dados Fonte: http://www.developerforce.com/

Para interação com esta estrutura foi padronizado a linguagem de programação SQL, Structured Query Language. O SQL é uma linguagem criada para manipular, armazenar e obter dados de uma base de dados. A mesma é composta por comandos que formam um filtro para obter os dados desejados. A tabela abaixo apresenta os comandos mais comuns de SQL e sua respectiva função. [43]

Tabela 2.1: Comandos SQL

Comando	Descrição
Desenvolvimento eficiência alimentar gado	Seleciona uma tabela para aplicar um determinado filtro, por exemplo, SELECT * FROM NOMETABELA. Este comando seleciona todos os dados da tabela NOMETABELA
INSERT, INTO, VALUES	Insere um determinado valor em uma tabela seleciona, por exemplo, INSERT INTO NOMETAELA (coluna1, coluna2) VALUES (valor1, valor2)
UPDATE, SET	O comando UPDATE atualiza o valor que está em uma determina célula. No exemplo a seguir ele atualiza os campos coluna1 e coluna2 quando o id da linha é igual a 1 UPDATE NOMETABELA SET coluna1=ex1, coluna2=ex2 WHERE id=1
DELETE,FROM	O comando DELETE apaga a linha de uma determinada tabela, sempre deve ser usada com WHERE. Exemplo, DELETE FROM NOMETABELA WHERE id=1
WHERE	É um filtro que é utilizado com os comandos acima. Sua função é procurar a linha onde a determinada coluna, ou campo, tem o valor especificado.
BETWEEN	É um outro filtro utilizado com o WHERE, este comando seleciona valores dentro de uma faixa. Por exemplo. SELECT * FROM TABELA WHERE id BETWEEN 1 and 2
ORDER BY	Ordena a exibição dos dados de acordo com a coluna desejada, por exemplo, select * from NOMETABELA order by id, os dados serão exibidos de forma crescente. Caso queria exibir em ordem decrescente utiliza-se o DESC ao final do comando.

2.3.7 Linguagens de programação orientada a objetos

A programação orientada a objetos é um padrão de desenvolvimento que tem como objetivo modelar os processos em formas de objetos reais com identidade, características e ações. Este conceito se chama abstração. [44]

Exemplificando, temos uma classe, também denominado de classe, cuja identidade, nome da mesma, será cachorro, dentro da classe existe as características, que são variáveis, como nome, sexo e idade, temos também as ações, que são métodos que retornam algum dado, como latir e dormir.

```
1  Classe Cachorro(){
2  variavel nome
3  variavel sexo
4  variavel idade
5  Latir(){
6          Retorna "au au";
7  }
8  Dormir(){
9  Retorna "ZZZzzz"
10 }
11 }
```

Portanto, toda vez que criamos um objeto chamado cachorro, estes compartilharão das mesmas ações e terão nome, idade e sexo.

Outros padrões da programação orientada a objetos são encapsulamento, herança e polimorfismo. [30] Encapsulamento é a propriedade da classe de esconder características ou métodos de acessos externos, por exemplo, a classe cachorro acima poderia declarar que o método latir é do tipo privado, portanto, quando o objeto cachorro é criado, esta função não poderá ser chamada externamente, seria necessário um outro método público chamar a função latir internamente. Herança é a propriedade da programação orientada a objetos de incluir características e métodos em uma classe sem ter que reescrevê-las. Isto poupa o programador de repetir código. [30]

Exemplificando, o cachorro é do tipo mamífero, logo, poderia ser criado uma classe "mamífero" que as variáveis são nome, idade, sexo e os métodos são dormir, porém nem todos os mamíferos latem.

```
1  Classe mamifero(){
2  Variavel nome
3  Variavel sexo
4  Variavel idade
5  Dormir(){
6          Retorna "ZZZZzzzz"
7  }
8  }
9  Classe cachorro() herda mamifero {
10         Latir(){
11                 Retorna "au au"
12 }
13 }
```

Com o exemplo acima, para criar o objeto cachorro, ainda seria necessário definir nome, idade e sexo e também poderia ser chamado a função dormir, pois todos os mamíferos dormem. O polimorfismo é outra característica que é inspirado em objetos reais implementada em uma linguagem de programação. Esta propriedade permite que uma função possa ser reescrita em uma classe filha. Exemplificando, um objeto eletrodoméstico tem a função ligar, porém geladeira e o fogão, que herdou características e métodos de eletrodoméstico, tem maneiras de diferentes de executar esta ação, logo a função ligar deve ser reescrita de acordo com a nova necessidade.

A principal vantagem de utilizar a programação orientada a objetos é que uma vez modelado um sistema com as características citadas acima, este trecho de código pode ser replicado em várias partes do programa sem a necessidade de reescrevê-lo. Assim é possível criar bibliotecas com códigos mais claros para permitir a reutilização.

2.3.8 FrameWorks

Frameworks são um conjunto de códigos que propõe resolver um problema de um certo domínio de forma genérica. Geralmente são feitos sob a licença open-source, licença que permite que outras pessoas utilizem o código sem pagar por eles, são desenvolvidos em uma determinada linguagem e provém uma série de benefícios como redução de bugs, facilidade de aprendizado e padronização de código, pois os frameworks são amplamente testados e documentados pelas pessoas que o utilizam e contribuem para o seu desenvolvimento. [45]

Alguns exemplos de frameworks são:

BootStrap: desenvolvido em JavaScript, CSS e HTML, auxilia o desenvolvimento de sites e aplicações web fornecendo componentes como botões, tabelas, menus entre outros que ajustam seu tamanho de acordo com a tela que a mesma está sendo visualizada, assim é possível que o mesmo site seja visto de forma diferente se acessado pelo celular ou pelo notebook.

PhoneGap: Framework para desenvolvimento de aplicações mobile híbridas, que rodam em sistemas como android, ios e Windows phone, utilizando aplicações construídos em HTML, JAVASCRIPT e CSS com acesso a funcionalidades nativas como câmeras, GPS, notificações.

2.3.9 Webservices

Webservices são serviços que permitem a comunicação entre dispositivos eletrônicos programados de formas distintas. A comunicação, por sua vez, é padronizada e segue algumas regras para ser implantada e entendida por qualquer sistema. [46] Dentre estes padrões temos o SOAP, Simple Object Access Protocol. Seu funcionamento parte do princípio de transmissão de um arquivo XML via protocolo de rede SOAP HTTP. A estrutura do XML, chamado de envelope, é escrita em uma linguagem de programação WSDL, semelhante ao HTML, tem um cabeçalho em que são definidos dados referentes a configuração de entrega da mensagem, autenticação ou regras de autorização ou contexto das transações e um corpo com as informações a serem enviadas. A figura 2.14 apresenta a estrutura do envelope SOAP. [47]

Um exemplo de uma transmissão de um protocolo SOAP HTTP:

```
1 POST /InStock HTTP/1.1
2 Host: www.example.org
3 Content-Type: application/soap+xml; charset=utf-8
4 Content-Length: nnn
5
6 <?xml version="1.0"?>
7 <soap:Envelope
8 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

```
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

A resposta a esta requisição seria:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

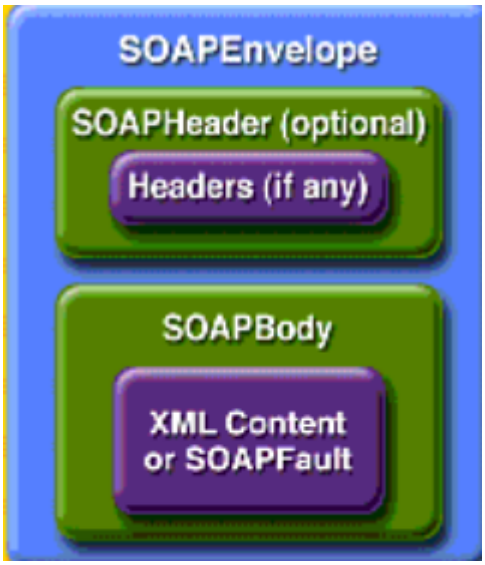

Figura 2.14: Esturutura SOAP Adaptado Fonte:[2]

Outro padrão de webservice é o REST. Diferente do SOAP que é um protocolo de comunicação, este utiliza somente o protocolo HTTP para transmitir informações como

GET, POST, PUT e DELETE. Seu funcionamento consiste em enviar uma requisição para o servidor, através de uma URI com informações relativas ao método e variáveis no corpo da mensagem. Um exemplo de requisição utilizando HTTP:

```
POST /item HTTP/1.1
Host: 189.123.255.239
Content-Type: text/plain
Content-Length: 200
```

Caso a requisição ocorra com sucesso, será retornado ao cliente:

```
200 OK
Content-Type: text/plain
Content-Length: 200
```

Ou

```
400 Bad Request
Content-Length: 0
```

Em caso de falha.

Capítulo 3

Desenvolvimento

Neste capítulo são apresentadas as especificações de projeto determinadas pela dupla, levando-se em consideração as pesquisas realizadas no mercado, e os procedimentos adotados para a realização do produto, utilizando os conceitos de engenharia Mecatrônica abordados na fundamentação teórica deste trabalho 2.3.

3.1 Sensor

Foram abordadas na fundamentação teórica2.3 alguns tipos de célula de carga, que possuem funcionamento diferente entre si. Visto que durante a graduação de engenharia Mecatrônica é lecionada a disciplina de Instrumentação Industrial, e nesta disciplina, é dado um enfoque maior em sistemas de medição baseados em extensômetros, será utilizada uma célula de carga fundamentada neste princípio de funcionamento.

Ainda não foi determinado o modelo ou fabricante da célula utilizada na confecção do protótipo final, visto que os preços variam de acordo com a época e quantidade produtos adquiridos, por isso, o modelo será determinado durante a execução da disciplina de TCC II.

Definido o princípio de funcionamento do elemento sensor, foi escolhido o circuito de uma Ponte de *Wheatstone* para funcionar como condicionador de sinal. Nesta topologia, podemos alcançar valor de tensão igual a zero, o que é uma vantagem para medição de peso, visto que poderá ser determinada uma faixa de operação, aumentando assim a precisão dos dados coletados pela balança. A figura 3.1 mostra uma ponte de *wheatstone*.

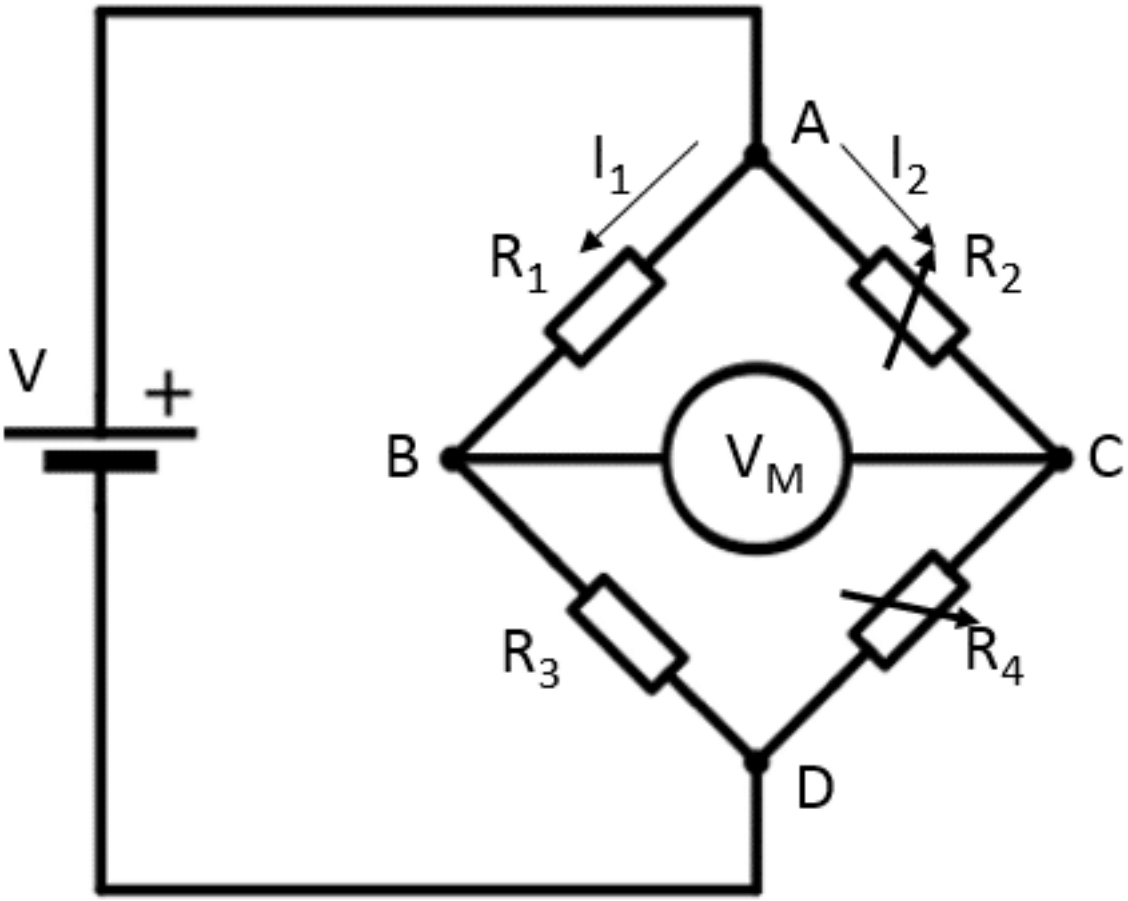

Figura 3.1: Ponte de Wheatstone Fonte: https://embarcados.com

Para que a ponte funcione de maneira correta, deve ser levado em consideração no projeto a variação de resistência devido à temperatura, e os valores possíveis para a fonte de alimentação do circuito.

O projeto da ponte será feito para que a tensão de saída não exceda 3,3 *volts*, tendo assim uma margem de segurança para se atingir a tensão máxima suportada na entrada do microcontrolador.

Para que se obtenha valor de tensão igual a zero, a ponde deve ser projetada para ter valores de resistência iguais quando o resistor da célula de carga se encontrar em seu valor mínimo. Para isso, é necessário que se tenha em mãos a célula de carga a ser utilizada, para que sejam realizadas as medições e calibrações.

3.2 Microcontrolador

Para a determinação do microcontrolador a ser utilizado, levou-se em consideração os seguintes requisitos para projeto:

- Compatibilidade com os protocolos de comunicação mais utilizados atualmente, sem

que seja necessária a utilização de outros módulos, facilitando a implementação, tornando-a mais barata e melhor otimizada.

- Capacidade de processamento suficiente para gerenciar conexões, implementar filtro digital, processar sinais, consumir *webservices*, e salvar arquivos em cartões de memória.

- Baixo consumo energético, tanto para processamento de sinais quanto para gerenciamento da rede.

Após pesquisar por microcontroladores comerciais, e tendo como base o conhecimento adquirido ao longo do curso de Engenharia Mecatrônica, foram selecionados três microcontroladores para serem analisados de forma mais profunda, são eles:

- ATmega328p3.2
- ESP-WROOM-32D3.3
- nRF528323.4

Figura 3.2: Microcontrolador ATmega328p Fonte: https://br.mouser.com/

Figura 3.3: Microcontrolador ESP-WROOM-32 Fonte: https://br.mouser.com/

Figura 3.4: Microcontrolador nRF52832 Fonte: Próprio autor.

Para realizar a escolha do microcontrolador, foi montada a tabela 3.1. As informações presentes na tabela foram extraídas dos *datasheets* de cada produto[48][49][50]. Os preços podem variar de acordo com a quantidade de componentes comprados.

Tabela 3.1: Comparativo *ATmega328p*,*ESP-WROOM-32D* e *nRF52832*

Microcontrolador	ATmega328p	ESP-WROOM-32D	nRF52832
Necessita de módulos para comunicação sem fio	Sim	Compatível com BLE e WiFi	Compatível com BLE
Número de núcleos	1	2	1
Velocidade do processador	20MHz	240MHz	64MHz
Memória *flash*	32KB	448KB	512KB
Memória RAM	2KB	520KB	64KB
Consumo mínimo de corrente	0.3mA	80mA	1.9uA
Tensão de alimentação	4.5 - 5.5V	2.7 - 3.6V	1.7 - 3.6V
Preço	U$ 2,50	U$ 10,00	U$ 4,30

A partir da análise da tabela 3.1, decidiu-se pela utilização do nRF52832 no desenvolvimento do trabalho. Além da alta eficiência energética do microcontrolador em questão, o fato de possuir um maior suporte para implementação de redes sem fio foi fator fundamental na escolha do mesmo

3.2.1 Hardware

O microcontrolador escolhido, nRF52832 possui algumas características úteis ao produto, além das citadas na tabela 3.1. Graças ao baixo consumo energético, alto poder de processamento e facilidade para implementação de redes BLE, o nRF é muito utilizado em aplicações embarcadas, o que o torna uma boa escolha para o trabalho aqui proposto. O nRF conta com 32 portas configuráveis como entrada ou saída, sensor de temperatura interno, 4 canais PWM, 3 SPIs, 2 I2Cs, 3 RTCs, 1 UART e 1 ADC SAR de 12 bits com 8 canais.

Os canais SPI serão utilizados, assim como os canais I2C, para estabelecer a comunicação do microcontrolador com dispositivos periféricos, através de redes físicas. É de extrema importância que a balança seja confiável, por este motivo, sistemas redundantes serão utilizados, como por exemplo um cartão de memória, que salvará os dados que são enviados para a nuvem.

A UART(*Universal Asynchronous Receiver/Transmitter*), será utilizada para estabelecer comunicação serial, configurada via *firmware*, com periféricos que utilizem protocolos de comunicação serial.

O conversor ADC poderá ser utilizado para facilitar a comunicação entre a célula de carga e o microcontrolador responsável pelo tratamento de dados provenientes da pesagem.

3.2.2 Filtro

Como abordado na fundamentação teórica 2.3, filtros podem ser implementados em circuitos eletrônicos ou em códigos, através de filtros digitais. Para o desenvolvimento da balança, o grupo optou pela utilização de filtros digitais.

A escolha por filtro digital teve grande influência de dois aspectos, o primeiro, o custo para a implementação e o segundo, a facilidade de ajustes no filtro digital.

Pelo fato de não necessitar de componentes eletrônicos, o filtro digital representa uma diminuição no preço do produto, pois, o microcontrolador responsável por tratar os sinais provenientes da balança possui capacidade de processamento maior que a necessária para filtragem de sinais, o que significa que não será necessária a aquisição de outro componente.

Filtros digitais são implementados através de equações a diferenças, o que significa que são linhas de código rodando na rotina do *firmware*. Graças a essa característica, esses filtros permitem inúmeros ajustes, sem que seja necessária a mudança de componente, valores de resistores ou alterações do gênero.

Para que seja projetado o filtro digital, é necessário conhecer a natureza dos sinais provenientes da medição, como faixa de frequência e frequência do ruído.

Um dos intuitos de se construir um filtro para a balança, é acelerar a dinâmica da medição, permitindo que o peso seja aferido de maneira mais rápida. Para isso, são necessários testes e levantamento de modelos, para que possam ser gerados os parâmetros do sistema.

O levantamento de dados e estudos de sinais da célula de carga serão realizados durante a execução de TCC II, e por isso, o projeto do filtro não será abordado neste relatório.

3.2.3 Comunicação

Para a escolha do protocolo de comunicação, foram analisados os seguintes requisitos de projeto:

- Baixo consumo energético para a utilização do protocolo
- Grande número de dispositivos conectados a uma mesma rede
- Compatibilidade com dispositivos utilizados pelo usuário (*tablets, smartphones* e *notebooks*, para possível interação *off-line*

Levando-se em consideração os requisitos de projeto, e o fato de possuir uma grande comunidade de desenvolvedores, o que auxilia na implementação de produtos, foi escolhido como protocolo a ser utilizado neste trabalho o BLE. Fato este que teve grande influência também pelo fato de o nRF ter seu foco principal de aplicação neste tipo de rede.

O BLE permite que uma balança seja ao mesmo tempo periférico e gerador de rede, o que facilita a implementação do sistema no campo.

Foi determinado que os valores obtidos em cada medição de peso, serão disseminados na rede através de mensagens de *advertising*, que serão geradas por cada balança. Desta forma, o microcontrolador poderá permanecer em *sleep mode* durante a maior parte do tempo, poupando energia e tornando o sistema mais otimizado.

Após implementado o filtro e o banco de dados, será determinada a necessidade ou não de uma central, a depender da quantidade de processamento e memória empregadas na utilização do *webservice*.

3.2.4 Funcionamento

Com base nas funções do produto já especificadas, foi determinada uma base de funcionamento para os microcontroladores, com o intuito de otimizar o consumo energético do produto.

Como somente serão realizadas medições perante a presença de animais, os nRF permanecerão em *sleep mode* até que seja gerada uma interrupção, seja ela de presença de algum animal ou gerada por outro dispositivo, como um celular, por exemplo.

Ainda será determinada a frequência de gravação de informações no banco de dados, para isso, serão determinadas quais variáveis serão enviadas junto ao peso, como hora ou temperatura.

3.2.5 Firmware

Foi desenvolvida uma parte de uma central, que funciona como *scanner* de *advertising*, e uma parte de um dispositivo que envia *advertising* levando em consideração os requisitos de funcionamento. O trecho de código em C abaixo, mostra a como é feito o *scanner*.

```
1
2 static void scan_start(void)
3 {
4     ret_code_t ret;
5
6     ret = sd_ble_gap_scan_start(&m_scan_params);
7     APP_ERROR_CHECK(ret);
8 }
```

A função *sd-ble-gap-scan-start()* é responsável por acessar o iniciar o processo de *scan*, levando em consideração as posições de memória e registradores necessários para acessar a antena.

```
1
2 static ble_gap_scan_params_t const m_scan_params =
3 {
```

```
    .active   = 1,
    .interval = SCAN_INTERVAL,
    .window   = SCAN_WINDOW,
    .timeout  = SCAN_TIMEOUT,
    .selective   = 0,
    p_whitelist = NULL,
};
```

A *struct* acima é utilizada para modularização do código. Essa *struct* específica é responsável pela determinação dos parâmetros do *scanner*, como tempo ativo, intervalo entre *scan* entre outros.

Para gerar mensagens de *advertising*, foi desenvolvido o seguinte código

```

static void advertising_init(void)
{
    uint32_t               err_code;
    ble_advertising_init_t init;

    memset(&init, 0, sizeof(init));

    init.advdata.name_type          = BLE_ADVDATA_FULL_NAME;
    init.advdata.include_appearance = false;
    init.advdata.flags              =
        BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE;

    init.srdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) /
        sizeof(m_adv_uuids[0]);
    init.srdata.uuids_complete.p_uuids  = m_adv_uuids;

    init.config.ble_adv_fast_enabled  = true;
    init.config.ble_adv_fast_interval = APP_ADV_INTERVAL;
    init.config.ble_adv_fast_timeout  = APP_ADV_TIMEOUT_IN_SECONDS
        ;

    init.evt_handler = on_adv_evt;

    err_code = ble_advertising_init(&m_advertising, &init);
    APP_ERROR_CHECK(err_code);

    ble_advertising_conn_cfg_tag_set(&m_advertising,
        APP_BLE_CONN_CFG_TAG);
}
```

3.3 Programação

A programação do sistema será feita de acordo com o padrão de projetos MVC, separando em camadas: o modelo, que consiste nos métodos e ações do aplicativo, a visão, responsável pela interação com o usuário e o *controller* que intermediará as ações do usuá-

rio chamando o método no modelo. Para tanto, foi dividido a programação em *front-end*, ou seja, a interface de interação com o usuário e *back-end*, responsável por fazer executar a comunicação com banco de dados e métodos de cálculos.

A aplicação desenvolvida conterá uma tela que apresentará uma lista de animais cadastrados informando o número de identificação, último peso registrado e eficiência alimentar calculada. Ao clica em cada linha, será exibido um gráfico mostrando a evolução ao longo dos dias que o animal foi pesado e deste uma tabela informará dados como previsão de lucro e tempo médio para abate.

3.3.1 Front-End da aplicação web

Linguagem de programação

Como mencionado o *Front-End* é a interface de interação com o usuário e por se tratar de uma aplicação WEB foram adotadas as linguagens de programação mais comuns para esta atividade como HTML, CSS e JavaScript.

O HTML será utilizado para a marcação de tabelas, menus e gráficos. Esta linguagem é desenvolvida em forma de tags, conforme o exemplo abaixo que é o mínimo necessário para o navegador interpretar a linguagem.

```
<html>
<header>
</header>
<body>
</body>
</html>
```

Estas tags são interpretadas pelo navegador criando o elemento que ela foi designada, algumas usadas na aplicação são:

```
<button></button>
```

Para criação de botões.

```
<table>
<tr>
<td>
</td>
<tr>
</table>
```

Para criação de tabela.

```
<ul>
<li>
<li>
</li>
```

Para criação de listas.

O CSS é uma linguagem utilizada para formatar o layout das tags, atribuir cores, formatos, transição, opacidade, entre outras características utilizadas para melhorar a experiência do usuário e deixar o desenho da tela mais agradável para o mesmo.

O CSS geralmente é escrito em um arquivo separado do aplicativo e incluído com a tag HTML

```
<link rel="stylesheet" href="Caminho Arquivo css">
```

Um exemplo da sintaxe do código CSS utilizado na aplicação encontra-se abaixo:

```
.container-fluid {
  width: 100%;
  padding-right: 15px;
  padding-left: 15px;
  margin-right: auto;
  margin-left: auto;
}
```

Para atribuir o código CSS a tag HTML devemos incluir o atributo *Class* na mesma, da forma que se segue.

```
<button class="container-fluid"></button>
```

Dessa forma, o botão irá receber as propriedades do código CSS que neste caso fará o botão ocupar 100% da tela, dando um espaço de 15px da borda da direita e da esquerda.

O JavaScript será utilizado para executar ações em ocorrência de eventos que acontecerão na interface do usuário. Esta linguagem permite, por exemplo, acionar uma função sempre que um determinado botão foi clicado, conforme o exemplo abaixo.

```
<button onclick="escrevemensagem()"></button>
<script>
Function escrevemensagem(){
Alert("Botao clicado");
}
</script>
```

No código acima, a tag <script></script> define que tem um código de Javascript que precisa ser interpretado pelo navegador e o atributo *onclick* define que sempre que o botão for clicado a função “escrevemensagem()” será chamada e a mesma emite um alerta no navegador com a mensagem “Botão clicado”.

Bootstrap

Para otimizar o desenvolvimento da aplicação foi utilizado o *framework* chamado de Bootstrap. Desenvolvido pela Twitter, o Bootstrap tem uma coleção de componentes, como botões, tabelas, menus, com estilos pré-definidos deixando a aplicação visivelmente mais agradável. O *framework* já implementa em seus códigos a programação HTML, CSS e JavaScript, cabendo ao programador somente replicar o código onde for necessário.

A figura 3.5 apresenta uma série de componentes desenhados por esta ferramenta.

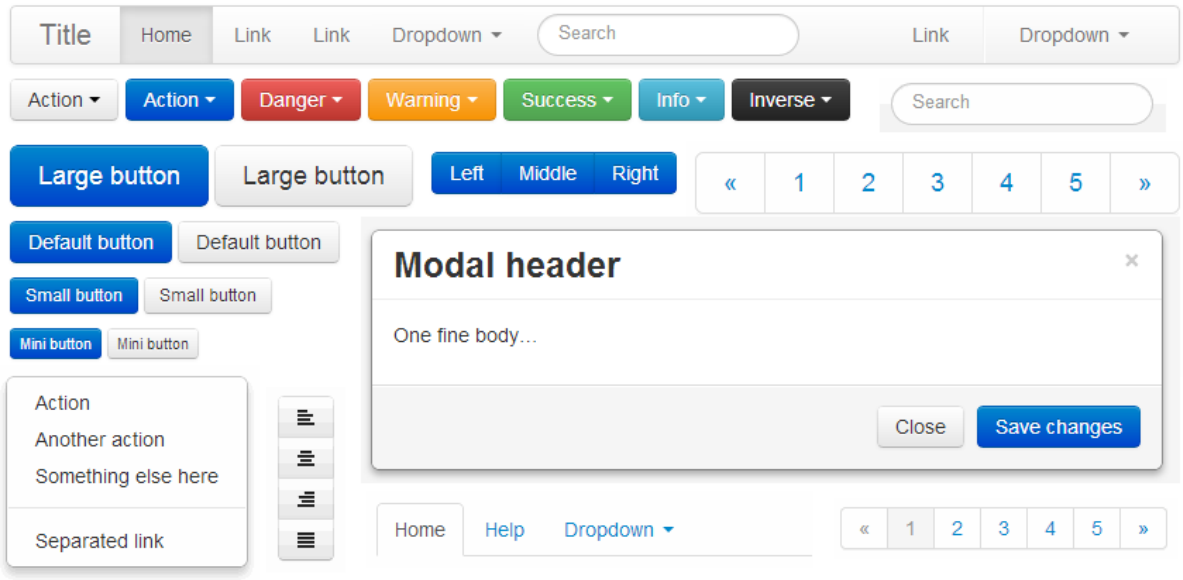

Figura 3.5: Componentes desenhados pelo Bootstrap Fonte: https://www.bootstrap.com

3.3.2 Back-End da aplicação web

Linguagem de programação

O *Back-End* será o responsável por efetuar as consultas ao banco de dados, fornecer um serviço de *webservice* e executar cálculos pertinentes ao sistema em um ambiente WEB. Para tanto, será necessário utilizar uma linguagem de programação que tenha recursos de orientação a objetos, seja compatível com o ambiente de desenvolvimento e promova uma facilidade na manutenção do código para futuras implementações. A linguagem mais adequada para estes requisitos é o PHP. A sua programação é interpretada por um servidor e escrita junto com o código HTML, CSS e Javascript, porém, ao entrar em execução, este código não é visualizado pelo usuário. Para a construção do sistema, a programação foi dividida em vários arquivos:

- index.php
- modelo.php
- sql.php
- view.php
- listabois.php
- tableboi.php

- chart.php

A função de *controller* do padrão MVC que recebe as interações da visão e interage com o modelo mais conveniente foi escrito no arquivo index.php. O arquivo modelo.php, contém códigos referente as ações que o sistema irá tomar, como fazer consultas no banco de dados e informar as modificações na *view*. O view.php, apresenta métodos que atualizarão a interface do usuário, chamando os códigos presentes nos arquivos listaboi.php e chart.php e para isto utilizarão métodos de consulta do banco de dados presente no arquivo sql.php. Os código estão disponíveis para consulta em `https://github.com/FelipePTO/appwebTCC`.

Capítulo 4

Metodologia

4.1 Projeto Balança e Cocho

O projeto da balança e do cocho foi desenvolvido utilizando o *software AutoCad*. O mesmo consistiu no desenho de suas vistas, a indicação dos materiais e de uma simulação, utilizando o software ANSYS para estudo de resistência do material. Todas as dimensões aplicadas nos desenhos, se não apresentadas, estão em milímetros.

4.1.1 Projeto da Balança

As figuras 4.1, 4.2, 4.3 apresentas as vistas frontal e superior e lateral da balança do projeto de montagem, respectivamente.

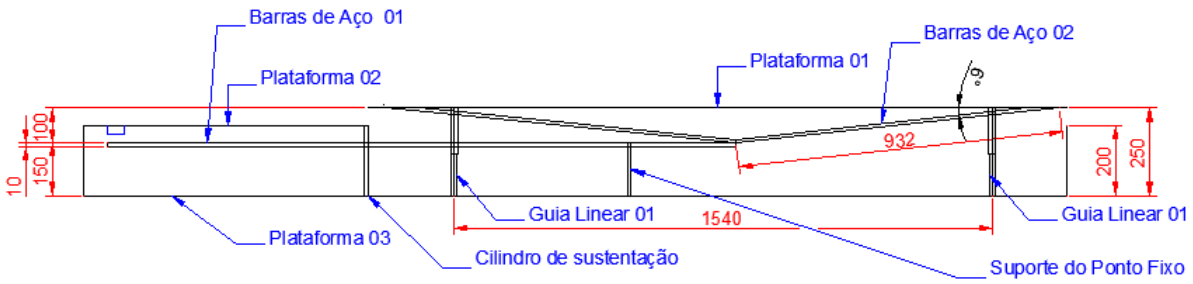

Figura 4.1: Vista Superior da balança Fonte: Próprio autor

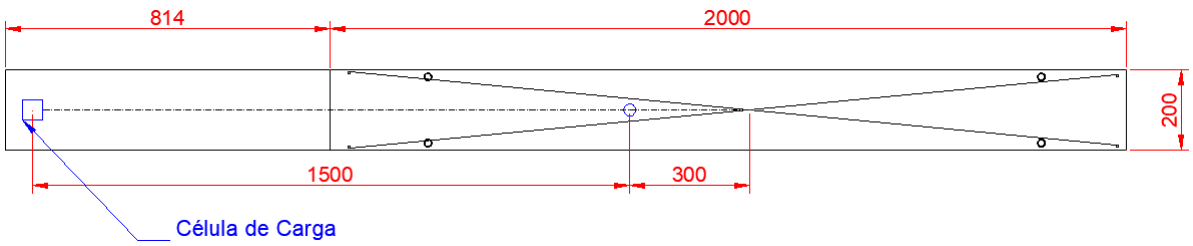

Figura 4.2: Vista Superior da balança Fonte: Próprio autor

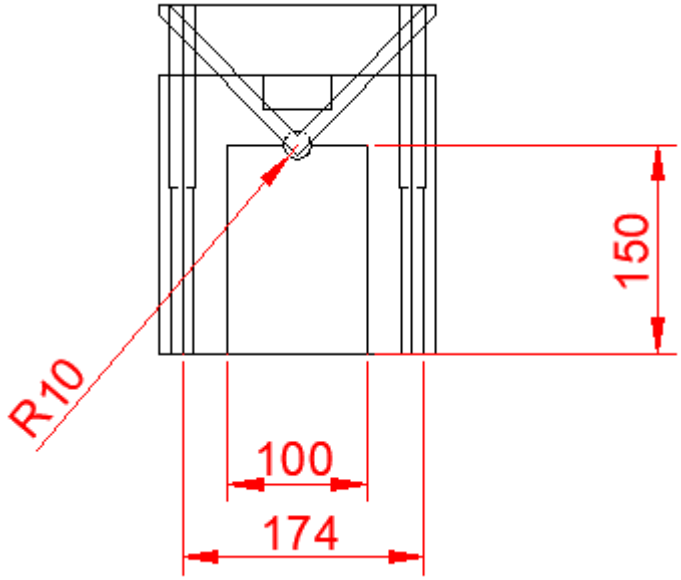

Figura 4.3: Vista Lateral da balança Fonte: Próprio autor

O projeto de fabricação foi feito de forma descritiva, pois não há necessidade de dobras e cortes especiais. Logo, para a fabricação desta balança são necessários os seguintes materiais.

- Quatro ”Guia Linear 01”: Composta por dois cilindros, um de 25mm e outro de 19mm de diâmetro, ambos com comprimento de 150 mm
- Dois ”Cilindro de sustentação”: 25mm de diâmetro, 200mm de comprimento
- Uma ”Plataforma 01”: Chapa de zinco 200x1200x5mm
- Uma ”Barra de aço 01”: Barra de aço 40x80x3 com comprimento de 1800mm
- Um ”Suporte de ponto fixo”: Chapa de 100x150x5mm com um furo de 25mm representado no desenho de montagem
- Quatro ”Barra de aço 02”: Barra de aço 40x80x3 com comprimento de 950mm. Em uma de suas extremidades é necessário um corte angular de 10º, para que a barra forme uma base para apoiar a ”Plataforma 01”
- Uma ”Célula de carga”: Posição na qual é indicado a localização do sensor de célula de carga

A “barra de aço 01” é responsável pela transmissão do momento gerado pelo peso do animal até a célula de carga. Por este motivo, é o elemento do sistema que sofrerá maior

esforço. Para que se tenha a certeza de que o material escolhido suporta o esforço ao qual será submetido, foram realizadas simulações estáticas do material.

O tópico 4.1.2 mostra os resultados obtidos a partir de simulação realizada no *software* ANSYS, que é baseado no método de elementos finitos.

4.1.2 Simulação

Foi realizada inicialmente a simulação utilizando a geometria descrita no tópico anterior, utilizando aço inoxidável 304, que é um material isentrópico e de fácil aquisição. As características configuradas para realizar a simulação foram as seguintes:

- Módulo de elasticidade: 193 Gpa
- Razão de Poison: 0,3
- Densidade: 8,00 g/cm^3
- Módulo de rigidez: 86,2 Gpa
- Limite de escoamento: 350 Mpa
- Limite de resistência : 770 Mpa

Para a simulação, foi desenhada a geometria da barra, e a partir dela foi gerada a malha para efetuação dos cálculos. Foi utilizada uma malha com elementos triangulares e de tamanho médio de 3mm de aresta, gerada automaticamente pelo *software*. A imagem 4.4 mostra o resultado deste processo.

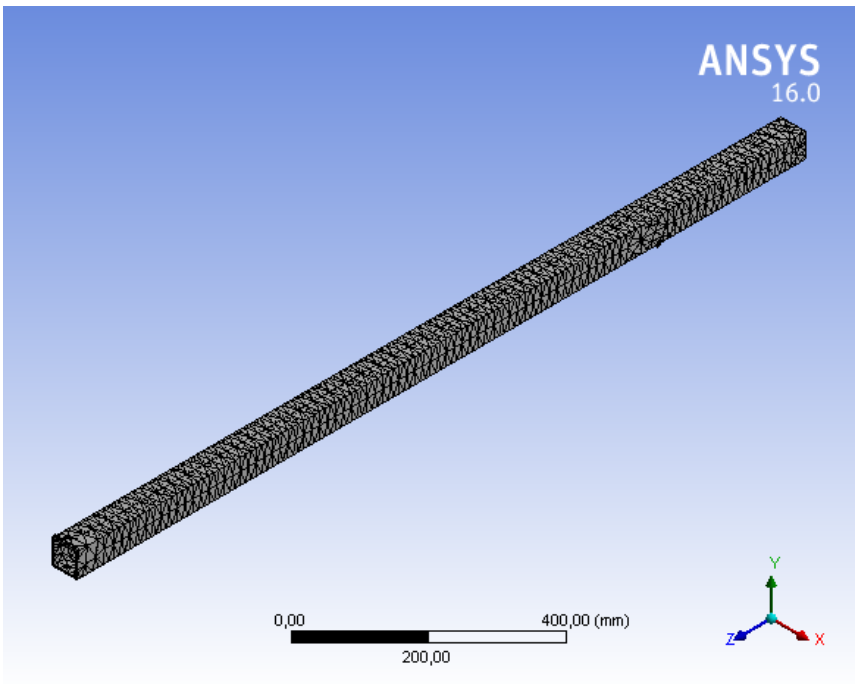

Figura 4.4: Malha gerada na geometria Fonte: Próprio autor

Foram determinados os pontos de apoio e de aplicação de força. A força utilizada para simulação foi de 19620 N, o que seria equivalente a força peso de um animal de 2000 Kg. Os pontos de apoio e de aplicação da força podem ser visualizados na figura 4.5 .

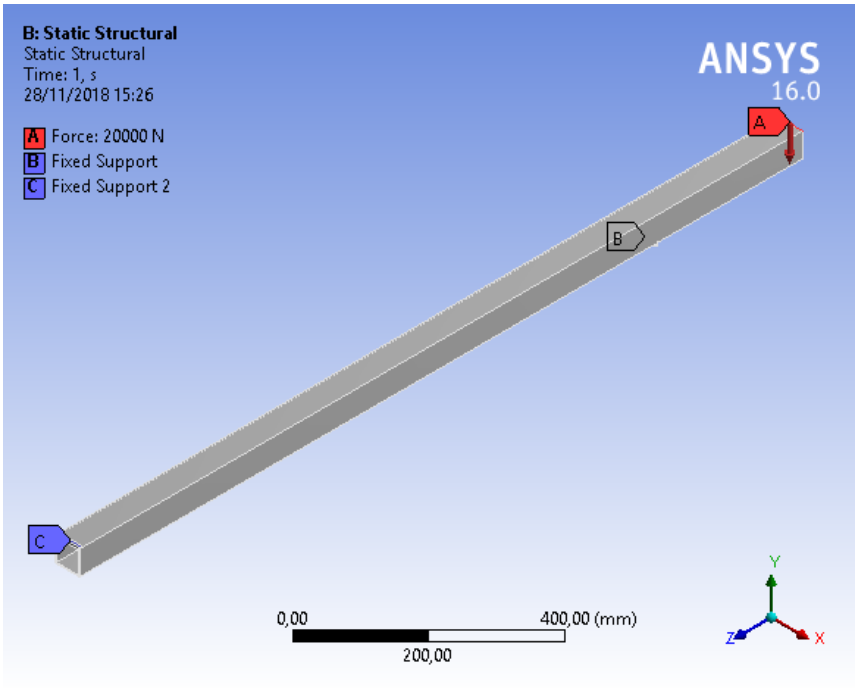

Figura 4.5: Pontos de apoio e aplicação de força Fonte: Próprio autor

Foi avaliada a deformação mecânica da barra, para que seja possível a implementação de uma plataforma que não gere um deslocamento superior a 50 mm, tornando mais prática e segura a balança.

A figura 4.6 mostra a deformação do material submetido aos esforços descritos anteriormente. Foi constada uma deformação de 179,85 mm, valor este muito superior ao requisito desejado. Para resolver este problema, foi testada uma outra geometria, também de fácil aquisição. O perfil escolhido foi de 50 mmx50 mm, e 2 mm de espessura.

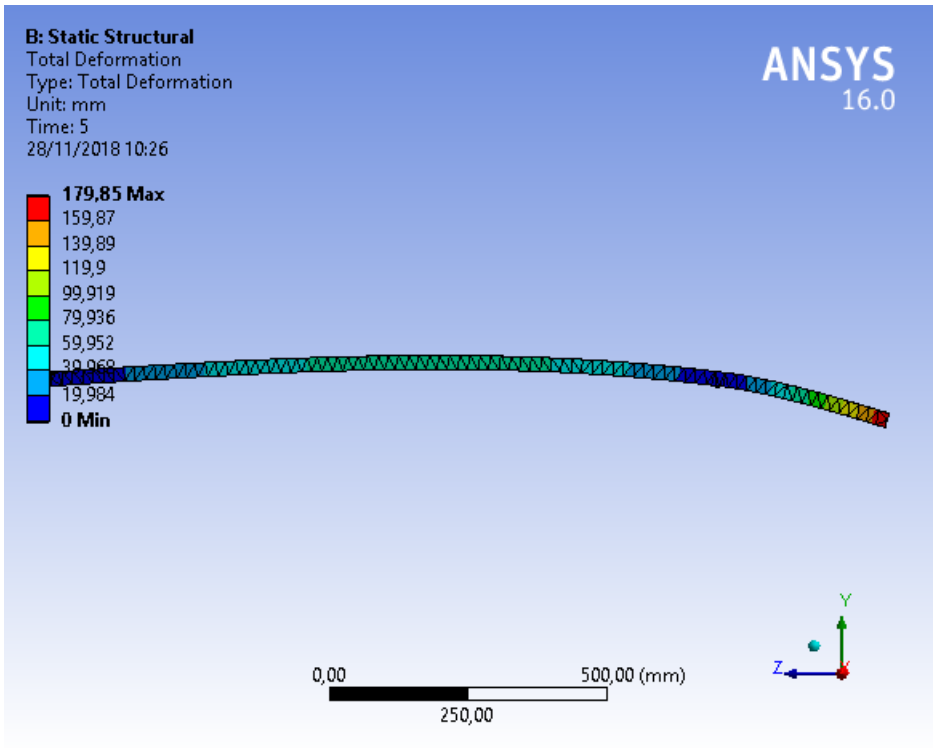

Figura 4.6: Deformação total (perfil 25x25) Fonte: Próprio autor

A figura 4.7 mostra a deformação da nova geometria, composta pelo mesmo material e submetida aos mesmos esforços descritos anteriormente. A deformação máxima obtida foi de 26,65 mm, o que é uma deformação aceitável levando em consideração o requisito adotado anteriormente.

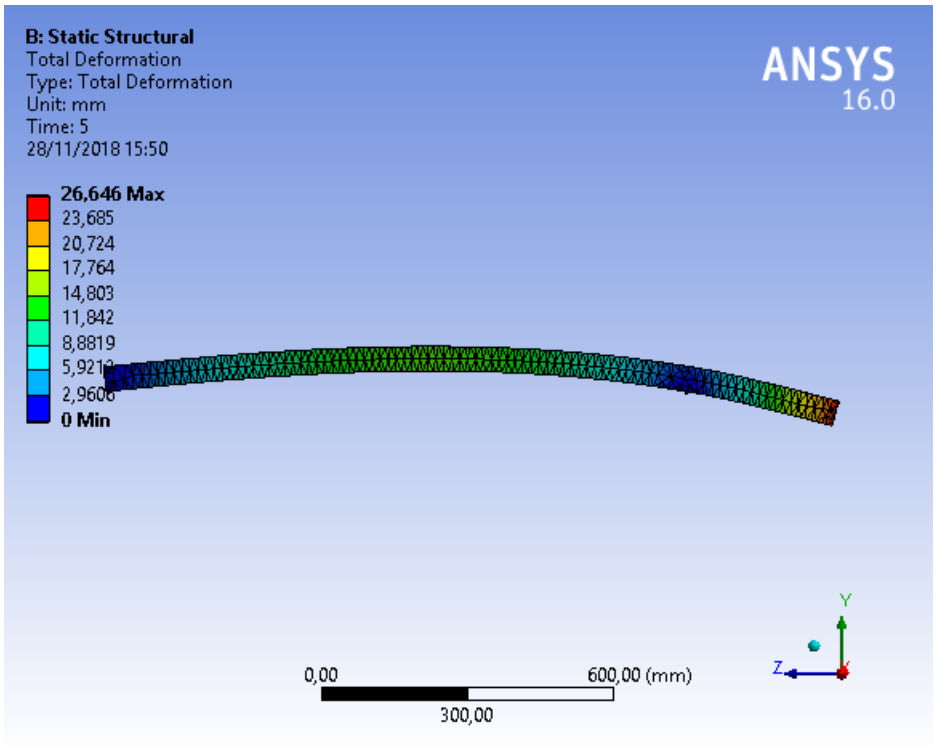

Figura 4.7: Deformação total (perfil 50x50) Fonte: Próprio autor

Como o material atendeu a especificação de deformação, foi realizada a simulação de tensão máxima ao qual o material será submetido. A figura 4.8 mostra o resultado obtido. Os valores de tensão ao qual será submetida a estrutura são superiores ao de limite de escoamento do material, ou seja, o material entrará em regime de deformação plástica, o que modifica a geometria inicial. Para sanar este problema, foi escolhido um perfil de 40 mmx80 mm, e 3 mm de espessura.

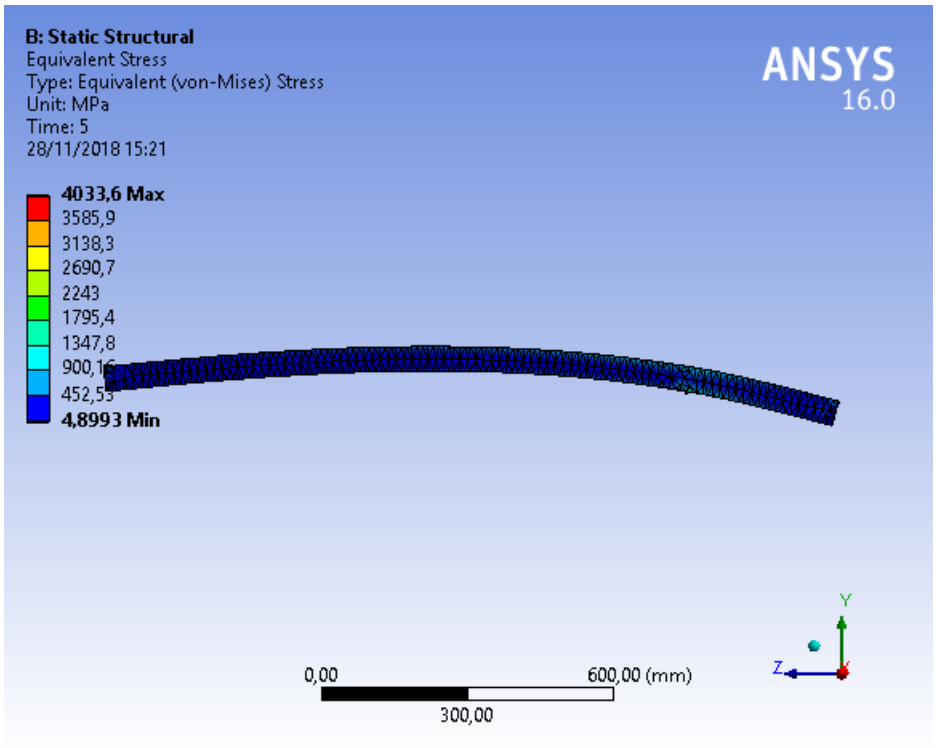

Figura 4.8: Deformação total (perfil 50x50) Fonte: Próprio autor

O resultado obtido a partir da simulação do perfil de 40 mmx80 mm e 3 mm de espessura foi de uma deformação total de 9,39 mm. A figura 4.9 mostra o resultado da simulação de deformação total.

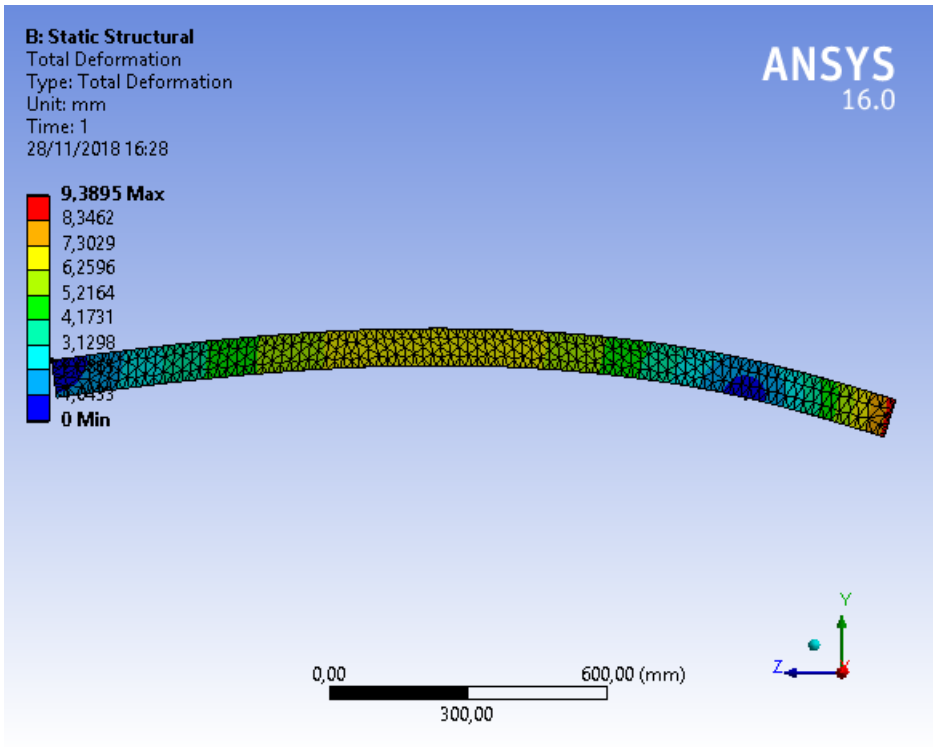

Figura 4.9: Deformação total (perfil 40x80) Fonte: Próprio autor

A figura 4.10 mostra o resultado da simulação de tensão máxima. É possível observar que o perfil escolhido suporta, sem deformar-se plasticamente, o esforço ao qual será submetido.

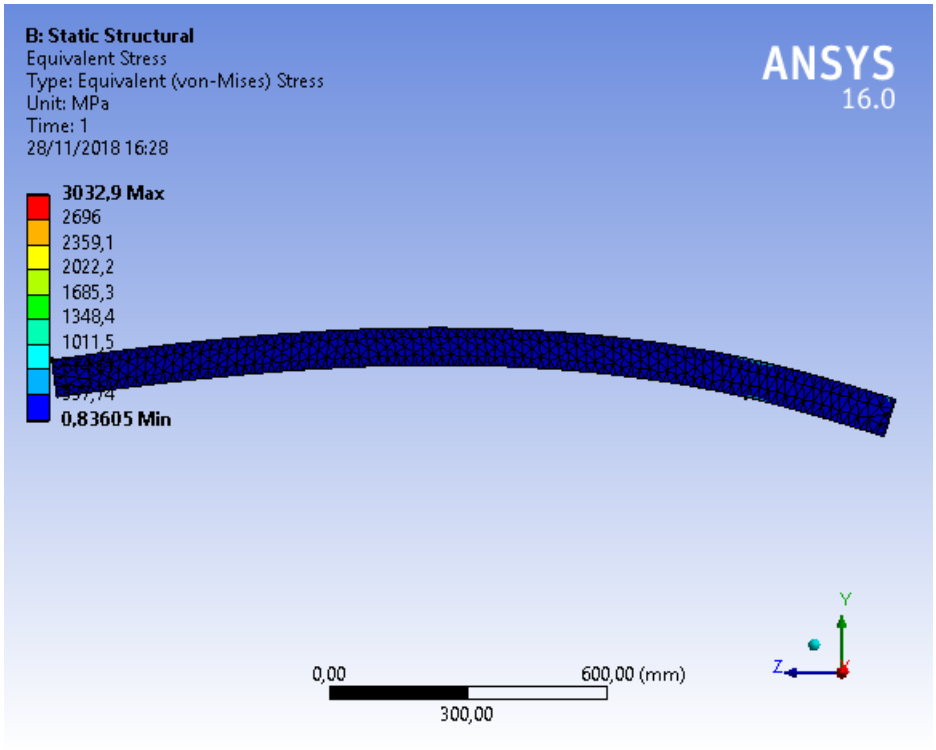

Figura 4.10: Tensão máxima (perfil 40x80) Fonte: Próprio autor

Com base nas simulações realizadas, foi modificado o projeto inicial, e a "barra de aço 01" teve seu perfil alterado para o retangular de dimensões $40mm$x$80mm$ e $3mm$ de espessura de parede.

4.1.3 Projeto do Cocho

Semelhante ao da balança, o projeto do cocho foi desenvolvido desenhos de suas vistas para o projeto de montagem e uma lista descritiva para o projeto de fabricação.

As figuras 4.11, 4.12 apresentam as vistas frontal e lateral respectivamente.

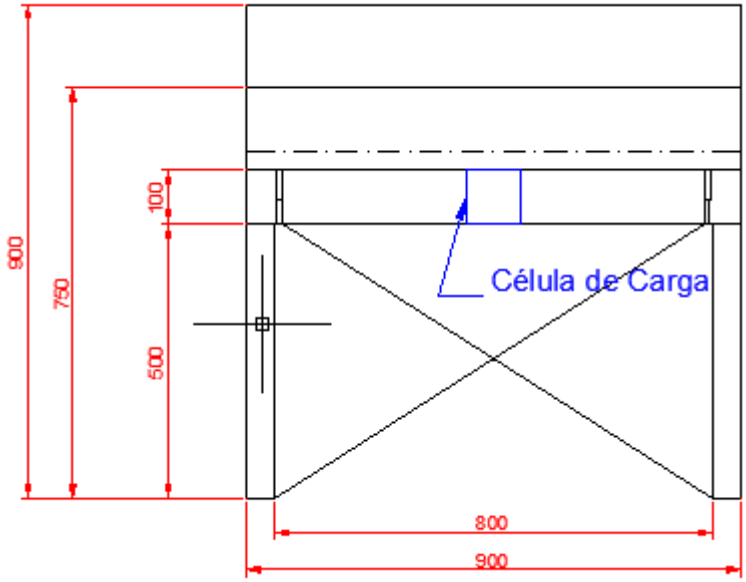

Figura 4.11: Vista Lateral do cocho Fonte: Próprio autor

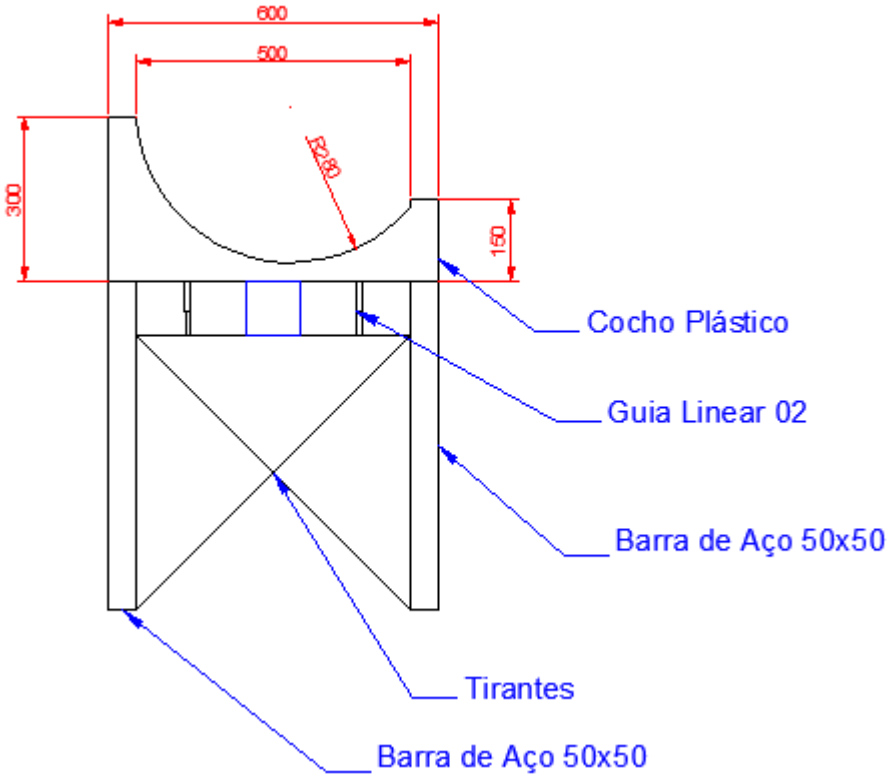

Figura 4.12: Vista Frontal do cocho Fonte: Próprio autor

- Coxo Plástico: Coxo pré-fabricado de Plástico
- 8 ”Tirantes”: Tirantes de aço de 1mm de diâmetro

- Duas ”Guia linear 02”: Composta por dois cilindros, um de 25mm e outro de 19mm, ambos com comprimento de 70 mm
- Quatro ”Barra de Aço”: Composta por barras de 40x80x600x3mm

4.2 Projeto da plataforma

A construção da plataforma foi baseada no princípio de alavancas. Este mecanismo é utilizado para ampliar ou reduzir a força aplicada em uma ponta da mesma em relação ao outro lado. Esta relação de forças varia a medida que a a distância de uma ponta aumenta ou diminui em relação ao ponto fixo. A figura conforme a figura 4.13.

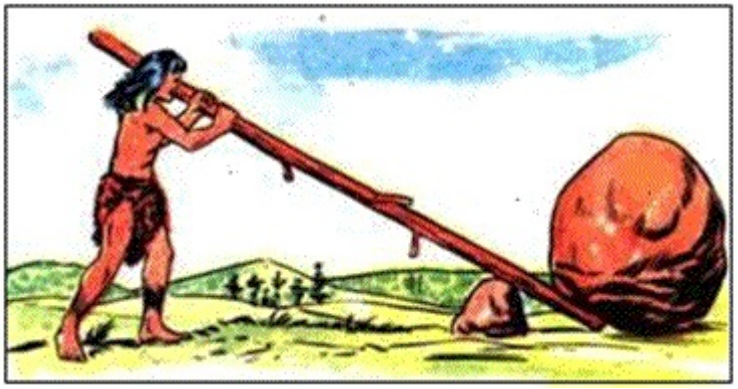

Figura 4.13: Aplicação clássica de uma alavanca Fonte: https://mundoeducacao.bol.uol.com.br/

Esta metodologia de construção foi escolhida, pois, assim como o exemplo clássico mostrado na figura 4.13 precisamos de uma redução de forças simples e dessa forma é possível utilizar uma célula de carga de menor amplitude, por sua vez, mais barata, e aumentar a capacidade da mesma para os níveis desejados de trabalho.

A célula de carga utilizada no protótipo tem a capacidade de pesar até 50 Kg e utilizando a alavanca, seu potencial será elevado para até 200kg. Com isso devemos ter uma relação de distância entre a aplicação da força e a célula de carga de 4 vezes.

A partir deste princípio foi desenhado no software 3D SolidWorks a seguinte plataforma, todas as medidas estão em milímetros.

Vista Superior

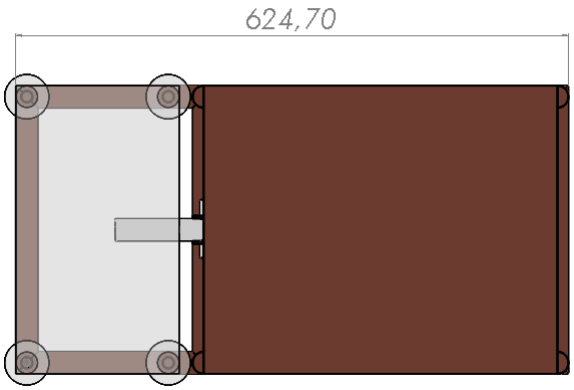

Figura 4.14: Vista superior da balança Fonte: Próprio autor

Vista Lateral

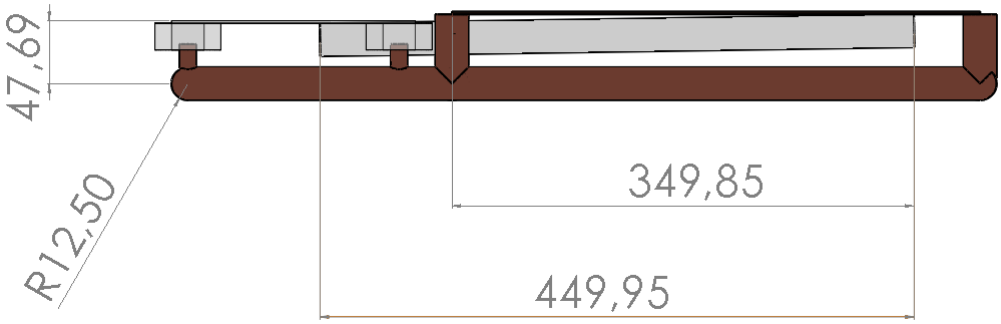

Figura 4.15: Vista lateral da balança Fonte: Próprio autor

Vista Isométrica

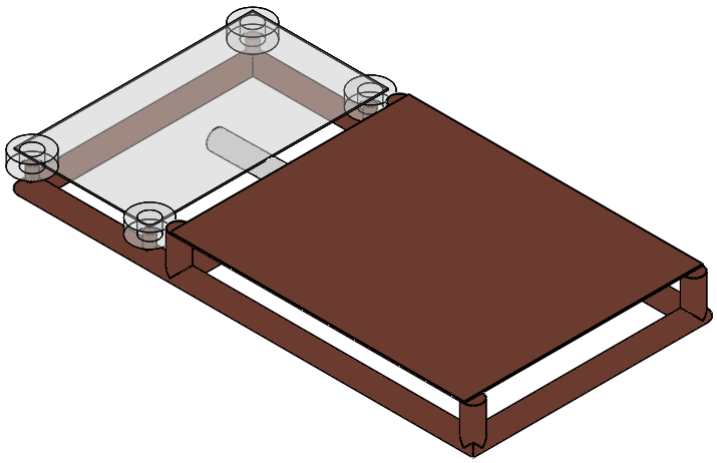

Figura 4.16: Vista isométrica da balança Fonte: Próprio autor

De acordo com as dimensões do desenho, temos a relação de forças dadas abaixo.

$$F_1 x D_1 = F_2 x D_2$$

Sendo

$$D_1 = 100$$

e

$$D_2 = 349$$

Temos

$$F_2/F_1 = 349/100$$

$$F_2/F_2 = 3{,}49$$

Portanto, para uma célula de carga de 50Kg, podemos aplicar no máximo, aproximadamente, 150kg.

Após desenhado a plataforma, foi realizado uma análise estática na montagem, porém, devido a limitações computacionais e de geometria, não foi possível efetuar a mesma. Portanto, a análise efetuada levou em consideração as forças máximas aplicadas e a resistência mecânica do material utilizado.

De acordo com [51] que realizou um teste de flexão de um tubo semelhante ao utilizado no projeto, temos um valor de aproximadamente 1700N, valor superior ao utilizado no projeto.

4.3 Construção da plataforma

A construção da plataforma foi feita utilizando canos de PVC de 25mm de diâmetro e canos de PVC de 30mm de diâmetro.

Estes materiais foram escolhidos por sua facilidade de manuseio, custo e resistência mecânica.

As etapas da construção pode ser vista nas figuras 4.17 e 4.18.

Figura 4.17: Construção da Plataforma que recebe a força Fonte: Próprio autor

Figura 4.18: Construção do alongamento para o braço Fonte: Próprio autor

Para as chapas, foram utilizados chapas de acrílico, pelo mesmos motivos mencionados para o uso dos canos.

4.4 Calibração do sensor

Para o funcionamento do sistema de medição, é necessária uma calibração inicial, tanto para determinação do zero do sistema como para a determinação do fator de escala.

O fator de escala é necessário devido aos ajustes para compensação da montagem física e da diferença da construção mecânica dos componentes.

Fator de escala é uma constante que multiplica o valor medido. Para obter o valor desta constante é necessário que se realize a pesagem de um objeto de peso conhecido, e aferir qual é o valor indicado no sistema de pesagem. Com o valor real e o valor medido em mãos, é feita uma simples divisão para que seja determinado o fator de escala.

No *firmware* desenvolvido, o fator de escala é ajustado de forma gradativa ao invés de ser determinado de maneira exata. Isso se deve ao fato de que a ordem de grandeza da variação de resistência da célula de carga ser muito pequena, o que dificulta cálculos exatos devido a grandes mudanças que podem ser ocasionadas a partir de pequenas variações.

4.5 Programação do firmware

Foi desenvolvido um *firmware* para o sistema de pesagem do animal. O detalhamento do desenvolvimento e funcionalidades do código são abordadas nesta seção.

O elemento sensor do sistema é uma célula de carga com princípio de funcionamento resistivo, ou seja, sua resistência varia de acordo com a deformação de sua estrutura física. A variação de valores gerados pela deformação é muito pequena, na ordem de grandeza de miliohms.

Para que a variação do sinal de medição possa ser interpretada e processada pelo microcontrolador, é necessário que o sinal seja amplificado, elevando a amplitude do sinal a ordens de grandezas mais perceptíveis . Para fazer a amplificação do sinal medido, foi utilizado um módulo Hx711.

4.5.1 Hx711

O módulo Hx711 é um conversor analógico-digital de 24-bits, que além de amplificar o sinal de entrada o transforma em uma saída digital, utilizando o protocolo de comunicação I2C. Para utilizar o circuito conversor, é necessária a implementação dos elementos de resistência de uma Ponte de Wheatstone. Essa implementação é necessária pois não é possível amplificar sinais de resistência, visto que o resistor é um elemento passivo. A

solução utilizada é a amplificação do valor de tensão obtido através de uma ponte de Wheatstone, que garante a variação da tensão medida a partir da variação da resistência do sensor.

4.5.2 Firmware

O *firmware* implementado recebe o valor de medição através do barramento I2C do microcontrolador. Os dados são requisitados pelo sistema sempre que identificada a aproximação de algum animal, desta forma a balança consome o mínimo de energia possível e evita medições indevidas de animais não cadastrados ou de elementos estranhos ao sistema. O trecho de código que segue é o responsável pelo estabelecimento da comunicação I2C.

```
1  void twi_init (void)
2  {
3      ret_code_t err_code;
4
5      const nrf_drv_twi_config_t twi_config = {
6         .scl                = ARDUINO_SCL_PIN,
7         .sda                = ARDUINO_SDA_PIN,
8         .frequency          = NRF_TWI_FREQ_100K,
9         .interrupt_priority = APP_IRQ_PRIORITY_HIGH,
10        .clear_bus_init     = false
11     };
12
13     err_code = nrf_drv_twi_init(&m_twi, &twi _config, twi_handler,
           NULL);
14     APP_ERROR_CHECK(err_code);
15
16     nrf_drv_twi_enable(&m_twi);
17 }
```

Após iniciada a comunicação, é necessária uma função que lide com os eventos provenientes do barramento I2C, essa função é chamada de *handler*. O *handler* é responsável pelo primeiro tratamento do sinal, identificando o seu tipo e qual a mensagem enviada pelo sensor. A função *twi-handler* é mostrada no trecho abaixo.

```
1
2  void twi_handler(nrf_drv_twi_evt_t const * p_event, void *
       p_context)
3  {
4      switch (p_event->type)
5      {
6          case NRF_DRV_TWI_EVT_DONE:
7              if (p_event->xfer_desc.type == NRF_DRV_TWI_XFER_RX)
8              {
9                  data_handler(m_sample);
10             }
11             m_xfer_done = true;
```

```
12                  break;
13              default:
14                  break;
15          }
16  }
```

A variável *m-xfer-done* é responsável por garantir que o barramento não seja ocupado simultaneamente por mensagens enviadas pelo emissor e pelo receptor. A partir dela é possível garantir que a comunicação ocorrerá sem problemas e sem congestionamento do barramento.

O *firmware* de identificação individual envia mensagens através de *broadcast* para o *observer*, que é o código desenvolvido para a tratar a pesagem do animal.

O sinal emitido pelo sensor presente em cada animal chega ao receptor com alguns dados característicos do microcontrolador e não da mensagem enviada por ele. Alguns destes dados são o *MAC* do sensor e o *RSSI (Received Signal Strength Indicator)*. O RSSI é medida da potência do sinal de rádio recebido pelo *observer*, que no sistema é o microcontrolador responsável pela pesagem do animal.

Utilizando a potência do sinal emitido por cada animal, a balança é capaz de identificar qual individuo está sendo pesado naquele momento. O *firmware* processa os dados provenientes do sensor e estampa juntamente com a identificação de cada animal em um cartão *microSD*, salvando em meio físico e sem a necessidade de conexões locais ou com a internet.

Para que a central possa acessar o cartão, inserir, modificar e gerar arquivos em seu diretório, é necessária a utilização de um módulo leitor de cartão. A imagem 4.19 mostra o adaptador utilizado para cumprir esta função.

Para gerenciar o acesso ao módulo, o firmware deve se comunicar através do protocolo SPI, que utiliza quatro fios e possui uma taxa de transmissão superior ao protocolo I2C. O microcontrolador possui três periféricos que se revezam entre os protocolos I2C e SPI. Por este motivo, é possível estabelecer a comunicação entre o módulo *Hx711* e o leitor de cartão. A parte do código responsável pela inicialização do módulo SPI do microcontrolador e pela gravação no cartão se encontra no Apêndice B, na função *write-data-sd()*.

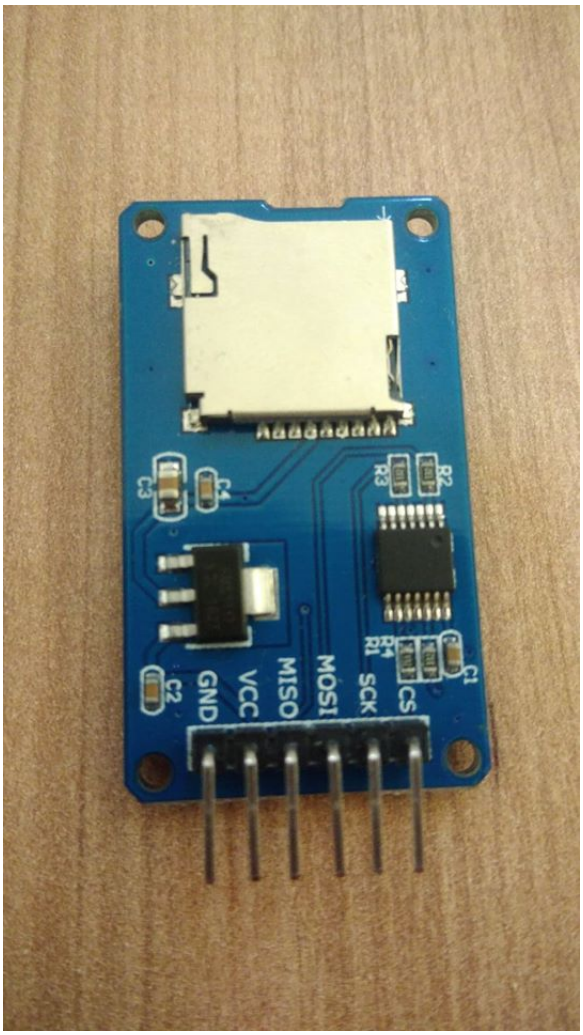

Figura 4.19: Leitor de cartão microSD Fonte: Próprio autor

4.6 Filtro

Para que um filtro seja desenvolvido, é necessário que se faça a aquisição dos sinais aos quais ele estará sujeito, tornando o projeto mais eficiente para o processo em que será inserido.

Para gerar os dados de medição, foi construída uma plataforma onde foi acoplada a célula de carga. Foram gerados dados com dois pesos distintos, sendo aplicados sobre a balança de forma dinâmica, simulando a passagem rápida de peso e a adição repentina de carga.

Partindo dos dados coletados, foi observado que a dinâmica do sistema é muito rápida, sendo desnecessária a implementação de um filtro. Os dados serão tratados via código, para que seja capturado apenas o maior valor de peso aferido pelo sistema.

Devido a maleabilidade da estrutura física, não foi possível realizar experimentos com grande impacto, o que pode gerar um falseamento nos dados coletados. Tomando como

válidos os esforços analisados, e levando-se em consideração a grande variação do fundo de escala da célula de carga de grande porte, os dados coletados foram considerados satisfatórios para o desenvolvimento da aplicação. Para que um filtro seja desenvolvido, é necessário que se faça a aquisição dos sinais aos quais ele estará sujeito, tornando o projeto mais eficiente para o processo em que será inserido.

Para gerar os dados de medição, foi construída uma plataforma onde foi acoplada a célula de carga. Foram gerados dados com dois pesos distintos, sendo aplicados sobre a balança de forma dinâmica, simulando a passagem rápida de peso e a adição repentina de carga.

Partindo dos dados coletados, conforme mostrado na figura 4.20, foi observado que a dinâmica do sistema é muito rápida, observando que cada divisão tem uma base de tempo é de 100ms, sendo desnecessária a implementação de um filtro. Os dados serão tratados via código, para que seja capturado apenas o maior valor de peso aferido pelo sistema.

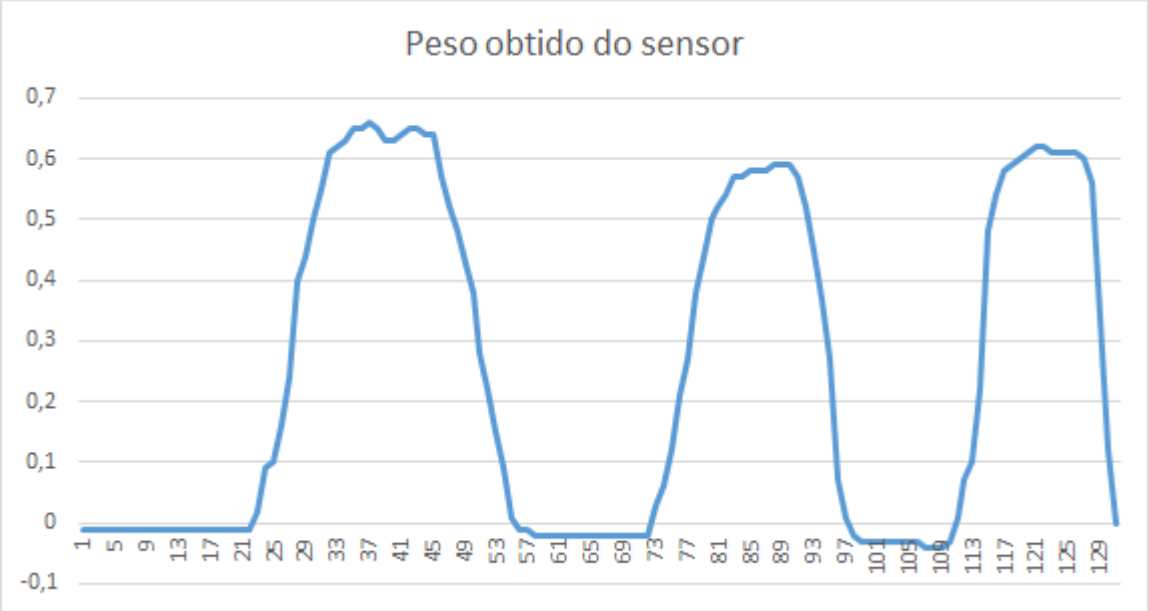

Figura 4.20: Dados provenientes do sensor Fonte: Próprio autor

Devido a maleabilidade da estrutura física, não foi possível realizar experimentos com grande impacto, o que pode gerar um falseamento nos dados coletados. Tomando como válidos os esforços analisados, e levando-se em consideração a grande variação do fundo de escala da célula de carga de grande porte, os dados coletados foram considerados satisfatórios para o desenvolvimento da aplicação.

4.7 Sistema de identificação individual

Foi desenvolvido o *firmware* para a identificação de cada animal, para que a pesagem possa ser correlacionada com cada indivíduo do rebanho.

Para fazer a identificação individual foi utilizada a tecnologia empregada em *beacons*. Esta tecnologia consiste na utilização de *tags* ativas, que enviam através da tecnologia *bluetooth 4.2* dados que podem ser interpretados por outros dispositivos compatíveis.

O funcionamento do sistema é baseado no envio e recepção de mensagens com *payload* variante ou não. O assunto foi abordado de maneira mais profunda no tópico 2.3.4.

O *firmware* desenvolvido gera um sinal com dados para serem enviados para o ambiente. Um dos *bytes* enviados pelo emissor é sempre um valor aleatório, que é modificado sempre que é iniciado o envio de mensagens. Este artificio é utilizado para que o dispositivo receptor possa ignorar sinais repetidos, diminuindo o consumo energético e garantindo que uma maior quantidade de dados possa ser transmitida pela rede. O trecho do código responsável por gerar os valores aleatórios é mostrado abaixo.

```
// Set randon key in data packet
err_code =sd_rand_application_vector_get(&m_adv_data[6],1);
APP_ERROR_CHECK(err_code);
err_code = ble_advdata_set(&advdata, NULL);
APP_ERROR_CHECK(err_code);
```

Além do valor aleatório, a mensagem enviada carrega um endereço *MAC* único, que é utilizado para distinguir um microcontrolador de outro. O firmware coleta também a temperatura do *chip*, e a tensão da bateria do sensor. Essas informações são disponibilizadas no *payload* e enviadas para a central. O código de aquisição de temperatura é mostrado abaixo.

```
uint8_t temperature_data_get(void)
{
    int32_t temp;

    uint32_t err_code = sd_temp_get(&temp);
    APP_ERROR_CHECK(err_code);

    temp = (temp>>1)+50;

                // if temperature below -25 C  saturate at -25 C
                   if temp over 102.5 saturate at 102.5
                if(temp<0)temp=0;
                else if(temp>255) temp=255;

    return (uint8_t)temp;
}
```

Para medir o nível de tensão da bateria do sensor é utilizado o ADC do micro, que faz toma como base de comparação o terra da bateria. A tensão da bateria cai a medida que ela perde sua carga, com isso, é possível fazer o acompanhamento da duração da bateria em cada sensor, e quando deverá ser substituída.

Para utilizar o ADC do micro como medidor de tensão da bateria é necessário que um

dos pinos que pode ser configurado como conversor seja curto-circuitado com a fonte de tensão da placa.

Para configurar o ADC é necessário utilizar o GPIO do microcontrolador, para que seja configurado como entrada o pino responsável pela medição do sinal. O trecho de código abaixo mostra a implementação do ADC, assim como a configuração do pino como *input*.

```
1  static void adc_init(void)
2  {
3  #ifndef NRF52832
4      ret_code_t err_code;
5      nrf_drv_adc_config_t adc_config = NRF_DRV_ADC_DEFAULT_CONFIG;
6
7      err_code = nrf_drv_adc_init(&adc_config, NULL);
8      APP_ERROR_CHECK(err_code);
9
10                 adc_channel_config.config.config.reference =
                       NRF_ADC_CONFIG_REF_VBG;
11                 adc_channel_config.config.config.input=
                       NRF_ADC_CONFIG_SCALING_SUPPLY_ONE_THIRD;
12                 adc_channel_config.config.config.resolution=
                       NRF_ADC_CONFIG_RES_8BIT;
13
14     nrf_drv_adc_channel_enable(&adc_channel_config);
15 #else
16                 ret_code_t err_code;
17     nrf_drv_saadc_config_t saadc_config;
18     nrf_saadc_channel_config_t channel_config;
19
20
21     //Configure SAADC
22     saadc_config.low_power_mode = true;
23     saadc_config.resolution = NRF_SAADC_RESOLUTION_8BIT;
24     saadc_config.oversample = NRF_SAADC_OVERSAMPLE_DISABLED;
25     saadc_config.interrupt_priority = APP_IRQ_PRIORITY_LOW;
26
27     //Initialize SAADC
28     err_code = nrf_drv_saadc_init(&saadc_config, saadc_callback);
29     APP_ERROR_CHECK(err_code);
30
31     //Configure SAADC channel
32     channel_config.reference = NRF_SAADC_REFERENCE_INTERNAL;
33     channel_config.gain = NRF_SAADC_GAIN1_6;
34     channel_config.acq_time = NRF_SAADC_ACQTIME_40US;
35     channel_config.mode = NRF_SAADC_MODE_SINGLE_ENDED;
36     channel_config.pin_p = NRF_SAADC_INPUT_VDD;
37     channel_config.pin_n = NRF_SAADC_INPUT_DISABLED;
38     channel_config.resistor_p = NRF_SAADC_RESISTOR_DISABLED;
39     channel_config.resistor_n = NRF_SAADC_RESISTOR_DISABLED;
40
```

```
41
42      //Initialize SAADC channel
43      err_code = nrf_drv_saadc_channel_init(0, &channel_config);
44      APP_ERROR_CHECK(err_code);
45  #endif
46  }
```

4.8 Aplicação web

A aplicação WEB foi composta de várias, partes ferramentas utilizadas, banco de dados, front-end e back end. Será dividido em tópicos para relatar o desenvolvimento dos mesmos.

4.8.1 Ferramentas Utilizadas

Para o desenvolvimento da aplicação foram necessários a utilização de algumas ferramentas, as quais serão listadas abaixo.

ATOM

Atom é um editor de texto de código aberto, desenvolvido na plataforma Electron e é amplamente utilizada para programação de aplicações WEB. Esta ferramenta contém pacotes que facilitam o trabalho do desenvolvedor como extensões que autocompletam palavras, identifica parênteses e chaves abertas e fechadas, além de mostrar todo conteúdo da pasta de projeto de forma amigável. A figura 4.21 apresenta o logo desta ferramenta.

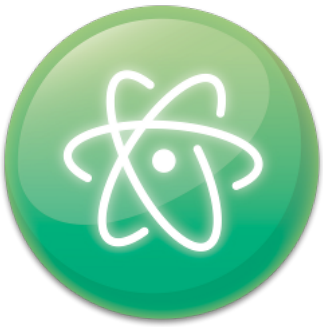

Figura 4.21: Simbolo do ATOM Fonte: https://atom.io/

NodeJS

NodeJS é um interpretador de JavaScript, linguagem utilizada no front-end. A grande vantagem da utilização do NodeJS é o seu gerenciado de pacotes. Com esta ferramenta é possível criar projetos do zero com uma velocidade considerável, portanto facilita bastanto o trabalho do desenvolvedor.

Figura 4.22: Simbolo do NodeJS Fonte: https://nodejs.org/en/

4.8.2 Banco de dados

O banco utilizado para armazenar os dados foi o MySQL 4.23.

Figura 4.23: Simbolo do MysQL Fonte: https://www.mysql.com/

Para a manipulação do banco de dados é necessário a criação de um *database* e dentro da mesma a criação de tabelas que receberão os registros a serem armazenados.

Inicialmente criamos um database com o comando abaixo.

```
1 CREATE DATABASE TCC;
```

Com o *database* criado, devemos criar as tabelas que receberão os dados a serem armazenados. As tabelas contém colunas e estas contém tipos que variam de acordo com a natureza da variável.

Para o TCC, serão criadas três tabelas:

- parametroboi: Que receberá o número de identificação do animal, assim como armazenará o valor do cálculo de eficiência, o peso inicial e a data que entrou que entrou no sistema de confinamento.
- listaboi: Que receberá os pesos registrados pela balança e associará o mesmo com a identificação do boi

- parametros: Que receberá variáveis do sistema, como valor da ração e valor da arroba

Para a tabela parametroboi temos as seguintes nomes das colunas com o tipo utilizado.

- numidentificao: inteiro
- eficiencia: float
- pesoinicial: float
- dataentrou: date

Utilizando as informações acima, utilizamos o comando abaixo para criar a tabela.

```
CREATE TABLE  parametroboi (
numidentificao INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
eficiencia FLOAT(8,4) null,
pesoinicial FLOAT(8,4) null,
dataentrou date null
);
```

Para a tabela listaboi temos as seguintes nomes das colunas com o tipo utilizado.

- numidentificao: inteiro
- datahora: datetime
- peso: float

Utilizando as informações acima, utilizamos o comando abaixo para criar a tabela.

```
CREATE TABLE  listaboi (
numidentificao INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
datahora datetime DEFAULT CURRENT_TIMESTAMP,
peso FLOAT(8,4) null
);
```

Para a tabela parametros temos as seguintes nomes das colunas com o tipo utilizado.

- id: inteiro
- valorracao: float
- valorarroba: float

Utilizando as informações acima, utilizamos o comando abaixo para criar a tabela.

```
CREATE TABLE  parametros (
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
valorracao FLOAT(8,4),
valorarroba FLOAT(8,4) null
);
```

Com as tabelas criadas, podemos dar inicio ao desenvolvimento do código que irá acessar e inserir dados nos mesmos.

4.8.3 Back-End

Para o desenvolvimento do Back-End foram criados alguns arquivos, como mencionado na revisão bibliográfica, que têm funções distintas.

O aquivo 'SQL.php' é responsável por acessar o banco de dados, executar comandos no mesmo e retornar o resultado para que o sistema interprete.

O código abaixo é o básico para o acesso ao banco de dados e foi amplamente utilizado no desenvolvimento da aplicação

```
$dbhost    = 'Host do banco de dados';
$db        = 'tcc';      #Nome do banco de dados
$user      = 'tcc';      #Nome do usuario
$password = 'Senha';    #Senha do usuario

$conn = mysqli_connect($dbhost, $user, $password);

if(! $conn ) {
    die('Could not connect: ' . mysqli_error());
}

mysqli_select_db($conn,"tcc");
$sql = "Comando SQL para consultar o banco de dados";

if ($result=mysqli_query($conn,$sql))
        {
                $i=0;
                while ($row=mysqli_fetch_assoc($result))
                {
                    //Comandos para exibir os dados na tela
                        $i++;
            }
//Free result set
    mysqli_free_result($result);
    }

        mysqli_close($conn);
        return $retorna;
}
```

O arquivo 'Index.php' é o arquivo controlador da aplicação. O mesmo idenfica qual página o usuário vai acessar e redireciona o conteúdo a ser exibido.

O trecho de código utilizado como controlador pode ser visto no código abaixo:

```
if (isset($_GET['menu'])) {
        $controller->index($_GET['menu']);
}
elseif (isset($_GET['id'])){
        $controller->boi($_GET['id']);
}else{
        $controller->index(1);
```

```
}
```

O 'model.php' é arquivo responsável pelas regras do negócio, onde são aplicados cálculos mapeados objetos.

Um trecho de código utilizado que exemplifica esta função está escrito abaixo. O trecho recebe os dados provenientes da consulta do SQL, mapeia e envia para o controlador que por sua vez chama a tela para a exibição do conteúdo.

```
public function boi($idboi=null)
{
    $controller = new Controller;
        $model = new Model;
        if(isset($idboi)){
                $Model->renderlistaboi($idboi);
                return $model;
        }else{
            return "";
        }
}
```

4.8.4 Front-End

O Front-End da aplicação, como mencionado na revisão bibliográfica, é a parte que o usuário irá interagir. Na aplicação do projeto, foi utilizado um *layout* como base chamado *ADMINLTE* [52]. A partir desta estrutura foram criados as estruturas do Menu, a página inicial e a página individual para cada animal.

Menu

O Menu é o componente posicionado no canto esquerdo da tela, sua função é dispor opções acessíveis para o usuário da aplicação. A figura apresenta o Menu desenvolvido.

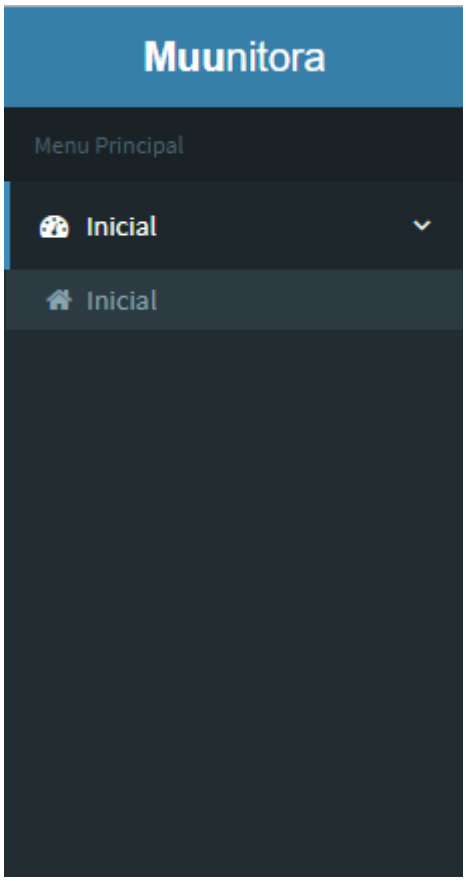

Figura 4.24: Menu da aplicação Fonte: Próprio autor

Pagina Inicial

A página inicial é o conteúdo exibido para o usuário, conforme visto na figura 4.25. A tela é composta por cartões com status dos animais. Desta forma é possível clicar em um dos cartões e será exibido um componente chamado *modal*, que exibe em uma tabela todos os animais com a situação do animal escolhida.

Figura 4.25: Pagina inicial Fonte: Próprio autor

A figura 4.26 apresenta o conteúdo exibido após clicar no cartão com a descrição ”Total”.

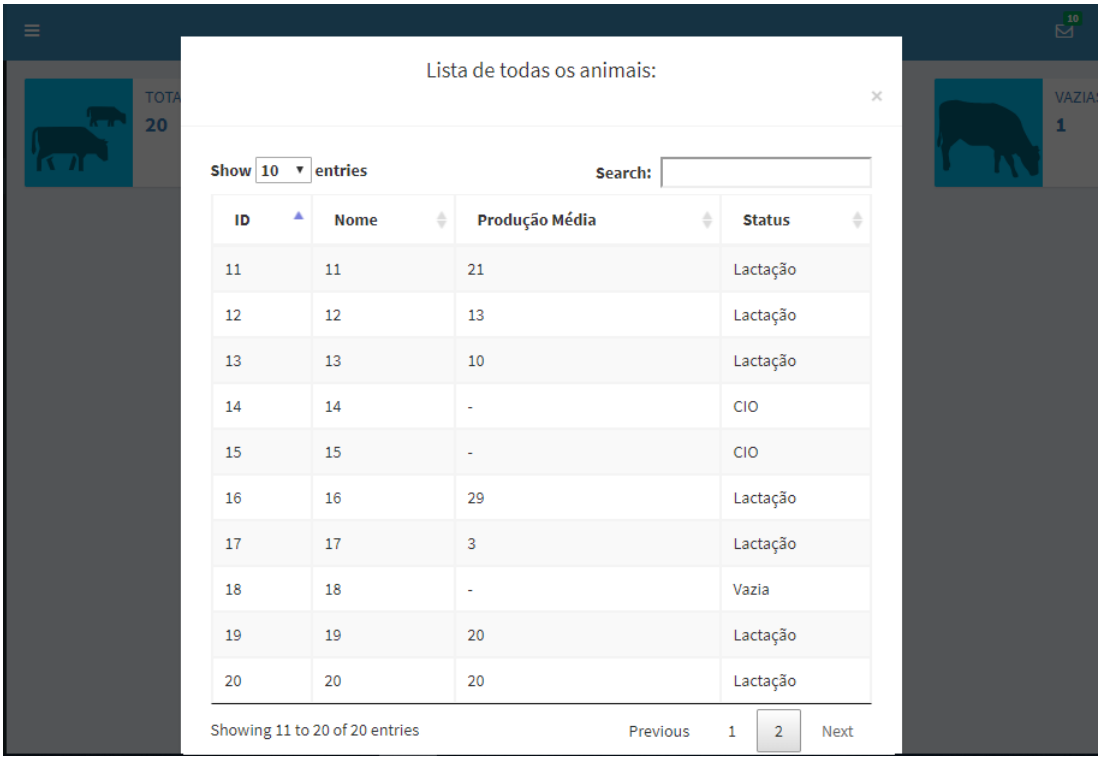

Figura 4.26: Modal Fonte: Próprio autor

Pagina Individual

Ao clicar na identificação do animal, a aplicação redireciona o usuário para a página individual que contém as informações do mesmo. Esta tela contém o gráfico de evolução de peso, além de informações de cadastro do mesmo. A figura 4.27 apresenta a tela informada.

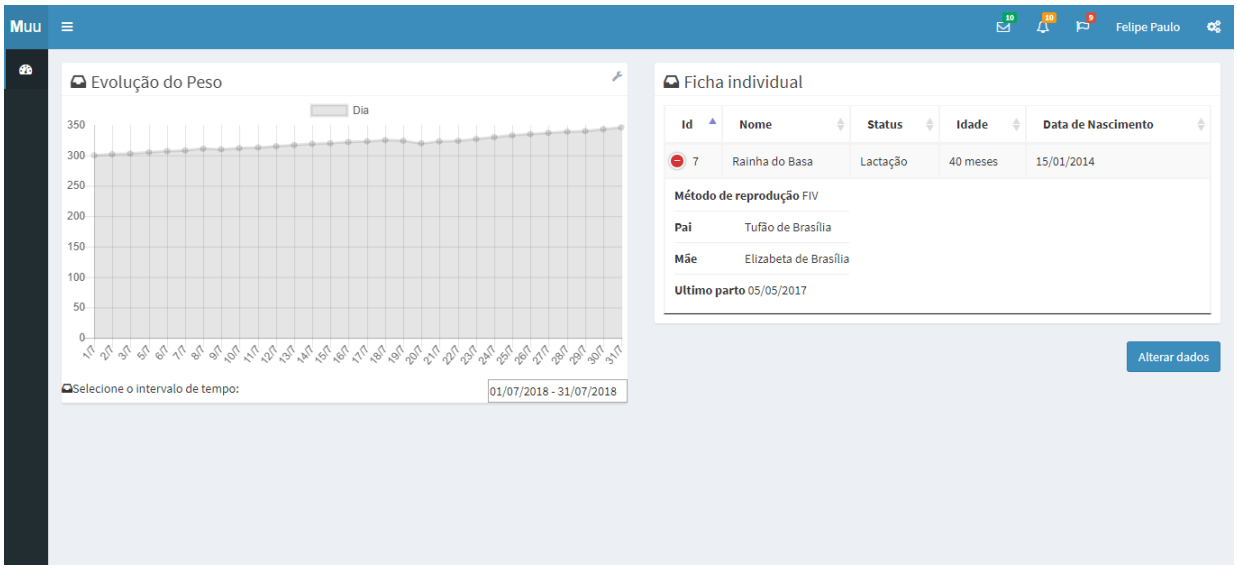

Figura 4.27: Pagina Individual Fonte: Próprio autor

Capítulo 5

Resultados obtidos

5.1 Sinais de medição de peso

O *firmware* da balança deve retornar como resultado um documento de texto no formato *txt*, que posteriormente é processado pela aplicação *web*.

Após os testes realizados com dois pesos fixos, e aproximação manual dos identificadores, foi gerado um arquivo que pode ser visualizado na imagem 5.1.

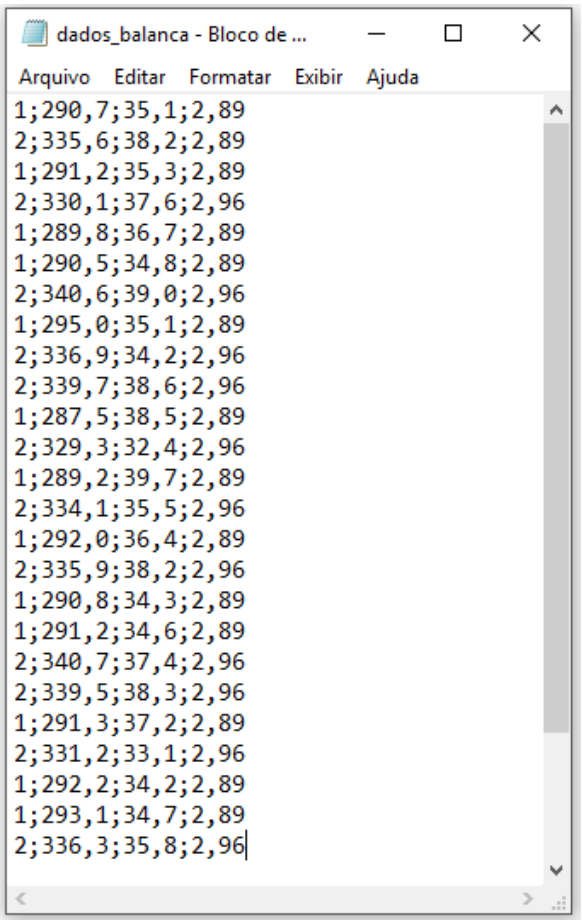

Figura 5.1: Dados de teste Fonte: Próprio autor

O fator de escalada calibrado para os testes foi alterado para que fosse possível observar uma variação maior na pesagem. Desta forma, foi possível observar de maneira representativa o resultado final do sistema de medição de peso.

5.2 Identificação

O *firmware* de identificação individual deve retornar como resultado um pacote de *advertising* contendo a identificação de cada animal, um número aleatório gerado a cada envio de mensagem, a medida de temperatura do microcontrolador e a tensão da bateria do sensor. De forma paralela, o sensor em conjunto com a central deve enviar o valor da potência do sinal, para que seja possível identificar a proximidade ou não do animal. Para facilitar a realização dos testes, foi utilizado um aplicativo para celular que recebe mensagens no protocolo BLE presentes no ambiente. As imagens 5.2 e 5.3 mostram a mensagem presente no pacote de *advertising* de um dos sensores, submetido a uma variação de temperatura para visualização da alteração dos *bytes* de temperatura e valor aleatório.

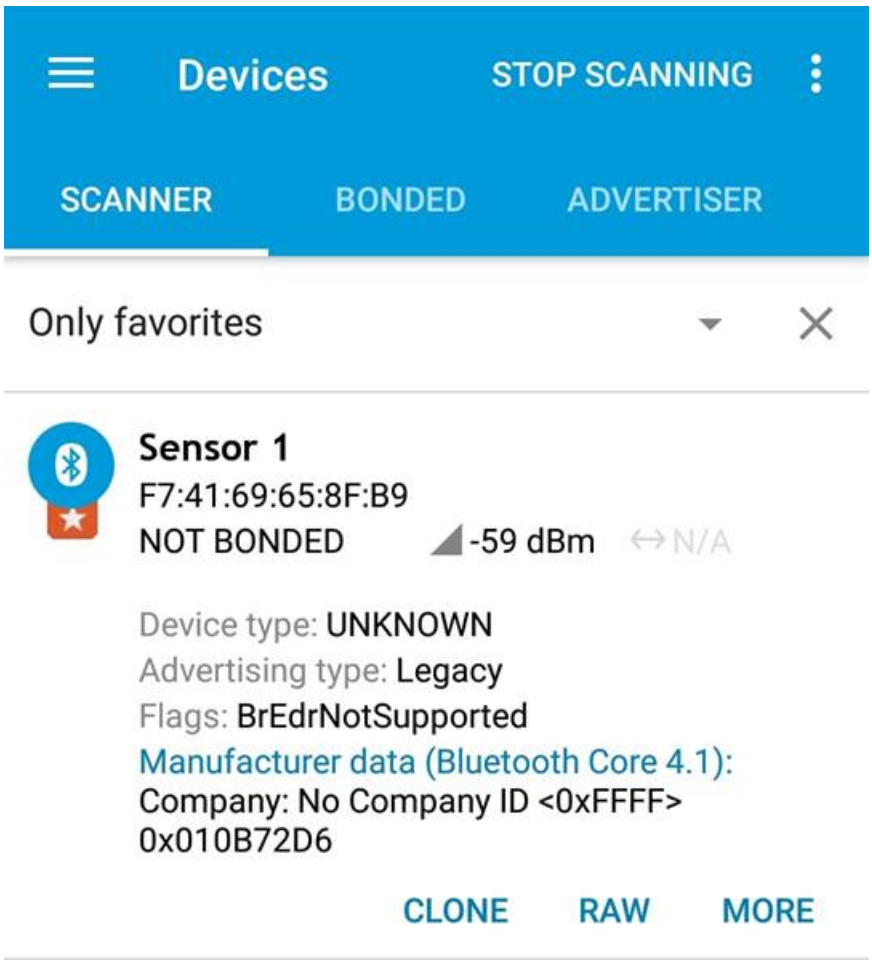

Figura 5.2: Mensagem de *advertising* 1 Fonte: Próprio autor

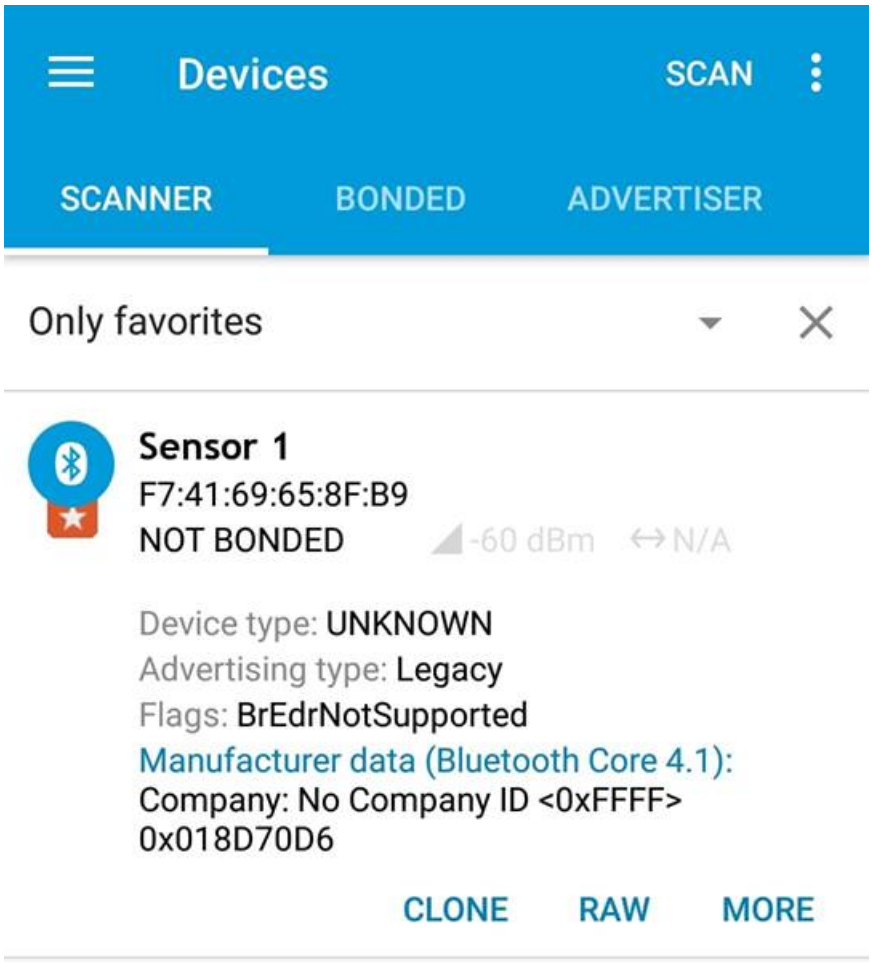

Figura 5.3: Mensagem de *advertising* 2 Fonte: Próprio autor

Os *bytes* da mensagem representam os seguintes valores: ID do animal, valor aleatório, temperatura, tensão na bateria. O valor 0xFFFF utilizado no campo *Company ID* é referente ao ID de desenvolvimento, utilizado por estudantes ou empresas que desenvolvem códigos e não possuem registro ativo na Bluetooth SIG.

Para avaliar a efetividade do filtro de distância baseado na interação sensor-central, tomando como medida o valor de RSSI, foi utilizado o *debugger* do microcontrolador.

A variável 'aux' observada na imagem 5.4 assume dois valores possíveis, 0x3C para sensores a uma distância de até aproximadamente 1,5 metros, e 0x64 para sensores à uma distância superior a 1,5 metros.

Para realizar o teste, foi utilizado um sensor e o valor da variável foi observado em dois instantes, um quando o sensor se encontrava próximo a central e outro quando o sensor se encontrava a uma distância de 2 metros da central.

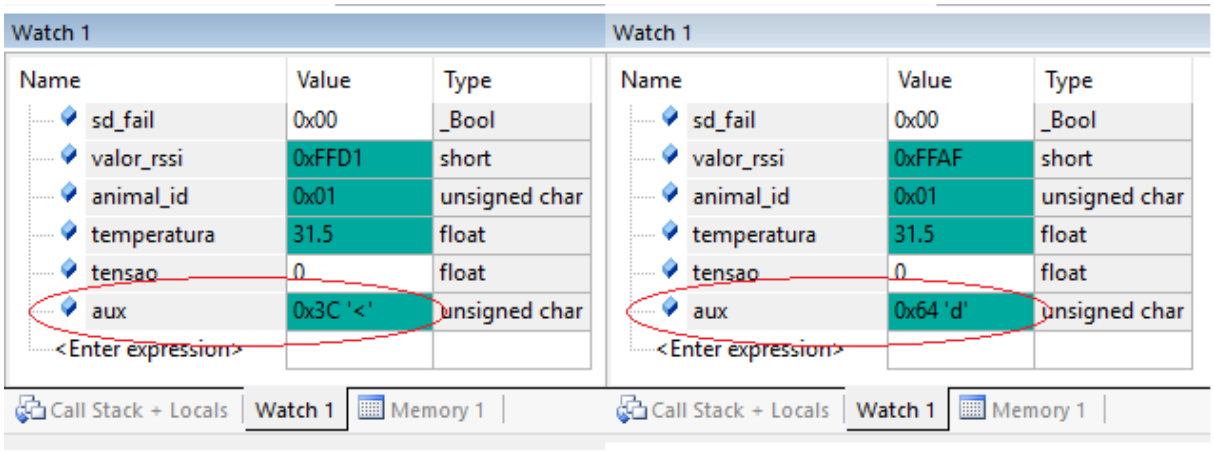

Figura 5.4: Visualização da variável *aux* Fonte: Próprio autor

Como pode ser observado na imagem 5.4, o filtro de distância funciona e é efetivo para determinar qual animal está sendo pesado naquele instante.

5.3 Aplicação Web

Com os dados obtidos da medição e gerado o arquivo *txt* da mesma, a aplicação web interage com o usuário em vários momentos. Nesta seção serão ilustrados as etapas que o sistema percorre para exibição dos dados das pesagens.

Inicialmente é enviado para o sistema o arquivo gerado através da tela representado na figura 5.5.

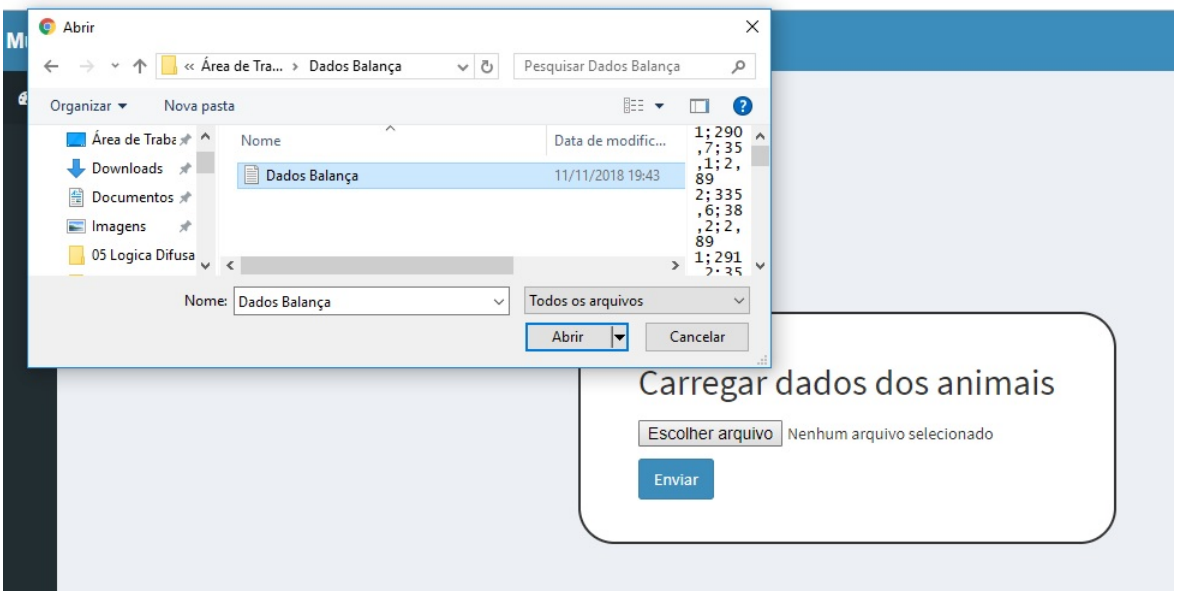

Figura 5.5: Carregar dados na aplicação Fonte: Próprio autor

Após esta etapa o sistema recebe os dados, interpreta e salva os valores no banco de

dados, conforme a figura 5.6.

```
MariaDB [tcc]> select * from webapp;
+----+--------+--------+-------------+---------+
| id | Codigo | Peso   | Temperatura | Bateria |
+----+--------+--------+-------------+---------+
|  1 |      1 | 290.70 |       35.10 |    2.89 |
|  2 |      2 | 335.60 |       38.20 |    2.89 |
|  3 |      1 | 291.20 |       35.30 |    2.89 |
|  4 |      2 | 330.10 |       37.60 |    2.96 |
|  5 |      1 | 289.80 |       36.70 |    2.89 |
|  6 |      1 | 290.50 |       34.80 |    2.89 |
|  7 |      2 | 340.60 |       39.00 |    2.96 |
|  8 |      1 | 295.00 |       35.10 |    2.89 |
|  9 |      2 | 336.90 |       34.20 |    2.96 |
| 10 |      2 | 339.70 |       38.60 |    2.96 |
| 11 |      1 | 287.50 |       38.50 |    2.89 |
| 12 |      2 | 329.30 |       32.40 |    2.96 |
| 13 |      1 | 289.20 |       39.70 |    2.89 |
| 14 |      2 | 334.10 |       35.50 |    2.96 |
| 15 |      1 | 292.00 |       36.40 |    2.89 |
| 16 |      2 | 335.90 |       38.20 |    2.96 |
| 17 |      1 | 290.80 |       34.30 |    2.89 |
| 18 |      1 | 291.20 |       34.60 |    2.89 |
| 19 |      2 | 340.70 |       37.40 |    2.96 |
| 20 |      2 | 339.50 |       38.30 |    2.96 |
| 21 |      1 | 291.30 |       37.20 |    2.89 |
| 22 |      2 | 331.20 |       33.10 |    2.96 |
| 23 |      1 | 292.20 |       34.20 |    2.89 |
| 24 |      1 | 293.10 |       34.70 |    2.89 |
| 25 |      2 | 336.30 |       35.80 |    2.96 |
| 26 |      1 | 290.70 |       32.20 |    2.89 |
```

Figura 5.6: Dados armazenados no Banco de dados Fonte: Próprio autor

Uma vez cadastrado no banco de dados, a aplicação fará a leitura dos animais que têm registros cadastrados e listará os animais no cartão ”TOTAL”.

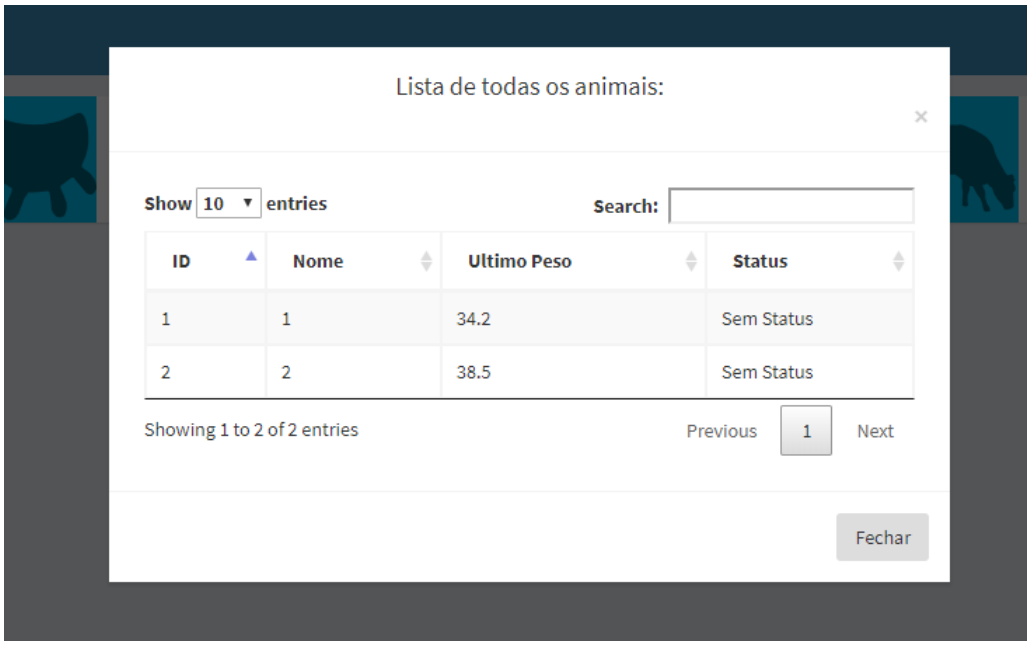

Figura 5.7: Listagem de animais Fonte: Próprio autor

Ao selecionar o animal, o sistema exibirá o gráfico com os pesos registrados e assim o produtor poderá analisar se há uma tendência de estabilizar a evolução de peso ou mesmo se a alimentação aplicada está dando os resultados desejados.

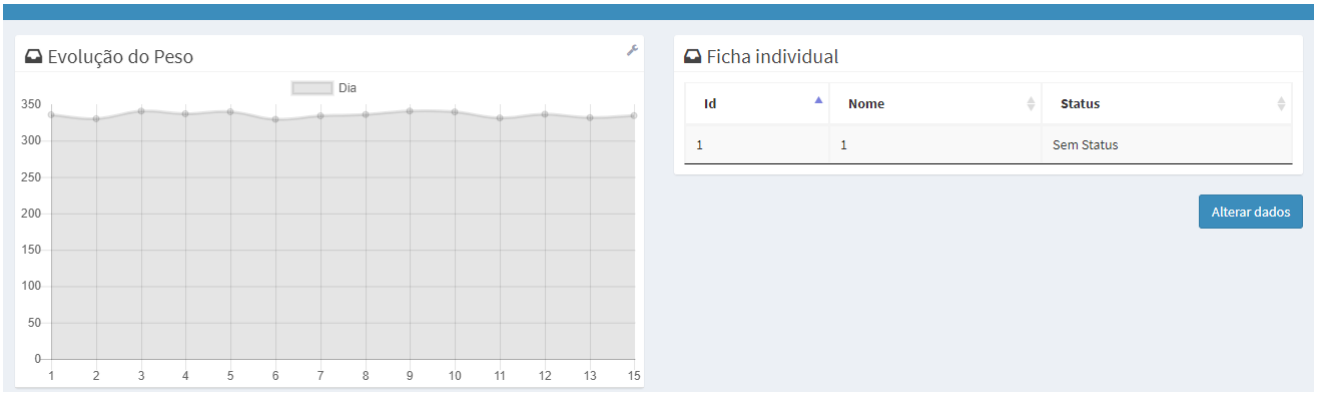

Figura 5.8: Dashboard individual Fonte: Próprio autor

5.4 Custos

Nesta seção será apresentado os custos relacionados a construção do projeto ideal, levando em consideração a balança para capacidade máxima e o cocho.

Tabela 5.1: Tabela de custos balança Fonte: Próprio autor

Quantidade	Unidade	Material	Preço (Real)
1	150mm	Tubo 1”	3,75
1	150mm	Tubo 3/8”	3,88
2	200mm	Tubo 1”	5,00
1	200x1200x1mm	Chapa de Aço	115,2
1	40x80x1800x3mm	Tubo de aço inox	270,00
1	100x150x5mm	Chapa de Aço	36,00
4	40x80x3x950mm	Tubo de aço inox	140,00
1	Peça	Cocho Plástico	180,00
8	10 m	Cabo de aço	30,00
4	40x80x600x3mm	Tubo de aço inox	90,00
1	Peça	Célula de carga 500Kg	330,00
1	Peça	Célula de carga 50Kg	20,00
2	Peça	Módulo Hx711	10,00
2	Peça	Placa Microcontrolador	80,00
2	Peça	Fonte de alimentação 5v	30,00
		Total	2368,83

A tabela 5.2 mostra o custo de implementação para cada animal a ser monitorado.

Tabela 5.2: Tabela de custos identificação Fonte: Próprio autor

Unidade	Material	Preço (Real)
Peça	Placa Microcontrolador	80,00
Peça	Bateria Cr2032	1,50
Peça	Caixa protetora	5,00
	Total	86,50

Capítulo 6

Considerações Finais

Neste trabalho foi desenvolvido um sistema mecatrônico que utilizou as áreas da engenharia eletrônica, mecânica e computação. Inicialmente o trabalho consistiu na construção de uma balança para gado criado em regime de confinamento.

Durante o desenvolvimento do projeto, foram encontradas algumas dificuldades técnicas que não puderam ser resolvidas a tempo e serão propostas como trabalhos futuros.

Uma das dificuldades encontradas foi a de construção da estrutura das balanças. O alto custo proveniente da célula de carga e da quantidade de aço utilizado na mesma impediu a aquisição dos materiais. As dimensões e peso do projeto também impossibilitou a construção das balanças. Portanto, não foi possível implementar nenhum dos dois projetos, sendo realizadas medições e validações em um protótipo de baixa fidelidade, que tem como objetivo demonstrar o funcionamento da parte eletrônica, computacional do sistema e a transmissão de forças da alavanca, não a parte estrutural mecânica do projeto.

A construção do protótipo da balança sofreu com problemas estruturais devido a maleabilidade das juntas de PVC, dessa forma, não foi possível simular um evento real de alto impacto gerado no movimento de subida e descida do animal na balança. Foram realizados testes colocando e retirando a carga da plataforma, de maneira rápida para simular uma dinâmica rápida de pesagem.

Tendo em vista as limitações apresentadas na execução do protótipo, os filtros descritos na fundamentação teórica e a implementação dos mesmos não foi necessária, devido a alta velocidade de resposta do sensor utilizado.

Com o trabalho realizado, foram desenvolvidos os *firmwares* e o *Web App* que em conjunto permitem, de forma simples e rápida, o acompanhamento do desenvolvimento e alterações de padrão de alimentação de bovinos criados em regime de confinamento.

Além de serem utilizados de forma integrada, os dois *firmwares* podem ser utilizados de maneira independente a partir de algumas mudanças. O código desenvolvido implementa

conceitos de microcontroladores e métodos de comunicação e tratamento de dados que podem ser reaproveitados para outros desenvolvimentos.

O código do sensor utiliza periféricos úteis a outras aplicações e desenvolve o protocolo BLE de maneira simples, podendo ser utilizado como fonte de estudo para outros projetos. O protocolo implementado é de grande utilidade para projetos que tenham como premissa a alta eficiência energética, tendo uma boa eficiência em processos que tenham como objetivo a aquisição de poucos dados, com o objetivo de alimentar dados para decisões com base estatística.

O código da central utiliza periféricos diferentes dos tratados no sensor, e possibilita a utilização do microcontrolador no modo *observer*, que pode ser utilizado como *scanner* para projetos de automação.

O microcontrolador utilizado foi subutilizado, tendo um poder de processamento muito superior ao necessário para o projeto e com uma eficiência enérgica melhor que outros MCUs utilizados atualmente. Desta forma, o trabalho realizado traz uma opção diferente das convencionais para desenvolvimento de soluções embarcadas.

A aplicação *Web* teve seu desenho alterado devido a opção de atender o sistema para outros mercados. Porém, o mesmo se comportou como descrito no desenvolvimento do projeto.

Como proposta de trabalhos futuros, existe a possibilidade de se desenvolver uma plataforma mais robusta, com elementos e uma geometria que permita maior aplicação de esforço. Além deste, a implementação de ferramentas de ciência de dados no aplicativo *web*, assim como uma melhora estética para tornar o sistema mais acessível ao usuário são possibilidades para a construção de um produto com maior valor agregado.

Apêndice A

Código do sensor de identificação individual

```
/*
Codigo Sensor TCC
Autores:                Vitor Cesar Tavares
                                Felipe Paulo Tavares de Oliveira
Orientador:     Luiz Claudio de Oliveira
*/

#include <stdbool.h>
#include <stdint.h>
#include "ble_advdata.h"
#include "nordic_common.h"
#include "softdevice_handler.h"
#include "app_timer.h"
#include "nrf_log.h"
#include "nrf_log_ctrl.h"
//Sd
#include "nrf.h"
#include "bsp.h"



#ifndef NRF52832
        #include "nrf_drv_adc.h"
#else
        #include "nrf_drv_saadc.h"
        #include "app_util_platform.h"
#endif

#define NRF_TX_POWER       4

                                                /**< Radio Power for Tx
    packets in dB*/
#define CENTRAL_LINK_COUNT         0
                                     /**< ? Number of central links
   used by the application. When changing this number remember to
   adjust the RAM settings*/
#define PERIPHERAL_LINK_COUNT          0
```

```
                                    /**< ? Number of peripheral
    links used by the application. When changing this number
    remember to adjust the RAM settings*/
32
33  #define IS_SRVC_CHANGED_CHARACT_PRESENT 0
                                    /**< ? Include or not the
    service_changed characteristic. if not enabled, the server's
    database cannot be changed for the lifetime of the device*/
34
35  #define APP_CFG_NON_CONN_ADV_TIMEOUT    5
                                    /**< Time for which the device
    must be advertising in non-connectable mode (in seconds). 0
    disables timeout. */
36  #define NON_CONNECTABLE_ADV_INTERVAL    MSEC_TO_UNITS(500,
    UNIT_0_625_MS) /**< The advertising interval for non-
    connectable advertisement (100 ms). This value can vary between
     100ms to 10.24s). */
37
38
39  #define APP_COMPANY_IDENTIFIER          0xFFFF
                                /**< Company identifier for
    developer*/
40
41  #define ANIMAL_ID
                                    0x01
42
43  #define DEAD_BEEF                       0xDEADBEEF
                            /**< Value used as error code on stack
    dump, can be used to identify stack location on stack unwind.
    */
44
45  #define APP_TIMER_PRESCALER             0
                                    /**< Value of the RTC1
    PRESCALER register. */
46  #define APP_TIMER_OP_QUEUE_SIZE         4
                                    /**< Size of timer operation
    queues. */
47
48  #define TEMP_SAMPLE_PERIOD
                      500

                             /**< Temperature Sample Period*/
49
50  #define NRF_CLOCK_LFCLKSRC  {.source=NRF_CLOCK_LF_SRC_XTAL,
51                               .rc_ctiv       = 0,
52                              .rc_temp_ctiv  = 0,
53                              .xtal_accuracy =
                                   NRF_CLOCK_LF_XTAL_ACCURACY_20_PPM}
54
55  // Handle of Temperature Aquisition Timer
56  APP_TIMER_DEF(temp_aq_timer_id);
```

```
57
58
59  #ifndef NRF52832
60  // Handle of ADC configuration
61  static nrf_drv_adc_channel_t adc_channel_config; /**< Channel
        instance*/
62  #endif
63
64  // Advertising global variables
65  static bool advertising_is_off=true;
66  static ble_advdata_t advdata;
67  static ble_advdata_manuf_data_t manuf_specific_data;
68  static ble_gap_adv_params_t m_adv_params;
69
70  // Advertising data structure
71  uint8_t m_adv_data[]={
72          ANIMAL_ID,
73          0x13,//randon key
74          0,
75          33// battery voltage
76  };
77
78  //-----------------------------------------------
79  // Prototypes of functions
80
81  static void advertising_start(void);
82  static void advertising_init(void);
83
84  //-------------------------------------
85  #ifdef NRF52832
86          void saadc_callback(nrf_drv_saadc_evt_t const * p_event){}
87  #endif
88
89  /** @brief ADC initialization.*/
90  static void adc_init(void)
91  {
92          #ifndef NRF52832
93      ret_code_t err_code;
94      nrf_drv_adc_config_t adc_config = NRF_DRV_ADC_DEFAULT_CONFIG;
95
96      err_code = nrf_drv_adc_init(&adc_config, NULL);
97      APP_ERROR_CHECK(err_code);
98
99                  adc_channel_config.config.config.reference =
                        NRF_ADC_CONFIG_REF_VBG;
100                 adc_channel_config.config.config.input=
                        NRF_ADC_CONFIG_SCALING_SUPPLY_ONE_THIRD;
101                 adc_channel_config.config.config.resolution=
                        NRF_ADC_CONFIG_RES_8BIT;
102
103     nrf_drv_adc_channel_enable(&adc_channel_config);
```

```
104         #else
105                 ret_code_t err_code;
106     nrf_drv_saadc_config_t saadc_config;
107     nrf_saadc_channel_config_t channel_config;
108 
109 
110     //Configure SAADC
111     saadc_config.low_power_mode = true;
112     saadc_config.resolution = NRF_SAADC_RESOLUTION_8BIT;
113     saadc_config.oversample = NRF_SAADC_OVERSAMPLE_DISABLED;
114     saadc_config.interrupt_priority = APP_IRQ_PRIORITY_LOW;
115 
116     //Initialize SAADC
117     err_code = nrf_drv_saadc_init(&saadc_config, saadc_callback);
118     APP_ERROR_CHECK(err_code);
119 
120     //Configure SAADC channel
121     channel_config.reference = NRF_SAADC_REFERENCE_INTERNAL;
122     channel_config.gain = NRF_SAADC_GAIN1_6;
123     channel_config.acq_time = NRF_SAADC_ACQTIME_40US;
124     channel_config.mode = NRF_SAADC_MODE_SINGLE_ENDED;
125     channel_config.pin_p = NRF_SAADC_INPUT_VDD;
126     channel_config.pin_n = NRF_SAADC_INPUT_DISABLED;
127     channel_config.resistor_p = NRF_SAADC_RESISTOR_DISABLED;
128     channel_config.resistor_n = NRF_SAADC_RESISTOR_DISABLED;
129 
130 
131     //Initialize SAADC channel
132     err_code = nrf_drv_saadc_channel_init(0, &channel_config);
                                    //Initialize SAADC channel 0
       with the channel configuration
133     APP_ERROR_CHECK(err_code);
134         #endif
135 }
136 //-----------------------------------------
137 /** @brief Internal Temperature sensor data get*/
138 uint8_t temperature_data_get(void)
139 {
140     int32_t temp;
141 
142     uint32_t err_code = sd_temp_get(&temp);
143     APP_ERROR_CHECK(err_code);
144 
145     temp = (temp>>1)+50;
146 
147                 // if temperature below -25 Celsius saturate at
                       -25 Celsius if temp over 102.5 Celsius saturate
                        at 102.5 Celsius
148                 if(temp<0)temp=0;
149                 else if(temp>255) temp=255;
150 
```

```
151     return (int8_t)temp;
152 }
153 //-----------------------------------------
154 /** @brief Timeout handler for the repeated timer */
155 static void temp_aq_handler(void * p_context)
156 {
157         uint32_t err_code;
158         int16_t voltage;
159 
160         // Fill data buffer
161         m_adv_data[2]=temperature_data_get();
162 
163         adc_init();
164 
165         // Set randon key in data packet
166         err_code = sd_rand_application_vector_get(&m_adv_data
            [1],1);
167         APP_ERROR_CHECK(err_code);
168 
169         // Set battery voltage
170         #ifndef NRF52832
171                 err_code = nrf_drv_adc_sample_convert(&
                    adc_channel_config,&voltage);
172         #else
173                 err_code = nrf_drv_saadc_sample_convert(0,&voltage
                    );
174         #endif
175 
176         NRF_LOG_INFO("Er ADC %d Val %d\r\n",err_code,voltage
            *36/255);//debug
177         APP_ERROR_CHECK(err_code);
178         m_adv_data[3]=(uint8_t)voltage;
179 
180         // Uninit ADC module to save battery
181         #ifndef NRF52832
182                 nrf_drv_adc_uninit();
183         #else
184                 nrf_drv_saadc_uninit();
185         #endif
186 
187         //Update advertising packet
188         err_code = ble_advdata_set(&advdata, NULL);
189         APP_ERROR_CHECK(err_code);
190 
191         // Verify advertising state and start
192         if(advertising_is_off){
193                 advertising_start();
194                 advertising_is_off=false;
195         }
196 }
197 //-----------------------------------------
```

```
198 /** @brief Create timers */
199 static void create_timers()
200 {
201     uint32_t err_code;
202 
203     // Create timers
204 
205     err_code = app_timer_create(&temp_aq_timer_id,
206                                 APP_TIMER_MODE_REPEATED,
207                                 temp_aq_handler);
208     APP_ERROR_CHECK(err_code);
209 
210                 err_code = app_timer_start(temp_aq_timer_id,
                       APP_TIMER_TICKS(TEMP_SAMPLE_PERIOD,
                       APP_TIMER_PRESCALER), NULL);
211     APP_ERROR_CHECK(err_code);
212 
213 }
214 //------------------------------------------------
215 /**@brief Callback function for asserts in the SoftDevice.
216  *
217  * @details This function will be called in case of an assert in
        the SoftDevice.
218  *
219  * @warning This handler is an example only and does not fit a
        final product. You need to analyze
220  *          how your product is supposed to react in case of
        Assert.
221  * @warning On assert from the SoftDevice, the system can only
        recover on reset.
222  *
223  * @param[in]   line_num   Line number of the failing ASSERT call.
224  * @param[in]   file_name  File name of the failing ASSERT call.
225  */
226 void assert_nrf_callback(uint16_t line_num, const uint8_t *
       p_file_name)
227 {
228     app_error_handler(DEAD_BEEF, line_num, p_file_name);
229 }
230 //------------------------------------------------
231 /**@brief Function for initializing the Advertising functionality.
232  *
233  * @details Encodes the required advertising data and passes it to
        the stack.
234  *          Also builds a structure to be passed to the stack when
        starting advertising.
235  */
236 static void advertising_init(void)
237 {
238     uint32_t      err_code;
239 
```

```
240                 manuf_specific_data.company_identifier =
                       APP_COMPANY_IDENTIFIER;
241
242     manuf_specific_data.data.p_data = (uint8_t *) m_adv_data;
243     manuf_specific_data.data.size   = sizeof(m_adv_data);
244
245     // Build and set advertising data.
246     memset(&advdata, 0, sizeof(advdata));
247
248     advdata.name_type               = BLE_ADVDATA_NO_NAME;
249     advdata.flags                   =
           BLE_GAP_ADV_FLAG_BR_EDR_NOT_SUPPORTED;
250     advdata.p_manuf_specific_data = &manuf_specific_data;
251
252     err_code = ble_advdata_set(&advdata, NULL);
253     APP_ERROR_CHECK(err_code);
254
255     // Initialize advertising parameters (used when starting
           advertising).
256     memset(&m_adv_params, 0, sizeof(m_adv_params));
257
258     m_adv_params.type        = BLE_GAP_ADV_TYPE_ADV_NONCONN_IND;
259     m_adv_params.p_peer_addr = NULL;
           // Undirected advertisement.
260     m_adv_params.fp          = BLE_GAP_ADV_FP_ANY;
261     m_adv_params.interval    = NON_CONNECTABLE_ADV_INTERVAL;
262     m_adv_params.timeout     = APP_CFG_NON_CONN_ADV_TIMEOUT;
263
264 }
265
266 //----------------------------------------------
267 /**@brief Function for starting advertising.*/
268 static void advertising_start(void)
269 {
270     uint32_t err_code;
271
272     err_code = sd_ble_gap_adv_start(&m_adv_params);
273                 NRF_LOG_INFO("Er ADV St %d\r\n",err_code);
274     APP_ERROR_CHECK(err_code);
275
276 }
277
278 //----------------------------------------------
279 /**@brief Function for the application's SoftDevice event handler.
280  *
281  * @param[in] p_ble_evt SoftDevice event.
282  */
283 static void on_ble_evt(ble_evt_t * p_ble_evt)
284 {
285    // uint32_t err_code;
286
```

```
287     switch (p_ble_evt->header.evt_id)
288     {
289         case BLE_GAP_EVT_TIMEOUT:
290             if (p_ble_evt->evt.gap_evt.params.timeout.src ==
                    BLE_GAP_TIMEOUT_SRC_ADVERTISING)
291             {
292                                 advertising_is_off=true;
293                         }
294                         break;
295                 }
296 }
297
298 //----------------------------------------------
299 /**@brief Function for initializing the BLE stack.
300  *
301  * @details Initializes the SoftDevice and the BLE event interrupt
        .
302  */
303 static void ble_stack_init(void)
304 {
305     uint32_t err_code;
306
307     nrf_clock_lf_cfg_t clock_lf_cfg = NRF_CLOCK_LFCLKSRC;
308
309     // Initialize the SoftDevice handler module.
310     SOFTDEVICE_HANDLER_INIT(&clock_lf_cfg, NULL);
311
312     ble_enable_params_t ble_enable_params;
313     err_code = softdevice_enable_get_default_config(
           CENTRAL_LINK_COUNT,PERIPHERAL_LINK_COUNT,&ble_enable_params
           );
314     APP_ERROR_CHECK(err_code);
315
316     //Check the ram settings against the used number of links
317     CHECK_RAM_START_ADDR(CENTRAL_LINK_COUNT,PERIPHERAL_LINK_COUNT)
           ;
318
319     // Enable BLE stack.
320     err_code = softdevice_enable(&ble_enable_params);
321     APP_ERROR_CHECK(err_code);
322
323                 err_code = softdevice_ble_evt_handler_set(
                       on_ble_evt);
324     APP_ERROR_CHECK(err_code);
325 }
326
327 //----------------------------------------------
328 /**@brief Function for doing power management.
329  */
330 static void power_manage(void)
331 {
```

```
332         #if (__FPU_USED == 1)
333                 __set_FPSCR(__get_FPSCR() & ~(0x0000009F));
334                 (void) __get_FPSCR();
335                 NVIC_ClearPendingIRQ(FPU_IRQn);
336         #endif
337
338     uint32_t err_code = sd_app_evt_wait();
339     APP_ERROR_CHECK(err_code);
340 }
341
342 //-------------------------------------------------
343 /*** @brief Function for application main entry.
344  */
345 int main(void)
346 {
347     uint32_t err_code;
348     // Initialize.
349     err_code = NRF_LOG_INIT(NULL);
350     APP_ERROR_CHECK(err_code);
351
352     APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_OP_QUEUE_SIZE,
        false);
353
354                 // Init ble stack
355     ble_stack_init();
356
357                 // Set Radio TX Power
358                 err_code = sd_ble_gap_tx_power_set(NRF_TX_POWER);
359                 APP_ERROR_CHECK(err_code);
360
361                 // Init the advertising structure, don't start the
                        advertising
362     advertising_init();
363
364                 //Enable DCDC regulator
365                 //The DCDC regulator is a more efficient voltage
                       regulator topology for some microcontroller
                       applications, ensuring lower current
                       consumption
366                 err_code = sd_power_dcdc_mode_set(
                       NRF_POWER_DCDC_ENABLE);
367                 APP_ERROR_CHECK(err_code);
368
369
370     // Start execution.
371     NRF_LOG_INFO("TCC sensor start\r\n");
372
373                 // Create timer for temperature aquisition
374                 create_timers();
375
376     // Enter main loop.
```

```
377     for (;;)
378     {
379         if (NRF_LOG_PROCESS() == false)
380         {
381             power_manage();
382         }
383     }
384 }
385 /**
386  * @}
387  */
```

Apêndice B

Código da central de pesagem e controle

```
/*
Codigo Central TCC
Autores:                Vitor Cesar Tavares
                              Felipe Paulo Tavares de Oliveira
Orientador:     Luiz Claudio de Oliveira
*/

#include <stdbool.h>
#include <stdint.h>
#include "ble_advdata.h"
#include "nordic_common.h"
#include "softdevice_handler.h"
#include "bsp.h"
#include "app_timer.h"
#include "nrf_log.h"
#include "nrf_log_ctrl.h"

//SD card includes
#include "ff.h"
#include "diskio_blkdev.h"
#include "nrf_block_dev_sdc.h"

//RTC includes (talvez falte o drv_clock)
#include "nrf_drv_rtc.h"

#include "app_uart.h"

#include "nrf_drv_wdt.h"



#define CENTRAL_LINK_COUNT              0
                                    /**< Number of central links
   used by the application. When changing this number remember to
   adjust the RAM settings*/
#define PERIPHERAL_LINK_COUNT           0
                                    /**< Number of peripheral links
```

```
    used by the application. When changing this number remember to
    adjust the RAM settings*/
34
35  #define APP_TIMER_PRESCALER                 0
                                        /**< Value of the RTC1
    PRESCALER register. */
36  #define APP_TIMER_OP_QUEUE_SIZE             4
                                        /**< Size of timer operation
    queues. */
37
38  #define SAMPLE_PERIOD
                                1000

                                        /**< Timer events period*/
39
40  #define TIME_FACTOR                         (1000/SAMPLE_PERIOD)
41
42  #define THIS_CENTRAL
                                        02
43  #define  NUMBER_OF_MODULES
                        8
44
45  #define DEAD_BEEF                           0xDEADBEEF
                                /**< Value used as error code on stack
    dump, can be used to identify stack location on stack unwind.
    */
46
47  #define APP_COMPANY_IDENTIFIER              0xFFFF
                                    /**< Company identifier for
    developer*/
48
49  //SD card pins
50  #define SDC_SCK_PIN     18//18  ///< SDC serial clock (SCK) pin.
51  #define SDC_MOSI_PIN    17//17  ///< SDC serial data in (DI) pin.
52  #define SDC_MISO_PIN    16//16  ///< SDC serial data out (DO) pin.
53  #define SDC_CS_PIN      19//19  ///< SDC chip select (CS) pin.
54
55  //SD card buffer length
56  #define TCC_DATA_LEN    80
57
58  //scan parameters
59  #define SCAN_INTERVAL                                       8000
                                         /**< Determines scan interval
    in units of 0.625 millisecond. */
60  #define SCAN_WINDOW                                         1500
                                         /**< Determines scan window in
     units of 0.625 millisecond. */
61  #define SCAN_TIMEOUT                                  0x0000
                                       /**< Timout when scanning. 0
    x0000 disables timeout. */
62
```

```
63 //UART
64 #define UART_TX_BUF_SIZE 256                          /**< UART TX
      buffer size. */
65 #define UART_RX_BUF_SIZE 256                          /**< UART RX
      buffer size. */
66 #define TCC_RX_PIN          15//15
67 #define TCC_TX_PIN          14//21//14
68
69
70 //Vector for data validation
71 int data_validation[255];
72
73 // Handle of Aquisition Timer
74 APP_TIMER_DEF(aq_timer_id);
75
76
77 nrf_drv_wdt_channel_id m_channel_id;
78
79 //buffers for write in SD card
80 char sd_buffer[TCC_DATA_LEN];
81
82 uint8_t sd_buf_len=0;
83
84 char sd_file_name[]="TCC.TXT";
85
86 //
87 bool can_get_weight=false;
88 bool sd_write_able=false;
89 bool sd_central_able=false;
90
91 bool sd_fail=false;
92
93 //times for print with temperature
94 int time_h=0;
95 int time_m=0;
96 uint8_t time_s=0;
97 int tick=0;
98
99
100 //variaveis TCC
101 float peso;
102 float tensao;
103 float temperatura;
104 uint8_t animal_id;
105 int16_t valor_rssi=0;
106 int16_t aux=0;
107
108 /**
109  * @brief  SDC block device definition
110  * */
111 NRF_BLOCK_DEV_SDC_DEFINE(
```

```
112         m_block_dev_sdc,
113         NRF_BLOCK_DEV_SDC_CONFIG(
114                 SDC_SECTOR_SIZE,
115                 APP_SDCARD_CONFIG(SDC_MOSI_PIN, SDC_MISO_PIN,
                       SDC_SCK_PIN, SDC_CS_PIN)
116          ),
117          NFR_BLOCK_DEV_INFO_CONFIG("Hedro", "Central", "1.00")
118 );
119
120
121 void wdt_event_handler(void)
122 {
123         APP_ERROR_CHECK(100);
124
125 }
126 void wdt_config()
127 {
128                 uint32_t err_code;
129     //Configure WDT.
130     nrf_drv_wdt_config_t config = NRF_DRV_WDT_DEAFULT_CONFIG;
131     err_code = nrf_drv_wdt_init(&config, wdt_event_handler);
132     APP_ERROR_CHECK(err_code);
133     err_code = nrf_drv_wdt_channel_alloc(&m_channel_id);
134     APP_ERROR_CHECK(err_code);
135     nrf_drv_wdt_enable();
136 }
137 /**
138  * @brief Parameters used when scanning.
139  */
140 static const ble_gap_scan_params_t m_scan_params =
141 {
142     .active   = 1,
143     .interval = SCAN_INTERVAL,
144     .window   = SCAN_WINDOW,
145     .timeout  = SCAN_TIMEOUT,
146     #if (NRF_SD_BLE_API_VERSION == 2)
147         .selective   = 0,
148         .p_whitelist = NULL,
149     #endif
150     #if (NRF_SD_BLE_API_VERSION == 3)
151         .use_whitelist = 0,
152     #endif
153 };
154
155 /**@brief Callback function for asserts in the SoftDevice.
156  *
157  * @details This function will be called in case of an assert in
       the SoftDevice.
158  *
159  * @warning This handler is an example only and does not fit a
       final product. You need to analyze
```

```
160  *          how your product is supposed to react in case of
        Assert.
161  * @warning On assert from the SoftDevice, the system can only
        recover on reset.
162  *
163  * @param[in]   line_num   Line number of the failing ASSERT call.
164  * @param[in]   file_name  File name of the failing ASSERT call.
165  */
166 void assert_nrf_callback(uint16_t line_num, const uint8_t *
        p_file_name)
167 {
168     app_error_handler(DEAD_BEEF, line_num, p_file_name);
169 }
170
171 /**
172  * @brief Function for demonstrating FAFTS usage.
173  */
174 static void write_data_sd()
175 {
176     static FATFS fs;
177     static DIR dir;
178     static FILINFO fno;
179     static FIL file;
180
181     uint32_t bytes_written;
182     FRESULT ff_result;
183     DSTATUS disk_state = STA_NOINIT;
184
185     // Initialize FATFS disk I/O interface by providing the block
            device.
186     static diskio_blkdev_t drives[] =
187     {
188             DISKIO_BLOCKDEV_CONFIG(NRF_BLOCKDEV_BASE_ADDR(
                    m_block_dev_sdc, block_dev), NULL)
189     };
190
191     diskio_blockdev_register(drives, ARRAY_SIZE(drives));
192
193     NRF_LOG_INFO("Initializing disk 0 (SDC)...\r\n");
194     for (uint32_t retries = 3; retries && disk_state; --retries)
195     {
196         disk_state = disk_initialize(0);
197     }
198     if (disk_state)
199     {
200                                     sd_fail=true;
201         NRF_LOG_INFO("Disk initialization failed.\r\n");
202         return;
203     }
204
205     uint32_t blocks_per_mb = (1024uL * 1024uL) / m_block_dev_sdc.
```

```
        block_dev.p_ops->geometry(&m_block_dev_sdc.block_dev)->
        blk_size;
206     uint32_t capacity = m_block_dev_sdc.block_dev.p_ops->geometry
        (&m_block_dev_sdc.block_dev)->blk_count / blocks_per_mb;
207     NRF_LOG_INFO("Capacity: %d MB\r\n", capacity);
208
209     NRF_LOG_INFO("Mounting volume...\r\n");
210     ff_result = f_mount(&fs, "", 1);
211     if (ff_result)
212     {
213         NRF_LOG_INFO("Mount failed.\r\n");
214         return;
215     }
216
217     NRF_LOG_INFO("\r\n Listing directory: /\r\n");
218     ff_result = f_opendir(&dir, "/");
219     if (ff_result)
220     {
221         NRF_LOG_INFO("Directory listing failed!\r\n");
222         return;
223     }
224
225     do
226     {
227         ff_result = f_readdir(&dir, &fno);
228         if (ff_result != FR_OK)
229         {
230             NRF_LOG_INFO("Directory read failed.");
231             return;
232         }
233
234         if (fno.fname[0])
235         {
236             if (fno.fattrib & AM_DIR)
237             {
238                 NRF_LOG_RAW_INFO("   <DIR>   %s\r\n",(uint32_t)fno
                    .fname);
239             }
240             else
241             {
242                 NRF_LOG_RAW_INFO("%9lu  %s\r\n", fno.fsize, (
                    uint32_t)fno.fname);
243             }
244         }
245     }
246     while (fno.fname[0]);
247     NRF_LOG_RAW_INFO("\r\n");
248
249     NRF_LOG_INFO("Writing to file %c%c%c%c...\r\n",sd_file_name
        [0],sd_file_name[1],sd_file_name[2],sd_file_name[3]);
250     ff_result = f_open(&file, sd_file_name, FA_READ | FA_WRITE |
```

```
        FA_OPEN_APPEND);
    if (ff_result != FR_OK)
    {
        NRF_LOG_INFO("Unable to open or create file: %c%c%c%c.\r\n
            ",sd_file_name[0],sd_file_name[1],sd_file_name[2],
            sd_file_name[3]);
        return;
    }

    ff_result = f_write(&file, sd_buffer, sd_buf_len, (UINT *) &
        bytes_written);
    if (ff_result != FR_OK)
    {
        NRF_LOG_INFO("Write failed\r\n.");
    }
    else
    {
        NRF_LOG_INFO("%d bytes written.\r\n", bytes_written);
                                nrf_drv_wdt_channel_feed(
                                    m_channel_id);
    }

    (void) f_close(&file);
    return;
}



/**@brief Function to start scanning.
 */
static void scan_start(void)
{
    ret_code_t ret;

    ret = sd_ble_gap_scan_start(&m_scan_params);
    APP_ERROR_CHECK(ret);
}
//----------------------------------------
/** @brief Timeout handler for the repeated timer */
static void aq_handler(void * p_context)
{
        tick++;
        if(tick==TIME_FACTOR){
                time_s++;
                tick=0;
        }
        if(time_s==60){
                time_s=0;
                time_m++;
        }
```

```
297         if(time_m==60){
298                 time_m=0;
299                 time_h++;
300                 sd_central_able=true;
301         }
302         if(time_h==24){
303                 time_h=0;
304         }
305 }
306 //-----------------------------------
307 /** @brief Create timers */
308 static void create_timers()
309 {
310     uint32_t err_code;
311 
312     // Create timers
313 
314     err_code = app_timer_create(&aq_timer_id,
315                                 APP_TIMER_MODE_REPEATED,
316                                 aq_handler);
317     APP_ERROR_CHECK(err_code);
318 
319                 err_code = app_timer_start(aq_timer_id,
                       APP_TIMER_TICKS(SAMPLE_PERIOD,
                       APP_TIMER_PRESCALER), NULL);
320     APP_ERROR_CHECK(err_code);
321 
322         /*
323                 err_code = app_timer_create(&adv_init_timer_id,
324                                 APP_TIMER_MODE_SINGLE_SHOT,
325                                 adv_init_handler);
326     APP_ERROR_CHECK(err_code);*/
327 
328 }
329 
330 /**@brief Function for handling the Application's BLE Stack events
       .
331  *
332  * @param[in] p_ble_evt  Bluetooth stack event.
333  */
334 static void on_ble_evt(ble_evt_t * p_ble_evt)
335 {
336     const ble_gap_evt_t * p_gap_evt = &p_ble_evt->evt.gap_evt;
337 
338     switch (p_ble_evt->header.evt_id)
339     {
340     case BLE_GAP_EVT_ADV_REPORT:
341         {
342         const ble_gap_evt_adv_report_t * p_adv_report = &p_gap_evt
              ->params.adv_report;
343                                 const uint8_t * data= p_adv_report
```

```
                                        ->data;
344
345                                     if(data[5]==(uint8_t)(
                                           APP_COMPANY_IDENTIFIER | 0x000f
                                            ) && data[6]==(uint8_t)(
                                           APP_COMPANY_IDENTIFIER>>8))
346                                     {
347
348                                             valor_rssi = p_adv_report
                                                ->rssi;
349
350                                             if(valor_rssi>(-55)){
351                                                     aux=60;
352                                             }else{
353                                                     aux=100;
354                                             }
355
356                                             if(data_validation[data
                                                [7]]!=data[8])
357                                             {
358                                                     animal_id=data[7];
359
360                                                     data_validation[
                                                        data[7]]=data
                                                        [8];
361
362                                                     temperatura=((
                                                        float)data
                                                        [9]-50)/2;
363
364                                                     tensao=(data
                                                        [10]*3.6)/255;
365
366                                                     can_get_weight=
                                                        true;
367                                             }
368                                     }
369                             }
370                     break;
371         default:
372             break;
373         }
374 }
375
376
377 /**@brief Function for dispatching a BLE stack event to all
       modules with a BLE stack event handler.
378  *
379  * @details This function is called from the scheduler in the main
         loop after a BLE stack event has
380  *          been received.
```

```
381  *
382  * @param[in] p_ble_evt  Bluetooth stack event.
383  */
384 static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
385 {
386     on_ble_evt(p_ble_evt);
387 }
388 
389 
390 /**@brief Function for initializing the BLE stack.
391  *
392  * @details Initializes the SoftDevice and the BLE event interrupt
        .
393  */
394 static void ble_stack_init(void)
395 {
396     uint32_t err_code;
397 
398     nrf_clock_lf_cfg_t clock_lf_cfg = NRF_CLOCK_LFCLKSRC;
399 
400     // Initialize the SoftDevice handler module.
401     SOFTDEVICE_HANDLER_INIT(&clock_lf_cfg, NULL);
402 
403     ble_enable_params_t ble_enable_params;
404     err_code = softdevice_enable_get_default_config(
           CENTRAL_LINK_COUNT,PERIPHERAL_LINK_COUNT,&ble_enable_params
           );
405     APP_ERROR_CHECK(err_code);
406 
407     //Check the ram settings against the used number of links
408     CHECK_RAM_START_ADDR(CENTRAL_LINK_COUNT,PERIPHERAL_LINK_COUNT)
           ;
409 
410     // Enable BLE stack.
411     err_code = softdevice_enable(&ble_enable_params);
412     APP_ERROR_CHECK(err_code);
413 
414                 // Register with the SoftDevice handler module for
                        BLE events.
415     err_code = softdevice_ble_evt_handler_set(ble_evt_dispatch);
416     APP_ERROR_CHECK(err_code);
417 }
418 
419 
420 /**@brief  Function for initializing the UART module.
421  */
422 
423 /**@snippet [Handling the data received over UART] */
424 void uart_event_handle(app_uart_evt_t * p_event)
425 {
426 
```

```
427     static uint8_t data_array;
428         static int u=0;
429 
430     switch (p_event->evt_type)
431     {
432     case APP_UART_DATA_READY:
433                 if(can_get_weight){
434                         app_uart_get(&data_array);
435                         //app_uart_put(data_array);
436                         if((data_array!=0xFF)&&(u<(TCC_DATA_LEN)))
437                         {
438                                 switch (u){
439                                         case 0:
440                                                 peso=data_array
                                                    *100;
441                                                 break;
442                                         case 1:
443                                                 peso+=data_array
                                                    *10;
444                                                 break;
445                                         case 2:
446                                                 peso+=data_array
                                                    *10;
447                                         break;
448                                         case 3:
449                                                 peso+=data_array;
450                                                 break;
451                                         case 4:
452                                                 peso+=data_array
                                                    /10;
453                                                 break;
454                                         case 5:
455                                                 peso+=data_array
                                                    /100;
456                                                 break;
457                                         default:
458                                             break;
459                                         }
460                                         u++;
461                                 }else{
462                                         u=0;
463                                         sd_write_able=true;
464                                 }
465                 }
466         break;
467     case APP_UART_COMMUNICATION_ERROR:
468         APP_ERROR_HANDLER(p_event->data.error_communication;
469         break;
470 
471     case APP_UART_FIFO_ERROR:
472         APP_ERROR_HANDLER(p_event->data.error_code);
```

```
473             break;
474
475         default:
476             break;
477         }
478 }
479 /**@brief  Function for initializing the UART module.
480  */
481 /**@snippet [UART Initialization] */
482 static void uart_init(void)
483 {
484     uint32_t                     err_code;
485     const app_uart_comm_params_t comm_params =
486     {
487         TCC_RX_PIN,
488         TX_PIN_NUMBER,
489         RTS_PIN_NUMBER,
490         CTS_PIN_NUMBER,
491         APP_UART_FLOW_CONTROL_DISABLED,
492         false,
493         UART_BAUDRATE_BAUDRATE_Baud9600
494     };
495
496     APP_UART_FIFO_INIT( &comm_params,
497                         UART_RX_BUF_SIZE,
498                         UART_TX_BUF_SIZE,
499                         uart_event_handle,
500                         APP_IRQ_PRIORITY_LOWEST,
501                         err_code);
502     APP_ERROR_CHECK(err_code);
503 }
504 /**@snippet [UART Initialization] */
505
506 /**@brief Function for doing power management.
507  */
508 static void power_manage(void)
509 {
510         #if (__FPU_USED == 1)
511                 __set_FPSCR(__get_FPSCR() & ~(0x0000009F));
512                 (void) __get_FPSCR();
513                 NVIC_ClearPendingIRQ(FPU_IRQn);
514         #endif
515
516     uint32_t err_code = sd_app_evt_wait();
517     APP_ERROR_CHECK(err_code);
518 }
519
520
521 /**
522  * @brief Function for application main entry.
523  */
```

```
524 int main(void)
525 {
526     uint32_t err_code;
527     // Initialize.
528     err_code = NRF_LOG_INIT(NULL);
529     APP_ERROR_CHECK(err_code);
530 
531                 APP_TIMER_INIT(APP_TIMER_PRESCALER,
                        APP_TIMER_OP_QUEUE_SIZE, false);
532 
533     ble_stack_init();
534 
535                 //Enable DCDC regulator
536                 err_code = sd_power_dcdc_mode_set(
                        NRF_POWER_DCDC_ENABLE);
537                 APP_ERROR_CHECK(err_code);
538 
539     //Start uart
540                 uart_init();
541 
542                 wdt_config();
543 
544                 // Create timer for temperature aquisition
545                 create_timers();
546 
547                 //Start scanning
548                 scan_start();
549 
550                 // Start execution.
551                 NRF_LOG_INFO("Central HEDRO\r\n");
552                 //NRF_LOG_FLUSH();
553 
554 
555                 app_uart_put('m');
556 
557     // Enter main loop.
558     for (;; )
559     {
560         if (NRF_LOG_PROCESS() == false)
561         {
562             power_manage();
563         }
564                 if(sd_central_able)
565                 {
566                         sd_central_able=false;
567                         memset(sd_buffer,0,sizeof(sd_buffer));
568                         sd_buf_len=sprintf(sd_buffer,"\r\nTROCA DE
                                HORA,%d",time_h);
569                         write_data_sd();
570                 }
571                 if(sd_write_able)
```

```
572                     {
573                             sd_write_able=false;
574                             sd_buf_len=sprintf(sd_buffer,"\r\n%d;%.2f
                                   ;%.1f;%.2f",animal_id,peso,temperatura,
                                   tensao);
575                             write_data_sd();
576                     }
577                     if(sd_fail){
578                             NVIC_SystemReset();
579                     }
580
581                     NRF_LOG_FLUSH();
582         }
583 }
584
585
586 /**
587  * @}
588  */
```

Referências

[1] Erich Gamma. *Padrões de Projetos: Soluções Reutilizáveis*. Bookman editora, 2009.

[2] Alexsander Lopes de Oliveira. Web service para geração de vrs no contexto de posicionamento gps. 2010.

[3] Rodrigo da Costa Gomes e Gelson Luiz Dias Feijó e Lucimara Chiari. Evolução e Qualidade da Pecuária Brasileira. Campo Grande: EMBRAPA. Technical report, EMBRAPA, 03 2017.

[4] Scot Consultoria. Problemas no metabolismo de bovinos confinados, 2006. Disponível em `https://www.scotconsultoria.com.br/noticias/artigos/21233/problemas-no-metabolismo-de-bovinos-confinados.htm` Acessado em: 18 Abril 2018.

[5] Darlene Santiago. Pecuária de precisão: nova plataforma revoluciona a pesagem do gado., 2017. Disponível em `http://sfagro.uol.com.br/pecuaria-de-precisao-nova-plataforma-revoluciona-pesagem-gado/` Acessado em: 18 Abril 2018.

[6] Clauber Eduardo Marchezan Scherer e Alexandre Alves Porsse. Eficiência produtiva regional da agricultura brasileira: uma análise de fronteira estocástica. 2017.

[7] Marina Salles. Balança de pesagem promete melhorar gestão do gado., 2017. Disponível em `http://www.portaldbo.com.br/Revista-DBO/Noticias/Balanca-de-passagem-promete-melhorar-gestao-do-gado/20131/` Acessado em: 18 Abril 2018.

[8] Fabiano Alvim Barbosa e outros autores. Cenários para a pecuária de corte amazônica, 2015. Disponível em `http://csr.ufmg.br/pecuaria/pdf/contexto.pdf` Acessado em: 18 Abril 2018.

[9] BOSCH. `https://pecuaria.bosch.com.br`, 2018. Acessado em: 18 Abril 2018.

[10] DeLaval. http://www.delaval.com.br/-/Produtos--Solucoes/Gerenciamento-de-Rebanho/Solucoes/Sistema-de-Pesagem-Automatica-AWS100/, 2018. Acessado em: 18 Abril 2018.

[11] Toledo do Brasil. https://www.toledobrasil.com.br/balanca/balancas-para-pesar-animais/mgr4000, 2018. Acessado em: 18 Abril 2018.

[12] Júlio César Brandini. Doenças em bovinos confinados, 1996. Disponível em http://ainfo.cnptia.embrapa.br/digital/bitstream/item/135848/1/DOC-65.pdf Acessado em: 18 Abril 2018.

[13] Everton Zaccaria Nadalin. Determinação da força peso, a partir dos impactos de pisadas, utilizando um sensor piezoelétrico., 2017. Disponível em http://www.repositorio.unicamp.br/bitstream/REPOSIP/262031/1/Nadalin_EvertonZaccaria_M.pdf Acessado em: 18 Abril 2018.

[14] PF Arthur, G Renand, and D Krauss. Genetic and phenotypic relationships among different measures of growth and feed efficiency in young charolais bulls. *Livestock Production Science*, 68(2):131–139, 2001.

[15] Maurício Manduca Ferreira, Antonio Carlos Manduca Ferreira, and Jane Maria Bertocco Ezequiel. Avaliação econômica da produção de bovinos confinados: estudo de caso. *Informações Econômicas, São Paulo*, 34(7):8–20, 2004.

[16] Cauê Freire de Abreu. Confinamento de bovinos e suas técnicas para viabilidade da atividade pecuária. 2014.

[17] Rogério T de Faria, Fernando de SM Campeche, and Eduardo Y Chibana. Construção e calibração de lisímetros de alta precisão. *R. Bras. Eng. Agríc. Ambiental*, 10(1):237–242, 2006.

[18] Célula de carga. https://www.toledobrasil.com.br/blog/artigos/detalhe/voce-sabe-como-funciona-uma-celula-de-carga. acessado em 02/06/2018.

[19] D. I BARNA, A.; PORAT. Integrated circuits in digital electronics. 1973.

[20] Vega LLapapasca, Roberto Jossimar, et al. Análise e caracterização de desempenho de conversores analógico-digitais. 2013.

[21] Thiago Brito Bezerra et al. Desenvolvimento de um conversor a/d integrador com faixa de entrada e resolução programável a capacitor chaveado. 2012.

[22] Rodrigo Durães de Vasconcellos. Projeto de um conversor analógico/digital por aproximações sucessivas de 12 bits. 2011.

[23] Thales Exenberger Becker. Estudo dos efeitos de single event transients em conversor ad sar do tipo redistribuição de carga. 2015.

[24] Paulo Henrique Marchetti and Marcos Duarte. Instrumentação em eletromiografia. *Laboratório de Biofísica, Escola de Educação Física e Esporte. São Paulo: Universidade de São Paulo*, 2006.

[25] André Luiz Lins Miranda. *Projetos de filtros digitais para análise de sinais do sistema elétrico*. PhD thesis, UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 2005.

[26] Dirk Lindeke et al. Projeto de um filtro ativo paralelo de 1kva usando técnicas de controle analógico e digital. 2003.

[27] Ernande Eugenio Campelo Morais. Estudo e projeto de filtros passivos para atenuação de harmônicos em instalações elétricas industriais. 2011.

[28] Enio Roberto Ribeiro et al. Filtros ativos série para a compensação de harmônicas de tensão. 2003.

[29] M Oliveira Junior and R de O DUARTE. Apostila sobre introdução ao projeto com microcontroladores e programação de periféricos. *Departamento de Engenharia Eletrônica–Escola de Engenharia–Universidade Federal de Minas Gerais*, 2011.

[30] Ivan Luiz Marques Ricarte. Programação orientada a objetos: uma abordagem com java. *Universidade Estadual de Campinas, Campinas, SP, Brasil*, 2001.

[31] Elisabeth Andrade. Programação estruturada. 2018.

[32] Bianca Deordica and Marian Alexandru. Advertisement using bluetooth low energy. *Review of the Air Force Academy*, (2):65, 2014.

[33] Robin Heydon. *Bluetooth low energy: the developer's handbook*, volume 1. Prentice Hall Upper Saddle River, 2013.

[34] Bin Yu, Lisheng Xu, and Yongxu Li. Bluetooth low energy (ble) based mobile electrocardiogram monitoring system. In *Information and Automation (ICIA), 2012 International Conference on*, pages 763–767. IEEE, 2012.

[35] Rodrigo Franco Gonçalves, Vagner Luiz Gava, MARCELO SCHNECK DE PAULA PESSÔA, and MAURO DE MESQUITA SPINOLA. Uma proposta de processo de produção de aplicações web. *Production*, 15(3):376–389, 2005.

[36] Lucinéia Souza Maia. *Um processo para o desenvolvimento de aplicações Web acessíveis*. PhD thesis, Dissertação–Universidade Federal do Mato Grosso do Sul, 2010.

[37] Marco Winckler and Marcelo Soares Pimenta. Avaliação de usabilidade de sites web. *Escola de Informática da SBC SUL (ERI 2002) ed. Porto Alegre: Sociedade Brasileira de Computação (SBC)*, 1:85–137, 2002.

[38] Adolfo Henrique Schneider. Desenvolvimento web com client side rendering: combinando single page application e serviços de backend. 2016.

[39] Daniel Welfer et al. Padrões de projeto no desenvolvimento de sistemas de processamento de imagens. *Universidade Federal de Santa Maria*, 2005.

[40] AR Dantas, GO Veronese, AL Correa, José Ricardo Xavier, and CML Werner. Suporte a padrões no projeto de software. *Caderno de Ferramentas do XVI Simpósio Brasileiro de Engenharia de Software, Gramado, Rio Grande do Sul, Brasil, Outubro*, 2002.

[41] Tiago Barros, Mauro Silva, and E Espínola. State mvc: Estendendo o padrão mvc para uso no desenvolvimento de aplicações para dispositivos móveis. In *Sexta Conferência Latino-Americana em Linguagens de Padrões para Programação*, 2007.

[42] Abraham Silberschatz, Henry Korth, and S Sundarshan. *Sistema de banco de dados*. Elsevier Brasil, 2016.

[43] Christopher J Date. *Introdução a sistemas de bancos de dados*. Elsevier Brasil, 2004.

[44] Márcio Bueno. Programação orientada a objeto.

[45] Flávia Maria Santoro, Marcos R da S Borges, and Neide Santos. Um framework para estudo de ambientes de suporte à aprendizagem cooperativa. *Revista Brasileira de Informática na Educação*, 4(1):51–68, 1999.

[46] Vitor Fernando Pamplona. Web services: construindo, disponibilizando e acessando web services via j2se e j2me. *Javafree. org*, 2004.

[47] Sérgio Nunes and Gabriel David. Uma arquitectura web para serviços web. *XATA-2005*, 2005.

[48] Espressif Systems. Esp-wroom-32 datasheet, Junho 2016. Disponível em: `https://br.mouser.com/`Acessado em 20 Maio 2018.

[49] Atmel. Atmega328p datasheet, Junho 2012. Disponível em: `https:/www.alldatasheet.com/`Acessado em 01 Junho 2018.

[50] Nordic Semiconductor. nrf52 datasheet, Agosto 2017. Disponível em: `https://www.nordicsemi.com/`Acessado em 01 Junho 2018.

[51] J. V. L. Hoeflich e outros autores. Ensaios de resistência do pvc, 2016. Disponível em `http://www.damec.ct.utfpr.edu.br/assistiva/downloadsAssitiva/resistPVC20.doc` Acessado em: 07 Novembro 2018.

[52] Almsaeed Studio. Adminlte, 2016. Disponível em `https://adminlte.io` Acessado em: 07 Novembro 2018.