Universidade Federal de Santa Catarina
Centro de Blumenau
Departamento de Engenharia de
Controle, Automação e Computação

Gabriel Tiskoski Serratine

Sistema de Contagem Automática e Compartilhamento de Dados Via Rede de Microcontroladores em uma Linha de Produção de Injeção de PVC

Blumenau
2021

Gabriel Tiskoski Serratine

Sistema de Contagem Automática e Compartilhamento de Dados Via Rede de Microcontroladores em uma Linha de Produção de Injeção de PVC

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos necessários para a obtenção do Título de Engenheiro de Controle e Automação.
Orientador: Prof. Dr. Ciro André Pitz

Universidade Federal de Santa Catarina
Centro de Blumenau
Departamento de Engenharia de
Controle e Automação e Computação

Blumenau
2021

Gabriel Tiskoski Serratine

Sistema de Contagem Automática e Compartilhamento de Dados Via Rede de Microcontroladores em uma Linha de Produção de Injeção de PVC

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Engenheiro de Controle e Automação.

Comissão Examinadora

Documento assinado digitalmente
Ciro Andre Pitz
Data: 20/05/2021 17:11:25-0300
CPF: 055.728.089-38
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Ciro André Pitz
Universidade Federal de Santa Catarina
Orientador

Documento assinado digitalmente
Adao Boava
Data: 20/05/2021 18:15:03-0300
CPF: 645.640.969-15
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Adão Boava
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Marcos Vinicius Matsuo
Data: 20/05/2021 19:19:36-0300
CPF: 052.580.029-85
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Marcos Vinicius Matsuo
Universidade Federal de Santa Catarina

Blumenau, 20 de maio de 2021

Dedico esse trabalho a todos que algum dia duvidaram de seu próprio potencial.

Agradecimentos

Agradeço a todos que de qualquer maneira me apoiaram.

Agradeço primeiramente minha família, que nunca mediu esforços para me apoiar, não importa quão difícil eu tornasse essa tarefa. Especialmente meu pai, minha mãe, meu irmão. Vocês são as pessoas mais especiais da minha vida.

Agradeço meu orientador, Professor Dr. Ciro André Pitz, que não poupou esforços para me auxiliar e responder minhas dúvidas, por mais perdido que eu estivesse.

Agradeço a meus amigos de faculdade, que com certeza levarei para toda essa vida, por sempre alegrarem meus dias e compartilharem dessa época tão especial comigo.

Agradeço também meus companheiros de estágio na Howe, por todos os conselhos e auxílios durante o desenvolvimento desse trabalho.

A todos, meu muito obrigado.

"Life is not the amount of breaths you take,
it's the moments that take your breath away"
(Hitch)

Resumo

É proposto neste trabalho um sistema de contagem automática para ambientes industriais com o intuito de facilitar a produção de grandes lotes de produtos. O sistema proposto é baseado em uma célula de carga associada à um microcontrolador. O novo sistema conta com uma interface intuitiva para o usuário, sendo que os operários podem selecionar o número de produtos e realizar apontamentos. Os dados obtidos com o sistema são enviados através de uma rede sem fio (mantida em funcionamento pelos próprios contadores) até um *hub* central que, por sua vez, os encaminhará para o servidor via rede Wi-Fi.

Palavras-Chave: 1. Automação. 2. Células de carga. 3. Contagem Automática. 4. Microcontroladores. 5. Wi-Fi.

Abstract

In this work, an automatic counting system for industrial environments is proposed in order to facilitate the production of large batches of products. The proposed system is based on a load cell associated with a microcontroller. The new system has an intuitive user interface, and operators can select the number of products and make notes. The data obtained with the system are sent over a wireless network (which is maintained online by the counters themselves) to a central hub that in turn forward the data to the server via a Wi-Fi network.

Keywords: 1. Automation. 2. Load Cells. 3. Automatic Counting. 4. Microcontrollers. 5. Wi-Fi.

Lista de figuras

Lista de tabelas

Lista de Siglas e Abreviaturas

ACK	*Acknowledgement*
AD	*Analógico-Digital*
ADC	*Analog-Digital Converter*
AP	*Access Point*
CI	*Circuito Integrado*
DA	*Digital-Analógico*
DHCP	*Dynamic Host Configuration Protocol*
EEPROM	*Electrically Erasable Programmable Read-Only Memory*
IoT	*Internet of Things*
I/O	*Input/Output*
IP	*Internet Protocol*
ISR	*Interrupt Service Routine*
LAN	*Local Area Network*
LCD	*Liquid Crystal Display*
LDR	*Light-Dependant Resistor*
LED	*Light-Emiting Diode*
MAN	*Metropolitan Area Network*
P2P	*Peer-to-Peer*
PAN	*Personal Area Network*
PCB	*Printed Circuit Board*
PP	*Polipropileno*
PVC	*Policloreto de Vinila*
RF	*Rádio-Frequência*
ROS	*Robot Operating System*
SPI	*Serial Peripheral Interface*
SPS	*Samples Per Second*
SSID	*Service Set Identifier*
UDP	*User Datagram Protocol*
UFSC	*Universidade Federal de Santa Catarina*
WAN	*Wide Area Network*
dBm	*Decibel miliwatt*
T_s	*Tempo de assentamento do conversor AD*

Sumário

1 Introdução

Muito antes dos termos automação e otimização existirem, já buscava-se sempre realizar tarefas com o mínimo de esforço e o máximo de recompensa. Pequenas otimizações de ações cotidianas, como pegar o caminho mais rápido quando se deseja ir à algum lugar, são fáceis de serem implementadas quando comandadas por seres humanos. Afinal, seres humanos são inteligentes e sabem interpretar comandos abstratos ou ambíguos. Porém, algumas definições de automação retiram humanos da equação. Segundo Mikell Groover, automação pode ser definida como "a tecnologia pela qual um processo ou procedimento é realizado sem assistência humana. Humanos podem estar presentes como observadores ou até participantes, mas o processo em si opera ante sua própria direção." [1].

Automação não é somente algo realizado em altíssimo nível somente para e por indivíduos com extremo conhecimento técnico sobre esse tópico. Automação tem sua raiz em tecnologias simples, utilizadas no cotidiano para tornar mais fácil a vida de todos. Desde mecanismos que utilizavam o nível de água para manutenção de relógios [2], até autômatos cuja função era simplesmente servir chá [3], tinha-se em mente que as pessoas poderiam usufruir dos benefícios trazidos pela automação mesmo que não possuíssem o conhecimento necessário para compreender totalmente seu funcionamento. A partir desse conhecimento foi possível evoluir-se a tecnologia, a ponto de *experts* dizerem que "Crianças que hoje estão no ensino fundamental irão, quando adultos, exercer profissões que sequer existem ainda" [4].

Ainda segundo Groover [1], um processo de automação necessita de 3 elementos essenciais: energia, um conjunto de instruções, e um sistema de controle que executará tais instruções. Apesar dos requisitos gerais serem simples, ainda há uma grande parte do setor industrial brasileiro que é dependente de processos manuais. Afinal, apesar de o Brasil contar com mais de 15 mil robôs industriais [5], liderando a América do Sul, esse número representa somente 1 robô a cada 815 trabalhadores no setor industrial. Mas é claro que não só de robôs é feita a automação industrial, e apesar da visão de tirar o fôlego que são plantas industriais totalmente automatizadas, alguns autores defendem que automação pode ser algo simples e integrado juntamente ao operador humano, de maneira que "com sistemas automatizados, o operador pode alocar recursos, como atenção, para outras tarefas que executem concorrentemente" [6].

É nesse contexto que o presente trabalho é desenvolvido, trazendo projetos de automação no ambiente industrial sem o intuito de remover ou substituir o operador humano, mas sim para auxiliá-lo à focar somente nas tarefas que nenhum robô pode realizar por ele. Há diversas maneiras de aumentar ganhos ou diminuir perdas em processos manuais, e sempre haverá um compromisso de tempo em relação à qualidade do produto entregue

quando operários humanos estão envolvidos. Permitir que operários tenham mais tempo para realização de suas atividades é a meta do sistema proposto.

1.1 Objetivos

1.1.1 Objetivo geral

O objetivo geral deste trabalho é desenvolver um sistema automatizado que auxilie operadores em uma linha de produção predominantemente manual. O novo sistema deve ser capaz de realizar a contagem automática de produtos, bem como possibilitar que operadores realizem o apontamento de suas atividades diretamente de suas estações produtivas. Também é objetivo deste trabalho desenvolver uma rede sem fio para compartilhamento de dados entre os dispositivos desenvolvidos, com o intuito de reduzir o tempo gasto e auxiliar operadores a permanecerem concentrados em outras tarefas importantes. Dados enviados pela rede sem fio irão ter como destino um *hub* central, que os enviará ao servidor da empresa via Wi-Fi.

1.1.2 Objetivos específicos

Os objetivos específicos dos projetos presentes neste trabalho são listados a seguir:

- Realizar a contagem automática e com precisão da produção de novas peças no setor de injeção de PVC para plugues de tomada.
- Reduzir perdas por superprodução de produtos que ficariam em estoque desnecessariamente, bem como evitar a subprodução a fim de garantir que ordens de produção sejam cumpridas de maneira correta.
- Implementar um sistema para coleta e compartilhamento de dados. Esse sistema deverá integrar diversos microcontroladores comunicando-se com um *hub* central que, então, enviará os dados ao banco de dados.
- Realizar a integração do sistema proposto a um sistema de banco de dados já existente, de maneira que não sejam necessárias modificações severas na arquitetura pré-existente.
- Abstrair a complexidade do sistema e, assim, eliminar a necessidade de conhecimento técnico na área de automação para utilizá-lo.

1.2 Organização

Este trabalho pretende apresentar um conjunto de projetos de caráter simples porém que atuam em conjunto para formar um sistema com objetivo de melhorar a eficiência de uma célula produtiva. Assim, ao referir-se a "sistema", o autor refere-se ao sistema como um todo, e "projeto" refere-se ao projeto específico abordado no capítulo. Caso seja necessário referir-se a conceitos ou ações entre projetos então ambos são devidamente identificados de antemão. Além disso pode-se dividir o trabalho em quatro capítulos distintos, cada qual tem seu conteúdo listado à seguir.

- Capítulo 2: Descrição do processo produtivo. Nesse capítulo são explicados os processos produtivos de uma linha de injeção de PVC para plugues de tomada, os problemas encontrados pelos operadores ao longo da produção, possíveis fontes de perdas e pontos da produção que podem ser melhorados com o sistema proposto por este trabalho.

- Capítulo 3: Sistema de contagem automática. Nesse capítulo é apresentado o projeto de um contador automático de produtos, apresentação e discussão sobre todos os componentes que o integram, bem como sobre o código que estará em execução. Além disso, é apresentado o algoritmo de detecção de novos produtos criado para o sistema, bem como as interações possíveis entre operador e dispositivo.

- Capítulo 4: Sistema de comunicação sem fio. Nesse capítulo é descrita a rede sem fio criada para compartilhamento de dados entre cada módulo contador e um *hub* central. É abordada a topologia da rede e os protocolos de comunicação utilizados, o envio e tratamento de dados entre diferentes dispositivos, bem como o envio de diferentes tipos de dados, e como o *hub* fará o processamento de tais dados para enviá-los ao servidor da empresa. Como esse envio se dá por uma rede Wi-Fi, a mesma também é abordada, bem como o protocolo de comunicação utilizado. Nesse mesmo capítulo são tratadas possíveis melhorias para o futuro do sistema, como a implementação de novos componentes ou otimização das várias diferentes funções sendo executadas no seu código.

- Capítulo 5: Resultados e discussão. Nesse capítulo, são apresentados resultados da utilização do sistema em um ambiente industrial real, os ganhos proporcionados pelo mesmo e suas vantagens e desvantagens quando comparado ao processo produtivo atual.

- Capítulo 6: Conclusões. São apresentadas as conclusões finais sobre o sistema apresentado.

2 Descrição do processo produtivo

O processo de fabricação de diversos tipos de cabos elétricos ainda é um processo predominantemente manual. Apesar de haverem diversas máquinas que realizam etapas específicas do processo produtivo, a maior parte do tempo de produção está relacionada com atividades executadas por operadores humanos. Apesar de ser um produto simples, um cabo PP injetado passa por cinco etapas de fabricação diferente. Cabos do tipo PP são muito utilizados em aparelhos eletrodomésticos e na indústria automotiva, pois oferecem flexibilidade e alta resistência a abrasões externas, devido à seu encapamento de dupla camada.

O principal material do cabo PP, o polímero Polipropileno, é o segundo plástico *commodity* mais produzido no mundo [9]. O mercado global de polipropileno é esperado de atingir um total de 192 bilhões de dólares até 2023 [10]. O material é amplamente utilizado, por exemplo, em aplicações aonde necessitam-se dobradiças plásticas, como tampas de embalagens, devido à sua alta resistência mecânica. Além disso, o material possui a maior temperatura de fusão dentre todas as *commodities* plásticas e excelente resistência química e térmica [11].

Já o outro material utilizado na fabricação dos cabos PP, o PVC, ou Policloreto de Vinila, também se encontra no topo da lista de *commodities* plásticas mais produzidas. Com sua invenção datando desde o século XIX, o PVC é encontrado tanto em forma rígida quanto flexível, e possui aplicações que vão desde a indústria de canos e tubulações até a produção de cartões de crédito e aplicações na indústria têxtil. É um material com ótima resistência química e é bastante utilizado para isolamento de cabos elétricos, porém solta vapores quando queimado e, portanto, não é recomendado para utilização em ambientes com pouca ventilação.

No Brasil a norma NBR 14136 de 2012 [12] é responsável pela regulamentação de plugues de tomada para uso doméstico em corrente alternada. Essa norma trata de plugues com capacidade de até 20 ampères e 250 V AC.

2.1 O processo de injeção de PVC para plugues de tomada

Dentre os demais produtos produzidos na empresa, o setor de injeção de PVC foi o escolhido para os testes preliminares devido ao alto fluxo de produtos por dia. Há, em média, a produção de 7 lotes de cabos com plugues injetados por dia, podendo conter até mil unidades por lote.

2.1.1 Corte

Na primeira etapa do processo produtivo devem-se cortar os cabos de acordo com a extensão desejada. Cabos são produzidos em bobinas e entregues como um rolo contínuo de material, como demonstra a Figura 1. Essa etapa do processo produtivo encontra-se bastante automatizada na planta atual, contando com maquinário que mede e corta até 20 unidades de cabo por minuto. Apesar de não possuírem conectividade para realizar o processo de apontamento[1] automaticamente, as máquinas contam com *displays* que tanto exibem informações como permitem a configuração da operação da mesma, deixando a cargo dos operadores realizar apontamento antes e depois da produção. Um detalhe importante é que um mesmo comprimento de cabo PP pode ser utilizado para a produção de diversos tipos de plugues, assim normalmente são reunidas várias ordens de produção para uma só operação do maquinário. A estação de corte também é responsável pelos diversos outros tipos de cabo produzidos na empresa, não ficando restrita somente à cabos PP.

Figura 1 – Bobinas de cabos. Fonte: Elaborada pelo autor.

[1] O processo de apontamento serve para parear cada processo produtivo ao longo da cadeia produtiva de um lote de produtos aos operadores que os realizaram. É realizado pelo operador no início e final da execução de sua tarefa, e é utilizado internamente para medir a produtividade de operadores individuais e dos setores produtivos dentro da empresa.

2.1.2 Aplicação de pinos e separador

Após o corte, os cabos seguem para as estações de aplicações de pino e separador. Nessa etapa os fios ainda encontram-se soltos na extremidade do cabo, então é necessário o uso de um separador antes do processo de injeção, uma vez que a distância entre os pinos é fortemente regulamentada. O processo de aplicação dos pinos aos fios chama-se crimpagem. A crimpagem, além de um processo mecânico, é um método de conexão elétrica sem solda aonde a conexão entre os terminais é fixada de maneira mecânica através da deformação de um, ou mais, terminais. É um processo irreversível mas que, quando realizado corretamente, sela completamente o contato, de maneira similar à um processo de solda à frio.

No processo produtivo essa etapa ainda é predominantemente manual, apesar do processo de crimpagem em si ser realizada com o auxílio de maquinário especializado, como exemplo o maquinário apresentado na Figura 2. Há máquinas que realizam automaticamente o alinhamento de fios, encaixe de pinos (já com separador), e crimpagem dos terminais, porém ainda assim necessitam que operadores humanos realizem a alimentação de cabos para as mesmas. Tal maquinário pode possuir contadores internos, dependendo do modelo utilizado, mas similarmente às maquinas de corte, não possuem conectividade para realizar apontamentos.

Figura 2 – Exemplo de maquinário para o processo de aplicação de pino, já com separador, e crimpagem. Fonte: Elaborada pelo autor.

2.1.3 Injeção de PVC: luva

A injeção da luva é o processo mais demorado da produção do plugue. As luvas são peças injetadas na extremidade contrária aos pinos e servem para auxiliar na fixação do cabo em eletrodomésticos e afins. Como os painéis por onde o cabo passa para acessar os contatos internos do aparelho são metálicos, os mesmos podem eventualmente possuírem superfícies cortantes em contato com o cabo, levando à seu eventual desgaste e expondo a fiação interna resultando em curto-circuitos e choques elétricos. Assim, as luvas são utilizadas como intermediário entre os painéis metálicos e os cabos, possuindo uma maior resistência à choques e danos mecânicos de natureza cortante. Além disso as mesmas oferecem proteção térmica, e, segundo a norma NBR NM 60335 [13], são obrigatórias para aparelhos que apresentem um aumento de temperatura determinado durante sua operação contínua.

2.1.4 Injeção de PVC: plugue

Depois de terem terminado de esfriar após o processo de injeção da luva, os cabos são levados até a estação de injeção do plugue. Apesar de utilizarem o mesmo processo e mesmo material, essa etapa é consideravelmente mais rápida que a injeção da luva, por necessitar de menos mão de obra manual. A etapa de resfriamento é aonde a maior parte do tempo é gasto, pois, diferente do processo de injeção da luva, não pode-se utilizar água como agente de resfriamento. O uso de água para resfriamento de plugues recém injetados pode resultar em infiltrações, por vez devido à maleabilidade do PVC ainda quente ou pelos próprios pinos, que no processo de fabricação por estampa podem apresentar fissuras. Infiltrações em plugues podem resultar em fuga de corrente[2], uma das condições mais comuns para reprovação de plugues de tomada, devido ao alto risco apresentado na utilização de aparelhos conectados à cabos defeituosos. Normalmente cabos que apresentam essas condições tornam-se refugos. Assim, o processo de resfriamento de plugues de tomada recém-injetados é realizado por ventiladores industriais.

2.1.5 Inspeção e dobra

Cabos de energia são componentes normatizados, e como tal, devem ser submetidos à uma bateria de testes para comprovar conformidade com tais normas. Alguns dos testes são:

- Teste de flexão mecânica. A ligação entre cabo e plugue é chamada de pescoço, e normalmente possui uma estrutura injetada semelhante à uma mola. Essa estrutura

[2] O fenômenos da fuga de corrente se dá quando uma corrente elétrica flui por caminhos indesejados, normalmente por falta de isolamento. A ocorrência de correntes de fuga pode gerar desde um consumo maior de energia até choques em operadores.

confere maleabilidade para o encaixe, evitando o rompimento de cabos nesse ponto. O teste de flexão mecânica realiza torção nesse ponto enquanto mantém o cabo tensionado através do uso de um peso. Por norma cabos devem resistir dezenas de milhares de flexões, e caso reprovados no teste deve-se checar a estrutura do pescoço ou os componentes utilizados na injeção para tentar-se aumentar a maleabilidade do conjunto.

- Teste para vazamento de tensão. Cabos elétricos em conforme com a norma NBR 14136 [12] possuem três terminais distintos: duas fases e um terra. Assim como qualquer circuito elétrico, ligar dois destes terminais em curto-circuito representa um risco à segurança do utilizador desse produto.

Produtos que encontrem-se fora das especificações devem prosseguir para a etapa de retrabalho, aonde tentam-se consertar os problemas encontrados nos mesmos. Produtos que não estão aptos para o processo de retrabalho e não poderão mais ser úteis à produção são chamados de refugos. Refugos são uma das maiores fontes de perda em processos produtivos, pois perde-se não somente material como também tempo de mão de obra.

Uma vez constatado o funcionamento correto dos cabos os mesmos são direcionados à estação de dobra. Nessa última etapa os cabos são então dobrados, amarrados com um fitilho, contados, e embalados. Após essa estação os mesmos podem tanto encontrar-se finalizados quanto serem direcionados à outras etapas da produção, para processos extras como, por exemplo, a montagem de terminais às extremidades desencapadas após a luva. Essa é a etapa aonde mais ocorrem erros de contagem.

3 Sistema de contagem automática

Como explicado no Capítulo 1 e tendo em vista o processo produtivo descrito no Capítulo 2, neste capítulo é abordada a proposta e desenvolvimento de um contador automático de produtos com estação de apontamento, além da criação de uma rede sem fio para compartilhamento de dados entre diversos dispositivos e envio dos dados à um servidor remoto. Cada contador faz uso de uma plataforma de medição de peso, que conta com quatro células de carga em uma configuração de ponte de Wheatstone [15], para obtenção do número de produtos. As células conectam-se à um módulo HX711 [16], que serve como amplificador e conversor analógico-digital. O sinal digital de saída do amplificador é lido por um microcontrolador que determina automaticamente o número de produtos na pesagem. Esses dados são exibidos para o operador através de um monitor LCD e indicadores luminosos, e enviados automaticamente ao servidor da empresa via uma rede de comunicação sem fio formada entre os dispositivos e um *hub* central. Além disso o projeto também contempla o design e criação de uma placa de circuito impresso e modelagem e impressão de uma *case* para armazenamento dos componentes. Cada um dos componentes é abordado em detalhes na Seção 3.3.1. A criação da rede de comunicação sem fio é apresentada no Capítulo 4, enquanto a integração do *hub* à rede pré-existente para envio de dados ao servidor ficará à cargo da Seção 4.5.

3.1 Problemática

Em sistemas produtivos de grande volume e com processos predominantemente manuais há uma grande dispersão no tempo gasto em cada etapa do processo e na qualidade dos produtos finais. Além do mais também podem ocorrer problemas devido à erros humanos. Especialmente em setores com processos repetitivos e lotes na ordem de centenas de unidades, um dos erros mais comuns é a contagem errônea do número de produtos em um lote produtivo. Essa contagem pode causar repercussões ao longo da produção, levando diferentes setores a receberem quantidades faltantes ou exageradas de produtos, e podendo estender-se até a entrega de um lote defeituoso ao cliente. Assim, neste trabalho é proposto um contador automático, onde o usuário poderá inserir o número de produtos a serem contados e preocupar-se somente com a produção dos mesmos, minimizando contagens erradas e aumentando a qualidade do produto final. O contador proposto deve:

- Contar com precisão a adição ou retirada de produtos da plataforma de pesagem.
- Avisar para o usuário quando o número de produtos atingir o esperado, podendo também avisar caso haja excesso de produtos no lote.

- Realizar o processo de contagem com o mínimo de interação necessária por parte do operador.
- Calcular automaticamente a tara da plataforma de pesagem e o peso unitário do produto a ser contado, podendo atualizar esse peso ao longo do tempo para obter um valor de detecção mais preciso.
- Possuir capacidade para contar lotes na casa de centenas ou até milhares de produtos.
- Realizar o apontamento de uma ordem de produção diretamente na estação do operador.

Uma rede de comunicações sem fio permite que dados sejam enviados imediatamente até uma estação para serem processados. Apesar da comunicação sem fio não ser tão confiável ou veloz, tanto em velocidade de transmissão quanto em largura de banda, quanto soluções convencionais com fio, sua flexibilidade é seu maior diferencial. Poder distribuir Nós de uma rede livremente dentro do alcance da mesma permite que o sistema, em sua configuração atual, não esteja preso à um setor ou processo produtivo específico. Além disso, fatores como velocidade e confiabilidade não apresentam perdas reais ao sistema, uma vez que podem ser eliminados de diversas maneiras. O ganho proporcionado por altas velocidades de propagação de sinais elétricos (uma fração da velocidade da luz) e grandes larguras de banda (1Gb/s de velocidade máxima e 100MHz de largura de banda máxima para cabos Ethernet Cat 5e, os mais comuns atualmente [17]) oferecidas por redes de comunicação cabeadas são desprezíveis quando é realizado o envio de pequenos pacotes na rede, como é o caso para o sistema desenvolvido. A confiabilidade de redes sem fio é uma de suas maiores desvantagens, sendo suscetíveis à interferências que causam desde oscilações na transmissão até perda total de sinal. Uma maneira de contornar a perda de pacotes em uma rede sem fio é via software, onde são implementadas rotinas que realizam o *acknowledgement*, ou reconhecimento, de pacotes recebidos, bem como utilizar frequências com baixo tráfego de informações. Assim, a rede de comunicações sem fio deve atender os seguintes pré-requisitos:

- Enviar com baixa latência os dados necessários.
- Permitir a conexão de múltiplos dispositivos de maneira mais transparente possível para o usuário.
- Nós devem ser capazes de se conectar automaticamente à rede, o usuário não deve ser obrigado a configurar cada dispositivo individualmente.
- Nós devem ser capazes de encaminhar pacotes de/para Nós remotos, aumentando assim o alcance da rede.

- Cada Nó deve, periodicamente, verificar sua conexão com a rede. O operador deve ser informado em caso de perda de conexão, fator que pode prejudicar ou até eliminar a comunicação com Nós remotos.
- Realizar o *acknowledgement* de pacotes recebidos, minimizando problemas resultantes da perda de pacotes.
- A rede deve evitar ao máximo possível a faixa de frequência de 2.4 GHz reservada à rede Wi-Fi pré-existente na empresa.

Como detalhado na Seção 4.1, a rede criada para comunicação entre os dispositivos de campo não possui conexão com a internet e os dispositivos de campo não se encontram conectados à rede Wi-Fi disponível no ambiente industrial. É função do *hub* realizar a ponte entre comunicação RF e Wi-Fi. Os dados são enviados entre o *hub* e o servidor via pacotes UDP (do inglês *User Datagram Protocol*, o protocolo UDP é abordado teoricamente na Seção 4.5.1.4 e na prática na Seção 4.5.4), e devem ser formatados de uma maneira que o servidor possa entendê-los. Por sua vez, o *hub* deve:

- Atuar como Nó mestre na rede de comunicação com dispositivos de campo, deve designar identificadores e endereços automaticamente para novos dispositivos que se conectarem na rede.
- Realizar periodicamente verificações de que os dispositivos marcados como conectados realmente estão conectados na rede e disponíveis para comunicação. Caso contrário os dispositivos devem ter seus identificadores e endereços liberados para permitir que novas conexões tomem seus lugares.
- Receber pacotes com diferentes tipos de dados, identificados pelo tipo do *header* do pacote.
- Permitir configuração para rede Wi-Fi sem a necessidade de ID e senha serem *hard coded*.
- Formatar dados recebidos para envio ao servidor via UDP de uma maneira convencionalmente aceita, sem que haja a necessidade de realizar ajustes no próprio servidor somente para acomodação do *hub*.
- Realizar o *acknowledgement* dos dados enviados ao servidor para evitar duplicatas e perda de pacotes.

3.2 Soluções existentes no mercado

Há uma extensa seleção de medidores de peso e contadores de produtos disponíveis no mercado. Desde contadores que prometem precisão na casa de microgramas até contadores com a versatilidade de pesar dezenas de quilos. Porém, nenhum desses contadores abrange todos os tópicos desejados para o projeto, como descrito na Seção 3.1. Além disso, contadores extremamente precisos não tem capacidade de pesagem de muitos itens. De maneira similar, contadores que prometem medições de milhares de produtos muitas vezes não oferecem a precisão desejada. Também não foi encontrado no mercado algum contador que possuísse a conectividade exigida, onde lotes são pesados e, uma vez que atinjam o número desejado de peças, os dados são automaticamente enviados ao banco de dados da empresa. Por fim, grande parte das soluções prontas disponíveis apresenta um custo mais elevado do que o projeto, mesmo em vista da vasta gama de componentes utilizados pelo mesmo. Apesar da qualidade de construção do protótipo final não não ser equivalente a de produtos comerciais, reitera-se que o mesmo é meramente um protótipo para uso interno, e apesar de ter sido feito um design de *case* para o mesmo tal atitude foi realizada pensando-se em ganhos práticos e segurança dos componentes e operadores enquanto o protótipo encontra-se em uso.

3.3 Idealização do projeto

Desde o início de seu desenvolvimento, o projeto seguiu o princípio de abordar uma tarefa de cada vez. O primeiro passo para o desenvolvimento de um projeto que integre diversos componentes é que todos os componentes estejam funcionando adequadamente de maneira individual. Para isso, realizaram-se testes de hardware e software, pequisas em *datasheets*, fóruns, leitura de documentações, revisões teóricas, com o intuito de compreender ao máximo o funcionamento de cada componente e seu efeito no projeto como um todo. As tarefas individuais realizadas ao longo do desenvolvimento do sistema são apresentadas a seguir:

1. Realizar medidas precisas com as células de carga e módulo HX711.
2. Integrar indicadores luminosos e de número de produtos sobre a balança. Nessa etapa ainda utilizava-se um potenciômetro para que o usuário realizasse a seleção do número de produtos a serem pesados.
3. Comunicação direta e sem fio entre dois microcontroladores idênticos, ainda sem estabelecer uma rede para múltiplos dispositivos.
4. Envio de dados entre dois dispositivos conectados diretamente.
5. Inserção de dados via *keypad* com divisor de tensão.

6. Envio de dados entre diferentes microcontroladores.

7. Estabelecer uma rede de comunicação entre dispositivos e *hub*. Endereços dinâmicos são implementados pela biblioteca utilizada, identificadores dinâmicos devem ser implementados pelo projetista.

8. Enviar dados do *hub* até um servidor de testes.

9. Ciclo de utilização completo por parte do contador, *hub* conecta-se automaticamente na rede Wi-Fi, configura rede RF para recebimento dos dados dos contadores, e realiza envio dos dados ao servidor.

3.3.1 Componentes

O microcontrolador escolhido como central do projeto foi o controlador ATmega328P, encontrado nas placas Arduino UNO e Nano (ATMEGA328). Originalmente com a ideia de utilizá-lo sem a placa, eventualmente o autor optou por utilizar a plataforma Arduino pois a mesma já inclui, entre outras funcionalidades, controlador de tensão na placa, cristal oscilador com capacitores acoplados, e *bootloader* já carregado no microcontrolador. A plataforma Arduino UNO conta com 14 pinos de I/O, dos quais 6 provém sinais de saída PWM (modulação por largura de pulso, do inglês *Pulse-Width Modulation*), e dividem-se em pinos analógicos e digitais. Todos os pinos podem ser configurados como entrada ou saída, observando-se certas restrições, e pinos analógicos além de funcionalidade analógica podem atuar com o mesmo comportamento de pinos digitais. Além disso, a plataforma conta com pinos de alimentação de componentes em 5V e 3.3V sem a necessidade de implementação de um divisor de tensão para alimentação de componentes. A Tabela 1 demonstra o uso de cada pino no escopo do projeto, além da descrição de sua função principal e restrições do uso. Com breves descrições de seu funcionamento e *links* para seus *datasheets*, quando disponíveis, os componentes utilizados para o desenvolvimento do sistema, tanto contadores quando *hub* de comunicações, são listados a seguir:

- **Arduino UNO**
  - Microcontrolador dos contadores, controla o funcionamento do mesmo e gerencia comunicação e tratamento de dados com o resto dos componentes.
- **Célula de carga 50Kg**
  - Dispositivo transdutor, varia seu valor resistivo proporcionalmente à força aplicada sobre a célula permitindo fazer a leitura e calcular a intensidade da força [18].
- **Demux 74HC155**

  - Expande as saídas digitais do microcontrolador, permite fazer o acionamento de 3 LEDs utilizando somente duas portas digitais [19]. A tabela verdade do mesmo pode ser encontrada na Seção 3.3.2.5.

- ***Display* LCD 16x02**

  - Fornece ao usuário informações sobre o funcionamento do contador, instruções sobre os próximos passos da operação, bem como possíveis erros que tenham acontecido durante a operação [20].

- **ESP8266**

  - Microcontrolador do *hub*. Gerencia a distribuição de novos IDs e endereços automaticamente na rede RF, recebe e envia dados dos contadores, e estabelece comunicação com o servidor via Wi-Fi para envio de dados com o protocolo UDP [21].

- **Hex Inverter 74HC04**

  - Inverte os sinais de saída do demux, necessário conforme informado na Seção 3.3.2.5 [22].

- **HX711**

  - Faz a leitura do valor resistivo das células de carga, amplifica o sinal lido e o converte para um sinal digital utilizando um conversor AD (analógico-digital) com resolução de 24 bits [16].

- ***Keypad***

  - Dispositivo para inserção de dados do operador, necessários para realizar o apontamento de ordens de produção.

- **LEDs**

  - Sinalizadores luminosos para informar o operador do estado de operação do contador e presença ou ausência de conexão com a rede de contadores.

- **nRF24L01+**

  - Módulo responsável pela comunicação sem fio entre contadores e entre contador e *hub*. Comunica-se com os microcontroladores via protocolo SPI. É o único componente que opera com tensão de alimentação 3.3V [23].

- **Pushbutton**

– *Input* do operador, sinaliza que o mesmo deseja recalibrar a balança ou o *keypad* ao final de um ciclo de operação.

Tabela 1 – Pinos do Arduino UNO: funções e utilização no projeto

Pino	Utilização no projeto	Observações
D0	Nenhuma	a
D1	Nenhuma	a
D2	Pushbutton	b
D3	Saída de dados HX711	b,c
D4	Entrada de dados LCD	
D5	Entrada de dados LCD	c
D6	*Enable* LCD	c
D7	*Register Select* LCD	
D8	Entrada de dados LCD	
D9	CSN nRF24L01+	c
D10	CE nRF24L01+	c
D11	MOSI nRF24L01+	c,d
D12	MISO nRF24L01+	d
D13	SCK nRF24L01+	d
A0	Entrada *keypad*	e
A1	Demux B	e
A2	Demux A	e
A3	*Clock* HX711	e
A4	Led Wi-Fi	e,f
A5	Entrada de dados LCD	e,f

a. Pinos D0 e D1 são utilizados para comunicação serial com o microcontrolador. São instáveis durante o período de *upload* de novos binários para o mesmo e é aconselhado que não sejam utilizados, salvo quando deve-se estabelecer comunicação serial com outros microcontroladores ou dispositivos.
b. Pinos D2 e D3 capaz de receber interrupções externas. Interrupções param o funcionamento do programa imediatamente e realizam instruções curtas e simples antes de retomar a execução do ponto onde parou. Apesar de possuírem suporte para interrupções podem ser utilizados como pinos digitais comuns.
c. Pinos D3, D5, D6, D9, D10, D11 são capazes de gerar sinais analógicos utilizando o método PWM. As portas operam com frequência de 490Hz, salvo para os pinos D5 e D6 onde a frequência de operação é de 960Hz.
d. Pinos D11, D12, D13 são utilizados para comunicação com dispositivos via protocolo SPI, onde correspondem aos barramentos MOSI, MISO, e SCK, respectivamente. O protocolo SPI é explicado com mais detalhes na Seção 3.3.2.4.
e. Pinos com prefixo "A" são pinos que possuem capacidade de realizar leituras analógicas. O sinal lido é convertido para um sinal digital e a resolução do conversor AD no Arduino UNO é de 10 bits, podendo gerar um sinal digital com até 1024 (2^{10}) valores distintos. Esses pinos também podem ser utilizados como pinos digitais comuns, realizando operações de entrada de sinais digitais (*HIGH* e *LOW*) e saída digital.
f. Pinos A4 e A5 são utilizados para comunicação com dispositivos via protocolo I^2C. No presente projeto nenhum dispositivo utiliza este protocolo, porém há a possibilidade de utilizá-lo para comunicação com o *display* LCD através de um módulo conversor I^2C/Serial.

3.3.2 Esquema elétrico

O esquema elétrico do projeto pode ser divido com base em seus componentes, uma vez que todos se conectam ao microcontrolador. As conexões entre microcontrolador e componentes para o contador são exibidas na Figura 3, enquanto para o *hub* são exibidas na Figura 4.

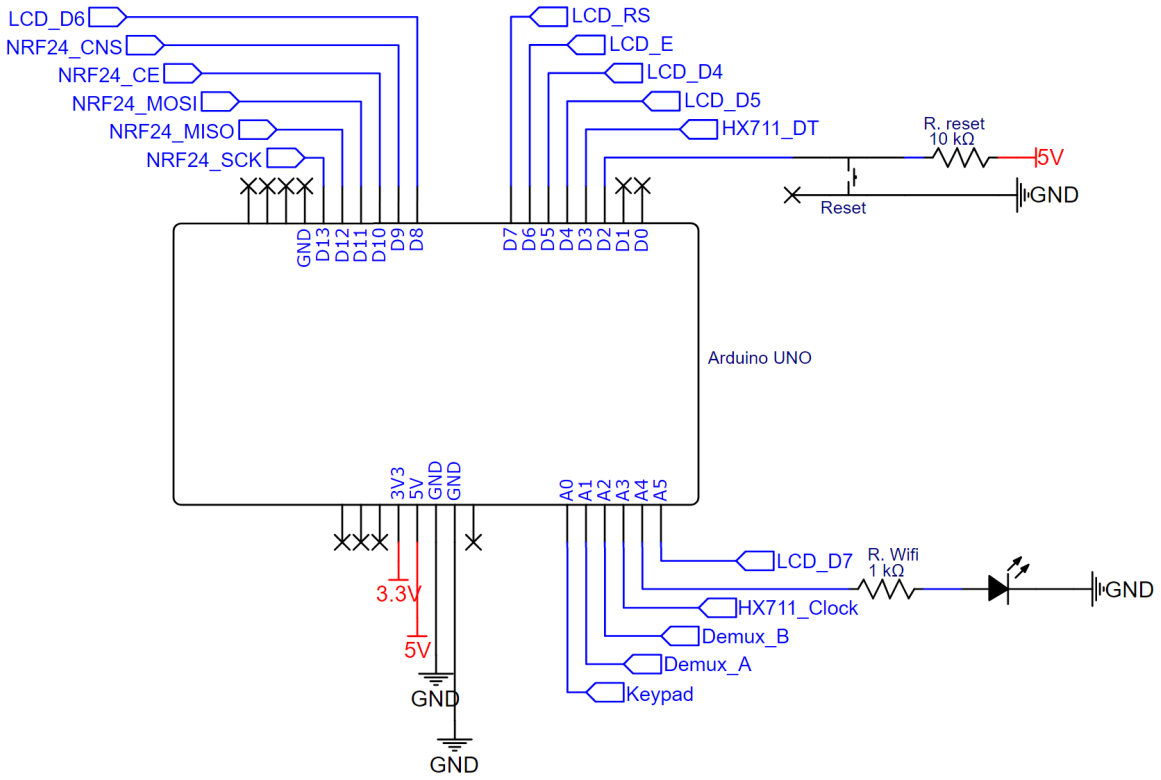

Figura 3 – Circuito do microcontrolador e componentes para o contador. Fonte: Elaborada pelo autor.

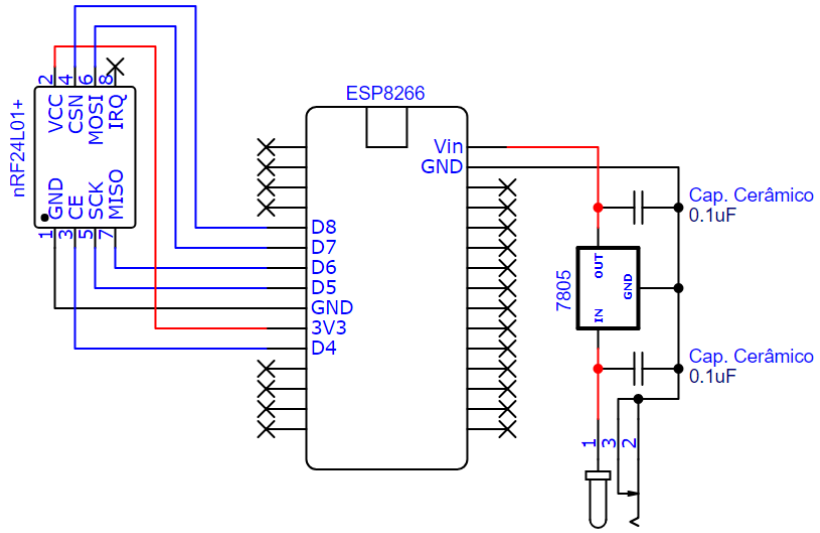

Figura 4 – Circuito do microcontrolador e componentes para o *hub*. Fonte: Elaborada pelo autor.

3.3.2.1 Circuito do *display* LCD

Conforme ilustrado na Figura 5, o monitor LCD utilizado possui 16 conexões com o microcontrolador. Destas, 6 podem ser ligadas diretamente à tensão de alimentação de 5V ou ao terra do microcontrolador, com duas sendo destinadas à alimentação do componente, duas dedicadas para alimentação dos LEDs que iluminam a tela, uma utilizada para regular o contraste de cada caractere, e uma utilizada para diferenciar entre leitura/escrita de dados do/no componente. As restantes 10 conexões são para dados e dividem-se em uma conexão *enable*, com função de habilitar o processamento dos dados enviados ao dispositivo, uma conexão *register select*, utilizada para diferenciar envio de instruções (*LOW*) ou dados (*HIGH*) ao dispositivo, e oito conexões de um barramento de dados de 8 bits em paralelo (nomeadas D0 à D7). O dispositivo também pode ser configurado via software para receber instruções de 4 bits, restringindo assim seu barramento de dados à utilização das entradas D4 à D7. Quanto à tela, o mesmo conta com 32 caracteres, distribuídos em duas linhas e 16 colunas. Cada caractere é composto por uma matriz de 5x8 *pixels* que pode, sem programação extra, exibir até 192 caracteres (entre eles números, letras maiúsculas e minúsculas, caracteres especiais, e o silabário *katakana* japonês), e o componente também aceita a definição, via software, de até oito caracteres customizados.

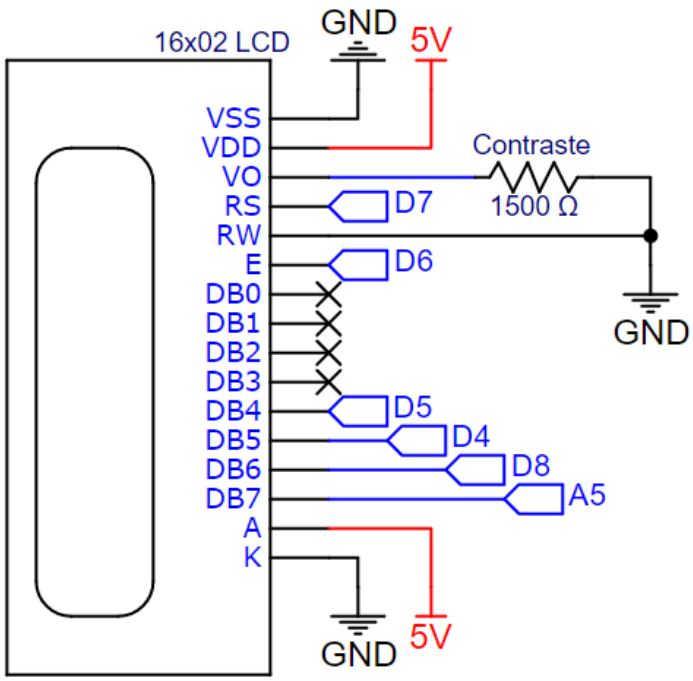

Figura 5 – Conexões do *display* LCD. Fonte: Elaborada pelo autor.

3.3.2.2 Circuito do amplificador e células de carga

Para o presente projeto, como demonstrado por Beckwith [24], a variação de resistência nas células de carga para todo o intervalo de operação (0-50Kg) é de $\pm 10\Omega$, correspondendo à uma variação de 1% em relação à sua resistência em repouso. Assumindo que a resistência varia linearmente em relação à força aplicada sobre a célula, é possível achar um valor que relaciona variação de resistência por grama: $0.2m\Omega/g$. Mesmo fazendo uso da configuração em ponte de Wheatstone [15], onde os *strain gauges* [24] são distribuídos alternadamente de forma a alternar tensão e compressão visando aumentar a sensibilidade da leitura, esse valor é muito pequeno para ser lido com precisão pelo conversor AD presente no microcontrolador. Com resolução de 10 bits, o conversor AD presente no microcontrolador lê 1024 valores, que distribuídos entre 0V e 5V resultam em uma precisão de aproximadamente $\pm 4,88mV$. Assim, usa-se o módulo HX711 que amplifica e converte o sinal lido com 24 bits de resolução. Sua conexão com as células de carga é demonstrada na Figura 6.

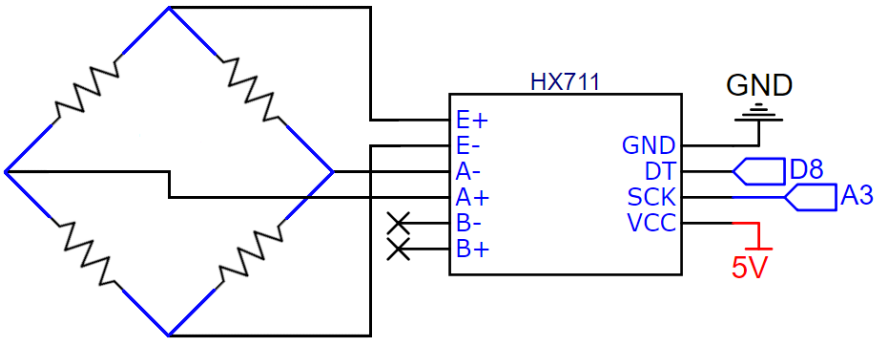

Figura 6 – Circuito do amplificador e células de carga em configuração de ponte de Wheatstone. Fonte: Elaborada pelo autor.

Apesar do módulo não fornecer 24 bits de conversão sem ruído (na prática notou-se que as leituras puras do sinal convertido oscilavam os últimos 6 bits, conforme ilustrado na Figura 7) a resolução de 18 bits restante mostrou-se satisfatória para o projeto por apresentar uma precisão de 0,763 gramas por nível lógico, conforme mostra a Equação (3.1). Nota-se na Figura que os valores apresentados possuem 32 bits, isso se deve simplesmente à utilização de uma variável de 32 bits para armazenamento dos dados, dessa forma os 8 bits mais significativos podem ser desprezados.

$$P = \frac{P_T}{Res} = \frac{200.000[g]}{2^{18} níveis} \approx 0,763 \frac{g}{nível} \tag{3.1}$$

Uma vez que o sinal tenha sido convertido para um sinal digital ele é então enviado ao microcontrolador na forma de um número do tipo `long`, um tipo de variável de 32 bits capaz de armazenar valores até $\pm 2^{31} - 1$ (2.147.483.647, o bit mais significativo denota se o número é positivo ou negativo). Esse número é a leitura pura do conversor e ainda não corresponde à medida de peso. Para que esse número seja apresentado ao operador de forma legível, primeiro devem-se obter duas outras medidas: o fator de tara (*tare factor*) e o fator de calibração, ou calibragem (*cal factor*). O fator de tara está relacionado ao peso medido "em vazio", ou seja, quando não há produtos sob a plataforma. Esse peso é proveniente não só da própria plataforma de pesagem como também de qualquer recipiente dentro do qual os produtos sejam pesados, como caixas ou demais tipos de embalagens. Já o fator de calibragem é relacionado ao ganho da conversão AD e relaciona a medida adimensional proveniente do módulo HX711 à um peso de referência conhecido. Enquanto o fator de tara é descontado do sinal convertido através de uma subtração, o fator de calibragem é uma constante pela qual o sinal convertido deve ser dividido. Ambos valores podem ser calibrados pelo operador durante a operação do contador.

O módulo HX711 possui algumas configurações para que seu funcionamento seja adequado à funcionalidade desejada. Tais configurações influenciam principalmente o ganho aplicado ao sinal analógico antes de sua conversão (padrão 128), o número de amostras

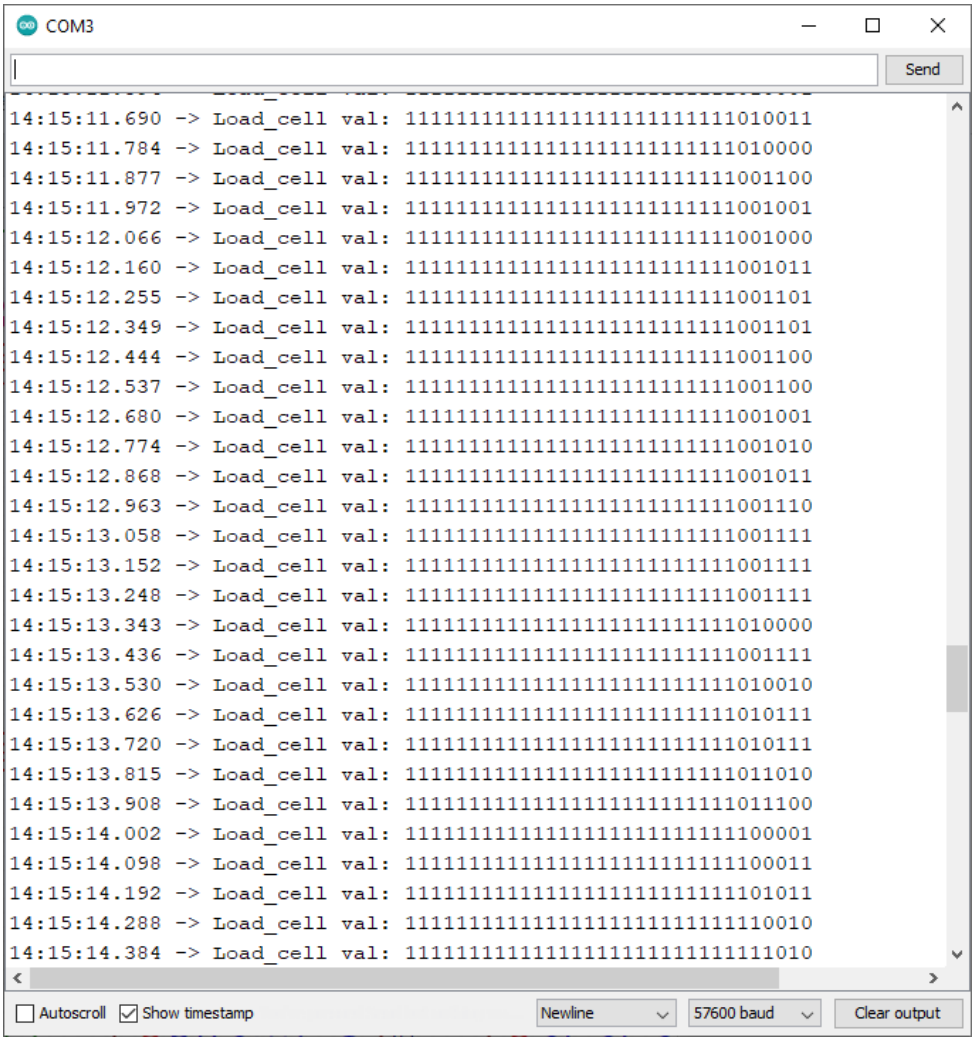

Figura 7 – Amostras obtidas do módulo HX711 em formato binário. Fonte: Elaborada pelo autor.

por segundo (padrão 10), o número de amostras no conjunto temporário (padrão 16), e o tempo de assentamento do sistema (padrão 1.8s) quando há uma mudança abrupta de peso sob as células. O número de amostras por segundo, ou SPS (do inglês *samples per second*), refere-se ao número de medidas de sinal digital convertido que são enviadas ao microcontrolador a cada segundo. Já o número de amostras no conjunto temporário refere-se ao número de medições digitais utilizadas para obter-se uma medição definitiva. Quando o conjunto temporário é preenchido, então faz-se a média de suas amostras rejeitando, ou não, a menor e maior amostra encontrada. Essa média é a amostra final enviada ao microcontrolador. Quanto maior o tamanho do conjunto temporário mais precisa será a amostra enviada ao microcontrolador, porém o tempo de assentamento também será mais longo. Para o presente projeto as configurações foram mantidas pois apresentam comportamento satisfatório. Os valores disponíveis são de 32, 64, e 128 para o ganho, 2^n para $n \leq 7$ para o número de amostras no conjunto temporário, e 10 ou 80 para o número de amostras enviado ao microcontrolador a cada segundo, enquanto o valor do tempo de

assentamento pode ser calculado através da seguinte expressão:

$$T_s = (N_S + IGN_{HS} + IGN_{LS})/SPS \tag{3.2}$$

onde T_s é o tempo de assentamento em segundos, N_S é o número de amostras do conjunto temporário, IGN_{HS} e IGN_{LS} são *flags* para que o maior e menor valor do conjunto temporário sejam descartados (por padrão ambas são iguais a 1, indicando que há o descarte dessas amostras), e SPS é o número de amostras enviadas ao microcontrolador a cada segundo. Como foram mantidas as configurações padrões, pode-se concluir que o contador desenvolvido para o projeto apresenta um tempo de assentamento de 1.8 segundos, suficiente pois, como explicado na Seção 3.3.3.4, o algoritmo de detecção utiliza a rápida variação de peso resultante da adição ou subtração de produtos para identificar a mudança no número de produtos sobre a plataforma de pesagem.

3.3.2.3 Circuito do *keypad* para inserção de dados

A adição de um teclado de membrana ao projeto tornou possível que o usuário não apenas selecione o número de produtos no lote a ser pesado como também informe os dados que serão posteriormente enviados para o servidor. Esses dados e sua formatação para envio são discutidos na Seção 4.5.4. Normalmente a implementação deste tipo de dispositivo é feita através de bibliotecas pré-existentes, que implementam funções de *scan* do hardware para detecção do aperto de botões. No entanto tais implementações requerem que todas as saídas do teclado estejam conectadas ao microcontrolador, e com as restrições impostas pela utilização de outros dispositivos tornou-se necessário implementar o *keypad* de outra maneira. Foi utilizado o conceito de divisor de tensão [15] para dimensionar a tensão lida por uma porta analógica de maneira que cada botão resultasse em um valor distinto, conforme apresenta o esquemático da Figura 8. Vale notar que cada circuito equivalente gerado pela pressão de algum botão possui uma resistência equivalente única e, portanto, gera um valor de tensão único na porta analógica. O cálculo desse valor de resistência equivalente, bem como os valores das resistências utilizadas são abordados a seguir. Outra observação importante é que cada botão possui uma resistência associada em série, representada no esquemático de forma RKn onde n corresponde à tecla pressionada. Os valores destas resistências são discutidos e exibidos na Tabela 4.

Como o *keypad* é composto por múltiplos botões, observa-se o efeito de *bounce* após o operador apertar qualquer um deles. *Bounce* é um comportamento típico de botoeiras e *switches* mecânicas, e pode ser observado na Figura 9. Imediatamente após os contatos serem fechados ocorrem várias pequenas flutuações no sinal lido. Isso se dá pois o encaixe desse tipo de dispositivo não é perfeito, e demora alguns instantes até se estabilizar. As flutuações geradas nesses instantes de instabilidade podem, e muitas vezes geram, falsas leituras, como se o botão houvesse sido pressionado várias vezes.

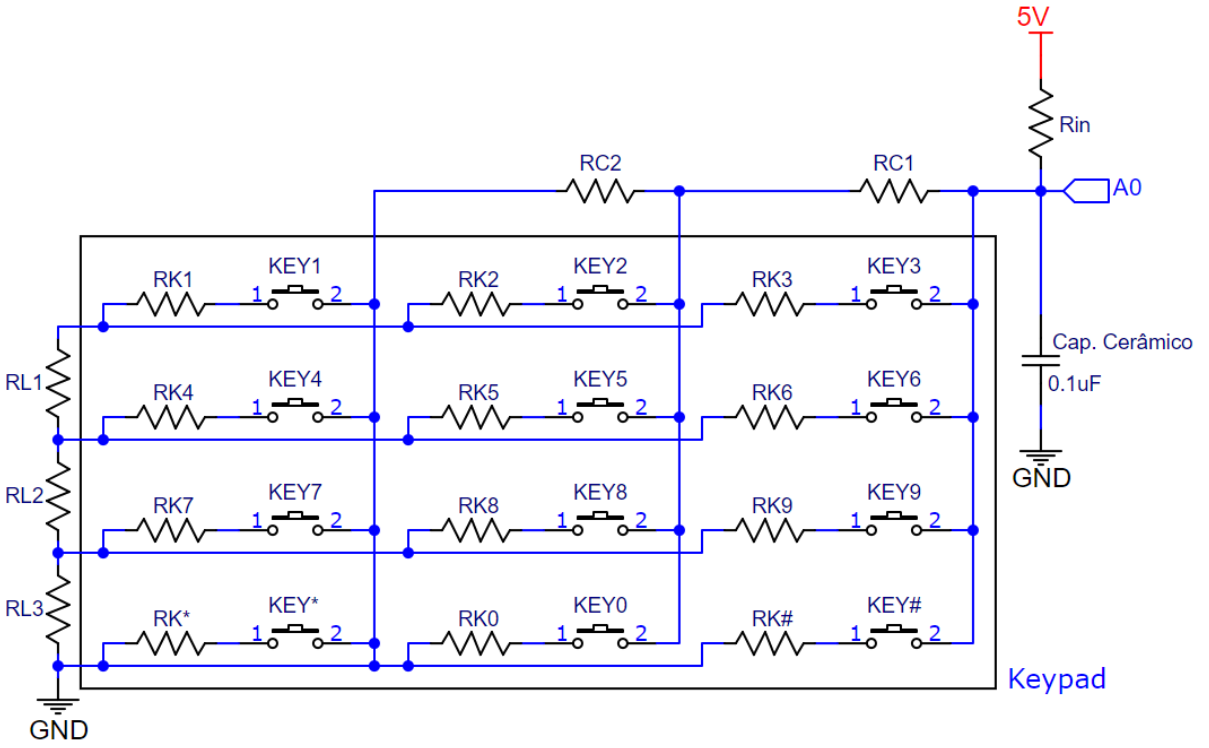

Figura 8 – Divisor de tensão do *keypad*. Fonte: Elaborada pelo autor.

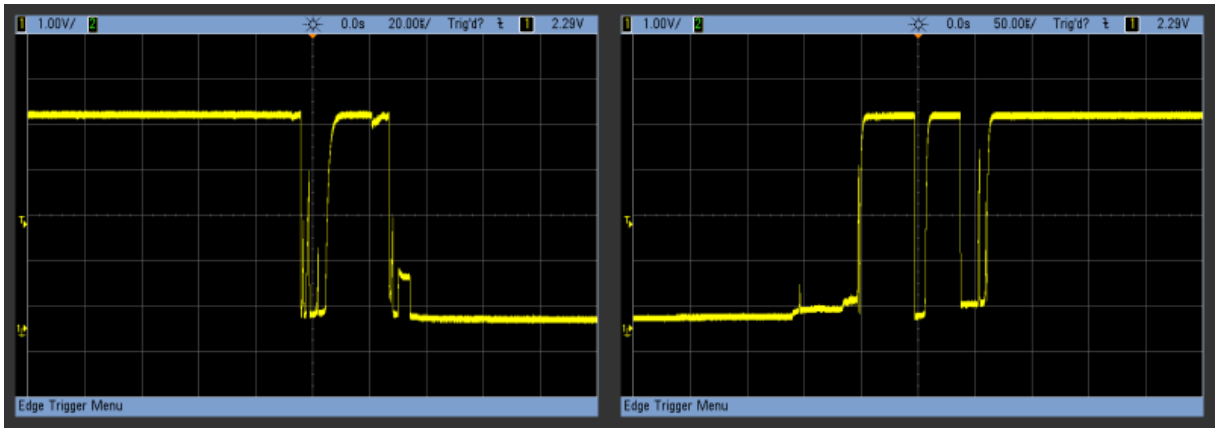

Figura 9 – Exemplo de *bounce* em um botão ou *switch* mecânica [25].

Para minimizar o efeito de *bounce* a nível de hardware é utilizado um capacitor de desacoplamento para a filtragem de sinais variáveis no período transitório. O efeito de *bounce* também é compensado via software, a função é abordada em mais detalhes na Seção 3.3.3.3. O projetista optou pelo uso de um capacitor cerâmico de $0.1\mu F$.

A partir das leis de Kirchhoff pode-se obter a tensão lida pelo ADC (conversor analógico-digital, do inglês *Analog-Digital Converter*). Sabendo que portas do microcontrolador configuradas como entrada equivalem à um resistor de $100M\Omega$ [26] em série, pode-se desprezar a corrente que por ali flui, visto que para uma tensão de alimentação de $5V$ a mesma é da ordem de *nano ampères* e, portanto, não influenciará nas medidas obtidas

por ser 3 ordens de grandeza menor que a menor corrente que flui pelo circuito. Assim, considera-se que esse nó encontra-se em aberto e que toda a corrente fornecida pela fonte de alimentação passará pelo circuito resultante do aperto de cada botão. O cálculo da corrente é feito através dos valores da resistência equivalente do circuito resultante e da tensão da fonte de alimentação, ou seja,

$$i_n = \frac{V_{in}}{R_{eq}} \tag{3.3}$$

sendo que o subscrito n serve para referir-se à um circuito específico gerado pelo fechamento do botão n.

Uma vez calculada a corrente pode-se calcular a queda de tensão gerada pelo resistor de entrada, e ao subtraí-la da tensão fornecida ao sistema tem-se o valor que é lido pelo ADC, isto é,

$$V_{ADC} = V_{in} - R_{in} * i_n = V_{in} - \frac{R_{in}V_{in}}{R_{eq}} = V_{in}\left(1 - \frac{R_{in}}{R_{eq}}\right) \tag{3.4}$$

Todas as expressões para o cálculo de R_{eq} encontram-se na Tabela 2. Uma expressão geral pode ser dada por:

$$R_{eq} = \sum_{i=a}^{b} RC_i + \sum_{i=c}^{d} RL_i + R_{in} \tag{3.5}$$

onde os limites a, b, c, e d dos somatórios variam para cada botão, sendo apresentados na Tabela 2.

Tabela 2 – Expressões para resistores equivalentes

Botão	a	b	c	d	Resistência equivalente (Ω)
1	1	2	1	3	$R_{eq} = RC_1 + RC_2 + RL_1 + RL_2 + RL_3 + R_{in}$
2	2	2	1	3	$R_{eq} = RC_2 + RL_1 + RL_2 + RL_3 + R_{in}$
3	0	0	1	3	$R_{eq} = RL_1 + RL_2 + RL_3 + R_{in}$
4	1	2	2	3	$R_{eq} = RC_1 + RC_2 + RL_2 + RL_3 + R_{in}$
5	2	2	2	3	$R_{eq} = RC_2 + RL_2 + RL_3 + R_{in}$
6	0	0	2	3	$R_{eq} = RL_2 + RL_3 + R_{in}$
7	1	2	3	3	$R_{eq} = RC_1 + RC_2 + RL_3 + R_{in}$
8	2	2	3	3	$R_{eq} = RC_2 + RL_3 + R_{in}$
9	0	0	3	3	$R_{eq} = RL_3 + R_{in}$
*	1	2	0	0	$R_{eq} = RC_1 + RC_2 + R_{in}$
0	2	2	0	0	$R_{eq} = RC_2 + R_{in}$
#	0	0	0	0	$R_{eq} = R_{in}$

A escolha dos valores das resistências foi feita de maneira a maximizar a diferença entre os valores de tensão lidos enquanto minimizando o espaço ocupado pelos mesmos na PCB. Assim, o uso de resistores em série para obtenção de valores mais precisos foi desconsiderado. Inicialmente foi compilada uma lista de possíveis valores válidos, onde denota-se um valor válido aquele que possui:

1. Um componente com tal valor resistivo disponível em tecnologia SMT (tecnologia de montagem de superfície, do inglês *Surface-mount technology*)

2. Devido às restrições de tempo para desenvolvimento do protótipo, o componente deve estar disponível em estoque no fornecedor escolhido.

3. Valor resistivo entre 10Ω e $47k\Omega$. Esses valores foram obtidos experimentalmente e constatou-se que valores fora desta faixa não apresentam resultados desejados de leitura de tensão pelo ADC.

A lista compilada junto ao fornecedor de componentes escolhido para o projeto reuniu 73 valores possíveis no intervalo de 10Ω a $47k\Omega$, dos quais 11 foram descartados por não cumprirem com as demais restrições impostas. A lista final, com 62 valores possíveis e 62^6 (56.800.235.584) combinações entre eles, foi então passada por um programa escrito pelo autor para selecionar as melhores resistências dado um certo intervalo. Diferente do contador e *hub* desenvolvidos, o programa foi escrito em Java. Como demonstrado na Tabela 4 e ilustrado na Figura 13, os valores de tensão lidos pelo ADC para cada botão não se distribuem uniformemente no intervalo entre o botão 1, que gera o maior valor, e o botão #, que gera o menor. Sabe-se que o valor lido pelo ADC de 10 bits do microcontrolador é um valor inteiro entre 0 e 1023, com o intervalo entre cada nível correspondente à tensão de alimentação dividida pela resolução do ADC (2^{10}, 1024). A diferença entre a tensão lida para um botão qualquer e o botão anterior, conforme a lista da Tabela 2, é o critério utilizado para a escolha da combinação de resistores pelo programa, pois deseja-se que a tensão lida pelo ADC diminua conforme os botões são percorridos na ordem disposta na Tabela 2. Assim, o programa dimensiona as resistências utilizadas no divisor de tensão de forma a aumentar ao máximo o intervalo entre a tensão lida por um botão e para o botão seguinte. Para isso, primeiramente calcula-se a tensão lida pelo ADC para cada botão, dada uma combinação de valores dos resistores. Em seguida é encontrada a diferença mínima entre as tensões calculadas. Caso essa menor diferença seja um valor negativo então a combinação de resistências é descartada e o programa segue diretamente para a próxima combinação. Esse comportamento é tal pois assume-se que conforme a lista de botões prossegue, seguindo a ordem da Tabela 2, as resistências equivalentes diminuem, e, consequentemente, diminui-se também o valor de tensão lido pelo ADC. Quando a resistência equivalente aumenta ao invés de diminuir corre-se o risco de diferentes botões possuírem valores de tensão lida similares, resultando em interpretações incorretas de qual botão foi pressionado. Caso o valor da menor diferença seja maior que zero então o programa irá compará-lo ao maior valor encontrado até o momento. Caso o valor da presente combinação seja maior que o valor da combinação favorita anterior, então essa combinação será marcada como a nova favorita e a execução continuará. O ciclo se repete várias vezes, cada vez encontrando novas combinações favoritas que ofereçam a maior

diferença mínima entre tensões. No final da execução o programa irá fornecer a combinação encontrada que melhor satisfaz essa condição. Há, porém, algumas considerações importantes sobre o funcionamento do programa:

- O programa só mede uma combinação por vez.
- Além da maior diferença mínima há também a potência dissipada pelos resistores e a corrente que flui pelo sistema que devem ser levadas em consideração. Afinal o circuito é alimentado pelo microcontrolador, e o mesmo possui limites no fornecimento de tensão para os componentes à ele conectados.
- Como o valor de tensão lido pelo ADC é um número inteiro pode que múltiplas combinações diferentes de resistores apresentem uma mesma diferença mínima, havendo a necessidade de um critério de desempate. Esse critério de desempate é a média entre as diferenças de tensão, e o circuito com a maior média é automaticamente escolhido pelo programa.
- A mudança de valores resistivos deve seguir uma ordem definida, para que a execução seja mais organizada. A ordem escolhida foi: primeiro varia-se RL_3, em seguida RL_2, RL_1, RC_2, RC_1, e por fim R_{in}. Para cada variação no valor de R_{in} são testadas 916.132.832 combinações.
- Como demonstrado pela Tabela 4, o *keypad* possui resistências não-nulas. Tais resistências não foram levadas em consideração em execuções iniciais do programa, porém para a escolha final as mesmas foram adicionadas ao cálculo.

Uma vez que o programa percorreu toda a lista de combinações possíveis, testando em média 8,38 milhões de combinações por segundo, é possível verificar se a combinação escolhida está de acordo com as considerações listadas acima. Caso sim então a combinação é escolhida, caso contrário deve-se ajustar o intervalo de possíveis valores para a resistência e tentar novamente. Alguns exemplos de execução do programa podem ser visualizados na Figura 10, demonstrando o *output log* parcial quando é encontrado, ou não, uma nova combinação favorita, na Figura 11, demonstrando o uso do critério de desempate para escolha da melhor combinação favorita, e na Figura 12, demonstrando o *output* ao final da execução, contendo a combinação favorita final, número de combinações verificadas, tempo de execução (em *ms*), e número de combinações testadas por segundo.

Os resultados reportados foram colocados em uma planilha de Excel, utilizada para verificar se os mesmos atendiam de fato as especificações do projeto sem que fossem refeitos os cálculos manualmente. Além disso o circuito foi montado e simulado no software LTSpice para uma verificação extra, principalmente em respeito à potência dissipada pelos resistores pois todos são avaliados em até $125mW$, um fator que também auxiliou

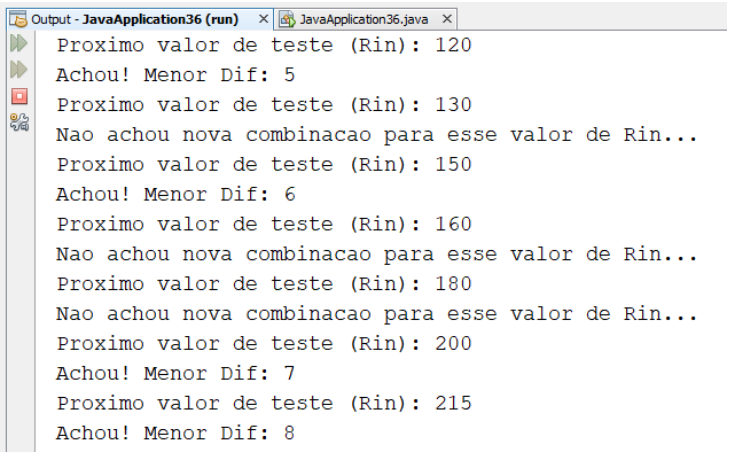

Figura 10 – Programa de seleção de valores de resistência, *log* parcial 1. Fonte: Elaborada pelo autor.

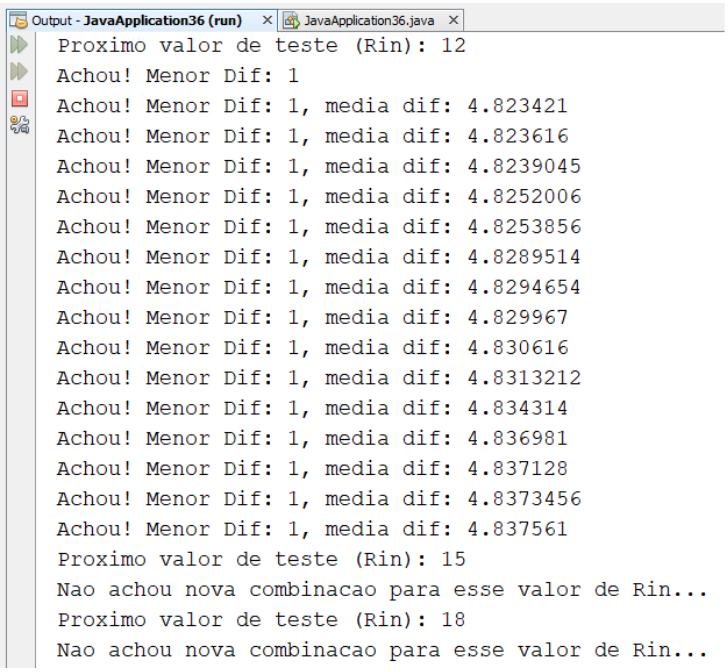

Figura 11 – Programa de seleção de valores de resistência, *log* parcial 2. Fonte: Elaborada pelo autor.

na escolha entre combinações que oferecessem comportamentos similares. Os valores finais encontrados e utilizados são apresentados na Tabela 3.

Ao inicialmente comparar-se os valores de tensão medidos para cada circuito equivalente com os valores teóricos esperados notou-se que o *keypad* possuía resistências equivalentes diferentes para cada circuito resultante. Assim, tais resistências foram medidas e os valores teóricos calculados foram ajustados. Na Tabela 4 estão listados os valores de tensão calculados sem (VC_1) e com (VC_2) as resistências internas do *keypad*, os valores de tensão realmente lidos pelo ADC (VL), o erro do valor medido comparado ao valor teórico VC_2, bem como os valores das resistências obtidos empiricamente.

```
Proximo valor de teste (Rin): 39000
Nao achou nova combinacao para esse valor de Rin...
Proximo valor de teste (Rin): 47000
Nao achou nova combinacao para esse valor de Rin...

Resistencias: 5900, 1000, 383, 2000, 1800, 1800,
Tensoes: 2.7216558, 2.5877013, 2.4465506, 2.304459, 2.1143498, 1.9106712, 1.7724288, 1.496021, 1.19
Tensoes ADC: 557, 529, 501, 471, 433, 391, 362, 306, 243, 200, 109, 6,
Resistencias eqs: 12948, 12229, 11553, 10944, 10223, 9549, 9140, 8419, 7745, 7334, 6610, 5936,
Melhores valores: 5900, 1000, 383, 2000, 1800, 1800,  (Rin, R1, R2, R3, R4, R5)
Operações total: 56800235584
Tempo decorrido: 6775860.0
Operações por segundo: 8382734
BUILD SUCCESSFUL (total time: 112 minutes 56 seconds)
```

Figura 12 – Programa de seleção de valores de resistência, *log* final. Fonte: Elaborada pelo autor.

Tabela 3 – Valores finais dos resistores para leitura analógica do *keypad*

Resistor	Valor (Ω)	Potência mínima (mW)	Potência máxima (mW)
R_{in}	5900	0,8797	4,1862
R_{C1}	1000	0,1491	0,7095
R_{C2}	383	0,0571	0,2717
R_{L1}	2000	0,2982	1,4190
R_{L2}	1800	0,2684	1,2771
R_{L3}	1800	0,2684	1,2771

Tabela 4 – Valores de tensão calculados e lidos para cada entrada do *keypad*

Tecla	$VC_1(V)$	$VC_2(V)$	$VL(V)$	Erro %	Resistência (Ω)
1	2,710	2,722	2,711	-0,37%	65,4
2	2,517	2,588	2,585	-0,09%	346,3
3	2,434	2,446	2,435	-0,46%	53,2
4	2,289	2,304	2,288	-0,73%	61,5
5	2,015	2,114	2,108	-0,31%	340,3
6	1,895	1,911	1,894	-0,89%	49,1
7	1,752	1,772	1,760	-0,72%	57,2
8	1,350	1,496	1,486	-0,65%	336,3
9	1,169	1,191	1,166	-2,11%	44,9
*	0,949	0,978	0,958	-1,98%	51,2
0	0,305	0,537	0,536	-0,18%	326,9
#	0,000	0,030	0,026	-14,42%	35,9

O nível lido pelo ADC para cada botão, bem como a diferenças real e ideal, a diferença ideal sendo de um sistema que utilizasse todo o intervalo distribuído igualmente para essa leitura, são apresentadas na Figura 13.

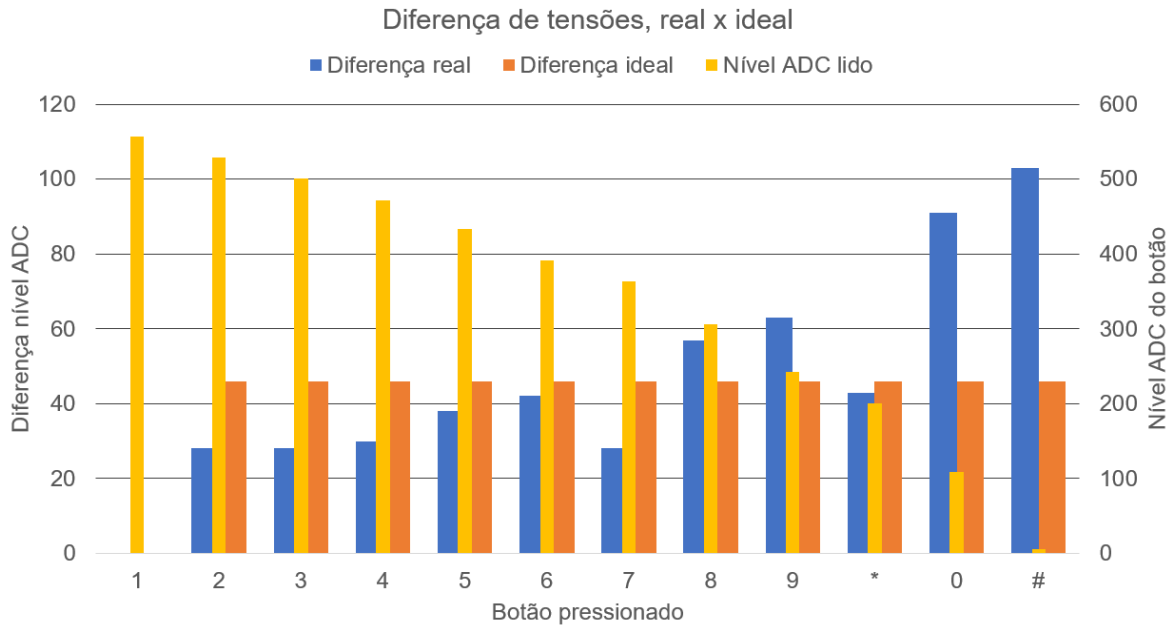

Figura 13 – Diferença na tensão lida para diferentes botões do *keypad*. Fonte: Elaborada pelo autor.

3.3.2.4 Interface entre microcontrolador e módulo RF: O protocolo SPI

O protocolo SPI (do inglês *Serial Peripheral Interface*), criado em 1979 pela Motorola, é um protocolo de comunicação síncrona muito utilizado para componentes de circuitos embarcados devido à sua ênfase em comunicação de curta-distância. É utilizado para dispositivos como cartões SD, diversos tipos de sensores, conversores AD/DA (analógico-digital e digital-analógico, respectivamente), displays, e até mesmo memórias embarcadas como a EEPROM (tipo de memória não volátil que permite leitura e escrita de bytes individuais, do inglês *electronically erasable programmable read-only memory*). Com uma arquitetura de mestre-escravo esse protocolo permite a comunicação entre um mestre e múltiplos escravos, todos compartilhando os barramentos de dados enquanto cada escravo possui um barramento de seleção individual. O protocolo opera com quatro fios possuindo funções distintas. São eles:

- *SCK (Serial Clock)*: Utilizado pelo *master*, sincroniza os *clocks* dos *slaves* e é necessário para comunicação síncrona. É um barramento compartilhado entre todos os dispositivos.
- *MOSI (Master Out Slave In)*: Saída de dados do *master* e entrada de dados do *slave*, utilizado para envio de dados entre o mestre e o dispositivo selecionado. Também é compartilhado entre todos os dispositivos.
- *MISO (Master In Slave Out)*: Entrada de dados do *master* e saída de dados do *slave*, utilizado para recebimento de dados entre o mestre e o dispositivo selecionado.

Também é compartilhado entre todos os dispositivos.

- *SS/CS (Slave Select/Chip Select)*: Barramento utilizado pelo mestre para selecionar qual dispositivo deverá ler o barramento de dados. É mantido em *HIGH* para os dispositivos que devem ignorar o barramento e colocado em *LOW* no dispositivo que realizará a leitura dos dados. Cada dispositivo deve possuir uma conexão direta com o mestre referente à esse barramento.

A Figura 14 demonstra o uso do protocolo SPI para comunicação com múltiplos escravos.

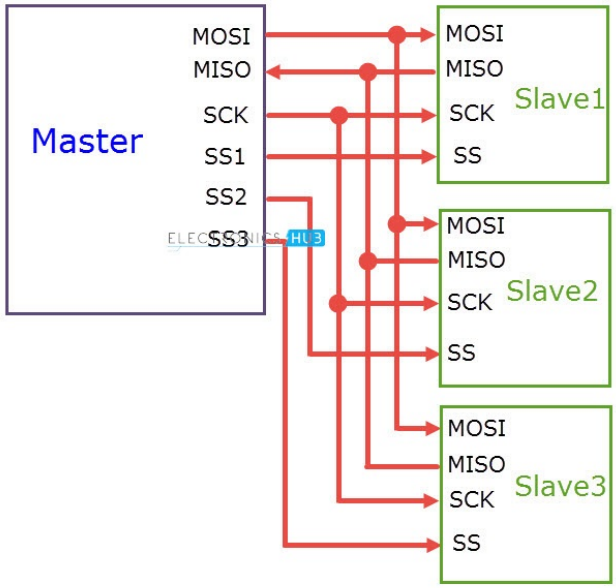

Figura 14 – Figura demonstrando os diferentes barramentos do protocolo SPI [27], modificada pelo autor.

Como discutido na Seção 3.3.1, o módulo de RF utilizado foi o módulo nRF24L01+, criado e produzido pela empresa Nordic Semiconductor. O uso deste módulo para recebimento e envio de dados na rede de comunicação é abordado no Capítulo 4, por enquanto é discutida apenas sua maneira de comunicar-se com o microcontrolador. Além dos barramentos pertencentes ao protocolo SPI o módulo ainda possui uma conexão, CE ou *Chip Enable*. Essa conexão permite que o microcontrolador configure o módulo de maneira a alternar entre os modos TX (transmissor) e RX (receptor). Este módulo é compacto, podendo ser facilmente incluído na PCB do projeto, apresenta alta taxa de transmissão de dados, considerando o contexto de dispositivos embarcados (até 2 Mbps), baixo consumo de energia, e opção de antena embutida na placa ou conetor de antena externa, o que aumenta seu alcance para um máximo de 1000 metros sem interferências e obstáculos, segundo informações do fabricante. Além disso, apesar de ser alimentado com 3.3V seus contatos podem ser operados em tensão de 5V, assim podem ser ligados diretamente às

portas digitais do microcontrolador sem que haja preocupação em alterar o valor de tensão das mesmas. O esquema de suas conexões com o microcontrolador é demonstrado na Figura 15.

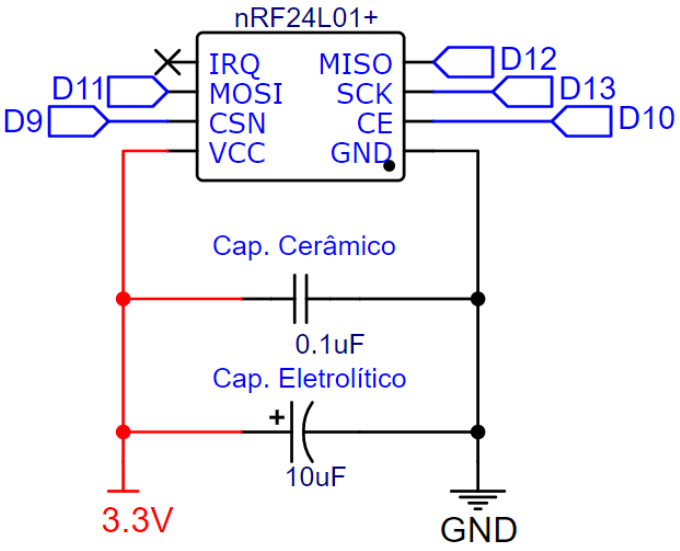

Figura 15 – Conexões do módulo nRF24L01+. Fonte: Elaborada pelo autor.

Como o módulo opera em alta velocidade e lida com sinais analógicos de alta frequência há a preocupação de que ruídos na alimentação do mesmo possam interferir em seu funcionamento. Assim, foram utilizados dois capacitores em paralelo próximos aos pinos de alimentação e terra do módulo com a função de realizar desacoplamento. Esses capacitores permitem que sinais de tensão contínua passem para o módulo enquanto filtram perturbações e demais sinais analógicos, ligando-os diretamente com o terra e os impedindo de chegar até o módulo.

A Tabela 5 apresenta informações sobre as condições de operação, de acordo com seu *datasheet*, disponível na Seção 3.3.1.

Tabela 5 – Condições operacionais do módulo nRF24L01+

Medida	Mínimo	Típico	Máximo	Observações
Tensão de alimentação	$-0.3V$		$3.6V$	
Tensão nos contatos	$-0.3V$		$5.25V$	
Corrente em *stand-by*		$320\mu A$		
Corrente em modo de transmissão	$7.0mA$	$7.0mA$	$11.3mA$	a
Corrente em configuração modo TX		$8.0mA$		b
Corrente em modo de recepção	$12.6mA$	$13.1mA$	$13.5mA$	c
Corrente em configuração modo RX		$8.9mA$		d

a. Valor mínimo corresponde à transmissão com ganho da antena em $-18dBm$, enquanto o valor máximo corresponde a $0dBm$. O valor típico foi a configuração utilizada para o projeto ($-18dBm$).
b. Consumo médio enquanto inicia o módulo em modo TX e na troca de modo RX para TX.
c. Valor mínimo corresponde à uma taxa de recepção de 250kbps, enquanto o valor máximo corresponde a 2Mbps. O valor típico foi a configuração utilizada para o projeto (1Mbps).
d. Consumo médio enquanto inicia o módulo em modo RX e na troca de modo TX para RX.

3.3.2.5 Demais componentes: LEDs e botão de *reset*

Outra maneira que o operador possui de realizar a entrada de dados no sistema é através de um botão de *reset*. Esse botão, ligado à porta digital 2 do microcontrolador, faz uso de uma interrupção com gatilho de *falling edge* para detectar quando o operador o pressiona, podendo assim alterar seu estado a qualquer ponto durante a execução do código. Interrupções são instruções especiais na execução de um programa. Quando provocada, uma interrupção sinaliza para o microprocessador que, assim que possível, pare a execução atual, salve seu estado, e execute uma função especial para tratar da interrupção, chamada de ISR (rotina de serviço de interrupção, do inglês *Interrupt Service Routine*). Ao término da execução da ISR o microprocessador continua a execução do código do ponto aonde foi interrompido. No microcontrolador utilizado é recomendado que ISRs sejam o mais breve possível, assim quando deseja-se a ocorrência de um evento desencadeado por interrupção normalmente a ISR meramente levantará uma *flag*, ou seja, modificará uma variável booleana, e deixará a execução do evento por conta de outras funções, chamadas normalmente. Além disso, variáveis modificadas por uma função ISR devem ser declaradas como voláteis, ou `volatile`, e são carregadas diretamente da memória RAM quando utilizadas. A *tag* `volatile` indica que uma variável pode ser modificada por outros fragmentos de código que não o código atual, como é o caso com múltiplas *threads* ou interrupções. Funções de interrupção no microcontrolador Arduino UNO podem sem invocadas de quatro maneiras diferentes [28]: `LOW`, `CHANGE`, `RISING`, e `FALLING`. Como o botão de *reset* utiliza um resistor de *pull-up*, demonstrado na Figura 16, e deseja-se que a interrupção seja invocada somente uma vez, a única opção viável é

`FALLING`. Tal opção indica que somente invocará a interrupção quando o pino D2 ir do estado *HIGH* para *LOW*.

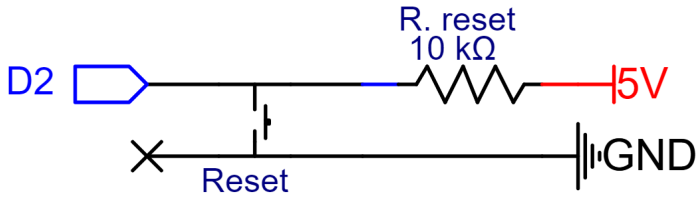

Figura 16 – Conexões do botão de *reset*, também utilizado como seletor. Fonte: Elaborada pelo autor.

Há também a possibilidade de configurar o botão com um resistor de *pull-down*. Resistores de *pull-up* e *pull-down* são necessários pois a porta digital deve, em todos os momentos, estar conectada à uma tensão conhecida ou ao terra. Portas que encontram-se sem conexão são ditas como flutuantes, e podem representar tanto *HIGH* quanto *LOW*, normalmente resultando em problemas. Nos microcontroladores Arduino que operam em 5V, como o UNO, o nível lógico *HIGH* em portas de entrada significa uma tensão lida de 3.0V ou mais. Já o nível lógico *LOW* significa uma tensão lida de 1.5V ou menos. Vale notar que, como o intervalo de tensão para detecção do nível *LOW* é menor, resistores de *pull-down* normalmente possuem valores menores de resistência quando comparados a resistores de *pull-up*. Além disso, resistores de *pull-up* e *pull-down* também servem para limitar a corrente que flui pelo circuito quando o botão é pressionado. A Figura 17 demonstra como é feita a conexão destes resistores para um botão.

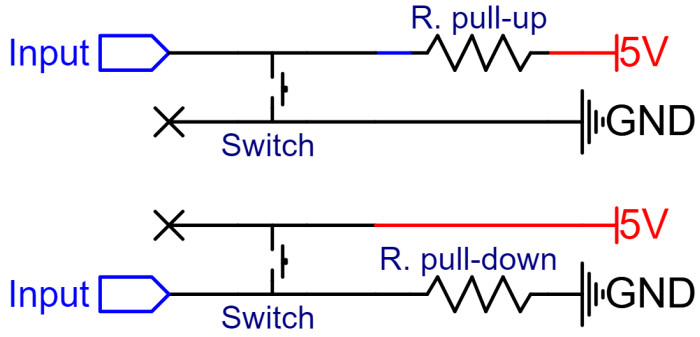

Figura 17 – Diferença na conexão de resistor *pull-up* e *pull-down*. Fonte: Elaborada pelo autor.

Um importante fator do contador automático é a habilidade de avisar o operador quando um número x de produtos é alcançado. Há diferentes tipos de dispositivos de alerta que podem ser usados para tal função e os mais utilizados são os indicadores sonoros e/ou luminosos. Apesar de simples, indicadores sonoros são muito eficientes para sinalização, pois mesmo simples *buzzers* emitem sons de até 85dB, suficientes para avisar

operadores em ambientes calmos que estejam até 10 metros de distância. Infelizmente ambientes industriais são altamente ruidosos, e devido à isso indicadores sonoros não foram utilizados para o projeto, apesar da concepção inicial do mesmo incluí-los. Em seu lugar foram utilizados luzes de *LED* (do inglês *Light-Emitting Diode*). O uso de LEDs como indicadores possui a vantagem de um menor consumo de energia, em contrapartida, como desvantagem tem-se uma maior dificuldade em informar operadores que não estejam prestando atenção aos indicadores. Tendo em vista porém que o sistema é usado em estações onde os operadores encontram-se estáticos, bem como o posicionamento dos LEDs de sinalização próximos ao *display* LCD, conforme apresentado na Seção 3.3.4.1 e Seção 3.3.4.2, os mesmos mostram-se efetivos em obter a atenção do operador quando há a necessidade de transmitir-lhe alertas. O projeto conta com três indicadores luminosos, nas cores vermelho, verde, e amarelo, e suas conexões são demonstradas na Figura 18. Essas cores foram escolhidas pois, além de serem três das quatro cores de LED mais amplamente disponíveis no mercado, representam três níveis distintos de alerta, sendo verde uma situação de operação normal, amarelo uma situação onde o operador deve prestar atenção à possíveis alertas do sistema, e vermelho uma situação crítica ou de erro, necessitando ação ou atenção redobrada do operador.

Como demonstrado na Figura 18 o acionamento dos indicadores de LED é feito através do uso de dois CIs (circuitos integrados) e não há a inclusão de resistores. Circuitos integrados são dispositivos de baixo custo, normalmente baixa complexidade, compostos de diversos circuitos elétricos, e que servem propósitos específicos. A utilização de tais componentes se dá pelo desejo de acionar um número de LEDs superior ao número de portas disponíveis. Para tal, faz-se uso de um demultiplexador. O demultiplexador, ou demux, é um dispositivo capaz de encaminhar um sinal único para múltiplas saídas, necessitando n canais seletores para selecionar entre 2^n saídas. A Tabela verdade das entradas e saídas do demux utilizado pode ser visualizada na Figura 19. Nota-se que ao ligar as entradas `DATA` (`1C`) em 5V e `STROBE` (`1G`) ao terra, podem-se usar apenas duas entradas digitais para controlar o estado de quatro saídas diferentes. Nota-se também que a saída selecionada fica no estado LOW, enquanto as demais encontram-se em $HIGH$, demonstrando a necessidade de um inversor. A ausência de resistores para controle da corrente fornecida aos LEDs se dá devido à limitações do fornecimento de corrente dos próprios CIs. Ao consultar seus *datasheets* nota-se que o demux é capaz de fornecer um máximo de $16mA$ para saídas de nível LOW, enquanto o inversor fornece no máximo $8mA$. Caso ligado diretamente à tensão de 5V da porta digital tal corrente seria o equivalente à utilização de um resistor de 625Ω, e é aceitável perante os requisitos de consumo de energia do sistema.

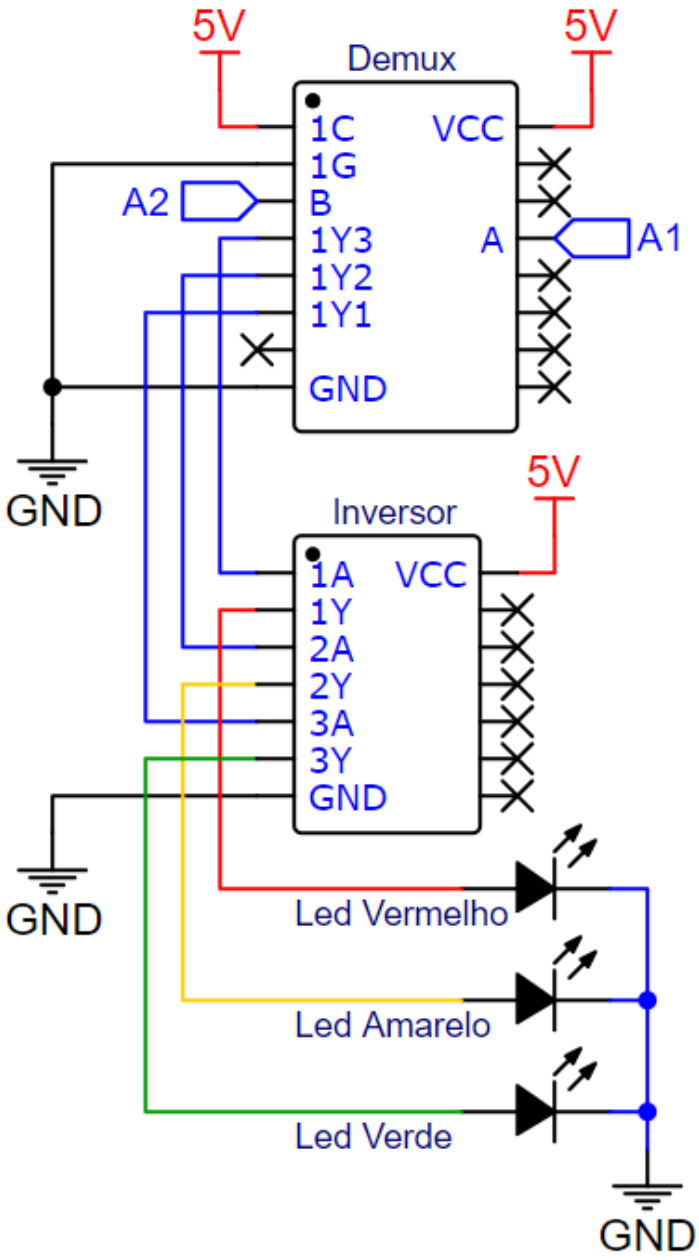

Figura 18 – Conexões dos LEDs com CIs de demux e inversor. Fonte: Elaborada pelo autor.

3.3.3 Software

Toda a programação do sistema foi realizada na IDE (ambiente de desenvolvimento integrado, do inglês *Integrated Development Environment*) Arduino, salvo o programa para cálculo dos valores de resistência para o divisor de tensão do *keypad*, apresentado na Seção 3.3.2.3. As linguagens utilizadas foram C e C++. A filosofia para criação do código, tanto do contador quanto do *hub*, é de que todo comportamento que se repete mais de uma vez receba sua própria função. Além disso funções devem ser breves, e realizar

INPUTS				OUTPUTS			
SELECT		STROBE	DATA				
B	A	$1\overline{G}$	1C	1Y0	1Y1	1Y2	1Y3
X	X	H	X	H	H	H	H
L	L	L	H	L	H	H	H
L	H	L	H	H	L	H	H
H	L	L	H	H	H	L	H
H	H	L	H	H	H	H	L
X	X	X	L	H	H	H	H

INPUTS				OUTPUTS			
SELECT		STROBE	DATA				
B	A	$2\overline{G}$	2C	2Y0	2Y1	2Y2	2Y3
X	X	H	X	H	H	H	H
L	L	L	L	L	H	H	H
L	H	L	L	H	L	H	H
H	L	L	L	H	H	L	H
H	H	L	L	H	H	H	L
X	X	X	H	H	H	H	H

Figura 19 – Tabela verdade do demux [19].

um propósito especifico, podendo beneficiar-se de *overloading* quando realizam propósitos ligeiramente diferentes baseado no tipo de dado que lhes é passado como parâmetro. Outra consideração importante é sobre o uso de *delays*. A função `delay()` é uma função especial, que recebe como parâmetro um número do tipo `unsigned long` correspondente à milissegundos, e interrompe o funcionamento do microprocessador durante estes milissegundos. É bastante útil para temporizar comandos e realizar certos comportamentos com dependências temporais, porém pode levar a diversos tipos de problemas quando há a necessidade de leitura de dados continuamente. A rede sem fio criada para compartilhamento de dados, como é explicado no Capítulo 4, necessita que os Nós estejam conectados e prontos para receber ou enviar pacotes a todo momento, pois pacotes de Nós remotos precisam ser encaminhados por outros Nós para chegar ao *hub*. Assim, Nós que estejam "travados" em *delays* atrasam, e por vezes até interrompem, a comunicação em rede. Similarmente, a leitura de dados de dispositivos como o *keypad* ou as células de carga é contínua, e caso seja interrompida por um *delay* pode resultar na perda de dados importantes. Assim, foi implementada uma função especial `Delay`, que realiza o atraso desejado em diversas etapas do código sem que haja o bloqueio de funções importantes como leitura de *inputs* e manutenção da rede de comunicações. Atrasos são um funcionamento desejado em diversas etapas do funcionamento, desde manter informações para o operador no *display* LCD até aguardar um intervalo para tentar novamente a conexão à rede após uma tentativa falha, e assim a implementação da função `Delay` mantém essa funcionalidade com um impacto mínimo no resto do código.

Foram utilizadas diversas bibliotecas externas para implementação das funcionalidades

desejadas, removendo assim a necessidade de reescrever códigos que, após serem utilizados por milhares de usuários, encontram-se extremamente otimizados. Cada biblioteca possui uma licença à ela associada, e a lista de bibliotecas utilizadas e suas respectivas licenças pode ser encontrada na Tabela 6.

Tabela 6 – Bibliotecas utilizadas e licenças

Biblioteca	Licença	Projeto
`LiquidCrystal`	LGPL	Contador
`HX711_ADC`	MIT	Contador
`RF24`	GPLv2	Ambos
`RF24Network`	GPLv2	Ambos
`RF24Mesh`	GPLv2	Ambos
`ArduinoJson`	MIT	*Hub*
`AutoConnect`	MIT	*Hub*
`ESP8266WiFi`	GPLv2.1	*Hub*
`ESP8266WebServer`	GPLv2.1	*Hub*
`WiFiUdp`	LGPL	*Hub*

3.3.3.1 Ciclo de funcionamento

O contador conta com três etapas de funcionamento distintas: configuração, medição, e *reset*, podendo alternar entre elas de acordo com o fluxograma exposto na Figura 20. A etapa de configuração é a primeira a ser executada uma vez que o sistema é energizado e engloba a declaração de variáveis globais, e a função `setup()`, que realiza as configurações necessárias para comunicação em rede e com os dispositivos. As bibliotecas utilizadas pelo contador e suas funções são:

- `LiquidCrystal`. Biblioteca nativa da IDE Arduino, a biblioteca `LiquidCrystal` é responsável pela comunicação com o *display* LCD. Durante a execução do código é criado um objeto da classe LiquidCrystal, e são passados como parâmetros ao construtor os pinos *enable*, *rs*, *rw* (opcional), e os pinos de dados (D0 à D7 para modo do controle com 8 pinos de dados, D4 à D7 para modo de controle com 4 pinos de dados). Como explicado na Seção 3.3.2.1 o pino *rw* corresponde à leitura (*HIGH*) ou escrita (*LOW*) de dados, e como somente há interesse em realizar a escrita de dados o mesmo é ligado diretamente no terra e não é passado como parâmetro para o construtor. A biblioteca implementa métodos que facilitam a utilização do *display* sem que haja a necessidade de preocupação com formato de instruções, *timing*, endereçamento de registradores, *bit shifting*, entre outras facilidades.

- `HX711_ADC`. Biblioteca terceirizada, implementa métodos para controle do hardware do conversor, bem como várias das configurações abordadas na Seção 3.3.2.2. Durante a execução do programa é criado um objeto da classe HX711_ADC, que recebe

como parâmetros os pinos DT e SCK. Além disso, ao iniciar-se a balança a operação de tara já é automaticamente realizada durante um intervalo, que também pode ser passado como parâmetro. Alguns dos métodos importantes implementados por essa biblioteca incluem a função de tara, com e sem o bloqueamento da execução do programa, funções de *power up* e *power down* para uso em sistemas onde há a preocupação com baixo consumo de energia, além de funções restritas como a conversão AD e obtenção do conjunto de medições temporário citado na Seção 3.3.2.2. Além disso a biblioteca conta com um arquivo de configuração, através do qual é possível alterar vários parâmetros de utilização, como o número de amostras por segundo (SPS), ganho do amplificador, e tamanho do conjunto temporário.

- `RF24`. Biblioteca terceirizada, é responsável pela comunicação e configuração do módulo nRF24L01+. Implementa métodos para abertura e fechamento de canais de transmissão de dados (*pipes*, o módulo nRF24L01+ pode possuir até 5 canais simultaneamente), envio de pacotes diretamente entre módulos, seleção de frequência e configuração de parâmetros de transmissão, como velocidade de transmissão de dados e ganho da antena. Bibliotecas RF24 subsequentes constroem sobre os métodos por ela estabelecidos.

- `RF24Network`. Biblioteca terceirizada, é responsável pela comunicação de múltiplos módulos nRF24L01+ em rede. Implementa métodos para criação de *headers*, envio de pacotes em rede, gerenciamento de comunicações com diversos módulos, bem como conexão e desconexão da rede (especialmente útil para módulos que enviam dados esporadicamente e consomem pouca energia).

- `RF24Mesh`. Biblioteca terceirizada, responsável pela rede *mesh*. Em redes *mesh* os Nós se distribuem livremente, conectando-se com múltiplos outros Nós e sem hierarquia, porém devido à limitações do hardware nRF24L01+ em somente aceitar cinco conexões simultâneas, a biblioteca implementa algumas das funcionalidades de redes *mesh* para assegurar o comportamento dinâmico da rede, enquanto mantendo sua topologia de árvore. São implementados pela biblioteca métodos para endereçamento automático de Nós, encaminhamento de pacotes à Nós remotos, e aumento de confiabilidade na conexão através de módulos de renovação de endereço e reconexão automática na perda de sinal.

3.3.3.2 Estrutura de dados a ser enviada

Considerando o sistema produtivo exposto no Capítulo 2 e que operadores devem sempre realizar o apontamento das ordens de produção antes e depois da sua realização, foi então criada uma estrutura para manter os dados do apontamento organizados. Os dados contidos nessa estrutura são descritos na Tabela 7.

Figura 20 – Fluxograma do ciclo de funcionamento do contador. Fonte: Elaborada pelo autor.

A Tabela 7 em código C é declarada da seguinte maneira:

```
const uint8_t sizeArray = 9;
struct Data {
    char OP[sizeArray];
    uint16_t operacao;
    uint16_t operador;
    uint16_t linha;
    uint16_t produtos;
};
Data dados;
```

Nota-se que é declarada uma variável própria para o tamanho do vetor de caracteres, e o nome da variável é utilizada na declaração do vetor, ao invés de declarar o vetor utilizando diretamente o valor. Além de manter consistência com outras partes do código tal operação torna possível acessar facilmente o tamanho do vetor de caracteres a qualquer momento no código, sem a necessidade de realizar mais operações utilizando os métodos próprios da linguagem C, como `sizeof()` ou `strlen()`.

3.3.3.3 Inserção de informações: *keypad* e botão de *reset*

Em alguns momentos durante o funcionamento do contador é necessário o uso dados que são fornecidos pelo operador. Estes dados correspondem à identificadores na linha de produção, como a ordem de operação (OP) do produto, a operação realizada na estação, e o próprio código de identificação do operador ou da linha de produção. Tais dados são obtidos pelo código na forma de números, de formato `long` para a ordem de produção e `uint16_t` para os demais.

Com as considerações expostas na Seção 3.3.3 em mente foram criadas três funções diferentes para obtenção dos dados via leitura do *keypad* e inserção dos dados na estrutura, listadas na Tabela 8.

A função `getKeypadInput` lê os números inseridos pelo *keypad* e realiza manipulações aritméticas para obtê-los na ordem que foram pressionados como um número só, sem que seja necessária a utilização de vetores ou *type casting* para essa manipulação. Em prática

Tabela 7 – Dados necessários para apontamento

Dado	Tipo	Obs
Ordem de produção (OP)	Vetor de `char`	a
Número da operação	`Unsigned int 16 bits`	b
Código do operador	`Unsigned int 16 bits`	b,c
Número da linha	`Unsigned int 16 bits`	b,c
Número de produtos	`Unsigned int 16 bits`	b

a. O sistema pré-existente na empresa gera ordens de operação iniciadas em 0, algo que é automaticamente removido em variáveis numéricas. A conversão de número para vetor de caracteres é simples e pode ser realizada tanto nos Nós contadores quanto no *hub*. Afim de minimizar o número de operações realizadas pelo *hub* decidiu-se deixar essa conversão a cargo de cada Nó contador. Não há a necessidade de usar um tipo `uint8_t` ou similar na declaração de variáveis `char` quando trabalha-se nas linguagens C e C++ pois, independente do sistema, uma variável `char` sempre representa e ocupa 8 bits. Caso trabalhe-se com componentes programados em linguagens diferentes, por exemplo Java, há a necessidade de especificar o tamanho da variável de maneira similar aos demais dados da estrutura.
b. É importante declarar variáveis numéricas contendo exatamente o número de bits necessários quando enviam-se dados entre diferentes microcontroladores. Variáveis do tipo `int` representam um diferente número de bits em diferentes sistemas. Para os microcontroladores utilizados neste projeto, Arduino UNO e ESP8266, este tipo de variável representa 16 e 32 bits, respectivamente. 16 bits `unsigned` (sem sinal) são suficientes para representar números entre 0 e 65.535, o que, para o sistema, engloba todos os possíveis números de operação, códigos de operador, números de linha, e números de produtos em um lote. Futuramente é possível substituir estas variáveis por tipos que representem 32 bits (`uint32_t`) caso necessário.
c. O código do operador e número da linha são valores mutuamente exclusivos, ou seja, apenas um dos dois é utilizado em cada apontamento. Isso se deve ao fato de que cada processo produtivo é realizado exclusivamente por um operador ou uma linha de operadores, e caso seja realizado pela linha de operadores o sistema pré-existente na empresa já faz a associação entre número da linha e código dos operadores que nela trabalham. Assim, ao realizar-se o apontamento para uma linha basta informar o número da mesma.

Tabela 8 – Funções para inserção e validação de dados

Função	Parâmetros	Retorno
`getKeypadInput`	`bool`, `byte`, `byte`, `bool`	`long`
`checkInput`	`long`, `bool`, `bool`	`bool`
`preencheDados`	Nenhum	`void`

é utilizado um vetor de variáveis `char` contendo os dígitos correspondentes às teclas do *keypad*, pois o *keypad* possui caracteres não-numéricos, e através da leitura da porta analógica e comparação com os resultados esperados, demonstrados na Seção 3.3.2.3, é determinada qual tecla foi pressionada. O resultado dessa leitura é o caractere lido, no tipo `char`. Com uma breve consulta à uma Tabela *ascii*, como demonstrada na Figura 21, nota-se que os caracteres de 0 a 9 possuem código *ascii* igual à seu valor numérico + 48, então para obter um valor numérico equivalente à tecla pressionada simplesmente subtrai-se 48 do valor obtido na leitura. Isso resulta em valores de 0 a 9 para dígitos numéricos, -6 para o dígito "*" e -13 para o dígito "#".

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Figura 21 – Tabela Ascii [29].

A manipulação aritmética realizada para obter-se o valor sequencial é simples. Primeiro cria-se uma variável do tipo `long`, chamada "`soma`", que é a variável de retorno da função, e uma variável do tipo `int` para recordar-se da última tecla pressionada. Para um número de 0 à 9 realizam-se as seguintes operações:

```
1 if (numero <= 9 && numero >= 0) {
2     soma = soma * 10;
3     soma += numero;
4     ultimoNum = numero;
```

Dessa maneira a soma incorpora o número atual, que passa a ser o número anterior e é armazenado na variável `ultimoNum`. Esse armazenamento é necessário pois os caracteres

não-numéricos do *keypad* possuem funções especiais. A tecla "*" possui a função de apagar o último número inserido, e caso pressionada duas vezes zera a `soma`. Assim é necessário saber qual foi o último número pressionado, pois a variável número irá assumir valor `-6` quando a tecla "*" for pressionada, para que possam-se realizar as operações inversas na soma. As operações aritméticas realizadas pelo aperto do botão "*" são apresentadas a seguir:

```
if (numero == -6) {
    if (ultimoNum == -6) {
        soma = 0;
    } else {
        soma = soma - ultimoNum;
        soma = soma / 10;
        ultimoNum = -6;
```

A tecla "#" é utilizada como tecla de confirmação para inserção de dados. Uma vez que for pressionada então a função irá terminar sua execução e retornar a variável `soma`. Há também um limite de 10 números, inerente do espaço de armazenamento de variáveis `long`, para a função de inserção de dados. Caso sejam inseridos 10 números a mesma irá assumir comportamento similar à tecla "#" e retornar o valor da variável `soma`, mesmo que o número inserido não tenha sido confirmado. Vale notar que apagar caracteres com a tecla "*" diminui o contador, e zerar a soma, apertando "*" duas vezes seguidas, também zera o contador. A função retornar uma soma de 10 dígitos não é problemática para o funcionamento das demais partes do código, pois tal número sempre dará erro na verificação e o processo de inserção de dados recomeçará (ou a calibração do *keypad* será realizada, caso tenham sido feitas 3 tentativas incorretas de inserção de dados).

Além disso, como visto na Seção 3.3.2.3, o efeito de *bounce* é minimizado no hardware através do uso de um capacitor ligando o ponto onde a tensão é lida pelo ADC com o terra. Mesmo assim, ainda há a necessidade de realizar-se uma compensação via software. É trabalho desta função realizar o *debouncing* do *keypad*, para tal a função implementa uma chamada para a função especial `Delay` quando lê que alguma tecla foi pressionada. O curto intervalo proporcionado pela função `Delay` permite a estabilização do contato para, ao retornar à função `getKeypadInput`, executar o restante da função. Além de *debouncing* a função também é responsável por exibir a variável `soma` no *display* LCD, e faz o posicionamento do cursor através dos parâmetros passados em sua chamada. O primeiro parâmetro *bool* refere-se a exibir (`true`), ou não (`false`) a variável no *display*. Os parâmetros `byte` seguintes cuidam do posicionamento, enquanto o parâmetro `bool` final é restrito a ser chamado somente para teste de calibragem do *keypad*. Por fim, a tecla "#" encerra a execução da função e retorna a variável `soma`.

A função `checkInput` verifica se o número passado como parâmetro `long` é um dado válido, baseado no estado das *flags* booleanas, também passadas como parâmetro, que definem o intervalo ao qual o número deve pertencer e se zero é considerado um número

válido. A função retorna `true` (número válido) caso o mesmo encontre-se dentro do intervalo especificado e seja maior (ou igual) à zero. A primeira *flag* diz respeito ao intervalo, e diferencia testar se o número pertence ao intervalo `long` (variável `true`, intervalo de ± 2.147.483.647) ou `unsigned int` (variável `false`, intervalo de 0 a 65.535). A segunda *flag* indica se zero é um número válido (`true`) ou inválido (`false`). A função `checkInput` normalmente é chamada passando-se a variável `soma` retornada pela função `getKeypadInput` como parâmetro para verificação.

Já a função `preencheDados` é responsável por, sequencialmente, chamar as funções anteriores para receber dados do operador e armazená-los na estrutura para envio. Essa função lida com mecanismos de *timing* para apresentar instruções ao operador sobre o tipo de dado a ser inserido, bem como dá-lo tempo suficiente para ler mensagens de erro ou confirmação causadas pelo retorno da função `checkInput`. É nessa função também que é feita a manipulação do número correspondente à ordem de produção em um vetor de caracteres liderado por "0" e terminado em caractere `NULL`. Usa-se a função `ltoa` padrão da linguagem C para a conversão do número para o vetor de caracteres. Em seguida insere-se o caractere "0" no primeiro elemento e percorre-se o vetor deslocando todos os elemento em uma posição. É importante notar que o caractere "0" corresponde ao símbolo de código 48 na Tabela *ascii* (Figura 21), e não ao de código 0. O símbolo de código 0 é o caractere `NULL` inserido no final do vetor. O deslocamento de informações no vetor é apresentado a seguir:

```
1 ltoa(soma, dados.OP, 10);
2 char temp = dados.OP[0];
3 char anterior = dados.OP[0];
4 dados.OP[0] = '0';
5 for (byte i = 1; i < sizeArray; i++) {
6     temp = dados.OP[i];
7     dados.OP[i] = anterior;
8     anterior = temp;
9 }
```

Como mencionado anteriormente, os dados `operador` e `linha` são mutuamente exclusivos, e é responsabilidade da função `preencheDados` de inserir somente o dado preenchido pelo operador na estrutura. O dado que não for preenchido não será enviado para o *hub*. Caso algum dos dados inseridos pelo operador seja inválido então o processo de apontamento reinicia automaticamente, e caso sejam feitas 3 tentativas inválidas em sequência o *keypad* é recalibrado.

3.3.3.4 Algoritmo de detecção de produtos

O desenvolvimento e aperfeiçoamento do algoritmo de detecção de novos produtos tomou uma grande parte do tempo total de desenvolvimento. Como a detecção é baseada no peso sob a plataforma, em teoria basta que divida-se o peso total na plataforma pelo peso

unitário do produto. Esse método, porém, apresenta diversos tipos de problemas. Primeiramente há a questão de que pequenas incertezas e variações no peso de cada produto único geram grandes inconsistências quando pesados em lotes de centenas de unidades. Assim haveria não só a necessidade de que as referências de peso de cada produto fossem obtidas com uma precisão considerável, como também a certeza de que variações de peso fossem mantidas em um mínimo, algo que nem sempre é possível em linhas de produção manuais. Lotes de produtos não podem ser descartados por apresentarem algumas gramas a mais de material, e para a linha de injeção, onde cada produto pesa, em média, 100g, uma variação na faixa de poucas gramas resulta em uma medição exagerada do número de produtos, acusando mais do que a realidade. Além disso, seria necessário gerar uma referência de peso para todos os produtos da empresa, e atentar-se para que a mesma estivesse sempre atualizada. Como são produzidos centenas de produtos diferentes, ao total são mais de 800 tipos de cabos no banco de dados da empresa, esse processo torna-se inviável no longo prazo.

Com isso em mente foi utilizada uma abordagem diferente para a identificação de novos produtos. Como explicado na Seção 3.3.2.2, o módulo HX711 possui um tempo de assentamento entre medidas que pode ser calculado sabendo seus parâmetros de configuração. Para os parâmetros utilizados, também discutidos na mesma seção, o tempo de assentamento teórico é de 1.8 segundos, e com base em testes práticos, a exemplo a Figura 22, nota-se que corresponde satisfatoriamente com a realidade. Assim, pode-se usar esse período transitório como gatilho para identificação de um novo produto.

Com tempo de assentamento próximo de 2 segundos e taxa de amostras por segundo igual a 10, nota-se que mantendo em memória as 20 últimas amostras pode-se obter valores de equilíbrio antes e depois da variação. Sabe-se também que o valor retornado pelo módulo HX711 é um número do tipo `float`, uma vez que, como explicado na Seção 3.3.2.2, é resultado de uma divisão de dois valores, e assim contém casas decimais. Assim, há um vetor de variáveis do tipo `float` que a todo momento mantém um registro das últimas 20 amostras obtidas pela plataforma. Para determinar se há um novo produto compara-se a amostra atual à amostra mais antiga contida no vetor, de forma que se obtém a diferença absoluta entre ambas. Utiliza-se o valor absoluto pois o algoritmo deve ser capaz de detectar produtos tanto sendo adicionados quanto retirados da plataforma de pesagem, e comparar se uma diferença negativa é maior do que um valor positivo sempre resultará em falhas. A diferença absoluta entre as amostras é comparada ao fator de detecção, que corresponde à uma fração do peso unitário do produto. A obtenção do peso unitário é automática, e é abordada em detalhes na Seção 3.3.3.5. Essa fração foi ajustada experimentalmente, mas sabe-se que a mesma deve maior que a metade do peso unitário, afim de evitar que produtos com peso ligeiramente acima da média disparem o algoritmo duas vezes, e menor que o peso total unitário, afim que produtos ligeiramente abaixo do peso médio também possam disparar a detecção. O valor encontrado experimentalmente que

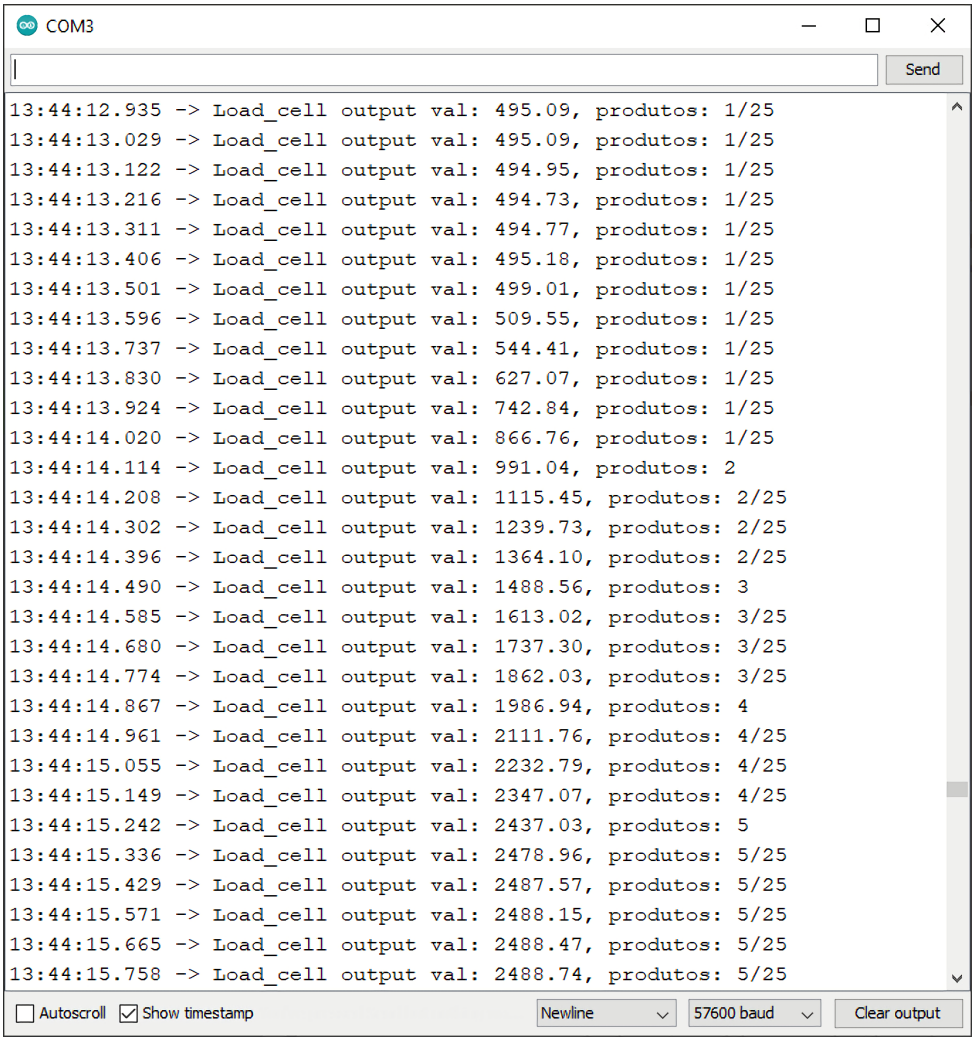

Figura 22 – Medição de peso com períodos permanentes e transitório. Fonte: Elaborada pelo autor.

corresponde à maior precisão para a detecção foi a fração de 80% do peso unitário. Caso note-se que a diferença absoluta entre a media atual e a medida mais antiga o algoritmo de detecção irá atualizar a *flag booleana* `novoProduto`, bem como a *flag* `maisMenos`, que indica se há adição ou subtração de produtos. Caso contrário, então atualiza-se a amostra mais antiga como sendo a medição atual e aguarda-se a próxima amostra enviada pelo módulo HX711. A Figura 23 exemplifica o funcionamento do algoritmo de detecção.

Após ser constatado que há um novo produto sobre a plataforma primeiramente o contador de produtos é incrementado e seu valor na estrutura de dados é atualizado. Esse valor é exibido ao operador no *display* LCD, juntamente com o peso total sobre a plataforma. A seguir é chamada uma função que preenche todo o vetor de amostras históricas com o valor da medida atual, afim de evitar que a próxima comparação resulte em falsos positivos. Assim, sabe-se que amostras futuras que sejam "80% do peso unitário" maiores que amostras históricas indicam novos produtos. Devido ao tempo de assentamento (1.8 segundos) e o número de amostras por segundo (10 *SPS*) o algoritmo de detecção consegue

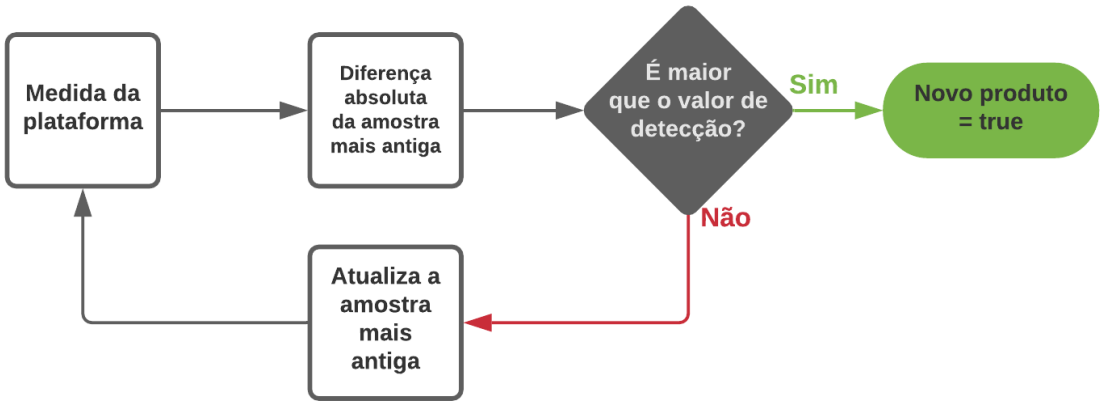

Figura 23 – Fluxograma do algoritmo de detecção. Fonte: Elaborada pelo autor.

identificar, em teoria, até 18 produtos novos sob a plataforma a cada regime transitório. Na prática, conforme discutido no Capítulo 5, a identificação de múltiplos produtos simultaneamente é bem abaixo do valor teórico esperado. Enfim, após atualizado o número de produtos é comparado ao número inserido pelo operador durante o processo de apontamento. Caso o número de produtos atual seja menor que o desejado então a função retorna a *flag* `novoProduto` para `false` e aguarda o envio da próxima medida. Caso o número de produtos tenha atingido o número desejado então o sistema notifica o operador através dos indicadores luminosos citados na Seção 3.3.2.5, para em seguida realizar os mesmos procedimentos mencionados anteriormente. A Figura 24 demonstra essa etapa do funcionamento do algoritmo de detecção.

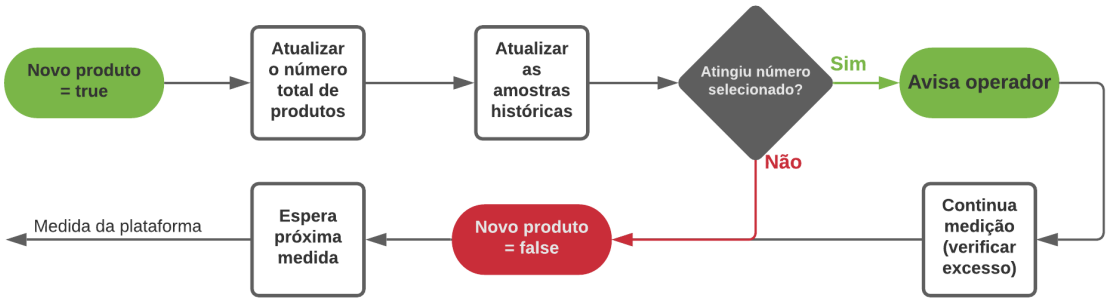

Figura 24 – Fluxograma do algoritmo de detecção após novo produto. Fonte: Elaborada pelo autor.

Vale notar que mesmo atingido o número limite o sistema continua realizando medições, e avisa o operador em caso de excessos. A medição só é interrompida, a qualquer ponto de sua execução, quando o operador pressiona o botão de *reset*. O evento de apertar o botão de *reset* não somente interrompe as medições do contador, como envia ao *hub* o

apontamento do número de produtos no instante do evento, e prepara o dispositivo para um *reset*. O envio de dados é abordado na Seção 3.3.3.6.

3.3.3.5 Medição automática de peso unitário

Como mencionado anteriormente, o fator de detecção de um novo produto sobre a plataforma é uma fração do peso unitário dos produtos sendo contados. Para isso é necessário que saiba-se o peso unitário. Porém, como dito na Seção 3.3.3.4, essa informação não deve requerer que o operador conheça com precisão o peso do produto, mas deve ser obtida automaticamente. Para tal, foi implementada uma função especial de medição de peso, de nome `medePrimeiroProduto`, e chamada após a calibragem da plataforma de pesagem.

A função de calibragem da plataforma de medição é chamada obrigatoriamente na primeira inicialização de cada estação, e pode ser chamada após cada *reset*, caso o operador assim deseje. Uma vez chamada, a função automaticamente realiza a tara da plataforma e, em seguida, pergunta ao operador se deve recalcular o fator de calibragem (*cal factor* e tara foram explicados na Seção 3.3.2.2), finalmente fazendo-o, ou não, para então encerrar sua execução.

Uma vez que a plataforma esteja calibrada, a função `medePrimeiroProduto` aguarda o envio de dados do módulo HX711, de maneira similar à função de medição. A detecção de novos produtos também é feita com base no mesmo procedimento de comparação da diferença entre amostras, porém com um valor de detecção diferente, fixo em 25 gramas. Esse valor de 25 gramas foi escolhido por ser o valor de precisão das células de carga utilizadas. Diferentemente da função de medição normal, aonde flutuações de peso ao longo do tempo e pequenas oscilações em torno do valor de peso correto (inerentes das células de carga, todos os dispositivos dessa categoria apresentam esse comportamento) são desprezíveis por serem lentas demais e não influenciarem a detecção de novos produtos, na obtenção de peso unitário é fundamental que o valor lido pela plataforma seja o mais próximo possível do peso unitário real do produto. Assim, caso não sejam corrigidas, tais erros podem deslocar a medição de peso unitário e resultar em um comportamento indesejado durante o resto da execução do programa. A correção de erros durante o período de medição do primeiro produto é realizada através do cálculo de um "ruído médio" na medição, a partir de amostras históricas. Esse valor é simplesmente a média das últimas 20 amostras obtidas da plataforma, e é descontado do valor de peso unitário final. É recalculado a cada 5 amostras, e caso atinja um valor absoluto superior a 25 gramas sem que haja a detecção de um novo produto a função de calibragem é chamada automaticamente e a balança é recalibrada. Uma vez que seja detectado um produto sobre a plataforma, a medição do ruído médio cessa de ser atualizada e espera-se até a estabilização do peso sobre a plataforma para que seja feito o cálculo do peso unitário. Após realizado o cálculo do peso unitário e subtraído o valor do ruído médio então atualizam-se

os valores das variáveis globais `pesoProduto` e `valorDeteccao`, o número de produtos é incrementado, e a função `medePrimeiroProduto` termina sua execução, dando lugar à função de medição normal que roda em *loop*. Vale notar que os valores de peso unitário e detecção podem ser recalculados, basta que o valor de peso atinja um regime permanente e encontre-se estável. Para ser considerada em regime permanente, a plataforma deve medir uma diferença absoluta entre valor atual e amostra mais antiga menor do que 10% do peso unitário durante, pelo menos, 10 amostras (1 segundo). O peso total é então dividido pelo número de produtos sobre a plataforma e, caso esse valor não difira em mais de 5% do peso unitário atual, atualizam-se o peso unitário e o valor de detecção. A Figura 25 demonstra o funcionamento dessa função.

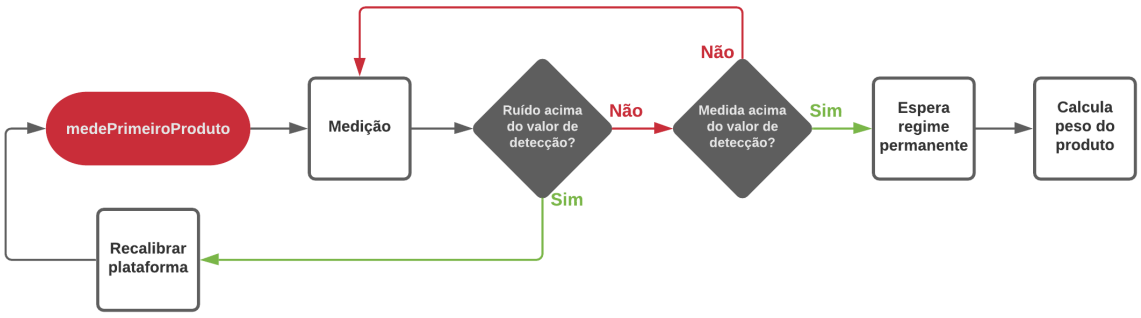

Figura 25 – Fluxograma da função `medePrimeiroProduto`. Fonte: Elaborada pelo autor.

3.3.3.6 Funções para envio de informações na rede sem fio

O envio de informações na rede sem fio é realizado através do módulo nRF24L01+. A rede é abordada em detalhes no Capítulo 4, enquanto nessa seção são abordadas as funções implementadas para envio, recebimento, e verificação de pacotes nela enviados por parte dos Nós, listadas na Tabela 9.

Tabela 9 – Funções para envio, recebimento, e verificação de pacotes na rede sem fio

Função	Parâmetros	Retorno	Observações
`encodePacket`	`unsigned char*, Data*, bool`	`void`	
`sendMessage`	`Data`	`bool`	a
`sendMessage`	`uint8_t`	`bool`	a
`sendMessage`	`bool`	`bool`	a
`checkSend`	`bool`	`bool`	
`checkReceive`	`uint32_t`	`bool`	

a. A declaração de duas funções com o mesmo nome mas diferentes parâmetros denota o comportamento de *overloading*. *Overloading* é muito comum quando deseja-se comportamentos similares porém ligeiramente diferentes dado certas condições. Nesse caso deseja-se enviar pacotes diferentes, identificados por diferentes tipos de *header*, aonde o tipo de *header* é baseado no tipo de dado enviado.

O envio de estruturas de dados entre diferentes microcontroladores não é tão direto quanto simplesmente enviar pacotes contendo um único dado. Especialmente para microcontroladores com diferentes tamanhos de barramentos de dados, como os utilizados pelo projeto (8 bits para o Arduino e 32 bits para o ESP8266). Estruturas de dados na linguagem C possuem uma propriedade chamada *padding*, responsável por realizar o alinhamento de dados com o intuito de minimizar o número de ciclos de CPU para leitura de variáveis. A quantidade de *padding* depende do tamanho de palavra lida por cada processador, o que por sua vez depende da arquitetura do mesmo. Para processadores de 32 bits, como o processador L106 Diamond, a partir do qual o processador do ESP8266 é baseado, o tamanho de palavra é de 4 bytes. Assim, realiza-se *padding* para garantir que todas as variáveis estejam alinhadas na memória quando precisem ser acessadas. Devido ao uso de um vetor de caracteres com tamanho diferente de uma potência de 2 (variáveis `char` ocupam 1 byte de espaço, e assim quando agrupadas em múltiplos de 4 não necessitam de *padding* para processadores de até 32 bits) nota-se que haverá *padding* no ESP8266, algo que não acontece no Arduino devido à seu processador de 8 bits com tamanho de palavra também de 8 bits. A solução para esse problema é codificar a estrutura enviada em um vetor, contendo todos os dados sequencialmente, e enviá-lo no lugar de enviar a própria estrutura de dados. Para isso a função `encodePacket` é chamada quando deseja-se enviar uma estrutura de dados através da rede RF.

A função `encodePacket` recebe como parâmetros o endereço de um vetor de caracteres, o endereço da estrutura de dados a ser codificada, e uma *flag* do tipo `bool` indicando qual dado, entre número da linha e código do operador, deve ser copiado para o vetor. Conforme apresentado a seguir, a função utiliza a função da linguagem C `memcpy` para copiar um número "i" de bytes de um endereço de memória à outro:

```
1  void encodePacket(unsigned char* vetor, struct Data* dados, bool linha) {
2      size_t i = 0;
3      memcpy(&vetor[i], &dados->OP, sizeof(dados->OP));
4      i += sizeof(dados->OP);
5
6      ...
7
8      if (linha) {
9          memcpy(&vetor[i], &dados->linha, sizeof(dados->linha));
10         i += sizeof(dados->linha);
11     } else {
12         memcpy(&vetor[i], &dados->operador, sizeof(dados->operador));
13         i += sizeof(dados->operador);
14     }
```

Uma vez que a estrutura esteja codificada no vetor, então a mesma pode ser enviada para o *hub*. O envio de pacotes ao servidor fica à cargo da função `sendMessage`. Essa função é *overloaded*, pois deseja-se que trate do envio não somente de estruturas mas

também do pacote de requisição de ID único, que não requer codificação por ser somente um dado. O primeiro passo para o envio de pacotes é realizar a verificação da conexão através da função `conectionCheck`, cujo funcionamento é explicado na Seção 3.3.3.7, pois se o Nó encontra-se sem conexão à rede o envio de dados obviamente não será bem sucedido. Uma vez que a conexão esteja funcionando então é determinado o tipo de *header* utilizado para o pacote enviado. Os tipos de *header* utilizados pelo contador estão disponíveis na Tabela 10. Caso a mensagem envie dados de contagem e apontamento então é criado um vetor para envio e chamada a função `encodePacket` para preenchimento do vetor. Em seguida, independente do tipo de variável enviado, a função monta o pacote e o envia, retornando `true` caso o envio seja bem sucedido (recebe *ACK* do módulo que recebeu o pacote, o *auto acknowledgement* de pacotes é explicado na Seção 4.3, e `false` caso contrário.

Tabela 10 – Tipos de *header* recebidos e enviados pelo contador

Header type	Tipo de dado	Observações
`M`	`Dados`	a
`L`	`Dados`	a
`N`	`uint8_t`	b
`d`	`bool`	c
`t`	`uint8_t`	d

a. *Headers* tipo `M` e `L` são utilizados para envio de dados de apontamento. Como mencionado na Seção 3.3.3.2, o contador pode enviar uma estrutura contendo código do operador ou linha, e como a formatação de ambos é diferente para envio ao servidor então existem *headers* separados. Um *header* `M` indica que a estrutura enviada contém código de operador, enquanto um *header* `L` indica a presença do número da linha.
b. Pacotes com *header* `N` são enviados durante a configuração do ID único, e seu conteúdo não é importante.
c. Pacotes com *header* `d` são utilizados para verificação de conexão. Ao receber um pacote tipo `d` o Nó responde com um pacote, também de tipo `d` contendo uma variável `bool` para confirmar que continua conectado à rede.
d. O *header* `t` é exclusivo para recebimento. Pacotes recebidos com *headers* `t` são pacotes confirmando que os dados enviados foram entregues ao servidor.

Uma vez que as mensagens tenham sido enviadas, a função `checkSend` verifica o código de retorno gerado pela função `sendMessage`. Caso o envio tenha sido bem sucedido então a única operação realizada é a notificação do operador de que o envio foi bem sucedido e a função termina sua execução retornando `true`. Caso contrário então tenta-se diagnosticar o motivo da falha no envio. Primeiramente testa-se a conexão à rede. Caso o Nó esteja conectado à rede então o operador é avisado que a mensagem falhou mas que o Nó continua conectado, e a função termina sua execução retornando `false`. Caso contrário então tenta-se renovar o endereço do Nó dentro da rede. Uma renovação bem sucedida notifica o operador de que o Nó moveu-se dentro da rede, exibindo à ele o novo endereço do mesmo, e a função termina sua execução retornando `false`. Caso a renovação de

endereço não tenha sucesso então tenta-se iniciar uma nova conexão à rede e a função termina sua execução retornando `false`.

Há pacotes aonde deseja-se uma confirmação de envio bem sucedido mais completa. Como é explicado na Seção 4.3, o reconhecimento de pacotes recebidos é realizado pelo próprio módulo à nível de hardware e apenas trata do primeiro envio, não havendo a confirmação concreta de que o pacote chegou à seu destino. Para aguardar a confirmação de entrega de pacotes, principalmente quando enviando dados ao *hub* que devem ser encaminhados ao servidor, foi criada a função `checkReceive`. A função recebe como parâmetro um intervalo durante o qual o Nó aguardará um pacote de retorno contendo o *header* `t` e um número de confirmação de entrega. A inclusão do número de confirmação é apenas uma maneira de preparar o sistema para possíveis atualizações e melhorias no futuro, aonde o número recebido seria único para cada transação. Por enquanto esse número é fixo, tanto para Nós quanto para o *hub*. O pacote com *header* `t` é enviado ao Nó somente após a confirmação de recebimento de pacote UDP por parte do servidor, assim ao recebê-lo a função notifica o operador que seus dados foram recebidos com sucesso e termina sua execução retornando `true`. Caso o pacote recebido do *hub* contenha outro tipo de dado então o mesmo é ignorado, e uma vez que o intervalo informado como parâmetro seja esgotado a função notifica ao operador que houve falha no envio de dados ao servidor e termina sua execução retornando `false`.

Vale notar que uma mensagem de erro de envio ao servidor pode ser causada por uma confirmação entre *hub* e Nó interrompida, como é explicado na Seção 4.4. Nesse caso os dados já constam no servidor, e operadores são instruídos a notificar supervisores caso encontrem mensagens de erro para que os mesmos verifiquem se os dados foram inseridos e/ou os corrijam caso necessário.

3.3.3.7 Outras funções implementadas

Como explicado na Seção 3.3.3, comportamentos repetitivos devem ser simplificados através de funções. As funções que de alguma maneira interagem com o operador, ou complementam o funcionamento do sistema, e não foram abordadas anteriormente estão contidas na Tabela 11.

Tabela 11 – Outras funções implementadas no contador

Função	Parâmetro	Retorno	Observações
`printMsg`	`String`	`void`	a
`printMsg`	`String`, `String`	`void`	a
`printLCD`	`float`, `bool`	`void`	
`acendeLed`	`int`	`void`	
`connectionCheck`	Nenhum	`void`	
`configMesh`	Nenhum	`void`	

a. Novamente demonstrado *overloading* de funções.

As funções `printMsg` são chamadas para imprimir mensagens no *display* LCD. Cada parâmetro `String` passado para a função é impresso em uma linha, assim o motivo para utilização de *overloading* é simplesmente para a utilização da segunda linha do *display*. Quando a função `printMsg(String, String)` é chamada, a mesma chama a função `printMsg(String)` para realizar a impressão da primeira linha, ao invés de reutilizar o código, e somente realiza a impressão da segunda linha antes de terminar sua execução.

Apesar da função `printLCD` também imprimir dados no *display* LCD, a natureza dos dados requer uma implementação diferente. Essa função é atualizada a cada nova amostra recebida do módulo HX711, e imprime no *display* o peso registrado e o número de produtos, tanto atuais quando o total desejado. A variável `float` passada como parâmetro é a amostra recebida do amplificador, que por ser uma variável local da função `loop()` não pode ser acessada por outras funções, enquanto a variável `bool` significa alteração no número de produtos, para qual o monitor irá primeiro limpar todos os dados exibidos para, então, atualizar a nova pesagem e o novo número de produtos. Como o número de produtos é uma variável global então não há a necessidade de passá-la como parâmetro, visto que todas as funções podem acessá-la.

A função `acendeLed` recebe como parâmetro um número correspondente à combinação do estado das portas de entrada do demux necessária para acender o LED indicado. A combinação de entradas e LEDs acesos pode ser vista na Figura 19. Para simplificar o funcionamento dessa função foram declaradas variáveis globais `ledVerde`, `ledAmarelo`, e `ledVermelho`, correspondentes aos números utilizados pela função para selecionar a combinação desejada. O número à elas associado é aleatório e não influencia o funcionamento do código, tanto que para duas variáveis o valor até sofre *overflow*, importando apenas que seja único para todas. A implementação da função é apresentada a seguir:

```
1  const byte ledAmarelo = 150;
2  const byte ledVermelho = 292;
3  const byte ledVerde = 347;
4
5  ...
6
7  void acendeLed(int pin) {
8      apagaLeds();
9      switch (pin) {
10         case ledVermelho:
11             digitalWrite(demuxA, HIGH);
12             break;
13         case ledAmarelo:
14             digitalWrite(demuxB, HIGH);
15             break;
16         case ledVerde:
17             digitalWrite(demuxA, HIGH);
18             digitalWrite(demuxB, HIGH);
```

```
19                 break;
20             }
21 }
```

As funções `connectionCheck` e `configMesh` adicionam funcionalidades extras à rede. A primeira faz uso do reconhecimento automático de pacotes implementado pelas bibliotecas RF24 (mais detalhes na Seção 4.3) para realizar periodicamente a verificação da conexão por parte dos clientes. São realizados dois testes e três tentativas de reconexão na rede, com a primeira falha já avisando o operador por meio do desligamento do LED de conexão Wi-Fi. Apesar de ser nomeado LED Wi-Fi, esse LED não indica conexão Wi-Fi, mas sim conexão à rede de RF. Primeiro usa-se o método `checkConnection()` da biblioteca `RF24Mesh`. Esse método tenta enviar para o Nó mestre da rede (*hub* de comunicações) uma requisição para saber o endereço do Nó remetente na rede, fornecendo seu ID para a consulta. Caso o pacote envie com sucesso é iniciado um *timer* para seu retorno. Um retorno bem sucedido irá fazer a função retornar o valor do endereço, o que indicará uma conexão bem sucedida à rede. Falha no pacote de retorno do *hub* (*timer* atinge o limite, *timeout*) ou falha no envio da requisição ao *hub* fazem a função retornar -1, o que indica falha na conexão. Vale notar que a função também irá indicar erro caso utilizada pelo Nó mestre, pois um retorno de endereço igual à 0 (endereço do Nó mestre) dispara a condição de erro de conexão da função. Caso o primeiro teste indique falha de conexão então o Nó realiza o segundo teste de conexão, na forma de uma tentativa de renovação de seu endereço na rede, através da função `renewAddress()`, também da biblioteca `RF24Mesh`. A função `renewAddress()` estabelece uma nova conexão expressa com a rede, utilizando um endereço padrão (0, endereço do Nó mestre). Tentar conectar na rede com endereço 0 irá gerar uma requisição ao Nó mestre por um novo endereço. A função então irá retornar o valor de seu novo endereço. Caso a conexão não tenha sido estabelecida a função retorna 0, o endereço padrão. Essa função não requisita um novo identificador ao Nó mestre. Caso esse teste também falhe, significando que as tentativas anteriores de consulta de endereço e renovação de endereço também falharam, é realizada uma última tentativa de conexão. Essa tentativa de conexão, que é o mesmo método utilizado por Nós que tentam se conectar na rede pela primeira vez, é a função `configMesh`.

A função `configMesh` foi implementada para que não houvesse a necessidade de configurar cada Nó para possuir um identificador único, e ao invés simplesmente deixar o Nó conectar-se à rede com um identificador padrão para, então, receber um novo identificador do Nó mestre. O primeiro passo é configurar o identificador padrão do Nó para requisição (identificador 1) e solicitar a conexão expressa na rede, de maneira similar à função `renewAddress()`. Essa conexão expressa serve para adquirir um novo endereço na rede, e a distribuição automática de endereços é explicada em mais detalhes na Seção 4.3. Após ter conseguido um novo endereço o Nó irá enviar ao *hub* um pacote do tipo `"N"` e iniciar um *timer* para verificação. O conteúdo desse pacote não é importante, e não é

processado pelo *hub*. A maneira utilizada pelo *hub* para distribuição de identificadores é abordada em mais detalhes na Seção 4.3. Após encontrar um novo identificador o *hub* o envia ao Nó. Ao receber seu novo identificador único o Nó irá liberar seu endereço atual e iniciar uma nova conexão, já com seu novo identificador. A liberação de endereço é uma função também pertencente à biblioteca `RF24Mesh`, chamada `releaseAddress()`, e simplesmente envia ao Nó mestre um pacote vazio de tipo "197". O Nó mestre, ao receber um pacote desse tipo, irá marcar o endereço do Nó remetente como liberado e enviar uma última mensagem de confirmação ao Nó, que, ao recebê-la, cessará comunicação com a rede.

Uma vez que a nova conexão seja aceita então a configuração de rede está finalizada, o LED indicador de conexão Wi-Fi é aceso, e o programa pode iniciar ou resumir sua execução.

3.3.4 Construção do protótipo

Cada componente adicionado ao projeto aumenta sua complexidade e o número de tarefas a ser realizadas pelo microcontrolador. Todos os componentes passaram por baterias de testes iniciais antes de serem integrados aos demais, para verificar seu funcionamento e validar que poderiam funcionar em conjunto. Tanto os testes de componentes quanto as primeiras etapas de integração dos mesmos e validação do software foram realizadas em *breadboards* (também chamadas de *protoboards*, ou placas de prototipagem/ensaio), por possuírem maior facilidade no arranjo de componentes. Como o circuito não lida de maneira crítica com sinais analógicos de alta velocidade ou demais componentes muito sensíveis à ruídos e interferências então não há restrições severas no uso de *breadboards* para prototipagem. A Figura 26 demonstra a montagem do circuito em uma *breadboard*, enquanto a Figura 27 demonstra a plataforma de medição de peso conectada à ele.

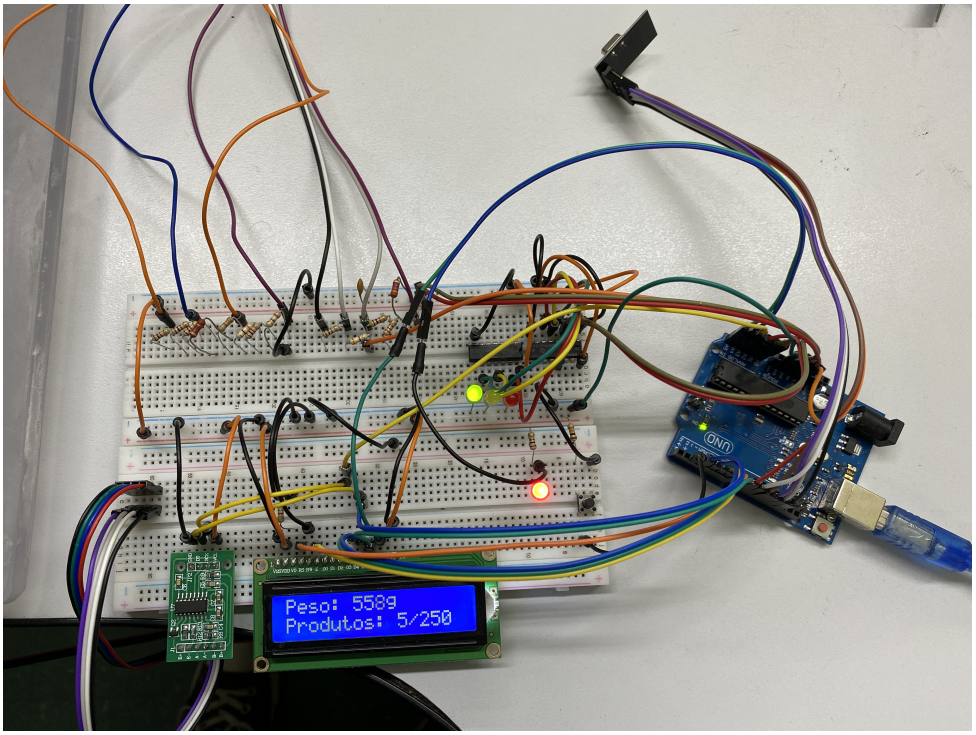

Figura 26 – Circuito montado em uma *breadboard*. Fonte: Elaborada pelo autor.

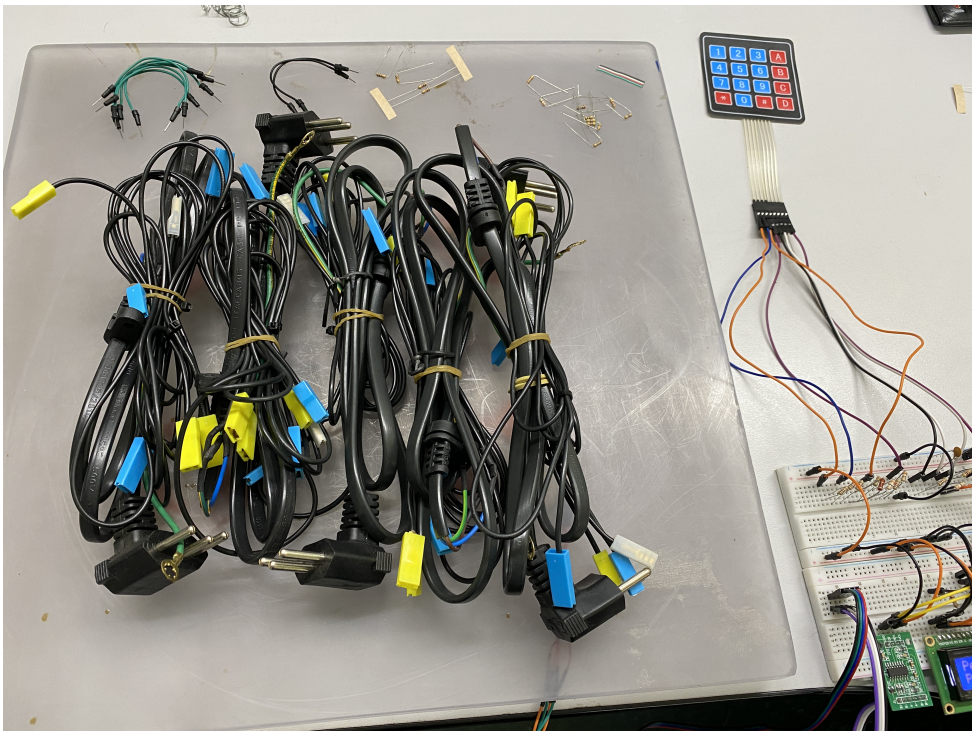

Figura 27 – Plataforma de medição de peso. Fonte: Elaborada pelo autor.

3.3.4.1 Design de PCB

Apesar do uso de *breadboards* não demonstrar empecilhos para a integração inicial dos componentes e validação de seu uso, para um produto final é desejado o design de

uma placa de circuito impressa (do inglês *printed circuit board*, PCB), capaz de integrar os componentes de forma mais organizada, profissional, e funcional, reduzindo ruídos provenientes do uso de *jumpers* e placas sem solda. Algumas considerações feitas pelo autor quanto ao design da PCB são:

1. Deseja-se usar a placa de maneira similar a um *shield*, de maneira que a mesma esteja encaixada sobre o microcontrolador.

2. Deve-se prestar atenção quanto ao posicionamento do *display* LCD, de forma que nenhum outro componente obstrua ou atrapalhe a leitura dos dados ali exibidos.

3. Os LEDs de sinalização devem estar próximos ao *display* LCD, para facilitar a observação de avisos por parte do operador.

4. O componente HX711 deve-se estar localizado de forma que suas conexões com as células de carga se encontrem em alguma extremidade da placa, com intuito de facilitar a conexão e desconexão da plataforma de pesagem com o sistema.

5. O componente nRF24L01, responsável pela comunicação sem fio, também deve-se estar localizado de forma que sua antena embutida esteja em uma extremidade da placa. Caso note-se que é necessário o uso de uma antena externa então deve-se planejar tanto o design da placa quanto da *case* que guardará todos os componentes de forma a permitir este encaixe. Além disso é necessário que os capacitores de *decoupling* encontrem-se o mais próximo possível dos pinos de alimentação do componente.

Tendo em mente essas considerações, o design final da placa de circuito impresso é demonstrado nas Figuras 28 e 29.

3.3.4.2 Design e impressão 3D de *case*

Uma vez que o design da PCB esteja finalizado então a última etapa no desenvolvimento de um produto final é o design de um invólucro, que, ao longo desta seção, será chamado de *case*. Um bom design para a *case* além de oferecer uma aparência mais profissional ao produto final também auxilia na proteção contra partículas externas, como poeira e demais resíduos, e choques físicos contra os delicados componentes elétricos do projeto. Antes de iniciar o processo de design foram feitas algumas considerações pelo autor:

1. A *case* deve manter os componentes protegidos quando fechada, incluindo contra choques entre os mesmos e a própria *case*. Para isso todos devem estar bem posicionados e fixados. Ainda assim, deve ser possível removê-los da *case* para eventual manutenção ou troca.

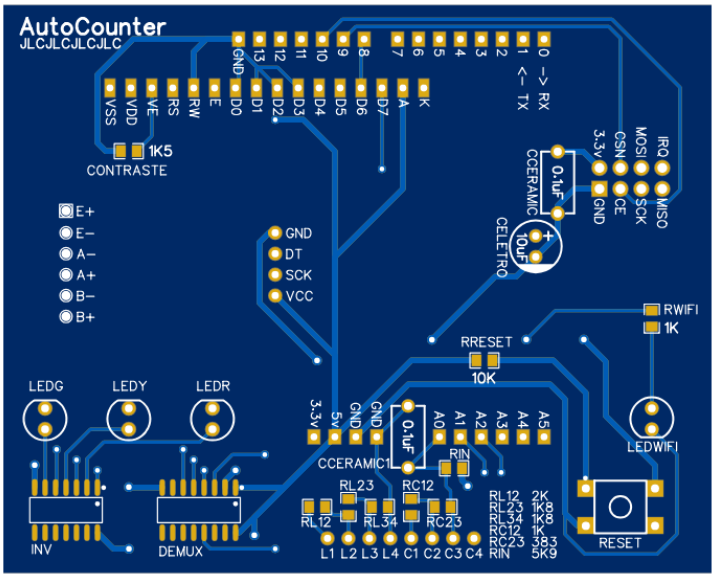

Figura 28 – Vista frontal da PCB. Fonte: Elaborada pelo autor.

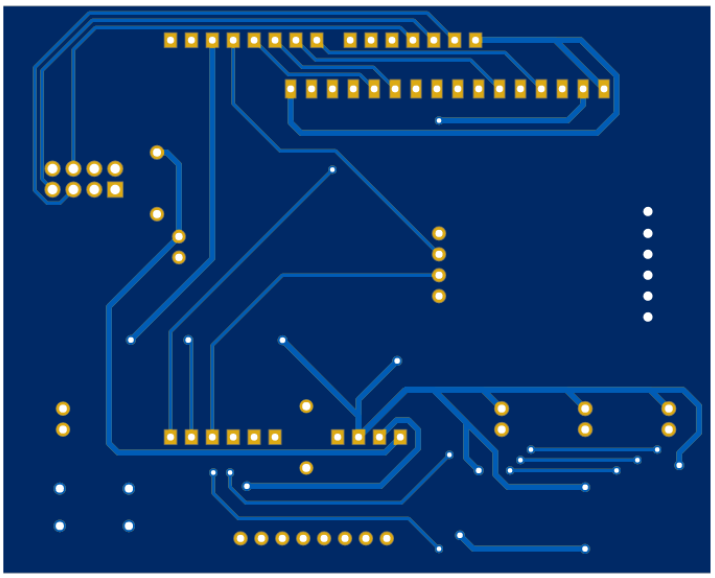

Figura 29 – Vista traseira da PCB. Fonte: Elaborada pelo autor.

2. Deve-se atentar para conexões com dispositivos externos (por exemplo células de carga), bem como conexões usb e de energia do microcontrolador.

3. Com o intuito de minimização de custos a *case* deverá utilizar o mínimo de material possível.

4. O projeto deve ser capaz de estabelecer comunicação sem fio com outros dispositivos, assim a *case* deverá ser transparente para ondas de radiofrequência, ou seja, permissível à passagem de sinais com uma absorção mínima de energia.

5. Deve ser de fácil montagem e desmontagem, permitindo acesso ao sistema, caso necessário.

Uma vez enumerados os objetivos então tem início o processo de design. Primeiramente são obtidas as dimensões físicas do microcontrolador, PCB, e demais componentes. Em seguida é pensada na fixação dos componentes dentro da *case*. Como deseja-se que os mesmos possam ser removidos, e então é vedado o uso de adesivos ou similares, optou-se por utilizar os furos presentes na placa do microcontrolador como encaixes para suportes afim de evitar o deslocamento horizontal do mesmo. Pode-se também dimensionar a altura da *case* para evitar o deslocamento vertical do protótipo, fixando-o assim de maneira removível dentro da *case*. O design da *case* foi feito pensando em uma futura impressão 3D da mesma, atentando-se para artefatos como geometria e normais. O design final da *case* pode ser visualizado nas Figuras 30 à 33.

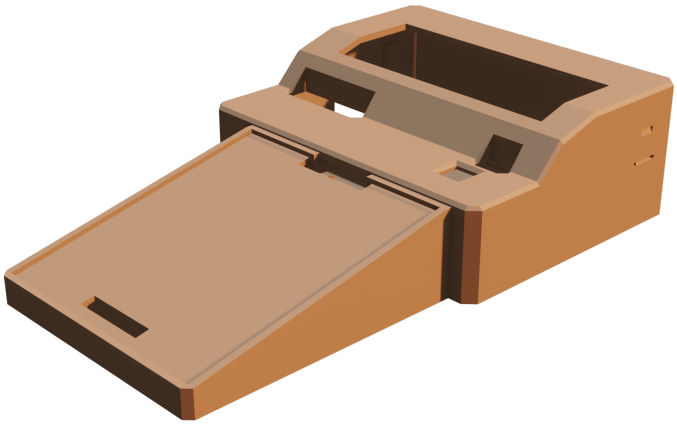

Figura 30 – *Case*, vista frontal. Fonte: Elaborada pelo autor.

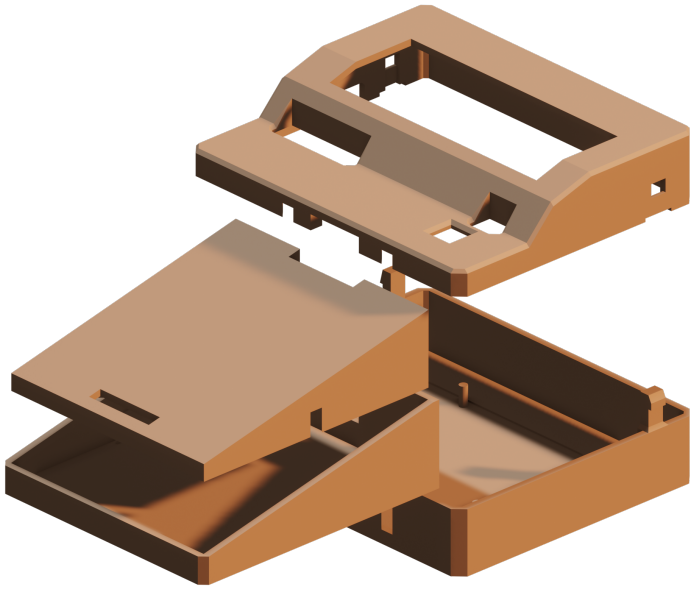

Figura 31 – *Case*, vista frontal explodida. Fonte: Elaborada pelo autor.

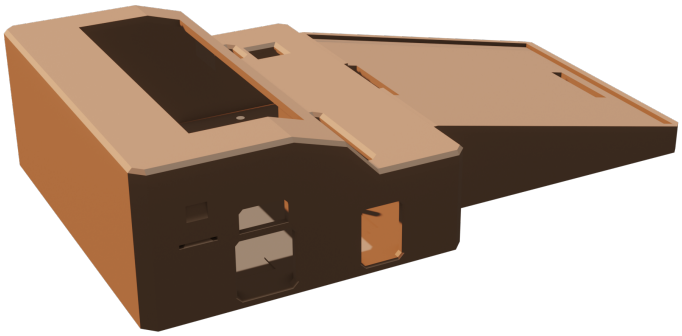

Figura 32 – *Case*, vista traseira. Fonte: Elaborada pelo autor.

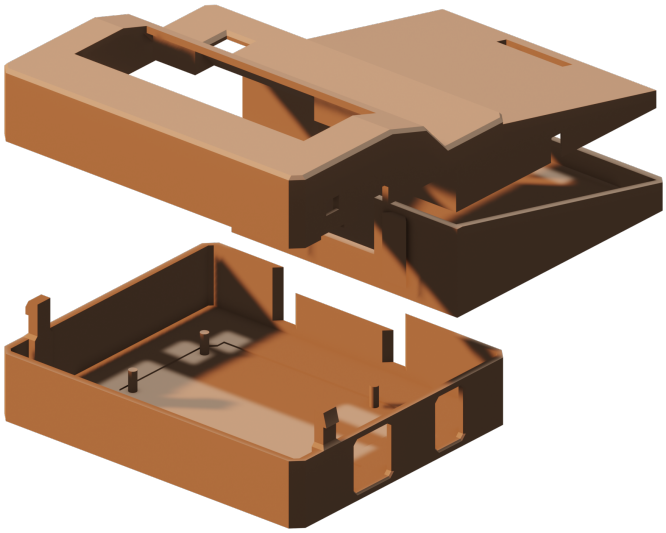

Figura 33 – *Case*, vista traseira explodida. Fonte: Elaborada pelo autor.

4 Sistema de comunicação sem fio

Apesar de problemas com confiabilidade dos dados enviados, interferências, velocidades e conexões inconsistentes, é inegável que o envio de dados em uma rede sem fio traz muito mais mobilidade e praticidade à um sistema. Com o envio de pacotes pequenos e métodos para verificação de recebimento, a rede sem fio implementada pelo projeto não é afetada pelos problemas ditos acima enquanto mantendo a mobilidade de Nós dentro do ambiente industrial sem a preocupação de estar atrelado à cabos. Ambientes industriais são ruidosos e longe de ideais para envio de dados. Além de contar com diversas redes Wi-Fi, o ambiente industrial conta com diversas fontes de ruído eletromagnético que podem influenciar não só a transmissão de dados como o próprio circuito do transmissor ou receptor. Há também os diversos obstáculos físicos que reduzem o alcance e confiabilidade dos sinais de rádio, algo que seria crítico caso o sistema fosse empregado em longas distâncias e que poderia acarretar na necessidade de uso de repetidores de sinal. Porém, com a habilidade de reenvio de pacotes dentro da rede, operação ligeiramente fora da faixa de frequência delimitada para redes Wi-Fi, e envio de pacotes pequenos com protocolos de reconhecimento de envio, esses problemas tem baixa, se alguma, influência no funcionamento da rede de comunicação sem fio.

Ainda assim, foram realizados diversos testes para determinar o alcance máximo da rede. No ambiente industrial, com diversos tipos de obstáculos e interferências, o alcance máximo encontrado foi de aproximadamente 16,80 metros, enquanto para o mesmo ambiente sem obstáculos os testes foram realizados até 30 metros sem quaisquer problemas. Assim assume-se que sob condições normais o alcance máximo varia entre 16,8 e 30,0 metros. Tais medidas são abaixo das especificações do fabricante, segundo o *datasheet* do módulo nRF24L01+ disponível na Seção 3.3.1, porém para o presente projeto são satisfatórias, pois, devido à estrutura da rede, mensagens para Nós remotos podem ser reenviadas por outros Nós até encontrá-los, e sua distribuição física dentro da empresa não irá conter distâncias maiores que 16,8 metros entre Nós.

Na parte da implementação do código foram utilizadas bibliotecas que abstraem controle de hardware do módulo de RF. Além disso, as bibliotecas facilitam a implementação da rede *mesh*, implementando métodos para endereçamento automático e roteamento de pacotes. A requisição e distribuição de identificadores automaticamente, bem como verificações de conexão por parte dos Nós, e liberação de identificadores e endereços sem conexão por parte do *hub* foram algumas das implementações extras realizadas pelo autor.

Como citado na Seção 3.3.3, Tabela 6, as bibliotecas utilizadas para a construção da rede estão disponíveis sob a licença GPL.

4.1 Rede de comunicação

Há diversas maneiras de se construir uma rede de comunicações via RF, porém os passos a serem seguidos são quase sempre os mesmos. Primeiramente deve-se pensar na topologia da rede. A topologia de uma rede é essencialmente sua estrutura, como serão feitos os caminhos de comunicação entre cada dispositivo. A rede segue a topologia de árvore, apesar de ser uma rede *mesh*, e terá sua topologia abordada em mais detalhes na Seção 4.2. Em seguida deve-se considerar as diferentes maneiras de endereçamento dos dispositivos, afinal todo pacote enviado na rede deve conter o endereço de destino do mesmo. O endereçamento de dispositivos pode ser feito de duas maneiras, endereços estáticos ou endereços dinâmicos. Endereços estáticos são utilizados quando Nós não se movem fisicamente dentro da área de cobertura da rede, são configurados manualmente e qualquer alteração na rede ou nos dispositivos requer alterações manuais. Endereços dinâmicos são fornecidos automaticamente pelo Nó mestre, e podem ser redistribuídos quando liberados sem a necessidade de ajustes manuais. Endereçamento dinâmico é especialmente interessante quando há a necessidade de minimizar o gasto de bateria nos dispositivos de campo, por permitir que os mesmos liberem seus endereços na rede e desativem os módulos de comunicação enquanto não há necessidade de transmissão de dados, permitindo que novas conexões tomem seus lugares automaticamente. Uma vez que se conectem à rede novamente irão receber um novo endereço do Nó mestre e poderão, então, enviar os dados normalmente. A rede desenvolvida utiliza endereçamento estático para o Nó mestre (endereço 0) e Nós que estejam estabelecendo conexão à rede (endereço 1), porém uma vez que estejam conectados então recebem um endereço dinâmico distribuído pelo Nó mestre, junto com um identificador único, também dinâmico. A distribuição de endereços e identificadores é abordada na Seção 4.3.

Outra consideração importante é sobre os pacotes transmitidos pela rede e suas limitações, tanto em software quando hardware. Segundo seu *datasheet*, os módulos de rádio nRF24L01+ possuem a limitação de 32 bytes como tamanho máximo de pacotes enviados, e 8 destes bytes são ocupados pelo *header* do pacote, como demonstra a Figura 34. Para o sistema desenvolvido essa limitação não é problemática, pois a estrutura de dados enviada, explicada na Seção 3.3.3.2, ocupa um máximo de 18 bytes (até 17 bytes de dados e um byte extra de *padding* caso seja ímpar), porém em casos onde há interesse em enviar pacotes maiores que o limite deve-se trabalhar com repartição de pacotes, e em adequadamente juntá-los quando entregues ao destino. Apesar de não terem sido utilizados, as bibliotecas `RF24Network` e `RF24Mesh` possuem métodos para implementação de fragmentação de pacotes.

Na Figura 34, `from_node` é o endereço do Nó remetente, no formato octal, `to_node` é o endereço do Nó de destino, também no formato octal, `id` é o identificador do pacote, incrementado toda vez que o Nó envia um novo pacote, `type` indica o tipo do *header*, e

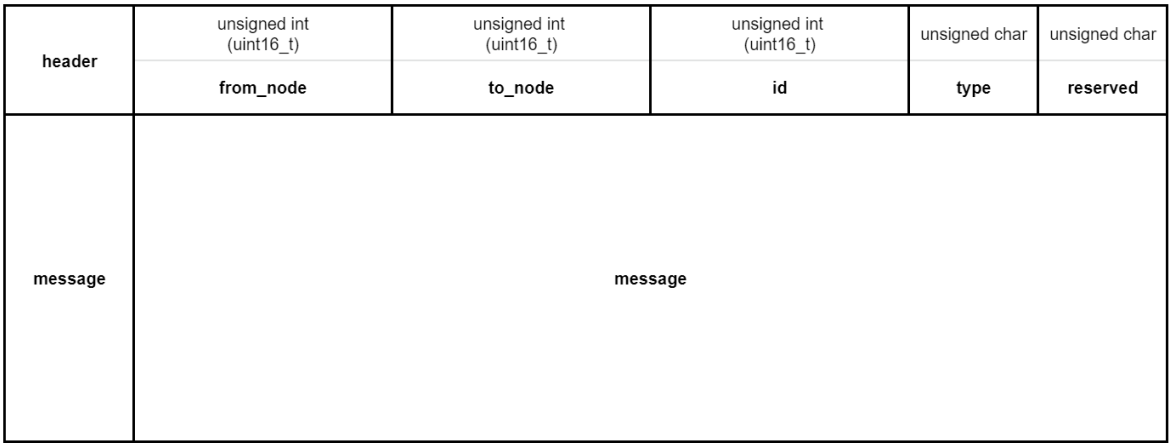

Figura 34 – Datagrama de pacotes enviados na rede RF. Fonte: Elaborada pelo autor.

`reserved` é para uso interno da biblioteca em pacotes como *ACK* (*acknowledgement*, ou confirmação) do recebimento de mensagens, por exemplo. O tipo de *header* é uma variável `unsigned char`, semelhante ao tipo `byte`, que serve para indicar ao Nó de destino o tipo de variável sendo enviada. Assim, o Nó de destino pode realizar diferentes tipos de ações de tratamento de dados dependentes do tipo de variável recebida antes mesmo de ler o conteúdo do pacote. Da mesma maneira, pode-ser usar *overloading* de funções de envio de dados para produzir pacotes utilizando os mesmos procedimentos mas tipos de dados, e consequentemente *headers*, diferentes.

4.2 Topologia de rede

A topologia de uma rede é influenciada principalmente por limitações físicas dos módulos de comunicação, localização física dos Nós dentro da rede, mobilidade dos Nós, e caminhos percorridos pelos pacotes. Os módulos nRF24L01 possuem seis *pipes* de comunicação, permitindo a comunicação simultânea com até 6 outros módulos. Essa limitação não diz que a comunicação com mais módulos é impossível, entretanto caso seja feita a tentativa com todos os canais ocupados o pacote será perdido. Com isso em mente foi utilizada uma topologia de rede do tipo árvore, onde cada Nó possui um mestre (exceto o Nó mestre, neste caso o *hub* de comunicações) e até 5 filhos, ou escravos. Pacotes enviados na rede *mesh* podem passar por vários Nós antes de chegarem à seu destino e todos os Nós estão configurados pela biblioteca `RF24Mesh` a reenviarem pacotes que não sejam destinados à seu endereço. O módulo nRF24L01+ implementa à nível de hardware a função de confirmação de recepção de pacotes. Essa confirmação, porém, é limitada ao primeiro Nó que recebe seu pacote, que não necessariamente é o Nó de destino.

Devido ao endereçamento e identificação automáticos caso um Nó encontre problemas de funcionamento e se desconecte da rede antes de liberar seu endereço e identificador,

seu lugar será automaticamente liberado e eventualmente tomado por outro dispositivo que conectar-se à rede. Caso nenhum dispositivo se conecte e o Nó falhado esteja no caminho de algum pacote enviado então o pacote falhará, gerando uma mensagem de erro no Nó remetente que irá, automaticamente, tentar assumir o lugar do Nó falhado na rede. Essa rotina de reconexão é explicada em mais detalhes nas Seções 3.3.3.6 e 3.3.3.7, e a distribuição de novos endereços na Seção 4.3.

Apesar de utilizar uma biblioteca chamada `RF24Mesh` e ser referida por rede *mesh* múltiplas vezes, a topologia da rede é na forma de árvore, e não *mesh*, como demonstra a Figura 35. A diferença entre redes com topologia de árvore e topologia *mesh* é na maneira como conexões entre os Nós são estabelecidas. Redes com topologia tipo árvore possuem hierarquia. Essas redes partem do nível mais alto na hierarquia, consistindo de um Nó no nível inicial conectado à dois ou mais Nós no nível secundário. Estes Nós, por sua vez, conectam-se à Nós no terceiro nível e assim por diante. Assim a rede estende-se como "galhos de uma árvore" [30]. Redes com topologia de árvore devem possuir pelo menos três níveis de hierarquia, pois apenas dois níveis caracterizam uma topologia de estrela. Os Nós nessa topologia agem como escravos para o Nó localizado no nível superior e mestres para os Nós localizados nos níveis inferiores, e Nós que desejem enviar pacotes entre si sem estarem diretamente conectados devem rotear seus pacotes pela rede. O número de Nós ligados à um mestre é chamado de *branching factor* e é limitado pelas características da rede, no caso desse projeto a limitação é o número de canais de comunicação dos módulos, e deve ser maior ou igual à 2 pois um *branching factor* igual à 1 corresponde à uma topologia linear.

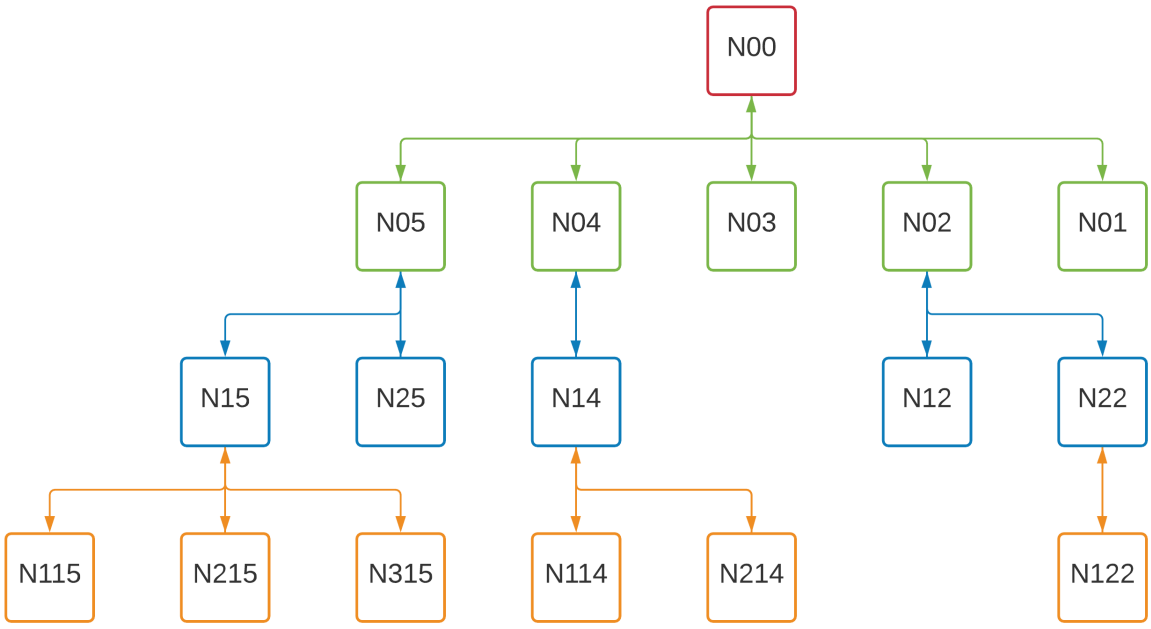

Figura 35 – Topologia da rede RF. Fonte: Elaborada pelo autor.

Já uma rede *mesh* conecta os múltiplos Nós diretamente e entre si, formando uma estrutura similar à uma teia. Nós de uma rede *mesh* são ditos totalmente conectados quando qualquer Nó da rede possui uma conexão direta com qualquer outro Nó, caso contrário diz-se que os mesmos são parcialmente conectados. Normalmente, redes *mesh* só são totalmente conectadas em pequena escala, pois o número de conexões cresce conforme

$$C = \frac{N * (N - 1)}{2} \tag{4.1}$$

onde C é o número de conexões necessárias para tornar a rede totalmente conectada, e N é o número de Nós conectados à rede. Assim, normalmente constroem-se redes *mesh* parcialmente conectadas aonde a maioria dos Nós, ou até mesmo todos os Nós, contém mais de uma conexão. As topologias de árvore e *mesh* são demonstradas pelas Figuras 36 e 37, respectivamente.

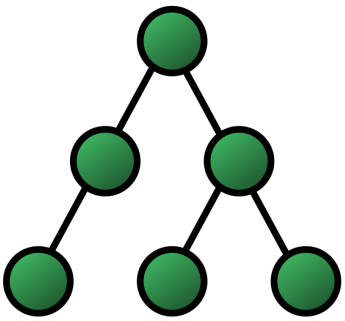

Figura 36 – Rede com topologia de árvore [31].

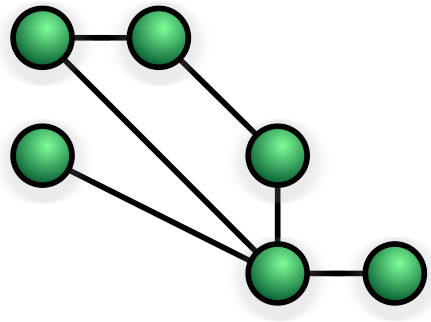

Figura 37 – Rede com topologia *mesh* parcialmente conectada [31].

4.3 Algoritmo de identificação de novas conexões e atribuição automática de identificadores e endereços

Para que a rede torne-se a mais dinâmica possível, deseja-se que não seja necessário configurar cada Nó individualmente, pois Nós devem ser capazes de se conectar, desconectar, e trocar de lugar dentro da rede automaticamente. Assim, foram criadas algumas funções para realizar a distribuição de endereços e identificadores dinamicamente. Essas funções são apresentadas na Tabela 12 e explicadas ao longo desta seção.

Tabela 12 – Funções para distribuição automática de endereços e identificadores únicos

Função	Parâmetros	Retorno
`getNextAvailableID`	Nenhum	`uint8_t`
`checkConnections`	`bool`	`void`
`checkData`	Nenhum	`void`

A distribuição de novos endereços na rede ocorre de maneira similar ao protocolo DHCP (*Dynamic Host Configuration Protocol*), utilizado em redes cuja identificação é feita por meio do protocolo de internet (IP, do inglês *Internet Protocol*). O protocolo DHCP é utilizado para designar automaticamente endereços IP à dispositivos que se conetem à rede através da comunicação entre dispositivo e servidor pelo meio de pacotes UDP, e consiste em quatro etapas:

- *Discovery*. O dispositivo, ao conertar-se na rede, envia um pacote *broadcast* do tipo `DHCPDISCOVER` na rede.
- *Offer*. Ao receber um pacote UDP do tipo `DHCPDISCOVER`, o servidor então envia um pacote do tipo `DHCPOFFER` ao endereço MAC do dispositivo, contendo, entre outras informações, o endereço IP ofertado pelo servidor.
- *Request*. O dispositivo então envia ao servidor um pacote do tipo `DHCPREQUEST` requisitando o endereço IP que lhe foi ofertado.
- *Acknowledge*. Caso aceita, o servidor responde a requisição com uma confirmação do tipo `DHCPACK`.

O pacote *broadcast* enviado por clientes buscando um endereço IP na rede é necessário pois o endereço do servidor é dinâmico, e, assim, pode mudar entre conexões e requisições DHCP. A grande diferença para o processo de distribuição de endereços na rede RF é o fato do Nó mestre, que nesse caso atua com o papel do servidor, sempre possuir o endereço fixo 0, eliminando assim a necessidade de *broadcast* (apesar da rede oferecer suporte para *broadcast* e *multicast*).

Além de endereços únicos, cada Nó conectado à rede possui um identificador único. Ao passo que em redes dinâmicas endereços podem mudar, os identificadores são únicos e persistem após a desconexão. Para tal, cada identificador deve ser configurado manualmente, algo que não era desejado para a rede criada. No intuito de deixá-la o mais dinâmica possível, e sabendo que os pacotes enviados pelos Nós contém informações suficientes para identificar suas localizações físicas caso necessário, foram então criados métodos para distribuição de IDs únicos automaticamente no momento de conexão à rede. Esses IDs são utilizados pela biblioteca `RF24Mesh` e na mesma possuem tamanho de 8 bits, limitando sua faixa de valores à 256 (entre 0 e 255). Esse valor pode ser modificado caso necessário.

A distribuição de IDs únicos acontece no momento de conexão à rede. Cada Nó que deseja-se conectar à rede deve configurar seu ID para o ID reservado "1" e conectar-se à rede. Com uma conexão bem sucedida o Nó recebe um endereço automático, como citado anteriormente. Uma vez conectado à rede o mesmo envia um pacote de controle para o Nó mestre. O pacote de controle nada mais é do que um pacote com *header* de tipo "`N`" (78, segundo a Figura 21), contendo um byte irrelevante de conteúdo. Esse tipo de pacote também é enviado quando testa-se a conexão à rede por parte dos Nós.

Há duas maneiras do *hub* obter o ID único que é enviado para o Nó. A primeira maneira é através da função `getNextAvailableID`, que percorre a lista de Nós conectados procurando por endereços liberados. A lista de Nós conectados é proveniente da biblioteca `RF24Mesh`, e, por padrão, Nós não são removidos uma vez que percam conexão. Ao invés de implementar métodos para remoção de Nós inativos da lista, uma vez determinados como inativos simplesmente marca-se seus endereços como disponíveis, ou seja, endereço 0. Tal funcionalidade é semelhante à função `releaseAddress()`, da biblioteca `RF24Mesh`, utilizada por Nós que queiram liberar seus endereços antes de colocar os módulos nRF24L01+ em modo de baixo consumo de energia, por exemplo. Assim, caso a função `getNextAvailableID` encontre um Nó cujo endereço seja 0, então o ID desse Nó é retornado. Caso contrário a função retorna 0, sinalizando que não há IDs sobrando na lista de Nós conectados.

A segunda maneira é através de uma variável global, atualizada a cada nova conexão, e que indica o menor valor para ID único que ainda não foi utilizado. Inicialmente assumindo valor 2, a variável é atualizada sempre que a função `checkConnections` é chamada. Para encontrar o valor desejado utilizam-se duas variáveis locais enquanto a lista de Nós conectados é percorrida. A primeira variável assume o valor do maior ID encontrado na lista, enquanto a segunda tenta encontrar valores entre 2 e o maior valor que ainda não tenham sido utilizados. Para isso inicia-se a variável com valor 2, e a cada Nó que possua ID igual à seu valor então incrementa-se a variável em uma unidade. Após percorrer a lista caso essa variável possua valor maior do que a variável contendo o maior valor então esse número é retornado e atualiza-se a variável global com ele. Caso contrário então a lista de Nós conectados é percorrida novamente, para verificar se o ID encontrado realmente

não está presente. Caso a variável não altere seu valor, indicando que realmente o valor não está presente, então a variável global é atualizada com este valor. Caso contrário a variável contendo o maior valor é incrementada e esse é o valor para o qual a variável global é atualizada. O fluxograma da Figura 38 demonstra a lógica desse funcionamento.

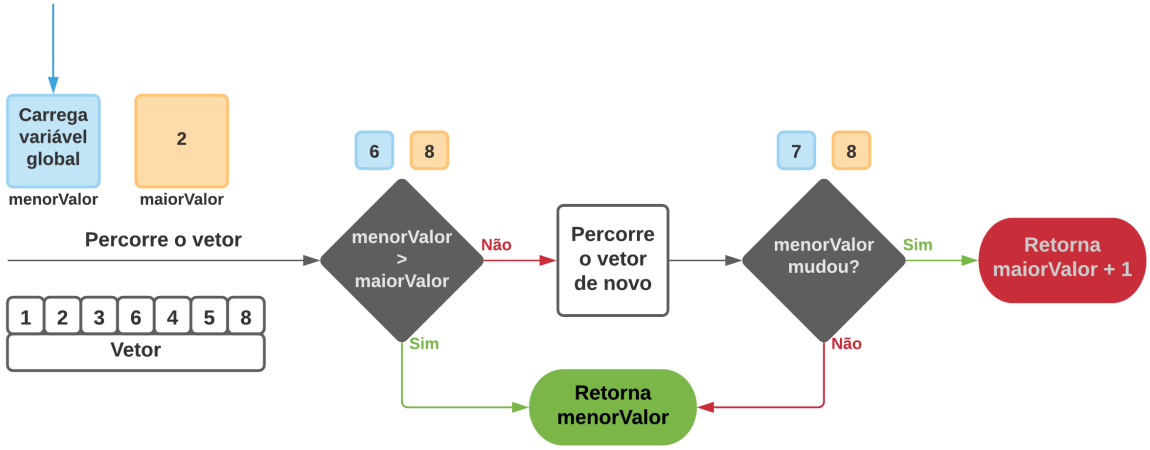

Figura 38 – Fluxograma da obtenção de novo ID. Fonte: Elaborada pelo autor.

A função `checkConnections` foi escolhida para encontrar esse valor pois também é responsável por verificar a conexão de Nós à rede por parte do servidor, e, assim, percorre a lista de Nós conectados procurando por Nós inativos. A função atua de maneira diferente para Nós localizados no nível secundário e níveis subsequentes da rede. Os níveis de uma rede com topologia de árvore são explicados na Seção 4.2. Para Nós do nível secundário basta que o envio de um pacote seja feito com sucesso, uma vez que o próprio módulo de RF realiza o *acknowledgement* de pacotes enviados entre módulos. Esse método, porém, não funciona quando há o encaminhamento de pacotes, uma vez que o *acknowledgement* à nível de hardware é realizado somente para a primeira etapa, não detectando falhas nos envios seguintes, conforme mostra a Figura 39.

Assim, para Nós cujo endereço indica níveis inferiores ao secundário é necessário realizar o envio de uma mensagem e esperar uma resposta dos mesmos. Essa mensagem é de tipo "`d`" (100, segundo a Figura 21) e, de maneira semelhante às mensagens de tipo "`N`", seu conteúdo não é importante. Uma vez que marcar um Nó como inativo não é algo crítico, pois todos os Nós verificam o estado da conexão e, caso necessário, solicitam novos endereços antes de enviar qualquer mensagem ao *hub*, após o envio o *hub* aguarda somente $500ms$ por uma resposta. A função responsável pela obtenção e processamento da resposta é a função `checkReceive`, explicada na Seção 3.3.3.6 para o contador, mas que possui o mesmo funcionamento para o *hub*. Vale notar que demais pacotes recebidos enquanto o *hub* aguarda resposta são processados normalmente, sem que haja perda de informações. Caso o *hub* receba uma resposta dentro desse intervalo então o Nó mantém

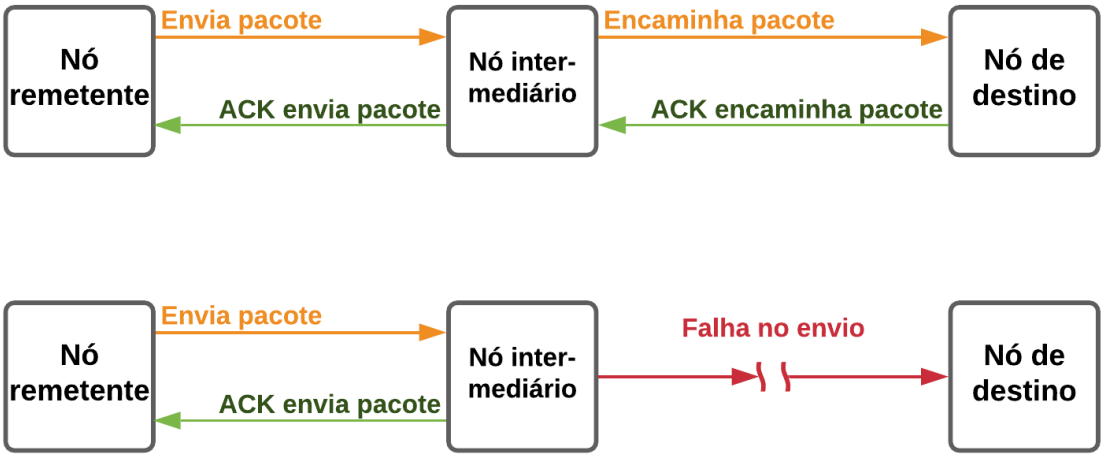

Figura 39 – Falha no envio de pacote encaminhado e *acknowledgement* do hardware. Fonte: Elaborada pelo autor.

seu endereço na lista e a função `checkConnections` segue adiante para verificar o próximo Nó.

4.4 Funções para recebimento de informações na rede sem fio

Qualquer pacote recebido pelo *hub* é lido e interpretado pela função `checkData`. O primeiro passo é verificar o *header* do pacote, para obter informações como tipo de dado sendo transmitido e ID do remetente. Os tipos de *header* recebidos e enviados pelo *hub* são apresentados na Tabela 13.

Assim, uma vez que o tipo de dado seja conhecido o *hub* pode realizar o processamento devido. Para *headers* `N` e `d` nenhum processamento é necessário, o recebimento de tais pacotes já serve como confirmação. Já pacotes de tipo `M` e `L` necessitam de processamento, pois devem ter seus dados armazenados, formatados, e enviados ao servidor da empresa. Como visto na Seção 3.3.3.6, a função `encodePacket` copia as informações contidas na estrutura de dados para um vetor, que é então enviado. Ao ser recebido é necessário que essas informações sejam transferidas do vetor para a estrutura de dados, e para isso a função `decodePacket` realiza os mesmos processos de cópia de memória, porém copiando do vetor para a estrutura. Além disso, a função `decodePacket` recebe uma variável `bool` como parâmetro, indicando a presença ou ausência do número da linha. O estado dessa variável depende do tipo de *header* lido.

Uma vez que os dados tenham sido extraídos do vetor e armazenados na estrutura, então é chamada a função `sendUdpPacket`, responsável pela formatação dos dados para

Tabela 13 – Tipos de *header* recebidos e enviados pelo *hub*

Header type	Tipo de dado	Observações
`M`	`Dados`	a
`L`	`Dados`	a
`N`	`uint8_t`	b
`d`	`bool`	c
`t`	`uint8_t`	d

a. *Headers* tipo `M` e `L` são utilizados para envio de dados de apontamento. Como mencionado na Seção 3.3.3.2, o contador pode enviar uma estrutura contendo código do operador ou linha, e como a formatação de ambos é diferente para envio ao servidor então existem *headers* separados. Um *header* `M` indica que a estrutura enviada contém código de operador, enquanto um *header* `L` indica a presença do número da linha.
b. Pacotes com *header* `N` são utilizados para requisições de novos IDs por parte dos Nós. Apesar de terem seu conteúdo lido, nenhuma operação é aplicada sobre tal conteúdo.
c. Como mencionado anteriormente, pacotes com *header* `d` são utilizados para verificação de conexão com Nós que estejam em níveis inferiores ao secundário.
d. O *header* `t` é exclusivo para envio, sendo tratado como qualquer outro tipo de *header* não incluso na lista caso recebido. Pacotes enviados com *headers* `t` são pacotes de confirmação de entrega UDP direcionados ao Nó que produziu os dados. O envio, recebimento, e confirmação de pacotes UDP é abordado na Seção 4.5.4.

envio ao servidor via protocolo UDP. O formato escolhido para envio de dados ao servidor foi o formato JSON. O formato JSON é independente de linguagem e utiliza texto legível para armazenar e enviar dados tornando-o fácil de ler e interpretar, tanto por máquinas quanto humanos. As informações são mantidas em objetos ou vetores, na forma de pares "nome/valor" [32]. Por ser um formato leve, aberto, independente de linguagem, e fácil de ler e interpretar, o formato JSON é amplamente utilizado para envio de dados entre aplicações web e servidores. Muitas linguagens fornecem suporte nativo para formatação de JSON, e, apesar da plataforma Arduino não oferecer suporte nativo, os métodos e ferramentas para criação, edição, e interpretação de documentos JSON são implementados na plataforma pela biblioteca terceirizada `ArduinoJSON`.

A biblioteca `ArduinoJSON` torna fácil, com poucos comandos, criar e formatar um documento JSON. Uma vez que o documento contenha todos os pares "nome/valor" necessários o mesmo é convertido em uma variável `String` para, então, ser enviado. O uso de variáveis `String` não é recomendado quando há preocupações com memória, porém a variável declarada para envio é local, e permanece na memória somente durante a execução da função `sendUdpPacket`. A criação do documento JSON e sua conversão em uma variável `String` é apresentada a seguir:

```
1 StaticJsonDocument<200> doc;
2 doc["OP"] = dados.OP;
3 doc["Operacao"] = dados.operacao;
4 if (dados.operador > 0) {
5     doc["Operador"] = dados.operador;
```

```
}
if (dados.linha > 0) {
    String linha = String("L" + String(dados.linha));
    doc["Linha"] = linha;
}
doc["Produtos"] = dados.produtos;
String envio;
envio = doc.as<String>();
```

A seguir são apresentados exemplo de documentos em formato JSON:

```
{"OP":"0123456","Operacao":"205","Operador":"702","Produtos":"0"}
{"OP":"0123456","Operacao":"322","Linha":"L32","Produtos":"150"}
```

Uma vez que a `String` "envio" contenha o documento JSON então a mesma pode ser enviada para o servidor. O envio de pacotes UDP é discutido na Seção 4.5.4. São realizadas três tentativas de envio do pacote UDP ao servidor, e uma vez enviado o pacote então o *hub* aguarda uma confirmação, que também será encaminhada ao Nó de origem dos dados através de um pacote com *header* `t`. Uma vez recebida a resposta então chama-se a função `checkSend`, que realiza três tentativas de envio de pacotes `t`. A verificação de recebimento de tais pacotes não é implementada, pois em caso de falha os dados apenas não serão reenviados mas já constam no servidor. E, como discutido na Seção 3.3.3.6, o operador também é notificado que houve falha nos dados, podendo requisitar que um supervisor consulte o servidor e realize correções, se necessário. Durante o período de espera por resposta do servidor o *hub* não checa por novos pacotes na rede, porém uma vez que o tempo de espera tenha-se esgotado, ou uma resposta tenha sido obtida e enviada ao Nó de origem, os pacotes já são lidos e processados.

4.5 Integração do sistema desenvolvido à um sistema pré-existente

A maior vantagem de um sistema sem fio com rede própria é que o único componente ao qual é necessário realizar integração com o sistema pré-existente é o *hub* de comunicações. Os demais Nós da rede RF são invisíveis à rede Wi-Fi do ambiente industrial, e vice-versa. Além disso, como o envio de pacotes ao servidor é realizado através de protocolos e normas altamente regulamentados há fácil configuração e alto grau de interoperabilidade entre *hub* e servidor.

4.5.1 Revisão teórica: protocolos de comunicação Wi-Fi

Há diversas maneiras de enviar dados por uma rede sem fio, e muitas das tecnologias empregadas utilizam protocolos e normas bem definidos. Os protocolos e normas para

comunicação sem fio, por exemplo, são definidos pelo padrão IEEE 802.11. A IEEE (Instituto de Engenheiros Elétricos e Eletrônicos, do inglês *Institute of Electrical and Electronics Engineers*) é uma associação para engenheiros nas áreas de engenharia elétrica, eletrônica, e áreas associadas. Existente desde a década de 1960, é atualmente a associação de caráter técnico com o maior número de membros mundialmente, contando com quase 400 mil membros. A associação é responsável por regulamentar muitos dos protocolos mais utilizados na área de telecomunicações, como a família de protocolos IEEE 802, responsável por protocolos para comunicação em redes locais (LAN, do inglês *Local Area Networks*), pessoais (PAN, *Personal Area Network*), e municipais (MAN, *Metropolitan Area Networks*).

Os protocolos da família IEEE 802, porém, fazem parte uma categoria maior de protocolos, chamados de suíte de protocolos Internet, comumente referenciada como TCP/IP, e é através de protocolos dessa suíte que a comunicação via internet é realizada. Abordar todos os tópicos da suíte TCP/IP é além do escopo deste trabalho, apesar disso serão abordados os protocolos utilizados: UDP, IP, Ethernet, e Wi-Fi. Dados enviados através de redes Wi-Fi muitas vezes utilizam múltiplos protocolos, e para tais casos acabam sendo "envelopados" por diferentes protocolos, como demonstra a Figura 40.

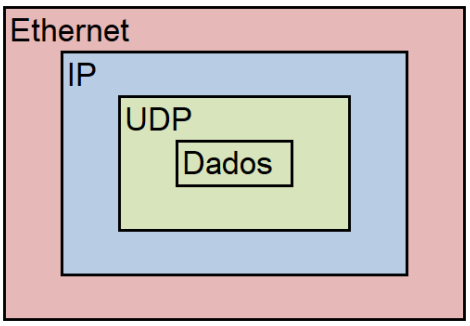

Figura 40 – Dados enviados via Wi-Fi encapsulados por múltiplos protocolos. Fonte: Elaborada pelo autor.

4.5.1.1 Ethernet

O protocolo Ethernet é um protocolo para comunicação em redes LAN, MAN, e WAN através de cabos (originalmente coaxiais, atualmente par trançado). Dados transmitidos por Ethernet são divididos em pedaços menores, chamados de datagramas. Esses datagramas contém informações como o endereço MAC de destino e remetente, EtherType, e bits para verificação de erros no envio, além dos próprios dados sendo transmitidos. Endereços MAC são endereços dados à placas de rede de dispositivos, e apesar de muitas placas poderem alterar seus endereços, muitas vezes são disponibilizados pelos próprios fabricantes dos dispositivos. EtherType é um conjunto de 2 bytes contendo uma descrição do protocolo que é encapsulado pelo datagrama Ethernet. A Figura 41 demonstra um

datagrama Ethernet (em vermelho) encapsulando um datagrama IP (em azul) que, por sua vez, encapsula um datagrama UDP (em verde). Normalmente os bits para verificação de erros do datagrama Ethernet (*frame check sequence, FCS*) encontram-se ao final do pacote, porém foram descritos justamente com os demais campos do datagrama Ethernet para facilitar a leitura.

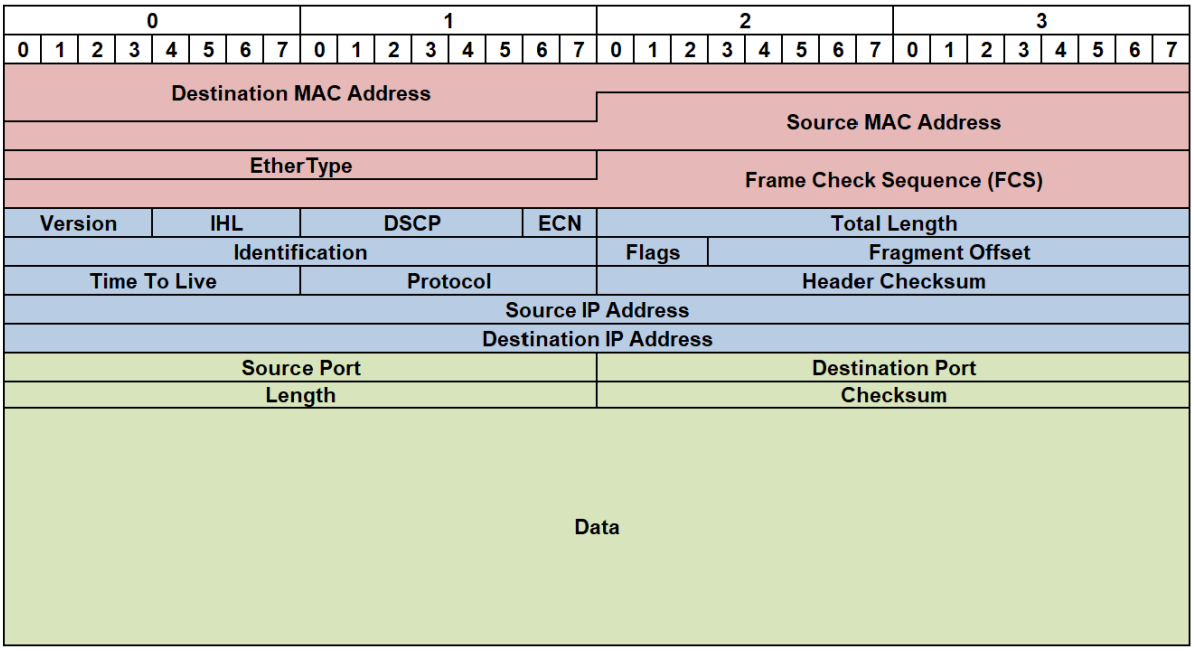

Figura 41 – Datagrama de um pacote UDP enviado na rede e recebido por um computador com conexão cabeada. Fonte: Elaborada pelo autor.

4.5.1.2 Wi-Fi

O padrão IEEE 802.11, conhecido como Wi-Fi, é um conjunto de padrões que estabelecem técnicas para utilizar frequências de 2.4 GHz e 5 GHz para transmissão de dados por radiofrequência. Existem dois tipos de arquiteturas para redes Wi-Fi: redes *ad hoc* e redes de infraestrutura. Redes *ad hoc* estabelecem comunicação P2P (ponto a ponto, do inglês *peer-to-peer*) entre duas estações, e diversas conexões podem ser criadas de maneira que a rede eventualmente feche um *loop*. Já redes de infraestrutura criam conexões entre múltiplas estações e um ponto de acesso (AP, do inglês *Access Point*). Ambas arquiteturas são demonstradas na Figura 42.

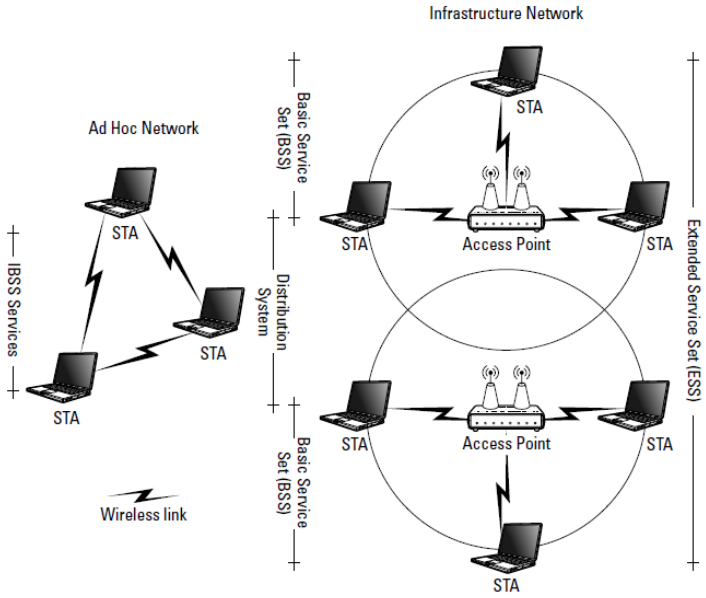

Figura 42 – Arquitetura de redes Wi-Fi *ad hoc* e de infraestrutura [30].

Vale notar que cada padrão da família IEEE 802.11 determina diferentes normas, padrões de modulação, bandas de operação, taxas máximas de transmissão, entre outros detalhes. Porém a interoperabilidade de dispositivos, contando com mais de 3 bilhões de dispositivos mundialmente em 2019 [33], resultante das normas IEEE 802 ajudou a transformar o Wi-Fi na revolução para a comunicação sem fio "tão grande quanto celulares para comunicações telefônicas" [30].

4.5.1.3 IP

O protocolo Internet, ou somente IP, é um protocolo de comunicações utilizado para transmissão de datagramas em rede. Há duas versões do protocolo em uso, IPv4, a versão mais utilizada, e IPv6, uma versão atualizada atualmente em processo de implementação. Pacotes gerados pelo protocolo IP, por exemplo para comunicação TCP ou UDP, contém os endereços de destino e origem, e não fazem demandas excessivas à rede, liberando assim cada roteador intermediário a decidir o caminho pelo qual o pacote passará. Além disso, é responsável por gerar os endereços de origem e destino, algo que é diferente para as duas versões do protocolo. Endereços IPv4 ocupam 32 bits e são expressos em formato decimal, com pontos separando cada byte (xx.xx.xx.xx), enquanto endereços IPv6 ocupam 128 bits e são expressos em formato hexadecimal, com dois pontos separando cada intervalo de 2 bytes (xx:xx:xx:xx:yy:yy:yy:yy). Além disso ambas versões possuem tipos diferentes de transmissão.

O datagrama IP contém informações como versão do protocolo utilizada, tamanho do *header* do datagrama, tamanho total do pacote, protocolo encapsulado pelo datagrama IP, *checksum* para verificação de erros, e endereços de origem e destino. A Figura 41

demonstra um datagrama IP (em azul) encapsulando um datagrama UDP (em verde) e encapsulado por um datagrama Ethernet (em vermelho).

4.5.1.4 UDP

O protocolo UDP (do inglês, *User Datagram Protocol*) é um protocolo para transporte de pequenos dados em um datagrama, enviados entre *sockets* de dois dispositivos em uma rede IP. É um protocolo sem métodos definidos para verificação de entrega de pacotes, especialmente quando há fragmentação de dados. O protocolo UDP é dito *connectionless*, ou não-orientado a conexão, pois não requer uma conexão estabelecida entre o ponto de origem e destino para o envio de dados. Assim, dados são enviados de forma mais rápida porém menos confiável, com a implementação de rotinas de *acknowledgement* do recebimento de pacotes ficando à cargo do usuário. O protocolo UDP é utilizado por diversos outros protocolos da suíte TCP/IP, como o protocolo DHCP mencionado na Seção 4.3. O *header* de um datagrama UDP é mínimo, de forma a diminuir o *overhead* na rede, contendo apenas as portas de origem e destino, o tamanho do datagrama, e um *checksum* para verificação de erros no envio. A Figura 41 demonstra um datagrama UDP (em verde) encapsulado por datagramas IP (em azul) e Ethernet (em vermelho). Nas Figuras 43 e 44 são apresentados exemplos de um datagrama real enviado pelo *hub* ao servidor, na forma hexadecimal de origem e convertidos para forma decimal para melhor visualização, respectivamente.

0								1								2								3							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
7c								8b								ca								00							
b2								29								f0								bf							
97								0f								66								e7							
08								00								-								-							
-								-								-								-							
04				05				000				0				00								e1							
ff								72								0	0	0	0												
80								11								00								00							
c0								a8								00								32							
c0								a8								00								e6							
c4								a2								1b								5d							
00								cd								83								47							
86								00								00								00							
06								22								00								00							
22								01								00								22							
02								00								22								03							
00								22								04								00							
22								05								00								22							
06								00								22								07							
00								22								08								00							
⋮								⋮								⋮								⋮							

Figura 43 – Exemplo de dados contidos em um datagrama conforme a Figura 41, em formato hexadecimal. Fonte: Elaborada pelo autor.

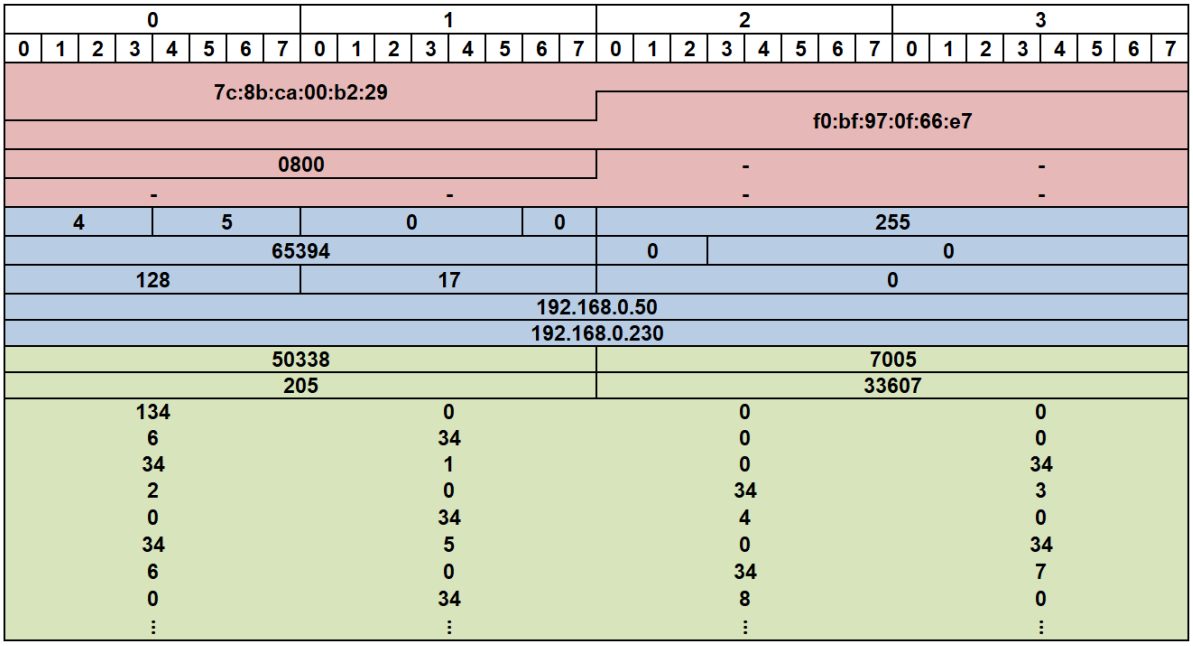

Figura 44 – Exemplo de dados contidos em um datagrama conforme a Figura 41, em formato decimal. Fonte: Elaborada pelo autor.

4.5.2 Sistema já existente

A empresa conta com um sistema de apontamento através de estações fixas em pontos chaves do ambiente industrial. Operadores se deslocam até as estações ao receberem uma ordem de produção a ser realizada, inserem os dados solicitados, informam o início da produção, e retornam à suas estações para iniciar o processo produtivo. Uma vez que o processo produtivo esteja finalizado o operador, novamente, se desloca até a estação e repete o procedimento, dessa vez informando o número de produtos que o mesmo produziu. Os dados são enviados via Wi-Fi, através de pacotes UDP, para o servidor, que contabiliza também o tempo levado para a produção do lote. Supervisores tem acesso a mais ferramentas através de terminais especializados, e podem alterar informações incorretas que tenham acidentalmente sido apontadas.

4.5.3 Conexão à rede Wi-Fi existente sem credenciais *hard coded*

Conforme discutido nas diretrizes para o projeto, na Seção 3.1, a conexão do *hub* à rede Wi-Fi pré-existente deve ser realizada de maneira dinâmica, sem que o identificador da rede (SSID, do inglês *Service Set Identifier*) e senha sejam *hard coded*. Para tal, utiliza-se a biblioteca `AutoConnect`. A biblioteca `AutoConnect` torna possível inserir SSID e senha da rede Wi-Fi à qual deseja-se conectar o microcontrolador ESP8266 após a inicialização do mesmo. Uma vez inseridos então as credenciais são gravadas na memória EEPROM do

dispositivo, e acessadas toda vez que o mesmo é reinicializado, sem que haja a necessidade de serem reinseridas a cada reinicialização. Para tal, primeiramente o microcontrolador é configurado no modo `SOFT_AP`, criando sua própria rede Wi-Fi à qual o usuário pode ser conectar. Uma vez conectado um *captive portal* irá abrir automaticamente contendo o menu de configuração AutoConnect. A partir do menu o usuário pode escolher em qual rede deseja conectar o microcontrolador, e ao inserir SSID e senha o mesmo reinicia automaticamente. Uma vez reiniciado, agora em modo `STA` o mesmo conecta-se à rede determinada pelo usuário e encerra sua própria rede Wi-Fi. O funcionamento do sistema pode ser observado na Figura 45.

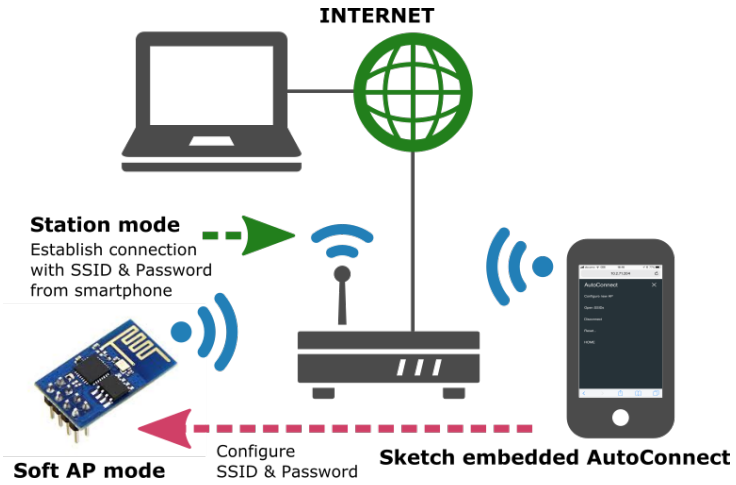

Figura 45 – *Overview* do funcionamento da biblioteca `Autoconnect` [34].

4.5.4 Envio de dados do *hub* ao servidor

O envio de pacotes do *hub* ao servidor é realizado por meio de pacotes UDP. As únicas informações necessárias para o envio de um pacote UDP são a porta de destino e o IP do destinatário. Ambos podem ser facilmente descobertos caso sejam dinâmicos através de técnicas como *broadcast* e métodos semelhantes à DHCP, porém como tanto o servidor quanto a porta UDP do sistema pré-existente na empresa são estáticos ambos foram *hard coded* no *hub*. Uma vez que sabe-se o destino do pacote UDP basta que os dados estejam formatados de uma maneira que o servidor os entenda e então os mesmos podem ser enviados. A formatação dos dados obtidos dos Nós para o formato JSON é abordada na Seção 4.4.

Uma vez que os dados foram obtidos e formatados corretamente, o envio de pacotes fica à cargo da função `sendUdpPacket`. Essa função utiliza métodos implementados pela biblioteca `WiFiUdp` para criação, preenchimento, e envio de pacotes UDP. São realizadas três tentativas de envio de pacotes, com o sucesso de alguma pulando tentativas consequentes e passando a esperar um retorno. Como explicado na Seção 4.5.1.4, o protocolo UDP não possui nativamente o reconhecimento da entrega de pacotes, assim deve-se es-

perar uma confirmação do próprio servidor. A função chamada para esperar respostas é a função `checkUdp`, que recebe como parâmetros um intervalo de espera e uma `String` contendo a resposta desejada. Uma vez que o *hub* receba uma resposta do servidor então é chamada a função `checkSend`, que envia um pacote ao Nó de origem dos dados informando-o do sucesso no envio dos dados ao servidor. A rotina de envio de pacote UDP, confirmação de recebimento, e resposta ao Nó é apresentada a seguir:

```
uint8_t tentativa = 0;
uint8_t maxTentativas = 3;
while (tentativa < maxTentativas) {
    udp.beginPacket(serverIP, udpPort);
    udp.print(envio);
    if (udp.endPacket() == 1) {
        tentativa = 0;
        while (tentativa < maxTentativas) {
            if (checkUdp(2000, "recebido")) {
                tentativa = 0;
                while (tentativa < maxTentativas) {
                    if (checkSend()) {
                        break;
                    } else {
                    tentativa++;
                    }
                }
                break;
            } else {
            tentativa++;
            }
        }
        break;
    }
}
```

O método `print()` utilizado para o preenchimento do pacote UDP não faz parte da biblioteca `WiFiUdp`, que contém somente métodos sob o nome de `write()` para tais funções, mas é utilizado por possuir suporte para variáveis do tipo `String`, eliminando assim a necessidade de conversão em um vetor de caracteres. Este método é implementado pela biblioteca `Print.h`, incluída na biblioteca `Stream.h`, que por sua vez é incluída na biblioteca `Udp.h`, que, finalmente, é incluída na biblioteca `WiFiUdp`. A função `checkUdp`, responsável pela validação da resposta recebida do servidor, é apresentada a seguir:

```
bool checkUdp(uint32_t timeout, String confirmacao) {
    uint32_t timerCheck = millis();
    while (millis() < timerCheck + timeout) {
        if (udp.parsePacket() > 0) {
            String req;
            req = "";
```

```
7               while (udp.available() > 0) {
8                   char z = udp.read();
9                   req += z;
10              }
11              if (req == confirmacao) {
12                  return true;
13              }
14          }
15      }
16    return false;
17  }
```

Um exemplo do envio de dados pode ser encontrado na Figura 46. Para captura dos pacotes foi utilizado o software Wireshark. Vale notar que esse pacote é meramente um exemplo, e não a comunicação real com o servidor, pois pacotes enviados por meios sem fio em redes com padrão de segurança WPA se encontram ocultos abaixo da camada IP, e, portanto, não são capturados pelo software Wireshark. Porém, para assegurar que os pacotes de teste apresentam comportamento similar à pacotes enviados ao servidor real, o servidor utilizado para recebimento dos pacotes mostrados na Figura 46 é uma cópia do servidor real.

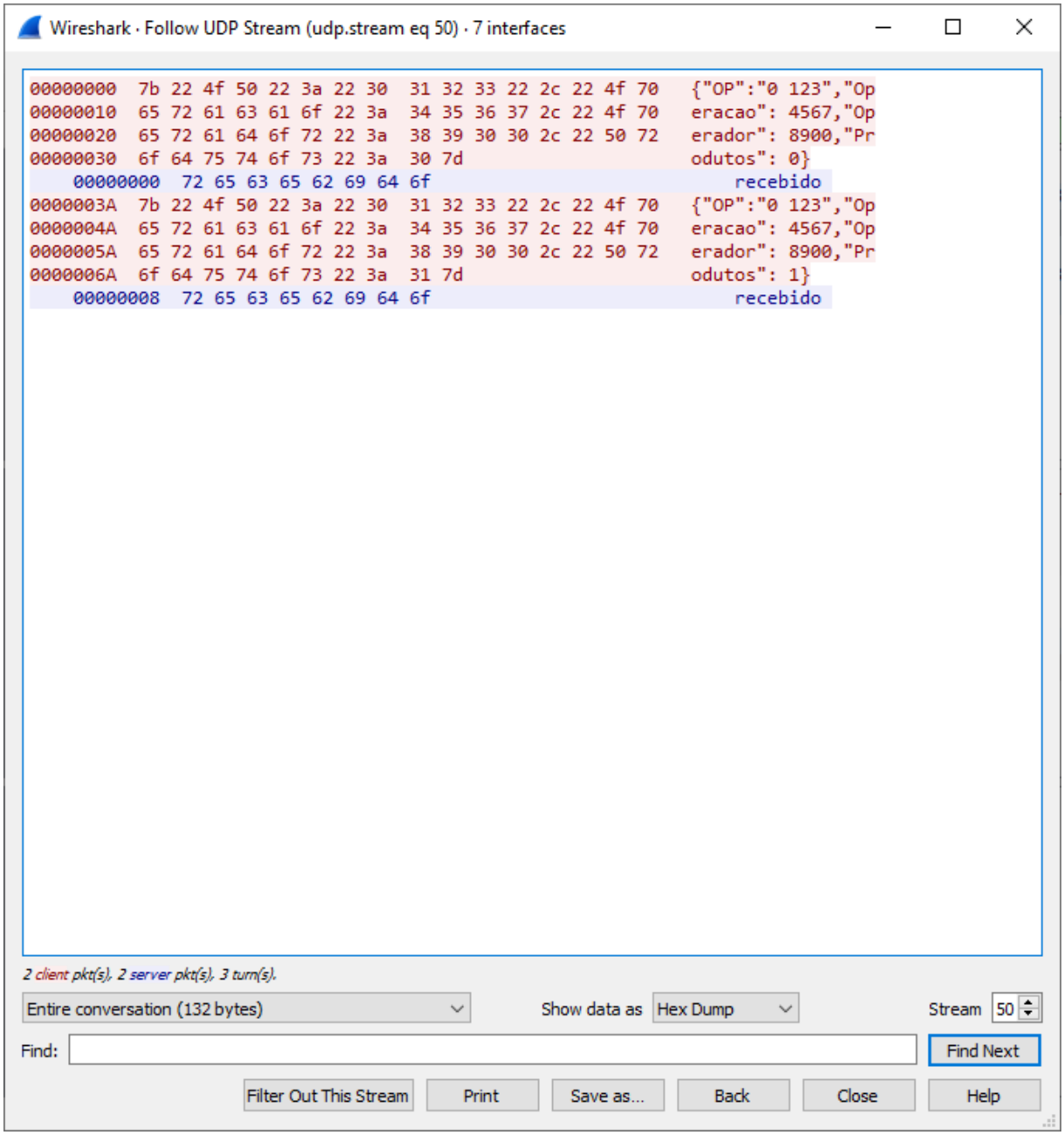

Figura 46 – *Stream* UDP entre cliente e servidor capturada no software Wireshark. Fonte: Elaborada pelo autor.

5 Resultados e discussão

5.1 Economia de tempo

Como mencionado no Capítulo 2, operários devem se deslocar de suas estações de trabalho até uma estação de apontamento ao iniciar e terminar uma nova ordem de produção. Devido à ausência de um leitor de QR Code, proposto como futura melhoria na Seção 5.6.1, e também por conta das limitações impostas pelo software para controle do efeito de *bounce* no *keypad*, o apontamento com o sistema desenvolvido é mais lento. Mesmo assim, distribuir sistemas de contadores que também sejam capazes de realizar apontamentos pelo ambiente fabril prova-se mais eficiente por reduzir o deslocamento de operadores pela fábrica e diminuindo esperas quando necessitam-se realizar apontamentos simultâneos. Operadores levam, em média, 32 segundos para inserir dados de apontamento no sistema desenvolvido, e precisam inseri-los apenas uma vez para cada ordem de produção, visto que o sistema automaticamente realiza o apontamento inicial (com 0 produtos produzidos) na seleção do número de produtos a serem pesados. Enquanto isso, o tempo para apontamento de uma ordem de produção atualmente, considerando apontamento inicial e final, pode chegar a minutos, dependendo do tráfego de operadores nas estações de apontamento. Como a maior parte do tempo é gasto em deslocamento e espera, a dispersão entre valores medidos para o tempo de apontamento atual é grande demais para realizar-se conclusões.

Além disso, como mencionado por Breton [6] no Capítulo 1, o uso do sistema de contagem automática permite que o operador esteja focado e dedicando toda sua atenção à tarefa produtiva, deixando a contagem de unidades produzidas a cargo do sistema.

5.2 Testes

Ao longo do desenvolvimento foram realizadas várias baterias de teste para garantir que o projeto funcionasse da maneira esperada. As células de carga foram regularmente colocadas sob estresse para testar sua precisão, bem como deixadas ligadas durante longos intervalos para verificar a confiabilidade de suas medições ao longo do tempo. O algoritmo de detecção foi testado com a adição e subtração de múltiplos produtos simultaneamente, mostrando ser capaz de detectar com tranquilidade até 5 produtos adicionados ou removidos simultaneamente, apesar de sua precisão variar um pouco quando tratam-se de números maiores, como mostra a Tabela 14.

Apesar disso, quando produtos são inseridos de maneira unitária não houveram problemas de precisão, sendo testados lotes de até 125 produtos.

Tabela 14 – Precisão na adição e subtração de vários produtos simultaneamente

Número de produtos	Precisão para adição	Precisão para subtração
6	90%	80%
7	70%	50%
8	60%	40%
9	40%	10%
10	10%	10%

Obs. Foram usados produtos com peso unitário de 125g

Similarmente foram testados intensivamente o tráfego de pacotes nas redes RF e Wi-Fi. Para a confirmação de envio e recepção de pacotes na rede RF foram utilizadas diferentes combinações dos microcontroladores UNO e ESP8266 como transmissor e receptor, bem como o envio de diferentes tipos e tamanhos de dados. O alcance de transmissão de dados também foi testado múltiplas vezes, em diferentes ambientes ambientes, e com diferentes configurações. O envio e recepção de pacotes UDP na rede Wi-Fi também foi testado extensivamente, utilizando software especializado, como Wireshark, para a captura de pacotes na rede e visualização do conteúdo de tais pacotes. Foi testado também o envio de pacotes UDP não somente na rede local mas para IPs remotos, localizados fora da mesma.

5.3 Erros de medição

Assim como qualquer sistema, há a possibilidade de erros de contagem no sistema desenvolvido. Foram realizados dois tipos de testes, utilizando produtos de diferentes pesos, para a verificação do nível de precisão do contador. O primeiro teste tem como objetivo verificar quão rápida pode ser a produção até que o contador encontre-se sobrecarregado, e os resultados encontrados são expostos na Tabela 15. Para tal, foram realizados testes com lotes de 10 produtos sendo inseridos e retirados no intervalo especificado. Para cada intervalo foram realizados 10 testes de adição e 10 testes de retirada, recalibrando a balança após cada retirada. O intervalo foi medido utilizando outro microcontrolador Arduino, que acendia e apagava um LED a cada n segundos, e apresenta uma precisão suficiente para os intervalos desejados (a operação `digitalWrite` utilizada para o acendimento do LED executa em, aproximadamente, 3,4 microssegundos [35]).

Intervalos maiores que 4 segundos e menores que 1,5 segundos não foram testados pois, baseado em resultados de testes em tempos similares, pode-se deduzir com confiança o comportamento do sistema. O teto de 4 segundos foi escolhido por ser o tempo demorado pela etapa produtiva de dobra de cabos PP, introduzida na Seção 2.1.5.

O segundo teste realizado é sobre precisão à longo prazo. O contador foi ligado e deixado em vazio durante 30 minutos aguardando o primeiro produto. Após o término

Tabela 15 – Precisão da plataforma de medição para inserção e retirada periódica de produtos

Intervalo (s)	% erros adição	% erros retirada
4,0	0%	0%
3,0	0%	0%
2,5	0%	0%
2,4	0%	0%
2,3	0%	0%
2,2	0%	0%
2,1	10%	0%
2,0	0%	10%
1,9	10%	10%
1,8	20%	40%
1,7	30%	70%
1,6	60%	100%
1,5	80%	100%

Obs. Foram utilizados produtos com peso unitário de 125 gramas.

desse período foi então inserido o primeiro produto e deixado-o em repouso por outros 15 minutos. Uma vez que o segundo intervalo tivesse sido concluído foi então pesado um lote de controle de 50 unidades. O procedimento foi repetido para produtos de diferentes pesos unitários, e os resultados são expostos pela Tabela 16.

Tabela 16 – Medição de lotes após plataforma de pesagem ociosa por 15 minutos

Peso unitário	Peso total	Erro	Houve erro de contagem?
70 g	3477 g	-23 g	Não
80 g	4013 g	13 g	Não
82 g	4088 g	-12 g	Não
106 g	5254 g	-46 g	Não
110 g	5510 g	10 g	Não
125 g	6274 g	24 g	Não

Obs. Os testes foram realizados em dias diferentes, é possível que fatores externos, como temperatura, influenciem na defasagem observada.

Nota-se uma clara possibilidade de melhoria para o sistema, na forma de medições mais rápidas. O baixo tempo de assentamento combinado à adição rápida de produtos acaba gerando falsos positivos ao final de quase todas as medições quando produtos são adicionados em intervalos menores ou iguais a 1,5 segundos. Para tal poderiam-se modificar os parâmetros citados na Seção 3.3.2.2 de forma a melhorar o tempo de resposta do sistema. Vale notar que, como citado na mesma seção, um tempo de resposta menor implica um número menor de amostras no conjunto temporário, prejudicando a precisão do sistema. Para aplicações com tempo de produção média acima de 2 segundos por unidade, próximo ao tempo de assentamento do sistema, nota-se que a precisão do sistema

é satisfatória, apresentando uma taxa combinada de 2,5% de erro (2 medições falhas em 80 medições totais).

Quanto às medidas ao longo do tempo, os resultados encontrados foram bastante satisfatórios. Um lote de cabos PP injetados chega a demorar mais de uma hora para ser dobrado, e observa-se que a medição ao longo de mais de 45 minutos demonstrou uma defasagem de menos de um produto. Além disso realizar medições de lotes após um longo intervalo com a plataforma ociosa é um pré-requisito, visto que o sistema deve possuir a capacidade para funcionar durante períodos de várias horas.

5.4 Testes de rede RF e Wi-Fi

Uma das maiores preocupações com o compartilhamento de dados via uma rede sem fio é assegurar que os dados sejam enviados com sucesso ao servidor. Apesar de todas as rotinas implementadas para *acknowledgement* de dados, ainda assim notaram-se eventuais perdas de pacotes. Testes para determinar fatores decisivos para a perda de pacotes, infelizmente, mostraram-se inconclusivos, com o único fator definitivo encontrado sendo a distância entre Nós. Devido à restrições de tempo para testes, a rede de compartilhamento de dados também não pode ser testada no ambiente fabril chegando à níveis inferiores ao secundário, assim a distribuição de identificadores e endereços, bem como a verificação de conexão para Nós de níveis inferiores permanecem como conceitos testados somente em laboratório.

De maneira similar, é difícil de se estabelecer um motivo específico para perdas de pacotes enviados via Wi-Fi ao servidor real, uma vez que os mesmos não são capturados pelo software Wireshark, como mencionado na Seção 4.5.4. Testes realizados com cópias do servidor rodando em máquinas locais também não se mostraram muito úteis pois perdas de pacote na rede eram ocorrências raras. Com todas as rotinas estabelecidas de aguardo de respostas e aviso para operadores em caso de problemas de conexão, a perda de pacotes na rede acaba não sendo um problema grave.

5.5 Observações dos operadores com o sistema proposto

O *feedback* de operadores é algo crucial para o bom desenvolvimento do projeto, afinal deseja-se que o mesmo seja intuitivo e visto como uma ferramenta, ao invés de um empecilho. Assim, operadores que tiveram a oportunidade de testar o sistema foram sempre incentivados à oferecer reclamações e sugestões. Algumas das sugestões de operadores que acabaram fazendo parte da versão final do projeto são:

- Peso exibido no *display* LCD varia demasiadamente. Versões iniciais exibiam os valores das variáveis `float` obtidas do módulo HX711 diretamente, oscilando medidas na casa de miligramas até 20 vezes por segundo, causando distrações. Foi então realizada uma alteração para que o valor exibido fosse truncado em um valor inteiro para gramas, oscilando menos e deixando o *display* com dados mais relevantes. Além disso, após a medida de peso ultrapassar mil gramas, então a unidade exibida passa a ser Kg e os valores são truncados para precisão de 10 gramas.

- Tempo de exibição de dados muito curto. Para solucionar esse problema, foi criada a função `Delay`, explicada na Seção 3.3.3, com o intuito de manter informações exibidas no *display* LCD por mais tempo, sem que o funcionamento do dispositivo fosse interrompido para todas as demais funções.

- *Keypad* informando dados incorretos. A leitura analógica de tensão não garante que leituras do *keypad* sejam precisas 100% a todo tempo, assim há a opção de recalibrá-lo a cada inserção de dados. Como mencionado na Seção 3.3.3.3, há também a calibragem automática caso sejam detectados 3 erros seguidos na inserção de dados.

Sempre que possível tentou-se conciliar o sistema desejado por operadores com a visão do projetista para o sistema. Assim, os problemas acima foram resolvidos, enquanto outros problemas, a exemplo o uso de um leitor de QR Code, tiveram sua implementação restrita ao planejamento de melhorias futuras.

5.6 Melhorias futuras

5.6.1 Melhorias no contador

É possível melhorar alguns aspectos do contador com tempo e recursos. Uma dessas possíveis melhorias é realizar a integração de um módulo I^2C para comunicação com o *display* LCD. Tais módulos custam menos da metade do *display* em si, e diminuem o número de conexões de dados necessárias de 6 para 2 (porém com restrições, é obrigatório que utilizem os pinos analógicos A4 e A5, pois somente os mesmos possuem capacidade de comunicação com protocolo I^2C). Tal melhoria iria possibilitar o uso de mais dispositivos no sistema, como por exemplo novos indicadores luminosos ou mecanismos de inserção de dados, que tornariam a experiência mais dinâmica para o operador. Um porém do uso deste módulo de comunicação seria uma total reestruturação da placa de circuito impresso e *case* para armazenamento do projeto.

Ao fazer uso do protocolo I^2C pode-se também utilizar outros componentes para inserção de dados e identificação do operador. A identificação do operador pode ser feita via tecnologia RFID (identificação por radio-frequência, do inglês *Radio-Frequency Identifiation*). Tal tecnologia permite escanear dados armazenados em crachás ou *tags* na

forma de campos eletromagnéticos. É necessário um módulo extra para ler tais dados, porém como o módulo possui pinos para comunicação via protocolos I^2C e SPI então o mesmo ocupará, no máximo, mais dois pinos de I/O do microcontrolador (no caso do protocolo SPI, um pino é para a função de seleção, CS, e outro pino é para uso interno do módulo e corresponde à função de *reset*). Os dados lidos pelo sensor são exibidos na Figura 47, aonde podem-se observar os bytes que poderiam ser correspondentes à identificação do operador. Essa identificação seria enviada ao servidor, onde a mesma poderia ser referenciada junto à um banco de dados de operadores para identificar o operador responsável pelo lote. Esse método é mais rápido que o método de inserção de identificador via *keypad* utilizado neste projeto, porém requer o envio de pacotes maiores via RF, os operadores que utilizem esse método de apontamento devem ter sempre à mão seus crachás ou *tags* de identificação, e há a necessidade de referenciar os dados quando entregues ao servidor, pois a identificação lida via RFID não corresponde à identificação previamente utilizada na empresa.

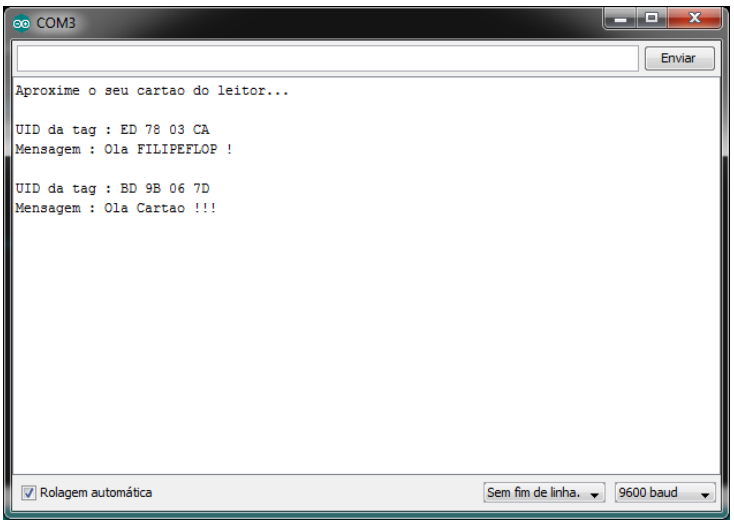

Figura 47 – Dados lidos via RFID. [36].

Outra maneira de agilizar o processo de apontamento é através de um leitor de *QR Code* capaz de automaticamente obter dados referentes ao processo de produção de um produto, como ordem de produção e operação que será realizada. A implementação de um leitor de *QR Code* utilizando o microcontrolador Arduino UNO não é recomendada, pois o mesmo não possui o poder de processamento necessário para que seja feita de maneira eficiente, porém pode ser feita via um leitor de terceiros, visto que os mesmos enviam os dados via comunicação USB serial. O envio de dados entre um dispositivo desse tipo e o microcontrolador Arduino não pode ser realizada pela porta USB padrão do Arduino, pois a mesma comunica-se somente como dispositivo USB, e não como *host*. A configuração do Arduino em modo *host* pode ser implementada através de um Shield USB Host e, assim, pode ser realizado o envio de dados entre um leitor de QR Code terceirizado e o microcontrolador.

Há também melhorias possíveis para utilização do sistema como estação de apontamento. Futuramente podem-se implementar funcionalidades extras, como a habilidade de editar e excluir apontamento previamente enviados. Um módulo *data logger* poderia ser utilizado para armazenar informações sobre os apontamentos realizados pelo contador em um cartão SD, permitindo, caso houvessem, verificar discrepâncias com os dados enviados para o servidor.

A própria PCB também apresenta possíveis mudanças. A reorganização de componentes, fazendo uso do espaço acima do microcontrolador Arduino, a tornaria mais compacta, Apesar que diminuir as dimensões da placa não alterariam seu preço, tais alterações diminuiriam a quantidade de material para confecção da *case* para o sistema.

Melhorias na parte de software incluem testes e *feedback* dos operadores para tornar o funcionamento mais intuitivo. Além disso é possível tornar o código do sistema mais eficiente, devido às otimizações para espaço, ao invés de eficiência, realizadas pelo autor, como comentado na Seção 3.3.3.4.

Outras melhorias incluem o uso de componentes mais caros e precisos, como um teclado mecânico para a entrada de dados ao invés do teclado de membrana utilizado, um ADC mais preciso, ou um *display* com capacidade para apresentar mais dados para o operador. Tais componentes somente não foram considerados para este projeto por aumentarem os custos do mesmo.

5.6.2 Melhorias na rede e no envio de dados

Melhorias na rede criada incluem a implementação de mecanismos de controle e correção de erros, com o intuito de otimizá-la o máximo possível. É possível implementar estações de *relay* com antenas externas para aumentar significantemente o alcance da rede de compartilhamento de dados. No envio de dados ao servidor, criação de rotinas para verificação da entrega de pacotes são essenciais.

O módulo nRF24L01+ possui um pino, de código `IRQ`, que possui suporte à interrupções. Através da utilização desse pino é possível alertar o microcontrolador sobre a presença de pacotes a serem lidos. Assim pode-se usar a configuração de *power down* do módulo, sendo que seu consumo é de $900nA$, segundo seu *datasheet*. Tal configuração é muito útil quando deseja-se criar um sistema embarcado com limitações de consumo de energia, tais como sistemas que são ligados à pilhas ou baterias, o que acaba não sendo o foco do projeto, porém pode eventualmente ser implementado. Tal mudança requer um novo *layout* da *PCB*, pois o *layout* atual, apesar de possuir um encaixe para o pino `IRQ`, não leva conexões até o mesmo.

6 Conclusões

Foi descrito neste trabalho o desenvolvimento e implementação de um sistema automatizado para auxiliar operadores em uma linha de produção manual através da contagem automática de unidades produzidas. O sistema permite que operadores possam apontar suas ordens de produção diretamente de suas estações de trabalho, sendo que todos os dados são enviados por uma rede sem fio, também desenvolvida ao longo do trabalho.

Cada módulo contador possui uma plataforma de pesagem para medir com precisão lotes de centenas de produtos, auxiliando operadores na realização de suas tarefas sem disrupção do processo fabril atual. Para isso, o módulo faz uso de algoritmos para identificação e atualização automática de peso unitário dos produtos contados, identificação de adição ou subtração de produtos sobre a plataforma, além de permitir a calibragem por parte do operador.

A rede para envio de dados sem fio opera contendo um Nó central, atuando como um *hub* de comunicações. Nós contadores são livres para conectar, desconectar, enviar a receber dados quando quiserem, inclusive encaminhando dados para Nós distantes, além do alcance do *hub*. A rede é segura e auto suficiente, não influenciando em outras redes presentes no ambiente industrial, como Wi-Fi, e sendo mantida em funcionamento pelos próprios Nós contadores e *hub*.

Dados enviados pela rede sem fio chegam ao *hub* para serem encaminhados ao servidor. Através do uso de pacotes UDP, uma tecnologia amplamente utilizada e regulamentada, são enviados pacotes e confirmações para e do servidor. O uso de protocolos de comunicação adotados amplamente permite uma configuração mínima por parte do servidor, garantindo também a possibilidade de interoperabilidade com próximas versões do próprio servidor e do sistema proposto. Como Nós contadores não conectam-se à rede Wi-Fi, o próprio *hub* os notifica do sucesso ou fracasso de envio de dados ao servidor.

A precisão da contagem de produtos encontrou-se satisfatória ao longo do desenvolvimento. Conseguiu-se medir com baixa taxa de erros a adição e subtração de produtos sobre a plataforma de pesagem para intervalos bastante próximos ao tempo de assentamento do sistema. Além disso, o mesmo é configurável via software, podendo também atender à processos de maior velocidade ao custo de uma menor precisão. O sistema também lida bem com múltiplos produtos sendo pesados simultaneamente, podendo medir até 5 produtos em uma única detecção sem que fossem constatados erros. Além disso, sua estabilidade e consistência para medidas realizadas ao longo de grandes intervalos de tempo se mostrou bastante satisfatória, tendo atendido com sucesso aos requisitos estabelecidos na seção 3.1.

De maneira similar, a rede para compartilhamento de dados atendeu à todos os pré-

requisitos estabelecidos. Foram constatadas pouquíssimas perdas de pacote durante sua utilização, baixa latência, e alta confiabilidade dos dados enviados. O encaminhamento de pacotes mostrou-se suficiente para enviá-los em longas distâncias, e testes demonstraram que o alcance mínimo das unidades de RF é satisfatório. A distribuição de endereços e identificadores de maneira dinâmica conferem dinamismo à rede e impedem que problemas em Nós individuais prejudiquem todo o sistema. A configuração do *hub*, conexão na rede Wi-Fi, e formatação e envio de dados via Wi-Fi são todos realizados de forma rápida e intuitiva, e as rotinas criadas para confirmação de envio e entrega de pacotes mostram-se bastante eficientes em notificar operadores de erros.

Ao longo do desenvolvimento foram realizadas várias mudanças e *upgrades* no sistema, com o intuito de deixá-lo sempre mais eficiente e intuitivo. Há várias possibilidades de mudanças para uma nova atualização futura, mantendo-se sempre em mente que processos automatizados podem ser desenvolvidos para auxiliar operadores, ao invés de substituí-los.

Referências Bibliográficas

1 GROOVER, M. P. *Fundamentals of Modern Manufacturing: Materials, Processes, and Systems*. 5. ed. [S.l.]: Wiley, 2012.

2 GUARNIERI, M. *The Roots of Automation Before Mechatronics [Historical]*. 2010. 42-43 p.

3 ANCIENT Discoveries. [S.l.]: History Channel, 2009. (45 min.), color. Episódio 10.

4 THE WORLD BANK. *The Changing Nature of Work*. <http://documents1.worldbank.org/curated/en/816281518818814423/pdf/2019-WDR-Report.pdf>, 2019. Acesso em: 11 mar. 2021.

5 ROBOTICS, I. F. of. *IFR presents World Robotics Report 2020*. 2020. <https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe>.

6 BRETON, R.; BOSSÉ, É. The cognitive costs and benefits of automation. In: WARSAW, POLAND. *The Role of Humans in Intelligent and Automated Systems*. [S.l.], 2003. p. 12.

7 BRASIL, G. *Índice de Automação do Mercado Brasileiro e de Consumidores*. [S.l.], 2020.

8 RUSSEL, S. J.; NORVIG, P. *Artificial Intelligence: a modern approach*. Englewood Cliffs, New Jersey: Prentice Hall, 1995.

9 RESEARCH, T.; STRATEGY, K. M. D.; BLOCK, P. I. *Global polypropylene market outlook*. [S.l.], 2017.

10 GLOBAL Polypropylene Market Report 2020 - Rising Demand for Thermoplastic Materials. <https://finance.yahoo.com/news/global-polypropylene-market-report-2020-092354027.html>.

11 MAIER, C.; CALAFUT, T. *Polypropylene: The Definitive User's Guide and Databook*. Elsevier Science, 2008. (PDL handbook series). ISBN 9780080950419. Disponível em: <https://books.google.com.br/books?id=AWaSJd9Non8C>.

12 TéCNICAS, A. B. de N. *NBR14136: Plugues e tomadas para uso doméstico e análogo até 20 A/250 V em corrente alternada - Padronização*. Rio de Janeiro, 2012.

13 TéCNICAS, A. B. de N. *NBR60335: Segurança de aparelhos eletrodomésticos e similares. Parte 1: Requisitos gerais*. Rio de Janeiro, 2010.

14 NILSSON, J. W.; RIEDEL, S. A. *Circuitos Elétricos*. 8. ed. São Paulo: Pearson Prentice Hall, 2009. 574 p.

15 ALEXANDER, C. K.; SADIKU, M. N. O. *Fundamentos de Circuitos Elétricos*. 5. ed. Porto Alegre: McGraw-Hill, 2013.

16 HX711 Datasheet. <https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf>.

17 HOW to Choose and Ethernet Cable | Digital Trends. <https://www.digitaltrends.com/computing/different-types-of-ethernet-cables-explained/>. Acesso em: 11 abr. 2021.

18 LOADSENSOR Datasheet. <https://www.sparkfun.com/datasheets/Sensors/loadsensor.pdf>.

19 HD74HC155 Datasheet. <https://pdf1.alldatasheet.com/datasheet-pdf/view/63834/HITACHI/74HC155.html>.

20 1602A-1 Datasheet. <https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf>.

21 ESP8266EX Datasheet. <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>.

22 74HC04 Datasheet. <https://www.alldatasheet.com/datasheet-pdf/pdf/15523/PHILIPS/74HC04.html>.

23 NRF24L01+ Datasheet. <https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf>.

24 BECKWITH, T. G.; BUCK, N. L.; MORANGONI, R. D. *Mechanical measurements*. 3. ed. Reading, Massachusetts: Addison-Wesley, 1982.

25 EMBEDDED with Elliot: Debounce Your Noisy Buttons, Part I. <https://hackaday.com/2015/12/09/embed-with-elliot-debounce-your-noisy-buttons-part-i/>.

26 DIGITAL Pins | Arduino. <https://www.arduino.cc/en/Tutorial/Foundations/DigitalPins>. Acesso em: 06 abr. 2021.

27 BASICS of Serial Peripheral Interface (SPI). <https://www.electronicshub.org/basics-serial-peripheral-interface-spi/>.

28 ATTACHINTERRUPT() - Arduino Reference. <https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/>. Acesso em: 30 abr. 2021.

29 ASCII Table - ASCII character codes and html, octal, hex, and decimal. <http://www.asciitable.com/>.

30 SOSINSKY, B. *Networking Bible*. Indianapolis, Indiana: Wiley, 2009. 890 p. ISBN 9780470431313.

31 WIKIMEDIA Commons. <https://commons.wikimedia.org/>.

32 JSON. <https://www.json.org/json-en.html>.

33 GLOBAL Wi-Fi Enabled Devices Shipment Forecast, 2020-2024. <https://www.researchandmarkets.com/reports/5135535/global-wi-fi-enabled-devices-shipment-forecast>.

34 OVERVIEW - AutoConnect for ESP8266/ESP32. <https://hieromon.github.io/AutoConnect/index.html>.

35 ARDUINO Fast digitalWrite - The Robotics Back-End. <https://roboticsbackend.com/arduino-fast-digitalwrite>.

36 CONTROLE de Acesso usando Leitor RFID com Arduino - FilipeFlop. <https://www.filipeflop.com/blog/controle-acesso-leitor-rfid-arduino/>.