Detecção de Falhas em Sistemas Ciberfísicos: Um Estudo Aplicado em Baterias de Robôs Móveis

Matheus Hiroyuki Donadi Ikeda - 52483

Dissertação apresentada à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do grau de Mestre em Engenharia Industrial.

Trabalho orientado por:

Prof. Dr. Paulo Leitão

Prof. Dr. Flávio Luiz Rossini

Bragança

2023

Detecção de Falhas em Sistemas Ciberfísicos: Um Estudo Aplicado em Baterias de Robôs Móveis

Matheus Hiroyuki Donadi Ikeda - 52483

Dissertação apresentada à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do grau de Mestre em Engenharia Industrial.

Trabalho orientado por:

Prof. Dr. Paulo Leitão

Prof. Dr. Flávio Luiz Rossini

Bragança

2023

Dedicatória

Aos meus amados familiares, com um carinho especial à minha querida mãe Leoni e avó Leonilda. Seu amor, dedicação e apoio incondicionais têm sido uma fonte constante de inspiração ao longo da minha vida. Vocês são exemplos de força, sabedoria e bondade, que me orientam e encorajam a ser a melhor versão de mim mesmo. Amo vocês além das palavras.

Agradecimentos

Agradeço primeiramente a Deus por me guiar ao longo desta jornada.

Agradeço ao meu orientador Paulo Leitão, pelo seu tempo, conhecimento e orientação ao longo do desenvolvimento deste trabalho. Suas orientações valiosas e feedbacks construtivos foram essenciais para moldar e aprimorar minhas ideias.

Agradeço também ao pesquisador e doutorando Luis Piardi que contribuiu para a realização deste trabalho. Sua experiência, conhecimento e dedicação foram fundamentais.

Aos meus familiares, meu amor e gratidão são infinitos, vocês têm sido uma fonte constante de apoio e amor ao longo desta trajetória. Seu encorajamento, paciência e compreensão foram essenciais para o meu crescimento.

Não posso deixar de agradecer aos meus amigos, pela amizade sincera, pelo apoio mútuo e pela energia positiva que compartilhamos. Vocês foram uma fonte constante de motivação e inspiração.

Meu sincero agradecimento a todos os mencionados e a todos aqueles que, de alguma forma, estiveram presentes em minha jornada. Seu apoio e dedicação são imensamente apreciados.

Resumo

Esta dissertação tem como objetivo abordar o desafio das falhas em Sistemas Ciberfísicos (CPSs), que são sistemas integrados e combinam componentes físicos e computacionais. Os CPSs são essenciais em várias áreas, erros nesses sistemas podem causar problemas significativos, como interrupções de serviços, perda de dados ou até mesmo ameaças à segurança. Com base nesse contexto, o objetivo deste estudo foi desenvolver um protótipo para detectar falhas em um CPS laboratorial. Especificamente, o foco foi direcionado para a detecção de falhas em baterias de Polímero de Lítio (Li-Po), o mesmo tipo de bateria utilizado nos robôs móveis do CPS laboratorial.

A estrutura implementada para o protótipo seguiu duas abordagens, a primeira é baseada em uma estrutura central no sistema Robot Operating System (ROS) que recebe todas as informações dos robôs móveis em um único nó ROS e a outra é baseada em uma estrutura local, onde cada robô móvel possui um nó ROS dedicado. Para enfrentar esse desafio, foram aplicadas técnicas de Aprendizado de Máquina (ML) do tipo supervisionado e não supervisionado, visando comparar qual tipo desempenharia melhor. Para a estrutura central, os modelos de ML foram treinados utilizando dados generalizados, já para a estrutura local, os modelos de ML foram treinados utilizando dados específicos de cada robô. Essas abordagens permitem que o sistema aprenda a reconhecer padrões e associações nos dados coletados, possibilitando a detecção automática de falhas.

Foram coletados dados em relação ao comportamento das baterias de Li-Po por meio de experimentos em laboratório, simulando duas condições de operação, a primeira é relacionada à descarga da bateria quando o robô do CPS se encontra parado e a segunda é quando o robô está em movimento. Esses dados foram preparados para treinar e validar

os modelos de ML. Os modelos de ML treinados foram introduzidos nos protótipos de estrutura central e local, o desempenho deles foi avaliado utilizando dados fornecidos por um ambiente de simulação no CoppeliaSim, onde foi simulado dois tipos de falhas que podem ocorrer nas baterias de Li-Po.

O desempenho dos modelos de ML aplicados e dos protótipos de detecção de falha foi medido através de métricas como acurácia e F1-score. Os resultados obtidos demonstraram que os modelos de ML supervisionados desempenharam melhor do que os não supervisionados para esta abordagem, assim como o protótipo com estrutura local se sobressaiu em relação ao protótipo com estrutura central. Em especial o modelo de aprendizagem de máquina Floresta Aleatória (RF) utilizado no protótipo de estrutura local, sendo capaz de detectar as falhas nas baterias com uma acurácia de 98,8% e F1-score(0) (relacionado a classe dos valores consistentes) de 1, e F1-score(1) relacionado a classe dos valores inconsistentes) de aproximadamente 0,68. Isso indica que o protótipo com estrutura local utilizando o modelo de aprendizagem de máquina RF pode ser efetivamente aplicado para identificar problemas em baterias de Li-Po e, potencialmente, em outros componentes do CPS.

Em conclusão, este estudo contribuiu para a área de detecção de falhas em CPSs. Os métodos utilizados baseados em modelos de ML supervisionados demonstraram ser eficientes na detecção de falhas em baterias de Li-Po em um ambiente laboratorial.

Palavras-chave: Sistemas Ciberfísicos, Detecção e Diagnóstico de Falhas, Aprendizado de Máquina.

Abstract

This dissertation aims to address the challenge of failures in Cyber-Physical Systems (CPSs), which are integrated systems that combine physical and computational components. CPSs are essential in various areas, and errors in these systems can cause significant problems such as service interruptions, data loss, or even security threats. Based on this context, the objective of this study was to develop a prototype for detecting failures in a laboratory CPS. Specifically, the focus was on detecting failures in Lithium Polymer (Li-Po) batteries, the same type of battery used in the mobile robots of the laboratory CPS.

The implemented structure for the prototype followed two approaches. The first approach is based on a centralized framework in the Robot Operating System (ROS), which receives all the information from the mobile robots in a single ROS node. The second approach is based on a local framework, where each mobile robot has a dedicated ROS node. To address this challenge, supervised and unsupervised Machine Learning (ML) techniques were applied to compare which type would perform better. For the centralized structure, the ML models were trained using generalized data, while for the local structure, the ML models were trained using specific data from each robot. These approaches allow the system to learn to recognize patterns and associations in the collected data, enabling automatic fault detection.

Data regarding the behavior of Li-Po batteries were collected through laboratory experiments, simulating two operating conditions. The first condition is related to battery discharge when the CPS robot is stationary, and the second is when the robot is in motion. These data were prepared to train and validate the ML models. The trained ML models

were introduced into the prototypes of the centralized and local structures, and their performance was evaluated using data provided by a simulation environment in CoppeliaSim, where two types of failures that can occur in Li-Po batteries were simulated.

The performance of the applied ML models and fault detection prototypes was measured using metrics such as accuracy and F1-Score. The results demonstrated that the supervised ML models performed better than the unsupervised models for this approach, and the local structure prototype outperformed the centralized structure prototype. In particular, the Random Forest (RF) machine learning model used in the local structure prototype was able to detect battery failures with an accuracy of 98.8%, an F1-Score(0) (related to the class of consistent values) of 1, and an F1-Score(1) (related to the class of inconsistent values) of approximately 0.68. This indicates that the local structure prototype using the RF machine learning model can effectively be applied to identify problems in Li-Po batteries and potentially in other components of CPSs.

In conclusion, this study contributed to the field of CPS fault detection. The methods used, based on supervised ML models, proved to be efficient in detecting failures in Li-Po batteries in a laboratory environment.

Keywords: Cyber-Physical Systems, Fault Detection and Diagnosis, Machine Learning.

Conteúdo

Lista de Tabelas

Lista de Figuras

Acrônimos

ABOD Detecção de Outliers Baseada em Ângulo.

AR Realidade Aumentada.

CPS Sistema Ciberfísico.

DT Árvore de Decisão.

ESC Curto-Circuito Externo.

I4.0 Indústria 4.0.

IA Inteligência Artificial.

IF Floresta de Isolamento.

IoS Internet de Serviços.

IoT Internet das Coisas.

ISC Curto-Circuito Interno.

KNN K-Vizinhos Mais Próximos.

Li-Po Polímero de Lítio.

LIB Bateria de Íon de Lítio.

LOF Fator Atípico Local.

m Coeficiente Angular.

M2M Máquina a Máquina.

MCD Determinante de Covariância Mínima.

ML Aprendizado de Máquina.

NMR Redundância N-Modular.

OCSVM Máquina de Vetores de Suporte de Uma Classe.

RF Floresta Aleatória.

ROS Robot Operating System.

SOC Estado de Carga.

SVM Máquina de Vetores de Suporte.

TMR Redundância Modular Tripla.

V-REP Virtual Robot Experimentation Platform.

VR Realidade Virtual.

WSN Redes de Sensores Sem Fio.

Capítulo 1

Introdução

Com a introdução da Indústria 4.0 (I4.0), o uso de tecnologias digitais em ambientes industriais como Sistemas Ciberfísicos (CPSs), tornou-se fundamental para a digitalização e o aumento da produtividade. Esses sistemas são caracterizados por representarem um modelo complexo, integrado e interconectado, constituídos de componentes físicos, capazes de se conectar a um ou vários programas de computador e que possuem duas camadas distintas, conhecidas como camada cibernética e física. Cada camada tem uma função dentro do sistema, a camada física consiste nos objetos físicos que possuem a capacidade de interagir com o ambiente, já a camada cibernética está relacionada com a comunicação entre os dispositivos e o sistema, juntamente com o processamento computacional [1]. Estes sistemas podem resultar em sistemas autônomos, basicamente, um sistema autônomo decide de forma independente o que realizar e quando realizar com base na análise dos dados recebidos. Existem vários níveis de autonomia usados em diferentes aplicações, mas geralmente os sistemas autônomos são usados em áreas críticas onde os seres humanos não podem chegar, onde há risco de vida, em atividades longas e repetitivas ou em atividades que requerem um tempo de reação rápido [2].

A preocupação dos projetistas em relação à criação de sistemas mais robustos em relação a falhas tem sido estudada e aplicada mesmo antes da concepção da I4.0, principalmente para sistemas críticos, como sondas espaciais, aviões e controles industriais em

tempo real [3]. À medida que o nível de tecnologia implementada nos processos de fabricação e máquinas evolui, principalmente para resolver problemas mais complexos, também há uma grande necessidade de empregar técnicas mais eficazes e eficientes para monitorar as condições das máquinas em tempo real, como os CPSs. As condições das máquinas podem incluir: a detecção do início, progressão e propagação de falhas, erros, defeitos, bem como a tomada de decisões corretas para que essas irregularidades não resultem em falhas, interrupções e tempos de inatividade prejudiciais [4].

A utilização de abordagens orientadas a análise de dados se tornou comum, isso pois, com o desenvolvimento da I4.0, a quantidade de diversos recursos de dados obtidos por meio da Internet das Coisas (IoT) aumentou significativamente [5]. Neste sentido, houve e há a necessidade de obter informações úteis a partir de dados. Este tipo de abordagem está totalmente interligada com o conceito de aprendizado de máquina, visto que existe uma dependência por parte do ML em relação a análise de dados, pois, o treinamento de modelos de ML requer uma preparação adequada desses dados.

Métodos de ML em conjunto com análise de dados são comumente utilizados para a detecção de falhas, alguns tipos de modelagem preditiva utilizados com este objetivo incluem o de classificação, regressão, redes neurais dentre outros. A escolha do método de ML depende do tipo de dados disponíveis, da natureza das falhas que se pretende detectar e do contexto específico do problema. A análise de dados também é fundamental para a preparação dos dados, a escolha dos recursos adequados e a avaliação do desempenho dos modelos de ML para detecção de falhas.

Este trabalho aborda aspectos gerais relacionados a falhas, aplicações atuais e benefícios relacionadas à análise de dados no contexto de detecção de falhas em CPSs, funcionamento de algoritmos de ML e a construção de um protótipo de detecção de falhas em um ambiente de CPS de pequena escala.

1.1 Objetivos

Esta dissertação de mestrado possui objetivos relacionados com técnicas de detecção de falhas em CPSs.

- Levantamento das aplicações e benefícios ao utilizar abordagens orientadas a análise de dados para detectar falhas em CPSs;
- Comparação entre modelos de ML tanto do tipo supervisionado quanto tipo não supervisionado aplicado a detecção de falhas em CPSs;
- Desenvolvimento de um protótipo de detecção de falhas em um CPS laboratorial utilizando modelos de ML. O CPS laboratorial consiste em um armazém logístico em pequena escala que inclui múltiplos robôs móveis. Dois tipos de abordagens serão utilizadas, a primeira utilizará uma estrutura centralizada contendo informações de todos os robôs do CPS e a segunda utilizará uma estrutura local, onde cada robô possuirá uma estrutura responsável por processar e detectar falhas localmente, ou seja, de forma individual.
- Comparação dos dois tipos de estruturas utilizadas, central e local.

1.2 Estrutura do Trabalho

Esta dissertação está organizada em 6 capítulos, que inicialmente descreve o contexto, o estado da arte e fundamentação teórica a respeito do tema, o método e ferramentas utilizadas, o desenvolvimento e implementação do protótipo de detecção de falhas em um CPS de pequena escala, os testes feitos, os resultados apresentados e, por fim, a conclusão da dissertação.

No Capítulo 2 intitulado “Estado da Arte”, aspectos gerais de falhas são abordados, além de conceitos e algoritmos de ML e análise de dados.

No Capítulo 3 “Abordagens e Ferramentas Utilizadas”, são demonstradas as ferramentas e abordagens utilizadas para a criação do protótipo de detecção de falhas.

No Capítulo 4 “Desenvolvimento e Implementação”, são exibidos e explicados as etapas utilizadas para o desenvolvimento do protótipo.

No Capítulo 5 “Testes e Resultados”, são demonstrados e explicados os testes feitos tanto nos modelos de ML testados quanto nos protótipos de detecção de falhas desenvolvido, apresentando as métricas obtidas nestes testes.

No Capítulo 6 “Conclusão”, são discutidas as conclusões a cerca do tema, dos modelos de ML que melhor se adaptaram ao protótipo e qual estrutura do protótipo demonstrou melhor desempenho.

Capítulo 2

Estado da Arte

2.1 Indústria 4.0

Com o passar dos anos e consequentemente com as revoluções industriais, novos métodos foram utilizados para conseguir suprir a necessidade encontrada em cada época. A primeira revolução industrial foi marcada pelo uso de água e vapor para a mecanização de processos e produção, a segunda utilizou da energia elétrica para o desenvolvimento da produção em larga escala e a terceira revolução industrial é relacionada com o uso da tecnologia da informação e eletrônica para a automatização da produção [6]. Mesmo com todas essas tecnologias utilizadas na terceira revolução industrial, a humanidade conseguiu atingir patamares muito maiores de tecnologia e informação nos anos seguintes, onde o conceito de I4.0 surgiu pela primeira vez em 2011 a partir de um projeto de tecnologia do governo alemão, e o termo "Industria 4.0" foi apresentado publicamente na Feira de Hannover [7].

O estudo de Sacomano et al. (2018) [8] propõe uma classificação (não definitiva) dos elementos que compõe a I4.0, entre eles estão: elementos base ou fundamentais, elementos estruturantes e elementos complementares. Os elementos base ou fundamentais retratam a base tecnológica na qual a I4.0 está relacionada, como os CPSs, IoT e Internet de Serviços (IoS). Elementos estruturantes são conceitos ou tecnologias que devem estar presentes

em sua maioria em uma indústria para que a mesma possa ser considerada 4.0, são eles: análise de Big Data, automação, comunicação Máquina a Máquina (M2M), computação em nuvem, integração de sistemas, Inteligência Artificial (IA), cibersegurança. Elementos complementares são componentes que expandem as possibilidades da I4.0, porém, diferentemente dos elementos estruturantes, as unidades de produção que utilizam em sua maioria os elementos complementares não necessariamente se tornam 4.0, estes elementos são: QR Code, Realidade Aumentada (AR), etiquetas de RFID e Realidade Virtual (VR) [8].

Para Schwab (2019) [9] as inovações apresentadas não seriam apenas aspectos da terceira revolução industrial, mas sim uma ocorrência de uma nova revolução. As três razões que Schwab (2019) [9] cita para sustentar este pensamento são: a velocidade, amplitude e profundidade e o impacto sistêmico. Primeiramente a velocidade que está relacionada com o crescimento exponencial desta revolução. A amplitude, relacionada a vasta gama de tecnologias empregadas, e a profundidade que está ligada não somente em modificar como os processos são efetuados mas também modificar quem somos diante das novas tecnologias. O impacto sistêmico é a ideia que envolve a modificação e transformação de sistemas inteiros em toda a sociedade [9].

> "A indústria 4.0 assenta-se na integração de tecnologias de informação e comunicação que permitem alcançar novos patamares de produtividade, flexibilidade, qualidade e gerenciamento, possibilitando a geração de novas estratégias e modelos de negócio para a indústria, sendo por isso, considerada a Quarta Revolução Industrial ou o Quarto Paradigma de Produção Industrial ([8], p. 28)."

2.2 Sistemas Ciberfísicos

O termo "Sistema Ciberfísico" (CPS) surgiu por volta de 2006, citado primeiramente por Helen Gil na Fundação Nacional de Ciência nos Estados Unidos, tendo como base a "cibernética", termo criado por Norbert Wiener durante a Segunda Guerra Mundial, o

mesmo descreve cibernética como o conjunto de controle e comunicação [10]. A definição de CPSs é discutida por alguns autores da área. Sacomano et al. (2018) [8] afirma que:

> "Os CPSs são sistemas mecatrônicos compostos por sensores e atuadores, controlados por software que, monitorando uma série de dados, supervisionam e controlam processos industriais no campo físico ([8] p. 34)."

Já para Liu et al. (2017) [11]:

> "CPSs são sistemas multidisciplinares para realizar controle de *feedback* em sistemas de computação embarcada amplamente distribuídos pela combinação de tecnologias de computação, comunicação e controle. Eles são a transformação e integração dos sistemas de rede existentes e sistemas embarcados tradicionais ([11], p.27)."

É possível notar que em ambas definições de CPSs, o controle dos sistemas através da obtenção de dados de sensores é fundamental, onde os dados da camada física são passados para a camada digital, que é onde ocorre o processamento dos dados e consequentemente os sinais de controle são enviados para o sistema físico.

O estudo de Liu et al. (2017) [11] analisa o CPS como a integração de sistemas diferentes que desempenham diversas funções, onde não há um modelo unificado e as pesquisas que abrangem o tema de sistemas ciberfísicos são realizadas sob a perspectiva de aplicações no campo de interesse daquela própria pesquisa e que os modelos de CPSs devem ser modificados e integrados com base no sistema físico existente, sistema de rede e estrutura do sistema de computador.

O modelo proposto por Liu et al. (2017) [11] pode ser dividido em três camadas. Camada de usuário, que executa a estratégia e proteção em ambiente de interação humano-computador e também faz a consulta de dados. Camada de sistemas de informação, que faz a transmissão e processamento de todos os dados que foram coletados no meio físico através de sensores ou outros dispositivos. Por último a Camada de Sistema Físico, composta por vários dispositivos, como sistemas embarcados e sensores, estes fazem a

coleta e transmissão das informações e também são encarregados do controle por meio dos sinais gerados. É possível perceber que o modelo proposto tem a seguinte linha de execução: o sistema físico envia as informações obtidas no meio físico para a camada de sistemas de informação, que processa estes dados e, de acordo com os resultados obtidos, envia sinais de controle para a camada de sistema físico, executando assim os comandos de controle.

O trabalho de Pires (2020) [12] descreve os CPSs como uma nova terminologia que representa a integração da computação e das capacidades físicas, citando algumas áreas de aplicação, como: controle de processos, controle de tráfego, sistemas automatizados avançados e estruturas inteligentes. Também comenta sobre a importância do CPS ser confiável e seguro em relação ao tempo crítico, pois, o tempo de resposta entre a camada virtual e física do CPS deve ser imediata e objetiva para não prejudicar as tomadas de decisões dentro do sistema.

Tendo a segurança de um CPS como elemento crítico, o mesmo deve se mostrar fiável. Pires (2020) [12] lista alguns aspectos em que a fiabilidade ou confiabilidade está englobada: manutenção, disponibilidade, segurança física, segurança lógica, restrições de tempo real. Manutenção está ligada a probabilidade da solução da falha do sistema ser realizada em um determinado período de tempo. Disponibilidade é a probabilidade do sistema estar disponível, para isso, a fiabilidade e facilidade de manutenção devem estar em primeiro lugar, para que o sistema esteja disponível a maior parte do tempo possível. Segurança física é uma propriedade do sistema relacionada a segurança do meio físico ou do ambiente em que o sistema está envolvido em relação à não causar nenhum tipo de dano a este espaço. Segurança lógica é uma propriedade ligada à confidencialidade dos dados e a garantia de comunicação. Restrições em tempo real diz respeito ao intervalo de tempo em que as informações serão processadas, dependendo do tempo em que isso ocorra, pode influenciar de forma negativa na segurança e confiabilidade do sistema [12].

2.3 Aspectos Gerais Sobre Falhas

Com a utilização cada vez maior de sistemas computadorizados, a busca pela segurança e confiabilidade do mesmo se torna imprescindível para que o usuário tenha uma boa experiência. Weber (2003) [3] apresenta alguns conceitos de falhas, erros e defeitos.

- Defeito: Anomalias nos parâmetros do sistema de computação;
- Falha: Origem física ou algorítmica da falha;
- Erro: Se a etapa subsequente ao estado presente resultar em uma falha.

Na Figura 2.1 é possível entender melhor estes conceitos e em qual camada cada um deles se encontra.

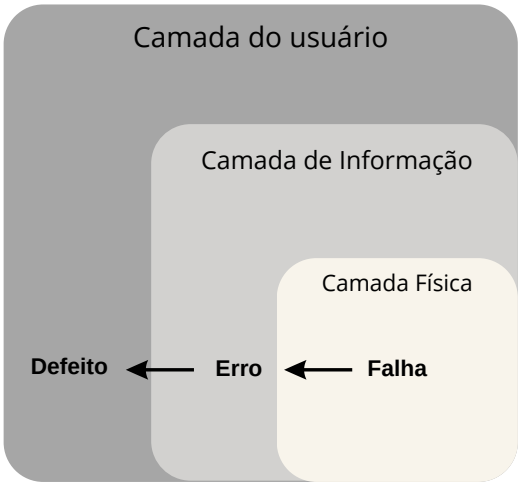


Figura 2.1: Falha, Erro e Defeito. Adaptado de [3].

Alguns conceitos como latência de falha e latência de erro também são definidas em Weber (2003) [3]:

- Latência de falha: Intervalo de tempo entre a ocorrência da falha e a manifestação do erro;

- Latência de erro: Tempo decorrido desde a ocorrência do erro até a manifestação do defeito.

Além da causa da falha, são considerados alguns conceitos para definir a mesma, como a natureza, duração, extensão e valor. Dentro dos tipos de falhas, as falhas de software e de projeto são consideradas um problema crítico dentro da computação. Alcançar a dependabilidade é fundamental para tolerância a falhas, a dependabilidade é caracterizada como a competência do atendimento oferecido por um determinado sistema [3].

Para alcançar a dependabilidade é necessário utilizar algumas técnicas e métodos, classificados na Tabela 2.1:

Métodos e Técnicas	**Descrição dos métodos e técnicas**
Prevenção de falhas	Evita que ocorram ou sejam introduzidas falhas. Engloba a escolha de abordagens de projeto e tecnologias apropriadas para os elementos envolvidos.
Tolerância a falhas	Garante a entrega do serviço desejado mesmo diante de possíveis falhas. Técnicas comuns: mascaramento de falhas, detecção de falhas, localização e outras.
Validação	Eliminação de defeitos, validação da existência de falhas.
Previsão de falhas	Avaliações da ocorrência de defeitos e avaliações dos impactos causados por falhas.

Tabela 2.1: Técnicas e Métodos. Adaptado de [3].

Existem algumas técnicas de tolerância a falhas que são utilizadas quando é necessário ter alta confiabilidade, as mesmas são baseadas em redundância, são elas:

- Mascaramento: As falhas não são detectadas, localizadas ou reconfiguradas, somente são mascaradas na origem (mais redundante);

- Detecção, localização e reconfiguração: Funcionamento oposto ao mascaramento, falhas são detectadas, localizadas e reconfiguradas.

Redundância está relacionada diretamente com tolerância a falhas e pode ser encontrada de diversas maneiras, cada tipo com sua própria técnica e vantagem [3], tais como:

- Redundância de hardware: Baseada na replicação de componentes, pode ser passiva ou estática (mascaramento de falhas), ativa ou dinâmica (detecção, localização e recuperação) e híbrida;
- Redundância de software: Utilização de programação em n-versões, blocos de recuperação e verificação de consistência;
- Redundância de informação: Fornecida através de códigos de correção de erro;
- Redundância de tempo: Repete a computação no tempo, evitando o custo de *hardware*, porém, aumentando o tempo de computação.

2.3.1 Redundância de Hardware Passiva

Na redundância de hardware passiva, componentes são replicados com o objetivo de mascarar possíveis falhas que possam ocorrer.

Redundância N-Modular

Uma técnica bastante conhecida por replicar o hardware em N módulos paralelos é a Redundância N-Modular (NMR). Esta técnica consiste na replicação do hardware em paralelo, onde a saídas dos N módulos são encaminhadas para um votador, que define a saída do sistema como sendo a maioria dos votos [13].

Na Figura 2.2 é possível entender melhor esta técnica.

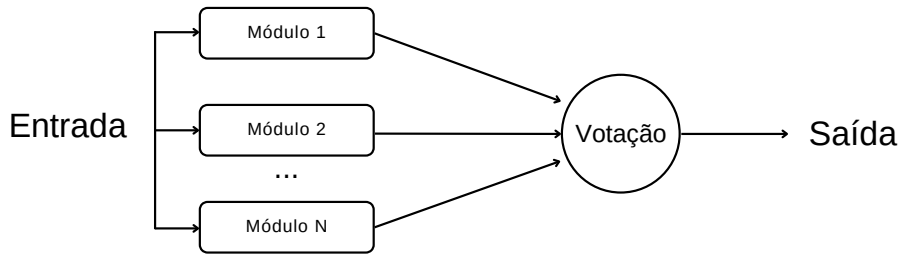


Figura 2.2: Técnica NMR. Adaptado de [14].

- Vantagem: Não suscetível a erros gerados por apenas um módulo;
- Desvantagem: Vulnerável quando há falha na maioria dos módulos, gerando uma saída equivocada; Ponto crítico de falha no sistema votação, pois, apresenta somente um componente.

Redundância Modular Tripla

Uma solução para aumentar a confiabilidade do sistema de votação seria replicá-lo na mesma ordem N em que o NMR foi implementado, por exemplo, para um sistema Redundância Modular Tripla (TMR), teríamos três módulos votadores ao invés de um módulo votador, e assim passando a ter três saídas que serão encaminhadas para os módulos seguintes do sistema [3]. A Figura 2.3 demonstra esta técnica.

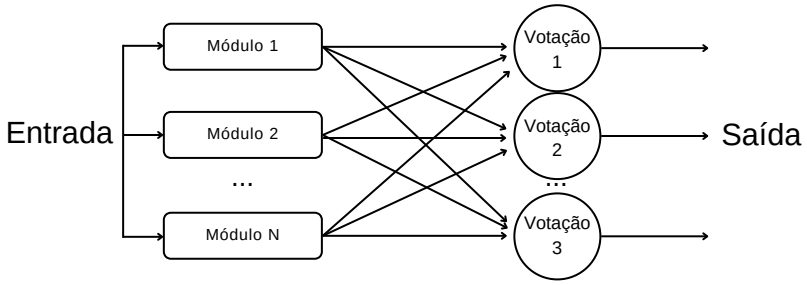


Figura 2.3: Técnica NMR com Multiplicação de Votadores.

Seleção de Valor Médio

Uma implementação utilizada do NMR é o TMR, como o nome já diz, é a implementação com três módulos e utilizando um sistema de votação [15]. A saída pode ser gerada pela maioria dos votos, ou, quando as saídas geradas pelos módulos são valores que podem variar muito, utiliza-se a técnica de Seleção de Valor Médio. A Figura 2.4 demonstra esta técnica.

Na seleção de valor médio é realizada a média das saídas dos módulos, o sistema votador é substituído por um componente que escolhe a saída em que o valor seja o mais

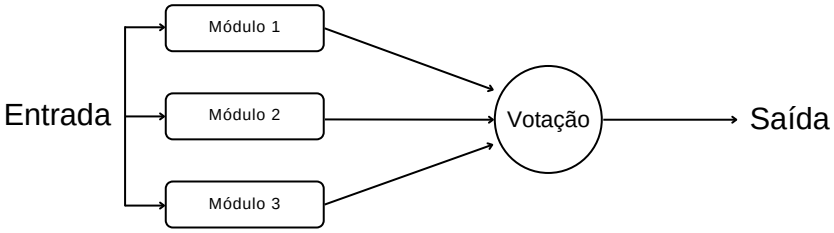


Figura 2.4: Técnica TMR. Adaptado de [16].

próximo do valor médio obtido, podendo estabelecer limites para as saídas de modo com que módulos defeituosos não interfiram negativamente na média final. O valor escolhido deve estar dentre os sinais não corrompidos, esta técnica pode ser aplicada a diferentes tipos de sistema, tendo que atender apenas à um requisito, o sistema deve utilizar um número ímpar de módulos [17]. A Figura 2.5 demonstra esta técnica.

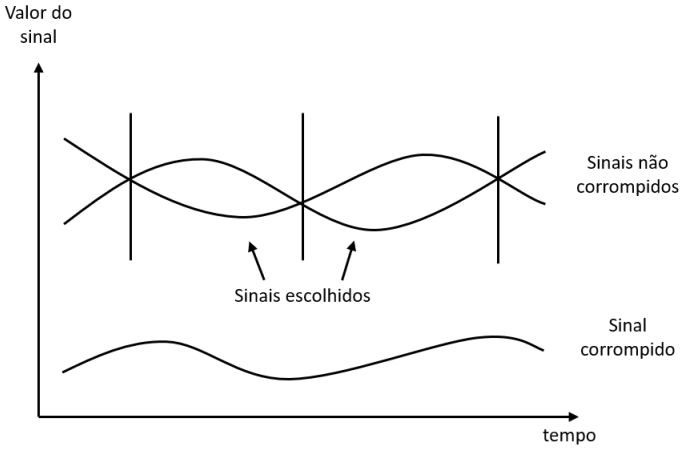


Figura 2.5: Técnica Seleção de Valor Médio. Adaptado de [18].

A seleção de valor médio geralmente é utilizada quando se tem a necessidade de utilizar sensores analógicos ou para sistemas de conversão analógico digital [19].

Flux Summing

A implementação da técnica de *Flux Summing* assim como a técnica TMR, utiliza módulos redundantes em que as saídas dos mesmos são enviadas para um transformador, porém,

a saída deste transformador é proporcional à soma das saídas do módulos. A saída do transformador é encaminhada novamente para a entrada dos módulos redundantes, ou seja, nesta técnica existe uma realimentação que proporciona a compensação no sinal de saída em caso de falha de algum dos módulos [13].

Portanto, não se trata de um sistema de votação mas continua tendo o efeito do mascaramento de falhas. Pode ser utilizado nas técnicas NMR em geral [17]. O funcionamento está demonstrado na Figura 2.6.

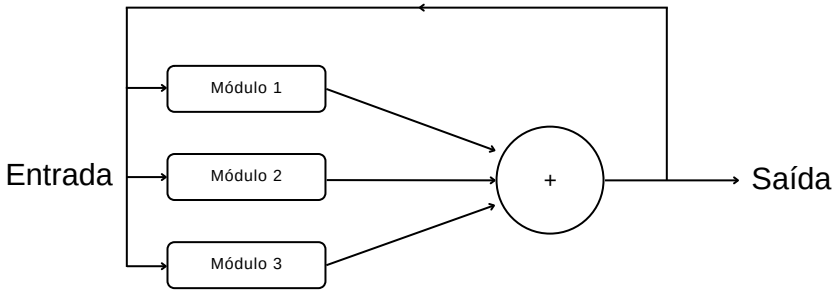


Figura 2.6: Técnica Flux Summing. Adaptado de [19].

2.3.2 Redundância de Hardware Ativa

Diferentemente da técnica de redundância de hardware passiva, que tem o objetivo de mascarar o erro, a redundância de hardware ativa utiliza as técnicas de detecção, localização e recuperação [3].

Técnica Standby Sparing

A utilização de módulos estepe ou *standby sparing* é um exemplo de implementação da redundância ativa, a mesma pode ser classificada em dois tipos diferentes, *hot standby* e *cold standby*.

O funcionamento da operação quando está em *hot standby* ocorre da seguinte maneira, os módulos redundantes que estão em *standby* operam paralelamente com o módulo principal, caso ocorra alguma falha com o mesmo, o módulo em *standby* está pronto para ser

utilizado. O fato dos módulos redundantes em *standby* estarem sempre ligados faz com que este modo de operação gaste mais energia que o *cold standby*, porém, com um tempo de recuperação menor. Em *cold standby* é necessário um tempo de recuperação maior, pois, diferentemente do hot standby, neste modo os módulos redundantes em *standby* não são alimentados todo o tempo, e só começam a funcionar quando são conectados. Pelo fato dos módulos não estarem sempre ligados, a vida útil dos componentes aumenta e o gasto de energia é menor [17].

2.3.3 Redundância de Software

Na redundância de *hardware* a replicação dos módulos era uma estratégia para que, em caso de um deles apresentassem falhas, o outro estaria apto a amenizar ou corrigir esta falha, dependendo do tipo de redundância. Quando o assunto é redundância de *software*, a replicação de um mesmo código ou algoritmo sempre terá a mesma saída, ou seja, se o sistema está apresentando falhas, estas falhas serão as mesmas, independente de quantas vezes o componente de *software* for replicado.

Para a redundância de *software* é preciso utilizar outras estratégias para se obter o resultado necessário, algumas destas técnicas são: programação em n-versões, *n-self-cheking programming* e blocos de recuperação.

Programação em N-Versões

Nesta técnica de redundância de *software* são utilizados programas desenvolvidos por N equipes diferentes e sem contato uma com a outra, todos os programas a serem desenvolvidos, necessariamente devem ter as mesmas especificações. Considerando que as equipes estão separadas, os programas desenvolvidos por cada uma delas tendem a ser diferentes em implementação, aumentado a diversidade do sistema, porém, não necessariamente aumentando a confiabilidade, já que mesmo separadas, os membros das equipes podem trocar algoritmos, vir a adotar a mesma ideia para o desenvolvimento ou a buscar ajuda em uma mesma fonte de conhecimento. Os programas desenvolvidos são executados em

módulos paralelos e as saídas assim como na técnica de redundância de hardware NMR, passam por um sistema de votação, que pode detectar erros de programação. Estes erros detectados devem demonstrar comportamentos diferentes uns dos outros por terem sido desenvolvidos de forma independente [17]. A Figura 2.7 demonstra esta técnica.

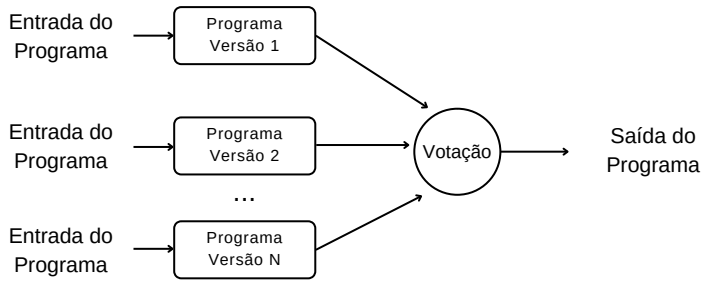


Figura 2.7: Programação em N-Versões. Adaptado de [17].

N-Self-Checking Programming

Existe também uma vertente estendida desta técnica, chamada *N-Self-Checking Programming*, a ideia é basicamente a mesma da programação em N-versões, porém, os programas não necessariamente são criados por pessoas diferentes, eles apenas devem ser uma versão única de um determinado programa, também há a adição de um módulo de verificação, que é responsável por verificar se a lógica do programa está correta, essa verificação é feita através de uma lista de testes que o programa deve passar. Os programas são executados em módulos paralelos assim como na programação de N-versões, e a saída selecionada neste método deve ter passado pelo próprio módulo de verificação. Esta técnica é semelhante à técnica de *hardware hot stadby sparing* [17]. A Figura 2.8 demonstra esta técnica.

Blocos de Recuperação

A técnica de blocos de recuperação também é uma extensão à programação em N-versões, mais especificadamente a programação *N-Self-Checking* pois necessitam passar por testes

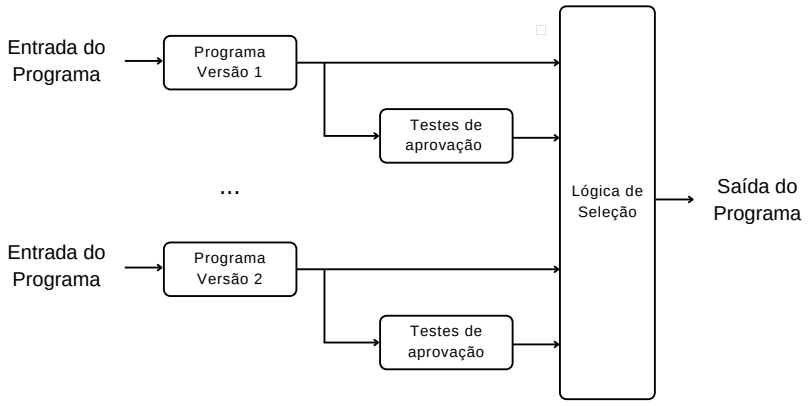


Figura 2.8: Programação N-Self-Checking. Adaptado de [17].

de aceitação. Apesar dos módulos estarem em paralelo, apenas um estará sendo executado por vez, o próximo só será executado na sequência se o anterior apresentar algum erro. Os programas serão executados até que um deles passe pelo módulo de verificação de corretude, uma falha no sistema ocorrerá se nenhum dos softwares passarem pela verificação. A aproximação desta técnica é semelhante a técnica de *hardware cold stadby sparing* e pode tolerar N-1 falhas [17]. A Figura 2.9 mostra a ideia desta técnica.

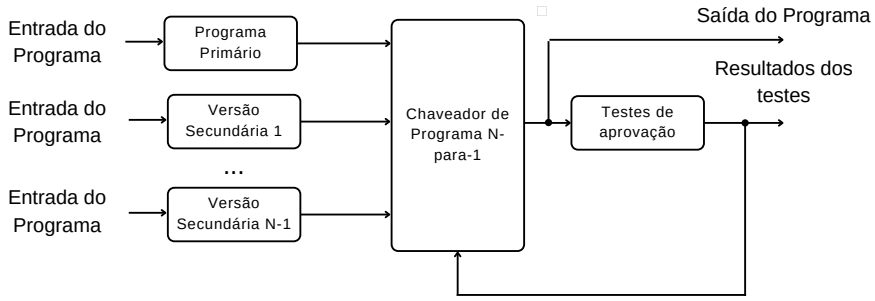


Figura 2.9: Blocos de Recuperação. Adaptado de [17].

2.3.4 Detecção e Diagnóstico de Falhas

Com a evolução e popularização das plataformas tecnológicas multifunções ou sistemas inteligentes, a utilização de técnicas de modelagem ou até mesmo de métodos baseados em inteligência computacional se tornaram populares, pois, houve a necessidade de superar as dificuldades encontradas nos métodos tradicionais utilizados em sistemas mais complexos [20].

A detecção de falhas é entendida como a constatação da presença de uma falha no sistema em um determinado período de tempo [21] [22] [23], através da observação e análise do comportamento do sistema ou de alguma variável, é possível perceber se o padrão de comportamento ou a variável está alterada. Outra maneira de realizar a detecção de falhas é analisar a diferença entre o valor da variável recebida e o valor estimulado por um modelo matemático, conhecida como detecção de falhas com redundância analítica [24]. Também é possível utilizar a redundância de *hardware* para detectar falhas, como visto anteriormente, a técnica NMR utiliza módulos extras para evitar falhas, a medição feita por este(s) módulo(s) redundantes pode ser utilizada para fins de comparação com o valor medido da variável.

O diagnóstico de falhas é a etapa posterior à detecção de falhas, onde é feita a caracterização das falhas detectadas, que de forma geral, caracterizam a localização e a causa da falha [24]. O tipo, magnitude, momento da falha e comportamento com o tempo também podem ser determinados [21] [22] [23] . É possível dividir o diagnóstico de falhas em duas etapas, a primeira relacionada ao isolamento, determinando o tipo, localização e instante de tempo em que a falha ocorreu [24]. A segunda etapa é relacionada com a identificação da falha, definindo a magnitude, o motivo para ocorrer a falha (causa) e comportamento com o tempo [25] [21] [22]. Algumas técnicas de diagnóstico de falhas baseadas em sistemas de regras e em árvores de decisão receberam uma maior aceitação na indústria [26].

Sistemas de regras se baseavam na experiência de técnicos, onde eram criadas regras baseadas nos sintomas e, a partir destes sintomas eram definidas as falhas. Árvores

de decisão de falhas também se baseia nos sintomas apresentados no sistema, podendo também decidir por qual "ramo" ou caminho seguir dependendo do resultado de um determinado teste, cada "ramo" é constituído por vários tipos de ações, como realizar novos testes e até reparos [20].

Como citado anteriormente, essas técnicas tradicionais de diagnóstico de falhas foram substituídas por técnicas mais avançadas no momento em que tais técnicas se tornaram inviáveis em sistemas mais complexos. Algumas abordagens podem utilizar a Lógica Fuzzy para o diagnóstico de falhas, ela tem como objetivo modelar ou representar conceitos aproximados ao invés de precisos, como é feito na lógica binária ou clássica. Uma das utilizações da Lógica Fuzzy é justamente em casos de sistemas mais complexos, na geração de modelos para estes sistemas e também na análise de resíduos, o que não era viável em técnicas tradicionais [20].

Em sistemas mais complexos como os CPSs o aumento do número de robôs instalados na linha de produção pode gerar um aumento nos casos de falhas, causando efeitos negativos como a perda de tempo e dinheiro [27]. De acordo com Piardi et al. (2020) [27], as investigações acerca das políticas de tolerância a falhas industriais em CPSs têm gerado propostas altamente eficazes que se fundamentam em tecnologias como IoT, IA e segurança cibernética, podendo ser uma alternativa em relação à detecção e diagnóstico de falhas. Normalmente, os métodos de tolerância a falhas, que levam em conta tecnologias como IoT e IA, são aplicados a processos e máquinas seguindo uma abordagem centralizada [28]. Embora essas tecnologias tenham evoluído gradualmente para suportar os Sistemas CiberFísicos Industriais (ICPS), as arquiteturas e metodologias relacionadas à detecção e diagnóstico de falhas não acompanharam essa evolução de forma sincronizada [1]. Sofrendo então, uma carência por parte das arquiteturas e metodologias.

2.3.5 Sistemas Tolerantes a Falhas

Com o avanço tecnológico, as operações industriais estão se tornando cada vez mais eficientes. Novas tecnologias, como automação, IA e IoT, estão sendo aplicadas para otimizar

processos, possibilitando a coleta de dados em tempo real sobre a planta industrial e a análise dessa grande quantidade de dados, podendo identificar padrões e prever falhas ou defeitos. No entanto, para garantir não apenas eficiência, mas também segurança, a pesquisa sobre sistemas tolerantes a falhas tem se intensificado cada vez mais. Esses sistemas desempenham um papel fundamental ao garantir a continuidade dos processos de forma segura. Os sistemas de tolerância a falhas visam identificar, analisar, isolar e restaurar todas as funcionalidades de um sistema diante da ocorrência de uma anomalia [27].

O estudo feito por Furquim (2017) [29] propõe e analisa uma abordagem de um sistema tolerante a falhas para detecção e previsão de eventos naturais adversos. O sistema é capaz de prever a possibilidade de falha na comunicação e a perda de nós durante o acontecimento do desastre natural, fazendo a distribuição de dados mesmo em casos críticos. Este sistema se mostrou funcional, emitindo alertas em tempo hábil para auxiliar na tomada de decisões em caso de desastres naturais, ele é baseado em IoT, Redes de Sensores Sem Fio (WSN) e ML.

A dissertação de Okada (2022) [30] implementa diversas abordagens de detecção e diagnóstico de falhas baseadas em diferentes tipos de filtros de Kalman e um controle tolerante para um quadricóptero. No estudo, foram abordadas distintas anomalias caracterizadas por comportamentos não lineares e a potencialidade de ocorrência simultânea nos dispositivos de controle e nas unidades de sensoriamento. Os resultados mostraram uma melhoria na segurança especialmente na utilização do filtro adaptativo.

2.4 Aprendizado de Máquina

Aprendizado de Máquina (*'Machine learning'*) (ML) é um ramo da IA e uma técnica que melhora o desempenho de sistemas aprendendo com as experiências ou dados, e utilizando métodos computacionais para isto. Assim como os humanos, os computadores também conseguem aprender através de experiências, no caso dos computadores, estas experiências são apresentadas na forma de dados. O principal objetivo do ML é desenvolver algoritmos de aprendizado, quando estes forem supridos com dados de experiências será possível obter

um modelo que poderá fazer predições em novas amostras [31].

O conjunto de dados utilizados constitui o que é chamado de *dataset*, no *dataset* são registradas as descrições de um evento ou objeto, a fruta maçã será utilizada como um exemplo. Dentro dessas descrições existem os atributos ou características do evento ou objeto, como por exemplo, cor, textura, formato e tamanho da maçã, os valores que são atribuídos a estas características (por exemplo, verde, vermelha, redonda, macia) são chamados de valores de atributos. A Figura 2.10 demonstra o exemplo de um conjunto de dados de maçãs.

Conjunto de dados da fruta maçã

ID	Cor	Formato	Tamanho (onças)	Maduro
1	Vermelha	Arredondado	13	verdadeiro
2	Amarela	Oblongo	8.4	falso
3	Vermelha	Oblato	10.5	verdadeiro

Figura 2.10: Exemplo de um Conjunto de Dados de Maçãs. Adaptado de [31].

O processo para criar os modelos através dos dados dos *dataset* utilizando ML é chamado de aprendizado (*'learning'*) ou treinamento (*'training'*). Nem sempre somente dados como, por exemplo, a cor, formato e tamanho são suficientes para determinar se a maçã está pronta para o consumo ou não. Portanto, para o modelo de predição ser efetivo, o resultado (neste caso, madura ou não) também deve estar disponível no *dataset*. Após o modelo ser treinado, é possível fazer a predição da saída desejada, o processo de fazer predições é chamado de *'testing'* [31].

Quando a saída a ser prevista é do tipo discreta, o estado de estar madura ou não, é chamado de problema de classificação. Quando a saída é contínua, é chamado de problema de regressão. Se a saída tiver apenas dois possíveis resultados, é chamado problema de

classificação binário. E se a saída tiver mais de duas classes, se trata de um problema de classificação de multiclasse [31].

Existem diversas abordagens algorítmicas para solucionar problemas de dados, e os cientistas de dados costumam destacar que não há uma solução única que consiga resolver todos os problemas. O tipo de algoritmo que melhor se adéqua dependerá de diversas variáveis, como por exemplo, o tipo de problema que é desejado resolver, quantas características e qual o tamanho do *dataset* a ser utilizado [32].

2.4.1 Algoritmos de ML

A técnica de ML pode ser ramificada em diferentes tipos de algoritmos, alguns mais comumente utilizados que outros, como por exemplo: aprendizagem supervisionada (*'Supervised Learning'*), aprendizagem não supervisionada (*'Unsupervised Learning'*), aprendizagem semi-supervisionada (*'Semi-supervised Learning'*), aprendizado conjunto (*'Ensemble Learning'*), redes neurais (*'Neural Network'*), aprendizagem baseada em instâncias (*'Instance Based Learning'*).

Aprendizagem Supervisionada

A aprendizagem supervisionada é uma tarefa crucial no campo de aprendizado de máquina, que consiste em construir um modelo capaz de mapear uma entrada para uma saída com base em exemplos previamente rotulados. Esses modelos dependem de assistência externa para aprender e melhorar seu desempenho ao longo do tempo [32].

Para isso, os dados de entrada são separados em conjuntos de treinamento e teste, sendo que o primeiro contém exemplos com variáveis de saída conhecidas que precisam ser previstas ou classificadas. Durante o processo de treinamento, os algoritmos de aprendizado de máquina supervisionados aprendem a identificar padrões nos dados de treinamento e, posteriormente, aplicam esses padrões para fazer previsões ou classificações em novos dados de teste.

Dentro do conceito de aprendizagem supervisionada, é possível citar alguns algoritmos

como o de Árvore de Decisão (DT), Naïve Bayes e Máquina de Vetores de Suporte (SVM). A Figura 2.11 demonstra um exemplo da aprendizagem supervisionada.

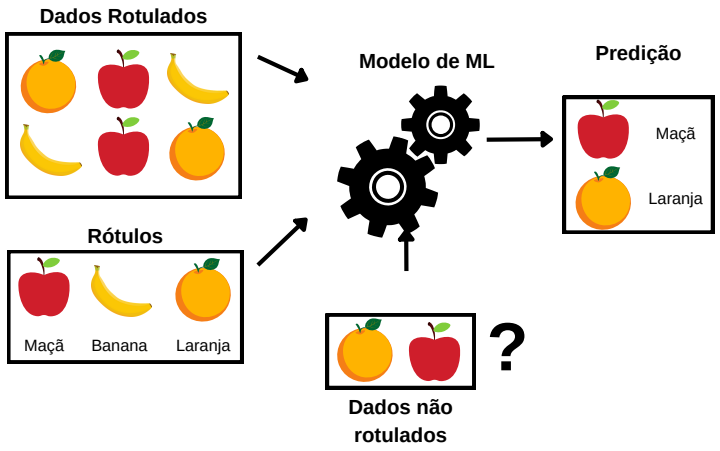


Figura 2.11: Exemplo da Aprendizagem Supervisionada. Adaptado de [33].

Árvore de Decisão

A Árvore de Decisão (*'Decision Tree'*) (DT) é uma representação visual que ajuda a entender as escolhas e suas respectivas consequências e um algoritmo de ML. Cada nó na árvore representa um evento ou opção a ser tomada, enquanto que as arestas da árvore correspondem às regras ou condições de decisão. A árvore é composta por nós e ramos, os nós são responsáveis por representar os atributos de um grupo que devem ser classificados e os ramos indicam um valor que pode ser atribuído ao nó. A Figura 2.12 mostra a árvore de decisão.

Máquina de Vetores de Suporte

Assim como a DT, outra técnica de ML bastante utilizada é a de Máquina de Vetores de Suporte (*'Support Vector Machine'*) (SVM), SVMs são modelos de aprendizado de máquina supervisionado que utilizam algoritmos de aprendizado para examinar dados e

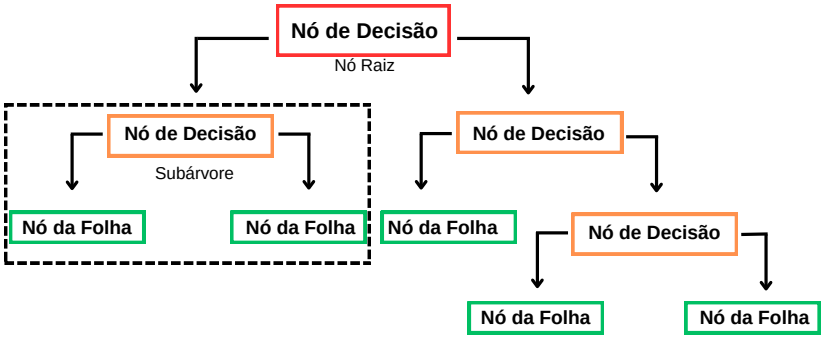


Figura 2.12: Árvore de Decisão. Adaptado de [34].

realizar análise de classificação, regressão e detecção de *outliers* (pontos de dados inconsistentes) [32]. Os dois principais objetivos do SVM são: utilizar a função Kernel, conhecido também como "truque de kernel" e encontrar a melhor classificação do hiperplano que separa os dados [35].

Os SVMs realizam a classificação linear e utilizando a função Kernel, é possível realizar a classificação não linear que ajuda a transformar as amostras de entrada para um espaço de dimensão superior, podendo assim, ser classificadas linearmente [32].

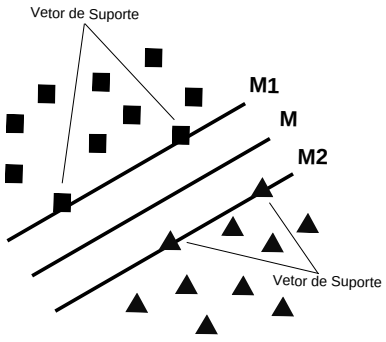


Figura 2.13: Ideia do SVM. Adaptado de [36].

Os quadrados e triângulos na Figura 2.13 representam os dois tipos de amostras, 'M' é a linha de classificação e 'M1' e 'M2' são linhas paralelas a 'M', que passam pelas amostras mais próximas à linha de classificação, essas amostras são chamadas de vetores

de suporte [37]. A Figura 2.14 demonstra a ideia da função Kernel.

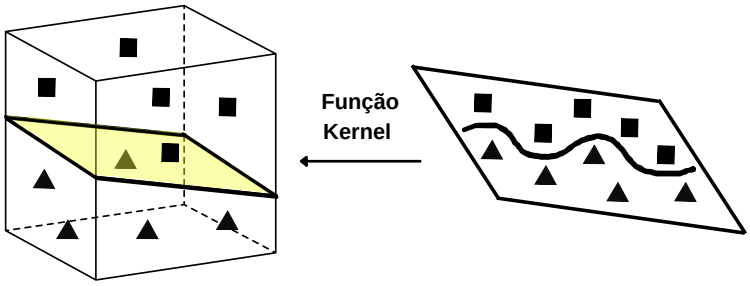


Figura 2.14: Função Kernel Transformando as Amostras para um Espaço de Dimensão Superior. Adaptado de [36].

Dentro da aprendizagem supervisionada existe no geral dois tipos de SVMs, o SVM de duas classes e o SVM de multiclasses. A metodologia de SVM de duas classes é a mais utilizada [37]. Além da classificação, o SVM também pode ser empregado para análise de regressão [38] [39].

Aprendizagem Não Supervisionada

Diferentemente do aprendizado supervisionado, no aprendizado não supervisionado não existem exemplos com variáveis de saída conhecidas que necessitam ser previstas ou classificadas, ao invés disso, o algoritmo apresenta uma estrutura na qual os dados estão dispostos, ele aprende limitadas particularidades sobre os dados, particularidades estas que serão utilizadas para reconhecer e agrupar novos dados. São utilizados principalmente para agrupamento e redução de características [32]. A Figura 2.15 demonstra um exemplo da aprendizagem não supervisionada.

Agrupamento K-Means

Dentro do âmbito do aprendizado não supervisionado, o Agrupamento K-means (*'K-Means Clustering'*) é um dos mais simples e mais populares algoritmos que resolvem um problema conhecido de agrupamento [32] [41].

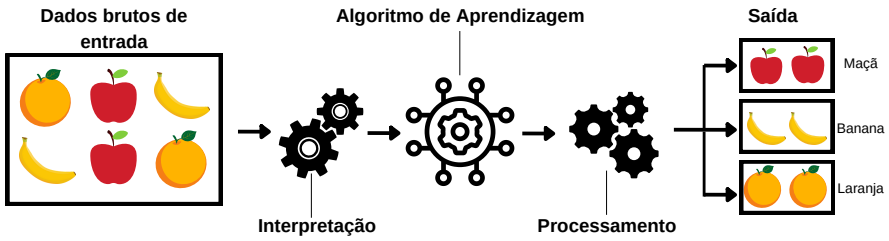


Figura 2.15: Exemplo da Aprendizagem Não Supervisionada. Adaptado de [40].

O principal objetivo do K-Means é definir K centros, sendo um para cada grupo ou *'cluster'*, os centros devem ser colocados o mais afastado possível uns dos outros para que os grupos fiquem bem definidos, após a definição dos centros, as amostras do conjunto de dados escolhido serão associadas ao centro mais próximo, quando todas já estiverem associadas, novos K centros serão recalculados levando em conta o centro real dos grupos formados anteriormente [32]. O Agrupamento K-Means tem sido estudado e aplicado em diversas áreas: [42] [43] [44] [45]. A Figura 2.16 demonstra o funcionamento do K-Means.

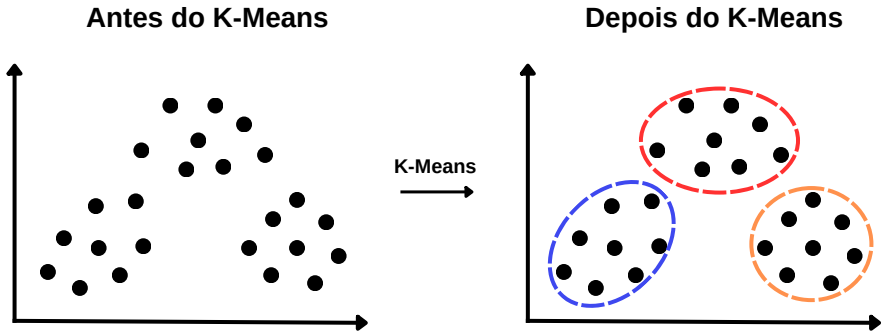


Figura 2.16: Exemplo do Agrupamento K-Means. Adaptado de [46].

Aprendizagem Semi-Supervisionada

Como o próprio nome já indica, a aprendizagem semi-supervisionada utiliza métodos tanto do aprendizado supervisionado quanto do aprendizado não supervisionado. Este

ramo utiliza dados rotulados e não rotulados no aprendizado, permitindo assim utilizar um conjunto grande de dados não rotulados juntamente com um conjunto menor de dados rotulados [47]. Essa abordagem é particularmente útil em casos em que obter dados rotulados é difícil ou dispendioso, permitindo aproveitar ao máximo os dados disponíveis. A Figura 2.17 demonstra um exemplo da aprendizagem semi-supervisionada.

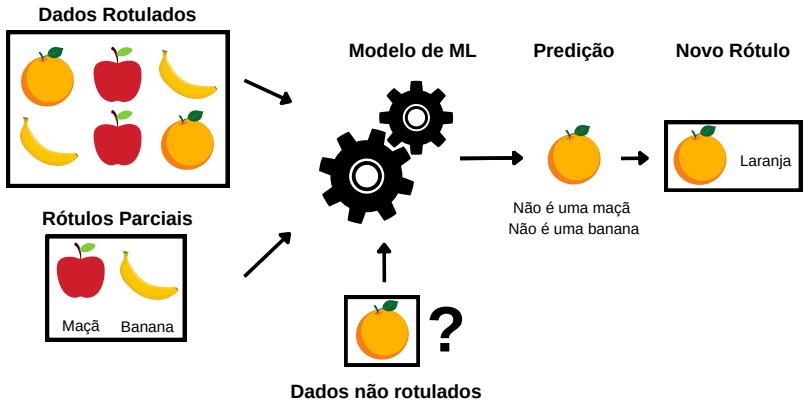


Figura 2.17: Exemplo da Aprendizagem Semi-Supervisionada. Adaptado de [33].

Self Training

No *Self-training* um modelo de classificação é inicialmente treinado com uma fração dos dados rotulados, e posteriormente, são inseridos dados não rotulados. Os pontos não rotulados são classificados usando o modelo, e as etiquetas previstas são adicionadas ao conjunto de dados rotulados. Para que o modelo consiga classificar com melhor desempenho, o processo é repetido múltiplas vezes [32].

2.4.2 Aprendizado em Conjunto

O aprendizado em conjunto (*'Ensemble Learning'*) é um conjunto de métodos pelo qual vários modelos (classificadores, regressores, etc.) são combinados de maneira estratégica para solucionar um problema específico [32]. Ao combinar vários modelos, o aprendizado em conjunto parte do princípio de que os erros de um único modelo serão compensados por

outros modelos. Dessa forma, espera-se que o desempenho de predição geral do conjunto seja melhor do que o de um único modelo [48].

Os métodos do aprendizado em conjunto podem ser divididos em duas estruturas: dependente e independente. Na estrutura dependente, a saída de cada algoritmo (aprendiz base) influencia na próxima saída que será gerada, ou seja, nesta estrutura, o algoritmo aprende a cada iteração. Já na estrutura independente, como o nome já diz, os aprendizes bases são construídos independentemente dos outros aprendizes [48]. Como exemplos de métodos de *ensemble*, é possível citar, *'bagging'*, *'boosting'*, *'stacking'* e *'blending'*, um algoritmo bastante conhecido que utiliza a técnica de *'bagging'* é o Floresta Aleatória (*'Random Forest'*) (RF). A Figura 2.18 demonstra um exemplo do aprendizado conjunto.

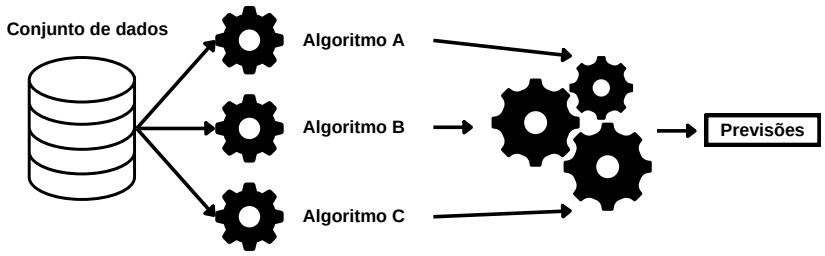


Figura 2.18: Aprendizado Conjunto. Adaptado de [49].

Floresta Aleatória

Um dos mais populares métodos de aprendizado em conjunto, se não for o mais popular, o método Floresta Aleatória (*'Random Forest'*) (RF) foi apresentado praticamente ao mesmo tempo por [50] e [51], e mais tarde popularizada e elaborada por [52]. As RFs são compostas por diversos preditores, cada um representado por uma árvore de decisão, cujos valores requerem um vetor aleatório obtido através de amostragem independente e com distribuição idêntica para todas as árvores presentes na floresta [52]. O treinamento da floresta aleatória é realizado utilizando o *'bagging'*, que melhora a classificação e regressão dos modelos conforme a estabilidade e a precisão da mesma. O algoritmo constrói sua decisão com base na contagem de votos dos preditores componentes em cada

classe e, posteriormente, a classe selecionada (vencedora) será a que possuir mais votos acumulados [53]. A Figura 2.19 demonstra a ideia do método Floresta Aleatória.

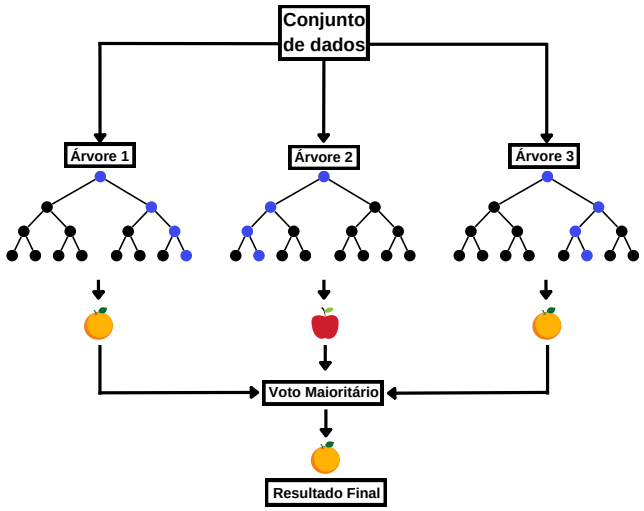


Figura 2.19: Floresta Aleatória. Adaptado de [54].

2.4.3 Outliers

O conceito de *outliers* é bastante utilizado no campo da estatística e análise de dados, os estudos previamente feitos na área de detecção de *outliers* utilizam técnicas baseadas em distância, densidade, distribuição, agrupamento e outras [55].

A técnica baseada em distância tem como proposta analisar a distância entre cada par de pontos de dados dentro de um conjunto de dados. Um algoritmo que utiliza este conceito é o K-Vizinhos Mais Próximo (*'K-Nearest-Neighbors'*) (KNN), onde pontos de dados que estão próximos de seus K vizinhos (onde K representa o número de vizinhos que estão sendo analisados) são considerados normais, e aqueles que estiverem relativamente distantes dos mesmos serão considerados *outliers* [56] [57].

A técnica baseada em agrupamento, agrupa os pontos de dados, pontos que estejam distantes de grupos maiores ou grupos pequenos de pontos de dados, são considerados *outliers*.

A técnica baseada em densidade analisa a densidade ao redor de cada ponto de dado dentro do conjunto de dados, uma comparação de densidade é feita entre os pontos. Pontos com alta densidade são considerados *inliers* (pontos de dados consistentes) e aqueles com baixa densidade são considerados *outliers*.

A técnica baseada em distribuição supõe que os pontos de dados normais seguirão uma distribuição pré definida como por exemplo a distribuição normal, serão considerados *outliers* pontos de dados que não seguirem esta distribuição [56].

Outliers podem ser definidos como dado por Hawkins [58]:

> "Um *outlier* é uma observação que se desvia tanto das outras observações a ponto de despertar suspeitas de que tenha sido gerado por um mecanismo diferente."

Detectar, diagnosticar e solucionar falhas e *outliers* em sistemas complexos como os CPSs utilizando abordagens de análise de dados tem vários usos e benefícios. Entre eles estão a detecção de anomalia, previsão de falhas e eficiência do sistema, redução de custos e melhoria da segurança. A análise de dados ajuda a identificar anomalias em tempo real nos CPSs para que ações imediatas possam ser tomadas para corrigir o problema [59]. A análise de dados pode também ajudar a prever falhas em CPSs, permitindo que ações sejam tomadas antes que ocorram falhas, assim, melhorando a eficiência do sistema [60], além de reduzir os custos de manutenção e reparo [61] e ajudar a melhorar a segurança dos CPSs.

A análise de dados é uma ferramenta poderosa para detectar, diagnosticar e corrigir erros do CPS. Ela pode antecipar possíveis problemas de desempenho do sistema, como a degradação de sensores e falhas de comunicação, podendo tomar ações em tempo real que sejam efetivas para garantir seu desempenho e segurança.

2.4.4 Modelos de ML Relacionados a Detecção de Outliers

Existem diversas abordagens e algoritmos de ML que podem ser aplicados com o objetivo de detecção de *outliers* dentro de um conjunto de dados. Os modelos utilizam diferentes

estratégias e procedimentos para identificar pontos de dados com características atípicas em relação ao restante dos dados, algumas dessas características podem estar relacionadas com a distância entre os pontos, densidade, ângulo, entre outros. Alguns modelos de ML amplamente utilizados para este propósito são: Floresta de Isolamento (IF), Fator Atípico Local (LOF), Máquina de Vetores de Suporte de Uma Classe (OCSVM), Detecção de Outliers Baseada em Ângulo (ABOD) e Determinante de Covariância Mínima (MCD).

Floresta de Isolamento

Floresta de Isolamento (*'Isolation Forest'*) (IF) é um algoritmo de aprendizagem não supervisionada baseado em árvores de decisão que isola os *outliers* de um certo conjunto de dados. Ele foi desenvolvido em 2008 pelos pesquisadores Fei Tony Liu, Kai Ming Ting e Zhi-Hua Zhou. O algoritmo funciona construindo aleatoriamente uma floresta de árvores de decisão, cada árvore é construída selecionando aleatoriamente um subconjunto das características e, em seguida, particionando recursivamente os dados usando essas características. A medida da anormalidade é baseada na facilidade com que uma amostra pode ser isolada a partir dos demais dados, ou seja, uma amostra isolada com poucas divisões, provavelmente será considerada uma anomalia [62].

Algumas características que dão vantagem ao IF sobre os outros algoritmos de detecção de anomalias são [62]:

- Rapidez: O IF tem um processamento muito rápido, especialmente para grandes *datasets*;
- Escalável: O IF pode ser facilmente escalável para grandes datasets;
- Robusto ao ruído: O IF é robusto ao ruído, isto ocorre pois, o algoritmo seleciona aleatoriamente as características que darão origem a cada árvore da floresta de árvores, diminuindo as chances de que as características irrelevantes ou ruidosas sejam escolhidas;

- Eficiente: O IF é bastante eficiente em termos de uso de memória mesmo tendo uma complexidade de tempo linear.

A Figura 2.20 demonstra a ideia deste algoritmo.

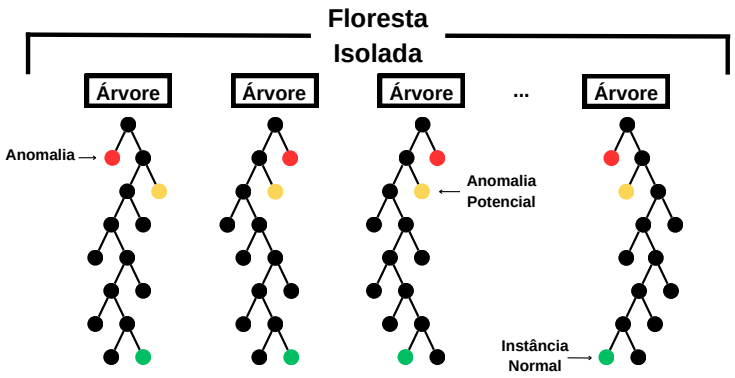


Figura 2.20: Floresta Isolada. Adaptado de [63].

Fator Atípico Local

Fator Atípico Local (*'Local Factor Outlier'*) (LOF) é um algoritmo de detecção de anomalias que identifica anomalias com base na densidade local de um ponto de dados em relação a seus vizinhos. Foi proposto por Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng e Jörg Sander em 2000.

O LOF é baseado na ideia de que *outliers* são pontos de dados que estão cercados por uma baixa densidade de outros pontos de dados. O algoritmo funciona calculando primeiro a densidade local de cada ponto de dados. A densidade local de um ponto de dados é o número de outros pontos de dados que estão a uma certa distância dele. Depois que a densidade local de cada ponto de dados é calculada, o LOF calcula a pontuação LOF para cada ponto de dados. A pontuação LOF de um ponto de dado é a razão de sua densidade local para a densidade local média de seus vizinhos. Pontos de dados com uma alta pontuação LOF são mais propensos a serem *outliers* [64]. A Figura 2.21 mostra a ideia deste algoritmo.

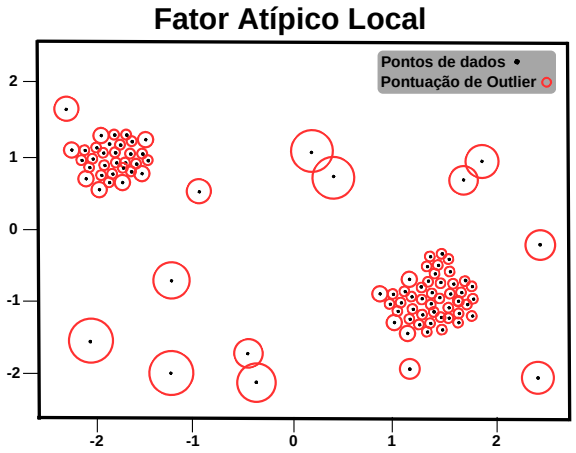


Figura 2.21: Fator Atípico Local. Adaptado de [65].

Máquina de Vetores de Suporte de Uma Classe

A Máquina de Vetores de Suporte de Uma Classe (*'One-Class Support Vector Machine'*) (OCSVM) é uma variante de SVMs utilizada para detectar *outliers*. A principal diferença do OCSVM em relação aos SVMs tradicionais explicados anteriormente, é que ele é treinado apenas em dados de uma única classe, ou seja, mesmo que os novos dados não sejam rotulados, este algoritmo é capaz de identificá-los, mesmo estes novos dados não pertencendo aos dados de treinamento [66].

Uma vantagem em relação a outros algoritmos é que o OCSVM pode ser utilizado mesmo com uma pequena quantidade de dados rotulados, em contrapartida, pode exigir um grande volume de recursos computacionais e a escolha errada do Kernel pode levar a uma piora no desempenho do modelo. A Figura 2.22 demonstra a ideia por trás do funcionamento deste algoritmo.

Detecção de Outliers Baseada em Ângulo

A ideia de utilizar ângulos para a detecção de *outliers* em alta dimensão foi introduzida por Knorr e Ng em 1998 [68] e em 2008 foi proposto o algoritmo de Detecção de Outliers Baseada em Ângulo (*'Angle-Based Outlier Detection'*) (ABOD) por Kriegel et al [69].

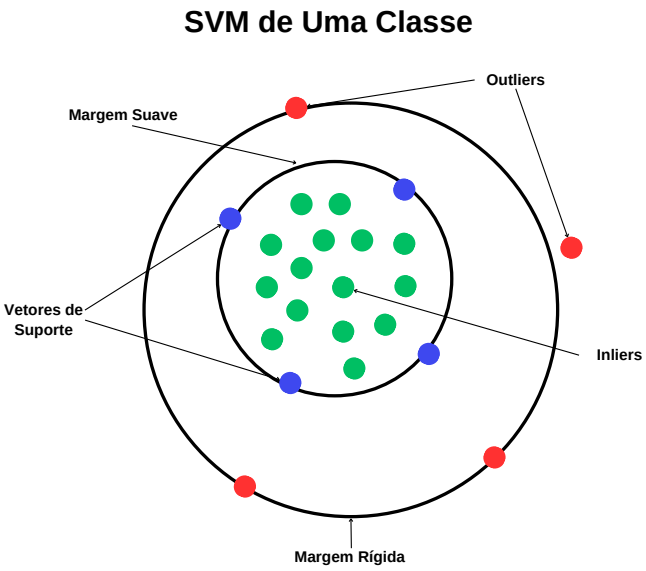


Figura 2.22: SVM de Uma Classe. Adaptado de [67].

Neste método, o ângulo é medido entre cada ponto do conjunto de dados e o centro de massa deste mesmo grupo de dados. São considerados *inliers* os pontos que apresentarem um ângulo pequeno com o centro de massa, e aqueles que apresentarem um ângulo maior em relação a este centro de massa são considerados *outliers*. Esta técnica apresenta robustez em relação a detecção de *outliers* e eficácia mesmo se tratando de dados de alta dimensão, isso uma vez que, os ângulos obtidos entre pontos do mesmo conjunto de dados são menos suscetíveis aos efeitos do ruído em relação as distâncias entre esses mesmos pontos. A Figura 2.23 demonstra a ideia do funcionamento deste método.

O funcionamento do Fator de Outlier Baseado em Ângulo (*'Angle-Based Outlier Factor'*) (ABOF) é similar ao funcionamento do ABOD, primeiro o algoritmo calcula o ângulo entre cada ponto de dados e o centro de massa do conjunto de dados, e posteriormente a isto, é feito o cálculo da variância dos ângulos. São considerados *'inliers'* os pontos de dados que apresentarem uma baixa variância de ângulos, enquanto os pontos de dados que possuem uma alta variância de ângulos são considerados *outliers* [69].

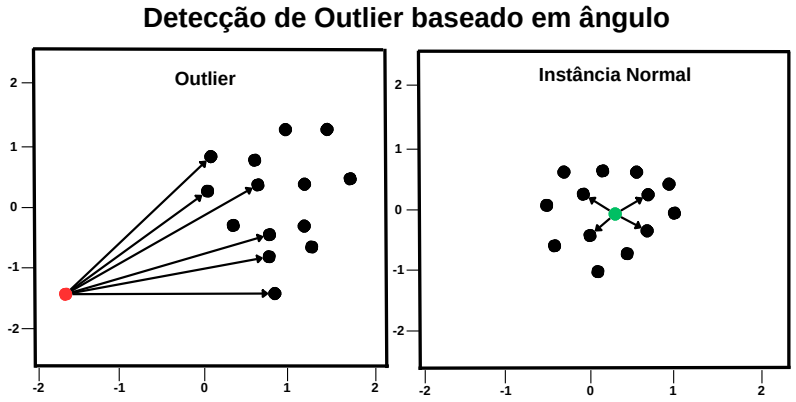


Figura 2.23: Detecção de Outlier Baseado em Ângulo. Adaptado de [70].

Determinante de Covariância Mínima

O método de Determinante de Covariância Mínima (*'Minimum Covariance Determinant'*) (MCD) foi criado por P.J. Rousseeuw em 1984. Ele é usado para identificar *outliers*, em conjuntos de dados multivariados. Esta abordagem seleciona aleatoriamente um subconjunto pertencente aos dados utilizados para estimar a matriz de covariância, a seguir, a distância de Mahalanobis é calculada para cada observação do conjunto de dados completo em relação à matriz de covariância [71].

A distância de Mahalanobis é uma técnica estatística que avalia a distância entre um ponto de dados e um conjunto de dados multivariados, levando em consideração a estrutura de covariância dos dados e a correlação entre as variáveis. [72].

As observações com as maiores distâncias de Mahalanobis são consideradas *outliers*, depois de identificar as observações atípicas, o MCD exclui essas observações e recalcula a matriz de covariância usando apenas as observações restantes. Esse processo é repetido várias vezes com diferentes subconjuntos aleatórios de dados para obter uma estimativa mais robusta da matriz de covariância, que leva em consideração a estrutura de covariância dos dados e é menos sensível a valores extremos [71]. A Figura 2.24 demonstra a ideia de funcionamento por trás deste método.

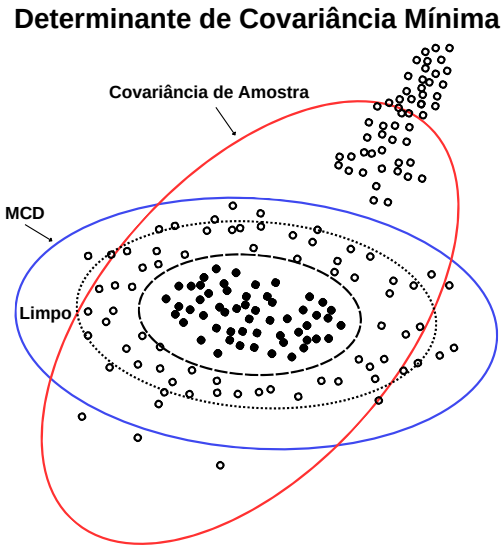


Figura 2.24: Determinante de Covariância Mínima. Adaptado de [73].

2.4.5 Outliers no aspecto de Bateria

Outliers relacionados à bateria podem ocorrer de várias maneiras, incluindo falhas de sensores, erros de medição e condições anormais da bateria. As condições anormais da bateria incluem superaquecimento, sobrecarga, descarga excessiva, Curto-Circuito Externo (ESC), Curto-Circuito Interno (ISC), envelhecimento acelerado, etc. Essas falhas geralmente estão inter-relacionadas. Por exemplo, sobrecarga e descarga excessiva podem causar uma variedade de reações indesejáveis dentro da bateria, acelerando sua deterioração. A geração anormal de calor ocorre sob várias condições, como reações desnecessárias durante sobrecarga e descarga, ISCs e ESCs [74].

Técnicas baseadas em dados comumente usadas no campo de solução de problemas incluem processamento de sinal, ML e fusão de informações. Os algoritmos de ML podem adaptar o conjunto de amostras de treinamento ajustando os parâmetros e extraindo conhecimento das amostras de treinamento atuais. Essa capacidade de adaptar e extrair conhecimento permite que os algoritmos de ML reconheçam padrões de maneira eficaz e tomem decisões com base nos dados disponíveis. A falta de dados de falha em baterias de Lítio-Íon pode causar problemas de *'overfitting'*, apresentando um desempenho muito

bom nos dados de treino, mas um desempenho ruim quando testado em novos dados [74]. Em veículos elétricos, incompatibilidades entre as células da bateria podem levar a falhas de sobrecarga e descarga excessiva, levando à redução da capacidade da bateria [75].

Falhas relacionadas a quedas rápidas de tensão na curva de descarga podem ser causadas por um ISC, a queda rápida de tensão ocorre devido ao contato direto entre cátodo e ânodo como resultado do rompimento do separador [76], o separador geralmente é uma película fina e porosa construída a partir de polímeros sintéticos que atua como uma barreira entre o cátodo e o ânodo. Z. Sun et al. [77] apresenta em seu trabalho este tipo de falha ocorrendo em diferentes tempos de operação e com diferentes magnitudes de queda de tensão, podendo chegar até 0,57 V.

A abordagem orientada a análise de dados para este tipo de falha depende muito de uma boa construção de características que distinguem as células com comportamento normal das células com ISC [78].

G. Zhu et al. [78] cita em seu trabalho, o estudo de M. Schmid et al. [79] que tem como objetivo detectar o ISC na fase inicial, o estudo utiliza a diferença de tensão entre as células de uma bateria como característica para a construção de um modelo de previsão de regressão não linear.

Outro tipo de falha são as falhas de relacionadas a súbitas quedas na tensão que permanecem até o esgotamento total da carga, Z. Chen et al. [80] apresenta um método de detecção de falhas para conjuntos de Baterias de Íon de Lítio (LIBs) utilizando LOF, essas falhas de acordo com [80] podem ser causadas por conexão incorreta, curto-circuito, colisão entre outros, o autor adiciona também que essas falhas podem causar um aumento da resistência interna, o que pode levar à diminuição do desempenho da bateria.

Em geral, muitas vezes é difícil detectar a falha da bateria usando medições de corrente, tensão e temperatura. Em vez disso, os recursos de erro são frequentemente extraídos de respostas anômalas induzidas por erro por meio do processamento de sinal [74].

Os trabalhos [81] [82] [83] [84] citados por [74] apresentam diferentes características utilizadas para a detecção de falhas em baterias, algumas delas são: diferença de tensão, função de flutuação da resistência interna, coeficiente de correlação entre as tensões das

células, entropia da temperatura e da tensão.

Trabalhos Relacionados à Detecção de Falhas em Baterias Utilizando ML

Gotz et al. [85] demonstra a aplicação de seis diferentes tipos de modelos de ML para a detecção de falhas em LIBs, os conjuntos de dados das baterias foram coletados em quatro diferentes tipos de situações críticas: sobrecarga, descarga excessiva, ESC e superaquecimento. O modelo de ML, RF se destacou em relação aos outros, porém, no geral, todos foram capazes de detectar as falhas com uma sensibilidade maior que 94% para falhas mecânicas, elétricas e térmicas.

Samanta et al. [86] demonstra uma revisão sobre as técnicas de detecção e diagnóstico de falhas baseadas em ML especialmente para o sistema de LIBs.

Yang et al. [87] analisa o impacto do Estado de Carga (SOC) e temperatura nas características de falhas de ESC de LIBs, incluindo a variação de corrente, tensão e aumento de temperatura, foi sugerido um método que utiliza o classificador RF para identificar vazamentos de eletrólito em células com ESC.

Kim et al. [88] propõe um novo método para diagnosticar falhas em tempo real em LIBs, o algoritmo de diagnóstico de falhas proposto inclui um algoritmo de detecção de *outliers* que identifica células anormais com base nos resultados do monitoramento de status e identifica tipos de falhas, como células com ISC e células com irregularidades.

Capítulo 3

Abordagem e Ferramentas Utilizadas

Neste capítulo serão abordadas as ferramentas utilizadas para a construção do protótipo de detecção de falhas, assim como as etapas necessárias para a implementação do mesmo. O protótipo tem o objetivo de detectar falhas com técnicas de ML em um cenário laboratorial de pequena escala, detalhes sobre o cenário são apresentados a seguir.

3.1 Sistema Ciberfísico Laboratorial

O sistema proposto por Piardi et al. [89] [90] [91] chamado ARENA consiste num armazém logístico de pequena escala baseado em um cenário real. O cenário proposto envolve a gestão de um conjunto de empilhadeiras autônomas, que estão conectadas de forma que a troca de informações entre elas é possível. As empilhadeiras são representadas por pequenos robôs.

O ambiente ARENA é composto por oito robôs móveis, que utilizam uma bateria recarregável de polímero de lítio (Li-Po) de 3,7V de tensão e 1350mAh de carga.

A Figura 3.1 demonstra o cenário real da ARENA onde é possível observar os oito robôs móveis, as prateleiras do armazém e os caminhos que os robôs percorrem.

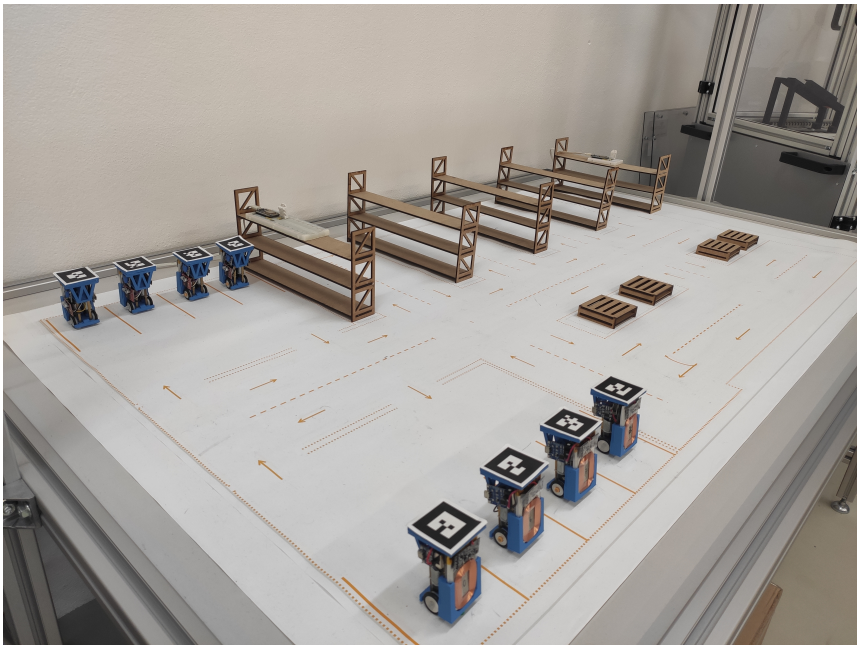


Figura 3.1: Ambiente Real da ARENA

3.2 Abordagem para o Desenvolvimento do Sistema de Detecção de falhas

O estudo adotado para a construção do protótipo de detecção de falhas seguirá uma série de etapas.

Primeiramente, é necessário conhecer e identificar falhas que possam ocorrer na ARENA. Esta etapa é importante pois, com base nela serão filtradas as falhas que serão estudadas e implementadas no protótipo. Posteriormente à escolha das falhas que serão abordadas, será necessário recolher e pré-processar os dados obtidos do CPS para entender o comportamento do sistema e para que o treinamento dos modelos de ML ser possível.

Será necessário identificar os algoritmos de ML que sejam capazes de identificar as falhas escolhidas, após a identificação, os algoritmos de ML serão implementados com base nos dados recolhidos e seu desempenho será comparado com outros tipos de algoritmos, visando identificar qual algoritmo de ML se adapta melhor para cada tipo de falha.

Com os modelos de ML já treinados, será possível criar o protótipo de detecção de falhas, o protótipo terá duas abordagens e estruturas diferentes. A primeira consiste em

um protótipo com estrutura centralizada que receberá as todas as informações necessárias dos robôs móveis do CPS em um único lugar, os modelos de ML para esta estrutura serão treinados utilizando dados gerais sobre o comportamento de cada falha. Já a segunda abordagem e estrutura será local, portanto, cada robô móvel do CPS possuirá uma unidade que irá detectar as falhas presentes e os modelos de ML serão treinados utilizando dados específicos de cada robô. Mediante à criação das duas estruturas do protótipo, será feita uma comparação entre o desempenho da estrutura central com dados gerais e a estrutura local com dados específicos.

Para que o protótipo de detecção de falhas possa ser construído, será utilizado um *framework* de *software open-source* chamado ROS e também para fins de simulação do CPS laboratorial, será utilizado o *software* de simulação de robótica CoppeliaSim (anteriormente chamado de Virtual Robot Experimentation Platform (V-REP)), além claro de algoritmos de ML.

3.3 Sistema Operacional de Robôs (ROS)

Robot Operating System (ROS) é um *framework* de *software open-source* para robótica que oferece uma coleção de bibliotecas, ferramentas e convenções para desenvolvimento de *software* para robôs [92]. O ROS ajuda os desenvolvedores a construir sistemas robóticos de forma mais eficiente, economizando tempo e recursos no processo de desenvolvimento. Algumas das ferramentas disponíveis no ROS incluem gerenciamento de pacotes, sistemas de comunicação entre processos, ferramentas de visualização de dados dos sensores e simulação de robôs. Além disso, o ROS é compatível com uma ampla gama de linguagens de programação, como C++, Python e MATLAB [93]. Em resumo, o ROS simplifica o processo de desenvolvimento de robôs, tornando-o mais acessível e eficiente.

Algumas vantagens do ROS, sobre outras plataformas de robótica, citadas por Joseph e Cacace (2018) [92]:

- Capacidades de alto nível: Capacidades já prontas para uso, como por exemplo os pacotes de navegação autônoma como a Localização e Mapeamento Simultâneo

(SLAM) e a Localização Adaptativa de Monte Carlo (AMCL);

- Suporte para sensores e atuadores de última geração: Estão incluídos nos pacotes do ROS sensores como o Velodyne-LIDAR, Kinect e atuadores como o servos DYNAMIXEL.

Joseph e Cacace (2018) [92] também listam algumas razões de porque as pessoas não utilizam o ROS:

- Dificuldade em aprender: ROS tem uma curva íngreme de aprendizado e pode ser difícil de aprender;

- Dificuldades na modelagem de robôs: A modelagem do robô em ROS deve ser realizada utilizando uma descrição baseada em XML (Extensible Markup Language);

- Potenciais limitações: Há uma falta de suporte de desenvolvimento de aplicativos nativos de tempo real.

O ROS possibilita a interação entre os diversos elementos de *software* que compõem um sistema robótico. Esse processo de interação é viabilizado por meio de um protocolo de comunicação baseado em mensagens leves e flexíveis, que permitem a transmissão de informações em tempo real entre diferentes processos [93].

Existem duas formas predominantes de estabelecer a rede no ROS: a primeira consiste em disponibilizar serviços que podem ser solicitados por outros nós; a segunda, por sua vez, envolve a criação de conexões *'publisher/subscriber'* com outros nós. Em ambos os casos, a comunicação entre os diferentes componentes da rede é realizada por meio de tipos de mensagem previamente especificados.

Na Figura 3.2 é demonstrada a estrutura de comunicação com os 'Nós' do ROS, onde é possível observar o 'Nó A' publicando no 'Tópico' e o 'Nó B' subscrevendo as informações do 'Tópico'.

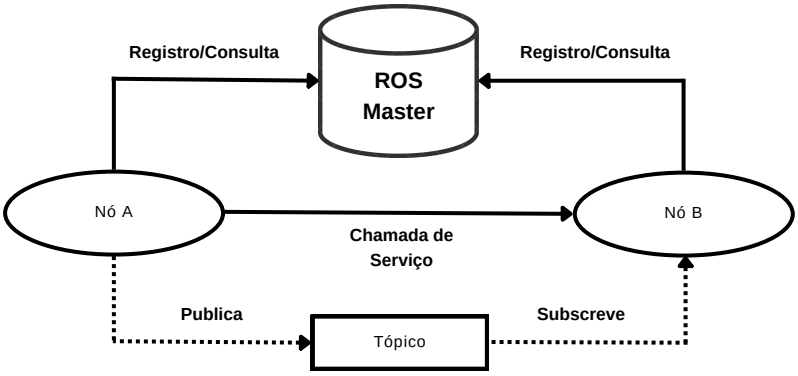


Figura 3.2: Comunicação do Nó ROS. Adaptado de [94].

3.4 Software de Simulação Robótica CoppeliaSim

O simulador de robótica CoppeliaSim, oferece um ambiente de desenvolvimento integrado baseado em uma arquitetura de controle distribuída. Isto significa que cada objeto ou modelo no simulador pode ser controlado individualmente por meio de códigos embutidos [95]. O simulador oferece uma ampla variedade de recursos, incluindo uma Interface Gráfica de Usuário (GUI) intuitiva, suporte para vários idiomas de programação, bibliotecas de modelos de robôs, simulação de sensores e ambientes físicos, além de suporte para ROS e outras tecnologias de robótica, sendo flexível e escalável [95].

O cenário da ARENA de estudo em questão também está disponível no CoppeliaSim, como é demonstrado na Figura 3.3. O arquivo de simulação foi disponibilizado por Piardi et al. [89].

3.5 Integração do CoppeliaSim com o ROS

Para este trabalho, a utilização do ROS é necessária principalmente para que a comunicação entre o protótipo e os robôs móveis do CPS laboratorial aconteça, assim, facilitando a integração e a troca de informações entre eles.

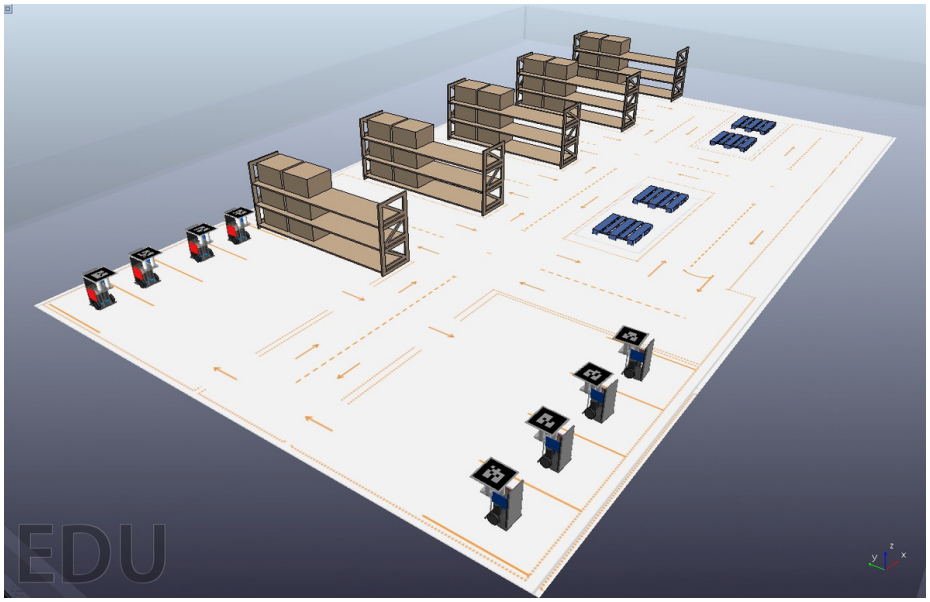


Figura 3.3: Ambiente de Simulação no CoppeliaSim.

O simulador CoppeliaSim é essencial para que os testes relacionados ao CPS laboratorial possam ser feitos virtualmente, diminuindo o custo de materiais e recursos humanos, permitindo a realização de diversas iterações em pouco tempo, sendo possível testar diferentes cenários e condições rapidamente.

Portanto, todas as informações do CPS laboratorial recebidas pelo ROS serão provindas diretamente do simulador CoppeliaSim, já que este tem suporte para o ROS e o ambiente de simulação existente retrata de forma fidedigna o sistema real. O CPS laboratorial real será utilizado para a aquisição de dados que o ambiente de simulação não consiga fornecer. Posteriormente à aquisição destes dados, o ambiente de simulação será adaptado.

Na Figura 3.4 está demonstrada a integração do CoppeliaSim com o ROS no envio de mensagens. Os controladores do ambiente de simulação enviam as informações da simulação para a interface ROS, estes dados são publicados em 'Nós' do ROS e estes dados posteriormente são acessados pelo protótipo de detecção de falhas.

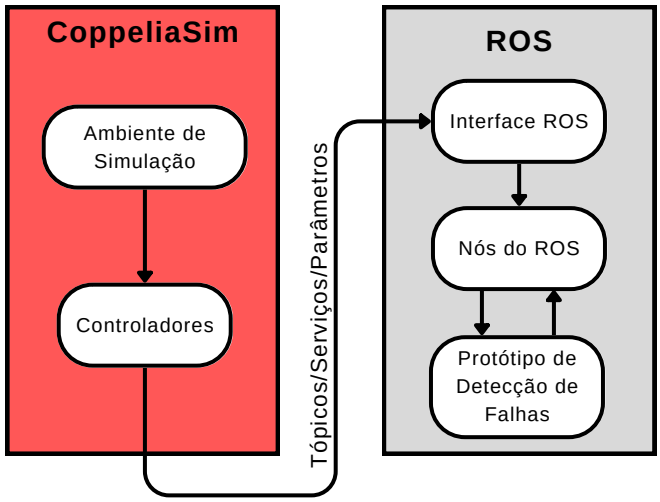


Figura 3.4: Integração do CoppeliaSim com o ROS.

Capítulo 4

Desenvolvimento e Implementação

Neste capítulo serão implementadas as etapas necessárias para a construção do protótipo de detecção de falhas com estrutura central e local.

4.1 Desenvolvimento das Etapas de Preparação

Nesta seção será explorado de forma detalhada a maneira em que o trabalho foi desenvolvido, demonstrando como será estruturado o protótipo de detecção de falhas proposto e como foi feita a aquisição de dados dos robôs.

4.1.1 Identificação das Possíveis Falhas Dentro do Sistema Ciberfísico

Os CPSs, como já mencionado anteriormente, são sistemas que integram a computação com o mundo físico, combinando uma série de elementos e tecnologias para controlar e monitorar os processos físicos em tempo real. Entre a sequência de eventos, desde a coleta de informações dos sensores, processamento de dados, tomada de decisões e controle dos dispositivos físicos, podem ocorrer várias falhas potenciais. No contexto do CPS estudado em questão, as falhas abordadas neste trabalho contemplarão as falhas de *hardware*. Os componentes físicos dos CPSs podem enfrentar possíveis falhas devido a fatores como o

desgaste ao longo do tempo, defeitos de fabricação e outros. As falhas de *hardware* podem afetar a precisão das medições e até causar danos físicos aos dispositivos.

No contexto de falhas de *hardware*, este trabalho tem como objetivo detectar falhas relacionadas às baterias dos robôs móveis do CPS laboratorial. Falhas relacionadas a quedas rápidas de tensão na curva de descarga serão chamadas de 'Falhas de Tipo 1' e falhas de relacionadas a súbitas quedas na tensão que permanecem até o esgotamento total da carga serão chamadas de 'Falhas de Tipo 2', essas falhas poder ocorrer em diferentes tempos de operação e foram abordadas na seção 2.4.5. O comportamento dessas falhas pode ser observado nas Figuras 4.1 e 4.2

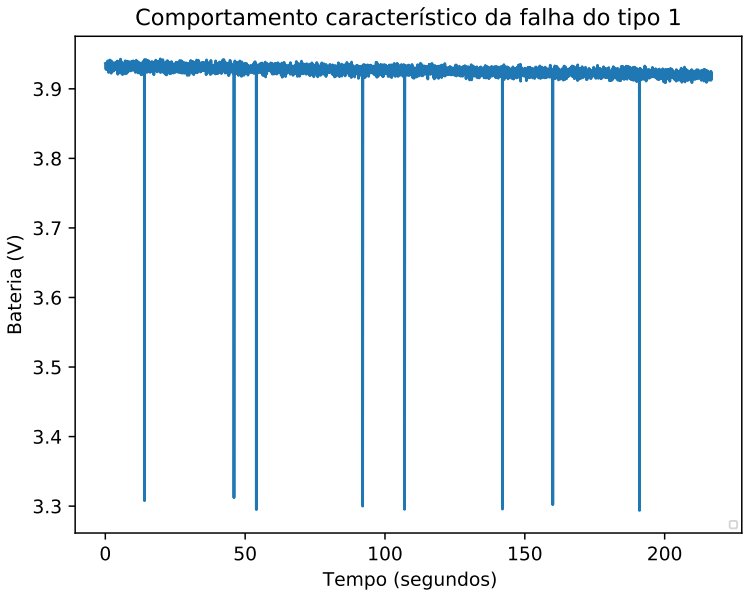


Figura 4.1: Comportamento Característico da Falha do Tipo 1.

O método utilizado nesta dissertação é baseado em dados e não depende de modelos analíticos ou experiência de especialistas, pois é realizado analisando e processando diretamente os dados em execução para detectar erros. No entanto, na maioria dos casos, o pré-processamento dos dados é necessário [74]. Os métodos baseados em dados podem apresentar algumas limitações como a necessidade de um grande conjunto de dados, alto custo computacional e complexidade de treinamento [96].

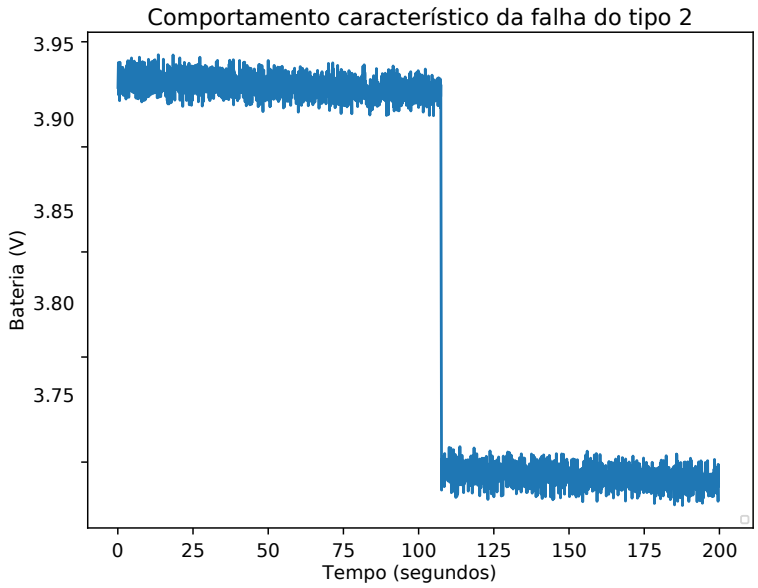


Figura 4.2: Comportamento Característico da Falha do Tipo 2.

4.1.2 Coleção, Pré-Processamento e Modelagem dos Dados Obtidos do Sistema

A coleta dos dados das baterias foi realizada por meio de experimentos laboratoriais na ARENA, como algumas informações presentes nos conjuntos de dados obtidos não serão utilizadas, é preciso fazer o tratamento destes dados e o pré-processamento para que estes fiquem formatados de uma maneira que facilite o seu uso posteriormente nos modelos de ML. A partir dos conjuntos de dados extraídos da ARENA, é necessário encontrar a função pelo tempo que descreve o comportamento de cada bateria, pois, o mesmo será implementado e simulado no CoppeliaSim quando os testes nos protótipos de detecção de falha forem feitos, podendo assim, extrair as informações e características da bateria em questão diretamente do simulador, sem depender fisicamente dos componentes da ARENA.

A bateria utilizada nos robôs móveis é do tipo Li-Po, o modelo da bateria pode ser encontrado como 'CELLEVIA BATTERIES LP503759' e é demonstrada na Figura 4.3. Os dados desta bateria obtidos nos testes laboratoriais resultaram em dois tipos de conjuntos

de dados, cada um relacionado a um estado do robô.

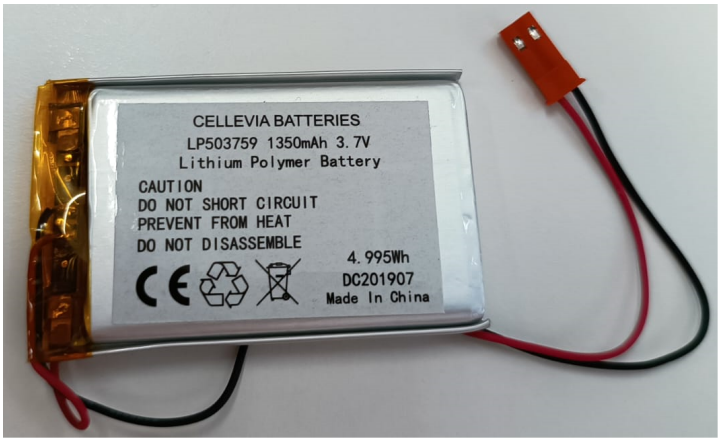


Figura 4.3: Bateria Utilizada nos Robôs Móveis.

O primeiro conjunto de dados extraído está relacionado ao funcionamento do robô sem movimento, ou seja, no primeiro teste realizado, foi obtido o conjunto de dados que representa o funcionamento do robô parado, sem ações adicionais. A Figura 4.4 demonstra a curva de descarga da bateria quando o robô se encontra parado.

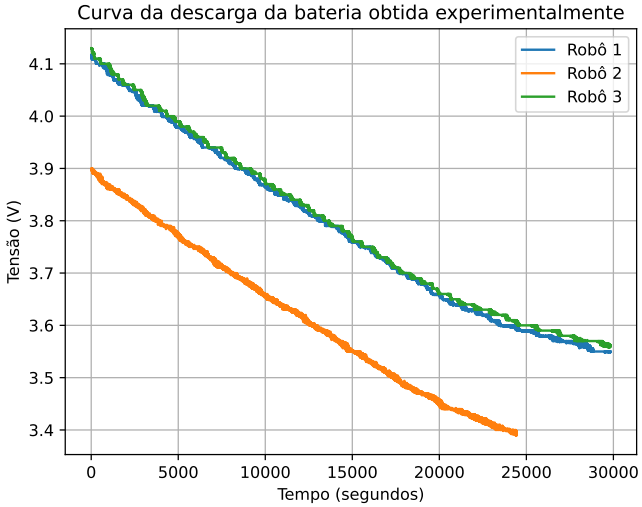


Figura 4.4: Curva de Descarga da Bateria quando o Robô se Encontra Parado.

O segundo conjunto de dados extraído é referente ao funcionamento do robô em movimento, neste segundo conjunto de dados é possível observar uma descarga mais rápida

da bateria, pois, o robô está utilizando mais recursos que no primeiro conjunto de dados. A Figura 4.5 demonstra a curva de descarga da bateria quando o robô se encontra em movimento.

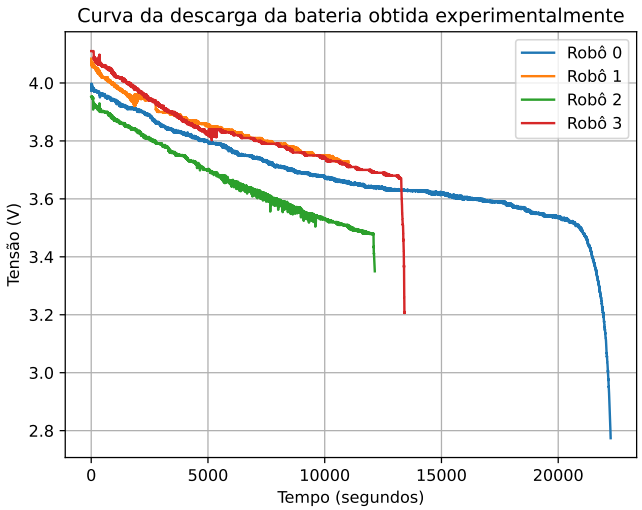
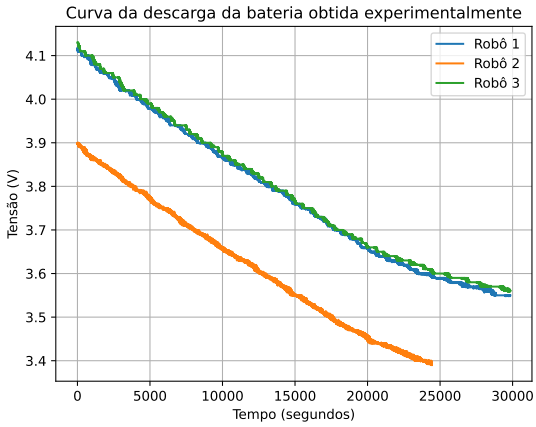


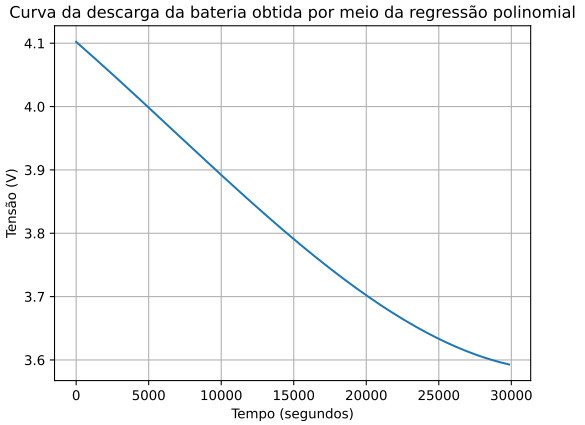
Figura 4.5: Curva de Descarga da Bateria quando o Robô se Encontra em Movimento.

A partir dos conjuntos de dados obtidos nos testes laboratoriais, foi possível encontrar a função f(t) que descreve a curva de descarga de tensão (V) pelo tempo (segundos). A função f(t) foi encontrada através de um método de análise estatística conhecido como regressão polinomial, onde é utilizado um polinômio como função de ajuste para descrever a relação entre a variável dependente que é o que queremos prever (tensão da bateria) e a variável independente que é a variável que usamos para prever (tempo de utilização da bateria) [97].

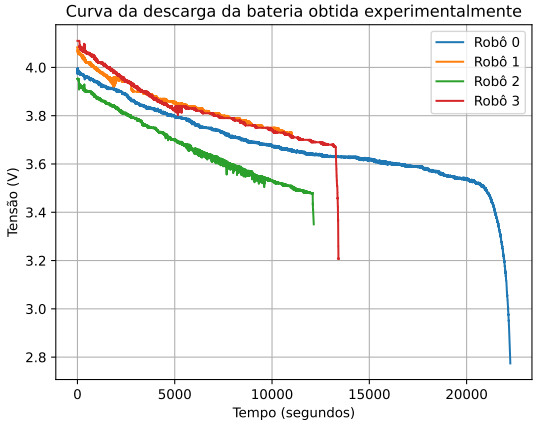
A regressão polinomial foi realizada utilizando o *software Microsoft Excel*, onde foram dispostos os dados do robô 1 de tensão para cada tempo (em segundos) obtidos experimentalmente quando o robô se encontra parado, e os dados do robô 3 quando o robô se encontra em movimento, estes robôs foram escolhidos por apresentarem menos variações na descarga da bateria. Após ajustar o grau da função polinomial na ferramenta do *Microsoft Excel*, foi possível obter dois polinômios de terceira ordem com 99% de acurácia. O polinômio que descreve a curva de descarga obtida em relação ao estado ocioso do robô



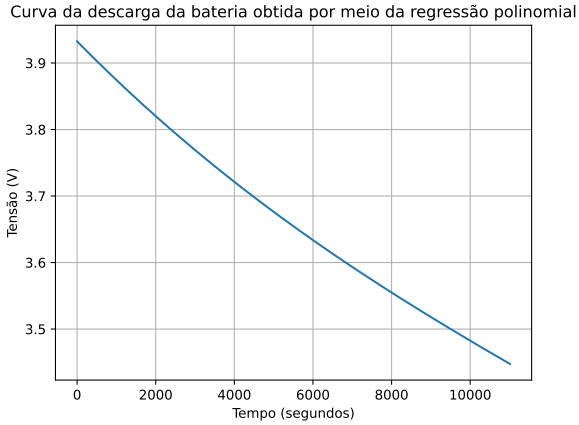
a) Descarga com o robô parado



b) Descarga com robô parado - Regressão Polinomial



c) Descarga com o robô em Movimento



d) Descarga com o robô em Movimento - Regressão Polinomial

Figura 4.6: Comparação Entre as Curvas de Descarga.

móvel se encontra na Equação 4.1 e a que descreve em relação ao robô em movimento se encontra na Equação 4.2.

$$f1(t) = (1 \cdot 10^{-14})t^3 - (2 \cdot 10^{-10})t^2 - (2 \cdot 10^{-5})t + 4,1021 \tag{4.1}$$

$$f2(t) = (-5 \cdot 10^{-14})t^3 + (2 \cdot 10^{-9})t^2 - (6 \cdot 10^{-5})t + 3,9325 \tag{4.2}$$

O gráfico de descarga de bateria comparado ao obtido experimentalmente pode ser

observado na Figura 4.6. É possível notar que a função encontrada representa bem o decaimento de tensão pelo tempo, porém, sem representar as pequenas oscilações observadas nas curvas obtidas experimentalmente, representando apenas de forma geral o comportamento da bateria.

4.1.3 Adaptação do Ambiente de Simulação em Relação aos Dados da Bateria

Seguidamente da obtenção da função pelo tempo f(t) que descreve o comportamento de descarga da bateria, é possível adaptar o ambiente de simulação para que os dados que antes eram obtidos experimentalmente, agora sejam fornecidos pelo simulador.

Foi criado uma função no ambiente de simulação dentro do código de cada robô com o objetivo de verificar o estado em que o robô se encontra (ocioso ou móvel) e a partir desses dados, publicar as informações de tensão (V) pelo tempo (s) em um tópico ROS. Na Figura 4.7 é possível observar o código do robô 0. O código contém variáveis responsáveis por armazenar o valor da função pelo tempo f(t) e o valor de tempo, que é obtido através da função *'sim.getSimulationTime()'*, além disso o código verifica algumas condições (por exemplo, movimento ou não do robô) para retornar o valor correto da bateria.

A taxa em que o ambiente de simulação publica os dados é de 50ms (milissegundos), portanto a taxa de amostragem utilizada será de 50ms, ou seja, as amostras estarão sendo coletadas pelo(s) Nó(s) ROS em intervalos de 50ms.

4.1.4 Abordagens Propostas para os Protótipos de Detecção de Falha Relacionados a Baterias

As Figuras 4.8 e 4.9 representam as estruturas dos protótipos criados para detecção falhas. Na prática, a estrutura centralizada recebe os dados publicados em tópicos ROS por meio do CoppeliaSim, cada robô dentro do ambiente de simulação publica seus dados em tópicos, estes dados podem ser: Bateria, Posição e outros. Neste caso, o nó central ROS irá subscrever os dados relacionados à bateria, os modelos de ML utilizados nesta

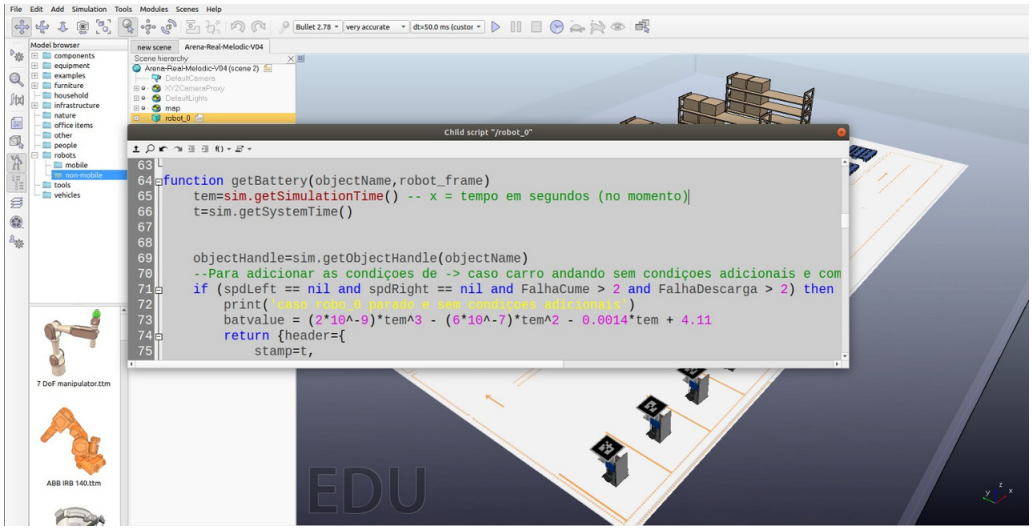


Figura 4.7: Função da Descarga da Bateria no Simulador.

abordagem serão treinados com dados generalizados obtidos diretamente da função pelo tempo que descreve o comportamento da bateria.

A estrutura local utilizará o mesmo procedimento feito para o nó central, porém, diferentemente do nó central que receberá os dados de todos os robôs e posteriormente realizará a detecção de falhas por meio de modelos de ML treinados com dados generalizados, a estrutura local irá possuir um nó local e também um modelo de ML customizado, treinado com base nos dados específicos de cada robô.

4.2 Implementação dos Modelos de ML

Para a implementação dos modelos de ML com o objetivo de detectar falhas, foi utilizado a biblioteca de aprendizado de máquina Scikit-Learn [98] totalmente *open source*, que oferece uma grande variedade de algoritmos e ferramentas.

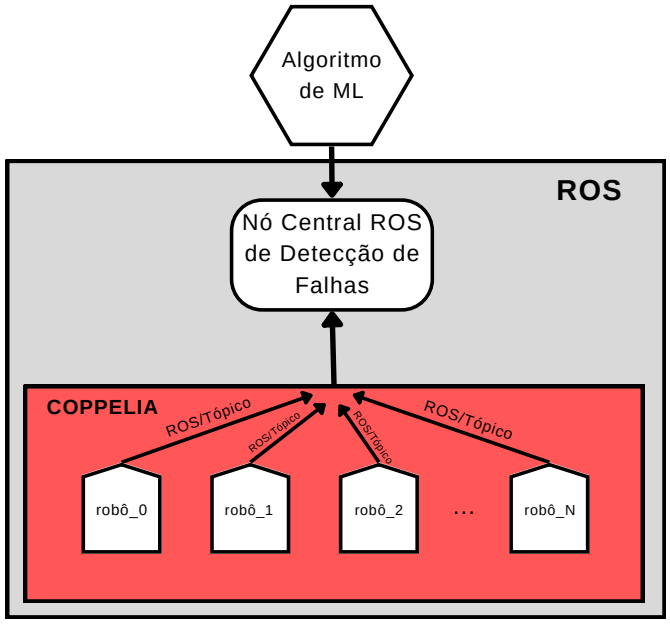


Figura 4.8: Estrutura Central de Detecção de Falhas para Baterias

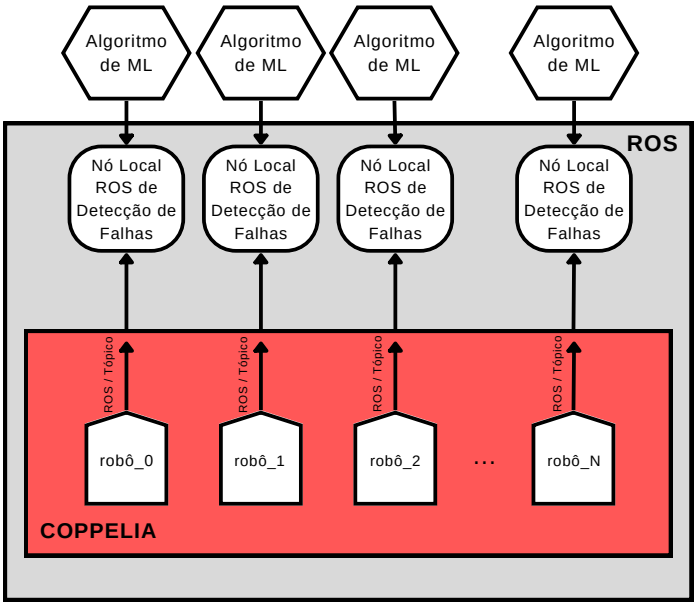


Figura 4.9: Estrutura Local de Detecção de Falhas para Baterias.

4.2.1 Modelos Selecionados para Detecção de falhas na Bateria

Com base nos trabalhos apresentados anteriormente no Capítulo 2, é possível observar que a detecção de falhas em baterias comumente é feita por modelos do tipo supervisionado, redes neurais e também modelos do tipo não supervisionado. Para esta aplicação, serão feitos testes tanto em modelos do tipo supervisionado que são capazes de fazer previsões com base nos rótulos do conjunto de dados, quanto modelos do tipo não supervisionado que são capazes de descobrir padrões ou estruturas no conjunto de dados, identificando como falhas pontos de dados que não seguem este padrão. A utilização destes dois tipos de modelos foi feita com o objetivo de comparar qual tipo de modelo de aprendizagem de máquina performa melhor nessa aplicação.

Os algoritmos do tipo supervisionado DT, K-Vizinhos Mais Próximos (KNN) e RF serão utilizados, já do tipo não supervisionado os algoritmos IF e LOF serão utilizados. A fim de realizar um melhor treinamento destes modelos, foram extraídas outras características dos conjuntos de dados obtidos experimentalmente da curva de descarga da bateria. Para a detecção de falhas em baterias será utilizada a diferença de tensão (V), relacionada entre um ponto de dado e o ponto de dado seguinte, além dessa característica, será utilizado o ângulo de inclinação formado pela reta entre dois pontos de dados conhecidos e o eixo ‘x’, que é obtido por meio do cálculo do coeficiente angular de uma reta, este cálculo e as características utilizadas serão esclarecidas de maneira mais detalhada na próxima subseção.

4.2.2 Treinamento dos Modelos de ML

Para treinar os modelos de ML escolhidos para detectar falhas em baterias é necessário utilizar conjuntos de dados que contenham características importantes em relação ao comportamento da bateria. Como o objetivo é detectar dois tipos de falhas, dois modelos de ML utilizando cinco diferentes algoritmos foram treinados (dois algoritmos do tipo não supervisionado e 3 algoritmos do tipo supervisionado), cada modelo foi treinado para detectar um tipo de falha, somando no total, dez modelos.

A primeira característica é relacionada ao ângulo obtido entre um par de pontos de dados e é obtida através do cálculo do coeficiente de uma reta, o Coeficiente Angular (m) representa a taxa de variação de 'y' e de 'x' da reta, o coeficiente angular positivo indica que a medida em que 'x' aumenta, 'y' também aumenta e para um coeficiente angular negativo, a medida em que 'x' diminui, 'y' também diminui. A equação 4.3 representa o coeficiente angular e pode ser utilizada quando existem dois pontos conhecidos. A Figura 4.10 demonstra este conceito.

$$m = \frac{yB - yA}{xB + xA} \tag{4.3}$$

Neste caso, o valor que está sendo utilizado para a característica é o ângulo tangente α formado entre a reta e o eixo 'x', que pode ser calculado a partir do coeficiente angular [99], como demonstrado na Equação 4.5.

$$m = \tan \alpha \tag{4.4}$$

$$\alpha = \arctan m \tag{4.5}$$

Valores anômalos possuirão ângulos com valores distintos em relação aos valores não anômalos, se houver algum pico ou vale de tensão (V) na distribuição de dados, o mesmo será destacado por conta do valor de ângulo diferente em relação aos demais pontos de dados.

A segunda característica é a da diferença de magnitude em relação a tensão (V) e é obtida realizando o cálculo da diferença do valor de tensão (V) da bateria no ponto de dado atual com o próximo ponto de dado. Esta característica visa caracterizar pontos de dados que tenham um desgaste maior de tensão, potencializando assim, a chance deste ponto ser uma falha.

Para o treinamento dos modelos, o conjunto de dados com essas características foram divididos em conjunto de treino e conjunto de teste, com a proporção de $\frac{2}{3}$ para treino e

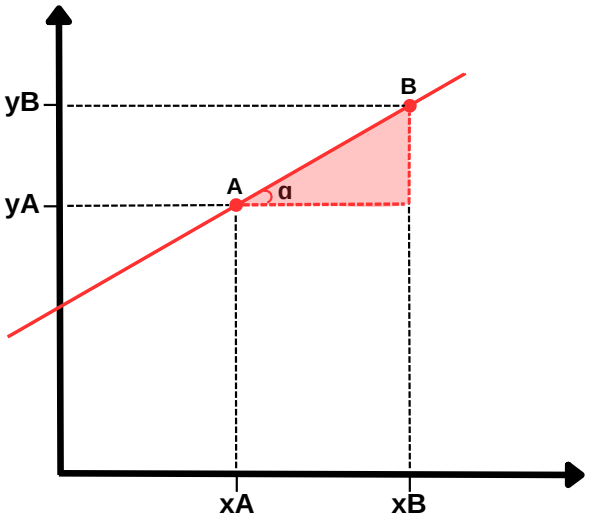


Figura 4.10: Ângulo Formado Entre a Reta e o Eixo 'x'.

$\frac{1}{3}$ para teste. A divisão do conjunto de dados é uma forma de validar os modelos de ML, pois, testa o modelo em dados ainda não vistos.

Em virtude da utilização de algoritmos do tipo supervisionado além de algoritmos do tipo não supervisionado, é necessário a adição de rótulos no conjunto de dados, indicando quais dados são considerados falhas e quais não são.

Devido à circunstância que os dados obtidos experimentalmente não continham pontos de dados que apresentavam falhas, pontos de dados falhos foram sinteticamente introduzidos no *dataset* com o objetivo de fazer o algoritmo entender quais são os padrões de falha e quais são os padrões normais.

A divisão do conjunto de dados foi realizada por meio da função *'train_test_split'* da biblioteca *scikit-learn* (todas as funções a seguir foram extraídas desta mesma biblioteca), a função recebe como entrada os conjuntos de recursos (comumente chamado de 'X') que são as características do conjunto de dados e os rótulos (comumente chamado de 'y') e divide aleatoriamente preservando a proporção desejada, a função retorna quatro variáveis: X_treino, X_teste, y_treino, y_teste.

Com os dados separados, o próximo passo é fazer com que o modelo de ML aprenda

os padrões presentes nos dados para que posteriormente o modelo possa fazer previsões. Para treinar o modelo foi utilizada a função *'fit'* que recebe os dados de treinamento, tanto o conjunto de recursos (X_treino) quanto o conjunto de rótulos (y_treino).

Após a conclusão da função *'fit'*, o modelo pode fazer previsões de novos dados utilizando a função *'predict'*, os novos dados devem seguir a mesma estrutura utilizada para treinamento. Quando o conjunto de dados é separado entre dados de treino e dados de teste, geralmente a função *'predict'* recebe como entrada os dados de teste (X_teste), essas previsões podem ser utilizadas posteriormente para avaliar o desempenho do modelo.

O código abaixo demonstra a utilização dessas funções:

```
#Dividindo o conjunto de dados
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y,
    tamanho_teste=1/3)

#Criando o modelo
ML = Modelo_ML()

#Treinamento do modelo
ML.fit(X_treino, y_treino)

#Utilizando o modelo para prever novos dados
y_pred = ML.predict(X_teste)
```

Listing 4.1: Exemplo de Treinamento de Modelos

4.2.3 Validação dos Modelos de ML

A validação dos modelos é importante, pois, é nesta etapa em que o desempenho dos modelos é verificado, com o propósito de assegurar a capacidade do mesmo de realizar previsões precisas em dados não vistos pelo modelo.

Algumas abordagens comuns para validar um modelo de ML são:

- Utilizar métricas de avaliação: Algumas métricas conhecidas para avaliar problemas

de classificação são: acurácia (corresponde a proporção de exemplos classificados corretamente em relação ao total de exemplos avaliados), precisão, *recall* e F1-score;

- Testes em dados não vistos: Mesmo após ter feito a separação do conjunto de dados, é importante que o modelo seja testado em dados completamente novos (dados diferentes dos conjuntos de treino e teste) para que o desempenho em cenários do mundo real sejam avaliados.

Com isto, a fim de avaliar o desempenho dos modelos empregados, a função *'classification_report'* foi utilizada, ela recebe como entrada o conjunto de rótulos corretos ('y_teste') e o conjunto das previsões do modelo ('y_pred') e retorna métricas como: precisão, *recall*, F1-score e suporte.

```
classification_report(y_test, y_pred)
```

Listing 4.2: Utilizando o classification_report

- Precisão pode ser definida como o número de verdadeiros positivos (TP) dividido por ele mesmo somado ao número de falsos positivos (FP) [100]. A equação 4.6 demonstra a fórmula da precisão.

$$Precisão(P) = \frac{TP}{TP + FP} \tag{4.6}$$

- O *recall* é definido como número de verdadeiros positivos (TP) dividido por ele mesmo somado aos falsos negativos (FN) [101]. A Equação 4.7 demonstra a fórmula do *recall*.

$$Recall(R) = \frac{TP}{TP + FN} \tag{4.7}$$

- O F1-score leva em conta os valores de precisão e *recall*, é definido como uma média harmônica desses valores [102]. A Equação 4.8 demonstra a fórmula do F1-score.

$$F1 - score(F1) = 2 \cdot \frac{P \cdot R}{P + R} \tag{4.8}$$

- O suporte indica a frequência em que determinada classe de pontos de dados aparece em um conjunto de dados.

Para analisar a quantidade de verdadeiros positivos (TP), falsos positivos (FP), falsos negativos (FN) e verdadeiros negativos (TN) foi empregado a função *'confusion_matrix'*, que nos retorna uma matriz disposta da mesma forma que a Figura 4.11. Todas as informações sobre a biblioteca e as funções do *scikit-learn* podem ser encontradas em [103].

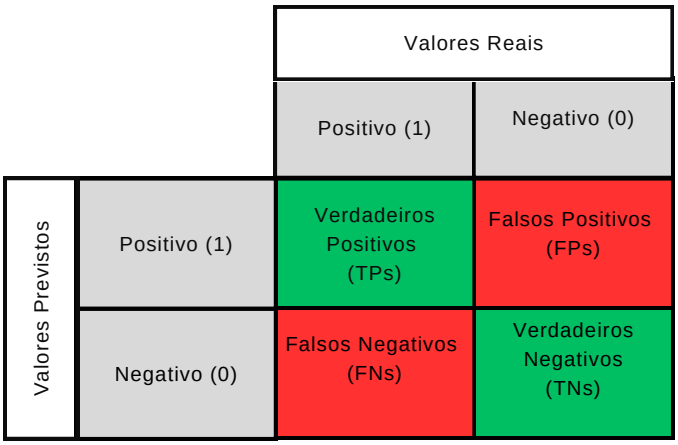


Figura 4.11: Matriz de Confusão. Adaptado de [104].

Os testes em novos dados serão feitos utilizando os dados fornecidos pelo CoppeliaSim após a implementação dos protótipos de detecção de falhas.

4.3 Implementação da Estrutura Central para Detecção de Falhas

Com o modelo de ML treinado e o ambiente de simulação adaptado, o próximo e último bloco a ser implementado é o nó ROS que receberá os dados do simulador e o modelo de ML treinado.

O nó ROS será criado por meio de um código na linguagem Python e receberá os dados de bateria de cada robô presente no ambiente de simulação, esses dados serão

armazenados em uma variável dentro do código Python. A Figura 4.12 mostra o nó ROS sendo subscrito por tópicos do tipo bateria, os tópicos são representados por retângulos e os nós por elipses. Esta representação pode ser obtida por meio da ferramenta gráfica 'rqt_graph' em um sistema ROS.

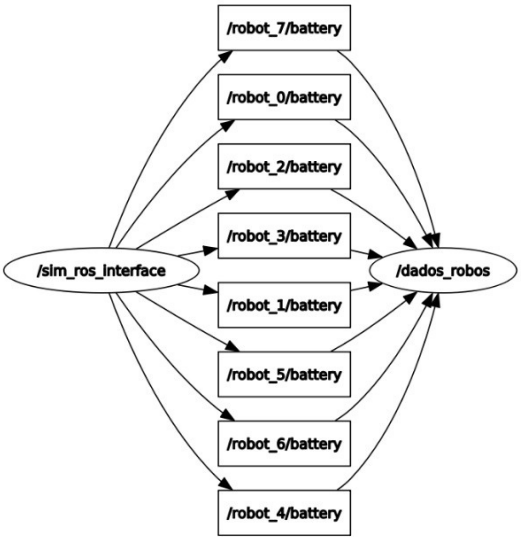


Figura 4.12: Nó Centralizado Recebendo os Tópicos de Bateria.

Posteriormente ao armazenamento de uma quantidade de dados, o código Python realiza o processamento destes dados, para que estes fiquem com mesmo formato e as mesmas características em que o modelo de ML foi treinado, isso é necessário para que os dados possam ser introduzidos no modelo de ML e as devidas predições sejam feitas.

O modelo de ML treinado é importado para o código Python utilizando o módulo Python conhecido como *'pickle'*, este módulo realiza a serialização e a desserialização de objetos Python [105].

Com o modelo importado e os novos dados formatados corretamente, basta inserir os novos dados no modelo de ML para que este faça a previsão dos dados, resultando em predições normais ou anormais.

Após a detecção ou não de falha pelo modelo de ML, o código Python analisa este

resultado e apresenta a detecção da falha, exibindo métricas como o tempo em que ocorreu e qual foi o tipo de falha detectada. Um exemplo disso pode ser observado na Figura 4.13.

```
[INFO] [1687750508.691747]: FALHA!!! Entre os segundos 211.9499969482422 e 212.0 do ROBO 0
[INFO] [1687750508.827494]: FALHA!!! Entre os segundos 212.0 e 212.0500030517578 do ROBO 0
[INFO] [1687750554.991867]: FALHA!!! Entre os segundos 234.9499969482422 e 235.0 do ROBO 0
[INFO] [1687750555.123889]: FALHA!!! Entre os segundos 235.0 e 235.0500030517578 do ROBO 0
[INFO] [1687750685.145525]: FALHA!!! Entre os segundos 299.95001220703125 e 300.0 do ROBO 0
^C[INFO] [1687751032.776727]: Finalizando
10.0 FALHAS DE VALE ENCONTRADA NO ROBÔ 0!
11.0 FALHAS DE DESCARGA ENCONTRADA NO ROBÔ 0!
FALHA DE VALE encontrada entre os pontos [ 3.25585073 13.        ] e [ 3.93493539 13.05000019]
FALHA DE VALE encontrada entre os pontos [ 3.25621179 38.        ] e [ 3.92861082 38.04999924]
FALHA DE VALE encontrada entre os pontos [ 3.24572756 70.        ] e [ 3.9283068  70.05000305]
```

Figura 4.13: Exemplo Detecção de Falha.

Portanto, basicamente o protótipo de detecção de falhas com estrutura central é um código Python que executa várias etapas. Ele cria o nó ROS central para comunicação e recebe os dados do CoppeliaSim. Após o pré-processamento dos dados, o modelo de aprendizado de máquina previamente treinado é importado. Os dados formatados são então inseridos no modelo, e com base nas previsões do modelo, o protótipo realiza a detecção de falhas. A escalabilidade dos protótipos tanto com estrutura central quanto o de estrutura local está ligada diretamente com o custo computacional, é possível adaptar os protótipos para lidar com mais robôs e também com diferentes cenários, porém, o custo computacional será maior e a eficiência do sistema pode ser afetada.

4.4 Implementação da Estrutura Local para Detecção de falhas

Os passos seguidos para a construção da estrutura local para detecção de falhas foram os mesmos adotados para a implementação da estrutura central, as diferenças entre eles são que, os modelos de ML utilizados nas estruturas locais, foram treinados de forma customizada, ou seja, cada modelo foi treinado com base no *dataset* de um robô em específico, e não mais em um *dataset* generalizado, como foi feito na estrutura central.

Outra diferença é que, para cada nó local criado para receber dados de determinado robô, será necessário um código Python, totalizando oito códigos. Na Figura 4.14 é possível observar a representação gráfica dos Nós Locais, obtida por meio da ferramenta 'rqt_graph'.

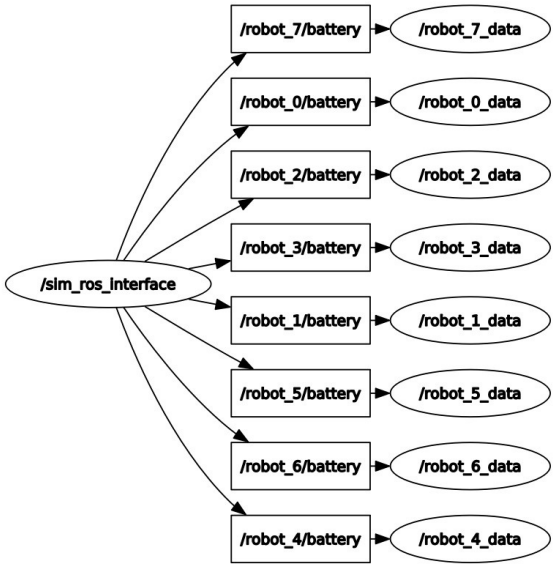


Figura 4.14: Nós Locais Recebendo os Tópicos de Bateria.

Capítulo 5

Testes e Resultados

Este capítulo apresenta os testes e resultados obtidos em relação à detecção de falhas de baterias, as métricas de desempenho obtidas nos testes dos modelos de ML, assim como a comparação entre o desempenho dos modelos utilizados.

5.1 Detecção de Falhas nos Protótipos

Com os modelos de ML implementados no protótipo de detecção de falhas, é possível obter as predições realizadas e através disso, analisar e informar por meio de mensagens, dados sobre o robô em que ocorreu a falha e o tempo em que a anomalia aconteceu. A Figura 5.1 demonstra a estrutura em que os novos dados estão saindo do simulador e a estrutura que está sendo inserida nos modelos de ML após o pré-processamento dos dados para realizar as predições no protótipo.

A tensão da bateria e tempo enviados pelo simulador, são utilizados para extrair as características que serão inseridas no modelo de ML, para que este faça a análise e a predição se é uma anomalia ou não. A 'Ocorrência de Falha' é o parâmetro enviado pelo simulador (0 ou 1) que representa se aquele ponto de dado apresenta o comportamento padrão da bateria ou se é uma falha inserida sinteticamente pelo simulador, esses parâmetros são armazenados em uma variável para que no final do teste os valores que foram preditos por meio do modelo de ML seja comparado com os valores enviados pelo simulador, obtendo

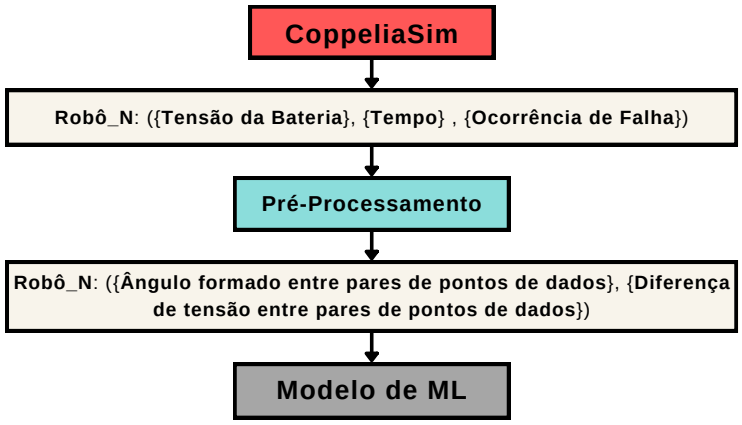


Figura 5.1: Estrutura dos Dados.

assim, as métricas de avaliação.

Para os testes no protótipo, foi inserido um ruído branco que varia entre os valores -1 e 1 na descarga da bateria de cada robô, este valor de ruído é multiplicado por outro parâmetro que representa a intensidade deste ruído, cada robô tem um parâmetro de intensidade de ruído, alguns robôs apresentam baixo ruído e outros um alto ruído, ocasionando assim um desafio maior em relação a classificação correta dos pontos de dados para os modelos.

Para ter uma melhor percepção de como o protótipo funciona, um exemplo será demonstrado a seguir. Na Figura 5.2 é demonstrada a curva de descarga de uma bateria. As falhas do tipo 1 foram inseridas sinteticamente no simulador para acontecer aleatoriamente entre 0 e 250 segundos de simulação, já a falha do tipo 2 foi inserida sinteticamente para ocorrer aleatoriamente entre 250 segundos e 300 segundos de simulação.

Nesta curva de descarga é possível identificar comportamentos anômalos que estão circulados de vermelho, a Figura 5.3 mostra a detecção de falhas realizada pelo protótipo para esta curva de descarga.

A detecção de falhas exibe em qual intervalo de tempo ocorreu a falha na execução do protótipo e, e após o final da análise dos dados e detecção, o protótipo exibe mais detalhes como o tipo de falha encontrada e entre quais pontos de dados foi encontrada a falha, mostrando também a tensão (V) e o tempo (s) destes pontos de dados falhos. A Figura 5.4 mostra a quantidade de falhas detectadas. A Figura 5.5 mostra dados da

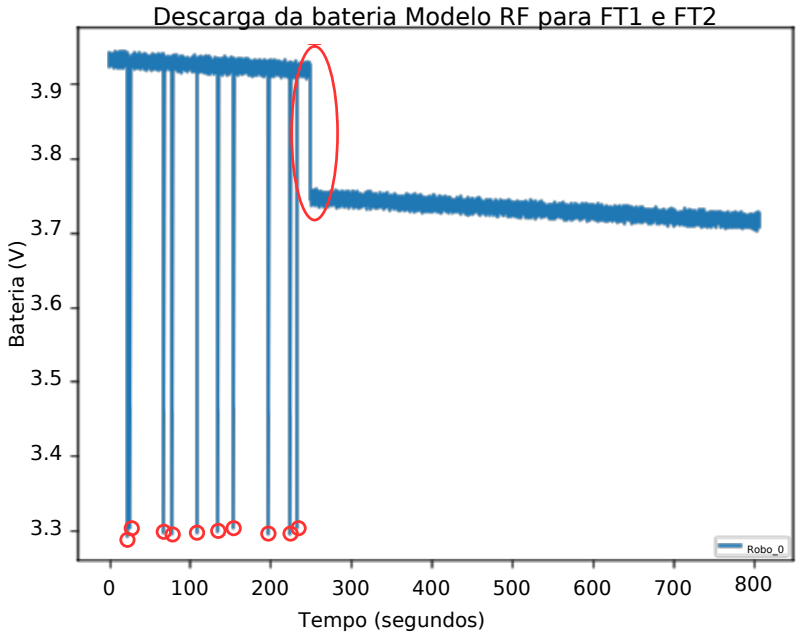


Figura 5.2: Curva de Descarga da Bateria com Pontos de Dados Anômalos Circulados em Vermelho.

```
[INFO] [1686814615.306311]: FALHA!!! Entre os segundos 22.950000762939453 e 23.0 do ROBO 0
[INFO] [1686814615.414324]: FALHA!!! Entre os segundos 23.0 e 23.049999237060547 do ROBO 0
[INFO] [1686814621.242944]: FALHA!!! Entre os segundos 25.950000762939453 e 26.0 do ROBO 0
[INFO] [1686814621.358732]: FALHA!!! Entre os segundos 26.0 e 26.049999237060547 do ROBO 0
[INFO] [1686814700.298647]: FALHA!!! Entre os segundos 67.94999694824219 e 68.0 do ROBO 0
[INFO] [1686814700.415570]: FALHA!!! Entre os segundos 68.0 e 68.05000305175781 do ROBO 0
[INFO] [1686814719.112580]: FALHA!!! Entre os segundos 77.94999694824219 e 78.0 do ROBO 0
[INFO] [1686814719.224298]: FALHA!!! Entre os segundos 78.0 e 78.05000305175781 do ROBO 0
[INFO] [1686814777.950019]: FALHA!!! Entre os segundos 108.94999694824219 e 109.0 do ROBO 0
[INFO] [1686814778.066762]: FALHA!!! Entre os segundos 109.0 e 109.05000305175781 do ROBO 0
[INFO] [1686814827.436016]: FALHA!!! Entre os segundos 134.9499969482422 e 135.0 do ROBO 0
[INFO] [1686814827.565166]: FALHA!!! Entre os segundos 135.0 e 135.0500030517578 do ROBO 0
[INFO] [1686814863.493910]: FALHA!!! Entre os segundos 153.9499969482422 e 154.0 do ROBO 0
[INFO] [1686814863.606232]: FALHA!!! Entre os segundos 154.0 e 154.0500030517578 do ROBO 0
[INFO] [1686814946.872487]: FALHA!!! Entre os segundos 197.9499969482422 e 198.0 do ROBO 0
[INFO] [1686814946.987077]: FALHA!!! Entre os segundos 198.0 e 198.0500030517578 do ROBO 0
[INFO] [1686814998.287164]: FALHA!!! Entre os segundos 224.9499969482422 e 225.0 do ROBO 0
[INFO] [1686814998.407027]: FALHA!!! Entre os segundos 225.0 e 225.0500030517578 do ROBO 0
[INFO] [1686815013.459755]: FALHA!!! Entre os segundos 232.9499969482422 e 233.0 do ROBO 0
[INFO] [1686815013.578600]: FALHA!!! Entre os segundos 233.0 e 233.0500030517578 do ROBO 0
[INFO] [1686815045.922147]: FALHA!!! Entre os segundos 249.9499969482422 e 250.0 do ROBO 0
```

Figura 5.3: Detecção de Falha Realizada pelo Protótipo.

detecção detalhada para falhas do tipo 1 (falha de vale) e a Figura 5.6 mostra para as falhas do tipo 2 (falha de descarga).

```
10.0 FALHAS DE VALE ENCONTRADA NO ROBÔ 0!
11.0 FALHAS DE DESCARGA ENCONTRADA NO ROBÔ 0!
```

Figura 5.4: Quantidade de Falhas Detectadas.

```
FALHA DE VALE encontrada entre os pontos [ 3.29308464 23.        ] e [ 3.92148664 23.04999924]
Falha Robo 0 encontrada no segundo: 23.049999237060547
FALHA DE VALE encontrada entre os pontos [ 3.30444275 26.        ] e [ 3.9343345  26.04999924]
Falha Robo 0 encontrada no segundo: 26.049999237060547
FALHA DE VALE encontrada entre os pontos [ 3.29840394 68.        ] e [ 3.92656613 68.05000305]
Falha Robo 0 encontrada no segundo: 68.05000305175781
FALHA DE VALE encontrada entre os pontos [ 3.29588856 78.        ] e [ 3.92568967 78.05000305]
Falha Robo 0 encontrada no segundo: 78.05000305175781
FALHA DE VALE encontrada entre os pontos [  3.29773475 109.        ] e [  3.92374237 109.05000305]
Falha Robo 0 encontrada no segundo: 109.05000305175781
FALHA DE VALE encontrada entre os pontos [  3.3 135. ] e [  3.92965436 135.05000305]
Falha Robo 0 encontrada no segundo: 135.0500030517578
FALHA DE VALE encontrada entre os pontos [  3.30396111 154.        ] e [  3.91575229 154.05000305]
Falha Robo 0 encontrada no segundo: 154.0500030517578
FALHA DE VALE encontrada entre os pontos [  3.29644425 198.        ] e [  3.91950218 198.05000305]
Falha Robo 0 encontrada no segundo: 198.0500030517578
FALHA DE VALE encontrada entre os pontos [  3.29692179 225.        ] e [  3.9210909  225.05000305]
Falha Robo 0 encontrada no segundo: 225.0500030517578
FALHA DE VALE encontrada entre os pontos [  3.30401214 233.        ] e [  3.92087154 233.05000305]
Falha Robo 0 encontrada no segundo: 233.0500030517578
```

Figura 5.5: Detecção Detalhada para a Falha do Tipo 1 Realizada pelo Protótipo.

As falhas do tipo 1 foram encontradas nos segundos: 23,04, 26,04, 68,05, 78,05, 109,05, 135,05, 154,05, 198,05, 225,05 e 233,05. Essas falhas podem ser confirmadas analisando a curva de descarga. Já para a falha do tipo 2, é possível notar que dez das onze falhas detectadas são falsos positivos (FP), a única detecção que realmente é do tipo 2 foi encontrada em 250 segundos, o momento e a característica do comportamento da falha pode ser observada na Figura 5.2.

A matriz de confusão para este teste apresentou 16140 verdadeiros positivos (TP), 11 verdadeiros negativos (TN) e apenas 10 falsos negativos (FN) ficando com uma acurácia de 0.99 ou 99% em relação a todos os dados previstos. A Figura 5.7 demonstra a matriz de confusão deste teste.

Todos os testes nos protótipos de estrutura central e local que serão realizados a partir de agora, foram feitos da mesma maneira apresentada neste exemplo, porém, para todos

```
FALHA DE DESCARGA encontrada entre os pontos [ 3.93145029 22.95000076] e [ 3.29308464 23.        ]
Falha Robo 0 encontrada no segundo: 23.0
FALHA DE DESCARGA encontrada entre os pontos [ 3.9260207  25.95000076] e [ 3.30444275 26.        ]
Falha Robo 0 encontrada no segundo: 26.0
FALHA DE DESCARGA encontrada entre os pontos [ 3.92636729 67.94999695] e [ 3.29840394 68.        ]
Falha Robo 0 encontrada no segundo: 68.0
FALHA DE DESCARGA encontrada entre os pontos [ 3.93133037 77.94999695] e [ 3.29588856 78.        ]
Falha Robo 0 encontrada no segundo: 78.0
FALHA DE DESCARGA encontrada entre os pontos [  3.92959691 108.94999695] e [  3.29773475 109.        ]
Falha Robo 0 encontrada no segundo: 109.0
FALHA DE DESCARGA encontrada entre os pontos [  3.92804239 134.94999695] e [  3.3 135. ]
Falha Robo 0 encontrada no segundo: 135.0
FALHA DE DESCARGA encontrada entre os pontos [  3.92331022 153.94999695] e [  3.30396111 154.        ]
Falha Robo 0 encontrada no segundo: 154.0
FALHA DE DESCARGA encontrada entre os pontos [  3.92698762 197.94999695] e [  3.29644425 198.        ]
Falha Robo 0 encontrada no segundo: 198.0
FALHA DE DESCARGA encontrada entre os pontos [  3.92089832 224.94999695] e [  3.29692179 225.        ]
Falha Robo 0 encontrada no segundo: 225.0
FALHA DE DESCARGA encontrada entre os pontos [  3.91442872 232.94999695] e [  3.30401214 233.        ]
Falha Robo 0 encontrada no segundo: 233.0
FALHA DE DESCARGA encontrada entre os pontos [  3.91971329 249.94999695] e [  3.74987031 250.        ]
Falha Robo 0 encontrada no segundo: 250.0
```

Figura 5.6: Detecção Detalhada para Falha do Tipo 2 Realizada pelo Protótipo.

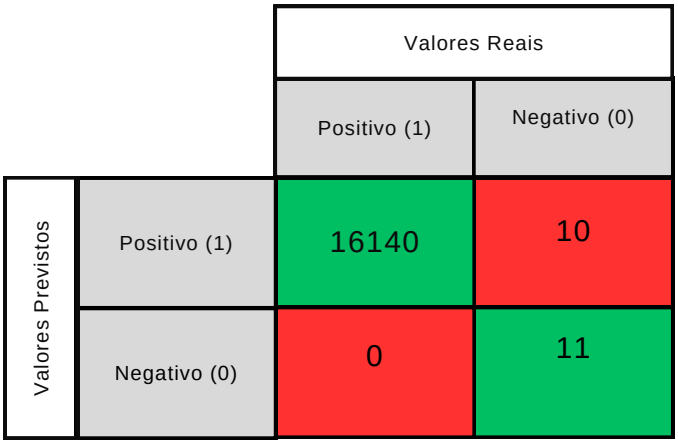


Figura 5.7: Matriz de Confusão.

os oito robôs e, não serão apresentadas as mensagens de detecção de falhas, apenas os resultados das métricas de acurácia e F1-score que podem ser obtidas por meio da matriz de confusão.

5.2 Testes dos Modelos de ML e do Protótipo com Estrutura Centralizada

Nesta seção serão abordados os testes dos modelos de ML aplicados e o teste do protótipo utilizado para detectar falhas na curva de descarga da bateria quando o robô se encontra parado e também quando está em movimento. Todos os modelos de ML foram treinados e testados utilizando as mesmas características, como os modelos não supervisionados (IF e LOF) não necessitam de rótulos para serem treinados, estes somente foram utilizados para a obtenção das métricas de avaliação do modelo. A Tabela 5.1 mostra todos os parâmetros utilizados para o treinamento e teste de cada modelo.

Modelos de ML	**Parâmetros Utilizados**	**Características Utilizadas**
Floresta Isolada (IF)	n_estimators=300, max_samples=0.8, contamination=float(0.2), max_features=2	Ângulo e diferença entre um par de pontos de dados.
Fator Atípico Local (LOF)	n_neighbors=10, novelty=True	Ângulo e diferença entre um par de pontos de dados.
Árvore de Decisão (DT)	max_depth=None, max_leaf_nodes=None	Ângulo e diferença entre um par de pontos de dados.
K-Vizinhos Mais Próximos (KNN)	n_neighbors=5	Ângulo e diferença entre um par de pontos de dados.
Floresta Aleatória (RF)	max_depth=None, max_leaf_nodes=None	Ângulo diferença entre um par de pontos de dados.

Tabela 5.1: Parâmetros e Características Utilizadas nos Modelos de ML

5.2.1 Testes dos Modelos de ML Não Supervisionados e Supervisionados - Curva de Descarga Robô Parado

Na Figura 5.8 é possível observar a acurácia e o F1-score dos modelos de ML não supervisionados, o termo 'FT1' e 'FT2' que aparecem na legenda do gráfico, são abreviações para 'Falha do Tipo 1' e 'Falha do Tipo 2'.

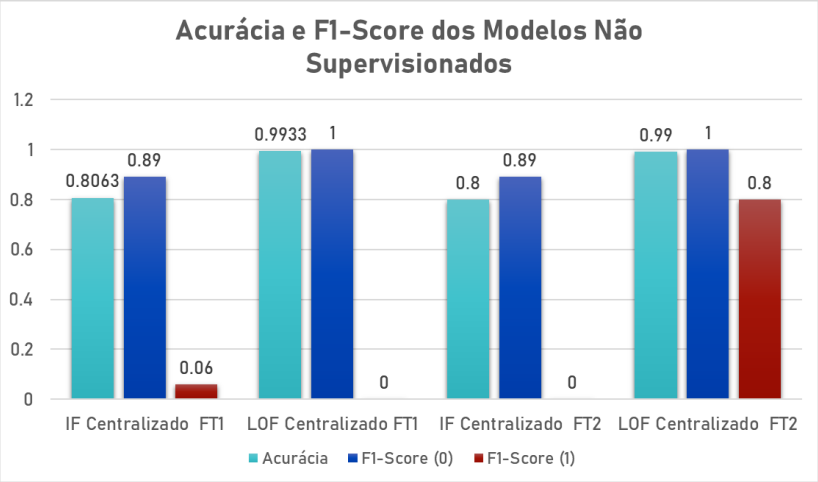


Figura 5.8: Acurácia e F1-score Modelos ML Não Supervisionados.

Pode-se observar que os modelos em geral tiveram uma acurácia e F1-score(0) relativamente bons, porém, com exceção do 'LOF Centralizado FT2', o restante dos modelos apresentaram um péssimo desempenho em relação a métrica de F1-score(1), ou seja, os modelos não conseguiram classificar corretamente os pontos de dados falhos, o que é ruim, pois o objetivo é detectar e classificar corretamente estes pontos.

Para os modelos supervisionados DT, KNN, RF, foi necessária a inserção de rótulos para serem treinados e testados. Na Figura 5.9 pode-se notar o gráfico de acurácia e F1-score dos modelos supervisionados.

É notório o melhor desempenho dos modelos supervisionados em relação aos modelos não supervisionados para este caso de falha em baterias, o conjunto de dados inserido nos modelos para realizar este teste é relacionado com a função pelo tempo f(t) que descreve a

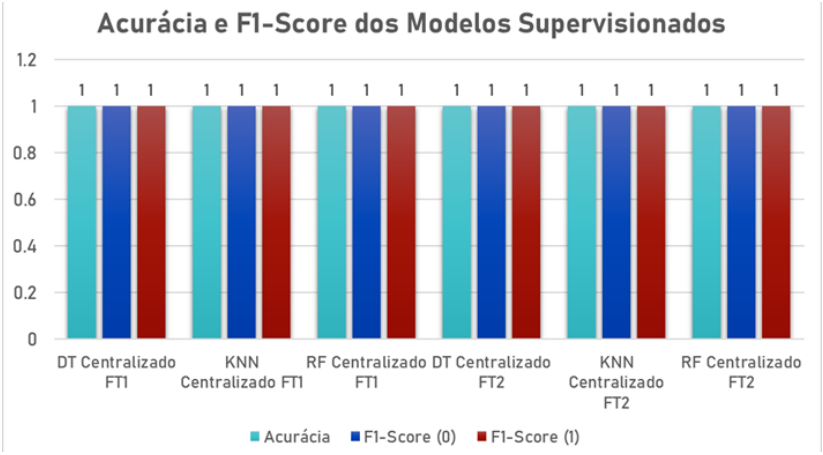


Figura 5.9: Acurácia e F1-score Modelos ML Supervisionados.

curva de descarga robô quando ele se encontra parado. Os modelos supervisionados conseguiram classificar 100% dos pontos de dados corretamente, sendo capazes de diferenciar os pontos que apresentavam comportamento anômalo dos que apresentavam comportamento normal, essa diferenciação ocorreu com base nas características apresentadas aos modelos e que pode ser observada na Tabela 5.1. Essa alta acurácia apresentada pelos modelos supervisionados pode ser consequência da limitação de características, já que o modelo de ML não precisa analisar muitas propriedades e consegue diferenciar totalmente com poucas condições os pontos de dados que estejam bem destacados. Esta acurácia é relacionada ao conjunto de dados de testes, testes utilizando novos dados serão feitos mais a frente.

O desempenho inferior dos modelos não supervisionados pode ter ocorrido pois a quantidade de dados extraídos das baterias para a criação dos *datasets* é limitado neste trabalho, os dados extraídos das baterias se resumem à tensão (V) pelo tempo (s). Muitas vezes a detecção de falhas se torna muito difícil utilizando apenas dados de corrente, tensão e temperatura como citado na subseção 2.4.5. E se torna ainda mais difícil utilizando apenas a medição de tensão (V) e de outras características extraídas dela. Provavelmente os modelos não supervisionados apresentariam melhores resultados de desempenho se houvessem mais características para o modelo ser treinado.

Devido aos fatores de desempenho demonstrados e as limitações de características, este trabalho utilizará apenas modelos supervisionados para realizar os testes de detecção de falhas em baterias.

5.2.2 Testes do Protótipo com Modelo de ML Supervisionado Utilizando Novos Dados - Curva de Descarga Robô Parado

Os testes no protótipo serão feitos utilizando modelos supervisionados com os dados sendo fornecidos diretamente do simulador e, portanto, dados que nunca foram vistos pelos modelos, com o objetivo de analisar o desempenho em novos dados.

Os testes serão divididos de forma que cada modelo de ML treinado para detectar a falha do tipo 1, seja testado em conjunto para cada modelo de ML treinado para a falha do tipo 2, portanto, o protótipo de detecção de falhas receberá dois modelos de ML, cada um treinado para um tipo de falha. O esquema de testes está representado na Figura 5.10.

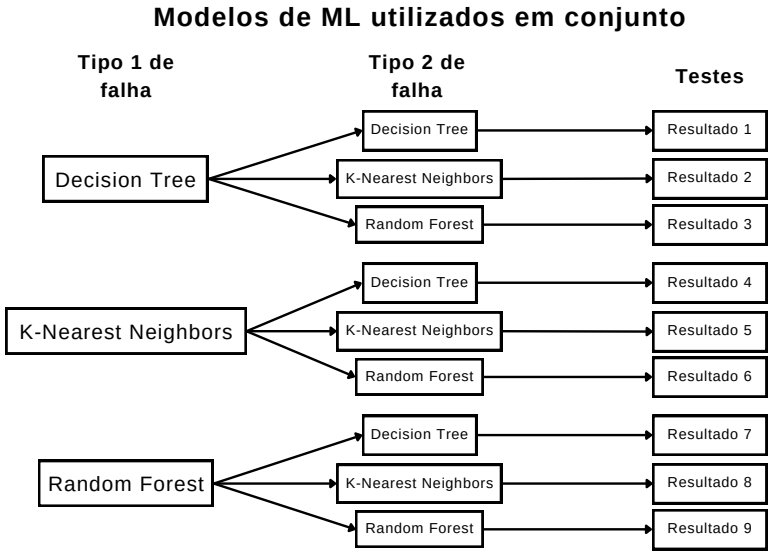


Figura 5.10: Esquema de Testes.

O primeiro teste utiliza o modelo de aprendizado de máquina DT para detectar falhas do tipo 1 e falhas do tipo 2, o segundo teste utiliza o modelo DT para detectar falhas do

tipo 1 e o modelo KNN para detectar o tipo de falha 2, os testes seguirão este padrão, cada teste apresentará as métricas de desempenho.

Na Figura 5.11 é visível as médias da acurácia e F1-score dos testes realizados no protótipo utilizando modelos supervisionados, cada teste envolvia métricas de oito robôs, com isto, foi utilizado as médias destes valores, os resultados dos testes de cada modelo em cada robô serão colocados no Apêndice A.

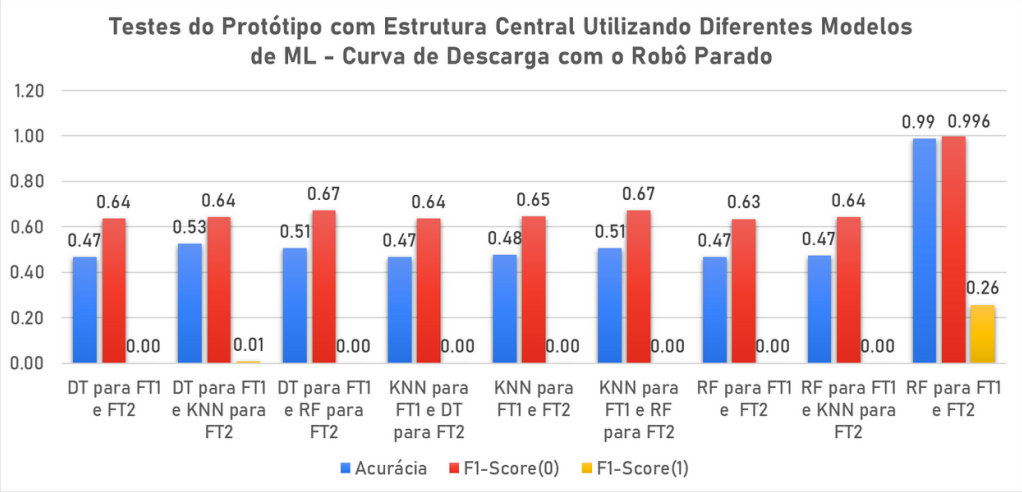


Figura 5.11: Médias das Métricas de Diferentes Modelos de ML Obtidas por Meio dos Testes Realizados com o Protótipo de Estrutura Central.

Observando o gráfico das médias das métricas, é aparente que a maioria dos modelos não obtiveram um desempenho satisfatório para a detecção de falhas em novos dados, mesmo apresentando um desempenho perfeito no teste com os dados generalizados, isso pode ter sido ocasionado pelo ruído introduzido nos sinais de descarga da bateria.

No entanto, o modelo RF para 'FT1' e 'FT2' demonstrou um desempenho significativamente melhor do que os outros, onde apresentou uma acurácia e um F1-score(0) de 0,99, o que significa que o modelo consegue classificar corretamente dados não anômalos 99% das vezes, além disto também apresentou uma média no F1-score(1) de 0,26, na Figura 5.12 é possível observar o teste detalhadamente do desempenho deste modelo.

Mesmo apresentando uma média de 26% no F1-score(1) que pode ser considerado baixa, no robô de número 3 (Robo_3) esta mesma métrica chega a 0,71 o que pode ser

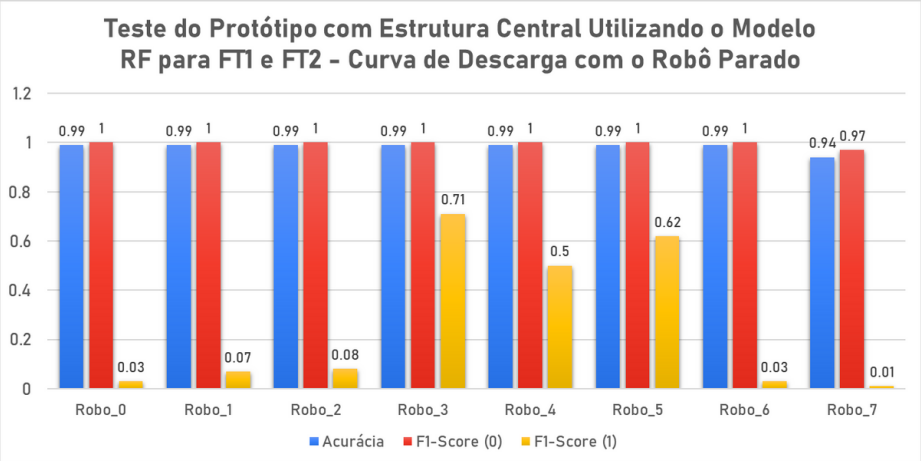


Figura 5.12: Teste do Protótipo com Estrutura Central Utilizando o Modelo RF para FT1 e FT2 - Curva de Descarga com o Robô Parado.

considerado relativamente bom, outro resultado favorável pode ser observado no robô de número 5 (Robo_5) onde o F1-score(1) chega a 0,62.

Utilizando como base este resultado satisfatório do modelo RF para alguns robôs, os modelos customizados de ML que serão construídos para o protótipo com estrutura local serão baseados no modelo RF.

As curvas de descarga demonstram diferentes ruídos e falhas em tempos diferentes, cada cor na Figura 5.13 representa a curva de descarga de bateria de um robô, é possível observar que entre 0 e 250 segundos ocorrem falhas do tipo 1 e na região dos 300 segundos, a falha do tipo 2 é injetada em todos os robôs, da mesma maneira em que foi demonstrado no exemplo de detecção de falhas. A diferença das métricas de F1-score(1) apresentadas na Figura 5.12 também podem estar relacionadas com a quantidade de ruído que foi inserido em cada sinal de bateria, pois, observando a Figura 5.13 é possível perceber que as curvas de descarga que estão mais ruidosas são as do robô 6 e robô 7 (robôs que apresentaram piores métricas), já as que são menos ruidosas são as do robô 3 e robô 5 (robôs que apresentaram melhores métricas).

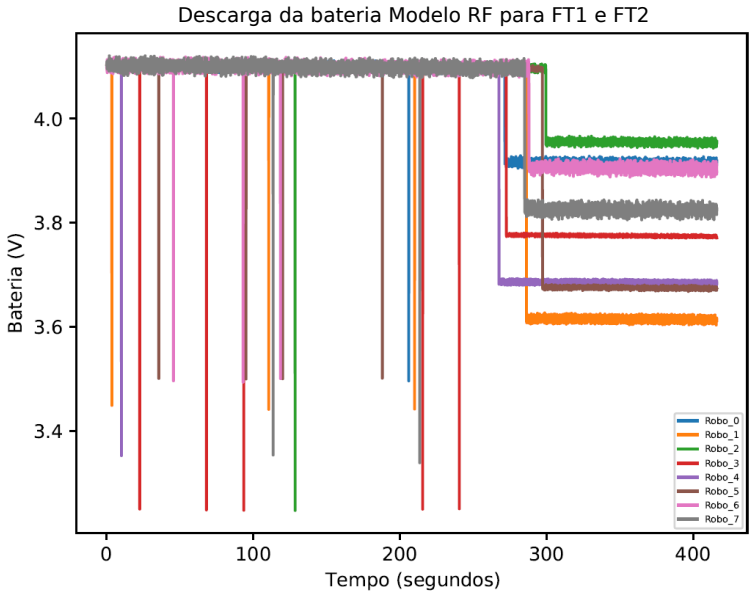


Figura 5.13: Descarga de Bateria Obtida no Teste com o Protótipo de Estrutura Central Utilizando o Modelo RF para FT1 e FT2 - Robô Parado.

5.2.3 Teste do Modelo de ML Supervisionado e do Protótipo Utilizando Novos Dados - Curva de Descarga Robô em Movimento

A comparação feita entre os testes dos modelos supervisionados de ML demonstrou que o modelo de aprendizagem de máquina RF teve um melhor desempenho em relação aos outros, por este motivo, o modelo de aprendizagem de máquina utilizado será o RF para o protótipo com estrutura centralizada com o objetivo de detectar falhas na curva de descarga de bateria dos robôs em movimento.

Assim como os testes dos modelos supervisionados citados anteriormente, este também foi capaz de classificar corretamente todos os pontos de dados do conjunto de teste. As métricas de desempenho do modelo estão dispostas na Figura 5.14.

Tal como o desempenho demonstrado para a estrutura central no contexto da curva de descarga da bateria quando o robô se encontra parado, a estrutura central utilizada

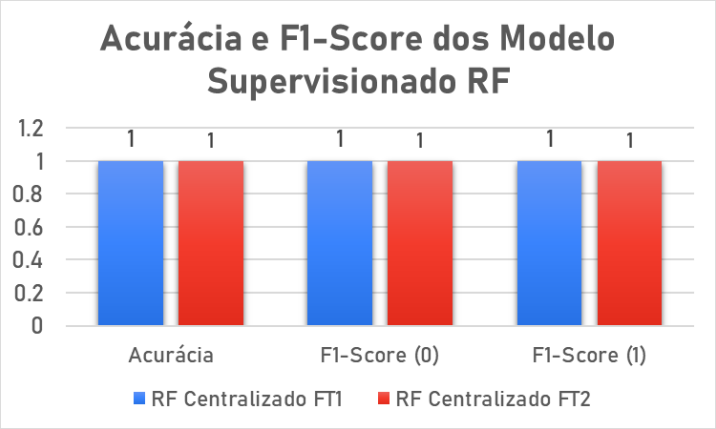


Figura 5.14: Métricas do Teste do Modelo RF - Curva de Descarga do Robô em Movimento.

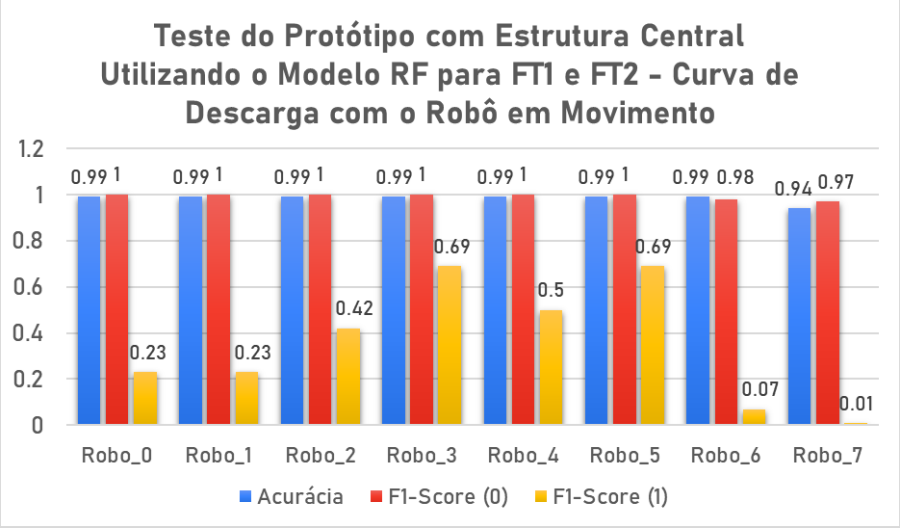


Figura 5.15: Teste do Protótipo com Estrutura Central Utilizando o Modelo RF para FT1 e FT2 - Curva de Descarga do Robô em Movimento.

para quando o robô se encontra em movimento também apresentou um resultado razoável para o F1-score(1) com um valor de 0,69, os valores de acurácia e de F1-score(0) chegaram muito próximos do valor máximo de 1. A Figura 5.15 exibe a acurácia e o F1-score de cada teste para cada robô utilizando o modelo RF.

A Figura 5.16 representa as curvas de descarga de bateria para este teste. Pode-se notar um decaimento maior da tensão na descarga de bateria quando o robô está em movimento em relação a quando o robô está parado.

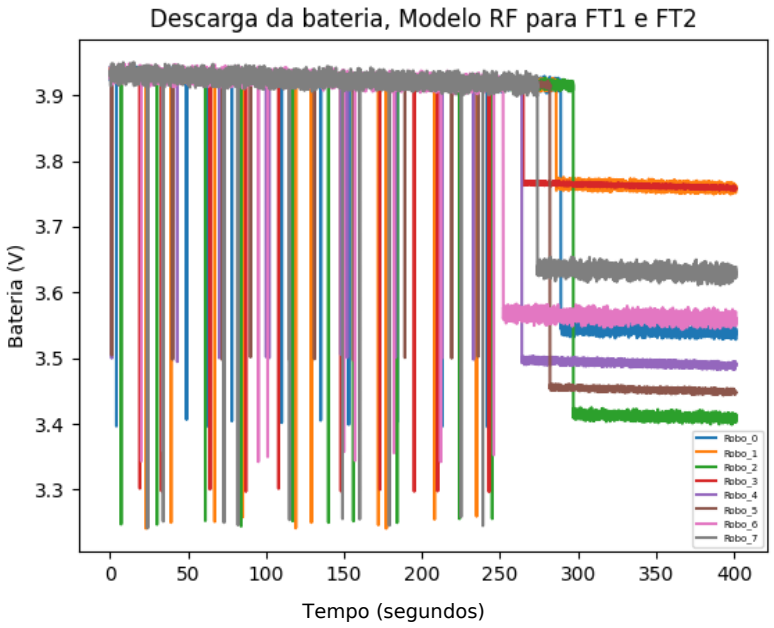


Figura 5.16: Descarga de Bateria Obtida no Teste com o Protótipo de Estrutura Central Utilizando o Modelo RF para FT1 e FT2 - Robô em Movimento.

5.3 Testes dos Modelos de ML e do Protótipo com Estrutura Local

Conforme mencionado anteriormente, os modelos de ML que serão utilizados no protótipo de detecção de falhas com estrutura local terão como base o algoritmo RF, cada modelo

de ML será testado utilizando o conjunto de dados de teste de seu respectivo robô.

5.3.1 Teste do Modelo de ML Aplicado e do Protótipo Utilizando Novos Dados - Curva de Descarga Robô Parado

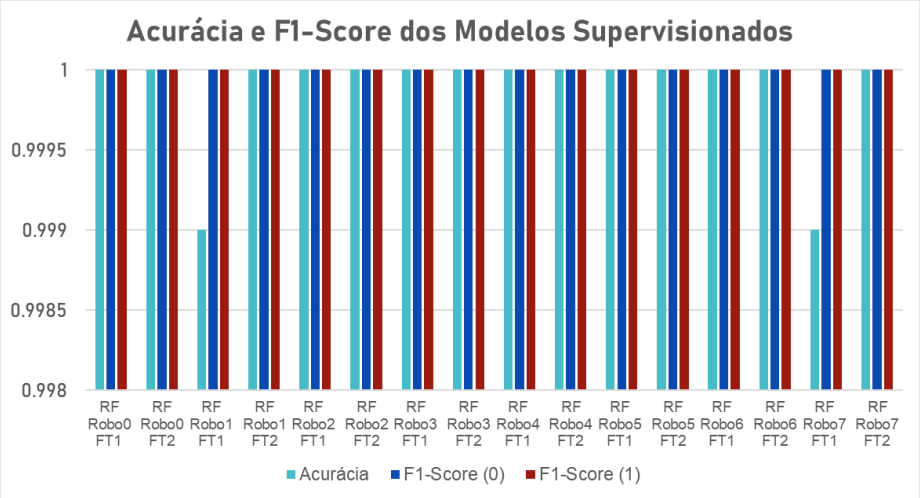


Figura 5.17: Métricas do Treinamento do Modelos de ML para o Protótipo com Estrutura Local.

Na Figura 5.17 está demonstrado o desempenho nos testes dos modelos customizados para cada robô. Com o gráfico de desempenho dos modelos, é notável que assim como o teste dos modelos supervisionados, o modelo RF conseguiu classificar 99,9% dos pontos de dados corretamente, é necessário agora testá-lo com novos dados sendo fornecidos diretamente do simulador.

A Figura 5.18 demonstra os resultados dos testes para cada robô utilizando o protótipo com estrutura local.

O gráfico que demonstra os resultados dos testes para cada robô, evidencia que os robôs de número 6 e 7 (Robo_6 e Robo_7) tiveram um F1-score(1) menor que os demais robôs, isso pode ter sido causado pela diferença de ruído introduzido na curva de descarga

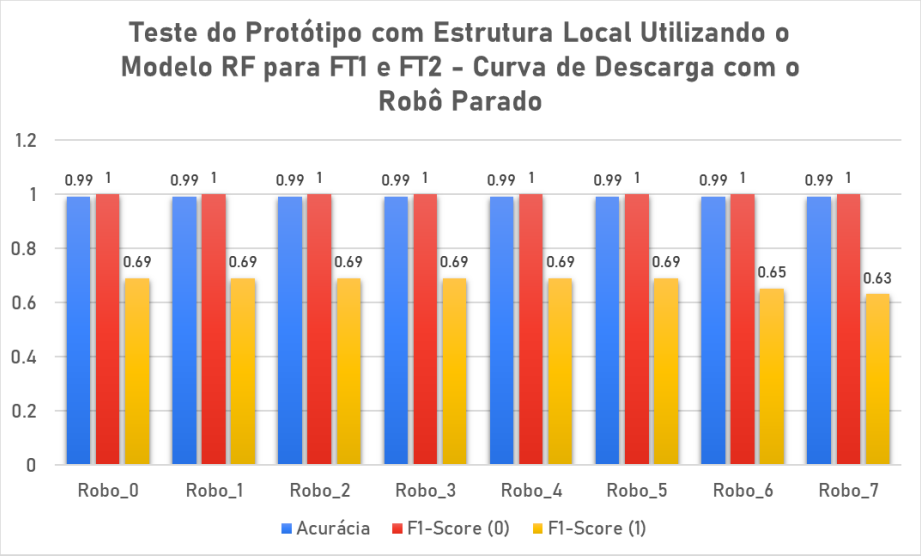


Figura 5.18: Teste do Protótipo com Estrutura Local Utilizando o Modelo RF para FT1 e FT2 - Curva de Descarga do Robô Parado.

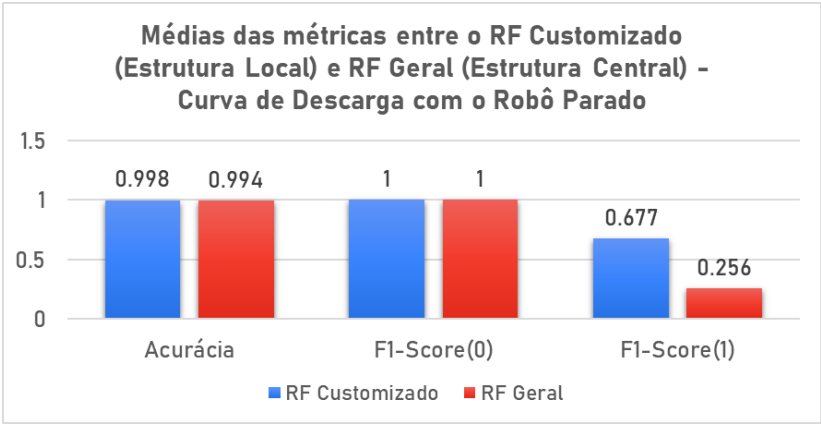


Figura 5.19: Comparação dos Modelos RF Geral e RF Customizado.

da bateria desses robôs, em geral, o modelo apresentou um bom resultado. A comparação entre o modelo RF geral utilizado na estrutura central e o modelo RF customizado utilizado na estrutura local pode ser observada na Figura 5.19. Os gráficos gerados com os testes estão disponíveis no Apêndice A.

Analisando o gráfico de comparação, percebe-se que os valores da acurácia e do F1-score(0) não tiveram uma grande alteração, porém, o F1-score(1) aumentou consideravelmente. Para o modelo RF geral utilizado na estrutura central o valor era de aproximadamente 0,26 e com o modelo RF customizado utilizado na estrutura local este valor chega próximo aos 0,68, um aumento de aproximadamente 160%, ou seja, o valor do F1-score(1) para o modelo customizado utilizado na estrutura local é 2,6 vezes maior que do modelo geral utilizado na estrutura central.

5.3.2 Teste do Modelo de ML Aplicado e do Protótipo Utilizando Novos Dados - Curva de Descarga Robô em Movimento

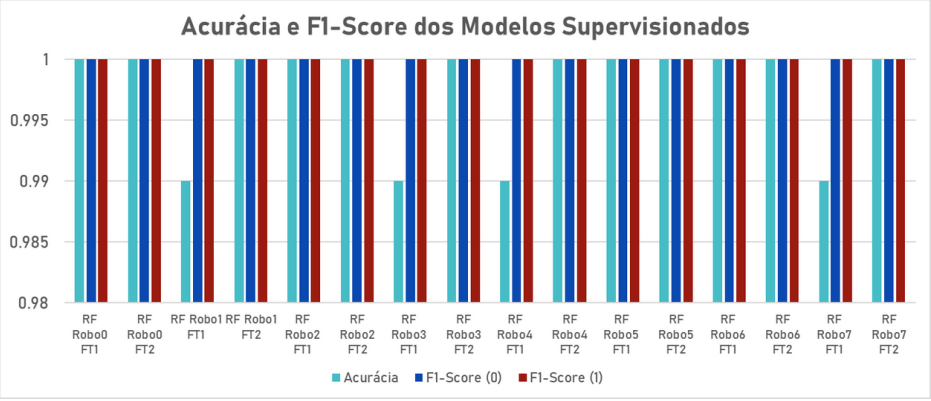


Figura 5.20: Métricas do Treinamento para o Protótipo com Estrutura Local - Curva de Descarga do Robô em Movimento.

As métricas do teste demonstram que o modelo classificou corretamente os pontos de dados do conjunto de teste, assim como os testes anteriores envolvendo modelos de ML

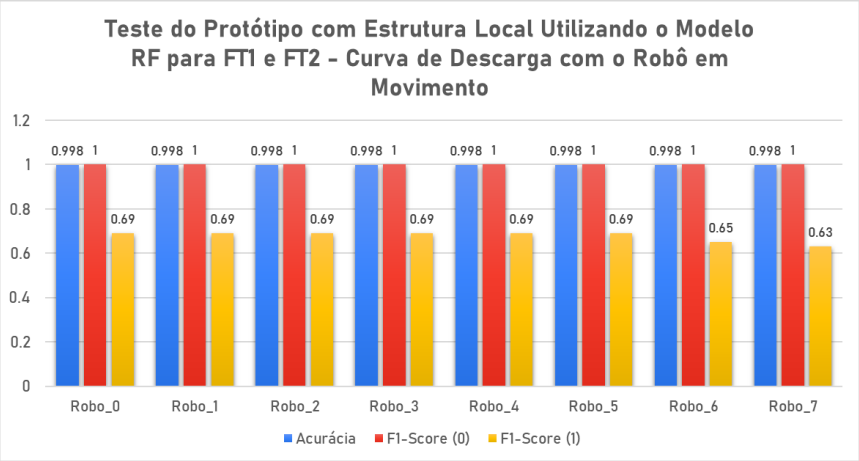


Figura 5.21: Teste do Protótipo com Estrutura Local Utilizando o Modelo RF para FT1 e FT2 - Curva de Descarga do Robô em Movimento

supervisionados. Essas métricas podem ser observadas na Figura 5.20.

A Figura 5.21 demonstra os resultados dos testes para cada robô utilizando o protótipo com estrutura local. As curvas de descarga para cada teste estão no Apêndice A.

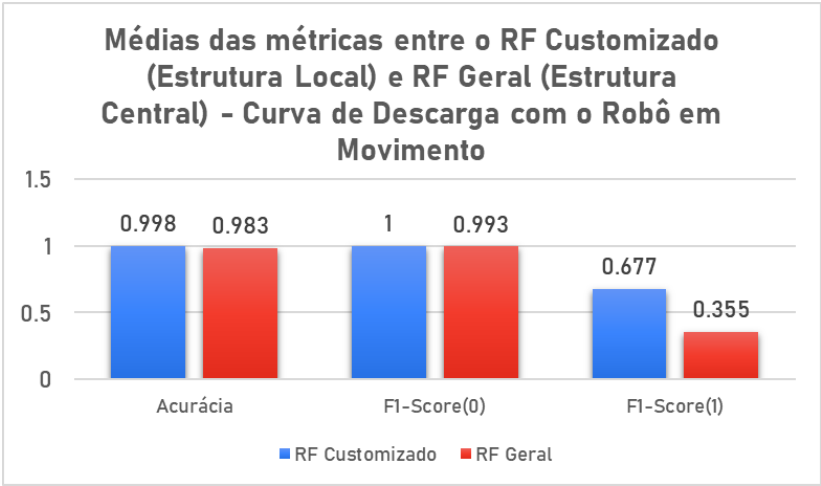


Figura 5.22: Comparação dos modelos RF Geral e RF Customizado.

O gráfico da Figura 5.21 mostra que os Robôs de número 6 e 7 (Robô_6 e Robô_7) também tiveram um F1-score(1) menor que os demais robôs, o mesmo ocorreu no teste da estrutura local para a descarga de bateria com o robô parado, o modelo de ML utilizado na estrutura local apresentou um bom resultado em geral. A comparação entre o modelo RF geral utilizado na estrutura centralizada e o modelo RF customizado utilizado na

estrutura local pode ser observada na Figura 5.22.

Assim como no teste do protótipo com estrutura local feito anteriormente, os valores da acurácia e do F1-Score(0) não se alteraram muito, porém, o F1-score(1) aumentou cerca de 90% em relação ao f1-score do modelo geral, o valor de F1-score do modelo RF geral era de aproximadamente 0,36 e com o modelo RF customizado este valor chega próximo aos 0,68.

5.4 Comparações Gerais

O modelo de ML do tipo não supervisionado que apresentou melhores resultados nos testes foi o LOF para a detecção de falha do tipo 2, com acurácia e F1-score(0) de 0,99 e 1 respectivamente, além do F1-score(1) de 0,8. Para os modelos de ML do tipo supervisionado, todos apresentaram um ótimo resultado, conseguindo classificar todos os pontos de dados presentes no conjunto de dados de teste. A Figura 5.23 demonstra a comparação entre o modelo de aprendizagem de máquina não supervisionada com o modelo supervisionado.

Nos testes dos protótipos utilizando novos dados, o modelo de aprendizagem de máquina supervisionado RF apresentou melhores resultados que os modelos DT e KNN. Para o protótipo com estrutura central utilizando este modelo de ML e relacionado a curva de descarga quando o robô está parado, o protótipo demonstrou desempenhos satisfatórios para alguns robôs. Para este mesmo modelo de ML, porém, customizado para ser utilizado na estrutura local, o protótipo demonstrou desempenhos satisfatórios para todos os robôs e não mais somente para alguns, o mesmo ocorreu em relação a curva de descarga quando o robô está em movimento. Portanto, o melhor desempenho foi apresentado pela estrutura local utilizando o modelo de aprendizagem de máquina RF. As Figuras 5.24 e 5.25 mostram as comparações entre a estrutura central e local utilizando o mesmo modelo de ML para curvas de descargas diferentes.

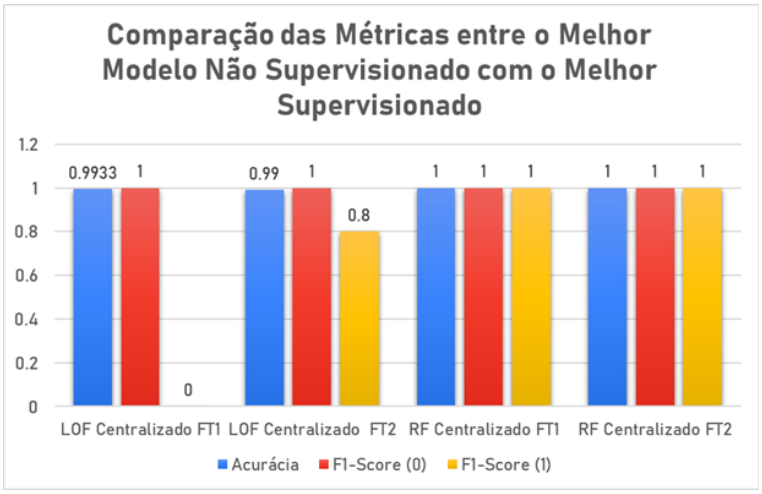


Figura 5.23: Comparação dos Melhores Modelos de ML de Cada Tipo.

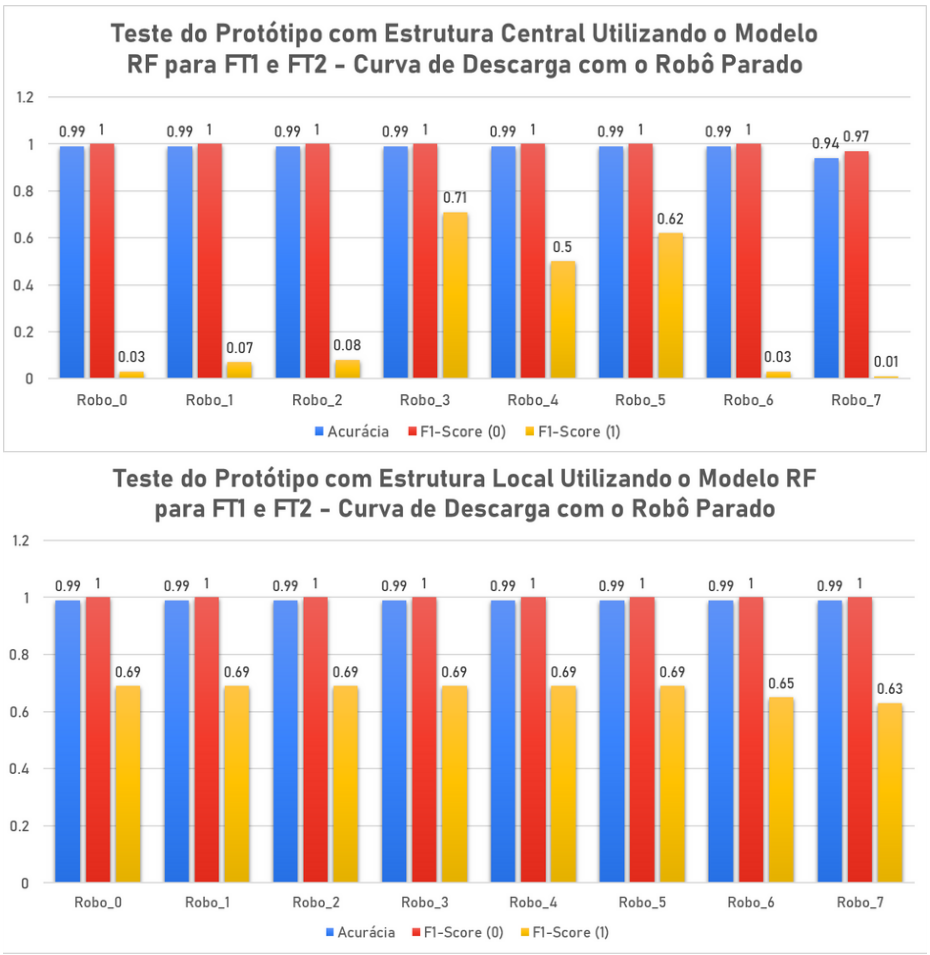


Figura 5.24: Comparação da Estrutura Central e Local Utilizando o Modelo RF para a Curva de Descarga com o Robô Parado.

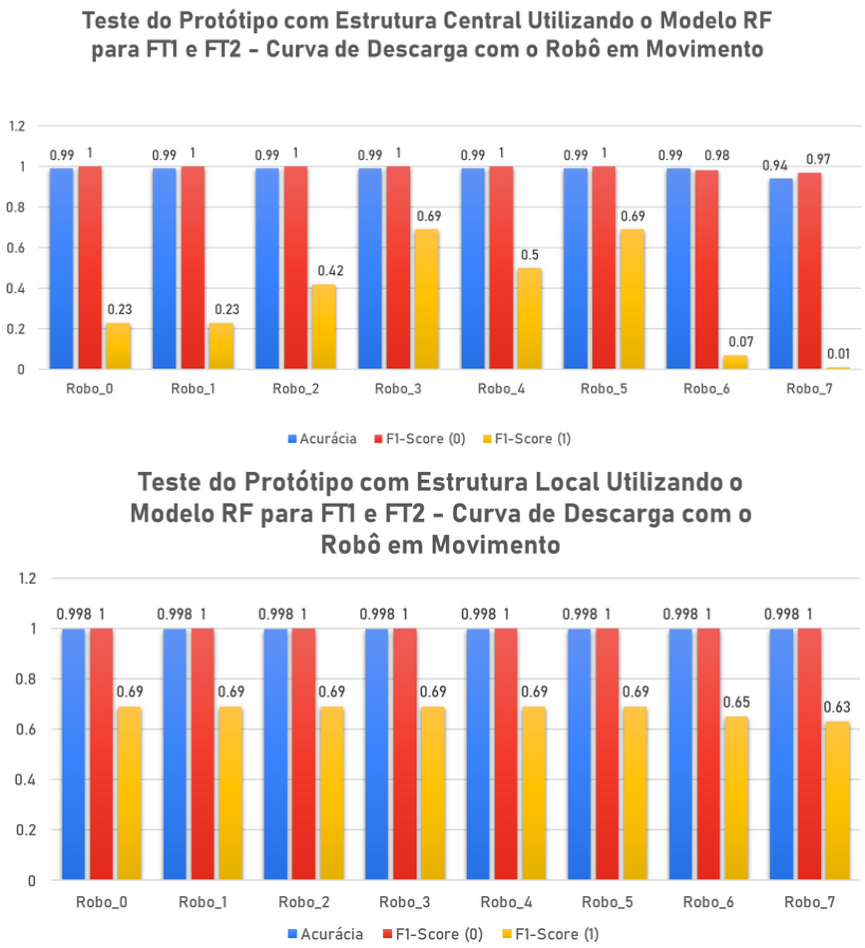


Figura 5.25: Comparação da Estrutura Central e Local Utilizando o Modelo RF para a Curva de Descarga com o Robô em Movimento.

Capítulo 6

Conclusões

Na presente dissertação, foram explorados conceitos de CPSs dentro da I4.0, aspectos gerais sobre falhas e sistemas tolerantes a falhas, detecção de falhas utilizando ML por meio da abordagem orientada a análise de dados e também a construção de um protótipo de detecção de falhas em um CPS laboratorial com ênfase em falhas em baterias.

Ao longo do estudo, foram examinadas diferentes abordagens para a construção de um sistema tolerante a falhas assim como técnicas utilizadas para este propósito, abordando a importância de um sistema ser seguro. Além disso, foram analisados algoritmos de ML de diferentes tipos que poderiam ser utilizados para a detecção de falhas ou *outliers*. A ideia de *outliers* no contexto de baterias foi analisada e dois tipos específicos de falhas em baterias foram utilizados.

No enquadramento da construção do protótipo de detecção de falhas em um CPS laboratorial, foram realizados a aquisição e tratamento de dados relacionados à bateria, adaptação do ambiente de simulação do CPS no CoppeliaSim com base nos dados recolhidos, criação e implementação de duas estruturas (Central e Local) no sistema ROS para a detecção de falhas, além da comparação de desempenho entre os modelos de ML e as estruturas central e local.

Os dados da bateria foram obtidos em dois estados, o primeiro quando o robô se encontrava parado e o segundo quando o robô se encontrava em movimento. Com isso foi possível treinar cada modelo em duas situações, aumentando a precisão.

Em relação ao treinamento dos modelos de ML, a falta de dados relacionados à falhas em baterias de Li-Po tornaram o treinamento e os testes dos modelos tarefas desafiadoras.

Os testes realizados mostraram que os modelos de ML do tipo supervisionado desempenharam melhor em relação aos não supervisionados, devido a esta circunstância, foram utilizados para a detecção de falhas no CPS modelos de ML do tipo supervisionado. A falta de características no conjunto de dados também pode ter sido um dos fatores que fizeram com que um determinado tipo de algoritmo de ML apresentasse desempenho pior do que o outro.

A comparação entre os modelos de aprendizagem de máquina demonstrou a sobreposição do modelo RF em relação aos outros. O modelo RF obteve resultados superiores em relação os modelos DT e KNN no teste do protótipo com estrutura central para a detecção de falhas na descarga de bateria com o robô parado. Com isto, o modelo de aprendizagem de máquina RF foi utilizado para os demais testes.

Para a detecção de falhas na descarga de bateria com o robô parado, o protótipo com estrutura local utilizou o RF com treinamento customizado, o desempenho em relação ao protótipo com estrutura central foi bem positivo, com um aumento de 160% do valor de F1-score(1). Para o caso com o robô em movimento, o protótipo com estrutura local também apresentou melhor desempenho, aumentando cerca de 90% a métrica de F1-score(1) em relação à estrutura central.

Concluindo, a detecção de falhas em CPSs é fundamental para garantir a segurança e eficiência do sistema. Esta pode ser feita de diversas maneiras, neste caso foi utilizado modelos de ML. Dentre os modelos do protótipo, o protótipo com estrutura local utilizando RF apresentou resultados melhores que o protótipo com estrutura central, se mostrando eficaz neste contexto.

Para trabalhos futuros, os protótipos poderiam ser aprimorados para conseguir detectar outros tipos de falhas que possam ocorrer em um CPS e posteriormente diagnosticá-las, como por exemplo a falha de comunicação entre agentes de um CPS, além da aplicação dos protótipos em diferentes tipos de CPSs com o propósito de investigar a aplicabilidade dos métodos propostos em diferentes áreas.

Bibliografia

[1] L. Piardi, P. Costa, A. Oliveira e P. Leitão, "Collaborative Fault Detection and Diagnosis Architecture for Industrial Cyber-Physical Systems," em *2022 IEEE International Conference on Industrial Technology (ICIT)*, 2022, pp. 1–6. DOI: 10.1109/ICIT48603.2022.10002786.

[2] M. Fisher, L. Dennis e M. Webster, "Verifying autonomous systems," *Communications of the ACM*, vol. 56, n.º 9, pp. 84–93, 2013.

[3] T. Weber, "Tolerância a falhas: conceitos e exemplos," jan. de 2003.

[4] N. Bayar, S. Darmoul, S. Hajri-Gabouj e H. Pierreval, "Fault detection, diagnosis and recovery using Artificial Immune Systems: A review," *Engineering Applications of Artificial Intelligence*, vol. 46, pp. 43–57, ago. de 2015. DOI: 10.1016/j.engappai.2015.08.006.

[5] M.-L. Tseng, T. P. T. Tran, H. M. Ha, T.-D. Bui e M. K. Lim, "Sustainable industrial and operation engineering trends and challenges Toward Industry 4.0: A data driven analysis," *Journal of Industrial and Production Engineering*, vol. 38, n.º 8, pp. 581–598, 2021.

[6] S. Klaus, "The Fourth Industrial Revolution," rel. téc., dez. de 2015.

[7] W. D. L. H. Kagermann e W. Wahlster, "Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution," rel. téc., abr. de 2011.

[8] J. B. Sacomano, R. F. Gonçalves, S. H. Bonilla, M. T. da Silva e W. C. Sátyro, *Indústria 4.0*. Editora Blucher, 2018.

[9] K. Schwab, *A quarta revolução industrial*. Edipro, 2019.

[10] E. Lee e S. Seshia, *: Introduction to Embedded Systems-A Cyber-Physical Systems Approach. LeeSeshia. org*, 2011.

[11] Y. Liu, Y. Peng, B. Wang, S. Yao e Z. Liu, "Review on cyber-physical systems," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, n.º 1, pp. 27–40, 2017.

[12] L. Pires, "Sistemas ciber-fisicos: o futuro da Manutenção Industrial," *Obtido em*, 2020.

[13] D. K. Pradhan, *Fault-tolerant computer system design*. Prentice-Hall, Inc., 1996.

[14] T. Lehtonen, "On Fault Tolerance Methods for Networks-on-Chip," jun. de 2023.

[15] R. E. Lyons e W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM journal of research and development*, vol. 6, n.º 2, pp. 200–209, 1962.

[16] A. A. Amin e K. Mahmood-Ul-Hasan, "Hybrid fault tolerant control for air-fuel ratio control of internal combustion gasoline engine using Kalman filters with advanced redundancy," *Measurement and Control -London- Institute of Measurement and Control-*, vol. 52, abr. de 2019. DOI: 10.1177/0020294019842593.

[17] B. W. Johnson, "An introduction to the design and analysis of fault-tolerant systems," *Fault-tolerant computer system design*, vol. 1, pp. 1–84, 1996.

[18] J. Barry, "Design and analysis of fault-tolerant digital systems," *Massachusetts: Addison-Wesley Publishing Company*, 1989.

[19] A. C. dos Oliveira Santos e S. V. Cavalcante, "Inserção Automática de Técnicas de Tolerância a Falhas em Descrições VHDL1,"

[20] J. L. M. do Amaral, "Sistemas imunologicos artificiais aplicados a deteccao de falhas," tese de doutoramento, PUC-Rio, 2006.

[21] R. Isermann e P. Balle, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control engineering practice*, vol. 5, n.º 5, pp. 709–719, 1997.

[22] P. Frank, E. A. Garcıa e B. Köppen-Seliger, "Modelling for fault detection and isolation versus modelling for control," *Mathematics and computers in simulation*, vol. 53, n.º 4-6, pp. 259–271, 2000.

[23] V. Venkatasubramanian, R. Rengaswamy, K. Yin e S. N. Kavuri, "A review of process fault detection and diagnosis: Part I: Quantitative model-based methods," *Computers & chemical engineering*, vol. 27, n.º 3, pp. 293–311, 2003.

[24] I. Sartori, C. Amaro, M. de Souza Jr e M. Embiruçu, "Detecção, diagnóstico e correções de falhas: Uma proposição consistente de definições e terminologias," *Ciência Engenharia*, vol. 21, pp. 41–53, abr. de 2017. DOI: 10.14393/19834071.2012.13183.

[25] J. J. Gertler, "Survey of model-based failure detection and isolation in complex plants," *IEEE Control systems magazine*, vol. 8, n.º 6, pp. 3–11, 1988.

[26] W. G. Fenton, T. M. McGinnity e L. P. Maguire, "Fault diagnosis of electronic systems using intelligent techniques: A review," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 31, n.º 3, pp. 269–281, 2001.

[27] L. Piardi, P. Leitão e A. S. de Oliveira, "Fault-tolerance in cyber-physical systems: literature review and challenges," em *2020 IEEE 18th International Conference on Industrial Informatics (INDIN)*, IEEE, vol. 1, 2020, pp. 29–34.

[28] L. Piardi, P. Leitao, P. Costa e A. S. de Oliveira, "Fault-Tolerance in Cyber-Physical Systems Using Holonic Multi-agent Systems," *Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future: Proceedings of SOHOMA 2021*, vol. 1034, p. 51, 2022.

[29] G. A. Furquim, "Uma abordagem tolerante a falhas para a previsao de desastres naturais baseada em IoT e aprendizado de máquina," tese de doutoramento, Universidade de São Paulo, 2017.

[30] K. F. Á. Okada et al., "Controle tolerante de quadricópteros em cenários com falhas em atuadores e sensores," 2022.

[31] Z.-H. Zhou, *Machine learning*. Springer Nature, 2021.

[32] B. Mahesh, "Machine learning algorithms-a review," *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020.

[33] Ravish Raj, *Supervised, Unsupervised and Semi-Supervised Learning With Real-life Usecase*, Disponível em: `https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning`, Acessado em: 20 de junho de 2023.

[34] Javatpoint, *Decision Tree Classification Algorithm*, Disponível em: `https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm`, Acessado em: 20 de junho de 2023.

[35] X. Yao, L. Tham e F. Dai, "Landslide susceptibility mapping based on support vector machine: a case study on natural slopes of Hong Kong, China," *Geomorphology*, vol. 101, n.º 4, pp. 572–582, 2008.

[36] X. Yao, L. Tham e F. Dai, "Landslide susceptibility mapping based on Support Vector Machine: A case study on natural slopes of Hong Kong, China," *Geomorphology*, vol. 101, n.º 4, pp. 572–582, 2008, ISSN: 0169-555X. DOI: `https://doi.org/10.1016/j.geomorph.2008.02.011`. URL: `https://www.sciencedirect.com/science/article/pii/S0169555X08000585`.

[37] Y. Huang e L. Zhao, "Review on landslide susceptibility mapping using support vector machines," *Catena*, vol. 165, pp. 520–529, 2018.

[38] B. Scholkopf, S. Mika, C. J. Burges et al., "Input space versus feature space in kernel-based methods," *IEEE transactions on neural networks*, vol. 10, n.º 5, pp. 1000–1017, 1999.

[39] D. M. Tax, A. Ypma e R. P. Duin, "Pump failure detection using support vector data descriptions," em *Advances in Intelligent Data Analysis: Third International Symposium, IDA-99 Amsterdam, The Netherlands, August 9–11, 1999 Proceedings 3*, Springer, 1999, pp. 415–425.

[40] M. Yan, K. Liu, Z. Guan, X. Xinkai, X. Qian e H. Bao, "Background Augmentation Generative Adversarial Networks (BAGANs): Effective Data Generation Based on GAN-Augmented 3D Synthesizing," *Symmetry*, vol. 10, p. 734, dez. de 2018. DOI: 10.3390/sym10120734.

[41] K. P. Sinaga e M.-S. Yang, "Unsupervised K-means clustering algorithm," *IEEE access*, vol. 8, pp. 80 716–80 727, 2020.

[42] M. Alhawarat e M. Hegazi, "Revisiting k-means and topic modeling, a comparison study to cluster arabic documents," *IEEE Access*, vol. 6, pp. 42 740–42 749, 2018.

[43] Y. Meng, J. Liang, F. Cao e Y. He, "A new distance with derivative information for functional k-means clustering algorithm," *Information Sciences*, vol. 463, pp. 166–185, 2018.

[44] Z. Lv, T. Liu, C. Shi, J. A. Benediktsson e H. Du, "Novel land cover change detection method based on K-means clustering and adaptive majority voting using bitemporal remote sensing images," *Ieee Access*, vol. 7, pp. 34 425–34 437, 2019.

[45] J. Zhu, Z. Jiang, G. D. Evangelidis, C. Zhang, S. Pang e Z. Li, "Efficient registration of multi-view point sets by K-means clustering," *Information Sciences*, vol. 488, pp. 205–218, 2019.

[46] Alan Jeffares, *K-means: A Complete Introduction*, Disponível em: ----https://towardsdatascience.com/k-means-a-complete-introduction-1702af9cd8c, Acessado em: 20 de junho de 2023.

[47] J. E. Van Engelen e H. H. Hoos, "A survey on semi-supervised learning," *Machine learning*, vol. 109, n.º 2, pp. 373–440, 2020.

[48] O. Sagi e L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, n.º 4, e1249, 2018.

[49] Mohammed Alhamid, *Ensemble Models: What Are They and When Should You Use Them?* Disponível em: `https://builtin.com/machine-learning/ensemble-model`, Acessado em: 20 de junho de 2023.

[50] Y. Amit e D. Geman, "Randomized Inquiries About Shape: An Application to Handwritten Digit Recognition.," Chicago Univ IL Dept of Statistics, rel. téc., 1994.

[51] T. K. Ho, "Random decision forests," em *Proceedings of 3rd international conference on document analysis and recognition*, IEEE, vol. 1, 1995, pp. 278–282.

[52] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[53] C. Procópio, "Aplicação do Algoritmo Random Forest como Classificador de Padrões de Falhas em Rolamentos de Motores de Indução,"

[54] TIBCO, *O que é uma floresta aleatória?* Disponível em: `https://www.tibco.com/pt-br/reference-center/what-is-a-random-forest`, Acessado em: 20 de junho de 2023.

[55] W. I. D. Mining, "Data mining: Concepts and techniques," *Morgan Kaufinann*, vol. 10, pp. 559–569, 2006.

[56] B. Gras, A. Brun e A. Boyer, "Identifying grey sheep users in collaborative filtering: a distribution-based technique," em *Proceedings of the 2016 conference on user modeling adaptation and personalization*, 2016, pp. 17–26.

[57] S. Ramaswamy, R. Rastogi e K. Shim, "Efficient algorithms for mining outliers from large data sets," em *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 427–438.

[58] D. M. Hawkins, *Identification of outliers*. Springer, 1980, vol. 11.

[59] G. R. Castanhel, T. Heinrich, F. Ceschin e C. A. Maziero, "Detecçao de anomalias: Estudo de técnicas de identificaçao de ataques em um ambiente de contêiner," em *Anais Estendidos do XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, SBC, 2020, pp. 169–182.

[60] D. F. F. D. Fernandes, "Uma Abordagem para Manutenção Preditiva baseada em Sistemas Multiagente e Machine Learning," tese de doutoramento, 2020.

[61] K. Vassakis, E. Petrakis e I. Kopanakis, "Big data analytics: applications, prospects and challenges," *Mobile big data: A roadmap from models to technologies*, pp. 3–20, 2018.

[62] F. T. Liu, K. M. Ting e Z.-H. Zhou, "Isolation forest," em *2008 eighth ieee international conference on data mining*, IEEE, 2008, pp. 413–422.

[63] Y. Regaya, F. Fadli e A. Amira, "Point-Denoise: Unsupervised outlier detection for 3D point clouds enhancement," *Multimedia Tools and Applications*, vol. 80, pp. 1–17, jul. de 2021. DOI: 10.1007/s11042-021-10924-x.

[64] M. M. Breunig, H.-P. Kriegel, R. T. Ng e J. Sander, "LOF: identifying density-based local outliers," em *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[65] Scikit-Learn, *Outlier detection with Local Outlier Factor (LOF)*, Disponível em: https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html, Acessado em: 20 de junho de 2023.

[66] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola e R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, n.º 7, pp. 1443–1471, 2001.

[67] Garima, *One Class SVM(OC-SVM)*, Disponível em: https://medium.com/@mail.garima7/one-class-svm-oc-svm-9ade87da6b10, Acessado em: 20 de junho de 2023.

[68] E. M. Knorr, R. T. Ng e V. Tucakov, "Distance-based outliers: algorithms and applications," *The VLDB Journal*, vol. 8, n.º 3, pp. 237–253, 2000.

[69] H.-P. Kriegel, M. Schubert e A. Zimek, "Angle-based outlier detection in high-dimensional data," em *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008, pp. 444–452.

[70] H.-P. Kriegel, P. Kröger e A. Zimek, "Outlier detection techniques," *Tutorial at KDD*, vol. 10, pp. 1–76, 2010.

[71] P. J. Rousseeuw e A. M. Leroy, *Robust regression and outlier detection*. John wiley & sons, 2005.

[72] J. F. Hair, "Multivariate data analysis," 2009.

[73] Jan Kalina, *Robust Classifiers in Multivariate Statistics and Machine Learning*, Disponível em: `https://kizi.vse.cz/wp-content/uploads/old/kalina.pdf`, Acessado em: 20 de junho de 2023.

[74] X. Hu, K. Zhang, K. Liu, X. Lin, S. Dey e S. Onori, "Advanced Fault Diagnosis for Lithium-Ion Battery Systems: A Review of Fault Mechanisms, Fault Features, and Diagnosis Procedures," *IEEE Industrial Electronics Magazine*, vol. 14, n.º 3, pp. 65–91, 2020. DOI: `10.1109/MIE.2020.2964814`.

[75] W. Chao, Z. Chunbo, S. Jinlei e J. Jianhu, "Fault mechanism study on Li-ion battery at over-discharge and its diagnosis approach," *IET Electrical Systems in Transportation*, vol. 7, n.º 1, pp. 48–54, 2017.

[76] C. Yuan, L. Wang, S. Yin e J. Xu, "Generalized separator failure criteria for internal short circuit of lithium-ion battery," *Journal of Power Sources*, vol. 467, p. 228 360, 2020.

[77] Z. Sun, Y. Han, Z. Wang et al., "Detection of voltage fault in the battery system of electric vehicles using statistical analysis," *Applied Energy*, vol. 307, p. 118 172, 2022, ISSN: 0306-2619. DOI: `https://doi.org/10.1016/j.apenergy.2021.`

118172. URL: https://www.sciencedirect.com/science/article/pii/S0306261921014434.

[78] G. Zhu, T. Sun, Y. Xu, Y. Zheng e L. Zhou, "Identification of Internal Short-Circuit Faults in Lithium-Ion Batteries Based on a Multi-Machine Learning Fusion," *Batteries*, vol. 9, n.º 3, p. 154, 2023.

[79] M. Schmid, J. Kleiner e C. Endisch, "Early detection of internal short circuits in series-connected battery packs based on nonlinear process monitoring," *Journal of Energy Storage*, vol. 48, p. 103 732, 2022.

[80] Z. Chen, K. Xu, J. Wei e G. Dong, "Voltage fault detection for lithium-ion battery pack using local outlier factor," *Measurement*, vol. 146, pp. 544–556, 2019.

[81] M. Ouyang, M. Zhang, X. Feng et al., "Internal short circuit detection for battery pack using equivalent parameter and consistency method," *Journal of Power Sources*, vol. 294, pp. 272–283, 2015.

[82] B. Xia, Y. Shang, T. Nguyen e C. Mi, "A correlation based fault detection method for short circuits in battery packs," *Journal of power Sources*, vol. 337, pp. 1–10, 2017.

[83] J. Hong, Z. Wang e P. Liu, "Big-data-based thermal runaway prognosis of battery systems for electric vehicles," *Energies*, vol. 10, n.º 7, p. 919, 2017.

[84] Z. Wang, J. Hong, P. Liu e L. Zhang, "Voltage fault diagnosis and prognosis of battery systems based on entropy and Z-score for electric vehicles," *Applied energy*, vol. 196, pp. 289–302, 2017.

[85] J. D. Gotz, G. C. Guerrero, J. R. H. de Queiroz, E. R. Viana e M. Borsato, "Diagnosing failures in lithium-ion batteries with Machine Learning techniques," *Engineering Failure Analysis*, p. 107 309, 2023.

[86] A. Samanta, S. Chowdhuri e S. S. Williamson, "Machine learning-based data-driven fault detection/diagnosis of lithium-ion battery: A critical review," *Electronics*, vol. 10, n.º 11, p. 1309, 2021.

[87] R. Yang, R. Xiong, H. He e Z. Chen, "A fractional-order model-based battery external short circuit fault diagnosis approach for all-climate electric vehicles application," *Journal of cleaner production*, vol. 187, pp. 950–959, 2018.

[88] T. Kim, A. Adhikaree, R. Pandey et al., "Outlier mining-based fault diagnosis for multiceli lithium-ion batteries using a low-priced microcontroller," em *2018 IEEE applied power electronics conference and exposition (APEC)*, IEEE, 2018, pp. 3365–3369.

[89] L. Piardi, V. C. Kalempa, M. Limeira, A. S. de Oliveira e P. Leitão, "Arena—augmented reality to enhanced experimentation in smart warehouses," *Sensors*, vol. 19, n.º 19, p. 4308, 2019.

[90] M. A. Limeira, L. Piardi, V. C. Kalempa, A. S. de Oliveira e P. Leitão, "WsBot: A Tiny, Low-Cost Swarm Robot for Experimentation on Industry 4.0," em *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, 2019, pp. 293–298. DOI: 10.1109/LARS-SBR-WRE48964.2019.00058.

[91] M. Limeira, L. Piardi, V. Cremer Kalempa, A. Schneider e P. Leitão, "Augmented Reality System for Multi-robot Experimentation in Warehouse Logistics," em *Robot 2019: Fourth Iberian Robotics Conference: Advances in Robotics, Volume 1*, Springer, 2020, pp. 319–330.

[92] L. Joseph e J. Cacace, *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*. Packt Publishing Ltd, 2018.

[93] M. Quigley, K. Conley, B. Gerkey et al., "ROS: an open-source Robot Operating System," em *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[94] I. Malavolta, G. A. Lewis, B. Schmerl, P. Lago e D. Garlan, "Mining guidelines for architecting robotics software," *Journal of Systems and Software*, vol. 178, p. 110 969, 2021.

[95] E. Rohmer, S. P. N. Singh e M. Freese, "V-REP: A versatile and scalable robot simulation framework," em *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326. DOI: 10.1109/IROS.2013.6696520.

[96] V. Venkatasubramanian, R. Rengaswamy e S. N. Kavuri, "A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies," *Computers & chemical engineering*, vol. 27, n.º 3, pp. 313–326, 2003.

[97] E. Ostertagová, "Modelling using polynomial regression," *Procedia Engineering*, vol. 48, pp. 500–506, 2012.

[98] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[99] H. O. Rodrigues, "Aprendizagem significativa de sistemas lineares através dos coeficientes angular e linear da reta utilizando como recurso didático o geoplano," *Revista Conapesc*, 2018.

[100] F. Ertam e M. Kaya, "Classification of firewall log files with multiclass support vector machine," em *2018 6th International symposium on digital forensic and security (ISDFS)*, IEEE, 2018, pp. 1–4.

[101] K. M. Ting, "Precision and Recall," em *Encyclopedia of Machine Learning*, C. Sammut e G. I. Webb, eds. Boston, MA: Springer US, 2010, pp. 781–781, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_652. URL: https://doi.org/10.1007/978-0-387-30164-8_652.

[102] Z. C. Lipton, C. Elkan e B. Naryanaswamy, "Optimal thresholding of classifiers to maximize F1 measure," em *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II 14*, Springer, 2014, pp. 225–239.

[103] L. Buitinck, G. Louppe, M. Blondel et al., "API design for machine learning software: experiences from the scikit-learn project," em *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[104] Bernardo Lago, *Matriz de confusão*, Disponível em: `https://medium.com/@bernardolago/matriz-de-confusÃčo-7c0e36468323`, Acessado em: 20 de junho de 2023.

[105] *pickle — Python object serialization*, `https://docs.python.org/3/library/pickle.html`, Acesso em: maio 2023, Python Software Foundation, 2021.

Apêndice A

Resultados Complementares

A.1 Resultados Detalhados dos Testes do Protótipo com Estrutura Central Utilizando Modelos de ML - Curva de Descarga com o Robô Parado

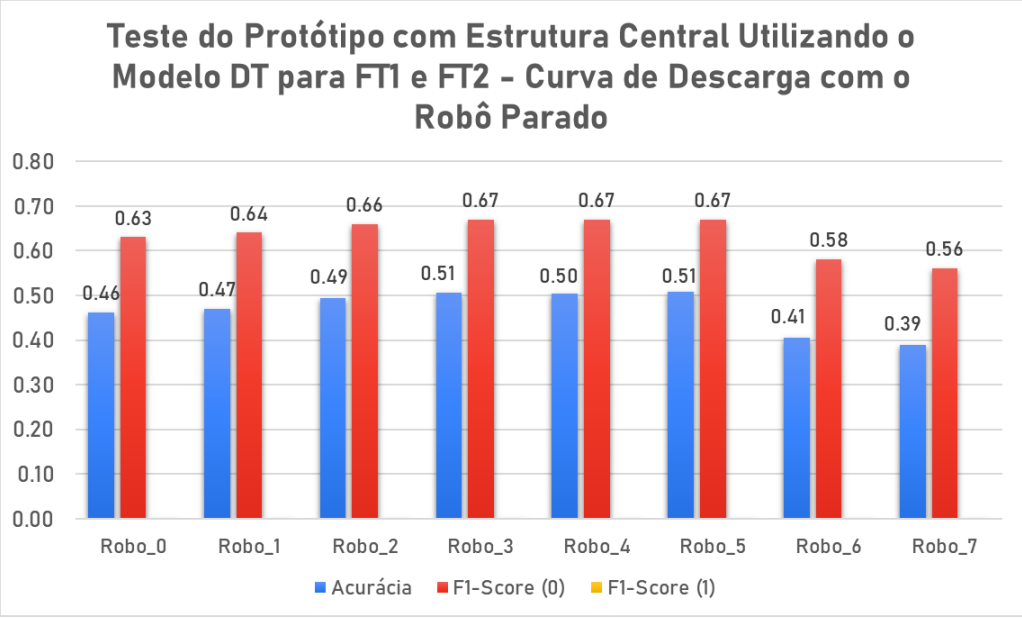


Figura A.1: Teste do Protótipo com Estrutura Central Utilizando o Modelo DT para FT1 e FT2.

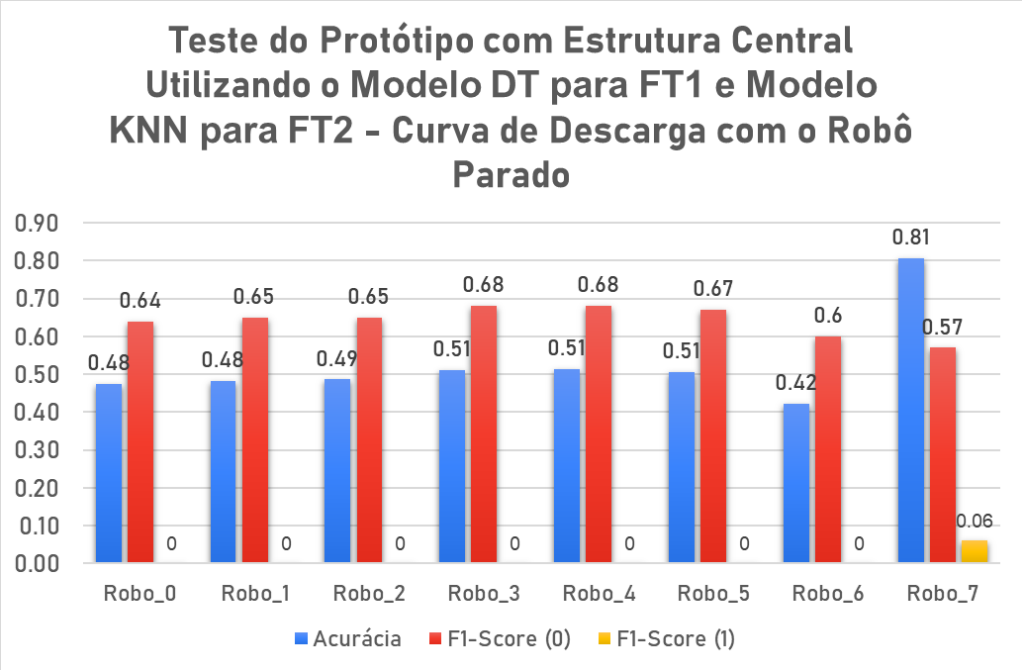


Figura A.2: Teste do Protótipo com Estrutura Central Utilizando o Modelo DT para FT1 e o Modelo KNN para FT2.

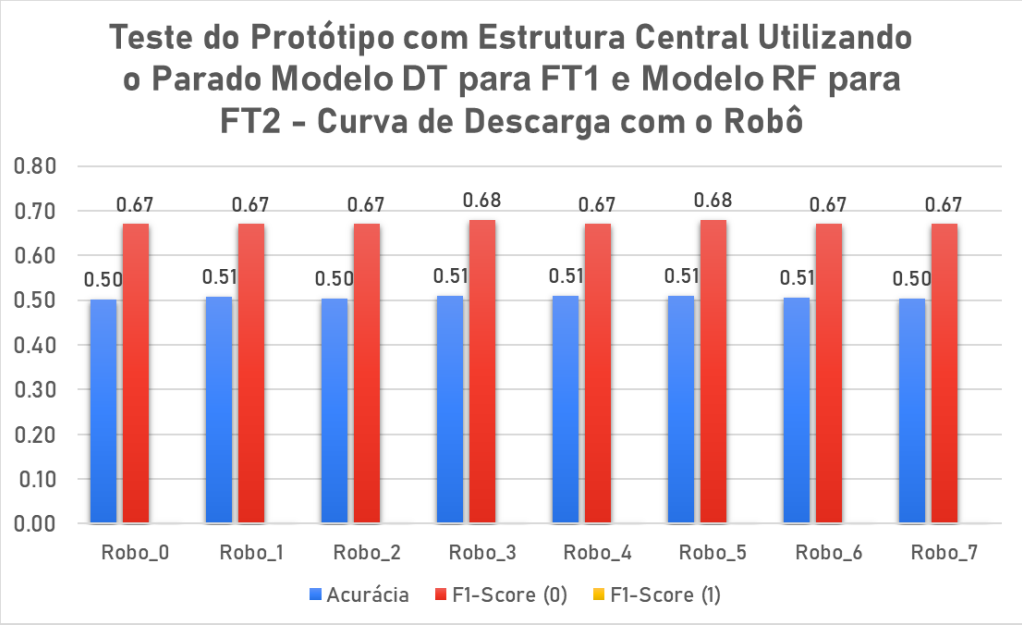


Figura A.3: Teste do Protótipo com Estrutura Central Utilizando o Modelo DT para FT1 e o Modelo RF para FT2.

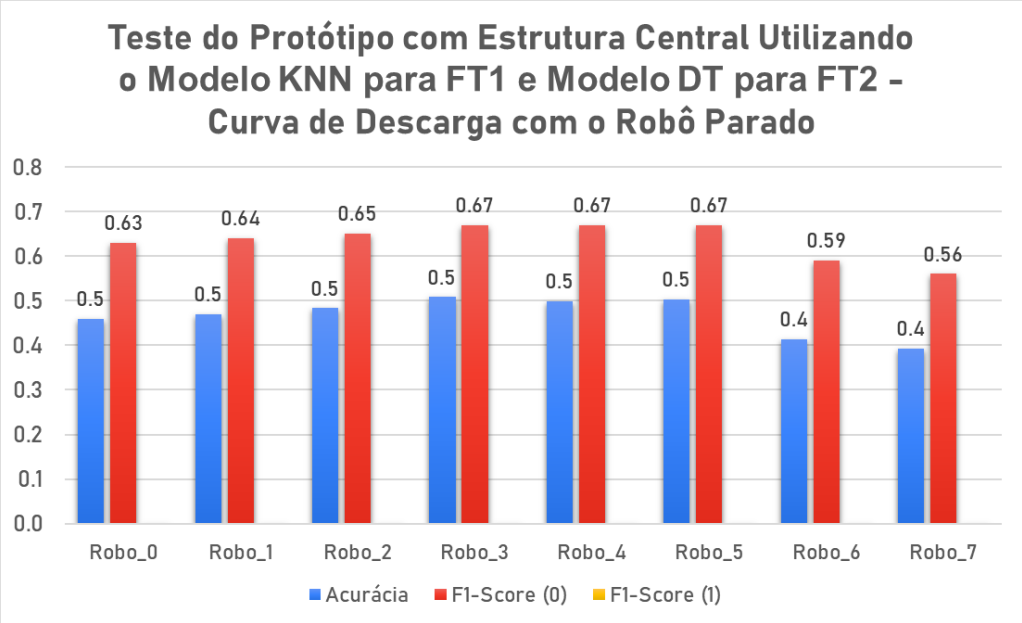


Figura A.4: Teste do Protótipo com Estrutura Central Utilizando o Modelo KNN para FT1 e o Modelo DT para FT2.

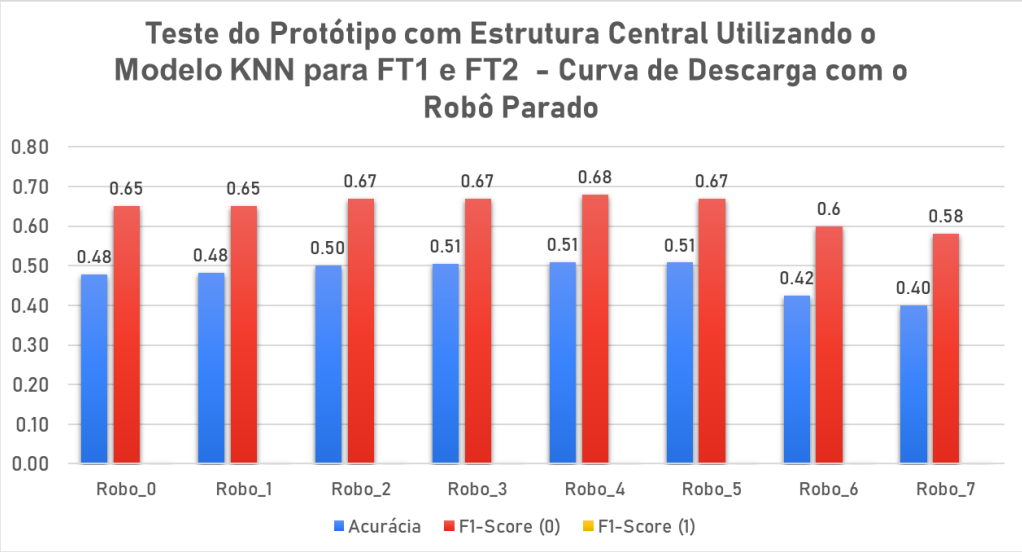


Figura A.5: Teste do Protótipo com Estrutura Central Utilizando o Modelo KNN para FT1 e FT2.

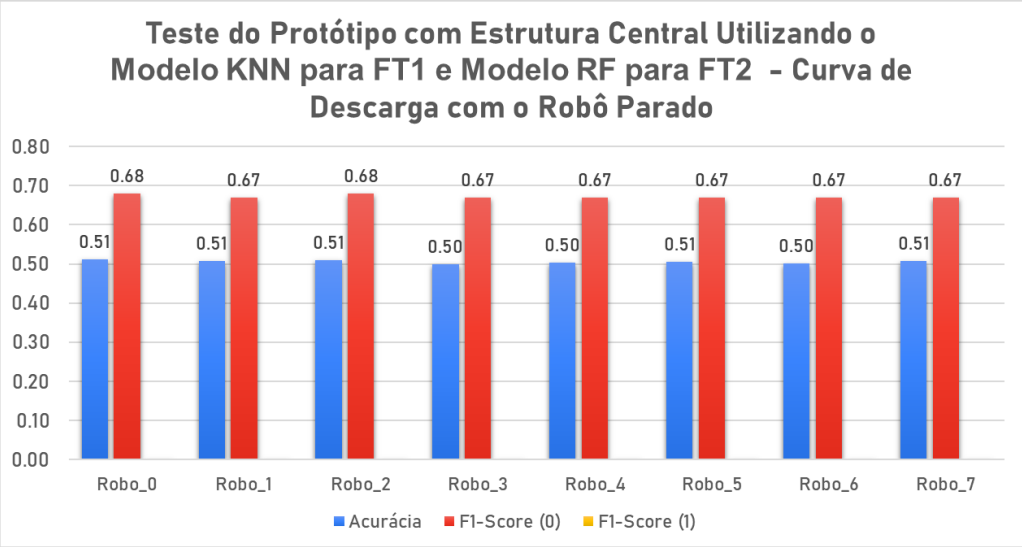


Figura A.6: Teste do Protótipo com Estrutura Central Utilizando o Modelo KNN para FT1 e o Modelo RF para FT2.

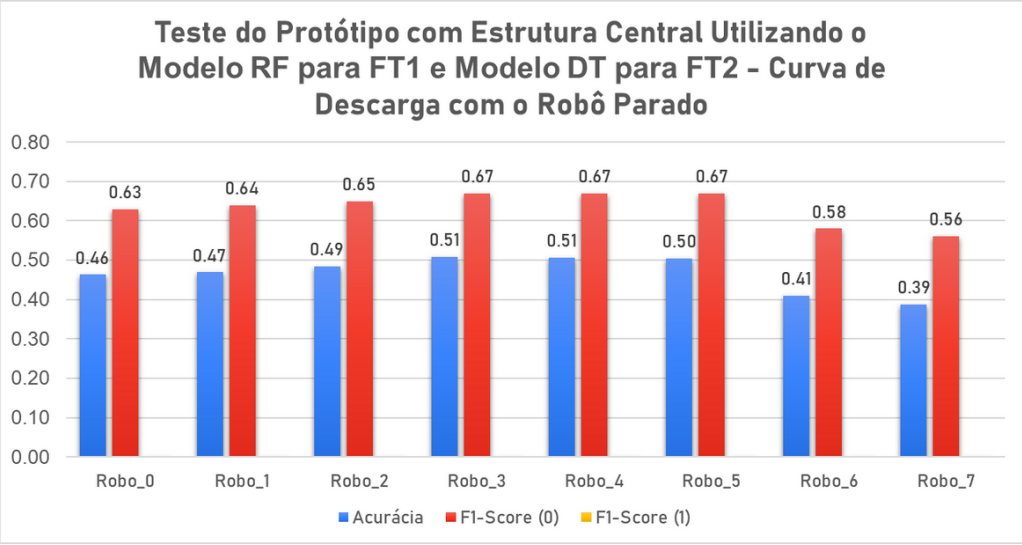


Figura A.7: Teste do Protótipo com Estrutura Central Utilizando o Modelo RF para FT1 e o Modelo DT para FT2.

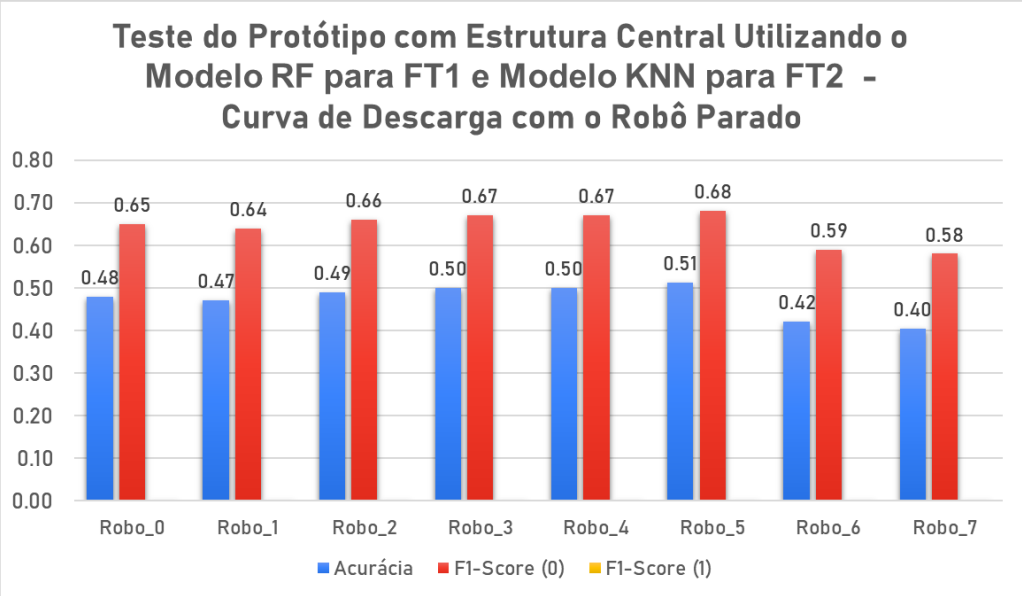


Figura A.8: Teste do Protótipo com Estrutura Central Utilizando o Modelo RFF para FT1 e o Modelo KNN para FT2.

A.2 Curvas das Descargas de Bateria Obtidas nos Testes dos Protótipos

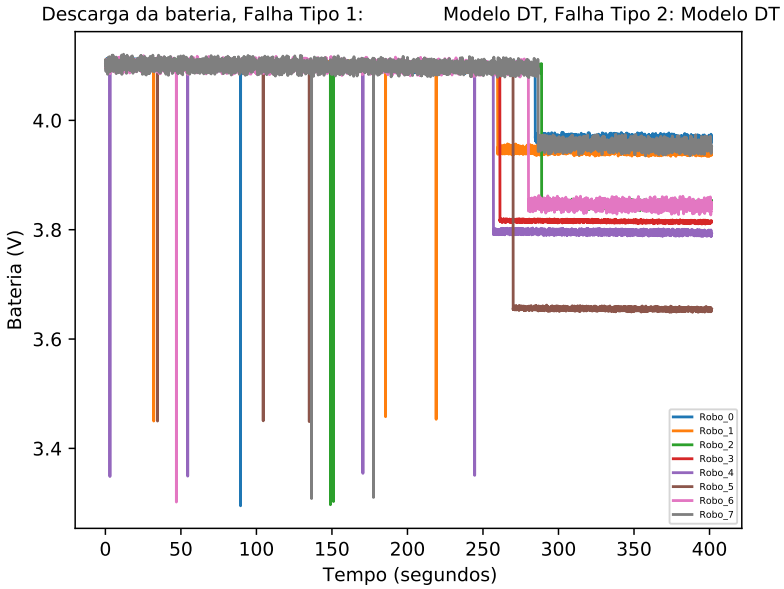


Figura A.9: Descarga da Bateria Teste Protótipo Estrutura Central, Modelo DT para FT1 e FT2 - Robô Parado.

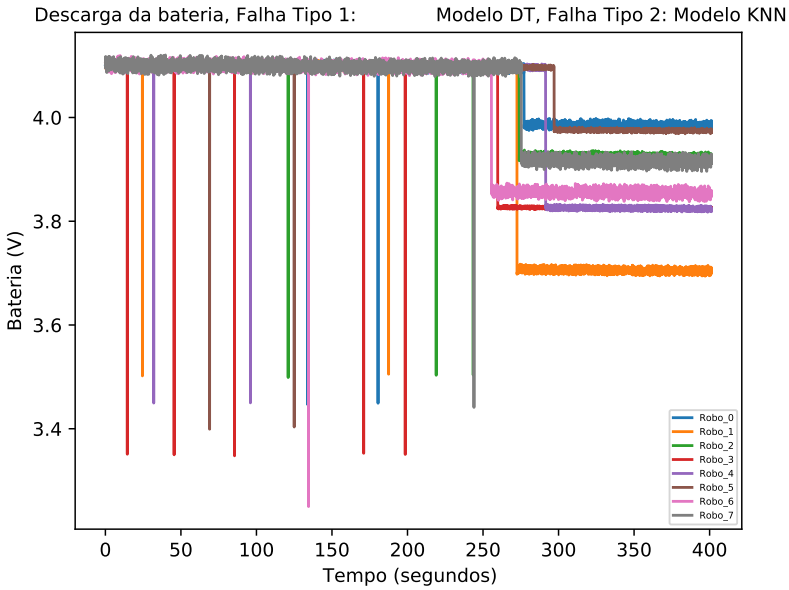


Figura A.10: Descarga da Bateria Teste Protótipo Estrutura Central, Modelo DT para FT1 e Modelo KNN para FT2 - Robô Parado.

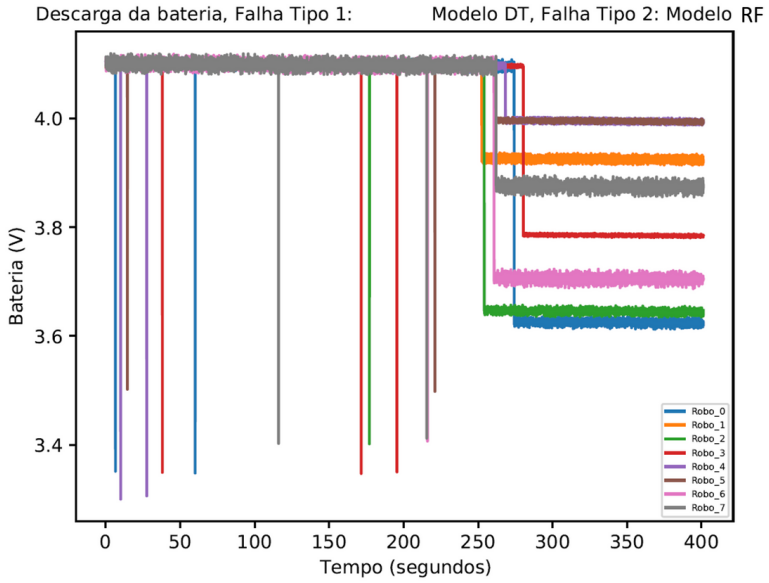


Figura A.11: Descarga da Bateria Teste Protótipo Estrutura Central, Modelo DT para FT1 e Modelo RF para FT2 - Robô Parado.

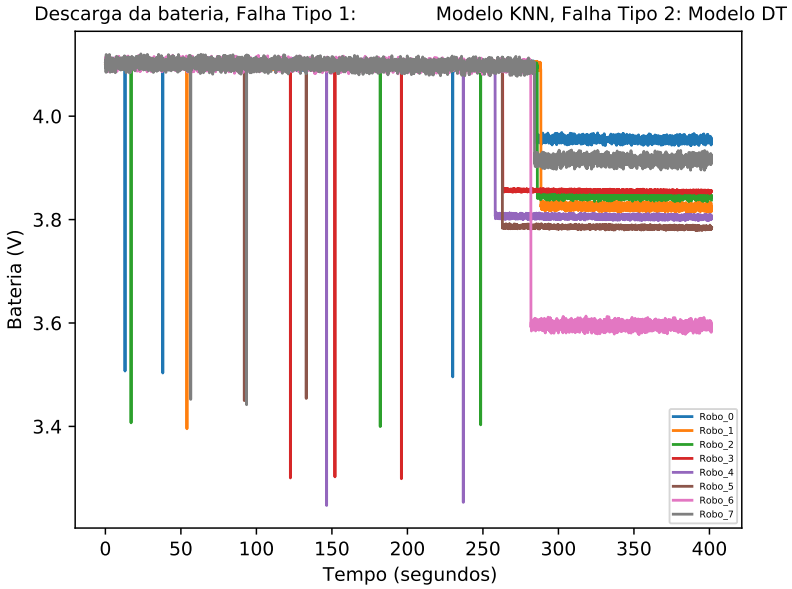


Figura A.12: Descarga da Bateria Teste Protótipo Estrutura Central, Modelo KNN para FT1 e Modelo DT para FT2 - Robô Parado.

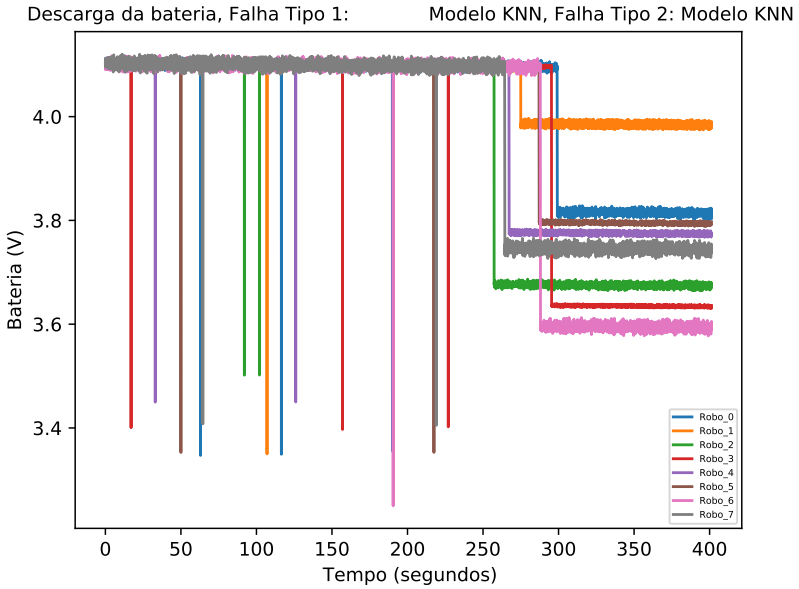


Figura A.13: Descarga da Bateria Teste Protótipo Estrutura Central, Modelo KNN para FT1 e FT2 - Robô Parado.

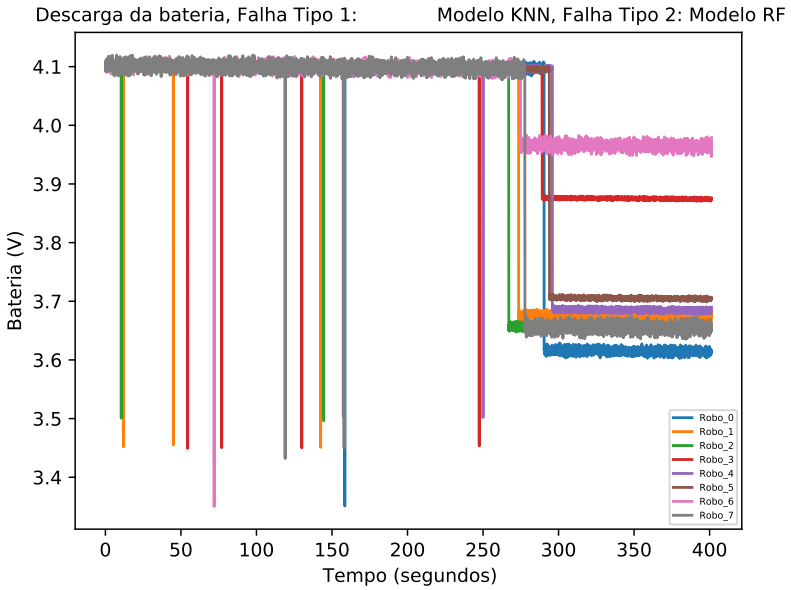


Figura A.14: Descarga da Bateria Teste Protótipo Estrutura Central, Modelo KNN para FT1 e Modelo RF para FT2 - Robô Parado.

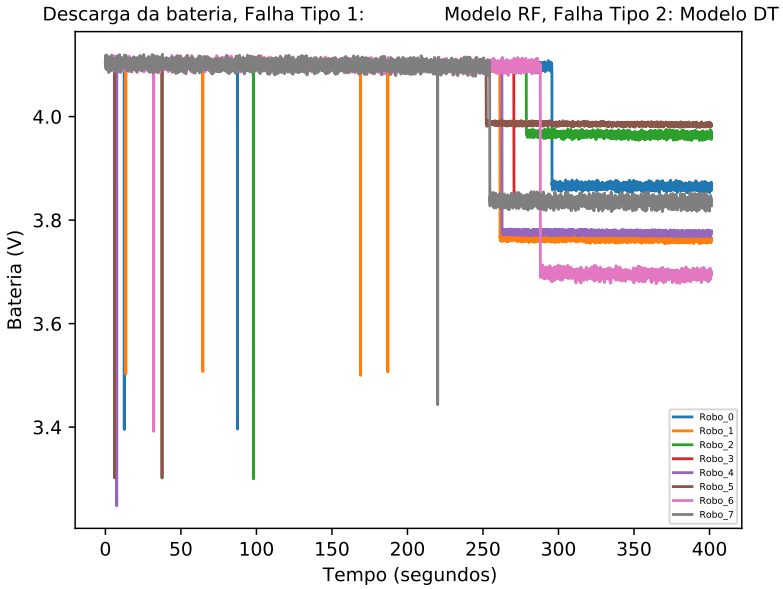


Figura A.15: Descarga da Bateria Teste Protótipo Estrutura Central, Modelo RF para FT1 e Modelo DT para FT2 - Robô Parado.

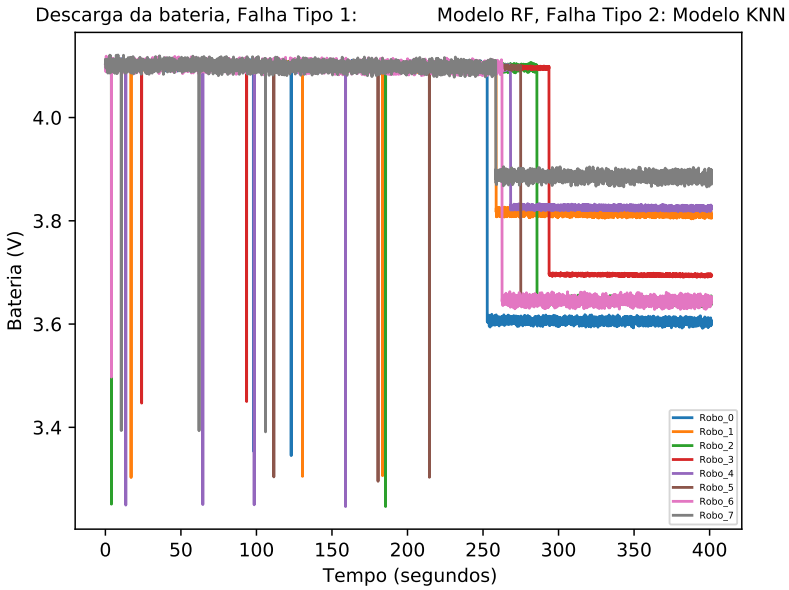


Figura A.16: Descarga da Bateria Teste Protótipo Estrutura Central, Modelo RF para FT1 e Modelo KNN para FT2 - Robô Parado.

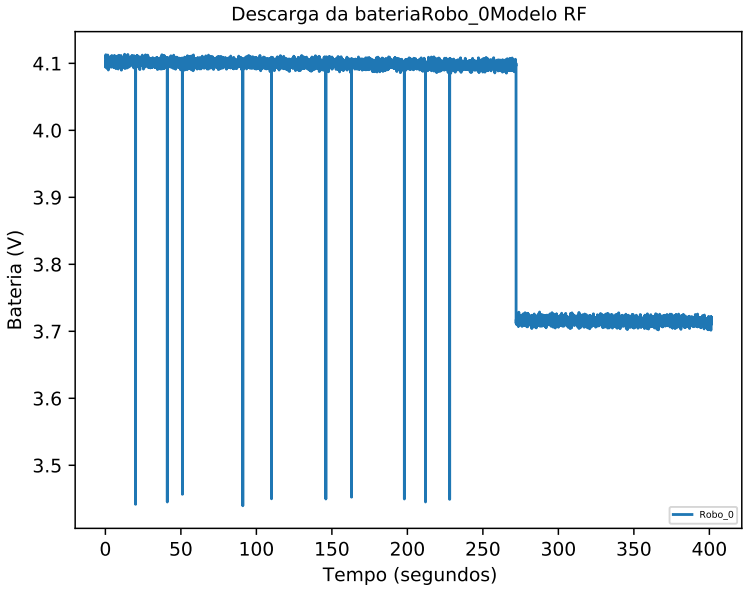


Figura A.17: Descarga da Bateria do Robô 0, Teste Protótipo Estrutura Local - Robô Parado

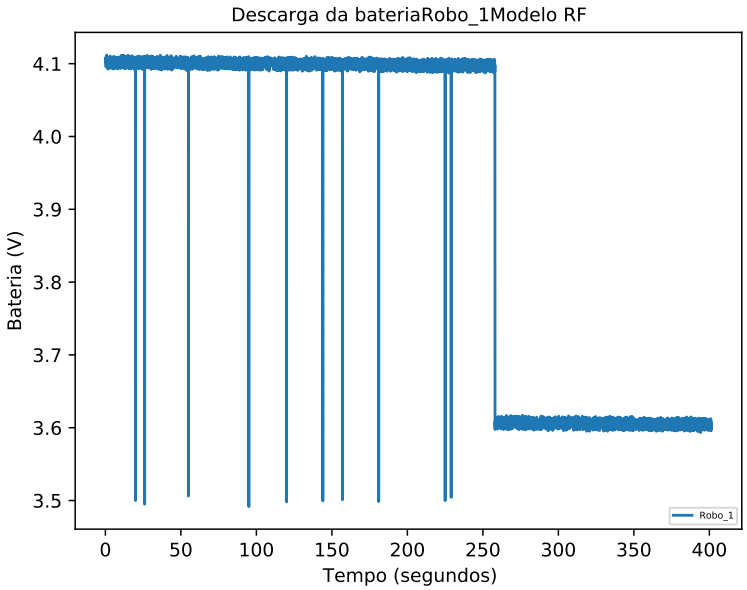


Figura A.18: Descarga da Bateria do Robô 1, Teste Protótipo Estrutura Local - Robô Parado

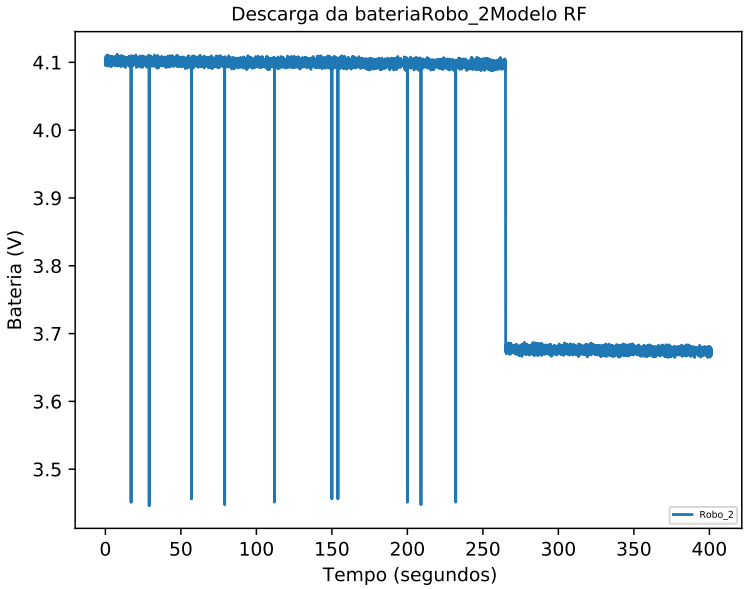


Figura A.19: Descarga da Bateria do Robô 2, Teste Protótipo Estrutura Local - Robô Parado

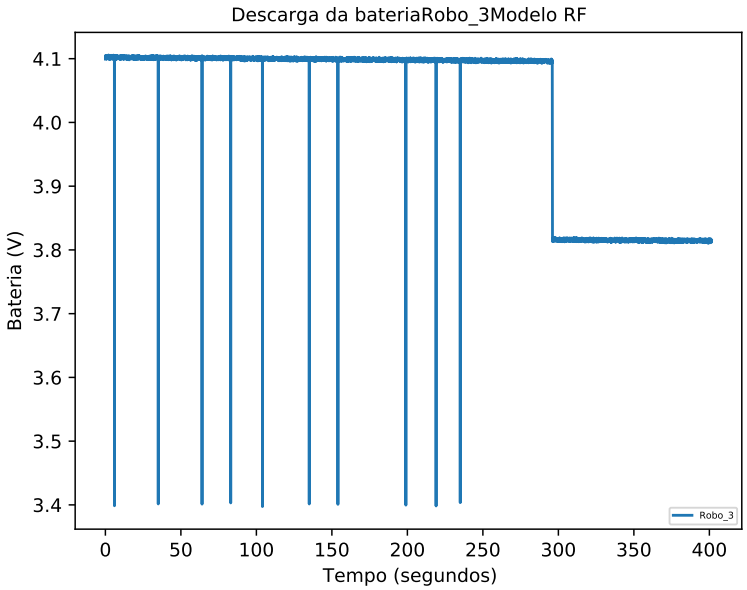


Figura A.20: Descarga da Bateria do Robô 3, Teste Protótipo Estrutura Local - Robô Parado

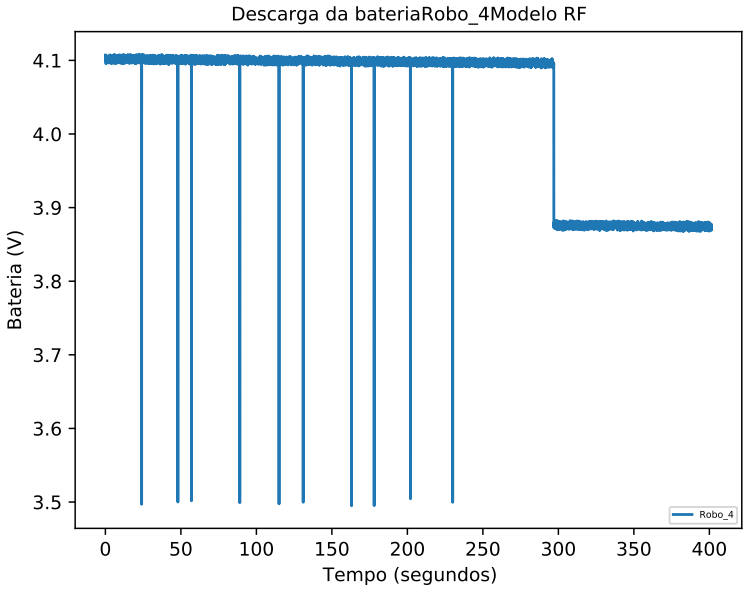


Figura A.21: Descarga da Bateria do Robô 4, Teste Protótipo Estrutura Local - Robô Parado

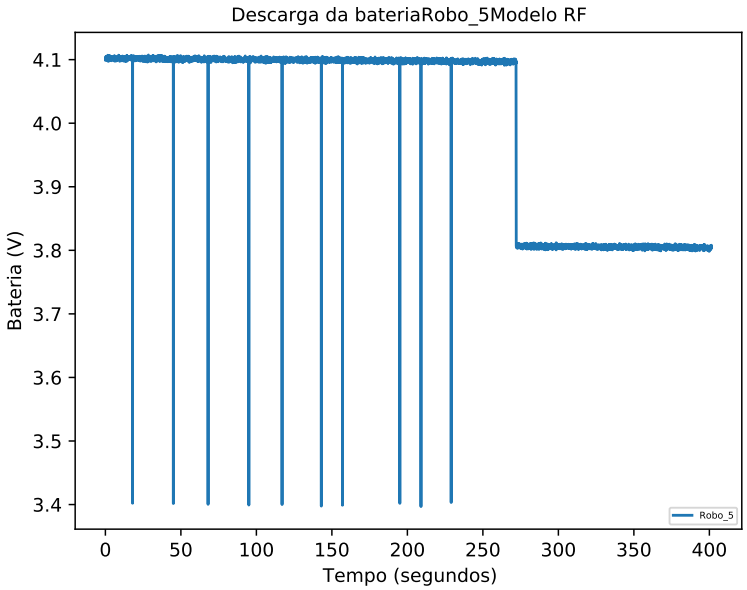


Figura A.22: Descarga da Bateria do Robô 5, Teste Protótipo Estrutura Local - Robô Parado

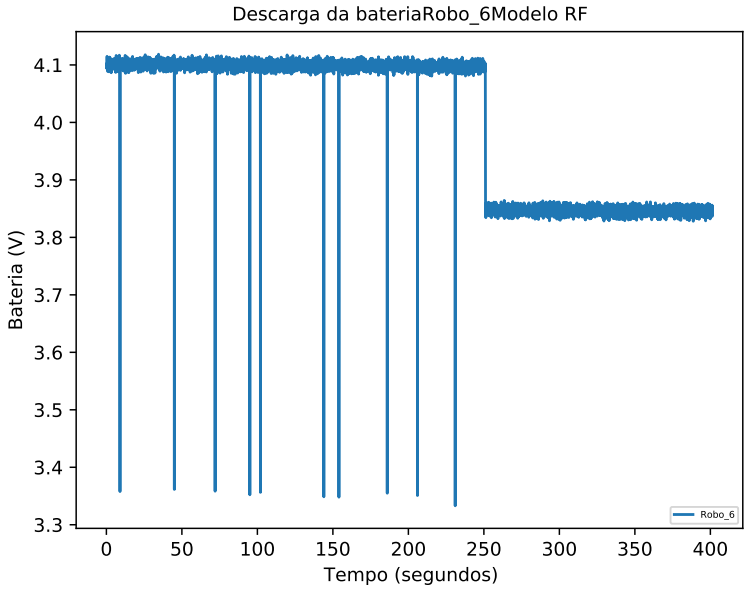


Figura A.23: Descarga da Bateria do Robô 6, Teste Protótipo Estrutura Local - Robô Parado

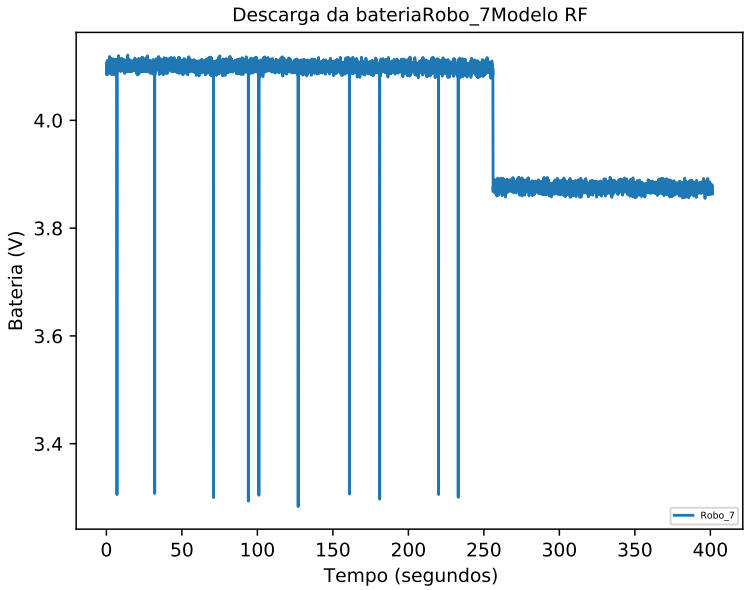


Figura A.24: Descarga da Bateria do Robô 7, Teste Protótipo Estrutura Local - Robô Parado

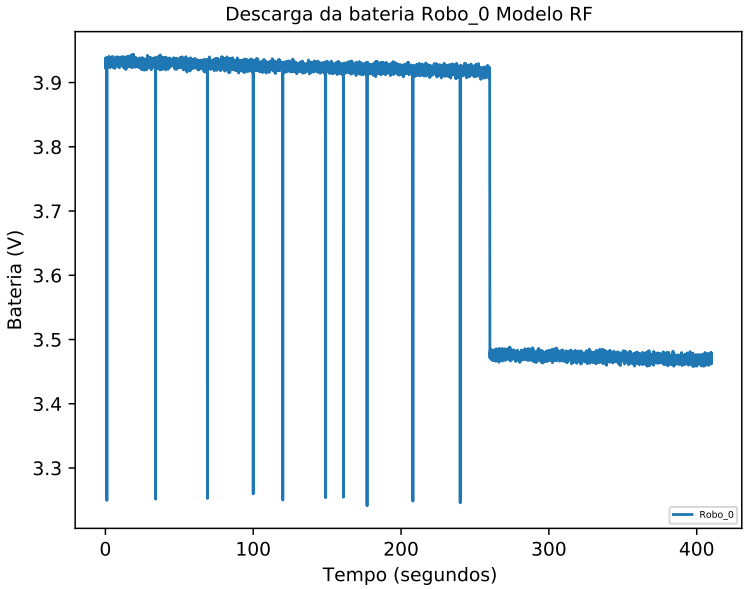


Figura A.25: Descarga da Bateria do Robô 0, Teste Protótipo Estrutura Local - Robô Movimento

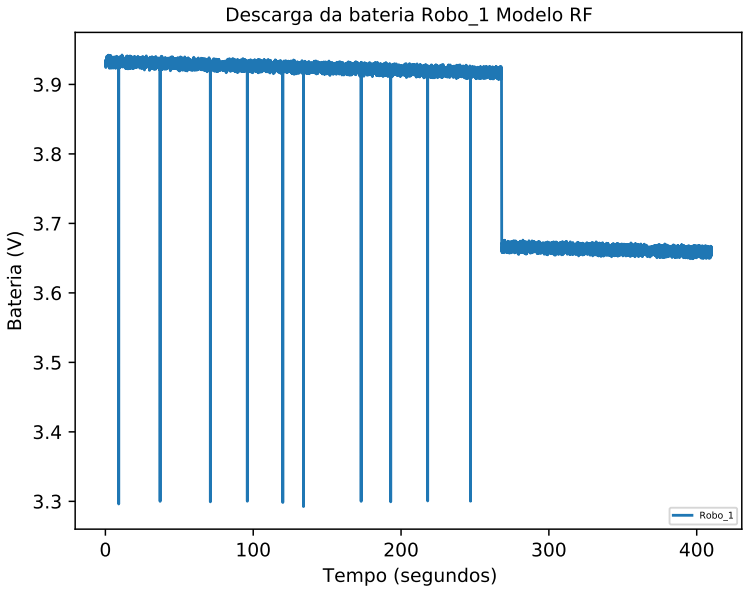


Figura A.26: Descarga da Bateria do Robô 1, Teste Protótipo Estrutura Local - Robô Movimento

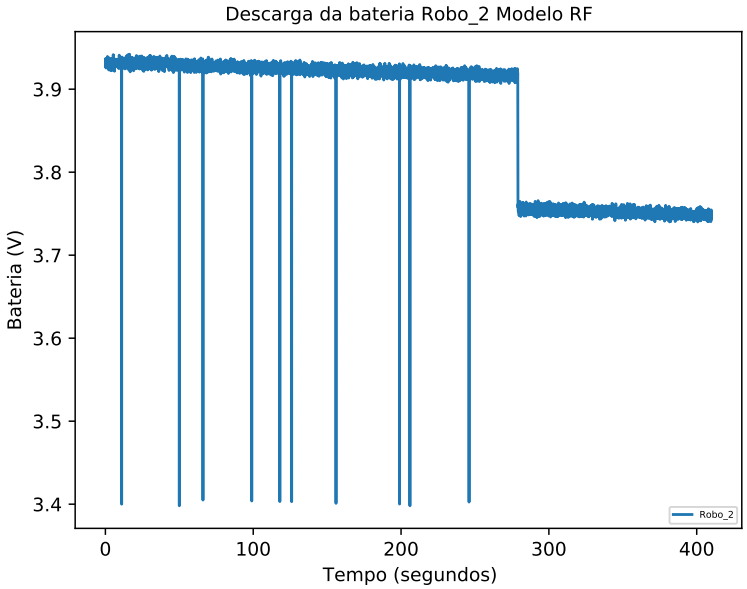


Figura A.27: Descarga da Bateria do Robô 2, Teste Protótipo Estrutura Local - Robô Movimento

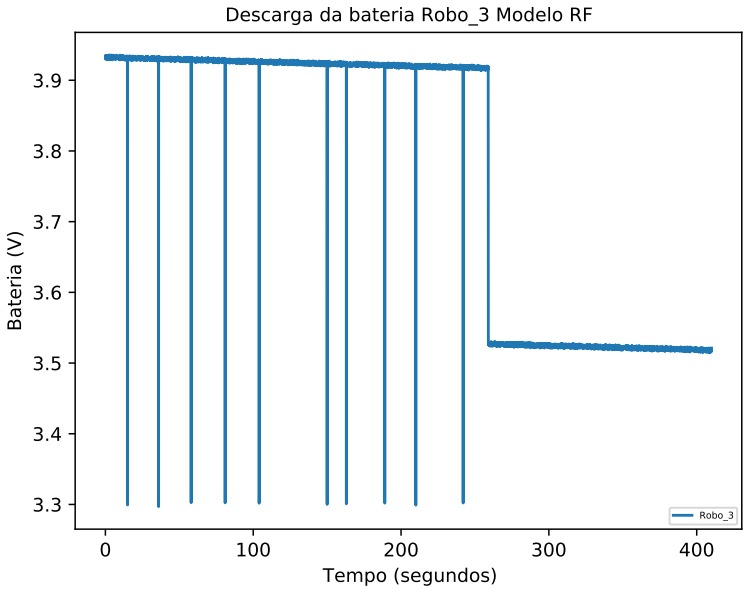


Figura A.28: Descarga da Bateria do Robô 3, Teste Protótipo Estrutura Local - Robô Movimento

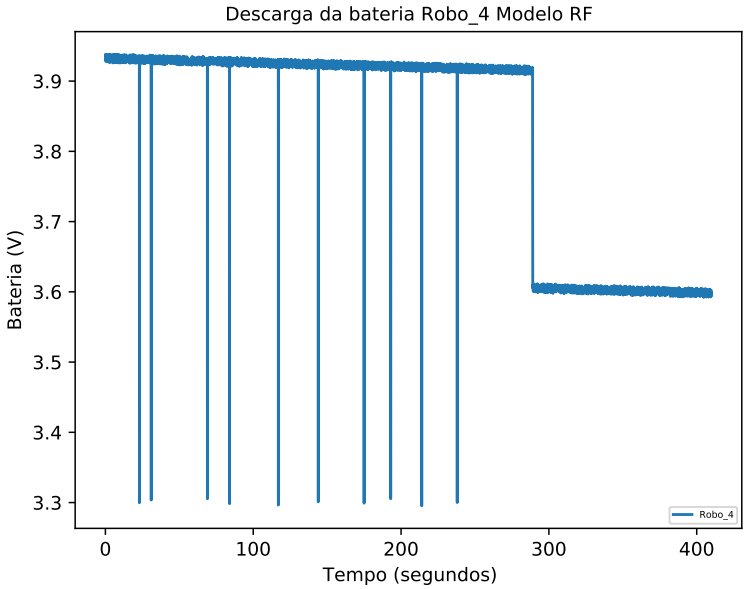


Figura A.29: Descarga da Bateria do Robô 4, Teste Protótipo Estrutura Local - Robô Movimento

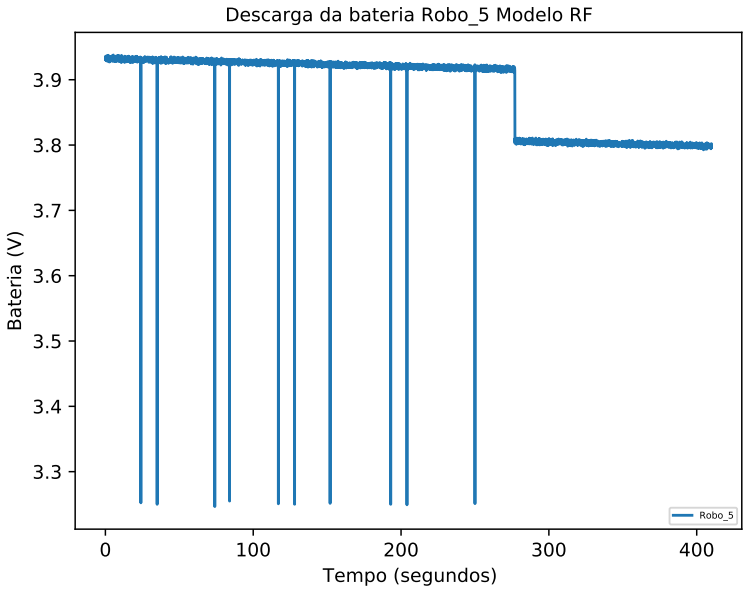


Figura A.30: Descarga da Bateria do Robô 5, Teste Protótipo Estrutura Local - Robô Movimento

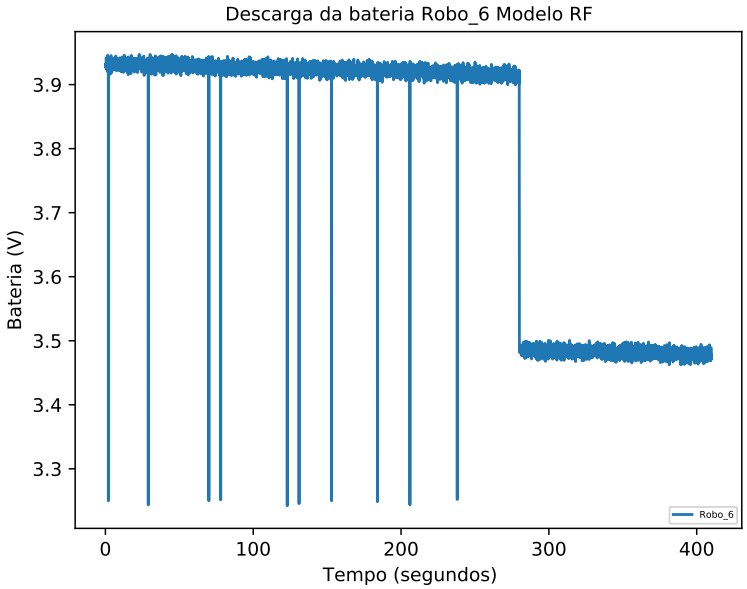


Figura A.31: Descarga da Bateria do Robô 6, Teste Protótipo Estrutura Local - Robô Movimento

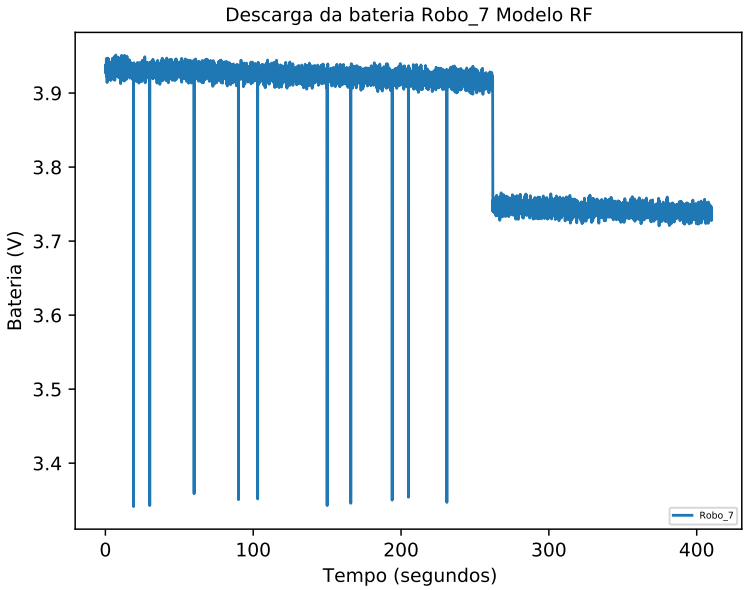


Figura A.32: Descarga da Bateria do Robô 7, Teste Protótipo Estrutura Local - Robô Movimento