UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO MULTIDISCIPLINAR

MARIO NOGUEIRA DE SANT'ANNA DINIZ

Viabilidade do uso de simulação para o treinamento de sistemas de localização indoor por fingerprint

Prof. Marcel William Rocha da Silva, D.Sc.
Orientador

Nova Iguaçu, Dezembro de 2024

Viabilidade do uso de simulação para o treinamento de sistemas de localização indoor por fingerprint

Mario Nogueira de Sant'Anna Diniz

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

Mario Nogueira de Sant'Anna Diniz

Aprovado por:

Prof. Marcel William Rocha da Silva, D.Sc.

Prof. Ubiratam Carvalho de Paula Junior, D.Sc.

Prof. Leandro guimarães marques alvim, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Dezembro de 2024

DOCUMENTOS COMPROBATÓRIOS Nº 28573/2024 - CoordCGCC (12.28.01.00.00.98)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 23/12/2024 11:03)
LEANDRO GUIMARAES MARQUES ALVIM
PROFESSOR DO MAGISTERIO SUPERIOR
DeptCC/IM (12.28.01.00.00.83)
Matrícula: ###008#2

(Assinado digitalmente em 21/12/2024 17:24)
MARCEL WILLIAM ROCHA DA SILVA
PROFESSOR DO MAGISTERIO SUPERIOR
PPGIHD (11.39.00.16)
Matrícula: ###807#6

(Assinado digitalmente em 23/12/2024 08:46)
UBIRATAM CARVALHO DE PAULA JUNIOR
PROFESSOR DO MAGISTERIO SUPERIOR
DeptCC/IM (12.28.01.00.00.83)
Matrícula: ###426#4

(Assinado digitalmente em 21/12/2024 12:35)
MARIO NOGUEIRA DE SANT'ANNA DINIZ
DISCENTE
Matrícula: 2017#####1

*Visualize o documento original em https://sipac.ufrrj.br/documentos/ informando seu número: **28573**, ano: **2024**, tipo: **DOCUMENTOS COMPROBATÓRIOS**, data de emissão: **21/12/2024** e o código de verificação: **13ea48f5a7***

Agradecimentos

Agradeço aos meus pais por todo apoio dado não somente na graduação mas por toda a vida, oferecendo todo o suporte possível para que eu me formasse não só em quesitos acadêmicos mas também como pessoa.

Agradeço também a minha família por ter sempre apoiado e acreditado em mim desde sempre.

Agradeço a minha melhor amiga(e namorada) e a todos meus amigos que além de apoiar, sempre estiveram presentes em todos âmbitos da minha vida e muitas vezes ficaram ao meu lado em momentos muito difíceis.

Agradeço também ao meu orientador por toda a paciência, apoio e atenção durante todo o período escrevendo este trabalho.

RESUMO

Viabilidade do uso de simulação para o treinamento de sistemas de localização indoor por fingerprint

Mario Nogueira de Sant'Anna Diniz

Dezembro/2024

Orientador: Marcel William Rocha da Silva, D.Sc.

Ao longo dos últimos anos, houve um aumento em estudos na área de localização *indoor*, utilizando tecnologias como *Wi-Fi*, *Bluetooth* e UWB para obter a localização. Este trabalho foca em posicionamento passivo baseado em *fingerprint* usando RSSI, com fases offline e online. A fase offline coleta dados dos *Access Points* para criar uma base de dados de *fingerprint*, enquanto a fase online prediz a localização com base nesses dados. Contudo, mudanças frequentes em ambientes podem exigir refazer a fase offline, o que eleva custos. Como alternativa, está sendo analisado o uso de simuladores como o ns-3, para modelar ambientes internos, reduzindo a necessidade de coletas físicas de dados. A análise comparativa entre simulação e ambiente real mostra a viabilidade da simulação para resolver problemas de localização *indoor*.

ABSTRACT

Viabilidade do uso de simulação para o treinamento de sistemas de localização indoor por fingerprint

Mario Nogueira de Sant'Anna Diniz

Dezembro/2024

Advisor: Marcel William Rocha da Silva, D.Sc.

Over the past years, there has been an increase in studies on indoor location, utilizing technologies such as Wi-Fi, Bluetooth, and UWB to obtain location data. This work focuses on passive positioning based on fingerprint using RSSI, with offline and online phases. The offline phase collects data from Access Points to build a fingerprint database, while the online phase predicts the location based on these data. However, frequent changes in environments may require redoing the offline phase, increasing costs. As an alternative, the use of simulators such as ns-3 is being analyzed to model indoor environments, reducing the need for physical data collection. The comparative analysis between simulation and the real environment shows the feasibility of simulation to address indoor location issues.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas e Siglas

ILBSs	Indoor location-based services
FP	*fingerprint*
RSSI	*Received Signal Strength Indication*
APs	*Access Points*
AP	*Access Point*
PR	Ponto de Referência
IWL	*Internal Wall Loss*
KNN	*K Nearest Neighbours*

Sumário

Capítulo 1

Introdução

Desde os primórdios da humanidade a localização é de suma importância para a sobrevivência da espécie e com o passar do tempo torna-se cada vez mais indispensável perante à sociedade como um todo. À medida em que a evolução do ponto de vista social ganha espaço entre as pessoas durante esses milhares de anos. A partir desta observação, a relevância da orientação pode ser notada.

À princípio, os caçadores-coletores já sofriam impactos que apontam a significância da localização pois necessitavam dispor de conhecimento prévio a respeito de locais apropriados para realizar suas colheitas de acordo com ciclo do ano, uma vez que ao deparar-se com a falta de alimentos, seria preciso alterar a rota com a finalidade de encontrar territórios em condições mínimas de abastecimento alimentar (CUMMINGS PETER JORDAN, 2014).

Com a mudança de organização para sedentários e em consequência o surgimento da agropecuária era necessário saber onde estavam rios para se aproveitar da água e se instalar próximo a esses locais. Ainda assim podemos citar posteriormente as viagens entre reinos, as grandes navegações com expedições marítimas entre continentes, revoluções industriais e até mesmo em guerras.

E é nesse contexto de guerra, mais especificamente na guerra fria, que surgiu o *Global Positioning System* (GPS), uma peça chave para a história da localização. Esse utiliza uma tecnologia via satélite que permite determinar a sua posição sobre a Terra

em latitude, longitude e altitude. Os receptores GPS medem os sinais provenientes de 3 ou mais satélites de maneira simultânea e determinam a sua posição através da trilateração destes sinais (MARINO, 2012).

Atualmente, o *Global Positioning System* (GPS) está presente em praticamente todos os setores da nossa sociedade e é utilizado em diversas áreas, como: agronomia, mapeamento de terrenos, estudos climáticos, monitoramento de placas tectônicas, padrões de migração de animais, logística, navegação e etc. Entretanto, apesar de ser amplamente difundido, apresenta um ponto limitante em seu desempenho que é inviabilizar sua funcionalidade em determinadas situações como em ambientes fechados, por exemplo, dentro de aeroportos, shoppings, edifícios e outros locais de comum acesso no dia-a-dia (CAMPOS, 2024).

Com isso, ao longo dos últimos anos, há uma crescente de estudos na área de localização *indoor* com o objetivo de resolver esse problema de diversas formas. Normalmente utilizando tecnologias do nosso dia a dia, tais como *Bluetooth*, *Ultra-Wideband (UWB)*, infravermelho e, principalmente, as redes *Wi-Fi* para obter a localização.

Este trabalho limita-se a localização *indoor* do tipo de posicionamento passivo baseado em *fingerprint* (FP) de *Received Signal Strength Indication* (RSSI), ou seja, é um processo que consiste em uma fase *offline* e uma fase *online*, em que a fase *offline* utiliza os níveis de sinais RSSI vindo dos APs que são dispositivos que emitem sinal wi-fi e esses dados são coletados em pontos chaves, chamados de Ponto de Referência (PR). Desta forma, é possível coletar FPs que representem uma localização única. E deste modo construir uma base de dados constituído dos sinais de RSSI vindo dos APs e assim podemos treinar um modelo de predição onde a partir de um padrão de sinais RSSI é possível obter a localização. Já na fase *online* é onde um dispositivo envia o sinal recebido dos APs e de acordo com modelo de predição pré definido na fase *offline* é informado a localização do dispositivo que realizou o envio do sinal.

Podemos aplicar esse tipo de técnica para resolver diversos tipo de problemas como, por exemplo, chegar de uma loja a outra dentro de um shopping, a partir de uma *tag* saber onde uma peça de roupa está dentro da loja, saber onde está

determinado objeto dentro de casa, conseguir localizar um carro no estacionamento e etc.

Entretanto podem vir a surgir alguns problemas práticos de usabilidade e implementação, pois com qualquer mudança física no local, mudança de organização ou de equipamentos, tem como consequência o custo de refazer a fase offline do processo. Isso pode aparecer como uma característica ruim em lugares onde essa aplicação poderia ser impactada com mudanças frequentes.

Referindo-se ao exemplo de ir de uma loja a outra em um shopping, poderia acontecer de em uma semana adicionarem um brinquedo grande no local, logo após um palco para apresentações, em seguida ficar vazio novamente e assim por diante, e em cada um desses acontecimentos a fase offline seria feita novamente ou seja, seria necessário coletar dados no local para formar uma base de dados, treinar um modelo preditivo e testá-lo para validar.

Como uma possível alternativa a fim de resolver essa questão, podemos citar o trabalho de conclusão de curso Simulação de Redes Sem Fio para o Problema da Localização Indoor (FERREIRA RENATO. QUEIROZ, 2021). Este trabalho consiste em mostrar a opção de que com um simulador (NSNAM, 2024) é possível com o módulo Building simular blocos/prédios com sinais wi-fi e diversos modelos de propagação para elaborar uma situação que represente o cenário onde seria implementado um serviço de *localização indoor*. Desta forma, torna-se possível poupar o custo físico da fase offline do projeto.

Considerando a gama de possibilidades que o tema abordado pode apresentar, foi possível contribuir com esta linha de pesquisa analisando a viabilidade da simulação como uma possível alternativa para resolução dos problemas citados até então, ou seja, poupar esforços físicos, financeiros e de tempo, com este objetivo, surge este trabalho que visa realizar uma análise comparativa entre utilizar o simulador e o ambiente real para coletar dados para a fase *offline* do algoritmo.

O contexto dessa análise ocorre em duas salas do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro (UFRRJ), nas quais foram instalados

3 APs e coletados diversos *fingerprints* vindos destes APs em 6 PRs posicionados em cada sala, totalizando 12 possíveis localizações nesse cenário. Desta forma de acordo com a planta baixa do prédio, foi possível criar um ambiente no simulador exatamente com a mesma medida e com a mesma quantidade de APs.

Assim, possibilitando a construção de uma base de dados com dados simulados e outra base de dados com dados coletados em ambiente real, para que seja analisado a possibilidade de realizar a fase *offiline* do algoritmo com os dados simulados a fim de poupar esforços físicos, financeiros e de tempo que são necessários para coletar os dados em ambiente real.

Capítulo 2

Fundamentação teórica

Neste capítulo, serão apresentados conceitos-chave sobre localização *indoor* e algoritmos de classificação, essenciais para o entendimento da análise comparativa entre simulações com ns-3 e dados coletados em ambiente real.

2.1 Trabalhos relacionados

Este trabalho está relacionado aos trabalhos (SILVA MARCEL WILLIAM ROCHA; ZAMITH, 2022) e (NIU, 2020), os quais forneceram informações detalhadas e abrangentes sobre a *localização indoor*, além de apresentarem a base conceitual e técnica essencial para o desenvolvimento deste estudo. Oferecendo uma compreensão mais detalhada dos desafios e das soluções existentes nesse campo. Dessa forma, este trabalho dá continuidade a essas discussões, ampliando o conhecimento e explorando novas abordagens e aplicações para o uso de tecnologias de localização em ambientes internos.

2.2 Localização Indoor

Durante a Guerra Fria, uma nova tecnologia foi desenvolvida pelo Departamento de Defesa dos Estados Unidos: o GPS. Possuindo como objetivo criar um sistema

de navegação altamente preciso para uso militar. Atualmente, essa tecnologia está amplamente difundida na sociedade e é utilizada desde o mapeamento de áreas por satélite para auxiliar o agronegócio em atividades como plantio e colheita, até a mobilidade urbana, traçando trajetos de um local ao outro.

Apesar de ser extremamente útil e possuir diversas aplicações, o GPS apresenta algumas limitações, entre elas, a dificuldade de operar em locais fechados. É nesse contexto que surge nossa área de estudo, a localização *indoor*.

A localização *indoor* é um processo utilizado principalmente quando há ausência ou ineficiência do Global Positioning System (GPS) em ambientes internos. Essa prática se beneficia de tecnologias comuns em nosso cotidiano, tais como Bluetooth, Ultra-Wideband (UWB), infravermelho e, principalmente, as redes *Wi-Fi* com o objetivo de substituir o GPS e agir como um sistema que consegue nos fornecer a localização de um dispositivo.

As redes *Wi-Fi* são capazes de emitir sinais por meio de dispositivos âncoras, conhecidos como Access Points (APs). Desta forma conseguimos utilizar dispositivos receptores, como smartphones, notebooks, tags, entre outros, para captar esse sinal e assim construir uma base de dados onde cada elemento será denominado como FP.

Nesse contexto, determinar a localização atual de um receptor passa a se mostrar como um desafio que envolve a aplicação de algoritmos de classificação, tendo em vista que teremos padrões e assim poderemos separar em diversas classes que serão justamente os possíveis locais. Desta forma é possível inferir a posição atual do dispositivo receptor dentro do ambiente analisado.

Para elucidar o texto acima podemos citar alguns exemplos de aplicação da localização *indoor* para melhor compreensão do tema abordado:

(a) Traçar rotas de uma loja até outra dentro de um shopping center.

(b) Encontrar uma peça de roupa dentro de uma loja.

(c) Identificar em que lugar de um cômodo um objeto está localizado.

O processo de localização *indoor* baseada em redes *Wi-Fi* pode ser classificada em dois tipos de posicionamentos: ativo e passivo (vide Figura 2.1). Para um melhor entendimento, serão abordadas as definições do posicionamento ativo e do posicionamento passivo. No entanto, para este trabalho, consideraremos apenas o posicionamento passivo baseado com FP de acordo com o valor de RSSI.

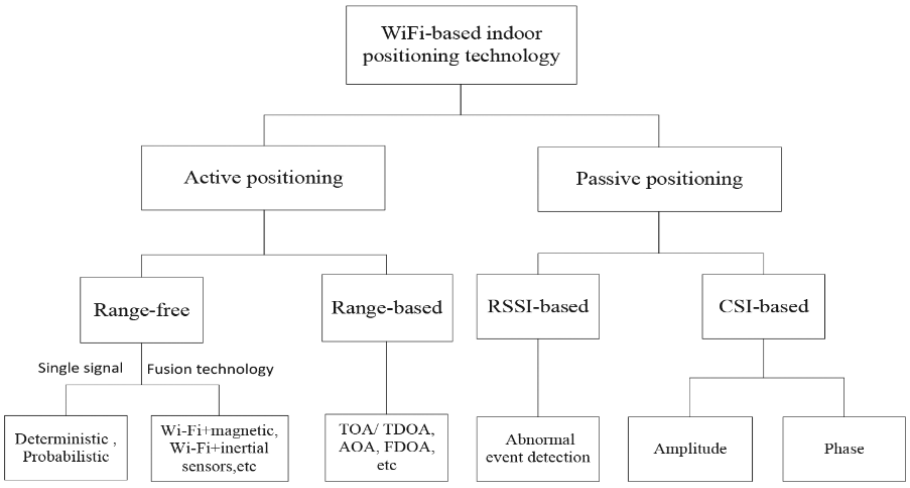


Figura 2.1: Classificação da localização *indoor* baseada no *Wi-Fi*

Fonte: (NIU, 2020)

2.2.1 Posicionamento ativo

Neste tipo de posicionamento o dispositivo deve conter o equipamento necessário para enviar os sinais, como por exemplo, um smartphone com *Wi-Fi*, que transmite ao receptor as informações necessárias em cada método. Seja sinal de rádio, bluetooth, RSS e etc. Pode ser usado também de forma combinada a fim de atingir uma acurácia mais alta.

2.2.2 Posicionamento passivo

A fim de compensar a falta do posicionamento ativo *indoor*, o posicionamento passivo pode ajudar as pessoas a alcançar vários Indoor location-based services (ILBSs) sem depender de equipamentos específicos (NIU, 2020), mas é necessário

que já tenha o classificador ou algum algoritmo que irá retornar a localização que possa ser executado no equipamento. É possível determinar a localização de acordo com o nível de RSSI que varia de acordo com o posicionamento do usuário.

2.2.2.1 RSSI

O RSSI é um indicador de intensidade de sinal de rádio. Através deste indicador, é possível estabelecer, em dBm, uma medida de intensidade do sinal recebido por um determinado receptor. Esta medida de energia é bastante difundida, por sua simplicidade de medição e por não necessitar de hardware adicional para a medida da intensidade do sinal (UOMALA J.; HAKALA, 2012).

$$RSSI = P_0 + 10nLog(\frac{d}{d_0}) + X \tag{2.1}$$

(a) P_0 é a potência recebida na medida de referência do modelo (em dBm).

(b) 10n é o resultado do cálculo do path loss da variável aleatória da influência do ambiente no sinal, visto que a influência do ambiente no sinal é uma variável cujo cálculo é influenciado por muitas outras variáveis, tornando seu cálculo, muitas vezes, impreciso e complexo.

(c) d é a distância que o sinal percorreu (em cm)

(d) d_0 é a distância de referência do modelo

(e) X é a variável que representa a influência do ambiente no sinal.

Com a Equação 2.1, é possível obter o valor de RSSI em dBm para uma determinada transmissão (NI, 2012).

2.2.3 Posicionamento passivo baseado em FingerPrint e RSSI

No posicionamento passivo baseado em FP de acordo com o valor do RSSI a implementação é divida em duas fases: Offline e Online.

Na fase Offline é definido um mapa onde os sinais RSSI originam-se de um ou mais APs correspondem a um lugar específico na área definida, desta forma, é possível obter um conjunto de FP que funcionarão como um dataset, podendo seguir o exemplo da tabela 2.1.

Tabela 2.1: Exemplo do dataset de *Fingerprints*

Coordenadas	RSSI(AP_1)	RSSI(AP_2)	RSSI(AP_n)
(X_1,Y_1)	-30dBm	-20dBm	-10dBm
(X_2,Y_2)	-10dBm	-30dBm	-30dBm
(X_3,Y_3)	-30dBm	-50dBm	-35dBm
(X_4,Y_4)	-55dBm	-20dBm	-10dBm
(X_5,Y_5)	-40dBm	-15dBm	-20dBm

Na fase Online é onde o dispositivo envia o sinal RSSI atual que recebe de cada *Access Point* (AP), desta forma podemos aplicar algoritmo(s) para comparar os sinais recebidos com o mapa definido previamente e assim saber a localização do dispositivo.

2.3 Fingerprint

No contexto deste trabalho o *Fingerprint* é uma representação única de uma localização baseada no RSSI recebido de um ou mais APs. Podemos definir também da seguinte forma:

$$FP_m = \{RSSI(AP_1), RSSI(AP_2), ..., RSSI(AP_n)\} \tag{2.2}$$

A definição ainda pode ser ilustrada com a figura 2.2

2.4 ns-3

O ns-3 é um simulador de rede de eventos discretos, voltado principalmente para pesquisa e uso educacional. ns-3 é um software livre, licenciado sob a licença

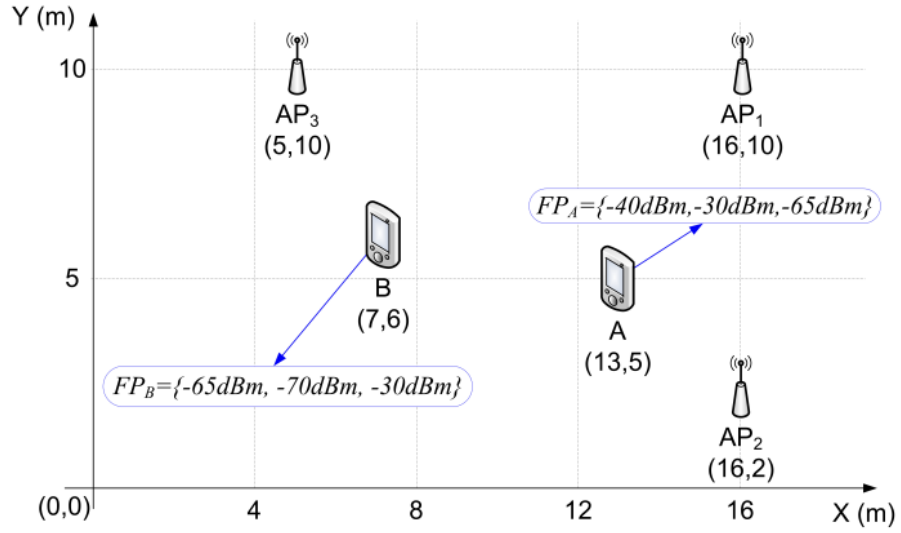


Figura 2.2: Exemplo de *Fingerprint*

Fonte: (SILVA MARCEL WILLIAM ROCHA; ZAMITH, 2022)

GNU GPLv2, e está disponível publicamente para pesquisa, desenvolvimento e uso (NSNAM, 2024).

O objetivo do projeto ns-3 é desenvolver um ambiente de simulação preferencial e aberto para pesquisa em redes: ele deve estar alinhado com as necessidades de simulação da pesquisa moderna em redes e deve encorajar a contribuição da comunidade, a revisão por pares e a validação do software (NSNAM, 2024).

2.4.1 Modelos de Simulação

O núcleo de simulação ns-3 suporta pesquisa em redes baseadas em IP e não-IP. No entanto, a grande maioria de seus usuários se concentra em simulações sem fio/IP que envolvem modelos para Wi-Fi, WiMAX ou LTE para camadas 1 e 2 e uma variedade de protocolos de roteamento estáticos ou dinâmicos, como OLSR e AODV para aplicativos baseados em IP (NSNAM, 2024).

O ns-3 também oferece suporte a um agendador em tempo real que facilita uma série de casos de uso de "simulação em loop" para interação com sistemas reais. Por exemplo, os usuários podem emitir e receber pacotes gerados pelo ns-3 em dispositivos de rede reais, e o ns-3 pode servir como uma estrutura de interconexão para adicionar

efeitos de link entre máquinas virtuais (NSNAM, 2024).

Outra ênfase do simulador está na reutilização de aplicativos reais e código do kernel. Frameworks para executar aplicações não modificadas ou toda a pilha de rede do kernel Linux dentro do ns-3 estão atualmente sendo testados e avaliados (NSNAM, 2024).

2.4.2 Módulo Building e trabalhos relacionados

O módulo de construção do NS3 possibilita a criação de ambientes destinados à simulação. Desta forma, torna-se viável a construção de edifícios, com a opção de escolher o tipo de construção(residencial, escritório ou comercial), características das paredes externas(madeira, concreto com janela, concreto sem janela ou blocos de pedra), quantidade de andares e número de cômodos desejados.

Porém existem algumas limitações como o edifício ser sempre representado como um paralelepípedo retangular e todos os cômodos do prédio possuírem o mesmo tamanho. No entanto, há um trabalho em andamento dos alunos Rodolfo Cláudio e Marcus Gonçalves de Ciência da computação da UFRRJ para permitir a construção de formas mais complexas através de coordenadas, aproximando ainda mais a simulação de ambientes reais.

2.5 Algoritmo de classificação

Um algoritmo de classificação é um método computacional de aprendizagem supervisionada empregada no âmbito de ciência de dados, cuja função é identificar padrões nos dados com o objetivo de atribuir categorias ou classes a novas instâncias. Normalmente, este processo envolve várias etapas, tais como:

(a) Aquisição e limpeza de dados

(b) Escolha do algoritmo (Ex: Rede neural, Árvore de decisão e etc)

(c) Treinamento

(d) Validação

(e) Aplicação de métricas

(f) Predição

2.5.1 Aquisição e limpeza de dados

Logo após os dados serem adquiridos ou criados, os dados são preparados e isso pode incluir limpeza (remoção de dados incompletos, sem sentidos, nulos e etc) e transformações (normalização, hot encoding e etc).

2.5.2 Escolha do algoritmo

Neste passo é verificado o algoritmo de classificação mais adequado com base na natureza dos dados e do problema específico. As escolhas mais comuns podem incluir Redes Neurais Artificiais (RNAs), Naive Bayes, Regressão Logística, Árvores de Decisão e etc.

2.5.3 Treinamento

O modelo é treinado utilizando um conjunto de dados de treinamento rotulado. Durante o treinamento, o modelo ajusta seus parâmetros para otimizar a capacidade de generalização, ou seja, para fazer previsões precisas em novos dados não vistos. O treinamento é feito com a parte especifica do dataset destinado a ele.

2.5.4 Validação

Após o treinamento, o modelo é validado usando um conjunto de dados de validação ou através de técnicas como validação cruzada(separar um parte do dataset para testar e ir alternando essa fatia). Isso ajuda a avaliar o desempenho do modelo e ajustar os parâmetros quando necessário.

2.5.5 Aplicação de métricas

O desempenho do modelo é medido utilizando métricas apropriadas para problemas de classificação, como precisão, recall, F1-score e matriz de confusão. Isso determina quão bem o modelo pode generalizar para novos dados.

2.5.6 Predição

Uma vez validado e avaliado, o modelo treinado pode ser utilizado para fazer previsões em novos dados não rotulados, atribuindo-lhes uma classe com base nos padrões aprendidos durante o treinamento com os dados já existentes e com veracidade confirmada.

Capítulo 3

Proposta

Este capítulo aborda a metodologia utilizada em torno deste trabalho para realizar a confecção de duas bases de dados de FPs. Uma construída a partir de um ambiente real e a outra advinda de um ambiente virtual preparado no simulador ns-3. Para que assim seja realizada uma análise com o objetivo de verificar o potencial de obter a localização de um dispositivo a partir de um FP capturado em ambiente real mas com um classificador que foi treinado em um ambiente simulado.

3.1 Ambiente real

Para realização do experimento em um ambiente real com o objetivo de realizar a criação de uma base de dados, foram utilizadas duas salas de aula do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro (UFRRJ) que possuíam a mesma medida e eram posicionadas uma ao lado da outra, havendo uma parede que dividiam as duas.

A distribuição do número de PRs e APs foi definida levando em consideração a limitação do número de dispositivos Raspberrys disponíveis, de forma a garantir uma alocação eficiente e equilibrada. Em cada uma das salas foram posicionados 6 PRs equidistantes, totalizando 12 PRs, além disso havia um total de 3 APs, que foram distribuídos da seguinte forma: 2 APs na sala 1 e o outro AP que foi posicionado na

sala 2.

Tanto os APs quanto os PRs foram posicionados em cima das cadeiras, ficando a aproximadamente 75 centímetros de altura em relação ao solo. Pode-se observar visualmente o posicionamento dos elementos do experimento a partir da figura 3.1.

3.1.1 Coleta de dados em ambiente real

No ambiente experimental descrito, foram utilizados um total de 9 dispositivos Raspberrys, sendo que 6 desses foram designados como pontos de referência PR e 3 como pontos de acesso AP. Devido à limitação no número de Raspberrys disponíveis, não foi possível disponibilizar simultaneamente os 12 PRs necessários. Desta forma, os 6 Raspberrys destinados aos PRs foram posicionados na sala 1, onde permaneceram por 20 minutos coletando *fingerprints*. Após esse período, os dispositivos foram desligados e transportados para a sala 2, sendo reposicionados em locais equivalentes aos da sala 1 e monitorando pela mesma quantidade de tempo. Por outro lado, os APs permaneceram fixos e inalterados durante toda a duração do experimento.

Os Raspberrys que seriam designados como APs tiveram suas interfaces de rede alteradas para operar em modo de gerenciamento através do software hostapd, a fim de possibilitar serem vistos como APs. Por sua vez, os PRs foram alterados para funcionarem em modo monitor ou seja, capturar pacotes vindos de APs, porém foram todos programados para descartar quaisquer pacotes de rede cujos endereços MAC não correspondessem a um dos endereços atribuídos aos APs.

Para a construção da base de dados, cada Raspberry executou um programa em C++ utilizando a biblioteca libtins que era iniciado assim que o dispositivo era ligado, no caso dos Raspberrys que são APs, o software hostpad fazia com que eles enviassem pacotes de beacon como um AP convencional. Já nos Raspberrys que funcionaram como PRs, o script capturava esses pacotes e assim eram gerados logs no formato CSV, contendo três campos separados por vírgula. O primeiro campo correspondia ao timestamp da captura, o segundo ao endereço MAC do dispositivo, e o terceiro ao nível de sinal RSSI registrado.

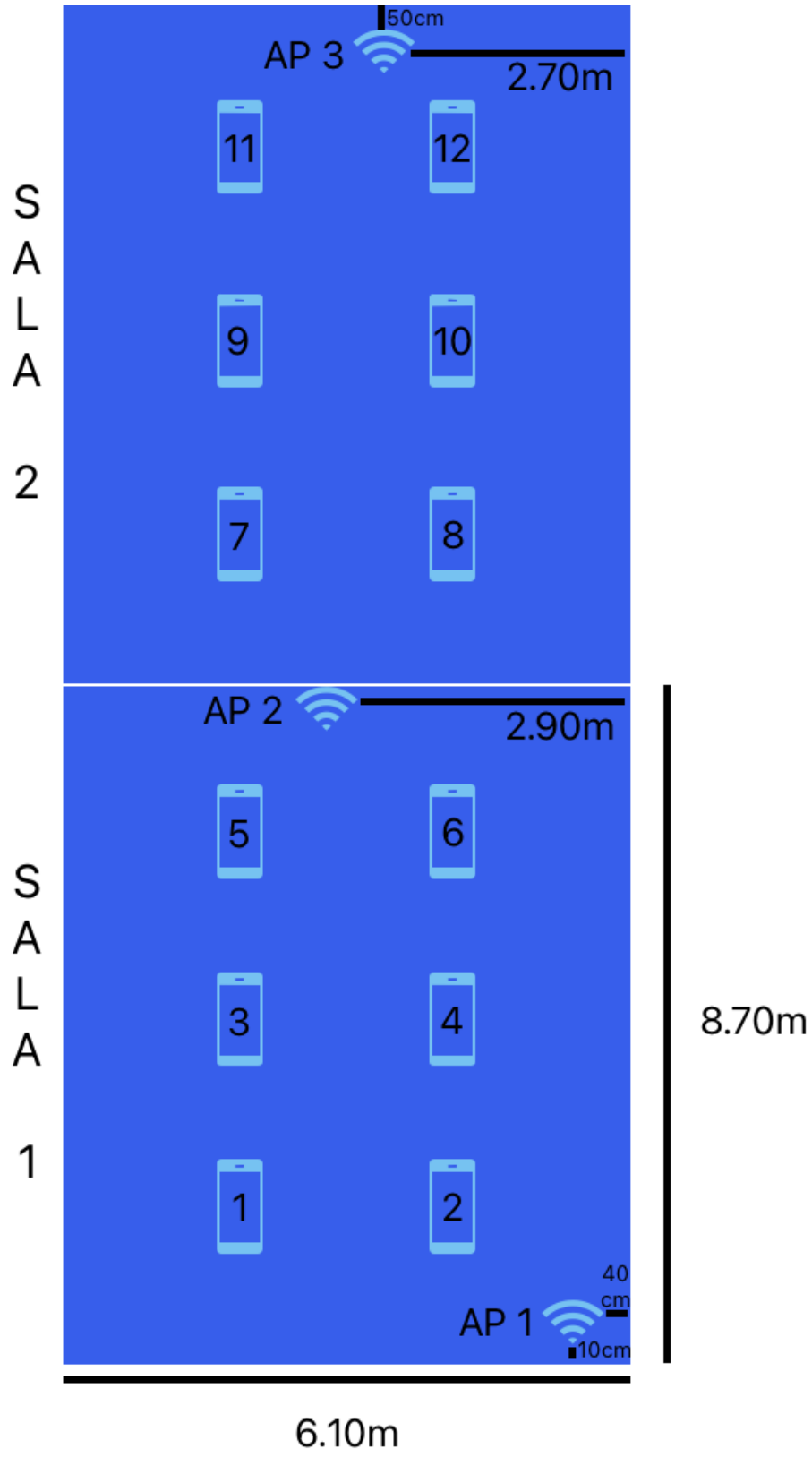


Figura 3.1: Numerações de 1 a 12 correspondem a enumeração dos pontos de referência

No que se refere a captura de pacotes e geração de logs, consegue-se através do código 3.1 verificar o modo em que os mesmos são realizados. O script desenvolvido em C++ com funcionalidades da biblioteca libtins recebe como parâmetros respectivamente, uma interface de rede, o nome do arquivo onde serão salvos os dados e uma lista de endereços MAC para que seja possível realizar uma filtragem com apenas os endereços desejados.

Após definir a interface e o filtro de MACs, é realizada a configuração do filtro de captura de pacotes com o objeto SnifferConfiguration que será passado para o objeto Sniffer, que é um analisador de pacotes, facilitando seu manejamento. Nessa configuração é inserido o filtro de endereços MAC que foi recebido via parâmetro e o modo promíscuo é ativado para que se torne possível capturar os pacotes na interface de rede passada no construtor do sniffer junto com a configuração. Por fim é realizado um sniff_loop, que recebe uma função de *callback* como parâmetro que é executada toda vez que chega um pacote. A função de *callback* é responsável por, a partir de um pacote, realizar a extração dos itens desejados para geração dos logs.

A partir de um Packet é possível obter o PDU que é o pacote entregue pelo software de captura libPCAP, Neste caso, como a interface de rede *Wi-FI* utilizada na captura está no modo monitor, a placa de rede adiciona um cabeçalho do tipo RadioTap, o qual contém informações de camada física do pacote capturado, como por exemplo, a potência de recepção do pacote (dbm_signal). Sendo assim, após pegar o timestamp do pacote, o endereço MAC do transmissor do pacote de beacon e potência de recepção do pacote RSSI, estas informações são gravadas em um arquivo e salvo em formato CSV.

Código 3.1: Gerando logs

```
bool callback(const Packet& packet) {
        const PDU* pdu = packet.pdu();
    const RadioTap& radiotap = pdu->rfind_pdu<RadioTap>();
    const Dot11Beacon& beacon = pdu->rfind_pdu<Dot11Beacon>();
        Timestamp ts = packet.timestamp();
        microseconds us = ts;
        cerr << us.count() << "," << beacon.addr2() << "," << (int)
            radiotap.dbm_signal() << "\n";
        out << us.count() << "," << beacon.addr2() << "," << (int)
            radiotap.dbm_signal() << "\n";
        out.flush();
    return true;
}
int main(int argc, char* argv[]) {
    if (argc < 3) {
        cout << "Usage: " <<* argv << " <interface> <outputfile> <
            space sep list of macs>" << endl;
        return 1;
    }
    string iface = argv[1];
    out.open (argv[2], std::ios::app);
        string mac_filter = "";
        for (int i=3; i < argc; i++) {
                        mac_filter = mac_filter + argv[i];
                        if (i != (argc-1))
                                        mac_filter = mac_filter + "
                                            or ";
        }
    SnifferConfiguration config;
    config.set_promisc_mode(true);
        if (!mac_filter.empty()) config.set_filter("subtype beacon
            and wlan addr2 " + mac_filter);
        else config.set_filter("subtype beacon");
        config.set_rfmon(true);
    Sniffer sniffer(iface, config);
        sniffer.sniff_loop(callback);
        out.close();
}
```

Como resultado do script de monitoramento dos pacotes, podemos verificar a partir da listagem 3.2, um exemplo de saída do mesmo. Com o primeiro item sendo o timestamp, o segundo o endereço MAC do transmissor e o terceiro item o nível de intensidade de sinal.

Código 3.2: Exemplo do arquivo de logs gerado

```
1669233426092984,b8:27:eb:93:12:b9,-17
1669233426297724,b8:27:eb:93:12:b9,-19
1669233426400121,b8:27:eb:93:12:b9,-17
1669233426502524,b8:27:eb:93:12:b9,-21
1669233426604940,b8:27:eb:93:12:b9,-19
1669233426707387,b8:27:eb:93:12:b9,-17
1669233426809725,b8:27:eb:93:12:b9,-21
1669233426912128,b8:27:eb:93:12:b9,-19
1669233427014525,b8:27:eb:93:12:b9,-19
1669233427116938,b8:27:eb:93:12:b9,-21
1669233427219320,b8:27:eb:93:12:b9,-17
1669233427321719,b8:27:eb:93:12:b9,-19
1669233427424119,b8:27:eb:93:12:b9,-19
1669233427526523,b8:27:eb:93:12:b9,-19
1669233427631262,b8:27:eb:93:12:b9,-17
```

3.1.1.1 Construção da base de dados

No total, considerando que foram utilizados 12 PRs, obteve-se um total de 12 arquivos de saída, conforme exemplificado no código 3.2. Foi necessário converter os dados dos RSSIs individuais dos pacotes em fingerprints e após isso a concatenação desses arquivos para formar uma base de dados completa, contendo as informações de todos os PRs.

Adicionalmente, a base de dados resultante deve apresentar algumas modificações em relação ao exemplo original. Todos os endereços MAC, que representam os APs, passaram a ser atributos dessa base, com o valor do RSSI correspondente a cada timestamp associado.

Dado que os pacotes eram enviados com intervalos de 100 milissegundos, os dados foram concatenados até completar 1 segundo. Caso algum valor estivesse ausente dentro desse intervalo, ele foi preenchido com a média dos valores adjacentes. Nos casos em que não havia nenhum valor registrado para aquele intervalo, o campo foi preenchido com o valor 0. Além disso, foi necessário incluir um atributo adicional denominado "target", ou seja, o alvo, que indica o PR que registrou esses dados, ou seja, o PR em que esses dados devem ser classificados.

A fim de esclarecer mais ainda esta transformação, vide exemplo da tabela 3.1 que é um exemplo de uma base formatada como a de um script de saída.

Tabela 3.1: Exemplo do dataset original

Timestamp	MAC	RSSI
1	b8:27:eb:b5:2f:a6	-40
1	b8:27:eb:b5:2f:a6	-50
2	b8:27:eb:29:66:6f	-50
3	b8:27:eb:e0:52:68	-20
3	b8:27:eb:29:66:6f	-15

Depois é possível observar na tabela 3.2 como fica formatada para ser concatenada com os outros arquivos de PRs que passarão pelo mesmo processo.

Tabela 3.2: Exemplo do dataset após processamento

Timestamp	b8:27:eb:b5:2f:a6	b8:27:eb:29:66:6f	b8:27:eb:e0:52:68	Target
1	-45	0	0	1
2	0	-50	0	1
3	0	-15	-20	1

Para realizar o tratamento dos arquivos de saída, foi utilizado um script em Python, conforme ilustrado no código 3.3. O código executa um loop que itera de 1 a 12, abrangendo todos os arquivos, uma vez que cada arquivo é nomeado de acordo com o PR responsável pela captura. Inicialmente, o arquivo CSV é aberto e os nomes dos atributos são adicionados, visto que o arquivo contém apenas os valores dos dados. Em seguida, os dados são ordenados com base no timestamp, e é definido um intervalo temporal de 1 segundo para o agrupamento dos dados. O próximo passo envolve um loop que agrupa os dados em intervalos de 1 segundo. Posteriormente, os dados são agrupados por endereço MAC, e a tabela é "pivotada"de modo que

cada valor de MAC seja associado ao seu respectivo valor de RSSI no timestamp correspondente. Dado que podem existir valores ausentes — devido a intervalos temporais nos quais pacotes de um ou mais APs não foram capturados —, é utilizada a função fillna(0) para substituir os valores ausentes (NaN) por 0. Aproximadamente 6% da base de dados possui algum atributo com o valor NaN. Finalmente, a base de dados resultante de cada arquivo é armazenada em uma lista. Após o processamento de todos os arquivos (do PR1 ao PR12), as bases de dados são concatenadas, gerando o DataFrame completo, armazenado na variável data_frame_complete.

Código 3.3: Processamento do arquivo de saída

```
pr_size = 12
cols_names = ['Timestamp', 'MAC', 'RSSI']
data_frame = []
for index in range(1, pr_size + 1):
  file = 'PR{}'.format(index)
  df = pd.read_csv(file, header=None, names=cols_names)
  df = df.sort_values(by='Timestamp')
  first_timestamp = df['Timestamp'].iloc[0]
  intervalos = []
  interval_start = first_timestamp
  interval_time_in_microsecond = 1000000 #1 segundo em microsegundo
  for i, row in df.iterrows():
      timestamp = row['Timestamp']
      if timestamp > interval_start + interval_time_in_microsecond:
        interval_start = timestamp
      intervalos.append(interval_start)
  df['Timestamp'] = intervalos
  df_agrupado = df.groupby(['Timestamp', 'MAC']).agg({'RSSI': 'mean
     '}).reset_index()
  df_pivot = df_agrupado.pivot_table(index='Timestamp', columns='
     MAC', values='RSSI', aggfunc='mean')
  df_pivot = df_pivot.fillna(0)
  df_pivot['Target'] = index
  data_frame.append(df_pivot)
data_frame_complete = pd.concat(data_frame)
```

3.2 Ambiente simulado

Para realização do experimento em um ambiente simulado, com o objetivo de realizar a criação de uma base de dados, foi executado um script que utiliza o módulo buildings do simulador NS-3, que permitiu recriar as salas utilizadas na universidade com as mesmas medidas e posicionar tanto os APs quanto os PRs.

Como parâmetro o script recebeu o número de beacons que seriam enviados (15000), o *Internal Wall Loss* (IWL) que simula a perda de sinal dentro de construções, que acontece pois há uma parede entre as salas, então foi simulado com IWL variando entre os valores 3 dBm, 5 dBm e 7 dBm.

O recorte do script presente no código 3.4, é responsável pela construção que referem-se as salas do ambiente real, conforme a figura 3.1 é possível ver que as salas possuem 6.10 metros de largura por isso a variável xsize recebe o valor de 6.1, o ysize tem o valor de 17.4 pois a cada sala tem 8.70 metros de comprimento, mas este o método de construção espera o valor total da construção, por isso definimos o valor de grid é definido como 2, assim o ambiente é formado por duas salas de 6.10m x 8.70m conforme o ambiente real. Ainda nesta primeira parte são definidos os números de APs e PRs, 3 e 12 respectivamente.

Código 3.4: Construção do ambiente

```
xsize = 6.1;
ysize = 17.4;
grid = 2;
naps = 3;
nprs = 12;

Ptr<Building> b = CreateObject <Building> ();
b->SetBoundaries (Box(0.0, xsize, 0.0, ysize, 0.0, 3.0));
b->SetBuildingType (Building::Residential);
b->SetExtWallsType (Building::ConcreteWithWindows);
b->SetNFloors (1);
b->SetNRoomsX (1);
b->SetNRoomsY (grid);
```

Em ordem, conforme o código 3.4 é realizado a criação um objeto do tipo Building, são configuradas as coordenadas referente a construção cujo o último valor é referente ao eixo z, que se refere a altura. Em seguida são configurados os valores do tipo de construção e do tipo de parede. Para recriar a sala exatamente como as do ambiente real, temos apenas 1 andar, numero de cômodos no eixo x é apenas 1 enquanto no eixo y igual a 2, vide figura 3.1.

Para inserir os APs e os PRs no ambiente simulado, pode-se observar o código 3.5, os valores usados estão todos baseados na figura 3.1 e nas medidas nela apresentada.

Os APs foram colocados em suas devidas posições espelhando o ambiente real, os posicionamentos APs foram calculados da seguinte forma, no eixo x considerando o ponto (0,0) no canto inferior esquerdo, foram inseridos nos pontos mostrados no código 3.5 de acordo com as medidas presente na imagem 3.1. Seguindo para o eixo y, foi definido com base na distância da parte mais inferior até a parte mais superior, podendo ser visto na figura 3.1.

Para a inserção dos PRs, que são distribuídos em 2 fileiras em relação a largura, foi observado que então foram gerados 3 espaçamentos, 1 antes da primeira fileira, 1 entre as fileiras e outro após a segunda fileira. Sendo assim dividindo a largura por 3, temos o valor de 2.03m, ou seja, todos os PRs da primeira fileira estão localizados a 2.03 metros do início do eixo x, seguindo o mesmo raciocínio a segunda fileira está sempre a 4.06m do início do eixo x. Em relação ao comprimento, foram gerados 4 espaçamentos, tendo em vista que são 3 fileiras deste ponto de vista. Dividindo 8.7m/4 espaçamentos, é sabido então que cada espaçamento vertical dentro da sala tem o valor de 2.175m, sendo assim, observa-se no trecho de código 3.5 que, conforme se muda o PR de fileira a um incremento de 2.175m, e após inserir todos os PRs da primeira sala, para inserir os PRs da segunda sala, é adicionado também o valor de 8.7m, que correspondem ao comprimento da sala 1.

Todos APs e PRs estavam a aproximadamente 75 centímetros de altura, por isso, em todas as linhas do trecho do código 3.5 o último parâmetro é passado como 0.75.

Desta forma, todos os APs e PRs foram adicionados em suas devidas posições,

estando de acordo com o ambiente real conforme figura 3.1.

Código 3.5: Adicionando APs e PRs no ambiente simulado

```
SetPosition(aps.Get(0), Vector(6.0, 0.4, 0.75));
SetPosition(aps.Get(1), Vector(3.2, 8.5, 0.75));
SetPosition(aps.Get(2), Vector(3.4, 16.9, 0.75));

SetPosition(prs.Get(0), Vector(2.03, 2.175, 0.75));
SetPosition(prs.Get(1), Vector(4.06, 2.175, 0.75));
SetPosition(prs.Get(2), Vector(2.03, 4.35, 0.75));
SetPosition(prs.Get(3), Vector(4.06, 4.35, 0.75));
SetPosition(prs.Get(4), Vector(2.03, 6.525, 0.75));
SetPosition(prs.Get(5), Vector(4.06, 6.525, 0.75));
SetPosition(prs.Get(6), Vector(2.03, 8.7 + 2.175, 0.75));
SetPosition(prs.Get(7), Vector(4.06, 8.7 + 2.175, 0.75));
SetPosition(prs.Get(8), Vector(2.03, 8.7 + 4.35, 0.75));
SetPosition(prs.Get(9), Vector(4.06, 8.7 + 4.35, 0.75));
SetPosition(prs.Get(10), Vector(2.03, 8.7 + 6.525, 0.75));
SetPosition(prs.Get(11), Vector(4.06, 8.7 + 6.525, 0.75));
```

3.2.1 Coleta de dados no ambiente simulado

Após o script de simulação ser executado, como resultado há um arquivo único, em formato de CSV, usando tab como separação de atributos. Como primeiro atributo surge o tempo de execução em que foi registrado o recebimento do beacon, em segundos. Em seguida tem-se respectivamente, o endereço MAC do dispositivo que enviou o beacon, ou seja, o AP e o endereço MAC do dispositivo que recebeu o beacon, ou seja, o PR. E como último atributo aparece o nível de sinal medido em SNR, que pode ser definido pela equação 3.1.

$$SNR = RSSI(dbm) - ruido\_de\_fundo(dbm) \tag{3.1}$$

Para exemplificar uma saída deste script, basta verificar o código 3.6, que traz esta representação.

Código 3.6: Exemplo de saída do script de simulação

```
0.049425  00:00:00:00:00:01  00:00:00:00:00:05  70.0736
0.049425  00:00:00:00:00:01  00:00:00:00:00:04  50.5792
0.049425  00:00:00:00:00:01  00:00:00:00:00:07  51.6377
0.049425  00:00:00:00:00:01  00:00:00:00:00:06  51.6475
0.049425  00:00:00:00:00:01  00:00:00:00:00:09  46.2907
0.049425  00:00:00:00:00:01  00:00:00:00:00:08  46.7051
0.049425  00:00:00:00:00:01  00:00:00:00:00:02  39.0784
0.049425  00:00:00:00:00:01  00:00:00:00:00:0b  42.5751
0.049425  00:00:00:00:00:01  00:00:00:00:00:0a  43.1791
```

3.2.1.1 Criação da base de dados do ambiente simulado

Para transformar a saída do script de simulação em uma base de dados estruturada da mesma forma que a base de dados do ambiente real, ou seja, o primeiro atributo sendo o timestamp, em seguida os 3 APs, com seus respectivos valores de RSSI em um determinado timestamp e por último o target que é o PR que recebeu o pacote foi executado o código 3.7.

Código 3.7: Transformando o arquivo de saída de simulação em base de dados semelhante a do ambiente real.

```
cols_names_simulado = ['Timestamp', 'MAC','MAC_RECEIVER', 'RSSI']
iwls = [3,5,7]
data_frames_iwl = {}
qty_aps = 3

for iwl in iwls:
  file = 'trace-15000-6.1-17.4-2-3-12-0-1.5-{}-1'.format(iwl)
  df_simulado = pd.read_csv(file, header=None, names=
      cols_names_simulado,delimiter='\t')
  df_simulado = df_simulado.sort_values(by='Timestamp')
  df_simulado['Target'] = [int(value[-1],16) - qty_aps for value in
       df_simulado['MAC_RECEIVER']]
  df_simulado = df_simulado.drop(axis=1,columns=['MAC_RECEIVER'])
  df_simulado = df_simulado[df_simulado['Target'] >= 1]
  df_simulado['RSSI'] = df_simulado['RSSI'] - 98

  first_timestamp = df_simulado['Timestamp'].iloc[0]
  intervalos = []
  interval_start = first_timestamp

  interval_time_in_second = 1

  for i, row in df_simulado.iterrows():
      timestamp = row['Timestamp']
      if timestamp > interval_start + interval_time_in_second:
        interval_start = timestamp
      intervalos.append(interval_start)

  df_simulado['Timestamp'] = intervalos
  df_agrupado_simulado = df_simulado.groupby(['Timestamp', 'MAC','
     Target']).agg({'RSSI': 'mean'}).reset_index()
  df_pivot_simulado = df_agrupado_simulado.pivot_table(index=['
     Timestamp','Target'], columns=['MAC'], values='RSSI', aggfunc=
     'mean')
  df_pivot_simulado = df_pivot_simulado.fillna(0)
  data_frames_iwl[iwl] = df_pivot_simulado.reset_index().drop(axis
     =1,columns=['Timestamp'])
```

O script foi executado para retornar 3 arquivos, o que muda de uma execução para outra é o valor da variável de IWL, por isso, o primeiro loop, itera sobre estes valores.

Os MACs dos APs e PRs neste código, foram atribuídos de forma sequencial e em hexadecimal. Com isso os APs ficaram com os valores de 1 a 3, já os PRs tiveram seus valores atribuídos de 4 a F (pois está em hexadecimal, ou seja, 15). Por isso após ler o arquivo, e ordenar pelo timestamp, o target é formato com int(value[-1],16), isto significa que ao iterar sobre a lista de targets ele pega o último elemento do endereço MAC e transforma para um inteiro, e depois remove a quantidade de APs, pois os targets são apenas os PRs.

Com isso é possível verificar que em seguida são filtradas as linhas da base de dados para que permaneçam apenas linhas capturadas apenas por PRs. Após essa filtragem, todos os RSSIs são somados de -98 dBm, isto serve para que neste experimento os valores de SNR dos pacotes recebidos sejam convertidos em RSSIs considerando um ruído de fundo típico em canais *Wi-Fi* de 20 MHz de -98 dBm. Depois disso, é realizado o mesmo procedimento aplicado na base de dados do ambiente real, que é agrupar por endereços MACs, e realizada média dos valores para preencher o campo de RSSI, onde cada endereço tenha um valor de RSSI em um determinado timestamp. E caso algum valor faltante, ou seja, NaN, é substuído por 0.

Pode-se observar as tabelas 3.3 e 3.4 para verificar o antes e depois dos arquivos passarem pelo código 3.7.

Tabela 3.3: Exemplo do arquivos antes do script

Timestamp	MAC_SENDER	MAC_RECEIVER	SNR
1	00:00:00:00:00:01	00:00:00:00:00:04	70
1	00:00:00:00:00:01	00:00:00:00:00:06	50
2	00:00:00:00:00:02	00:00:00:00:00:01	50
3	00:00:00:00:00:03	00:00:00:00:00:07	45
3	00:00:00:00:00:02	00:00:00:00:00:07	50

As principais mudanças a serem observadas são:

(a) Agora os MACs dos APs, se tornaram atributos da base de dados

(b) A linha com timestamp igual a 2 foi filtrada pois foi capturada por um AP

(c) Target é o número do AP que recebeu o pacote

(d) RSSI setado para 0, onde não tinha valores correspondentes

(e) Transformação de SNR para RSSI (somando -98 dBm do ruído de fundo).

Tabela 3.4: Exemplo do arquivos depois do script

Timestamp	00:00:00:00:00:01	00:00:00:00:00:02	00:00:00:00:00:03	Target
1	-28	0	0	4
1	-48	0	0	6
3	0	-48	-53	7

Ao final desta execução são salvas 3 bases dados, cada uma construída com um IWL diferentes, citados anteriormente.

Capítulo 4

Experimentos

Neste capítulo, será analisada a viabilidade de classificar dados reais com base em um modelo treinado exclusivamente com dados simulados no NS-3. O objetivo é otimizar o uso de tempo e recursos físicos, avaliando a possibilidade de determinar a localização de um dispositivo em ambientes fechados utilizando apenas o simulador.

Por isso, após a montagem das bases de dados, sendo uma coletada em ambiente real e a outra em ambiente simulado, foi escolhido um algoritmo de classificação para realizar o treinamento de um modelo de predição, o *K Nearest Neighbours* (KNN).

4.1 K Nearest Neighbours (KNN)

O KNN, é um algoritmo que busca resolver problemas de classificação usando a distância de um ponto para seus K vizinhos mais próximos, desta forma, ao comparar as K menores distâncias é encontrado a classe mais frequente entre essas K, ou seja, a que é mais provável deste ponto pertencer.

O passo a passo do algoritmo KNN pode ser descrito da seguinte forma:

(a) Definir um conjunto de dados em que cada amostra é representada por um conjunto de atributos, com uma classe associada a cada amostra.

(b) Dividir o conjunto de dados em duas partes: uma para treinamento e outra para

teste.

(c) Para cada ponto de teste, calcular a distância (como a distância Euclidiana, Manhattan, entre outras) entre o ponto de teste e todos os pontos do conjunto de treinamento.

(d) Ordenar as distâncias calculadas e selecionar os K vizinhos mais próximos do ponto de teste.

(e) A classe do ponto de teste é atribuída com base na maioria das classes dos K vizinhos mais próximos.

A escolha de um número K, não deve ser muito baixa, para que o modelo não fique muito especifico e sensível a dados restritos de determinada classes, por outro lado, K também não deve ser muito alto para que não inclua pontos não semelhantes por estar sendo influenciado por classes mais distantes.

A técnica utilizada neste trabalho consiste em realizar o treinamento do modelo diversas vezes, assim é possível verificar a acurácia(proporção entre número de acertos de modelo e o número total de amostras) em cada um dos modelos e escolher o número de K que representa melhor a base de dados.

4.2 Análise das bases de dados

Antes de treinar o modelo, foi realizada uma análise preliminar dos dados. Esse processo foi repetido também após o treinamento, com o objetivo de avaliar e compreender melhor os resultados obtidos. E ainda verificar se havia possibilidades para melhorias do modelo.

A primeira observação feita, foi a constatação de que no ambiente real nem todas as classes estão representadas com a mesma quantidade, pois podem haver perdas de pacotes, interferências físicas entre outros ruídos, já na base de dados simulada, todas as classes possuem a mesma frequência.

Utilizando a biblioteca matplotlib, com o trecho de código 4.1 foi possível exibir

um histograma, onde o eixo x representa as classes, ou seja, cada PR e no eixo y a frequência com que a classe aparece.

Código 4.1: Exibindo histograma

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
data_frame_complete['Target'].value_counts().plot(ax=ax, kind='bar'
    )
```

A figura 4.1 representa a distribuição das classes para a base de dados coletada em ambiente real, e na figura 4.2 tem-se a representação para a base de dados do simulador.

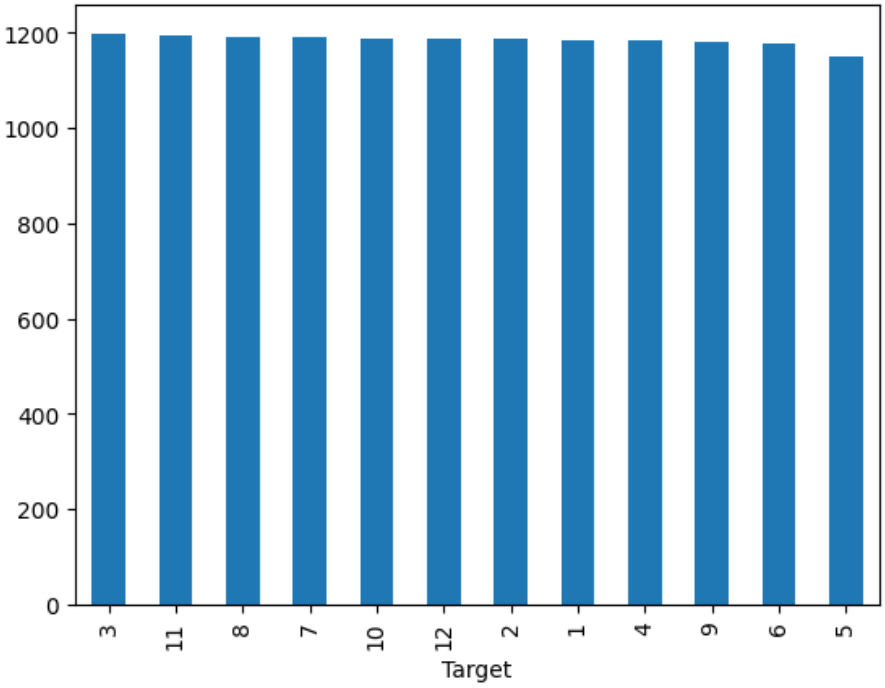


Figura 4.1: Base de dados coletada em ambiente real

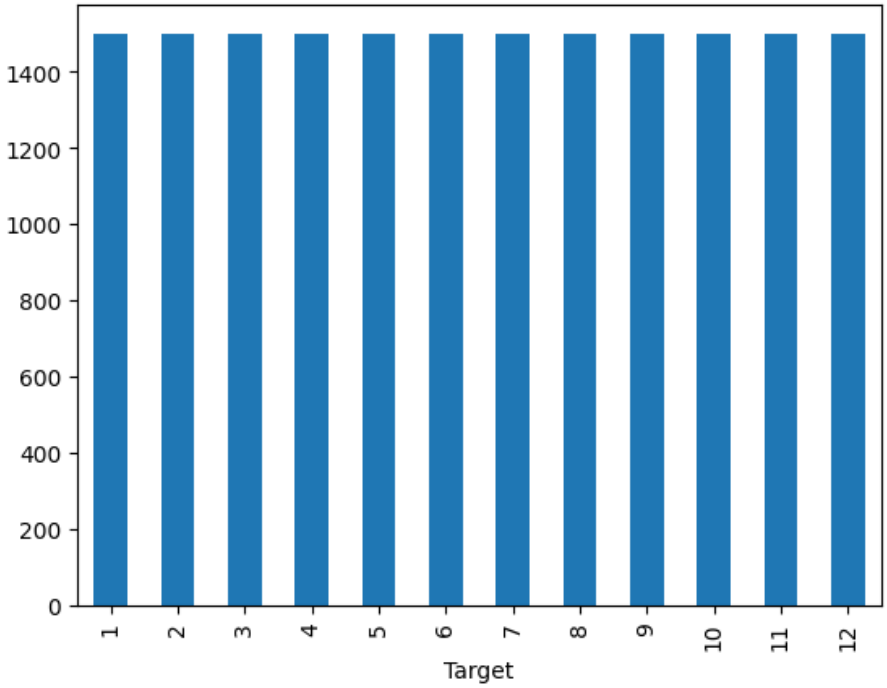
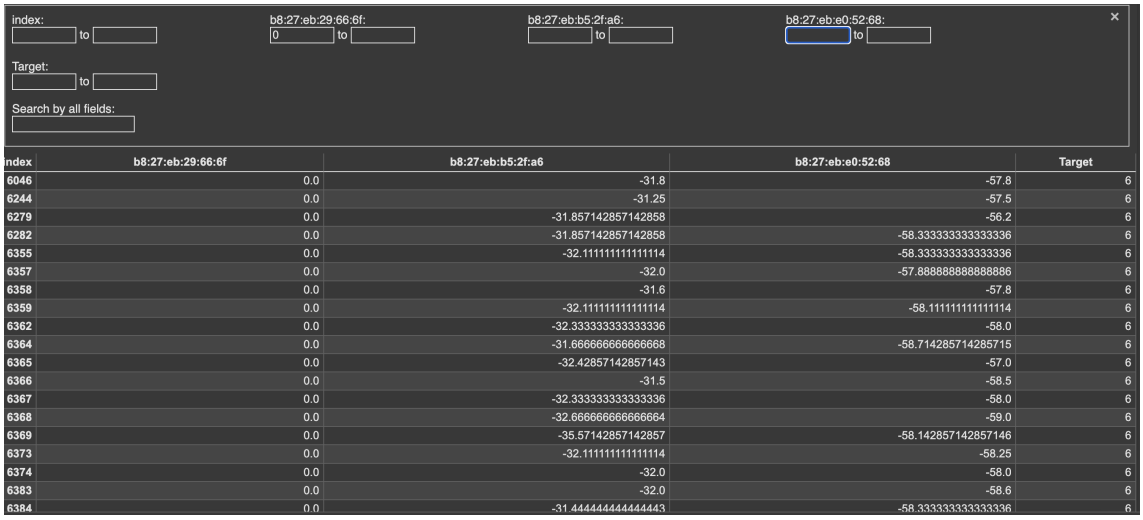


Figura 4.2: Base de dados do simulador

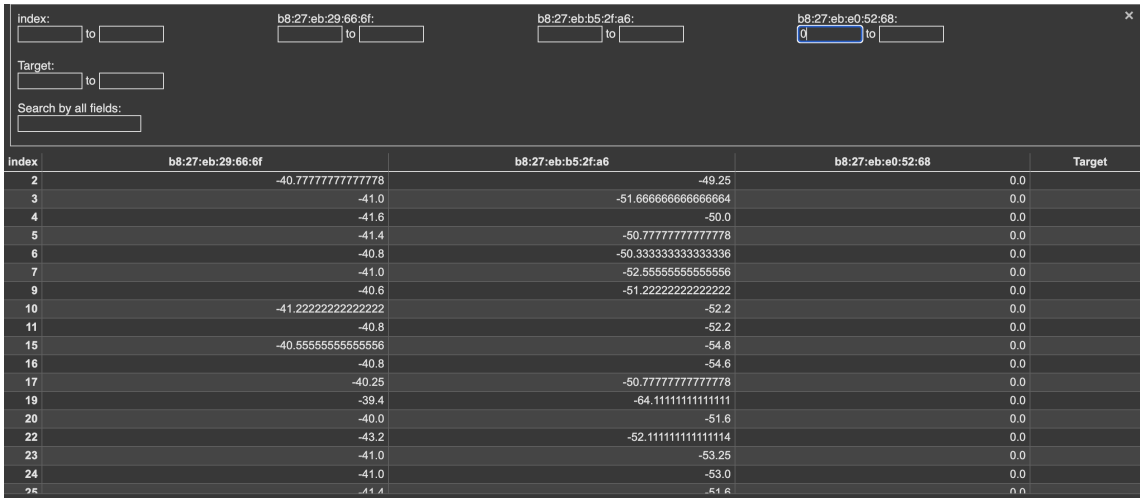
Outra observação importante foi que, na base de dados coletada em ambiente real, diversas linhas apresentaram valores de RSSI igual a 0, o que indica que, no intervalo de 1 segundo, não foram recebidos pacotes de um ou mais dos três APs. Esse fenômeno foi identificado especialmente nos APs localizados nas extremidades do ambiente real.

Essa informação foi confirmada utilizando a ferramenta de inspeção de base de dados do Google Colab. Ao analisar as figuras 4.3 e 4.4, observa-se que os AP1 e AP3 apresentam várias linhas contendo um dos valores de RSSI igual a zero. Por outro lado, na figura 4.5 que representa a busca por valor 0 no AP2, não há nenhuma linha com este valor.



index	b8:27:eb:29:66:6f	b8:27:eb:b5:2f:a6	b8:27:eb:e0:52:68	Target
6046	0.0	-31.8	-57.8	6
6244	0.0	-31.25	-57.5	6
6279	0.0	-31.857142857142858	-56.2	6
6282	0.0	-31.857142857142858	-58.333333333333336	6
6355	0.0	-32.111111111111114	-58.333333333333336	6
6357	0.0	-32.0	-57.888888888888886	6
6358	0.0	-31.6	-57.8	6
6359	0.0	-32.111111111111114	-58.111111111111114	6
6362	0.0	-32.333333333333336	-58.0	6
6364	0.0	-31.666666666666668	-58.714285714285715	6
6365	0.0	-32.42857142857143	-57.0	6
6366	0.0	-31.5	-58.5	6
6367	0.0	-32.333333333333336	-58.0	6
6368	0.0	-32.666666666666664	-59.0	6
6369	0.0	-35.57142857142857	-58.142857142857146	6
6373	0.0	-32.111111111111114	-58.25	6
6374	0.0	-32.0	-58.0	6
6383	0.0	-32.0	-58.6	6
6384	0.0	-31.444444444444443	-58.333333333333336	6

Figura 4.3: Filtragem por valores 0 no AP1



index	b8:27:eb:29:66:6f	b8:27:eb:b5:2f:a6	b8:27:eb:e0:52:68	Target
2	-40.77777777777778	-49.25	0.0	
3	-41.0	-51.666666666666664	0.0	
4	-41.6	-50.0	0.0	
5	-41.4	-50.77777777777778	0.0	
6	-40.8	-50.333333333333336	0.0	
7	-41.0	-52.55555555555556	0.0	
9	-40.6	-51.22222222222222	0.0	
10	-41.22222222222222	-52.2	0.0	
11	-40.8	-52.2	0.0	
15	-40.55555555555556	-54.8	0.0	
16	-40.8	-54.6	0.0	
17	-40.25	-50.77777777777778	0.0	
19	-39.4	-64.11111111111111	0.0	
20	-40.0	-51.6	0.0	
22	-43.2	-52.111111111111114	0.0	
23	-41.0	-53.25	0.0	
24	-41.0	-53.0	0.0	
25	-41.4	-51.6	0.0	

Figura 4.4: Filtragem por valores 0 no AP3

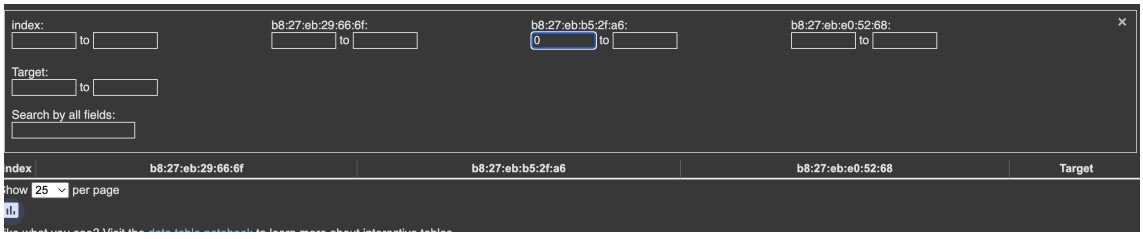


Figura 4.5: Filtragem por valores 0 no AP2

Este tipo de ruído não acontece na base de dados do ambiente simulado, por isso, surgiram algumas alternativas para tentativa de contornar este problema.

Uma opção seria remover todas as linhas da base de dados que possuem algum valor igual a 0, mas isto faria com que desbalanceasse ainda mais as classes, o que poderia afetar o treinamento e em consequência o resultado, vide figura 4.6 para observar as alterações na distribuição das classes. Outro modo de lidar com este problema seria não remover esses dados porém como citado anteriormente, este ruído não acontece na simulação, então para o atual problema, que é classificar dados reais com modelo treinado com dados simulados, iria intensificar a diferença entre os padrões vindo de uma base de dados e de outra. Por isso afim de minimizar os danos que isto poderia causar no modelo, todos os valores que eram iguais a 0 foram substituídos pela média do RSSI do respectivo PR.

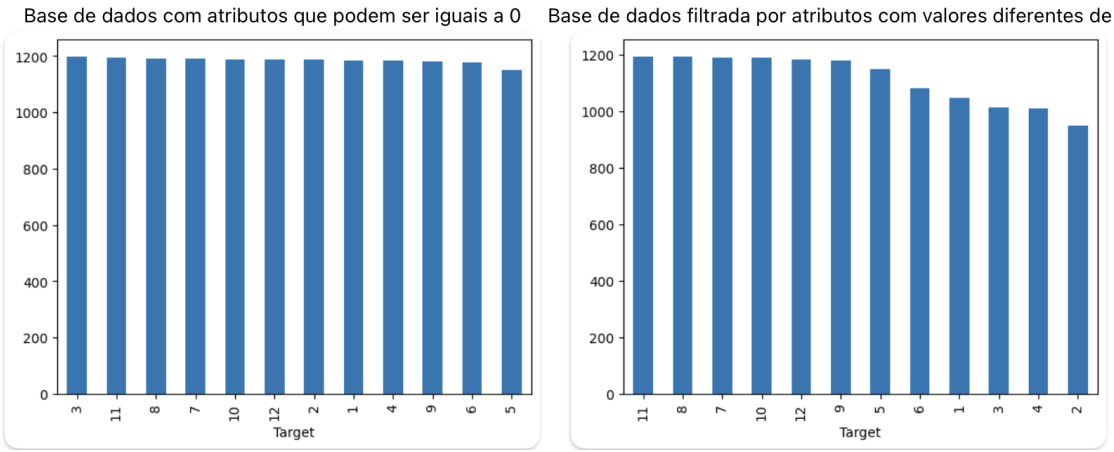


Figura 4.6: Comparação da base de dados filtrada e não filtrada

Após as mudanças sobre os dados serem feitas obteve-se a distribuição de classes das bases de dados através do t-SNE *(t-distributed Stochastic Neighbor Embedding)*, que é um algoritmo de redução de dimensionalidade usado para visualizar dados de alta dimensão em 2D ou 3D. Ele preserva as relações locais entre os pontos, agrupando itens semelhantes próximos no espaço reduzido. É amplamente utilizado para explorar e identificar padrões em grandes conjuntos de dados.

Com base nas imagens apresentadas 4.7, 4.8, 4.9, 4.10, é possível observar

algumas semelhanças entre os dados gerados pelo simulador e os dados reais, no entanto, não é evidente um padrão claro que estabeleça uma correlação consistente entre ambas as bases de dados. A análise visual sugere que, embora haja algumas similaridades superficiais, as discrepâncias entre as distribuições e características dos dados dificultam a identificação de uma relação direta e confiável entre os dois conjuntos.

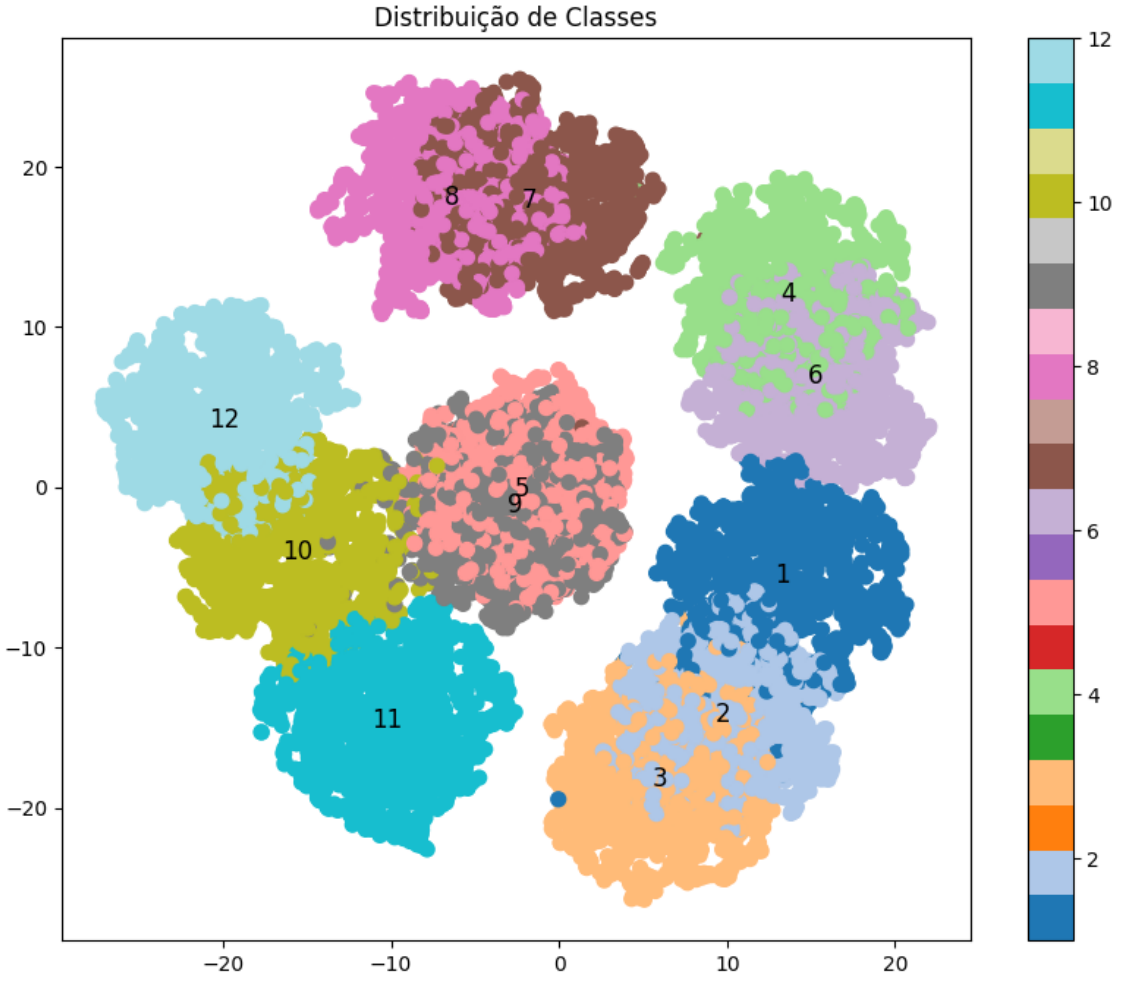


Figura 4.7: Distribuição das classes da base de dados do simulador com valor de iwl = 3

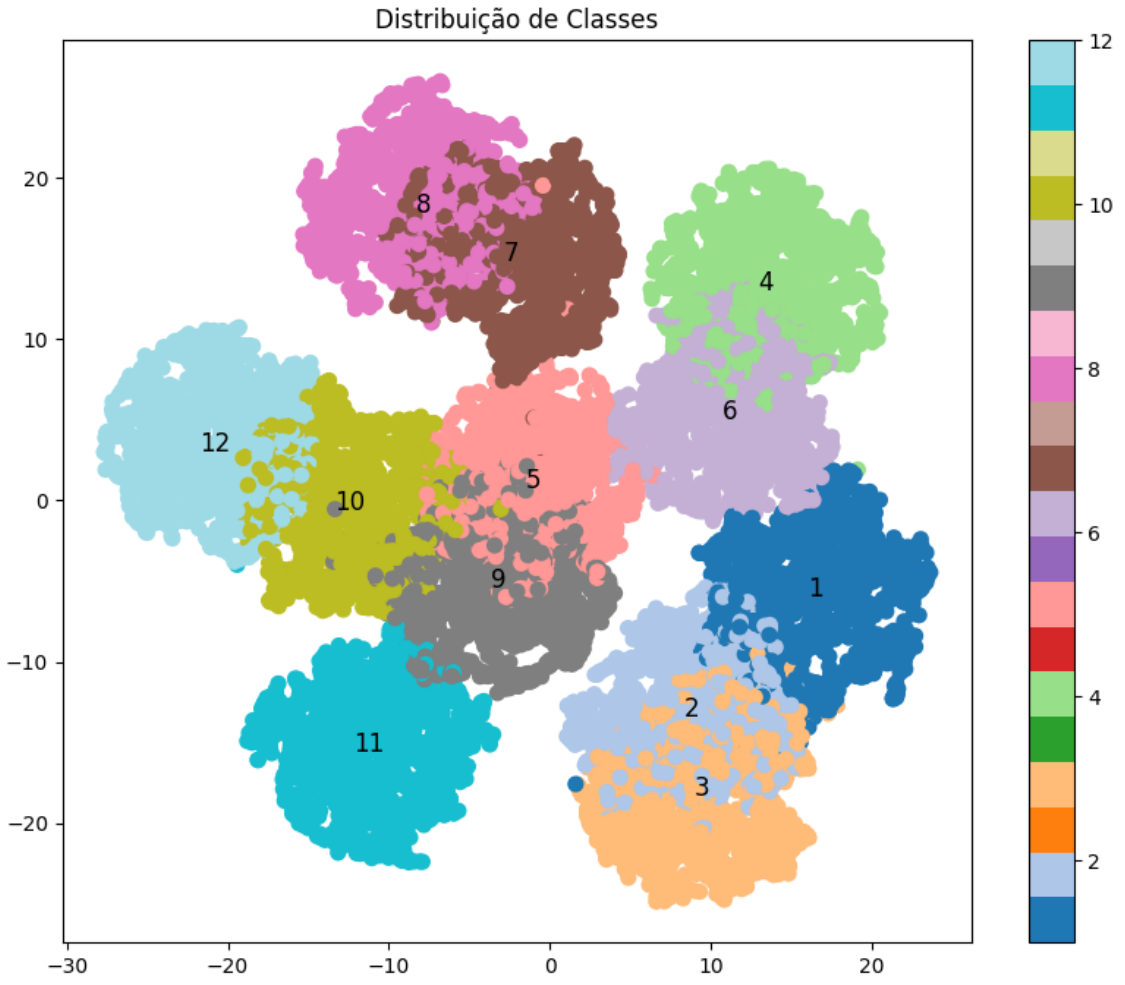


Figura 4.8: Distribuição das classes da base de dados do simulador com valor de iwl = 5

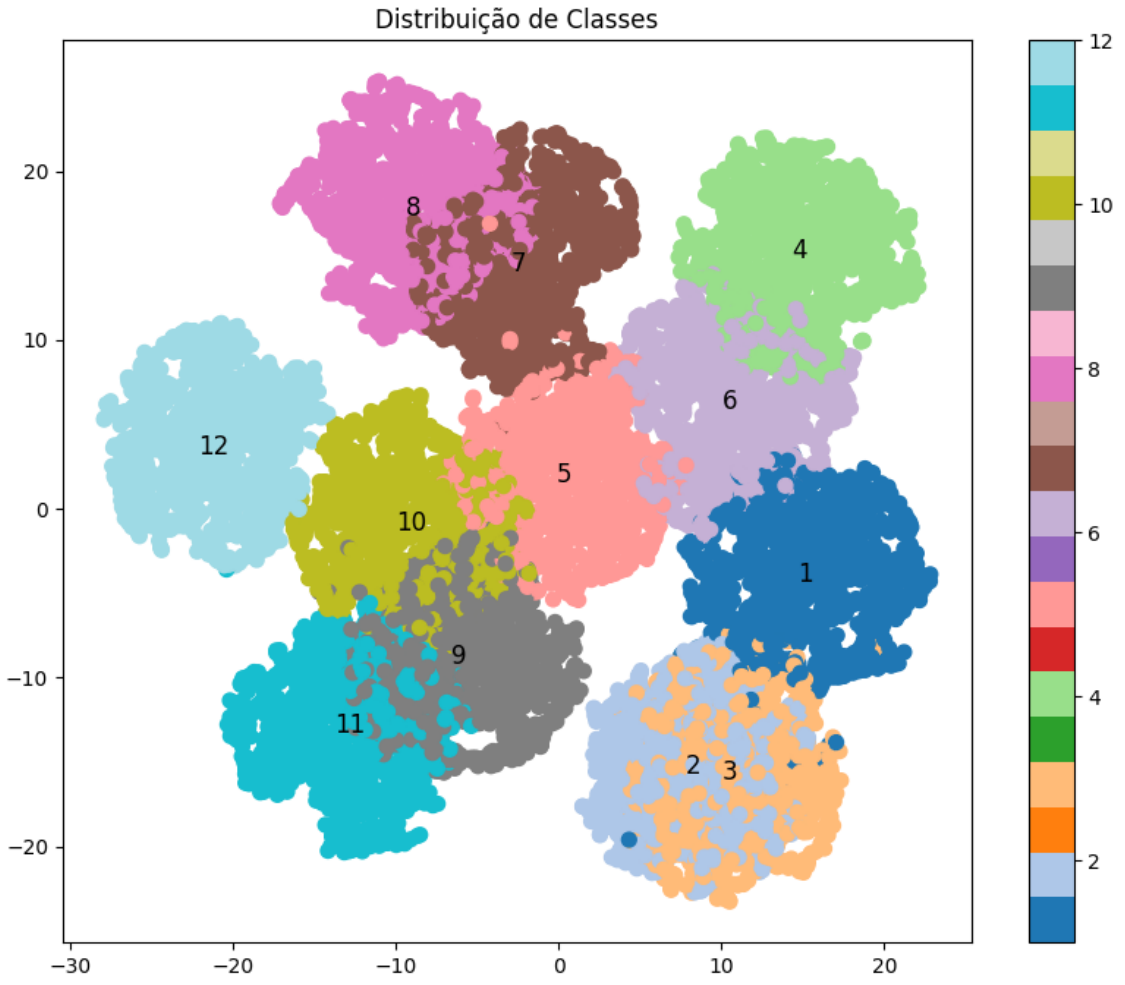


Figura 4.9: Distribuição das classes da base de dados do simulador com valor de iwl = 7

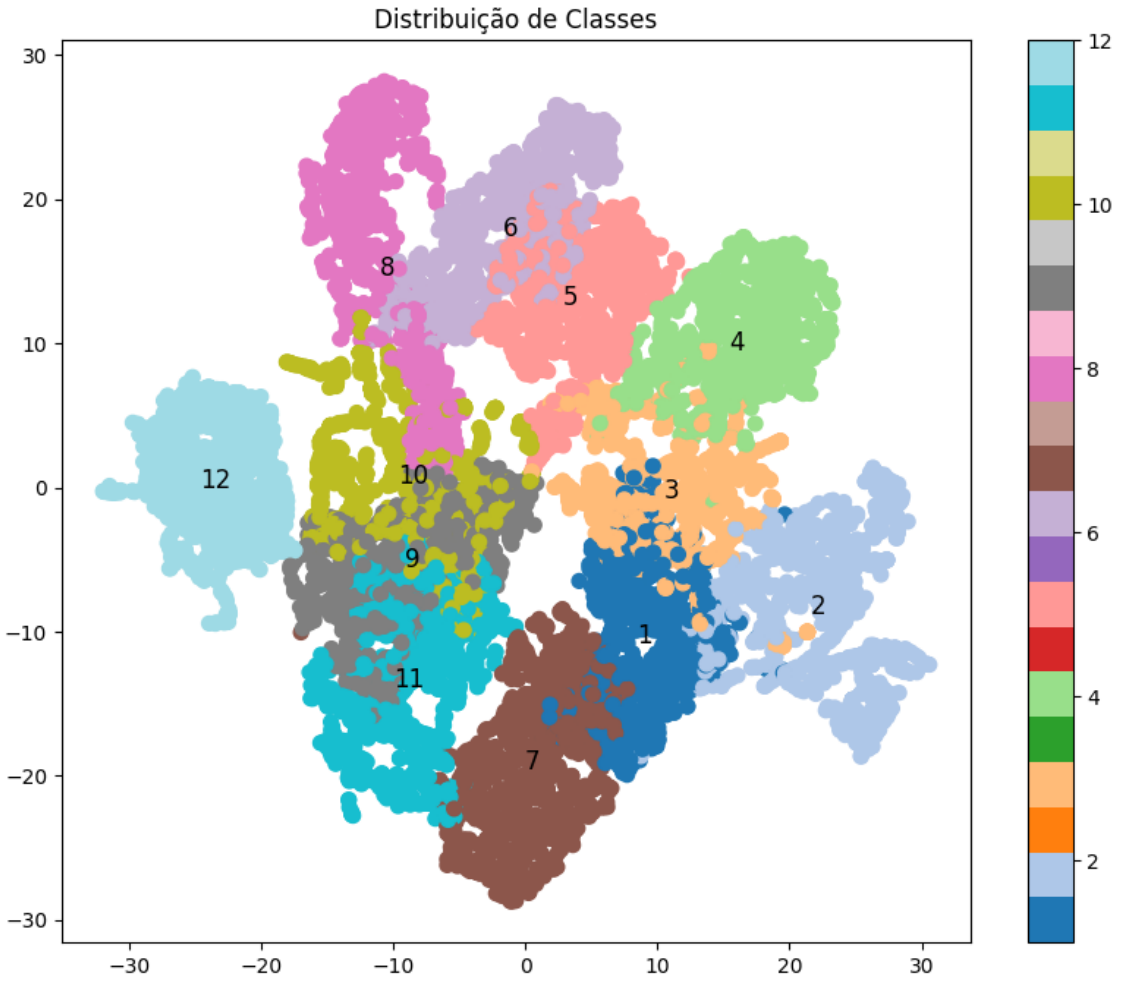


Figura 4.10: Distribuição das classes da base de dados coletada em ambiente real

4.3 Treinamento e validação do modelo

Em suma, o treinamento do modelo junto a validação do resultado foi estruturado da seguinte forma, para cada IWL, a base de dados referente era embaralhada, após este passo era realizado o treinamento com a base de dados do ambiente simulado, após esta etapa, com o melhor modelo para cada IWL era realizada a classificação, com os dados de entrada sendo a base de dados coletada em ambiente real. Após a predição, obtêm-se algumas métricas como acurácia e a matriz de confusão.

Ao percorrer o código 4.2, pode-se identificar a forma como é feito o treinamento e validação do modelo. Na primeira linha da função, os dados são normalizados aplicando a fórmula 4.1, com objetivo de diminuir a sensibilidade à escala, pois se trata de um algoritmo que é baseado em distância, logo as grandezas dos atributos impactam diretamente na classificação.

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma} \quad \text{onde} \quad x \text{ é o valor original do atributo, } \mu \text{ é a média, e } \sigma \text{ é o desvio padrão.} \tag{4.1}$$

A função train_test_split, separa a base de dados em 2 conjuntos, um de teste, que é usado para o treinamento do modelo e outro para testar o modelo, cujo tamanho é definido pelo parâmetro test_size, que neste caso foi atribuido o tamanho de 0.3, ou seja, nesta base de dados 70% será utilizada para treinar o modelo e 30% para realizar os testes.

A função fit_transform, faz as transformações de normalização de acordo com o StandardScaler e cria uma nova base de dados de acordo com os novos valores calculados, essas novas bases que serão utilizadas para treinamento e validação do modelo.

Após os ajustes nos dados visando melhorar o desempenho do modelo, é realizada a escolha do número ideal de K-vizinhos para a base de dados. Para isso, a técnica de validação cruzada (cross-validation) é aplicada, repetindo o processo N vezes(neste código 30 vezes). Em cada iteração, cria-se uma instância do modelo KNN e, com

um parâmetro cv=5 (neste caso), a base de dados é dividida em 5 partes iguais. O modelo é então treinado utilizando quatro dessas partes, enquanto a quinta é utilizada para testes. Esse processo é repetido até que todas as partes sejam usadas tanto para treinamento quanto para teste. Essa abordagem tem como objetivo avaliar a capacidade de generalização do modelo, ou seja, garantir que o modelo esteja aprendendo padrões gerais e não apenas memorizando os dados específicos.

Combinando as bibliotecas seaborn e matplotlib, é possível visualizar a escolha do melhor número de K-vizinhos, criando um gráfico correlacionando a acurácia e o valor de K. Vide figura 4.11 que mostra a escolha do melhor K para a base de dados referente ao IWL com o valor igual a 7.

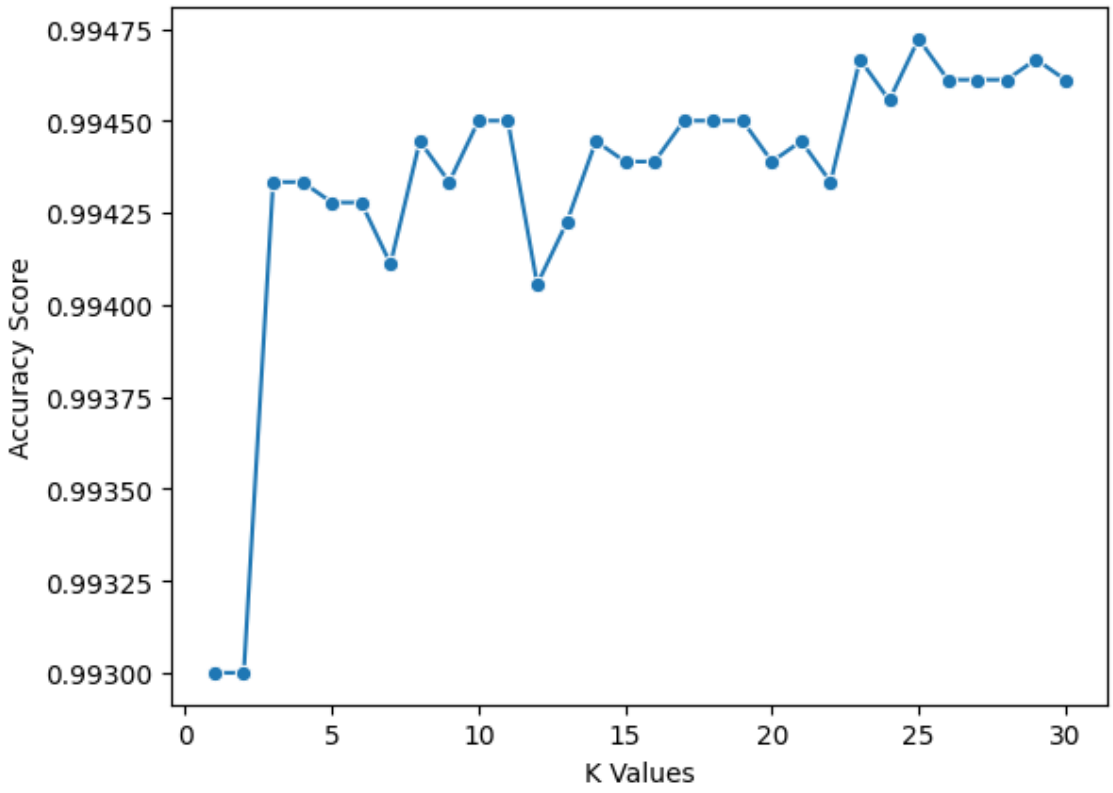


Figura 4.11: Gráfico para o IWL=7

Após a escolha do melhor K, é instanciada um novo modelo de KNN, com a melhor número para K, então é aplicada a função fit, que é responsável por realizar o treinamento do modelo, por outro lado, a função predict é responsável por realizar

a classificação, com base no treinamento realizado pela função fit.

A função accuracy_score, faz uma comparação entre a classificação esperada e classificação obtida e retorna a proporção entre número de acertos de modelo e o número total de amostras. Esta é uma métrica que traz o quanto o modelo está acertando, funciona bem para casos em que as classes estão bem balanceadas, que é o caso da base de dados simulada, caso contrário pode não exatamente dizer isto, por exemplo, em uma base hipotética que tem 16 itens A e 4 itens B, se o modelo apenas acertasse a classe A, ainda teria uma acurácia de 80% mas não significa que é um bom modelo pois nunca acerta a classe B.

Por último, antes de retornar o melhor modelo, é construída uma matriz de confusão, que é uma métrica utilizada para entender melhor a predição do modelo, nesta matriz que tem o tamanho NxN, sendo N o número de classes da base de dados, é possível verificar quando uma classe foi classificada corretamente, ou quais classes estão sendo confundidas, ou classificadas juntas.

Código 4.2: Treinamento e validação

```
def trainWithKNN(train_x,train_y):
  scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(train_x,
     train_y, test_size=0.3)
  X_train = scaler.fit_transform(X_train)
  X_test = scaler.fit_transform(X_test)
  k_values = [i for i in range (1,31)]
  scores = []
  X = scaler.fit_transform(train_x)

  for k in k_values:
      knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
      score = cross_val_score(knn, X, train_y, cv=5)
      scores.append(np.mean(score))
  sns.lineplot(x = k_values, y = scores, marker = 'o')

  plt.xlabel("K Values")
  plt.ylabel("Accuracy Score")

  best_index = np.argmax(scores)
  best_k = k_values[best_index]

  knn = KNeighborsClassifier(n_neighbors=best_k)
  knn.fit(X_train, y_train)
  y_pred = knn.predict(X_test)

  accuracy = accuracy_score(y_test, y_pred)
  print("Accuracy:", accuracy)

  index = ['PR1','PR2','PR3','PR4','PR5','PR6','PR7','PR8','PR9','
     PR10','PR11','PR12']
  cm_matrix = confusion_matrix(y_test,y_pred)
  cm = [[j/sum(i) for j in i] for i in cm_matrix]
  cm = pd.DataFrame(cm,index,index)
  plt.figure(figsize=(10,5))
  sns.heatmap(cm, annot=True,fmt=".2%")

  return knn
```

4.4 Resultados

O treinamento do modelo, independentemente do valor de IWL, resultou em uma acurácia de aproximadamente 99%. Em outras palavras, ao treinar o modelo com 70% da base de dados e utilizá-lo para prever os 30% restantes, observou-se que, em média, 99 de cada 100 predições feitas pelo modelo estavam corretas.

Alguns fatores que podem ter contribuído para a alta acurácia do modelo incluem as características da base de dados utilizada. Inicialmente, o arquivo CSV gerado pelo script de simulação possuía 629.867 linhas, mas, após os processamentos e tratamentos descritos no capítulo de proposta, a base foi reduzida para 18.000 linhas, com 12 classes bem definidas e balanceadas, cada uma contendo exatamente 1.500 amostras. Essa combinação de uma base grande, poucas classes, e equilíbrio na quantidade de amostras por classe favorece a aprendizagem eficiente do modelo, contribuindo para seu desempenho elevado.

Em contra partida, o modelo treinado com os dados do simulador não apresentou um bom desempenho para classificar a base de dados coletada em ambiente real, vide tabela 4.1 que contém a acurácia de cada modelo para seu respectivo IWL.

Tabela 4.1: Acurácia aproximada para cada modelo com respectivo IWL

	IWL=3	IWL=5	IWL=7
acurácia	38%	34%	32%

Com a métrica da matriz de confusão é possível compreender o motivo desta baixa acurácia. Respectivamente as figuras 4.12, 4.13, 4.14, mostram a matriz de confusão para os IWL 3, 5 e 7 e traz os dados de erros e acertos de predição, como as classes preditas erradamente foram classificadas. Assim dando um poder maior para analisar o motivo por trás disto.

As matrizes nos mostra que quanto maior o IWL nos PR1,PR2 e PR3, mais o modelo agrupou essas classes, errando a predição. Por exemplo, com o IWL=3,

61.86% PR1 foi foi classificado como PR3, já com IWL=7, 99.41% do PR1 foi classificado como PR3. Por outro lado quando se trata do PR4 o aumento do IWL teve o efeito inverso, acabou obtendo mais erros, confundindo com o PR1.

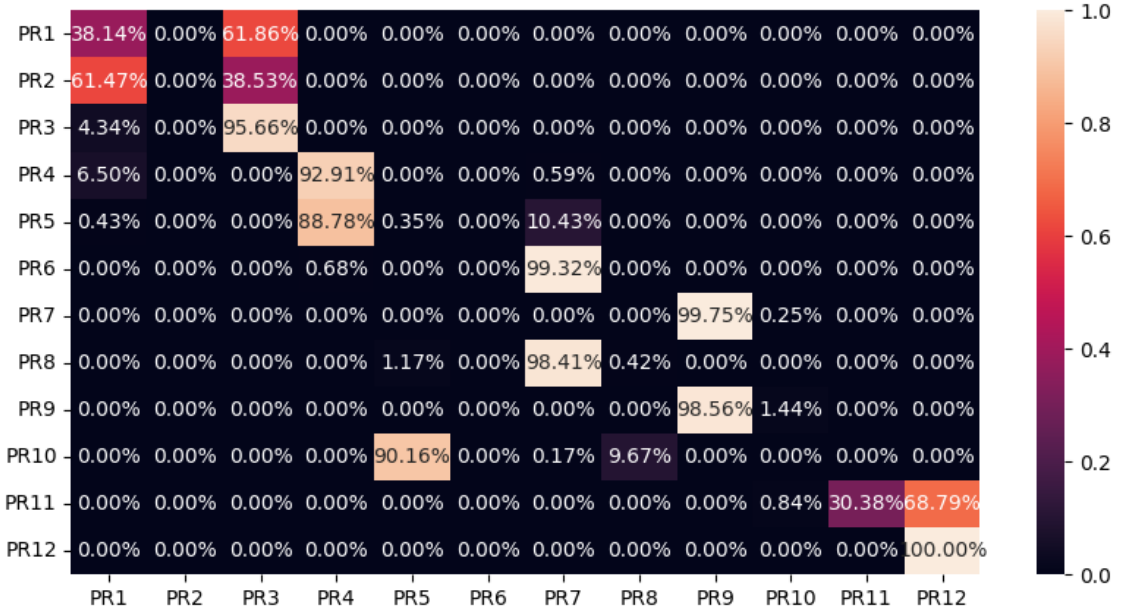


Figura 4.12: Matriz de confusão com IWL=3

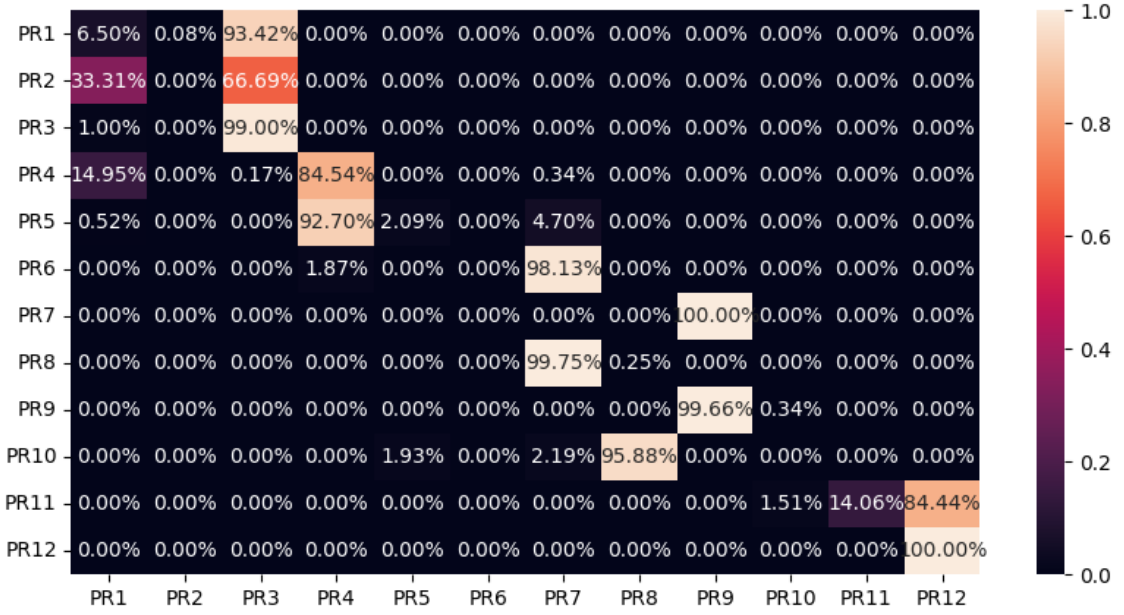


Figura 4.13: Matriz de confusão com IWL=5

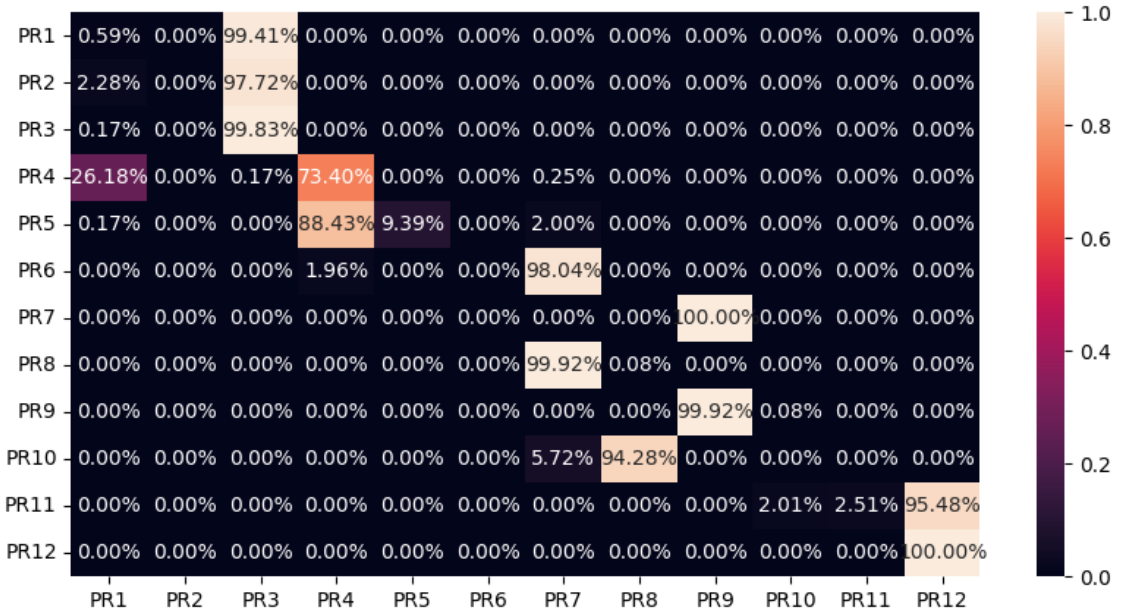


Figura 4.14: Matriz de confusão com IWL=7

Analisando as matrizes de confusão junto a figura 3.1, não é possível relacionar diretamente o agrupamento de classes, com a posição das mesmas nas salas. Por exemplo, o PR1, PR2 e PR3 foram classificados como se fossem os mesmos PRs, ou seja, quando era para classificar um ponto do base de dados como PR3, era classificado corretamente, no entanto, o PR1 e PR2 também foram classificados como PR3. Apesar de o PR4 e o PR2 estarem mais próximos, não foram classificados como o mesmo PR nenhuma vez. O que mostra que pode não ser apenas um erro no cálculo de distância do modelo ou um erro no número de K-vizinhos.

Tendo isso em vista, um fato importante a ser percebido é que independente do valor de IWL, somando todas as predições, o PR2 e o PR6 receberam menos de 1% das classificações tanto corretas quanto incorretas. Também acontece algo parecido com o PR10. Para tentar analisar as razões pelas quais isto pode ter acontecido, foi feito para cada valor de IWL, um gráfico que traz a média de RSSI para cada AP em relação a classe correspondente, o mesmo foi feito para a base de dados coletada em ambiente real para que possa haver uma comparação entre os padrões.

Com os gráficos representados nas figuras 4.15, 4.16, 4.17, pode-se verificar que

apesar da mudança do valor de IWL, o padrão dos dados continuam parecidos. O mesmo não ocorre se comparado com o padrão da base de dados do ambiente real representado pela figura 4.18, que apresenta padrões bem diferentes, o que pode ter contribuído para baixa acurácia do modelo para classificar os dados do ambiente real.

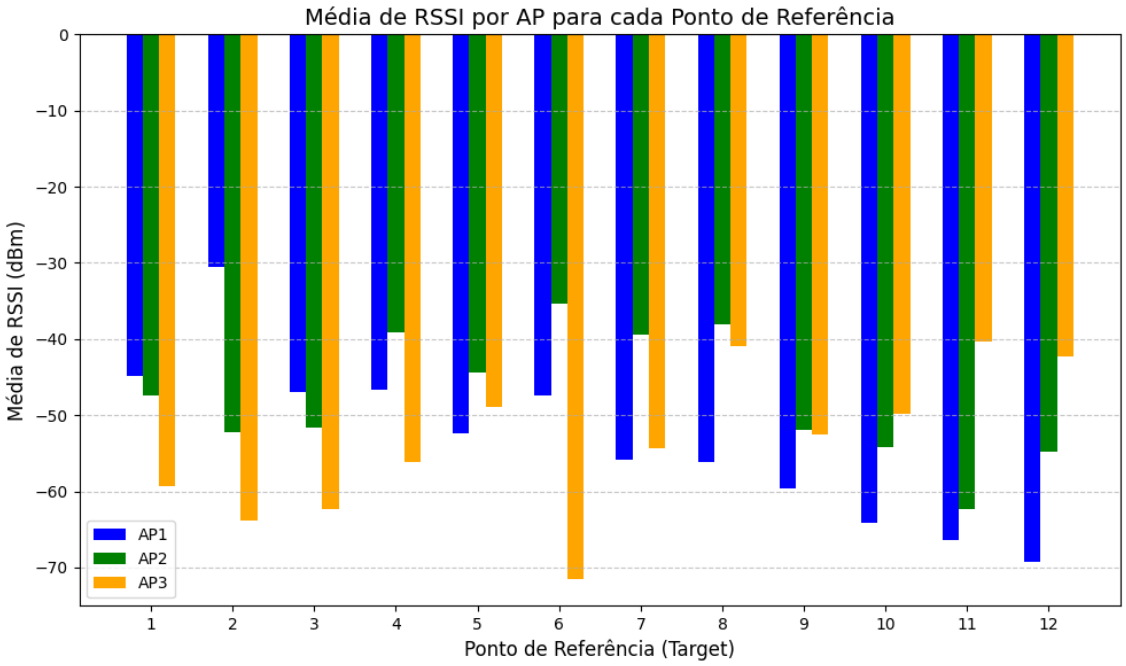


Figura 4.15: Distribuição de RSSI por AP com IWL=3

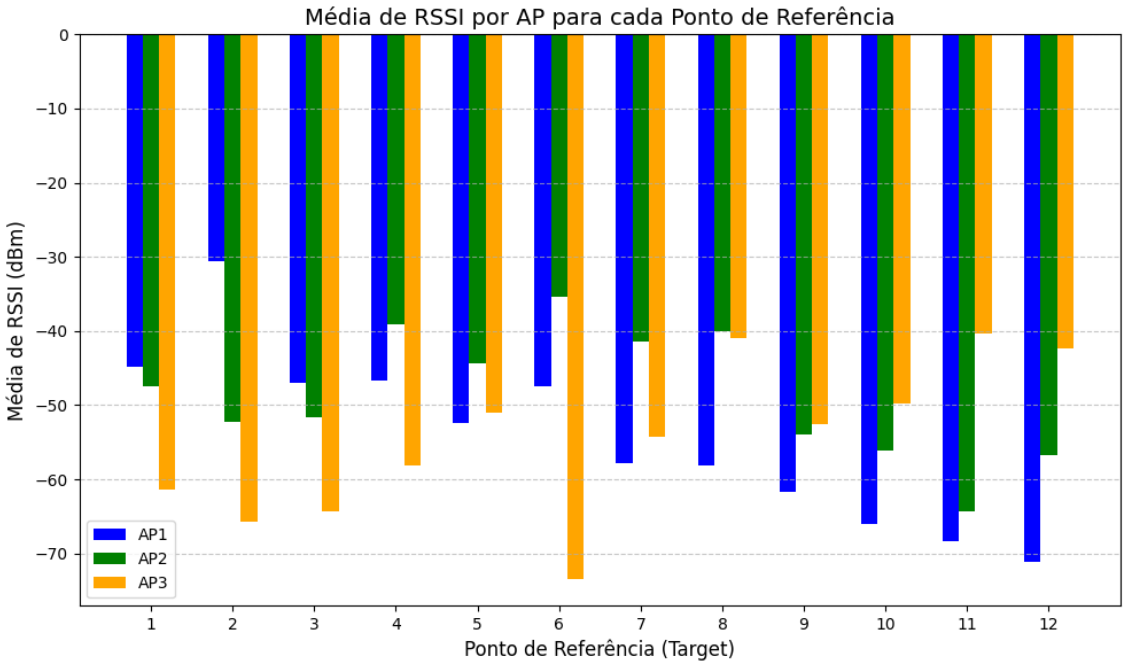


Figura 4.16: Distribuição de RSSI por AP com IWL=5

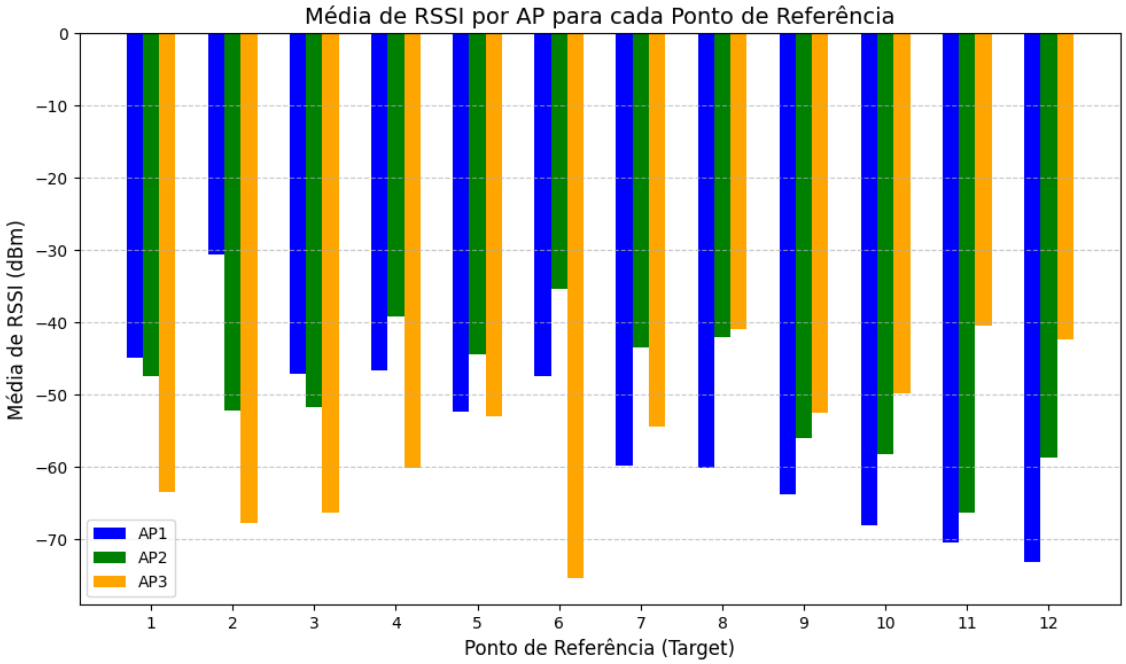


Figura 4.17: Distribuição de RSSI por AP com IWL=7

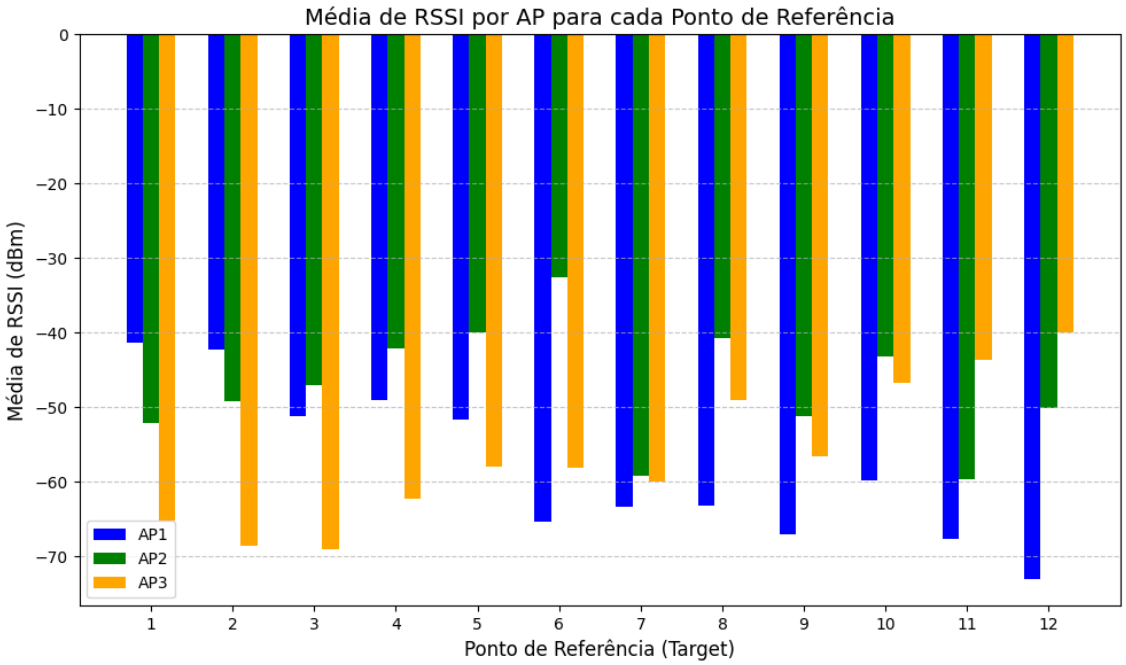


Figura 4.18: Distribuição de RSSI por AP no ambiente real

Pode-se observar que, no caso do PR6, que apresentou 0% de classificação pelo modelo, há uma clara discrepância nos padrões de RSSI entre os dados simulados e os dados coletados no ambiente real. Nos dados simulados, o RSSI do AP3 é o menor entre os APs, seguido pelo AP1, enquanto o maior valor é registrado no AP2. Já na base de dados do ambiente real, essa ordem se inverte: o AP1 apresenta o menor valor, seguido pelo AP3, com o AP2 registrando o maior valor.

Além disso, a diferença nos valores absolutos de RSSI entre o AP1 e o AP3 no simulador é de quase 30, enquanto no ambiente real essa diferença é de aproximadamente 10. Esse exemplo ilustra a diferença significativa nos padrões das bases de dados simuladas e reais, um comportamento que também se observa em outras classes (PRs).

Com isso, um dos motivos da queda de desempenho entre a predição na base de dados com o conjunto de teste do simulador e os dados do ambiente real pode ser por essa disparidade de padrão entre as bases de dados.

Capítulo 5

Conclusão

Neste capítulo serão feitas considerações finais sobre o trabalho realizado e a relação entre o resultado esperado com o resultado obtido, além de falar sobre limitações e possíveis melhorias que podem ser aplicadas em trabalhos futuros.

5.1 Considerações finais

Este trabalho teve como objetivo principal analisar a viabilidade de utilizar dados coletados em ambientes simulados para estimar a localização em ambientes reais, oferecendo uma alternativa que reduz a necessidade de esforços físicos, economiza tempo e otimiza recursos financeiros e logísticos. A relevância dessa abordagem é evidente, pois afeta diretamente a experiência de pessoas que dependem de sistemas de localização precisos e confiáveis, especialmente em cenários onde o funcionamento do principal meio de localização atual, o GPS, é limitado.

O GPS desempenha um papel crucial na navegação ao ar livre, mas sua precisão diminui muito em ambientes internos devido à perca do sinal, interferências e obstáculos físicos. Em consequência disto, a localização *indoor* ainda enfrenta desafios significativos, mas pode servir como uma alternativa neste campo.

Por isso, este trabalho propõe uma alternativa para tentar superar essas limitações, mas também explora um campo de pesquisa que pode transformar a maneira como

os sistemas de localização funcionam em ambientes *indoor*.

O desempenho observado pode ser parcialmente atribuído a limitações físicas inerentes ao sistema, como os erros médios causados pela distância entre os pontos de referência PRs. Esses erros podem ser mitigados por meio de estratégias como o agrupamento dos PRs, o que ajudaria a reduzir a dispersão e melhorar a precisão do sistema. Outra explicação relevante para o desempenho observado está nas diferenças significativas entre as bases de dados geradas pelo simulador e aquelas coletadas no ambiente real. Tais discrepâncias podem ser originadas por vários fatores, incluindo simplificações nos modelos de simulação, que não conseguem capturar com total fidelidade a complexidade do ambiente real. Além disso, as variações nos padrões de propagação do sinal, que podem ser influenciadas por obstáculos físicos, interferências e outros fenômenos imprevisíveis, bem como a presença de ruídos específicos do ambiente real, contribuem para essas diferenças. Como resultado, essas inconsistências nos *fingerprints* — representações da intensidade do sinal em diferentes pontos do ambiente — afetaram a precisão do sistema de localização.

5.2 Trabalhos futuros

Para trabalhos futuros, recomenda-se aprofundar a investigação em métodos para melhorar a simulação do ambiente real, incorporando uma gama mais ampla de fatores que possam influenciar a forma como os dados são transmitidos pelos APs e recebidos pelos PRs. Pode-se buscar variar mais os fatores de ruídos, propagação de sinal entre outros fatores.

Além disso, é possível explorar diferentes formatos de *fingerprints*. Por exemplo, um fingerprint pode ser representado como um vetor das diferenças entre os valores de RSSI de pares de Access Points, como [AP1-AP2, AP2-AP3, AP3-AP1]. Para trabalhos que exigem monitoramento ao longo do tempo, é recomendável incluir o timestamp, pois o horário pode influenciar significativamente o sinal devido a fatores como a movimentação de pessoas ou mudanças na densidade de dispositivos conectados à rede.

Referências

ANLIX. *SNR e RSSI: tudo o que você precisa saber sobre*. [S.l.], 2022. Disponível em: <https://anlix.io/rssi-e-snr-tudo-o-que-voce-precisa-saber-sobre/>.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. [S.l.]: Springer Science+Business Media, LLC, 2006.

CAMPOS, M. *GPS - Sistema de posicionamento global*. [S.l.], 2024. Disponível em: <https://mundoeducacao.uol.com.br/geografia/gps-sistema-de-posicionamento-global.htm#:~:text=As%20caracter%C3%ADsticas%20do%20GPS%20permitem,e%20mapeamento%20de%20diferentes%20%C3%A1reas.>

CLáUDIO RODOLFO. GOLçAVES, M. *TCC - Rodolfo e Marcus*. [S.l.], 2024.

CUMMINGS PETER JORDAN, M. Z. V. *The Oxford Handbook of the Archaeology and Anthropology of Hunter-Gatherers*. [S.l.]: Oxford University Press, 2014.

FERREIRA RENATO. QUEIROZ, V. *Simulação de Redes Sem Fio para o Problema da Localização Indoor*. [S.l.], 2021.

HASTIE T., T. R. . F. J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.)*. [S.l.]: Springer Science+Business Media, LLC, 2009.

IEEE. *IEEE 802.11*. [S.l.], 2024. Disponível em: <https://libtins.github.io/tutorial/802.11/>.

JAMES G., W. D. H. T. . T. R. An introduction to statistical learning: with applications in r. *Springer Science+Business Media, LLC*, 2013.

MARINO, T. B. *GPS - Sistema de Posicionamento por Satélites Artificiais*. [S.l.], 2012. Disponível em: <http://www.ufrrj.br/lga/tiagomarino/aulas/7%20-%20GPS.pdf>.

NI, H. An improved method of self-adaptive localization for wireless sensor network in dynamic indoor environment. *31st Chinese Control Conference.*, 2012.

NIU, F. L. J. L. Y. Y. W. W. D. H. P. C. Q. Survey on wifi-based indoor positioning techniques. *IET The Institution of Engineering and Technology*, Elsevier, v. 14, n. 9, p. 1372–1383, 2020.

NSNAM. *ns-3*. [S.l.], 2024. Disponível em: <https://www.nsnam.org/about/>.

SILVA MARCEL WILLIAM ROCHA; ZAMITH, M. Avaliação de técnicas de localização indoor por fingerprint de rssi com simulações no ns-3. 2022.

UOMALA J.; HAKALA, I. Towards adaptive localization in wireless sensor networks. *Ubiquitous Positioning, Indoor Navigation, and Location Based Service*, p. 1–8, 2012.