UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Társis Natan Boff da Silva

SISTEMA DE LOCALIZAÇÃO EM AMBIENTES FECHADOS UTILIZANDO LORA WAN

Santa Maria,
RS 2019

Társis Natan Boff da Silva

SISTEMA DE LOCALIZAÇÃO EM AMBIENTES FECHADOS UTILIZANDO LORA WAN

Engenharia de Computação apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de Engenheiro de Computação.

ORIENTADOR: Prof. Carlos Henrique Barriquello

Santa Maria, RS
2019

AGRADECIMENTOS

Agradeço primeiramente a meus pais, Gení e Gilmar, que sempre me deram grande apoio e me permitiram cursar a graduação de Engenharia.

Agradeço a minha irmã Bárbara, que sempre acreditou em mim e esteve comigo nos momentos de dificuldades, me ajudando de toda a maneira possível.

Aos meus amigos Victor e Luandy, que foram e são uma importante parte da minha trajetória, se colocando por diversas vezes como minha segunda família.

Ao meu professor e orientador Barriquello, por dispor de tempo, recursos e conhecimentos necessários para compor o presente trabalho.

Enfim, agradeço a todos que me ajudaram a seguir firme na caminhada e nunca desistir de correr atrás dos meus sonhos.

"Se vi mais longe, foi por estar apoiado sobre ombros de gigantes."

Isaac Newton

"Ninguém vai bater mais forte do que a vida. Não importa como você bate e sim o quanto aguenta apanhar e continuar lutando; o quanto pode suportar e seguir em frente. É assim que se ganha."

Stallone S. em Rocky *Balboa, 2007*

RESUMO

SISTEMA DE LOCALIZAÇÃO EM AMBIENTES FECHADOS UTILIZANDO LORA WAN

AUTOR: Társis Natan Boff da Silva
ORIENTADOR: Carlos Henrique Barriquello

O presente trabalho mostra um estudo com caráter de pesquisa tecnológica, relacionado ao uso da tecnologia LoRa pra geolocalização indoor, bem como, métodos comumente utilizados em tecnologias de rede sem fio para a obtenção de geolocalização. Serão abordados conceitos a respeito do protocolo LoRaWAN e dois algoritmos de aprendizagem de máquina. Em seguida será implementada uma rede LoraWAN e feitos experimentos, para verificar o comportamento da intensidade do sinal irradiado pelos nós da rede em relação a distância entre eles, analisando assim, a ideia de utilizar os algoritmos e a intensidade do sinal para tentar fazer predições a cerca da localização de um dos nós da rede.

Palavras-chave: IoT. Geolocalização. LoRa. KNN. SVM. Machine Learning.

ABSTRACT

INDOOR LOCATION SYSTEM USING LORA WAN

AUTHOR: Társis Natan Boff da Silva
ADVISOR: Carlos Henrique Barriquello

The present work shows a study of technological research related to the use of LoRa technology for indoor geolocation, as well as methods commonly used in wireless network technologies to obtain geolocation.Concepts will be approached regarding the LoRaWAN protocol and two machine learning algorithms. Then, a LoraWAN network will be implemented and experiments will be carried out to verify the behavior of the signal intensity radiated by the network nodes in relation to the distance between them, analyzing the idea of using the algorithms and the signal strength to try to predict about the location of one of the network nodes.

Key words: IoT. Geolocalização. LoRa. KNN. SVM. Machine Learning.

Conteúdo

1 INTRODUÇÃO

Em 1999 o termo “internet das coisas” era usado pela primeira vez na história, pelo pesquisador Kevin Ashton, do Massachusetts Institute of Technology (MIT), em uma apresentação à empresa Procter & Gamble. Nesta apresentação Ashton descreveu um sistema em que as “coisas do mundo real” (computadores, carros, eletrodomésticos, objetos...) estariam conectadas entre si, trocando informação a todo tempo, dando origem ao que chamamos hoje de sistema IoT(*internet of things*). (FINEP, 2015)

Por tratar-se de uma idéia abstrata, não existe um consenso do que de fato é um sistema IoT. Porém pode ser ilustrado como uma rede, cujos nós (coisas) podem estar em qualquer lugar no espaço, conectados entre si através da internet via endereçamento IP. Os nós podem trocar informações entre si e reportar seu estado de funcionamento, dando origem a um sistema inteligente de sensores e atuadores (Sônego, 2016).

No Brasil existe um grande esforço para utilizar idéias IoT em três principais segmentos: competitividade econômica, criando novos modelos de negócios mais eficientes; conectar a sociedade, trazendo melhorias no gerenciamento de recursos em cidades; promover a cadeia produtiva de equipamentos IoT, com objetivo de exportar tecnologia . (BNDES, 2017). Segundo o BNDES, estima-se que até 2025 o país pode movimentar até U$132,00 bilhões nos quatro principais setores de uso de IoTs (Cidades, Rural, Industrial e Saúde).

1.1 GEOLOCALIZAÇÃO E REDES IOT

Em alguns casos é necessário saber a posição geográfica em que os nós de uma IoT se encontram. Bons exemplos que ilustram essa necessidade são sistemas de navegação, busca de objetos, monitoramento de recursos distribuídos num espaço geoGráfico, etc.

A tecnologia mais popular para geolocalização hoje é o Sistema de Posicionamento Global (GPS), porém seu uso se restringe a áreas abertas devido à necessidade de existir linha

de visada entre os satélites e o ponto que se quer encontrar. Para soluções em espaço indoor, existem várias opções de tecnologias de comunicação sem fio como *wifi*, *Bluetooth Low Energy* (BLE), *zigbee*, juntamente com inúmeros algoritmos que usam propriedades do sinal, como tempo de chegada, ângulo de chegada ou a potência do sinal recebida para determinar o posicionamento de um determinado nó da rede (Haute, 2016).

O presente trabalho se trata de uma pesquisa de iniciação tecnológica, na qual se tenta investigar e apontar possíveis resoluções para o problema da geolocalização indoor dos nós de uma rede IoT usando a tecnologia LoRaWan, uma tecnologia recente e com grande crescimento mundial no mercado IoT.

LoRaWan é uma arquitetura de rede feita a partir de dispositivos com modulação LoRa (*Long Range*). Foi desenvolvida em 2012 pela LoRa Alliance, uma aliança de tecnologia sem fins lucrativos com mais de quinhentas empresas associadas, na França. Sua proposta, é integrar dispositivos IoT através de grandes distâncias consumindo pouca energia, caracterizando uma LPWAN (*Low Power Wide Area Network*)(LORA ALLIANCE, 2019) Estima-se que em 2019, dispositivos LoRa tomaram 75% do mercado de IoT, contra 25% do 5G(Blackman, 2019).

1.2 MOTIVAÇÃO

Um sistema que reconheça geolocalização de nós de uma rede IoT pode auxiliar-se de dados como localização de máquinas ativas e pessoas, para ajudar no processo de tomada de decisões e tornar processos de alocação de recursos mais eficientes.

Tem grande utilidade em casos de gerenciamento de máquinas móveis que possuem um custo de locomoção caro dentro de espaços muito segmentados. Um bom exemplo é um equipamento de eletrocardiograma em um hospital, é uma máquina cara e de difícil locomoção, que pode mudar de sala e andar dentro de um hospital várias vezes em um dia. Saber a localização deste equipamento agiliza sua busca, podendo economizar tempo de espera em caso de uso de emergência. Outro exemplo é ilustrado no setor de logística, em que traz a possibilidade de escolher a melhor rota para equipamentos de movimentação interna como empilhadeiras e robôs, assim como o monitoramento dos recursos e produtos (NOVIDA, 2019). Um exemplo geral e voltado para problemas cotidianos seria um sistema de busca em uma área residencial. Nesse caso o usuário teria um mapeamento de todas as

salas e lugares de sua casa que desejasse cadastrar no sistema, podendo saber o local ou rota definida por um agente, que neste caso podem ser objetos de valor ou pets de estimação. Este é o caminho a se seguir nesta pesquisa, verificar se é possível utilizar sistemas IoT em rede LoRaWan, para mapear quadrantes ou espaços contidos em salas, previamente definidos pelo usuário (ex: mesa de centro da sala de estar, estante do quarto, área de serviço) com alguma precisão.

1.3 TRABALHOS SEMELHANTES

Inicialmente discutiu-se sobre implementar um sistema de geolocalização para áreas abertas (outdoor), porém torna-se mais difícil realizar testes em um ambiente amplo e externo por necessitar-se de receptores (gateways) mais potentes e não ter-se muita utilidade prática, visto que para esse tipo de problema, o GPS é uma solução muito mais confiável em termos de precisão - quinze a cem metros para uso civil, e um metro para uso militar (Lawrence, 2000) - contra uma margem superior a cem metros utilizando LoRa (Fargas, 2015).

Um dos poucos trabalhos com a iniciativa de utilizar LoRaWan para localização indoor, é o trabalho “Localização indoor utilizando a tecnologia LoRaWAN e aprendizado de máquina”, de Oliveira, G. C (2017), onde é feita a proposta de mapear salas do campus de IFSC-SJ. O autor propõe utilizar classificadores de *machinie learning* (ML) para obter uma relação entre a intensidade do sinal dos nós da rede (RSSI), com a sala em que o nó emissor se encontra. Foram aferidos valores RSSI em diversas salas, e foi utilizado um software fechado para utilizar os algoritmos (*software R*), sem um enfoque na implementação destes.

Alguns algoritmos tiveram boa taxa de sucesso em aprender a localizar o nó emissor de sinal da rede dentro das salas treinadas, algoritmos como *support vector machine* (SVM) e *K-Nearest Neighbors* (KNN) obtiveram um sucesso de até noventa e nove por cento em acertar as predições do *dataset* de testes segundo os resultados do autor, também sendo citados em outros trabalhos como alternativa de resolver o problema de geolocalização (Aernouts, 2018).

1.4 OBJETIVOS ESPERADOS

Os principais objetivos desse trabalho são:

a) Implementar uma rede LoRaWan utilizando o servidor de rede The things Network;

b) Analisar como o sinal de modulação LoRa varia com a distância dos nós Emissor e Receptor (Experimento I);
c) Criar um programa em python, rodando em uma máquina local que funcione como um cliente da rede e tenha acesso aos dados dos pacotes trocados pelos nós da rede;
d) Criar outro programa, que permita realizar a coleta os dados do programa mencionado anteriormente e formalização dos mesmos para aplicar algoritmos de aprendizagem de máquina (ML), que realizem a predição do local em que o nó emissor dos pacotes está situado, tendo como entrada a potência do sinal recebida pelos nós receptores (Experimento II);
e) Os programas serão utilizados em dois experimentos distintos: um para tentar predizer a posição em que o nó emissor de pacotes se encontra em relação a um percurso unidimensional segmentado em intervalos, com o objetivo de estudar a precisão em metros que se pode obter em um experimento controlado (sem objetos por perto e com linha de visada entre todos os dispositivos); o outro será feito em um apartamento, onde se tentará ensinar aos algoritmos ML a identificar lugares e salas contidas em um pequeno espaço bidimensional, com objetos diversos nos ambientes (móveis e eletrodomésticos) e sem linha de visada em alguns pontos (Experimento III);
f) Por fim, concluir se é válido o uso de tecnologia LoRa para geolocalização indoor em espaços de dimensões residenciais, quais são suas limitações e possíveis melhorias para trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA E CONCEITOS

Nas próximas seções serão descritos o funcionamento de uma rede LoRa WAN, os métodos utilizados para obtenção da posição de um nó de uma rede *wireless*, e também serão apresentados os algoritmos de predição que serão usados neste trabalho.

2.1 LORAWAN

O LoRaWAN é um protocolo LPWAN sem fio da camada MAC (*Media Access Control*, ou Controle de Acesso ao Meio) do modelo OSI (*Open System Interconnection*), com foco principal em aplicações da IoT que exigem grande área de cobertura e baixo custo

energético. A arquitetura de rede LoRaWAN, é implantada em uma topologia de estrelas em que os gateways transmitem mensagens entre os dispositivos finais e um servidor de rede central (THE THINGS NETWORK, 2019).

A arquitetura da rede é dividida em quatro níveis.

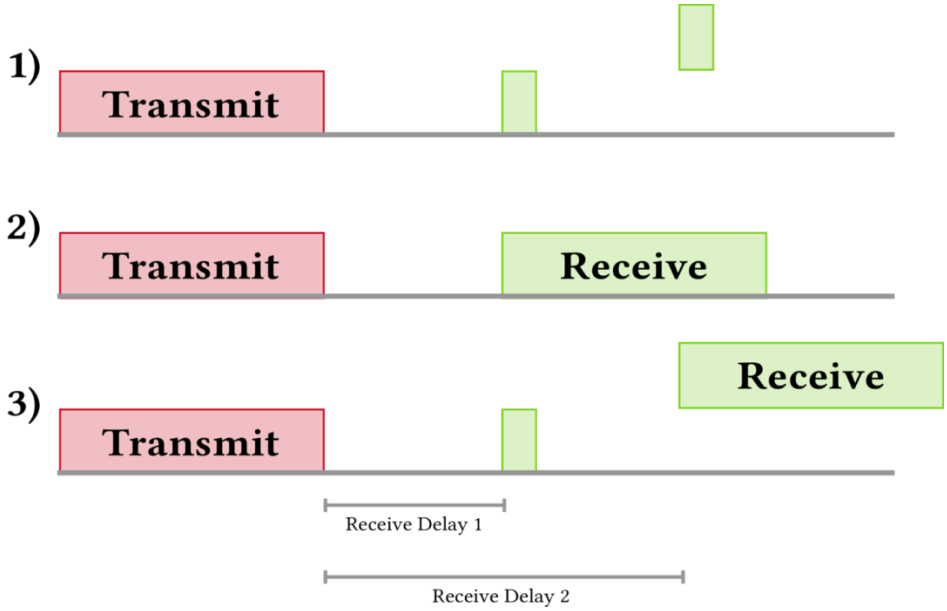
2.1.1 End Devices

São os dispositivos finais da rede, podendo receber (*downlinks*) e enviar (*uplinks*) dados para o servidor de rede por meio dos gateways. Podem ser sensores ou atuadores, com um chip radio *transceiver* LoRa embarcado. Os *End devices* podem ainda ser divididos em três classes:

a) Classe A: Suporta comunicação bidirecional entre o dispositivo e os *gateways*. Podem enviar mensagens de *uplink* a qualquer momento, caso o servidor não responda, a mensagem precisa ser reenviada pelo dispositivo aos gateways (THE THINGS NETWORK, 2019).

b) Classe B: Os dispositivos de classe B estendem a classe A adicionando janelas de recebimento agendadas para mensagens de *downlink* do servidor. Usando balizas sincronizadas no tempo transmitido pelo *gateway*, os dispositivos abrem periodicamente janelas de recepção (THE THINGS NETWORK, 2019).

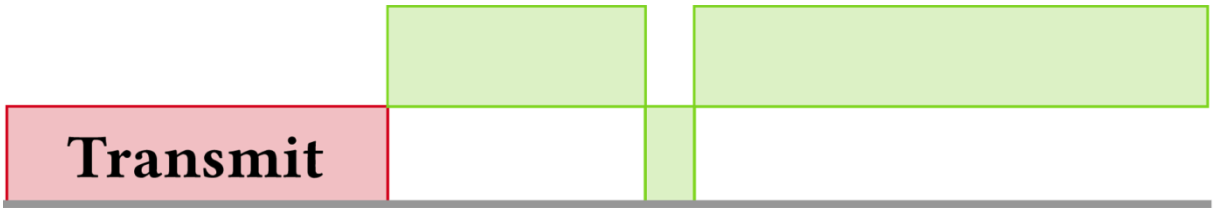
FIGURA 2.01 – Janela de transmissão Classe B.



Fonte: THE THINGS NETWORK (2019)

c) Classe C: Dispositivos da Classe C estendem a Classe A mantendo as janelas de recepção abertas a menos que estejam transmitindo. Isso permite uma comunicação de baixa latência, mas consome muito mais energia do que os dispositivos Classe A (THE THINGS NETWORK, 2019).

FIGURA 2.02 – Janela de transmissão Classe C.



Fonte: THE THINGS NETWORK (2019)

2.1.2 Gateways(GW)

Os *gateways* são dispositivos conectados ao servidor de rede por meio de conexões IP padrão e atuam como uma ponte, simplesmente convertendo pacotes RF em pacotes IP e vice-versa.

Todos os gateways ao alcance de um dispositivo receberão as mensagens do dispositivo e as encaminharão para o servidor de rede, convertendo pacotes RF em pacotes IP

e vice-versa. Este processo é denominado *packet forwarder*, que é responsável por gerenciar, verificar e redirecionar os pacotes LoRa demodulados para o *network server*, utilizando algum protocolo de rede como UDP ou MQTT previamente definido em sua configuração.

É importante salientar que o servidor de aplicação e os *end nodes* não precisam conhecer o *gateway* utilizado para o procedimento, podendo utilizar qualquer um que estiver disponível em seu alcance e cadastrado no servidor de rede, mostrando uma independência entre os *gateways* e a camada superior e inferior da rede (THE THINGS NETWORK, 2019).

2.1.3 Network Server(NS)

Também denominado sistemas *backend* não é um Hardware Lora propriamente dito, podendo ser um computador ou servidor local, é responsável pela maior parte da lógica do controle e segurança das mensagens recebidas e enviadas pelos *gateways* e os nós finais da rede. Requer apenas que esteja conectado à mesma rede dos *gateways*.

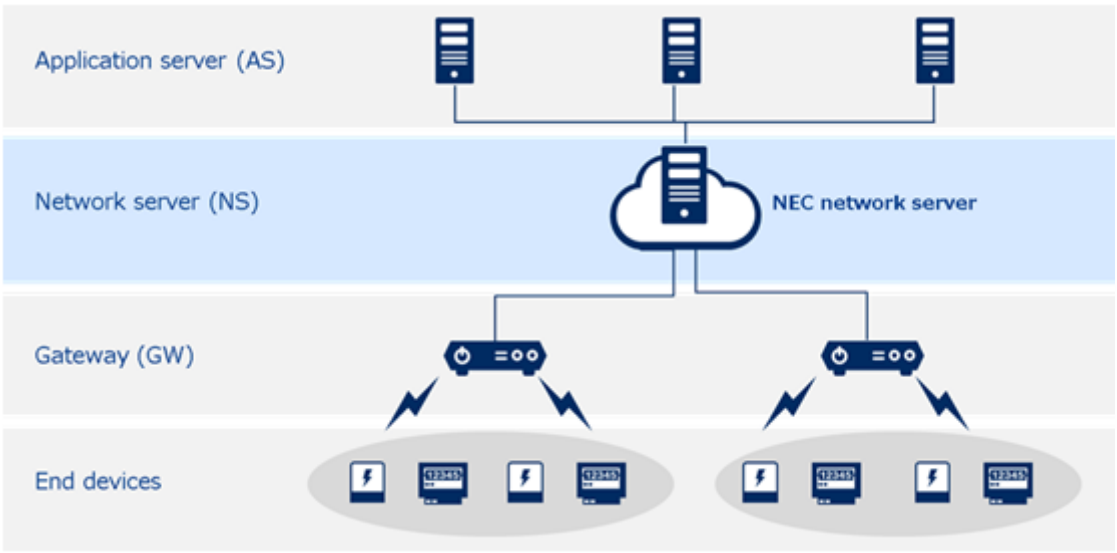
É composto por um *software* que faz o roteamento e o controle de fluxo da rede. A comunidade The Things Network, oferece um serviço de servidor de rede, em que é possível cadastrar gateways e aplicações para o usuário (THE THINGS NETWORK, 2019).

2.1.4 Application Server (AS)

É a camada de aplicação da rede, basicamente um software compatível com a camada do servidor da rede que descriptografa os pacotes transformando-os em informação legível para o usuário ou prepara pacotes para serem enviados até os nós terminais.

Permite que clientes da rede realizem o acesso aos dados trafegados (THE THINGS NETWORK, 2019).

Figura 2.03 – Ilustração da arquitetura da rede LoRaWAN com seus dispositivos



Fonte: (the things network, 2019)

2.2 MODULAÇÃO LORA

É importante distinguir a diferença entre LoRa e LoRaWAN. Como citado no item 2.1.2, LoraWan é um protocolo de acesso ao meio (MAC) que controla o tráfego entres nós de uma rede com dispositivos LoRa e fornece acesso a aplicações pela internet. Por outro lado, o LoRa é uma modulação para meio sem fio proprietária da *Semtech Corporation* projetada para operar com longo alcance e baixo consumo de potência, que utiliza a tecnologia de espalhamento espectral *Chirp Spread Spectrum* (CCS) para a codificação e transmissão de pacotes (THE THINGS NETWORK, 2019).

2.3 SISTEMAS DE POSIÇÃO INDOOR (IPS)

Como já citado na seção 1.1 deste trabalho, a necessidade da existência de sistemas de localização e posicionamento indoor ou IPS, se faz necessária visto que sistemas de localização por satélite como GPS ou *Glonass*, precisam oferecer linha de visada entre os nós para poder calcular a posição. Logo, existem inúmeras tecnologias sem fio em que se pode implementar sistemas para estimar a posição ou a proximidade de objetos. Tecnologias como

wifi, Bluetooth Low Energy (BlE), Zigbee, são viáveis para implementar um IPS, cada uma com suas particularidades como melhor desempenho energético ou alcance (Haute,2016).

Em qualquer sistema IPS existem dois tipos de nós:

a) Nós móveis: são os dispositivos móveis embarcados a agentes que se deseja localizar no espaço (pessoas, máquinas, animais e etc.);

b) nós âncoras: são os nós fixos da rede, que servem como referência para calcular a posição dos Nós móveis. Fazendo uma analogia com o GPS, os nós âncoras seriam os satélites.

Alguns parâmetros do sinal entre a comunicação dos nós móveis e âncoras variam com a distância entre eles; partindo desta premissa, é possível utilizar estes dados para tentar fazer a predição da distância ou o local em que os nós móveis se encontram. A seguir serão apresentados estes parâmetros e as equações que descrevem suas variações com a distância em uma situação ideal, em que o sinal entre nós âncoras e móveis é perfeitamente inteligível e livre de ruído por reflexão e sombreamento que é intrínseco ao sinal em locais fechados (Haute,2016).

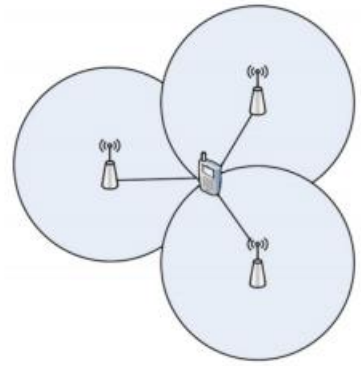
2.4 PROPRIEDADES DO SINAL E TÉCNICAS UTILIZADAS

As técnicas baseadas no sinal Wireless mais comumente empregadas são *Time Of Arrival* (TOA), *Angle Of Arrival* (AOA) e *Receive Signal Strength Indication* (RSSI), que utilizam, respectivamente, o tempo de chegada do sinal a um nó âncora, o ângulo de chegada do sinal em relação aos nós âncora, e a intensidade do sinal, para calcular a posição de um objeto.

2.4.1. Time Of Arrival (ToA)

A técnica baseada no Tempo de Chegada (*Time Of Arrival-ToA*) estima a distância através do tempo que o sinal leva para sair de um sensor móvel e chegar aos nós âncora. Isso exige que ambos os lados possuam um relógio sincronizado, para que o receptor saiba calcular, junto com a velocidade de propagação já conhecida, a distância do transmissor. Esta sincronização pode ser feita pelos recursos do sistema ou por aparelhos como GPS. Apesar da potência do sinal variar a diferentes distâncias, no método ToA não há degradação, já que se baseia no tempo (Oliveira, 2017).

Figura 2.04 – Esquema de triangulação usando TOA



Fonte: (Carvalho, 2016)

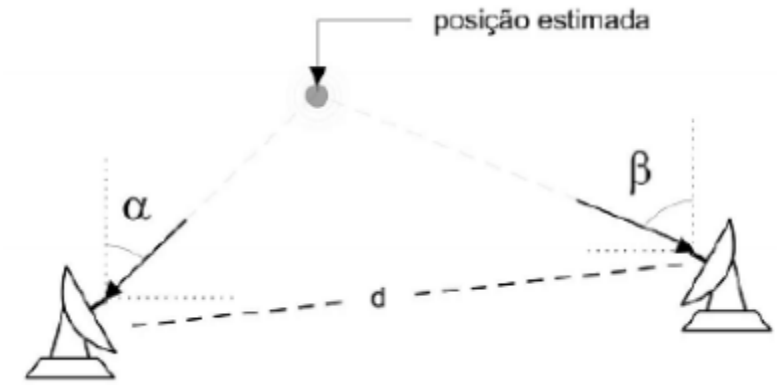
Uma possível forma de estimar as medidas é com triangulação dos nós (FIGURA), semelhante à solução para GPS.

O problema principal para um sistema baseado em tempo é a sincronização dos relógios dos dispositivos, consequentemente o mesmo acontece com sistemas baseados com TOA, especialmente considerando a alta velocidade com que o sinal se propaga. Pelo fato dos sistemas serem sensíveis ao tempo, há também a possibilidade de atrasos nas leituras realizadas pelo hardware, pois muito se gasta para sincronização dos relógios, e assim devem ser contabilizados para o cálculo das distâncias. (Carvalho, 2016).

2.4.2 Angle of Arrival

No método AOA utiliza-se o ângulo do sinal ao chegar às antenas e este processo exige a instalação de receptores e antenas especiais nos nós âncora. O objeto a ser localizado emite sinais e são necessários ao menos dois receptores, como mostra a Figura 2, para que seja possível saber a fonte do sinal, ou seja, a posição de onde o sinal foi emitido.

Figura 2.05 – Cálculo da posição através do ângulo do sinal recebido



Fonte: (Carvalho, 2016).
A posição é estimada conforme o ângulo de defasagem do sinal recebido em cada nó âncora.

2.4.3 RSSI

A intensidade de um sinal propagado no meio sem fio sofre interferência de vários fatores, incluindo a potência do transmissor, a sensibilidade do receptor, a atenuação do meio, a perda de caminho (*path loss*) e a influência de outros sinais presentes no meio. É medida em decibéis (*dBm*) numa escala negativa, sendo assim, quanto maior o módulo em *dBm*, menor é a intensidade do sinal. O valor do RSSI representa qual a força do sinal transmitido por um determinado ponto de acesso (AP), ou seja, essa técnica de medição se baseia no fato de que o sinal perde força à medida que se propaga. Quanto mais próximo do ponto de acesso, maior será a intensidade do sinal e menos negativo será o valor em decibéis. Este valor é uma indicação relativa de medição, por isso as leituras podem variar dependendo do dispositivo utilizado. (Oliveira, 2017).

Idealmente, o valor de RSSI apresenta variação correspondente com *P(d)* na Equação (2.1), referente à perda de percurso (Oliveira, 2017).

$$P(d) = Po - 10.n.\log_{10}(d/do) \quad (2.1)$$

Onde *n* é o fator de perda de percurso, *P(d)* é a potência recebida na distância *d*, e *Po* é a potência recebida em uma distância de referência *do*.

Existem ainda dois fatores que afetam na potência de sinal recebida: multipercurso e sombreamento. Multipercurso é o efeito causado por diversas componentes de um sinal que chegam até o receptor por trajetos diferentes, podendo causar interferência construtiva ou

destrutiva. Já o efeito de sombreamento se refere à atenuação do sinal devido a obstáculos presentes no percurso (Oliveira, 2017).

Conforme o artigo de Regilane L (2016), por conta destes fatores, o mapeamento de ambientes indoor pode ser bem caótico, conforme é explicitado nos resultados de simulações exibidos na Figura 2.06. Nas simulações foram considerados os efeitos de sombreamento que as paredes do ambiente simulado podem influenciar nas medidas.

Figura 2.06 – Mapeamento RSSi estimado para quatro APs (simulação)

Fonte: (Regilane L., 2016).
Valores em dBm.

A Técnica de medir a distância entre dois nós da rede usando o parâmetro RSSI do sinal não necessita de sincronia entre os dispositivos (ToA) e nem do uso de antenas adicionais (AoA), tornando a implementação mais simples e barata. Por estas conveniências, foi a técnica escolhida.

2.5 ALGORÍTMOS DE APRENDIZAGEM DE MÁQUINA

Nas seções seguintes, serão apresentados os algoritmos de ML utilizados neste trabalho.

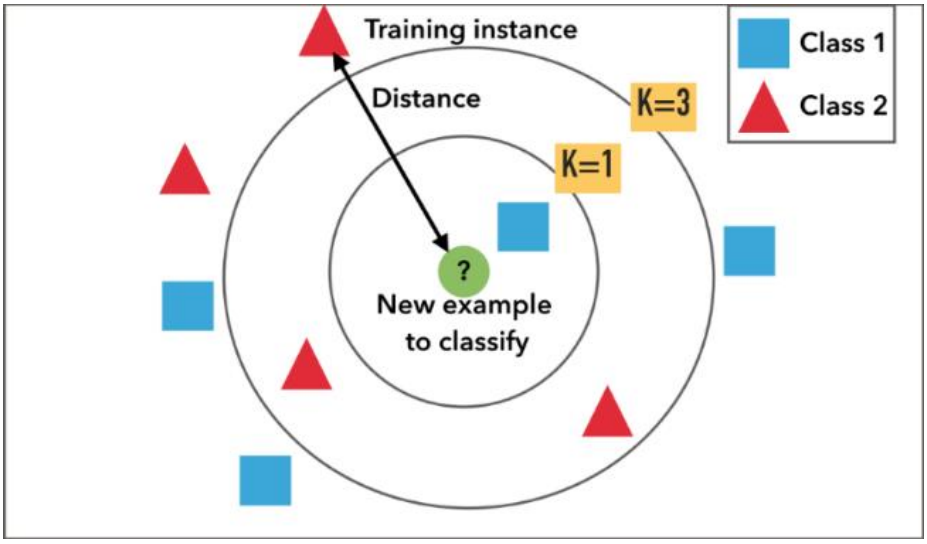
2.5.1 K–Nearest Neighbor

O algoritmo *K- Nearest Neighbor* (KNN), em português “k- vizinhos mais próximos”, é um algoritmo de aprendizagem de máquina supervisionado. Portanto um modelo classificador que tem como entrada um conjunto de dados de treinamento e certo resultado alvo na saída (*target*), que deve ser validado com outra parte dos dados usada para teste. O

classificador não tenta construir um modelo interno geral, mas simplesmente armazena instâncias dos dados de treinamento. A classificação é calculada a partir de uma votação por maioria simples dos vizinhos mais próximos de cada ponto: um ponto de consulta é atribuído à classe de dados que possui mais representantes dentro dos vizinhos mais próximos do ponto. (SCIKIT-LEARN DEVELOPERS, 2019).

A idéia é que o classificador consiga determinar a classe de uma amostra M em um espaço amostral *Rn*, de acordo com a distância entre seus *k* vizinhos mais próximos, tal que *k >= 1*. Ou seja, a amostra *M* vai ser classificada de acordo com a ocorrência das classes dos pontos pertencentes a sua vizinhança.

Figura 2.07 – Representação gráfica do KNN



Fonte: (Santos, 2009).
Imagem representando algoritmo KNN, se K = 1, a nova amostra é classificada como classe 1 (quadrados), se K = 3, é classificada como classe 2 (triângulos).

Portanto, em cada predição, o algoritmo deve estimar a distância entre os *k* pontos mais próximos da amostra *M* e sua classe será a moda das classes dos *k* pontos.

Para um ponto *A(x,y)* e um ponto *B(x,y)*, pode-se definir a distância entre os pontos pela equação euclidiana:

$$D = \sqrt{(Ax - Bx)^2 + (Ay - By)^2} \quad (2.2)$$

Para a distancia *D* de dois pontos num espaço *Rn*:

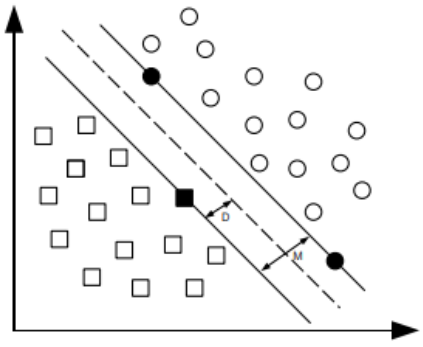
$$D = \sqrt{\sum_{i=1}^{n}(An - Bn)^2} \tag{2.3}$$

O número de *k* define a complexidade computacional do algoritmo, visto que, com muitos vizinhos, para cada predição será necessário fazer a computação destas distâncias, esta é a sua principal desvantagem. O numero *k* a ser escolhido depende da dispersão do conjunto de amostras medidas, ao se escolher um *k* muito pequeno, pode-se estar alimentando o algoritmo apenas com ruído amostral, em contraponto, um valor muito elevado para *k* pode gerar predições mais imprecisas. (SCIKIT-LEARN DEVELOPERS, 2019).

2.5.2 Suport Vector Machinie (SVM)

O método SVM foi inicialmente definido para a classificação de padrões linearmente separáveis. Em outras palavras, padrões cujas categorias possam ser separadas por um hiperplano. Dado o conjunto de treino $Tr = \{di, c\,j\}\,|Tr|\; i = 1 \subset D$, onde c $j \in \{círculo, quadrado\}$. O objetivo do método SVM é construir um hiperplano como superfície de decisão, que separe as categorias distintas do conjunto Tr com a maior margem de separação possível. A Figura 2.07 ilustra o exemplo desse hiperplano, representado por uma linha tracejada (Santos, 2009).

Figura: 2.08: Representação gráfica do SVM



Fonte: (Santos, 2009).

Sendo x um ponto arbitrário que representa um vetor de amostras a ser classificado, o vetor w define a direção do hiperplano perpendicular ao ponto x e o termo b possibilita deslocar o hiperplano separador do conjunto Tr conforme a Equação (2.4) (Santos, 2009).

$$(w \cdot x) + b = 0 \tag{2.4}$$

Para determinar a classe de *x*, é necessário calcular sua posição relativa ao hiperplano da equação (2.5), pode-se usar a relação (Santos, 2009).

$$y = \begin{cases} +1\,(círculo) se\,(w \cdot x) + b \geq 0 \\ -1\,(quadrado)\,se\,(w \cdot x) + b < 0 \end{cases} \tag{2.5}$$

Denominam-se vetores suporte, os elementos do conjunto *Tr* (pontos escuros da FIGURA 2.08) que formam a maior margem de separação entre duas classes diferentes; portanto, são os pontos mais próximos do hiperplano separador, pode ser calculada pela equação (2.6) (Santos, 2009).

$$M = \frac{2}{||w||} \tag{2.6}$$

Calculando a margem de separação, é possível saber a distância entre o vetor suporte *x* e o hiperplano *D* conforme a equação (2.7) (Santos, 2009).

$$D = \frac{M}{2} = \frac{|(w.x)+b|}{||w||} = \frac{1}{w} \tag{2.7}$$

Em muitos casos, os padrões não são linearmente separáveis. Nesses casos, os padrões de entradas (espaço de entrada) são transformados em um vetor de características com alta dimensionalidade, cujo objetivo é separar linearmente as características no espaço através do uso de funções não lineares especiais chamadas de *kernel*. O *kernel* possibilita a construção de

um hiperplano de separação ótimo no espaço de características sem considerar explicitamente o próprio espaço de características (Santos, 2009).

3 DESENVOLVIMENTO E RESULTADOS

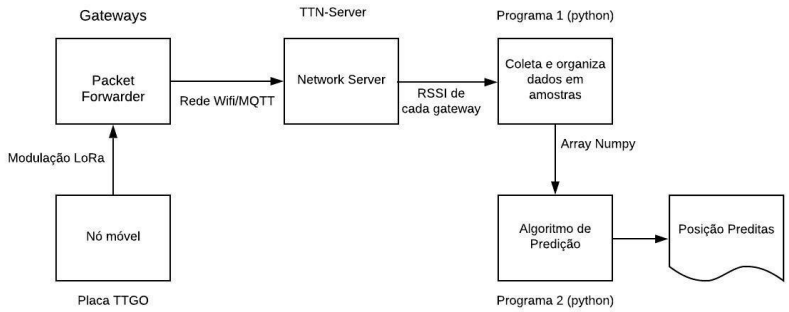
O desenvolvimento do trabalho foi feito em três experimentos. Primeiramente foi feita uma investigação, de como o parâmetro RSSI do sinal varia com a distância entre o nó móvel e os nós âncoras. Após a investigação, foi feito um segundo experimento simplificado (percurso unidimensional), a fim de verificar se era possível implementar um modelo para encontrar a posição do nó móvel em relação aos nós âncoras (*gateways)*. Por fim, um último experimento foi feito empregando os mesmos algoritmos para ML do experimento anterior, porém, em um ambiente mais próximo da realidade (espaço bidimensional composto por salas), verificando a viabilidade do uso da tecnologia LoRa para um IPS utilizando os métodos propostos.

Nas seções a seguir serão descritos como foi implementada a rede LoRaWAN, bem como os experimentos aqui citados.

3.1 HARDWARE E SOFTWARES UTILIZADOS

Na Figura 3.01 é ilustrado o fluxo de dados do sistema, iniciando no nó móvel (Placa TTGO), até a aplicação, gerando um arquivo separado por vírgulas (CSV) com as amostras preditas pelos algoritmos.

Figura 3.01 – Fluxo dos dados no sistema desenvolvido para os experimentos

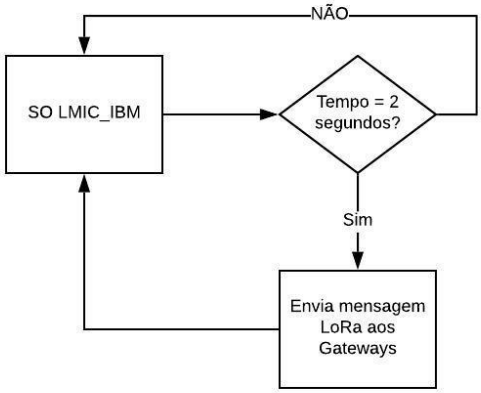


Fonte: Autor.

3.1.1 Nó Móvel

Para o nó móvel, utilizou-se a placa de desenvolvimento ESP32 TTGO (FIGURA 3.03). A placa se trata de uma versão genérica, sem fabricante especificado, utiliza o microcontrolador ESP32 (ESP32, 2019) com um radio LoRa embarcado (LORA ALLIANCE, 2019). O software foi programado utilizando a Arduino IDE (ARDUINO, 2019) e o sistema operacional LMIC da IBM (Matthijskooijman, 2017), quando executado, envia um pacote LoRa aos gateways com a periodicidade de dois segundos.

Figura 3.02 - Diagrama UML do programa no nó móvel



Fonte: Autor.

Figura 3.03 – Foto da placa TTGO utilizada como nó móvel

Fonte: Foto tirada pelo autor

3.1.2 Gateways

Como não se tinha disponível um gateway LoRa comercial, utilizou-se placas *Raspberry PI* (FIGURA 3.04) com radio LoRa embarcado. Com um programa em *C++* clonado do github (Telkamp, 2017), e acesso uma rede *wifi* foi possível implementar um *gateway* LoRa para realizar processos de *packet forwarder*. As placas foram obtidas sob empréstimo do professor orientador deste trabalho.

No artigo de Oliveira, L. R.(2018), estão presentes instruções de como programar um *gateway* LoRa mono canal utilizando placas *Raspberry Pi* e cadastrar no servidor TTN.
As placas já continham o sistema operacional *Raspbian* (RARPBIAN, 2019) baseado em *Debian*, bastando apenas clonar o *github* (Telkamp, 2017), e modificar o trecho do código referente à freqüência 915MHz a 928MHz (Plano da Austrália) que é a banda utilizada no Brasil para dispositivos LoRa.

Figura 3.04 – Foto da placa Raspberry PI utilizada como gateway

Fonte: Foto tirada pelo autor

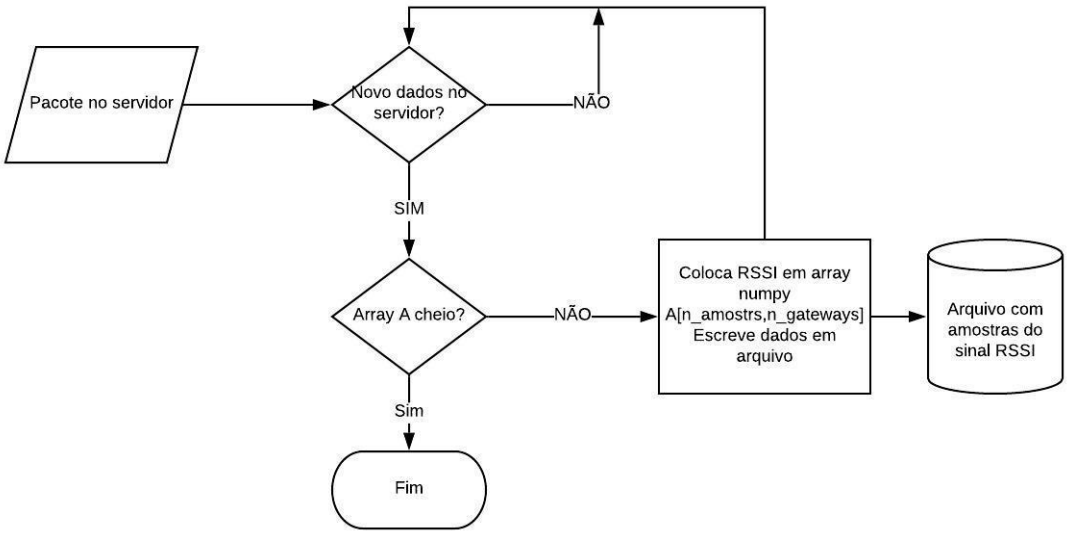
3.1.3 Aplicação

Para a aplicação, foram criados dois programas em *python*. O primeiro acessa o servidor de rede por protocolo MQTT utilizando a biblioteca *paho mqtt* (PAHO MQTT, 2018); coleta os pacotes recebidos em cada um dos *gateways* e guarda os valores do

parâmetro RSSI em um diretório de arquivos. O segundo programa carrega os arquivos e organiza os dados em um *array* utilizando a biblioteca *Numpy* (NUMPY DEVELOPERS, 2019); em seguida, executa os algoritmos de aprendizagem de máquina e exibe os resultados impressos no console (*Pyton Shell*).

O primeiro programa, simplesmente coleta os dados e organiza um *dataset* em arquivos separado em diretórios, para cada local em que as medidas foram feitas.

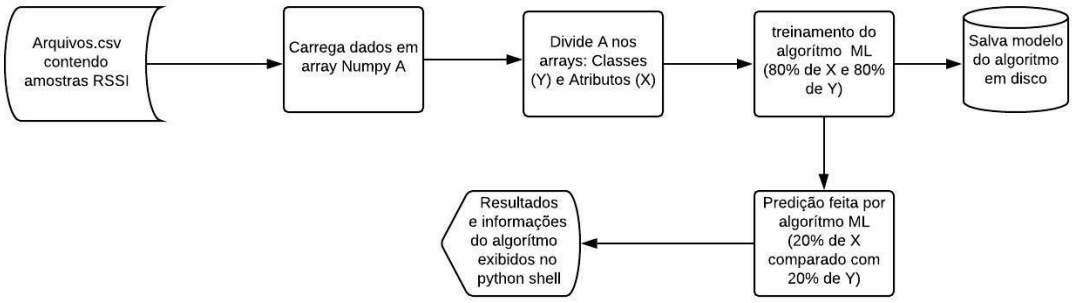
Figura 3.05 – Diagrama UML do Primeiro programa



Fonte: Autor.

No segundo programa, foram implementados os algoritmos de ML utilizando a biblioteca *SciKit-learn (*SCIKIT-LEARN DEVELOPERS, 2019). Os dados contidos nos arquivos são formatados em um *array numpy*, sendo que 75% dos dados serão utilizados para o treinamento do algoritmo e 25% para a validação. O modelo treinado é salvo com a biblioteca *pickle* (PICKLE, 2019), permitindo o seu uso posterior, não sendo necessário treinar o algoritmo toda a vez que executar o programa.

Figura 3.06 – Diagrama UML do Segundo programa



Fonte: Autor.

Tabela 3.01 – Funções utilizadas para treinar modelo ML

Função	Descrição
Np.randon.shuffle(A)	Embaralha as linhas do array 'A'
Split(A,x_train,y_train,x_test,y_dest)	Separa as features e classes em conjuntos de treinamento e teste: 'x_train' = 80% das *features* de 'A' 'y_train' = 80% das classes de 'A' 'x_test' = 20% das *features* de 'A' 'y_test' = 20% das classes de 'A'
Clf =svm.SVC(kernel = 'linear')	Cria um classificador 'Clf' SVM com *kernel* linear
Clf =KNeighborsClassifier(k)	Cria um classificador 'Clf' KNN com k vizinhos
Clf.fit(x_train,y_train)	Ensina o algoritmo a fazer a predição com arrays separados para treino (80% das amostras)
Y_pred = Clf.predict(x_test)	Modelo 'Clf' criado faz a predição do *array* de testes 'x_test' (20% das amostras) e armazena resultados em *array* ' y_pred'
Metrics.classification(y_test,y_pred)	Calcula as taxas de porcentagem de acertos do algoritmo, relacionando as classes preditas dos dados de teste, com os suas classes verdadeiras (score).
Metrics.confusion_matrix(y_test, y_pred)	Calcula a matriz de confusão entre as classes verdadeiras e as classes preditas

Fonte: Autor.

A verificação da validade do modelo foi feita utilizando a matriz de confusão e *scores* fornecidos por funções da própria biblioteca *SciKit-learn*, que simplesmente compara as

saídas preditas pelo algoritmo com as saídas esperadas do conjunto de dados de teste. Mais detalhes serão abordados na seção 3.4.3.

Onde A é o *array* de dados carregado dos arquivos CSV, com *M* linhas e *N* colunas. As *features* (medidas RSSI de cada *gateway*) compõem as colunas *0* até *N-1*, restando a última coluna para as classes correspondente ao local onde foram feitas as medidas. *M* corresponde ao número de amostras coletadas.

Posteriormente juntaram-se os dois programas citados acima em um único, em que foi adicionada uma função para plotar os resultados, utilizada no Experimento III – Primeira Etapa.

A partir da aquisição dos dados, foram feitos três experimentos para avaliar a viabilidade de um sistema IPS utilizando tecnologia LoRa em espaços pequenos.

a) Experimento I: Investigação do comportamento do parâmetro RSSI dos gateways em relação à distância do nó móvel;

b) Experimento II: Implementação de dois algoritmos ML em um trajeto unidimensional;

c) Experimento III: Implementação de dois algoritmos ML em um espaço de duas dimensões.

3.2 METODOLOGIA DO EXPERIMENTO I

O objetivo do primeiro experimento trata-se de uma investigação com o intuito de verificar a relação que existe entre a potencia do sinal recebido pelos nós âncoras da rede com a variação da posição do nó móvel, verificando se é possível a construção de um modelo *f* que forneça a posição do emissor *Pe*, dado o parâmetro RSSI do sinal em cada receptor *G*.

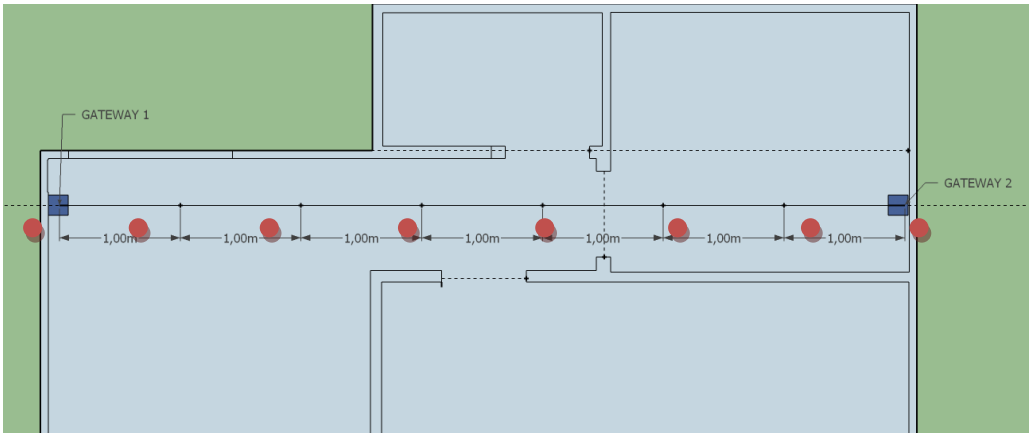
Para o caso de *n gateways* na rede:

$$Pe = f(Grssi_1, Grssi_2 \dots Grssi_n) \tag{3.1}$$

Para verificar o quanto o meio pode interferir nas medidas, serão feitas medidas em dois ambientes diferentes, em um corredor fechado com distância de sete metros e em um salão aberto sobre um trajeto de dez metros.

3.2.1 Primeira Etapa

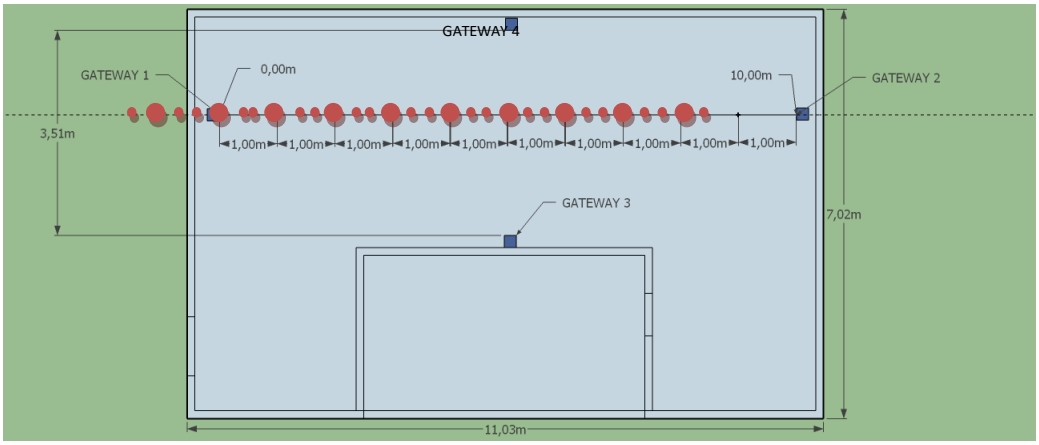
Figura 3.07 – Planta baixa do local onde foi feita a primeira etapa do Experimento I



Fonte: Autor.
Diagrama do experimento I, primeira etapa.
Distância dos nós âncora: 7 metros.
Em vermelho: Medidas coletadas com o nó móvel.
Em azul: Nó âncora (gateways 1 e gateway 2).
Local das medidas: limites dos intervalos.
Dataset gerado: 55 amostras de cada gateway em cada ponto de medida.

3.2.2 Segunda Etapa

Figura 3.08 - Planta baixa do local onde foi feita a segunda etapa do Experimento I



Fonte: Autor.
Diagrama do experimento I, segunda etapa.
Distância dos nós âncora: 10 metros (10 intervalos de 1 metro)
Em vermelho: Medidas coletadas com o nó móvel
Em azul: Nó âncora (somente gateway 1 e gateway 2 utilizados)
Local das medidas: no meio e nos limites dos intervalos
Dataset gerado: 55 amostras de cada gateway em cada ponto de medida.

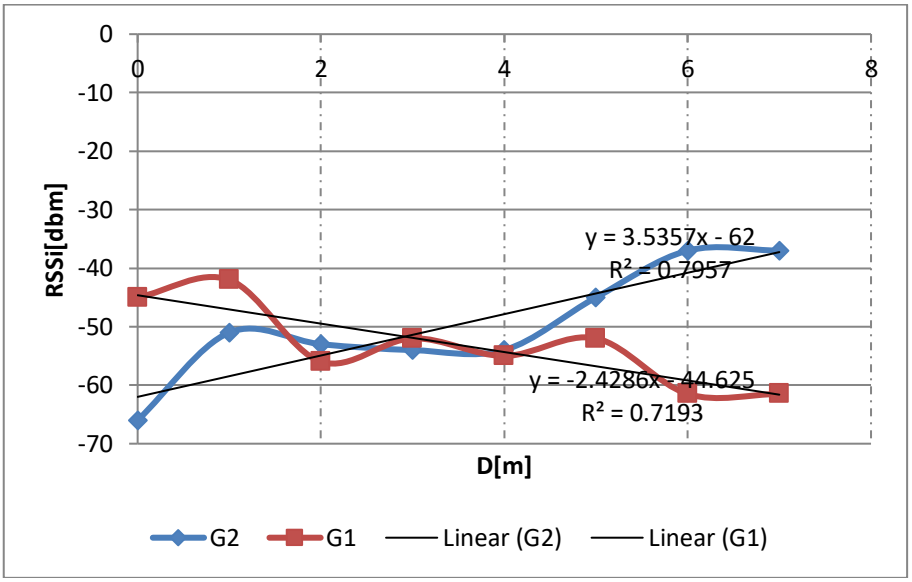
Note que no diagrama da FIGURA 3.5 estão presentes quatro gateways como nós âncoras. Os dados dos Gateways três e quatro não serão utilizados neste experimento, sendo úteis e justificados posteriormente na seção 3.4.

3.3 RESULTADOS DO EXPERIMENTO I

Nas seções seguintes serão mostrados os resultados do Experimento I graficamente, todos os dados medidos (datasets) estão contidos em arquivos TXT no *github* do autor (Silva, T. N. B, 2019).

3.3 .1 Primeira Etapa

Gráfico 3.01 – Dados da primeira etapa do Experimento I



Fonte: Autor.
Gráfico gerado no Excel com as medias dos valores RSSI nos limites dos intervalos.

Com o auxílio do software Excel, as curvas G1 e G2, representam a relação entre o parâmetro RSSI e a distância entre os *gateways* um e dois (em azul na Figura 3.07), e o nó móvel.

As curvas foram linearizadas utilizando o software Excel, resultando nas equações de reta Linear (G1) e Linear (G2) conforme as equações (3.2) e (3.3) respectivamente.

$$y = -2{,}428x - 44{,}62 \quad (3.2)$$

$$y = 3{,}535x - 62 \quad (3.3)$$

O Excel fornece o coeficiente de determinação da linearização *(R^2)* (Gráfico 3.01), que corresponde ao quanto o modelo consegue justificar a relação entre as amostras RSSI [*dbm*] com a variável independente *d[m]*. O valor é dado em porcentagem, quanto menor, menos o modelo linearizado descreve a situação real.

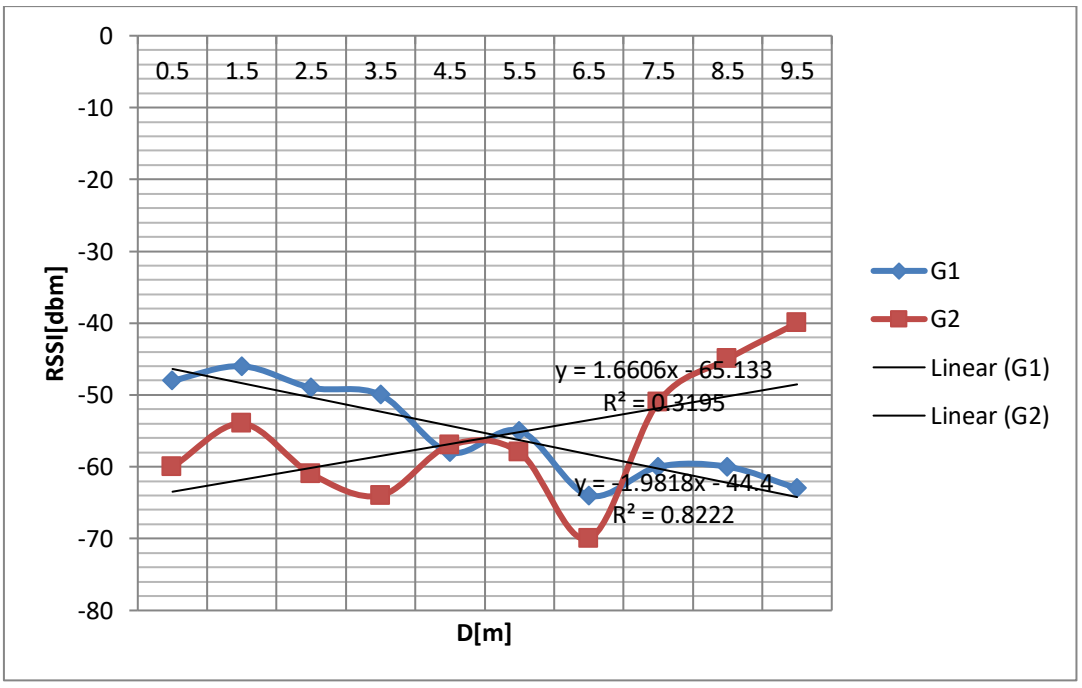
Tabela 3.02 – Coeficiente de determinação da linearização

Curva	R^2
G1	71,9%
G2	79,5%

Fonte: Autor.
Referente aos dados do Gráfico 3.01.

3.3.2 Segunda Etapa

Gráfico 3.02 - Dados da segunda etapa do Experimento I



Fonte: Autor.
Gráfico gerado no Excel com as médias dos valores RSSI medidos nos meios dos intervalos

Análogo à primeira etapa do experimento, ambas as curvas foram linearizadas resultando em *Linear (G1)* e *Linear (G2)* conforme as equações 3.4 e 3.5.

$$y = -1{,}981x - 44{,}4 \tag{3.4}$$

$$y = 1{,}660x - 65{,}13 \tag{3.5}$$

Tabela 3.03 – Coeficiente de determinação da linearização

Curva	R^2
G1	82,2%
G2	31,9%

Fonte: Autor.
Referente aos dados do Gráfico 3.02.

Note que é visível o decaimento com tendência linear no sinal com o aumento da distância entre o nó móvel e os nós âncoras.

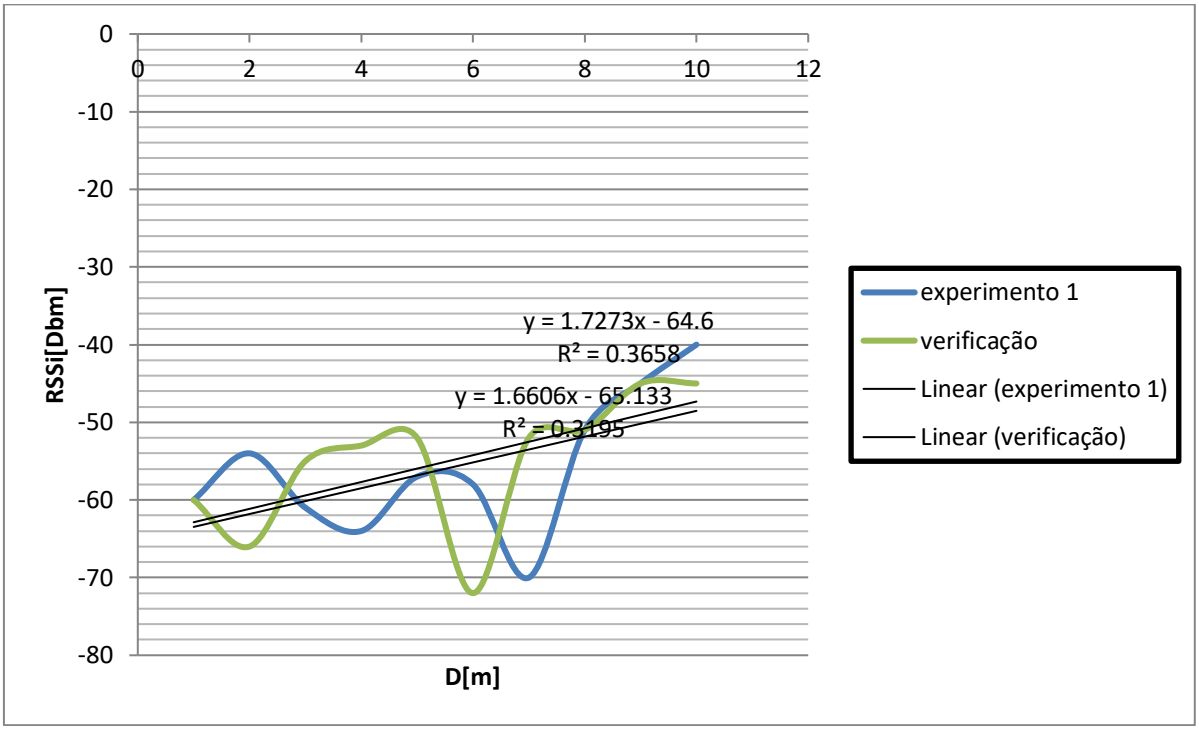
São observadas, duas discrepâncias entre os dois resultados; nos coeficientes angulares das equações (G2-1) e (G2-2) e uma atenuação da curva do sinal do gateway 2, no intervalo entre 6 e 7 metros do percurso da segunda etapa, inexistente no ambiente testado na primeira etapa.

Provavelmente por este motivo, o coeficiente de determinação da curva linearizada resultou em apenas 31%, ou seja, menos de um terço das amostras estão relacionadas ao modelo.

Em função das anomalias percebidas, foi repetida a segunda etapa do experimento, na tentativa de verificar se a razão das diferenças no resultado das duas etapas foi devido a algum erro sistemático na aferição das medidas ou à mudança do meio.

3.3.3 Verificação dos resultados

Gráfico 3.03 – Dados da verificação do Experimento I



Fonte: Autor.
Gráfico gerado no Excel com as medias dos valores RSSI medidos nos intervalos

$$y = 1{,}660x - 65{,}13 \quad (3.6)$$

$$y = 1{,}727x - 64{,}6 \quad (3.7)$$

Tabela 3.04 – Coeficiente de determinação da linearização

Curva	R^2
Experimento 1	31,9%
Verificação	36,5%

Fonte: Autor.
Referente aos dados do Gráfico 3.03.

É notória a semelhança entre as duas curvas na seção 3.3.3 (*experimento 1* e *verificação*), sugerindo que a mudança do ambiente em que foram feitas as medidas em 3.3.1 e 3.3.2 tenha sido responsável por parte das distorções. Logo, para cada local diferente que se

tente implementar o IPS, o modelo terá de ser diferente, o que dificulta utilizar abordagens lineares e modelos ideais como vistos na seção 2.

3.4 METODOLOGIA DO EXPERIMENTO II

Um possível caminho para contornar o problema encontrado no Experimento I, pode ser utilizar métodos de aprendizagem de máquina, que adaptem um modelo específico para cada ambiente.

Em vez de obter uma saída no modelo, com o valor contínuo da distância entre o nó móvel e os gateways, um algoritmo pode aprender com as medidas aferidas nos intervalos conhecidos e gerar um modelo preditivo do intervalo que o nó móvel se encontra tendo como entrada os valores RSSI medidos para cada gateway, sendo a saída, portanto, discreta.

Figura 3.09 – Diagrama da predição

Fonte: Autor
A entrada do modelo é composta pelos valores RSSI referentes a cada gateway, e a saída corresponde ao intervalo predito.

O segundo experimento é dividido em duas etapas:

a) Primeira Etapa: Predição dos intervalos com dois nós âncora;

b) Segunda Etapa: Predição dos intervalos com quatro nós âncora.

O ambiente do experimento está referido no diagrama da FIGURA 3.08, para melhor compreensão, cada intervalo foi rotulado conforme a TABELA 3.05.

Tabela 3.05 – Rótulo dos intervalos (classes)

Limite superior do intervalo [m]	1	2	3	4	5	6	7	8	9	10
Rótulo do intervalo	A	B	C	D	E	F	G	H	I	J

Fonte: Autor.

3.4.1 Análise da eficiência dos algoritmos

Para analisar a eficiência da aprendizagem do algoritmo, utiliza-se um componente denominado matriz de confusão. Este, relaciona as classes verdadeiras e as classes preditas do conjunto em uma matriz, de forma que é possível saber quais as predições são verdadeiras e quais foram erros de predição do algoritmo.

A predição ideal, com 100% de acertos, deve conter itens apenas na diagonal da matriz, como ilustra o exemplo da Tabela 3.06 contendo quatro classes (A, B, C, D).

Tabela 3.06 – Matriz de confusão

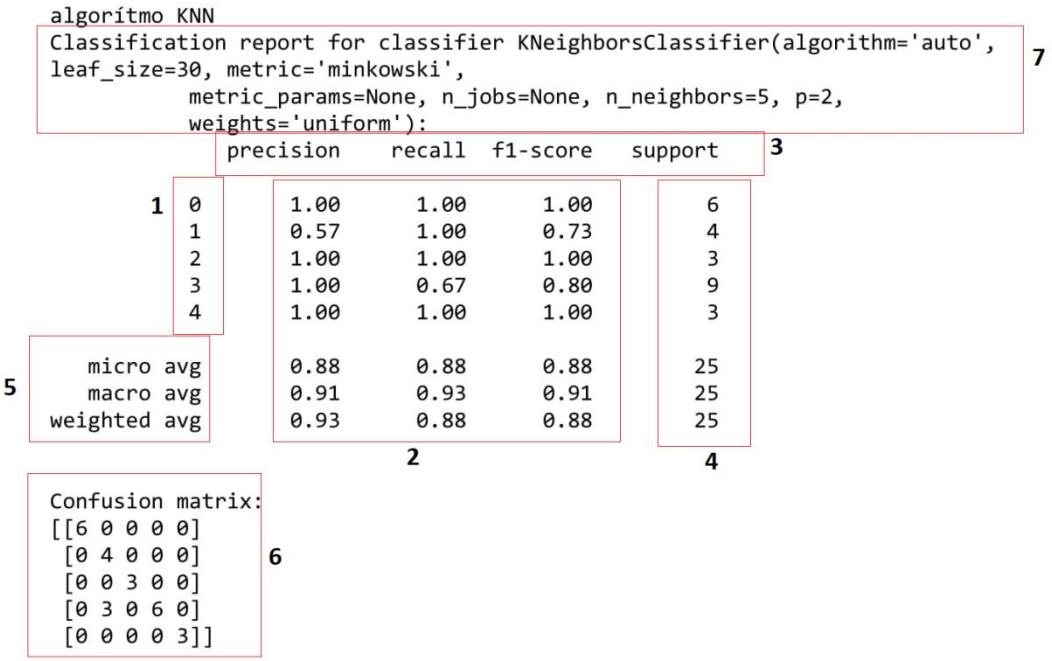
Classes	A predição	B predição	C predição	D predição
A verdadeiro	Predição Correta	Predição Falsa	Predição Falsa	Predição Falsa
B verdadeiro	Predição Falsa	Predição Correta	Predição Falsa	Predição Falsa
C verdadeiro	Predição Falsa	Predição Falsa	Predição Correta	Predição Falsa
D verdadeiro	Predição Falsa	Predição Falsa	Predição Falsa	Predição Correta

Fonte: Autor.

Como mencionado na Tabela 3.01, a funções *sklearn.metrics.classification_report()* e *sklearn.metrics. confusion_matrix (),* fornecem as porcentagens de erros e acertos da predição (*score*) e a relação entre os resultados certos e errados com a matriz de confusão.

Na Figura 3.10, é ilustrado como serão entendidos os resultados do *score* do modelo. Em todos os parâmetros se utilizou SVM com *kernel* linear e KNN com $k = 5$. O restante dos parâmetros não foi modificado.

Figura 3.10 – Visualização dos resultados no *Python Shell*



Fonte: Autor.

a) Classes do conjunto de testes (saídas).
b) Resultados da predição das classes (média de acertos).
c) Precision: Capacidade do classificador de não rotular uma amostra na classe incorreta
Recall: Capacidade do classificador de rotular uma amostra na classe correta
F1-score: Resultado mais importante; corresponde à média entre precision e recall.
d) Número de ocorrência das features, do conjunto de testes em cada classe.
e) Micro avg: Média das médias dos resultados por classe
Macro avg: Média total dos acertos.
Weigthted avg: Média das médias ponderadas das ocorrências (suport) por classe
f) Matriz de confusão.
g) Parâmetros do algoritmo. (SCIKIT-LEARN DEVELOPERS).

3.4.1 Primeira Etapa

A análise dos dados a partir daqui, será feita usando diagramas de dispersão das amostras *Am*, coletadas em cada intervalo. Tal que, cada amostra *Am*, é composta pelo par ordenado dos valores RSSI dos gateways um e dois conforme a Equação 3.8.

$$Am = RSSI(G1, G2) \tag{3.8}$$

Nesta Etapa serão utilizados todos os dados da segunda etapa do Experimento I, incluindo medidas feitas no meio e nos extremos dos intervalos. Os dados serão armazenados

em um arquivo CSV, com linhas contendo as amostras em todos os intervalos e três colunas com RSSi do *gateway1* e *gateway2* e o intervalo em que a medida foi feita respectivamente.

As duas primeiras colunas correspondem às *features* do modelo, a terceira, corresponde às classes (saídas) que o algoritmo deve aprender a classificar encontrando uma relação entre as *features*.

Tabela 3.07 – Ilustração do dataset resultante

Linha	Gateway 1	Gateway 2	Intervalo correspondente
1	RSSI [0] ,	RSSI [0],	A
[...]	[...] ,	[...],	[...]
1650	RSSI [1649],	RSSI [1649],	J

Fonte: Autor.
Ilustração do dataset resultante da primeira etapa do Experimento II armazenado como arquivo CSV.
1650 amostras de cada gateway em cada ponto de medida.

O arquivo csv ilustrado na TABELA 3.07 será importado pelo *python* como um *array numpy* e embaralhado pela função *shuflle()* , para então ser separado em duas partes, treinamento e teste, com isso pode-se aplicar o algoritmo SVM e KKN, já descritos na seção 2.5 e verificar os resultados.

Tabela 3.08 - Array de Treino:

Linha	Gateway 1	Gateway 2	Intervalo correspondente
1	Valor RSSI aleatório	Valor RSSI aleatório	Rótulo aleatório
[...]	[...]	[...]	[...]
1320	Valor RSSI aleatório	Valor RSSI aleatório	Rótulo aleatório

Fonte: Autor.
Ilustração do array numpy para treinamento dos algoritmos.

Tabela 3.09 - Array de Testes:

Linha	Gateway 1	Gateway 2	Intervalo correspondente
1	Valor RSSI aleatório	Valor RSSI aleatório	Rótulo aleatório
[...]	[...]	[...]	[...]
330	Valor RSSI aleatório	Valor RSSI aleatório	Rótulo aleatório

Fonte: Autor.
Ilustração do array numpy para teste dos algoritmos.

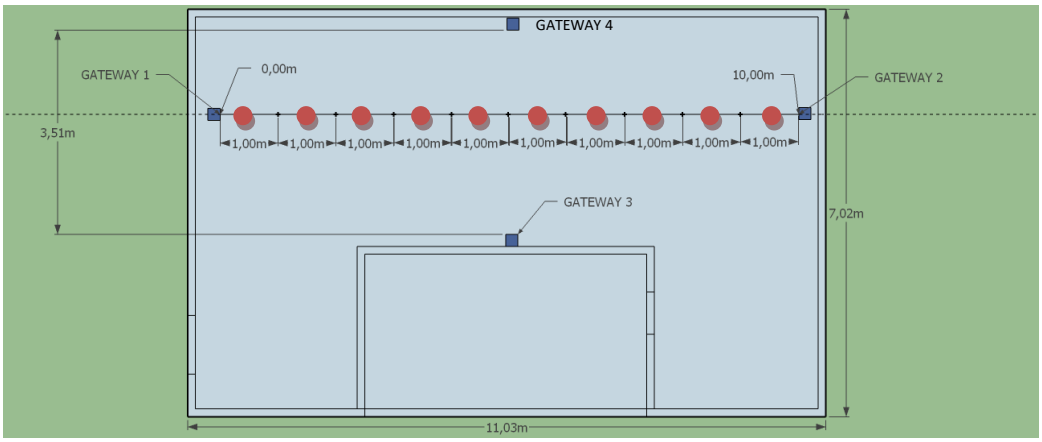
3.4.2 Segunda Etapa

Nesta etapa, os algoritmos serão testados com mais *features*, que serão importantes no Experimento III. Com quatro gateways, as amostras *Am* serão representadas por quatro coordenadas:

$$A = RSSI(G1, G2, G3, G4) \tag{3.9}$$

Foram feitas novas medidas em uma configuração semelhante à representada pelo diagrama da Figura 3.08 da seção 3.2.2.

Figura 3.11 - Planta baixa do local onde foi feita a segunda etapa do Experimento II



Fonte: Autor.
Diagrama do experimento I, segunda etapa.
Distância dos nós âncora: 10 metros (10 intervalos de 1 metro).
Em vermelho: Medidas coletadas com o nó móvel.
Em azul: Nó âncora (4 usados) .
Local das medidas: no meio dos intervalos.

Os próximos passos para fazer o treinamento são idênticos aos da etapa anterior, com exceção que agora o arquivo CSV possui mais duas colunas de *features*.

Tabela 3.10 - Ilustração do dataset resultante

Linha	Gateway 1	Gateway 2	Gateway 3	Gateway 4	Intervalo correspondente
1	RSSI [0] ,	RSSI [0],	RSSI [0],	RSSI [0],	A
[...]	[...] ,	[...],	[...],	[...],	[...]
1000	RSSI [999],	RSSI [999],	RSSI [999],	RSSI [999],	J

Fonte: Autor.
Ilustração do dataset resultante da segunda etapa do Experimento II armazenado como arquivo CSV.
1000 amostras de cada gateway em cada ponto de medida.

3.4.3 Verificação

Para verificar a validade dos modelos de predição feitos com o SVM e o KNN na seção anterior, será gerado um novo *dataset* no mesmo ambiente, porém, com medidas em lugares aleatórios dentro dos intervalos, diferente do *dataset* utilizado para a construção dos modelos em que foram feitas medidas apenas no meio dos intervalos.

Os dados dos modelos serão mostrados em dois Gráficos, com os intervalos preditos e os intervalos verdadeiros, onde idealmente a amostra deveria ser classificada. O dataset para a validação será composto por 300 amostras.

Tabela 3.11 - Ilustração do dataset resultante

Linha	Gateway 1	Gateway 2	Gateway 3	Gateway 4	Intervalo correspondente
1	RSSI [0] ,	RSSI [0],	RSSI [0],	RSSI [0],	A
[...]	[...] ,	[...],	[...],	[...],	[...]
300	RSSI [299],	RSSI [299],	RSSI [299],	RSSI [299],	J

Fonte: Autor.
Ilustração do dataset resultante da verificação do Experimento II armazenado como arquivo CSV.
Ilustração do dataset: 300 amostras de cada gateway em cada ponto de medida.

3.5 RESULTADOS DO EXPERIMENTO II

Analisando as amostras nos diagramas de dispersão nos Gráficos 3.04 e 3.05, é possível concluir que existe agrupamento das amostras nos intervalos, porém alguns intervalos, principalmente os que se encontram na metade do percurso (D, E, F) se encontram sobrepostos, o que torna difícil para a classificação dos algoritmos. Note, que esta afirmação é

coerente com os erros de predição registrados pelo *python shell* na Figura 3.12 e na Figura 3.13.

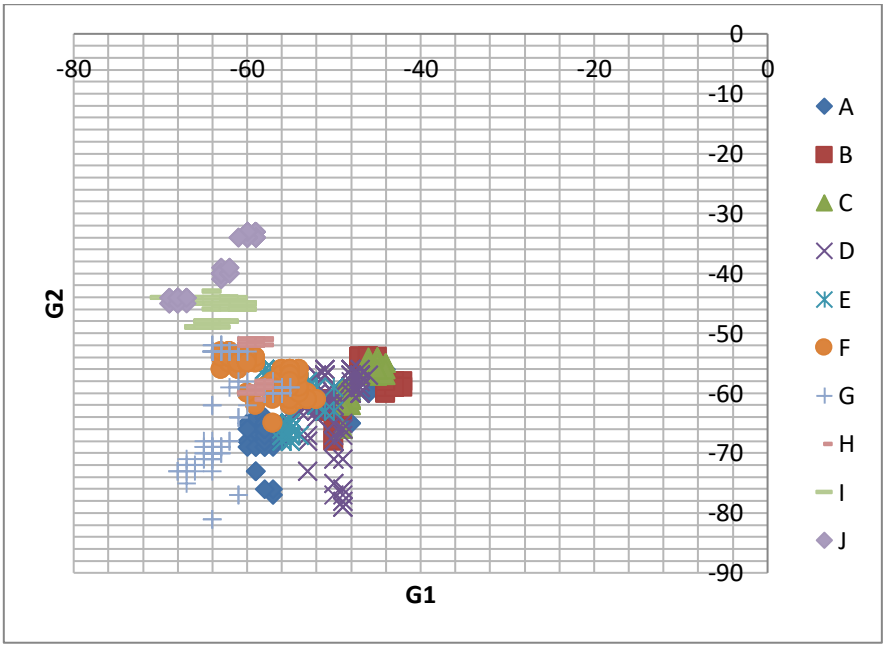
Na segunda etapa em que se utilizaram quatro gateways, foi incluído mais um par de parâmetros nas amostras, criando um espaço amostral de quatro dimensões e aumentando portanto, o número de *features* dos modelos. Como fica difícil demonstrar os dados nessa situação, foram feitos quatro Gráficos com o software Excel, com os pares: RSSIg1 versus *RSSIg2*, *RSSIg1* versus *RSSIg3*, *RSSIg2* versus *RSSIg4*, e *RSSIg3* versus *RSSIg4* (Gráficos 3.06, 3.07, 3.08 e 3.09). Como esperado, com mais *features* no modelo a predição de ambos os algoritmos cometeu pouquíssimos erros, conforme as Figuras 3.16 e 3.17.

Na verificação notou-se uma grande diferença entre o desempenho do algoritmo KNN e SVM, como mostrado nos Gráficos 3.10 e 3.11. O algoritmo SVM errou apenas uma amostra, contra 163 erradas com o KNN, conforme pode ser conferido nas matrizes de confusão das Figuras 3.18 e 3.19.

Nas seções seguintes serão mostrados os resultados do Experimento II graficamente e os resultados do desempenho dos algoritmos no console do *python Shell*. Os códigos utilizados estão nos apêndices deste trabalho, todos os dados medidos (*datasets*) estão contidos em arquivos TXT no *github* do autor (Silva, 2019).

3.5.1 Resultados Primeira etapa

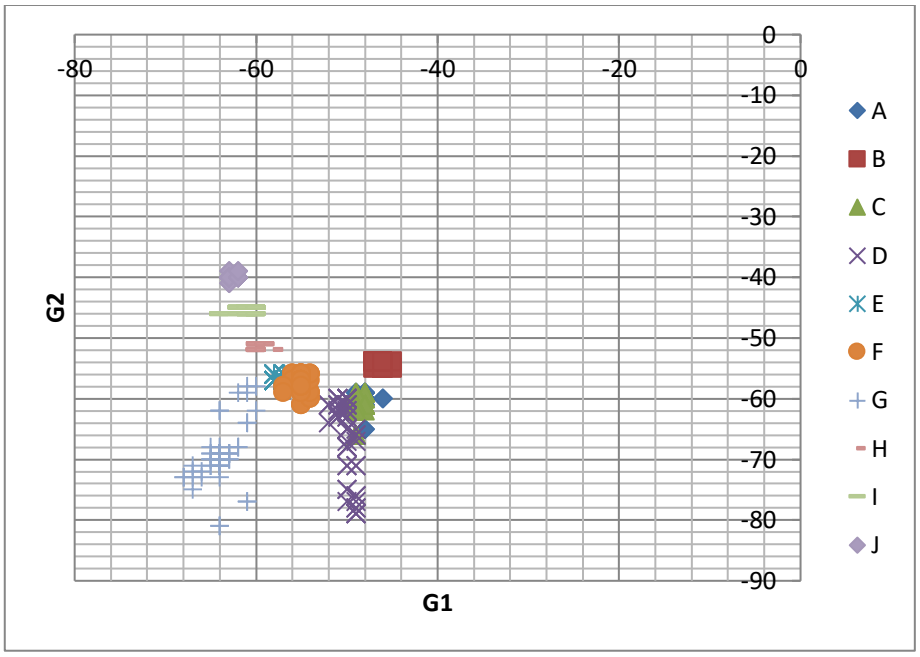
Gráfico 3.04 – Dados da primeira etapa do Experimento II:



Fonte: Autor.

Diagrama de dispersão gerado no Excel com todas as medias dos valores RSSI medidos nos intervalos.

Gráfico 3.05 – Dados da primeira etapa do Experimento II:



Fonte: Autor.

Diagrama de dispersão gerado no Excel com as medias dos valores RSSI medidos apenas nos meios dos intervalos.

Figura 3.12 – Resultados da primeira etapa do Experimento II (SVM)

```
Algoritmo SVM:
Classification report for classifier SVC(C=1.0, cache_size=200,
class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False):
              precision    recall  f1-score   support

           1       0.24      0.57      0.34        30
           2       0.64      0.26      0.37        35
           3       0.41      0.67      0.51        30
           4       0.86      0.19      0.31        32
           5       0.28      0.22      0.24        37
           6       0.47      0.39      0.43        38
           7       0.49      0.54      0.51        37
           8       0.71      0.61      0.65        28
           9       1.00      0.94      0.97        31
          10       0.94      1.00      0.97        32

   micro avg       0.52      0.52      0.52       330
   macro avg       0.60      0.54      0.53       330
weighted avg       0.60      0.52      0.52       330


Confusion matrix:
[[17  4  6  0  1  0  2  0  0  0]
 [12  9 14  0  0  0  0  0  0  0]
 [10  0 20  0  0  0  0  0  0  0]
 [13  1  9  6  3  0  0  0  0  0]
 [17  0  0  1  8 11  0  0  0  0]
 [ 0  0  0  0 11 15  9  3  0  0]
 [ 2  0  0  0  5  6 20  4  0  0]
 [ 0  0  0  0  1  0 10 17  0  0]
 [ 0  0  0  0  0  0  0  0 29  2]
 [ 0  0  0  0  0  0  0  0  0 32]]
>>>
```

Fonte: Autor.

Dados do console do *python shell* ao aplicar o algoritmo SVM usando as medidas dos meios e extremos dos intervalos.

Figura 3.13 – Resultados da primeira etapa do Experimento II (KNN)

```
Algoritmo KNN:
Classification report for classifier KNeighborsClassifier(algorithm='auto',
leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'):
              precision    recall  f1-score   support

           1       0.78      0.74      0.76        38
           2       0.89      0.92      0.91        37
           3       0.74      0.81      0.77        31
           4       0.83      0.74      0.78        34
           5       0.90      0.90      0.90        41
           6       0.78      0.94      0.85        31
           7       1.00      0.89      0.94        28
           8       0.97      0.94      0.95        33
           9       1.00      1.00      1.00        31
          10       1.00      1.00      1.00        26

   micro avg       0.88      0.88      0.88       330
   macro avg       0.89      0.89      0.89       330
weighted avg       0.89      0.88      0.88       330


Confusion matrix:
[[28  1  8  1  0  0  0  0  0  0]
 [ 0 34  1  2  0  0  0  0  0  0]
 [ 2  2 25  2  0  0  0  0  0  0]
 [ 3  1  0 25  2  3  0  0  0  0]
 [ 2  0  0  0 37  2  0  0  0  0]
 [ 0  0  0  0  1 29  0  1  0  0]
 [ 1  0  0  0  1  1 25  0  0  0]
 [ 0  0  0  0  0  2  0 31  0  0]
 [ 0  0  0  0  0  0  0  0 31  0]
 [ 0  0  0  0  0  0  0  0  0 26]]
>>>
```

Fonte: Autor.
Dados do console do *python shell* ao aplicar o algoritmo KNN usando as medidas dos meios e extremos dos intervalos

Figura 3.14 – Resultados da primeira etapa do Experimento II (SVM)

```
---
Algoritmo SVM:
Classification report for classifier SVC(C=1.0, cache_size=200,
class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False):
              precision    recall  f1-score   support

           1       0.14      0.40      0.21         5
           2       0.83      0.36      0.50        14
           3       0.38      0.89      0.53         9
           4       0.30      0.27      0.29        11
           5       0.50      0.73      0.59        11
           6       0.83      0.33      0.48        15
           7       0.60      0.38      0.46        16
           8       0.88      0.70      0.78        10
           9       1.00      0.89      0.94         9
          10       0.91      1.00      0.95        10

   micro avg       0.56      0.56      0.56       110
   macro avg       0.64      0.59      0.57       110
weighted avg       0.67      0.56      0.57       110


Confusion matrix:
[[ 2  0  2  1  0  0  0  0  0  0]
 [ 0  5  5  4  0  0  0  0  0  0]
 [ 0  0  8  1  0  0  0  0  0  0]
 [ 1  1  6  3  0  0  0  0  0  0]
 [ 2  0  0  1  8  0  0  0  0  0]
 [ 7  0  0  0  2  5  1  0  0  0]
 [ 2  0  0  0  6  1  6  1  0  0]
 [ 0  0  0  0  0  0  3  7  0  0]
 [ 0  0  0  0  0  0  0  0  8  1]
 [ 0  0  0  0  0  0  0  0  0 10]]
>>>
```

Fonte: Autor.
Dados do console do *python shell* ao aplicar o algoritmo SVM usando apenas as medidas dos meios dos intervalos.

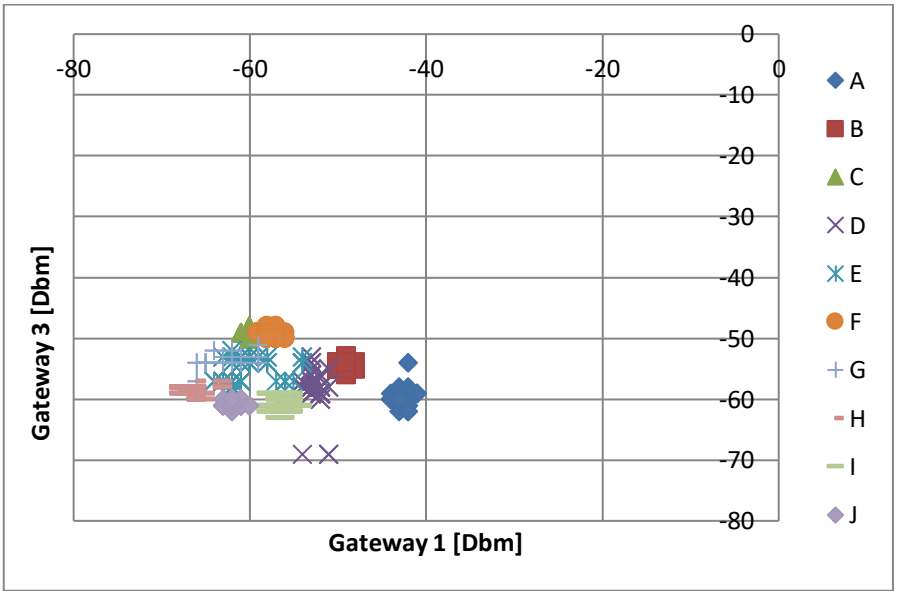
Figura 3.15 – Resultados da primeira etapa do Experimento II (KNN)

```
Algoritmo KNN:
Classification report for classifier KNeighborsClassifier(algorithm='auto',
leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'):
              precision    recall  f1-score   support

           1       0.64      0.60      0.62        15
           2       0.69      1.00      0.82         9
           3       0.50      0.71      0.59         7
           4       1.00      0.55      0.71        11
           5       0.78      0.88      0.82         8
           6       1.00      0.82      0.90        17
           7       1.00      0.92      0.96        13
           8       0.78      1.00      0.88         7
           9       1.00      1.00      1.00        11
          10       1.00      1.00      1.00        12

   micro avg       0.84      0.84      0.84       110
   macro avg       0.84      0.85      0.83       110
weighted avg       0.86      0.84      0.84       110


Confusion matrix:
[[ 9  1  5  0  0  0  0  0  0  0]
 [ 0  9  0  0  0  0  0  0  0  0]
 [ 1  1  5  0  0  0  0  0  0  0]
 [ 2  2  0  6  1  0  0  0  0  0]
 [ 1  0  0  0  7  0  0  0  0  0]
 [ 1  0  0  0  1 14  0  1  0  0]
 [ 0  0  0  0  0  0 12  1  0  0]
 [ 0  0  0  0  0  0  0  7  0  0]
 [ 0  0  0  0  0  0  0  0 11  0]
 [ 0  0  0  0  0  0  0  0  0 12]]
>>>
```

Fonte: Autor.
Dados do console do *python shell* ao aplicar o algoritmo KNN usando apenas as medidas dos meios dos intervalos.

3.5.2 Resultados Segunda Etapa

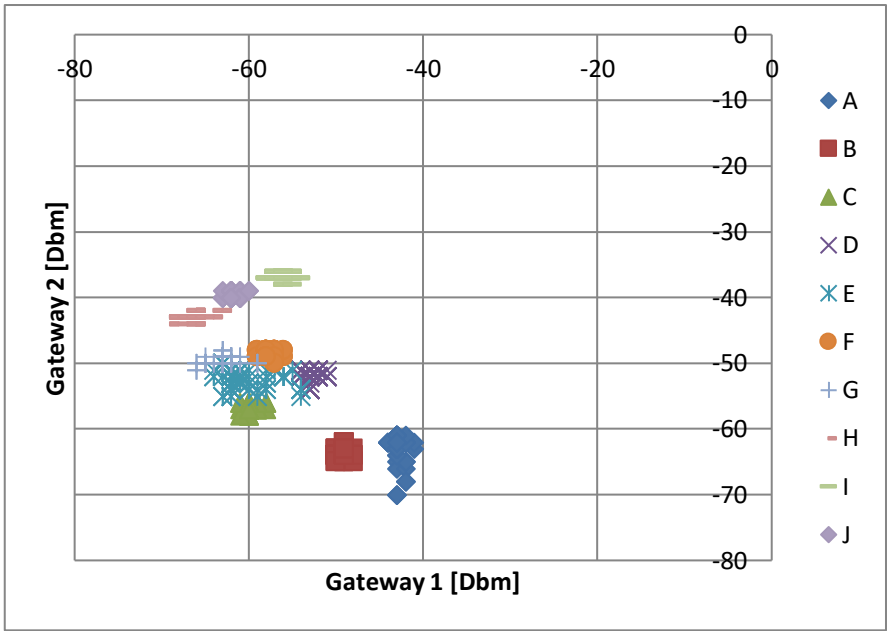
Gráfico 3.06 – Dados da segunda etapa do Experimento II:



Fonte: Autor.

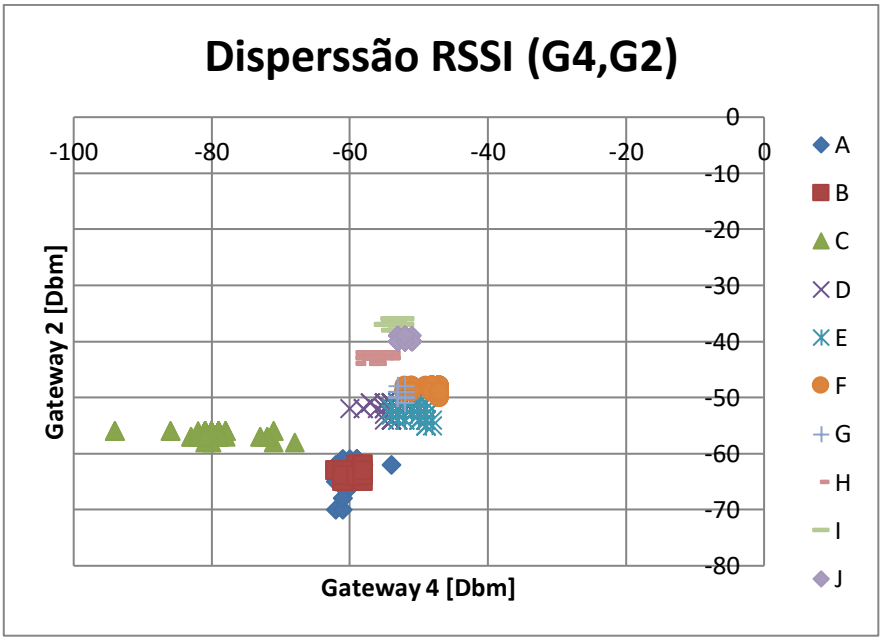
Diagrama de dispersão referente ao Gateway 1e o Gateway 3 gerado no Excel com as medias dos valores RSSI medidos no meio de cada intervalo.

Gráfico 3.07 – Dados da segunda etapa do Experimento II:



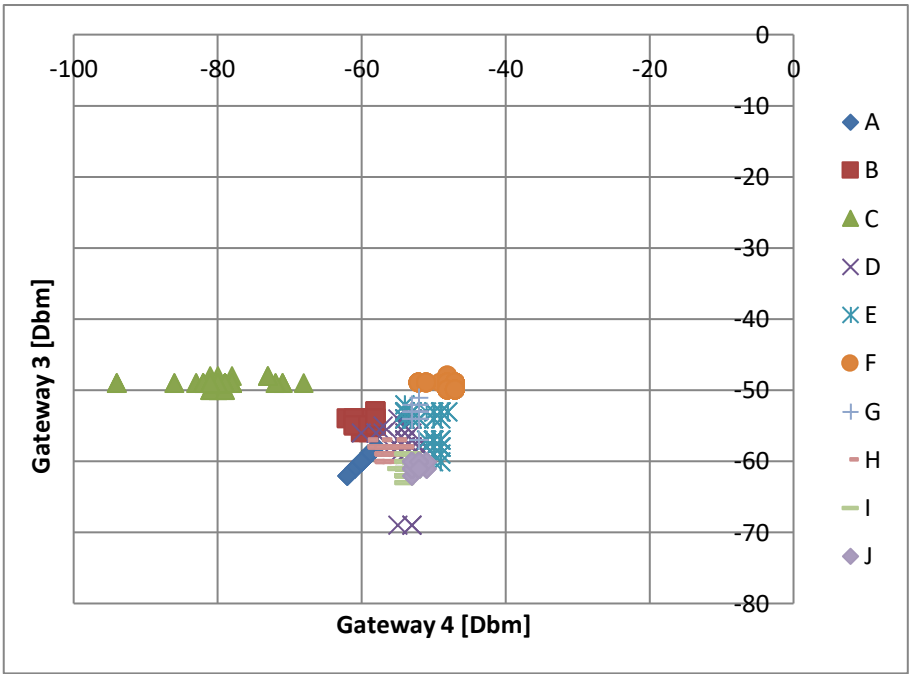
Fonte: Autor.
Diagrama de dispersão referente ao Gateway 1e o Gateway 2 gerado no Excel com as medias dos valores RSSI medidos no meio de cada intervalo.

Gráfico 3.08 – Dados da segunda etapa do Experimento II:



Fonte: Autor.
Diagrama de dispersão referente ao Gateway 4 e o Gateway 2 gerado no Excel com as medias dos valores RSSI medidos no meio de cada intervalo.

Gráfico 3.09 – Dados da segunda etapa do Experimento II:



Fonte: Autor.
Diagrama de dispersão referente ao Gateway 4e o Gateway 3 gerado no Excel com as medias dos valores RSSI medidos no meio de cada intervalo.

Figura 3.16 – Resultados da segunda etapa do Experimento II (KNN)

```
Algoritmo KNN
Classification report for classifier KNeighborsClassifier(algorithm='auto',
leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'):
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        20
           2       1.00      1.00      1.00        18
           3       1.00      1.00      1.00        16
           4       1.00      1.00      1.00        20
           5       0.96      1.00      0.98        26
           6       1.00      1.00      1.00        23
           7       1.00      0.95      0.98        21
           8       1.00      1.00      1.00        21
           9       1.00      1.00      1.00        12
          10       1.00      1.00      1.00        23

   micro avg       0.99      0.99      0.99       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      0.99      0.99       200


Confusion matrix:
[[20  0  0  0  0  0  0  0  0  0]
 [ 0 18  0  0  0  0  0  0  0  0]
 [ 0  0 16  0  0  0  0  0  0  0]
 [ 0  0  0 20  0  0  0  0  0  0]
 [ 0  0  0  0 26  0  0  0  0  0]
 [ 0  0  0  0  0 23  0  0  0  0]
 [ 0  0  0  0  1  0 20  0  0  0]
 [ 0  0  0  0  0  0  0 21  0  0]
 [ 0  0  0  0  0  0  0  0 12  0]
 [ 0  0  0  0  0  0  0  0  0 23]]
>>>
```

Fonte: Autor.

Dados do console do *python shell* ao aplicar o algoritmo KNN usando apenas as medidas dos meios dos intervalos.

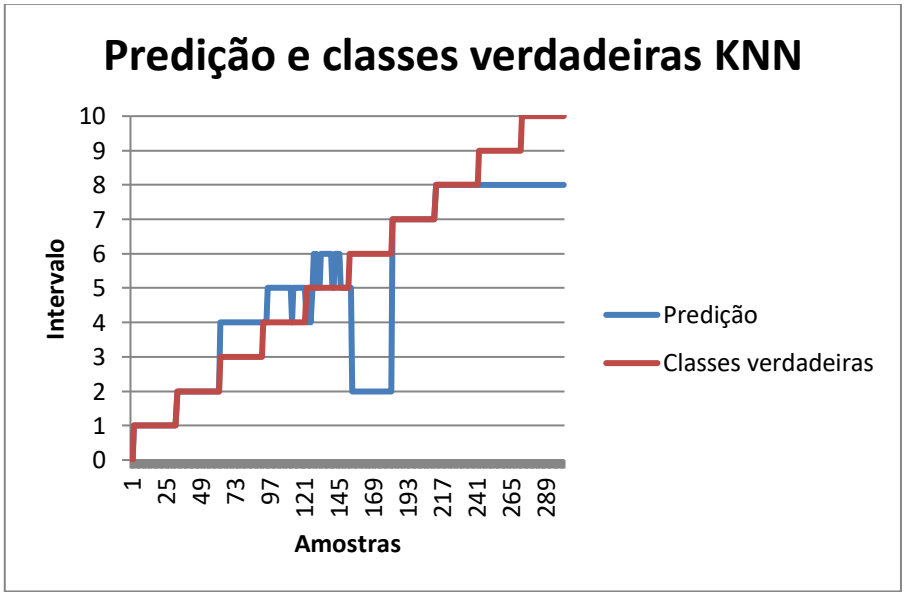
Figura 3.17 – Resultados da segunda etapa do Experimento II (SVM)

```
Algoritmo SVM
Classification report for classifier SVC(C=1.0, cache_size=200,
class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False):
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        19
           2       1.00      1.00      1.00        22
           3       1.00      1.00      1.00        28
           4       1.00      1.00      1.00        18
           5       0.92      1.00      0.96        11
           6       1.00      1.00      1.00        26
           7       1.00      0.95      0.98        21
           8       1.00      1.00      1.00        16
           9       1.00      1.00      1.00        21
          10       1.00      1.00      1.00        18

   micro avg       0.99      0.99      0.99       200
   macro avg       0.99      1.00      0.99       200
weighted avg       1.00      0.99      1.00       200


Confusion matrix:
[[19  0  0  0  0  0  0  0  0  0]
 [ 0 22  0  0  0  0  0  0  0  0]
 [ 0  0 28  0  0  0  0  0  0  0]
 [ 0  0  0 18  0  0  0  0  0  0]
 [ 0  0  0  0 11  0  0  0  0  0]
 [ 0  0  0  0  0 26  0  0  0  0]
 [ 0  0  0  0  1  0 20  0  0  0]
 [ 0  0  0  0  0  0  0 16  0  0]
 [ 0  0  0  0  0  0  0  0 21  0]
 [ 0  0  0  0  0  0  0  0  0 18]]
>>>
```

Fonte: Autor.
Dados do console do *python shell* ao aplicar o algoritmo KNN usando apenas as medidas dos meios dos intervalos.

3.5.3 Verificação do Modelo

Gráfico 3.10 - Modelo de predição gerado com KNN

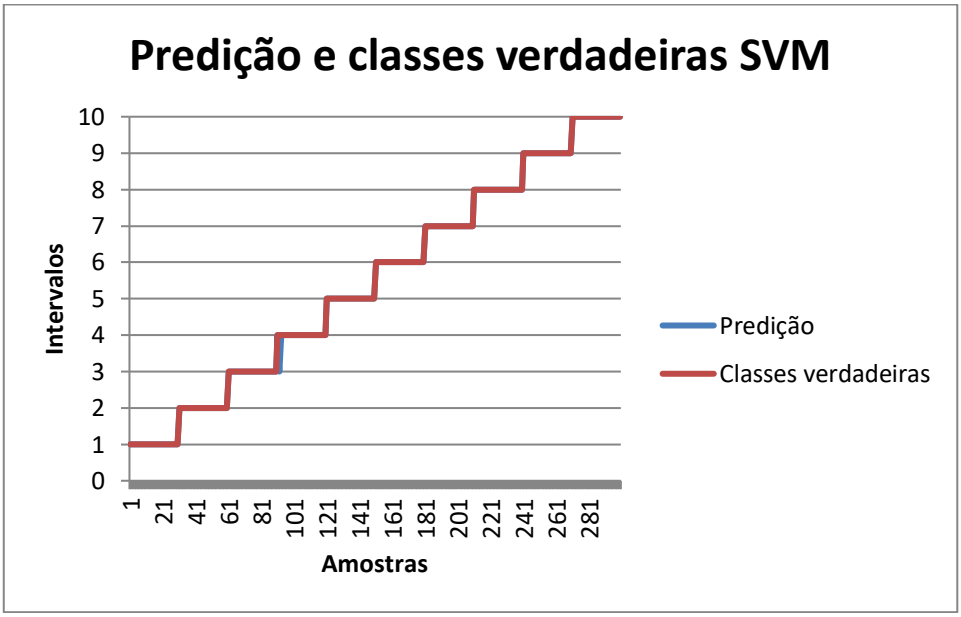


Fonte: Autor.

Gráfico gerado no Excel comparando os intervalos preditos com os verdadeiros.

Gráfico 3.11 - Modelo de predição gerado com SVM

Gráfico 3.11 - Modelo de predição gerado com KNN



Fonte: Autor.
Gráfico gerado no Excel comparando os intervalos preditos com os verdadeiros.

Figura 3.18 – Resultados da Verificação do Experimento II (KNN)

```
Classification report for classifier KNeighborsClassifier(algorithm='auto',
leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'):
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        30
           2       0.52      1.00      0.68        30
           3       0.00      0.00      0.00        30
           4       0.11      0.13      0.12        30
           5       0.32      0.43      0.37        30
           6       0.00      0.00      0.00        30
           7       1.00      1.00      1.00        30
           8       0.33      1.00      0.50        30
           9       0.00      0.00      0.00        30
          10       0.00      0.00      0.00        30

   micro avg       0.46      0.46      0.46       300
   macro avg       0.33      0.46      0.37       300
weighted avg       0.33      0.46      0.37       300


Confusion matrix:
[[30  0  0  0  0  0  0  0  0  0]
 [ 0 30  0  0  0  0  0  0  0  0]
 [ 0  0  0 30  0  0  0  0  0  0]
 [ 0  0  0  4 26  0  0  0  0  0]
 [ 0  0  0  3 13 14  0  0  0  0]
 [ 0 28  0  0  2  0  0  0  0  0]
 [ 0  0  0  0  0  0 30  0  0  0]
 [ 0  0  0  0  0  0  0 30  0  0]
 [ 0  0  0  0  0  0  0 30  0  0]
 [ 0  0  0  0  0  0  0 30  0  0]]


>>>
```

Fonte: Autor.
Dados do console do *python shell* ao aplicar o algoritmo KNN usando apenas as medidas dos meios dos intervalos.

Figura 3.19 – Resultados da Verificação do Experimento II (SVM)

```
Algoritmo SVM
Classification report for classifier SVC(C=1.0, cache_size=200,
class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False):
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        30
           2       1.00      1.00      1.00        30
           3       0.97      1.00      0.98        30
           4       1.00      0.97      0.98        30
           5       1.00      1.00      1.00        30
           6       1.00      1.00      1.00        30
           7       1.00      1.00      1.00        30
           8       1.00      1.00      1.00        30
           9       1.00      1.00      1.00        30
          10       1.00      1.00      1.00        30

   micro avg       1.00      1.00      1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300


Confusion matrix:
[[30  0  0  0  0  0  0  0  0  0]
 [ 0 30  0  0  0  0  0  0  0  0]
 [ 0  0 30  0  0  0  0  0  0  0]
 [ 0  0  1 29  0  0  0  0  0  0]
 [ 0  0  0  0 30  0  0  0  0  0]
 [ 0  0  0  0  0 30  0  0  0  0]
 [ 0  0  0  0  0  0 30  0  0  0]
 [ 0  0  0  0  0  0  0 30  0  0]
 [ 0  0  0  0  0  0  0  0 30  0]
 [ 0  0  0  0  0  0  0  0  0 30]]
```

Fonte: Autor.
Dados do console do *python shell* ao aplicar o algoritmo SVM usando apenas as medidas dos meios dos intervalos.

3.6 METODOLOGIA DO EXPERIMENTO III

No terceiro experimento foram feitas medidas em um espaço bidimensional, para serem classificadas pelos algoritmos de ML. O critério da classificação será feito de duas maneiras diferentes, em salas ou quadrantes, portanto pode-se dividir o Experimento III em três etapas:

a) Primeira Etapa: Modelo de predição das salas de um apartamento.

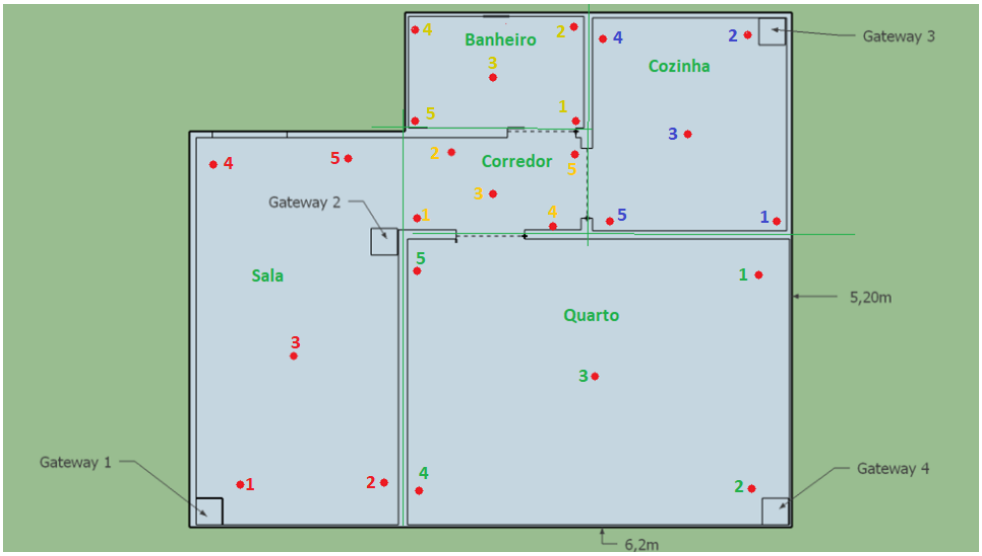
b) Segunda Etapa: Modelo de predição de ambientes e lugares das salas de um apartamento identificados por quadrantes.

c) Validação do Experimento III: Predição dos modelos feitos na Segunda Etapa para novas medidas RSSI coletadas.

3.6.1 Primeira Etapa

O Experimento II foi feito em um apartamento de 30 metros quadrados com a disposição conforme a Figura 3.20.

Figura 3.20 - Planta baixa do local onde foi feita a primeira etapa do Experimento III



Fonte: Autor.
O espaço foi dividido em cinco ambientes, em cada ambiente foram realizadas cinco medidas (pontos em vermelho) e em cada medida foi coletada cinco amostras (25 amostras por ambiente) dos sinais RSSI dos gateways.

Tabela 3.12 – Dataset gerado

	Sala	Corredor	Quarto	Banheiro	Cozinha
Gateway 1	25 amostras	25 amostras	25 amostras	25 amostras	25 amostras
Gateway 2	25 amostras	25 amostras	25 amostras	25 amostras	25 amostras
Gateway 3	25 amostras	25 amostras	25 amostras	25 amostras	25 amostras
Gateway 4	25 amostras	25 amostras	25 amostras	25 amostras	25 amostras

Fonte: Autor
Ilustração do dataset: 125 amostras de cada gateway em cada ponto de medida.

Na Figura 3.20, cada ponto numerado corresponde a uma medida em determinado ambiente (*Sala, Corredor,Quarto,Banheiro,Cozinha*).

A forma de predição da posição do nó móvel que foi adotada neste trabalho, não está atrelada às dimensões físicas do ambiente, visto que, nos resultados do Experimento I ficou evidente como mudanças no meio (salas com ambientes diversos) apresentam distorções na

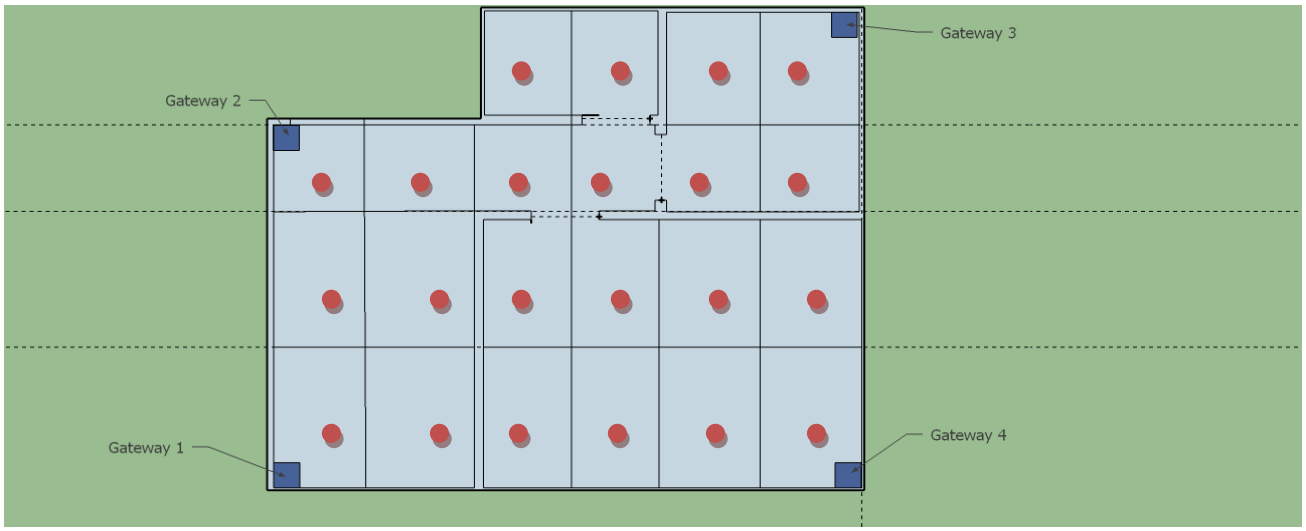
curva RSSI versus Distância, mostrando que pode ser difícil implementar um modelo que relacione diretamente a distância do nó móvel aos nós âncoras.

Nesta etapa se fará a predição da sala em que se encontra o nó móvel com os algoritmos KNN e SVM e em seguida, serão analisados os resultados.

3.6.2 Segunda Etapa

Essa etapa é muito semelhante à anterior, porém mistura um pouco da metodologia do Experimento II. O ambiente em que será treinado o algoritmo é o mesmo apartamento da primeira etapa deste experimento, mudando a disposição do *gateway2*.

Figura 3.21 - Planta baixa do local onde foi feita a segunda etapa do Experimento III



Fonte: Autor.
O espaço foi dividido em 22 quadrantes, em cada quadrante foi realizada uma medidas (pontos em vermelho) e em cada medida foi coletada 10 amostras dos sinais RSSI dos gateways.

Conforme é mostrado na Figura 3.21, dividiu-se os ambientes do apartamento em quadrantes, endereçados por números de *1* até *22*. A proposta é o algoritmo tentar identificar o lugar na sala em que o nó móvel se encontra (por exemplo: *quadrante 1 = "mesa da sala", quadrante 19 = "criado mudo do quarto",* etc.), os *22* quadrantes compõem as possíveis saídas do modelo.

Em cada quadrante foi feita uma medida em seu centro (em vermelho no diagrama) contendo *10* amostras (*220* amostras por *gateway*), gerando um *dataset* conforme a Tabela 3.13.

Tabela 3.13 Dataset Gerado

Linha	Gateway 1	Gateway 2	Gateway 3	Gateway 4	Quadrante correspondente
1	RSSI [0] ,	RSSI [0],	RSSI [0],	RSSI [0],	1
[...]	[...] ,	[...],	[...],	[...],	[...]
222	RSSI [221],	RSSI [221],	RSSI [221],	RSSI [222],	22

Fonte: Autor.
Ilustração do dataset resultante da segunda etapa do Experimento III armazenado como arquivo CSV.
220 amostras de cada gateway em cada ponto de medida.

3.7 RESULTADOS EXPERIMENTO III

Nos resultados da primeira etapa (Figura 3.22), é mostrado que a predição feita pelo algoritmo SVM errou algumas amostras medidas nos ambientes Corredor (classe 1) e Quarto (classe 3), conforme a matriz de confusão, duas amostras coletadas no Corredor foram identificadas como se estivessem no Quarto.

O modelo gerado pelo KNN teve desempenho semelhante, errou a predição de três amostras nos mesmos ambientes (Figura 3.23).

No GRÁFICO 3.12, pode-se ver que as amostras coletadas no ambiente Quarto (em verde), ficaram dispersas em todos os diagramas. Uma hipótese é que, pelo fato deste ambiente ser muito grande em relação aos demais (ver Figura 3.20), a variação dos valores RSSI medidos pelos gateways fica muito parecida com a de outros ambientes.

Na Segunda Etapa do experimento (seção 3.7.2) em que se dividiram as salas em quadrantes e aumentou-se o número de classes, o *score* atingido pela predição dos dois algoritmos foi de *100%* (Figura 3.18 e Figura 3.19) em todas as classes, porém na verificação dos modelos (seção 3.7.3), quando se utilizou medidas coletadas aleatoriamente nos intervalos, o desempenho de ambos os algoritmos foi bem baixo em classificar os intervalos, conforme A Figura 3.30 e a Figura 3.32.

Das *66* amostras, o algoritmo SVM acertou a predição de dezenove amostras (*29%*), sendo que vinte e seis das predições erradas *(39%*) apontaram para quadrantes que estavam

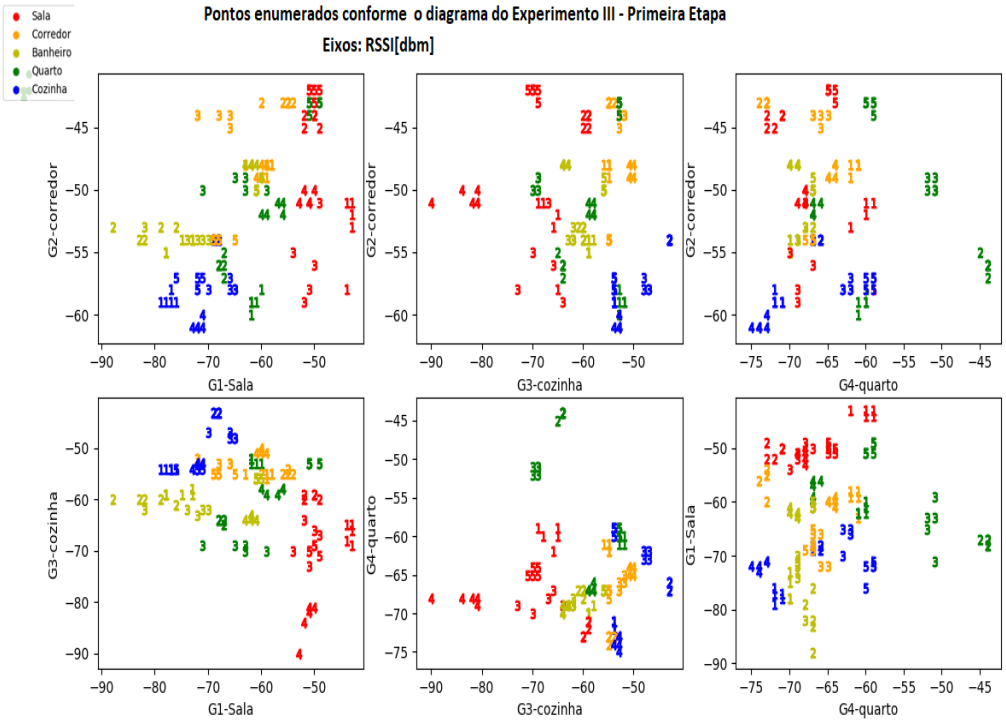
em contato com o quadrante correto. Já as *21* amostras restantes (*32%*) foram classificadas em quadrantes distantes dos verdadeiros, como no caso de uma das amostras coletadas no *Quadrante 7*, no ambiente *Sala*, o algoritmo fez predição para o *Quadrante 14* que está no ambiente *Banheiro*.

O KNN apresentou desempenho similar, classificando *18* amostras corretamente (*27%*), *29* amostras classificadas erroneamente em quadrantes próximos (*44%*) e as *19* amostras restantes preditas em quadrantes isolados (*29%*).

3.7.1 Resultados Experimento III Primeira Etapa

Nas seções seguintes serão mostrados os resultados do Experimento III graficamente e os resultados do desempenho dos algoritmos no console do *python Shell*. Os códigos utilizados estão nos apêndices deste trabalho, todos os dados medidos (*datasets*) estão contidos em arquivos TXT no *github* do autor (Silva, 2019).

Gráfico 3.12 Dados da primeira etapa do Experimento III



Fonte: Autor.
Gráfico gerado com *python* comparando a dispersão dos dados dos ambientes com referência aos quatro gateways utilizados..

Figura 3.22 – Resultados da primeira etapa do Experimento III (SVM)

```
Carregando modelo existente...
algorítmo SVM
Classification report for classifier SVC(C=1.0, cache_size=200,
class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       0.71      1.00      0.83         5
           2       1.00      1.00      1.00         2
           3       1.00      0.60      0.75         5
           4       1.00      1.00      1.00         7

   micro avg       0.92      0.92      0.92        25
   macro avg       0.94      0.92      0.92        25
weighted avg       0.94      0.92      0.92        25


Confusion matrix:
[[6 0 0 0 0]
 [0 5 0 0 0]
 [0 0 2 0 0]
 [0 2 0 3 0]
 [0 0 0 0 7]]
```

Fonte: Autor.
Dados do console do *python shell* ao aplicar o algoritmo SVM.

Figura 3.23 – Resultados da primeira etapa do Experimento III (KNN)

```
Carregando modelo existente...
algorítmo KNN
Classification report for classifier KNeighborsClassifier(algorithm='auto',
leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       0.57      1.00      0.73         4
           2       1.00      1.00      1.00         3
           3       1.00      0.67      0.80         9
           4       1.00      1.00      1.00         3

   micro avg       0.88      0.88      0.88        25
   macro avg       0.91      0.93      0.91        25
weighted avg       0.93      0.88      0.88        25


Confusion matrix:
[[6 0 0 0 0]
 [0 4 0 0 0]
 [0 0 3 0 0]
 [0 3 0 6 0]
 [0 0 0 0 3]]
```

Fonte: Autor.
Dados do console do *python shell* ao aplicar o algoritmo KNN..

Tabela 3.14 - Legenda das classes:

Classes do Algoritmo	Quadrante correspondente a classe
0	Sala
1	Corredor
2	Banheiro
3	Quarto
4	Cozinha

Fonte: Autor

3.7.2 Resultados Experimento III Segunda Etapa

Figura 3.24 – Resultados da segunda etapa do Experimento III (SVM) parte um

```
Carregando modelo...
Algoritmo SVM:
Classification report for classifier SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         3
           3       1.00      1.00      1.00         2
           4       1.00      1.00      1.00         4
           6       1.00      1.00      1.00         3
           7       1.00      1.00      1.00         3
           8       1.00      1.00      1.00         2
           9       1.00      1.00      1.00         1
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         2
          12       1.00      1.00      1.00         1
          14       1.00      1.00      1.00         2
          15       1.00      1.00      1.00         2
          16       1.00      1.00      1.00         2
          17       1.00      1.00      1.00         1
          18       1.00      1.00      1.00         2
          19       1.00      1.00      1.00         3
          20       1.00      1.00      1.00         2

   micro avg       1.00      1.00      1.00        44
   macro avg       1.00      1.00      1.00        44
weighted avg       1.00      1.00      1.00        44
```

Fonte: Autor.
Print do console do *python shell* ao aplicar o algoritmo SVM.

Figura 3.25 – Resultados da segunda etapa do Experimento III (SVM) parte dois

```
Confusion matrix:
[[4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2]]
```

Fonte: Autor.
Print do console do *python shell* ao aplicar o algoritmo SVM (matriz de confusão).

Figura 3.26 – Resultados da segunda etapa do Experimento III (KNN) parte um

```
Carregando modelo...
Algoritmo KNN:
Classification report for classifier KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         2
           4       1.00      1.00      1.00         2
           7       1.00      1.00      1.00         3
           8       1.00      1.00      1.00         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         2
          11       1.00      1.00      1.00         3
          12       1.00      1.00      1.00         2
          13       1.00      1.00      1.00         1
          14       1.00      1.00      1.00         4
          15       1.00      1.00      1.00         2
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         1
          18       1.00      1.00      1.00         2
          20       1.00      1.00      1.00         3
          21       1.00      1.00      1.00         2

   micro avg       1.00      1.00      1.00        44
   macro avg       1.00      1.00      1.00        44
weighted avg       1.00      1.00      1.00        44
```

Fonte: Autor.
Print do console do *python shell* ao aplicar o algoritmo KNN.

Figura 3.27 – Resultados da segunda etapa do Experimento III (KNN) parte dois

```
Confusion matrix:
[[2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2]]
```

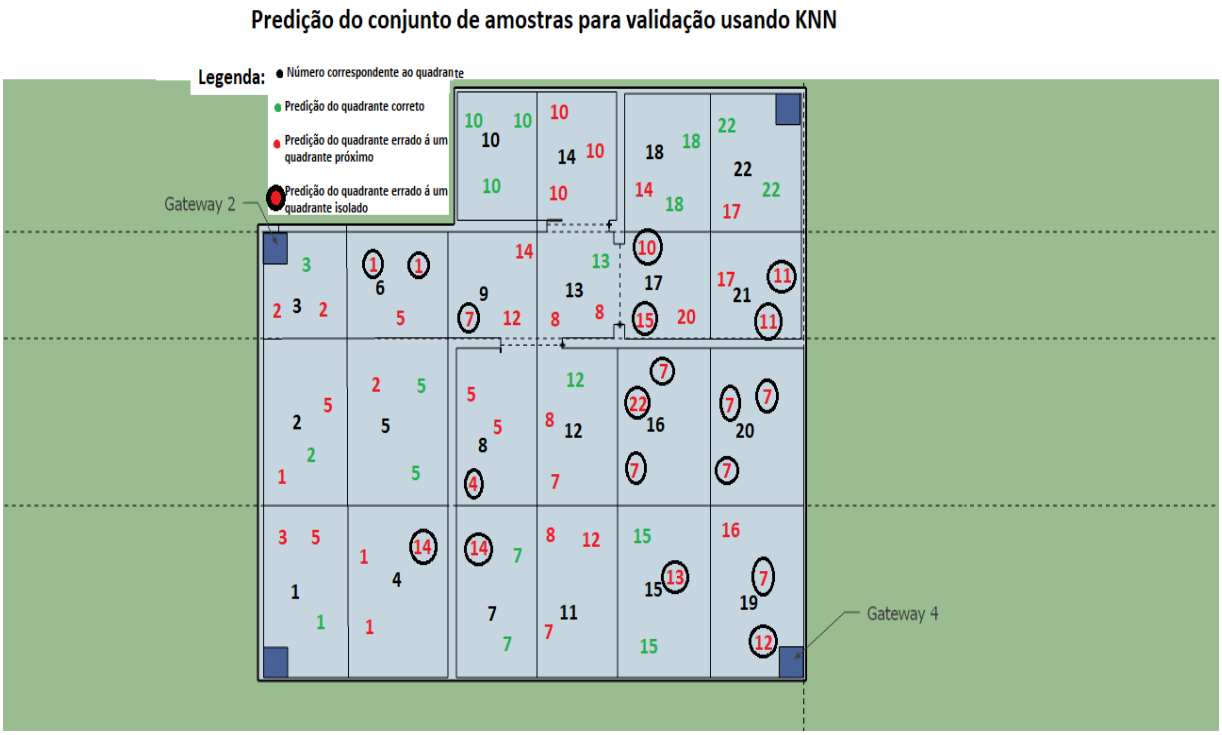
Fonte: Autor.
Print do console do *python shell* ao aplicar o algoritmo KNN (matriz de confusão).

3.7.3 Resultados da verificação do modelo

Figura 3.28 - Planta baixa do local onde foi feita a verificação do Experimento III (SVM)

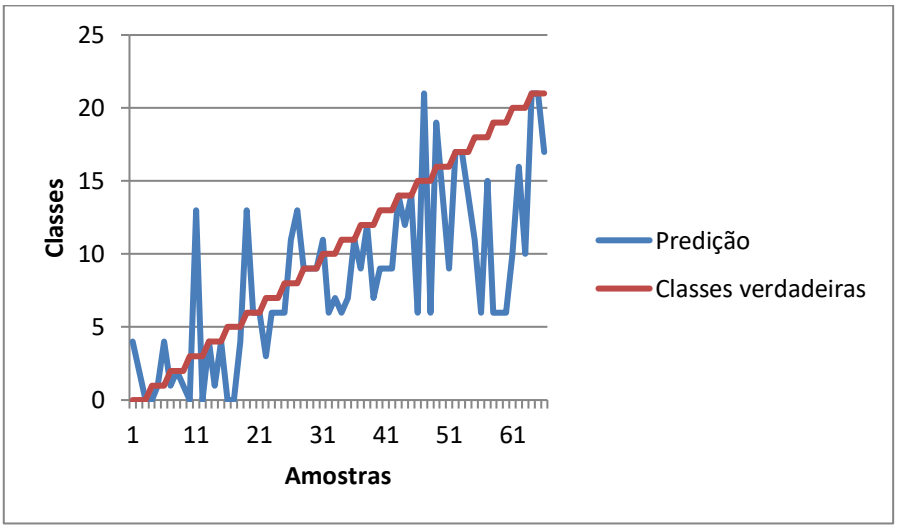
Fonte: Autor.

Figura 3.29 - Planta baixa do local onde foi feita a verificação do Experimento III (KNN)



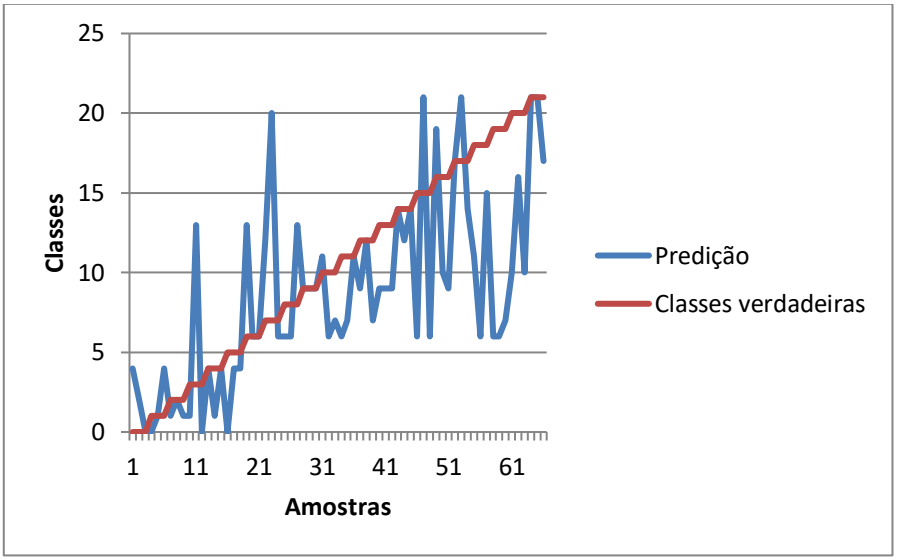
Fonte: Autor.

Gráfico 3.13- Modelo de predição gerado com KNN



Fonte: Autor.
Gráfico gerado no Excel comparando os intervalos preditos com os verdadeiros.

Gráfico 3.14- Modelo de predição gerado com SVM



Fonte: Autor.
Gráfico gerado no Excel comparando os intervalos preditos com os verdadeiros.

Figura 3.30 – Resultados da validação do Experimento III (SVM) parte um

```
Classification report for classifier SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False):
              precision    recall  f1-score   support

           0       0.25      0.33      0.29         3
           1       0.20      0.33      0.25         3
           2       0.50      0.33      0.40         3
           3       0.00      0.00      0.00         3
           4       0.33      0.67      0.44         3
           5       0.00      0.00      0.00         3
           6       0.17      0.67      0.27         3
           7       0.00      0.00      0.00         3
           8       0.00      0.00      0.00         3
           9       0.38      1.00      0.55         3
          10       0.00      0.00      0.00         3
          11       0.33      0.33      0.33         3
          12       0.33      0.33      0.33         3
          13       0.00      0.00      0.00         3
          14       0.67      0.67      0.67         3
          15       0.00      0.00      0.00         3
          16       0.00      0.00      0.00         3
          17       0.50      0.33      0.40         3
          18       0.00      0.00      0.00         3
          19       0.00      0.00      0.00         3
          20       0.00      0.00      0.00         3
          21       0.50      0.67      0.57         3

   micro avg       0.26      0.26      0.26        66
   macro avg       0.19      0.26      0.20        66
weighted avg       0.19      0.26      0.20        66
```

Fonte: Autor.
Print do console do *python shell* ao aplicar o algoritmo SVM.

Figura 3.31 – Resultados da validação do Experimento III (SVM) parte dois

```
Confusion matrix:
[[1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 2 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 2 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 2 0 0 0 0 0 0]
 [0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1]
 [0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 2]]
```

Fonte: Autor.
Print do console do *python shell* ao aplicar o algoritmo SVM (matriz de confusão).

Figura 3.32 – Resultados da validação do Experimento III (KNN) parte um

```
Classification report for classifier KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'):
              precision    recall  f1-score   support

           0       0.17      0.33      0.22         3
           1       0.25      0.33      0.29         3
           2       0.50      0.33      0.40         3
           3       0.00      0.00      0.00         3
           4       0.40      0.67      0.50         3
           5       0.00      0.00      0.00         3
           6       0.15      0.67      0.25         3
           7       0.00      0.00      0.00         3
           8       0.00      0.00      0.00         3
           9       0.38      1.00      0.55         3
          10       0.00      0.00      0.00         3
          11       0.25      0.33      0.29         3
          12       0.50      0.33      0.40         3
          13       0.00      0.00      0.00         3
          14       0.50      0.67      0.57         3
          15       0.00      0.00      0.00         3
          16       0.00      0.00      0.00         3
          17       0.67      0.67      0.67         3
          18       0.00      0.00      0.00         3
          19       0.00      0.00      0.00         3
          20       0.00      0.00      0.00         3
          21       0.67      0.67      0.67         3

   micro avg       0.27      0.27      0.27        66
   macro avg       0.20      0.27      0.22        66
weighted avg       0.20      0.27      0.22        66
```

Fonte: Autor.
Print do console do *python shell* ao aplicar o algoritmo KNN.

Figura 3.33 – Resultados da validação do Experimento III (KNN) parte um

```
Confusion matrix:
[[1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [2 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [2 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 2 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 2 0 0 0 0 0]
 [0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 2 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 2]]
```

Fonte: Autor.
Print do console do *python shell* ao aplicar o algoritmo KNN (matriz de confusão).

4 CONCLUSÃO

Após a conclusão dos três experimentos, fica nítida uma das principais dificuldades existentes na construção de um sistema IPS utilizando a intensidade do sinal (RSSI): a influência sofrida por obstáculos e existência de situações peculiares onde não existe linha de visada por parte de algum gateway. Neste caso Gateways com sinal muito fraco deveriam ser desprezados.

Também é interessante ver a diferença nas duas primeiras etapas do Experimento II, em que foram utilizados quatro gateways. No caso de apenas dois gateways a média de acertos nas classes foi menor que 50%, em contraponto, na segunda etapa atingiu quase 100% de média em todas as classes em ambos os modelos (99% no SVM e 100% no KNN), mostrando a importância do aumento dos números de *features* no modelo.

A etapa de verificação do Experimento II deixou claro que em um ambiente aberto, pouco segmentado por salas e sem objetos entre os nós de comunicação é possível atingir precisões de um metro com o uso do algoritmo SVM, que fez a predição correta de 100% das amostras coletadas para validar o modelo. No KNN, porém, apesar de predições erradas se encontrarem em intervalos próximos, se obteve apenas 37% de média de acertos em todas as classes, deve-se considerar que se trata de um algoritmo mais simples que o SVM.

Na primeira etapa do Experimento III, não foi possível construir um modelo com taxas altas de acertos de predições das salas em que foram feitas as medidas com o nó móvel, talvez pela dispersão das amostras ficarem muito homogêneas da forma como foi tentado classificar (em salas), dificultando os algoritmos de conseguirem separar os conjuntos. Quando separadas em quadrantes (segunda etapa) todas as predições do conjunto de testes em ambos os algoritmos foram feitas corretamente. Porém, quando se mediu pontos em lugares diferentes dos que se mediram inicialmente para criar o modelo (centro dos quadrantes), a taxa de predições corretas caiu bastante. Nos Gráficos 3.14 e 3.15 pode-se ver como as predições acompanham a tendência do resultado ideal, mas apresentam enorme variação, principalmente por conta de predições que sugerem quadrantes muito afastados do correto. A variação pode ser oriunda de efeitos como sombreamento e multipercurso do sinal, explicados na seção 2.4.3, porém não foram feitos estudos conclusivos a respeito neste trabalho. É

interessante que a maioria das predições incorretas sugeriu um quadrante que estava próximo (em contato) com o correto, sugerindo que apesar de toda a instabilidade dos fatores oriundos do meio, os algoritmos conseguem definir razoavelmente bem a posição relativa do Nó móvel da rede com os nós Âncora.

Outro fator que pode ter sido decisivo na construção de modelos desse tipo é o aumento no número de gateways. Esta conclusão é provada no Experimento II, porém seria ideal realizar o Experimento III em um lugar aberto e sem obstáculos, para tirar melhores conclusões quando os espaços são bidimensionais.

A implementação da rede *LoRaWAN* mostrou-se bem simples e relativamente barata, sendo os gateways os dispositivos mais caros, o que motiva uma continuação da pesquisa tecnológica realizada neste trabalho.

4.1 SEGMENTOS DA PESQUISA

No trabalho, não houve tempo de pesquisar a fundo os parâmetros utilizados pelos algoritmos, principalmente no SVM, em que existem inúmeras versões não lineares que poderiam ser mais apropriadas para a situação. O algoritmo KNN não é muito sofisticado, sendo que o único parâmetro ajustável é o número de *k* vizinhos, e como isso não resultou em diferenças significativas, não foi abordado neste trabalho.

Uma possível abordagem para melhores resultados em ambientes que afetam demasiadamente a relação RSSI e distância (descrita na equação 2.1) é a técnica de Mapa de Assinatura (Regilane, 2016), em que se levantam dados RSSI no máximo de lugares possíveis da sala e é feito um mapeamento da intensidade do sinal de cada gateway, porém tornaria o processo de treinamento (aquisição das amostras) pouco prático, sendo necessário realizar um experimento trabalhoso em cada lugar que fosse implementado o IPS.

5 REFERÊNCIAS

Aernouts, M.; Berkvens, R; Van, K; Vlaenderen; Weyn, M.; **Sigfox and LoRaWAN Datasets for Fingerprint Localization in Large Urban and Rural Areas.** University of Antwerp, 5 de abril de 2018. Acessado em: 15 de maio de 2019. Disponível em: https://www.mdpi.com/2306-5729/3/2/13.

ARDUINO. **Arduino IDE.** Portal Arduino, 2019. Acessado em 12 de abril de 2019. Disponível em: https://www.arduino.cc/en/Main/Software

Blackman, J. **LoRa shoots for 75% of IoT market, versus 25% for 5G; aims for stars with satellite constellation.** Enterprise iot insights, 18 de junho de 2019**.** Acesso em 20 de junho de 2019 . Disponível em https://enterpriseiotinsights.com/20190618/channels/news/lora-shoots-for-75pc-of-iot-market .

BNDES, **Aspiração do Brasil em Internet das Coisas 2017**.Relatório parcial BNDES, 2017. Disponível em https://www.bndes.gov.br/wps/wcm/connect/site/d1348f10-c93d-408e-8c2d-a2c2a6e4efb1/170614_Produto_Parcial_Frente+3_Aspiracao_IoT_Final.pdf?MOD=AJPERES&CVID=lOOiRj1. Acessado em 25 de junho de 2019.

Carvalho, E.C. **Investigação em Arquiteturas de Redes sem Fio para Localização em Ambientes Internos Usando Aprendizado de Máquina.** Universidade Federal do Pará – UFPA, 2016. Acessado em 14 junho de 2019. Disponível em: http://ppgcc.propesp.ufpa.br/Disserta%C3%A7%C3%B5es_2016/Eduardo%20Costa%20de%20Carvalho_Disserta%C3%A7%C3%A3o.pdf.

ESP32. **FEATURES & SPECIFICATIONS.** 2019. Acessado em 12 de março de 2019. Disponível em: http://esp32.net/.

Fargas, Carbones, B.; Petersen; Nordal**, M.GPS-free Geolocation using LoRa in Low-Power WANs.** Global Internet of Things Summit (GIoTS), 2017. Acessado em 5 de abril de 2019. Disponível em: https://orbit.dtu.dk/files/130478296/paper_final_2.pdf.

FINEP**. Kevin Ashton – entrevista exclusiva com o criador do termo "Internet das Coisas"**. Portal da FINEP, 2015. Disponível em (http://finep.gov.br/noticias/todas-noticias/4446-kevin-ashton-entrevista-exclusiva-com-o-criador-do-termo-internet-das-coisas).

Haute, T. V.; Poorter, E.; Crombez, P.; Lemic, F.; Handziski V; Wirström, N; Wolisz A.; Voigt T.; Moerman I. **Performance analysis of multiple Indoor Positioning Systems in a healthcare environment**. International Journal of Health Geographics, 2016. Acesso em 20

de junho de 2019. Disponível em: (https://ijhealthgeographics.biomedcentral.com/track/pdf/10.1186/s12942-016-0034-z.

Lawrence, L. **GPS Made Easy**. Canadá Coast Guard 2000 Edition. Acesso em 17 de junho de 2019. Disponível em http://www.ccg-gcc.gc.ca/folios/00021/docs/dgps_guide-eng.pdf.

LORA ALLIANCE. **About LoRa Alliance.** 2019. Acessado em 12 de abril de 2019. Disponível em:
https://www.arduino.cc/en/Main/Software.
Matthijskooijman. **Arduino-LMIC library.** Github, 2017. Acessado em 16 de abril de 2019. Disponível em https://github.com/matthijskooijman/arduino-lmic.

NOVIDA, **Geolocalização Indoor: Conheça as aplicações.** 2019. Disponível em: https://novida.com.br/blog/geolocalizacao-indoor/. Acesso em 21 junho de 2019

NUMPY DEVELOPERS, **NumPy.** Scipy, 2019. Acessado em 20 de junho de 2019. Disponível em:https://www.numpy.org/.

Oliveira, G. C.; **Localização indoor utilizando a tecnologia LoRaWAN e aprendizado de máquina.** Instituto Federal de Santa Catarina – IFSC, 2017. Acessado em 15 de abril de 2019. Disponível em:
https://wiki.sj.ifsc.edu.br/wiki/images/a/ad/TCC290_Giulio_Cruz_de_Oliveira.pdf.

Oliveira, L. R. **Setup LoRa com Arduino, Raspberry Pi e shield Dragino.** Embarcados.com.br, 28 de maio de 2018. Acessado em 14 de abril de 2019. Disponível em: https://www.embarcados.com.br/lora-arduino-raspberry-pi-shield-dragino/.

PAHO MQTT. **Paho-mqtt 1.4.0.** Pypi, 2 de setembro de 2018. Acessado em 10 de abril de 2019. Disponível em: https://pypi.org/project/paho-mqtt/

PICKLE. **Python object serialization.** Docs Python, 2019. Acessado em 16 de maio de 2019. Disponível em: https://docs.python.org/3/library/pickle.html.

RARPBIAN. **Raspbian Pi Desktop**. 2019. Acessado em 14 de maio de 2019. Disponível em: https://www.raspberrypi.org/downloads/.

Regilane L. Paiva, A. R; Cavalcanti, R. P.; Nascimento, H. J. B. **Avaliação de Algoritmos de Localização Indoor baseados em Mapa de Assinatura de WLANs.** XXXIV SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES, Santarém, PA. 30 de setembro de 2016. Acessado em 17 de junho de 2019. Disponível em:
http://www.sbrt.org.br/sbrt2016/anais/ST13/1570274889.pdf.

Santos, F. C. **Variações do Método kNN e suas Aplicações na Classificação Automática de Texto.** UNIVERSIDADE FEDERAL DE GOIÁS, Goiania, 2009. Acessado em 10 maio de abril de 2019. Disponível em:
http://www.inf.ufg.br/mestrado/sites/www.inf.ufg.br.mestrado/files/uploads/Dissertacoes/Fernando%20Chagas.pdf#page=89&zoom=100,0,781.

SCIKIT-LEARN DEVELOPERS. **Documentation of scikit-learn 0.21.2.** 2019. Acessado em 13 de abril de 2019. Disponível em: https://scikit-learn.org/stable/documentation.html.

SKETCHUP, Software de modelagem 3D. Acessado em: 5 de abril de 2019. Disponível em: https://www.sketchup.com/pt-BR

Silva, T. N. B. **IPS com Lora WAN.** Github, 2019. Acessado em 26 de junho de 2019. Disponível em: https://github.com/TarsisNatan/IPS-com-LoRa-WAN.

Sônego A.;Marcelino, R.; Gruber, V. **A Internet das Coisas aplicada ao conceito de eficiência**
energética:uma análise quantitativo-qualitativa do estado da arte da literatura. Universidade Federal de Santa Catarina, 28 de julho de 2016.Disponível em: https://revistas.ufpr.br/atoz/article/view/47860/30163. Acesso em 22 de junho de 2019

Telkamp, T. **Single Channel LoRaWAN Gateway.** Github, 2017. Acessado em 14 de Abril de 2019. Disponível em: https://github.com/tftelkamp/single_chan_pkt_fwd.

THE THINGS NETWORK. **Documentation**, 2019. Acessado em 20 de maio de 2019. Disponível em https://www.thethingsnetwork.org/docs/.

6 APÊNDICES

6.1 PROGRAMA DO NÓ MÓVEL

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <MPU9250_asukiaaa.h>

#include <Arduino.h>
#include <stdint.h>
#include<string.h>


#ifdef _ESP32_HAL_I2C_H_
#define SDA_PIN 21
#define SCL_PIN 22
#endif
// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.

static const PROGMEM u1_t NWKSKEY[16] = { 0xBA, 0xDD, 0x0E, 0x68, 0xCF, 0x67, 0xAB,
0x40, 0x44, 0xD1, 0xA8, 0x78, 0x9C, 0x08, 0xA3, 0xBA };

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const u1_t PROGMEM APPSKEY[16] = { 0x84, 0x28, 0x4C, 0x8B, 0xA3, 0x2E, 0x87,
0x62, 0x2F, 0xA2, 0x44, 0xD5, 0xC3, 0x70, 0x0A, 0xC8 };

// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x26031132 ; // <-- Change this address for every node!

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t mydata[12] = {"2222"};
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 2;

// Pin mapping
const lmic_pinmap lmic_pins = {
```

```
45 .nss = 18,
46 .rxtx = LMIC_UNUSED_PIN,
47 .rst = 23,
48 .dio = {26, 33, 32},
49 };
50 uint8_t flag_sent = 1;
51 //IMU
52 MPU9250 imu;
53 uint8_t sensorId;
54 float aX_f, aY_f, aZ_f, aSqrt, gX_f, gY_f, gZ_f, mDirection, mX_f, mY_f, mZ_f;
55 //declarar "estrtura imu" mais completa posteriormente
56
57
58 void onEvent (ev_t ev) {
59 Serial.print(os_getTime());
60 Serial.print(": ");
61 switch(ev) {
62 case EV_SCAN_TIMEOUT:
63 Serial.println(F("EV_SCAN_TIMEOUT"));
64 break;
65 case EV_BEACON_FOUND:
66 Serial.println(F("EV_BEACON_FOUND"));
67 break;
68 case EV_BEACON_MISSED:
69 Serial.println(F("EV_BEACON_MISSED"));
70 break;
71 case EV_BEACON_TRACKED:
72 Serial.println(F("EV_BEACON_TRACKED"));
73 break;
74 case EV_JOINING:
75 Serial.println(F("EV_JOINING"));
76 break;
77 case EV_JOINED:
78 Serial.println(F("EV_JOINED"));
79 break;
80 case EV_RFU1:
81 Serial.println(F("EV_RFU1"));
82 break;
83 case EV_JOIN_FAILED:
84 Serial.println(F("EV_JOIN_FAILED"));
85 break;
86 case EV_REJOIN_FAILED:
87 Serial.println(F("EV_REJOIN_FAILED"));
88 break;
89 case EV_TXCOMPLETE:
90 Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
91 if (LMIC.txrxFlags & TXRX_ACK)
92 Serial.println(F("Received ack"));
93 if (LMIC.dataLen) {
94 Serial.println(F("Received "));
95 Serial.println(LMIC.dataLen);
96 Serial.println(F(" bytes of payload"));
97 }
98 // Schedule next transmission
99 os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL),
do_send);
100 break;
101 case EV_LOST_TSYNC:
102 Serial.println(F("EV_LOST_TSYNC"));
103 break;
```

```
104 case EV_RESET:
105 Serial.println(F("EV_RESET"));
106 break;
107 case EV_RXCOMPLETE:
108 // data received in ping slot
109 Serial.println(F("EV_RXCOMPLETE"));
110 break;
111 case EV_LINK_DEAD:
112 Serial.println(F("EV_LINK_DEAD"));
113 break;
114 case EV_LINK_ALIVE:
115 Serial.println(F("EV_LINK_ALIVE"));
116 break;
117 default:
118 Serial.println(F("Unknown event"));
119 break;
120 }
121 }
122
123 void do_send(osjob_t* j){
124 //prepara dados do imu para enviar
125 int16_t ax;
126 uint8_t x_byte[2], *axp = (uint8_t*)&ax;
127 char strnumber[3];
128 //Serial.print("accelX float: ");
129 //Serial.println(aX_f);
130 Serial.print("accelX int: ");
131 ax = (int16_t)(aX_f);
132 Serial.println(ax);
133 sprintf(strnumber, "%d", (ax) );
134
135 Serial.print("accelX bytes: ");
136 Serial.print(strnumber[0]);
137 Serial.print(strnumber[1]);
138 Serial.println(strnumber[2]);
139
140
141
142
143 // Check if there is not a current TX/RX job running
144 if (LMIC.opmode & OP_TXRXPEND) {
145 Serial.println(F("OP_TXRXPEND, not sending"));
146 } else {
147 // Prepare upstream data transmission at the next possible time.
148 LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
149 Serial.println(F("Packet queued"));
150
151 //for(int i = 0;i<8;i++)Serial.print(mydata[i]);
152 }
153 // Next TX is scheduled after TX_COMPLETE event.
154 }
155
156 void float_to_string(double number, char *res, int afterpoint)
157 {
158 // parte inteira em aux
159 int32_t aux = (int32_t) number;
160 // parte fracionária em aux2
161 float aux2 = number - (float) aux;
162
163 // salva a parte inteira do número na string e já coloca o ponto
```

```
164 sprintf(res, "%d.", aux);
165
166 uint8_t loop;
167 for (loop = 0; loop < afterpoint; loop++)
168 {
169 aux2 = aux2 * 10;
170 printf("%f\n", aux2 );
171 aux = (int32_t) aux2;
172 char strnumber[2];
173 sprintf(strnumber, "%d", abs(aux) );
174 strcat(res, strnumber);
175
176 aux2 = aux2 - aux;
177 }
178 }
179 void setup() {
180 Serial.begin(115200);
181 Serial.println(F("Starting"));
182
183 // imu esp32 setup/////
184 #ifdef _ESP32_HAL_I2C_H_ // For ESP32
185 Wire.begin(SDA_PIN, SCL_PIN); // SDA, SCL
186 #else
187 Wire.begin();
188 #endif
189 imu.setWire(&Wire);
190 imu.beginAccel();
191 imu.beginGyro();
192 imu.beginMag();
193 sensorId = imu.readId();
194 ////////////////////
195 #ifdef VCC_ENABLE
196 // For Pinoccio Scout boards
197 pinMode(VCC_ENABLE, OUTPUT);
198 digitalWrite(VCC_ENABLE, HIGH);
199 delay(1000);
200 #endif
201
202 // LMIC init
203 os_init();
204 // Reset the MAC state. Session and pending data transfers will be discarded.
205 LMIC_reset();
206
207 // Set static session parameters. Instead of dynamically establishing a session
208 // by joining the network, precomputed session parameters are be provided.
209 #ifdef PROGMEM
210 // On AVR, these values are stored in flash and only copied to RAM
211 // once. Copy them to a temporary buffer here, LMIC_setSession will
212 // copy them into a buffer of its own again.
213 uint8_t appskey[sizeof(APPSKEY)];
214 uint8_t nwkskey[sizeof(NWKSKEY)];
215 memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
216 memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
217 LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
218 #else
219 // If not running an AVR with PROGMEM, just use the arrays directly
220 LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
221 #endif
222
223
```

```
224 #if defined(CFG_us915)
225 // NA-US channels 0-71 are configured automatically
226 // but only one group of 8 should (a subband) should be active
227 // TTN recommends the second sub band, 1 in a zero based count.
228 //
https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-global_conf.json
229 //LMIC_selectSubBand(1);
230 for (int channel=0; channel<63; ++channel) { // set frequency by
choosing the active channel (this case 914.9Mhz = channel 63)
231 LMIC_disableChannel(channel);
232 }
233 for (int channel=64; channel<72; ++channel) { // set frequency by
choosing the active channel (this case 914.9Mhz = channel 63)
234 LMIC_disableChannel(channel);
235 }
236
237
238 #endif
239
240 // Disable link check validation
241 LMIC_setLinkCheckMode(0);
242
243 // TTN uses SF9 for its RX2 window.
244 LMIC.dn2Dr = DR_SF7;
245
246 // Set data rate and transmit power for uplink (note: txpow seems to be ignored
by the library)
247 LMIC_setDrTxpow(DR_SF7,14);
248
249 // Start job
250 do_send(&sendjob);
251 }
252
253 void loop() {
254 if (flag_sent)
255 {
256 do_send(&sendjob);
257
258
259 flag_sent = 0;
260 }
261
262 os_runloop_once();
263 }
```

6.2 PROGRAMA DA APLICAÇÃO (PROGRAMA 1: CLIENTE MQTT)

```
1 import paho.mqtt.client as mqtt
2 import json
3 import base64
4 import time
5 import iso8601
6 from datetime import datetime
7
8
```

```
9 now = datetime.now()
10 APPEUI = "70B3D57ED001B5DF"
11 APPID = "tccteste2"
12 PSW = 'ttn-account-v2.I9nmkZIDsgIPgoccEs4XdO94YJt-4Wy2rDULg99CKgA'
13 g_id = 0
14 t = time.localtime()
15 data = str(t.tm_mday) + "," + str(t.tm_hour) + "," + str(t.tm_min) + "," +
str(t.tm_sec)
16 experiment = str(t.tm_year) + str(t.tm_mon) + str(t.tm_mday) + str(t.tm_hour) +
str(t.tm_min) + str(t.tm_sec)
17 packets_count = [0]
18 packets_sent = [0]
19 last_counter = []
20 ids = []
21 index = 0
22
23 g1i = 0
24 g2i = 0
25 g3i = 0
26 g4i = 0
27
28 #Call back functions
29
30 # gives connection message
31 def on_connect(mqttc, mosq, obj,rc):
32 print("Connected with result code:"+str(rc))
33 # subscribe for all devices of user
34 mqttc.subscribe('+/devices/+/up')
35
36 # gives message from device
37 def on_message(mqttc,obj,msg):
38 global g1i
39 global g2i
40 global g3i
41 global g4i
42 if msg._topic.count(b"payload"):
43 return
44 x = json.loads(msg.payload.decode('utf-8'))
45 device = x["dev_id"]
46 payload_raw = x["payload_raw"]
47 payload_plain = base64.b64decode(payload_raw)
48 datetime = x["metadata"]["time"]
49 counter = x["counter"]
50 freq = x["metadata"]["frequency"]
51 rssi = x["metadata"]["gateways"][0]["rssi"]
52 snr = x["metadata"]["gateways"][0]["snr"]
53 rf_chain = x["metadata"]["gateways"][0]["rf_chain"]
54 timestamp = x["metadata"]["gateways"][0]["timestamp"]
55 timestring = x["metadata"]["gateways"][0]["time"]
56 gtw_id = x["metadata"]["gateways"][0]["gtw_id"]
57
58 if(g1i < 100):
59 if (gtw_id) == "eui-b827ebffff6155c1":
60 arquivo = open('G1.txt', 'a')
61 arquivo.write(str(rssi) + ",\n")
62 arquivo.close()
63 g1i = g1i + 1
64 if(g1i == 99):
65 print("gateway1 com: " + str(g1i) + "amostras")
66 if(g2i < 100):
```

```
67 if (gtw_id) == "eui-b827ebffff95e2c7":
68 arquivo = open('G2.txt', 'a')
69 arquivo.write(str(rssi) + ",\n")
70 arquivo.close()
71 g2i = g2i + 1
72 if(g2i == 99):
73 print("gateway2 com: " + str(g2i) + "amostras")
74 if(g3i < 100):
75 if (gtw_id) == "eui-b827ebffff4436f1":
76 arquivo = open('G3.txt', 'a')
77 arquivo.write(str(rssi) + ",\n")
78 arquivo.close()
79 g3i = g3i + 1
80 if(g3i == 99):
81 print("gatewa3 com: " + str(g3i) + "amostras")
82 if(g4i < 100):
83 if (gtw_id) == "eui-b827ebffff270e92":
84 arquivo = open('G4.txt', 'a')
85 arquivo.write(str(rssi) + ",\n")
86 arquivo.close()
87 g4i = g4i + 1
88 if(g4i == 99):
89 print("gatewa4 com: " + str(g4i) + "amostras")
90
91
92
93 ts = iso8601.parse_date(timestring)
94
95 ts = time.mktime(ts.timetuple())
96
97 global last_counter,packets_count,packets_sent,index
98 if ids.count(device) == 0:
99 ids.insert(index,device)
100 index = index+1
101
102
103 idx = ids.index(device)
104 if not last_counter:
105 last_counter.insert(idx,counter)
106 if last_counter[idx] != counter:
107 cnt = counter-last_counter[idx]
108 if cnt < 0:
109 cnt = cnt + 65536
110 packets_sent[idx] = packets_sent[idx] + cnt
111 res = "\n" + str(packets_sent[idx]) + "," + str(packets_count[idx]) + "," +
str(snr) + "," + str(rssi) + "," \
112 + str(freq) + "," + str(rf_chain) + "," + str(counter) + "," + str(ts) +
"," + str(payload_plain)
113 fname = str(device) + "_" + str(experiment) + "unique.txt"
114 with open(fname, "a") as myfile:
115 myfile.write(res)
116 myfile.close()
117
118 packets_count[idx] = packets_count[idx] + 1
119 last_counter[idx] = counter
120 #rssi = -1
121 #print(device + ": " + payload_raw + " ==> " + payload_plain + ", RSSI ["+
str(rssi) + "] @" + datetime )
122 res = "\n" + str(packets_sent[idx]) + "," + str(packets_count[idx]) + "," +
str(snr) + "," + str(rssi) + "," \
```

```
123 + str(freq) + "," + str(rf_chain) + "," + str(counter) + "," + str(ts) + ","
+ str(payload_plain)
124 print(res)
125 fname = str(device) + "_" + str(experiment) + ".txt"
126 with open(fname, "a") as myfile:
127 myfile.write(res)
128 myfile.close()
129
130 def on_publish(mosq, obj, mid):
131 print("mid: " + str(mid))
132
133
134 def on_subscribe(mosq, obj, mid, granted_qos):
135 print("Subscribed: " + str(mid) + " " + str(granted_qos))
136
137 def on_log(mqttc,obj,level,buf):
138 print("message:" + str(buf))
139 print("userdata:" + str(obj))
140
141 mqttc= mqtt.Client()
142
143 mqttc.on_connect=on_connect
144 mqttc.on_message=on_message
145
146 mqttc.username_pw_set(APPID, PSW)
147 mqttc.connect("brazil.thethings.network",1883,60)
148
149 run = True
150
151 while run:
152 mqttc.loop_forever()
```

6.3 PROGRAMA DA APLICAÇÃO (PROGRAMA 2: KNN E SVM)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn import svm
4 import pandas as pd
5 from matplotlib import style
6 from sklearn import datasets, svm, metrics
7 from sklearn.preprocessing import StandardScaler
8 from mpl_toolkits.mplot3d import Axes3D
9 from sklearn.metrics import classification_report, confusion_matrix
10 from sklearn.neighbors import KNeighborsClassifier
11 import pickle
12
13 #Array com todas as amostras e classes
14 data10 = np.loadtxt('10classes.csv', delimiter=";", dtype='int')
15
16 # número de amostras
17 n_amostras = 1000
18 # número correspondente a 80% das amostras
19 n_treino = 800
20 def svm_():
21 global data10, n_amostras, n_treino
22 print("Algoritmo SVM")
23 #embaralha array
24 np.random.shuffle(data10)
25 #separa entradas e saídas para treino
```

```
26 Xtrain = data10[:n_treino, :4]
27 ytrain = data10[:n_treino, 4::]
28 ytrain = ytrain.ravel()
29 #separa entradas e saídas para testes
30 Xtest = data10[n_treino:n_amostras, :4]
31 ytest = data10[n_treino:n_amostras, 4::]
32 ytest = ytest.ravel()
33 #aprendizagem e predição do algoritmo
34 clf = svm.SVC(kernel="linear")
35 clf.fit(Xtrain,ytrain)
36 y_pred = clf.predict(Xtest)
37 #salva modelo em formato pickle
38 with open('SVM_10classes','wb') as f:
39 pickle.dump(clf, f)
40
41 #resultados
42 print("Classification report for classifier %s:\n%s\n"
43 % (clf, metrics.classification_report(ytest, y_pred)))
44
45 print("Confusion matrix:\n%s" % metrics.confusion_matrix(ytest, y_pred))
46
47
48 def kkn_10classes():
49 global data10, n_amostras, n_treino
50 print("Algoritmo KNN")
51 #embaralha array
52 np.random.shuffle(data10)
53 #separa entradas e saídas para treino
54 Xtrain = data10[:n_treino, :4]
55 ytrain = data10[:n_treino, 4::]
56 ytrain = ytrain.ravel()
57 #separa entradas e saídas para testes
58 Xtest = data10[n_treino:n_amostras, :4]
59 ytest = data10[n_treino:n_amostras, 4::]
60 ytest = ytest.ravel()
61 #aprendizagem e predição do algoritmo
62 classifier = KNeighborsClassifier(n_neighbors = 5)
63 classifier.fit(Xtrain, ytrain)
64 y_pred = classifier.predict(Xtest)
65 #salva modelo em formato pickle
66 with open('KNN_10classes','wb') as f:
67 pickle.dump(classifier, f)
68
69 #resultados
70 print("Classification report for classifier %s:\n%s\n"
71 % (classifier, metrics.classification_report(ytest, y_pred)))
72
73 print("Confusion matrix:\n%s" % metrics.confusion_matrix(ytest, y_pred))
74 #main program
75 svm_()
76 kkn_10classes()
```