

Universidade Federal de Minas Gerais
Escola de Engenharia
Curso de Graduação em Engenharia de Controle e Automação

Projeto de Final de Curso
Desenvolvimento de um Sistema de Localização em Tempo
Real Utilizando Transmissores e Receptores de
Radiofrequência (RFID)

Felipe Saleimen Nader

Orientador: Prof. Alair Dias Junior, Dr.
Supervisor: Eng. Rafael Padovezi Miranda

Belo Horizonte, Novembro de 2018

Monografia

Desenvolvimento de um Sistema de Localização em Tempo Real Utilizando Transmissores e Receptores de Radiofrequência (RFID)

Monografia submetida à banca examinadora designada pelo Colegiado Didático do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Minas Gerais, como parte dos requisitos para aprovação na disciplina Projeto Final de Curso II.

Belo Horizonte, Novembro de 2018

Resumo

Detectar a localização de ativos e pessoas é um desafio recorrente na Engenharia. Sistemas de Localização em Tempo Real são uma tecnologia em constante estudo e possuem aplicações em diversas áreas, como segurança, controle de processos, logística e saúde. Este trabalho apresenta o projeto de um sistema de localização em tempo real baseado na força do sinal recebido (*Received Signal Strength Indication - RSSI*) e sua avaliação em aplicações de localização *Indoor*. O RSSI, embora apresente como desvantagens uma alta variabilidade e interferência, possui um custo de implementação em hardware menor em comparação a outros métodos. Devido a isso, foram objetivos do projeto avaliar a relação do RSSI com a distância e a posição de transmissores de radiofrequência inseridos em ambientes conhecidos, desenvolver modelos que relacionassem essas grandezas e implementá-los em software. No projeto, foram feitos dois experimentos: Um para avaliar a relação entre a distância e o valor de RSSI e outro para avaliar a relação entre a posição e o valor medido. Com os dados obtidos nos experimentos, alguns modelos baseados em redes neurais foram desenvolvidos e comparados. Após essa análise, o modelo com melhor desempenho foi implementado em um componente de software e utilizado dentro de um Sistema de Localização em Tempo Real completo. Ao final do projeto, o sistema obtido foi capaz de reduzir erros causado pela alta variabilidade do sinal e obter posições com um erro médio de 0.4 metros.

Abstract

People and asset positioning is a recurring challenge in Engineering. Real-Time Location Systems are a technology in constant study and have applications in a variety of areas, including security, process control, logistics and healthcare. This work presents the design of a Received Signal Strength Indication (RSSI) based Real Time Locating System and its evaluation in indoor positioning applications. In despite of having high variability and interference as disadvantages, RSSI has a lower hardware implementation cost compared to other methods. Due to this, the objectives of the project were to evaluate the relationship of RSSI with the distance and position of radio transmitters inserted in known environments, to develop models that relate these values and implement them in software. In the project, two experiments have been made: The first one to evaluate the relationship between distance and RSSI and the other one to evaluate the realtionship between the signal and the position. With the data obtained in the experiments, some models based on neural networks were developed and compared. After this analysis, the best performing model was implemented in a software component and used within a complete Real-Time Location System. At the end of the project, the obtained system was able to reduce errors caused by the high variability of the signal and obtain positions with an average error of 0.4 meters.

Agradecimentos

Em primeiro lugar, agradeço aos meus pais, André e Patrícia, à minha namorada Anny e aos meus amigos, por todo o amor, apoio e incentivo que me deram durante toda a vida.

Agradeço ao Prof. Alair Dias Junior pela orientação, pela paciência e por todo o apoio técnico e acadêmico durante a execução deste trabalho. Agradeço também aos Engenheiros Luis Carlos Dil Junges e Rafael Padovezi Miranda da Radix Engenharia e Software, que me ajudaram durante a execução do projeto, fornecendo tanto os equipamentos necessários quanto seus conhecimentos técnicos.

Por fim, agradeço à Universidade Federal de Minas Gerais, aos colegas de classe e aos professores, por terem contribuído tanto para minha formação como Engenheiro de Controle e Automação.

Sumário

Resumo	i
Abstract	iii
Agradecimentos	v
1 Introdução	1
1.1 Formulação do Problema	2
1.2 Objetivos	2
1.3 Justificativa	3
1.4 Metodologia	3
1.5 Contribuições	4
1.6 Local de Realização	4
2 Fundamentação Teórica	7
2.1 Identificação por Radiofrequência (RFID)	7
2.2 Received Signal Strength Indication (RSSI) e seu uso na Localização Indoor	9
2.3 Redes Neurais Artificiais	11
2.3.1 Estrutura e Características	11
2.3.2 Funções de Ativação	13
2.3.3 Pré-processamento dos dados	14
2.3.4 Treinamento e Avaliação de Desempenho	15
3 Caracterização dos Modelos	17
3.1 Metodologia	17
3.1.1 Experimento 1	18
3.1.2 Experimento 2	20
3.2 Análise e Resultados	21
3.2.1 Experimento 1	22
3.2.2 Experimento 2	25
3.3 Discussão	33
4 Implementação do Software	35
4.1 Metodologia	35
4.2 Arquitetura	35
4.3 Desenvolvimento	37

4.3.1	Recebimento de pacotes UDP	37
4.3.2	Consumidor de pacotes	38
4.3.3	Processador	39
4.3.4	Estimador de Posição	41
4.3.5	Interface Gráfica	42
5	Conclusão	45
	Referências Bibliográficas	46

Lista de Figuras

2.1	Principais componentes de um sistema de identificação por radiofrequência: <i>Transponder</i> e Leitor (FINKENZELLER, 2010)	7
2.2	<i>Transponder</i> passivo, formado por um microchip e um elemento de acoplamento (como uma antena). Quando esses dispositivos entram na área de leitura dos leitores, eles modulam o sinal emitido e o retransmitem com seu código de identificação.(FINKENZELLER, 2010)	8
2.3	<i>Transponder</i> semi-passivo, formado por um microchip, um elemento de acoplamento e uma bateria. Da mesma forma que os transmissores passivos, eles modulam o sinal recebido dos leitores para transmitir sua identificação. Entretanto, uma bateria interna alimenta o chip e permite que ele responda em distâncias maiores .(FINKENZELLER, 2010)	8
2.4	<i>Transponder</i> ativo, formado por um transmissor ativo (TX), um receptor passivo (RX) e uma fonte de alimentação. Para transmitir dados para o receptor, o transmissor é ligado e um campo eletromagnético de alta frequência é gerado.(FINKENZELLER, 2010)	9
2.5	No gráfico da esquerda, é apresentada a comparação entre a distância aproximada pelas equações 1 e 2 e os valores de RSSI brutos medidos. No gráfico da direita essa comparação é feita com valores de RSSI filtrados por médias móveis. (DONG; DARGIE, 2012)	10
2.6	Comparação entre os resultados obtidos com o método de Zhang e Shi (linhas verdes) e com o modelo tradicional (linhas vermelhas). No gráfico da esquerda, são apresentados os resultados obtidos nos eixos x e y, enquanto no gráfico da direita é apresentado um indicador de erro das medições. Podemos perceber que a linha verde (algoritmo) proposto, apresentou melhores resultados que o método tradicional.(ZHANG; SHI, 2012)	10
2.7	Representação de um neurônio utilizando o modelo de McCulloch Pitts.(ZHANG; SHI, 2012)	12
2.8	Estrutura básica de um perceptron multicamadas (ISOKAWA; NISHIMURA; MATSUI, 2012)	13
2.9	(a) Função de etapa binária. (b) Função linear por partes. (c) Função Sigmoide. (HAYKIN; NETWORK, 2004)	14
3.1	A!Prox - Tag	17
3.2	A!Prox - Checkpoint	18
3.3	Representação da montagem feita durante o primeiro experimento	18

3.4	Fotos do experimento 1, onde foram medidos os sinais de RSSI, a distância e o ângulo de incidência entre a <i>tag</i> e o <i>checkpoint</i>	19
3.5	Representação da montagem feita durante o segundo experimento	20
3.6	Fotos do experimento 2, onde foram medidos os sinais de RSSI e a posição da <i>tag</i> móvel.	21
3.7	Dados obtidos no experimento 1 para um ângulo de incidência de 0º. No gráfico superior, é apresentado o valor de RSSI medido para cada medição, enquanto no gráfico inferior são apresentadas as distâncias medidas.	22
3.8	Valores de RSSI filtrados através do filtro de médias móveis das medianas. .	23
3.9	Valores de erro médio quadrático obtido para os conjuntos de treinamento e validação, conforme o número de neurônios utilizados.	24
3.10	Distâncias reais (em preto) e estimadas (em azul) pela rede neural de 4 neurônios para os mesmos valores de RSSI medidos. É possível observar que os resultados obtidos pela rede neural foram insatisfatórios.	25
3.11	Topologia do modelo 1, onde a entrada da rede neural são os valores de RSSI medidos pelos <i>checkpoints</i> 1 e 2, enquanto a saída são as coordenadas X e Y.	26
3.12	Valores do erro médio quadrático obtidos para os conjuntos de treinamento e validação, conforme o número de neurônios utilizado na camada oculta da rede neural para a topologia 1.	26
3.13	Melhor posição estimada para o modelo de topologia 1, utilizando o <i>dataset</i> de treinamento e comparação com a posição real da <i>tag</i>	27
3.14	Topologia do modelo 2, onde a entrada da rede neural são os valores de RSSI medidos pelos <i>checkpoints</i> 1 e 2 para o <i>tag</i> móvel e para os <i>tags</i> de referência, enquanto a saída são as coordenadas X e Y.	27
3.15	Valores de erro médio quadrático obtidos para os conjuntos de treinamento e validação, conforme o número de neurônios utilizado na camada oculta da rede neural para a topologia 2.	28
3.16	Melhor posição estimada para o modelo de topologia 2, utilizando o <i>dataset</i> de treinamento.	29
3.17	Topologia do modelo 3, formado por uma rede neural de topologia 1 e uma rede neural de compensação.	29
3.18	Valores de erro médio quadrático obtidos para os conjuntos de treinamento e validação, conforme o número de neurônios utilizado na camada oculta da rede neural para a topologia 3.	30
3.19	Melhor posição estimada para o modelo de topologia 3, utilizando o <i>dataset</i> de treinamento.	30
3.20	Topologia do modelo 4, formado por uma rede neural de classificação que estima a probabilidade da distância ser maior que 3.5 metros e três redes neurais de topologia 2 para cálculo da posição.	31
3.21	Valores de erro médio quadrático obtidos para os conjuntos de treinamento e validação, conforme o número de neurônios utilizado na camada oculta da rede neural para a topologia 4.	32
3.22	Melhor posição estimada para o modelo de topologia 4, utilizando o <i>dataset</i> de treinamento.	32

4.1 Componentes principais e visão geral do Prox!Locator	36
4.2 Arquitetura de software do Prox!Locator.	38
4.3 Diagrama de classes do <i>ProxLocatorConsumer</i>	39
4.4 Diagrama de classes do armazenamento de mensagens no MessagesProcessor.	40
4.5 Diagrama de classes do MessagesProcessor.	41
4.6 Diagrama de classes do estimador de posição baseado em redes neurais. . .	42
4.7 Interface gráfica do ProxLocator, onde é possível visualizar um gráfico de posição em tempo real, botões de controle e as coordenadas obtidas pelo software.	43

Capítulo 1

Introdução

Detectar a localização de ativos e pessoas é um desafio recorrente na Engenharia. Sistemas de Localização em Tempo Real ou RTLS (do inglês *Real Time Locating Systems*) possuem aplicações em diversas áreas, como segurança, controle de processos, logística e saúde.

Na saúde, por exemplo, é possível utilizar sistemas de localização para estimar a posição de pacientes, médicos e equipamentos, a fim de otimizar a utilização de equipamentos e materiais, facilitar a comunicação entre equipes e monitorar os pacientes.

Em armazéns ou industrias, sistemas de localização também são utilizados para controlar acesso à áreas restritas, monitorar a posição de equipamentos, garantir a segurança dos colaboradores em áreas de risco e controlar a entrada e saída de material.

Em ambientes de grande circulação de pessoas, como feiras e eventos, sistemas de localização em tempo real podem facilitar o processo de organização, proporcionar uma experiência mais agradável para os visitantes e gerar informações sobre o padrão de comportamento dessas pessoas. Em uma feira corporativa, por exemplo, é possível acompanhar quanto tempo os visitantes estão passando em cada estande e analisar os motivos pelos quais os estandes mais visitados estão atraindo a atenção do público.

Nas mais diversas áreas, conhecer a posição dos elementos inseridos em um ambiente pode aumentar o grau de liberdade de sistemas autônomos, permitindo que eles executem tarefas que anteriormente só podiam ser executadas por pessoas, como escolher objetos em uma estante, transportar materiais dentro de um pátio ou desviar de obstáculos móveis.

Com a necessidade de um mundo cada vez mais conectado e o advento da Internet das Coisas, Sistemas de Localização em Tempo Real vêm sendo tema recorrente de pesquisa, com a utilização das mais diversas tecnologias. GPS, Identificação por Radiofrequência (*Radio-Frequency Identification - RFID*), Bluetooth, Ultrassom e Wi-fi são alguns exemplos. Da mesma forma, diversas estratégias de obtenção de dados e cálculo foram desenvolvidas, utilizando variáveis e conceitos diferentes.

Devido a esses aspectos, a Radix, empresa onde o autor atualmente trabalha como pro-

gramador, desenvolveu o A!Prox, uma solução proprietária que utiliza RFID para detectar proximidade de equipamentos e pessoas em plantas industriais, avaliar rotas dos ativos e gerar alarmes. O A!Prox utiliza transmissores de 915Mhz acoplados aos equipamentos e receptores fixos, que recebem as informações dos transmissores e enviam através de protocolo UDP para a rede local. Essas informações são então processadas por software e fornecidas através de aplicações Web e Mobile para os usuários.

Neste Projeto de Fim de Curso, pretende-se analisar os resultados do uso do *hardware* previamente projetado pela empresa para calcular a localização de objetos específicos em duas ou três dimensões, desenvolvendo um Sistema de Localização em tempo real. Com isso, pretende-se obter não só indicadores da eficiência desse *hardware*, mas também um algoritmo implementado para fazer o cálculo da posição.

1.1 Formulação do Problema

Para desenvolver sistemas de localização em tempo real, tipicamente são utilizados transmissores de RFID (muitas vezes chamados de etiquetas ou, do inglês, *tags*) e receptores, responsáveis por obter dados dos transmissores, como potência ou tempo de chegada do sinal. Em seguida, softwares específicos recebem as informações de diversos receptores e utilizam isso para calcular a posição relativa dos *tags*.

Uma estratégia utilizada para esse cálculo é a obtenção do *Received Signal Strength Indication* (RSSI), ou força do sinal recebida. O RSSI informa, em valores relativos, a potência do sinal enviado pelo transmissor. Aplicando essa variável em um modelo matemático de queda de potência por distância, é possível extrair a distância entre o transmissor e receptor.

Esse método não necessita de *hardware* adicional no transmissor, o que garante um custo de produção mais baixo e um tempo de bateria maior. Entretanto, a obtenção de um modelo preciso que relate a distância do sensor e a potência do sinal é um dos grandes desafios de seu uso, visto que a variabilidade de medições geradas normalmente é muito grande. Ao mesmo tempo, para obter estimativas de posição satisfatórias, são necessários diversos receptores posicionados no ambiente.

1.2 Objetivos

Tendo em vista o exposto acima, o projeto tem como objetivo principal avaliar o uso do RSSI enviado por transmissores de RFID 915Mhz para receptores em posições conhecidas e desenvolver um algoritmo de cálculo de posição baseado nisso. Para alcançar esse objetivo, foram propostos cinco objetivos específicos:

1. Medir valores da força do sinal recebido (RSSI), os ângulos e as distâncias medidas entre o transmissor e receptor, com o objetivo de traçar uma curva de relação entre

essas grandezas;

2. Com os valores amostrados, desenvolver um modelo que relate o valor de RSSI com a distância e o ângulo entre o transmissor e receptor;
3. Implementar o modelo desenvolvido em um componente de software reutilizável;
4. Desenvolver um componente de software responsável por calcular a posição de um transmissor, utilizando valores de RSSI recebidos por diversos receptores posicionados em uma sala;
5. Implementar uma estratégia para reduzir o erro de medição causado por obstáculos que possam ser colocados na sala, selecionando as antenas que serão utilizadas para executar os cálculos.

1.3 Justificativa

Embora detectar proximidade possa solucionar diversos problemas da indústria, algumas aplicações mais complexas demandam o cálculo da posição de equipamentos e pessoas para serem bem sucedidas. No A!Prox, mesmo com a possibilidade de obter rotas e eventos através de *checkpoints*, a falta de um mecanismo de cálculo de posição impede seu uso em demandas mais complexas.

Seus transmissores ativos e antenas receptoras foram projetados para ter baixo custo de produção, consumo de energia e uso da rede, objetivo que foi alcançado com a remoção de componentes desnecessários e implementação de um *firmware* compacto de envio e recepção periódica de RSSI. A implementação de um novo *hardware* e *firmware* com envio de informações além do RSSI seria um projeto custoso e pouco escalável, devido à todo esforço que já foi gasto implementando a solução atual e produzindo os equipamentos em escala.

Dessa forma, é interessante avaliar implementações em software para cálculo de posição utilizando os equipamentos já produzidos. Se os resultados forem satisfatórios, o sistema poderá ser acrescentado ao A!Prox. Se os resultados não forem satisfatórios, os dados e o software obtidos poderão justificar a reelaboração do *hardware* ou servir de referência para estudos envolvendo outras tecnologias e estratégias.

1.4 Metodologia

Para alcançar os objetivos determinados, o projeto foi dividido em etapas:

Na primeira etapa, foi montado um ambiente para avaliar o sinal de potência do transmissor (*tag*), o ângulo e a distância entre ele e o receptor (antena), constituído do *hardware* e do sistema de captação de dados. Em seguida, foram realizadas diversas medidas de potência e distância, a fim de montar uma base de dados que pudesse ser utilizada para obter os modelos

matemáticos que relacionassem essas grandezas.

Na segunda etapa, os dados obtidos na etapa um foram analisados e utilizados para obter alguns modelos matemáticos que representassem o sistema. No final dessa etapa, os modelos obtidos foram validados e o melhor modelo foi selecionado.

Na terceira etapa, foi montado um ambiente para avaliar o sinal de RSSI e sua relação com a posição do transmissor no ambiente, a fim de montar uma base de dados da mesma forma que foi feito durante a primeira etapa.

Na quarta etapa, os dados obtidos na terceira etapa foram analisados para obter modelos matemáticos para representar o sistema. Os modelos obtidos foram avaliados, comparados com os gerados na segunda etapa e o melhor foi implementado em um componente de software.

Na quinta etapa, o componente desenvolvido na etapa quatro foi utilizado na implementação de um software em tempo real capaz de estimar a posição da *tag* em um ambiente de antenas posicionadas em locais conhecidos. Esse software deveria lidar não só com o cálculo da posição, mas também com as outras etapas necessárias para esse cálculo, como obtenção de dados, tratamento e apresentação para o usuário.

Por fim, todos os resultados obtidos foram comparados e apresentados nesta monografia.

1.5 Contribuições

Após a conclusão deste trabalho, foi obtido não só um protótipo de sistema de localização em tempo real, mas também um estudo apresentando a viabilidade de utilizar o RSSI para esse tipo de medição. As contribuições obtidas após a conclusão deste projeto foram:

1. A avaliação do uso do RSSI para estimativa de posição e mensuração dos seus erros;
2. Estratégias para redução dos erros relacionados ao uso do RSSI como variável de cálculo de distância;
3. Software desenvolvido para estimar a posição através do recebimento periódico de sinais de potência de transmissores RFID de 915Mhz.

1.6 Local de Realização

O projeto foi desenvolvido dentro do escritório de Belo Horizonte da Radix Engenharia e Software, uma empresa brasileira que atua com projetos de Engenharia, Software e Automação Industrial. Fundada em 2010, possui escritórios no Rio de Janeiro, Belo Horizonte, São Paulo, Volta Redonda, São José dos Campos e Houston. Ela é dividida nas seguintes Unidades de Negócio: Óleo e Gás e Energia, Serviços, Agroindústria e Metais e Mineração. O escritório de Belo Horizonte, em sua maioria, está inserido dentro da UN de Metais

e Mineração, onde executa projetos de Software, Automação e Indústria 4.0 para grandes empresas dessa área de atuação. A localização do escritório em Belo Horizonte é:

Rua Santa Rita Durão, 444 – 6º andar
Funcionários, Belo Horizonte – MG – CEP: 30140-110
+55 31 2531-0444
+55 21 3725-1110

Capítulo 2

Fundamentação Teórica

2.1 Identificação por Radiofrequência (RFID)

Identificação por Radiofrequência, ou *Radio-Frequency Identification* (RFID), é um método de identificação feito através da transmissão e recepção de ondas de rádio. Ao contrário de outros sistemas de identificação, como *Smart Cards*, a transferência de informações no RFID é feita sem contato físico, através de campos magnéticos ou eletromagnéticos. (FINKENZELLER, 2010)

Um sistema de RFID é composto por dois componentes principais, que podem ser vistos na figura 2.1:

1. *Transponders*: Dispositivos que ficam conectados aos objetos que se deseja identificar, também chamados de etiquetas ou *tags*;
2. Leitores: Responsáveis por identificar a presença dos *transponders* em seu raio de detecção.

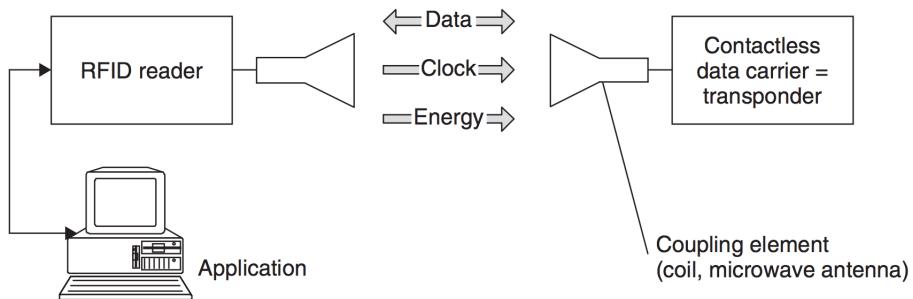


Figura 2.1: Principais componentes de um sistema de identificação por radiofrequência: *Transponder* e Leitor (FINKENZELLER, 2010)

As *tags* normalmente possuem um elemento de acoplamento, um microchip e, opcionalmente, uma fonte de alimentação. As não alimentadas podem ser classificadas como passivas, por serem incapazes de gerar um sinal eletromagnético sem excitação externa. Ao

entrar na área de atuação dos leitores, essas *tags* são alimentadas pela própria energia fornecida pelo leitor, e emitem o sinal indicando sua presença. Na figura 2.2 são apresentados os componentes de uma *tag* passiva.

Quando alimentadas, podem ser classificadas como semi-passivas e ativas. As semi-passivas utilizam sua bateria interna para alimentar o chip modulador, reduzindo a energia necessária para se comunicar com os leitores. Ainda assim, o sinal de identificação é gerado com a energia recebida, obrigando que a *tag* esteja dentro da área de detecção das antenas. Na figura 2.3 são apresentados os componentes de uma *tag* semi-passiva.

Já as ativas, representadas pela figura 2.4, utilizam a fonte de energia interna para alimentar o chip e um transmissor ativo, que emite um sinal eletromagnético de alta frequência. Conceitualmente esse tipo de *tag* não segue o princípio de funcionamento de um sistema de RFID tradicional, mas sim de sistemas de rádio de curto alcance (*short-range radio devices - SRD*). Entretanto, a possibilidade de gerar o próprio sinal de identificação permite que eles possam ser identificados a dezenas de metros.

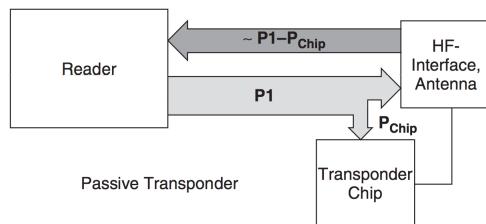


Figura 2.2: *Transponder* passivo, formado por um microchip e um elemento de acoplamento (como uma antena). Quando esses dispositivos entram na área de leitura dos leitores, eles modulam o sinal emitido e o retransmitem com seu código de identificação.(FINKENZELLER, 2010)

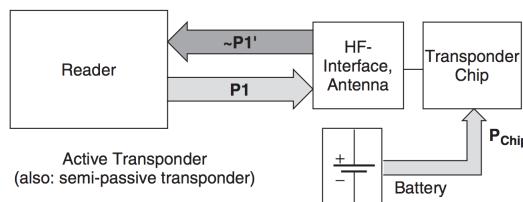


Figura 2.3: *Transponder* semi-passivo, formado por um microchip, um elemento de acoplamento e uma bateria. Da mesma forma que os transmissores passivos, eles modulam o sinal recebido dos leitores para transmitir sua identificação. Entretanto, uma bateria interna alimenta o chip e permite que ele responda em distâncias maiores .(FINKENZELLER, 2010)

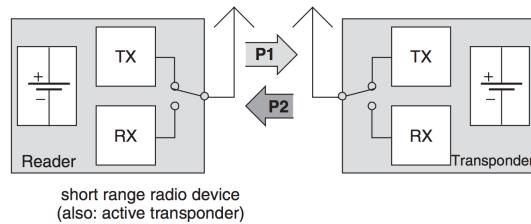


Figura 2.4: *Transponder* ativo, formado por um transmissor ativo (TX), um receptor passivo (RX) e uma fonte de alimentação. Para transmitir dados para o receptor, o transmissor é ligado e um campo eletromagnético de alta frequência é gerado.(FINKENZELLER, 2010)

2.2 Received Signal Strength Indication (RSSI) e seu uso na Localização Indoor

O primeiro passo para se obter a localização através de transmissores e receptores de RFID é obter uma grandeza que possa relacionar a distância entre o transmissor e receptor. Uma das técnicas utilizadas para isso é a obtenção da força do sinal recebida, ou *Received Signal Strength Indication* (RSSI). Essa grandeza representa a relação entre a potência de sinal transmitida e a recebida em um leitor e, em um ambiente sem obstáculos, pode ser idealmente relacionada à distância pela lei do quadrado inverso.

Entretanto, em situações reais, diversos fatores podem influenciar a potência recebida pelo leitor, como reflexão, refração, difração e as condições de propagação do meio. Devido a isso, é necessário determinar uma taxa de decaimento γ para representar as características do meio. Após considerar essa taxa, uma equação simplificada que representa a distância (d) e a potência do sinal (P_r) é dada pela equação 2.1. (DONG; DARGIE, 2012)

$$P_r \propto d^{-\gamma} \quad (2.1)$$

Outro fator importante que deve ser considerado na utilização do RSSI é o alinhamento da polarização entre a antena e o transmissor. Caso esses dois elementos estejam com a mesma orientação, a potência obtida será máxima. Caso eles não estejam alinhados, a perda de potência (L) pode ser expressa pela equação 2.2. (DONG; DARGIE, 2012)

$$L = 20 \log(\cos(\theta)) \quad (2.2)$$

Na prática, os fatores físicos capazes de influenciar a medição acabam gerando uma variabilidade muito grande no sinal de RSSI, inviabilizando a aplicação direta das equações 1 e 2 para obtenção da distância. Em seu artigo, DONG; DARGIE concluem que os resultados obtidos através das equações exponenciais não se mostram confiáveis e que outras variáveis devem ser utilizadas para a obtenção de localização. Na figura 2.5 são exibidos dois gráficos de comparação entre a distância obtida pelas equações 1 e 2 e os valores de RSSI brutos e filtrados. É possível observar que, em ambos os experimentos, os valores obtidos apresentam um erro significativo em comparação ao valor real.

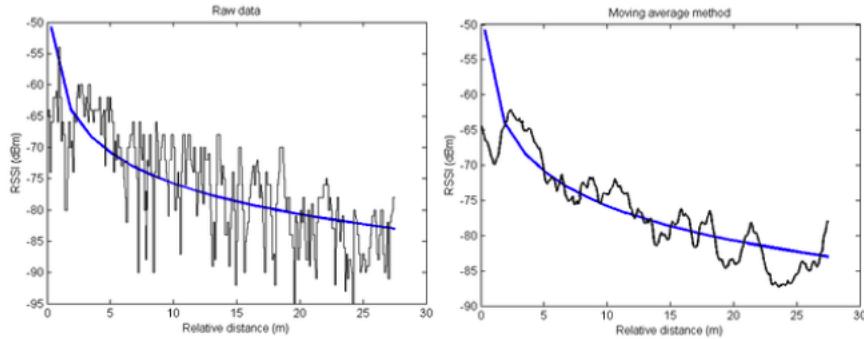


Figura 2.5: No gráfico da esquerda, é apresentada a comparação entre a distância aproximada pelas equações 1 e 2 e os valores de RSSI brutos medidos. No gráfico da direita essa comparação é feita com valores de RSSI filtrados por médias móveis. (DONG; DARGIE, 2012)

Alguns outros pesquisadores obtiveram melhores resultados ao utilizar o RSSI com outras técnicas matemáticas, como redes neurais e modelos estocásticos. Em seu artigo, ZHANG; SHI utilizou Redes Neurais Artificiais e Expansões de Séries de Taylor para obter um modelo de localização. Os resultados obtidos em seu experimento apresentaram uma melhora significativa de precisão em comparação com o método de redução da potência pela distância e podem ser vistos na figura 2.6.

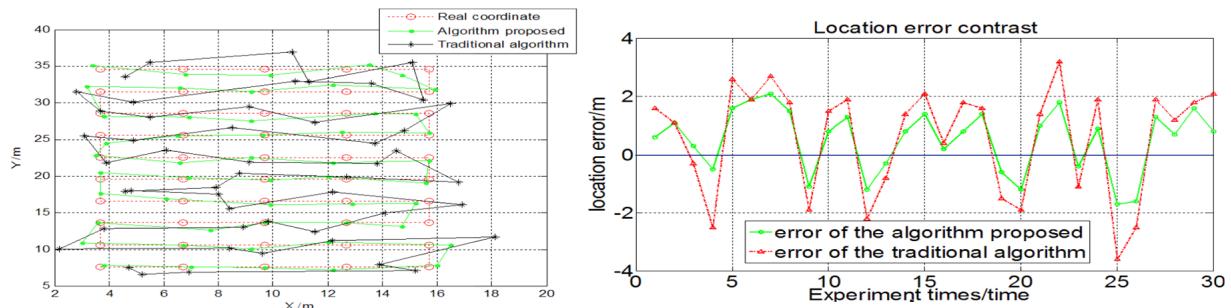


Figura 2.6: Comparação entre os resultados obtidos com o método de Zhang e Shi (linhas verdes) e com o modelo tradicional (linhas vermelhas). No gráfico da esquerda, são apresentados os resultados obtidos nos eixos x e y, enquanto no gráfico da direita é apresentado um indicador de erro das medições. Podemos perceber que a linha verde (algoritmo) proposto, apresentou melhores resultados que o método tradicional.(ZHANG; SHI, 2012)

No artigo LANDMARC: Indoor Location Sensing Using Active RFID (NI et al., 2003), foi discutida a utilização de *tags* de referência para melhorar a medição de posição com o uso da potência do sinal. O método proposto utilizava o sinal medido de transmissores instalados em posições fixas para reduzir os erros causados por condições físicas do

ambiente. Embora utilizasse um hardware bastante limitado, o método proposto apresentou resultados promissores em comparação com sistemas de localização comerciais da época.

Já WANG et al. utilizou redes neurais baseadas em otimização de enxame de partículas (*Particle Swarm Optimization Artificial Neural Network - PSO-ANN*) para localização e comparou os resultados obtidos com o uso de Redes Neurais Artificiais (ANN), Redes Neurais Artificiais utilizando o algoritmo de Backpropagation (BP-ANN) e método de LANDMARC. Os resultados obtidos no experimento apresentaram uma redução do erro durante o uso da técnica de PSO-ANN, mesmo em comparação com o uso de Redes Neurais Artificiais, que já apresenta resultados melhores do que o método de redução da potência pela distância.

Por fim, OTARAWANNA; CHAROENSUK utilizou Redes Neurais de Atraso Temporal (*Time Delay Neural Networks*) para implementar um sistema de localização em tempo real com dispositivos Wireless. Nessa abordagem, uma rede neural foi alimentada não só com os valores instantâneos de sinal de RSSI, mas também com os valores dos últimos períodos medidos. Dessa forma, os erros de medição foram reduzidos em comparação com redes neurais tradicionais, visto que foi considerado também o comportamento temporal do sinal medido.

2.3 Redes Neurais Artificiais

2.3.1 Estrutura e Características

O cérebro humano é um mecanismo capaz de computar informações de forma bem diferente da feita por computadores tradicionais. O cérebro é um computador paralelo complexo e não linear, capaz de reestruturar seus componentes para executar diferentes tarefas de forma muito mais rápida que um computador digital. Para fazer isso, o cérebro possui a habilidade de construir suas próprias regras e equações, conforme passa por experiências durante a vida.

Redes neurais artificiais são estruturas de processamento paralelo constituídas de múltiplas células de processamento interconectadas, chamadas de neurônios. Seu princípio de funcionamento e estrutura são inspirados na forma com que o cérebro reconhece e interpreta informações. Da mesma forma que no cérebro, o conhecimento nas redes neurais é obtido através de treinamento e armazenado dentro da própria rede, em pesos sinápticos. Sua arquitetura permite que ela não só aprenda, mas também generalize, encontrando bons resultados para entradas que não estavam presentes nos dados de treinamento. (HAYKIN; NETWORK, 2004)

As principais vantagens no uso de redes neurais são:

1. Não-linearidade: Redes neurais podem ser lineares e não lineares. Essa característica permite que elas possam lidar com informações que não são lineares, como por exemplo voz;
2. Mapeamento Entrada-Saída: O treinamento de redes neurais não exige que seja obtido um modelo paramétrico para representar o problema a ser resolvido. Em cada iteração,

os algoritmos de treinamento verificam a saída do modelo, comparam ela com a saída desejada e modificam os pesos sinápticos de forma que o erro seja reduzido na próxima iteração;

3. Adaptabilidade: Redes neurais possuem uma capacidade intrínseca de adaptar seus pesos à alterações no ambiente externo através de um novo treinamento. Também é possível projetar a rede neural para que ela altere seus pesos sinápticos em tempo real, a fim de se adaptar a ambientes não estacionários;
4. Confiabilidade de Respostas: Junto com uma saída estimada, uma rede neural também é capaz de fornecer o nível de confiabilidade desse resultado. Essa informação é útil para avaliar a resposta e posteriormente melhorar um modelo;
5. Informação Contextual: Devido à sua estrutura de neurônios interconectados, uma rede neural é capaz de lidar muito bem com o contexto geral das informações fornecidas;
6. Tolerância a falhas: Em casos de falhas de neurônios específicos (em redes implementadas em hardware), o desempenho da rede é reduzido de forma gradativa, devido à sua natureza distribuída;
7. Universalidade de análise e projeto: Devido à sua estrutura formada por pequenas unidades de processamento (neurônios) interligados, as etapas de projeto e análise de redes neurais são relativamente simples. (HAYKIN; NETWORK, 2004)

O modelo mais utilizado em neurônios é chamado de M-P (McCulloch Pitts) e representa uma abstração simplificada de um neurônio real.

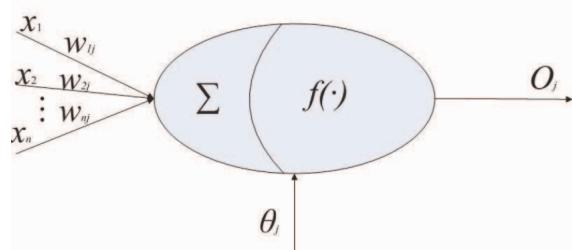


Figura 2.7: Representação de um neurônio utilizando o modelo de McCulloch Pitts.(ZHANG; SHI, 2012)

Nesse modelo, representado na figura 2.7, cada neurônio é composto de N entradas, que são compostas do sinal de saída de um neurônio anterior da rede (x) multiplicado por um peso sináptico (ω). Em seguida, o valor do somatório de todas essas entradas é subtraído de um valor θ e inserido em uma função de ativação. Dessa forma, seu sinal de saída pode ser representado pela equação 2.3.(ZHANG; SHI, 2012)

$$y = f\left(\sum_{i=1}^n \omega_i x_i - \theta\right) \quad (2.3)$$

Os neurônios são comumente organizados em uma estrutura chamada de Perceptron Multicamadas. Nessa estrutura, existe uma camada de entrada, onde as entradas do sistema são conectadas, N camadas ocultas e uma camada de saída. Quanto mais camadas ocultas existirem na rede, maior será a ordem do modelo estimado por ela. Na figura 2.8 é apresentada a estrutura básica de um perceptron multicamadas, onde são vistas a camada de entrada, a camada oculta e a camada de saída.

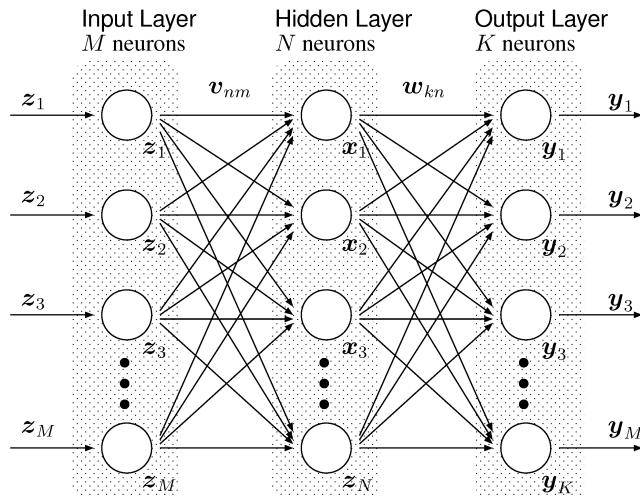


Figura 2.8: Estrutura básica de um perceptron multicamadas (ISOKAWA; NISHIMURA; MATSUI, 2012)

2.3.2 Funções de Ativação

Existem três tipos básicos de função de ativação que são utilizadas em redes neurais artificiais. A primeira delas é a Função de Etapa Binária, que é um classificador baseado em limiar. Essa função tem como saída 1 quando sua entrada é maior ou igual a 0 e 0 quando sua entrada é menor que 0. Essa função pode ser vista na equação 2.4 e seu gráfico característico é apresentado na figura 2.9 (a).

$$f(n) = \begin{cases} 1, & \text{if } n \geq 0 \\ 0, & \text{if } n < 0 \end{cases} \quad (2.4)$$

A segunda função de ativação básica é a função linear por partes, que apresenta comportamento linear dentro de uma área unitária e valores de saturação fora dessa área. Sua equação pode ser vista na equação 2.5 e seu gráfico característico é apresentado na figura 2.9 (b).

$$f(x) = \begin{cases} 1, & \text{if } x \geq 1/2 \\ v + 1/2, & \text{if } -1/2 < x < 1/2 \\ 0, & \text{if } x \leq -1/2 \end{cases} \quad (2.5)$$

A terceira, e mais utilizada, função de ativação básica é a sigmoide, que possui a forma

apresentada na equação 2.6, onde a é o coeficiente de inclinação da reta. Na figura 2.9 (c) é possível ver seu gráfico característico.

$$f(x) = \frac{1}{(1 + \exp(-ax))} \quad (2.6)$$

Esta é uma função não linear e diferenciável, que varia seu sinal de saída entre 0 e 1, em um formato de S. Em alguns casos, é utilizada a função da tangente hiperbólica para representar esse comportamento (equação 2.7).

$$f(x) = \tanh(x) \quad (2.7)$$

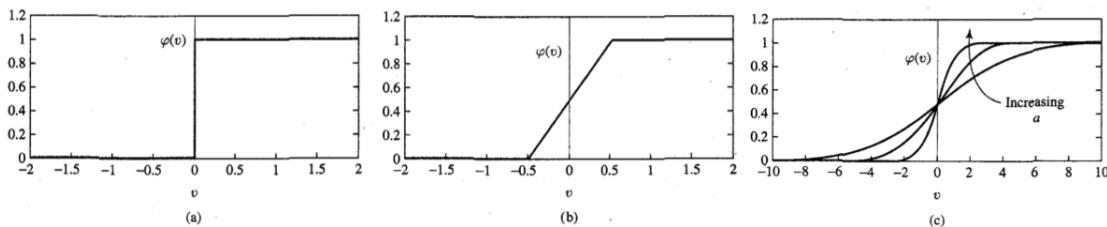


Figura 2.9: (a) Função de etapa binária. (b) Função linear por partes. (c) Função Sigmoidal. (HAYKIN; NETWORK, 2004)

2.3.3 Pré-processamento dos dados

Para treinar e validar uma rede neural através do aprendizado supervisionado, é necessário existir uma massa de dados para o treinamento e verificação da rede neural. Esse conjunto de dados, composto de entradas e saídas, é chamado de *dataset* e deve ser dividido aleatoriamente em dados de treinamento, validação e testes.

O *dataset* de treinamento é utilizado para executar o treinamento do modelo. Com esses dados, a rede neural aprende a generalizar e resolver o problema proposto.

O *dataset* de validação é utilizado para executar uma avaliação imparcial do modelo através de dados que não estão relacionados ao conjunto de treinamento. Esse conjunto de dados permite analisar a capacidade de generalização da rede e resolução de problemas que nunca foram apresentados a ela. Com essa informação, é possível avaliar a arquitetura da rede e os parâmetros utilizados em seu treinamento.

O *dataset* de treinamento é utilizado para avaliar o modelo final obtido, após o treinamento e ajuste da arquitetura da rede. O desempenho da rede neural com esse conjunto de dados é o indicador final da qualidade do modelo e representa seu comportamento no momento de execução.

Também é importante pré-processar os dados para simplificar a etapa de treinamento das redes neurais, através de técnicas que reduzam a complexidade matemática do problema.

A primeira técnica utilizada é a redução de ruído, ou filtragem. Essa etapa consiste em remover distorções nos valores de entrada da rede neural.

A segunda técnica utilizada para isso é a normalização, onde os dados de entrada são ajustados para uma escala comum. Esse ajuste permite que entradas que possuem escalas maiores não influenciem mais no resultado da rede do que entradas com escalas pequenas. (LI; CHEN; HUANG, 2000)

Por fim, uma terceira técnica utilizada é extração de características, através de conhecimentos específicos do problema que precisa ser resolvido. Essa técnica visa reduzir a dimensão dos dados de entrada, analisando-os previamente e removendo dados que não estão relacionados ao problema. (LI; CHEN; HUANG, 2000)

2.3.4 Treinamento e Avaliação de Desempenho

Algoritmos de treinamento são utilizados para ajustar a rede neural a fim de mapear corretamente entradas e saídas. Existem diversos tipos de algoritmos utilizados, cada qual com suas vantagens e desvantagens.

O primeiro passo do treinamento de uma rede neural é definir uma função que represente o nível de precisão de suas estimativas. Essa função é chamada de Função de Custo, e pode ser representada pelo somatório do quadrado dos erros encontrados dividido por dois (Equação 2.7). Essa função é normalmente escolhida por ser convexa, o que garante que ela não tenha mínimos locais e facilite o processo de treinamento.

$$f(e) = \sum \frac{1}{2} e^2 \quad (2.8)$$

Com a função de custo definida, o treinamento da rede se torna um problema de otimização, onde os pesos sinápticos devem ser ajustados para que a função de custo alcance seu valor mínimo. Embora seja um conceito simples, calcular todas as possibilidades da função de custo se torna um problema computacionalmente caro, visto que a complexidade do problema aumenta exponencialmente com a quantidade de neurônios na rede.

Dessa forma, diversos algoritmos de treinamento procuram diminuir a complexidade desse problema de otimização. Um dos principais conceitos utilizados dentro desses algoritmos é o de gradiente descendente, que consiste em calcular o gradiente da função de custo em determinados pontos e definir o caminho para a região ótima conforme o gradiente se aproxima de zero.

O conceito de gradiente descendente é utilizado dentro do método de retropropagação, em inglês *backpropagation*. Nesse método, a rede neural tem seus pesos inicializados aleatoriamente e sua saída é calculada, na etapa chamada de *forward pass*. Em seguida, na etapa de *backpropagation* o gradiente de cada um dos neurônios é calculado, utilizando a regra da cadeia. Os pesos desses neurônios são então atualizados conforme o valor de gradiente

obtido, a fim de que o valor da função de custo seja reduzido na próxima iteração. Após diversas etapas de *forward pass* e *backpropagation*, os valores dos gradientes em cada neurônio estarão próximos de zero, identificando que os parâmetros da rede neural atingiram seus pontos ótimos. (HAYKIN; NETWORK, 2004)

Diversos outros algoritmos de treinamento foram desenvolvidos para redes neurais, tais como Gradiente conjugado, retropropagação resiliente de Quasi-Newton, otimização por enxame de partículas, Levenberg-Marquardt, entre outros. A seleção do algoritmo a ser utilizado acaba se tornando uma tarefa complexa, visto que cada um pode se comportar melhor em alguns casos e pior em outros.(CÖMERT; KOCAMAZ, 2017)

Além de utilizar funções de custo para treinar as redes neurais, é interessante obter um estimador capaz de avaliar o desempenho de uma rede neural. Um indicador comumente utilizado é o Erro Quadrático Médio, ou *Mean Square Error* (MSE). O MSE é definido como a média da diferença entre o valor do estimador e do parâmetro ao quadrado, representado pela equação 2.8. Durante o desenvolvimento do projeto, o erro quadrático médio será um indicador utilizado para avaliar a qualidade dos resultados obtidos. (MORETTIN; BUSSAB, 2017)

$$MSE = E(T - \theta)^2 \quad (2.9)$$

Capítulo 3

Caracterização dos Modelos

3.1 Metodologia

Durante a execução do projeto, foram feitos dois experimentos de coleta de dados utilizando os checkpoints e tags do A!Prox. O A!Prox é um sistema composto de dois dispositivos principais: As *tags* e os *checkpoints*.

As *tags*, que podem ser vistos na figura 3.1, são pequenos dispositivos microcontrolados que enviam sinais de radiofrequência 915 Mhz a cada segundo, indicando seu nível de bateria e identificação. Já os *checkpoints*, apresentados na figura 3.2, são dispositivos responsáveis por identificar a presença dos *tags* e enviar suas informações para o sistema de localização. Eles contêm uma antena multidirecional que detecta os *tags* e um sistema microcontrolado, que transmite os dados recebidos dentro da rede local através do protocolo UDP. Nos pacotes enviados pelo checkpoint, são contidos o identificador do *tag*, seu nível de bateria e o valor de RSSI do sinal recebido.



Figura 3.1: A!Prox - Tag



Figura 3.2: A!Prox - Checkpoint

Os *checkpoints* são conectados na rede local através de cabos de rede RJ45 e alimentados por um cabo de força de 110/220V. A *tag* é alimentada por uma bateria interna, que pode ser substituída ao término de seu uso.

Para gerar os modelos de localização, os *checkpoints* foram instalados em posições fixas e conectados em uma rede LAN isolada, na qual transmitiam os pacotes UDP. Nessa mesma rede, foi conectado um computador responsável por capturar os dados do experimento.

O primeiro experimento tinha o objetivo de analisar a relação entre os sinais de RSSI e a distância entre o *tag* e *checkpoint*, enquanto o segundo tinha o objetivo de relacionar esse sinal com a posição de um *tag* no ambiente. Esses experimentos também foram utilizados para auxiliar na modelagem das redes neurais do sistema e gerar os *datasets* de treinamento, teste e validação.

3.1.1 Experimento 1

No experimento 1, foi utilizado um *checkpoint* mantido em local fixo e *tags* posicionados em distâncias de 0.5 até 10 metros por intervalos de 1 minuto, conforme o diagrama apresentado na figura 3.3.



Figura 3.3: Representação da montagem feita durante o primeiro experimento

Foram feitas quatro séries de medição, com os ângulos de incidência entre a antena de

0°, 30° e 60°. Esse experimento resultou em um conjunto de dados relacionando distância, ângulo de incidência e valor do RSSI, que foi utilizado posteriormente para projeto de um modelo que relacionasse essas grandezas. Na figura 3.4 são apresentadas fotos da montagem e realização do experimento.

Esse experimento apresentou as seguintes etapas:

1. Instalação do *checkpoint* na rede local e alimentação;
2. Demarcação de pontos a cada 0,5 metros no chão, para posicionamento dos *tags*;
3. Posicionamento do *checkpoint* em uma posição fixa, com o ângulo de incidência desejado;
4. Posicionamento do *tag* no ponto de medição;
5. Captação dos dados de RSSI, distância e ângulo por software durante 60 segundos;
6. Alteração da posição do *tag*;
7. Repetição das etapas 4, 5 e 6 até a conclusão dos pontos de medição;
8. Repetição dos pontos 3 a 7 até a conclusão dos ângulos de medição.



Figura 3.4: Fotos do experimento 1, onde foram medidos os sinais de RSSI, a distância e o ângulo de incidência entre a *tag* e o *checkpoint*.

3.1.2 Experimento 2

O experimento 2 teve como objetivo relacionar os valores de RSSI com a posição bidimensional da *tag*. Um ambiente de 3.7 x 7 metros foi demarcado e, dentro dele, foram posicionados dois *checkpoints* e duas *tags* em posições fixas, conforme a montagem apresentada na figura 3.5.

As *tags* em posição fixa tiveram a função de ser elementos de referência, responsáveis por reduzir a influência de condições físicas do ambiente nos cálculos, conforme estudado por NI et al.. Já a terceira *tag* foi posicionada em cada um dos pontos de medição e mantida neles durante cerca de 60 segundos, enquanto os valores de RSSI recebidos pelos dois *checkpoints* foram registrados.

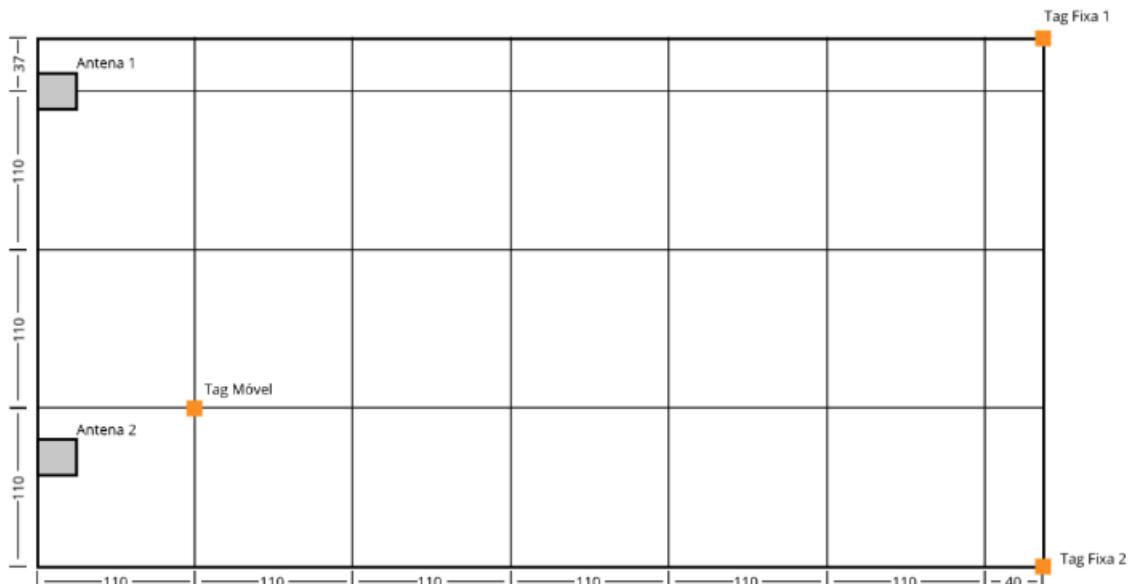


Figura 3.5: Representação da montagem feita durante o segundo experimento

Esse experimento resultou em um conjunto de dados que relaciona a posição da *tag* móvel com os valores de RSSI medidos pelas duas antenas, tanto para ela quanto para as duas *tags* fixas. O conjunto de dados foi, posteriormente, utilizado para o desenvolvimento do modelo de localização final contido neste projeto. Também foi gerado um segundo *dataset* para validação, onde a *tag* foi posicionada em posições aleatórias dentro do espaço de testes. Na figura 3.6 são apresentadas fotos do experimento, mostrando os *checkpoints* e *tags* posicionados.

Esse experimento apresentou as seguintes etapas:

1. Instalação dos *checkpoints* na rede local e alimentação;
2. Demarcação de pontos equidistantes nos eixos X e Y, a cada 110 cm. Por limitações de espaço no ambiente de testes, também foram demarcados pontos com distâncias menores, de cerca de 40 cm.

3. Posicionamento dos *checkpoints* nas posições (0, 330) e (0, 74);
4. Posicionamento dos *tags* fixos nas posições (700, 0) e (700, 370);
5. Posicionamento do *tag* na posição de interesse e medição dos valores de RSSI durante cerca de 60 segundos;
6. Alteração da posição do *tag*
7. Repetição das etapas 5 e 6 até a conclusão dos pontos de medição.



Figura 3.6: Fotos do experimento 2, onde foram medidos os sinais de RSSI e a posição da *tag* móvel.

3.2 Análise e Resultados

Com os conjuntos de dados obtidos nos dois experimentos, o próximo passo do projeto consistiu em analisar os dados obtidos e obter modelos matemáticos, tanto para estimar a distância pelo RSSI, quanto para estimar a posição.

3.2.1 Experimento 1

O objetivo da análise do experimento 1 foi obter um modelo que relacionasse os valores de RSSI (entrada) com a distância entre a *tag* e o *checkpoint* (saída).

Ao imprimir os dados obtidos durante o experimento 1 em um gráfico de linhas, foi possível observar que os valores de RSSI apresentavam um ruído muito grande, como pode ser observado na figura 3.7. Dessa forma, foi necessário implementar um filtro para que essa variabilidade fosse atenuada.

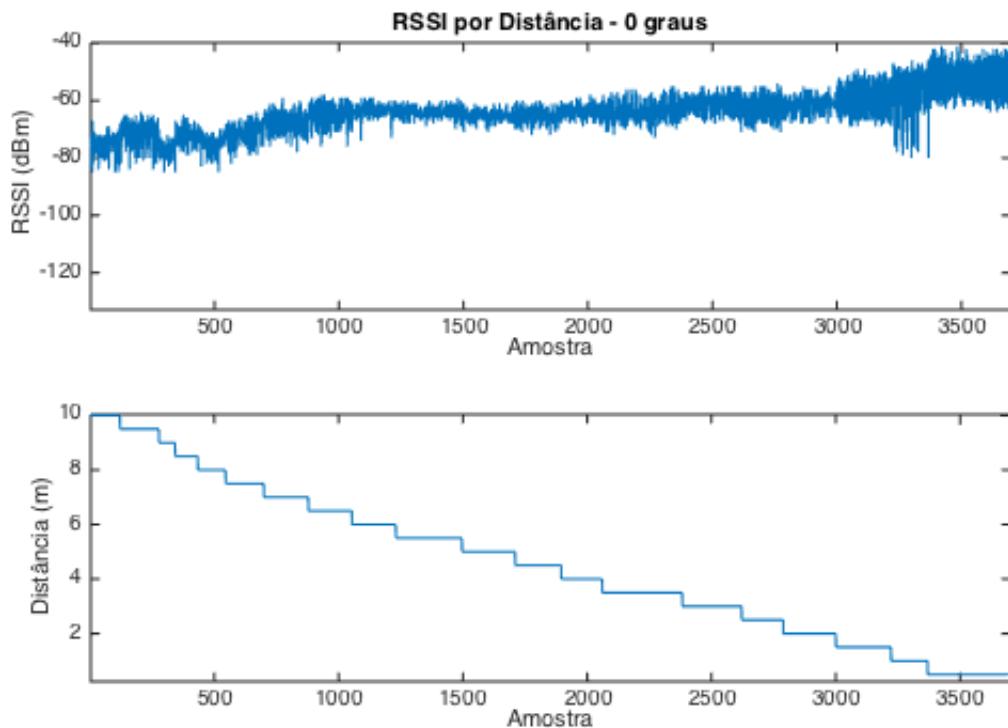


Figura 3.7: Dados obtidos no experimento 1 para um ângulo de incidência de 0º. No gráfico superior, é apresentado o valor de RSSI medido para cada medição, enquanto no gráfico inferior são apresentadas as distâncias medidas.

O filtro escolhido para executar essa função foi baseado na média móvel das medianas do sinal, seguindo o seguinte algoritmo:

1. Em uma janela móvel de 3 períodos, é extraída a mediana do sinal;
2. Em uma janela móvel de 7 períodos, é extraída a média ponderada das 7 medianas calculadas, onde o valor disponível mais recente possui peso 2;
3. Caso não existam valores disponíveis para a filtragem, o filtro retorna o valor de entrada.

Após a filtragem, o sinal apresentou uma redução significativa do ruído e manteve seu comportamento de baixa frequência, o que era desejado para que pudesse ser extraído um

modelo de predição. Na figura 3.8, são apresentados os valores filtrados através do filtro de médias móveis das medianas.

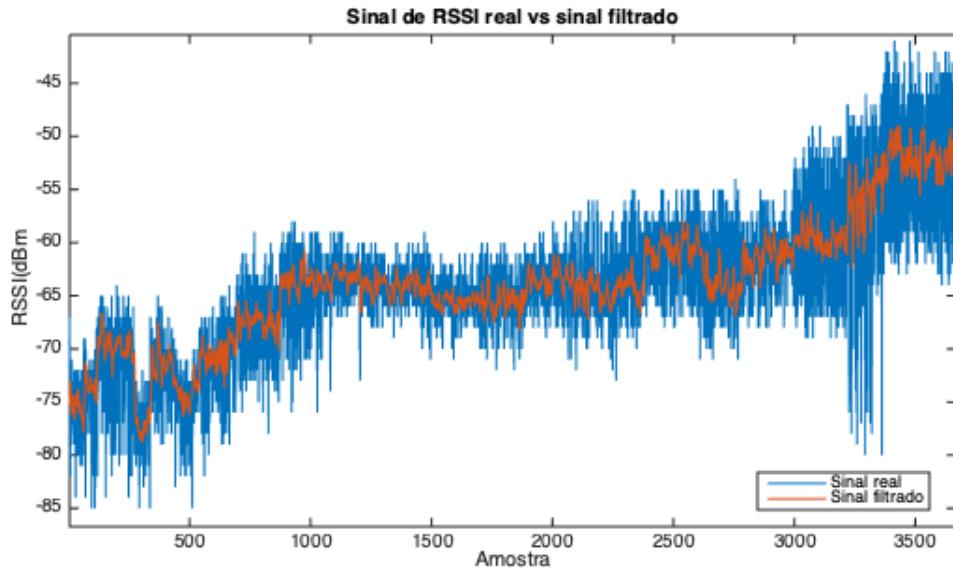


Figura 3.8: Valores de RSSI filtrados através do filtro de médias móveis das medianas.

Em seguida, foi necessário colocar os valores de entrada e saída em escalas normalizadas, de forma a facilitar a obtenção da rede neural. A normalização foi feita dividindo-se os valores dos vetores de entrada e saída por sua norma euclidiana. Os valores de norma obtidos para os vetores de entrada e saída foram armazenados, para que fosse possível converter e desconverter os valores futuramente.

Com os vetores de entrada filtrados e normalizados, o próximo passo foi dividir o *dataset* em conjuntos de treinamento (70%), validação (15%) e testes (15%) e treinar as redes neurais. Além disso, foi utilizado um *dataset* extra para uma segunda validação da rede neural e garantia de que o modelo obtido fosse capaz de generalizar entradas não previstas.

Por se tratar de um problema de entrada/saída, foi utilizada uma *Feed Foward Neural Network* com apenas uma camada oculta. O número de neurônios nessa camada foi definido experimentalmente através da avaliação do erro médio quadrático gerado. Foram utilizados dois *datasets*:

1. Valores de RSSI e distância medidos para um ângulo de 0º: Utilizado como *dataset* de treinamento.
2. Valores de RSSI e distância medidos para um ângulo de 30º: Utilizado como *dataset* de validação.

O treinamento e avaliação das redes neurais seguiu o seguinte algoritmo:

1. Treina a rede neural utilizando o *dataset* de treinamento e calcula seu erro quadrático médio (MSE);
2. Executa a rede neural obtida com os dados do *dataset* de validação e retorna seu erro quadrático médio;
3. Salva os valores de MSE obtidos nos dois *datasets* e avalia a qualidade da rede. Um erro grande nos dois *datasets* indica uma rede neural com desempenho ruim, enquanto um erro pequeno no *dataset* de treinamento e grande no *dataset* de validação indica que a rede neural apresentou *overfitting*, que é a incapacidade de generalizar resultados para entradas não esperadas.

Com essa análise, foi possível observar que uma rede neural com 4 neurônios na camada oculta apresentou os melhores resultados, o que pode ser observado visualmente na figura 3.9, que relaciona o número de neurônios utilizado com os resultados de treinamento e validação.

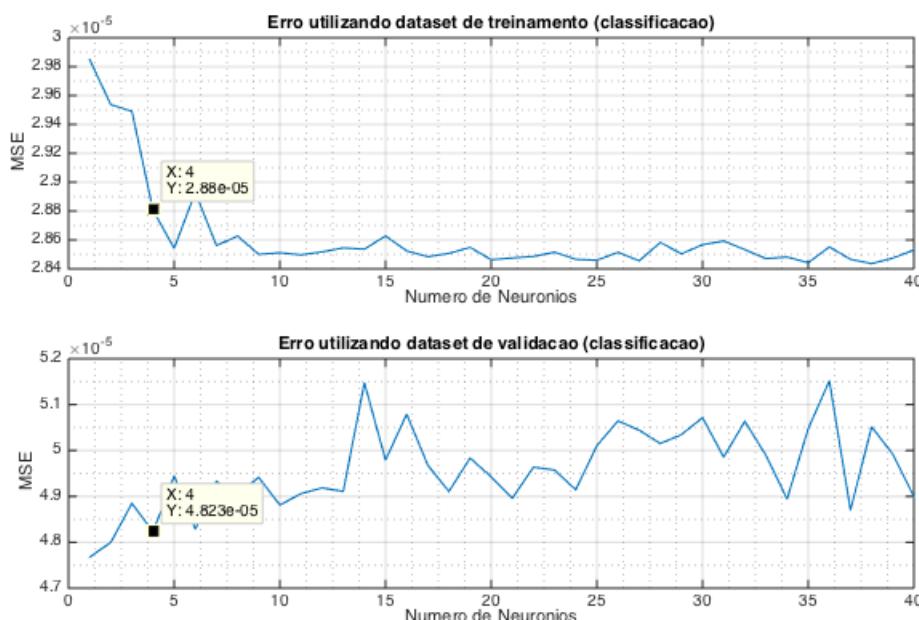


Figura 3.9: Valores de erro médio quadrático obtido para os conjuntos de treinamento e validação, conforme o número de neurônios utilizados.

Ao avaliar a rede neural obtida, o MSE encontrado após a desnormalização foi de 3.0035 m, o que indica que as previsões de distância obtidas pela rede apresentavam em média 1.73 metros para mais ou para menos. Também foi observado que a variabilidade do sinal de saída era muito grande e existia a necessidade de filtrar o resultado obtido. As distâncias calculadas e os valores reais de medição podem ser visualizados na figura 3.10.

Com essa análise, foi possível concluir que o uso dos valores de RSSI medidos apenas por um *checkpoint* não resultavam em um cálculo de distância preciso e seria necessário combinar os valores de RSSI medidos por mais de um *checkpoint* para obter resultados melhores.

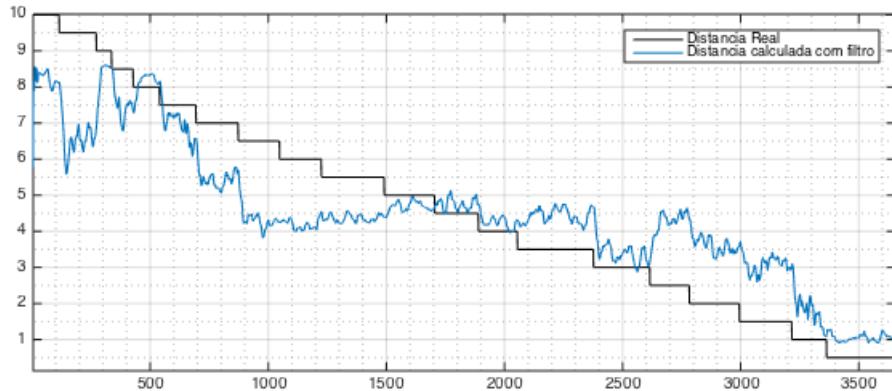


Figura 3.10: Distâncias reais (em preto) e estimadas (em azul) pela rede neural de 4 neurônios para os mesmos valores de RSSI medidos. É possível observar que os resultados obtidos pela rede neural foram insatisfatórios.

3.2.2 Experimento 2

O objetivo do experimento 2 foi gerar um modelo capaz de estimar a posição da *tag* móvel através dos valores de RSSI medidos por dois *checkpoints* e dos valores de RSSI de duas *tags* de referência. Da mesma forma que no experimento 1, os dados obtidos foram filtrados e normalizados. Em seguida, eles foram organizados em vetores de entrada e saída, com a seguinte configuração:

Entrada:

1. RSSI da *tag* móvel, medido pelo *checkpoint* 1;
2. RSSI da *tag* móvel, medido pelo *checkpoint* 2;
3. RSSI da *tag* fixa 1, medido pelo *checkpoint* 1;
4. RSSI da *tag* fixa 1, medido pelo *checkpoint* 2;
5. RSSI da *tag* fixa 2, medido pelo *checkpoint* 1;
6. RSSI da *tag* fixa 2, medido pelo *checkpoint* 2.

Saída:

1. Posição X da *tag* móvel;
2. Posição Y da *tag* móvel.

Após o pré-processamento dos dados, foram obtidas algumas redes neurais que relacionassem os sinais de RSSI medidos e a posição do *tag* móvel em 4 topologias diferentes. Para cada topologia, foram treinadas redes neurais com 1 a 40 neurônios e seu erro médio (mse) foi avaliado tanto para o *dataset* de treinamento quanto para o *dataset* de validação.

Topologia 1

A topologia 1, apresentada na figura 3.11 tem como entrada os valores medidos de RSSI da *tag* móvel para os *checkpoints* 1 e 2, e saída as coordenadas X e Y da *tag*. Durante o treinamento e execução das redes com os *datasets* de treinamento e validação, foram obtidos os valores de MSE apresentados na figura 3.12. Ao analisar esses dados, foi observado que a rede neural com o melhor desempenho apresentava 21 neurônios na camada oculta.

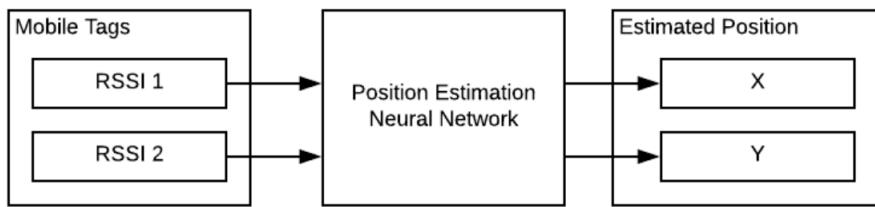


Figura 3.11: Topologia do modelo 1, onde a entrada da rede neural são os valores de RSSI medidos pelos *checkpoints* 1 e 2, enquanto a saída são as coordenadas X e Y.

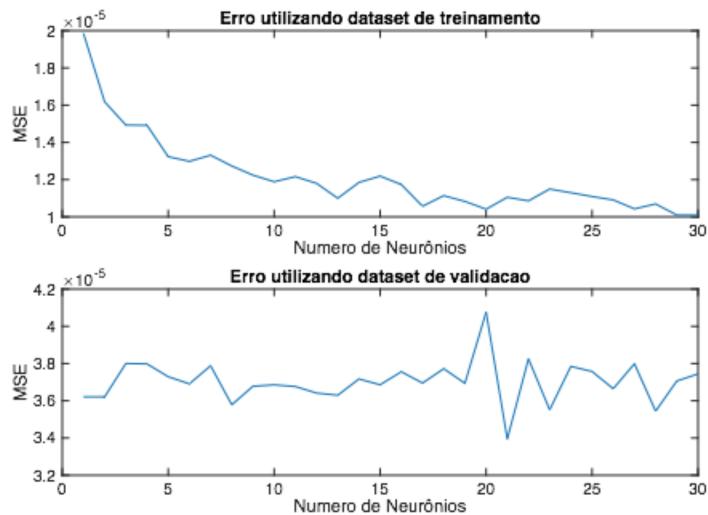


Figura 3.12: Valores do erro médio quadrático obtidos para os conjuntos de treinamento e validação, conforme o número de neurônios utilizado na camada oculta da rede neural para a topologia 1.

A rede neural encontrada apresentou um valor de MSE após a desnormalização de 0.799 m, o que indica que as previsões de posição obtidas pela rede apresentavam em média 0.89 metros para mais ou para menos. As melhores posições estimadas pela rede podem ser visualizadas na figura 3.13.

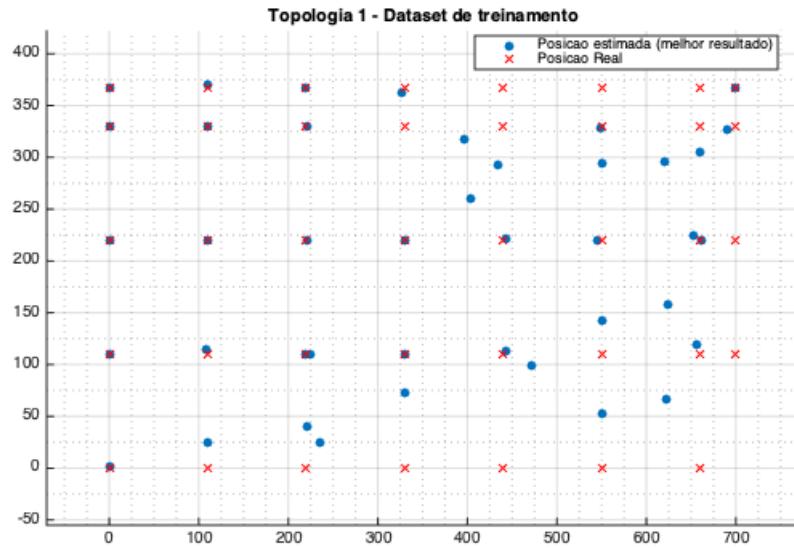


Figura 3.13: Melhor posição estimada para o modelo de topologia 1, utilizando o *dataset* de treinamento e comparação com a posição real da *tag*.

Topologia 2

A topologia 2, apresentada na figura 3.14 tem como entrada os valores medidos de RSSI dos *tags* fixos e móvel para os *checkpoints* 1 e 2, e saída as coordenadas X e Y da *tag*. Durante o treinamento e execução das redes com os *datasets* de treinamento e validação, foram obtidos os valores de MSE apresentados na figura 3.15. Ao analisar esses dados, foi observado que a rede neural com o melhor desempenho apresentava 16 neurônios na camada oculta.

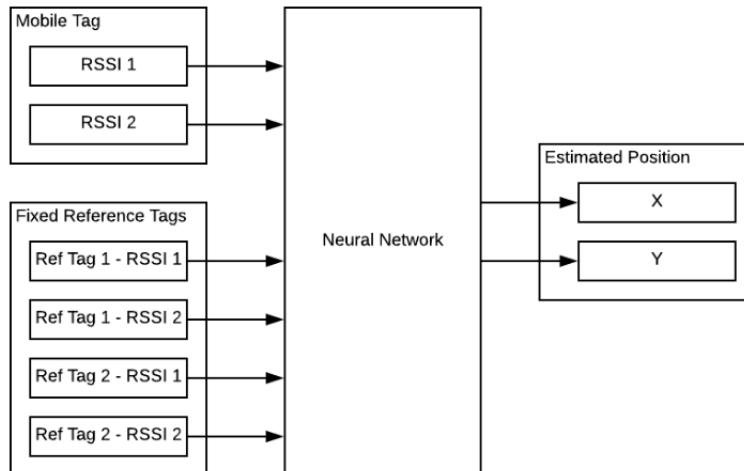


Figura 3.14: Topologia do modelo 2, onde a entrada da rede neural são os valores de RSSI medidos pelos *checkpoints* 1 e 2 para o *tag* móvel e para os *tags* de referência, enquanto a saída são as coordenadas X e Y.

A rede neural encontrada apresentou um valor de MSE após a desnormalização de 0.4053 m, o que indica que as previsões de posição obtidas pela rede apresentavam em média 0.63 metros para mais ou para menos. As melhores posições estimadas pela rede podem ser visualizadas na figura 3.16.

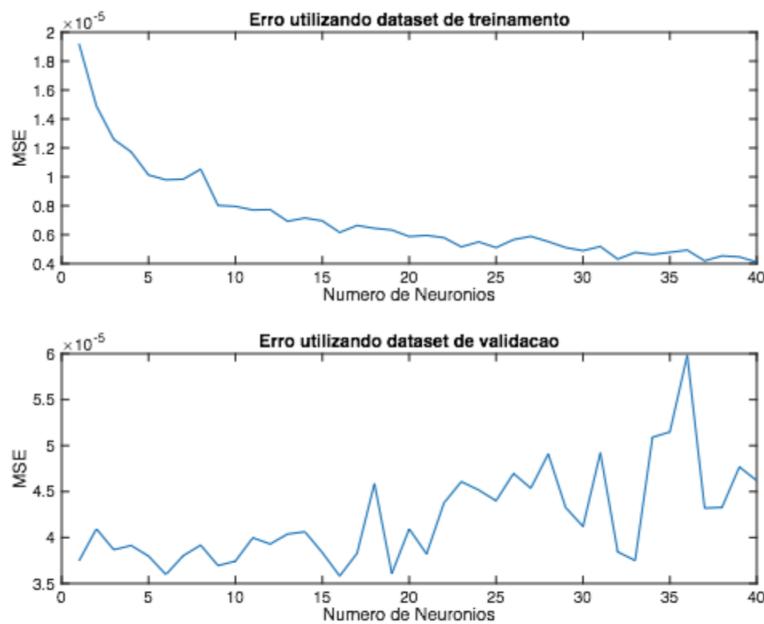


Figura 3.15: Valores de erro médio quadrático obtidos para os conjuntos de treinamento e validação, conforme o número de neurônios utilizado na camada oculta da rede neural para a topologia 2.

Topologia 3

A topologia 3, apresentada na figura 3.17 tem como entrada a posição estimada por uma rede da Topologia 1 que, em seguida, passa por uma rede neural de compensação. A rede de compensação tem como entrada a posição estimada pela rede 1 e os os valores de RSSI das tags de referência.

Durante o treinamento e execução das redes com os *datasets* de treinamento e validação, foram obtidos os valores de MSE apresentados na figura 3.18. Ao analisar esses dados, foi observado que a rede neural com o melhor desempenho apresentava 21 neurônios na camada oculta da rede de posicionamento e 9 neurônios na camada oculta da rede de compensação.

O modelo encontrado apresentou um valor de MSE após a desnormalização de 0.5436 m, o que indica que as previsões de posição obtidas pela rede apresentavam em média 0.73 metros para mais ou para menos. As melhores posições estimadas pela rede podem ser visualizadas na figura 3.19.

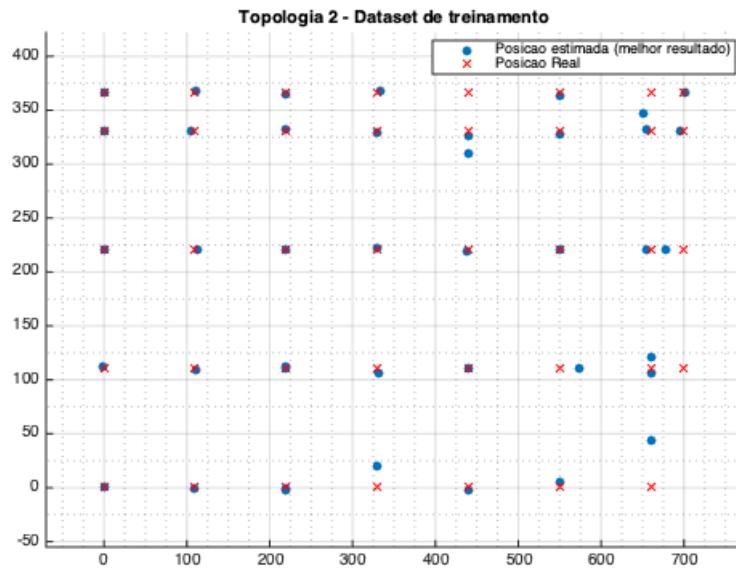


Figura 3.16: Melhor posição estimada para o modelo de topologia 2, utilizando o *dataset* de treinamento.

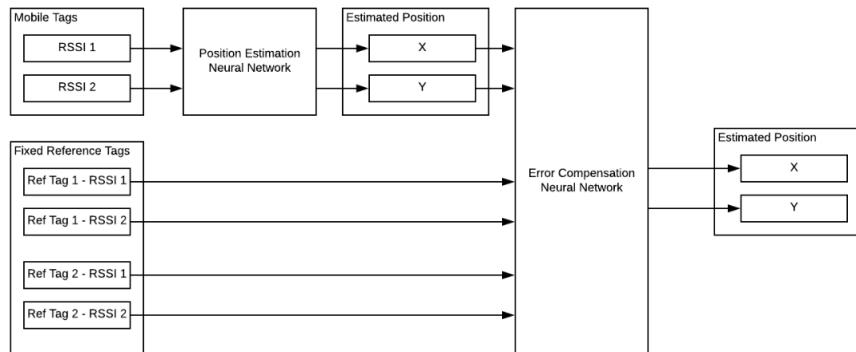


Figura 3.17: Topologia do modelo 3, formado por uma rede neural de topologia 1 e uma rede neural de compensação.

Topologia 4

Ao Analisar os resultados das topologias 1 a 3, foi possível perceber que a precisão dos algoritmos de previsão ao determinar a componente Y da posição diminuiu muito em posições X maiores que 3.5 metros, embora a precisão de X não tenha diminuído. Dessa forma, na topologia 4 é proposto o modelo da figura 3.20, formado por uma rede neural de classificação para determinar a probabilidade da distância ser maior que 3.5 metros e três redes neurais para determinação da posição do *tag* dependendo dessa probabilidade, conforme o seguinte algoritmo:

1. Calcula a probabilidade da distância ser maior que 3.5 metros através da rede neural

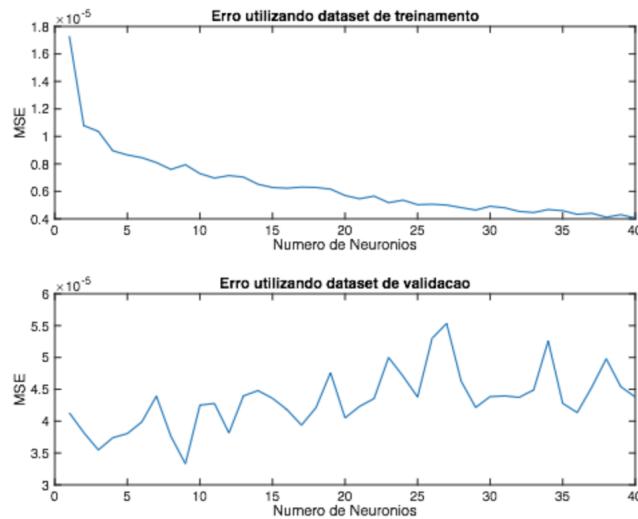


Figura 3.18: Valores de erro médio quadrático obtidos para os conjuntos de treinamento e validação, conforme o número de neurônios utilizado na camada oculta da rede neural para a topologia 3.

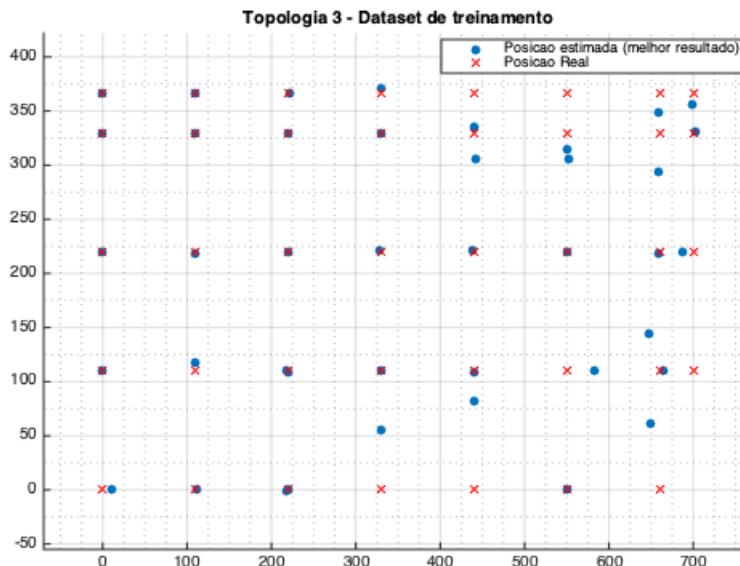


Figura 3.19: Melhor posição estimada para o modelo de topologia 3, utilizando o *dataset* de treinamento.

de classificação;

2. Se a probabilidade for menor que 0.35, é utilizada a rede neural de topologia 2 treinada para distâncias menores que 3.5 metros;
3. Se a probabilidade tiver valor entre 0.35 e 0.65, é utilizada uma rede neural de topologia

- 2 treinada para todas as distâncias;
4. Se a probabilidade tiver valor maior que 0.65, é utilizada uma rede neural de topologia 2 treinada para distâncias maiores que 3.5 metros.

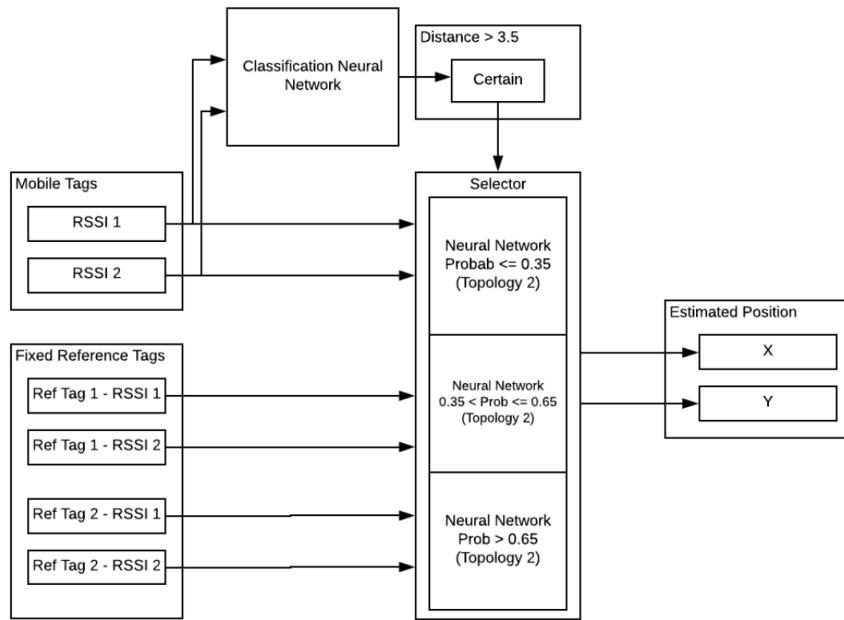


Figura 3.20: Topologia do modelo 4, formado por uma rede neural de classificação que estima a probabilidade da distância ser maior que 3.5 metros e três redes neurais de topologia 2 para cálculo da posição.

Durante o treinamento e execução das redes com os *datasets* de treinamento e validação, foram obtidos os valores de MSE apresentados na figura 3.21. Ao analisar esses dados, foi observado que a rede neural com o melhor desempenho apresentava 26 neurônios na rede neural de classificação e 16 neurônios nas redes neurais de cálculo de posição.

O modelo encontrado apresentou um valor de MSE após a desnormalização de 0.429 m, o que indica que as previsões de posição obtidas pela rede apresentavam em média 0.65 metros para mais ou para menos. As melhores posições estimadas pela rede podem ser visualizadas na figura 3.22. Embora esse modelo tenha apresentado um erro quadrático médio similar à topologia 2, foi possível observar que, para distâncias maiores que 3.5 metros, os resultados foram mais precisos.

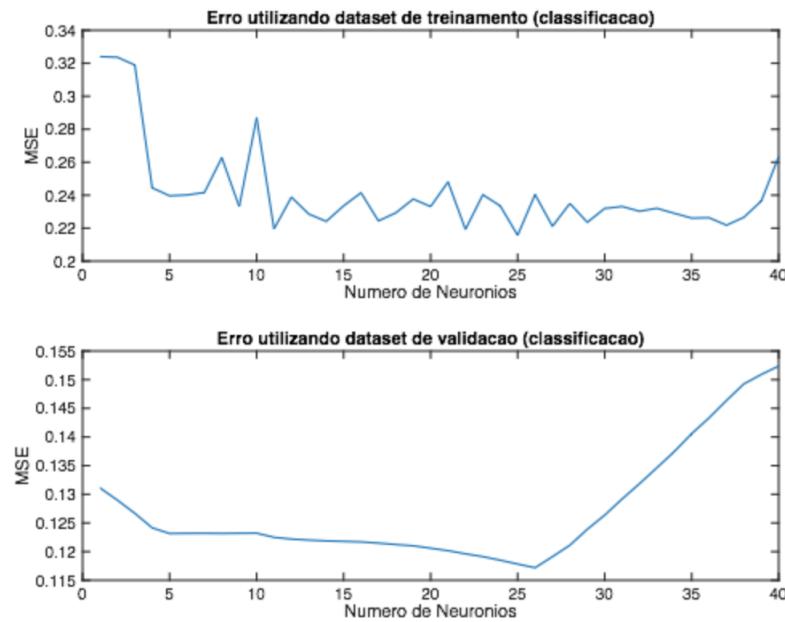


Figura 3.21: Valores de erro médio quadrático obtidos para os conjuntos de treinamento e validação, conforme o número de neurônios utilizado na camada oculta da rede neural para a topologia 4.

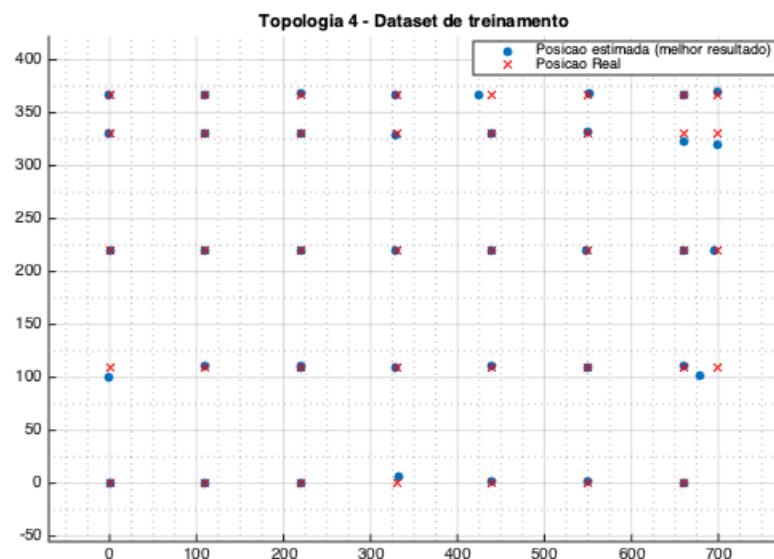


Figura 3.22: Melhor posição estimada para o modelo de topologia 4, utilizando o *dataset* de treinamento.

3.3 Discussão

Após a conclusão do primeiro experimento, foi possível observar que embora o valor do RSSI apresentasse certa relação com a distância, os modelos obtidos apresentaram um baixo grau de precisão. Esse resultado insatisfatório é justificado pela alta interferência dos fatores físicos do meio e da propagação multicaminho do sinal, e demonstrou a ineficiência de modelos de triangulação baseados em distância para cálculo da posição.

No segundo experimento, ao relacionar os valores de RSSI medidos por antenas em posições distintas, os resultados foram bem melhores. Isso permitiu concluir que o sinal de RSSI se relacionava melhor com mapeamento do meio do que com a distância direta entre o *checkpoint* e a *tag*. A utilização de redes neurais nesse caso também simplificou o problema, pois permitiu que fossem encontradas relações que não poderiam ser feitas por métodos matemáticos diretos.

O segundo experimento também permitiu concluir que a adição de *tags* de referência em posições fixas aumentava bastante a precisão do modelo, com o erro quadrático médio passando de 0.799 m para 0.4053 m com a aplicação direta da técnica. O aumento do erro conforme aumento da distância mostrou que o posicionamento dos *checkpoints* e *tags* também influenciava na precisão do modelo, assim a adição de novos *checkpoints* e um posicionamento mais bem distribuído deles resultaria em modelos mais precisos.

Finalmente, o modelo de topologia 4 foi selecionado para ser implementado dentro do software de localização. Embora esse modelo tenha apresentado um erro quadrático médio similar à topologia 2, os resultados foram mais precisos para distâncias maiores que 3,5 metros, fornecendo um erro mais bem distribuído. Nessas distâncias, o erro quadrático médio obtido com a Topologia 4 foi de 0.533 m, enquanto na topologia 2 foi de 0.561 m.

Capítulo 4

Implementação do Software

4.1 Metodologia

Para desenvolver um sistema de localização em tempo real baseado em RSSI, o projeto foi dividido nas fases de *blueprint*, planejamento, execução, controle e conclusão.

Na fase de *blueprint*, foi feita uma análise conceitual das dificuldades do projeto, do funcionamento do hardware existente, das técnicas que já foram utilizadas em projetos semelhantes, resultados obtidos e possíveis estratégias. A maior parte dessa etapa foi documentada no capítulo de Fundamentação Teórica e teve como objetivo criar um entendimento geral sobre o problema.

Na etapa de planejamento, com um entendimento melhor do problema e das possíveis estratégias para solucioná-lo, foi definido o escopo do projeto e as etapas necessárias para a sua execução.

A fase de execução e controle foi subdividida em etapas menores, para simplificar sua execução e gerenciamento. Foi adotado um processo iterativo, de forma que a cada etapa concluída o resultado era analisado e as etapas anteriores revistas:

1. Caracterização dos Modelos (apresentada no capítulo 3);
2. Arquitetura;
3. Implementação do Software;

Por fim, a etapa de conclusão tinha como o objetivo apresentar o produto obtido, avaliar seu funcionamento e validar seus requisitos.

4.2 Arquitetura

O software de localização desenvolvido neste trabalho foi batizado de Prox!Locator e possuía o objetivo de calcular a posição de uma *tag* móvel através do recebimento de pacotes UDP dos *checkpoints*. Para fazer isso, eram suas responsabilidades receber os pacotes, organizar

os dados recebidos e executar o cálculo da posição.

A fim de garantir a escalabilidade do sistema e possibilitar alterações como a adição de antenas e *tags*, troca dos modelos de cálculo e inclusão de regras de negócio adicionais, o Prox!Locator foi organizado em componentes, conforme apresentado na figura 4.1 e com as seguintes responsabilidades:

1. Recebimento de pacotes UDP, gerenciamento de conexões e envio dos pacotes recebidos para os componentes responsáveis por sua interpretação;
2. Tradução das mensagens recebidas, organização dos valores de RSSI por *tag/antena*, filtragem e envio dos dados no formato correto para cálculo da posição;
3. Cálculo da posição do *tag* através dos valores de RSSI obtidos;
4. Apresentação dos dados recebidos e/ou processados para o usuário, através de uma interface gráfica.

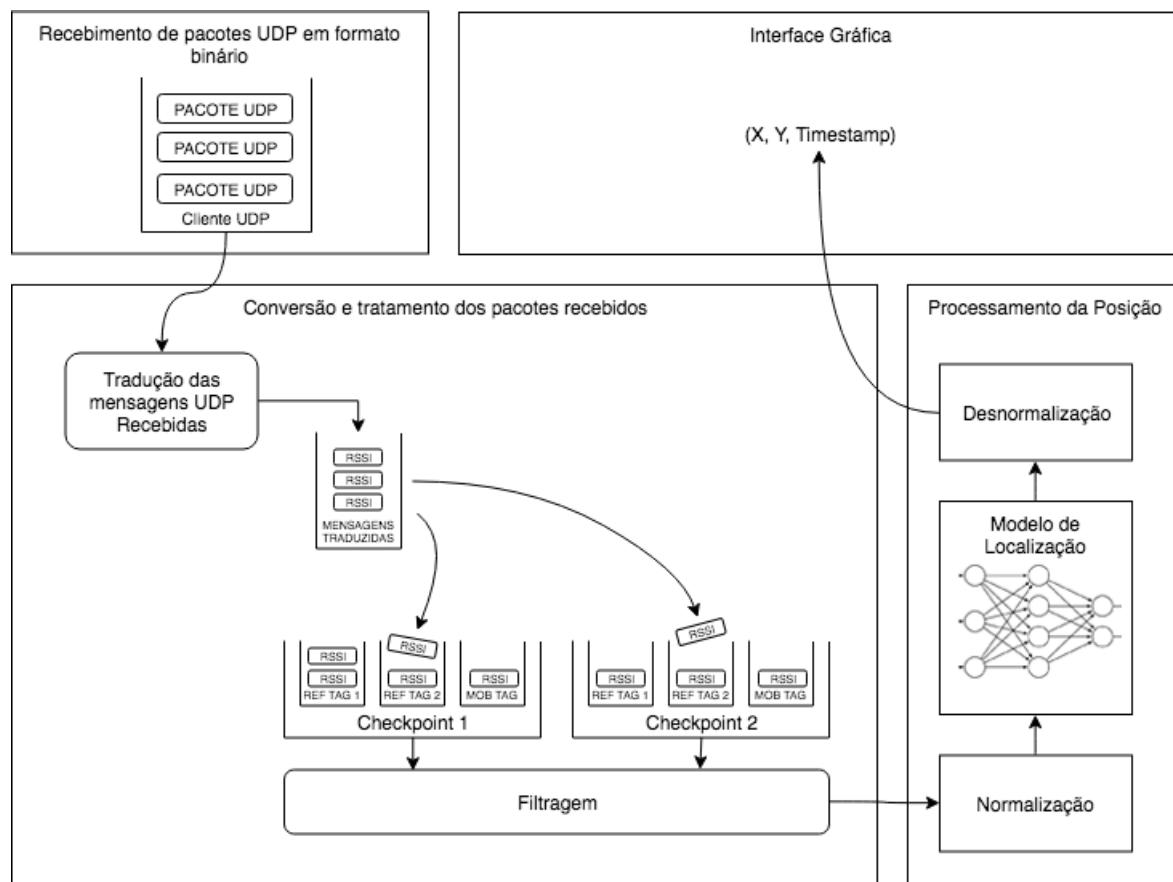


Figura 4.1: Componentes principais e visão geral do Prox!Locator

A principal tecnologia escolhida para o desenvolvimento do sistema foi o *.NET Framework* e a linguagem C#, por sua robustez, grande variedade de bibliotecas disponíveis e

consolidação no mercado. O *framework* apresenta bibliotecas nativas para uso de *sockets* UDP, estruturas de dados, gerenciamento de *threads*, entre outras funcionalidades necessárias durante o desenvolvimento.

A interface gráfica utiliza o *Windows Presentation Foundation (WPF)*, um *framework* de desenvolvimento de interfaces gráficas contido no *.NET Framework*. A plataforma é largamente utilizada para o desenvolvimento de aplicações para *Windows* e compartilha das vantagens do *.NET Framework*.

Devido às pesquisas feitas na fase de *blueprint*, as redes neurais artificiais foram escondidas como base do modelo de localização do projeto. Embora existam bibliotecas para o design, teste e execução de redes neurais no *.NET Framework*, outras tecnologias apresentam melhores recursos e são mais recomendadas para esse tipo de análise. Dessa forma, a modelagem inicial das redes neurais utilizadas no projeto é feita utilizando as bibliotecas de Redes Neurais do *MATLAB* e a implementação e treinamento dessas redes é feita utilizando a linguagem *Python*. Por fim, os modelos de redes neurais desenvolvidos no *Python* podem ser exportados e implementados dentro do *.NET Framework*, através do uso da biblioteca *NNSharp*.

4.3 Desenvolvimento

Os componentes do sistema foram especificados através de interfaces. Interfaces são contratos que definem quais os métodos e propriedades das classes sem definir sua implementação. Essa abordagem fornece algumas vantagens, pois permite que componentes que implementem a mesma interface sejam substituídos entre si sem que precisem ser refatorados.

Já as classes concretas são implementações das interfaces, contendo toda a lógica necessária para fornecer os métodos e propriedades esperadas por ela. Na figura 4.2 são apresentadas as interfaces e as implementações concretas dos componentes previstos na arquitetura. Dessa forma, cada componente criado durante o projeto não só pode ser utilizado separadamente como pode ser reutilizado dentro da própria aplicação.

4.3.1 Re却imento de pacotes UDP

O primeiro componente do sistema é o receptor de pacotes, responsável por criar um cliente UDP, receber as mensagens e enviar os pacotes recebidos para um consumidor de pacotes. Esse componente recebe como entradas a porta e onde o cliente UDP deverá ser executado e o consumidor de dados. Essa camada foi desenvolvida dentro de uma biblioteca chamada de *X.Sockets*, que segue o seguinte pseudocódigo:

1. Recebe um objeto do tipo consumidor (que implemente a interface *ISocketConsumer*) e o número da porta que será utilizada pelo *Socket* no seu construtor
2. Inicia um cliente UDP na porta especificada
3. Aguarda por novas mensagens

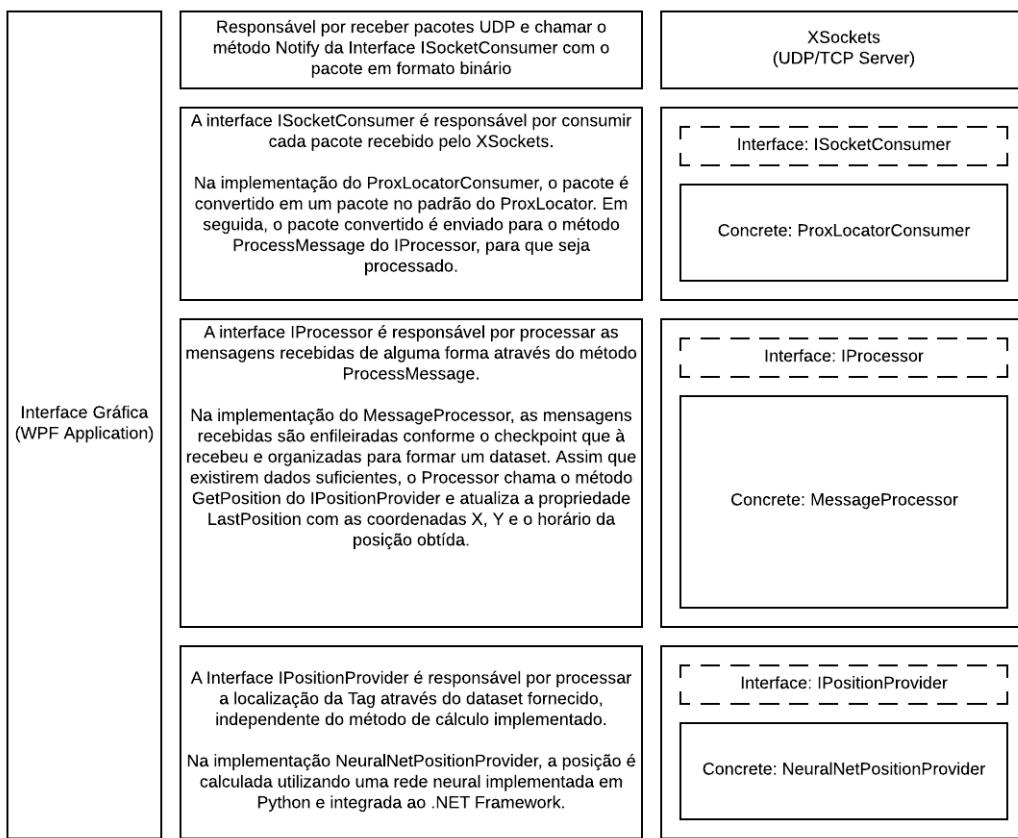


Figura 4.2: Arquitetura de software do Prox!Locator.

4. Chama o método `Notify()` do componente Consumidor sempre que uma nova mensagem for recebida.

4.3.2 Consumidor de pacotes

O segundo componente é o consumidor que deverá ser injetado dentro do receptor de pacotes. Esse componente implementa a interface `ISocketConsumer` e é responsável por tratar os pacotes recebidos pelo cliente UDP.

Interface: `ISocketConsumer`

A interface `ISocketConsumer` exige que os consumidores implementem os seguintes métodos:

1. **SetSocket(Socket socket):** Método utilizado para criar um canal de comunicação entre a aplicação e o *endpoint* que enviou a mensagem;
2. **Notify(byte[] message, IPEndPoint endpoint):** Método chamado toda vez que uma nova mensagem binária é recebida pelo cliente UDP. Dentro desse método deverá ser implementada toda a lógica de processamento dos dados recebidos.

Implementação Concreta: ProxLocatorConsumer

A implementação concreta do consumidor foi feita na classe *ProxLocatorConsumer*, representada pelo diagrama de classes da figura 4.3. Em seu método *Notify*, o consumidor decodifica o pacote binário recebido e cria uma instância de um objeto de mensagem, contendo o identificador da *tag*, o IP do *checkpoint* que recebeu a mensagem e o valor de RSSI recebido. Em seguida, ele envia esse pacote decodificado para o próximo componente, o processador, através do método *ProcessMessage*.

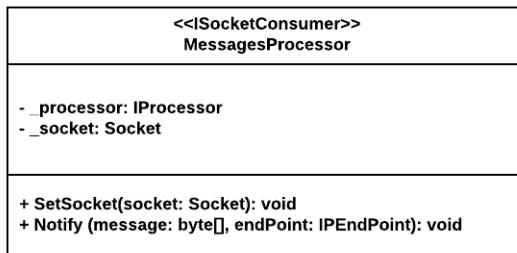


Figura 4.3: Diagrama de classes do *ProxLocatorConsumer*.

4.3.3 Processador

O terceiro componente é o processador que deverá ser injetado dentro do Consumidor de pacotes. Esse componente implementa a interface *IProcessor* e é o responsável por todo o processamento lógico das mensagens para obtenção da posição, como a separação das mensagens em filas por *checkpoint* e *tag*, filtragem, descarte de mensagens inapropriadas, organização dos *datasets*, chamada do mecanismo de cálculo e disponibilização da última posição obtida.

Interface: IProcessor

A interface *IProcessor* exige que os processadores implementem os seguintes métodos e propriedades:

1. **LastPosition:** Propriedade que indica a última posição calculada, com a data/horário de cálculo e posições x e y;
2. **ProcessMessage(LocatorMessage message):** Método chamado toda vez que uma nova mensagem é traduzida no consumidor. Esse método é a entrada lógica para todo o processamento feito nessa etapa.

Implementação Concreta: MessagesProcessor

O *MessagesProcessor* é chamado toda vez que uma nova mensagem é recebida e deve ser capaz de processar essas mensagens independentemente da ordem de chegada. Para isso,

ele conta com uma estrutura de dados interna para armazenar as mensagens recebidas em filas.

Essa estrutura, apresentada no diagrama de classes da figura 4.4 consiste em uma lista de objetos do tipo *Checkpoint*. Cada *checkpoint* encapsula uma lista de *Tags* e possui métodos para enfileiramento e desenfileiramento de mensagens. Cada *tag* possui uma fila para armazenar as mensagens recebidas e um filtro. Assim, cada mensagem recebida é filtrada e armazenada em uma fila referente ao seu par *tag/checkpoint*.

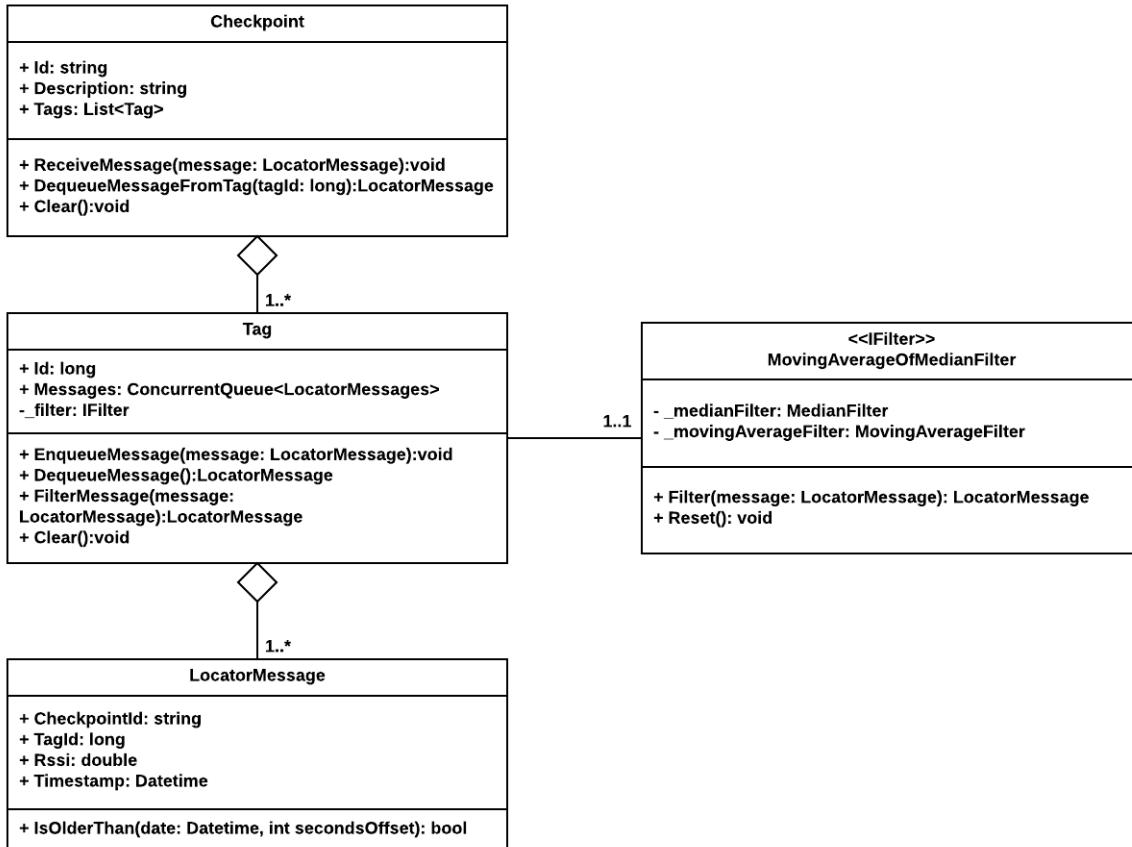


Figura 4.4: Diagrama de classes do armazenamento de mensagens no *MessagesProcessor*.

O filtro é implementado a partir da interface *IFilter* e deve possuir um método para filtragem e outro para limpeza do *buffer* interno. Para o projeto do Prox!Locator, foram implementados um filtro de médias móveis, um filtro de medianas e um filtro de médias móveis das medianas, composto dos dois anteriores.

Na figura 4.5 é possível ver a classe principal do *MessageProcessor*, com os métodos e propriedades definidas nela. Conforme as mensagens chegam e são enfileiradas, o processador verifica se existem dados suficientes para fazer uma previsão através dos métodos privados *RefreshDataSet*, *EnqueueCurrentRow* e *RefreshCurrentDataRow*.

Se existirem valores de RSSI do mesmo momento para todas as *tags* e *checkpoints*, esses dados são enviados para o estimador de posição através do método *CalculatePosition*, que retorna a posição calculada. Essa posição fica disponível na propriedade pública *LastPosition*, podendo ser acessada por componentes externos (como a interface do usuário).

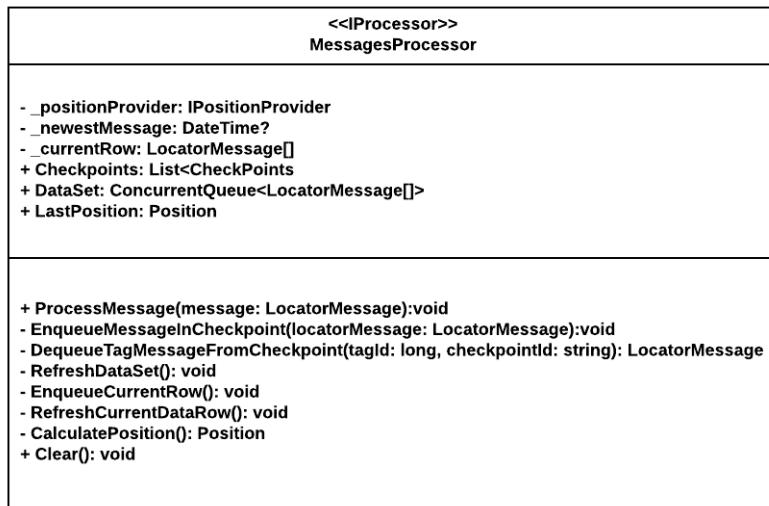


Figura 4.5: Diagrama de classes do MessagesProcessor.

4.3.4 Estimador de Posição

O estimador de posição é o componente responsável por calcular a posição da *tag* através de um vetor de mensagens recebidas. A técnica utilizada e os dados necessários nesse vetor dependem da implementação escolhida.

Interface: IPositionProvider

A interface IPositionProvider define o seguinte método:

1. `GetPosition(LocationMessage[] dataRow)`: Método que recebe um vetor de mensagens de posição e retorna a posição calculada.

Implementação Concreta: NeuralNetPositionProvider

Durante o capítulo de Caracterização dos Modelos, diferentes modelos de posicionamento foram avaliados. No fim dos experimentos, foi constatado que o modelo de topologia 4 apresentava o melhor desempenho para o problema proposto. Assim, esse modelo foi implementado dentro do Estimador do Posição do ProxLocator, através de redes neurais.

Primeiramente, os modelos foram montados e treinados utilizando a linguagem *Python* e a biblioteca *Keras*. Em seguida foram exportados para um arquivo de extensão *json* e

posteriormente importados dentro do código C#, com o uso da biblioteca *NNSharp*.

Dentro da classe *NeuralNetPositionProvider*, representado pelo diagrama de classes da figura 4.6, as redes neurais importadas foram encapsuladas em objetos do tipo *NeuralNetwork*. Ao receber os dados de RSSI através do método *GetPosition*, o estimador executa o cálculo da posição conforme o algoritmo definido para o modelo:

1. Calcula a probabilidade da distância ser maior que 3.5 metros através da rede neural de classificação (*distanceClassificationModel*);
2. Se a probabilidade for menor que 0.35, é utilizada a rede neural treinada para distâncias menores que 3.5 metros (*shortDistanceModel*);
3. Se a probabilidade tiver valor entre 0.35 e 0.65, é utilizada uma rede neural treinada para todas as distâncias (*anyDistanceModel*);
4. Se a probabilidade tiver valor maior que 0.65, é utilizada uma rede neural treinada para distâncias maiores que 3.5 metros (*longDistanceModel*).

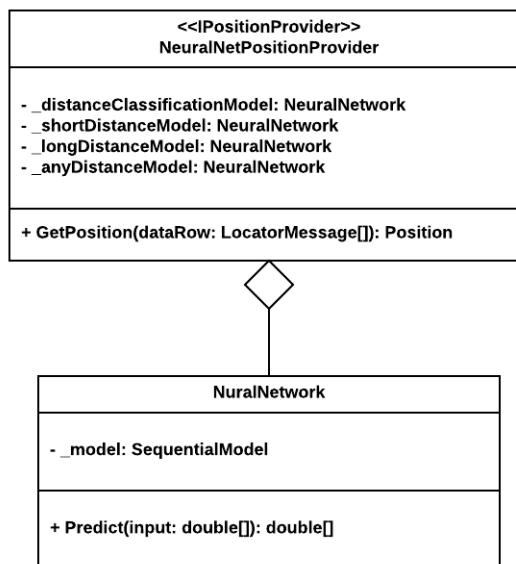


Figura 4.6: Diagrama de classes do estimador de posição baseado em redes neurais.

4.3.5 Interface Gráfica

A interface gráfica desenvolvida para o Prox!Locator foi feita utilizando o *Windows Presentation Foundation (WPF)* com o objetivo de não só apresentar a posição em tempo real das *tags*, mas também de controlar a execução do mecanismo de recebimento de pacotes /

processamento da posição.

A estrutura da aplicação gráfica foi baseada no padrão de arquitetura MVVM (*Model View ViewModel*), que tem o objetivo de separar a lógica da aplicação da interface gráfica implementada. Assim, foram criadas duas classes:

1. MainWindow.cs/MMainWindow.xaml: Classe onde todos os elementos gráficos da tela são definidos e organizados através de um arquivo no padrão XAML.
2. ProxLocatorViewModel: Classe onde todo o processamento de dados é feito e os valores que serão exibidos na tela são disponibilizados.

Ao criar a tela na classe MainWindow, o *framework* WPF permite que uma propriedade chamada *DataContext* seja definida. Assim, essa propriedade é definida como o *ProxLocatorViewModel*, e a aplicação faz a conexão lógica entre as propriedades da classe e a interface gráfica.

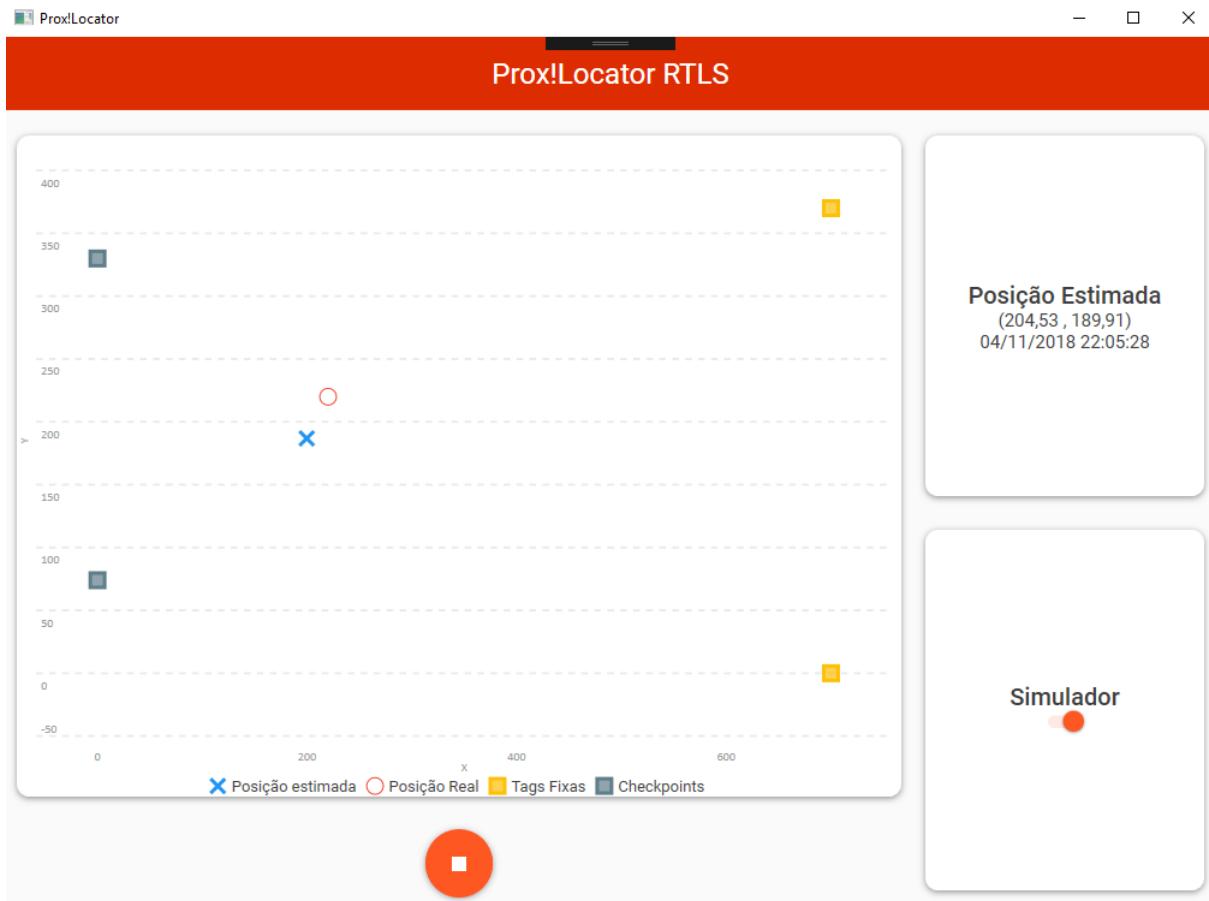


Figura 4.7: Interface gráfica do ProxLocator, onde é possível visualizar um gráfico de posição em tempo real, botões de controle e as coordenadas obtidas pelo software.

Na interface gráfica, apresentada na figura 4.7, são exibidos os seguintes elementos:

1. Um gráfico apresentando a posição dos *checkpoints*, *tags* fixas, localização real da *tag* móvel (no caso de simulação) e sua localização estimada;
2. Um cartão com as coordenadas estimadas da *tag* e a data/hora do último cálculo feito;
3. Um botão de início e parada do cálculo de posição;
4. Uma chave para seleção do modo de simulação

Caso o modo de simulação esteja desligado, o programa irá iniciar o receptor de pacotes UDP e todos os componentes de processamento de mensagens e cálculo de posição. Caso o modo de simulação esteja ativado, os componentes de processamento serão inicializados sem o receptor de pacotes UDP e os dados de RSSI serão obtidos de um arquivo de texto.

Já os botões de iniciar/parar a aplicação são utilizados para iniciar e interromper o processamento das mensagens, que tem os resultados exibidos em tempo real tanto no gráfico de posição quanto no card de resultados.

Capítulo 5

Conclusão

A proposta inicial do trabalho foi avaliar modelos de localização baseados em RSSI e desenvolver um protótipo de software de localização em tempo real. Após a etapa de Blueprint, as redes neurais foram escolhidas para a implementação dos modelos e uma arquitetura de software foi proposta.

Durante os experimentos, foi possível observar que os modelos que relacionavam os valores de RSSI com a posição da tag apresentavam resultados melhores do que os que relacionavam esses valores diretamente com a distância, chegando a erros médios de 40 cm. Ao mesmo tempo, o uso das tags fixas também melhorou os resultados, como foi observado.

Entretanto, os modelos obtidos apresentaram algumas limitações evidentes, como a forte relação deles com o meio onde os dados foram coletados, posição dos elementos de hardware e diminuição da precisão com o aumento da distância. Em ambientes mais dinâmicos como os pátios industriais, os resultados seriam fortemente impactados.

Considerando essas limitações, os próximos passos necessários seriam obter dados em ambientes maiores, durante mais tempo, com mais tags fixas e checkpoints. O reposicionamento dos checkpoints em posições bem distribuídas também aumentaria a eficácia do modelo, visto que reduziria os erros por distância.

O software desenvolvido também atendeu aos objetivos propostos, pois envolveu todas as etapas necessárias para o cálculo da posição em tempo real, desde a obtenção dos dados dos checkpoints até a execução do modelo. Com a arquitetura utilizada, esse software permitiu o reutilização de seus componentes, aspecto crucial para sua evolução. Entretanto, as limitações do modelo impactaram bastante a versatilidade do sistema, pois sua utilização em ambientes diferentes do utilizado nos experimentos iria exigir obrigatoriamente uma nova coleta de dados e treinamento.

Uma possível solução para esses problemas seria a implementação de um modelo autodidata no lugar das redes neurais já treinadas. Para isso, seria necessário captar não só os valores de RSSI das tags, mas também sua posição real através de outro mecanismo de posicionamento mais preciso. Após certo tempo de treinamento, o mecanismo mais preciso

poderia ser desligado e o modelo baseado em RSSI passaria a ser utilizado como fonte primária da localização.

Por fim, embora o sistema obtido ainda não possua condições de ser utilizado em aplicações reais, o uso do sinal de RSSI em conjunto com técnicas de inteligência artificial apresenta indicativos de que pode ser utilizado como uma opção mais barata para sistemas de localização em tempo real.

Referências Bibliográficas

- CÖMERT, Z.; KOCAMAZ, A. F. A study of artificial neural network training algorithms for classification of cardiotocography signals. *Bitlis Eren University Journal of Science and Technology*, v. 7, n. 2, p. 93–103, 2017.
- DONG, Q.; DARGIE, W. Evaluation of the reliability of rssi for indoor localization. In: IEEE. *Wireless communications in unusual and confined areas (ICWCUCUA), 2012 international conference on*. [S.l.], 2012. p. 1–6.
- FINKENZELLER, K. *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. [S.l.]: John Wiley & Sons, 2010.
- HAYKIN, S.; NETWORK, N. A comprehensive foundation. *Neural networks*, v. 2, n. 2004, p. 41, 2004.
- ISOKAWA, T.; NISHIMURA, H.; MATSUI, N. Quaternionic multilayer perceptron with local analyticity. *Information*, Multidisciplinary Digital Publishing Institute, v. 3, n. 4, p. 756–770, 2012.
- LI, H.; CHEN, C. P.; HUANG, H.-P. *Fuzzy neural intelligent systems: Mathematical foundation and the applications in engineering*. [S.l.]: CRC Press, 2000.
- MORETTIN, P. A.; BUSSAB, W. O. *Estatística básica*. [S.l.]: Editora Saraiva, 2017.
- NI, L. M. et al. Landmarc: indoor location sensing using active rfid. In: IEEE. *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*. [S.l.], 2003. p. 407–415.
- OTARAWANNA, P.; CHAROENSUK, W. Rssi-based positioning for health care service using artificial neural network approach. In: IEEE. *Biomedical Engineering International Conference (BMEiCON), 2014 7th*. [S.l.], 2014. p. 1–4.
- WANG, C. et al. An rfid indoor positioning system by using particle swarm optimization-based artificial neural network. In: IEEE. *Audio, Language and Image Processing (ICALIP), 2016 International Conference on*. [S.l.], 2016. p. 738–742.
- ZHANG, H.; SHI, X. A new indoor location technology using back propagation neural network to fit the rssi-d curve. In: IEEE. *Intelligent Control and Automation (WCICA), 2012 10th World Congress on*. [S.l.], 2012. p. 80–83.