UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU

HEITOR AUGUSTO DE NOVAIS

SISTEMA DE POSICIONAMENTO INDOOR BASEADO EM BLUETOOTH LOW ENERGY UTILIZANDO IOT

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Ouro Preto, 2022

HEITOR AUGUSTO DE NOVAIS

SISTEMA DE POSICIONAMENTO INDOOR BASEADO EM BLUETOOTH LOW ENERGY UTILIZANDO IOT

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Agnaldo José da Rocha Reis, D.Sc.

Ouro Preto
Escola de Minas – UFOP
2022

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

N935s Novais, Heitor Augusto de.
Sistema de posicionamento indoor baseado em Bluetooth Low Energy utilizando IOT. [manuscrito] / Heitor Augusto de Novais. - 2022.
67 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Agnaldo José da Rocha Reis.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Escola de Minas. Graduação em Engenharia de Controle e Automação .

1. In-Plane Switching (IPS). 2. Internet das Coisas (IoT). 3. Bluetooth. 4. received signal strength indication (RSSI). I. Reis, Agnaldo José da Rocha. II. Universidade Federal de Ouro Preto. III. Título.

CDU 681.5

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita - CRB-1716

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA CONTROLE E AUTOMACAO

FOLHA DE APROVAÇÃO

Heitor Augusto de Novais

Sistema de Posicionamento Indoor Baseado em Bluetooth Low Energy Utilizando IoT

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheiro de Controle e Automação

Aprovada em 10 de junho de 2022

Membros da banca

Dr. Agnaldo José da Rocha Reis - Orientador (Universidade Federal de Ouro Preto)
Dr. Eduardo José da Silva Luz - Examinador (Universidade Federal de Ouro Preto)
Dr. Alan Kardek Rêgo Segundo - Examinador (Universidade Federal de Ouro Preto)

Agnaldo José da Rocha Reis, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 24/06/2022.

Documento assinado eletronicamente por **Agnaldo Jose da Rocha Reis**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 24/06/2022, às 17:25, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0351451** e o código CRC **13E32B1C**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: 3135591533 - www.ufop.br

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter estado sempre comigo, me ajudando a superar todos os obstáculos. Dedico a Ele essa conquista.

Ao meu pai, Hélio, minha mãe, Maria das Graças e ao meu irmão Rafael, que foram minha base e meu exemplo, estando sempre presentes, confiando e me apoiando.

À minha namorada, Patrícia, que sempre me apoiou e incentivou, em todos os momentos.

À todos os professores, que compartilharam seu conhecimento, me despertaram brilho nos olhos e me prepararam para a vida.

À todos os meus amigos, pelo apoio, companheirismo e pelas incontáveis histórias que compartilhamos.

‘Qualquer idiota é capaz de escrever código que um computador possa entender. Bons programadores escrevem código que seres humanos podem entender.” (Martin Fowler)

RESUMO

O posicionamento de objetos tem sido um tema bastante relevante, pois é necessário localizar pessoas, orientá-las a um determinado local e auxiliar empresas e organizações na gestão de seus ativos. Embora existam soluções que ofereçam cobertura global, como o GPS, estas enfrentam dificuldades de receber sinais de satélites em ambientes fechados. Em vista disso, criou-se uma demanda de tecnologias para localização *Indoor*. Neste trabalho, é abordado o desenvolvimento de um Sistema de Posicionamento *Indoor* (IPS) baseado em beacons BLE e no Indicador de Força do Sinal Recebido (RSSI), utilizando ferramentas de código aberto e de baixo custo. Este sistema pode ser empregado em diversas aplicações, como, por exemplo, em corredores de supermercados para notificar clientes sobre ofertas de produtos localizados nas proximidades, em estacionamentos de shoppings para localizar a vaga em que o veículo foi estacionado, ou em galerias de arte para apresentar na tela do celular a descrição de uma obra de arte que está sendo observada. É apresentado em detalhes a criação do firmware dos beacons, o desenvolvimento do sistema de que calcula o posicionamento em tempo real baseado no RSSI e a aplicação do algoritmo de trilateração. Por fim, os resultados obtidos foram bastante precisos. Para as distância de 1m, 2m e 3m da referências os erros médios obtidos foram de 7cm, 15cm e 53cm, respectivamente. A partir de distâncias maiores as estimativas apresentaram alta dispersão e erros significativos, devido as atenuações sofridas pelo sinal de radiofrequência.

Palavras-chaves: IPS. IoT. Trilateração. Beacon. Bluetooth. RSSI.

ABSTRACT

The positioning of objects has been a very relevant topic, as it is necessary to locate people, guide them to a certain location and help companies and organizations in the management of their assets. Although there are solutions that offer global coverage, such as GPS, they face difficulties in receiving satellite signals indoors. In this work, the development of an Indoor Positioning System (IPS), based on beacons and the Received Signal Strength Indicator (RSSI) will be approached, using open-source and low-cost tools. This system can be used in a variety of applications, such as, for example, in supermarket aisles to notify customers of product offerings located nearby, in shopping mall parking lots to locate the place where the vehicle was parked or in art galleries. present on the cell phone screen the description of a work of art being observed. The creation of the beacons firmware, the development of the real-time positioning calculation system based on RSSI and the application of trilation are presented in detail. Finally, the results obtained were quite accurate. For distances of 1m, 2m and 3m from the references, the average errors obtained were 7cm, 15cm and 53cm, respectively. From greater distances, the estimates showed high dispersion and significant errors due to the attenuations suffered by the radiofrequency signal.

Key-words: IPS. IoT. Trilateration. Beacon. Bluetooth. RSSI.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Até a primeira metade do século XX a geolocalização era primordialmente baseada em mapas. Usar satélites para determinar a localização de pontos na superfície da terra é uma ideia que nasceu após o lançamento do Sputnik 1, o primeiro satélite artificial da história.

Em 1958, o Estados Unidos iniciou o primeiro sistema de navegação global baseado em satélites, o Transit, que revolucionou a tecnologia de navegação da época. Esse programa chegou a contar com 36 satélites em órbita (DARPA, 2022). Em 1978 o Transit deu lugar ao GPS (*Global Positioning System*). Inicialmente ele foi criado para fins militares e consiste em uma constelação de satélites que transmitem sinais de rádio aos usuários. Ele conta com 24 satélites que mantém cobertura em qualquer ponto do planeta (GPS, 2022).

O GPS mudou radicalmente a forma de como a sociedade lida com o planeta. Com isso, ele trouxe outros avanços tecnológicos sem precedentes, como nas áreas de agricultura, robótica, transporte, ecologia e esportes. Seu funcionamento é baseado na disponibilidade de pelo menos quatro satélites em qualquer latitude, longitude e altitude. Basicamente, o GPS calcula as distâncias entre o nó desconhecido e os satélites baseado no tempo de transmissão do sinal e obtém a localização aplicando o método da trilateração. Porém, o GPS não apresenta bom desempenho em ambientes fechados.

Dessa forma criou-se uma demanda de tecnologias de localização *indoor*, como o IPS (*Indoor Positioning System*). O IPS é uma solução que possibilita localizar pessoas e ativos em ambientes internos, como em aeroportos, shoppings e estacionamentos. Ele funciona de maneira similar ao GPS, porém sem o uso do satélites. Como referência, o IPS faz uso de internet das coisas e de tecnologias de localização em tempo real, baseadas em WiFi e beacons BLE (NOERTJAHYANA; WIJAYANTO; ANDJARWIRAWAN, 2017).

Beacons BLE são dispositivos Bluetooh Low Energy que emitem sinais de rádio com baixo consumo de energia, ideal para ambientes fechados (Cay et al., 2017). Com o beacon é possível determinar a posição com precisão usando algoritmos de posicionamento (Li et al., 2016). Na literatura podem ser encontradas várias técnicas para calcular distância, como tempo de chegada (TOA), diferença de tempo de chegada (TDOA), ângulo de chegada (AOA) e potência do sinal recebido (RSSI) (AZIZ; OWENS; ZAMAN, 2018). Porém, TOA e TDOA não podem ser utilizadas com eficiência em sinais de onda estreita, como o Bluetooth (Wang et al., 2011). AOA exige maior complexidade para ser implementada, comparada ao RSSI.

Desta forma, o RSSI apresenta-se como a técnica mais confiável e prática, uma vez que oferece melhores resultados contra as interferências do ambiente (Wang et al., 2011) e exige menor complexidade comparada ao RSSI. Sobre os algoritmos de posicionamento para localização indoor, a trilateração e aqueles baseados em aprendizado de máquina estão entre os

mais populares (FILÍPEK; KOVAROVA, 2016).

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um sistema de posicionamento *indoor* em tempo real de propósito geral que possa ser utilizado em aplicações que precisam estimar posição de forma precisa em um raio de até 3m das referências.

Compõe esse objetivo oferecer uma solução com baixo custo e flexível para ser alterada e otimizada visando melhorias e aplicações especificas.

1.1.2 Objetivos Específicos

- Desenvolvimento de sistema que integre hardware e software e que apresente o resultado em tempo real;
- Utilização de ferramentas de código aberto e licença gratuita;
- Desenvolvimento de algoritmo baseado em trilateração;
- Estudo do comportamento do RSSI;
- Estudo das configurações e recursos do Bluetooth Low Energy;

1.1.3 Organização do Texto

Este texto está organizado como se segue. No Capítulo 2 é realizada uma revisão da bibliografia nos conceitos pertinentes a um Sistema de Localização *Indoor* baseado em beacons, nos materiais, nas técnicas de programação e nas métricas de erro utilizadas ao longo deste trabalho. O Capítulo 3 contempla a construção do sistema embarcado e o desenvolvimento da aplicação que calcula o posicionamento. Os resultados e comparações das estimativas de distância e posição são apresentados no Capítulo 4. Finalmente, o Capítulo 5 traz as conclusões e sugestões para melhoria dos resultados obtidos com o trabalho.

2 REVISÃO BIBLIOGRÁFICA

2.1 Bluetooth Low Energy (BLE)

O Bluetooth Low Energy é uma tecnologia de rede sem fio desenvolvida para transmitir dados com baixo consumo de energia. Em comparação com o Bluetooth comum, o BLE tem a vantagem de consumir bem menos energia. Além disso, como as ondas emitidas são de rádio, com frequência de 2.4Ghz, o alcance e a penetração nas estruturas, como concreto, é bem maior. Por outro lado o BLE possui uma largura de banda menor, o que faz com que ele não seja ideal para aplicações que precisam de grandes processamentos.

Suas características fazem com que um dispositivo BLE atue na maior parte do tempo em modo *sleep*, sendo ativado apenas para realizar conexões que duram milésimos de segundos. Com isso, é possível garantir a transmissão de dados por períodos muito mais longos, ideal para aplicações que enviam informações esporadicamente.

O Bluetooth Low Energy recebe grande atenção em questões médicas, devido à sua portabilidade. Ele é visto como um meio tecnológico para a segurança de pessoas com problemas de saúde, como por exemplo disparar uma mensagem de celular, avisando a um parente ou médico, quando um paciente está com alterações no seu ritmo cardíaco, podendo ser resultado do início de um enfarto.

É também muito utilizado para automatizar tarefas do dia-a-dia em uma residência doméstica, como acender a luz quando alguém entrar no cômodo ou destrancar porta ao receber a aproximação de uma pessoa, e no varejo para notificar clientes sobre ofertas de produtos que estão nas proximidades.

2.2 Beacon BLE

O Beacon BLE é um pequeno dispositivo que utiliza a tecnologia Bluetooth Low Energy (BLE), que emite um sinal intermitente de ondas de rádio que consegue localizar seu smartphone em um determinado raio. É ideal para ambientes fechados, com precisão maior que um sistema de GPS, por exemplo.

Cada beacon tem um ID Exclusivo feito de números e letras e as informações de identificação são transmitidas por Bluetooth várias vezes a cada segundo. Os beacons bele são dispositivos bastante simples e consistem apenas em três partes: baterias, uma unidade de processamento central (CPU) e o rádio.

Existem muitos tipos diferentes de beacons, mas as configurações mais populares utilizam os protocolos de comunicação *iBeacon* e *Eddystone*. O *iBeacon* é mantido pela Apple e o *Eddystone* é mantido pela Google.

Figura 1 – iBeacon

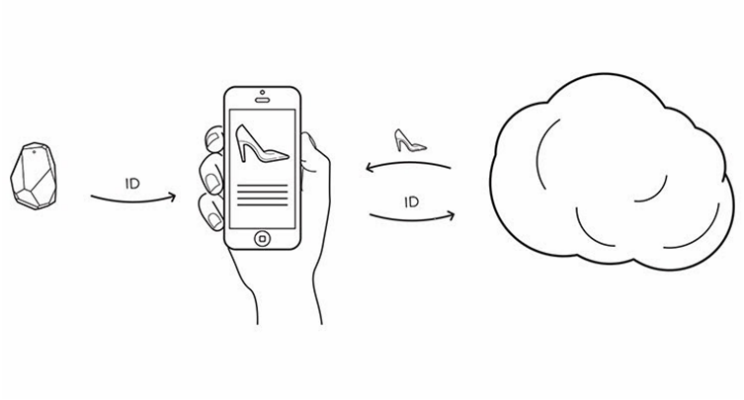
Fonte: ibeaconinsider (2021a).

Com o *iBeacon*, a Apple padronizou o formato para *BLE Advertising*. Sob este formato, um pacote de publicidade consiste em quatro peças principais de informação:

- UUID: Esta é uma sequência de 16 bytes usada para diferenciar um grande grupo de beacons relacionados. Por exemplo, se a Coca-Cola mantivesse uma rede de beacons em uma cadeia de supermercados, todos os beacons da Coca-Cola compartilhariam o mesmo UUID. Isso permite que o aplicativo dedicado para smartphones da Coca-Cola saiba quais anúncios de beacon vêm de balizas de propriedade da Coca-Cola.
- Major: Esta é uma sequência de 2 bytes usada para distinguir um subconjunto menor de beacons dentro do grupo Major. Por exemplo, se a Coca-Cola tivesse quatro beacons em um supermercado particular, todos os quatro teriam o mesmo Major. Isso permite que a Coca-Cola saiba exatamente em qual loja seu cliente está.
- Menor: Esta é uma sequência de 2 bytes destinada a identificar beacons individuais. De acordo com o exemplo da Coca-Cola, um beacon na frente da loja teria seu próprio Menor único. Isso permite que o aplicativo dedicado da Coca-Cola saiba exatamente onde o cliente está na loja.
- Tx Power: É usado para determinar a proximidade (distância) do beacon como uma medida de calibração. Como isso funciona? A potência TX é definida como a força do sinal exatamente a 1 metro do dispositivo. Isso tem que ser calibrado e codificado com antecedência. Os dispositivos podem então usar isso como uma linha de base para dar uma estimativa de distância aproximada.

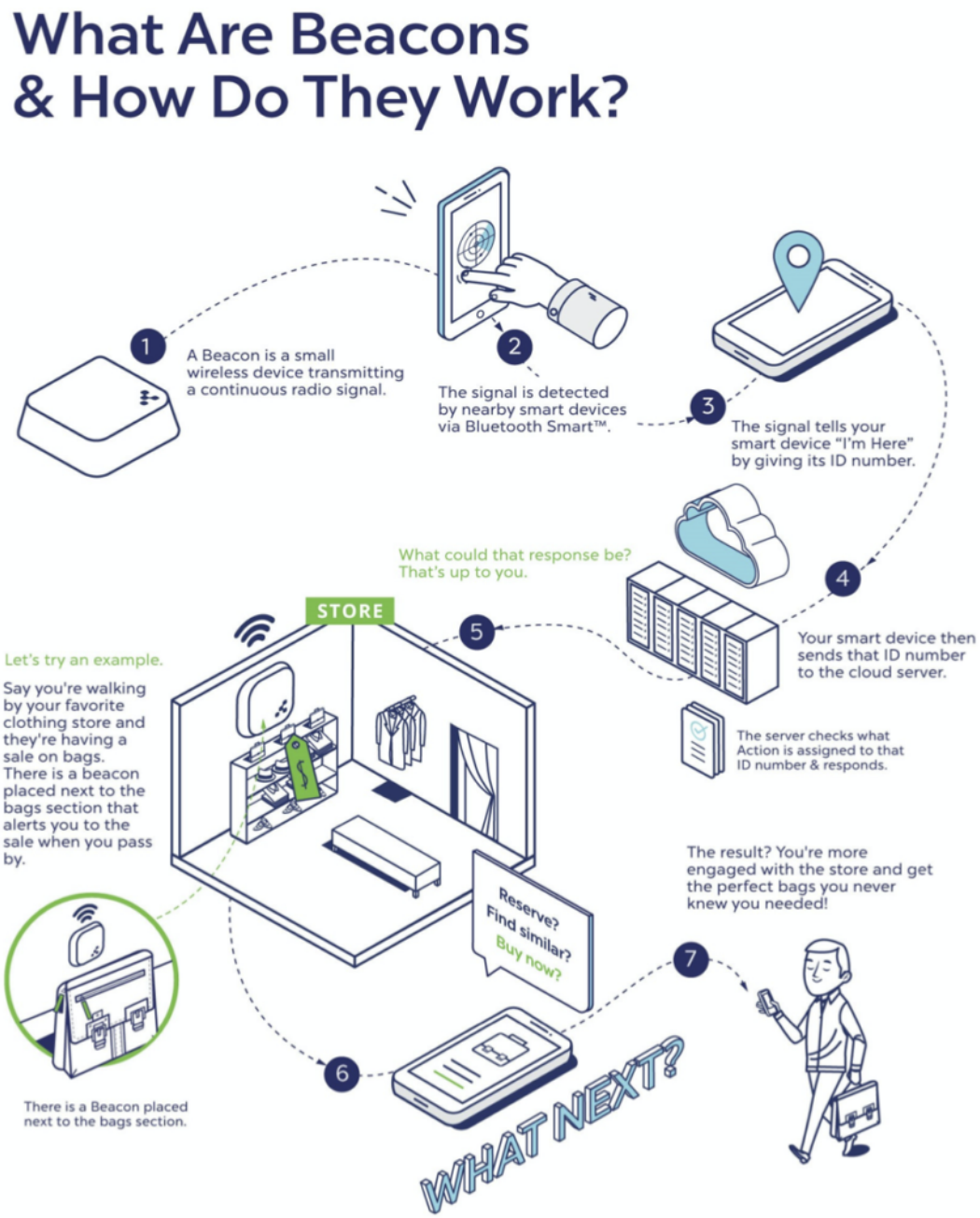
Com uma rede *iBeacon*, qualquer marca, varejista, aplicativo ou plataforma será capaz de entender exatamente onde um cliente está no ambiente. Isso oferece uma oportunidade de enviar aos clientes mensagens e anúncios altamente contextuais, hiper-locais e significativos em seus smartphones.

Figura 2 – Anuncios e ofertas através dos iBeacons



Fonte: ibeaconinsider (2021b).

Figura 3 – O que são beacons e como eles trabalham



Fonte: Kontakt.io (2021).

2.3 RSSI

O RSSI, que significa Indicador de Força do Sinal Recebido, é uma medida estimada do nível de energia que um dispositivo cliente de radiofrequência está recebendo da fonte. É um número relativo, representado em decibéis (dB), que mede o quão forte é um sinal quando é recebido por um dispositivo. Quando medido em números negativos, o número que está mais perto de zero geralmente significa um sinal melhor. Como exemplo, -50dB é um bom sinal, -75dB é bastante razoável e -100dB é nenhum sinal.

Devido à polarização do sinal elétrico nas ondas de rádio, o sinal de rádio é fortemente influenciado por fatores externos devido aos obstáculos relacionados ao ambiente, fazendo com que o sinal não chegue ao destino esperado ou que sofra dispersão quando se ramifica ou colide com outros sinais e superfícies presentes no ambiente, gerando distúrbios.

Por meio do RSSI é possível determinar uma relação entre a amplitude do sinal e distância da fonte. Obtendo a distância, torna-se viável estimar a posição aplicando técnicas de geolocalização.

2.4 ESP32

O ESP32 é uma placa de desenvolvimento microcontrolada que possui conectividade pelos módulos Bluetooth e WiFi integrados. Esta configuração facilita muito em projetos IoT, devido a constante troca de informações com a rede.

Figura 4 – Placa ESP32

Fonte: AthosElectronics (2021).

Especificações:

- CPU: Xtensa® Dual-Core 32-bit LX6;
- Memória ROM: 448 KBytes;
- Clock máximo: 240MHz;

- Memória RAM: 520 Kbytes;
- Memória Flash: 4 MB;
- Wireless padrão 802.11 b/g/n;
- Conexão Wifi de 2.4Ghz (máximo de 150 Mbps);
- Antena embutida na placa;
- Conector micro USB para comunicação e alimentação;
- Wi-Fi Direct (P2P), P2P Discovery, P2P Group Owner mode e P2P Power Management;
- Modos de operação: STA/AP/STA+AP;
- Bluetooth BLE 4.2;
- Portas GPIO: 11;
- GPIO com funções de PWM, I2C, SPI, etc;
- Tensão de operação: 4,5 9V;
- Conversor analógico digital (ADC).

2.5 MQTT

MQTT é um protocolo de mensagens baseado na pilha TCP/IP para internet das coisas (IoT). Ele foi projetado como um transporte de mensagens de publicação/assinatura extremamente leve, que é ideal para conectar dispositivos remotos com hardware limitado e largura de banda restrita. O MQTT hoje é usado em uma grande variedade de indústrias, como automotivo, manufatura, telecomunicações, petróleo e gás, etc.

O protocolo MQTT define dois tipos de entidades na rede: um *message broker* e inúmeros clientes. O *broker* é um servidor que recebe todas as mensagens dos clientes e, em seguida, roteia essas mensagens para os clientes de destino relevantes. Um cliente é qualquer dispositivo que possa interagir com o broker e receber mensagens. Um cliente pode ser um sensor de IoT em campo ou um aplicativo em um data center que processa dados de IoT.

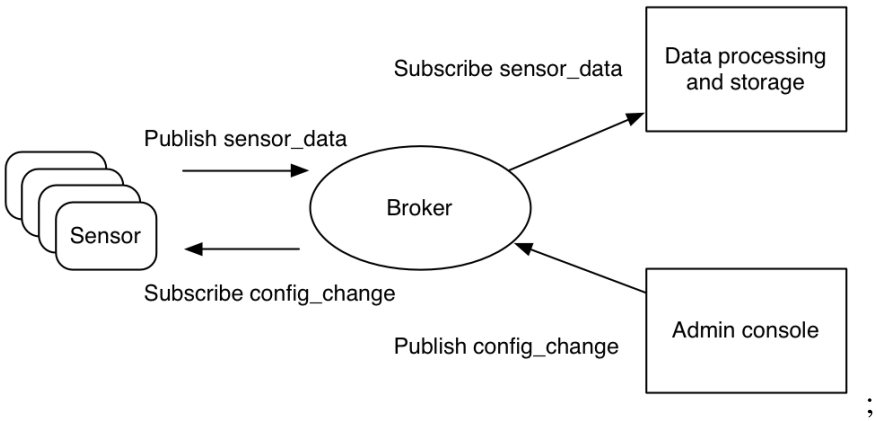
A comunicação MQTT pode ser exemplificada pelas seguintes etapas:

1. O cliente conecta-se ao *broker*. Ele pode assinar qualquer tópico de mensagem no *broker*. Essa conexão pode ser uma conexão TCP/IP simples ou uma conexão TLS criptografada para mensagens sensíveis.
2. O cliente publica as mensagens em um tópico, enviando a mensagem e o tópico ao *broker*.

3. Em seguida, o *broker* encaminha a mensagem a todos os clientes que assinam esse tópico.

Como as mensagens do MQTT são organizadas por tópicos, o desenvolvedor de aplicativos tem a flexibilidade de especificar que determinados clientes podem, somente, interagir com determinadas mensagens.

Figura 5 – Publicação e Assinatura



;

Fonte: Yuan (2017).

2.6 Programação

2.6.1 Programação Orientada a Objetos (POO)

O paradigma da Programação Orientada a Objetos é um modelo de análise, projeto e programação baseado na aproximação entre o mundo real e o mundo virtual, através da criação e interação entre objetos, atributos, códigos, métodos, entre outros.

Esse paradigma se baseia principalmente em dois conceitos chave: classes e objetos. Todos os outros conceitos, igualmente importantes, são construídos em cima desses dois.

A seguir, são apresentadas as características de um software desenvolvido com POO:

- Confiabilidade: A geração de código baseado no conceito de objetos e classes fornece uma grande independência ao programa. Assim, qualquer intervenção que seja necessária não afetará outros pontos do sistema. Isso confere robustez e confiabilidade.
- Agilidade: A criação paralela de código torna todo o processo bastante ágil. Ganha-se em tempo e eficiência, tornando um software com paradigma POO oportuno. Várias equipes podem trabalhar no mesmo projeto de forma independente.
- Ajustável: Essa característica diz respeito ao processo de manutenção do código-fonte. Ao atualizar uma parte pequena, o conceito de herança garante que, automaticamente, todas as partes que utilizarem tal método sejam beneficiadas.

- Extensível: O uso do princípio da reutilização do software adiciona funcionalidades ao sistema já existente. Não é preciso “reinventar a roda”, reescrevendo o código. Isso confere maior capacidade de estender um sistema já existente.
- Natural: O conceito de POO que traz para a programação o mundo concreto, tal qual vemos no dia-a-dia, faz com que se ganhe naturalidade. O ponto de vista humano se torna mais próximo do virtual. Assim, pensa-se no problema geral, em vez de concentrar o foco em pormenores.

2.6.2 C++

C++ é uma poderosa linguagem de programação de propósito geral. Ela pode ser usado para desenvolver sistemas operacionais, sistemas embarcados, navegadores, jogos e assim por diante. C++ suporta diferentes formas de programação como procedural, orientada a objetos, funcional, dentro outros.

2.6.3 Python

Python é uma linguagem de propósito geral de alto nível, multiparadigma, que suporta os paradigmas orientado a objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens.

2.6.4 JSON

O formato JSON (JavaScript Object Notation) é utilizado para estruturar dados em formato de texto e permitir a troca de dados entre aplicações de forma simples, leve e rápida.

Ele é construído sobre duas estruturas:

- Uma coleção de pares de nomes/valores. Em várias linguagens, isso é realizado como um objeto, registro, estrutura, dicionário, tabela hash, lista de chaves ou matriz associativa.
- Uma lista ordenada de valores. Na maioria dos idiomas, isso é realizado como uma matriz, vetor, lista ou sequência.

São estruturas de dados universais que praticamente todas linguagens de programação modernas suportam.

2.7 Frameworks e Bibliotecas

2.7.1 Arduino

Arduino é uma plataforma de prototipagem de projetos eletrônicos *open-source* constituída tanto de hardware e software fáceis de usar.

As placas Arduino são capazes de ler entradas - luz em um sensor, um dedo em um botão ou uma mensagem do Twitter - e transformá-la em uma saída - ativando um motor, ligando um LED, publicando algo online. Você pode dizer à sua placa o que fazer enviando um conjunto de instruções para o microcontrolador na placa. Para isso você utiliza a linguagem de programação Arduino (baseada em *Wiring*), e o Software Arduino (IDE) , baseado em *Processing* (ARDUINO, 2018).

Devido às importantes contribuições da comunidade, que abrange desde estudantes, hobistas, artistas à programadores profissionais, a plataforma dispõe de uma enorme gama de conhecimento acessível, que pode ser de grande ajuda para novatos e especialistas.

2.7.2 Eclipse Mosquitto

O Eclipse Mosquitto é um *message broker* de código aberto que implementa o protocolo MQTT. O Mosquitto é leve e adequado para uso em todos os dispositivos com acesso à rede, desde computadores de hardware limitado até servidores robustos (MOSQUITTO, 2022).

2.7.3 Matplotlib

Matplotlib é uma biblioteca que oferece suporte para criar visualizações estáticas, animadas e interativas em Python (MATPLOTLIB, 2022).

2.8 Métricas de Avaliação do Erro

- Erro Absoluto Médio (Mean Absolute Error - MAE)

  O erro absoluto médio (MAE) é a medida de erros entre observações que expressam o mesmo fenômeno, como por exemplo, o valor observado versus o previsto. É uma métrica comum para mensurar erros de estimativa. O MAE é calculado como:

$$MAE = \frac{\sum_{t=1}^{n} |y_t - x_t|}{n} = \frac{\sum_{t=1}^{n} |\epsilon_t|}{n} \tag{2.1}$$

  Em que y_t é o valor previsto e x_t é o valor observado. Portanto, $|\epsilon_t| = |y_t - x_t|$ é uma média aritmética dos valores absolutos.

- Erro Absoluto Percentual Médio (Mean absolute percentage error - MAPE)

  O erro absoluto percentual médio (MAPE) é a média dos erros percentuais absolutos das previsões. Também é comumente utilizado para mensurar erros de estimativa, mas neste caso resultando em uma porcentagem. O MAPE é calculado como:

$$MAPE = \left(\frac{1}{n} \sum_{t=1}^{n} \frac{|y_t - x_t|}{|y_t|} \right) * 100 \tag{2.2}$$

  Em que y_t é o valor observado e x_t é o valor previsto.

3 METODOLOGIA

Este trabalho consiste em estimar a posição de dispositivos móveis em ambientes fechados com a possibilidade de monitoramento em tempo real. Para isso, foi instalada uma rede de beacons para monitorar constantemente a presença de qualquer dispositivo móvel BLE que estivesse emitindo sinal de radiofrequência que pudesse identificá-lo. O ato de um objeto BLE emitir sinal de radiofrequência com dados de identificação também é conhecido como *advertising*.

A rede de beacons foi construída com quatro módulos ESP32, que receberam a denominação de "estações". Estes foram programados para receber o *advertising* e enviar ao servidor os dados de identificação e o RSSI medido do dispositivo móvel.

Após receber as medições de RSSI, o servidor teve como atribuição processar os dados e calcular as distâncias entre as estações de monitoramento que efetuaram a medição e o dispositivo móvel. De posse das distâncias tornou-se possível estimar a posição aplicando um algoritmo de geolocalização. Dessa forma o resultado pôde ser entregue a um cliente para que este pudesse renderizar uma animação em tempo real.

3.1 Cálculo da Distância

Existem muitas fórmulas para se calcular a distância entre dois objetos baseando-se na intensidade do sinal. Porém, algumas delas levam em conta fatores que não retratam a realidade desse experimento. A fórmula a ser apresentada é bastante recorrente em trabalhos similares (Bellecieri; Jabour; Jabour, 2015) e possui parâmetros condizentes com o caso em questão, a exemplo do *path loss*, calculado experimentalmente em Kurose e Ross (2006):

$$RSSI(dBm) = A - 10 \times n \times log(d) \tag{3.1}$$

Isolando *d* temos que:

$$d = 10^{\frac{RSSI-A}{-10\times n}} \tag{3.2}$$

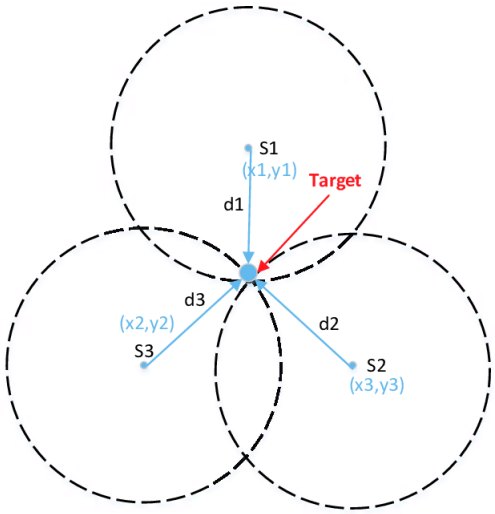
Em que:

- RSSI (dBm) é a potência do sinal recebido.
- A é a referência de RSSI (dB) medido a uma distância de 1 metro.
- n é o *path loss*, ou grandeza que corresponde a atenuação do sinal (dB).
- d é a distância entre dois objetos que se comunicam.

3.2 Cálculo da Posição

O método utilizado para estimar a posição foi a trilateração. A trilateração determina a localização geográfica do nó desconhecido a partir da posição previamente conhecida dos nós de referência. Traçando três circunferências com centros nos nós de referência e raios iguais às estimativas de distância, tem-se na interseção das circunferências a representação do nó desconhecido (SAVVIDES C. C. HAN, 2001).

Figura 6 – Descrição do método de trilateração



Fonte: (QIAN et al., 2018)

Pela Figura 6, S1, S2 e S3 são os pontos de referência e d_1, d_2 e d_3 são as distâncias conhecidas para o nó desconhecido. Com isso, a posição do nó desconhecido pode ser encontrada resolvendo o seguinte sistema de equações quadráticas, de onde pode ser obtidos os valores de x_0 e y_0 (YANG et al., 2020):

$$\begin{cases} (x_1 - x_0)^2 + (y_1 - y_0)^2 = d_1^2 \\ (x_2 - x_0)^2 + (y_2 - y_0)^2 = d_2^2 \\ (x_3 - x_0)^2 + (y_3 - y_0)^2 = d_3^2 \end{cases} \tag{3.3}$$

Reescrevendo a Equação 3.3 na forma matricial, tem-se a Equação 3.4.

$$\begin{bmatrix} (x_1 - x_0)^2 + (y_1 - y_0)^2 \\ (x_2 - x_0)^2 + (y_2 - y_0)^2 \\ (x_3 - x_0)^2 + (y_3 - y_0)^2 \end{bmatrix} = \begin{bmatrix} d_1^2 \\ d_2^2 \\ d_3^2 \end{bmatrix} \tag{3.4}$$

A fim de isolar as variáveis x_0 e y_0, a Equação 3.4 pode ser reescrita novamente, chegando-se a Equação 3.5.

$$2 \cdot \begin{bmatrix} (x_2 - x1) & (y_2 - y1) \\ (x_3 - x1) & (y_3 - y1) \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) \\ (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) \end{bmatrix} \tag{3.5}$$

Ao desenvolver a Equação 3.5, chega-se a:

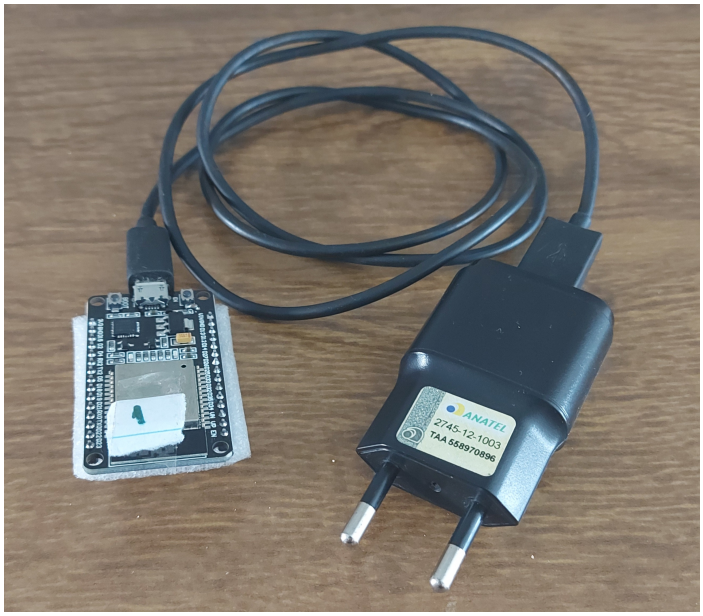
$$\begin{cases} x_0 = \dfrac{\begin{vmatrix} (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) & 2(y_2 - y1) \\ (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) & 2(y_3 - y1) \end{vmatrix}}{\begin{vmatrix} 2(x_2 - x1) & 2(y_2 - y1) \\ 2(x_3 - x1) & 2(y_3 - y1) \end{vmatrix}} \\ y_0 = \dfrac{\begin{vmatrix} 2(x_2 - x1) & (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) \\ 2(x_3 - x1) & (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) \end{vmatrix}}{\begin{vmatrix} 2(x_2 - x1) & 2(y_2 - y1) \\ 2(x_3 - x1) & 2(y_3 - y1) \end{vmatrix}} \end{cases} \tag{3.6}$$

Por fim, ao substituir os valores conhecidos e resolver os determinantes na Equação 3.5, obtém-se as coordenadas do nó desconhecido.

3.3 Estações de Monitoramento

Cada estação foi constituída por um sistema embarcado baseado em ESP32 (2.4) alimentada por uma fonte CA-CC.

Figura 7 – Estações ESP32 com fonte CA-CC



Fonte: O Autor.

Com ESP32 é possível utilizar todo o ecossistema Arduino na prototipagem. Dessa forma foi feito o uso da IDE do Arduino para codificar o firmware em C++ e AVR C, conforme Apêndice A.1.

Com o auxílio do Gerenciador de Bibliotecas da IDE, foram importadas as bibliotecas *BLEDevice*, *BLEUtils*, *BLEScan* e *BLEBeacon* para manipular módulo BLE embarcado. Dessa forma as estações foram configuradas para rastrear outros beacons e emitir *advertising* no padrão *iBeacon*.

Devido a natureza do sinal de radiofrequência, o RSSI se mostrou bastante sensível a ruídos, e como o RSSI é medido em escala logarítmica, qualquer variação mínima representa impacto significativo. Então, com o intuito suavizar as oscilações da medição e também otimizar o tráfego de dados, as estações foram configuradas com tempo de *scan* de 1 segundo.

Um tempo menor não apresentou nenhuma melhora na estimativa e ainda capturou mais oscilações do sinal, aumentando a sensação de movimento mesmo o objeto estando parado. Por outro lado um tempo maior se mostrou indesejável para rastrear objetos em movimento, pois pela diminuição do tempo de amostragem adicionou-se um atraso na estimativa.

Continuando os esforços para manter as medições com mínimo de atenuações e ruídos, fez-se importante escolher um ambiente plano, livre de obstáculos e com ausência de interferências eletromagnéticas. Dessa forma foram posicionadas quatro estações em uma área plana

quadrada de 5x5 metros, equidistantes e à uma altura de 1.5m, visando realizar as medições com a menor distância possível.

Figura 8 – Espaço de trabalho

Fonte: O Autor.

Após realizar as configurações do módulo BLE e a adequação do ambiente, fez-se necessário estruturar as medições e os dados de identificação do dispositivo móvel para enviar ao servidor. Portanto, as mensagens foram estruturadas no formado JSON (2.6.4), por ser mais leve e mais simples de manipular comparado à outros formatos, como por exemplo o XML. A montagem da mensagem está exposta no Apêndice A.1.

O próximo passo foi conectar as estações à rede para que pudessem se comunicar com o servidor. Uma das facilidade de se trabalhar com módulos ESP32 é que eles também possuem módulo WiFi embutido. Dessa forma bastou somente configurar o módulo com os dados da rede WiFi disponível.

Em seguimento, devido as facilidades de uso em aplicações de IoT, foi feita opção pelo protocolo de comunicação MQTT. Para implementá-lo bastou importar a biblioteca *PubSubClient*, que também pode ser adicionada pelo Gerenciador de Bibliotecas da IDE, e então configurar o caminho do servidor e os tópicos de leitura e escrita. Desse modo as estações tiveram suas configurações concluídas e apresetavam-se prontas para realizar o monitoramento.

3.4 Dispositivo Móvel

Uma dos requisitos deste trabalho é que o dispositivo móvel ofereça suporte ao Bluetooth Low Energy. Dessa forma os experimentos foram realizados com um smartphone Samsung Galaxy A71 com o sistema operacional Android 10.

Figura 9 – Ilustração do dispositivo móvel

Fonte: Samsung (2022).

Para manipular as configurações do Bluetooth como a identificação, a potência do sinal e o modo de funcionamento foi necessário instalar o aplicativo nRF Connect.

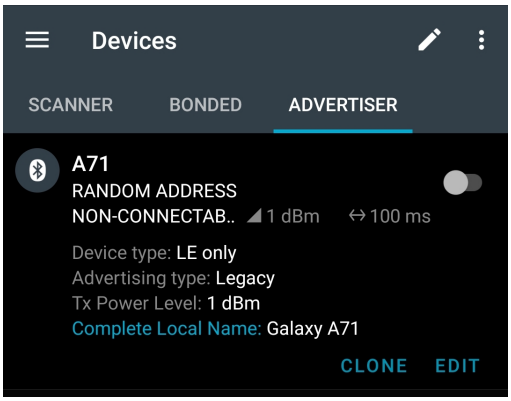
O nRF Connect é uma ferramenta mantida pela Nordic Semicondutor que permite testes e desenvolvimento de produtos Bluetooth Low Energy e células IoT. Ela permite fácil configuração de conexões com outros dispositivos e usa essas conexões para ler e gravar informações.

Dessa forma o smartphone foi configurado para emitir *advertising* a cada 160ms, intervalo mínimo disponível nas configurações do nRF Connect, e o TX Power foi configurado para fornecer a potência máxima do dispositivo, 1dB, conforme Figura 10.

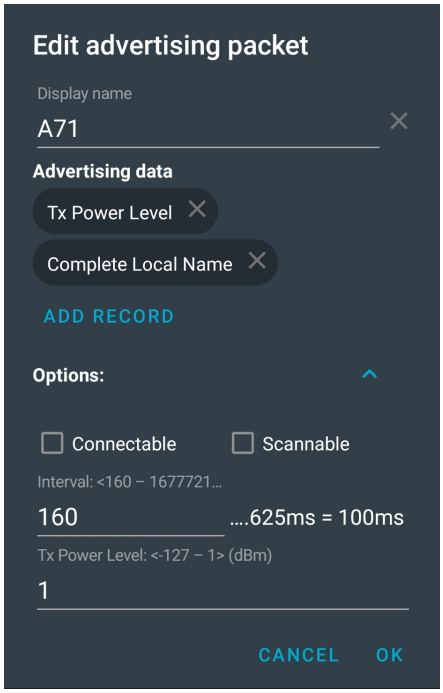
Foi de interesse que o *advertising* fosse realizado com o menor intervalo de tempo possível, pois assim haveria menos perdas de informações durante a amostragem do sinal.

Desse modo a configuração do smartphone foi concluída e este passou a ser visível para as estações instaladas.

Figura 10 – Telas do aplicativo nRF Connect



(a) Aba "Devices"



(b) Configuração do advertising

Fonte: O Autor.

3.5 Servidor

O servidor é um *host* que executa um ou mais serviços que compartilham recursos com os clientes.

3.5.1 Hardware

O hardware que ofereceu suporte a este servidor foi um computador Intel(R) Core(TM) i5-7200U com 8GB de memória RAM e com sistema operacional Linux Ubuntu 20.04 LTS Focal Fossa.

3.5.2 Message Broker MQTT

Assim como foi criada uma estrutura nos módulos ESP32 para organizar os dados coletados e estruturá-los em mensagens, o servidor também precisa estar preparado para receber e interpretar essas mensagens.

O primeiro passo foi instalar um programa para gerenciar o protocolo MQTT (2.5). Esse programa é conhecido *message broker*. O *message broker* é responsável por gerenciar os tópicos e notificar os clientes que novos dados foram recebidos. Para este trabalho foi utilizado o Eclipse Mosquitto (2.7.2), configurado em localhost na porta 1883.

Para receber as informações entregues pelas estações de monitoramento foi criado um tópico com o nome de "station_data".

3.6 Desenvolvimento da Aplicação

Com a conclusão da camada de infraestrutura a próxima etapa passou pelo desenvolvimento da aplicação responsável por calcular efetivamente o posicionamento do dispositivo móvel.

A aplicação foi desenvolvida em Python e foi baseada no paradigma Orientado a Objetos. O código foi composto por classes e módulos, que de forma coesa se relacionam entre si para produzirem as estimativas de posição.

Uma das vantagens de se utilizar o paradigma Orientado a Objetos é que há uma relação bastante próxima entre o mundo real e o virtual, como consequência tem-se um código mais amigável. Outra vantagem é possibilidade de reaproveitar código utilizando herança, o que reduz bastante o tempo de desenvolvimento e o tamanho do código.

3.6.1 Representação dos Objetos

Por este trabalho ter como objeto principal dispositivos BLE, o desenvolvimento foi iniciado pela criação da classe que define este objeto. Essa classe descreve todas as características

de um objeto BLE, como possuir uma identificação e um RSSI associado, além de outros atributos conforme Apêndice B.1.

Logo em seguida foi definida a classe que representa as estações, conforme Apêndice B.2. Como uma estação também é dispositivo BLE, sua classe foi herdada da classe BLE, que em resumo significa que ela passou a possuir todos os atributos que a classe BLE possui, além de seus atributos adicionais, como uma lista de dispositivos BLE rastreados e suas coordenadas no ambiente físico.

Para finalizar a representação dos objetos BLE, foi definida a classe que representa o dispositivo móvel, conforme Apêndice B.3. Essa classe também herda da classe BLE.

O passo seguinte foi definir as classes que representam as coordenados de um dispositivo BLE no plano cartesiano (Apêndice B.4) e classe que define a conexão entre dois dispositivos BLE (Apêndice B.5). Esta última também é responsável por estimar a distância entre esses dois dispositivos.

Concluída a representação de todos os objetos do ambiente físico fez-se necessário a implementação dos módulos.

3.6.2 Módulo de Mensageria

O Módulo de Mensageria foi responsável pela comunicação com o *message broker* MQTT. Sua tarefa era assinar o tópico "station_data", receber as mensagens do broker e direcioná-las para tratamento. A implementação pode ser verificada no Apêndice B.6.

3.6.3 Módulo de Sincronização

O Módulo de Sincronização tinha a responsabilidade de converter as mensagens em objetos e sincronizar os estados entre os ambientes real e virtual. Ou seja, ele adicionava novos dispositivos encontrados e removia dispositivos que estavam fora da área de cobertura. Sua implementação pode ser vista no Apêndice B.7.

3.6.4 Módulo de Posicionamento

O Módulo de Posicionamento teve como atribuição aplicar o algoritmo de geolocalização.

Primeiramente ele consultava o estado atual, depois estabelecia as conexões entre o dispositivo móvel e as estações, em seguida filtrava as três conexões com menor distância, por fim aplicava o cálculo da trilateração (3.3), conforme Apêndice B.8.

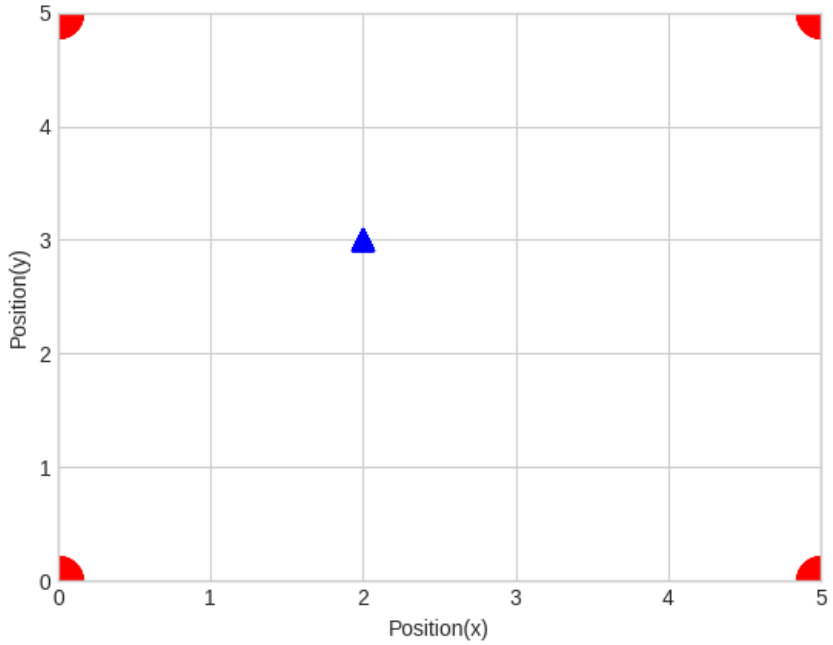
O resultado final era a estimativa de posição relativa do dispositivo móvel no ambiente, que era entregue ao Módulo de Mensageria para consumo do cliente.

3.6.5 Cliente

O cliente poderia ser qualquer aplicação ou serviço capaz de consumir o resultado da estimativa de posição e apresentá-la ao usuário.

Posto isso, foi desenvolvido um módulo para receber as estimativas de posição do módulo de mensageria, e a medida que elas eram entregues estas eram apresentadas graficamente em um plano cartesiano com as mesmas dimensões do espaço real. Dessa forma foi possível acompanhar a posição do dispositivo móvel em tempo real.

Figura 11 – Representação gráfica da posição



Fonte: O Autor.

O módulo desenvolvido foi baseado na biblioteca Matplotlib (2.7.3).

4 RESULTADOS E DISCUSSÃO

Este trabalho apresentou um Sistema de Posicionamento *Indoor* com aspecto multidisciplinar, englobando conceitos de Automação, Eletrônica e Computação. Foi também abordado um tema de grande relevância atualmente, que é a Internet das Coisas (IoT), um dos pilares da Indústria 4.0. A metodologia aplicada tentou envolver ao máximo os itens comentados nos objetivos geral e específico, alcançando a meta de construir um sistema com hardware e software integrados de baixo custo e viável de ser implementado utilizando ferramentas de código aberto e licença pública.

As tabelas 1, 2, 3, 4 e 5 mostram os resultados obtidos ao calcular a distância baseada no RSSI, considerado o *path loss* igual a 2, que é o recomendado para ambientes que dispõe de espaço livre (KUROSE; ROSS, 2006). Elas mostram o erro médio absoluto e o erro percentual médio absoluto do conjunto de dados, composto por 10 amostras em cada referência, sendo elas em 1 metro, 2 metros, 3 metros, 4 metros e 5 metros, respectivamente.

Pode-se notar que para distâncias de 1 a 3 metros os resultados são bastante exatos para o objeto deste trabalho. Porém, para distâncias maiores tem-se um erro significativo.

Tabela 1 – Comparação entre as estimativas de distância e a referência - 1 metro

	MAE	$MAPE(\%)$
Distância(m)	0.07	7.00

Tabela 2 – Comparação entre as estimativas de distância e a referência - 2 metros

	MAE	$MAPE(\%)$
Distância(m)	0.15	8.00

Tabela 3 – Comparação entre as estimativas de distância e a referência - 3 metros

	MAE	$MAPE(\%)$
Distância(m)	0.53	18.00

Tabela 4 – Comparação entre as estimativas de distância e a referência - 4 metros

	MAE	$MAPE(\%)$
Distância(m)	0.91	23.00

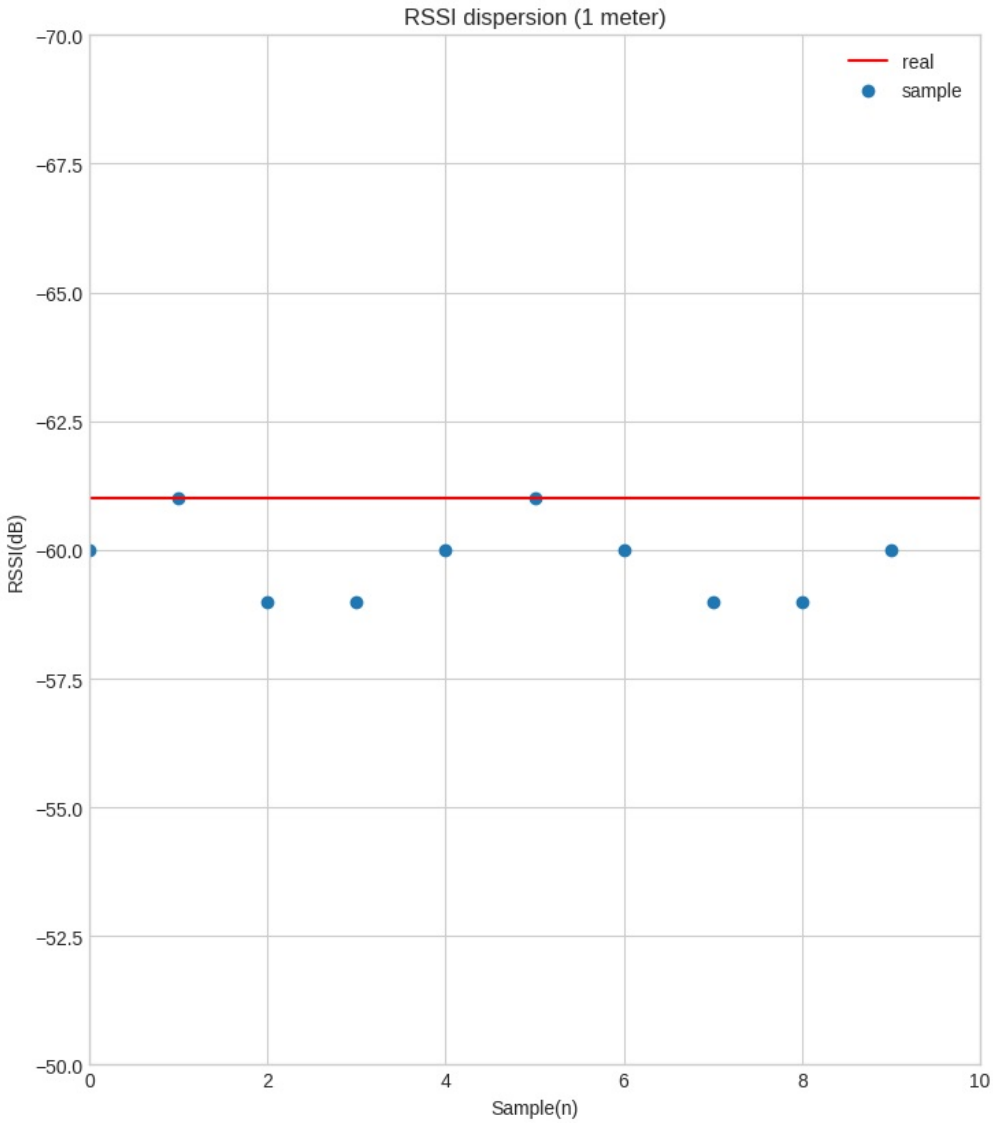
Tabela 5 – Comparação entre as estimativas de distância e a referência - 5 metros

	MAE	$MAPE(\%)$
Distância	2.91	58.00

As figuras 12, 13, 14, 15, 16, 17, 18, 19, 20, 21 mostram as dispersões das medidas de RSSI e das distâncias estimadas.

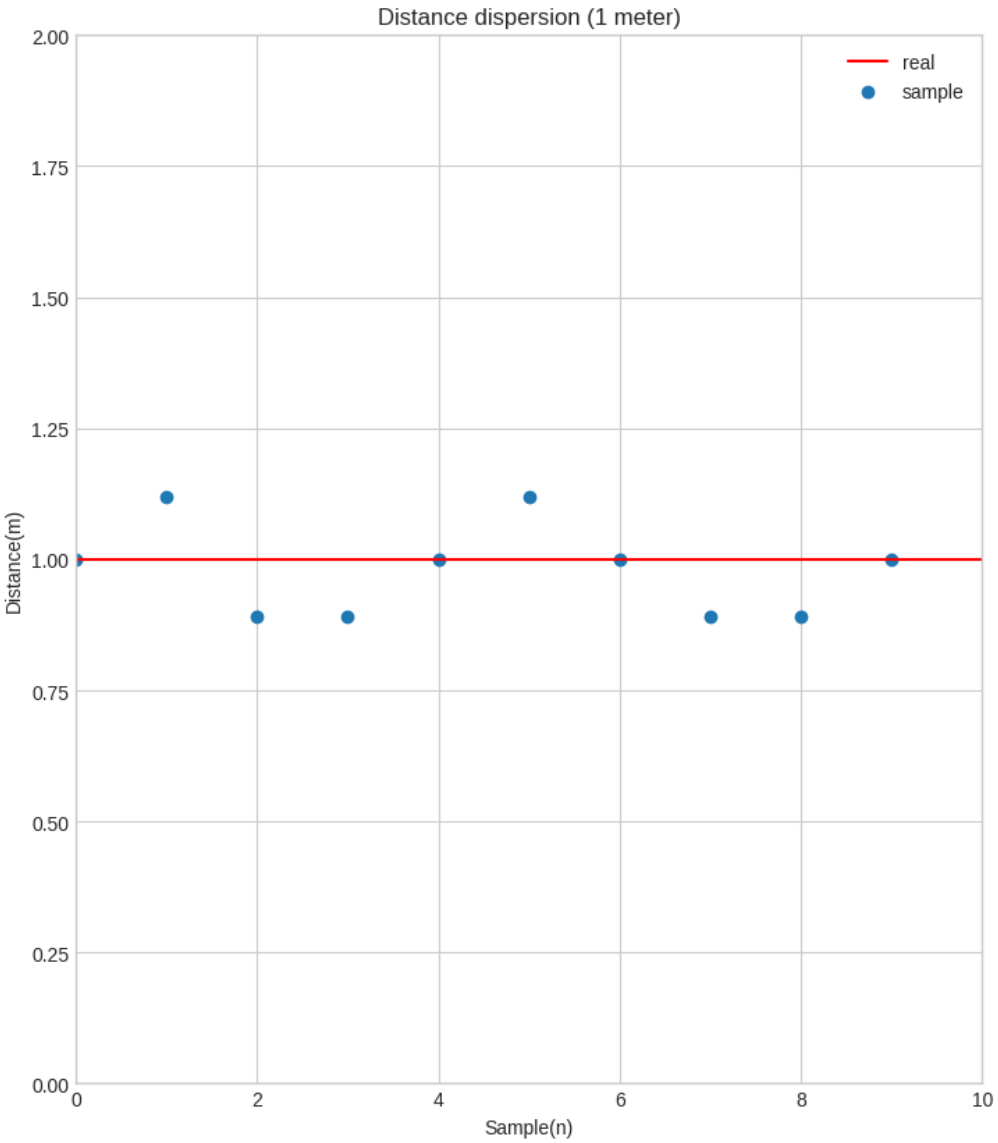
Pode-se observar que as amostras em 1 metro e 2 metros são bastante concentradas em torno da média, apresentando baixa dispersão. A partir de 3 metros a dispersão aumenta significativamente. É possível notar que uma pequena variação no RSSI implica numa grande variação na distância estimada, em virtude da escala logarítmica.

Figura 12 – Dispersão do sinal RSSI(dB) em relação à referência de 1 metro



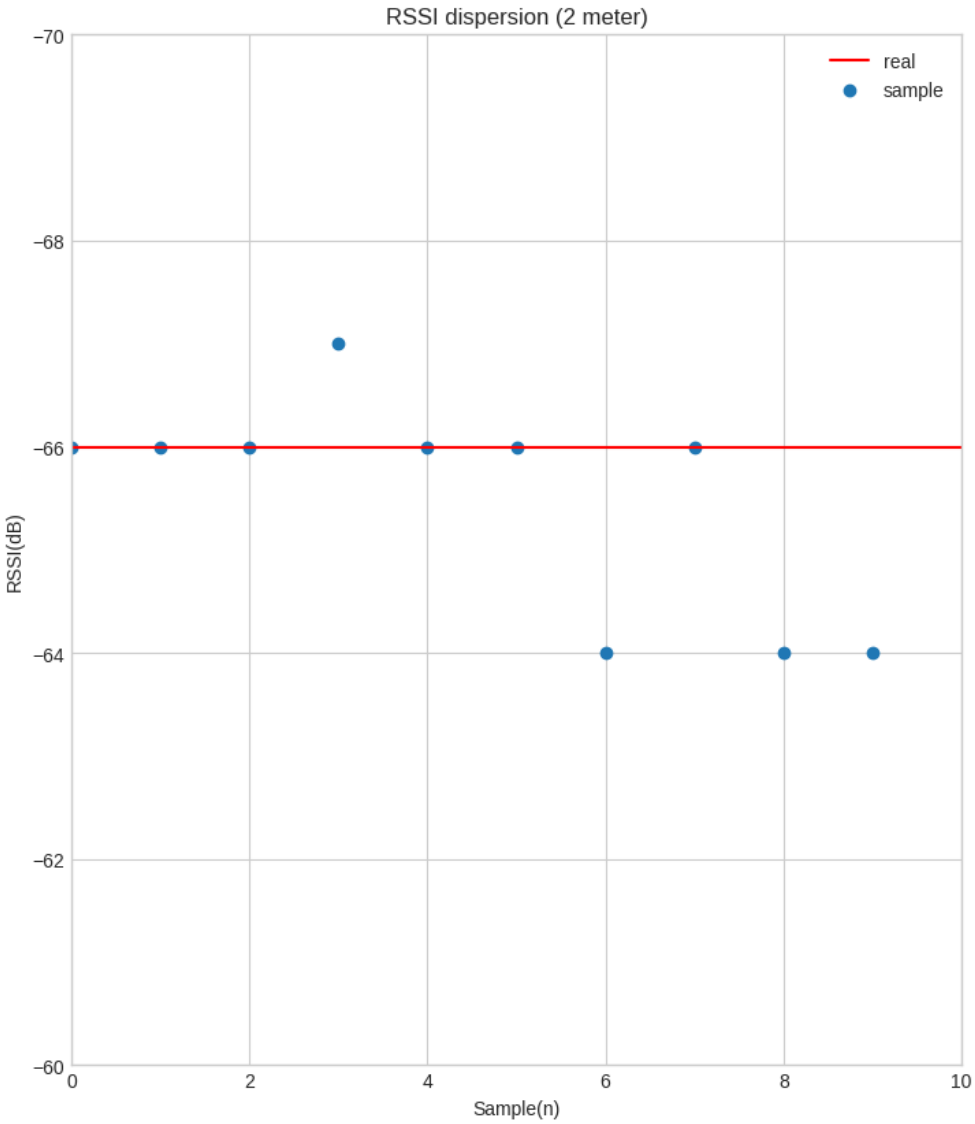
Fonte: O Autor.

Figura 13 – Dispersão da distância(m) estimada em relação à referência de 1 metro



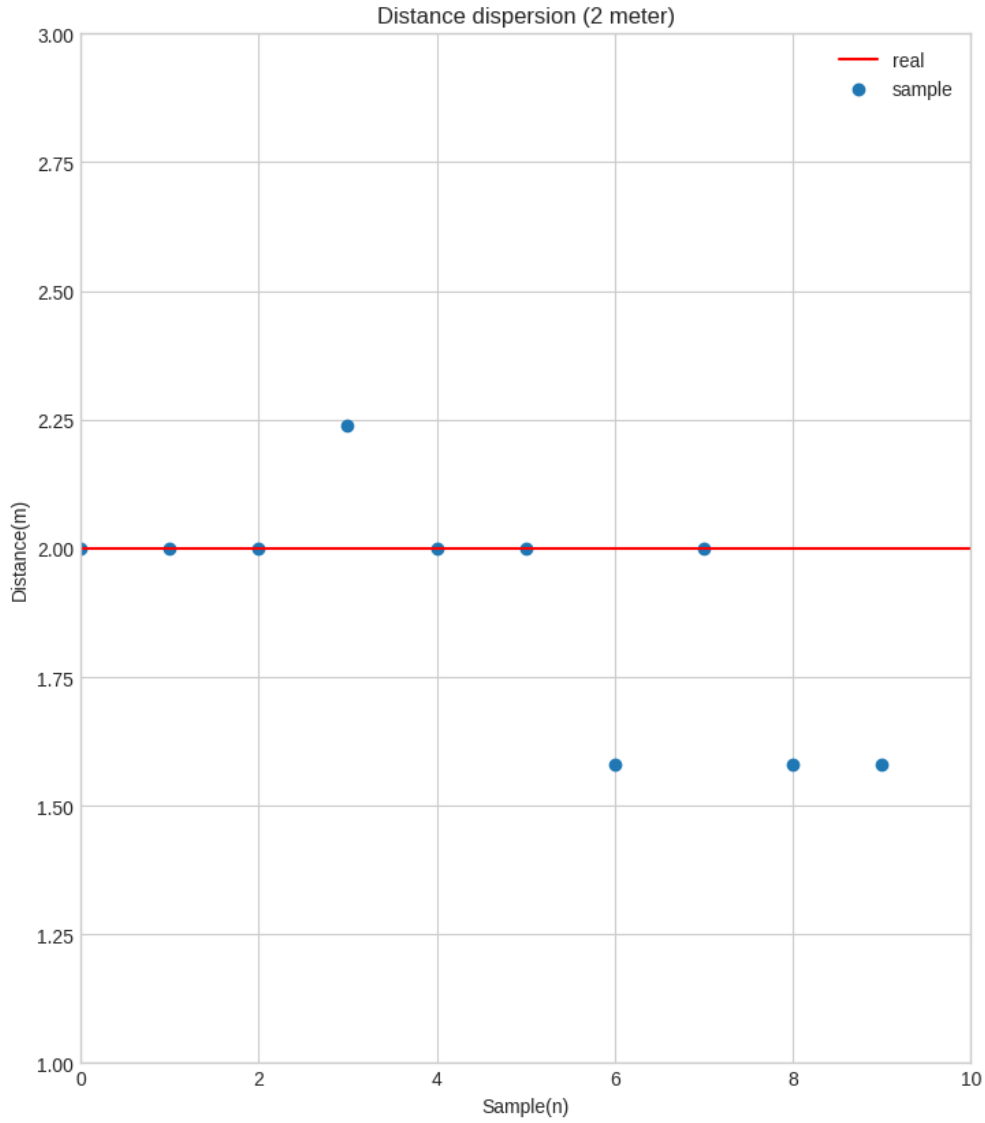
Fonte: O Autor.

Figura 14 – Dispersão do sinal RSSI(dB) em relação à referência de 2 metros



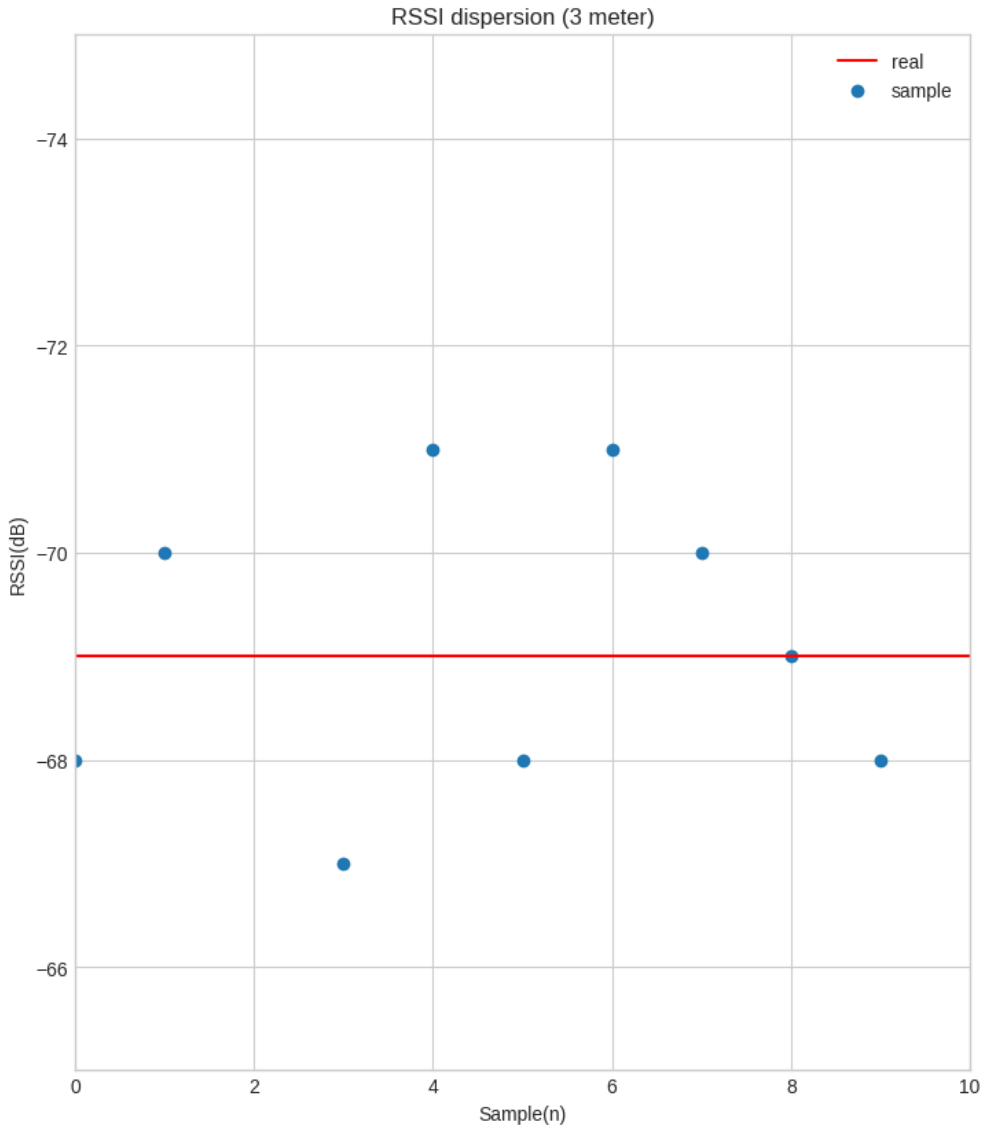
Fonte: O Autor.

Figura 15 – Dispersão da distância(m) estimada em relação à referência de 2 metros



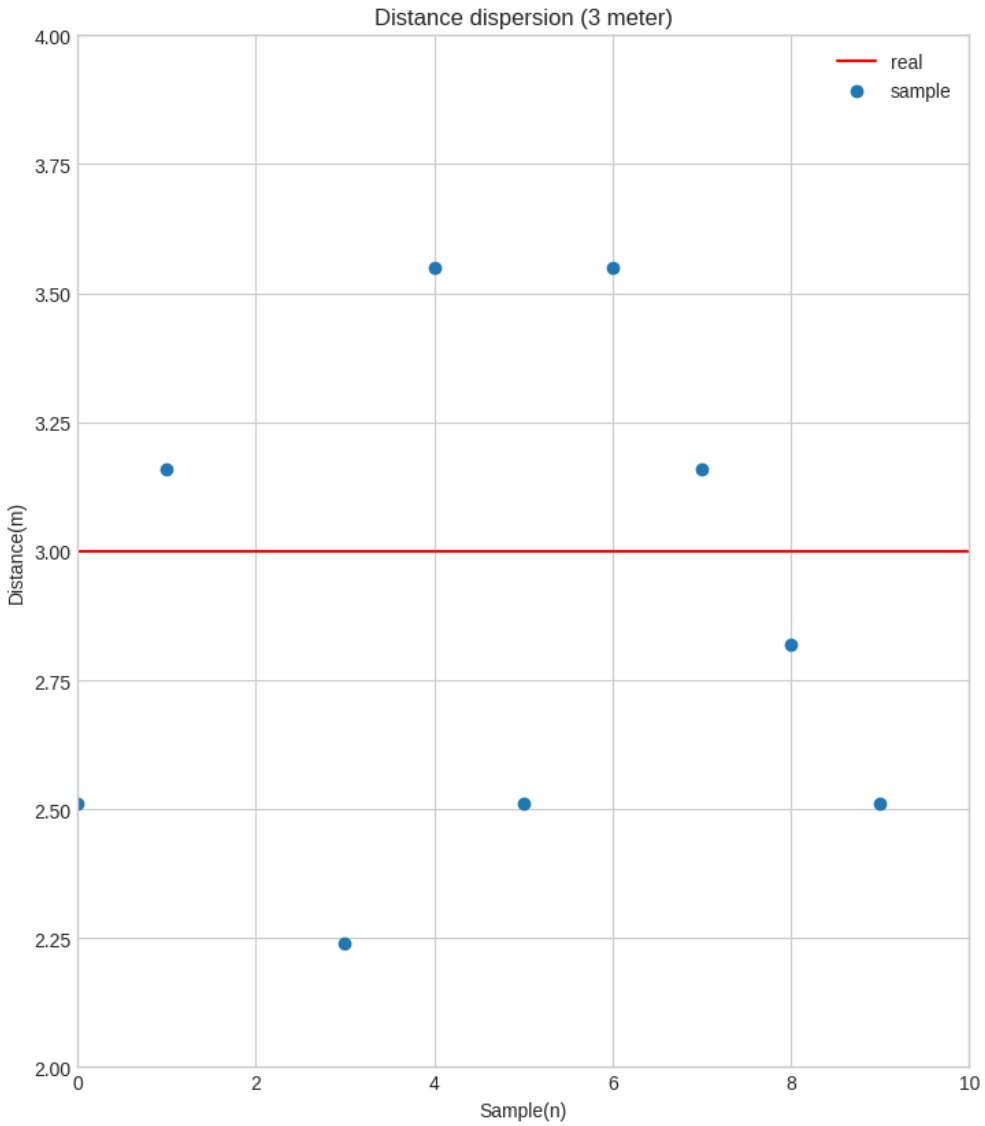
Fonte: O Autor.

Figura 16 – Dispersão do sinal RSSI(dB) em relação à referência de 3 metros



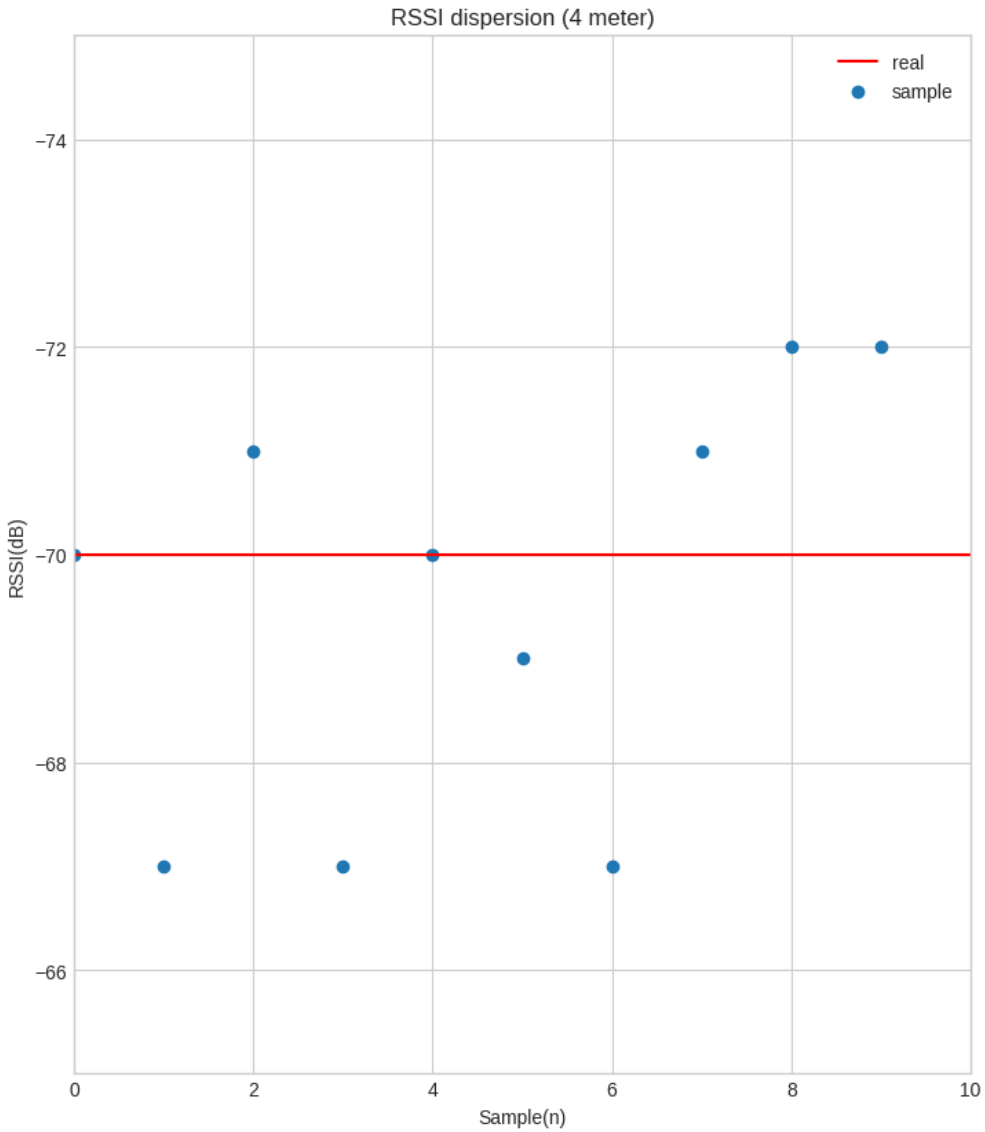
Fonte: O Autor.

Figura 17 – Dispersão da distância(m) estimada em relação à referência de 3 metros



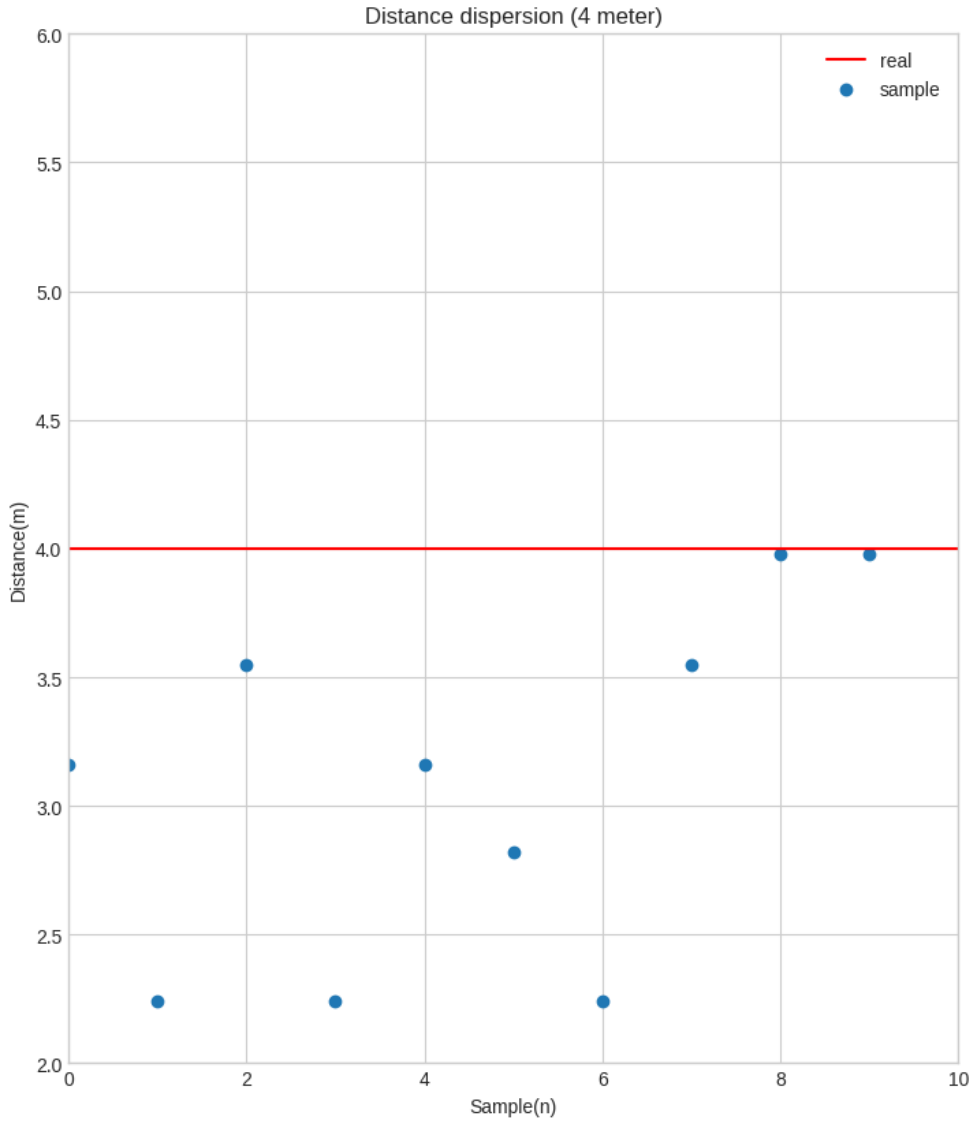
Fonte: O Autor.

Figura 18 – Dispersão do sinal RSSI(dB) em relação à referência de 4 metros



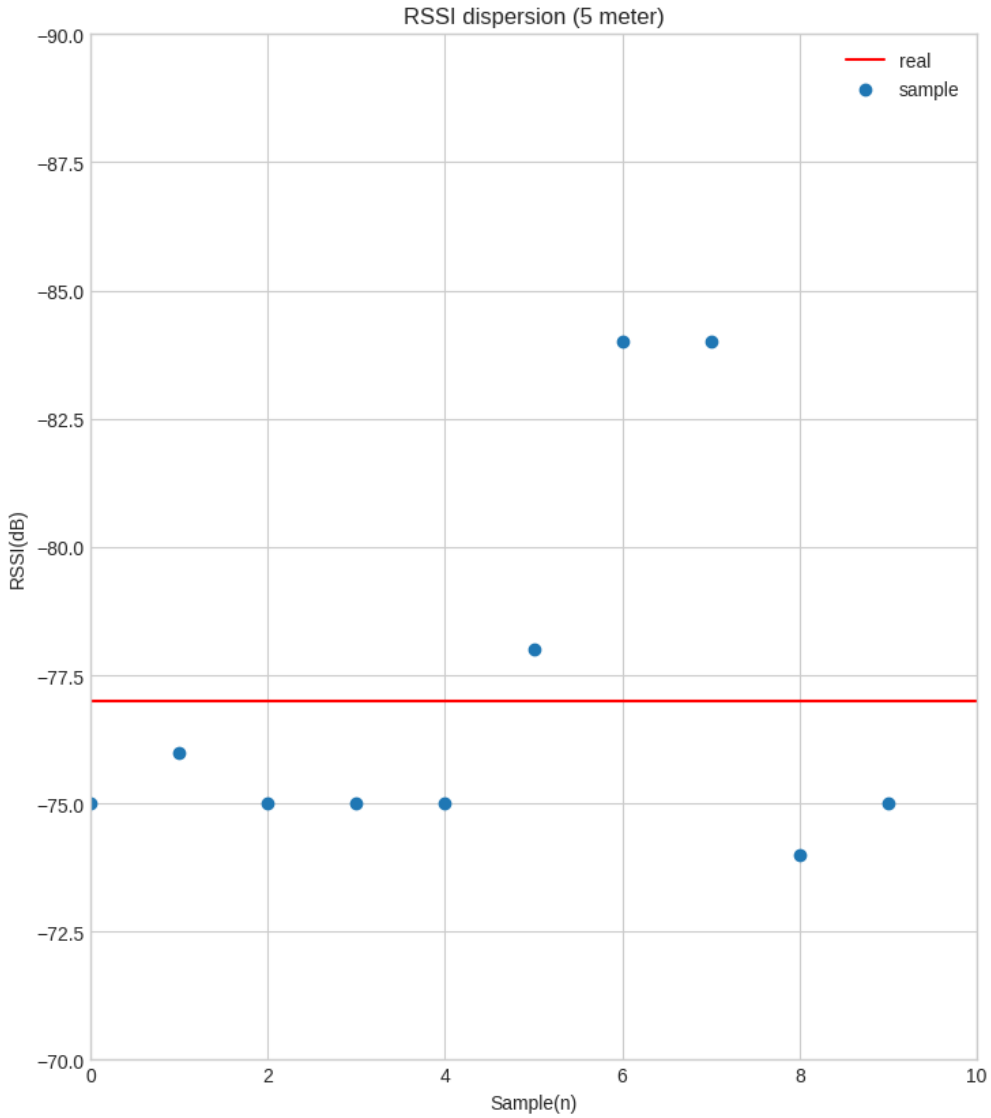
Fonte: O Autor.

Figura 19 – Dispersão da distância(m) estimada em relação à referência de 4 metros



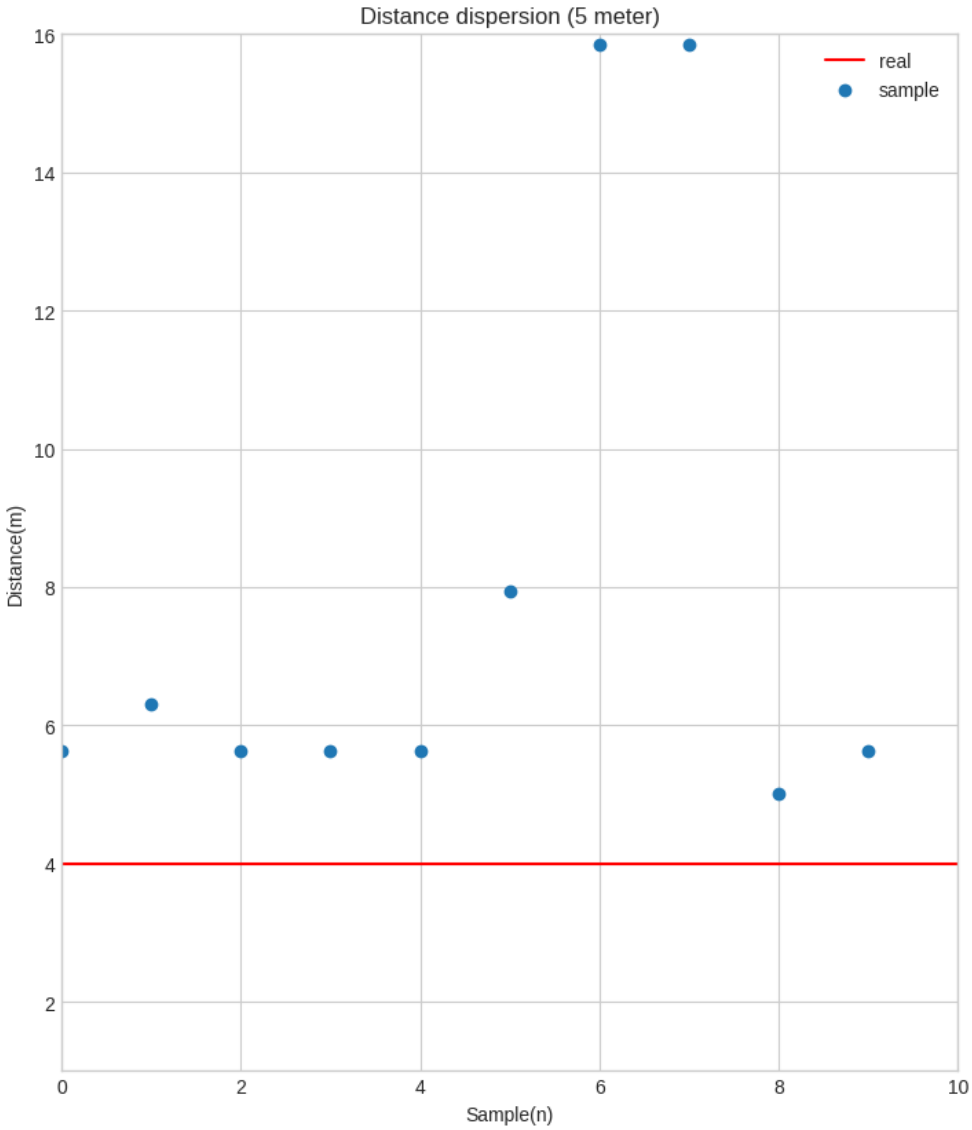
Fonte: O Autor.

Figura 20 – Dispersão do RSSI(dB) em relação à referência de 5 metros



Fonte: O Autor.

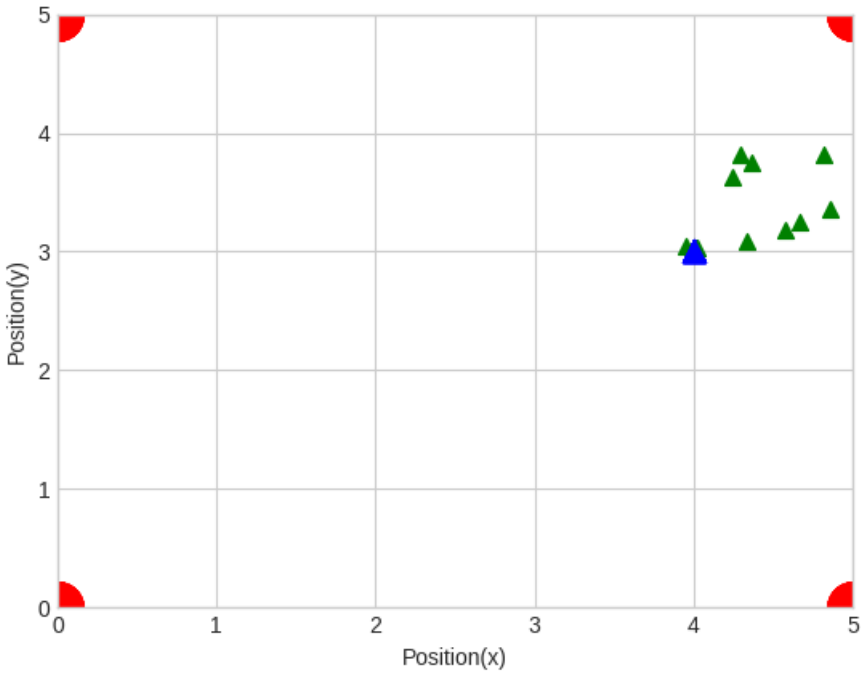
Figura 21 – Dispersão da distância(m) estimada em relação à referência de 5 metros



Fonte: O Autor.

Na Figura 22 é apresentado um conjunto de 10 amostras de estimativa de posição em torno da referência. O resultado apresentou ótima precisão, mantendo as estimativas de posição dentro do mesmo metro quadrado, tendo em vista que o objetivo geral é oferecer estimativas de posição em um raio de até 3m da referência. Portanto, comprova-se que os objetivos propostos inicialmente foram alcançados.

Figura 22 – Estimativas de localização em torno da referência



Fonte: O Autor.

5 CONCLUSÃO

Neste trabalho foi desenvolvido um sistema de localização *indoor* baseado em Bluetooth Low Energy, utilizando ferramentas de código aberto e hardwares de licença pública. Os testes foram realizados em laboratório e verificou-se que as estimativas de posição apresentadas pelo sistema foram bastante próximas das posições reais.

Com os resultados apresentados anteriormente, pode-se concluir que houve uma solução eficaz na busca de estimar com precisão a localização em ambiente interno. As ferramentas de código aberto e hardwares de licença pública se mostraram excelentes soluções de versatilidade e viabilidade de implementação. Este projeto pode ser aplicado com baixo custo, comparado as soluções oferecidas por grandes fabricantes, e sem grandes intervenções no ambiente.

Porém, foram identificadas algumas dificuldades ao longo do trabalho que são objetos consideráveis para seguir a pesquisa. A natureza sensível do RSSI é um fato a ser observado em ambientes onde há fluxo de pessoas ou que não há espaço livre. Lugares com grandes interferências eletromagnéticas também devem ser observados. Outro fato relevante é que a construção do hardware das estações de monitoramento não é dedicada para a aplicação de beacon BLE, com isso as estações apresentam um alto gasto energético, que contrasta com a grande autonomia dos beacons BLE oferecidos pelos grande fabricantes no mercado.

Pode-se perceber que para longas distâncias o sinal torna-se pouco confiável. Para esses casos é indicado inserir antenas externas ou aumentar a quantidade de estações para que a distância seja sempre a mais curta possível.

O sinal de RSSI funciona bem para objetos estáticos, porém demora a estabilizar quando o objeto está em movimento. Para esse caso um filtro baseado em média móvel pode ajudar atenuar a varição do sinal.

Por fim, conclui-se que os resultados obtidos foram bastante precisos considerando que o objetivo geral deste trabalho que é oferecer estimativas de posição em um raio de até 3m da referência.

5.1 Trabalhos Futuros

O cálculo da distância baseado em RSSI esbarra em um grande obstáculo que é a flutuação do sinal devido às interferências do ambiente. Um objeto de pesquisa interessante é a leitura bidirecional do RSSI (Bellecieri; Jabour; Jabour, 2015). Essa abordagem considera também o RSSI que o beacon recebe das estações, portanto poderia ser utilizado para minimizar o erro das estimativas.

Outra ideia para trabalhos futuros é aplicação de outros métodos de estimativa de posição

como, por exemplo, métodos probabilísticos para efeito de comparação.

REFERÊNCIAS

ARDUINO. *Arduino*. 2018. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 15th February 2022. Citado na página 23.

ATHOSELECTRONICS. *Placa ESP32*. 2021. Disponível em: <https://athoselectronics.com/esp32/>. Acesso em: 21th March 2021. Citado na página 19.

AZIZ, M. I.; OWENS, T.; ZAMAN, U. K. uz. Rssi based localization of bluetooth devices using trilateration: An improved method for the visually impaired. In: *2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. [S.l.: s.n.], 2018. p. 1–5. Citado na página 12.

Bellecieri, Y.; Jabour, F. C.; Jabour, E. G. Localização indoor baseada na leitura bidirecional do rssi. 2015. Citado 2 vezes nas páginas 24 e 46.

Cay, E. et al. Beacons for indoor positioning. In: *2017 International Conference on Engineering and Technology (ICET)*. [S.l.: s.n.], 2017. p. 1–5. Citado na página 12.

DARPA. *Defense Advanced Research Projects Agency*. 2022. Disponível em: <https://www.darpa.mil/about-us/timeline/transit-satellite>. Acesso em: 3rd June 2022. Citado na página 12.

FILÍPEK, P.; KOVAROVA, A. Indoor localization based on beacons and calculated by particle filter. In: *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*. New York, NY, USA: ACM, 2016. (CompSysTech '16), p. 269–276. ISBN 978-1-4503-4182-0. Disponível em: <http://doi.acm.org/10.1145/2983468.2983507>. Citado na página 13.

GPS. *GPS*. 2022. Disponível em: <https://www.gps.gov/systems/gps/space/>. Acesso em: 3rd June 2022. Citado na página 12.

IBEACONINSIDER. *iBeacon*. 2021a. Disponível em: <http://www.ibeacon.com/what-is-ibeacon-a-guide-to-beacon/>. Acesso em: 29th March 2021. Citado na página 16.

IBEACONINSIDER. *Anuncios e ofertas através dos iBeacons*. 2021b. Disponível em: <http://www.ibeacon.com/what-is-ibeacon-a-guide-to-beacon/>. Acesso em: 29th March 2021. Citado na página 17.

KONTAKT.IO. *O que são beacons e como eles trabalham*. 2021. Disponível em: <https://kontakt.io/what-is-a-beacon/>. Acesso em: 29th March 2021. Citado na página 18.

KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet: Uma abordagem top-down*. Trad. 3 ed. São Paulo: Addison Wesley, 2006. Citado 2 vezes nas páginas 24 e 34.

Li, X. et al. Design and implementation of indoor positioning system based on ibeacon. In: *2016 International Conference on Audio, Language and Image Processing (ICALIP)*. [S.l.: s.n.], 2016. p. 126–130. Citado na página 12.

MATPLOTLIB. *Matplotlib*. 2022. Disponível em: <https://matplotlib.org/>. Acesso em: 8th June 2022. Citado na página 23.

MOSQUITTO. *Eclipse Mosquitto*. 2022. Disponível em: <https://mosquitto.org/>. Acesso em: 8th June 2022. Citado na página 23.

NOERTJAHYANA, A.; WIJAYANTO, I. A.; ANDJARWIRAWAN, J. Development of mobile indoor positioning system application using android and bluetooth low energy with trilateration method. In: *2017 International Conference on Soft Computing, Intelligent System and Information Technology (ICSIIT)*. [S.l.: s.n.], 2017. p. 185–189. Citado na página 12.

QIAN, H. et al. A novel loss recovery and tracking scheme for maneuvering target in hybrid wsns. *Sensors*, v. 18, p. 341, 01 2018. Citado na página 25.

SAMSUNG. *Smartphone Galaxy Samsung A71*. 2022. Disponível em: <https://www.samsung.com/br/smartphones/galaxy-a/galaxy-a71-black-128gb-sm-a715fzkjzto/>. Acesso em: 1st June 2022. Citado na página 29.

SAVVIDES C. C. HAN, M. B. S. A. Dynamic fine grained localization in ad-hoc sensor networks. in proceedings of acm mobile communications (mobicom). 2001. Citado na página 25.

Wang, Q. et al. Improving rss-based ranging in los-nlos scenario using gmms. *IEEE Communications Letters*, v. 15, n. 10, p. 1065–1067, 2011. ISSN 2373-7891. Citado na página 12.

YANG, B. et al. A novel trilateration algorithm for rssi-based indoor localization. *IEEE Sensors Journal*, v. 20, n. 14, p. 8164–8172, 2020. Citado na página 25.

YUAN, M. *Conhecendo o MQTT*. 2017. Disponível em: <https://developer.ibm.com/br/technologies/iot/articles/iot-mqtt-why-good-for-iot/>. Acesso em: 29th March 2021. Citado na página 21.

APÊNDICE A – CÓDIGO-FONTE DO MÓDULO ESP32

Código-fonte das estações de monitoramento implementadas com ESP32.

A.1 Código-fonte do programa principal

```
/**
   @file main.cpp
   @brief Beacon Station ESP32
   @warning
   @todo esp sleep; diagnostic tool (connections, temperature and so on);
    watchdogs
   @bug
   @copyright Heitor Novais
*/
#include <Arduino.h>
#include <vector>
#include <sstream>
#include <string>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEBeacon.h>
#include "Connectivity.h"
#include <PubSubClient.h>
/*****************************************************************
    DEFINES
*****************************************************************/
#define SCAN_TIME 1
#define BEACON_UUID "16bffbc8-8ff8-11ea-bc55-0242ac130002"
/*****************************************************************
    TYPEDEFS
*****************************************************************/
/**
   @brief Struct with Location by station
*/
typedef struct {
  float x = 0.0;
  float y = 0.0;
} Location;
/**
   @brief Struct with MAC Address and RSSI of finded beacon
*/
typedef struct {
  char name[20];
```

```
39   char address[17];  // ex.: 67:f1:d2:04:cd:5d
40   int rssi = 0;
41   int txPower = 0;
42   char manufacturer[20];
43 } BeaconData;
44 /*************************************************************
45     GLOBAL VARIABLES
46 *************************************************************/
47 /**
48    @brief Buffer of finded beacons
49 */
50 std::vector<BeaconData> buffer;
51 /**
52    @brief Advertising object
53 */
54 BLEAdvertising *pAdvertising;
55 /**
56    @brief Station location objetct
57 */
58 Location location;
59 /**
60    @brief Station name
61 */
62 const char* stationName = "station1";
63 /**
64    @brief Advertising appearance name
65 */
66 const char* appearanceName = "ESP32-01";
67 /*************************************************************
68     CLASSES
69 *************************************************************/
70 /**
71    @brief Define callback class
72 */
73 class MyAdvertisedDeviceCallbacks : public BLEAdvertisedDeviceCallbacks
74 {
75   public:
76     /**
77        @brief Callback function
78     */
79     void onResult(BLEAdvertisedDevice advertisedDevice)
80     {
81       extern std::vector<BeaconData> buffer;
82       BeaconData beacon;
83 
84       if (advertisedDevice.haveRSSI())
85       {
```

```
86          beacon.rssi = advertisedDevice.getRSSI();
87          strcpy(beacon.address, advertisedDevice.getAddress().toString().
    c_str());
88        }
89        else
90          return;
91
92        if (advertisedDevice.haveName())
93          strcpy(beacon.name, advertisedDevice.getName().c_str());
94
95        if (advertisedDevice.haveTXPower())
96          beacon.txPower = advertisedDevice.getTXPower();
97
98        if (advertisedDevice.haveManufacturerData())
99          strcpy(beacon.manufacturer, advertisedDevice.getManufacturerData().
    c_str());
100
101       buffer.push_back(beacon);
102
103       // Print everything via serial port for debugging
104       Serial.printf("Name: %s \n", advertisedDevice.getName().c_str());
105       Serial.printf("MAC: %s \n", advertisedDevice.getAddress().toString().
    c_str());
106       Serial.printf("RSSI: %d \n", advertisedDevice.getRSSI());
107       Serial.printf("TXPower: %d \n", advertisedDevice.getTXPower());
108       Serial.printf("ManufacturerData: %s \n", advertisedDevice.
    getManufacturerData());
109     }
110 };
111 /*****************************************************************
112     LOCAL PROTOTYPES
113 *****************************************************************/
114 /**
115    @brief Set station location
116 */
117 void setLocation();
118 /**
119    @brief Configure advertising
120 */
121 void setBeacon();
122 /**
123    @brief Start advertising
124 */
125 void startAdvertising();
126 /**
127    @brief Discover beacons
128 */
```

```
129 void scanBeacons();
130 /**
131    @brief Mount JSON message
132 */
133 String mountMessage();
134 /**
135    @brief Validade beacon
136 */
137 bool validadeBeacon(BeaconData beacon);
138 /*****************************************************************
139     IMPLEMENTATION
140 *****************************************************************/
141 void setup()
142 {
143   Serial.begin(115200);
144   setLocation(0.0, 10.0);
145   connectWiFi();
146   connectMQTT();
147   BLEDevice::init("");
148   startAdvertising();
149 }
150 
151 void loop()
152 {
153   Serial.println("Devices found:");
154   scanBeacons();
155   checkConnections();
156   client.loop();
157   String message = mountMessage();
158   bool result = client.publish("station_data", message.c_str(), true);
159   Serial.print("PUB Result: ");
160   Serial.println(result ? "Success!" : "Failed!");
161   Serial.println("--\n");
162   buffer.clear();
163 }
164 /*****************************************************************
165     FUNCTIONS
166 *****************************************************************/
167 /**
168    @brief Set station location
169 */
170 void setLocation(float x, float y)
171 {
172   location.x = x;
173   location.y = y;
174 }
175 /**
```

```
176    @brief Configure advertising
177 */
178 void setBeacon()
179 {
180   BLEBeacon oBeacon = BLEBeacon();
181   oBeacon.setManufacturerId(0x4C00); // fake Apple 0x004C LSB (
       ENDIAN_CHANGE_U16!)
182   oBeacon.setProximityUUID(BLEUUID(BEACON_UUID));
183   oBeacon.setMajor((1 & 0xFFFF0000) >> 16);
184   oBeacon.setMinor(1 & 0xFFFF);
185   BLEAdvertisementData oAdvertisementData = BLEAdvertisementData();
186   BLEAdvertisementData oScanResponseData = BLEAdvertisementData();
187
188   oAdvertisementData.setFlags(0x04); // BR_EDR_NOT_SUPPORTED 0x04
189   oAdvertisementData.setName(appearanceName);
190
191   std::string strServiceData = "";
192
193   strServiceData += (char)26;     // Len
194   strServiceData += (char)0xFF;   // Type
195   strServiceData += oBeacon.getData();
196   oAdvertisementData.addData(strServiceData);
197
198   pAdvertising->setAdvertisementData(oAdvertisementData);
199   pAdvertising->setScanResponseData(oScanResponseData);
200 }
201 /**
202    @brief Start advertising
203 */
204 void startAdvertising()
205 {
206   pAdvertising = BLEDevice::getAdvertising();
207   setBeacon();
208   pAdvertising->start(); // Start advertising
209   Serial.println("Advertizing started...");
210 }
211 /**
212    @brief Discover beacons
213 */
214 void scanBeacons()
215 {
216   BLEScan *pBLEScan = BLEDevice::getScan(); //create new scan
217   MyAdvertisedDeviceCallbacks cb;
218   pBLEScan->setAdvertisedDeviceCallbacks(&cb);
219   pBLEScan->setActiveScan(true); //active scan uses more power, but get
       results faster
220
```

```
221   BLEScanResults foundDevices = pBLEScan->start(SCAN_TIME);
222
223   // Stop BLE
224   pBLEScan->stop();
225   Serial.println("Scan done!");
226   Serial.println();
227 }
228 /**
229    @brief Mount JSON message
230 */
231 String mountMessage()
232 {
233   /*
234    * Ex.:
235      {\"name\":\"station1\",\"mac\":\"aa:bb:cc:dd:aa\",\"manufecturer\":\"
     espressif\",\"location\":{\"x\":0,\"y\":0},\"beacons_found\":[{\"name
     \":\"beacon1\",\"mac\":\"aa:bb:cc:dd:ee\",\"manufecturer\":\"espressif
     \",\"rssi\":-20,\"tx_power\":-30},{\"name\":\"beacon2\",\"mac\":\"aa:bb:
     cc:dd:ff\",\"manufecturer\":\"espressif\",\"rssi\":-30,\"tx_power
     \":-30}]}"
236   */
237   String json = "{\"name\":\"" + String(stationName) + "\",\"mac\":\"" +
      String(WiFi.macAddress()) + "\",\"manufacturer\":\"" + "espressif" + "
      \",\"location\":{\"x\":" + String(location.x) + ",\"y\":" + String(
      location.y) + "},\"beacons_found\":[";
238   for (uint8_t i = 0; i < buffer.size(); i++)
239   {
240     BeaconData beacon = buffer.at(i);
241     if (validadeBeacon(beacon))
242     {
243       json += "{";
244       json += "\"name\":\"";
245       json += String(beacon.name) + "\",";
246       json += "\"mac\":\"";
247       json += String(beacon.address) + "\",";
248       json += "\"manufacturer\":\"";
249       json += String(beacon.manufacturer) + "\",";
250       json += "\"rssi\":";
251       json += String(beacon.rssi) + ",";
252       json += "\"tx_power\":";
253       json += String(beacon.txPower);
254       json += "}";
255       if (i < buffer.size() - 1)
256         json += ",";
257     }
258   }
259   json += "]}";
```

```
260   Serial.print("Message:");
261   Serial.println(json);
262   Serial.println();
263 
264   return json;
265 }
266 
267 bool validadeBeacon(BeaconData beacon)
268 {
269   if (sizeof(beacon.name) == 0) return false;
270   if (sizeof(beacon.address) == 0) return false;
271   if (sizeof(beacon.manufacturer) == 0) return false;
272   if (beacon.rssi > 0) return false;
273   return true;
274 }
```

A.2 Código-fonte do arquivo com as credenciais de rede e caminho do servidor

```
1  /**
2   * @file Credentials.h
3   * @brief Credentials for connecting to the server
4   * @copyright Heitor Novais
5  */
6 
7  #ifndef CREDENTIALS_H_
8  #define CREDENTIALS_H_
9 
10 /*****************************************************************
11 *   CONSTANTS
12 *****************************************************************/
13 /**
14  * WiFi credentials
15 */
16 const char *ssid = "nome_da_rede";
17 const char *password = "senha";
18 
19 /**
20  * MQTT credentials
21 */
22 const char *mqttServer = "192.168.0.12";
23 const uint16_t mqttPort = 1883;
24 const char *mqttClient = "ESP32-1";
25 const char *mqttTopico = "station_data";
26 
27 #endif
```

A.3 Código-fonte do arquivo de cabeçalho da interface de rede do programa principal

```
#ifndef CONNECTIVITY_H_
#define CONNECTIVITY_H_

#include <WiFi.h>
#include <PubSubClient.h>

extern WiFiClient esp32;
extern PubSubClient client;


void connectWiFi();

void connectMQTT();

void checkConnections();

#endif
```

A.4 Código-fonte do arquivo do arquivo de interface de rede do programa principal

```
/**
 * @file Connectivity.cpp
 * @brief Connects ESP32 station to server
 * @copyright Heitor Novais
*/

#include <Arduino.h>
#include "Connectivity.h"
#include "Credentials.h"

/*****************************************************************
*   GLOBAL VARIABLES
*****************************************************************/
WiFiClient esp32;
PubSubClient client(esp32);

/*****************************************************************
*   BODY FUNCTIONS
*****************************************************************/
void connectWiFi()
{
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
```

```
26      Serial.println("Connecting to WiFi..");
27    }
28    Serial.println("Connected to the WiFi network");
29    Serial.println("IP address: ");
30    Serial.println(WiFi.localIP());
31    Serial.print("ESP Board MAC Address:  ");
32    Serial.println(WiFi.macAddress());
33  }
34
35  void connectMQTT()
36  {
37    client.setServer(mqttServer, mqttPort);
38    Serial.println("Connecting to MQTT...");
39    if (client.connect(mqttClient))
40    {
41      Serial.println("Connected!");
42    }
43    else
44    {
45      Serial.print("failed with state ");
46      Serial.println(client.state());
47      delay(2000);
48    }
49  }
50
51  /**
52   * Attempts to reconnect if disconnected
53  */
54  void checkConnections()
55  {
56    while (WiFi.status() != WL_CONNECTED)
57    {
58      connectWiFi();
59    }
60
61    while (!client.connected())
62    {
63      connectMQTT();
64    }
65  }
```

APÊNDICE B – CÓDIGO-FONTE DA APLICAÇÃO

Código-fonte da aplicação que calcula o posicionamento do dispositivo móvel.

B.1 Código-fonte da classe BLE

```
class BLE:
    def __init__(self, name: str, mac: str, manufacturer: str,
                 rssi: int = None, tx_power: int = None):
        self._name = name
        self._mac = mac
        self._manufacturer = manufacturer
        self._rssi = rssi
        self._tx_power = tx_power

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def mac(self):
        return self._mac

    @mac.setter
    def mac(self, value):
        self._mac = value

    @property
    def manufacturer(self):
        return self._manufacturer

    @manufacturer.setter
    def manufacturer(self, value):
        self._manufacturer = value

    @property
    def rssi(self):
        return self._rssi

    @rssi.setter
    def rssi(self, value):
```

```
40          self._rssi = value
41
42      @property
43      def tx_power(self):
44          return self._tx_power
45
46      @tx_power.setter
47      def tx_power(self, value):
48          self._tx_power = value
49
50      def __str__(self):
51          return f'Name: {self._name}\nMAC: {self._mac}\nManufecturer: {self.
    _manufecturer}'
```

B.2 Código-fonte da classe Station

```
1  from ble import BLE
2  from beacon import Beacon
3  from location import Location
4
5
6  class Station(BLE):
7      def __init__(self, name: str, mac: str, manufecturer: str,
8                   location: Location):
9          super().__init__(name, mac, manufecturer)
10         self._beacons_found: dict = {}
11         self._location = location
12
13     @staticmethod
14     def parse(msg: dict):
15         if msg is None:
16             raise NotImplementedError()
17
18         station_attributes = list(vars(BLE).keys()) + \
19             list(vars(Station).keys())
20         msg_attributes = msg.keys()
21
22         is_station = True
23         for atrribute in msg_attributes:
24             if atrribute not in station_attributes:
25                 is_station = False
26                 break
27
28         if is_station:
29             station = Station(msg['name'],
30                               msg['mac'],
31                               msg['manufacturer'],
```

```
32                          Location(msg['location']['x'], msg['location']['y'
    ]))
33
34              for item in msg['beacons_found']:
35                  if (item['name'].startswith('ESP32')):
36                      continue
37
38                  station.add_beacon(Beacon(
39                      item['name'],
40                      item['mac'],
41                      item['manufacturer'],
42                      item['rssi'],
43                      item['tx_power']))
44
45              return station
46          else:
47              return None
48
49      def add_beacon(self, beacon: Beacon):
50          self._beacons_found[beacon.mac] = beacon
51
52      @property
53      def beacons_found(self):
54          return self._beacons_found
55
56      @property
57      def location(self) -> Location:
58          return self._location
59
60      def __str__(self):
61          return f'Station: {self._name} Location: {self._location}'
```

B.3 Código-fonte da classe Beacon

```
1  from ble import BLE
2
3
4  class Beacon(BLE):
5      def __init__(self, name: str, mac: str, manufacturer: str,
6                   rssi: int = None, tx_power: int = None):
7          super().__init__(name, mac, manufacturer, rssi, tx_power)
8
9      @staticmethod
10     def parse(msg: dict):
11         if msg is None:
12             raise NotImplementedError()
13
```

```
        beacon_attributes = list(vars(BLE).keys()) + \
            list(vars(Beacon).keys())
        msg_attributes = vars(msg).keys()

        is_beacon = True
        for atrribute in msg_attributes:
            if atrribute[1:] not in beacon_attributes:
                is_beacon = False
                break

        if is_beacon:
            return Beacon(msg['name'],
                          msg['mac'],
                          msg['manufacturer'],
                          msg['rssi'],
                          msg['tx_power'])
        else:
            return None


    def __str__(self):
        return f'Beacon: {self._name} MAC: {self._mac} Manufecturer: {self._manufacturer}'
```

B.4 Código-fonte da classe Location

```
class Location:
    def __init__(self, x: float, y: float):
        self._x = x
        self._y = y

    @property
    def x(self) -> float:
        return self._x

    @property
    def y(self) -> float:
        return self._y

    def __str__(self):
        return "(X: {:.2f} Y: {:.2f})".format(self._x, self._y)
```

B.5 Código-fonte da classe Link

```
from station import Station
from beacon import Beacon
```

```
from typing import Optional

class Link:
    def __init__(self, station: Station, beacon_mac: str):
        self._station = station
        self._beacon = self._station.beacons_found[beacon_mac]

    @property
    def station(self) -> Station:
        return self._station

    @property
    def beacon(self) -> str:
        return self._beacon

    @property
    def distance(self) -> Optional[float]:
        measured_power = -60
        n = 2
        return 10**((measured_power - self._beacon.rssi) / (10*n))

    def __str__(self):
        return f'{self._beacon.rssi};{round(self.distance, 2)}'
```

B.6 Código-fonte do Módulo de Mensageria

```
import ble_services
import paho.mqtt.client as mqtt
import utils
import threading
import positioning
import view
from location import Location
from random import randrange

def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    client.subscribe("test/topic/#")


def on_message(client, userdata, msg):
    print(f'[on_message] {msg.topic} {str(msg.payload)}')
    payload_contents = utils.get_payload_contents(msg.payload)
    ble_services.refresh_devices(payload_contents)
    locations = positioning.run()
    view.show(locations)

```

```
22
23
24 client = mqtt.Client()
25 client.on_connect = on_connect
26 client.on_message = on_message
27
28 t1 = threading.Timer(1.0, ble_services.show_devices)
29 t1.start()
30
31 client.connect("localhost", 1883, 60)
32 try:
33     client.loop_forever()
34 except Exception:
35     client.loop_forever()
```

B.7 Código-fonte do Módulo de Sincronização

```
1  from typing import Set, Dict
2  from station import Station
3  import threading
4  import json
5  import sys, traceback
6
7  on_site_beacons: Set[str] = set()
8  on_site_stations: Dict[str, Station] = dict()
9
10
11 def refresh_devices(payload: str):
12     try:
13         message = json.loads(payload)
14         station = Station.parse(message)
15         on_site_stations[station.mac] = station
16
17         for beacon_mac in station.beacons_found:
18             on_site_beacons.add(beacon_mac)
19
20         for beacon_mac in on_site_beacons:
21             beacon_found = False
22             for station in on_site_stations.values():
23                 if beacon_mac in station.beacons_found:
24                     beacon_found = True
25                     break
26
27             if beacon_found is False:
28                 on_site_beacons.remove(beacon_mac)
29     except Exception:
30         print("Exception in user code:")
```

```
        print("-"*60)
        traceback.print_exc(file=sys.stdout)
        print("-"*60)
        pass


def show_devices():
    for station in on_site_stations.values():
        print(f'[show_devices]  {str(station)}')
        print("\t\tBeacons found:")
        for beacon in station.beacons_found.values():
            print(f'\t\t\t{str(beacon)}')
    t2 = threading.Timer(3.0, show_devices)
    t2.start()
```

B.8 Código-fonte do Módulo de Posicionamento

```
from ble_services import on_site_beacons, on_site_stations
from location import Location
from link import Link
from typing import Optional, List, Dict
import numpy as np


def trilateration() -> Optional[Dict[str, Location]]:
    locations: Dict[str, Location] = dict()
    links: List[Link] = establish_links()

    for beacon_mac in on_site_beacons:
        beacon_links = get_beacon_links(links, beacon_mac)
        if len(beacon_links) >= 3:
            d = []
            x = []
            y = []

            shorter_links: List[Link] = get_shorter_links(beacon_links)

            for link in shorter_links:
                station = link.station
                d.append(link.distance)
                x.append(station.location.x)
                y.append(station.location.y)

            A = np.array([[2*(x[0] - x[2]), 2*(y[0] - y[2])],
                          [(2*x[1] - x[2]), 2*(y[1] - y[2])]])
            b = np.array([x[0] ** 2 - x[2] ** 2 + y[0] ** 2 + d[2] ** 2 - d
    [0] ** 2,
```

```
                        x[1] ** 2 - x[2] ** 2 + y[1] ** 2 + d[2] ** 2 - d
    [1] ** 2])

            result = np.linalg.solve(A, b)
            location = Location(result[0], result[1])
            locations[beacon_mac] = location

            print(f'[trilateration]  Beacon: {beacon_mac} Location: {
    location}')

    return locations


def establish_links() -> List[Link]:
    links: List[Link] = list()
    for beacon_mac in on_site_beacons:
        for station in on_site_stations.values():
            if beacon_mac in station.beacons_found:
                link = Link(station, beacon_mac)
                #print(f'[establish_links]  {str(link)}')
                print(str(link))
                links.append(link)
    return links


def get_beacon_links(links: List[Link], beacon_mac: str) -> List[Link]:
    beacon_links: List[Link] = list()
    for link in links:
        if link.beacon.mac == beacon_mac:
            beacon_links.append(link)
    return beacon_links


# get the best three distances
def get_shorter_links(beacon_links: List[Link]) -> List[Link]:
    sorted_links = sorted(beacon_links, key=lambda k: k.distance) #useful.
    sort_links(beacon_links)
    better_links: List[Link] = list()
    for i in np.arange(3):
        better_links.append(sorted_links[i])
    return better_links


def run() -> Optional[Dict[str, Location]]:
    return trilateration()
```

B.9 Código-fonte do arquivo de utilidades

```
from link import Link
from typing import List


def get_payload_contents(payload) -> str:
    aux = str(payload)
    return aux[2:(len(aux) - 1)]


def cloning(li1) -> list:
    li_copy = li1[:]
    return li_copy


def sort_links(links: List[Link]) -> List[Link]:
    return sorted(links, key=lambda k: k.distance)
```