TÉCNICO LISBOA

Museu interativo

Pedro Miguel André da Silva

Dissertação para obtenção do Grau de Mestre em

Engenharia de Telecomunicações e Informática

Orientador: Prof. Teresa Maria Sá Ferreira Vazão Vasques

Júri

Presidente: Prof. Ricardo Jorge Fernandes Chaves
Orientador: Prof. Teresa Maria Sá Ferreira Vazão Vasques
Vogal: Prof. Alberto Manuel Ramos da Cunha

Novembro 2019

Resumo

Este trabalho tem como objetivo criar um sistema de navegação e localização interior para o Museu da Computação do *Taguspark*, que servirá de suporte a um futuro sistema que permita a interação entre os visitantes e o ambiente, de modo a criar um experiência imersiva. Para tal, foram estudadas várias técnicas e tecnologias relacionadas, bem como exemplos de museu interativos, que permitiram determinar quais as soluções a usar. Através da caracterização do movimento e da possível trajetória dos utilizadores, definiu-se a arquitetura do sistema, que utiliza odometria para estimar a posição dos visitantes, e *Fingerprinting* para corrigir erros no cálculo da trajetória. Com base nesta arquitetura, foi desenvolvido uma aplicação protótipo, que implementa os vários elementos idealizados, que a partir dos dados da sensorização múltipla, calculam o percurso do utilizador e por consequente as suas coordenadas. Para o *Fingerprinting*, foi utilizado *machine learning* para criar modelos que possam fazer previsões com as medidas recolhidas pela aplicação. A precisão destas duas técnicas foi avaliada no espaço do museu, revelando resultados interessantes.

Palavras-chave: Museu, Localização, Sensores, Interação, *Bluetooth*, Sistema, *Smartphones*

Abstract

The objective of this work is to create an interior navigation and localization system for the Museum of Computation in Taguspark, which will support a future system that allows interaction between the visitors and the environment, so as to create an immersive experience. For such, several related techniques and technologies were studied, as well as examples of interactive museums, which allowed to determined which solutions to use. Through the characterization of the movement and the possible user trajectories, the system architecture was defined, using odometry to estimates the visitor's positions, and Fingerprinting to correct errors in the trajectory calculation. Based on this architecture, a prototype application, that implements the many idealized elements, which from the multiple sensing data, calculate the user's route e by consequence his coordinates. For Fingerprinting, machine learning was used to build models that can make predictions based on the measures collect by the app. The precision of these two techniques was evaluated in the museum space, revealing interesting results.

Keywords: Museum, Localization, Sensors, Interaction, Bluetooth, System, Smartphones

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Enquadramento

Um museu é definido pelo Conselho Internacional de Museus (ICOM) como ”uma instituição permanente, sem fins lucrativos, ao serviço da sociedade e do seu desenvolvimento, aberta ao público e que adquire, conserva, investiga, difunde e expõe os testemunhos materiais do homem e do que o rodeia, para educação e deleite da sociedade”[1].

As peças expostas, sejam elas arte, objetos históricos, ou de outro tipo, transportam os visitantes para uma outra realidade. A evolução das tecnologias de sensorização e comunicação, aliadas ao uso massivo dos *smartphones* torna possível o acesso a ambientes interactivos, que permitam a interação entre o mundo físico e virtual. Aumentando a imersão dos utilizadores nos espaços museológicos é possível proporcionar um tipo de visita diferente, que torna o museu um ambiente vivo e interactivo, o que certamente irá marcar os visitantes de forma positiva a atrair o público mais jovem. Para tal é necessário providenciar conteúdos e informação de uma forma automática, e promover a interação do visitante com o museu, permitindo que este participe em jogos e desafios, enquanto se move pelo local e interage com a tecnologia de sensorização envolvente. Em qualquer dos casos, pretende-se que haja o menor envolvimento possível do utilizador para a obtenção dos conteúdos, permitindo no entanto a sua interacção com o museu, promovendo assim o factor surpresa. Por exemplo, receber informação sobre uma dada peça apenas por passar perto dela, ou ser seleccionado para participar num jogo apenas por passar num dado local e accionar um actuador escondido.

Existem, no entanto, vários problemas que dificultam a implementação deste tipo de soluções no espaço de um museu, tais como a instrumentação do próprio espaço físico e a capacidade de localizar o visitante, com precisão suficiente para lhe fornecer o conteúdo mais adequado. Esta tese visa contribuir para a resolução da questão da localização em espaços interiores, sem custos significativos de instalação de infraestruturas.

1.2 Contribuições da tese

A ideia original desta tese era a criação de um sistema interativo para o Museu da Computação, que aumentasse a imersão dos visitantes, sendo a informação da localização a chave para cumprir este objetivo. Como tal, este foi o primeiro aspeto a ser trabalhado, por ser a base sobre a qual tudo o resto seria construído. No entanto, os desafios na caracterização do movimento e da trajetória, dificultaram e atrasaram o desenvolvimento do sistema, tendo o foco desta tese mudado de um sistema interativo para um de navegação e localização.

A solução encontrada parte da premissa que o visitante recorre a um telemóvel para obter a informação durante a visita através duma aplicação própria. A localização será obtida com recurso à interpretação das medidas dos sensores do telemóvel, sendo complementada com recurso á técnica de *Fingerprinting*, descrita na secção 2.1.2, cuja informação será usada para aumentar a precisão de posicionamento e corrigir erros de odometria. Para isso, são utilizados sensores localizados no ambiente – *beacons Bluetooth Low Energy* – que em conjunto com outros dados dos sensores, permitem reconhecer a posição atual do utilizador. A técnica funciona comparando as medidas atuais com uma série de medidas recolhidas anteriormente. Esta parte terá de ser feita num servidor remoto, que será construído para o efeito. Estas contribuições são as bases do museu interativo, sobre as quais as funcionalidades e técnicas, que irão oferecer uma experiência diferente, serão implementadas.

1.3 Estrutura do relatório

A estrutura deste relatório segue a cronologia das atividades realizadas neste trabalho.

Começa com o estado de arte, descrito no capítulo 2, onde são apresentados os resultados da pesquisa efetuada durante a fase do projeto. Segue-se a arquitetura da solução implementada, na secção 3, descrevendo os requisitos e características, bem como a apresentação dos diferentes módulos do sistema. No capítulo seguinte, é apresentada a implementação do sistema, na secção 4, detalhando o desenvolvimento de cada elemento da arquitetura. No capítulo 5, é feita uma avaliação, não só aos vários elementos e componentes, mas também ao sistema num todo.

No final, é apresentada uma síntese do trabalho realizado, bem como as conclusões finais e o trabalho futuro, face aos resultados obtidos e aos desafios encontrados durante o desenvolvimento do sistema. Os anexos incluem o código desenvolvido para as aplicações e programas criados, tal como outros dados produzidos durante esta tese.

Capítulo 2

Trabalho relacionado e estado da arte

Neste capítulo são apresentadas tecnologias relacionadas e soluções de museus interativos existentes, que foram encontradas durante a fase de pesquisa.

Divide-se em três partes: na primeira parte são apresentadas as tecnologias relevantes para a construção dum museu interativo, na secção 2.1; na segunda parte, são descritos trabalhos realizados no contexto dos museus interativos, na secção 2.2; e, para terminar, na última parte é feita uma síntese das principais ideias apresentadas que podem ser relevantes no contexto do projeto a desenvolver, na secção 2.3.

2.1 Tecnologias para localização interior

Existem três formas diferentes de efetuar localização interior, que serão descritas a seguir: com recurso a tecnologias de redes sem fios; a sensores simples; ou a sensorização múltipla. No final de cada secção é apresentada uma síntese das técnicas descritas, na qual os principais aspetos em analise são apresentados sobre a forma de tabela.

2.1.1 Aspetos em análise

De forma a poder efetuar uma comparação das diversas opções tecnológicas que permita selecionar as opções mais adequadas, foram identificados os principais aspetos que se deveriam analisar e que incluem:

- **Tecnologias de suporte** que podem ser usadas para aplicar a técnica de localização, como as tecnologias de redes sem fios WiFi, o *Bluetooth*, ou outras;
- Precisão de localização, em que são analisados dois aspetos:
  - **Tipo de local identificado**, que define o âmbito da estimativa da localização: circulo, vizinhança de um ponto ou uma posição;
  - **Erro de precisão**, que estima o valor do erro, de acordo com resultados de estudos efetuados.

- **Equipamento**, que identifica o tipo de *hardware* que é necessário para que a técnica funcione, seja na infraestrutura do museu ou nos sistemas terminais dos utilizadores;
- ***Software***, que identifica as necessidades de instalação de *software*, seja na infraestrutura do museu ou nos sistemas terminais dos utilizadores;
- **Custos**, que avalia os custos de instalação, configuração e de operação associados ao uso da técnica.

2.1.2 Localização com recurso a redes sem fios

Existem três formas diferentes de localização com recurso a redes sem fios: estimativa da distância, triangulação e *fingerprinting*. Cada uma destas técnicas será descrita nos parágrafos seguintes.

Estimativa da distância

A técnica de localização baseada em redes sem fios, mais simples que se pode conceber, consiste em utilizar as medições de ”distância”entre o utilizador e um Ponto de Acesso (AP - *Access Point*), cuja posição no edifício é conhecida, para calcular a posição do utilizador. As medições de distância resultam da leitura de características da comunicação que estão relacionadas com a distância física a que o utilizador se encontra do AP, como sejam o *Received Signal Strength* (RSS) [2] [3] ou o *Time-of-Flight* (ToF) dos pacotes [4]. Esta técnica pode ser usada com várias tecnologias sem fios, tais como o WiFi [2] [4] e Bluetooth [3], encontrando-se ilustrada na figura 2.1.

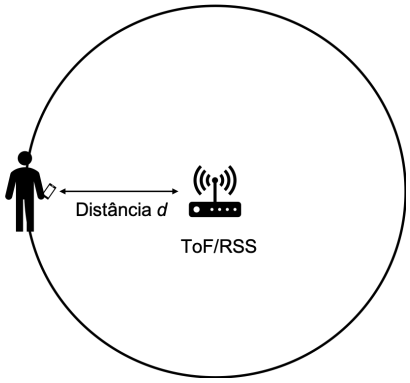


Figura 2.1: Esquema de estimativa da distância ao ponto de acesso.

A grande vantagem desta técnica reside na facilidade de uso, uma vez que requer apenas a existência dum AP, não sendo necessário instalar uma infraestrutura adicional. Deste ponto de vista, o seu uso é adequado para espaços pequenos. No entanto, a localização estimada não é precisa, uma vez que a informação obtida apenas permite estimar uma área circular, com centro no AP, em que o utilizador se encontra, dado que o resultado obtido é o raio. Adicionalmente, os problemas inerentes a propagação em ambientes fechados, nomeadamente o efeito da propagação multi-caminho, conduzem a obtenção de valores de distância incorretos e variáveis ao longo do tempo [4].

No trabalho apresentado por Alex Mariakakis *et al*, em *SAIL: Single Access Point-Based Indoor Localization*, explora o uso duma técnica complementar - *Dead Reckoning* (2.1.4) - para reduzir o impacto negativo do efeito da propagação multi-caminho, explorando o uso de múltiplas antenas e a mobilidade humana.

Triangulação

A técnica de localização por triangulação baseia-se no facto de existirem múltiplos APs ao alcance dum utilizador e de ser possível obter uma localização mais precisa cruzando a informação da estimativa da distância do utilizador a cada AP. Para que tal seja possível, algoritmos de localização comparam os valores de potência de transmissão para determinar o AP mais próximo [5][6]. Este tipo de técnica encontra-se ilustrado na figura 2.2.

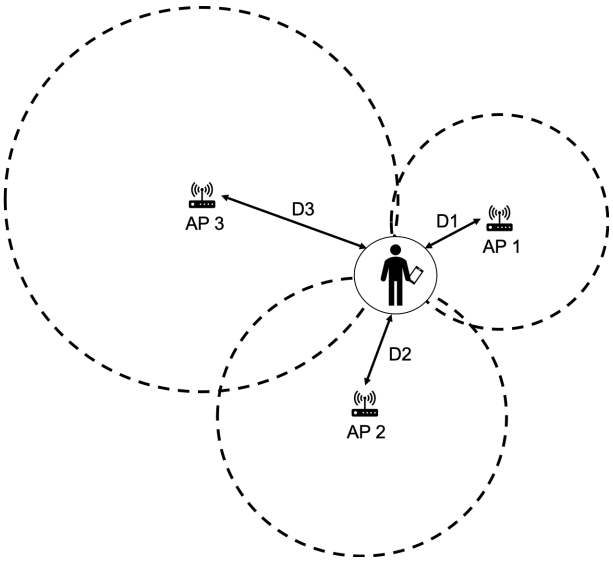


Figura 2.2: Esquema de estimativa da distância com base na triangulação.

Em relação à técnica de estimação de distância apresentada anteriormente, a técnica de triangulação permite obter uma localização mais precisa. Todavia, partilha os problemas anteriormente descritos, associados à variabilidade do sinal, decorrente dos efeitos da propagação multi-caminho.

Esta técnica é usada em dois sistemas de museus interativos [5], [6], que serão descritos na secção 2.2. Este último é o único que apresenta resultados de estimativas de distância, nomeadamente uma tabela de dez medidas, em que a precisão desta técnica esteve sempre acima dos 90%.

Fingerprinting

A técnica de *Fingerprinting* tem por objetivo melhorar a precisão da localização obtida por triangulação, criando uma base de dados com informação de medidas de distância em coordenadas previamente conhecidas, cuja informação serve para validar as medições de distância obtidas durante a fase de localização [7] [8] [9]. Tal como as técnicas anteriores, o cálculo da distância também usa o RSS do sinal recebido.

A técnica envolve duas fases: a **fase de treino** e a **fase de inquérito** (*query*) [7] [8] [9].

Na **fase de treino** é construída um base de dados ou um mapa de *fingerprinting* [8]. Para o efeito, recolhe-se a informação dos valores de RSS que foram medidos para cada AP detetado, em vários pontos do edifício e registam-se esses valores - *fingerprints* - em conjunto com a sua localização, previamente conhecida.

A **fase de inquérito** ocorre quando um utilizador requer a sua localização. Os valores de RSS de cada AP detetado que foram medidos são comparados com os valores armazenados na base de dados. A posição escolhida - localização aproximada - será aquela que mais se aproxima das medições efetuadas. Um método proposto para estimar a localização consiste em ordenar, de forma crescente a sequência de valores de RSS associados a uma zona (sub-espaço da área a mapear) e usar essa ordenação para estimar a posição [8].

A vantagem desta técnica reside no facto de, atualmente, muitos edifícios já terem uma infraestrutura de redes sem fios, pelo que não é necessário equipamento extra. Existem, no entanto, várias desvantagens associadas ao uso desta técnica. A fase de treino, necessária para obter a localização, é bastante complexa e tediosa, tendo de ser repetida se existirem alterações no ambiente, pois as *fingerprints* recolhidas deixam de ser válidas [10] [2]. A variabilidade do sinal, devido ao efeito das condições de propagação, torna as características observadas pelo utilizador diferentes daquelas registadas durante a fase de treino, tendo impacto na qualidade dos resultados [8]. Por último, o erro de localização pode ser elevado, dependendo da tecnologia usada e da cobertura de rede [9] [8].

Existem outras soluções que tentam resolver estes problemas. As RSS observadas pelos utilizadores podem ser utilizadas para atualizar a base de dados, sendo a localização da *fingerprint* observada obtida através de *Dead Reckoning* [9]. Em alternativa, também se podem usar outras tecnologias sem fios que permitam obter uma localização mais precisa, como por exemplo Bluetooth, e usar estas tecnologias para dividir o mapa em sub-espaços. O uso de tecnologias de comunicação complementares tem por objetivo que as as *fingerprints* observadas sejam únicas, melhorando assim a precisão [7].

Síntese e discussão

A tabela seguinte apresenta as características principais das diferentes técnicas de localização com recurso a redes sem fios.

Pretende-se utilizar um sistema de localização, que forneça informação precisa, sem, no entanto, isso implicar um sistema muito complexo, dispendioso e de difícil manutenção, ou o uso de dispositivos terminais próprios.

O facto de todas as técnicas descritas se poderem implementar com recurso a redes WiFi, é importante, na medida em que não se torna necessário adicionar infraestruturas de rede adicionais para o efeito.

Relativamente à precisão da localização, a técnica de *fringerprinting* oferece melhores resultados. A técnica de triangulação também oferece resultados interessantes, sobretudo quando se recorre à tecnologia Bluetooth, ou outra de curto alcance, para reduzir o erro de localização.

A grande desvantagem destas duas técnicas reside no facto de não haver garantia que funcionem

	Tecnologias de suporte			Precisão de localização		Equipamento necessário		*Software* necessário		Custos	
Técnica	*WiFi*	*Bluetooth*	Outras	Tipo de local identificado	Erro de precisão	Infraestrutura do museu	Sistemas Terminais	Infraestrutura do museu	Sistemas Terminais	Instalação/ Configuração	Operação
Estimativa da distância	X	X		Círculo	Elevado	1 AP por espaço	Dispositivos móveis (*smartphones* e *tablets*)	Sim	Não	Reduzido	Reduzido
Triangulação	X	X		Vizinhança de um ponto	Medio	Pelo menos 3 APs por espaço (mesma tecnologia)	Dispositivos móveis (*smartphones* e *tablets*)	Não	*App* para o sistema terminal	Reduzido (*Hw*) Medio (*Sw*)	Reduzido
Fingerprinting	X	X	RFID, etc	Posição	Pequeno (em função do mapa de *fringerprinting*)	Pelo menos 3 APs por espaço (uso possível de multiplas tecnologias)	Dispositivos móveis (*smartphones* e *tablets*)	Base de dados de *fingerprints* e algoritmos de comparação	*App* para o sistema terminal	Elevado	Médio (manter mapa de *fingerprintings*)

Figura 2.3: Tabela de comparação entre as diferentes técnicas de localização usando redes sem fios.

corretamente, se o número de APs ao alcance do utilizador for inferior a 3, ou se as condições de propagação sofrerem grandes variações, derivadas da própria dinâmica do espaço (por exemplo, se estiverem presentes muitas pessoas).

De realçar ainda que, a técnica de triangulação oferece pode ser implementada exclusivamente numa aplicação a correr em qualquer sistema terminal, enquanto que, a técnica de *fringerprinting* também recorre a *software* acessível através da rede e à base de dados de *fringerprints*, sendo por isso bastante mais complexa e de difícil manutenção.

2.1.3 Localização por sensorização simples

É possível utilizar sensores simples para obter a posição de um utilizador através dos dados que recolhe. Existem vários tipos de sensores que podem ser usados neste contexto, tais como: sensores de infravermelhos [11] e os sensores de pressão [12]. Estes dois tipos de sensores serão descritos no resto desta secção.

Será descrita ainda a solução de sensorização simples, com recurso á tecnologia *Near Field Communication* (NFC).

Sensores de infravermelhos

Os sensores de infravermelhos medem o comprimento de onda infravermelho emitido pelos humanos, entre os 9.4 e 10.4 µm. Em virtude do seu baixo custo, estes sensores são bastante comuns, existindo, por exemplo, nas luzes com detetores de presença/movimento [11].

Para garantir maior precisão, estes sensores são colocados no teto, de forma a que as áreas de deteção de sensores adjacentes se sobreponham. Cada sensor tenta localizar um utilizador durante um período constante. Um terminal recebe os resultados de todos os sensores e processa essa informação para determinar a posição [11].

Sendo que a localização dos sensores em cada divisão são conhecidas, se um utilizador entrar na área de deteção, então o sensor envia um sinal ”ON”, enquanto que os outros, que não detetam

o utilizador, enviam um sinal ”OFF”. No caso de dois sensores adjacentes enviarem um sinal ”ON”, então a posição do utilizador é assumida como o ponto médio entre estes. Se forem três, o utilizador encontra-se na centróide dos centros dos sensores [11].

A precisão de localização desta técnica depende de três fatores: o número de sensores colocados, bem como o local e o raio de deteção. Este fatores determinam o número de áreas adjacentes e a existência de pontos cegos, ou seja, locais que não são cobertos pelos sensores [11]. Existem no entanto vários fatores que podem causar falsos alarmes, ou seja, o sistema indicar uma posição na qual não se encontra um utilizador. Isto pode ser causado por fontes de calor, que emitem ondas infravermelhas que podem ser detetadas pelos sensores; a presença de insetos ou outros animais e também correntes de ar, que causam uma mudança brusca na temperatura [13].

Outras soluções usam lentes de *Fresnel* instaladas nos sensores, que rejeitam formas de onda que não sejam humanas [11]. Também um limite de voltagem é usado no sinal elétrico, que é o *output* do sensor, para distinguir a variação no sinal infravermelho do utilizador de outros sinais infravermelhos [11].

Sensores de pressão

Os sensores de pressão medem esta grandeza. A deteção de objetos é realizada quando é exercida uma força perpendicular sobre a superfície do sensor.

Na solução descrita no artigo ” *The Smart Floor: A Mechanism for Natural User Identification and Tracking*”de Robert J. Orr *et al* [12], são colocados sensores de pressão no chão, de forma a detetar quando o utilizador pousa o pé. Estes dispositivos são chamados de *Ground Reaction Force* (GRF).

É possível distinguir entre utilizadores através do tipo de passo. Para tal, é criado um perfil para cada passada, que inclui várias características obtidas dos dados do sensor. Através da distinção de cada passada detetada é possível reconhecer qual o percurso de cada utilizador.

A obtenção de informação de percurso implica a distribuição dum *array* de sensores, em posições previamente conhecidas, que correspondam a pontos de passagem obrigatória ou de mudança de direção. Neste modelo, a direção do percurso é determinada comparando a informação dos sensores temporalmente.

A precisão de localização pode ser afetada pela presença de várias pessoas sobre o mesmo sensor, impedindo a distinção entre os utilizadores presentes.

Near Field Communication

Uma técnica de localização muito usada em museus, por conta da sua simplicidade, tem como base a tecnologia *Near Field Communication* (NFC). Apesar de normalmente utilizada para transferência de dados ou pagamentos, também é possível usar NFC para localização.

Tags NFC são colocadas em vários pontos do edifício ou perto de peças em exposição. Quando um utilizador, carregando um dispositivo móvel com um leitor NFC, se aproxima da *tag* colocada, o ID desta é enviado para o dispositivo. Como a posição das *tags* é conhecida, a posição do utilizador na

proximidade duma *tag* também fica a ser conhecida [14].

Ainda que simples, esta técnica tem uma precisão baixa, pois apenas consegue localizar o utilizador quando se encontra próximo da *tag*, sendo que para conseguir maior precisão, é necessário um elevado número de *tags*. Também a frequência usada limita a distância entre a *tag* e o dispositivo necessário para a transferência do ID, o que significa que, em certos casos, o utilizador pode ter de aproximar o dispositivo, sempre que for necessário obter a localização [15].

Síntese e discussão

A tabela seguinte apresenta as características principais das diversas técnicas de localização com recurso a sensores.

	Precisão de localização		**Equipamento necessário**		***Software* necessário**		**Custos**	
Tipo de sensor	Tipo de local identificado	Erro de precisão	Infraestrutura do museu	Sistemas Terminais	Infraestrutura do museu	Sistemas Terminais	Instalação/ Configuração	Operação
Infravermenhos	Posição ou círculo	Baixo (depende de certos fatores como presença de fontes de calor ou animais, *etc*)	Múltiplos sensores infravermelhos	Sem sistemas terminais	Mapear posição dos sensores e calcular a posição	Sem sistemas terminais	Elevado (montar um número elevado de sensores)	Médio (manter terminal que recebe e processa informação dos sensores)
Pressão	Posição	Baixo	Multiplos sensores de pressão	Sem sistemas terminais	Mapear posição dos sensores e calcular a posição	Sem sistemas terminais	Elevado (montar um número elevado de sensores)	Médio (manter terminal que recebe e processa informação dos sensores)
NFC	Posição (localização da *tag*)	Baixo (na presença da *tag*) Elevado (sem presença de *tag*)	Múltiplos *tags* NFC	Dispositivos móveis com leitor NFC (como *smartphones* e *tablets*)	Mapear posição das *tags* e determinar posição	*App* para leitura do ID da tag	Médio (montar um número elevado de *tags*)	Médio (manter terminal que recebe e processa informação recebida)

Figura 2.4: Tabela de comparação entre os diferentes tipos de sensores.

O uso de sensores simples tem várias vantagens, como a preservação da privacidade dos utilizadores e não requerer que os utilizadores carreguem um dispositivo extra. O mesmo não é verdade no caso do NFC, em que o utilizador necessita de carregar um dispositivo com um leitor. É necessário, no entanto, a instalação de toda uma nova infraestrutura, tanto dos sensores como do sistema terminal. Também a escalabilidade é um problema, sendo necessários vários sensores para localizar com precisão razoável, o que pode ter custos elevados em espaços maiores.

O artigo "*A Pyroelectric Infrared Sensor-based Indoor Location-Aware System for the Smart Home*"[11], que apresenta um sistema de localização baseado em sensores infravermelhos, não menciona nenhuma forma de distinguir entre utilizadores, algo necessário para o uso em museus, onde existem vários visitantes. Já o artigo "*The Smart Floor: A Mechanism for Natural User Identification and Trac-*

king"[12], que descreve uma solução de localização com sensores de pressão, evidencia a possibilidade de individualizar cada um dos utilizadores através da sua passada. No entanto, existe o problema da distinção quando mais do que um utilizador se encontra sobre o mesmo sensor. Já com o uso do NFC, isto não é um problema, pois a individualização é feita nos terminais carregados pelos utilizadores.

2.1.4 Localização por sensorização múltipla

É possível integrar informação proveniente de diversos sensores, para a partir daí obter informação mais precisa de localização.

Nesta secção vão ser apresentados dois exemplos de sistemas deste tipo: *Dead Reckoning* e *Urban sensing e Activity Recognition*.

Dead Reckoning

A localização por *Dead Reckoning* baseia-se no facto da posição inicial do utilizador, ao entrar no edifício, ser previamente conhecida, seja por GPS ou por outra maneira qualquer. Esta posição é o ponto de partida para esta técnica, que com base nesta posição conhecida consegue calcular a posição atual. Para isso, é necessário descobrir o deslocamento entre os dois pontos, algo que é possível com os sensores presentes nos *smartphones* que os utilizadores costumam transportar [10] [2] [16]. A figura 2.5 ilustra este tipo de sistemas.

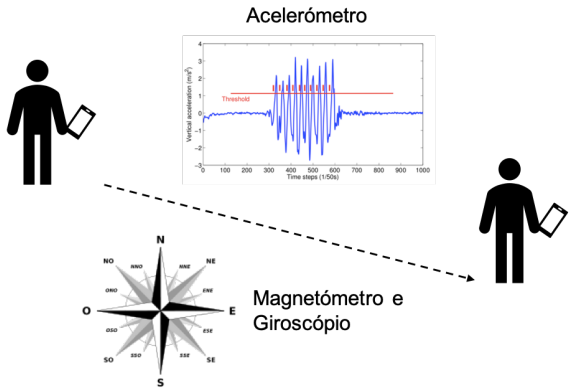


Figura 2.5: Esquema de funcionamento do *Dead Reckoning*.

Cada sensor têm um objetivo diferente. O acelerómetro mede a aceleração do dispositivo. O magnetómetro, que está na base da bússola, mede a direção e intensidade de um campo magnético próximo. A direção dos passos é obtida deste sensor. O giroscópio mede a velocidade angular em três dimensões, ou seja, fornece a orientação do dispositivo. Os dados deste sensor são usados para complementar os do magnetómetro, que é influenciado por equipamentos metálicos ou elétricos. Adicionalmente, também serve para revelar como o utilizador virou [10].

Um passo é identificado como um máximo do sinal do acelerómetro que está acima de um certo valor de limiar. Multiplicando o número de passos pelo tamanho do passo, obtém-se a distância percorrida. Normalmente os diversos tipos de sensores são usados em conjunto, de modo a melhorar o resultado obtido pelo acelerómetro.

A técnica de localização por *Dead Reckoning* é simples e não requer infraestrutura ou dispositivos extra, uma vez que usa exclusivamente os sensores presentes em quase todos os dispositivos móveis. No entanto, o ponto de partida tem de ser conhecido para funcionar, o que significa que a localização deve começar num ponto específico e não onde o utilizador quiser. Por outro lado, o resultado da medição depende da qualidade dos sensores, que são afetados pelo ruído, causando uma acumulação de erro ao longo do tempo. Por estas razões, o *Dead Reckoning* é normalmente é usado em conjunto com outras técnicas, tal como o *Urban Sensing e Activity Recognition*, descrito a seguir.

Urban sensing* e *Activity Recognition

No caso do *Dead Reckoning*, os sensores detetam que o utilizador está a caminhar, sendo que os dados que recolhem fornecem a indicação da direção e da distância. Isto significa que outras atividades também podem ser detetadas. Por exemplo, virar num corredor, subir as escadas ou usar um elevador, fazem com que as pessoas tenham certos comportamentos que causam flutuações nos dados medidos. Mas não é apenas o comportamento humano que pode causar estas flutuações. Certos elementos presentes no ambiente também podem causar flutuações nos dados dos sensores. Por exemplo, a distorção nos dados do magnetómetro pode ser causada por um campo magnético, criado por equipamento elétrico ou metálico presente no edifício. São usados os mesmos sensores que no *Dead Reckoning*, bem como outros, tais como o barómetro e o WiFi. Sendo a planta do espaço onde se pretende localizar o utilizador previamente conhecida, com a identificação destes padrões é possível determinar a localização precisa do utilizador e corrigir o erro acumulado no *Dead Reckoning*.

Consideremos o exemplo do elevador: o começo e a paragem causam uma flutuação única nos dados do acelerómetro, que é chamada de *signature*. O local onde esta *signature* é detetada é chamado de *landmark*. Neste caso específico, esta é considerada uma *Seed landmark*, pois é conhecida *a priori*, ou seja, sempre que o utilizador use um elevador, a mesma *signature* é observada. No entanto, as *landmarks* podem também ser descobertas pelos utilizadores. A estas dá-se o nome de *Organic landmarks*. Ao contrários das *Seed landmarks*, estas são encontradas organicamente, sendo classificadas como tal se múltiplos utilizadores observarem a mesma *signature* no mesmo local. Um exemplo de uma *Organic landmark* é a distorção no magnetómetro causada pela presença de equipamento elétrico ou metálico nas proximidades, tal como anteriormente mencionado. A figura 2.6 ilustra os diferentes tipos de *landmarks*.

A técnica de *Urban sensing e Activity Recognition* não é usada sozinha, pois apenas permite saber a localização do utilizador quando está perante uma *landmark*. É normalmente usada para reduzir o erro no *Dead Reckoning*. No inicio, a precisão de localização é baixa, porém, com o passar do tempo aumenta, com o aumento de *Organic landmarks*. No entanto, em espaços pequenos esta técnica pode não ser tão fiável, devido à menor probabilidade de existirem *landmarks*. Outro problema que foi identificado é o trabalho que têm de ser feito para identificar inicialmente as *signatures* associadas às *Seed Landmarks*.

(a) *Seed Landmark*.

(b) *Organic landmark*.

Figura 2.6: Exemplos de *landmarks*.

Síntese e discussão

A tabela seguinte apresenta as características principais das diversas técnicas de localização com recurso a sensorização múltipla.

	Tecnologia de suporte	**Precisão de localização**		**Equipamento necessário**		***Software* necessário**		**Custos**	
Técnica	Sensores usados	Tipo de local identificado	Erro de precisão	Infraestrutura do museu	Sistemas Terminais	Infraestrutura do museu	Sistemas Terminais	Instalação/ Configuração	Operação
Dead Reckoning	Acelerómetro, magnetómetro e giroscópio	Posição	Aumenta com passar do tempo	Não	Dispositivos móveis (como *smartphones* e *tablets*)	Não	*App* para recolha e procesamento dos dados dos sensores	Baixo (*Hw*) Médio (*Sw*)	Sem custos
Urban Sensing & *Activity Recognition*	Acelerómetro, magnetómetro, giroscópio, barómetro, WiFi	Área (local da *landmark*)	Depende da singularidade da *signature* da *landmark*	Sim	Dispositivos móveis (como *smartphones* e *tablets*)	Sim, para a identificação de *landmarks*	*App* para recolha de dados dos sensores	Médio (*Hw*) Médio (*Sw*)	Médio (manter mapa de *landmarks*)

Figura 2.7: Tabela e comparação entre os diferentes tipos de técnicas que utilizam sensorização múltipla.

O *Dead Reckoning* é uma técnica que não requer infraestrutura adicional, mas requer *software* capaz de interpretar os dados dos sensores, instalado nos dispositivos terminais. Este *software* tem de consolidar os dados de múltiplos sensores, como também tem de distinguir entre as diferentes posições do dispositivo, visto que as flutuações nos dados diferem. Isto aumenta o custo da fase de desenvolvimento. Porém, não existem custos operacionais.

O erro de precisão é um dos maiores problemas com esta técnica. O ruído nos sensores causa o aumentar do erro com o passar do tempo. Uma solução para este problema é o uso de *Urban Sensing* e *Activity Recognition*. *Landmarks* permitem reduzir este erro para próximo de zero, visto que cada *landmark* é única e a sua localização conhecida. Tal como o *Dead Reckoning*, esta técnica requer o uso dos múltiplos sensores dos dispositivos móveis, no entanto, é necessária infraestrutura extra

no edifício, para que seja possível identificar as *landmarks*, quer seja *a priori* (*Seed Landmarks*), ou durante o período de operação (*Organic Landmarks*). Os custos desta técnica são maiores que os do *Dead Reckoning*, por causa disto. Considerando que para conseguir uma precisão de localização razoável é necessário usar as duas técnicas em conjunto, pode-se concluir que o sistema final teria um custo elevado.

2.2 Estado da arte - Museu Interativo/Inteligente

A evolução tecnológica tem permitido melhorar de forma significativa a forma como a interação com o meio ambiente se processa. Este avanço tecnológico tem permitido desenvolver museus interativos, que serão descritos a seguir. Esta subsecção termina com uma síntese das propriedades mais interessantes que foram encontradas nos trabalhos descritos, e que podem ser usadas como base para o trabalho a desenvolver.

2.2.1 *Cultural Heritage Museum*

O caso de estudo apresentado por Angelo Chianese e Francesco Piccialli, em "*Designing a smart museum: when Cultural Heritage joins IoT*"[5], refere-se á solução tecnológica que foi criada para uma exposição temporária de esculturas de arte no Castelo *Maschio Angioino*, em Nápoles, Itália.

Como terminal, os visitantes usam o seu dispositivo móvel que executa uma aplicação específica. Nesta *app* é mostrada a peça mais próxima do utilizador, bem como informação detalhada, uma galeria de imagens e outros conteúdos multimédia relacionados.

Em termos de arquitetura, o sistema está organizado em três níveis, designados por *layers*:

- *Sensing layer*, que é responsável pela recolha e transferência de dados e colaboração dos nós em rede;
- *Network layer*, que é responsável pela transferência de dados entre as redes e aplicações;
- *Application layer*, onde as aplicações IoT são lançadas com as funcionalidades *middleware*.

A localização dos visitantes usa dois nós sensores: nós *Slave*, que criam um área Bluetooth, relacionada com um ou mais objetos, estando cada um destes nós ligado aos nós *Server*, que guardam conteúdo relacionado com os itens próximos e cria uma área de cobertura WiFi. Quando um utilizador com a aplicação a correr no seu dispositivo móvel, ligado à rede sem fios, entra na área *Bluetooth*, é detetado e o nó *Server* é notificado para entregar o respetivo conteúdo multimédia à aplicação.

A figura 2.8 apresenta os três elementos base deste sistema, nomeadamente:

- O servidor do *Cultural Heritage Informations System*, composto pelos *layers Network* e *Application*, cujo objetivo principal é armazenar e gerir o conteúdo;
- *Gateway*, que gere toda a comunicação entre os sensores e o servidor CHIS;

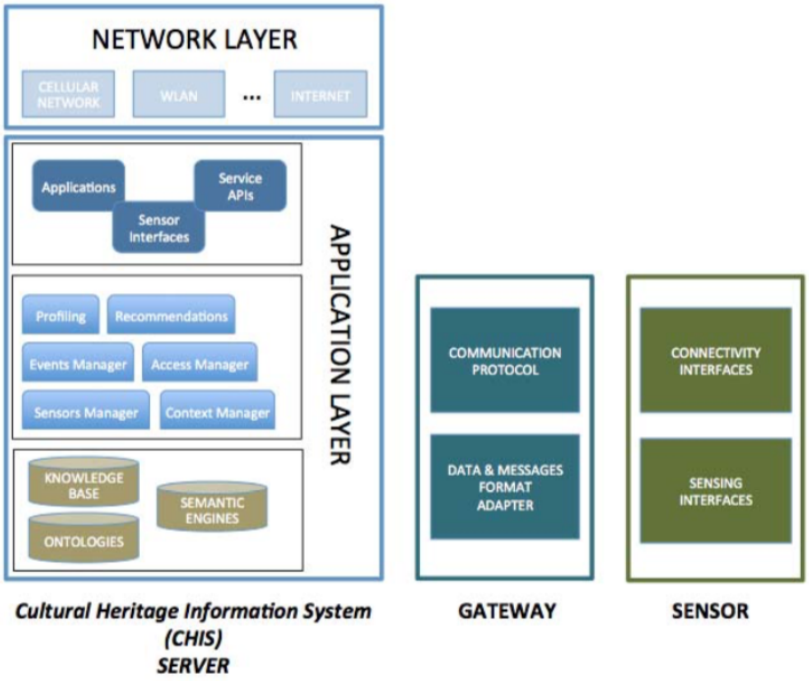


Figura 2.8: Representação da arquitetura do sistema [5].

- *Sensor*, que providencia interfaces de conectividade que habilitam as funcionalidades do sensor de acordo com os diferentes tipos de nós sensores.

2.2.2 *SmARTweet*

O objetivo do sistema *SmARTweet*, de Angelo Chianese *et al* [6], é entregar automaticamente aos utilizadores conteúdos multimédia relacionados com as peças do museu. Os visitantes utilizam um dispositivo móvel a correr a aplicação. O conteúdo é personalizado com base no perfil do utilizador, construído através do preenchimento de um questionário. A arquitetura geral do sistema encontra-se representada na figura 2.9.

Cada peça do museu está ancorada a um sensor que gera um sinal de rede. Para detetar qual a peça mais próxima, é usada a técnica da triangulação, em que o *Received Signal Stregth Indicator* (RSSI) de cada sensor é comparado para determinar qual é o valor mínimo. O ID vencedor é enviado para o sistema.

Para além da aplicação móvel e da infraestrutura da rede sem fios, existem dois elementos na arquitetura deste sistema que são relevantes descrever:

- *Gateway Server*, que recebe o ID vencedor e requisita ao *Multimedia Content Server* o conteúdo relacionado com a peça;

- *Multimedia Content Server*, que gere os conteúdos e aceita os pedidos do *Gateway Server*, criando uma história multimédia personalizada e uma estratégia de recomendação apropriada, que são enviadas para a aplicação.

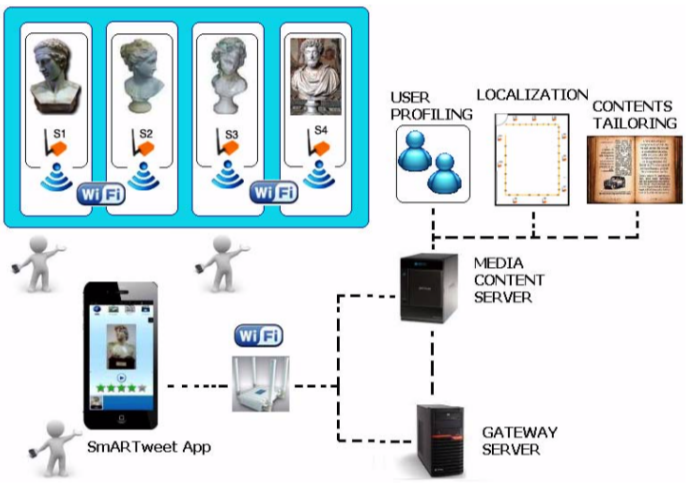


Figura 2.9: Arquitetura do sistema [6].

2.2.3 *Museum Navigation based on NFC Localization Approach and Automatic Guidance System*

O sistema oferece uma solução de localização interior baseada em NFC que permite selecionar os conteúdos a apresentar ao visitante com base na sua localização. Para o efeito, os visitantes têm de carregar durante a visita um dispositivo proprietário.

O sistema apresentado permite criar e editar os conteúdos a enviar aos visitantes [14].

Para localizar o utilizador, são colocadas *tags* NFC em vários pontos do museu. Estas posições estão mapeadas em *software*. Assim, quando um dispositivo deteta uma *tag*, é obtido um ID que identifica a sua localização.

São usadas duas tecnologias de comunicação para transferir dados com o dispositivo, *Infrared Radio Broadcasting* (IrDB), infravermelhos, para receção de dados no dispositivo (*downlink*); e rádio, usando uma rede pessoal, para envio de dados do dispositivo (*uplink*).

Quando estão a carregar na estação integrada de gestão, os dispositivos comunicam diretamente com o *IrDB-RF Hybrid Communication Controller*, que está ligado a um PC por USB. Por sua vez, este comunica com uma base de dados de conteúdos e história através da Internet. Esta base de dados armazena o conteúdo a apresentar aos visitantes. Este conteúdo é descarregado simultaneamente para todos os dispositivos, enquanto estão a carregar. Isto evita os problemas de tráfego quando múltiplos visitantes tentam aceder ao mesmo conteúdo ao mesmo tempo.

A arquitetura geral do sistema encontra-se representada na figura 2.10.

2.2.4 *An Indoor Location-Aware System for an IoT-Based Smart Museum*

O objetivo do sistema, descrito por Stefano Alletto *et al* [3], é melhorar a experiência dos utilizadores num museu de arte. Para o efeito, os visitantes têm acesso a um dispositivo terminal tipo *wearable*, que é configurado com o perfil do utilizador através de um questionário respondido anteriormente.

A arquitetura dos componentes de localização do sistema encontra-se representada na figura 2.11.

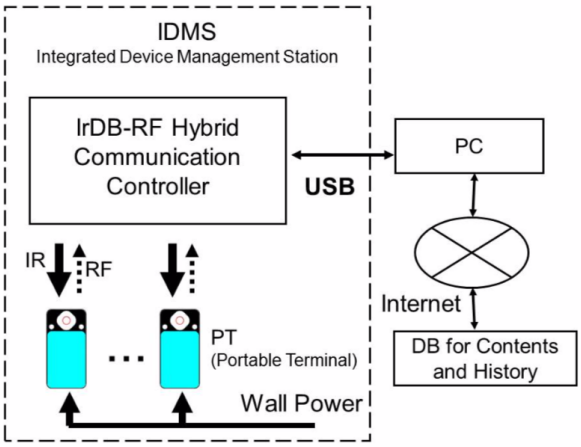


Figura 2.10: Arquitetura da estação integrada de gestão de dispositivos [14].

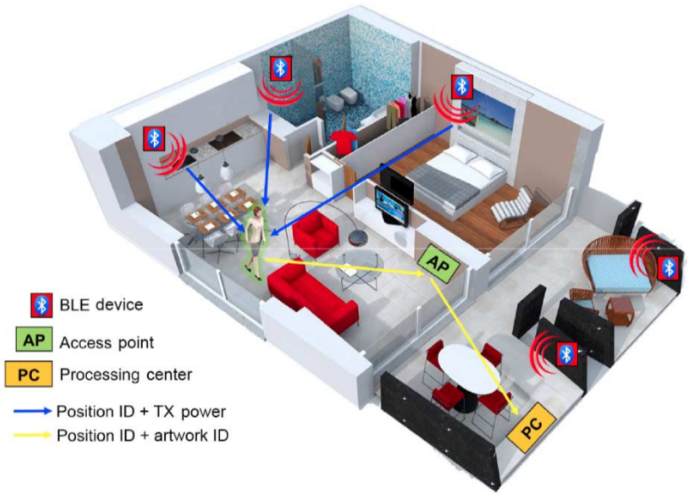


Figura 2.11: Principais componentes da arquitetura de localização [3].

Para detetar a posição dos visitantes, vários dispositivos com interface *Bluetooth Low Energy* (BLE) são colocados nas várias salas do museu. Cada dispositivo envia o seu ID de localização e um valor de potência de transmissão (TX). O *wearable* que o utilizador carrega recolhe informações de todos estes dispositivos, e usa essa informação para determinar a sala onde o visitante se encontra. Para o efeito, é usada a técnica de localização por triangulação.

A peça que o visitante está a apreciar é identificada através de um algoritmo de processamento de imagem, usando a câmara existente no *wearable*, para reconhecer a pintura.

A estrutura geral do sistema encontra-se representada na figura 2.12.

Da arquitetura deste sistema faz também parte o *Processing center*. O *Processing center* (Pc) recebe o ID vencedor da técnica de localização, enviado pela rede WiFi, acedendo á *Cloud* para ir buscar os conteúdos relacionados.

Dependendo do número de visitantes, o *Processing center* decide se é enviada uma descrição áudio para o *wearable* de cada utilizador ou se os conteúdos multimédia são exibidos na parede interativa do museu. Mais especificamente, se o número de visitantes está abaixo de um certo limite, a primeira

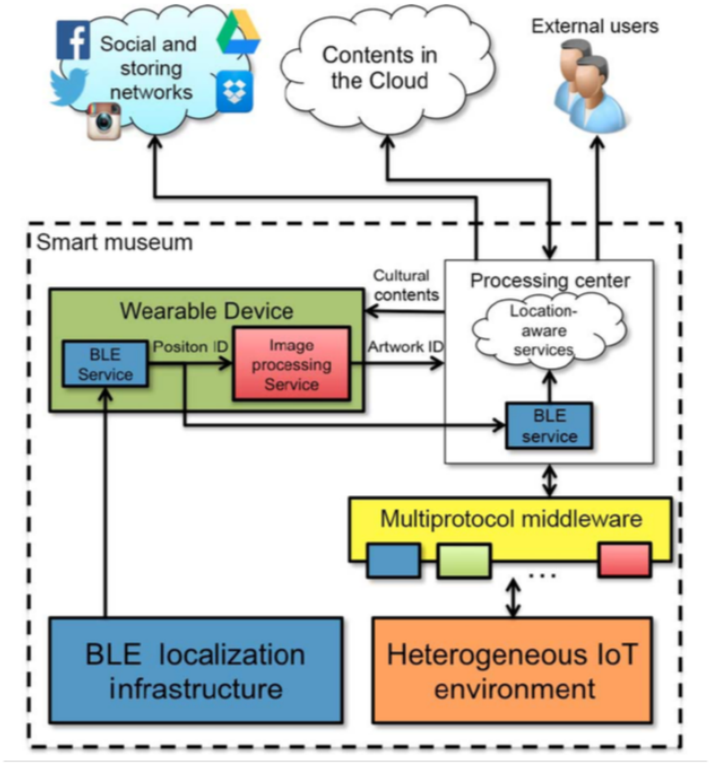


Figura 2.12: Estrutura geral do sistema [3].

opção é escolhida.

2.2.5 Síntese e discussão

A tabela seguinte apresenta as características principais dos sistemas de suporte aos museus interativos que foram apresentados.

Título	Dispositivos terminais	Técnica de localização	Tecnologias de comunicação	Arquitetura
Cultural Heritage Museum	Dispositivos móveis	Deteção de dispositivos por BLE	WiFi	• *Semantic Information Broker* • *Knowledge Processors*
SmARTweet	Dispositivos móveis	Triangulação (WiFi)	WiFi	• *Gateway server* • *Multimedia content server*
Museum Navigation based on NFC Localization Approach and Automatic Guidance System	Dispositivo móvel proprietário	*Tags* NFC	Infravermelhos e rádio	• *IrDB-RF Hybrid Communication Controller* • PC • Base de dados de conteúdos e história
An Indoor Location-Aware System for an IoT-Based Smart Museum	Dispositivo *wearable*	Triangulação (BLE)	WiFi	• *Processing center*

Figura 2.13: Tabela de comparação dos diferentes sistemas para museus interativos apresentados.

Alguns dos sistemas descritos usam dispositivos móveis, como *smartphones* e *tablets* [6] [5] , ao contrário dos outros, que usam dispositivos próprios como terminais [14] [3]. Por um lado, os visitan-

tes apenas precisam de descarregar a aplicação para usufruírem de uma experiência alargada. Por outro lado, apesar destes dispositivos serem bastante comuns, nem toda a gente possui um ou está disponível a usar o seu, o que torna a solução do dispositivo proprietário mais apropriada.

Em relação às técnicas de localização usadas, as *tags* NFC são uma forma simples, porém são necessárias várias para conseguir uma precisão razoável e a frequência usada limita a distância entre o dispositivo e a *tag* necessária para obter o ID [15]. No caso do *Cultural Heritage Museum* [5], são usados sensores *Bluetooth Low Energy* para detetar dispositivos que tenham entrado na zona de cobertura. A precisão de localização é baixa, pois apenas é possível determinar que o utilizador se encontra na área de deteção de um dispositivo *BLE*, sendo que nem todo o museu está coberto. Os outros sistemas usam a técnica da triangulação, descrita na secção 2.1.2, em que apenas é determinado o ponto de acesso mais próximo [3] [6]. Para além de ser suscetível ao efeito multi-caminhos, a técnica apenas fornece uma localização muito rudimentar, não sendo a mais apropriada para certos espaços.

2.3 Síntese final

Neste capitulo foram apresentadas tecnologias de localização e sistemas de suporte aos museus interativos.

As tecnologias estudadas dividem-se em localização com base em redes sem fios, suportada em sensores simples ou em múltiplos sensores. A solução que surge mais adequada para o caso em estudo na tese é o uso de *Dead Reckoning* em conjunto com o *Fingerprinting*, que servirá para reduzir o erro no cálculo da trajetória.

Foram também apresentados diversos sistemas de suporte aos museus interativos. Estes sistemas caracterizam-se por proporcionar ao visitante conteúdos multimédia e informação adicional, utilizando redes sem fios para comunicar com os sistemas terminais. Em termos de técnicas de localização, estes exemplos demonstram formas simples e básicas para posicionar os visitantes, sendo que algumas não se adequam ao Museu da Computação, nomeadamente as que envolvem a triangulação, dado este espaço ser muito menor do que os museus alvos destes sistemas, o que obriga a uma maior precisão de posicionamento. No entanto, são utilizadas aqui algumas tecnologias interessantes, como por exemplo os *beacons* BLE.

Capítulo 3

Arquitetura da solução

Neste capítulo, é apresentada a arquitetura do sistema desenvolvido. A secção começa com a apresentação dos aspetos de desenho do sistema, na secção 3.1, seguido de uma descrição e caracterização do espaço do museu, secção 3.2, bem como uma caracterização dos movimentos a identificar, e das possíveis trajetórias, secções 3.3 e 3.4, concluindo com a arquitetura final do sistema, detalhando os vários módulos e respetivos sub módulos, incluindo a lógica utilizada, apresentada na secção 3.5.2.

3.1 Aspetos de desenho do sistema

O objetivo desta tese é criar um sistema de navegação e localização interior para o Museu da Computação do Técnico que sirva de base ao futuro sistema de suporte do museu interativo.

Pretende-se que o visitante possa aceder aos conteúdos do museu através do seu telemóvel, sem ter de usar um dispositivo adicional como os tradicionais áudio-guias. Desta forma, tira-se partido a apetência das pessoas pelo uso do *smartphone*, como também se potencia a experiência do museu fora do tempo da visita. Assim sendo, o sistema de navegação e localização interior deve estar integrado na aplicação de interativa que o museu disponibiliza aos visitantes.

Existindo um conjunto de sensores disponíveis na generalidade dos telemóveis, estes poderão ser usados no processo de navegação e localização. Para efeitos de navegação, os sensores do telemóvel serão usados para caracterizar a trajetória, com base no conhecimento do movimento humano. A localização será definida com base no mapeamento da trajetória no espaço físico.

O museu deve estar equipado com a sua própria sensorização, de forma a garantir que a aplicação é usada mesmo que as capacidades de sensorização do telemóvel do visitante sejam limitadas. Nestas circunstancias, a localização e navegabilidade estão limitadas à informação fornecida pelos sensores externos. A informação de navegabilidade será muito restrita, uma vez que apenas se consegue saber se o visitante se encontra, ou não, próximo de determinado sensor.

Este sistema de sensorização do museu deve ser de fácil instalação e operação. Adicionalmente, a informação fornecida deve ser facilmente disponibilizada à aplicação móvel, não requerendo telemóveis com características especiais. Neste contexto, o uso da tecnologia *BLE* surge como a opção mais

adequada.

3.2 Descrição e caracterização do espaço

O conhecimento do espaço físico é decisivo na determinação da informação de navegação e posicionamento. Nas secções seguintes será efetuada uma caracterização do espaço físico.

3.2.1 Museu da Computação do IST

O Museu da Computação do Técnico iniciou o seu funcionamento aquando do encerramento do centenário da escola e contém um acervo de equipamento pertença do IST, doado por pessoas ou instituições parceiras. Este museu pretende dar a conhecer ao público a história da computação, inserida no contexto da Escola e das instituições a esta associada.

O museu está localizado no antigo centro de informática do campus do Taguspark e contém um conjunto de peças, que se agrupam tematicamente nos seguintes grupos:

- **Grupo IBM**: subconjunto de componentes do IBM 360, destinado ao cálculo científico, que foi instalado no centro de cálculo do IST no início da década de 70. Deste conjunto faz parte: uma impressora (IBM-R), uma perfuradora de cartões (IBM-29) e o CPU (IBM-2044).

- **Grupo HP**: conjunto completo dum micro-computador da HP usado para processamento digital de sinais, na década de 70, que pertencia ao Centro de Análise e Processamento de Sinais do IST.

- **Grupo Apple**: constituído por uma mesa que agrupa os equipamentos que ilustram o conceito de escritório pessoal da Apple na década de 70. Neste conjunto há que realçar uma peça que será um dos grandes atrativos do museu: um Macintosh aberto, com a assinatura de toda a equipa de projeto, incluindo Steve Jobs.

- **Grupo de Componentes**: constituído por uma mesa que ilustra a evolução da tecnologia eletrónica, englobando, diversas gerações de componentes, válvulas, relés, transístores e circuitos impressos, mas também memórias, CPUs etc..

- **Grupo Armazenamento**: constituído por uma mesa que agrupa diferentes gerações de equipamento de armazenamento, tais como discos, disquetes, *pens* etc..

- **Grupo Portáteis**: constituído por uma mesa que agrupa diferentes gerações e diferentes tipos de equipamentos portáteis.

- **Grupo Computadores Pessoais**: constituído por uma mesa com diferentes tipos de computadores pessoais.

- **Grupo de Redes**: constituído por uma mesa que agrupa diferentes gerações e tecnologias de equipamento de redes, que vão desde a rede telefónica até à Internet. Neste grupo é de realçar um objeto especifico: uma central RDIS, que por ter sido desenvolvida no contexto duma tese do IST, pode merecer especial interesse por parte dos visitantes.

3.2.2 Caracterização do espaço

A sala onde se encontra o Museu da Computação é um local com várias características relevantes para a implementação de um sistema de localização, nomeadamente o teto e chão falso, que permitem a instalação posterior de sensores específicos, para detetar a presença de visitantes num certo local. O chão é composto por múltiplas quadrículas, com dimensões 59.8 x 59.8 cm, sendo que a granularidade da posição foi definida como um quadrado. A localização é tida em referência a um sistema de coordenadas *x* e *y*, com estes eixos definidos como o limite inferior e o limite esquerdo da sala, respetivamente. Este espaço foi equipado com *beacons* BLE, colocados em posições estratégicas, que serão usados para efeitos de localização.

Dispersos pelos vários recantos do museu estão múltiplas mesas onde estão expostos vários equipamentos e componentes, bem como certas máquinas que possuem base ou suporte próprio. Neste trabalho, foram apenas consideradas as mesas com conteúdos mais relevantes, tendo outras sido marcadas como obstáculos. De modo a identificar locais específicos no museu, foram definidos grupos, que relacionam mesas com o mesmo tipo de materiais expostos. A cada grupo podem pertencer uma ou mais mesas, sendo que cada uma é identificada individualmente, sendo usada uma nomenclatura que a relaciona ao grupo.

Existem no espaço do museu alguns obstáculos, tais como pilares, e pequenas mesas, que restringem a circulação das pessoas. Apesar de existirem duas portas, apenas uma delas se encontra acessível. Dada a disposição das mesas em fila, existem cinco corredores principais onde os visitantes podem circular, três na vertical e dois na horizontal. Os utilizadores são livres de percorrer o museu como quiserem, não existindo percursos pré-definidos como guia.

A figura 3.1 ilustra a planta do espaço do museu utilizado para a demonstração da tese, com a inclusão da localização dos sensores BLE existentes.

3.3 Caracterização do movimento

A precisão da informação de navegação depende fortemente do conhecimento e caracterização do movimento humano. As secções seguintes descrevem o trabalho de caracterização do movimento que precedeu e condicionou a elaboração da arquitetura do sistema.

3.3.1 Aplicação para recolha de dados

A primeira etapa para realizar uma análise do movimento foi o desenvolvimento de uma aplicação personalizada para a recolha e armazenamento das medidas dos vários sensores em ficheiros, utilizados posteriormente para traçar gráficos. Esta *app* simples regista todos os valores medidos pelos três sensores considerados: **acelerómetro**, **giroscópio** e **magnetómetro**. A interface de utilizador contém dois botões, um para começar (START) e outro para terminar (STOP) a gravação dos dados, e um elemento para escrever o nome a atribuir aos ficheiros. Alguns aspetos da interface da aplicação, apresentada na

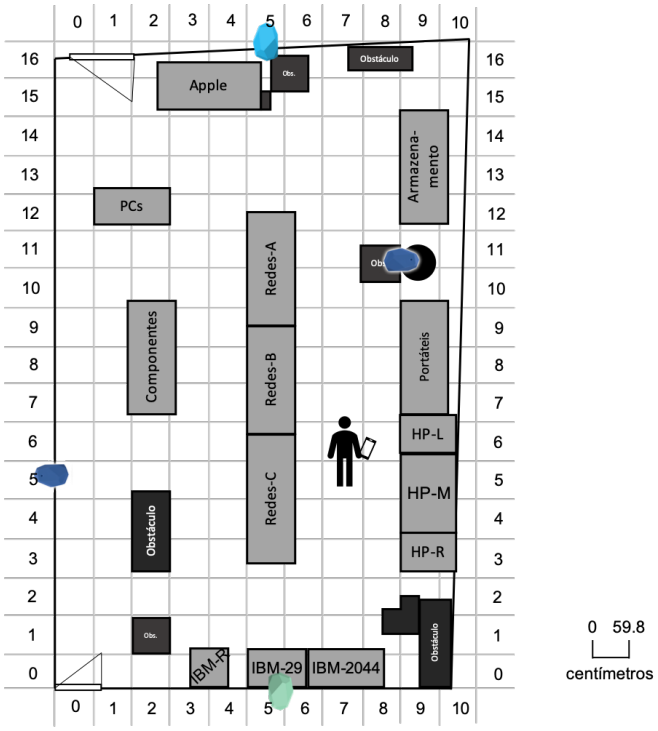


Figura 3.1: Planta do espaço do museu.

figura 3.2, foram pensados de modo a facilitar a utilização da aplicação durante a análise com múltiplos participantes.

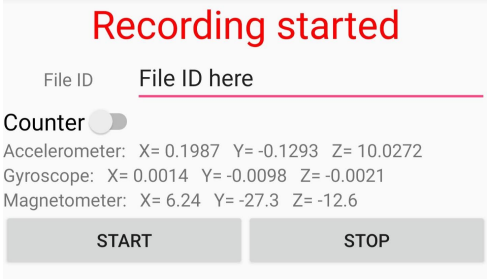


Figura 3.2: Interface da aplicação para recolha de dados.

Ao selecionar o botão START, a gravação é iniciada e de agora em diante todos os valores medidos serão guardados com o respetivo *timestamp* ou número do evento. É importante ter em conta que os novos dados de cada sensor são recebidos em diferentes instantes de tempo, de modo que para permitir a comparação entre os mesmos, os valores anteriores devem ser repetidos. Quando o registo é finalizado, ou seja, quando o utilizador seleciona o botão STOP, os valores registados são então guardados em ficheiros *.csv* individuais para casa sensor, utilizando uma biblioteca retirada de [17]. Apesar da simplicidade da aplicação, esta será extremamente útil para poder analisar como variam os valores medidos dos sensores, não só para o caminhar, mas também para outras movimentações como o virar numa esquina, algo essencial para estimar a sua localização. O código desta aplicação está presente no anexo A.1.

3.3.2 Análise dum passo

O movimento humano pode ser considerado, na sua forma mais simples, como um conjunto de passos. Tendo por base estudos anteriores de uso de telemóveis para análise do movimento [10], [2], pode-se considerar que a passada causa uma assinatura específica no sinal de alguns sensores. Esta secção tem por objetivo fazer a caracterização do passo humano, considerando diferentes tipos de pessoas.

Análise com sensorização múltipla

O primeiro conjunto de experiências teve como objetivo identificar a assinatura do sinal de cada sensor. Para o efeito, o sistema foi testado sempre com o mesmo telemóvel (Samsung Galaxy Note 8) e o mesmo utilizador. Utilizando a aplicação criada para o efeito, uma primeira experiência foi realizada, gravando as medidas registadas durante um percurso simples, em linha reta, em que foram dados apenas 5 passos num andar normal. Este percurso foi realizado por uma pessoa, (rapaz com 23 anos e 1,78 m de altura), num local interior com um chão plano. O dispositivo foi segurado com a mão direita numa posição próxima ao peito, com orientação vertical e o ecrã sempre voltado para cima, permanecendo imóvel em relação ao corpo. A experiência foi repetida várias vezes, de forma a obter dados relevantes e descartar erros. No entanto, para efeitos de análise foi selecionada apenas uma das experiências.

Os dados de cada sensor foram utilizados para construir a série temporal respeitante ao período de tempo em que se procedeu à recolha de dados durante o teste. Os gráficos foram desenhados com recurso à ferramenta Plotly [18], que foi usada por permitir manipular um volume de dados significativo de forma rápida e simples, pelo facto do processamento ser feito na *cloud*. Os gráficos resultantes são apresentados nas figuras 3.3, 3.4, 3.5, respetivamente para o acelerómetro, giroscópio e magnetómetro.

Figura 3.3: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso simples com 5 passos.

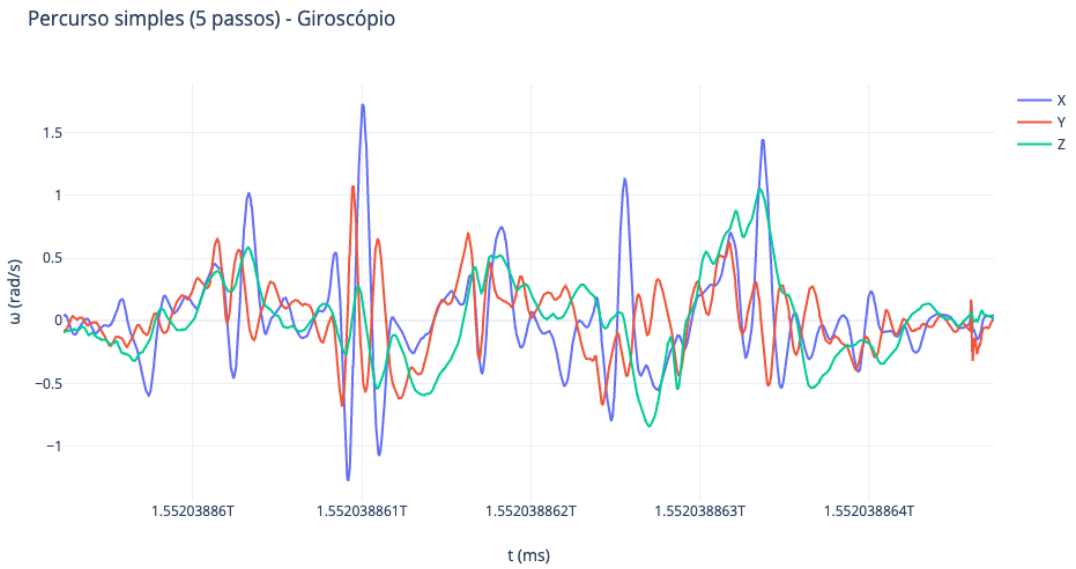


Figura 3.4: Gráfico traçado com as medidas do sensor giroscópio, recolhidas durante o percurso simples com 5 passos.

Figura 3.5: Gráfico traçado com as medidas do sensor magnetómetro, recolhidas durante o percurso simples com 5 passos.

A observação dos gráficos das medidas dos três sensores permite determinar que acelerómetro e o giroscópio são aqueles que mostram variações mais significativas. Isto deve-se ao facto de serem sensores de movimento, enquanto que o magnetómetro é um sensor de posição, tal como descrito na secção 3.5.2. Nos dois gráficos é possível perceber que a evolução de cada sinal exibe um comportamento que se ”repete”, ainda que, com variações: um passo é assinalado por uma série de extremos no sinal. Todavia, a forma desta assinatura difere entre os dois sensores, sendo por isso necessário analisar os dois gráficos independentemente.

Começando com o gráfico do acelerómetro, observe-se a assinatura do segundo passo, que tem maior amplitude, na figura 3.6. Quando não existe movimento, o sinal encontra-se por volta dos 8 m/s^2

para o eixo *z*, 4 m/s^2 para o eixo *y* e 0 m/s^2 para o eixo *x*. O balançar do braço durante a passada causa uma subida relativamente lenta no sinal até ser atingido o valor máximo, imediatamente seguido de uma queda abrupta até ao valor mínimo. No caso do eixo *x*, pode-se observar um descida mais lenta que nos outros eixos. Depois disso, o sinal volta a subir, quase estabilizando por volta dos valores normais. Esta assinatura é mais acentuada nos eixos *y* e *z*, mas principalmente neste último, onde a diferença entre os dois extremos é de quase 13 m/s^2. Isto acontece porque o movimento do braço e da mão causam um maior deslocação do dispositivo no eixo perpendicular ao ecrã (eixo *z*), do que no eixo vertical ao dispositivo (eixo *y*). Os máximos encontram-se entre os 5 e 7 m/s^2 no eixo *y* e entre 13 e 16 m/s^2 no eixo *z*, enquanto que os mínimos variam entre 1 e 3 m/s^2 no eixo *y* e entre 2 e 6 m/s^2 no eixo *z*.

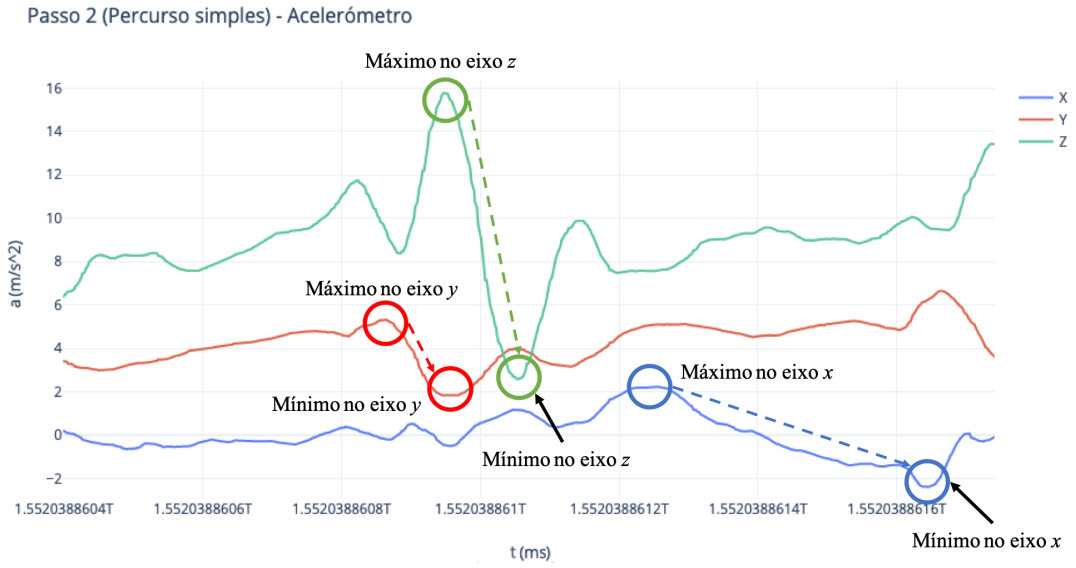


Figura 3.6: Gráfico traçado com as medidas do sensor acelerómetro, ampliando no segundo passo de modo analisar melhor a assinatura.

Relativamente ao gráfico do giroscópio, observe-se a figura 3.7. Aqui um passo é assinalado por uma espécie de efeito de ondulação na forma do sinal, que se verifica nos três eixos, sendo mais notável no eixo *x*. A assinatura caracteriza-se por uma ligeira subida até ao primeiro máximo, seguida de uma descida até ao primeiro mínimo, procedida de uma subida maior até ao máximo com maior amplitude, voltando a descer até ao segundo mínimo, subindo ligeiramente até um terceiro máximo e começa então a estabilizar. Os gráficos das figuras 3.8, 3.9 e 3.10, focam-se no sinal de cada um dos eixos individualmente.

Quando não existe movimento, a amplitude do sinal encontra-se por volta do valor 0 para todos os eixos. Porém, os diferentes sinais apresentam variações de amplitudes diferentes quando é dado um passo. No eixo do *x*, os máximos variam entre 0.5 e 1.8 rad/s, enquanto que os mínimos variam entre -1.3 e 0 rad/s. No eixo do *y*, os máximos variam entre 0.2 e 1.1 rad/s, enquanto que os mínimos variam entre -0.8 e 0 rad/s. Já no eixo do *x*, onde a variação é menor, os máximos variam entre 0.2 e 1.1 rad/s, enquanto que os mínimos variam entre -0.9 e 0 rad/s.

É importante notar que a descrição apresentada da forma da assinatura de um passo é generali-

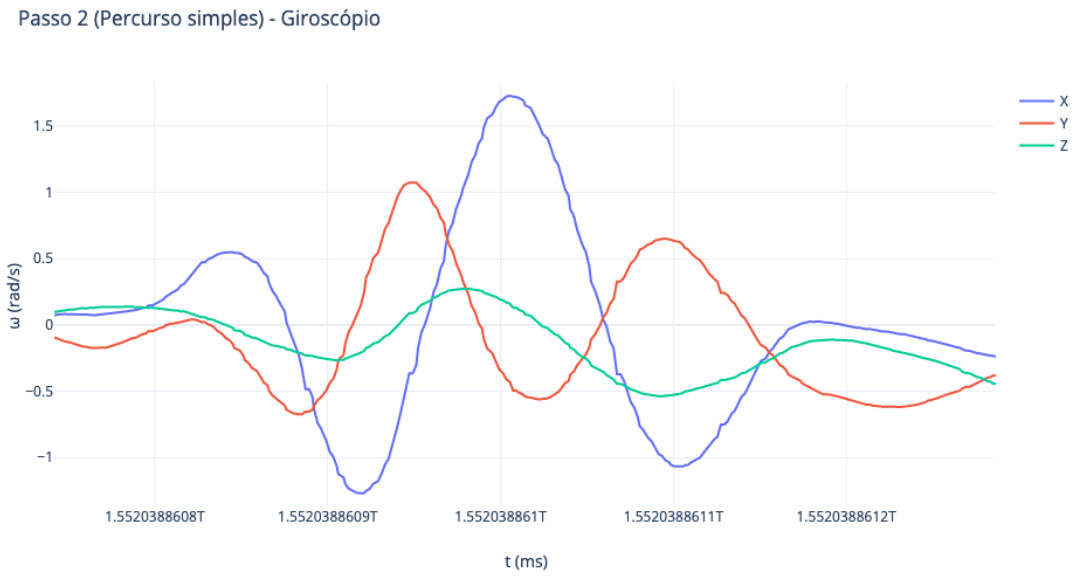


Figura 3.7: Gráfico traçado com as medidas do sensor giroscópio, ampliando no segundo passo dado.

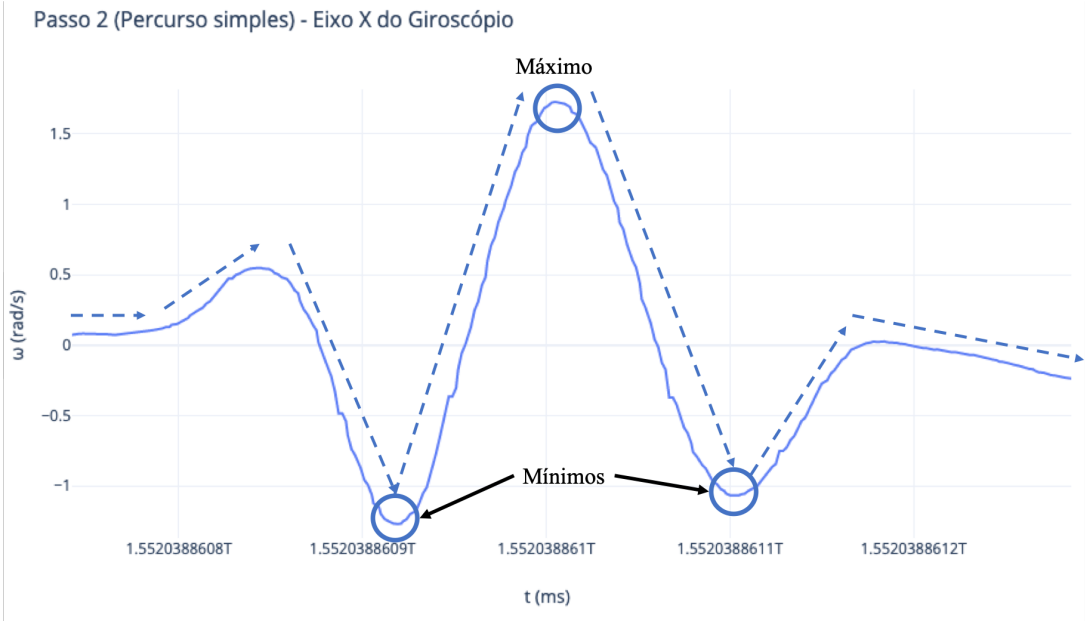


Figura 3.8: Gráfico traçado com as medidas do eixo *x* do sensor giroscópio, ampliando no segundo passo dado.

zada e que nem sempre ocorre desta forma, compare-se estes gráficos com outros traçados com os dados recolhidos numa outra experiência posterior, nas figuras 3.11 e 3.12, para ver as diferenças. Isto acontece por várias razões. O modo como o passo é dado, a posição do dispositivo, um movimento brusco feito durante ou previamente, tal como qualidade dos sensores, pode influenciar a forma real do sinal. Isto significa que é impossível construir um pedómetro perfeito, ou seja, vão haver sempre falsos positivos e negativos com que o sistema vai ter de lidar. É possível, no entanto, fazer uma distinção entre a assinatura de um passo e assinatura de certos movimentos aleatórios, no sinal de sensores como o giroscópio, que é mais suscetível, de modo a poder ratificar estes casos, algo que é analisado no final desta secção.

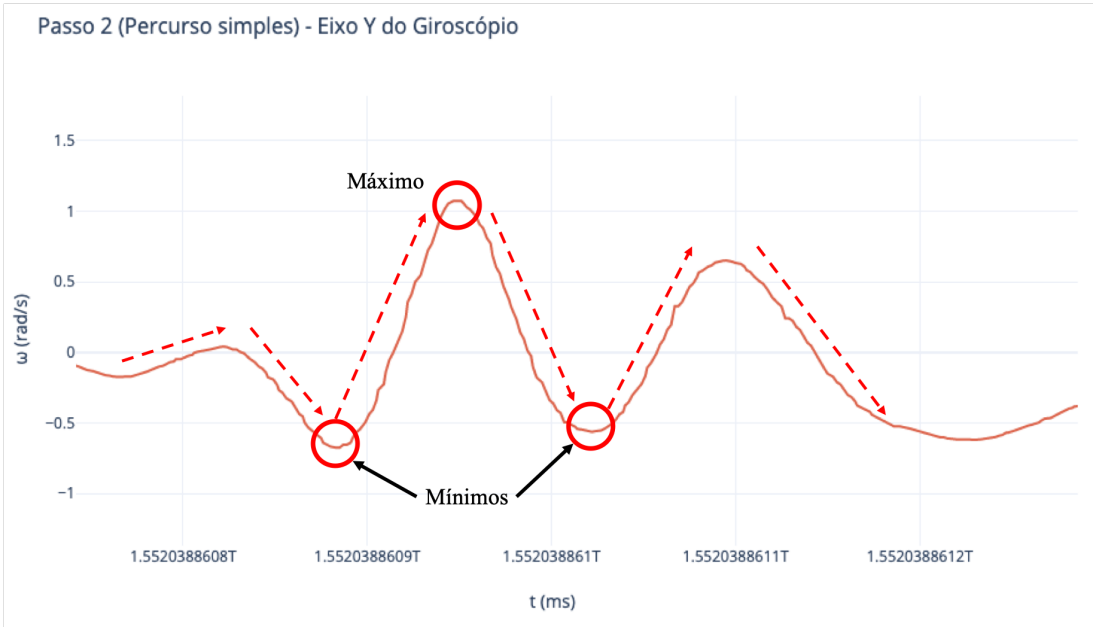


Figura 3.9: Gráfico traçado com as medidas do eixo *y* do sensor giroscópio, ampliando no segundo passo dado.

Figura 3.10: Gráfico traçado com as medidas do eixo *z* do sensor giroscópio, ampliando no segundo passo dado.

Comparando os gráficos traçados com os dados obtidos dos dois sensores, é possível concluir que o acelerómetro demonstra a assinatura mais simples e única, identificada apenas com um máximo e um mínimo. Por isso, foi selecionado como o principal sensor a usar no pedómetro do sistema, apesar de a possibilidade de usar o giroscópio também ter sido explorada. No entanto, o giroscópio será bastante útil para despistar os falsos positivos. Com esta primeira análise, foi possível começar a desenvolver uma versão final do pedómetro que cumprisse os requisitos estabelecidos para o sistema.

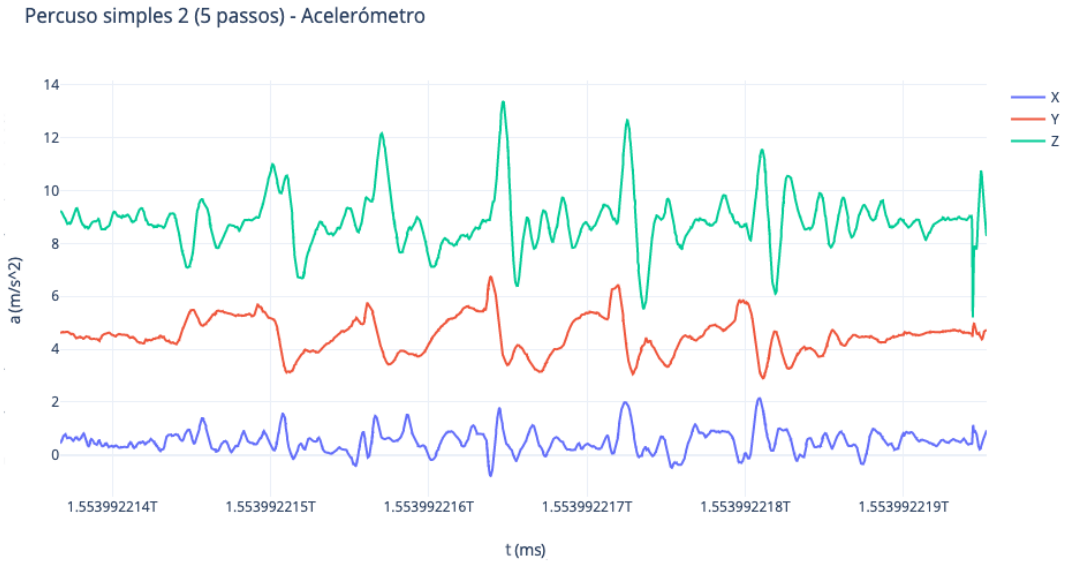


Figura 3.11: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante um segundo percurso simples com 5 passos.

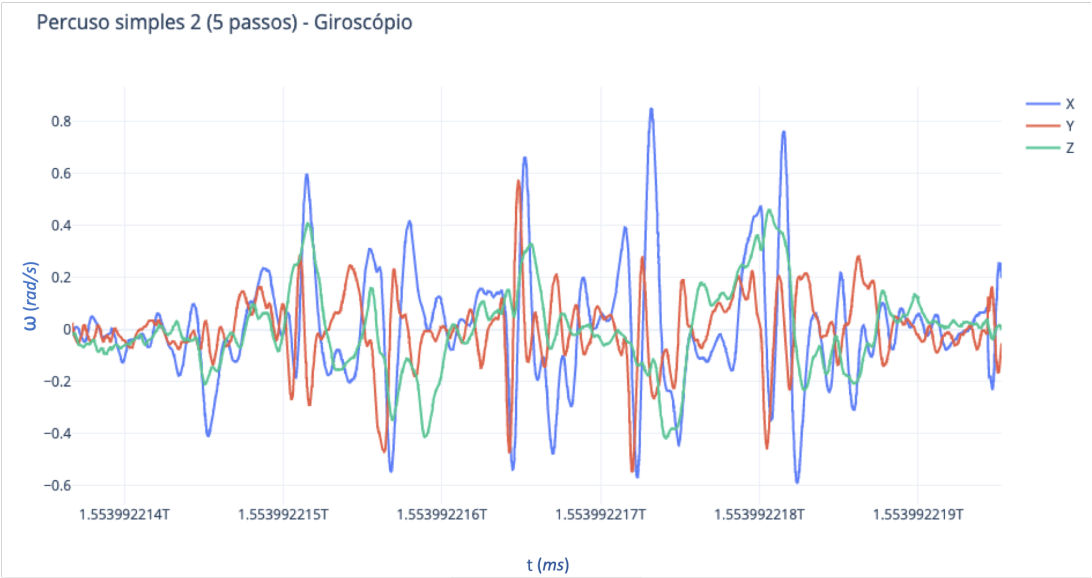


Figura 3.12: Gráfico traçado com as medidas do sensor giroscópio, recolhidas durante um segundo percurso simples com 5 passos.

Análise com múltiplos participantes

Sendo que os visitantes esperados no museu serão de diferentes géneros, idades e alturas, foi necessário garantir que estas assinaturas se verificam sempre, bem como observar as possíveis diferenças de pessoa para pessoa. Para tal, foi pedido a várias indivíduos, cujas idades variam entre os 4 e os 61 anos, que percorressem um pequena distância, de modo a analisar o seu movimento. O número de voluntários da amostra de certas faixas etárias pode ser considerado baixo e não representativo, no entanto, problemas com tempo, disponibilidade e a quantidade de dados limitaram a participação. A tabela da figura 3.13 descreve os vários participantes.

Nome	Idade	Altura (m)
Crianças e adolescentes		
Diogo	4	< 1
Gabriel	10	1.40
Mariana	12	1.55
Ana S.	15	1.55
Jovens adultos		
Daniel	20	1.74
Ana C.	23	1.52
Ricardo	34	1.82
Adultos de meia-idade		
Lúcia	44	1.62
Paula	44	1.57
José	53	1.63
Maria L.	53	1.55
Ilda	57	1.50
Maria H.	61	1.65

Figura 3.13: Identificação dos múltiplos participantes da experiência, incluindo a altura e idade, á data da recolha de dados.

Utilizando a aplicação construída para este propósito, os participantes foram instruídos a dar 5 passos em linha reta, com o seu andar normal, sempre num ambiente fechado, segurando o telemóvel com a mão direita, ou, no caso da criança mais pequena, com as duas mãos. Tal como nas experiências da análise inicial, a posição do dispositivo em relação ao corpo mantém-se, ficando á altura do peito e em orientação vertical (retrato), estando relativamente imóvel em relação ao corpo. O chão onde os participantes circularam foi diferente em certos casos, no entanto, sempre sobre terreno plano e em locais interiores. Antes de começarem, os participantes foram instruídos a selecionar o botão START, para começar a registar os valores, e após terem dado os cinco passos, selecionar o botão STOP, para terminar o registo. Apesar das instruções específicas que foram dadas, estas nem sempre foram cumpridas corretamente, por isso deve-se considerar alguns passos extra e movimentações do telemóvel. Em algumas situações, os participantes fizeram o percurso pedido, tendo no final invertido a sua orientação e repetido os passos. Os gráficos criados a partir dos dados obtidos do acelerómetro são apresentados nas figuras seguintes. De notar, que devido a um *bug* de origem desconhecida, nem todos os gráficos foram traçados com o *timestamp* das medidas no eixo X.

Observando os gráficos das passadas dos participantes mais novos, nomeadamente das crianças (figuras 3.14 e 3.15), torna-se claro que a assinatura distinta no eixo *z*, característica de um passo normal, é praticamente inexistente. Ampliando nalgumas zonas dos gráficos é possível encontrar evidências de um passo, no entanto, a forma do sinal dificulta a sua identificação. O mesmo acontece o caso das duas adolescentes (figuras 3.16 e 3.17), onde a assinatura é encontrada raramente. Estes resultados contrariam a percepção inicial de que apenas a altura está relacionada com o tamanho da passada e demonstram que a estatura do corpo e o padrão de mobilidade, também influenciam a variação nos dados do sensor. Observando estes gráficos, verifica-se que o sinal exibe uma variação muito aleatória quando o utilizador é muito jovem, pelo que dificilmente se pode usar para obter a assinatura dum passo. Neste contexto, a navegabilidade e posicionamento terão de ser relegados para os

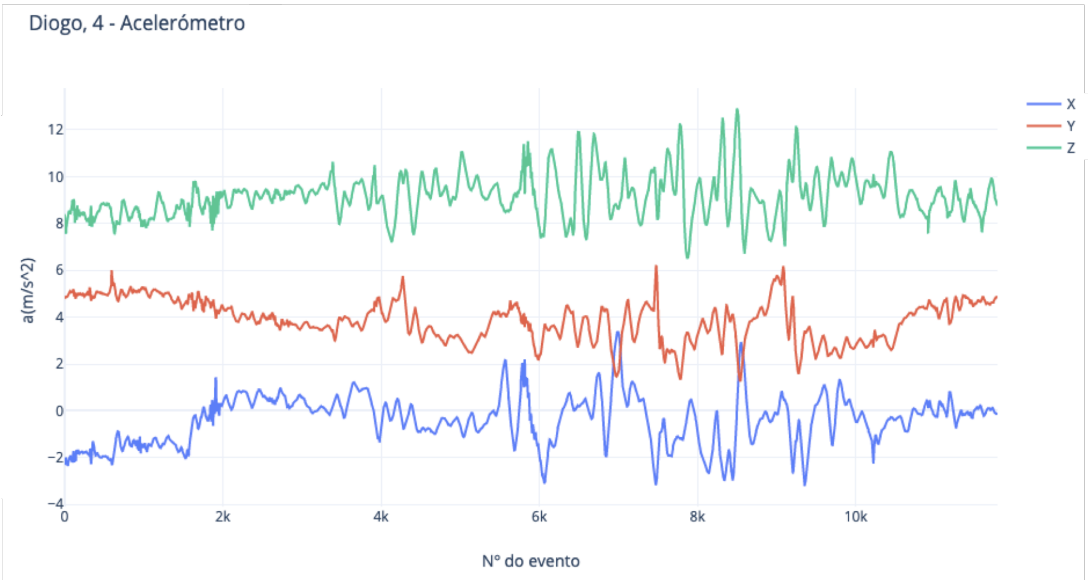


Figura 3.14: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Diogo (4 anos). De notar que devido a um problema de origem desconhecida, o gráfico é traçado com

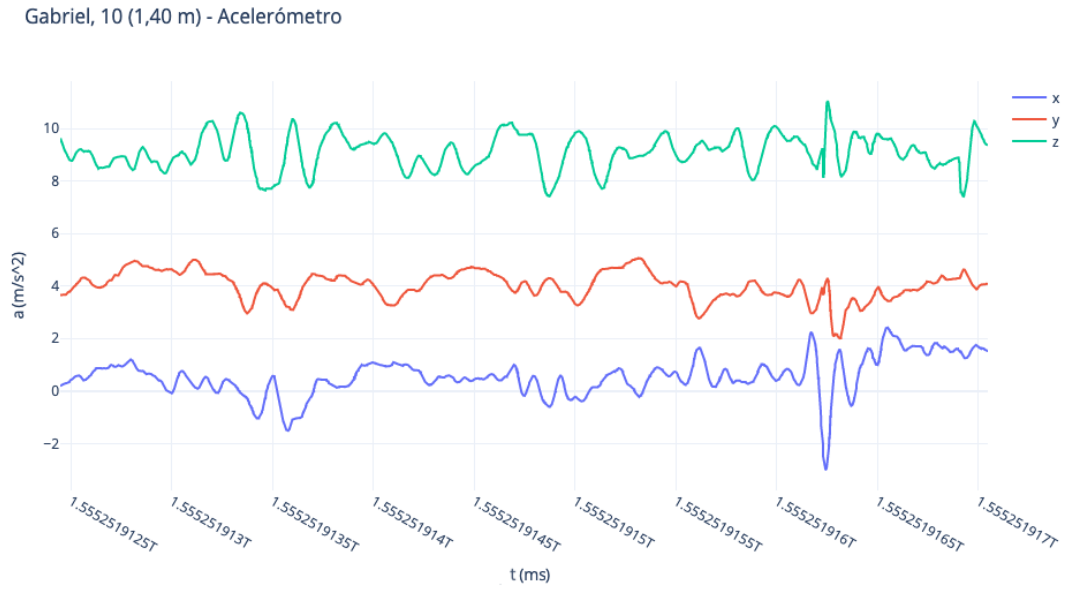


Figura 3.15: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Gabriel (10 anos).

sensores externos.

Já na casa dos 20 anos, a análise do gráfico do movimento de Ana C., na figura 3.19, revela uma situação curiosa. Durante a recolha de dados ocorreu uma inversão dos dados dos eixos *y* e *z*. Este evento poderia ser considerado apenas um *bug* na aplicação, não fosse a *app Sensor Kinectics*, usada complementarmente para visualizar o sinal dos sensores em tempo real, ter comprovado isto numa demonstração após o percurso. Apesar desta situação não parecer frequente (esta foi a única vez que foi observada), é possível que aconteça durante uma visita e cause um elevado erro na localização.

Nos gráficos do resto dos jovens adultos, dos 20 aos 34 anos (figuras 3.18 e 3.20), a assinatura

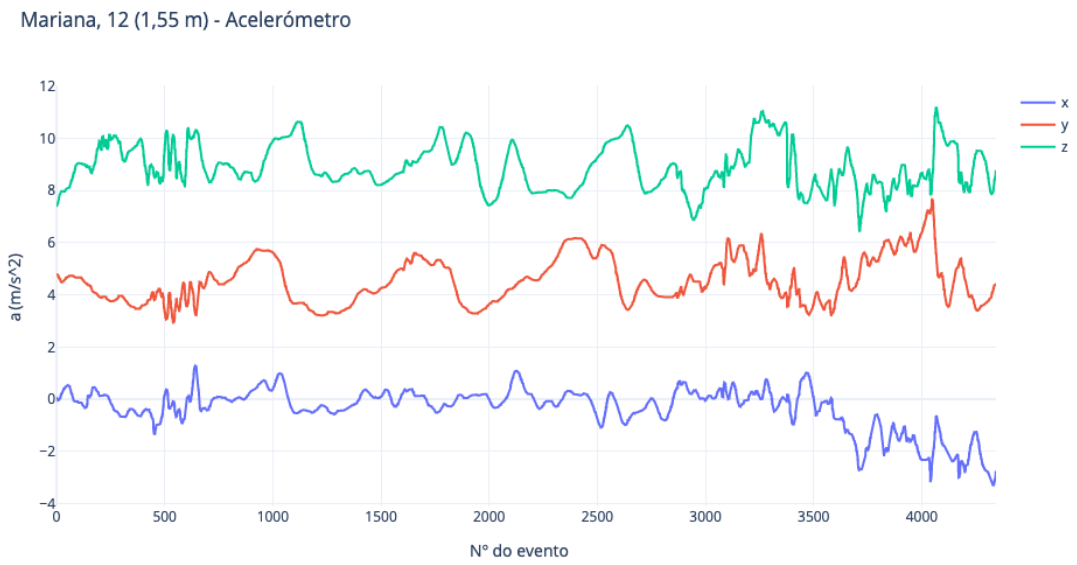


Figura 3.16: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Mariana (12 anos).

Figura 3.17: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Ana S. (15 anos).

específica de cada passo é relativamente fácil de identificar. Mesmo o gráfico da figura 3.19, em que ocorre esta situação incomum, ignorando a troca de eixos, as assinaturas dos cinco passos são facilmente reconhecíveis, sendo, no entanto, a variação do sinal menor que noutros gráficos, o que pode dever-se á altura. Nos primeiros passos, observa-se uma menor amplitude no máximo quando comparada com os seguintes, que se deve ao balançar da perna ser menor logo após o repouso, altura em que os dois pés estão juntos. Este aspeto ocorre no caso dos demais participantes.

Para os adultos de meia-idade, dos 44 aos 61 (figuras 3.21 a 3.26), a situação mantém-se, sendo que em alguns casos a interferência no sinal de certos movimentos de manobra do telemóvel, bem como algumas inversões de marcha, causaram algum ruído nas assinaturas dos passos. Também

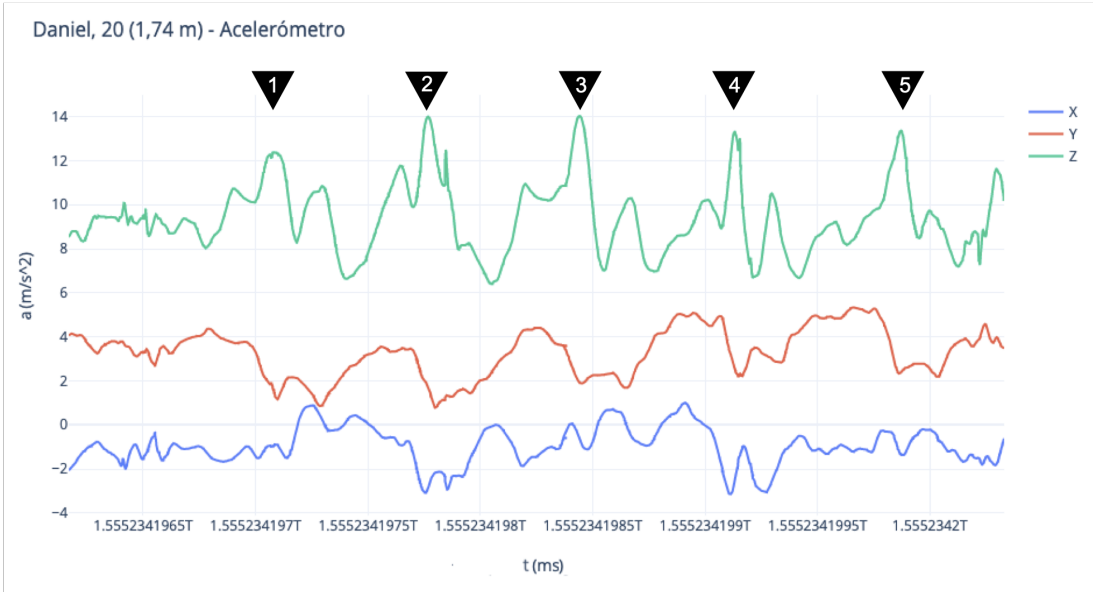


Figura 3.18: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Daniel (20 anos), com as assinaturas dos passos identificadas.

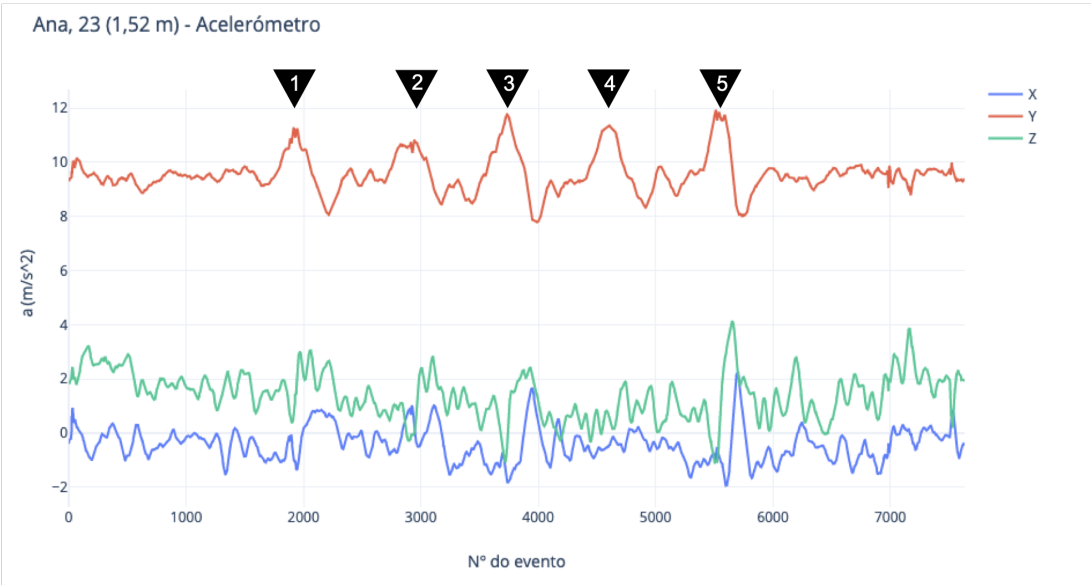


Figura 3.19: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Ana C. (23 anos), com as assinaturas dos passos identificadas.

podem ser considerados aqui outros aspetos relacionados com a idade. No entanto, a forma básica da assinatura mantém-se, com um par máximo-mínimo, que no entanto, tem diferentes amplitudes, dependentes da estatura de cada pessoa.

Análise de movimentos aleatórios

Um problema do pedómetro é a sensibilidade a movimentos bruscos, tais como estender o braço, mexer a mão ou baixar o tronco (este movimento é referido como 'vénia' em algumas experiências), que causam falsos positivos, reduzindo significativamente a precisão do sistema.

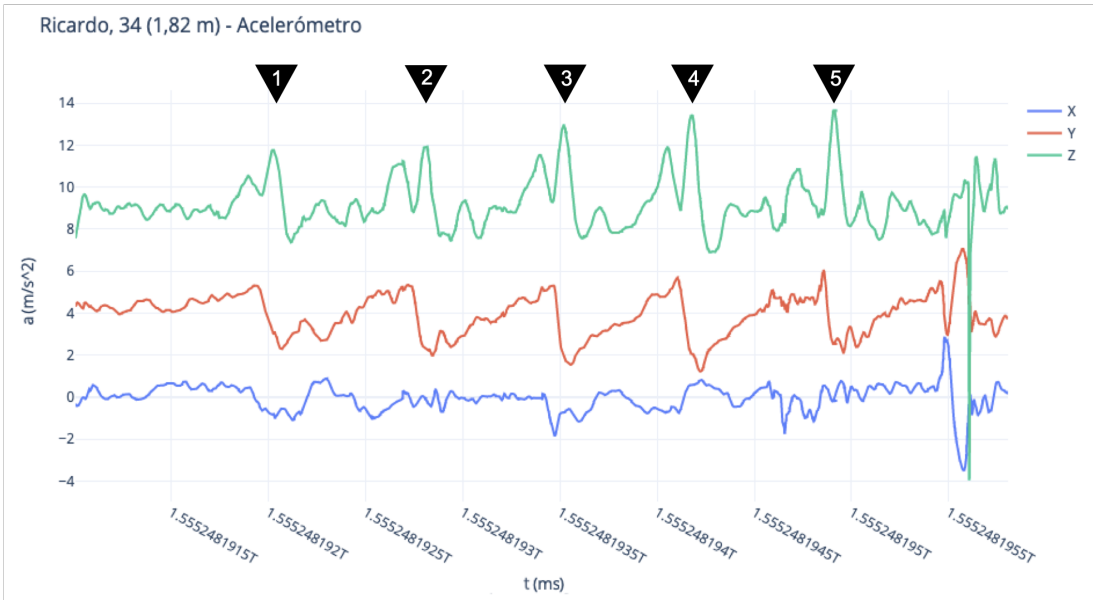


Figura 3.20: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Ricardo (34 anos), com as assinaturas dos passos identificadas.

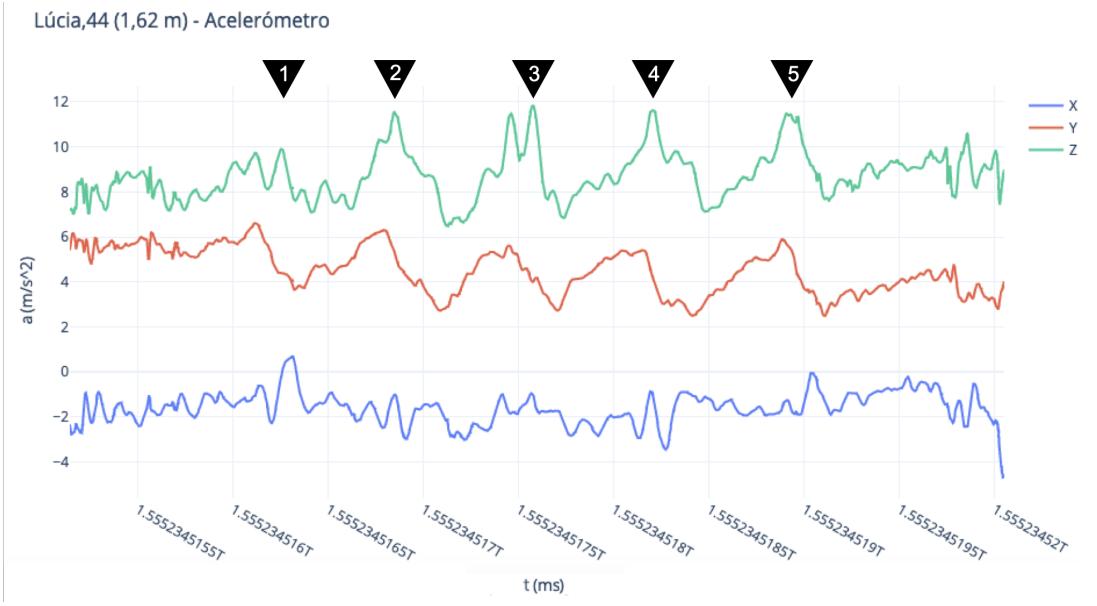


Figura 3.21: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Lúcia (44 anos), com as assinaturas dos passos identificadas.

A análise inicial demonstrou que o giroscópio exibe uma assinatura específica para a passada humana, podendo ser utilizado para ratificar um possível passo detetado. Contudo, para fazer esta assunção é necessário realizar uma análise especifica dos efeitos deste movimentos, de forma a que se possa comprovar que a forma do sinal resultante não corresponde à assinatura de um passo. Utilizando o mesmo telefone, num local interior e com o utilizador imóvel, foram feitas duas experiências distintas, para os vários tipos movimentos, exemplificados na figura 3.27:

- Movimento ”vénia”, em que o utilizador se debruça sobre algo, para observar melhor e volta a erguer-se.

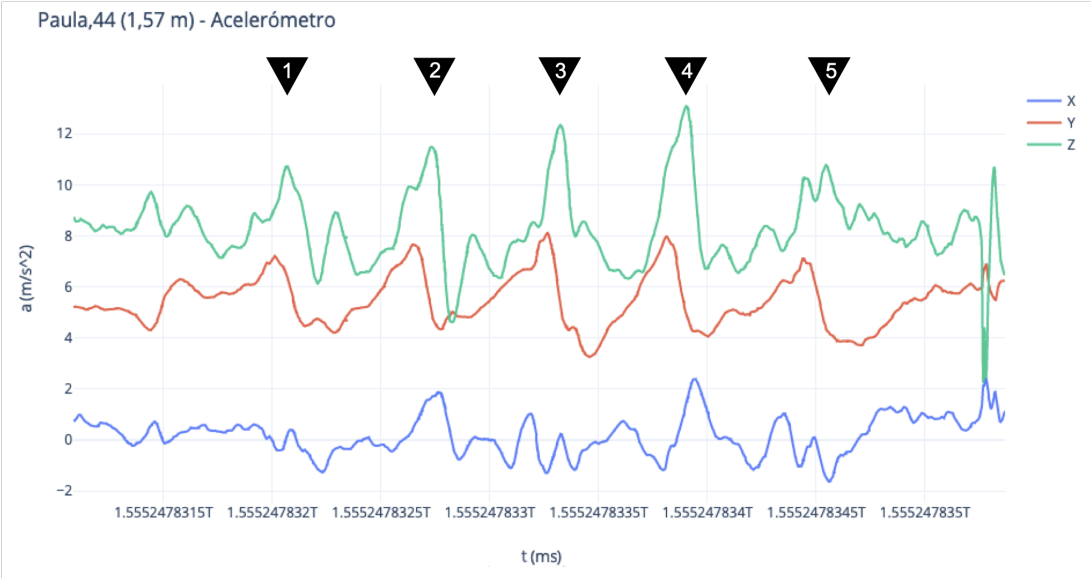


Figura 3.22: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Paula (44 anos), com as assinaturas dos passos identificadas.

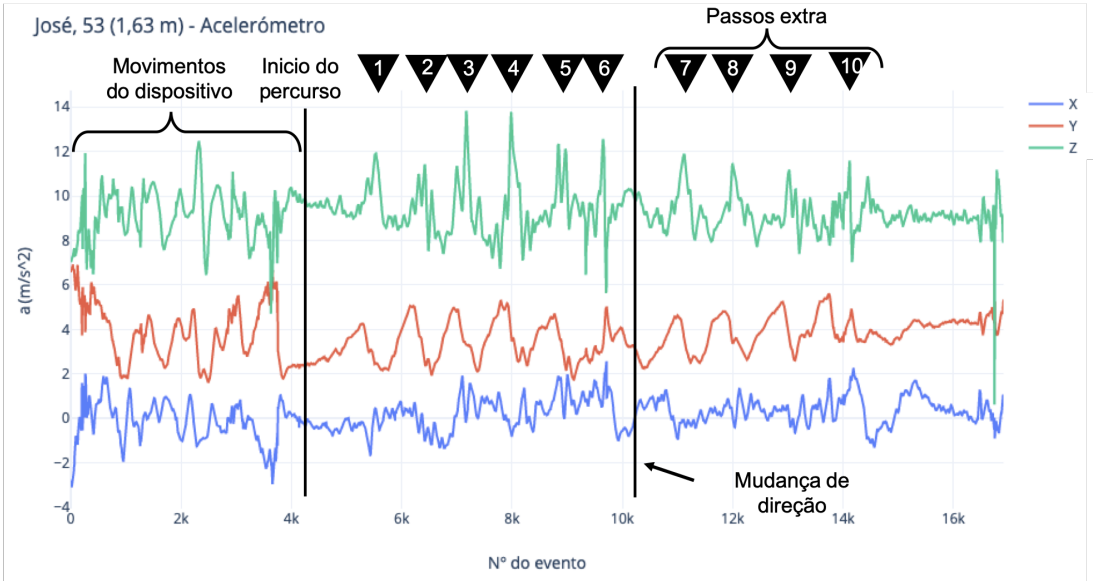


Figura 3.23: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de José (53 anos), com as assinaturas dos passos identificadas. De notar, que certos movimentos do dispositivo, que não são passos, são assinalados aqui, bem como uma mudança de direção e alguns passos extra. Isto deve-se a algumas dificuldades por parte do utilizador em compreender a interface da aplicação e em cumprir algumas das instruções, que obrigaram á intervenção do aluno.

- Movimentações do braço, tais como estendê-lo ou mover o telemóvel no ar.

Os gráficos resultantes das medidas do sensor giroscópio são apresentados nas figuras 3.28, para o movimento ”vénia”e 3.29 para os movimentos aleatórios do dispositivo.

Começando pelo gráfico do primeiro movimento, é possível observar dois picos em sentidos contrários no sinal do eixo *x*: um mínimo menor que -2 rad/s, durante o movimento para baixo; e um máximo maior que 2 rad/s, durante o movimento para cima. Estas assinaturas não são perfeitas, pois o sinal varia com frequência quando está a subir ou a descer. Observando de novo os gráficos dos percursos da

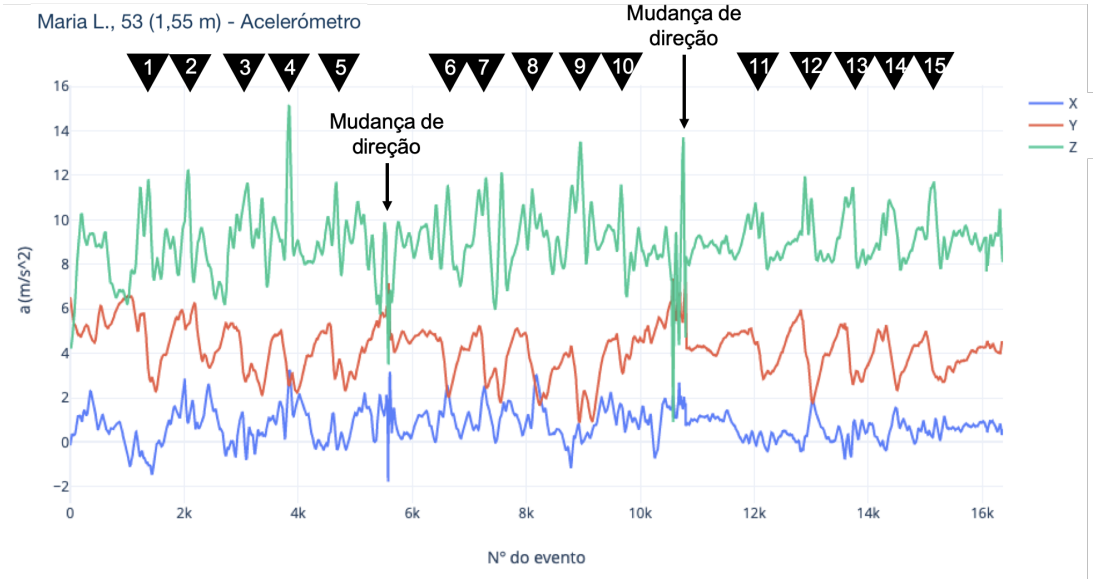


Figura 3.24: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Maria L. (53 anos), com as assinaturas dos passos identificadas. De notar, que durante o percurso foram feitas duas mudanças de direção e dados 10 passos extra. Isto deve-se a algumas dificuldades por parte do utilizador em compreender a interface da aplicação e em cumprir algumas das instruções, que obrigaram a intervenção do aluno.

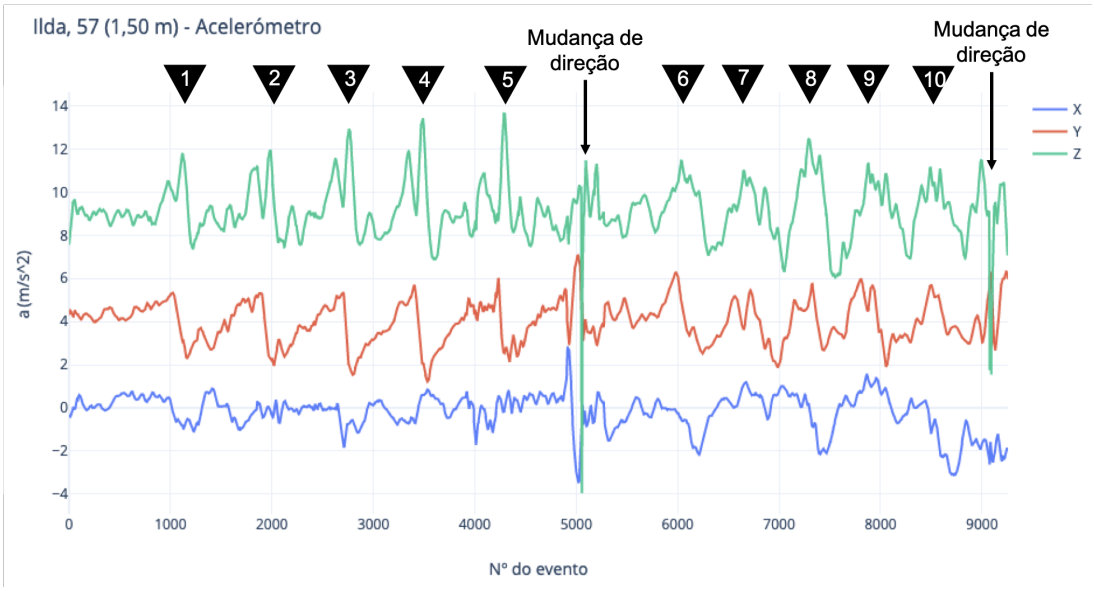


Figura 3.25: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Ilda (57 anos), com as assinaturas dos passos identificadas. De notar, que durante o percurso foram feitas duas mudanças de direção e dados 10 passos extra. Isto deveu-se ao incumprimento das instruções dadas.

análise inicial, é possível verificar que durante o passo, os valores no eixo x nunca chegam a estas amplitudes, variando normalmente entre os 1.8 e -1.5 rad/s, nos casos em que a amplitude é maior. Assim, para descartar este tipo de movimento durante a utilização do pedómetro, basta garantir que o sinal está dentro de certos limites.

Avançando para o gráfico das movimentações aleatórias do braço, é fácil identificar quando ocorre-

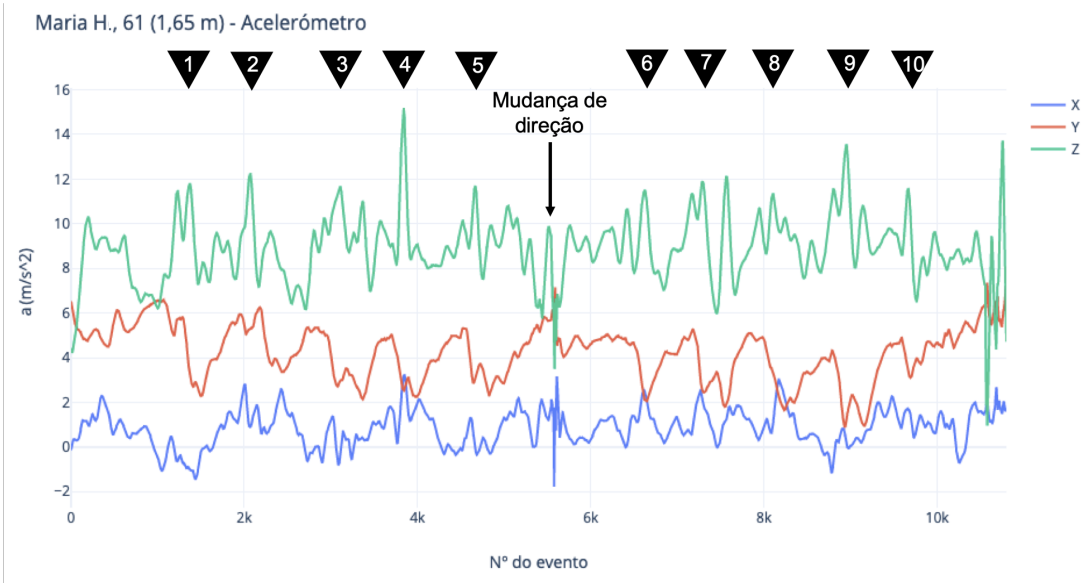


Figura 3.26: Gráfico traçado com as medidas do sensor acelerómetro, recolhidas durante o percurso de Maria H. (61 anos), com as assinaturas dos passos identificadas. De notar, que durante o percurso foram feitas duas mudanças de direção e dados alguns passos extra. Isto deveu-se a algumas dificuldades por parte do utilizador em cumprir as instruções.

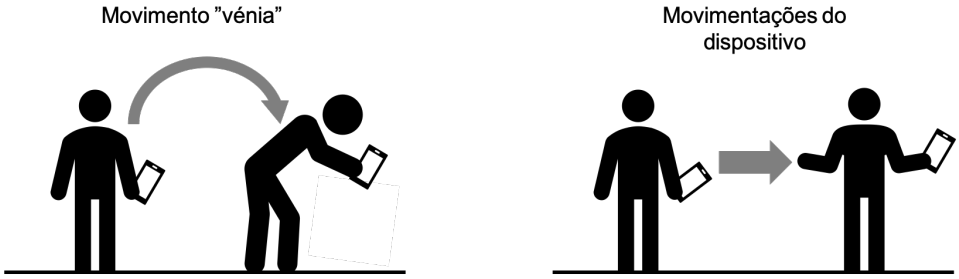


Figura 3.27: Demonstração dos movimentos aleatórios analisados. Movimento ”vénia”á esquerda, em que o utilizador se curva, enquanto segura o dispositivo; e á direita, um exemplo da movimentação do dispositivo, neste caso, o estender do braço.

ram gestos bruscos, mesmo não conhecendo o gesto em si através da análise das variações especificas nos eixos *x* e *y*, mais acentuadas que no eixo *z*. Tal como acontecia com o movimento ”vénia”, ocorrem vários picos, uns com maior amplitude que os outros e em diferentes sentidos. Os máximos nestes eixos encontram-se entre os 2 e 4.5 rad/s, enquanto que os mínimos variam entre os -2 e -6.5 rad/s. Voltando aos gráficos dos percursos da análise inicial do movimento humano, sabe-se que o sinal durante a passada nunca chega a atingir os valores acima, o que nos permite definir limites mais estritos para descartar falsos positivos.

Estes limites são definidos comparando a assinatura no acelerómetro com a do giroscópio, e verificando a variação do sinal ao longo do tempo. Considerando que apenas quando é identificado um mínimo, após encontrar um máximo e cumprindo certas condições, nesta altura é possível verificar o estado do sinal do giroscópio para despistar este tipo de movimentos. Nesta altura, o sinal está a descer e por isso definiu-se que o intervalo para os valores de todos os eixos é entre -0.5 e 0.5 rad/s.

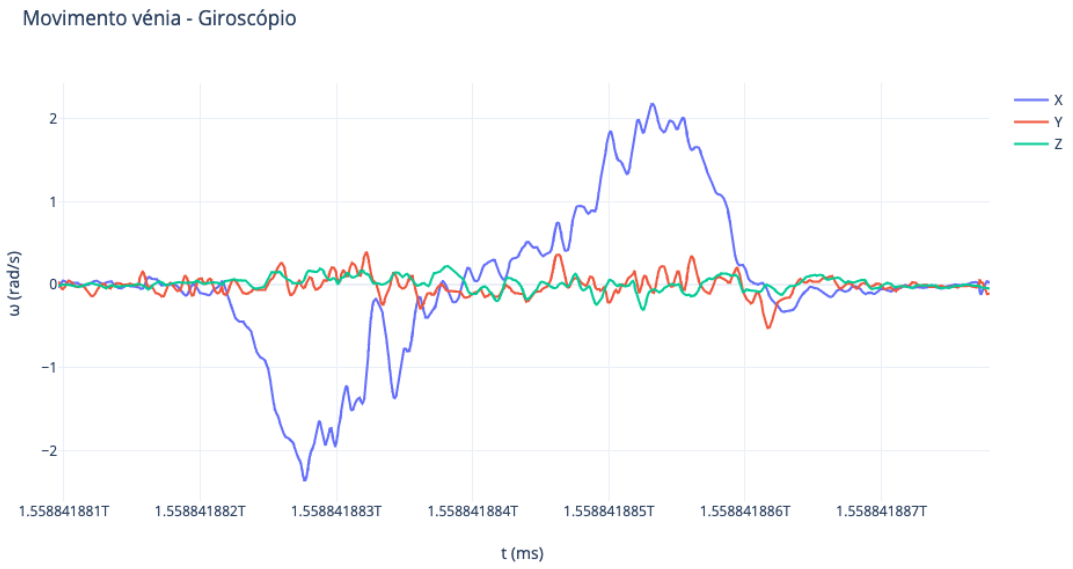


Figura 3.28: Gráfico traçado com as medidas do sensor giroscópio, durante a análise do movimento ”vénia”, em que o utilizar se debruça sobre algo.

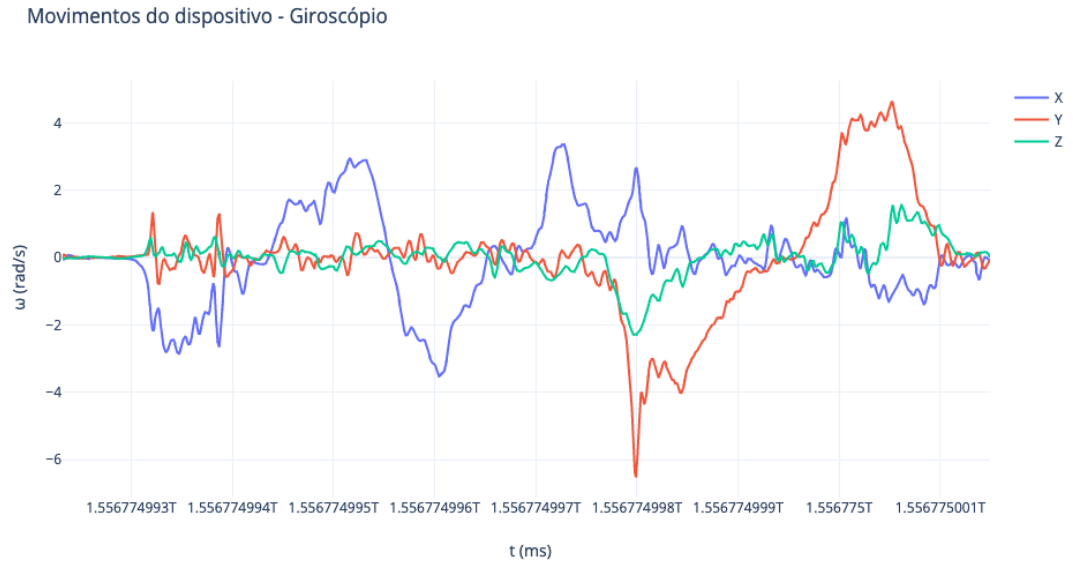


Figura 3.29: Gráfico traçado com as medidas do sensor giroscópio, durante a análise das movimentações aleatórias do dispositivo.

Síntese dos resultados

Tendo por base a análise efetuada foi possível definir o conjunto mínimo de características a incluir no sistema de medição de passos - o pedómetro. Duma forma geral, o eixo dos *z* foi aquele que melhor permitiu definir a assinatura da passada, pelo que será o que vai ser usado na concepção do sistema.

A análise dos máximos e mínimos do movimento de cada participante acima dos vinte anos permitiu então definir os limites, ou *thresholds*, necessários para identificar um passo. Assim, o máximo da assinatura deve ser maior que 11 m/s^2 e menor que 15 m/s^2, o mínimo deve ser menor que 8 m/s^2 e maior que 4 m/s^2, e a diferença entre o máximo e o mínimo tem de ser maior que 4 e menor que

9. Estes limites são o mínimo dos mínimos, sendo que aumentá-los ou diminuí-los, pode tornar o pedómetro mais suscetível a falsos positivos.

Este é um problema que se tenta resolver através da caracterização de certos movimentos aleatórios, que podem ser realizados pelos visitantes. Através da análise e comparação da forma destas assinaturas com a do passo, foi possível estabelecer limites para os dados do giroscópio, definidos como o intervalo de -0.5 a 0.5 rad/s, sem modificar aqueles determinados anteriormente, de modo a poder distinguir as duas situações. O objetivo será introduzir estas medidas para poder reduzir a possibilidade de falsos positivos.

3.3.3 Curva/viragem

Deteção por sensorização simples - giroscópio

Durante a análise inicial observou-se que quando os participantes da experiência invertiam a direção do seu percurso, ocorria uma alteração específica nos dados dos sensores do acelerómetro e do giroscópio. Esta assinatura poderia ser relevante para posicionar o utilizador no espaço, em conjunto com o pedómetro. Para tal, foi necessária a uma análise detalhada à forma como variam os dados recolhidos quando o utilizador vira. Apesar de tanto o acelerómetro como o giroscópio registarem esta variação, este último foi escolhido por estar relacionado com a rotação do dispositivo, bem como o facto de o acelerómetro já ser usado noutros elementos.

Para a recolha foi utilizada a aplicação já desenvolvida, a correr no telemóvel Samsung Galaxy Note 8. Foram feitos dois percursos, um com viragem á esquerda e outro á direita. Os percursos foram feitos em condições normais, em locais interiores, com chão plano, começando em linha reta e depois virando numa esquina, para que as mudanças de direção correspondessem a uma volta de 90º. Ambos os percursos começavam a alguma distância, de modo a observar como as variações causadas pelos passos afetavam a assinatura da mudança de direção.

Os gráficos traçados com os resultados obtidos são apresentados nos gráficos das figuras 3.30 (para a viragem á esquerda) e 3.31 (para a viragem á direita).

Começando por analisar o gráfico da viragem à esquerda, é possível observar um pico, um máximo que se distingue facilmente do resto do sinal, chegando aos 2.3 rad/s. No caso da viragem á direita, o pico ocorre na direção contrária, um mínimo igualmente distinto, perto dos 2 rad/s. Ambos estes extremos são mais acentuados no eixo *z*, razão pela qual é o único eixo apresentado nos gráficos. Isto deve-se facto do movimento do dispositivo durante a viragem ocorrer apenas no plano horizontal, permanecendo fixo no plano vertical. [1].

Síntese dos resultados

Tendo por base a análise efetuada, é possível conclui que este tipo de assinatura é mais simples e fácil de identificar do que a do pedómetro.

[1] Tal como referido na secção 3.5.3, o eixo *z* é perpendicular à face do dispositivo, rodando à volta deste eixo quando o utilizar muda de direção

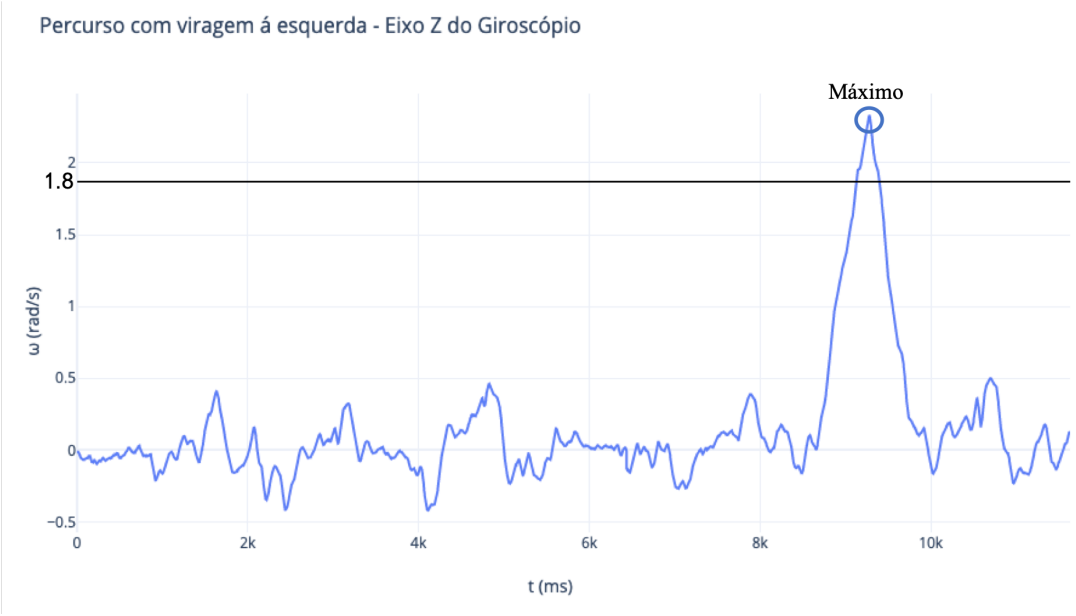


Figura 3.30: Gráfico traçado com as medidas do eixo *z* do sensor Giroscópio, recolhidas durante o percurso com viragem á esquerda.

Figura 3.31: Gráfico traçado com as medidas do eixo *z* do sensor Giroscópio, recolhidas durante o percurso com viragem á direita.

A assinatura duma viragem consiste apenas num máximo ou num mínimo, dependendo do lado para qual o utilizador vire. Estes extremos têm que estar acima de um certo limite, estabelecidos com base nos valores observados nos gráficos, de modo a distinguir a assinatura de outras variações causadas por certos movimentos. Para os máximos, que ocorrem nas viragens á esquerda, foi definido como limite 1.8 rad/s. Para os mínimos, que ocorrem nas viragens á direita, foi definido como limite -1.8 rad/s.

É possível, no entanto, que o utilizador movimente o dispositivo de outra forma antes ou durante

a viragem, causando uma distorção á forma do sinal, ou vire num ângulo diferente de 90º e o pico ser menor que os limites estabelecidos. Estas situações são impossíveis de prever *a priori*, e por isso podem causar problemas na estimativa da posição durante a utilização do sistema.

3.4 Caracterização da trajetória

No espaço do museu, os visitantes poder-se-ão deslocar livremente em todas as direções, com algumas restrições, tal como indica a figura 3.32. No caso vertente, não se consideram as situações em que o utilizador se move para trás por se considerar pouco ilustrativa das possíveis trajetórias num museu.

Considerando o posicionamento nas quadrículas, o utilizador poderá seguir em frente, virar à direita ou virar à esquerda. Quando o utilizador segue em frente apenas é considerada a distância percorrida. É no entanto possível que o utilizador siga em frente, mas efetue um percurso diagonal, que o conduza mais à direita ou mais à esquerda. A informação dada pela bússola é fundamental para distinguir este tipo de situações.

O utilizador também pode fazer viragens, por exemplo quando contorna um expositor do museu. Quando à viragem à direita ou à esquerda, à distância percorrida é associada a informação de mudança de direção, obtida através da deteção de curvas/viragens.

A utilização em conjunto de todos estes elementos permite então determinar o trajeto do utilizador.

É importante reforçar que este não se pode deslocar por cima de mesas ou obstáculos, e sendo assim o seu percurso não poderá passar por esses locais.

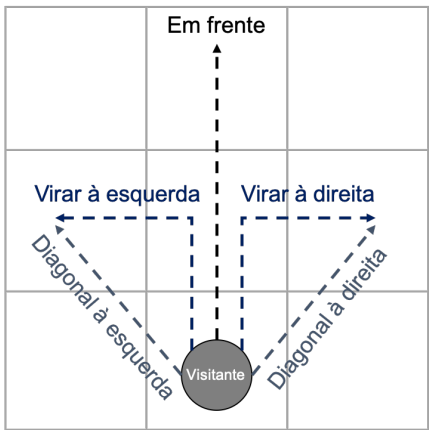


Figura 3.32: Esquema que exemplifica as possíveis direções da trajetória do utilizador no espaço do museu.

3.4.1 Estimativa da distância percorrida

Para calcular a distância percorrida pelo utilizador, o número de passos dados é multiplicado pelo tamanho do passo. A forma mais eficiente de determinar este valor seria contar o número de passos que o utilizador anda ao percorrer uma distância conhecida [10]. No entanto, esta maneira é mais complexa e requer a participação do utilizador no ajuste do tamanho do seu passo, o que torna o

sistema mais complexo e de difícil utilização. A alternativa usada tem por base o facto do comprimento do passo poder ser estimado através do sexo e da altura do utilizador [19]. Desta forma, a única intervenção que se pede aos utilizadores é que, quando usem a aplicação, forneçam informação sobre a sua altura e sexo. Com base nesta informação, o comprimento do passo é calculado com base na equação 3.1:

$$P = \mathsf{A} * alpha \tag{3.1}$$

Sendo que: P é o comprimento do passo, medido em metros; A, a altura do utilizador, medida em metros; e $alpha$, um valor que depende exclusivamente do sexo do utilizador. Assume o valor 0.413 para as mulheres e 0.415 para os homens [19].

Sendo a granularidade da posição definida com uma quadrado, é necessário converter o tamanho da passada em quadrículas percorridas. Este valor depende da direção do movimento do utilizador face ao referencial definido. Para o efeito, usa-se a equação 3.2:

$$N\_q = P / D\_ref \tag{3.2}$$

Sendo que: N_q é o número de quadriculas percorridas pelo utilizador; P é o comprimento do passo; e D_ref, a distância de referência que permite mapear o movimento no espaço físico. Este valor deverá ser igual ao lado da quadricula (0.589 m), caso o utilizador se movimente paralelamente a um eixo, ou à diagonal (0.8457 m), se o movimento do utilizador tiver outra orientação. Neste caso estamos a considerar apenas movimentos realizados a 45º.

3.4.2 Estimativa da posição

Estimativa baseada em deteção de curvas

A deteção de curvas indica quando o utilizador vira numa esquina, numa curva de 90º, sendo particularmente útil para identificar, no espaço do museu, em que corredor este se encontra. Em conjunto com a estimativa da distância percorrida, esta informação permite fornecer uma localização mais precisa.

Este componente permite identificar a que eixo do referencial do sistema de coordenadas o vetor correspondente ao percurso do utilizador é paralelo. Tendo por base esta informação é possível estimar as novas coordenadas do utilizador. Esta informação é cálculada quando um passo é detetado, ou quando se atualiza o valor de *x* ou o de *y* em função da alteração da posição deste vetor em relação aos eixos. É importante também ter em conta que o utilizador é livre de circular por onde quiser, e como tal, o valor da posição pode ser ser incrementado ou decrementado, conforme a direção.

Considere-se o caso em que o utilizador se move paralelamente a um dado eixo, genericamente designado por j, e que no instante t ocorre uma viragem; neste instante, caso o utilizador coloque o pé no chão atualiza-se a posição, incrementando o número de quadriculas percorridas na direção indicada. A partir deste momento, as alterações de posição serão realizadas no outro eixo, genericamente designado por i.

Em qualquer dos casos, a posição do utilizador deve ser atualizada com base na equação 3.3.

$$\begin{cases} Pos(i, t+1) \ = Pos(i, t) \ + DeltaPos. \\ Pos(j, t+1) \ = Pos(j, t) \ . \end{cases} \tag{3.3}$$

Sendo que: $DeltaPos$ assume o valor de N_q caso o utilizador se mova em sentido positivo e assume o valor de $-N\_q$, no caso contrário.

Estimativa baseada na orientação

A deteção do movimento com base na orientação visa dar resposta a outro tipo de padrão de mobilidade. Assim, considere-se o caso em que o visitante se pode aproximar ou afastar das mesas onde se encontram as peças em exposição, sem necessariamente efetuar uma viragem 90º. Durante o seu movimento, o visitante pode tomar um percurso diagonal aos eixos, sendo que a sua posição é atualizada conforme a posição do vetor do percurso. Dado a que a volta pode ser pouco pronunciada, este tipo de movimento poderá ser difícil de identificar com o detector de curvas. A utilização da bússola para estes casos é mais apropriada, sendo bastante útil nas alturas em que o utilizador voltar a um vetor de percurso paralelo a um dos eixos, dado ser possível comparar os valores do azimute medidos em diferentes instantes de tempo.

Considere-se o caso em que o utilizador se move inicialmente paralelo a um dado eixo, e que num instante t ocorre uma leve viragem, sendo o movimento agora diagonal aos dois eixos, genericamente designados por i e j; neste instante, caso o utilizador volte a pousar o pé, a posição é atualizada nas duas coordenadas, incrementado o número de quadrículas percorridas na direção indicada.

Neste caso, a posição do utilizador é atualizada com base na equação 3.4.

$$\begin{cases} Pos(i, t+1) \ = Pos(i, t) \ + DeltaPos(i). \\ Pos(j, t+1) \ = Pos(j, t) \ + DeltaPos(j). \end{cases} \tag{3.4}$$

Seja: N_q o valor de $DeltaPos_i$ caso o utilizador de mova em sentido seja positivo e $-N\_q$ caso contrário.

Esta situação mantém-se até que o utilizador regresse a um percurso paralelo a um dos eixos, seja o anterior ou um perpendicular. Quando esta situação acontece, poder-se-á voltar a usar o mecanismo de deteção de curvas anteriormente descrito.

Erros de posicionamento

A última etapa do processo de posicionamento consiste em identificar situações em que as coordenadas obtidas correspondam a estas posições inválidas e sinalizar estas situações como erro. Conhecendo a planta do museu e a localização e dimensão de todos os expositores e obstáculos, tal como os limites da sala, a posição estimada deve ser comparada com estas informações. Caso ocorra uma situação destas, o sistema deve corrigir a estimativa, a partir da última posição correta e somando o deslocamento ajustado.

3.5 Arquitetura da solução

3.5.1 Visão geral – Localização

Durante a visita ao museu o elemento central de informação ao visitante é um telemóvel, através do qual serão apresentados os conteúdos relativos à sua localização. Desta forma, o elemento central de localização é o telemóvel usado.

A informação de localização é fundamental para o fornecimento de informação ao visitante relativo à peça/espaço que este se encontra a ver. Uma vez técnicas usadas para proceder à localização interior estão sujeitas a diversos erros, optou-se por conjugar duas técnicas, como forma de tentar corrigir o erro e melhorar a experiência do visitante. Assim sendo, a localização dos utilizadores é feita recorrendo à técnica de ***Dead Reckoning*** (ver secção 2.1.4) e à técnica de ***Fingerprinting*** (ver secção 2.1.2). Ambas recorrem aos sensores presentes nos *smartphones* para obter a posição do utilizador.

Na primeira técnica, a posição é obtida calculando a distância percorrida e determinando a direção do movimento, isto é, identificando a trajetória seguida pelo visitante. Tal como descrito anteriormente, o *Dead Reckoning* acumula erro de precisão ao longo do tempo. A técnica de *Fingerprinting* é usada para corrigir esse erro, tendo por base medidas dos sensores e do RSSI dos *beacons* existentes no museu.

3.5.2 Visão geral – Arquitetura

O sistema desenhado tem por modelo a arquitetura Cliente-Servidor, tal como ilustrado na figura 3.33.

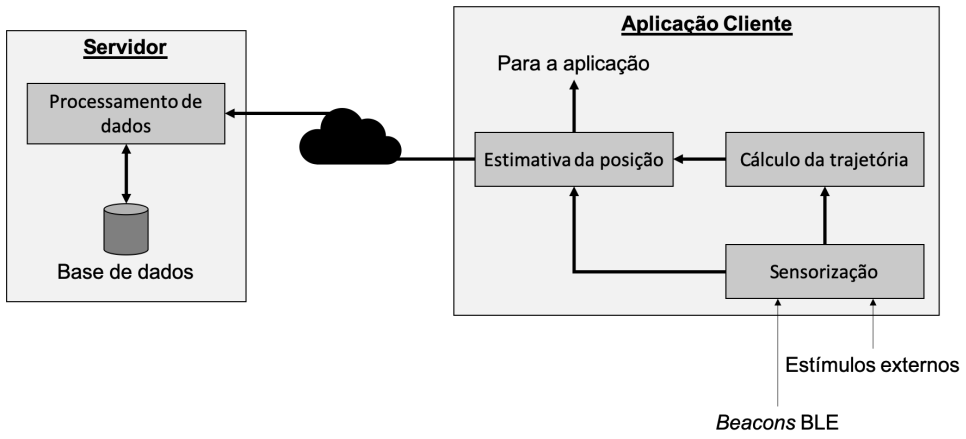


Figura 3.33: Esquema da arquitetura geral do sistema, com os dois elementos principais, o Cliente e o Servidor.

O Cliente é uma aplicação móvel que faz a recolha das medidas dos sensores do dispositivo móvel e dos *beacons* BLE para determinar a posição do utilizador. A informação recolhida é usada, localmente, para estimar a posição com base na trajetória. Tal como se ilustra na figura, existem três módulos principais: o módulo de Sensorização que é responsável pela aquisição e processamento de dados dos sensores; o módulo de Cálculo da trajetória que é responsável por calcular o percurso seguido pelo

visitante; e o módulo de Estimativa da posição que é responsável por indicar a localização do visitante em cada instante com base na trajetória seguida

O Cliente comunica com o Servidor para enviar a sua estimativa de posição e os dados medidos dos sensores, de forma a obter deste, uma correção à sua posição. Para o efeito de correção de posição, o servidor regista todas as medidas recebidos dos diversos clientes e usa a informação pré-recolhida para fornecer ao utilizar a melhor estimativa de posição corrente.

3.5.3 Aplicação Cliente – Módulo de Sensorização

A figura 3.34 ilustra a arquitetura do cliente, focada no módulo de Sensorização. Na base de tudo estão os sensores, que reagem aos estímulos externos, medindo aspetos específicos, tais como aceleração, mudança de orientação e orientação magnética, e o módulo interno de comunicação BLE que deteta os sinais enviados pelos *beacons*. O sub módulo Leitura dos sensores do telemóvel é responsável por ler e entregar os valores medidos por cada sensor ao módulos seguintes, onde os mesmos serão analisados. Já o sub módulo Leitura dos *beacons* BLE, deteta os *beacons* próximos, medindo o valor RSSI de cada um, sendo esta informação enviada para o módulo Estimativa da posição, antes de ser reencaminhada para o servidor central.

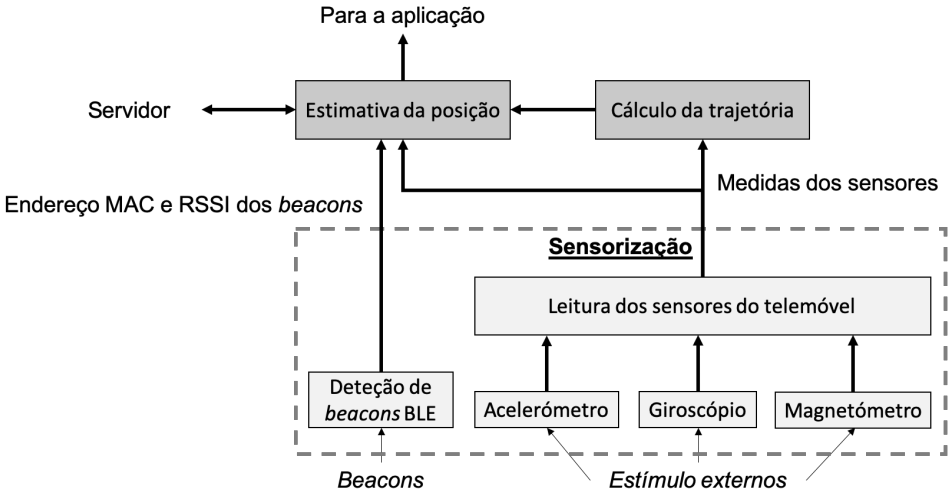


Figura 3.34: Esquema da arquitetura do cliente, com foco em especial no módulo da Sensorização.

Descrição dos sensores dos *smartphones*

Os sensores disponíveis tem múltiplas utilidades, em especial para a interação com o utilizador, providenciando dados em bruto (*raw*) com alta precisão e exatidão ás aplicações [20]. No caso desta tese, o uso de sensores tem como finalidade a determinação da localização do utilizador no museu. Para o efeito, os sensores de movimento e posição são os mais adequados para determinar a posição, dado que o museu é um espaço interior e por isso os sensores de ambiente não têm grande utilidade aqui. São considerados apenas três: o acelerómetro, o giroscópio e o magnetómetro. Cada um destes sensores tem várias funções diferentes e complementares que serão descritas a seguir.

Sensores de posição – Os sensores de posição permitem determinar a localização física do dispositivo [21]. Nesta aplicação, é utilizado apenas um sensor desta categoria, o magnetómetro. O magnetómetro é um sensor de efeito *Hall*, em miniatura, que mede a força do campo geomagnético ao longo dos eixos *x*, *y* e *z* em μT (microTeslas)[22]. À saída é produzida uma voltagem proporcional á força e polaridade do campo magnético ao longo de cada eixo. Esta voltagem é então convertida num sinal digital que representa a intensidade do campo magnético .

Com base nos dados recolhidos por este sensor é possível determinar a orientação geográfica do dispositivo, tal como se define numa bússola. Esta informação vai ser utilizada para calcular a orientação do utilizador, independentemente dos movimentos que este faça com o telemóvel.

O magnetómetro é sensor muito suscetível a interferências de equipamentos metálicos e elétricos. Estas interferências podem introduzir um erro significativo na estimativa da direção. Por esta razão, este sensor deve ser utilizado em conjunto com outros, tal como o giroscópio, de modo a diminuir a influência deste tipo de equipamentos nos resultados finais.

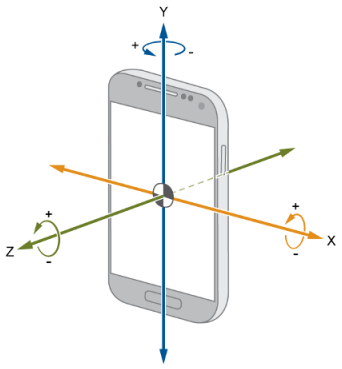


Figura 3.35: Esquema de um *smartphone* com os três eixos do sistema de coordenadas local assinalados [23] .

Sensores de movimento – Os sensores de movimento permitem monitorizar o movimento do dispositivo, normalmente causado pela acção direta do utilizador sobre o telemóvel, seja esta intencional ou não [24]. Atividades, como por exemplo caminhar ou virar numa esquina, podem ser detetadas com estes sensores. No entanto, não é possível determinar a localização do utilizador usando apenas este tipo de sensores, sendo necessário complementar a informação recebida com a que se obtém de outros sensores de posição, nomeadamente o magnetómetro.

Nesta aplicação, são utilizados dois sensores deste tipo: o acelerómetro e o giroscópio.

A posição do dispositivo no espaço tridimensional varia ao longo do tempo, caso este esteja em movimento. O acelerómetro mede a força da aceleração ao longo dos três eixos, *x*, *y* e *z* em m/s^2 [24]. Este tipo de dispositivo é essencialmente constituído por um *microchip* que contém uma parte móvel que gera impulsos elétricos diferentes, conforme a posição do dispositivo [**?**]. O esquema do sistema de coordenadas deste sensor encontra-se ilustrado na figura 3.36. Normalmente, este sensor é utilizado para detetar certos movimentos ou gestos, durante a utilização de aplicações ou jogos [25]. Nesta aplicação, tem duas funções importantes: a primeira é permitir a implementação de um pedómetro,

através da identificação da assinatura específica de um passo nos seus dados; a outra função é, em conjunto com o magnetómetro, estimar a orientação do utilizador.

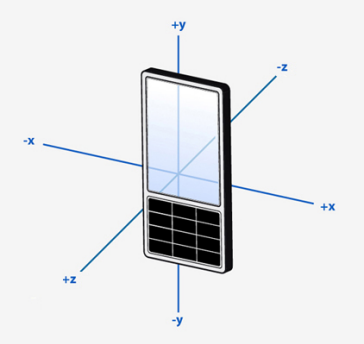


Figura 3.36: Esquema de um *smartphone* com os três eixos do sistemas de coordenadas local assinalados [25] .

O giroscópio é o sensor que fornece a orientação do telemóvel, medindo a velocidade (*rate*) de rotação ao redor dos três eixos em rad/s^2 [24]. Este tipo de sensor é muito utilizado em aplicações, especialmente naquelas que envolvem realidade aumentada ou virtual. O esquema do sistema de coordenadas deste sensor encontra-se ilustrado na figura 3.37. Nesta aplicação o giroscópio terá duas funções: a primeira, será ajudar a descartar falsos positivos no pedómetro, que sejam causados por movimentos bruscos do dispositivo. Estes movimentos causam uma grande variação nos dados deste sensor, algo que não ocorre quando o utilizador está apenas a caminhar. A segunda, está relacionada com a direção do percurso, usualmente obtida através do magnetómetro e acelerometro. O giroscópio é usado para corrigir erros causados pela presença de equipamentos metálicos e/ou elétricos nas proximidade.

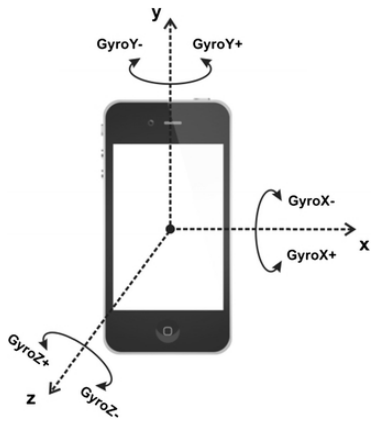


Figura 3.37: *Smartphone* com as definições do sistema de coordenadas local e as direções de rotação do giroscópio [26].

Sensores externos: Beacons BLE

Um *beacon* é um pequeno dispositivo, da classe *Bluetooth Low Energy* [27] que emite sinais rádio que podem ser recebidos pelos dispositivos em redor com interface BLE, neste caso os *smartphones* em

[2] rad/s – radiano por segundo

redor [28]. Cada um destes dispositivos transporta um identificador único universal que pode ser usado para o reconhecer [29]. A tecnologia BLE utiliza muito pouca energia elétrica e envia uma pequena quantidade de dados, sendo que a sua bateria pode durar 2-3 anos [28] [3]. O alcance do sinal é bastante grande, chegando a atingir distâncias de cerca de 70 m [28]. Porém, esta distância pode ser encurtada por causa dos problemas da propagação em ambientes fechados (ver secção 2.1.2).

Os *beacons* BLE são normalmente colocados em paredes ou tectos [28], de forma a permitir a maior cobertura sem obstruções. No caso presente, optou-se por proceder à sua colocação nas paredes e num pilar, em zonas altas. Foram usados 4 *beacons*: um junto à parede de entrada; um segundo no corredor da direita, no poste; um terceiro na parede oposta à entrada, e um quarto na parede do lado esquerdo, mais próximo da zona de entrada.

3.5.4 Aplicação Cliente – Módulo de Cálculo da trajetória

O módulo de Cálculo da trajetória é responsável por processar e interpretar os dados recolhidos pelos três sensores. As funções estão repartidas em três sub módulos, que serão descritos nesta secção: o Pedómetro, que analisa as medidas do acelerómetro para encontrar a assinatura específica do passo; o Detetor de curvas/viragens, que analisa as medidas do giroscópio para detetar a assinatura de uma viragem de 90º por parte do utilizador; e a Bússola, que utiliza as medidas de todos os sensores para calcular a orientação do telemóvel, designado por azimute. A Estimativa da posição é então notificada individualmente por cada um destes sub módulos, sendo enviadas as informações necessárias para a atualização da posição do utilizador. A figura 3.38 ilustra a arquitetura deste módulo, no contexto da aplicação cliente.

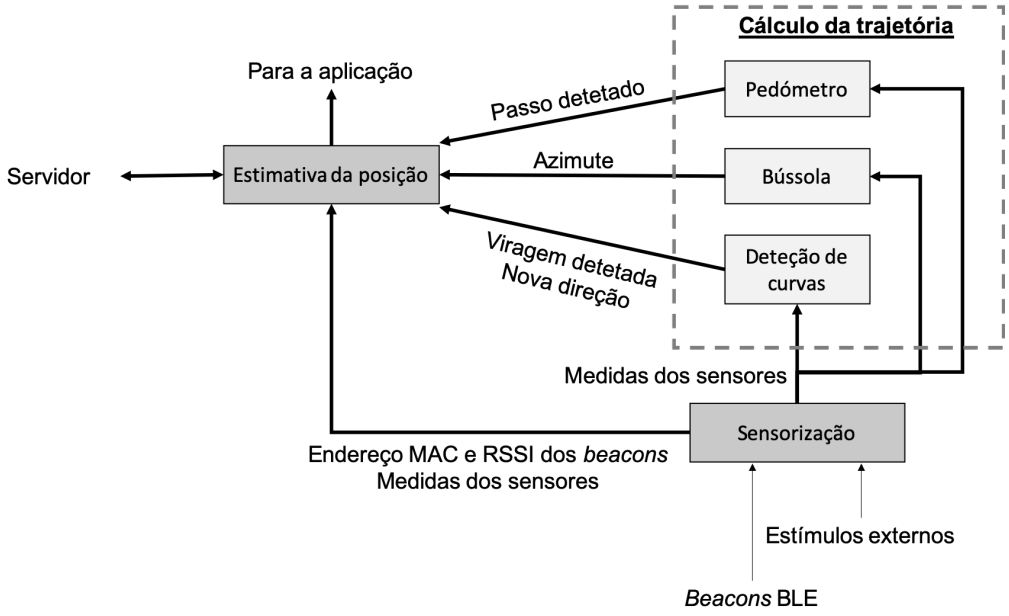


Figura 3.38: Esquema da arquitetura do cliente, com ênfase em especial no módulo do Cálculo da trajetória.

[3] a Estimote indica que os seus *beacons* podem duram mais que 5 anos em modo de baixo consumo

Pedómetro

O objetivo deste módulo é identificar a assinatura específica da passada do utilizador nas medidas do eixo *z* do sensor acelerómetro. Esta assinatura, descoberta durante a fase de análise do movimento humano descrita anteriormente, é constituída por um máximo seguido de um mínimo que devem cumprir certos requisitos, tal como o gráfico da figura 3.39 demonstra. Esta forma corresponde ao balançar do braço que segura o telemóvel, que faz variar a sua posição em relação a o eixo *z*, que como explicado acima, é perpendicular ao ecrã, algo que é ilustrado na figura 3.40.

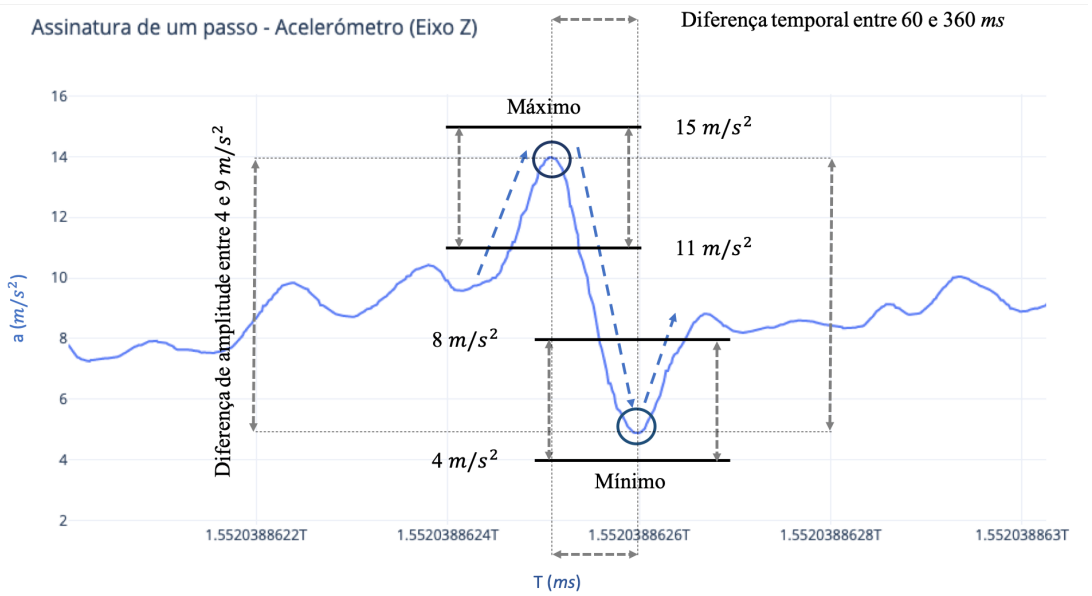


Figura 3.39: Gráfico que exemplifica a assinatura de um passo nas medidas do eixo *z* do acelerómetro, retirado de um dos percursos analisados.

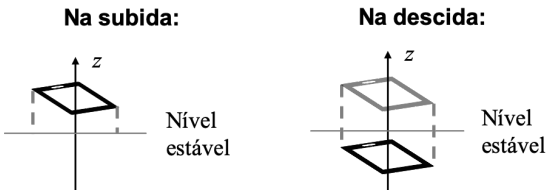


Figura 3.40: Demonstração do movimento do dispositivo durante a passada, enquanto segurado pelo utilizador.

A ratificação do passo é um processo complementar á identificação da assinatura, servindo para descartar falsos positivos, causados por certos movimentos do *smartphone*, que criam variações similares no sinal. Quando um passo é detetado, o módulo Estimativa da posição é notificado, para que a nova posição do utilizador seja determinada. Esta é uma parte muito importante do sistema, que fornece a distância percorrida pelo utilizador ao longo do tempo. No entanto, é necessário associar a direção á distância percorrida, para que possa ser definida uma posição específica, sendo esse o papel dos outros dois módulos.

Algoritmo: A implementação do pedómetro é detalhada na secção 4.3.1. O fluxograma da figura 3.41 exemplifica a lógica usada para a deteção de passos.

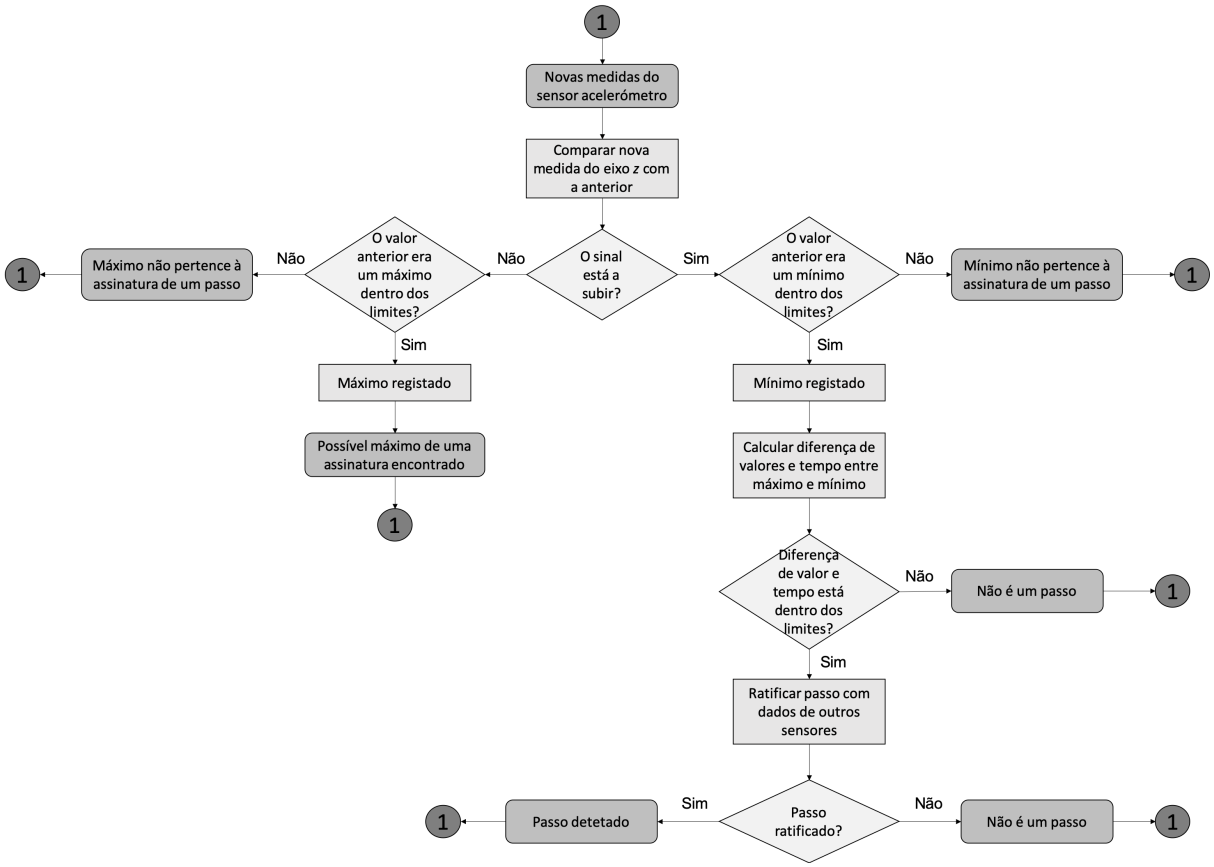


Figura 3.41: Fluxograma da lógica do pedómetro.

Deteção de curvas/viragens

A deteção de curvas, em conjunto com a bússola, fazem parte da determinação da orientação do percurso do utilizador, tendo, no entanto, papéis distintos. Neste sub módulo, o objetivo é identificar uma viragem de 90º para um dos lados, esquerda ou direita. Tal como aconteceu com o pedómetro, o desenvolvimento desta parte do sistema foi baseado na análise descrita acima. O objetivo aqui é identificar a assinatura da viragem no sinal do giroscópio, bem como a sua direção. Dependendo da direção da curva, esta assinatura pode constituir um máximo ou mínimo, que tem de estar dentro dos limites definidos.

Algoritmo: A implementação da deteção de curvas/viragens é descrita em detalhe na secção 4.3.2. O fluxograma da figura 3.42 exemplifica a lógica usada para a deteção de viragens, baseada no código implementado no sistema.

A mesma lógica para identificação de extremos do pedómetro é usada aqui, no entanto, apenas para detetar um máximo ou um mínimo. Uma técnica de bloqueio permite isolar este extremo, sendo

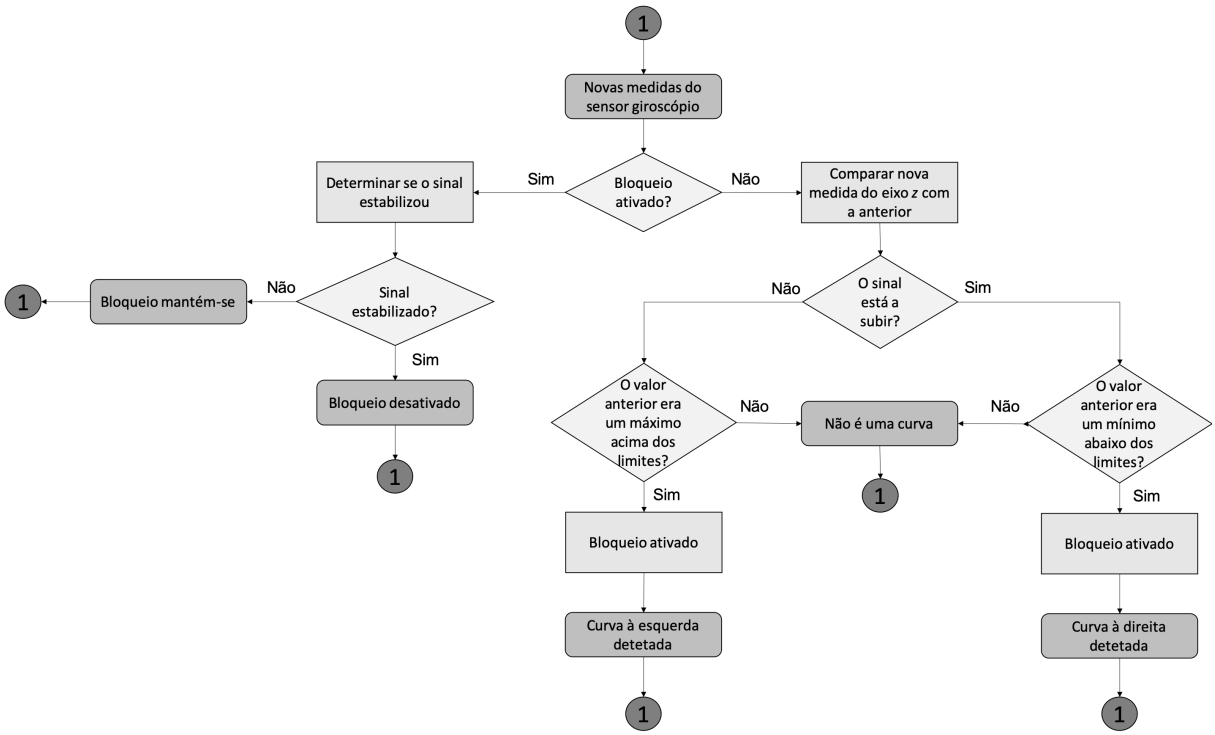


Figura 3.42: Fluxograma da lógica do detetor de curvas/viragens.

que apenas quando o sinal estabilizar é que será possível efetuar estas verificações. No final, o módulo Estimativa da posição é notificado da viragem detetada e da respetiva direção.

Bússola

A bússola indica a orientação geográfica do utilizador, uma informação necessária não só para determinar a direção do percurso, mas também para identificar para que lado este está virado em qualquer momento. A implementação usada neste sistema foi baseada em [30], que descreve como fazer a fusão de sensores para determinar a orientação final. Em suma, a orientação é calculada três vezes: com as medidas do magnetómetro e do acelerómetro; apenas com os dados do giroscópio; e finalmente com a fusão das duas estimativas anteriores. O cálculo com o giroscópio é bastante complexo, dado que este é um sensor local e a orientação é global, sendo necessária a transformação das medidas. A nova orientação, resultante desta fusão, é então enviada para o módulo da Estimativa da posição, sendo utilizado também na correção da localização, através do *Fingerprinting*.

Algoritmo: A implementação da bússola é detalhada na secção 4.3.3. O fluxograma da figura 3.43 exemplifica a lógica para a bússola, baseada no código implementado no sistema, a um nível de abstração mais alto, dada a complexidade e tamanho dos algoritmos.

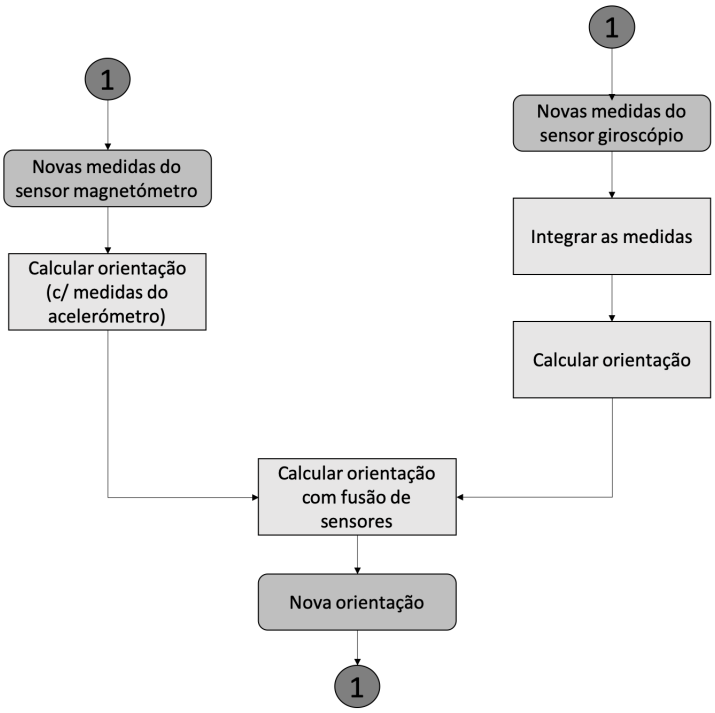


Figura 3.43: Fluxograma da lógica da bússola com fusão de sensores.

3.5.5 Aplicação Cliente – Módulo de Estimativa da posição

Com base nas informações recebidas do Cálculo da trajetória e da Sensorização, é neste módulo que a posição do utilizador é determinada. Enquanto o utilizador caminha, a estimativa é feita a partir da distância percorrida e da respetiva direção, no sub módulo Posicionamento baseado na trajetória. Quando o utilizador pára, é feito um pedido ao servidor para determinar a posição atual, com as medidas recolhidas, sendo que o envio e receção é feito pelo sub módulo Posicionamento baseado em *Fingerprinting*. A estimativa vinda do servidor, determinada com base no modelo de *machine learning* criado, é enviado para a Correção da posição, onde vai ser usada para repor a posição. É este módulo que é responsável por entregar as coordenadas da localização á aplicação. A figura 3.44 ilustra a arquitetura do cliente, como foco em especial nesta parte.

Posicionamento baseado na trajetória

As informações da Deteção de curvas e da Bússola são usadas de forma diferente mas complementar. Quando é notificada a deteção de uma viragem, são determinados os parâmetros necessários para a próxima atualização da posição, aquando do próximo passo, altura em que também é guardada a orientação da curva, para referência futura. Este parâmetros indicam a direção e localização do vetor do percurso em relação aos eixos do referencial. Ao receber uma notificação de um novo passo, é verificado se o utilizador mudou de direção desde o último, sendo que, em caso afirmativo, é necessário determinar qual a posição do novo vetor, ou seja, qual a situação atual de orientação. É aqui que entra a orientação da curva. Dependendo se o percurso for paralelo ou diagonal aos eixos, a atualização das coordenadas é feita de forma diferente.

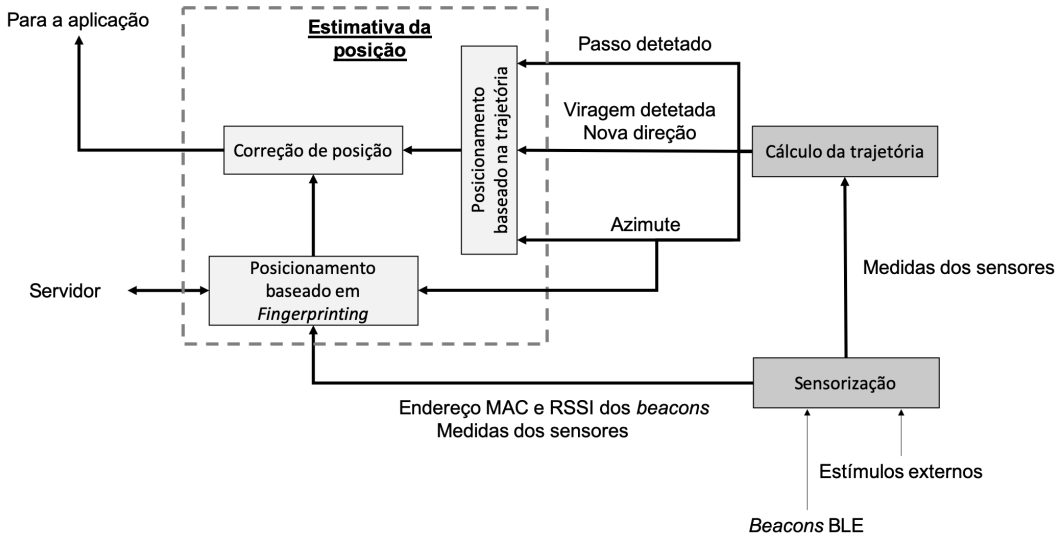


Figura 3.44: Esquema da arquitetura do cliente, descobrindo o módulo Estimativa da posição.

Algoritmo: O fluxograma da figura 3.45 ilustra a lógica usada neste módulo.

Esta lógica permite considerar todas as direções possíveis que um visitante pode tomar num campo de 180º. As direções contrárias não são tomadas em conta, dada a complexidade adicional que requeriam. No entanto, é possível que uma futura versão do sistema possa garantir isto. A posição calculada é então enviada para o módulo da Correção da posição, antes de ser reencaminhada para a aplicação.

Posicionamento baseado em *Fingerprinting*

Uma vez que o processamento dos dados é feito num servidor remoto, este sub módulo tem como objetivo recolher os valores RSSI de cada uma dos *beacons*, as medidas dos sensores e a orientação, de modo a construir a *fingerprint*, que será utilizada pelo servidor no processo de estimativa da posição. A informação colocada em cada *fingerprint* é a seguinte:

```
Tempo; RSSI Beacon 1, ..., Beacon n; valores x,y,z do magnetómetro; azimute
```

Nos casos em que por alguma razão a posição não possa ser estimada com base na trajetória, tal como acontece com as crianças e adolescentes, o *Fingerprinting* é a única forma de determinar a localização, e por esta razão a informação colocada na *fingerprint* terá apenas os valores RSSI dos *beacons*:

```
Tempo; RSSI Beacon 1, ..., Beacon n
```

A resposta é enviada de volta, sendo então reencaminhada para o módulo da Correção da posição, que irá então proceder á emenda dos valores das coordenadas.

Algoritmo: O fluxograma da figura 3.46 ilustra a lógica usada no algoritmo de *fringerprinting*.

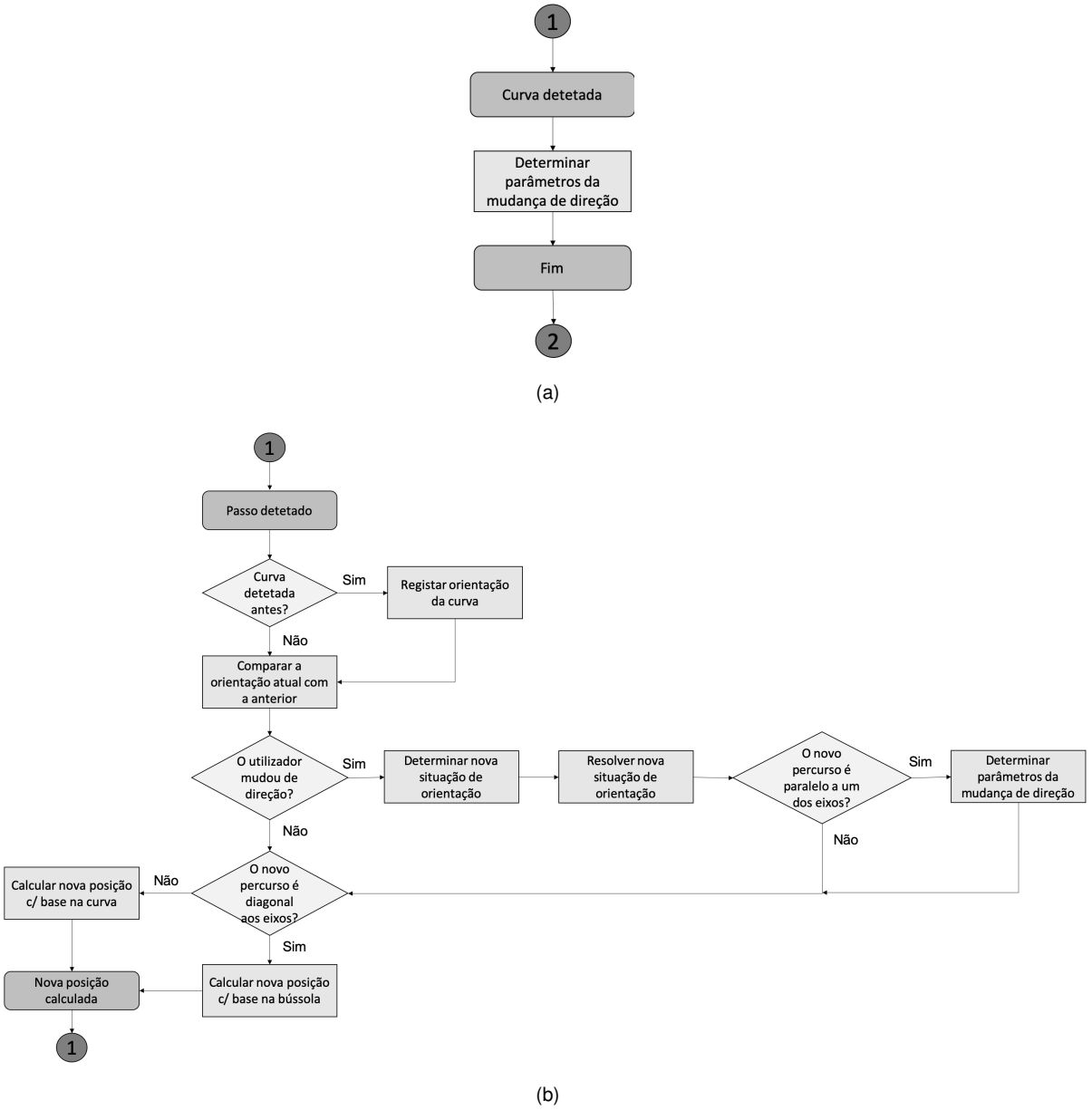


Figura 3.45: Fluxograma da lógica usada no módulo Posicionamento baseado na trajetória.

Correção da posição

Enquanto o utilizador circula pelo museu, a sua posição é exclusivamente determinada através da trajetória tomada, existindo ao longo do tempo uma acumulação do erro causado pelos falsos positivos e negativos. Quando o utilizador pára, ou seja, quando não são detetados passos durante uma certa quantidade de tempo, chega a oportunidade de corrigir a posição, pedindo uma estimativa com base em *Fingerprinting*. No caso de o utilizador voltar ao percurso antes de ser recebida uma resposta, a sua deslocação é guardada. Se a comunicação falhar e não for recebida nenhuma resposta, então esse deslocamento é adicionado á posição anterior. A estimativa recebida é comparada com aquela da trajetória, para determinar a distância entre as duas posições. Se esta distância for elevada, por exemplo, as duas posições corresponderem a sítios opostos do museu, então a estimativa recebida está incorreta e a posição não é corrigida. Para garantir a validade do *Fingerprinting*, no ponto inicial

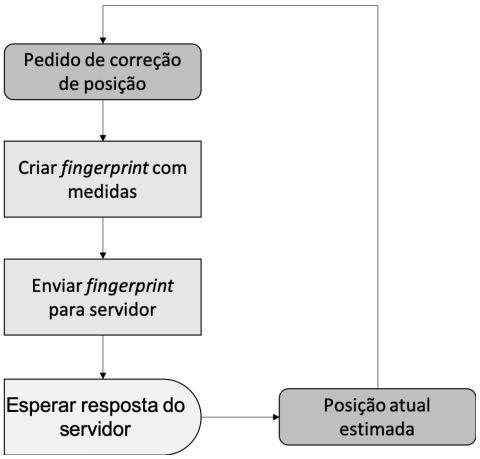


Figura 3.46: Fluxograma da lógica usada no módulo Posicionamento baseado em *Fingerprinting*.

(0,0), é pedida uma estimativa ao Servidor, sendo que se a mesma estiver incorreta então a correção da posição não acontecerá.

Algoritmo: O fluxograma da figura 3.47 exemplifica a lógica para do algoritmo para a correção da posição.

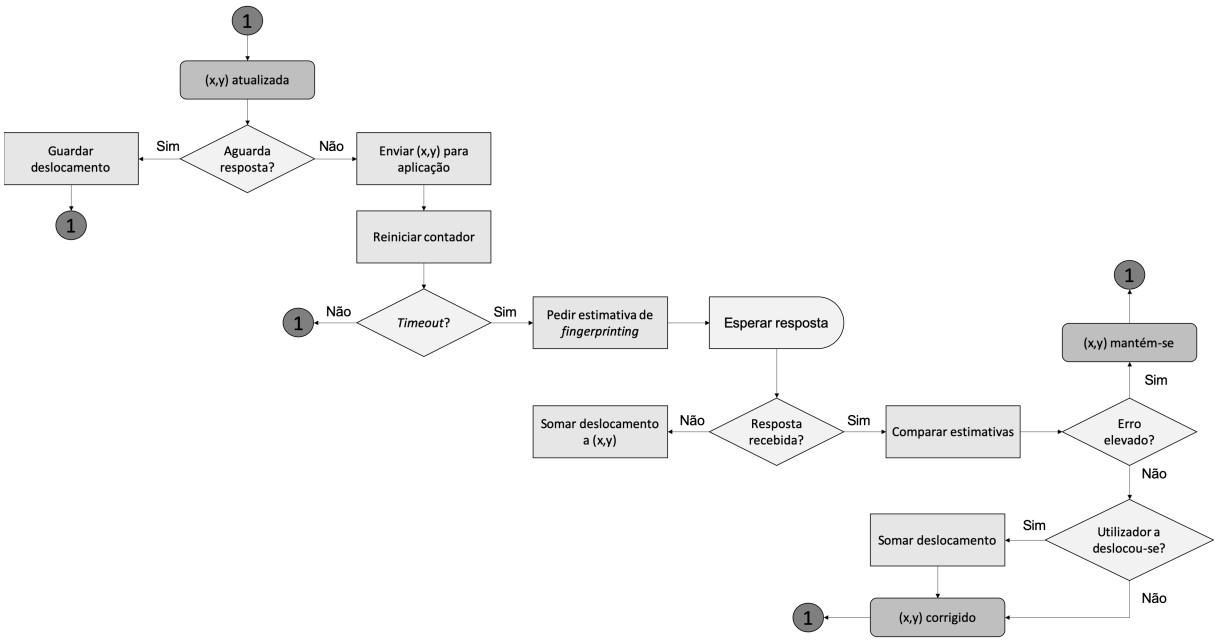


Figura 3.47: Fluxograma da lógica usada no módulo Correção da posição.

3.5.6 Servidor

O Servidor remoto é responsável pela criação e utilização da base de dados de *fingerprints*, recolhidas numa fase inicial. Através da comparação destas medidas com aquelas observadas pelo Cliente, é possível estimar a sua localização efetiva, permitindo ajustar a posição do utilizador. É constituído por

dois sub módulos, Treino e Previsão da posição, que fazem parte do módulo Processamento de dados. A figura 3.48 ilustra a arquitetura do Servidor.

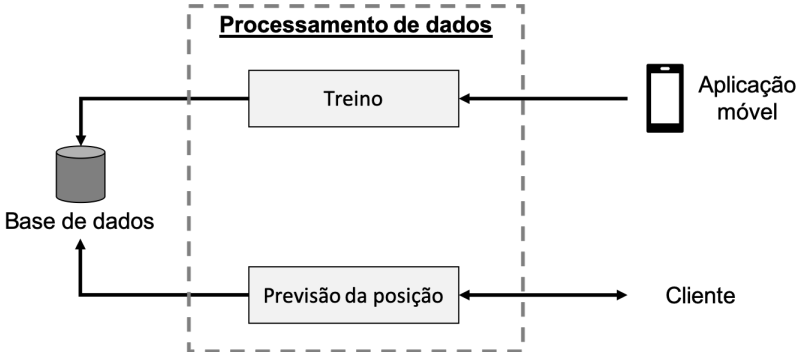


Figura 3.48: Esquema da arquitetura do servidor.

Treino

O treino envolve uma recolha inicial de dados, que consiste numa pessoa a percorrer o espaço, segurando um *smartphone* a correr uma aplicação que guarda para todas as posições, considerando as possíveis orientações, um conjunto de medidas associado, nomeadamente, os valores registados pelos vários sensores e os valores do RSSI medidos para cada um dos *beacons*, de modo a poder construir uma assinatura única. Os dados são então processados para criar um intervalo de valores possíveis para cada *fingerprint*, com uma certa confiança, que serão então armazenadas na base de dados.

Previsão da posição

Durante a utilização do sistema, o Cliente inquire sobre a sua posição, enviando a *fingerprint* registada no momento, podendo esta ser simples, contendo apenas informação sobre o RSSI dos *beacons*, ou complexa, contendo também as medidas do sensor magnetómetro e o valor do azimute. Estes valores são comparados com as gamas de valores armazenadas na base de dados, de modo a prever qual a localização do utilizador. Mesmo que ocorra sobreposição entre certos valores medidos em locais próximos, os outros campos devem permitir fazer a distinção entre estes. O resultado final pode ser um de três opções: a posição efetiva; a vizinhança de um ponto; ou erro, quando não é possível determinar uma localização válida.

Algoritmo: O fluxograma da figura 3.43 exemplifica a lógica para a previsão da posição.

3.6 Síntese final

Este capítulo começa com uma caracterização do espaço do museu, onde o sistema será implantado, estabelecendo um sistema de coordenadas para a localização no espaço. De seguida, é feita uma caracterização a diferentes tipos de movimentos, com base na análise realizada, para criar *software* capaz de identificar as assinaturas destes, nomeadamente o passo e as viragens, e assim poder calcular

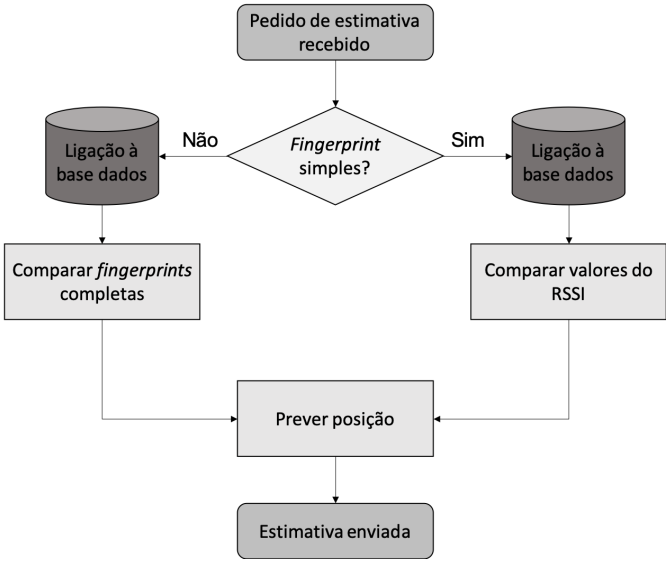


Figura 3.49: Fluxograma da lógica da previsão da posição.

a trajetória do utilizador. Foram também aqui estudados certas movimentações passíveis de ocorreram durante a visita, em especial durante a observação detalhada de certas peças, e que podem causar falsos positivos na deteção de passos. Esta trajetória corresponde a diversos caminhos, com diferentes direções e distâncias, que o utilizador pode seguir, os quais foram também caracterizados. A forma de atualização das coordenadas está dependente das últimas orientações.

Por último, é apresentada a arquitetura do sistema, que inclui dois elementos principais, o Cliente e o Servidor. O Cliente é responsável por recolher as medidas dos variados sensores para determinar a deslocação do utilizador, bem como a leitura dos *beacons* BLE, que em conjunto com outros dados, permitem obter um estimativa mais precisa para corrigir a posição calculada anteriormente. É constituído por três módulos: Sensorização; Cálculo da trajetória e Estimativa da posição. O Servidor implementa a técnica de *Fingerprinting*, utilizando as medidas recolhidas no espaço durante a fase inicial de treino, para prever qual a localização do visitante, comparando estas com as observadas pelo Cliente. É constituído por apenas um módulo, o Processamento de dados.

Capítulo 4

Implementação do sistema

Neste capítulo é detalhada a implementação dos vários elementos deste sistema, introduzidos anteriormente na arquitetura. Na primeira parte, secção 4.1, é apresentada a visão geral do sistema; seguidamente é apresentada a descrição dos vários módulos da aplicação Cliente. Esta descrição inicia-se pelo módulo de sensorização, descrito na secção 4.2, ao qual se segue o cálculo da trajetória, na secção 4.3, e por fim, a estimativa da posição, na secção 4.4. O capítulo termina com a descrição do sistema do processamento de dados básico, apresentado na na secção 4.5, que foi implementado no contexto do servidor.

4.1 Visão geral do sistema

4.1.1 Requisitos e opções de implementação

A aplicação a desenvolver deverá ser descarregada para os *smartphones* dos visitantes, antes do inicio da visita. Apenas está disponível para o sistema operativo *Android*, tendo sido desenhada para versões com nível da API de 27 [1]. Todavia, o sistema desenvolvido funciona em versões a partir do nível 21 [2].

Os telemóveis devem ser compatíveis com as tecnologias de comunicação sem fios *Bluetooth* e BLE e ter incluídos os sensores mencionados anteriormente.

A escolha de desenvolver a aplicação móvel para o sistema operativo *Android* foi motivada por duas razões. Primeiro, o domínio deste sistema no contexto do mercado móvel global e, segundo, a experiência do aluno neste ambiente de desenvolvimento. Em termos de linguagem de programação, a escolha recaiu sobre a linguagem Java, por ser de muito conhecida e utilizada.

Pretendendo-se com o protótipo testar a viabilidade de utilização deste sistema para efeitos de navegação e posicionamento no espaço do museu, por isso, considerou-se pouco relevante considerar as situações em que o utilizador não possa correr a aplicação, seja por não ter dispositivo próprio consigo, ou pelo seu *smartphone* não suportar a aplicação. Este aspeto só será importante equacionar numa fase posterior, caso o sistema implementado ofereça as garantias de localização necessárias à

[1] Android 8.0 ou Oreo
[2] Android 5.0 ou Lollipop

sua usabilidade.

A escolha dos dispositivos *Android* condiciona, de certa forma, as opções em termos de localização por BLE. O museu está equipado com *beacons* da Estimote, que disponibiliza um ambiente de desenvolvimento para efeitos de localização, com serviços acessíveis através da nuvem. Todavia, algumas das técnicas que são disponibilizadas apenas funcionam no sistema operativo *iOS*. Desta forma, foi escolhida uma solução mais simples, que fizesse apenas uma varrimento dos *beacons* próximos

Em termos de portabilidade para outros sistemas operativos, nomeadamente o sistema operativo *iOS* da *Apple*, existe uma ferramenta *open source* da *Google*, J2ObjC, que traduz código Java para Objective-C [31]. Apenas o código fonte é utilizado como parte da nova aplicação, sendo que a interface de utilizador requererá um esforço adicional para que seja portável.

4.1.2 Arquitetura de *Software*

Para uma melhor compreensão da arquitetura da aplicação desenvolvida, a figura 4.1 apresenta o respetivo diagrama de classes simples, existindo no anexo B.1 uma versão mais completa, que inclui os atributos e métodos.

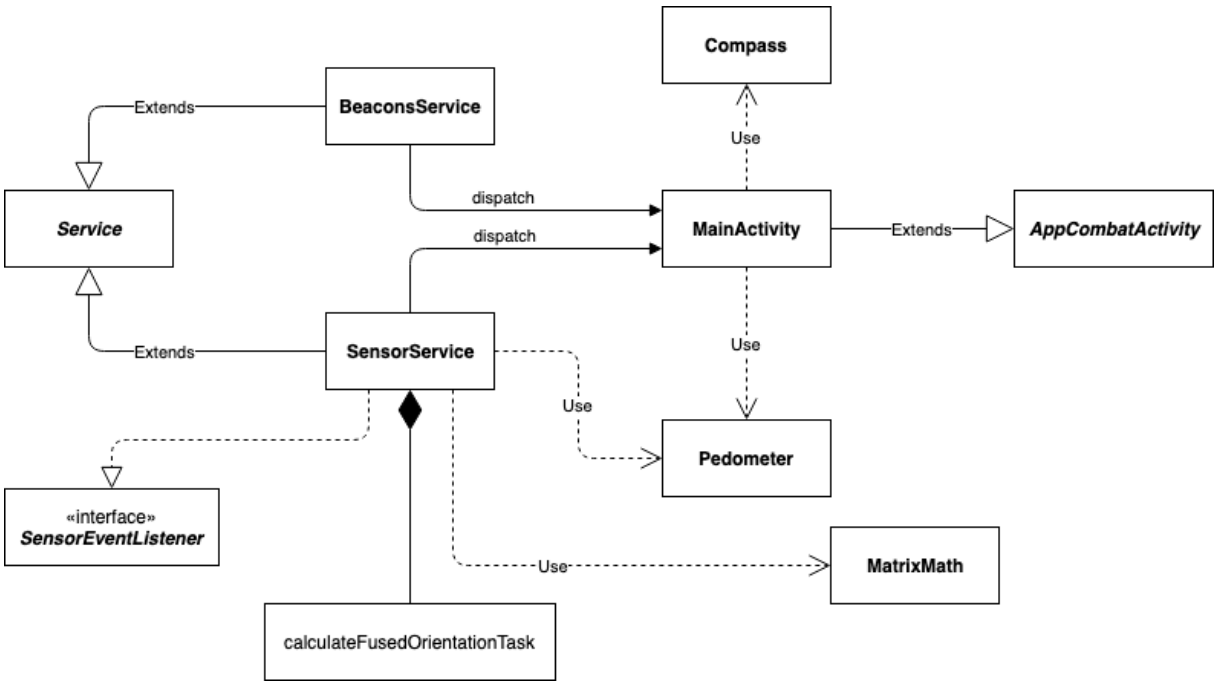


Figura 4.1: Diagrama de classes. Aquelas, cujo nome está em itálico, pertencem a bibliotecas do Android.

A aplicação foi construída utilizando dois componentes: *Activity* e *Service*. A *Activity* fornece uma interface de utilizador, sendo que cada *activity* tem uma janela associada onde é exibida essa interface [32], definida num ficheiro de *layout* em XML. O *Service* permite realizar operações em *background*, sem fornecer uma interface [33].

O *service* `SensorService` é onde está implementada a parte do módulo de Sensorização, relacionada com recolha de dados dos sensores e o módulo de Cálculo da trajetória, do qual fazem parte

o Pedómetro, a Deteção de curvas/viragens e a Bússola. Este inclui um `TimerTask`, denominada `calculateFusedOrientationTask` [30], que executa a fusão de dados dos sensores da Bússola.

A classe `Pedometer` e `MatrixMath` contêm métodos auxiliares utilizados no Pedómetro e na Bússola, respetivamente.

O *service* `BeaconsService` implementa a leitura dos *beacons* BLE, que também faz parte da Sensorização. Estes duas classes derivam da classe `Service` da biblioteca `android.app`.

A *activity* `MainActivity`, que deriva da classe `AppCombatActivity` da mesma biblioteca, relacionada com a interface da aplicação, implementa o módulo de Estimativa da posição, recebendo notificações dos *services* através dos métodos definidos na interface `Callbacks`. São utilizados aqui as funções da classe `Pedometer`, relacionados com o cálculo do incremento da posição. Para identificar a orientação/direção do utilizador, são utilizados os métodos auxiliares da classe `Compass`.

Por uma questão de simplicidade, estão omissas no diagrama as classes e interfaces relacionadas com a comunicação entre os *services* e a `MainActivity`, adaptadas de [34].

4.2 Cliente - Módulo de Sensorização

Esta secção tem por objetivo descrever a implementação do módulo de sensorização. Inicialmente, é descrita a Leitura dos sensores do telemóvel, seguidamente da Deteção de *beacons* BLE.

4.2.1 Leitura dos sensores do telemóvel

A *framework* de sensores do Android fornece o acesso a diferentes tipos de sensores, quer estes estejam implementados em *software* ou em *hardware* [20]. Os sensores a utilizar nesta *app* são todos físicos, embutidos nos telemóveis [20].

Neste sistema todo o processo de leitura e distribuição dos dados destes sensores é feito no `SensorService`, que implementa a interface `SensorEventListener`, usada para receber notificações de novas medidas [35].

Nesta secção é descrita a inicialização, monitorização dos eventos e respetiva obtenção dos dados, que serão então distribuídos pelos restantes módulos.

Inicialização

A inicialização dos sensores tem por objetivo dar às aplicações acesso às medidas.

Para este efeito, é necessário obter a referência do serviço do sensor, criando uma instância da classe `SensorManager`, o que se concretiza quando se chama o método `getSystemService()` [20], tal como se demonstra a seguir:

```
1 private SensorManager sensorManager = (SensorManager) getSystemService(Context.
    SENSOR_SERVICE);
```

É através deste objeto que é possível selecionar e aceder aos dados dos sensores desejados. Para isso, é criada uma instância da classe `Sensor` invocando o método `getDefaultSensor()` e passando como argumento a constante que corresponde ao tipo de sensor pretendido [20]. Por exemplo, para o acelerómetro a constante a usar seria `TYPE_ACCELEROMETER` [20]. No caso desta aplicação, são utilizados três sensores: o **acelerómetro**; o **giroscópio** e o **magnetómetro**, cujas instâncias correspondentes são criadas como a seguir se ilustra:

```
// Para o acelerometro:
Sensor accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
// Para o giroscopio:
Sensor gyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
// Para o magnetometro:
Sensor magneticField = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

Monitorizar eventos dos sensores

Para que seja possível receber dados, a aplicação tem de subscrever os eventos associados aos sensores pretendidos. Usa-se assim um padrão *Publish–subscribe*, onde a aplicação subscreve os eventos para poder receber as novas medidas.

Para isso, regista-se o `SensorEventListener` de cada uma das instâncias dos sensores, chamando o método `registerListener()`. Neste processo é definido o atraso de envio destes eventos [20]. No caso do acelerómetro e giroscópio usa-se o valor que melhor permite detetar os extremos – `SENSOR_DELAY_FASTEST` (atraso de 0 ms) [20]. No caso do magnetómetro usa-se o valor padrão, `SENSOR_DELAY_NORMAL` [20].

De notar que, estes valores e atraso poderão ser ajustados após a validação do protótipo, de modo a reduzir o consumo de energia.

Em seguida, exemplifica-se como se processa a monitorização de eventos:

```
// Para o acelerometro:
sensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_FASTEST);
// Para o giroscopio:
sensorManager.registerListener(this, gyroscope, SensorManager.SENSOR_DELAY_FASTEST);
// Para o magnetometro:
sensorManager.registerListener(this, magneticField, SensorManager.SENSOR_DELAY_NORMAL);
```

Obtenção de valores do evento do sensor

Tendo a aplicação subscrito os eventos de um dado sensor, cada evento recebido contém as medidas do sensor, sem qualquer tipo de processamento (*raw data*) [20]. Isto é feito através da invocação do método `onSensorChanged()`, da interface `SensorEventListener`. Para distinguir entre os vários tipos de eventos, é utilizado o operador condicional `if` no método `onSensorChanged()`, conforme se ilustra em seguida:

```
// Se o novo evento for do sensor acelerometro
if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
```

```
    // Operar sobre o novo evento do sensor
}
// Se o novo evento for do sensor magnetometro
else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD){
    // Operar sobre o novo evento do sensor
}
// Se o novo evento for do sensor giroscopio
else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE){
    // Operar sobre o novo evento do sensor
}
```

Em geral, os sensores usam um sistema de coordenadas de 3 eixos para expressar os valores dos seus dados [36], o que acontece com os sensores utilizados neste sistema. Um dos campos da classe `SensorEvent` é um vetor que contem os valores para as três coordenadas:

```
sensorData = event.values;      // Copiar valores do evento para variavel
```

Outro campo relevante guardado por este objeto do evento é o *timestamp*, que contém o instante de tempo em que a medida é efetuada, medido em nanosegundos (ns) [36]. Este campo que é utilizado no Pedómetro após ser convertido para milissegundos.

4.2.2 Deteção de *beacons* BLE

É neste sub módulo que ocorre o *scan* ou varrimento dos *beacons* BLE circundantes. Neste componente apenas se efetua a recolha de dados, com o objetivo de registar todos os *beacons* detetados e o valor do RSSI medido para cada um deles.

Esta secção detalha a implementação desta parte do sistema, começando com a inicialização, seguido do *scan* efetivo destes dispositivos.

Inicialização

Verificar permissões – Enquanto que o acesso aos sensores é disponibilizado sem que seja necessário pedir ao utilizador, o mesmo não acontece com a deteção dos *beacons*. São necessárias permissões para o uso do *Bluetooth*, bem como para a localização do telemóvel, através desta tecnologia. Estas permissões são, em primeiro lugar, declaradas no manifesto da aplicação (`AndroidManifest`):

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Para que o sistema possa funcionar são necessárias as permissões de acesso à localização, sendo que se estas não estiverem garantidas, é necessário solicitar ao utilizador o acesso. Este acesso refere-se a dois tipos de permissões: `ACCESS_COARSE_LOCATION`, que permite á aplicação ter acesso á localização aproximada, e `ACCESS_FINE_LOCATION`, que permite à aplicação ter acesso à localização precisa [37]. Para o efeito, foi construído um método específico para verificar se é necessário reque-

rer permissão de uso, denominado `checkLocationPermissions()`. O código deste método pode ser consultado no anexo B.2.

***Bluetooth adapter* –** Um elemento necessário para fazer o *scan* dos *beacons* é o `BluetoothAdapter`, que representa o próprio adaptador do *smartphone* [38]. No código é declarada uma variável deste tipo, onde é depois guardado este *adapter* padrão:

```
BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();
```

Este `adapter` funciona apenas se o `Bluetooth` no telemóvel estiver ligado. Esta verificação é feita através do método `isEnabled()` do `BluetoothAdapter`, que retorna `true` se estiver ativo e pronto para uso [38]. Caso contrário, é criado um `intent` que pede ao utilizador para o ligar, através de um *pop-up* na interface. O método usado aqui é o `checkBluetoothPermissions()`, conforme se descreve em seguida:

```
public void checkBluetoothPermissions (){
    if (!adapter.isEnabled()) {
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        mainActivity.startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
    }
}
```

Scan dos *beacons*

Com o `adapter` inicializado, é possível então começar o *scan* para detetar sinais enviados pelos *beacons*. Neste sistema, isto é feito no método `scanBeaconsBLE()`. Este método começa por criar uma instância da classe `BluetoothLeScanner`, de forma a poder realizar as operações relacionadas com a varrimento de dispositivos BLE [39]. Este processo está ilustrado em seguida:

```
BluetoothLeScanner bluetoothLeScanner = adapter.getBluetoothLeScanner();
```

Antes de se iniciar a execução, é necessário definir o que fazer quando existirem resultados, através de um `ScanCallback`. A informação dos *beacons* é enviada para o módulo Posicionamento baseado em Fingerprinting, implementado na `Main Acitivity`. Isto é feito chamando o método `updateInfo Beacons()`, definido na interface `Callbacks` no `BeaconsService` e implementado na `MainAcitivity`. Os valores do *timestamp*, endereço MAC e do RSSI medido para o *beacon* são passados como argumentos, para que possa ser construída a *fingerprint*, se necessário. Esta invocação do método está contida no `onScanResult()` do `ScanCallback`:

```
ScanCallback scanCallback = new ScanCallback() {
    @Override
    public void onScanResult(int callbackType, ScanResult result) {
        super.onScanResult(callbackType, result);
        activity.updateInfoBeacons(System.currentTimeMillis(),
        result.getDevice().getAddress(), result.getRssi());
    }
};
```

O *scan* pode ser agora iniciado, invocando a função `startScan()` de `BluetoothLeScanner`, com o `ScanCallback` criado anteriormente como *input*:

```
bluetoothLeScanner.startScan(scanCallback);
```

4.3 Cliente - Módulo de Cálculo da trajetória

Esta secção descreve a implementação dos três sub-módulos associados ao cálculo da trajetória: o Pedómetro, que deteta o passo; a Deteção de curvas/viragens, que identifica viragens de 90º; e a Bússola, que calcula a orientação do utilizador.

4.3.1 Pedómetro

Como descrito na secção 2.1.4, o objetivo do pedómetro é detetar os passos do utilizador, para determinar então a distância percorrida. Nesta secção é descrita a implementação do pedómetro, incluindo a deteção e ratificação do passo, terminando com uma síntese que descreve o trabalho realizado e os desafios encontrados.

Implementação no sistema

Detetar o passo – O método `stepDetector()` implementa a lógica da deteção de passos descrita na secção 3.5.4, recebendo como *input* o último `SensorEvent`. O valor do eixo *z* é guardado numa variável e é então comparado com o valor registado anteriormente. Durante este processo são detetados um valor máximo e um valor mínimo, através da aplicação do algoritmo identificado na arquitetura. A estes valores está associado uma marca temporal, *timestamp*, correspondente ao instante de tempo em que o evento foi detetado.

Encontrado o par máximo-mínimo, é chamado o método auxiliar `checkMaxAndMinPair()`,que recebe como *input* o valor e o *timestamp* dos dois extremos. A diferença de amplitude e temporal é calculada e os resultados são comparados com os limites estabelecidos. Entre o máximo e o mínimo tem de ter passado mais de 60 ms e a diferença entre os dois tem que estar entre 4 e 9 m/s^2. Se este par cumprir estes requisitos, o método retorna `true`. A última verificação envolve a ratificação do passo, que é feita no método descrito a seguir.

Ratificar o passo – O método `ratifyStep` foi criado com base na análise em 3.3.2, para simplificar a forma de ratificar um possível passo, utilizando apenas os últimos valores registados pelo giroscópio, na altura em que é encontrado o mínimo da assinatura do acelerómetro. A lógica do método é simples: é recebido como *input* um *array* com os valores dos três eixos do giroscópio, guardados após o último evento do sensor; o vetor é então percorrido e cada valor verificado. Se todos os valores estiverem entre -0.5 e 0.5 rad/s, então o método retorna `true`, caso contrário, retorna `false`. É com base neste *output*, que o método responsável pela deteção vai assinalar ou não um novo passo. O código deste método pode ser consultado no anexo B.3.1.

Reinicializar as variáveis – Quando todas as verificações são cumpridas, a `MainActivity` é notificada, sendo responsável por atualizar a posição. Para tal, invoca-se o método `stepDetected()` da interface `Callbacks`. As variáveis que guardam o valor e o tempo do máximo são reinicializadas para 0, para que o processo possa começar de novo. De modo a evitar que máximos sem par bloqueiem a deteção futura, é estabelecido um limite de 350 ms após a ocorrência do extremo, para o qual estas variáveis são reinicializadas. O código para este método, `stepDetector()` e o auxiliar está presente em anexo B.3.1.

Observações finais

A lógica do pedómetro anteriormente descrita resultou de múltiplas iterações e alterações, que tiveram como objetivo criar uma versão simples e precisa do sistema.

A primeira versão construída com base na caracterização do movimento foi feita usando a linguagem de programação Python, pela simplicidade da sintaxe e pela necessidade de proceder a uma análise dos dados recolhidos que permitisse encontrar o máximo e o mínimo, característicos da assinatura dum passo. Nesta fase, os eixos *y* e *z* foram usados, mas os testes demonstraram que o eixo dos *y* era suficiente.

Tendo por base os resultados preliminares, o código foi portado para a plataforma Android e para a linguagem de programação Java. Testes experimentais realizados permitiram concluir que precisão registada anteriormente não se verificava na prática, sendo necessário definir gamas de valores para o máximo e mínimo. A avaliação desta versão foi positiva, com um erro médio de 1.14 passos, mas a sua complexidade e a contínua procura por uma maior precisão, levaram a uma nova versão. A lógica foi reformulada, passando a utilizar o eixo *z*, onde a assinatura é mais acentuada, e focada apenas nos dois extremos principais. Nesta versão foi introduzida a comparação entre os últimos valores registados, para verificar o estado do sinal, ou seja, se sobe ou desce. Foram também experimentados vários conceitos, nomeadamente o número de medidas e a diferença entre os extremos, tal como uma observação, que se provou errónea, de que entre o máximo e o mínimo, o sinal nunca voltava a subir.

A última versão criada simplificou ainda mais o código, eliminando por completo o *array* que era utilizado até aqui para guardar todos os valores registados. Foram estudadas novas possibilidades para melhorar ainda mais a precisão, relacionadas com a diferença de tempo, entre dois máximos consecutivos e entre o máximo e o mínimo. A primeira hipótese falhou, devido á imprevisibilidade do ritmo do movimento do utilizador; no entanto, a segunda permaneceu. Esta versão foi melhorada na fase final desta tese, reorganizando o código de modo a ficar mais simples e legível.

O código de todas estas versões encontra-se no anexo B.3.1 desta tese.

4.3.2 Deteção de curvas/viragens

Com base na caracterização das viragens, na secção 3.3.3, foi então definida a lógica necessária para detetar a respetiva assinatura e distinguir entre as possíveis direções tomadas, esquerda ou direita. Nesta secção é descrita a implementação da deteção de curvas/viragens, concluindo com uma síntese

sobre o trabalho desenvolvido e os problemas encontrados.

Implementação no sistema

Sendo que, para detetar uma mudança de direção basta encontrar um extremo no sinal do eixo *z* do giroscópio, foi reutilizada a lógica do pedómetro relacionada com a identificação de subidas e descidas no sinal, através da comparação entre o valor medido anteriormente e o atual. O método `turnDetector()` foi construído para o efeito, implementado a lógica descrita na secção 3.5.4, e recebendo como *input* um `SensorEvent` do giroscópio. Para ser um máximo, o valor anterior deve estar acima de 1.8 rad/s e, para ser um mínimo, deve estar abaixo de -1.8 rad/s. Ao detetar uma viragem, o módulo responsável pela estimativa da posição é notificado, que neste caso é a `MainActivity` da aplicação (ver secção 4.4.1), enviado uma variável `boolean` que indica a direção, representada pelo valor `false` para a esquerda e `true` para a direita.

No entanto, esta lógica teve de ser adaptada para o facto de não existir uma altura específica em que o valor das variáveis auxiliares seja reinicializado, o que no pedómetro acontece aquando da identificação do mínimo. Para além disso, todos os valores posteriores aos extremos e ainda dentro dos limites, são aceites como uma nova viragem, devido á forma do algoritmo. Ao invés de simplesmente comparar o possível máximo/mínimo com aquele guardado, a solução escolhida foi a implementação de uma técnica de bloqueio ou trancas, que impeça que, após um extremo ser detetado, um novo valor possa acionar imediatamente outra notificação. Para isso, é utilizada uma variável `boolean`, cujo valor *default* é `false`, atualizada quando é encontrado um máximo ou um mínimo. Um `if` condicional permite bloquear a execução do resto da lógica. Quando o sinal, após uma viragem, volta a estabilizar perto do valor 0 rad/s, é então feito o desbloqueio. Assume-se uma margem de 0.1, dentro da qual o sinal é considerado como estável e a variável volta a ter o valor `false`. O código final deste método está presente no anexo B.3.2.

Observações finais

O desenvolvimento da deteção de curvas/viragens começou após a finalização do pedómetro, quando foi necessário encontrar uma forma de determinar a posição no espaço bidimensional do museu.

A bússola foi inicialmente considerada como o sensor mais adequado para este fim. No entanto, a forma como estava implementada no sistema tornou complicada a sua utilização, uma vez que usava o referencial global e não local ao museu. Esta seria melhor integrada no sistema posteriormente, porém, devido á variação de orientações durante a viragem, esta não seria a escolha mais apropriada. A primeira versão desenvolvida não continha a técnica de bloqueio, que veio a ser adicionada mais tarde, após o problema das curvas 'fantasmas' surgir durante os testes da integração com a estimativa da distância.

O código desta versão inicial da deteção de curvas está presente no anexo B.3.2.

4.3.3 Bússola

Uma parte fulcral de qualquer sistema que use *Dead Reckoning* para localização é a bússola, que indica a direção do utilizador ao longo do tempo. Contudo, há que ter em consideração que a presença de equipamentos eletromagnéticos no museu pode causar resultados incorretos, razão pela qual é utilizada a fusão de sensores no cálculo da orientação.

Nesta secção são descritas as várias fases de implementação da bússola no sistema. Algumas considerações iniciais são apresentadas, de modo a explicar detalhes relacionados com as medidas no sistema de coordenadas esférico.

Considerações iniciais

O resultado da estimativa da orientação obtém-se a partir do valor do azimute, uma medida de abertura angular do sistema de coordenadas horizontal [40], [41] [3].

De modo a poder relacionar o valor do azimute com uma direção específica, consideremos então a rosa-dos-ventos, composta por quatro pontos cardeais e quatro pontos colaterais. O valor do azimute é retornado em radiano (rad), sendo que o sistema de coordenadas vai de $-\pi$ a π rad, ambos correspondentes ao sul (S). O resto dos pontos correspondem assim:

- sudoeste (SO/SW) - $-\frac{3\pi}{8} rad$;
- oeste (O) - $-\frac{\pi}{2} rad$;
- noroeste (NO) - $-\frac{\pi}{4} rad$;
- norte (N) - $0\, rad$;
- nordeste (NE) - $\frac{\pi}{8} rad$;
- este (E) - $\frac{\pi}{2} rad$;
- sudeste (SE) - $\frac{3\pi}{4} rad$;

A disposição destes pontos no espaço com o valor do azimute correspondente, em radianos, é apresentado na figura 4.2.

Implementação no sistema

A implementação do bússola foi feita em duas fases. Na primeira fase, foi construída uma bússola simples, utilizando apenas os dados dos sensores acelerómetro e magnetómetro. O problema deste tipo de bússola é, recordando a descrição na secção 2.1.4, ser suscetível á interferência de componentes elétricos e metálicos, causando um erro elevado na estimativa de orientação. Assim, numa segunda fase foi implementada uma bússola com fusão de sensores, que utiliza a orientação calculada com o giroscópio e a da bússola base para determinar uma estimativa final. O desenvolvimento desta parte do sistema é descrito em detalhe nesta subsecção, abordando as duas fases de implementação.

[3] sistema que usa o horizonte local do observador como plano fundamental e mede o ângulo entre o vetor do norte e o vetor da direção do utilizador

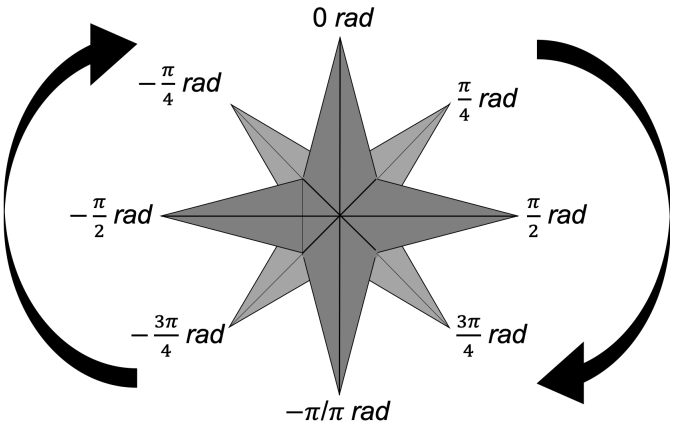


Figura 4.2: Rosa-dos-ventos: relação entre a abreviatura dos pontos cardeais e colaterais e os valores do azimute correspondentes (medidos em radianos)

Bússola base – A bússola base implementada foi adaptada de [42], que exemplifica uma forma simples de estimar a orientação do utilizador, com base nos sensores magnetómetro e acelerómetro. A razão pela qual este último é utilizado está relacionada com a transformação de um vetor do sistema de coordenadas do dispositivo para o sistema de coordenadas mundial [43]. Cada vez que um destes sensores gera um novo evento, os valores medidos são guardados em vetores específicos, sendo estes dados necessários para calcular a orientação. Um terceiro vetor é utilizado para guardar os resultados deste cálculo, que são:

- Índice 0 - azimute, o ângulo de rotação sobre o eixo *z*;
- Índice 1 - *pitch*, o ângulo de rotação sobre o eixo *y*;
- Índice 2 - *roll*, o ângulo de rotação sobre o eixo *x*.

A orientação é então calculada no método `getAccelMagneticOrientation()`, através de dois métodos da classe `SensorManager`: `getRotationMatrix()` e `getOrientation()`. Este primeiro, recebe o vetor `R`, onde é guardada a matriz de rotação resultante; o vetor `I`, que guarda a matriz de inclinação, que neste caso foi substituído por `null`; o vetor com os últimos valores guardados do acelerómetro e outro com os do magnetómetro [43]. Este método retorna um `boolean`, que assinala o sucesso ou falhanço da estimativa. O vetor `R` guarda:

- *x*, que é definido como o produto *y.z*. É tangencial ao chão no local onde o dispositivo se encontra, apontando para este;
- *y*, que é tangencial ao chão no local onde o dispositivo se encontra e aponta para o Polo Norte magnético;
- *z*, que aponta para o céu e é perpendicular ao chão [43].

A figura 4.3 demonstra a posições dos três eixos do sistema de coordenadas, em relação ao dispositivo. O método `getOrientation()` recebe então a matriz `R` e um vetor onde serão guardados os

resultados. Estes são, tais como descritos em cima, o azimute, o *pitch* e o *roll*. No entanto, para este sistema, é necessário apenas o azimute. Este resultado é retornado em radiano. O código desta bússola base está no anexo B.3.3.

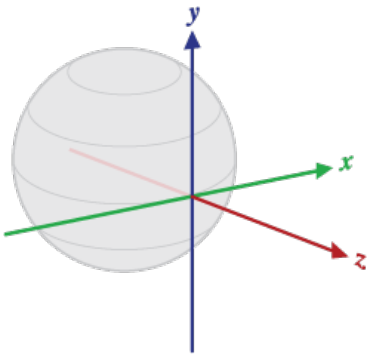


Figura 4.3: Sistema de eixos da matriz de rotação, em que o ponto (0,0,0) é a localização do dispositivo [43].

Bússola com fusão de sensores (Giroscópio) – O código da fusão de sensores foi adaptado de [30], que apresenta um tutorial sobre o assunto. Basicamente, as medidas do giroscópio são utilizadas para calcular uma outra estimativa, que será associada com a estimativa baseada no acelerómetro/magnetómetro, para determinar a orientação final [30]. Todavia, a direção fornecida por este sensor não é necessariamente a direção absoluta [10], razão pela qual é necessário processar estes dados de uma forma específica. Para obter esta orientação real, os valores da velocidade de rotação medidos pelo giroscópio são integrados ao longo do tempo, multiplicando estes valores com o intervalo de tempo entre o último e o atual evento do sensor, o que fornece o incremento de rotação [30]. A soma de todos este incrementos é então a direção absoluta do dispositivo [30]. Todo o código que será descrito nesta secção está presente no anexo B.3.3.

Existem dois vetores principais, inicializados logo na criação da atividade ou serviço: a matriz de rotação e o vetor de orientação do giroscópio, que contém o azimute, *pitch* e o *roll*. Este código é implementado no método `initializeGyroOrientation()`, construído para o efeito, com base no original. A função `getAccelMagneticOrientation()` mantém-se, pois a duas orientações (giroscópio e acelerómetro/magnetómetro) são estimadas de forma individual.

Para determinar a orientação, é usado o método `gyroOrientation`, que recebe um objeto `SensorEvent`, de onde são retirados os valores medidos e o respetivo *timestamp*. É aqui que os intervalos de rotação do giroscópio são adicionados á orientação absoluta baseada neste sensor [30]. No entanto, estas são matrizes de rotação e não ângulos, dado que não é possível simplesmente adicionar os intervalos de rotação, sendo aplicados através da multiplicação das matrizes [30]. Este método só pode ser iniciado se a orientação baseada no acelerómetro e magnetómetro tiver sido estimada anteriormente, já que esta é a orientação inicial considerada para o giroscópio [30]. A matriz de rotação é calculada a partir desta orientação e do vetor de rotação do sensor [30].

O vetor de rotação pode ser convertido para matriz pelo método `getRotationMatrixFromVector()` da classe `SensorManager`, no entanto, o autor original opta por desenvolver a sua própria função, de-

signada `getRotationMatrixFromOrientation()`, de modo a converter os ângulos de orientação numa matriz de rotação [30]. Esta conversão passa por criar três matrizes de rotação, uma para cada eixo, *x* (*pitch*), *y* (*roll*) e *z* (azimute), com os respetivos valores do seno e coseno. De notar que os eixos são trocados, sendo que o valor do eixo *x* do vetor da orientação, que guarda o azimute, passa para o eixo *z*, durante este cálculo. A matriz de rotação final é a multiplicação das três martizes, começando com *x* e *y*, depois multiplicando o resultado com *z*.

Continuando no método `gyroOrientation()`, os dados do giroscópio necessitam de processamento adicional, através da função `getRotationVectorFromGyro()`, que recebe o vetor dos valores medidos, o vetor de rotação vazio e o fator de tempo, que é igual ao intervalo de tempo a dividir por 2. Este vetor de rotação guarda o último intervalo de rotação que será aplicado á matriz de rotação [30]. A partir desta matriz é obtida a orientação, guardada no vetor da orientação do giroscópio.

É necessário agora a fusão da duas orientações, através de um filtro complementar, executado numa `TimerTask`. O tutorial apresentado em [30] difere aqui da versão implementada, dado que o autor resolveu fazer alterações na aplicação, de modo a corrigir um erro de transição entre π e $-\pi$. Assim, o código verifica se um dos dois ângulos de orientação é negativo, seja o obtido com o giroscópio ou com o par acelerómetro/magnetómetro. Se algum dos casos se verificar, então é somado 2π ao valor negativo e só depois é realizada a fusão dos dados dos sensores. O coeficiente `FILTER_COEFFICIENT` é um valor determinado heuristicamente através da rotação de um modelo 3D do *smartphone* do autor, tendo sido escolhido um valor de 0.98 com uma taxa de amostragem de 33 Hz, que define um intervalo de 30 ms entre execuções da `TimerTask` [30]. Após o cálculo do novo valor, este é enviado para a `MainActivity`.

Observações finais

A bússola é o único sub módulo do Cálculo da trajetória que foi adaptado de fontes externas. O intuito durante a fase de desenvolvimento foi sempre de utilizar o giroscópio no cálculo da orientação, tal como indicam alguns dos artigos no estado de arte [10], [2]. No entanto, nenhuma solução foi encontrada inicialmente, sendo a bússola base implementada como uma segunda opção. A técnica da fusão de sensores acabou por ser aplicada mais tarde, tendo a sua adaptação requerido uma extensa análise ao código original. A orientação retornada pelo sistema é o valor do azimute, que é resultado da integração de duas estimativas, uma calculada com as medidas dos sensores magnetómetro e acelerómetro, e outra com os dados do giroscópio. No entanto, como este sensor apenas fornece a direção relativa, as medidas registadas têm de ser processadas de forma diferente.

4.4 Cliente - Módulo de Estimativa da posição

Esta secção descreve a implementação do sub-módulo de estimativa de posição que foi implementado, o Posicionamento baseado em trajetória. Os dois sub-módulos referidos na arquitetura – o Posicionamento baseado em *Fingerprinting* e Correção da posição– não foram implementados. A razão deste

facto prende-se com as dificuldades já referidas de caracterização do movimento , mas também com os resultados obtidos durante a fase de testes.

4.4.1 Posicionamento baseado em trajetória

A caracterização da trajetória, detalhada na secção 3.4, especifica as possíveis direções consideradas para a deslocação do utilizador, que podem ser seguir em frente, virar ou circular na diagonal. Cada percurso tem uma distância e direção associada, sendo estas informações necessárias para determinar a localização. A atualização da posição é iniciada com a deteção de um passo no Pedómetro. A direção do percurso, baseada nas informações retornadas pela Deteção de curvas/viragens e pela Bússola, permite identificar como devem ser as coordenadas atualizadas. A Bússola também é utilizada para determinar, em qualquer altura, a orientação do utilizador em relação ao referencial do museu.

Estimativa da distância percorrida

Tal como indicado na secção 3.4.1, a distância percorrida é igual ao numero de passos multiplicado pelo tamanho da passada. No entanto, como neste protótipo a posição é atualizada sempre que o utilizador dá um passo, a distância percorrida será igual ao número de quadrículas percorridas, que é derivado da conversão do tamanho da passada. Este é calculado através do método `calculateStepSize()` que recebe como *input* a altura e um `boolean` que indica o sexo, com valor `true` para homens e `false` para mulheres. Estas informações são essenciais para a estimativa da posição, que não funcionará sem as mesmas. O código deste método, que implementa a fórmula 3.1, encontra-se no anexo B.4.

O método `calculateSqaure()` foi construído para fazer esta conversão, estimando o número de quadrículas percorridas, e recebendo como *input* o tamanho do passo e uma variável `boolean`, que indica se o utilizador se encontra a transitar diagonal aos eixos. Dependendo desta última variável, o resultado da divisão é arredondado para baixo com o método `Math.floor()` (retorna o maior inteiro que for menor ou igual ao argumento [44]), caso o valor seja `false`, o que indica um percurso paralelo a um dos eixos, ou é arredondado para cima com o método `Math.ceil()` (retorna o inteiro mais pequeno que seja maior ou igual ao argumento [45]), caso o valor seja `true`. O código deste método está presente no anexo B.4.

Estimativa da posição com base na deteção de curvas

A deteção de curvas é feita no `SensorService`, que notifica a `MainActivity`, da viragem do utilizador e da direção tomada. Aqui é implementado o método `notifyDirectionChange()`, definido na interface `Callbacks` no `SensorService`, chamado quando é detetada uma viragem. Uma variável `boolean`, que indica se o utilizador virou anteriormente, é atualizada para `true`. Recordando a metodologia da deteção de curvas, descrita na secção 4.3.2, a esquerda é representada pelo valor `false` e a direita pelo valor `true`. Dado que esta representação é mantida durante todo o funcionamento do sistema, é necessário estimar então os parâmetros da mudança, ou seja, qual a coordenada a ser atualizado

e se a esta será somado ou subtraído o número de quadrados percorridos. Estes parâmetros são determinados no método `determineDirectionChangeParameters()`, chamado logo após a notificação.

Recebendo a direção da viragem como *input*, este método utiliza esta informação em conjunto com os dados anteriores, para determinar os dois aspetos descritos acima. Em primeiro lugar, é necessário determinar qual a atualização ao valor, que depende da posição do vetor do percurso anterior á curva, em relação aos eixos, isto é, se é paralelo a *x* ou a *y*, e também a direção. Por exemplo, o utilizador começa um percurso, cujo vetor correspondente é coincidente com eixo *y*, virando posteriormente á direita. De acordo com o referencial, o novo vetor que corresponde ao percurso atual é paralelo ao eixo *x*, sendo esta coordenada incrementada, enquanto que a do *y* mantém-se. Num outro exemplo, considere-se outro vetor, que contínua a ser paralelo a *y*, no entanto, em direção contrária. Quando o utilizador vira á direita, o novo vetor é paralelo ao eixo *x*, com esta coordenada a ser subtraída. O mesmo acontece se o vetor inicial do percurso for paralelo ao eixo *x*. Estes exemplos demonstram a complexidade aquando da determinação da forma de atualização da posição, tornando a situação confusa e dificultando a escolha de uma solução para o problema.

Para identificar se a coordenada será incrementada ou reduzida, é utilizada uma variável `boolean` denominada, em que o valor `true` significa um acréscimo e o valor `false` significa um decréscimo, ou seja, se vai ser adicionado ou subtraído o número de quadrículos percorrido. O valor desta variável é determinado como base na direção da curva e no eixo a que o último vetor do percurso era paralelo. O último parâmetro determinado corresponde a uma outra variável `boolean`, que indica qual a coordenada a ser atualizada, que será sempre o inverso do valor anterior. Se o vetor do percurso for paralelo ao eixo *y*, a variável tem valor `true`, se for paralelo ao eixo *x*, terá valor `false`.

Com estes parâmetros definidos, a nova posição do utilizador é estimada no método `estimate CurrentPositionBasedOnTurnDetection()`. O método `signToValue()` transforma o número de qaudrículas percorrido num valor negativo ou positivo, conforme a situação atual. Neste momento, apenas é considerada a dimensão lateral da quadrícula. Este resultado será então somado á coordenada a atualizar, a qual foi determinada anteriormente. O código deste método e o auxiliar está presente no anexo B.4. A estimativa com base na deteção de curvas irá funcionar em conjunto com a utilização da orientação retornada pelo bússola, para determinar a localização do utilizador seja qual for a direção possível que este tomar.

Estimativa da posição com base na orientação

A bússola notifica a `MainActivity` quando é calculado um novo valor para a orientação, através da invocação do método `updateAzimuth()`. Isto permite identificar qual o valor do azimute para certas situações nomeadamente: após o último passo, após a a última curva e na direção avante no ponto (0,0). A comparação entre estes dados permitirá identificar qualquer que seja o tipo de percurso.

Ao ser detetado um novo passo, invocando `stepDetected()`, é chamado o método `estimateNewUser Position()`, responsável pela atualização da posição. Aqui, o primeiro passo, é determinar se a direção é a mesma desde o último passo, através da comparação entre os dois valores do azimute. Isto é feito, calculando a diferença entre os dois através da função `adjustAzimuth()`, que também ajusta o resul-

tado, caso este esteja fora do sistema de coordenadas ($-\pi$ a π rad), somando ou subtraindo 2π. O código deste método está presente no anexo B.4. Neste sistema é considerada uma margem de $\frac{\pi}{8}rad$ para uma direção, o que significa que se esta diferença estiver entre $-\frac{\pi}{8}rad$ e $\frac{\pi}{8}rad$, então considera-se que o utilizador não mudou de direção. Se o mesmo não se verificar, é necessário determinar qual a situação atual do percurso, comparando o valor do azimute deste passo com aquele registado na última viragem. O método `orientationSituation()`, cujo código está no anexo B.4, faz precisamente isto, retornando um número de 1 a 4, que indica o tipo de situação:

1. O novo vetor do percurso é perpendicular ao anterior e aponta para a esquerda (a diferença está entre $-\frac{5\pi}{8}rad$ e $-\frac{3\pi}{8}rad$);
2. O novo vetor do percurso é diagonal aos eixos e aponta para a esquerda (a diferença está entre $-\frac{3\pi}{8}rad$ e $-\frac{\pi}{8}rad$);
3. O novo vetor do percurso é perpendicular ao anterior e aponta para a direita (a diferença está entre $\frac{5\pi}{8}rad$ e $\frac{3\pi}{8}rad$);
4. O novo vetor do percurso é diagonal aos eixos e aponta para a direita (a diferença está entre $\frac{\pi}{8}rad$ e $\frac{3\pi}{8}rad$).

A distinção entre uma curva ou um desvio na diagonal é feita impedindo estas verificações imediatamente após a deteção. É nesta altura que o valor do azimute após a última curva é guardado, e este bloqueio é terminado, atribuindo o valor `false` á variável que indica a ocorrência de uma viragem.

Voltando á situação de orientação, o valor retornado é passado para o método `resolveOrientationSituation()`, que com base nisto irá então determinar os novos parâmetros para a atualização das coordenadas. No caso das situações 1 e 3, considera-se que ocorreu uma curva de 90º, dado que o novo vetor de percurso é perpendicular a um anterior, antes do caminho diagonal. A função `determineDirectionChangeParameters()` é chamada, passando como *input* a nova direção. Já no caso das situações 2 e 4, apenas é alterado o valor da variável que indica se o percurso atual é diagonal aos eixos para `true`.

O último passo é atualizar as coordenadas, conforme a direção do percurso. Caso o utilizador esteja a circular numa diagonal, é invocado o método `estimateCurrentPositionBasedOnCompass()`, que modifica o valor no *x* e *y*. Dependendo do eixo a que o vetor do percurso anterior era paralelo, uma das coordenadas manterá a mesma forma de atualização, enquanto que a outra é alterada com base na direção atual. Aqui também é utilizado o método `signToValue()` para atualizar a posição, considera-se porém a dimensão da diagonal, para calcular o número de quadrículas percorridas. O código de todos os métodos descritos está presente no anexo B.4.

Direção – Outro aspeto relevante do posicionamento, envolve o uso da orientação da bússola para identificar qual a direção do utilizador, num certo instante, em que este esteja parado. No protótipo criado, isto é feita quando ocorre um pedido de localização, em que é retornada a posição atual, que é calculcada sempre que é detetado um passo, e esta direção. Para simplificar a implantação no sistema,

optou-se por recorrer apenas á orientação da bússola. O método `estimateUserDirection()`, construído para o efeito, recebe como *input* dois valores do azimute, o último registado e o correspondente á direção frontal na posição inicial, que pode ser gravado manualmente ou após o primeiro passo. Este método funciona de forma similar ao `orientationSituation()`, calculando a diferença entre este dois valores para determinar qual a direção atual:

- Se a diferença estiver entre $-\frac{3\pi}{4}rad$ e $-\frac{\pi}{4}rad$, o utilizador está virado para a esquerda, sendo retornado 'L';
- Se a diferença estiver entre $\frac{\pi}{4}rad$ e $\frac{3\pi}{4}rad$, o utilizador está virado para a direita, sendo retornado 'R';
- Se a diferença estiver entre $-\frac{\pi}{4}rad$ e $\frac{\pi}{4}rad$, o utilizador está virado para a frente, sendo retornado 'F';
- Se a diferença estiver é maior que $\frac{3\pi}{4}rad$ ou menor que $-\frac{3\pi}{4}rad$, o utilizador está virado para trás, sendo retornado 'B'.

Dado que as direções diagonais não são consideradas aqui, cada direção tem uma margem de $\frac{\pi}{2}rad$. O código deste método está presente no anexo B.4.

Observações finais

A implementação da estimativa da posição foi complexa, não só por causa das possíveis direções, mas também pela utilização da Bússola para a identificação destas direções. Existem aqui vários elementos que funcionam de forma diferente, mas complementar. A lógica utilizada considera que cada vez que um passo é dado, a posição tem de ser alterada para a nova localização, somando ou subtraindo o número de quadrículas percorridas, calculado em função da dimensão do quadrado, que depende da direção, e do tamanho do passo. As coordenadas são atualizadas dependendo do tipo de percurso, paralelo ou diagonal aos eixos. No primeiro caso, a atualização é feita com base na Deteção de curvas, cuja informação permite determinar a que eixo o percurso é paralelo. Esta situação apresentou alguns desafios, nomeadamente no que diz respeito á influência que as direções anteriores têm na alteração da coordenada respetiva. No segundo caso, esta é feita com base na Bússola, que é utilizada para determinar se a orientação mudou desde o último passo, e em caso afirmativo, proceder á determinação da situação atual do percurso, da qual vai depender a atualização das coordenadas. A Bússola permite identificar a direção das movimentações do utilizador em relação ao referencial, nomeadamente desvios de percursos paralelos aos eixos e o eventual regresso a estes. A sua implementação foi complicada, designadamente o encaixe deste elemento na estimativa da posição. Apesar de as coordenadas serem atualizadas constantemente, o protótipo desenvolvido apenas retorna a posição quando o utilizador requerer, devolvendo também a direção para qual este voltado, esquerda, direita, frente ou trás. Esta informação é estimada através da orientação da Bússola.

4.5 Servidor - Módulo de Processamento de dados

Por razões relacionadas com tempo, não foi possível desenvolver por completo um servidor funcional para o sistema, no entanto, a técnica de *Fingerprinting* foi implementada com recurso a *machine learning*, ao invés de uma base de dados de *fingerprints*. Esta secção descreve as diferentes fases, começando com a recolha de dados, seguida da criação e treino do modelo, e previsão.

4.5.1 Modelo de previsão

Recolha de dados para treino

O primeiro passo da fase de treino consistiu no desenvolvimento de uma aplicação móvel que permitisse a recolha das *fingerprints* de cada uma das posições relevantes no museu. A interface é simples, permitindo identificar as coordenadas da localização e gravar as medidas. São guardadas 30 *fingerprints* para cada posição, cada uma com um intervalo de 100 ms, utilizando uma `TimerTask`. Cada *fingerprint* contém as seguintes informações:

```
Nº; RSSI Beacon 1, ..., Beacon 4; x,y,z do acelerómetro; x,y,z do giroscópio;
x,y,z do magnetómetro; azimute
```

No final, as 30 *fingerprints* são gravadas num ficheiro `.txt` individual.

Para a recolha de dados efetiva, são utilizados os quatros *beacons* colocados no espaço do museu, cujas posições são demonstradas na figura 3.1 e descritos na tabela 4.4. Para cada posição relevante do museu, um utilizador a segurar um telemóvel Samsung Galaxy Note 8, a correr esta aplicação, gravou as *fingerprints* manualmente, para cada uma das possíveis direções, em relação ao referencial do museu: frente; trás; esquerda e direita. Foram 119 posições, para as quais foram recolhidas 120 *fingerprints* cada, com algumas exceções, perfazendo um total de 14 220.

Beacon nº	Cor	ID	Endereço MAC	Posição (x,y)
1	Blueberry pie 1	42530	FC:05:40:8B:A6:22	(8, 10-11)
2	Icy marshmallow	57471	EE:A9:28:94:E0:7F	(5, 15)
3	Mint cocktail	56160	D6:AB:20:A4:DB:60	(5-6, 0)
4	Blueberry pie 2	42455	D1:34:AE:CB:A5:D7	(0, 5)

Figura 4.4: Tabela com as informações dos *beacons* utilizados no museu.

Criação e treino do modelo

Através de uma pesquisa na internet foram encontradas várias soluções para criar um modelo com *machine learning*, porém, a maior parte destas eram bastante complexas e requeriam bastante trabalho de compreensão e adaptação. As mais simples e interessantes utilizavam uma biblioteca de *open source* da Google chamada *TensorFlow*, cuja tutorial em [46] explica o seu funcionamento, através de um exemplo com classificação de imagens. A solução escolhida, cujo código está em [47] e descrita num artigo em [48], utiliza os valores de RSSI de mais de 500 WAPs para determinar uma localização. O modelo resultante é criado com dois conjuntos de dados: de treino e de validação.

Antes de poder construir o modelo, foi necessário usar as *fingerprints* recolhidas na fase de treino e adaptá-las para os ficheiros de treino e validação, através de um programa feito em Python, cujo código está no anexo B.5.1. As 30 *fingerprints* foram utilizadas para o treino, enquanto que para a validação, a média de cada valor era calculada para criar uma nova *fingerprint*. Para este conjunto de dados não foram considerados os valores do acelerómetro e do giroscópio, pois estes estão relacionados com o movimento do telemóvel. Cada *fingerprint* fica na forma:

```
RSSI Beacon 1, ... Beacon 4; x,y,z do magnetómetro; azimute, x, y, direção
```

A direção é definida como L para a esquerda, R para a direita, F para a frente e B para trás.

A solução baseada em [47] começa por ler os dados de treino e separar as medidas das classes (ou *labels*), que são o conjunto das informações de posição, representadas como `X_Y_DIREÇÃO`. Tanto as *fingerprints* como as classes são convertidas em *arrays* `numpy`. No código original, o autor opta por dividir estes dados em dois conjuntos, um para treino e outro para validação durante o construção do modelo, no entanto, na implementação neste sistema, apesar de esta divisão permanecer, todas as *fingerprints* são usadas para treino. As camadas (*layers*) são então configuradas e o modelo é compilado, sendo este treinado e validado inicialmente duas vezes, uma durante e outra após esta fase. O modelo final é então validado definitivamente com os respetivos dados, retornando no final a precisão, ou seja a quantidade de previsões certas, e a perda.

O modelo final, resultante da execução do código, é então gravado num ficheiro HDF5, tal como demonstrado em [49], que permite a sua fácil utilização noutros programas. A implementação aqui descrita, adaptada á situação desta tese, está presente no anexo B.5.2.

Fazer previsões

Fazer uma previsão com o modelo não retorna automaticamente a classe, ou *label*, correspondente á estimativa da posição. Ao invés disso, é retornado um *array* com valores que indicam a confiança do modelo de que a *fingerprint* corresponde a cada uma das diferente classes [46]. Para determinar aquela com o maior valor de confiança, basta determinar o máximo. No código presente no anexo B.5.2, isto é demonstrado na parte onde é feita a avaliação da capacidade de previsão do modelo. A partir do índice com a confiança máxima, é determina a *label* com a posição, através de uma lista que contém todas as classes ordenadas.

4.5.2 Observações finais

A implementação do modelo de *fingerprinting* começou após o término da aplicação Cliente, dada a necessidade de uma técnica que pudesse ser utilizada para reduzir o erro da estimativa com base na trajetória. O primeiro passo foi a recolha das medidas observadas para cada uma das posições no espaço do museu.

Ao fazer previsões sobre uma certa *fingerprint*, é necessário determinar qual a classe, que corresponde á posição, para a qual o modelo tem maior confiança. Os primeiros modelos construídos

foram treinados e validados com conjuntos de valores uniformizados, pelo método `scale()`, no entanto, aquando da previsão, não é possível fazer o mesmo, dado que apenas existe uma *fingerprint*. Por este motivo, foi necessário criar um modelo sem os dados uniformizados, o que causou uma redução da precisão para de 94.5% para 76%, com alguns criados posteriormente a registar entre uma precisão entre 83% a 91%.

Capítulo 5

Resultados Experimentais

Neste capítulo é apresentada a avaliação, e os respetivos resultados e análise, dos principais módulos implementados no sistema. A secção começa com avaliação final dos três elementos do Cálculo da trajetória, na secção 5.1. De seguida, é feita uma avaliação ao *Fingerprinting*, através de uma análise á cobertura dos *beacons* utilizados. Por fim, na secção 5.3, é apresentada a avaliação á estimativa da posição, baseada na trajetória, calculada no respetivo módulo, e baseada em *Fingerprinting*, através de uma simulação com o modelo.

5.1 Avaliação do Cálculo da trajetória

Após a finalização de cada um dos elementos deste módulo (Pedómetro, Deteção de curvas/viragens e Bússola), foi necessário validar a sua eficácia e precisão, antes que se pudesse avançar para as próximas etapas. Esta secção apresenta a metodologia de avaliação e os resultados obtidos, bem como a sua análise.

5.1.1 Pedómetro

O teste do pedómetro tem por objetivo avaliar se o sistema consegue contabilizar corretamente o número de passos dados.

Para o efeito, foi feita um experiência simples em que um utilizador (rapaz com 23 e 1,78 m de altura) efetuava um percurso de 7 passos, em linha reta, num local interior com chão plano. O utilizador transportava o telemóvel que foi usado para o teste (Samsung Galaxy Note 8), tentando mantê-lo sempre na mesma posição à altura do peito, na orientação vertical ou ”retrato”, permanecendo imóvel em relação ao corpo. A figura 5.1 demonstra o esquema utilizado para esta experiência.

Este percurso foi repetido sete vezes, de modo a obter uma maior amostra de resultados para análise. No final de cada teste, o número de passos detetado pela aplicação foi registado, sendo estes valores apresentados na tabela da figura 5.2.

Do conjunto dos 7 testes efetuados, ocorreram duas situações em que verificaram falsos negativos, sendo o erro médio de cerca de 0.29 passos. Nos dois testes em que o erro ocorreu foi detetado menos

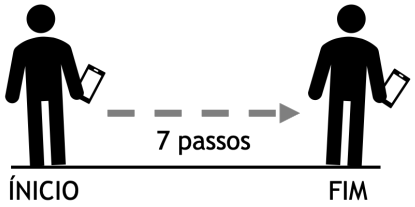


Figura 5.1: Esquema do teste efetuado ao Pedómetro.

Teste nº	Nº de passos detetados
1	7
2	7
3	7
4	6
5	7
6	7
7	6
Média	6,714
Erro médio	0,285714286

Figura 5.2: Tabela com os resultados da avaliação feita á versão final do pedómetro.

um passo.

5.1.2 Deteção de curvas/viragens

Para avaliar a eficácia da deteção de viragens, foi realizada uma experiência na qual o utilizador circula num percurso com múltiplas esquinas e obstáculos, sendo que para os contornar tem de fazer uma curva de 90º para um dos lados. As condições desta experiência foram as mesmas das da experiência descrita anteriormente.

Para testar também a rapidez de adaptabilidade, certas viragens realizam-se muito próximas. Foram definidas 5 curvas, tanto para a direita como para a esquerda, num total de 10. A figura 5.3 demonstra o percurso de teste usado.

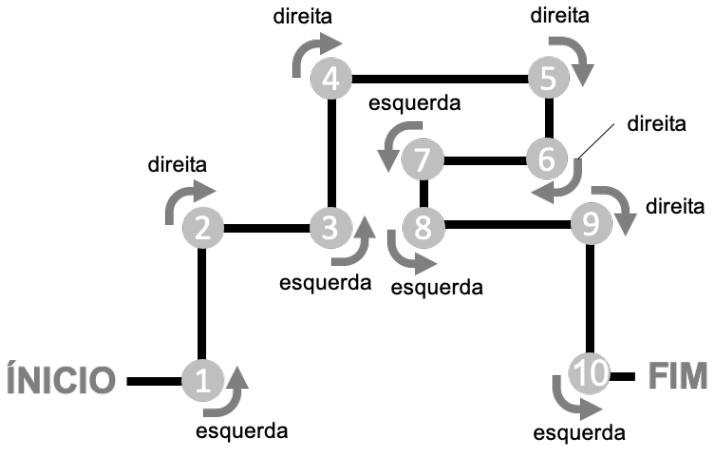


Figura 5.3: Ilustração do percurso usado para testar a deteção de viragens/curvas, com estas identificadas com a direção na perspetiva do utilizador.

Para fazer a análise dos resultados *a posteriori*, a aplicação regista num ficheiro a deteção uma curva, incluindo informação como a direção e o valor máximo medido. Também foi analisada a técnica de bloqueio implementado, mais propriamente o desbloqueio. O *output* em RAW do ficheiro gerado está presente no anexo C.1.1. A tabela da figura 5.4 demonstra melhor os resultados, relacionando-os com o percurso real.

Curva	Direção real	Direção estimada	Valor máximo/mínimo (*rad/s*)
1	esquerda	esquerda	2,48575
2	direita	direita	-2,7182112
3	esquerda	esquerda	3,088674
4	direita	direita	-1,8782713
5	direita	direita	-2,3119855
6	direita	direita	-2,913688
7	esquerda	esquerda	2,3134859
8	esquerda	esquerda	2,4313831
9	direita	direita	-2,5416708
10	esquerda	esquerda	2,765526

Figura 5.4: Tabela com os resultados do teste realizado para a avaliação da deteção de curvas.

Observando a tabela, é possível verificar que todas as curvas do percurso foram detetadas. Estes resultados demonstram uma alta precisão desta lógica, não só na deteção da viragem, mas também na identificação da nova direção tomada. A observação dos valores dos máximos/mínimos também demonstra que o limite escolhido foi o mais correto, sendo que em certos casos a diferença foi muito pequena. A simplicidade da assinatura e por consequente do código desenvolvido, explicam a alta precisão verificada.

5.1.3 Bússola

De modo a testar a bússola simples e com fusão de sensores foram feitas duas experiências: uma em que o telemóvel está no cimo de uma mesa, com superfície lisa, numa posição horizontal; e outra em que o utilizador está a segurar o telemóvel, com objetivo de analisar a interferência da movimentação do braço. Cada experiência foi repetida duas vezes, uma utilizando a bússola simples e outra com a fusão de sensores. No final, os valores do azimute calculados foram guardados num ficheiro, para análise posterior. Todos os testes foram realizados como o mesmo dispositivo, um *smartphone* Samsung Galaxy Note 8, em locais próximos, com menos de 1 m de distância. No primeiro teste, o dispositivo foi rodado no sentido dos ponteiros do relógio, começando com orientação norte até noroeste. No segundo teste, este foi segurado por uma pessoa, numa posição á altura do peito, sempre em orientação vertical. Um folha com a rosa-dos-ventos impressa e uma bússola real foram utilizados como referência, para determinar a posição real dos pontos cardeais em ambos os testes. As tabelas das figuras 5.5 e 5.6 demonstram os resultados das duas experiências. Os *outputs* em RAW dos ficheiros estão presentes no anexo C.1.2.

Observando a informação contida na tabela da figura 5.5, é possível observar que nas melhores condições, a bússola normal retorna um valor de orientação muito próximo do correto. De norte a sudeste, o valor do azimute cresce até próximo de π, sempre positivo, sendo que a partir do sul, o valor

Pontos cardeais	Azimute (*rad*)		
	Real	Normal	Com fusão de sensores
N	0	0,047725156	0,06915795
NE	0,785398163	0,889503	0,9212533
E	1,570796327	1,6814488	1,6620795
SE	2,35619449	2,3833117	2,4066088
S	-3,141592654	-3,114638	-2,6005502
SO	-2,35619449	-2,3305705	-2,3473856
O	-1,570796327	-1,6478666	-1,6357641
NO	-0,785398163	0,8370953	-0,80586314

Figura 5.5: Tabela com os resultados do teste realizado para a avaliação da bússola, simples (acelerómetro/magnetómetro) e com fusão de sensores (acelerómetro/magnetómetro e giroscópio), estando o dispositivo fixo numa mesa, sendo rodado para ficar orientado para os pontos cardeais.

Pontos cardeais	Azimute (*rad*)		
	Real	Normal	Com fusão de sensores
N	0	-0,07980154	-0,018328954
NE	0,785398163	0,69548875	0,7299832
E	1,570796327	1,5240033	1,5318103
SE	2,35619449	2,2349496	2,171878
S	-3,141592654	3,0731153	2,7590163
SO	-2,35619449	-2,4378886	-2,3067172
O	-1,570796327	-1,7165598	-1,6435639
NO	-0,785398163	-0,89886326	-0,87955093

Figura 5.6: Tabela com os resultados do teste realizado para a avaliação da bússola, simples (acelerómetro/magnetómetro) e com fusão de sensores (acelerómetro/magnetómetro e giroscópio), estando o utilizador a segurar o dispositivo, alterando a sua orientação conforme os pontos cardeais.

passa a negativo, dado que o valor do azimute aqui é π e $-\pi$, voltando a aumentar até perto de 0. Os valores do azimute registados não correspondem completamente aos valores reais nos pontos cardeais, pelo facto de a orientação do dispositivo não estar alinhada perfeitamente com o ponto cardeal, dado que os ajustes foi feitos manualmente, rodando o *smartphone*, por isso há que considerar o erro causado.

Comparando estes resultados com os que se obtêm com a fusão de sensores, verifica-se a mesma situação, com estimativas muito próximas dos valores reais, mas distintas do bússola normal. A única exceção é quando o dispositivo está apontado para sul, onde o erro é maior. Uma possível razão para isto pode dever-se ao método usado para corrigir o erro da transição entre π e $-\pi$, em conjunto com o posicionamento incorreto do dispositivo.

Na tabela da figura 5.6, observa-se quase a mesma coisa, existindo uma maior diferença entre as estimativas e o resultados reais, que podem dever-se à maior dificuldade em orientar o telemóvel para o ponto cardeal correto, quando o utilizador o está a segurar, tendo como referência uma imagem da rosa dos ventos no chão. Isto é mais provável do que ser derivado dos movimentos do utilizador, que não deve ter influência neste sensor, dado este ser de posição e não movimento. A situação observada com o sul mantém-se. No entanto, diminui um pouco a diferença com o valor real e dado que mesmo a bússola simples apresenta aqui um valor mais afastado, isto pode ser atribuído à situação descrita anteriormente. Note-se que esta situação deve ser analisada posteriormente, para que possa ser determinado o impacto e, se necessário, modificar a fusão dos dados, revertendo para o método

original, descrito em [30].

5.2 Avaliação do *Fingerprinting* - *Beacons* BLE

A recolha de dados efetuada para a fase de treino do *Fingerprinting*, descrita na secção 4.5.1, produziu uma enorme quantidade da dados, que foram utilizados para treinar e validar os modelos de *machine learning* criados. Foram gravados os valores do azimute, dos sensores e do RSSI medido para cada um dos *beacons* presentes no museu.

Nesta secção, é feita uma análise á variação da potência do sinal medido no espaço do museu, verificando o impacto da direção e dos próprios *beacons*, através de mapas de cobertura construídos para o efeito. Existem algumas situações em que não existe um valor para uma certa posição, o que poder ocorrer por duas razões: a direção naquele local não foi considerada, porque é improvável que um visitante esteja voltado para esse lado; ou o valor do RSSI medido ter sido 0.

5.2.1 Análise de cobertura

Impacto da direção do utilizador

Um aspeto que pode afetar o RSSI medido para alguns dos *beacons* é a direção do utilizador, dado que a presença deste entre o telemóvel e o dispositivo BLE pode obstruir e enfraquecer o sinal. Para analisar estas situações, observe-se os mapas seguintes, que apresentam a médias de todos os valores RSSI medidos para o *beacon* 1, nas várias posições no espaço do museu, com o utilizador voltado para a frente, na figura 5.7, para a direita, na figura 5.8, para trás, na figura 5.9, e para a esquerda, na figura 5.10. os mapas de cobertura dos outros *beacons* estão presentes no anexo C.2.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-63,86666667	-72,96666667		APPLE		OBS.	OBS.	-73	-82,96666667	15
14	-60,33333333	-66	-65,6	-68,1	-65,53333333	-66,6	-77,8	-86	-70	14
13	-62,36666667	-74	-62,5	-63,73333333	-69	-61,06666667	-69,93333333	-74,93333333	-72,33333333	13
12	-64,7	PCs		-60,8	-58,23333333	-69	-63,06666667	-68,76666667	-74,33333333	12
11	-67,46666667	-64,53333333	-75,1	-63,33333333	-62,13333333		-57,53333333	-59,23333333	OBS.	11
10	-75,3	-66	-71,06666667	-63,16666667	-65,23333333	REDES-A	-60,53333333	-54	-60,43333333	10
9	-64,4	-74,7		-62,4	-71		-63,63333333	-64,6	-61,86666667	9
8	-72,1	-68,73333333	COMPONETES	-65,9	-66,26666667		-65,4	-70,76666667	-81	8
7	-72	-72,53333333		-73,06666667	-75,26666667	REDES-B	-77,73333333	-74,53333333	-76	7
6	-61,86666667	-69,8	-77,16666667	-68,06666667	-71,4		-74,9	-71,9	-78,36666667	6
5	-67,46666667	-67,2	-67,06666667	-66	-71		-67,5	-74,86666667	-78,8	5
4	-80,43333333	-75,36666667	OBS.	-73,33333333	-71	REDES-C	-75,06666667	-75,73333333	-70,9	4
3	-75,33333333	-70,33333333		-71	-79,1		-74,6	-78,33333333	-69,8	3
2	-69,53333333	-70,26666667	-79	-77,53333333	-73	-70,36666667	-70,33333333	-74,76666667	-76,5	2
1	-71	-73,2	-70,6	-86,66666667	-77,93333333	-76	-74,7	-78,5	-76	1
0	-70,36666667	-71	n/d	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura 5.7: Mapa com a média dos valores RSSI registados para cada posição, com direção para a frente.

Analisando os vários mapas, é logo possível verificar a ocorrência dos fenómenos descritos em 2.1.2, inerentes à propagação em ambientes fechados. A presença do corpo humano entre o *beacon* e o telemóvel tem uma grande influência na diminuição da potência do sinal, mesmo em curtas distâncias, como por exemplo acontece no posição (14, 7), na figura 5.7.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-73,0333333	-70		APPLE		OBS.	OBS.	-72,1666667	-76	15
14	-75,6	-68,3	-70	-78,7333333	-72,8666667	-84	-69,8333333	-73,4	-76,9333333	14
13	-72	-67	-69,6666667	-71	-70,9	-77,1	-69,4333333	-67,6666667	-74,3333333	13
12	-71,4	PCs		-71,4666667	-75,1333333	-70,5333333	-67,3666667	-63	-76,3333333	12
11	-78	-68,9666667	-65,1	-67,3666667	-70,9333333		-81	-60,2666667	OBS.	11
10	-67,5	-69,3666667	-66,0333333	-64	-66,9333333	REDES-A	-70,8	-60,9333333	-72,1333333	10
9	-66	-70,9666667		-76,6666667	-73,2666667		-70,2	-67	-70,3333333	9
8	-67,7666667	-71,0666667	COMPONETES	-76,7333333	-77,7		-70,2666667	-67	-66,9	8
7	-72	-76		-71	-68,5333333	REDES-B	-70	-87	-73	7
6	-71,3333333	-75,2	-73,2333333	-70,2	-76,6		-71	-73,0333333	-75,2666667	6
5	-71,0666667	-73	-77	-70,8666667	-75,3333333		-71,2666667	-68	-85,1333333	5
4	-75,2	-77,8	OBS.	-71,2	-72,2	REDES-C	-71,9333333	-72,1	-80,2666667	4
3	-73,3333333	-75,9		-75,6666667	-79		-71	-76,2333333	-78,5333333	3
2	-75,4	-72,9666667	-75,2	-76,2333333	-72,1666667	-75,3666667	-72,9666667	-79,8666667	-77,8666667	2
1	-78,3	-78,6666667	-73,3333333	-79,2	-75,9333333	-82	-70,4333333	-70,8666667	-83	1
0	-75,2333333	-75	-79,7666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura 5.8: Mapa com a média dos valores RSSI registados para cada posição, com direção para a direita.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-72	-70,53333333		APPLE		OBS.	OBS.	-76,8	-69,66666667	15
14	-74	-72,3	-75,3	-76,76666667	-69,73333333	-72,73333333	-76,4	-69,56666667	-73,33333333	14
13	-90	-71,86666667	-69	-74	-72,2	-83	-73	-69	-69,2	13
12	-68,46666667	PCs		-64	-65,7	-73,26666667	-66,03333333	-71,2	-64,46666667	12
11	-73	-75,53333333	-73,26666667	-66,4	-67,93333333		-70	-65,36666667	OBS.	11
10	-71,13333333	-75,53333333	-75,93333333	-77,3	-63	REDES-A	-62,76666667	-68,8	-68,4	10
9	-70,83333333	-68		-67	-66,2		-78,63333333	-72,46666667	-77	9
8	-77,5	-65,06666667	COMPONETES	-71,13333333	-75,13333333		-78,4	-88	-73	8
7	-80,9	-71,3		-69,9	-73,93333333	REDES-B	-76	-74,53333333	-71,6	7
6	-69,73333333	-70,6	-68,06666667	-69,06666667	-70,26666667		-74,8	-74,66666667	-81,53333333	6
5	-71,5	-70	-75	-70	-74,06666667		-85,73333333	-72,2	-81	5
4	-75,53333333	-78,53333333	OBS.	-73,33333333	-77,96666667	REDES-C	-69,8	-68,4	-76	4
3	-75,36666667	-72		-82	-83,36666667		-76,2	-82,23333333	-75,2	3
2	-73,06666667	-74,4	-69,83333333	-71,86666667	-76,86666667	-73,4	-76,76666667	-80,13333333	-82,03333333	2
1	-74,46666667	-77	-69	-79	-80	-76	-77,46666667	-77	-76,26666667	1
0	-83,5	-75,06666667	n/d	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura 5.9: Mapa com a média dos valores RSSI registados para cada posição, com direção para trás.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-73,0333333	-70		APPLE		OBS.	OBS.	-72,1666667	-76	15
14	-75,6	-68,3	-70	-78,7333333	-72,8666667	-84	-69,8333333	-73,4	-76,9333333	14
13	-72	-67	-69,6666667	-71	-70,9	-77,1	-69,4333333	-67,6666667	-74,3333333	13
12	-71,4	PCs		-71,4666667	-75,1333333	-70,5333333	-67,3666667	-63	-76,3333333	12
11	-78	-68,9666667	-65,1	-67,3666667	-70,9333333		-81	-60,2666667	OBS.	11
10	-67,5	-69,3666667	-66,0333333	-64	-66,9333333	REDES-A	-70,8	-60,9333333	-72,1333333	10
9	-66	-70,9666667		-76,6666667	-73,2666667		-70,2	-67	-70,3333333	9
8	-67,7666667	-71,0666667	COMPONETES	-76,7333333	-77,7		-70,2666667	-67	-66,9	8
7	-72	-76		-71	-68,5333333	REDES-B	-70	-87	-73	7
6	-71,3333333	-75,2	-73,2333333	-70,2	-76,6		-71	-73,0333333	-75,2666667	6
5	-71,0666667	-73	-77	-70,8666667	-75,3333333		-71,2666667	-68	-85,1333333	5
4	-75,2	-77,8	OBS.	-71,2	-72,2	REDES-C	-71,9333333	-72,1	-80,2666667	4
3	-73,3333333	-75,9		-75,6666667	-79		-71	-76,2333333	-78,5333333	3
2	-75,4	-72,9666667	-75,2	-76,2333333	-72,1666667	-75,3666667	-72,9666667	-79,8666667	-77,8666667	2
1	-78,3	-78,6666667	-73,3333333	-79,2	-75,9333333	-82	-70,4333333	-70,8666667	-83	1
0	-75,2333333	-75	-79,7666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura 5.10: Mapa com a média dos valores RSSI registados para cada posição, com direção para a direita.

O *beacon* está a pouco mais de 3 quadrados de distância deste local, no entanto, o valor RSSI registado é o mais baixo naquela zona. Comparando este caso com os outros mapas, em que a direção do utilizador não causa a obstrução do sinal, o valor medido está sempre na casa dos -70 dBm. A projeção do sinal pode explicar o porquê da potência recebida ser inferior a outras posições. Esta

observação pode ser feita sobre outros locais, e em diferentes direções. Em alguns casos, a obstrução que ocorre é compensada pelo efeito da propagação multi-caminho, dado que o limite superior da sala é de vidro, e o sinal é refletido.

A direção tem de facto impacto no valor RSSI medido, no entanto, existem múltiplos aspetos e características do local, bem como dos próprios dispositivos *Bluetooth*, que podem influenciar os resultados observados.

Impacto da localização dos *beacons*

Nos mapas apresentados na secção anterior, foi possível verificar a influência de vários aspetos na cobertura do *beacon* 1.

Na secção seguinte pretende-se avaliar a cobertura de cada *beacon* no espaço do museu, considerando uma dada direção. O objetivo é averiguar se é possível mapear a cobertura em áreas específicas do museu.

Cobertura por *beacons* – Os resultados apresentados em seguida consideram o caso em que o utilizador está voltado para a frente. Os mapas seguintes apresentam as médias de todos os valores RSSI medidos para o *beacon* 2, na figura 5.11, para o *beacon* 3, na figura 5.12, e para o *beacon* 4, na figura 5.13.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-72,13333333	-71		APPLE		OBS.	OBS.	-72,9	-74	15
14	-76,13333333	-74	-73,2	-66,43333333	-64,2	-70,3	-69,66666667	-74,33333333	-76,33333333	14
13	-70	-73,86666667	-69,73333333	-69,66666667	-72	-70	-69,93333333	-70,06666667	-72,66666667	13
12	-67,53333333	PCs		-76,13333333	-72,66666667	-77,33333333	-69,26666667	-76	-72,76666667	12
11	-70,13333333	-73,2	-70,83333333	-73,73333333	-76,9		-70,1	-74,4	OBS.	11
10	-70,4	-68	-79,4	-77,06666667	-79,86666667	REDES-A	-74,23333333	-69,96666667	-74,13333333	10
9	-74,2	-69,53333333		-74,73333333	-75,46666667		-68,2	-80,4	-87,26666667	9
8	-72,73333333	-76,8	COMPONETES	-76	-75,2		-72,03333333	-75,36666667	-75,66666667	8
7	-70,8	-86		-76,8	-75,46666667	REDES-B	-69,73333333	-78	-83,6	7
6	-74,3	-79	-74,43333333	-76,2	-79		-78,76666667	-78,43333333	-78,53333333	6
5	-71,26666667	-73,6	-73,26666667	-74,93333333	-78,13333333		-76,76666667	-75,73333333	-76,4	5
4	-90,5	-75,06666667	OBS.	-79,83333333	-82	REDES-C	-79,9	-78,03333333	-80,13333333	4
3	-77	-73,5		-81	-76,23333333		-74,43333333	-82,36666667	-74,8	3
2	-72,93333333	-73,36666667	-71	-75,1	-81,86666667	-77,16666667	-83,56666667	-75,96666667	-76,2	2
1	-82,66666667	-77,03333333	-82,53333333	-76,03333333	-75,06666667	-73	-76,83333333	-77,53333333	-80	1
0	-78,63333333	-74,56666667	n/d	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura 5.11: Mapa com a média dos valores RSSI do *beacon* 2, registados para cada posição, com direção para a frente.

Comparando os mapas dos quatro *beacons*, é possível verificar que o 1 é aquele que tem melhor cobertura num maior alcance, sendo registados valores RSSI similares tanto nas posições mais próximas, como no outro extremo da sala. Isto pode dever-se á proximidade deste *beacon* da parede de vidro, onde o sinal é refletido, tal como descrito anteriormente.

Cobertura global e posicionamento – A cobertura de todos os *beacons* pode ser melhor observada no mapa da figura 5.14, que apresenta com base nas tabelas acima, para cada posição, qual o dispositivo BLE que regista, em média, o maior valor RSSI, quando o utilizador está voltado para a

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-69,46666667	-68,2		APPLE		OBS.	OBS.	-64,83333333	-63	15
14	-66,73333333	-72,06666667	-75	-70,8	-70,4	-74	-68,06666667	-72,5	-69,8	14
13	-72	-67	-70,43333333	-72	-68,03333333	-67,66666667	-66,46666667	-73,7	-76,36666667	13
12	-72,23333333	PCs		-67,6	-68,13333333	-70,63333333	-64,06666667	-72,2	-67,13333333	12
11	-70	-69,13333333	-76,8	-62,6	-75,33333333		-68,86666667	-71,4	OBS.	11
10	-75,36666667	-68,2	-68	-63	-73,46666667	REDES-A	-67,66666667	-67	-69,16666667	10
9	-68	-68		-62,5	-65,73333333		-62,26666667	-81	-64,33333333	9
8	-68,06666667	-72,73333333	COMPONETES	-65,56666667	-61,1		-63,3	-65	-55	8
7	-64	-66		-65,2	-68,46666667	REDES-B	-73,63333333	-63	-58,86666667	7
6	-64,33333333	-68,06666667	-62,8	-68,73333333	-64,36666667		-61	-65,26666667	-71,76666667	6
5	-69,93333333	-61	-62	-75,93333333	-59,4		-63,8	-60	-56,13333333	5
4	-83,53333333	-61,1	OBS.	-68,86666667	-64,06666667	REDES-C	-60,1	-56,86666667	-70,1	4
3	-68,86666667	-69,2		-61,96666667	-60,8		-58,93333333	-60,86666667	-61,76666667	3
2	-71,06666667	-70,16666667	-68,66666667	-72	-56,73333333	-64,43333333	-60,13333333	-62,13333333	-59,4	2
1	-76,06666667	-73,6	-63,36666667	-68,33333333	-59,2	-67,06666667	-53,9	-76,2	-66	1
0	-74,93333333	-67	-67,86666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura 5.12: Mapa com a média dos valores RSSI do *beacon* 3, registados para cada posição, com direção para a frente.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-65,06666667	-68,6		APPLE		OBS.	OBS.	-85	-70	15
14	-71,9	-66,63333333	-66	-68,33333333	-76	-75	-81,46666667	-77,96666667	-72	14
13	-67,73333333	-69	-64,9	-66	-76	-75,76666667	-72,2	-73,2	-77,76666667	13
12	-65,13333333	PCs		-73,2	-76	-71,16666667	-73,56666667	-76,6	-83,66666667	12
11	-65	-67,46666667	-73,4	-66,2	-72,56666667		-69	-77,8	OBS.	11
10	-60,6	-63	-62,16666667	-84	-70,33333333	REDES-A	-81,13333333	-80	-77,46666667	10
9	-84	-66		-71,3	-75,56666667		-73,1	-76	-78,46666667	9
8	-63,7	-69,2	COMPONETES	-68,76666667	-73		-75,33333333	-76	-70	8
7	-61,2	-74		-74,1	-70,83333333	REDES-B	-73	-76	-77,13333333	7
6	-65,53333333	-78,23333333	-75,73333333	-70	-72		-71	-76	-79,26666667	6
5	-76,7	-76,63333333	-70	-73,33333333	-72		-76,13333333	-76	-74,06666667	5
4	-65,46666667	-67,8	OBS.	-70,63333333	-77,26666667	REDES-C	-88	-79,93333333	-79,83333333	4
3	-65,6	-62		-67	-69,56666667		-68,96666667	-73,83333333	-71,13333333	3
2	-66,2	n/d	-71,63333333	-67	-73,03333333	-75,9	-74,53333333	-93	-74,03333333	2
1	-76	n/d	-69,66666667	-74,23333333	-69	-72,53333333	-75,8	-75,26666667	-70	1
0	-76	n/d	-70,66666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura 5.13: Mapa com a média dos valores RSSI do *beacon* 4, registados para cada posição, com direção para a frente.

frente. Neste mapa, verifica-se a conclusão anterior para o *beacon* 1, que regista uma melhor cobertura, nomeadamente na parte frontal do museu. O *beacon* 2 é irrelevante neste contexto, ocorrendo sobreposição dos *beacons* 1,3 e 4, na parte traseira da sala, onde o 3 domina.

Todavia, analisando os mapas para as outras direções, a situação é diferente. Começando pelo mapa da figura 5.15, em que o utilizador estava voltado para a direita, o *beacon* 3 regista melhor cobertura na maior parte do museu, com exceção do canto superior esquerdo, onde o 4 domina. Tal como o *beacon* 1 na direção frontal, também aqui a cobertura do 3 é aumentada pelo reflexo das janelas no limite direito da sala. O *beacon* 2 continua irrelevante, e a melhor cobertura do *beacon* 1 é dispersa por vários por várias locais.

Nas outras direções, cujos mapas estão presentes nas figuras 5.16, para a direção para trás, e 5.17, para a direção para a esquerda, o *beacon* que regista a melhor cobertura no museu, no entanto, mais dispersa. O *beacon* 4 tem uma cobertura ligeiramente inferior quando comparada com o 3, especialmente na parte direita do espaço. O *beacon* 2 continua relativamente irrelevante, sendo que a sua melhor cobertura verifica-se em mais posições, e a situação com o 1 mantém-se.

Y/X	0	1	2	3	4	BLE 2	6	7	8	X/Y
15	1	2		APPLE		OBS.	OBS.	3	2	15
14	1	1	1	2	2	1	2	4	1	14
13	1	4	1	1	1	1	1	2	3	13
12	1	PCs		1	1	1	1	1	2	12
11	1	1	2	1	1		1	1	OBS.	BLE1
10	3	1	4	1	1	REDES-A	1	1	1	10
9	1	2		1	4		1	1	1	9
8	3	3	COMPONETES	3	1		1	1	4	8
7	3	4		3	4	REDES-B	3	4	3	7
6	1	4	2	4	4		4	3	4	6
BLE4	4	1	3	3	4		1	3	3	5
4	4	4	OBS.	3	1	REDES-C	3	4	3	4
3	3	3		3	3		4	3	3	3
2	3	4	3	4	1	1	4	3	4	2
1	3	3	3	3	3	3	3	3	3	1
0	1	3	4	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	BLE3 6	7	8	X/Y

Figura 5.14: Mapa com o *beacon* que regista a maior média para o valor de RSSI, em cada posição, com direção para a frente.

Y/X	0	1	2	3	4	BLE 2	6	7	8	X/Y
15	4	3	1	4	1	4	4	3	3	15
14	3	4	4	4	3	2	3	3	3	14
13	4	1	4	4	3	3	3	1	2	13
12	4	PCs		3	3	1	3	1	3	12
11	4	4	1	3	1		3	1	OBS.	BLE1
10	4	4	4	3	1	REDES-A	3	1	3	10
9	1	4		3	3		3	1	3	9
8	4	4	COMPONETES	3	3		3	3	3	8
7	4	3		3	3	REDES-B	1	3	3	7
6	3	3	3	3	3		3	3	3	6
BLE4	3	3	3	1	3		3	3	3	5
4	4	3	OBS.	3	3	REDES-C	3	3	3	4
3	4	4		3	3		3	3	3	3
2	4	3	3	4	3	1	3	3	3	2
1	4	3	3	3	3	3	3	1	3	1
0	3	3	3	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	BLE3 6	7	8	X/Y

Figura 5.15: Mapa com o *beacon* que regista a maior média para o valor de RSSI, em cada posição, com direção para a direita.

Observações finais

Observando o conjunto de mapas, é possível verificar que, não é viável proceder à localização usando apenas os *beacons* existentes e nas posições em que estes se encontram colocados.

O efeito da propagação multi-caminho, a obstrução de certos elementos do espaço, ou mesmo do próprio utilizador, e a interferência entre os sinais, tem um impacto enorme na potência do sinal. Para além disto, outros aspetos como a altura a que foram colocados os *beacons*, ou mesmos as posições, que teoricamente aparentam ser as mais corretas, podem ter influência nos resultados finais.

Y/X	0	1	2	3	4	BLE 2	6	7	8	X/Y
15	4	4	1	4	1	4	4	2	2	15
14	3	4	2	2	1	4	3	2	2	14
13	3	4	1	4	2	3	1	3	3	13
12	4	PCs		1	3	4	1	4	1	12
11	4	3	3	1	4		3	1	OBS.	BLE1
10	4	1	4	4	1	REDES-A	1	1	1	10
9	4	4		1	1		4	3	3	9
8	3	1	COMPONETES	4	2		3	4	4	8
7	3	1		1	4	REDES-B	3	3	3	7
6	4	1	3	4	4		3	3	4	6
BLE4	4	3	3	1	4		3	3	4	5
4	3	4	OBS.	4	4	REDES-C	3	3	3	4
3	3	1		4	3		3	4	3	3
2	1	3	1	3	4	3	3	3	3	2
1	3	1	1	3	4	4	3	3	3	1
0	4	1	4	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	BLE3 6	7	8	X/Y

Figura 5.16: Mapa com o *beacon* que regista a maior média para o valor de RSSI, em cada posição, com direção para trás.

Y/X	0	1	2	3	4	BLE 2	6	7	8	X/Y
15	4	4	1	4	1	4	4	2	2	15
14	3	3	2	2	2	2	2	1	2	14
13	3	3	4	2	3	1	2	2	1	13
12	1	PCs		3	3	1	2	3	1	12
11	4	4	3	2	4		2	2	OBS.	BLE1
10	3	3	3	1	4	REDES-A	3	3	1	10
9	4	4		2	4		3	2	1	9
8	4	4	COMPONETES	3	1		1	3	1	8
7	4	3		3	3	REDES-B	1	1	2	7
6	4	4	3	3	4		1	1	1	6
BLE4	4	4	2	1	1		3	1	3	5
4	4	1	OBS.	3	3	REDES-C	3	3	3	4
3	4	4		3	3		3	3	3	3
2	4	4	3	4	3	4	3	3	4	2
1	3	4	4	3	3	3	3	3	4	1
0	4	4	3	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	BLE3 6	7	8	X/Y

Figura 5.17: Mapa com o *beacon* que regista a maior média para o valor de RSSI, em cada posição, com direção para a esquerda.

5.3 Avaliação da Estimativa de posição (Trajetória e *Fingerprinting*)

Embora o sistema não tenha sido terminado, nomeadamente a parte relacionada com o *Fingerprinting*, foi possível avaliar a estimativa de posição das duas técnicas, e determinar se de facto é possível corrigir o erro no cálculo do trajeto com as previsões do modelo de aprendizagem automática. Nesta secção é descrita a metodologia utilizada na avaliação, seguida dos resultados obtidos e da respetiva análise.

5.3.1 Metodologia

Para realizar a experiência, o protótipo desenvolvido, com o posicionamento com base na trajetória completo, foi modificado para guardar todos os valores do RSSI medido para os *beacons*, as medidas dos sensores e o valor do azimute, num ficheiro. A interface foi modificada que a posição calculada apenas fosse exibida a pedido do utilizador, sendo esta guardada num ficheiro, em conjunto com as coordenadas reais, inseridas anteriormente. Cada registo no ficheiro tem a seguinte forma:

```
Tipo; Tempo; posição real (x,y, direção); posição estimada (x,y, direção); RSSI Beacon 1,
..., Beacon 4; x,y,z do acelerómetro; x,y,z do giroscópio; x,y,z do magnetómetro; azimute
```

Existem quatro tipos de registos:

- SET, que indica quando e o valor do azimute definido para a direção frontal do museu;
- RAW, que indica a ocorrência de novos valores para o RSSI ou para um dos sensores;
- REQ, que indica um pedido de estimativa por parte do utilizador, com as coordenada atuais;
- RESP, que indica a resposta ao pedido feito, com a posição estimada com base na trajetória.

De notar que os campos que não são atualizados são colocados a 0. Através de um programa de simulação, é possível recriar o percurso efetuado, utilizando os registos presentes no ficheiro para fazer previsões com um modelo, e comparar estas com as estimativas baseadas na trajetória, que foram feitas em tempo real durante a execução da aplicação.

Foram realizados três percursos diferentes pelo museu, com o mesmo telemóvel e nas mesmas condições que outras experiências, sendo que, começando com a posição (0,0) da entrada, a cada duas quadrículas foi feito um pedido de estimativa. As figuras 5.18, 5.19 e 5.20, ilustram estes percursos, indicando todos os locais onde foram feitos os pedidos, sendo estes identificados sequencialmente com as letras do alfabeto.

5.3.2 Resultados

As tabelas das figuras 5.21, 5.23 e 5.25, e os mapas das figuras 5.22, 5.24 e 5.24, apresentam os resultados das estimativas feitas com as duas técnicas, em comparação com as posições reais, para o percurso 1, 2 e 3, respetivamente.

Começando com a posição baseada na trajetória, é possível observar erros de estimativa logo no segundo pedido feito nos percursos 1 e 2, no ponto (1,2), em que se observa um erro de 1 quadrado. De notar que aqui o erro é calculado em função do deslocamento desde o local do último pedido. Esta situação aparenta estar relacionada com a curva realizada entre a posição inicial e esta, e que ocorre em todas as outras viragens. O problema pode estar ligado aos passos dados aquando da viragem, que são interpretados pelo sistema como um deslocamento, quando de facto servem apenas para a mudança de direção. Durante os deslocamentos em linha reta também ocorrem erros de posicionamento, que se devem ao constante parar e iniciar da caminhada, sendo que o balançar da perna na

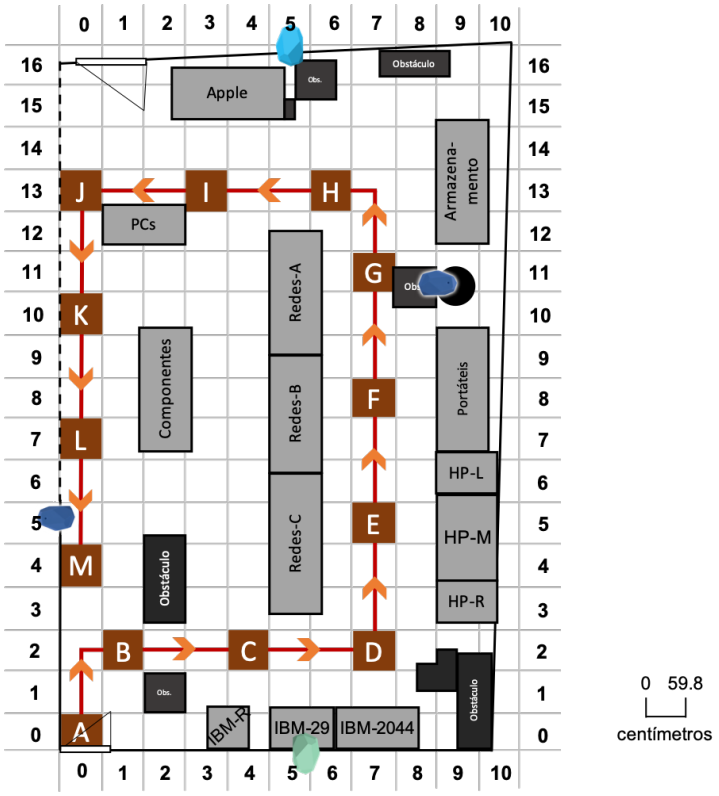


Figura 5.18: Ilustração do percurso 1.

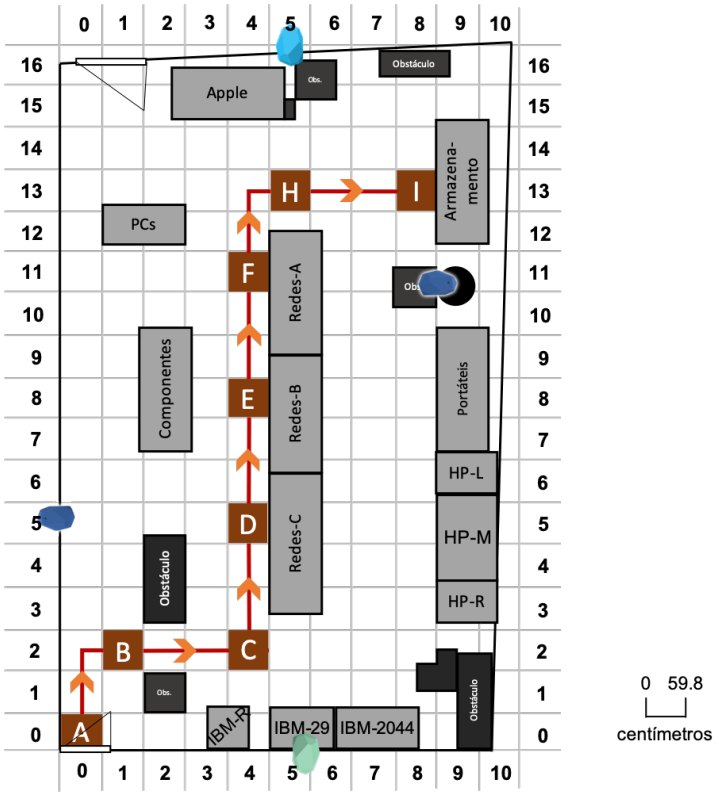


Figura 5.19: Ilustração do percurso 2.

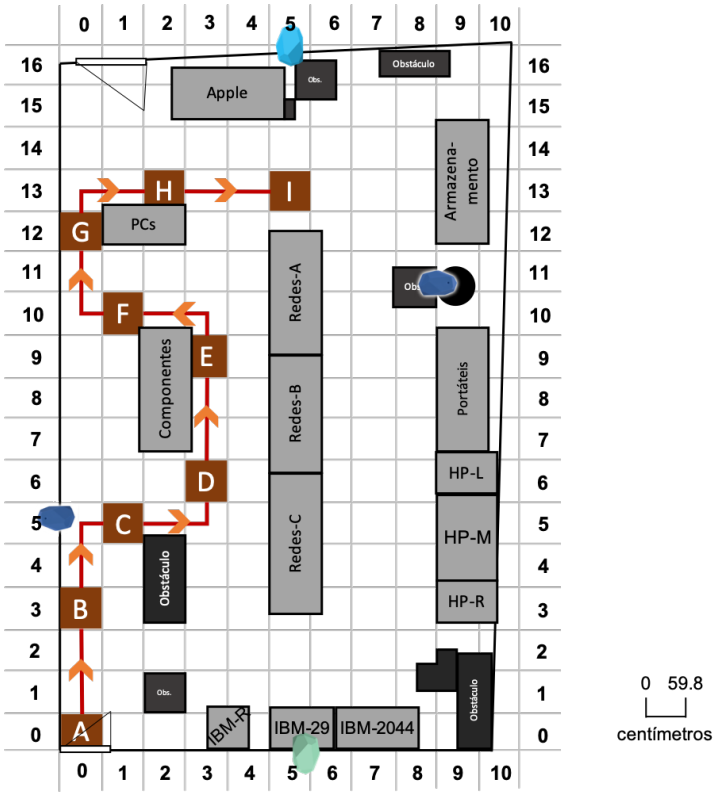


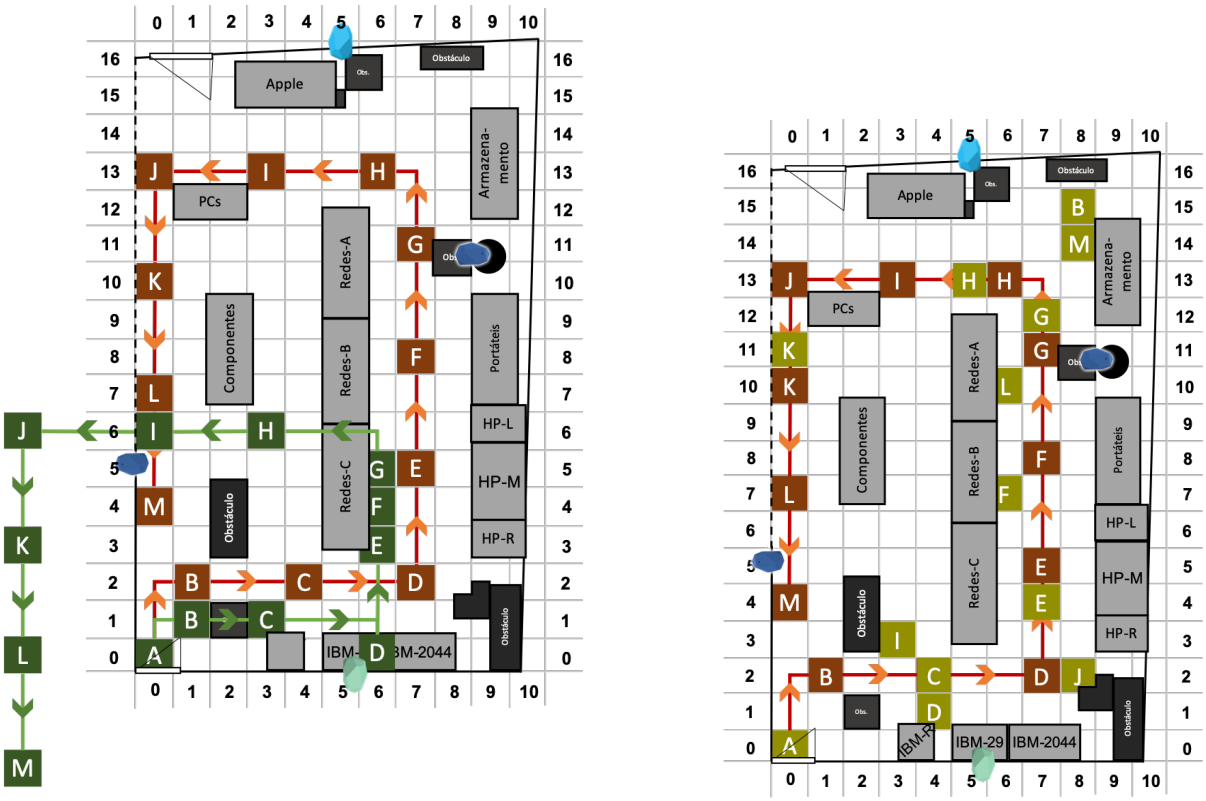
Figura 5.20: Ilustração do percurso 3.

ID	Posição real			Posição com base na trajetória				Posição com base no fingerprinting				
	X	Y	Orientação	X	Y	Orientação	Erro (quadículas)	X	Y	Orientação	Erro (quadrículas)	Atraso (*ms*)
A	0	0	F	0	0	F	0	0	0	F	0	67
B	1	2	R	1	1	R	1	8	15	R	14,76482306	3
C	4	2	R	3	1	B	1	4	2	R	0	2
D	7	2	R	6	0	R	1	4	1	R	3,16227766	3
E	7	5	F	6	3	R	0	7	4	F	1	2
F	7	8	F	6	4	R	2	6	7	F	1,414213562	3
G	7	11	F	6	5	F	2	7	12	F	1	2
H	6	13	L	3	6	L	2,236067977	5	13	L	1	2
I	3	13	L	0	6	L	0	3	3	L	10	2
J	0	13	L	-3	6	L	0	8	2	L	13,60147051	2
K	0	10	B	-3	3	B	0	0	11	B	1	3
L	0	7	B	-3	0	B	0	6	10	B	6,708203932	2
M	0	4	B	-3	-3	B	0	8	14	B	12,80624847	2

Figura 5.21: Tabela com os resultados do percurso 1.

passada durante estas alturas é menor, e por consequente a amplitude do máximo estará abaixo do limite definido. Também é possível a ocorrência de falsos positivos em alguns instantes. No entanto, estes casos apenas podem ser verificados em tempo real, tendo em atenção ao número de passos detetados. Ainda assim, regista-se normalmente um erro de apenas um quadrado, com algumas exceções, em que o erro atinge um valor igual ou superior a 2.

Na posicionamento baseado no *Fingerprinting*, a situação é diferente. A posição só é determinada nestes pontos, ao contrário do que acontece com base na trajetória. Em praticamente todas as previsões existe erro, sendo que, na maior parte dos casos este é bastante elevado, existindo locais em que é superior a 10 quadrículas. Estas situações são algo frequentes, tendo ocorrido pelo menos uma



(a) Baseado na trajetória.

(b) Baseada em *Fingerprinting*.

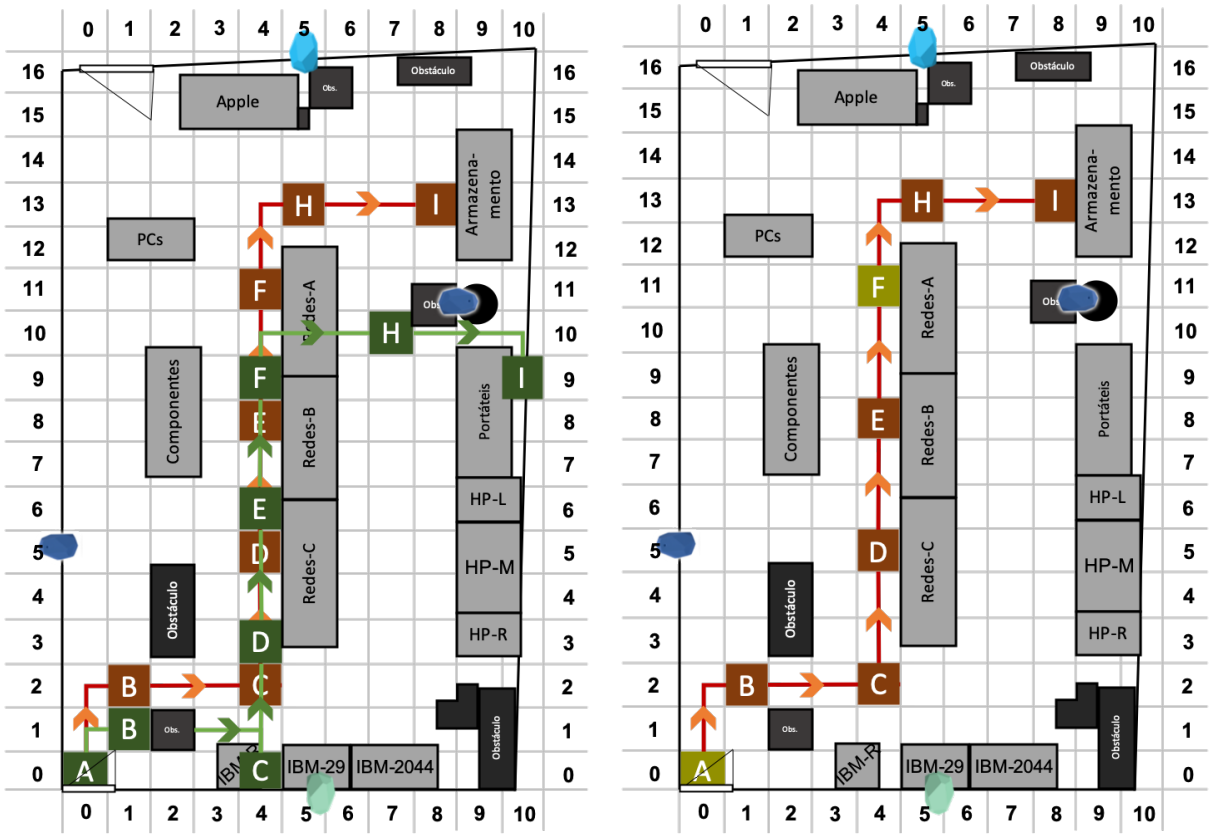
Figura 5.22: Comparação entre as posições reais do percurso 1 e as estimadas com base nas duas técnicas de localização.

ID	Posição real			Posição com base na trajetória				Posição com base no fingerprinting				
	X	Y	Orientação	X	Y	Orientação	Erro (quadículas)	X	Y	Orientação	Erro (quadrículas)	Atraso (*ms*)
A	0	0	F	0	0	F	0	0	0	F	0	55
B	1	2	R	1	1	B	1	4	1	R	3,16227766	3
C	4	2	R	4	0	B	1	4	1	R	1	2
D	4	5	F	4	3	R	0	3	4	F	1,414213562	2
E	4	8	F	4	6	F	0	5	12	F	4,123105626	2
F	4	11	F	4	9	F	0	4	11	F	0	2
G	5	13	R	7	10	R	2,236067977	8	14	R	3,16227766	3
H	8	13	R	10	9	B	1	5	1	R	12,36931688	3

Figura 5.23: Tabela com os resultados do percurso 2.

vez em todos os percursos, ficando a previsão feita pelo modelo fica num local oposto á posição real. O único elemento das coordenadas que esta técnica consegue estimar corretamente é a direção do utilizador, no momento ao pedido, que se deve ao uso dos dados do magnetómetro, bem como do valor do azimute, nas *fingerprints* criadas.

Os resultados obtidos podem dever-se a várias razões: a possível invalidade das *fingerprints* de treino, dado que entre a fase de treino e a realização do percurso houve um intervalo de quase dois meses, em que novas peças foram colocadas no museu; a propagação do sinal no espaço, que, tal como descrito na secção anterior, dificulta a utilização dos *beacons* para efeitos de localização.



(a) Baseado na trajetória.

(b) Baseada em *Fingerprinting*.

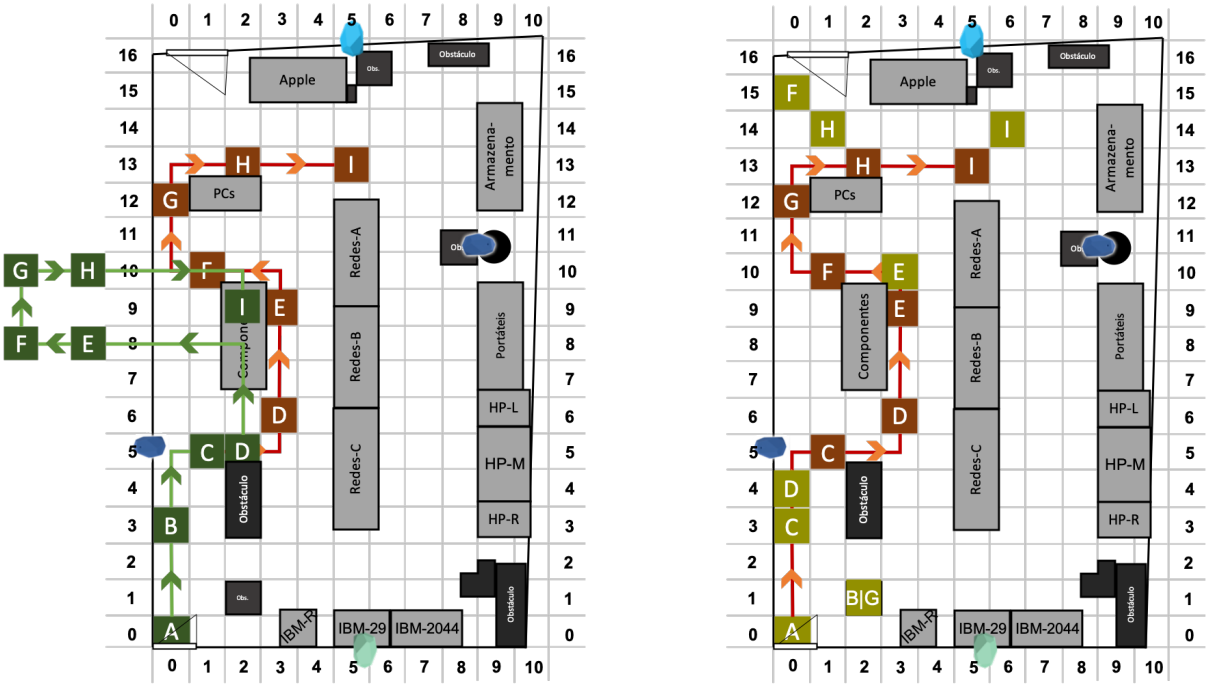
Figura 5.24: Comparação entre as posições reais do percurso 2 e as estimadas com base nas duas técnicas de localização.

ID	Posição real			Posição com base na trajetória				Posição com base no fingerprinting				
	X	Y	Orientação	X	Y	Orientação	Erro (quadículas)	X	Y	Orientação	Erro (quadrículas)	Atraso (*ms*)
A	0	0	F	0	0	F	0	0	0	F	0	104
B	0	3	F	0	3	F	0	2	1	F	2,828427125	4
C	1	5	R	1	5	R	0	0	3	R	2,236067977	12
D	3	6	F	2	5	R	1,414213562	0	4	F	3,605551275	3
E	3	9	F	-1	8	F	3	3	10	F	1	5
F	1	10	L	-3	8	L	1	0	15	L	5,099019514	4
G	0	12	F	-3	10	F	1	2	1	F	11,18033989	3
H	2	13	R	-1	10	R	1	1	14	R	1,414213562	4
I	5	13	R	2	9	R	1	6	14	R	1,414213562	3

Figura 5.25: Tabela com os resultados do percurso 3.

5.4 Síntese final

Durante a implementação dos vários elementos do módulo do Calculo da trajetória no protótipo desenvolvido, estes foram avaliados e testados a cada nova iteração, de modo a validar a lógica utilizada. Os resultados aqui apresentados representam a avaliação final destes elementos, onde aspetos como a eficácia a precisão são verificados. O Pedómetro apresentou uma alta precisão, com um baixo número de falsos positivos e negativos. A Deteção de curvas provou conseguir identificar todas as viragens e determinar corretamente a sua direção. A Bússola demonstrou bons resultados, ocorrendo alguma imprecisão nas orientações a sul.



(a) Baseado na trajetória.

(b) Baseada em *Fingerprinting*.

Figura 5.26: Comparação entre as posições reais do percurso 3 e as estimadas com base nas duas técnicas de localização.

Após a finalização do Posicionamento com base na trajetória, a técnica de *fingerprinting* foi selecionada para ser utilizada na correção do erro da estimativa calculada a partir do deslocamento. Para o efeito, realizou-se uma recolha de dados no espaço do museu, os quais foram tratados estatisticamente para criar os mapas apresentados. Observando os resultados, verificou-se que a propagação do sinal dos *beacons* é incerta, devido a vários fatores, relacionados com as condições do espaço, bem como a obstrução causada pelo próprio utilizador.

A avaliação final á Estimativa da posição consistiu numa experiência com três percursos distintos, sendo que em certas posições, foi gravada a respetiva estimativa com base na trajetória, e mais tarde, através da simulação dos percursos, e usando os dados registados, foi também feita uma estimativa com base em *Fingerprinting*. Os resultados obtidos com ambas as técnicas ficaram aquém do esperado. O posicionamento com base na trajetória, apesar do erro na estimação, demonstrou conseguir calcular a deslocação do utilizador entre estes locais, surgindo problemas aquando das curvas. Já o posicionamento com base em *fingerprinting*, demonstrou na maior parte das situações um erro relativamente elevado para localização interior.

Capítulo 6

Conclusão

6.1 Síntese

O objetivo principal desta tese foi a criação de um sistema de navegação e localização, que sirva como base para a implementação futura de um sistema interativo no Museu da Computação.

Para tal, em primeiro lugar, foi feita pesquisa sobre o estado de arte, que abordou várias técnicas e tecnologias para localização (com recurso a redes sem fios, sensorização simples e múltipla) e sistemas de museus interativos, de modo a formular uma ideia do que já existe nestes ramos.

Com base nos conceitos mais relevantes, estabeleceu-se uma visão inicial para a solução, a qual requereu uma análise e caracterização do movimento e da trajetória, dos quais depende o posicionamento.

A arquitetura do sistema foi então definida, incluindo dois elementos principais, o Cliente, responsável por estimar a localização atual com base nos dados dos sensores e *beacons*, e o Servidor, responsável por determinar uma estimativa que será usada para a correção do erro.

Protótipos foram então desenvolvidos, implementado os vários elementos definidos na arquitetura, servindo para testar a viabilidade do sistema de navegação no museu.

Cada um dos elementos implementados foi testado individualmente para garantir a sua validade. Também foi analisada a cobertura dos *beacons* no espaço do museu. Por último, foi feita uma avaliação final á estimativa de posição, com base na trajetória e no *Fingerprinting*, para verificar a precisão e eficácia do sistema final.

6.2 Conclusões

O resultado deste trabalho é um sistema de localização, cujo desenvolvimento e avaliação foram aqui especificados. Este sistema destaca-se de outros por utilizar duas técnicas, *Dead Reckoning* e *Fingerprinting*, de modo complementar, com o objetivo de obter uma maior precisão de posicionamento. Para além disso, nos casos em que, por variadas razões, não seja possível usar as medidas dos sensores, é possível recorrer á outra técnica para estimar a posição.

A localização é determinada de duas formas diferentes: com base na trajetória e com base em *Fingerprinting*. Na primeira, as medidas dos sensores são processadas e interpretadas, para identificar a trajetória tomada, e com base nesta informação, determinar a posição do utilzador. Na segunda, a localização é prevista, comparando as *fingerprints* para cada uma das posições (e direções), recolhidas numa fase de treino, com aquela observada pelo cliente.

Para testar a solução, foram dois criados protótipos que implementam este sistema. A avaliação final foi feita através de vários percursos no espaço do museu, sendo a estimativa baseada no trajeto calculada em tempo real, localmente, e a estimativa baseada em *Fingerprinting* feita posteriormente, através da simulação do percurso.

Os resultados demonstraram, para os três percursos, um erro médio de deslocamento de 0.767 quadrículas para o posicionamento com base na trajetória, e um erro médio de estimativa de 3.821 quadrículas para o posicionamento com base em *Fingerprinting*. Estes erros, apesar de relativamente baixos, são relevantes no contexto de localização interior, onde é necessária uma maior precisão. Ainda assim, ambas estas técnicas demonstram potencial, que pode ser trabalhado mais profundamente no futuro, de modo a obter melhores resultados.

6.3 Trabalho futuro

Os resultados da avaliação permitiram concluir que existe um aumento do erro de deslocamento logo após uma curva de 90º. Isto demonstra que é necessária uma melhor integração e tratamento das informações vindas dos sub módulos de cálculo da trajetória.

Em relação ao *Fingerprinting*, deve ser estudada uma nova abordagem, que pode ou não recorrer a *machine learning* para a sua implementação. Outro aspeto aqui a analisar é a colocação dos *beacons*, nomeadamente a quantidade e a localização destes dispositivos. Uma hipótese é a colocação destes no teto, o que em principio mitigaria o impacto da direção do utilizador e da localização dos *beacons*.

Finalmente, outras técnicas complementares também devem ser exploradas, nomeadamente a sensorização simples (que foi originalmente idealizada para esta solução), com o intuito de melhorar a precisão.

Bibliografia

[1] ICOM. Museum Definition. URL https://icom.museum/en/activities/standards-guidelines/museum-definition/. Online; acedido a 7 de dezembro de 2018.

[2] H. J. Q. Z. Y. C. S. L. X. Zhenghua Chen, Han Zou. Fusion of Wifi, Smartphone Sensors and Landmarks using the Kalman Filter for Indoor Location. *MDPI*, 2015. URL https://escholarship.org/content/qt8sh1p133/qt8sh1p133.pdf.

[3] G. D. F. L. M. V. M. L. P. G. S. Stefano Alletto, Rita Cucchiara. An Indoor Location-Aware System for an IoT-Based Smart Museum. *IEEE Internet of Things Journal*, 2016. URL https://ieeexplore.ieee.org/abstract/document/7348638.

[4] J. L. K.-H. K. Alex Mariakakis, Souvik Sen. SAIL: Single Access Point-Based Indoor Localization. *MobySys '14*, 2014. URL https://www.sigmobile.org/mobisys/2014/pdfMainConference/sys284-mariakakisP.pdf.

[5] F. P. Angelo Chianese. Designing a smart museum: when Cultural Heritage joins IoT. *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, 2014. URL https://ieeexplore.ieee.org/document/6982932.

[6] V. M. F. P. Angelo Chianese, Fiammetta Marulli. SmARTweet:A Location-based smart application for Exhibits and Museums. *2013 International Conference on Signal-Image Technology —& Internet-Based Systems*, 2013. URL https://ieeexplore.ieee.org/document/6727222.

[7] H. L. Artur Baniukevic, Christian S. Jensen. Hybrid Indoor Positioning With Wi-fi and Bluetooth: Architecture and Performance. *IEEE 14th International Conference on Mobile Data Management*, 2013. URL http://madalgo.au.dk/fileadmin/madalgo/OA_PDF_s/C290-signed.pdf.

[8] T.-N. D. Y. J. X. L. U.-X. T. Ran Liu, Chau Yuen. Indoor Positioning using Similarity-based Sequence and Dead Reckoning without Training. *IEEE Workshop on Signal Processing Advances in Wireless Communications*, 2017. URL https://arxiv.org/pdf/1705.04934.pdf.

[9] Y. L. Chenshu Wu, Zheng Yang. Smartphones based Crowdsourcing for Indoor Localization. *IEEE Transactions on Mobile Computing*, 2015. URL https://ieeexplore.ieee.org/document/6805641.

[10] A. E. M. F. M. Y. R. R. C. He Wang, Souvik Sen. No Need to War-Drive: Unsupervised Indoor Location. *MobiSys '12*, 2012. URL https://www.cs.purdue.edu/homes/hw/papers/unloc_mobisys12.pdf.

[11] K. C. L. Suk Lee, Kyoung Nam Ha. A Pyroelectric Infrared Sensor-based Indoor Location-Aware System for the Smart Home. *IEEE Transactions on Consumer Electronics*, 2006. URL https://ieeexplore.ieee.org/document/4050061.

[12] R. J. O. e Gregory D. Abowd. The Smart Floor: A Mechanism for Natural User Identification and Tracking. *College of Computing and GVU Center*, 2000. URL https://www.cc.gatech.edu/fce/ahri/publications/floor-short.pdf.

[13] M. Rosa. Serenar: 3 principais causas de alarme falso com sensor infravermelho passivo. URL http://serenarseguranca.com.br/3-causas-de-alarme-falso-com-sensor-infravermelho-passivo/. Online; acedido a 2 de dezembro de 2018.

[14] D. Cai. Museum Navigation based on NFC Localization Approach and Automatic Guidance System. *International Journal of Computer Applications*, 2015. URL https://pdfs.semanticscholar.org/bb55/0a4576e42c7550cba1d9aa93952d7ae4602e.pdf.

[15] Sankar. Quora: What is the maximum distance for NFC communication? URL https://www.quora.com/What-is-the-maximum-distance-for-NFC-communication. Online; acedido a 22 de novembro de 2018.

[16] Y. H. Wonho Kang. SmartPDR: Smartphone-Based Pedestrian Dead Reckoning for Indoor Localization. *IEEE Sensors Journal*, 2015. URL https://ieeexplore.ieee.org/abstract/document/6987239.

[17] Múltiplos autores. FastCSV: High performance CSV reader and writer for Java, . URL https://github.com/osiegmar/FastCSV. Online; acedido a 1 de março de 2019.

[18] Plotly. Chart Studio. URL https://plot.ly/#/. Online; acedido a 4 de março de 2019.

[19] M. Dolan. WikiHow - how to Measure Stride Length. URL https://www.wikihow.com/Measure-Stride-Length. Online; acedido a 22 de fevereiro de 2019.

[20] Android Developers. Sensors Overview, . URL https://developer.android.com/guide/topics/sensors/sensors_overview. Online; acedido a 18 de fevereiro de 2019.

[21] M. Priyadarshini. Which Sensors Do I Have In My Smartphone? How Do They Work?, 2018. URL https://fossbytes.com/which-smartphone-sensors-how-work/. Online; acedido a 27 de maio de 2019.

[22] RotoView. Magnetometer in Smartphones and Tablets. URL https://www.rotoview.com/magnetometer.htm. Online; acedido a 24 de maio de 2019.

[23] MathWorks. Magnetometer. URL https://www.mathworks.com/help/supportpkg/android/ref/magnetometer.html. Online; acedido a 24 de maio de 2019.

[24] Android Developers. Motion sensors, . URL https://developer.android.com/guide/topics/sensors/sensors_motion. Online; acedido a 21 de fevereiro de 2019.

[25] S. Govender. Using the Accelerometer on Android, 2014. URL https://code.tutsplus.com/tutorials/using-the-accelerometer-on-android--mobile-22125. Online; acedido a 27 de maio de 2019.

[26] A. K. Anton Umek. Validation of smartphone gyroscopes for mobile biofeedback applications. *Personal and Ubiquitous Computing*, 2016. URL https://doi.org/10.1007/s00779-016-0946-4.

[27] Múltiplos autores. Wikipedia - Bluetooth low energy beacon, . URL https://en.wikipedia.org/wiki/Bluetooth_low_energy_beacon. Online; acedido a 24 de setembro de 2019.

[28] Pointr. Beacons: Everything you need to know. URL https://blog.pointr.tech/beacons-everything-you-need-to-know. Online; acedido a 24 de setembro de 2019.

[29] D. Addey. iBeacons. URL https://web.archive.org/web/20131203014352/http://daveaddey.com/?p=1252. Online; acedido a 24 de setembro de 2019.

[30] P. Lawitzki. Android Sensor Fusion Tutorial. URL http://plaw.info/articles/sensorfusion/. Online; acedido a 19 de março de 2019.

[31] Google Developers. J2ObjC. URL https://developers.google.com/j2objc. Online; acedido a 14 de outubro de 2019.

[32] Android Developers. Acitivities, . URL https://developer.android.com/guide/components/activities.html. Online; acedido a 27 de setembro de 2019.

[33] Android Developers. Services overview, . URL https://developer.android.com/guide/components/services. Online; acedido a 17 de outubro de 2019.

[34] e. a. Jitesh Dalsaniya, Moti Bartov. StackOverflow - Communication between Activity and Service. URL https://stackoverflow.com/questions/20594936/communication-between-activity-and-service. Online; acedido a 29 de abril de 2019.

[35] Android Developers. SensorEventListener, . URL https://developer.android.com/reference/android/hardware/SensorEventListener. Online; acedido a 26 de setembro de 2019.

[36] Android Developers. SensorEvent, . URL https://developer.android.com/reference/android/hardware/SensorEvent. Online; acedido a 26 de setembro de 2019.

[37] Android Developers. Manifest.permission, . URL https://developer.android.com/reference/android/Manifest.permission. Online; acedido a 27 de setembro de 2019.

[38] Android Developers. BluetoothAdapter, . URL https://developer.android.com/reference/android/bluetooth/BluetoothAdapter. Online; acedido a 27 de setembro de 2019.

[39] Android Developers. BluetoothLeScanner, . URL https://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner. Online; acedido a 27 de setembro de 2019.

[40] Múltiplos autores. Wikipédia - Azimute, . URL https://pt.wikipedia.org/wiki/Azimute. Online; acedido a 20 de março de 2019.

[41] Múltiplos autores. Wikipedia - Azimuth, . URL https://en.wikipedia.org/wiki/Azimuth. Online; acedido a 20 de março de 2019.

[42] Y. M. StackOverflow - How to get Direction in Android (Such as North, West). URL https://stackoverflow.com/questions/8315913/how-to-get-direction-in-android-such-as-north-west. Online; acedido a 22 de março de 2019.

[43] Android Developers. Sensor Manager, . URL https://developer.android.com/reference/android/hardware/SensorManager.html. Online; acedido a 22 de março de 2019.

[44] Tutorial Point. Java - floor() Method, . URL https://www.tutorialspoint.com/java/number_floor.html. Online; acedido a 4 de abril de 2019.

[45] Tutorial Point. Java.lang.Math.ceil() Method, . URL https://www.tutorialspoint.com/java/lang/math_ceil.htm. Online; acedido a 29 de maio de 2019.

[46] Google Brain. TensorFlow - Treine sua primeira rede neural: classificação básica, . URL https://www.tensorflow.org/tutorials/keras/classification. Online; acedido a 1 de julho de 2019.

[47] S. Mall. GitHub - Low-Effort Place Recognition with WiFi Fingerprints Using Deep Learning (Keras), . URL https://github.com/mallsk23/place_recognition_wifi_fingerprints_deep_learning/blob/master/README.md.

[48] S. Mall. Low-effort place recognition with WiFifingerprints using deep learning, . URL https://arxiv.org/pdf/1611.02049v1.pdf. Online; acedido a 1 de julho de 2019.

[49] Google Brain. TensorFlow - Save and load models, . URL https://www.tensorflow.org/tutorials/keras/save_and_load#save_the_entire_model. Online; acedido a 30 de julho de 2019.

[50] C. Echterling. What is the difference between an Azimuth and Bearing? URL http://chrisechterling.com/blog/2009/09/07/what-is-the-difference-between-an-azimuth-and-bearing/. Online; acedido a 22 de março de 2019.

Anexo A

Arquitetura

A.1 Aplicação de recolha dos dados dos sensores (secção 3.3.1)

Para a recolha de dados dos sensores, durante a análise ao movimento, foi desenvolvida uma aplicação móvel para o sistema Android, utilizando a respetiva *framework* para aceder aos sensores acelerómetro, giroscópio e magnetómetro. O código desta aplicação é o seguinte:

```
public class MainAcitivity extends AppCombatActivity implements
        SensorEventListener {
    // ArrayList para guardar todos os valores medidos em cada instante
    Collection<String[]> accerelometerData = new ArrayList<>();
    Collection<String[]> magneticData = new ArrayList<>();
    Collection<String[]> gyroData = new ArrayList<>();

    // Vetores que guardam os utlimos valores medidos
    private float accValues[] = new float[3];
    private float gyroValues[] = new float[3];
    private float magValues[] = new float[3];

    boolean recording = false;  // Indica se a gravacao de dados foi iniciada
    ...
    @Override
    public void onSensorChanged (SensorEvent event){
        // Converte o timestamp para milisegundos
        long timeInMillis = (new Date()).getTime()
                    + (event.timestamp - System.nanoTime()) / 1000000L;

        // Se a gravacao de dados tiver sido iniciada
        if (recording) {
            // Evento do sensor acelerometro
            if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
                accValues = event.values;
            }
            // Evento do sensor giroscopio
            if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
```

```
                gyroValues = event.values;
            }
            // Evento do sensor magnetometro
            if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
                magValues = event.values;
            }

        // E utilizado um contador ao inves do timestamp
        if (counterOn){
            accerelometerData.add(new String[]{String.valueOf(counter), String.valueOf(
    accValues[0]), String.valueOf(accValues[1]), String.valueOf(accValues[2])});
            gyroData.add(new String[]{String.valueOf(counter), String.valueOf(gyroValues
    [0]), String.valueOf(gyroValues[1]), String.valueOf(gyroValues[2])});
            magneticData.add(new String[]{String.valueOf(counter), String.valueOf(
    magValues[0]), String.valueOf(magValues[1]), String.valueOf(magValues[2])});
        }
        // E utilizado o timestamp
        else {
            accerelometerData.add(new String[]{String.valueOf(timeInMillis), String.
    valueOf(accValues[0]), String.valueOf(accValues[1]), String.valueOf(accValues[2])});
            gyroData.add(new String[]{String.valueOf(timeInMillis), String.valueOf(
    gyroValues[0]), String.valueOf(gyroValues[1]), String.valueOf(gyroValues[2])});
            magneticData.add(new String[]{String.valueOf(timeInMillis), String.valueOf(
    magValues[0]), String.valueOf(magValues[1]), String.valueOf(magValues[2])});
        }

        counter++;      // Incrementa o contador
    }
}
```

Omitidos estão os métodos relacionados com a escrita em ficheiros, por serem irrelevantes no contexto desta tese. Uma biblioteca externa, retirada de [17], foi utilizada para a escrita de ficheiros `.csv`.

Anexo B

Implementação

B.1 Arquitetura de *Software* (secção 4.1.2)

A figura B.1 apresenta o diagrama de classes completo, incluindo os atributos e métodos principais, excluindo aqueles relacionados com a comunicação entre os *services* e a `MainActivity`.

B.2 Cliente - Módulo de Sensorização (secção 4.2)

checkLocationPermissions() – Este método verifica as permissões de localização e requer ao utilizador a sua autorização. Está implementado no `BeaconsService`. O código desta função é o seguinte:

```
1  // Verifica (e requer se necessario) permissao para a localizacao
2  public void checkLocationPermissions (){
3      int permissionCheckFineLocation =
4      ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION);
5      int permissionCheckCoarseLocation =
6      ContextCompat.checkSelfPermission(this, android.Manifest.permission.
       ACCESS_COARSE_LOCATION);
7
8      if (permissionCheckFineLocation != PackageManager.PERMISSION_GRANTED){
9          ActivityCompat.requestPermissions(mainActivity, new String[]
10         {android.Manifest.permission.ACCESS_FINE_LOCATION},
       PERMISSION_ACCESS_FINE_LOCATION);
11     }
12
13     if (permissionCheckCoarseLocation != PackageManager.PERMISSION_GRANTED){
14         ActivityCompat.requestPermissions(mainActivity, new String[]
15         {android.Manifest.permission.ACCESS_COARSE_LOCATION},
       PERMISSION_ACCESS_COURSE_LOCATION);
16     }
17 }
```

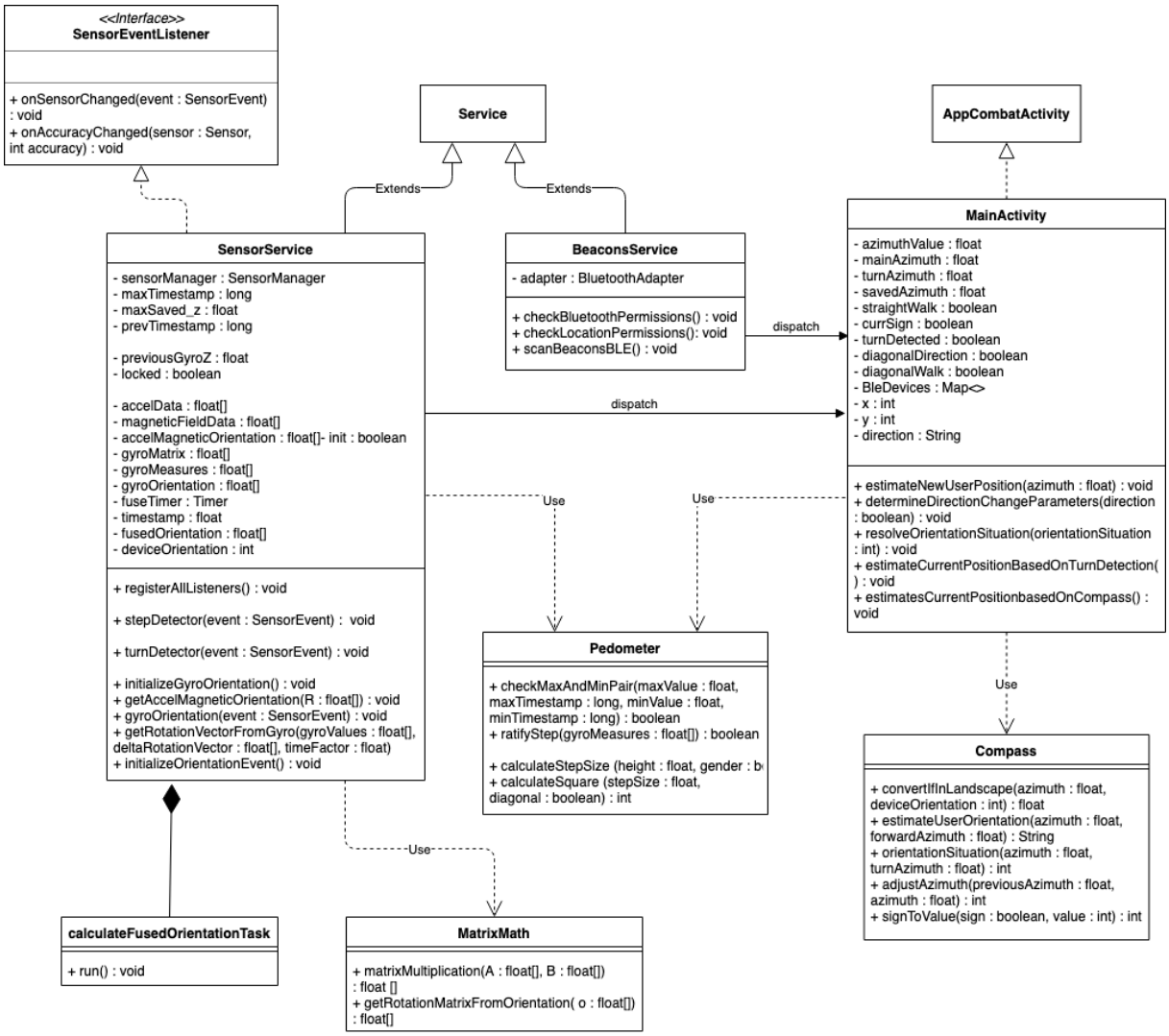


Figura B.1: Diagrama de classes completo, incluindo atributos e métodos.

B.3 Cliente - Módulo de Cálculo da trajetória (secção 4.3)

B.3.1 Pedómetro (secção 4.3.1)

Durante a implementação do Pedómetro, foram desenvolvidas três versões principais para Android, que serão descritas neste anexo. É importante ter em conta que todas estas iterações tiveram múltiplas alterações, pegando em novas ideais e conceitos que permitiram melhorar e simplificar o código, não existindo uma lógica definitiva para cada versão.

Versão 1.0

A primeira iteração desta versão foi adaptada da lógica inicialmente desenvolvida em Python. Para além das óbvias modificações necessárias por causa das diferenças e duas linguagens, outras alterações foram feitas com o objetivo de melhorar a precisão, nomeadamente: a utilização de dois limites para os extremos; a verificação de medidas noutros eixos; e a identificação do segundo máximo da assinatura.

Durante os vários testes realizados, registou-se um erro médio de 1.14 passos, em 7 percursos de 7 passos. Ocorreram, no entanto, algumas situações em que a aplicação falhou em detetar a maior parte dos passos. Outro dos problemas com esta versão foi a complexidade do código, por causa da procura pelo segundo máximo, o que também causava um atraso na deteção da passada.

O código final desta versão é o seguinte:

```
public class SensorService extends Service implements
        SensorEventListener {

  List<Float> accMeasures_x = new ArrayList<>();   // Guarda todos os valores do eixo do
    x
  List<Float> accMeasures_y = new ArrayList<>();   // Guarda todos os valores do eixo do
    y

  List<Float> secAccMeasures_y = new ArrayList<>();   // Guarda todas as medidas do eixo
    y apos a detecao do minimo principal

  boolean stepIndicator = false;      // Indica se o sinal esta a descer
  float  minSaved_x,  minSaved_y;     // Guarda os valores minimos do sinal detetados
    para os eixos x e y

  int numberSteps=0;
  ...
  @Override
  public void onSensorChanged (SensorEvent event){

      if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){

          if (!accMeasures_y.isEmpty()) {
              float previousMeasure = accMeasures_y.get(accMeasures_y.size() - 1);

              // Obter o ultimo valor colocado no array
              if (previousMeasure > data_y) {
                  minSaved_x = data_x;    // Guardar minimo no eixo x
                  minSaved_y = data_y;    // Guardar minimo no eixo y
                  // Sinal esta em descida
              }
              else {
                  // Sinal esta em ascensao
                  if (!stepIndicator) {
                      // Verificar se os minimos estao entre os limites definidos
                      if (minSaved_y > 0 && minSaved_y < 4 && minSaved_x > -5) {
                          float max = Collections.max(accMeasures_y);     // Valor maximo
     encontrado

                          // Verificar se os maximos estao entre limites definidos
                          if ((max - minSaved_y) > 3 && max > 6 && max < 10) {
                              stepIndicator = true;   // Possivel assinatura de passada
    encontrada
```

```
38
39                              // Reinicializar as variaveis e os arrays
40                              accMeasures_x.clear();
41                              accMeasures_y.clear();
42                          }
43                      }
44                  }
45                  else {
46                      // Se o tamanho do array dos y for menor do que 50
47                      if (secAccMeasures_y.size() <= 50) {
48                          float secMax = Collections.max(secAccMeasures_y);   //
    Determinar o segundo maximo da assinatura
49                           // Segundo maximo encontrado
50
51                          // Se a diferenca entre o primeiro minimo e este segundo maximo
     for maior que 0.5 m/s^2
52                          if ((secMax - minSaved_y) > 0.5) {
53                              // Passo detetado
54                              numberSteps++;  // Incrementa numero total de passos
55
56                              // Renicializar as variaveis e arrays
57                              secAccMeasures_y.clear();
58                              stepIndicator = false;
59                          }
60                      } else {
61                          stepIndicator=false;
62                      }
63                  }
64                  minSaved_x = 0.0f;
65                  minSaved_y = 0.0f;
66              }
67          }
68          accMeasures_x.add(data_x);
69          accMeasures_y.add(data_y);
70
71          // Se um par maximo-minimo tiver sido encontrado, todos os valores do eixo z
    passam a ser guardados num array
72          if (stepIndicator) {
73              secAccMeasures_y.add(data_y);
74          }
75      }
76  }
77 }
```

Versão 2.0

Para esta segunda versão, a lógica foi parcialmente reformulada, passando a utilizar o eixo *z*, pois a assinatura do passo é mais acentuada aqui. Outras ideias foram consideradas, tal como: adicionar

um *threshold* para o número de eventos entre o máximo e o mínimo; verificar se a diferença entre os extremos está dentro de certos limites; e uma observação, incorreta, de que entre o máximo e o mínimo, o sinal continua sempre a descer.

Esta lógica, apesar das supostas melhorias, acabou por apresentar piores resultados que as anteriores, o que levou á sua reformulação. Existem duas possíveis razões para isto. Em primeiro lugar, a caracterização do passo comprovou que o sinal pode de facto subir entre o máximo e o mínimo. Para além disso, o número de eventos entre estes extremos não está correlacionado com a assinatura do passo.

O código final desta versão é o seguinte:

```
1  public class SensorService extends Service implements
2          SensorEventListener {
3
4    private List<Float> accMeasures_z = new ArrayList<Float>(80);   // Guarda as ultimas
       medidas do eixo do z
5
6    float maxSaved_z;   // Guarda o valor maximo identificado do sinal
7    float  previousSaved_x, previousSaved_y;     // Guarda os valores anteriores do sinal
       detetados para os eixos x e y
8
9    boolean inDescent = false;  // Identifica se o sinal esta a descer
10   int isInDescentCounter = 0;     // Conta o numero de eventos na descida, entre o maximo
        identificado ate a um possivel minimo
11
12   int numberSteps=0;
13   ...
14   @Override
15   public void onSensorChanged (SensorEvent event){
16
17       if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
18           // Copia os valores do novo evento para variaveis
19           data_x = e.values[0];   // Eixo X
20           data_y = e.values[1];   // Eixo Y
21           data_z = e.values[2];   // Eixo Z
22
23           // Se o numero de valores contados em descida for maior que 80
24           if (isInDescentCounter > 80){
25               inDescent=false;
26               isInDescentCounter=0;
27           }
28
29           // Se o array de medidas nao estiver vazio
30           if (!accMeasures_z.isEmpty()){
31               float previousMeasure = accMeasures_z.get(accMeasures_z.size() - 1);     //
       Obter o ultimo valor colocado no array do valor do eixo Z
32
33               if (previousMeasure > data_z){
34                   // Sinal esta em descida
```

```
                if (!inDescent) {
                    // Verificar se o possivel maximo esta dentro dos limites definidos
                    if (previousMeasure > 10 && previousMeasure < 16) {
                        maxSaved_z = previousMeasure;   // Maximo encontrado
                        inDescent = true;   // O sinal esta em descida
                    }
                }
            } else {
                // O sinal esta em ascencao
                if (previousMeasure < 9 && previousMeasure > 2.5 && previousSaved_x < 1.5 && previousSaved_y < 5){

                    // Se o sinal esteve em descida
                    if (inDescent){
                        float difference = maxSaved_z - previousMeasure;    // Calcular a diferenca entre o valor maximo e o minimo

                        // Verificar se a diferenca entre os extremos esta dentro dos limites
                        if (difference > 5 && difference < 14){
                            // Passo detetado

                            // Incrementa o numero total de passos
                            numberSteps++;
                        }
                    }
                }

                // Renicializar as variaveis
                inDescent = false;
                maxSaved_z = 0;
            }
        }

        // O sinal esta em descida apos um maximo e a contagem foi iniciada
        if (inDescent){
            isInDescentCounter++;   // Incrementa o numero de eventos apos o maximo
        }

        previousSaved_x = data_x;   // Guardar o valor do eixo X
        previousSaved_y = data_y;   // Guardar o valor do eixo Y

        accMeasures_z.add(data_z);  // Adicionar o valor do eixo Z no array
    }
  }
}
```

Versão 3.0

Nesta versão final, a lógica para a identificação do estado do sinal foi mantida, eliminando o `ArrayList` para guardar todos os valores do eixo *z*, substituindo este por variáveis que guardam a amplitude e o *timestamp*. Para minimizar o impacto dos falsos positivos, foram também estudados alguns movimentos aleatórios, cuja caracterização levou á implementação do método `ratifyStep()`. Outra possibilidade que foi explorada para melhorar a precisão, foi a diferença de tempo entre o máximo e o mínimo da possível assinatura encontrada.

O código desta versão final é o seguinte:

```
1  public class SensorService extends Service implements
2          SensorEventListener {
3
4    float previousMeasure;       // Guarda o ultimo valor registado no eixo Z do sensor
       acelerometro
5    float maxSaved_z;       // Guarda o ultimo valor maximo registado no eixo Z do sensor
       acelerometro
6
7    float previousSaved_x, previousSaved_y;     // Guarda os ultimos valores do eixos X e Y
        do sensor acelerometro
8
9    long maxSavedTimestamp;     // Guarda o timestamp do evento que registou o valor maximo
        no eixo Z do acelerometro
10   long previousSavedTimestamp;    // Guarda o timestamp do evento anterior do
       acelerometro
11
12   private float[] gyroMeasures = new float[3];    // Medidas do eixo X,Y e Z do
       giroscopio
13   ...
14   @Override
15   public void onSensorChanged(SensorEvent event) {
16
17       if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
18           // Copia valores do evento para variaveis
19           float data_x = event.values[0];     // Valor do eixo X
20           float data_y = event.values[1];     // valor do eixo Y
21           float data_z = event.values[2];     // Valor do eixo Z
22
23           // Para encontrar um valor maximo
24           if (previousMeasure > data_z){
25
26               // O maximo tem de estar entre os 11 e 15 m/s^2
27               if (previousMeasure > 11 && previousMeasure < 15 && previousMeasure >
       maxSaved_z){
28                   maxSaved_z = previousMeasure;
29                   maxSavedTimestamp = previousSavedTimestamp;
30               }
31           }
```

```
32          // Para encontrar um valor minimo
33          else {
34
35              // O minimo tem de estar entre os 5 e 8 m/s^2
36              if ((previousMeasure < 8 && previousMeasure > 5)){
37
38                  // Analisar os outros eixos de modo a descartar falsos positivos
39                  if (maxSaved_z!=0 && (previousSaved_x < 2 && previousSaved_x > -4) && (
    previousSaved_y < 5 && previousSaved_y > 0)){
40
41                      float difference = maxSaved_z - previousMeasure;    // Calcula a
    diferenca entre os valores maximo e minimo
42                      float timeDifference = maxSavedTimestamp - previousSavedTimestamp;
        // Calcula a diferenca entre os tempos dos valores maximo e minimo
43
44                      // Verificar se as diferencas estao dentro dos limites definidos,
    tanto o valor da aceleracao, como dos timestamps correspondentes
45                      if ((difference > 3 && difference < 6) && (timeDifference > 100 &&
    timeDifference < 250)) {
46
47                          // Ratificar o possivel passo com os valores do giroscopio
48                          if (ratifyStep(gyroMeasures)) {
49
50                              // Incrementa o numero total de passos
51                              numberSteps++;
52
53                              maxSaved_z = 0;     // Reinicializa a variavel
54                          }
55                      }
56                  }
57              }
58          }
59
60          // Guarda os valores e timestamp atuais para comparacao no proximo evento do
    sensor
61          previousSaved_x = data_x;
62          previousSaved_y = data_y;
63          previousMeasure = data_z;
64          previousSavedTimestamp = (new Date()).getTime()
65                      + (event.timestamp - System.nanoTime()) / 1000000L;
66      }
67
68      // Se o novo evento for do sensor giroscopio
69      else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE){
70          gyroMeasures = event.values;
71      }
72  }
73 }
```

Este código foi simplificado, removendo os múltiplos condicionais emaranhados e movendo certas

operações para novos métodos. A verificação das medidas dos outros eixos do acelerómetro foi removida, dependendo apenas do método `ratifyStep()` para descartar os falsos positivos. A análise dos pares máximo/mínimo foi alterada, sendo as comparações entre os extremos efetuadas num novo método, denominado `checkMaxAndMinPair()`. Para além disso, a comparação com o maior limite temporal foi movida para fora da procura dos extremos, para considerar os casos em que não existe um mínimo aceitável, de modo a reinicializar as variáveis. Este código simplificado foi colocado num método para o efeito, `stepDetector()`, que é chamado sempre que houver um novo evento do sensor acelerómetro.

O código da versão final simplificada é o seguinte:

```
1  public class SensorService extends Service implements
2          SensorEventListener {
3
4      private long maxTimestamp,       // Guarda o timestamp correspondente ao ultimo maximo
        encontrado
5      private long prevTimestamp;     // Guarda o timestamp do ultimo evento do acelerometro
6      private float maxSaved_z;        // Guarda o ultimo valor do eixo Z do aceleromtetro
7
8      private float[] gyroMeasures = new float[3];     // Medidas dos tres eixos do
       giroscopio
9
10   // Deteta um passo com base nos dados do acelerometro
11   // Se detetar um minimo apos um maximo
12   // Ambos devem estar dentro dos limites definidos
13   public void stepDetector (SensorEvent event){
14       // Copiar valores do evento para variaveis
15       float curData_Z = event.values[2];
16       float prevData_Z = accelData[2];
17
18       switch (Float.compare(prevData_Z, curData_Z)){
19           // Para encontrar um valor maximo
20           case 1:
21               if ((prevData_Z > 11 && prevData_Z < 15) && prevData_Z > maxSaved_z){
22                   maxSaved_z = prevData_Z;
23                   maxTimestamp = prevTimestamp;
24               }
25               break;
26
27           // Para encontrar um valor minimo
28           case -1:
29               if (prevData_Z < 8 && prevData_Z > 4){
30                   // verifica par maximo/minimo
31                   boolean isSignature = Pedometer.checkMaxAndMinPair(maxSaved_z,
     maxTimestamp, prevData_Z, prevTimestamp);
32
33                   // Ratifica passo
34                   boolean stepRatified = Pedometer.ratifyStep(gyroMeasures);
35
```

```
                if (isSignature && stepRatified){
                    activity.stepDetected();

                    // Renicializar variaveis
                    maxSaved_z = 0;
                    maxTimestamp = 0;
                }
            }
        break;
    }

    // Renicializa variaveis se o valor maximo estiver preso num para o qual nao existe minimo aceitavel
    if (prevTimestamp - maxTimestamp > 350){
        maxSaved_z=0;
        maxTimestamp =0;
    }

    prevTimestamp = (new Date()).getTime()
            + (event.timestamp - System.nanoTime()) / 1000000L;     // Guarda o timestamp atual para comparacao posterior
  }
}
```

Na classe `Pedometer`, estão definidos os métodos `ratifyStep()`, que faz a ratificação de um possível passo, e `checkMaxAndMinPair()`. O código desta classe é o seguinte:

```
public class Pedometer {

    // Calcula a diferenca de valor e temporal entre o maximo e o minimo encontrados
    // Se este valores estiverem dentro dos limites, entao a funcao retorna true,
    // considerando a assinatura causada por um passo do utilizador
    public static boolean checkMaxAndMinPair (float maxValue, long maxTimestamp, float minValue, long minTimestamp){

        // Calcula a diferenca temporal e de valor entre os dois extremos
        float valueDifference = maxValue - minValue;
        float timeDifference = minTimestamp - maxTimestamp;

        if ((valueDifference > 4 && valueDifference < 9) && (timeDifference > 60)) {
            return true;
        }

        return false;
    }

    // Verifica se os ultimos valores do giroscopio estao entre -1 e 1
    // Retorna true ou false, dependendo dos valores no vetor
    // O giroscopio e mais sensivel a movimentacoes do dispositivo, por isso e usado aqui para descartar falsos positivos
```

```
22      public static boolean ratifyStep(float[] gyroMeasures){
23
24          // Para cada um dos valores dos eixo X, Y e Z
25          for (float value : gyroMeasures){
26              if (!(value > -1 && value < 1)){
27                  return false;
28              }
29          }
30          return true;
31      }
32  }
```

B.3.2 Deteção de curvas/viragens (secção 4.3.2)

A primeira versão do método `turnDetector()`, adapta a lógica do Pedómetro para a identificação do estado do sinal (sobe ou desce). Neste caso apenas é necessário identificar um extremo, que pode ser um máximo ou um mínimo.

O código desta versão é seguinte:

```
1  public void turnDetector (SensorEvent event){
2      float gyroZ = event.values[2];
3
4      // Para encontrar um valor maximo (viragem a esquerda)
5      if (previousGyroZ > gyroZ) {
6          if (previousGyroZ > 1.8 && previousGyroZ > gyroMaxSaved) {
7              activity.notifyDirectionChange(false, fusedOrientation[0]); // Notifica um
       viragem a esquerda
8          }
9      }
10     // Para encontrar um valor minimo (viragem a direita)
11     else {
12         if (previousGyroZ < -1.8) {
13             activity.notifyDirectionChange(true, fusedOrientation[0]);      // Notifica
       uma viragem a direita
14         }
15     }
16
17     previousGyroZ = gyroZ;      // Guarda o valor atual do eixo Z para comparar no
       proximo evento
18 }
```

A versão final do método implementa uma técnica de bloqueio ou trancas, que impede após um extremo ser detetado, que um novo valor possa acionar outra notificação.

O código desta versão final é seguinte:

```
1  // Variaveis relacionadas a detecao de viragens
2  private float previousGyroZ;        // Guarda o ultimo valor do eixo Z do giroscopio
3
4  private boolean locked=false;       // Indica o estado do bloqueio
```

```
...
// Deteta uma viragem com base nos dados do girocopio
// Se detetar um maximo acima de 1.8 rad/s ou um minimo abaixo de -1.8 rad/s, entao o utilizador virou
public void turnDetector (SensorEvent event){
    float gyroZ = event.values[2];

    // Apenas se o sinal tiver estabilizado anteriormente (estiver perto de 0), e que as verificacoes da mudanca de direcao sao feitas
    if (!locked) {

        // Para encontrar um valor maximo (viragem a esquerda)
        if (previousGyroZ > gyroZ) {
            if (previousGyroZ > 1.8) {
                activity.notifyDirectionChange(false, fusedOrientation[0]);     // Notifica um viragem a esquerda
                locked=true;        // Bloqueia as verificacoes
            }
        }
        // Para encontrar um valor minimo (viragem a direita)
        else {
            if (previousGyroZ < -1.8) {
                activity.notifyDirectionChange(true, fusedOrientation[0]);      // Notifica um viragem a direita
                locked=true;        // Bloqueia as verificacoes
            }
        }
    } else {

        // Se os dados do eixo Z estiverem estabilizados (estarem perto de 0), entao a variavel e atualizada para false (desbloquear)
        if (gyroZ < 0.1 && gyroZ -0.1) {
            locked = false;
        }
    }

    previousGyroZ = gyroZ;      // Guarda o valor atual do eixo Z para comparar no proximo evento
}
```

B.3.3 Bússola (secção 4.3.3)

Tal como descrito, a bússola foi implementada em duas fases. Numa fase inicial, foi criada uma bússola base, seguida de uma bússola com fusão de sensores.

Neste anexo é apresentado o código desenvolvido para estas duas fases.

Bússola base

O código para a bússola base, adaptado de [42], foi implementado no `SensorService`:

```
public class SensorService extends Service implements SensorEventListener {

  ...
  public static final int TIME_CONSTANT = 30;
    public static final float FILTER_COEFFICIENT = 0.98f;
    private Timer fuseTimer = new Timer();

    public void onCreate(Bundle savedInstanceState) {
      ...

        // Espera por 1 segundo ate que os dados do giroscopio e
        // acelerometro/magnetometro sejam inicializados e entao programa a task
        //do filtro complementar
        // Fonte: http://plaw.info/articles/sensorfusion/
        fuseTimer.scheduleAtFixedRate(new calculateFusedOrientationTask(),
                                      1000, TIME_CONSTANT);
    }

    // Inicializa a matriz de rotacao e o vetor de orientacao do giroscopio
    // Fonte: http://plaw.info/articles/sensorfusion/
    public void initializeGyroOrientation (){

        // Vetor de orientacao do giroscopio
        gyroOrientation[0] = 0.0f;
        gyroOrientation[1] = 0.0f;
        gyroOrientation[2] = 0.0f;

        // Inicializar gyroMatrix com a matriz de identidade
        gyroMatrix[0] = 1.0f; gyroMatrix[1] = 0.0f; gyroMatrix[2] = 0.0f;
        gyroMatrix[3] = 0.0f; gyroMatrix[4] = 1.0f; gyroMatrix[5] = 0.0f;
        gyroMatrix[6] = 0.0f; gyroMatrix[7] = 0.0f; gyroMatrix[8] = 1.0f;
    }

    @Override
    public void onSensorChanged(SensorEvent event) {

        // Se o evento for do tipo acelerometro
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
            System.arraycopy(event.values, 0, accelData, 0, 3);     // Para a estimativa
    da direcao
            ...
        }
        // Se o evento for do tipo magnetometro
        else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD){
            System.arraycopy(event.values, 0, magneticFieldData, 0, 3);

```

```
46          float[] R = new float[9];             // Matriz de rotacao onde os resultados
    de getRotationMatrix() serao guardados
47          getAccelMagneticOrientation(R);     // Para obter a orientacao com base no
    acelerometro/magnetometro
48      }
49      // Se o evento for do tipo giroscopio
50      else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE){
51          ...
52          gyroOrientation(event);     // Para obter a orientacao do giroscopio
53      }
54  }
55
56  // Esta funcao realiza a integracao dos dados do giroscopio.
57  // Escreve a orientacao baseada no giroscopio no vetor gyroOrientation
58  // Fonte: http://plaw.info/articles/sensorfusion/
59  public void gyroOrientation (SensorEvent event){
60
61      // Apenas pode comecar apos a orietancao do acelerometro/giroscopio ter sido
    adquirida
62      if (accelMagneticOrientation == null){
63          return;
64      }
65
66      // Inicializacao
67      if (init){
68          float[] initialMatrix = new float[9];
69          initialMatrix = getRotationMatrixFromOrientation(accelMagneticOrientation);
    // Obtem a matriz de rotacao a partir da orientacao do acelerometro/magnetometro
70
71          float[] test = new float[3];
72          SensorManager.getOrientation(initialMatrix, test);      // obtem a orientacao
     com base na matriz de rotacao
73
74          gyroMatrix = matrixMultiplication(gyroMatrix, initialMatrix);       //
    Calculo da matriz de rotacao do giroscopio, atraves da multiplicacao das matrizes
75      }
76
77      float[] deltaVector = new float[4];
78      if(timestamp != 0) {
79          final float dT = (event.timestamp - timestamp) * NS2S;
80          System.arraycopy(event.values, 0, gyroMeasures, 0, 3);      // Copia os novos
     valores do giroscopio para o vetor
81
82          getRotationVectorFromGyro(gyroMeasures, deltaVector, dT / 2.0f);    //
    Converte os dados em RAW num vetor de rotacao
83      }
84
85      timestamp = event.timestamp; // Guarda o timestamp para o proximo intervalo
86
```

```
87          // Converte o vetor de rotacao numa matriz de rotacao
88          float[] deltaMatrix = new float[9];
89          SensorManager.getRotationMatrixFromVector(deltaMatrix, deltaVector);
90
91          gyroMatrix = matrixMultiplication(gyroMatrix, deltaMatrix);     // Aplica o novo
    intervalo de rotacao na matriz de rotacao baseada no giroscopio
92
93          SensorManager.getOrientation(gyroMatrix, gyroOrientation);      // Obtem a
    orientacao baseada no giroscopio a partir da matriz de rotacao
94      }
95
96      // Fonte: http://plaw.info/articles/sensorfusion/
97      class calculateFusedOrientationTask extends TimerTask {
98      public void run() {
99          float oneMinusCoeff = 1.0f - FILTER_COEFFICIENT;
100
101         // Azimute
102         if (gyroOrientation[0] < -0.5 * Math.PI && accelMagneticOrientation[0] > 0.0) {
103             fusedOrientation[0] = (float) (FILTER_COEFFICIENT * (gyroOrientation[0] + 2.0
     * Math.PI) + oneMinusCoeff * accelMagneticOrientation[0]);
104             fusedOrientation[0] -= (fusedOrientation[0] > Math.PI) ? 2.0 * Math.PI : 0;
105         }
106         else if (accelMagneticOrientation[0] < -0.5 * Math.PI && gyroOrientation[0] >
    0.0) {
107             fusedOrientation[0] = (float) (FILTER_COEFFICIENT * gyroOrientation[0] +
    oneMinusCoeff * (accelMagneticOrientation[0] + 2.0 * Math.PI));
108             fusedOrientation[0] -= (fusedOrientation[0] > Math.PI)? 2.0 * Math.PI : 0;
109         }
110         else {
111             fusedOrientation[0] = FILTER_COEFFICIENT * gyroOrientation[0] + oneMinusCoeff
     * accelMagneticOrientation[0];
112         }
113
114         ...
115
116         // Grava a matriz do giroscopio e a orientacao com a orientacao conjunta para
    compensar o flutuar do sensor
117         gyroMatrix = getRotationMatrixFromOrientation(fusedOrientation);
118         System.arraycopy(fusedOrientation, 0, gyroOrientation, 0, 3);
119
120         activity.updateAzimuth(System.currentTimeMillis(), fusedOrientation[0]);    //
    Notifica a mudanca no valor do azimute
121     }
122 }
```

Bússola com fusão de sensores

O código para a bússola com fusão de sensores, adaptado de [30], foi implementado no `SensorService`:

```
public class SensorService extends Service implements SensorEventListener {

  ...
  public static final int TIME_CONSTANT = 30;
    public static final float FILTER_COEFFICIENT = 0.98f;
    private Timer fuseTimer = new Timer();

    public void onCreate(Bundle savedInstanceState) {
      ...

        // Espera por 1 segundo ate que os dados do giroscopio e
        // acelerometro/magnetometro sejam inicializados e entao programa a task
        //do filtro complementar
        // Fonte: http://plaw.info/articles/sensorfusion/
        fuseTimer.scheduleAtFixedRate(new calculateFusedOrientationTask(),
                                      1000, TIME_CONSTANT);
    }

    // Inicializa a matriz de rotacao e o vetor de orientacao do giroscopio
    // Fonte: http://plaw.info/articles/sensorfusion/
    public void initializeGyroOrientation (){

        // Vetor de orientacao do giroscopio
        gyroOrientation[0] = 0.0f;
        gyroOrientation[1] = 0.0f;
        gyroOrientation[2] = 0.0f;

        // Inicializar gyroMatrix com a matriz de identidade
        gyroMatrix[0] = 1.0f; gyroMatrix[1] = 0.0f; gyroMatrix[2] = 0.0f;
        gyroMatrix[3] = 0.0f; gyroMatrix[4] = 1.0f; gyroMatrix[5] = 0.0f;
        gyroMatrix[6] = 0.0f; gyroMatrix[7] = 0.0f; gyroMatrix[8] = 1.0f;
    }

    @Override
    public void onSensorChanged(SensorEvent event) {

        // Se o evento for do tipo acelerometro
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
            System.arraycopy(event.values, 0, accelData, 0, 3);     // Para a estimativa
    da direcao
            ...
        }
        // Se o evento for do tipo magnetometro
        else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD){
            System.arraycopy(event.values, 0, magneticFieldData, 0, 3);

            float[] R = new float[9];            // Matriz de rotacao onde os resultados
    de getRotationMatrix() serao guardados
```

```
47          getAccelMagneticOrientation(R);     // Para obter a orientacao com base no
    acelerometro/magnetometro
48      }
49      // Se o evento for do tipo giroscopio
50      else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE){
51          ...
52          gyroOrientation(event);     // Para obter a orientacao do giroscopio
53      }
54  }
55
56  // Esta funcao realiza a integracao dos dados do giroscopio.
57  // Escreve a orientacao baseada no giroscopio no vetor gyroOrientation
58  // Fonte: http://plaw.info/articles/sensorfusion/
59  public void gyroOrientation (SensorEvent event){
60
61      // Apenas pode comecar apos a orietancao do acelerometro/giroscopio ter sido
    adquirida
62      if (accelMagneticOrientation == null){
63          return;
64      }
65
66      // Inicializacao
67      if (init){
68          float[] initialMatrix = new float[9];
69          initialMatrix = getRotationMatrixFromOrientation(accelMagneticOrientation);
    // Obtem a matriz de rotacao a partir da orientacao do acelerometro/magnetometro
70
71          float[] test = new float[3];
72          SensorManager.getOrientation(initialMatrix, test);      // obtem a orientacao
     com base na matriz de rotacao
73
74          gyroMatrix = matrixMultiplication(gyroMatrix, initialMatrix);       //
    Calculo da matriz de rotacao do giroscopio, atraves da multiplicacao das matrizes
75      }
76
77      float[] deltaVector = new float[4];
78      if(timestamp != 0) {
79          final float dT = (event.timestamp - timestamp) * NS2S;
80          System.arraycopy(event.values, 0, gyroMeasures, 0, 3);      // Copia os novos
     valores do giroscopio para o vetor
81
82          getRotationVectorFromGyro(gyroMeasures, deltaVector, dT / 2.0f);    //
    Converte os dados em RAW num vetor de rotacao
83      }
84
85      timestamp = event.timestamp; // Guarda o timestamp para o proximo intervalo
86
87      // Converte o vetor de rotacao numa matriz de rotacao
88      float[] deltaMatrix = new float[9];
```

```
89          SensorManager.getRotationMatrixFromVector(deltaMatrix, deltaVector);
90
91          gyroMatrix = matrixMultiplication(gyroMatrix, deltaMatrix);     // Aplica o novo
    intervalo de rotacao na matriz de rotacao baseada no giroscopio
92
93          SensorManager.getOrientation(gyroMatrix, gyroOrientation);      // Obtem a
    orientacao baseada no giroscopio a partir da matriz de rotacao
94      }
95
96      // Fonte: http://plaw.info/articles/sensorfusion/
97      class calculateFusedOrientationTask extends TimerTask {
98      public void run() {
99          float oneMinusCoeff = 1.0f - FILTER_COEFFICIENT;
100
101         // Azimute
102         if (gyroOrientation[0] < -0.5 * Math.PI && accelMagneticOrientation[0] > 0.0) {
103             fusedOrientation[0] = (float) (FILTER_COEFFICIENT * (gyroOrientation[0] + 2.0
     * Math.PI) + oneMinusCoeff * accelMagneticOrientation[0]);
104             fusedOrientation[0] -= (fusedOrientation[0] > Math.PI) ? 2.0 * Math.PI : 0;
105         }
106         else if (accelMagneticOrientation[0] < -0.5 * Math.PI && gyroOrientation[0] >
    0.0) {
107             fusedOrientation[0] = (float) (FILTER_COEFFICIENT * gyroOrientation[0] +
    oneMinusCoeff * (accelMagneticOrientation[0] + 2.0 * Math.PI));
108             fusedOrientation[0] -= (fusedOrientation[0] > Math.PI)? 2.0 * Math.PI : 0;
109         }
110         else {
111             fusedOrientation[0] = FILTER_COEFFICIENT * gyroOrientation[0] + oneMinusCoeff
     * accelMagneticOrientation[0];
112         }
113
114         ...
115
116         // Grava a matriz do giroscopio e a orientacao com a orientacao conjunta para
    compensar o flutuar do sensor
117         gyroMatrix = getRotationMatrixFromOrientation(fusedOrientation);
118         System.arraycopy(fusedOrientation, 0, gyroOrientation, 0, 3);
119
120         activity.updateAzimuth(System.currentTimeMillis(), fusedOrientation[0]);    //
    Notifica a mudanca no valor do azimute
121     }
122 }
```

A classe `MatrixMath` contém os métodos auxiliares `matrixMultiplication()` e `getRotationMatrixFromOrientation()`, utilizados na bússola com fusão de sensores.

O código desta classe é o seguinte:

```
1  public class MatrixMath {
2
3          private static final double EPSILON = 0.000001;
```

```
4
5   // Multiplica as matrizes A e B
6   // Fonte: http://plaw.info/articles/sensorfusion/
7   public static float[] matrixMultiplication(float[] A, float[] B) {
8       float[] result = new float[9];
9
10      result[0] = A[0] * B[0] + A[1] * B[3] + A[2] * B[6];
11      result[1] = A[0] * B[1] + A[1] * B[4] + A[2] * B[7];
12      result[2] = A[0] * B[2] + A[1] * B[5] + A[2] * B[8];
13
14      result[3] = A[3] * B[0] + A[4] * B[3] + A[5] * B[6];
15      result[4] = A[3] * B[1] + A[4] * B[4] + A[5] * B[7];
16      result[5] = A[3] * B[2] + A[4] * B[5] + A[5] * B[8];
17
18      result[6] = A[6] * B[0] + A[7] * B[3] + A[8] * B[6];
19      result[7] = A[6] * B[1] + A[7] * B[4] + A[8] * B[7];
20      result[8] = A[6] * B[2] + A[7] * B[5] + A[8] * B[8];
21
22      return result;
23  }
24
25  // Converte o vetor de rotacao numa matriz
26  // Fonte: http://plaw.info/articles/sensorfusion/
27  private float[] getRotationMatrixFromOrientation(float[] o) {
28      float[] xM = new float[9];      // Matriz para o eixo X
29      float[] yM = new float[9];      // Matriz para o eixo Y
30      float[] zM = new float[9];      // Matriz para o eixo Z
31
32      // Calculo dos senos e cossenos do valores do pitch, roll e azimute
33      float sinX = (float)Math.sin(o[1]);
34      float cosX = (float)Math.cos(o[1]);
35      float sinY = (float)Math.sin(o[2]);
36      float cosY = (float)Math.cos(o[2]);
37      float sinZ = (float)Math.sin(o[0]);
38      float cosZ = (float)Math.cos(o[0]);
39
40      // Rotacao sobre o eixo X (pitch)
41      xM[0] = 1.0f; xM[1] = 0.0f; xM[2] = 0.0f;
42      xM[3] = 0.0f; xM[4] = cosX; xM[5] = sinX;
43      xM[6] = 0.0f; xM[7] = -sinX; xM[8] = cosX;
44
45      // Rotacao sobre o eixo Y (roll)
46      yM[0] = cosY; yM[1] = 0.0f; yM[2] = sinY;
47      yM[3] = 0.0f; yM[4] = 1.0f; yM[5] = 0.0f;
48      yM[6] = -sinY; yM[7] = 0.0f; yM[8] = cosY;
49
50      // Rotacao sobre o eixo Z (azimute)
51      zM[0] = cosZ; zM[1] = sinZ; zM[2] = 0.0f;
52      zM[3] = -sinZ; zM[4] = cosZ; zM[5] = 0.0f;
```

```
        zM[6] = 0.0f; zM[7] = 0.0f; zM[8] = 1.0f;

        // Ordem de rotacao e X, Y, Z (roll, pitch, azimuth)
        float[] resultMatrix = matrixMultiplication(xM, yM);
        resultMatrix = matrixMultiplication(zM, resultMatrix);
        return resultMatrix;
    }
}
```

B.4 Cliente - Módulo de Estimativa da posição (secção 4.4)

O código que implementada o módulo o Posicionamento com base na trajetória, está presente na `MainActivity`:

```
public class MainActivity extends AppCompatActivity implements SensorService.Callbacks,
    BeaconsService.Callbacks {

  // Orientacao do percurso
    private float azimuthValue;      // Guarda o ultimo valor do azimute retonado pelo
    SensorService. Utilizado para definir o azimte para o percurso

    private float mainAzimuth;      // Guarda o valor do azimute registado quando o
    utilizador esta esta voltado para a frente da sala
    private float turnAzimuth;     // Guarda o valor do azimute registado imediatamente
    apos a ultima curva
    private float savedAzimuth;   // Guarda o valor do azimute registado aquando do
    ultimo paso

    private boolean diagonalWalk = false;  // Indica se o utilizador esta a caminha
    diagonal aos eixos

    // Detecao de curvas/viragens
    private boolean currSign = true;      // Guarda o tipo da ultima atualizacao das
    coordenadas [soma ou subtracao]
    private boolean turnDetected = false;      // Indica se o utilizaodr virou
    recentemente
    private boolean diagonalDirection;        // Indica a direcao da ultima curva
    private boolean straightWalk=true;       // Indica se o utilizador esta a circular
    num vetor paralelo ao eixo y

    private int x = 0, y = 0;   // Posicao do utilizador

  /*******************************************************
  COMUNICACAO COM OS SERVICES
  *******************************************************/

  // O utilizador deu um passo
    @Override
```

```
26    public void stepDetected() {
27    // O valor principal do azimute (mainAzimuth) fica guardado apos o primeiro passo
28        if (numberSteps==1){
29            mainAzimuth = azimuthValue;
30            turnAzimuth = azimuthValue;
31            savedAzimuth = azimuthValue;
32        }
33    estimateNewUserPosition(azimuthValue);       // Estima a nova posicao do utilizador
34    savedAzimuth = azimuthValue;  // Guarda o valor do azimute apos o passo
35    }
36
37    // O utilizador virou
38  @Override
39  public void notifyDirectionChange(boolean direction, float azimuth) {
40      turnDetected = true;
41      determineDirectionChangeParameters(direction);
42  }
43
44  // O valor do azimute mudou
45    @Override
46    public void updateAzimuth(long timestamp, float azimuth) {
47        azimuthValue = azimuth;
48    }
49
50    /*********************************************************
51  PARAMETROS DO PERCURSO
52  *********************************************************/
53
54    // Dependendo do eixo ao qual o vetor do percurso foi paralelo, da direcao da curva e
       da direcao anterior, a forma como a posicao vai ser utilizada e determinada
55    // Determina a que eixo o vetor e agora paralelo e qual o sinal de atualizao (soma ou
       subtracao)
56    public void determineDirectionChangeParameters(boolean direction) {
57      boolean sign = (straightWalk) == direction;     // A posicao do vetor anterior do
      percurso em relacao aos eixos influencia a direcao do novo
58      currSign = (currSign) == sign;      // A direcao que o utilizador tomou na ultima
      curva (representada pelo sinal) influencia a atual (e o sinal atual)
59      straightWalk = !straightWalk;      // O utilizador circula agora num vetor
      perpendicular ao anterior
60  }
61
62    // Com base na situacao de orientacao determinada antes, a forma correta de estimar a
       nova posicao do utilizador e escolhida
63    // Duas situacoes sao consideradas:
64    // 1) O utilizador circula diagonalmente em relacao aos eixos do referencial
      escolhido
65    // 2) O utilizador circula num percurso paralelo a um dos eixos do referencial, apos
      circular na diagonal
66  public void resolveOrientationSituation(int directionSituation) {
```

```
67      // O utilizador encontra-se num percurso perpendicular a um dos eixos
68      if (directionSituation % 2 == 1) {
69          diagonalDirection = directionSituation != 1;
70          determineDirectionChangeParameters(diagonalDirection); diagonalWalk = false;
71          }
72      // O utilizador encontra-se num percurso diagonal aos eixos
73      else if (directionSituation % 2 == 0) {
74          diagonalDirection = directionSituation != 2;
75          diagonalWalk = true;
76      } else {
77          // Direcao nao considerada
78      }
79  }
80
81  /*******************************************************
82  ESTIMATIVA DA POSICAO ATUAL
83  *******************************************************/
84
85  // Determina como estimar a nova posicao do utilzador, com base nas ultimas mudancas de
       direcao e curvas
86  public void estimateNewUserPosition(float azimuth) {
87      float azimuthDifference = Compass.adjustAzimuth(savedAzimuth, azimuth);     //
      Diferenca entre o valor do azimute deste e do passo anterior
88
89      // Se nenhuma curva tiver sido detetada
90      if (!turnDetected) {
91          // Se a dircao tiver mudado
92          if (azimuthDifference < -(Math.PI / 8) || azimuthDifference > (Math.PI / 8)) {
93              int directionSituation = Compass.orientationSituation(azimuth, turnAzimuth)
      ;       // Determina a nova situacao de orientacao
94              resolveOrientationSituation(directionSituation);    // Resolve a situacao
      atual
95          }
96      } else {
97          turnAzimuth = azimuthValue;     //Guarda o valor do azimute apos a curva
98          turnDetected = false;
99      }
100
101     // Se o percurso for diagonal
102     if (diagonalWalk) {
103         estimateCurrentPositionBasedOnCompass();
104     } else {
105         estimateCurrentPositionBasedOnTurnDetection();
106     }
107 }
108
109 // Com base na ultima mudanca de direcao, as coordenadas x e y sao atualizadas
110 public void estimateCurrentPositionBasedOnTurnDetection() {
111     int square = calculateSqaure(globalClass.getStepSize(), false);
```

```
        if (straightWalk) {
            y += signToValue(currSign, square);
        } else {
            x += signToValue(currSign, square);
        }
    }

    // Com base na ultima mudanca de direcao, as coordenadas x e y sao atualizadas
    public void estimateCurrentPositionBasedOnCompass() {
        int square = calculateSqaure(globalClass.getStepSize(), true);  // Estima quantas
      quadriculas o utilizador percorreu

        // Atualiza a posicao
        if (straightWalk) {
            diagonalDirection = (currSign) == diagonalDirection;    // Para os casos em que
       a posicao anterior tinha valor negativo
            y += signToValue(currSign, square);
            x += signToValue(diagonalDirection, square);
        } else {
            diagonalDirection = (currSign) != diagonalDirection;    // Para os casos em que
       a posicao anterior tinha valor negativo
            x += signToValue(currSign, square);
            y += signToValue(diagonalDirection, square);
        }
}

}
```

A classe `Pedometer` contém métodos auxiliares, relacionados com o cálculo do tamanho do passo e da estimativa da distância percorrida.

O código desta classe é o seguinte:

```
public class Pedometer {

    // Estima o tamanho da passada do utilizador com base na altura e sexo
    // Multiplica a altura por uma constante que e diferente para homens e mulheres
    // Fonte de informacao: https://www.wikihow.com/Measure-Stride-Length
    public static int calculateStepSize (float stepSize, boolean diagonal){
        return (gender) ? height * 0.415f : height * 0.413f;
    }

    // Calcula o numero de quadrados no chao que o utilizador percorreu, baseado na sua
      direcao e tamanho do passo
    // O comprimento do lado de uma quadricula e 0.598 m e a diagonal mede 0.846 m
    public static int calculateSqaure (float stepSize, boolean diagonal){
        return (diagonal) ? (int) Math.ceil(stepSize / 0.846) : (int) Math.floor(stepSize /
       0.598);
}
}
```

A classe `Compass` também contém métodos auxiliares, relacionados com a estimativa da direção, com base na orientação (valor do azimute).

O código desta classe é o seguinte:

```
public class Compass {

  // Com base no valor do azimute, estima-se a direcao do utilzador em relacao aos eixos
    // Trabalha em radiano (rad)
    static String estimateUserDirection(float azimuth, float forwardAzimuth){
        float difference = adjustAzimuth(forwardAzimuth, azimuth);

        // O utilizador esta voltado para a esquerda da sala
        if (difference < -(Math.PI/4) && difference >= -((3*Math.PI)/4)){
            return "L";
        }
        // O utilizador esta voltado para a direita da sala
        else if (difference >= (Math.PI/4) && difference < ((3*Math.PI)/4)){
            return "R";
        }
        // O utilizador esta voltado para a frente da sala
        else if (difference >= -(Math.PI/4) && difference < Math.PI/4) {
            return "F";
        }
        // O utilizador esta voltado para as traseiras da sala
        else if (difference >= ((3*Math.PI)/4) || difference < -((3*Math.PI)/4)) {
            return "B";
        }

        return "O";     // Direcao nao considerada
    }

  // Determina a situacao atual da direcao do percurso do utilizador
  // Se a direcao for a mesma ou tiver sido alterada
  // Retorna um numero que identifica a situacao atual
  // Trabalha em radiano (rad)
  public static int orientationSituation (float azimuth, float turnAzimuth){

        float difference = adjustAzimuth(turnAzimuth, azimuth);

        // O utilizador esta a andar na perpendicular a um dos eixos, apos virar para o
    lado esquerdo
        if (difference < -((3*Math.PI)/8) && difference > -((5*Math.PI)/8)){
            return 1;
        }
        // O utilizador esta a andar na perpendicular a um dos eixos, apos virar para o
    lado direito
        else if (difference > ((3*Math.PI)/8) && difference < ((5*Math.PI)/8)){
            return 3;
        }
```

```
44          // O utilizador afastou-se do caminho original e encontra-se a caminhar
    diagonalmente para a esquerda
45          else if (difference < -(Math.PI/8) && difference > -((3*Math.PI)/8)) {
46              return 2;
47          }
48          // O utilizador afastou-se do caminho original e encontra-se a caminhar
    diagonalmente para a direita
49          else if (difference > (Math.PI/8) && difference < ((3*Math.PI)/8)) {
50              return 4;
51          } else {
52              return -1;
53          }
54      }
55
56  // Se o valor do azimute estiver acima ou abaixo dos limites dos referencial definido,
57  // o valor deve ser ajustado se for maior que PI e menor que -PI
58  // Inputs e outputs em radiano (rad)
59  public static float adjustAzimuth (float previousAzimuth, float azimuth){
60      float azimuthDiff = azimuth - previousAzimuth;      // Calcula a diferenca entre os
    dois valores guardados
61
62      // Verifica se o resultado esta contido no referencial
63      if (azimuthDiff < -(Math.PI)){
64          azimuthDiff = (float) (azimuthDiff + (2*Math.PI));  // Se o resultado for menor
    que -PI, adiciona-se 2*PI
65      } else if (azimuthDiff > Math.PI){
66          azimuthDiff = (float) (azimuthDiff - (2*Math.PI));  // Se o resultado for maior
    que PI, subtrai-se 2*PI
67      }
68
69      return azimuthDiff;
70  }
71
72  // Se sign tiver o valor TRUE (+), entao o valor retornado sera positivo, se por
    contrario, for FALSE (-), entao o valor retornado sera negativo
73  static int signToValue(boolean sign, int value){
74      return (!sign) ? -1 * value : value;
75  }
76 }
```

B.5 Servidor - Processamento de dados (secção 4.5)

B.5.1 Dados de treino e validação

Para adaptar os dados recolhidos, ou *fingerprints*, para conjuntos de treino e validação, para o modelo de *machine learning*, foi usado o seguinte programa escrito em Python:

```
# Le os ficheiros das fingerprints gerados durante a fase de treino e retorna os ficheiros de treino e validacao adaptados
# para um aplicacao baseada em TensorFlow
# Criado por Pedro Silva in 3/7/2019
import os
import re
import csv
import sys

basepath = '' # Caminho para ficheiro omitido

header = 'BLE1,BLE2,BLE3,BLE4,MAGN_X,MAGN_Y,MAGN_Z,AZIMUTH,X,Y,DIRECTION\n' # Cabecalho
training_data = []      # Medidas para treino
validation_data = []    # Medidas para validacao

direction_dict = {'Frente':'F', 'Tras':'B', 'Esquerda':'L', 'Direita':'R'}      # Para traduzir o nome do ficheiro na direcao

# Calcula a media de todos os valores nas 30 medidas
def calculateAverageForMeasure (measures):
    sum = [0]*8
    for m in measures:
        k=0
        while k < len(m):
            sum[k] += m[k]
            k+=1

    # Divide a soma de todos os valores pelo numero de medidas adicionado
    l=0
    while l < len(sum):
        sum[l] = sum[l]/len(measures)   # Divisao
        l+=1

    return sum

# Le o ficheiro correspondente a um certa posicao e direcao
# e no cria um fingerprint valida para o treino e validacao do modelo
def createFingerprint (position, file):
    i=0
    k=0
    extension = False
    fingerprint_calc = []
    for line in file:
        line = line.rstrip('\n')  # Remove '\n' do final da linha

        sample = arrangeMeasureForSet(line, position)

    # Apenas os primeiros 30 elementos sao considerados para os dados de treino, o resto vai para a validacao
```

```
47          if i<31:
48              training_data.append(sample + position)
49              fingerprint_calc.append([float(i) for i in sample])     # Converte strings em
     float
50              i+=1
51          elif k<31:
52              validation_data.append(sample + position)
53              extension = True    # Indica que havia mais de 30 medidas no ficheiro
54              k+=1
55
56      # Calcula a media de todos os valores nas medidas para uso na validacao no caso de
     existirem menos de 30 medidas
57      if not extension:
58          avg = calculateAverageForMeasure(fingerprint_calc)  # Calcula a media
59          avg_str = [str(g) for g in avg] + position          # Adiciona a informacao da
     posicao
60          validation_data.append(avg_str)
61
62  # Le cada uma das medidas fingerprint e reorganiza-as em ficheiros de treino e validacao
63  def arrangeMeasureForSet (line, position):
64      measure = line.split(',')   # Divide os elementos da linha para um lista
65      measure = measure[1:]  # Remove o numero da linha na primeira posicao
66
67      # Remove os dados do acelerometro e do giroscopio
68      measure_part_one = measure[0:4]
69      measure_part_two = measure[10:14]
70      measure = measure_part_one + measure_part_two
71
72      sample = []
73      for e in measure:
74          sample.append(e)  # Adiciona cada um dos elementos da fingerprint a lista
75
76      return sample
77
78  # Abre ficheiros txt
79  def open_txt_file(fingerprint_file, directory, file_path):
80      if not fingerprint_file.startswith('.'):
81          fingerprint_name = re.split("\_|\.", fingerprint_file)
82          position = fingerprint_name[1:3]
83
84          position.append(direction_dict[directory])
85
86          fingerprint_path = file_path + '/' + fingerprint_file
87          createFingerprint(position, open(fingerprint_path, "r"))
88
89  # Escreve todas as medidas num ficheiro
90  def writeToFile (file_name, header, data):
91      f = open(file_name, 'w+')
92      f.write(header)
```

```
    for m in data:
        s = ",".join(m)
        f.write(s +'\n')
    f.close()


# Abre os diretorios e opera sobre os ficheiros da fase de treino
for entry in os.listdir(basepath):
    if os.path.isdir(os.path.join(basepath, entry)):
        file_name = list(entry)
        for directory in os.listdir(basepath + entry):
            if not directory.startswith('.'):
                for fingerprint in os.listdir(basepath + entry + '/' + directory):
                    open_txt_file(fingerprint, directory, (basepath + entry + '/' +
    directory))
        print ('Row ' + entry + ' completed', end='\n')

# Write lists to file
writeToFile('training_data.csv', header, training_data)
writeToFile('validation_data.csv', header, validation_data)
print ('Training and validation data written to files')
```

B.5.2 Criação do modelo de *machine learning*

O solução escolhida para a criação do modelo de *machine learning*, foi adaptada de [47].

O código implementado foi o seguinte:

```
# Machine learning para fingerprinting utilizando TensorFlow
# Codigo adaptado das seguintes fontes:
# - https://www.tensorflow.org/tutorials/keras/basic_classification
# - https://www.tensorflow.org/tutorials/keras/overfit_and_underfit
# - https://www.tensorflow.org/tutorials/keras/save_and_load
# - https://github.com/mallsk23/place_recognition_wifi_fingerprints_deep_learning
# Mudancas foram feitas de modo a adaptar o codigo para a situacao pretendida

from __future__ import absolute_import, division, print_function, unicode_literals

# TensorFlow e tf.keras
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import scale
from keras import metrics
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Bibliotecas de ajuda
import pandas as pd
import numpy as np
import time
```

```
23
24 print(tf.__version__)
25
26 labels_list = []
27
28 # Nome dos ficheiros de treino e validacao
29 path_train_file = 'training_data.csv'
30 path_validation = 'validation_data.csv'
31
32 # LER E PREPARAR DADOS DE TREINO ************************************
33
34 # Indicar explicitamente a presenca de um header (cabecalho) para poder substituir nomes
35 train_df = pd.read_csv(path_train_file,header = 0)
36 train_AP_strengths = train_df.ix[:,:8]     #seleciona os primeiros 8 valores (valores do
       RSSI dos beacons, do magnetometro e azimute)
37
38 # Na versao original
39 train_AP_features = scale(np.asarray(train_AP_strengths))  # Adaptar train_AP_strengths
       para um array np e padronizar os valores
40
41 # Para fazer previsoes
42 train_AP_features = np.asarray(train_AP_strengths)  # Adaptar train_AP_strengths para um
       array np
43
44 # Criacao dos labels (classes):
45 # Os seguintes objetos sao na verdade pandas.core.series.Series objects
46 x_position_str = train_df["X"].map(str)     #converte todas as coordenadas x para strings
47 y_position_str = train_df["Y"].map(str)     #converte todas as coordenadas y para strings
48 direction_str = train_df["DIRECTION"].map(str)      #converte todas a direcoes para
       strings
49
50 res = x_position_str + '_' + y_position_str + '_' + direction_str    #concatenacao dos
       elementos de X,Y e DIRECTION
51 train_labels = np.asarray(res)      # Adapta res para um array np
52
53 #converte as lables para valores categoricos, dummy_labels e do tipo 'pandas.core.frame.
       DataFrame'
54 dummy_labels = pd.get_dummies(train_labels)
55 train_labels = np.asarray(dummy_labels)
56
57 # Divide os dados de treino em dados de treino e validacao inicial:
58 train_val_split = np.random.rand(len(train_AP_features))        #gera len(
       train_AP_features) floats entre 0 e 1
59 #converte train_val_split num array de booleans: if elem < 0.7 = true, else: false
60 train_val_split = train_val_split < 0.70    #deve conter ~70% de true
61
62 # O conjunto de treino divide-se em treino+validacao
63 train_X = train_AP_features[train_val_split]
64 train_y = train_labels[train_val_split]
```

```
val_X = train_AP_features[~train_val_split]
val_y = train_labels[~train_val_split]

#Transformar o conjunto de validacao num conjunto de treino
test_df = pd.read_csv(path_validation,header = 0)
test_AP_features = scale(np.asarray(test_df.ix[:,0:8]))   # Na versao original
test_AP_features = np.asarray(test_df.ix[:,0:8])      # Para fazer previsoes
test_labels = np.asarray(test_df["X"].map(str) + '_' + test_df["Y"].map(str) + '_' +
    test_df["DIRECTION"].map(str)) #converte todas as coordenadas x,y e direcao para
    string e concatena-as
test_labels = np.asarray(pd.get_dummies(test_labels))

# CRIACAO DO MODELO E VALIDACAO INICIAL ***********************************

nb_epochs = 20
batch_size = 10
input_size = 8
num_classes = 474

def encoder():
    model = Sequential()
    model.add(Dense(256, input_dim=input_size, activation='tanh', bias=True))
    model.add(Dense(128, activation='tanh', bias=True))
    model.add(Dense(64, activation='tanh', bias=True))
    return model

def decoder(e):
    e.add(Dense(128, input_dim=64, activation='tanh', bias=True))
    e.add(Dense(256, activation='tanh', bias=True))
    e.add(Dense(num_classes, activation='tanh', bias=True))
    e.compile(optimizer='adam', loss='mse')
    return e

e = encoder()

d = decoder(e)
d.fit(train_AP_features, train_labels, nb_epoch=nb_epochs, batch_size=batch_size)  #
    Fornece ao modelo os dados de treino

def classifier(d):
    num_to_remove = 3
    for i in range(num_to_remove):
        d.pop()
    d.add(Dense(128, input_dim=64, activation='tanh', bias=True))
    d.add(Dense(128, activation='tanh', bias=True))
    d.add(Dense(num_classes, activation='softmax', bias=True))
    d.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return d

```

```
111 c = classifier(d)
112 c.fit(train_AP_features, train_labels, validation_data=(val_X, val_y), nb_epoch=nb_epochs
        , batch_size=batch_size)
113
114 loss, acc = c.evaluate(test_AP_features, test_labels)
115 print(loss, acc)
116
117 # Testar as previsoes
118 predictions = c.predict(test_AP_features)
119
120 matches=0 # numero de labels certos
121 mismatches=0  # numero de labels errados
122 index=0
123 while index < len(test_AP_features):
124     pred_max = np.argmax(predictions[index])  # Determinar a label prevista com base no
        indice com maior valor de confianca
125     actual_max = np.argmax(test_labels[index])  # Determina a verdadeira label
126
127   # Compara as labels
128     if pred_max==actual_max:
129         matches+=1
130         print (str(pred_max) + " == " + str(actual_max))
131     else:
132         mismatches+=1
133
134     index+=1;
135
136 print ('Matches= ' + str(matches) + ' and mismatches= ' + str(mismatches) + '\n')
137
138
139 # Guardar o modelo
140 # Guarda o modelo inteiro num ficheiro HDF5
141 c.save('model.h5')
142
143 # Recria o mesmo exato modelo, incluindo weights e optimizer.
144 new_model = keras.models.load_model('model.h5')
145 new_model.summary()
146
147 # Avalia o modelo restaurado
148 loss, acc = new_model.evaluate(test_AP_features, test_labels)
149 print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

Anexo C

Resultados

C.1 Avaliação do Cálculo da trajetória (secção 5.1)

C.1.1 *Output* em RAW do ficheiro gerado no teste de avaliação da deteção de curvas (secção 5.1.2)

```
Left turn detected with max= 2.48575
Unlocked with value 0.09299079
Right turn detected with mix= -2.7182112
Unlocked with value 0.00746966
Left turn detected with max= 3.088674
Unlocked with value 0.0911582
Right turn detected with mix= -1.8782713
Unlocked with value 0.00746966
Right turn detected with mix= -2.3119855
Unlocked with value 0.004415334
Right turn detected with mix= -2.913688
Unlocked with value 0.00746966
Left turn detected with max= 2.3134859
Unlocked with value 0.09482339
Left turn detected with max= 2.4313831
Unlocked with value 0.09604512
Right turn detected with mix= -2.5416708
Unlocked with value 0.011745717
Left turn detected with max= 2.765526
Unlocked with value 0.09726685
```

C.1.2 *Output* em RAW dos ficheiros gerados no teste de avaliação da bússola (secção 5.1.3)

O formato dos dados guardados no ficheiro é o seguinte [50]:

```
Ponto cardeal, azimute (radiano), azimute (graus), orientação do norte ou sul e o ângulo a este ou oeste
```

Dispositivo fixo em cima da mesa

Bússola normal:

```
N,0.047725156,3,N 3 E
NE,0.889503,51,N 51 E
E,1.6814488,96,S 84 E
SE,2.3833117,137,S 43 E
S,-3.114638,-178,S 2 W
SW,-2.3305705,-134,S 46 W
W,-1.6478666,-94,S 86 W
NW,-0.8370953,-48,N 48 W
```

Bússola com fusão de sensores:

```
N,0.06915795,4,N 4 E
NE,0.9212533,53,N 53 E
E,1.6620795,95,S 85 E
SE,2.4066088,138,S 42 E
S,-2.6005502,-149,S 31 W
SW,-2.3473856,-134,S 46 W
W,-1.6357641,-94,S 86 W
NW,-0.80586314,-46,N 46 W
```

Utilizador a segurar o dispositivo

Bússola normal:

```
N,-0.07980154,-5,N 5 W
NE,0.69548875,40,N 40 E
E,1.5240033,87,N 87 E
SE,2.2349496,128,S 52 E
S,3.0731153,176,S 4 E
SW,-2.4378886,-140,S 40 W
W,-1.7165598,-98,S 82 W
NW,-0.89886326,-52,N 52 W
```

Bússola com fusão de sensores:

```
N,-0.018328954,-1,N 1 W
NE,0.7299832,42,N 42 E
E,1.5318103,88,N 88 E
SE,2.171878,124,S 56 E
S,2.7590163,158,S 22 E
SW,-2.3067172,-132,S 48 W
W,-1.6435639,-94,S 86 W
NW,-0.87955093,-50,N 50 W
```

C.2 Avaliação do *Fingerprinting* - Análise da cobertura (secção 5.2.1)

C.2.1 Mapas para o *Beacon* 1

Colocado na posição (8, 10-11).

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-73,0333333	-70		APPLE		OBS.	OBS.	-72,1666667	-76	15
14	-75,6	-68,3	-70	-78,7333333	-72,8666667	-84	-69,8333333	-73,4	-76,9333333	14
13	-72	-67	-69,6666667	-71	-70,9	-77,1	-69,4333333	-67,6666667	-74,3333333	13
12	-71,4	PCs		-71,4666667	-75,1333333	-70,5333333	-67,3666667	-63	-76,3333333	12
11	-78	-68,9666667	-65,1	-67,3666667	-70,9333333		-81	-60,2666667	OBS.	11
10	-67,5	-69,3666667	-66,0333333	-64	-66,9333333	REDES-A	-70,8	-60,9333333	-72,1333333	10
9	-66	-70,9666667		-76,6666667	-73,2666667		-70,2	-67	-70,3333333	9
8	-67,7666667	-71,0666667	COMPONETES	-76,7333333	-77,7		-70,2666667	-67	-66,9	8
7	-72	-76		-71	-68,5333333	REDES-B	-70	-87	-73	7
6	-71,3333333	-75,2	-73,2333333	-70,2	-76,6		-71	-73,0333333	-75,2666667	6
5	-71,0666667	-73	-77	-70,8666667	-75,3333333		-71,2666667	-68	-85,1333333	5
4	-75,2	-77,8	OBS.	-71,2	-72,2	REDES-C	-71,9333333	-72,1	-80,2666667	4
3	-73,3333333	-75,9		-75,6666667	-79		-71	-76,2333333	-78,5333333	3
2	-75,4	-72,9666667	-75,2	-76,2333333	-72,1666667	-75,3666667	-72,9666667	-79,8666667	-77,8666667	2
1	-78,3	-78,6666667	-73,3333333	-79,2	-75,9333333	-82	-70,4333333	-70,8666667	-83	1
0	-75,2333333	-75	-79,7666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.1: Mapa com a média dos valores RSSI registados para o *beacon* 1, em cada posição, com direção para a direita.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-72	-70,53333333		APPLE		OBS.	OBS.	-76,8	-69,66666667	15
14	-74	-72,3	-75,3	-76,76666667	-69,73333333	-72,73333333	-76,4	-69,56666667	-73,33333333	14
13	-90	-71,86666667	-69	-74	-72,2	-83	-73	-69	-69,2	13
12	-68,46666667	PCs		-64	-65,7	-73,26666667	-66,03333333	-71,2	-64,46666667	12
11	-73	-75,53333333	-73,26666667	-66,4	-67,93333333		-70	-65,36666667	OBS.	11
10	-71,13333333	-75,53333333	-75,93333333	-77,3	-63	REDES-A	-62,76666667	-68,8	-68,4	10
9	-70,83333333	-68		-67	-66,2		-78,63333333	-72,46666667	-77	9
8	-77,5	-65,06666667	COMPONETES	-71,13333333	-75,13333333		-78,4	-88	-73	8
7	-80,9	-71,3		-69,9	-73,93333333	REDES-B	-76	-74,53333333	-71,6	7
6	-69,73333333	-70,6	-68,06666667	-69,06666667	-70,26666667		-74,8	-74,66666667	-81,53333333	6
5	-71,5	-70	-75	-70	-74,06666667		-85,73333333	-72,2	-81	5
4	-75,53333333	-78,53333333	OBS.	-73,33333333	-77,96666667	REDES-C	-69,8	-68,4	-76	4
3	-75,36666667	-72		-82	-83,36666667		-76,2	-82,23333333	-75,2	3
2	-73,06666667	-74,4	-69,83333333	-71,86666667	-76,86666667	-73,4	-76,76666667	-80,13333333	-82,03333333	2
1	-74,46666667	-77	-69	-79	-80	-76	-77,46666667	-77	-76,26666667	1
0	-83,5	-75,06666667	n/d	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.2: Mapa com a média dos valores RSSI registados para o *beacon* 1, em cada posição, com direção para trás.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-73,0333333	-70		APPLE		OBS.	OBS.	-72,1666667	-76	15
14	-75,6	-68,3	-70	-78,7333333	-72,8666667	-84	-69,8333333	-73,4	-76,9333333	14
13	-72	-67	-69,6666667	-71	-70,9	-77,1	-69,4333333	-67,6666667	-74,3333333	13
12	-71,4	PCs		-71,4666667	-75,1333333	-70,5333333	-67,3666667	-63	-76,3333333	12
11	-78	-68,9666667	-65,1	-67,3666667	-70,9333333		-81	-60,2666667	OBS.	11
10	-67,5	-69,3666667	-66,0333333	-64	-66,9333333	REDES-A	-70,8	-60,9333333	-72,1333333	10
9	-66	-70,9666667		-76,6666667	-73,2666667		-70,2	-67	-70,3333333	9
8	-67,7666667	-71,0666667	COMPONETES	-76,7333333	-77,7		-70,2666667	-67	-66,9	8
7	-72	-76		-71	-68,5333333	REDES-B	-70	-87	-73	7
6	-71,3333333	-75,2	-73,2333333	-70,2	-76,6		-71	-73,0333333	-75,2666667	6
5	-71,0666667	-73	-77	-70,8666667	-75,3333333		-71,2666667	-68	-85,1333333	5
4	-75,2	-77,8	OBS.	-71,2	-72,2	REDES-C	-71,9333333	-72,1	-80,2666667	4
3	-73,3333333	-75,9		-75,6666667	-79		-71	-76,2333333	-78,5333333	3
2	-75,4	-72,9666667	-75,2	-76,2333333	-72,1666667	-75,3666667	-72,9666667	-79,8666667	-77,8666667	2
1	-78,3	-78,6666667	-73,3333333	-79,2	-75,9333333	-82	-70,4333333	-70,8666667	-83	1
0	-75,2333333	-75	-79,7666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.3: Mapa com a média dos valores RSSI registados para o *beacon* 1, em cada posição, com direção para a direita.

C.2.2 Mapas para o *Beacon* 2

Colocado na posição (5, 15).

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-72,13333333	-71		APPLE		OBS.	OBS.	-72,9	-74	15
14	-76,13333333	-74	-73,2	-66,43333333	-64,2	-70,3	-69,66666667	-74,33333333	-76,33333333	14
13	-70	-73,86666667	-69,73333333	-69,66666667	-72	-70	-69,93333333	-70,06666667	-72,66666667	13
12	-67,53333333	PCs		-76,13333333	-72,66666667	-77,33333333	-69,26666667	-76	-72,76666667	12
11	-70,13333333	-73,2	-70,83333333	-73,73333333	-76,9		-70,1	-74,4	OBS.	11
10	-70,4	-68	-79,4	-77,06666667	-79,86666667	REDES-A	-74,23333333	-69,96666667	-74,13333333	10
9	-74,2	-69,53333333		-74,73333333	-75,46666667		-68,2	-80,4	-87,26666667	9
8	-72,73333333	-76,8	COMPONETES	-76	-75,2		-72,03333333	-75,36666667	-75,66666667	8
7	-70,8	-86		-76,8	-75,46666667	REDES-B	-69,73333333	-78	-83,6	7
6	-74,3	-79	-74,43333333	-76,2	-79		-78,76666667	-78,43333333	-78,53333333	6
5	-71,26666667	-73,6	-73,26666667	-74,93333333	-78,13333333		-76,76666667	-75,73333333	-76,4	5
4	-90,5	-75,06666667	OBS.	-79,83333333	-82	REDES-C	-79,9	-78,03333333	-80,13333333	4
3	-77	-73,5		-81	-76,23333333		-74,43333333	-82,36666667	-74,8	3
2	-72,93333333	-73,36666667	-71	-75,1	-81,86666667	-77,16666667	-83,56666667	-75,96666667	-76,2	2
1	-82,66666667	-77,03333333	-82,53333333	-76,03333333	-75,06666667	-73	-76,83333333	-77,53333333	-80	1
0	-78,63333333	-74,56666667	n/d	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.4: Mapa com a média dos valores RSSI registados para o *beacon* 2, em cada posição, com direção para a frente.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-72,26666667	-76,13333333		APPLE		OBS.	OBS.	-70,53333333	-76	15
14	-79,66666667	-77,63333333	-72	-72,86666667	-72,46666667	-71	-71,76666667	-74,33333333	-75,66666667	14
13	-76,73333333	-72	-68,86666667	-73	-74,7	-74,56666667	-69,06666667	-80,4	-73,33333333	13
12	-83,66666667	PCs		-78,86666667	-74,96666667	-72,46666667	-66,73333333	-75,23333333	-79,4	12
11	-81	-84,6	-71,26666667	-72,53333333	-71,03333333		-68,96666667	-80,16666667	OBS.	11
10	-75,26666667	-77,23333333	-75,96666667	-79	-72,2	REDES-A	-71,7	-70,03333333	-78,53333333	10
9	-79	-74		-76,53333333	-74,13333333		-73,9	-76	-76,43333333	9
8	-76,73333333	-80,8	COMPONETES	-72,1	-72,2		-73	-75	-71,1	8
7	-78,1	-82		-85,26666667	-74,16666667	REDES-B	-82,93333333	-71	-75,3	7
6	-73,73333333	-79,53333333	-80,53333333	-75,53333333	-81,5		-70	-75,33333333	-73,93333333	6
5	-78,5	-75	-79	-71,6	-76,6		-80,06666667	-69,96666667	-69,06666667	5
4	-77,13333333	-80,96666667	OBS.	-83,8	-70,86666667	REDES-C	-76	-72,2	-76,36666667	4
3	-77	-76,96666667		-76,3	-80		-70,13333333	-72,06666667	-75,8	3
2	-79,26666667	-78	-74,9	-75,06666667	-78,86666667	-79,2	-76,46666667	-75,76666667	-75,96666667	2
1	-81,46666667	-80,93333333	-76,46666667	-79,73333333	-74,26666667	-78,03333333	-81,03333333	-73,36666667	-79	1
0	-76,06666667	-86	-82	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.5: Mapa com a média dos valores RSSI registados para o *beacon* 2, em cada posição, com direção para a direita.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-79,03333333	-82,53333333		APPLE		OBS.	OBS.	-68	-68,06666667	15
14	-74,96666667	-74,16666667	-68,33333333	-66,2	-70,03333333	-71,93333333	-75,16666667	-67,13333333	-68,26666667	14
13	-73	-78,06666667	-73,5	-75,03333333	-71,5	-79,33333333	-73,73333333	-68,2	-72,96666667	13
12	-72,93333333	PCs		-73,9	-84,7	-74,7	-77,3	-81,53333333	-72,8	12
11	-78	-80,7	-75,73333333	-82,6	-78,3		-78,66666667	-70,4	OBS.	11
10	-78,4	-76,43333333	-78,2	-75,46666667	-78,33333333	REDES-A	-75,46666667	-81,73333333	-73,6	10
9	-78,6	-89		-76	-78,13333333		-84,13333333	-78,86666667	-79	9
8	-77,93333333	-75,36666667	COMPONETES	-77,26666667	-72		-82,33333333	-78	-76	8
7	-75,86666667	-79,53333333		-79,93333333	-79	REDES-B	-80	-79,13333333	-79,26666667	7
6	-86,86666667	-76,66666667	-76,86666667	-78	-78,33333333		-79,53333333	-79,23333333	-72,76666667	6
5	-79,56666667	-84	-76,66666667	-83	-80,53333333		-75,06666667	-75,03333333	-78,63333333	5
4	-76,7	-73,83333333	OBS.	-79,26666667	-76,3	REDES-C	-89,2	-79,16666667	-80,8	4
3	-86,4	-84		-87	-83,53333333		-81,76666667	-82,2	-77,8	3
2	-79,13333333	-84,26666667	-80,7	-83,03333333	-79,16666667	-81,6	-79,06666667	-78,03333333	-81,2	2
1	-77,4	-81,96666667	-81	-76,96666667	-81	-79	-80,1	-79,26666667	-80,8	1
0	-83,6	-81,53333333	n/d	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.6: Mapa com a média dos valores RSSI registados para o *beacon* 2, em cada posição, com direção para trás.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-79	-71,2		APPLE		OBS.	OBS.	-66,2	-70,2	15
14	-90,96666667	-71,9	-68,7	-68	-66,8	-60,5	-64	-79	-68,43333333	14
13	-89,5	-76,4	-72	-64,63333333	-68,93333333	-69,06666667	-60	-65,33333333	-72,7	13
12	-76,86666667	PCs		-75,4	-71,86666667	-72,66666667	-66,2	-76,73333333	-74,93333333	12
11	-68,43333333	-76,13333333	-71,8	-66,86666667	-72,6		-65,46666667	-67,86666667	OBS.	11
10	-78,73333333	-74,66666667	-73,66666667	-71,23333333	-84,13333333	REDES-A	-78,26666667	-70,73333333	-79	10
9	-74,66666667	-76		-70,46666667	-69,03333333		-68,13333333	-66,1	-72,6	9
8	-76	-85	COMPONETES	-75,33333333	-77,06666667		-69	-68,26666667	-82,1	8
7	-71,66666667	-76,4		-75	-77,23333333	REDES-B	-71,1	-72,23333333	-74,8	7
6	-80,53333333	-77,26666667	-72,33333333	-73	-73,53333333		-70,5	-73,06666667	-74,6	6
5	-72	-80,73333333	-67	-73,93333333	-73,1		-67,06666667	-66,46666667	-74,4	5
4	-73,2	-70,43333333	OBS.	-76,53333333	-73,53333333	REDES-C	-72,4	-75,36666667	-77,86666667	4
3	-78,33333333	-74,83333333		-75,93333333	-92		-81,46666667	-70,73333333	-76	3
2	-78,86666667	-74,46666667	-78,86666667	-86,33333333	-80,96666667	-76	-74,23333333	-71	-75,53333333	2
1	-76,86666667	-73	-73,13333333	-78,2	-79	-72,86666667	-88,5	-75,8	-74,16666667	1
0	-76,46666667	-83,73333333	-78	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.7: Mapa com a média dos valores RSSI registados para o *beacon* 2, em cada posição, com direção para a esquerda.

C.2.3 Mapas para o *Beacon* 3

Colocado na posição (5-6, 0).

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-69,46666667	-68,2		APPLE		OBS.	OBS.	-64,83333333	-63	15
14	-66,73333333	-72,06666667	-75	-70,8	-70,4	-74	-68,06666667	-72,5	-69,8	14
13	-72	-67	-70,43333333	-72	-68,03333333	-67,66666667	-66,46666667	-73,7	-76,36666667	13
12	-72,23333333	PCs		-67,6	-68,13333333	-70,63333333	-64,06666667	-72,2	-67,13333333	12
11	-70	-69,13333333	-76,8	-62,6	-75,33333333		-68,86666667	-71,4	OBS.	11
10	-75,36666667	-68,2	-68	-63	-73,46666667	REDES-A	-67,66666667	-67	-69,16666667	10
9	-68	-68		-62,5	-65,73333333		-62,26666667	-81	-64,33333333	9
8	-68,06666667	-72,73333333	COMPONETES	-65,56666667	-61,1		-63,3	-65	-55	8
7	-64	-66		-65,2	-68,46666667	REDES-B	-73,63333333	-63	-58,86666667	7
6	-64,33333333	-68,06666667	-62,8	-68,73333333	-64,36666667		-61	-65,26666667	-71,76666667	6
5	-69,93333333	-61	-62	-75,93333333	-59,4		-63,8	-60	-56,13333333	5
4	-83,53333333	-61,1	OBS.	-68,86666667	-64,06666667	REDES-C	-60,1	-56,86666667	-70,1	4
3	-68,86666667	-69,2		-61,96666667	-60,8		-58,93333333	-60,86666667	-61,76666667	3
2	-71,06666667	-70,16666667	-68,66666667	-72	-56,73333333	-64,43333333	-60,13333333	-62,13333333	-59,4	2
1	-76,06666667	-73,6	-63,36666667	-68,33333333	-59,2	-67,06666667	-53,9	-76,2	-66	1
0	-74,93333333	-67	-67,86666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.8: Mapa com a média dos valores RSSI registados para o *beacon* 3, em cada posição, com direção para a frente.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-69,46666667	-68,2		APPLE		OBS.	OBS.	-64,83333333	-63	15
14	-66,73333333	-72,06666667	-75	-70,8	-70,4	-74	-68,06666667	-72,5	-69,8	14
13	-72	-67	-70,43333333	-72	-68,03333333	-67,66666667	-66,46666667	-73,7	-76,36666667	13
12	-72,23333333	PCs		-67,6	-68,13333333	-70,63333333	-64,06666667	-72,2	-67,13333333	12
11	-70	-69,13333333	-76,8	-62,6	-75,33333333		-68,86666667	-71,4	OBS.	11
10	-75,36666667	-68,2	-68	-63	-73,46666667	REDES-A	-67,66666667	-67	-69,16666667	10
9	-68	-68		-62,5	-65,73333333		-62,26666667	-81	-64,33333333	9
8	-68,06666667	-72,73333333	COMPONETES	-65,56666667	-61,1		-63,3	-65	-55	8
7	-64	-66		-65,2	-68,46666667	REDES-B	-73,63333333	-63	-58,86666667	7
6	-64,33333333	-68,06666667	-62,8	-68,73333333	-64,36666667		-61	-65,26666667	-71,76666667	6
5	-69,93333333	-61	-62	-75,93333333	-59,4		-63,8	-60	-56,13333333	5
4	-83,53333333	-61,1	OBS.	-68,86666667	-64,06666667	REDES-C	-60,1	-56,86666667	-70,1	4
3	-68,86666667	-69,2		-61,96666667	-60,8		-58,93333333	-60,86666667	-61,76666667	3
2	-71,06666667	-70,16666667	-68,66666667	-72	-56,73333333	-64,43333333	-60,13333333	-62,13333333	-59,4	2
1	-76,06666667	-73,6	-63,36666667	-68,33333333	-59,2	-67,06666667	-53,9	-76,2	-66	1
0	-74,93333333	-67	-67,86666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.9: Mapa com a média dos valores RSSI registados para o *beacon* 3, em cada posição, com direção para a direita.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-75	-74,46666667		APPLE		OBS.	OBS.	-69,16666667	-72,73333333	15
14	-66,26666667	-74,93333333	-71,9	-70	-72,16666667	-74,4	-68,56666667	-78,4	-73,6	14
13	-67	-74,33333333	-69	-71	-77,6	-66,66666667	-73,06666667	-65	-68,8	13
12	-71,1	PCs		-71,66666667	-65	-78,93333333	-71,16666667	-70,83333333	-68,83333333	12
11	-73	-64,53333333	-64,33333333	-74,86666667	-78,46666667		-68,03333333	-73,46666667	OBS.	11
10	-69,66666667	-78,4	-66,53333333	-76	-69,86666667	REDES-A	-73,4	-76,2	-69,33333333	10
9	-69,66666667	-70		-67	-70,26666667		-74,1	-72,26666667	-69	9
8	-65,8	-76,2	COMPONETES	-75,53333333	-79		-65,43333333	-74	-72	8
7	-67,13333333	-80,16666667		-70,6	-67,03333333	REDES-B	-67,66666667	-62	-64,43333333	7
6	-68,46666667	-73,73333333	-66,06666667	-69,26666667	-68,66666667		-68,76666667	-61,86666667	-73,46666667	6
5	-73,53333333	-69	-67,66666667	-70	-65,33333333		-62,83333333	-56,03333333	-76,5	5
4	-65,86666667	-65,96666667	OBS.	-66,33333333	-81,2	REDES-C	-68,36666667	-62,26666667	-59,6	4
3	-67,33333333	-73		-83	-60,5		-62,33333333	-74,66666667	-70,06666667	3
2	-81,46666667	-71,6	-70,3	-66,66666667	-69,1	-59,73333333	-63,46666667	-61,93333333	-65,4	2
1	-70,73333333	-82	-70	-61,16666667	-69	-74	-62,9	-57,9	-65,26666667	1
0	-71,06666667	-77,66666667	n/d	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.10: Mapa com a média dos valores RSSI registados para o *beacon* 3, em cada posição, com direção para trás.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-73	-71,66666667		APPLE		OBS.	OBS.	-68,53333333	-77,3	15
14	-63,8	-69,93333333	-74,53333333	-71	-70,8	-71,76666667	-75,33333333	-75,73333333	-69,76666667	14
13	-69,66666667	-69,2	-69,86666667	-73,56666667	-68,83333333	-71,73333333	-70	-71,7	-75,5	13
12	-74	PCs		-71,13333333	-68,73333333	-73,46666667	-68	-62,06666667	-71,36666667	12
11	-65,8	-73,8	-67,4	-70,06666667	-74,86666667		-72,9	-70,56666667	OBS.	11
10	-64	-65,43333333	-66,4	-72,36666667	-78,93333333	REDES-A	-64,46666667	-67,1	-68	10
9	-70,66666667	-88		-74,7	-71,73333333		-66,93333333	-67,76666667	-69,7	9
8	-73	-76	COMPONETES	-64,36666667	-73,13333333		-70,83333333	-62,96666667	-75,26666667	8
7	-72,53333333	-62,66666667		-63,86666667	-70,53333333	REDES-B	-71	-71,1	-88,33333333	7
6	-77,86666667	-65,23333333	-61	-62	-68,43333333		-69,2	-75,3	-78,93333333	6
5	-75,63333333	-79,06666667	-68	-76	-66,66666667		-62,2	-67,86666667	-62,16666667	5
4	-74,33333333	-73,2	OBS.	-61,8	-60,66666667	REDES-C	-65,96666667	-62,53333333	-65,1	4
3	-70,26666667	-65,66666667		-66,56666667	-68,73333333		-62,86666667	-66,53333333	-61	3
2	-78,06666667	-72,93333333	-61,96666667	-68,6	-63,13333333	-75	-63,26666667	-69	-72,8	2
1	-73,46666667	-74	-67,26666667	-67,63333333	-67	-61,6	-66,33333333	-58,46666667	-69,06666667	1
0	-69,66666667	-80,96666667	-66	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.11: Mapa com a média dos valores RSSI registados para o *beacon* 3, em cada posição, com direção para a esquerda.

C.2.4 Mapas para o *Beacon* 4

Colocado na posição (0,5).

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-65,06666667	-68,6		APPLE		OBS.	OBS.	-85	-70	15
14	-71,9	-66,63333333	-66	-68,33333333	-76	-75	-81,46666667	-77,96666667	-72	14
13	-67,73333333	-69	-64,9	-66	-76	-75,76666667	-72,2	-73,2	-77,76666667	13
12	-65,13333333	PCs		-73,2	-76	-71,16666667	-73,56666667	-76,6	-83,66666667	12
11	-65	-67,46666667	-73,4	-66,2	-72,56666667		-69	-77,8	OBS.	11
10	-60,6	-63	-62,16666667	-84	-70,33333333	REDES-A	-81,13333333	-80	-77,46666667	10
9	-84	-66		-71,3	-75,56666667		-73,1	-76	-78,46666667	9
8	-63,7	-69,2	COMPONETES	-68,76666667	-73		-75,33333333	-76	-70	8
7	-61,2	-74		-74,1	-70,83333333	REDES-B	-73	-76	-77,13333333	7
6	-65,53333333	-78,23333333	-75,73333333	-70	-72		-71	-76	-79,26666667	6
5	-76,7	-76,63333333	-70	-73,33333333	-72		-76,13333333	-76	-74,06666667	5
4	-65,46666667	-67,8	OBS.	-70,63333333	-77,26666667	REDES-C	-88	-79,93333333	-79,83333333	4
3	-65,6	-62		-67	-69,56666667		-68,96666667	-73,83333333	-71,13333333	3
2	-66,2	n/d	-71,63333333	-67	-73,03333333	-75,9	-74,53333333	-93	-74,03333333	2
1	-76	n/d	-69,66666667	-74,23333333	-69	-72,53333333	-75,8	-75,26666667	-70	1
0	-76	n/d	-70,66666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.12: Mapa com a média dos valores RSSI registados para o *beacon* 4, em cada posição, com direção para a frente.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-65,06666667	-68,6		APPLE		OBS.	OBS.	-85	-70	15
14	-71,9	-66,63333333	-66	-68,33333333	-76	-75	-81,46666667	-77,96666667	-72	14
13	-67,73333333	-69	-64,9	-66	-76	-75,76666667	-72,2	-73,2	-77,76666667	13
12	-65,13333333	PCs		-73,2	-76	-71,16666667	-73,56666667	-76,6	-83,66666667	12
11	-65	-67,46666667	-73,4	-66,2	-72,56666667		-69	-77,8	OBS.	11
10	-60,6	-63	-62,16666667	-84	-70,33333333	REDES-A	-81,13333333	-80	-77,46666667	10
9	-84	-66		-71,3	-75,56666667		-73,1	-76	-78,46666667	9
8	-63,7	-69,2	COMPONETES	-68,76666667	-73		-75,33333333	-76	-70	8
7	-61,2	-74		-74,1	-70,83333333	REDES-B	-73	-76	-77,13333333	7
6	-65,53333333	-78,23333333	-75,73333333	-70	-72		-71	-76	-79,26666667	6
5	-76,7	-76,63333333	-70	-73,33333333	-72		-76,13333333	-76	-74,06666667	5
4	-65,46666667	-67,8	OBS.	-70,63333333	-77,26666667	REDES-C	-88	-79,93333333	-79,83333333	4
3	-65,6	-62		-67	-69,56666667		-68,96666667	-73,83333333	-71,13333333	3
2	-66,2	n/d	-71,63333333	-67	-73,03333333	-75,9	-74,53333333	-93	-74,03333333	2
1	-76	n/d	-69,66666667	-74,23333333	-69	-72,53333333	-75,8	-75,26666667	-70	1
0	-76	n/d	-70,66666667	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.13: Mapa com a média dos valores RSSI registados para o *beacon* 4, em cada posição, com direção para a direita.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-70,93333333	-65,66666667		APPLE		OBS.	OBS.	-69	-73	15
14	-68,26666667	-68,13333333	-70,7	-68	-80,7	-71	-75,86666667	-69	-68,36666667	14
13	-67	-65,13333333	-70,3	-70	-75,53333333	-71	-79	-69	-73	13
12	-67	PCs		-73	-67,46666667	-71	-70	-69	-80,96666667	12
11	-66,9	-72,36666667	-69,53333333	-73	-67,6		-76,06666667	-70,5	OBS.	11
10	-61,66666667	-76	-66,2	-74,53333333	-69	REDES-A	-71,43333333	-76,93333333	-73,23333333	10
9	-67	-63		-70	-67,13333333		-69,4	-75,66666667	-83	9
8	-74,73333333	-79,26666667	COMPONETES	-68,03333333	-75		-70,63333333	-73	-71	8
7	-68,53333333	-72,23333333		-79,86666667	-66,2	REDES-B	-71,93333333	-71,33333333	-71	7
6	-62,13333333	-70,93333333	-71	-62,1	-64,2		-76	-76,2	-71	6
5	-63,73333333	-72	-71	-73	-64,66666667		-70,93333333	-68	-71,8	5
4	-73,83333333	-62,76666667	OBS.	-64,23333333	-69,93333333	REDES-C	-68,9	-70,6	-70,03333333	4
3	-68,53333333	-73		-68	-71,6		-70,76666667	-71,93333333	-73,13333333	3
2	-78,13333333	-77	-70,86666667	-72,66666667	-66,93333333	-73	-65,2	-65,96666667	-66,53333333	2
1	-71	-77	-79	-67	-63	-68	-68,23333333	-64,86666667	-73,26666667	1
0	-71	-76,1	n/d	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.14: Mapa com a média dos valores RSSI registados para o *beacon* 4, em cada posição, com direção para trás.

Y/X	0	1	2	3	4	5	6	7	8	X/Y
15	-69	-69,53333333		APPLE		OBS.	OBS.	-72,66666667	-74	15
14	-71	-74	-68,86666667	-70	-75,66666667	-78,9	-70	-77,66666667	-72,73333333	14
13	-71	-70,6	-68,53333333	-71,46666667	-74,53333333	-74,43333333	-70	-74	-72,73333333	13
12	-72,8	PCs		-78,66666667	-76,33333333	-72,66666667	-70	-80,06666667	-83,06666667	12
11	-62,43333333	-69	-67,83333333	-69,93333333	-69,7		-70	-73,8	OBS.	11
10	-74	-69	-78	-70,16666667	-69,9	REDES-A	-70	-76,73333333	-68	10
9	-66,73333333	-69		-71,76666667	-68,06666667		-70	-69,86666667	-73,46666667	9
8	-67	-63,16666667	COMPONETES	-75,7	-72,03333333		-72	-76,6	-75,93333333	8
7	-66,4	-63		-74,93333333	-75,7	REDES-B	-76	-69	-77,66666667	7
6	-65,1	-63	-71,33333333	-75	-67,93333333		-69,96666667	-73	-71,66666667	6
5	-68	-64,83333333	-71	-74	-72,93333333		-69,73333333	-73,26666667	-78,9	5
4	-59,2	-74,86666667	OBS.	-67,53333333	-68,13333333	REDES-C	-67,53333333	-77,4	-75,23333333	4
3	-70	-64,7		-71	-74		-72,03333333	-69,53333333	-69	3
2	-70,36666667	-69,6	-62,8	-67,36666667	-63,63333333	-71	n/d	-82	-69	2
1	-83,23333333	-72	-62	-77,8	-70	-71	n/d	-69	-67,2	1
0	-73,2	-71,2	-69	IBM-R		IBM-29		IBM-2044		0
Y/X	0	1	2	3	4	5	6	7	8	X/Y

Figura C.15: Mapa com a média dos valores RSSI registados para o *beacon* 4, em cada posição, com direção para a esquerda.