INSTITUTO POLITÉCNICO DE BRAGANÇA Escola Superior de Tecnologia e Gestão

UTFPR UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Um sistema móvel híbrido para navegação interna e localização de produtos em supermercados

Leonardo Bazán Picinin - 61440

Dissertação apresentada à Escola Superior de Tecnologia e Gestão para obtenção do grau de mestre em Informática no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná

Orientadores:

Prof. Luisa Jorge

Prof. Paulo Melo

Prof. Marlon Marcon

Bragança

Dezembro de 2025

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Um sistema móvel híbrido para navegação interna e localização de produtos em supermercados

Leonardo Bazán Picinin - 61440

Dissertação apresentada à Escola Superior de Tecnologia e Gestão para obtenção do grau de mestre em Informática no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná

Orientadores:

Prof. Luisa Jorge

Prof. Paulo Melo

Prof. Marlon Marcon

Bragança

Dezembro de 2025

Agradecimentos

Agradeço, antes de tudo, aos meus pais, Gilmar Picinin e Fânia Hurtado Bazán Picinin, pelo amor incondicional, pela criação e pelo exemplo de dedicação que sempre me inspirou. Todo o esforço e todas as conquistas que me trouxeram até aqui são reflexo do vosso apoio e da vossa presença constante. Estendo também esse agradecimento a toda a minha família, pelo incentivo, carinho e palavras de encorajamento ao longo deste percurso.

Aos meus orientadores Luisa Jorge, Paulo Melo e Marlon Marcon, deixo o meu sincero reconhecimento pela orientação, pela paciência e pela partilha de conhecimento que foram essenciais para a concretização deste trabalho. A vossa disponibilidade e o vosso rigor académico foram determinantes para o meu crescimento pessoal e profissional.

Agradeço igualmente ao Instituto Politécnico de Bragança (IPB) e à Universidade Tecnológica Federal do Paraná (UTFPR) pela oportunidade de participar no programa de dupla diplomação, que me proporcionou uma experiência académica e cultural profundamente enriquecedora.

À minha namorada, Marlene Barreira, agradeço pelo amor, pela ajuda e pelo incentivo durante esta etapa. A tua presença foi fundamental para que eu mantivesse a motivação e o equilíbrio necessários para chegar até aqui.

Aos meus colegas e amigos Lucas Bonato, Vinícius Freitas, Alex Silva e Marco Petry, agradeço pela amizade, pela ajuda, pelas ideias partilhadas e pelos momentos que tornaram este percurso mais leve e significativo. Agradeço também aos demais colegas e amigos que, mesmo não sendo mencionados individualmente, contribuíram de diferentes formas. Eles sabem quem são e o quanto a sua amizade significou nessa caminhada me tornar na pessoa que sou.

Resumo

Este trabalho apresenta o desenvolvimento de uma aplicação móvel destinada a auxiliar utilizadores na localização de produtos em supermercados, através da integração de tecnologias de Realidade Aumentada (RA), Visão Computacional (VC) e Sistemas de Posicionamento em Interiores (IPS). O sistema propõe uma abordagem inovadora para melhorar a experiência de compra, permitindo que o utilizador pesquise um produto, seja guiado visualmente até à sua localização e confirme, por meio de análise de imagem, a prateleira correta.

A arquitetura da solução foi concebida em modelo cliente–servidor, com *frontend* desenvolvido em `React Native` e *backend* em `FastAPI`, suportado por base de dados `PostgreSQL` e modelo de VC implementado em `PyTorch`. Inicialmente, foi testada a utilização de Bluetooth de Baixa Energia (BLE) para posicionamento, mas devido à instabilidade do sinal, a estratégia foi substituída por códigos *QR*, que oferecem ancoragem determinística e maior precisão espacial.

Os resultados experimentais demonstraram tempos de resposta adequados e funcionamento coerente entre os módulos, validando a viabilidade técnica da proposta. As limitações identificadas centram-se na precisão do alinhamento e na generalização do modelo de VC.

Conclui-se que o sistema constitui uma prova de conceito funcional, evidenciando o potencial da integração entre RA, VC e navegação interna em contextos de retalho, e apontando caminhos claros para a sua evolução e aplicação em ambientes reais.

Palavras-chave: realidade aumentada, visão computacional, navegação interna, aplicações móveis.

Abstract

This work presents the development of a mobile application designed to assist users in locating products within supermarkets, through the integration of Augmented Reality (AR), Computer Vision (CV), and Internal Positioning Systems (IPS) technologies. The system proposes an innovative approach to enhance the shopping experience, allowing the user to search for a product, to be visually guided to its location, and to confirm the correct shelf through image analysis.

The solution's architecture was conceived with a client–server model, with a frontend developed in `React Native` and a backend implemented in `FastAPI`, supported by a `PostgreSQL` database and a CV model built in `PyTorch`. Initially, the use of Bluetooth Low Energy (BLE) beacons for positioning was tested, but due to signal instability, the strategy was replaced by QR codes, which provide deterministic anchoring and higher spatial precision.

Experimental results demonstrated adequate response times and coherent operation between modules, validating the technical feasibility of the proposed system. The identified limitations are mainly related to alignment accuracy and the generalization capacity of the CV model.

It is concluded that the system constitutes a functional proof of concept, demonstrating the potential of integrating AR, CV, and indoor navigation technologies in retail contexts, while outlining clear directions for its evolution and application in real environments.

Keywords: augmented reality, computer vision, indoor navigation, mobile applications.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Códigos

Acrónimos e Siglas

API Interface de programação de aplicações.

BLE Bluetooth de Baixa Energia.

CNN Rede Neuronal Convolucional.

CRUD Criação, Leitura, Atualização e Deleção.

GPS Sistema de Posicionamento Global.

IPB Instituto Politécnico de Bragança.

IPS Sistemas de Posicionamento em Interiores.

PLAN Posicionamento, Localização e Navegação.

RA Realidade Aumentada.

RFID Identificação por Radiofrequência.

RSSI Indicador de Intensidade do Sinal Recebido.

VC Visão Computacional.

Capítulo 1.

Introdução

Em ambientes comerciais como supermercados, onde a eficiência na localização de produtos impacta diretamente na experiência do consumidor, métodos tradicionais como sinalização estática ou mapas digitais demonstram limitações significativas. Estudos recentes indicam que localizar produtos em supermercados representa frequentemente a fase mais demorada do processo de compras, correspondendo a uma parte significativa do tempo total passado nas lojas [43, 29]. Além disso, pesquisas indicam que melhorias na sinalização podem reduzir em até 15% a necessidade de interação com colaboradores e aumentar as vendas em até 20%, reforçando a relevância de soluções tecnológicas para melhorar a experiência do utilizador nestes contextos [38].

Os avanços recentes em Realidade Aumentada (RA), Visão Computacional (VC) e Sistemas de Posicionamento em Interiores (IPS) possibilitam novas formas de interação espacial em ambientes fechados [22, 28]. Estas tecnologias permitem a criação de interfaces visuais intuitivas e oferecem precisão em ambientes fechados, algo difícil de alcançar por meio de abordagens tradicionais, como Sistema de Posicionamento Global (GPS) ou Identificação por Radiofrequência (RFID). Embora já estejam a ser exploradas com sucesso em setores como a educação, a agricultura e a segurança [13, 36, 42], a sua aplicação no contexto específico dos supermercados ainda é incipiente, mas altamente promissora. Este cenário cria uma oportunidade de explorar como a utilização combinada destas tecnologias pode melhorar a navegação em loja, reduzir o tempo de procura e otimizar a experiência de compra.

Este trabalho propõe um sistema móvel integrado para navegação interior e localização de produtos em supermercados. A solução combina RA para orientação visual em tempo real, VC para reconhecimento ao nível das prateleiras e Bluetooth de Baixa Energia (BLE) para a inicialização da posição do utilizador dentro de um sistema global de coordenadas. O sistema

integra um *frontend* em `React Native` e um *backend* desenvolvido com `FastAPI`, `PostgreSQL` e `PyTorch`, garantindo modularidade e uma comunicação fluida entre os dados de produto e os serviços de inferência visual.

Parte dos resultados apresentados nesta dissertação originou o artigo *A Hybrid Mobile System for Indoor Navigation and Product Localization in Supermarkets*, atualmente aceite para publicação na conferência internacional *International Conference on Internet of Things: Systems, Management and Security* (IOTSMS 2025). O manuscrito foi avaliado por revisores que destacaram a relevância prática do sistema desenvolvido, a clareza da arquitetura proposta e a consistência dos resultados experimentais, reconhecendo a solidez técnica da abordagem e a contribuição do estudo para o avanço de soluções híbridas de navegação interior.

1.1. Objetivos

Este trabalho tem como objetivo desenvolver um sistema integrado de navegação interna para supermercados, utilizando as tecnologias mencionadas, com o intuito de auxiliar os utilizadores na localização eficiente de produtos. Para tal, será desenvolvida uma aplicação móvel, com foco na usabilidade, com recursos de RA para indicar visualmente os produtos; será implementado um modelo de VC capaz de reconhecer prateleiras; e será integrada uma tecnologia de posicionamento interno baseada em BLE, com o objetivo de garantir precisão na localização do utilizador dentro do supermercado. Esta solução procura superar as limitações das abordagens tradicionais, proporcionando uma experiência de compra mais ágil e autónoma.

1.2. Âmbito e Limitações

A implementação do sistema será realizada com a utilização de recursos que favorecem a eficiência no desenvolvimento. Será empregada a tecnologia `React Native` [45], em conjunto com bibliotecas específicas que suportam a navegação interna e demais funcionalidades essenciais. O desenvolvimento e a avaliação ocorrerão em ambiente laboratorial, permitindo o treinamento dos modelos de VC, a gestão dos dados armazenados e a execução de testes sistemáticos voltados à verificação da eficácia do sistema.

1.3. Estrutura do Documento

Este documento encontra-se organizado em sete capítulos, além dos apêndices. No Capítulo **2**, é apresentado o enquadramento teórico e tecnológico que sustenta o trabalho, abordando os conceitos e ferramentas relacionados com aplicações móveis, RA, VC e navegação interna. O Capítulo **3** descreve o planeamento do sistema, incluindo as capacidades centrais, a arquitetura proposta e o design visual. Os Capítulos **4** e **5** detalham, respetivamente, o desenvolvimento do *backend* e do *frontend*, destacando a integração entre os módulos e as soluções técnicas adotadas. O Capítulo **6** reúne os resultados e as discussões, apresentando os testes realizados, a análise de desempenho e a avaliação da estratégia de navegação. Por fim, o Capítulo **7** apresenta as conclusões e as propostas de trabalhos futuros. Os apêndices complementam o conteúdo com informações adicionais sobre testes, requisitos, casos de uso e excertos de código-fonte.

Capítulo 2.

Contexto e Tecnologias

Este capítulo apresenta os fundamentos teóricos e tecnológicos que sustentam o desenvolvimento do sistema proposto, abordando as principais áreas envolvidas na sua conceção. São discutidas as abordagens de desenvolvimento móvel, com destaque para a adoção do *framework* `React Native`, as bases da RA e a sua integração através da biblioteca `ViroReact`, bem como os princípios da visão computacional aplicados à deteção de prateleiras por Rede Neuronal Convolucional (CNN). Por fim, é explorado o conceito de navegação interna, com ênfase na utilização de beacons BLE e na técnica de trilateração como suporte à localização do utilizador em ambientes fechados. Em conjunto, estas secções estabelecem o enquadramento técnico necessário para compreender as decisões de arquitetura e implementação descritas nos capítulos seguintes. No final do capítulo o contexto da aplicação desenvolvida é fortalecido pela apresentação de exemplos de trabalhos relacionados usando tecnologias similares.

2.1. Aplicações Móveis

Com o crescimento exponencial da utilização de dispositivos móveis, a escolha da abordagem de desenvolvimento adequada é fundamental para garantir a eficiência e a qualidade de uma aplicação. Existem duas principais abordagens para o desenvolvimento de aplicações móveis: o desenvolvimento nativo e o desenvolvimento multiplataforma. Cada uma destas abordagens possui as suas características, vantagens e limitações, que devem ser consideradas de acordo com os objetivos do projeto.

2.1.1. Opções de desenvolvimento

Segundo [41], no desenvolvimento nativo, as aplicações são desenvolvidas com as ferramentas e linguagens específicas de cada sistema operativo. Esta abordagem oferece acesso total aos recursos do dispositivo e, geralmente, melhor desempenho gráfico e computacional. No entanto, implica a necessidade de manter dois projetos distintos (Android e iOS), o que aumenta significativamente o custo de desenvolvimento e de manutenção.

Como alternativa, o desenvolvimento multiplataforma permite a criação de aplicações com uma única base de código, compatível com diferentes sistemas operativos [12]. Ferramentas como `React Native` e `Flutter` têm-se destacado neste cenário por oferecerem um desempenho próximo do nativo, além de facilitarem o desenvolvimento com reutilização de componentes e integração com bibliotecas de terceiros [49].

2.1.2. `React Native`

O `React Native` é um *framework* de desenvolvimento que permite criar aplicações a partir de uma única base de código, utilizando `JavaScript` e a biblioteca `React` [45]. Combina as vantagens do desenvolvimento nativo com a produtividade do desenvolvimento baseado em componentes, proporcionando a reutilização de código e uma experiência de desenvolvimento eficiente, resultando em aplicações de elevado desempenho e interfaces de utilizador nativas e fluidas. Além disso, o `React Native` possui integração direta com o `Expo`, uma ferramenta que acelera o ciclo de desenvolvimento e oferece suporte a bibliotecas direcionadas para funcionalidades avançadas, como RA, geolocalização e sensores [35, 47].

O estudo comparativo [32] avaliou o desempenho e a usabilidade de aplicações desenvolvidas com `React Native` em comparação com aplicações nativas. No teste de perceção, 28% dos utilizadores não conseguiram distinguir entre as aplicações desenvolvidas com as duas abordagens. Já nos testes de desempenho, embora o desenvolvimento nativo tenha apresentado vantagem em alguns aspetos, o `React Native` obteve resultados semelhantes em termos de consumo de bateria e utilização de memória, apresentando inclusive melhor eficiência energética em determinadas tarefas. No contexto deste projeto, o `React Native` foi selecionado pela sua maturidade, pelo suporte ativo da comunidade e pela compatibilidade com bibliotecas de RA. Em conjunto com o `Expo`, facilitou a execução de testes rápidos e o desenvolvimento iterativo, assegurando uma interação fluida entre diferentes dispositivos.

2.2. Realidade Aumentada

Segundo [1], a RA pode ser definida como uma variação dos ambientes virtuais que permite ao utilizador visualizar o mundo real com a sobreposição de objetos virtuais, sem substituir completamente a realidade, como ocorre na realidade virtual. Entre os principais componentes utilizados para este processo encontram-se câmaras, sensores de movimento, GPS, e algoritmos de rastreamento que permitem mapear o ambiente enquanto localizam o dispositivo simultaneamente [11]. A Figura **2.1** ilustra a utilização da RA em aplicações de diferentes contextos.

Figura 2.1: Exemplos de aplicações de RA em jogos e arquitetura doméstica. [21, 54]

2.2.1. `ViroReact`

O `ViroReact` é uma biblioteca de código aberto que permite o desenvolvimento de experiências de RA em dispositivos móveis através do `React Native` [48]. O seu funcionamento baseia-se no posicionamento de objetos virtuais num sistema de coordenadas tridimensionais, permitindo a criação de cenas interativas que combinam elementos físicos e digitais em tempo real. A biblioteca disponibiliza componentes prontos para deteção de planos, inserção de modelos 3D, controlo de câmara e interação com a cena, simplificando a construção de aplicações imersivas sem necessidade de recorrer à renderização gráfica de baixo nível.

A adoção do `ViroReact` neste trabalho deve-se à sua integração com o ecossistema `React Native`, ao suporte multiplataforma e à facilidade de implementação de funcionalidades essenciais para o sistema proposto. Para verificar a sua viabilidade e desempenho, foram realizados testes preliminares em ambiente controlado, cujos resultados encontram-se documentados no Apêndice **A**.

2.3. Visão Computacional

A VC é um campo da inteligência artificial que procura capacitar as máquinas a extrair, interpretar e agir sobre informações visuais a partir de imagens ou vídeos. Segundo [30], trata-se da área responsável por permitir que sistemas computacionais simulem a capacidade humana de compreender o ambiente visual. Esta tecnologia é amplamente utilizada em aplicações como sistemas de alarme, rastreamento de objetos e processamento de transformações em imagens [14]. No contexto deste trabalho, a VC será utilizada como ferramenta para detetar automaticamente os níveis de prateleiras num supermercado, auxiliando na localização precisa de produtos.

2.3.1. Linhas semânticas

Linhas semânticas são linhas significativas que definem o *layout* de uma imagem, desempenhando um papel essencial na análise de imagens e na compreensão de cenas. Elas separam diferentes regiões semânticas dentro de uma cena e, muitas vezes, não são linhas explícitas, mas sim implícitas, sugeridas pelos limites entre essas regiões [18]. A Figura **2.2** ilustra as linhas semânticas detetadas em imagens de paisagens.

Figura 2.2: Linhas semânticas detetadas em imagens, destacando limites entre regiões da cena. [18]

2.3.2. Redes neuronais e redes neuronais convolucionais

Segundo [3], uma rede neuronal é um paradigma de processamento de informação inspirado na forma como os sistemas nervosos biológicos, como o cérebro, processam informações. Estas são compostas por um grande número de unidades de processamento interligadas de forma ponderada, simulando neurónios, que trabalham em conjunto para resolver problemas específicos. A Figura **2.3** ilustra uma rede neuronal com duas camadas ocultas, facilitando a visualização das unidades de processamento.

Uma CNN é um tipo de rede neuronal profunda concebida especificamente para lidar com dados visuais, compostas por múltiplas camadas que extraem características hierárquicas das imagens através de filtros convolucionais [15, 58]. Esta arquitetura permite que o modelo aprenda automaticamente padrões espaciais e texturais relevantes, sem necessidade de extração manual

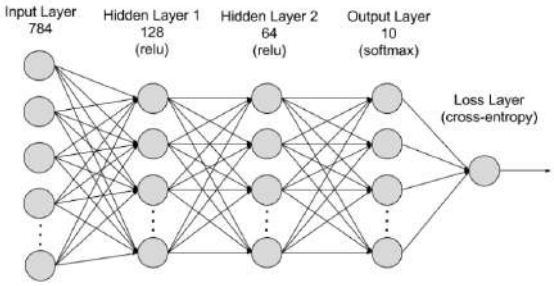

Figura 2.3: Estrutura esquemática de uma rede neuronal com múltiplas camadas ocultas. [59]

de características [19]. A Figura **2.4** apresenta a camada convolucional de uma CNN, na qual são realizadas as etapas de convolução e subamostragem, correspondentes ao pré-processamento da imagem antes da sua introdução na rede neuronal.

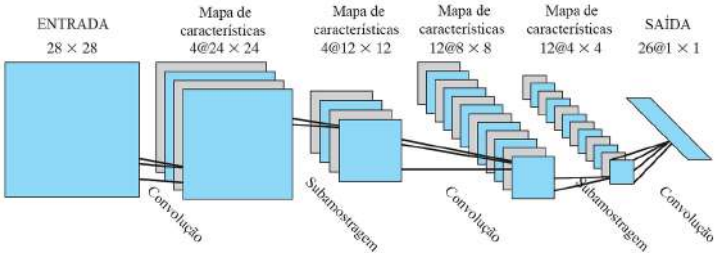

Figura 2.4: Representação de uma camada convolucional de uma CNN, mostrando as etapas de convolução e subamostragem. [58]

Neste projeto, uma CNN será utilizada para detetar linhas semânticas horizontais em imagens de prateleiras, permitindo identificar automaticamente seus diferentes níveis e, assim, localizar com precisão o posicionamento dos produtos.

2.4. Navegação Interna

Segundo [26], a navegação interna é definida como um sistema de Posicionamento, Localização e Navegação (PLAN) concebido para funcionar em ambientes fechados, onde as limitações dos sistemas convencionais, como o GPS, impedem a sua funcionalidade. Esta tecnologia é frequentemente utilizada para fornecer orientação em espaços complexos e multifuncionais, como aeroportos, universidades e centros de conferências, cuja arquitetura pode dificultar a deslocação e a identificação de pontos de interesse [9]. Entre as principais soluções disponíveis, destacam-se etiquetas RFID, sensores inerciais, Wi-Fi e *beacons* BLE, sendo esta última a tecnologia adotada neste trabalho.

2.4.1. *Beacons* BLE

Os *beacons* BLE são pequenos transmissores de rádio de baixo consumo energético que enviam sinais periódicos para dispositivos móveis compatíveis [23]. Ao contrário do *Bluetooth* tradicional, a sua eficiência energética permite que dispositivos como sensores médicos e relógios inteligentes operem com uma única bateria durante meses ou até anos [57]. Além disso, o seu custo de instalação reduzido e a ampla compatibilidade com *smartphones* tornam-no uma solução escalável e acessível para sistemas de localização interna, eliminando a necessidade de equipamentos adicionais.

2.4.2. Trilateração

A trilateração é uma técnica de localização utilizada para determinar a posição de um ponto desconhecido a partir das distâncias medidas em relação a três ou mais pontos fixos com coordenadas previamente conhecidas [6]. O princípio da trilateração consiste em encontrar a posição de um objeto no espaço com base na interseção de múltiplos volumes definidos por essas distâncias, círculos no caso bidimensional ou esferas quando o cálculo é realizado em três dimensões.

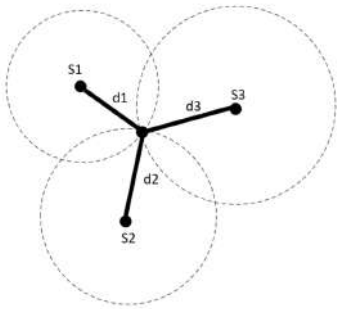

Figura 2.5: Trilateração entre três pontos (s1,s2,s3) indicando suas distâncias (d1,d2,d3).

Na Figura **2.5**, este conceito é ilustrado com três estações de referência representadas pelos pontos s_1, s_2 e s_3, e o ponto central correspondente à posição desconhecida que se pretende calcular. Cada distância entre o ponto central e os *beacons* (representadas como d_1, d_2 e d_3) define a superfície de uma esfera centrada em cada estação. A posição do objeto é obtida a partir da interseção dessas superfícies. Matematicamente, o modelo de trilateração em três dimensões pode ser descrito pelo seguinte sistema de equações:

$$(x - x_{si})^2 + (y - y_{si})^2 + (z - z_{si})^2 = r_{si}^2 \quad \text{para } i = 1, 2, 3 \qquad \textbf{Eq. 2.1}$$

Em que (x_{si}, y_{si}, z_{si}) são as coordenadas conhecidas de cada *beacon* BLE, e r_{si} representa

a distância estimada entre o dispositivo e cada um desses pontos, obtida a partir do Indicador de Intensidade do Sinal Recebido (RSSI). Esta equação será adotada como base para estimar, em tempo real, a posição do utilizador dentro do supermercado. Com os *beacons* instalados em posições estratégicas no ambiente físico, o sistema poderá calcular a localização relativa do dispositivo móvel e, consequentemente, a distância ao produto desejado. Estas informações serão integradas na interface, permitindo orientar o utilizador de forma visual precisa e intuitiva durante a navegação no espaço físico.

2.5. Comparação com Trabalhos Relacionados

Estudos recentes têm explorado a utilização de RA e de IPS com o objetivo de melhorar a navegação e o apoio à tomada de decisão em ambientes de venda a retalho. Trabalhos como o RetailOpt [50] e o ARShopping [31] integram dados de sensores, mapas e técnicas de fusão de dados para rastreamento em loja e pesquisa de produtos, enquanto estudos [16, 37, 39] salientam o potencial da RA para reforçar a orientação espacial e o fornecimento de informação contextual em tempo real. No contexto do retalho, a visualização imersiva e a interação com avatares têm demonstrado capacidade para enriquecer a experiência do cliente [44], e os mapas interiores baseados em RA contribuem para uma geovisualização mais intuitiva em *layouts* complexos [24]. O BLE continua a ser uma opção de baixo custo para localização, embora apresente elevada instabilidade sob interferência [39, 25], ao passo que a inicialização baseada em códigos *QR* oferece uma alternativa determinística e de baixo custo infraestrutural [16].

No domínio da compreensão visual de cenas, métodos de deteção de linhas semânticas e de aprendizagem profunda têm obtido bons resultados em ambientes controlados, mas enfrentam desafios de generalização sob diferentes perspetivas [34, 52]. *Frameworks* multiplataforma, como o `React Native`, permitem o desenvolvimento ágil de aplicações de RA [37]. Com base nestes avanços, o presente trabalho integra RA (via `ViroReact`), sensores BLE e VC numa aplicação móvel unificada para navegação em supermercados, mitigando a instabilidade do BLE através de alinhamento baseado em códigos *QR* e permitindo a confirmação visual ao nível da prateleira dentro de uma referência espacial coerente.

Capítulo 3.

Planeamento do Sistema

Este capítulo apresenta o planeamento do sistema, reunindo as definições essenciais que orientam a sua implementação. São descritas as capacidades centrais previstas, a arquitetura geral da solução e o fluxo de operação, bem como os principais aspetos de design visual. Estes elementos articulam-se para estabelecer uma visão integrada do funcionamento do sistema, assegurando a coerência entre os requisitos definidos e a experiência final do utilizador.

3.1. Capacidades Previstas para o Sistema

Segundo [4], os requisitos funcionais representam os comportamentos esperados do sistema, expressando-se como serviços, tarefas ou funcionalidades que este deve oferecer. Já os requisitos não funcionais, conforme [5], correspondem a restrições ou atributos de qualidade que influenciam o desempenho, a usabilidade, a segurança e a manutenção da aplicação. O trabalho de [2], por sua vez, define um caso de uso como uma coleção de possíveis sequências de interações entre o sistema em questão e os seus atores externos, relacionadas com um objetivo específico.

Com o objetivo de descrever de forma clara e estruturada as principais funcionalidades previstas para o sistema, os requisitos funcionais, não funcionais e os casos de uso foram agrupados em três grandes capacidades centrais. Cada uma destas capacidades representa uma etapa essencial da experiência do utilizador com a aplicação, reunindo os requisitos e interações que lhe dão suporte. A descrição completa dos requisitos funcionais (RFXX), dos não funcionais (RNFXX) e dos casos de uso (CUXX) podem ser consultadas no Apêndice **B**.

3.1.1. Pesquisa e seleção de produtos

Esta capacidade contempla as funcionalidades que permitem ao utilizador localizar, dentro da aplicação, o produto que pretende encontrar no supermercado. O sistema apresentará uma lista de produtos registados, acompanhada de um campo de pesquisa para facilitar a filtragem por nome (RF08). O utilizador poderá iniciar uma pesquisa a partir da leitura de um código *QR* localizado na entrada do supermercado, que ativa a aplicação e define a posição inicial para a navegação (RF01, CU01). O sistema exibirá informações relevantes de cada produto e inicializará a navegação ao selecionar o item pretendido (CU02). Estas funcionalidades garantem uma etapa inicial rápida e intuitiva, servindo de base para as restantes interações de localização no ambiente físico.

3.1.2. Navegação até o produto em tempo real

Uma vez selecionado o produto, o sistema fornecerá instruções visuais e métricas de distância para guiar o utilizador até à sua localização física no supermercado. Esta navegação será realizada através de marcadores atualizados continuamente de acordo com o deslocamento do utilizador (RF03, RF04). Durante o trajeto, o sistema enviará notificações por vibração ou mensagens visuais para indicar a proximidade do destino (RF05, RF06). Esta etapa tem como objetivo garantir que a pesquisa seja eficiente e fluida, evitando a necessidade de apoio por parte dos colaboradores. As interações previstas incluem iniciar a navegação a partir do ecrã de listagem de produtos, seguir as orientações visuais e receber alertas até estar a aproximadamente dois metros do item procurado (CU02, CU03). O desempenho desta navegação está diretamente relacionado com os requisitos não funcionais que asseguram um tempo de resposta rápido e atualização em tempo real (RNF04, RNF05).

3.1.3. Confirmação visual da prateleira correta

Ao aproximar-se do destino, o sistema permitirá confirmar visualmente a posição do produto na prateleira, utilizando técnicas de VC para destacar o nível correto onde este se encontra (RF07). Esta etapa será acionada após o utilizador apontar a câmara para a área indicada, capturando a imagem das prateleiras e identificando automaticamente aquela onde o produto está localizado (CU05). A capacidade de processamento de imagem (RNF06) garante que a deteção seja concluída em poucos segundos, fornecendo uma confirmação visual clara para finalizar a

pesquisa. Esta confirmação visual procura evitar erros na escolha do produto e melhorar a experiência de compra, especialmente em prateleiras com elevada densidade de itens semelhantes.

3.2. Arquitetura Geral da Solução

A solução proposta adota uma arquitetura cliente–serviços, estruturada em quatro blocos principais: (i) aplicação móvel (camada de apresentação e interação), (ii) serviços de apoio à navegação (cálculo de posição e visualização de indicadores), (iii) serviços de *backend* (pesquisa de produtos e inferência de prateleiras) e (iv) camada de dados (armazenamento do catálogo e metadados de localização). A comunicação entre os blocos ocorre através de requisições *HTTP* para a Interface de programação de aplicações (API) do *backend*, enquanto a estimativa de posição é processada localmente no dispositivo, a partir da leitura do RSSI dos *beacons*, da comunicação com o `ViroReact` e da integração com a câmara do dispositivo. A Figura **3.1** oferece uma visão geral da arquitetura da aplicação.

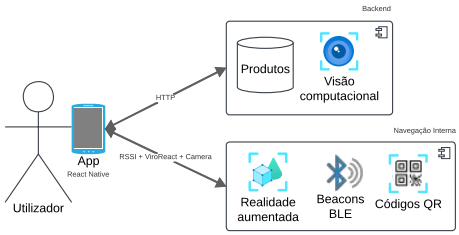

Figura 3.1: Arquitetura geral do sistema, integrando a aplicação móvel, *backend*, módulos de RA e mecanismos de navegação interna.

3.2.1. Aplicação e serviços de *backend*

A aplicação móvel está estruturada em três módulos principais: (i) módulo de pesquisa, responsável por consultar o *backend* e apresentar ao utilizador uma lista de produtos disponíveis; (ii) módulo de navegação, responsável por recolher sinais RSSI de *beacons* BLE, estimar distâncias e aplicar trilateração para atualizar, em tempo real, a direção e a distância até ao destino, incorporando recursos de RA para exibir marcadores virtuais que guiam o deslocamento; e (iii) módulo de confirmação, encarregado de capturar imagens das prateleiras e enviá-las ao *backend* para análise. A Figura **3.2** ilustra os principais componentes utilizados pela aplicação.

Os serviços de *backend* são disponibilizados através de APIs e organizados em dois grupos: (i) API de pesquisa, responsável por consultar a base de dados e devolver informações e metadados de localização dos produtos; e (ii) API de deteção de prateleiras, que processa as imagens

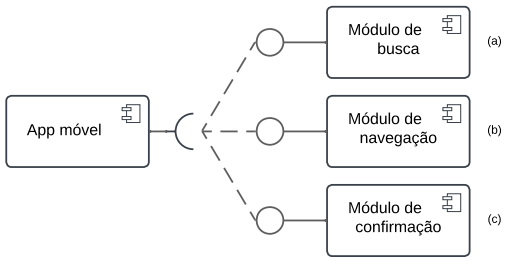

Figura 3.2: Componentes principais da aplicação móvel, incluindo módulos de pesquisa, navegação e confirmação visual.

capturadas pela aplicação, aplicando técnicas de VC para identificar e devolver o nível correto da prateleira onde o produto se encontra. A Figura **3.3** ilustra a estrutura do *backend* através de um diagrama de componentes.

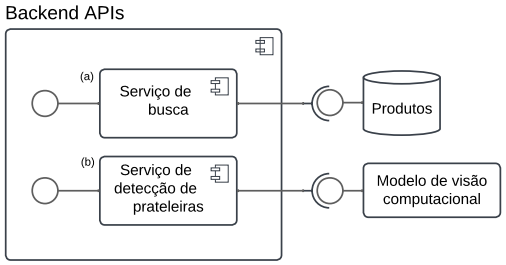

Figura 3.3: Estrutura dos serviços de *backend*, destacando as APIs de pesquisa e deteção de prateleiras e o modelo de VC.

3.2.2. Estrutura e organização dos dados

A camada de dados armazena a informação de cada produto, incluindo identificador, nome, imagem, preço, coordenadas no mapa do ambiente e nível da prateleira, conforme ilustrado na Figura **3.4** . Estes dados alimentam tanto a navegação (coordenadas) como a validação final (nível da prateleira). O registo será realizado previamente utilizando coordenadas de teste e ajustado de acordo com o mapeamento do ambiente, através de uma aplicação auxiliar desenvolvida para esse fim.

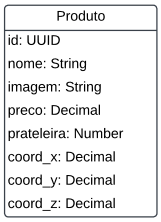

Figura 3.4: Modelo de dados da entidade Produto, com atributos de identificação, preço, imagem e localização no espaço.

3.3. Funcionamento e *Design* Visual

O funcionamento do sistema segue um fluxo que integra as três capacidades centrais descritas na Secção **3.1**, de forma sequencial e contínua. Este fluxo reflete a interação direta entre o utilizador e os módulos da aplicação, abrangendo desde a ativação inicial por código *QR* até à finalização da pesquisa com a validação por VC. Todos os passos estão alinhados com os requisitos funcionais e não funcionais definidos no Apêndice **B**, bem como com os casos de uso CU01 a CU05, assegurando coerência entre o planeamento e a execução. A Figura **3.5** sintetiza esta sequência operacional através de um fluxograma.

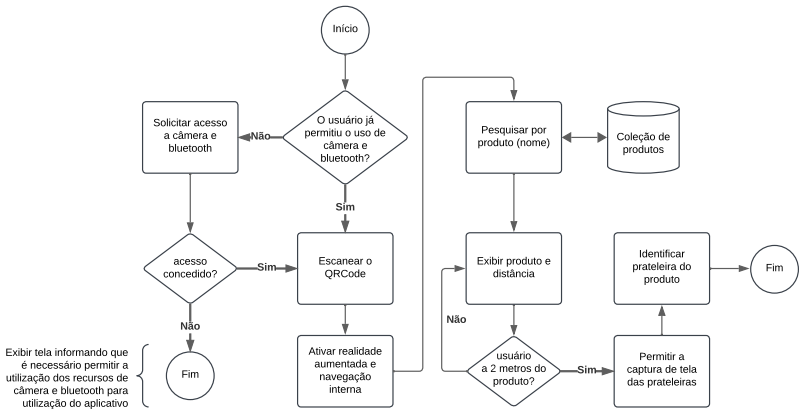

Figura 3.5: Fluxograma do processo de utilização da aplicação, desde a leitura do código *QR* até à validação visual do produto.

Com o objetivo de garantir uma interface funcional e coerente, foi realizada uma fase de prototipagem antes do desenvolvimento. Este processo permitiu validar a disposição dos elementos gráficos e o fluxo de interação, reduzindo retrabalho e garantindo uma experiência consistente. O protótipo contempla três ecrãs principais: *QR Code Screen*, *Home Screen* e *Find Product Screen*, sendo o último dividido em quatro etapas que conduzem o utilizador desde a localização do produto até à confirmação visual na prateleira, conforme ilustrado na Figura **3.6**.

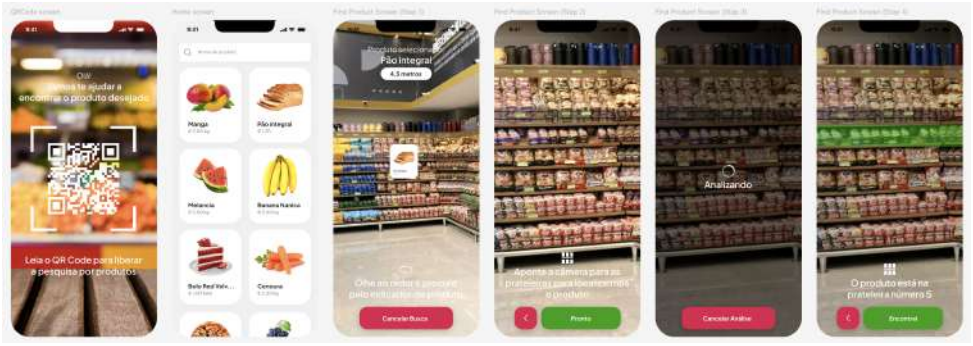

Figura 3.6: Protótipo de alta fidelidade ilustrando as principais interfaces da aplicação móvel.

O design da aplicação segue os princípios de *Flat Design* [20] e de *Design Minimalista* [7], priorizando a clareza visual, a facilidade de utilização e a consistência estética. Elementos como textos, ícones e botões são combinados com cores estratégicas para direcionar a atenção do utilizador e facilitar a execução das tarefas previstas.

Capítulo 4.

Desenvolvimento do `Backend`

Este capítulo descreve o processo de desenvolvimento do *backend* do sistema, abrangendo desde a modelação da base de dados até à disponibilização dos serviços através de uma API unificada. São apresentadas a estrutura da entidade `Produto`, a implementação das operações `CRUD` em `FastAPI` e a integração do modelo de VC baseado em `PyTorch` para deteção de prateleiras. Por fim, é detalhada a arquitetura final do *backend*, consolidando os componentes desenvolvidos e a comunicação entre os serviços.

4.1. Modelação da Base de Dados

O armazenamento da informação do sistema foi implementado utilizando o `PostgreSQL` [56], um sistema de gestão de bases de dados relacionais amplamente consolidado, escolhido pela sua robustez, pelo suporte nativo a tipos de dados como `UUID` e `NUMERIC`, e por se tratar de uma solução *open source* e de utilização gratuita. Para simplificar a configuração e garantir a portabilidade, a base de dados foi executada num contentor `Docker` [53], utilizando volumes para assegurar a persistência dos dados mesmo após reinicializações do serviço. Neste contexto, o *backend* foi estruturado de forma a oferecer suporte às principais funcionalidades da aplicação, garantindo não apenas a persistência, mas também a facilidade de consulta.

A entidade central definida na base de dados é o Produto, que concentra a informação necessária para possibilitar tanto a pesquisa no catálogo como a integração com os mecanismos de navegação e validação visual. Desta forma, a modelação desta entidade foi concebida a partir dos requisitos definidos no planeamento do sistema (Secção **3.2.2**), resultando na estrutura apresentada na Tabela **4.1**.

Tabela 4.1: Atributos da entidade Produto

Atributo	Tipo de dado	Descrição
id	UUID	Identificador único do produto
nome	VARCHAR(255)	Nome do produto
imagem	TEXT	URL ou referência à imagem do produto
preco	NUMERIC(10,2)	Preço monetário com duas casas decimais
prateleira	INTEGER	Nível da prateleira onde o produto se encontra
coord_x	DECIMAL(10,2)	Coordenada X no espaço do supermercado
coord_y	DECIMAL(10,2)	Coordenada Y no espaço do supermercado
coord_z	DECIMAL(10,2)	Coordenada Z no espaço do supermercado

Cada produto é identificado por um `UUID` gerado automaticamente, assegurando unicidade e evitando colisões mesmo em cenários distribuídos. Os atributos *nome* e *imagem* permitem identificar e exibir o produto ao utilizador, enquanto o atributo *preço* é armazenado com precisão de duas casas decimais, garantindo consistência nos valores monetários. Além disso, os atributos *prateleira*, *coord_x*, *coord_y* e *coord_z* representam informação espacial utilizada tanto na navegação como na etapa de validação visual. Estes campos permitem mapear a posição de cada item dentro do ambiente físico e identificar em que nível da prateleira este se encontra.

Durante os testes iniciais, foram inseridos registos de exemplos na base de dados com o objetivo de validar a comunicação entre o `PostgreSQL` e a API desenvolvida em `FastAPI`. Esta fase permitiu verificar o correto funcionamento das operações de inserção, consulta, atualização e remoção de produtos, além de confirmar a consistência dos tipos de dados adotados, como a utilização de identificadores `UUID`, valores monetários e atributos referentes à localização do produto. Deste modo, a modelação estabelecida serviu como base conceptual e como parte da integração entre a base de dados e os restantes módulos do *backend*, garantindo suporte às funcionalidades essenciais do sistema.

4.2. Implementação da API de Produtos

A linguagem de desenvolvimento adotada para a implementação do *backend* foi o `Python`, escolha motivada pela necessidade de integrar diretamente o modelo de VC construído no âmbito deste trabalho e cuja implementação será detalhada na Secção **4.3**. Optar por uma única linguagem para todo o *backend* permitiu evitar a fragmentação da solução em múltiplos serviços,

reduzindo a complexidade arquitetónica e eliminando a necessidade de mecanismos adicionais de integração, como *gateways* entre APIs distintas.

Para disponibilizar as funcionalidades relacionadas com o catálogo de produtos, foi desenvolvida uma API `REST` utilizando o *framework* `FastAPI`. A escolha desta tecnologia deve-se à sua simplicidade de configuração, ao suporte nativo para validação de dados com `Pydantic` e à documentação automática das rotas. Estas características agilizam o processo de desenvolvimento e tornam a API facilmente integrável à aplicação móvel. Esta foi organizada em torno de um conjunto de *endpoints* responsáveis pela gestão da entidade Produto, implementando as operações clássicas de Criação, Leitura, Atualização e Deleção (CRUD). Cada *endpoint* comunica-se diretamente com a base de dados `PostgreSQL` através da biblioteca `SQLAlchemy`, que atua como camada de mapeamento objeto-relacional, permitindo manipular registos da base de dados como objetos `Python`. A Tabela **4.2** ilustra os métodos e as rotas de cada operação CRUD.

Tabela 4.2: Endpoints disponibilizados pela API de produtos

Método	**Rota**	**Descrição**
GET	/products	Lista todos os produtos ou busca por nome
GET	/products/{id}	Retorna as informações de um produto específico
POST	/products	Cria um novo produto na base de dados
PUT	/products/{id}	Atualiza os dados de um produto existente
DELETE	/products/{id}	Remove um produto da base de dados

Além do registo e da listagem, a rota */products* também suporta um parâmetro de pesquisa textual denominado *search*, acessível através do método `GET`, que permite filtrar os resultados pelo nome do produto. Esta funcionalidade viabiliza a integração direta com a aplicação, onde o utilizador pode introduzir o nome do item pretendido e receber como retorno apenas os registos correspondentes, conforme ilustrado no protótipo da Secção **3.3**. Deste modo, a API de produtos garante a interface necessária entre a aplicação móvel e a base de dados, centralizando o acesso ao catálogo e possibilitando a manipulação consistente da informação.

Entre as funcionalidades disponibilizadas pela API de produtos, a aplicação utilizou apenas as operações de leitura (`GET`), para efetuar a pesquisa de produtos, e de atualização (`PUT`), para modificar os dados de posição (coordenadas e nível de prateleira) através de uma subaplicação de calibração desenvolvida para apoiar os testes, cuja descrição detalhada é apresentada na Secção **6.2.1**.

4.3. Desenvolvimento da Visão Computacional

O desenvolvimento do modelo de VC utilizado neste trabalho não partiu da construção de´ uma solução inédita, mas sim da adaptação de abordagens consolidadas na literatura. Entre os estudos que serviram de referência, destacam-se:

1. O trabalho *Deep Hough Transform for Semantic Line Detection* [27], proposto para a deteção de linhas semânticas em imagens, combina técnicas clássicas de transformação geométrica com aprendizagem profunda;

2. O estudo *Shelf Management System Based on Deep Learning* [46], voltado para a gestão visual de prateleiras, aplica metodologias semelhantes para identificar automaticamente os níveis das prateleiras em ambientes de retalho.

Estas referências foram selecionadas por apresentarem metodologias previamente validadas em contextos semelhantes ao deste projeto e por se alinharem ao objetivo de detetar, de forma fiável, as linhas horizontais que delimitam os níveis das prateleiras. Assim, a implementação desenvolvida neste trabalho baseou-se na adaptação dessas abordagens, recorrendo ao modelo disponibilizado pelo primeiro estudo da listagem anterior, treinado com o *dataset* fornecido pelo segundo estudo. O modelo de deteção de linhas semânticas foi integrado como elemento central do processo de validação visual adotado pelo sistema.

4.3.1. Preparação dos dados e treino do modelo

Como as anotações do conjunto de dados original estavam normalizadas em percentagens, foi necessário adaptá-las ao formato exigido pelo modelo *Deep Hough Transform*, que requer coordenadas absolutas em píxeis. Foi criado um *script* auxiliar para converter as coordenadas normalizadas em valores absolutos com base nas dimensões da imagem. Cada rótulo gerado segue a estrutura `image.jpg,1 x1 y1 x2 y2`, em que o primeiro número indica o número de linhas existentes na imagem e cada grupo subsequente de quatro valores define os pontos extremos de uma linha detetada. Quando várias linhas estão presentes, as respetivas coordenadas são concatenadas sequencialmente. O Algoritmo **4.1** apresenta um pseudocódigo que ilustra o processo de conversão.

```
for line in dataset:
    image_name, y_values = parse(line)
    H, W = getImageSize(image_name)
    label = f"{image_name},{len(y_values)}"
    for y in y_values:
        y_abs = int(y * H)
        label += f" 0 {y_abs} {W} {y_abs}"
    save(label)
```

Código 4.1: Pseudocódigo de conversão de rótulos normalizados para coordenadas absolutas

Após a etapa de pré-processamento, o modelo foi treinado em utilizando a mesma arquitetura apresentada em [27], com adaptações para manter a compatibilidade com as versões mais recentes do `PyTorch`. Hiperparâmetros, como a taxa de aprendizagem e o tamanho do *batch*, foram otimizados de forma a equilibrar a velocidade de convergência e a capacidade de generalização. O processo de treino foi acompanhado por meio da monitorização contínua das métricas de desempenho e de análise visual das predições geradas ao longo das épocas, assegurando a estabilidade e a coerência dos resultados.

No contexto da deteção de linhas semânticas, as métricas de precisão (*precision*), revocação (*recall*) e *F-measure* foram calculadas a partir da correspondência entre as linhas preditas e as linhas de referência anotadas no conjunto de validação. Para cada limiar de tolerância (variando de 0,01 a 0,99) o algoritmo avaliou o número de verdadeiros positivos (VP) e falsos negativos (FN). A precisão, então, foi definida como $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$, e a revocação como $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$, sendo a *F-measure* obtida pela média harmónica entre ambas [8, 10]. A métrica *F-measure* tradicional corresponde à média global dos valores obtidos em todos os limiares, enquanto a *F-measure@0.95* representa o valor específico calculado com um limiar de 0.95, refletindo o desempenho do modelo sob um critério mais rigoroso de correspondência geométrica entre linhas preditas e reais [27]. A Tabela **4.3** apresenta os resultados do modelo sobre o conjunto de dados de avaliação.

Tabela 4.3: Resultados das métricas de avaliação do modelo

Métrica	Valor
Precision	0.97038
Recall	0.95685
F-measure	0.96356
F-measure@0.95	0.91880

4.4. Inferência e Disponibilização via API

Após o treino, o modelo foi preparado para realizar a inferência sobre novas imagens, de forma a identificar os níveis horizontais correspondentes às prateleiras. Deste modo, o processo de inferência foi estruturado em três fases principais:

1. **Pré-processamento:** a imagem recebida é carregada, redimensionada para uma resolução fixa e normalizada, de modo a manter a compatibilidade com os parâmetros utilizados no treino.

2. **Inferência do modelo:** a imagem processada é encaminhada para a rede neural em `PyTorch`, que gera um mapa de probabilidades indicando as regiões associadas à presença de linhas horizontais.

3. **Pós-processamento:** os resultados do modelo são transformados em segmentos de linha no espaço da imagem, pelo uso de limiares, pela rotulagem de componentes e pelo mapeamento de coordenadas.

O Código **4.2** apresenta uma versão simplificada da função `predict`, destacando as etapas de carregamento da imagem, aplicação das transformações, chamada ao modelo e processamento da saída.

```
from PIL import Image
import torch
from torchvision import transforms

def predict(image_path, model, device):
    # Pre-processamento
    image = Image.open(image_path).convert('RGB')
    transform = transforms.Compose([
        transforms.Resize((400, 400)),
```

```
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])
    image_tensor = transform(image).unsqueeze(0).to(device)

    # Inferencia
    with torch.no_grad():
        output = model(image_tensor)
        output = torch.sigmoid(output)

    # Pos-processamento (simplificado)
    binary_map = output.squeeze().cpu().numpy() > 0.5
    # ...deteccao de linhas e conversao para coordenadas...

    return {"num_lines": len(binary_map.nonzero()[0])}
```

Código 4.2: Exemplo simplificado de inferência no modelo

O resultado final do processo é devolvido num formato estruturado, contendo o número de linhas detetadas e as coordenadas de cada uma delas. Este resultado permite que o *backend* associe a posição física das prateleiras aos produtos registados na base de dados, estabelecendo o elo entre a VC e os dados armazenados. A Figura **4.1** ilustra visualmente este processo, exibindo a imagem das prateleiras após a anotação das linhas horizontais detetadas.

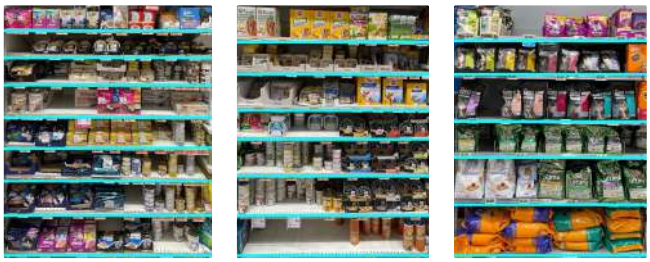

Figura 4.1: Resultado da inferência em imagens de prateleiras

Com o modelo de VC treinado e validado, a etapa seguinte consistiu em disponibilizar a sua funcionalidade através do *backend*. Para tal, foi implementado um *endpoint* de inferência utilizando as mesmas ferramentas empregues no módulo de produtos. Deste modo, o mesmo *backend* passou a centralizar tanto as operações de consulta à base de dados como a execução

do modelo de visão computacional.

O *endpoint* foi definido na rota `/predict`, recebendo como entrada uma imagem enviada por *upload*. Internamente, a imagem é processada de acordo com o procedimento descrito anteriormente, passando pelas etapas de pré-processamento, inferência e pós-processamento. O resultado gerado é devolvido em formato `JSON`, contendo o número de linhas detetadas e as respetivas coordenadas, que indicam a posição horizontal de cada prateleira na imagem analisada. O Código **4.3** ilustra a estrutura de dados devolvida pelo *endpoint* de predição: a primeira propriedade (`num_lines`) indica o número total de linhas identificadas e a segunda (`lines`) lista, para cada linha, quatro valores (x_1, y_1, x_2, y_2) que correspondem, respetivamente, às coordenadas de início e de fim (em píxeis).

Código 4.3: Exemplo de resposta da API /predict

```
{
  "num_lines": 3,
  "lines": [
    [15, 80, 600, 80],
    [20, 200, 610, 200],
    [18, 340, 608, 340]
  ]
}
```

A partir desta abordagem, a *API* pode ser acedida através de chamadas `HTTP` convencionais, e o retorno estruturado em `JSON` permite que os resultados sejam interpretados diretamente pelo lado do cliente. Esta solução simplificou o processo de comunicação entre as camadas, uma vez que tanto as operações sobre a base de dados como os procedimentos de VC estão disponíveis a partir de um único *backend*.

4.4.1. Adaptação do código original

Embora o modelo de VC tenha sido baseado em trabalhos prévios, a sua utilização prática exigiu um conjunto de adaptações no código original. Tal ocorreu principalmente devido a diferenças entre versões de bibliotecas, em especial do `PyTorch`, e à necessidade de ajustar o fluxo às particularidades deste projeto. Entre as principais alterações realizadas destacam-se:

- Compatibilidade com novas versões do `PyTorch`: funções obsoletas foram substituídas por alternativas atuais, como a troca de chamadas tensor.data<T>() por tensor.data_ptr<T>()

em extensões `CUDA/C++`, e nn.functional.upsample por nn.functional.interpolate nos diferentes *backbones* da rede.

- Ferramentas auxiliares de preparação do *dataset*: foram implementados *scripts* adicionais, como o `dataset_adapter`, responsável por converter anotações em formato `CSV` para ficheiros compatíveis com o modelo, e o `label?maker`, que automatiza a geração de listas de imagens a processar. Estas ferramentas foram essenciais para garantir consistência no treino.

- *Script* de inferência independente: foi desenvolvido um *script* auxiliar em `Python` capaz de carregar o modelo treinado e executar a inferência em imagens específicas através da linha de comandos. Este *script* inclui as etapas de pré-processamento, inferência e pós-processamento, permitindo validar resultados de forma rápida e modular, sem depender diretamente do *backend*.

- Ajustes no carregamento de *checkpoints*: o código foi adaptado para aceitar diferentes formatos de armazenamento de modelos, tanto em `state_dict` como em versões completas, garantindo maior flexibilidade na utilização dos ficheiros de treino.

Estas modificações, embora de natureza técnica, foram indispensáveis para viabilizar a aplicação do modelo no contexto do sistema proposto. No final deste processo, o modelo pôde ser integrado de forma consistente ao *backend*, garantindo a execução da inferência e uma comunicação fluida com o restante do sistema.

4.5. Arquitetura Final do Backend

A arquitetura final do *backend* consolidou todos os serviços num único servidor `Python/FastAPI`, responsável por disponibilizar tanto a API de produtos como o *endpoint* de VC baseado em `PyTorch`. O *backend* liga-se a uma base de dados `PostgreSQL` executada em `Docker`, com armazenamento persistente através de volumes; opcionalmente, o `pgAdmin` é utilizado para a administração da base de dados. Esta organização reduz a complexidade de integração, padroniza o ambiente de execução e permite que qualquer cliente `HTTP` consuma os *endpoints*, mantendo o processamento de inferência e a persistência dos dados centralizados no mesmo serviço. A Figura **4.2** apresenta uma visão geral da arquitetura do *backend* do sistema.

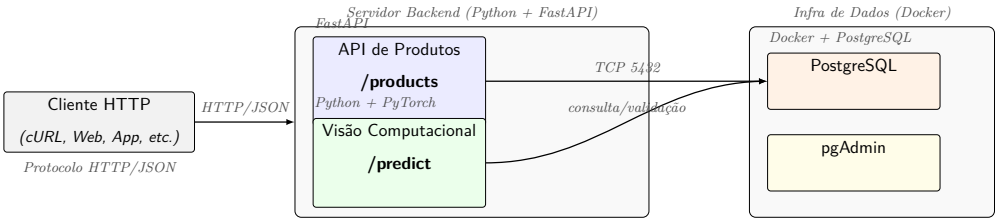

Figura 4.2: Arquitetura final do *backend*, integrando `FastAPI`, `PostgreSQL` e `PyTorch` para os serviços de produtos e VC.

Capítulo 5.

Desenvolvimento do `Frontend`

Este capítulo apresenta o desenvolvimento do *frontend* da aplicação, destacando as etapas de implementação, os desafios técnicos enfrentados e as soluções adotadas. São descritas as abordagens testadas para a localização do utilizador, inicialmente com *beacons* BLE e posteriormente com códigos *QR*, bem como a integração da RA através da biblioteca `ViroReact`. Adicionalmente, são detalhados os mecanismos de alinhamento do mundo virtual, a comunicação com os serviços do *backend* e os ajustes realizados na camada de apresentação, que garantem a coerência e a robustez da experiência de utilização.

5.1. Camada de Apresentação

O fluxo de ecrãs passou por ajustes com o objetivo de proporcionar uma navegação mais fluida. A `HomeScreen` foi renomeada para `SearchProductModal` e apresentada em formato modal, enquanto a `FindProductScreen` foi substituída por `AugmentedRealityScreen`. Para suportar as funcionalidades principais, a biblioteca `react-native-vision-camera` foi utilizada na leitura de códigos *QR*, e a captura de imagens ocorreu através de métodos disponibilizados pela API do `ViroReact`. A utilização do `Expo` simplificou a gestão de permissões e o acesso a recursos nativos, e os testes foram realizados em dispositivos físicos para validar a integração entre câmara, *Bluetooth* e interface.

A arquitetura do código foi organizada de forma modular, com base em componentes reutilizáveis e `hooks` personalizados, o que favoreceu a manutenção e a escalabilidade do sistema. A responsividade da interface foi assegurada através do uso de medidas relativas, permitindo a adaptação dinâmica dos elementos às diferentes dimensões de ecrã. Deste modo, a camada

de apresentação final, ilustrada na Figura **5.1**, manteve o alinhamento com o protótipo original, cumprindo os requisitos de usabilidade, consistência e eficiência, mesmo face às adaptações necessárias ao longo do desenvolvimento.

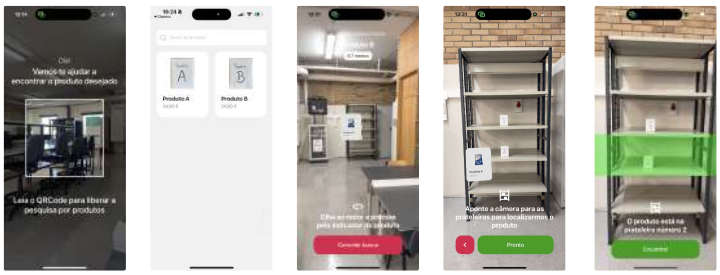

Figura 5.1: Capturas de ecrã representando as principais interfaces implementadas na aplicação móvel.

5.2. Integração com os Serviços do *Backend*

A integração do *frontend* com o *backend* foi realizada através de requisições `HTTP` simples, utilizando o próprio método `fetch` do `React Native`. Esta escolha teve como objetivo reduzir a complexidade da implementação, ainda que bibliotecas mais avançadas, como o `Axios` [51], pudessem oferecer ganhos adicionais de desempenho e flexibilidade. As requisições foram centralizadas em serviços que encapsulam a comunicação com a API, permitindo manter o código organizado e alinhado com os modelos de dados definidos anteriormente.

5.2.1. Integração com produtos

Para garantir a organização e a tipagem, foi definido no *frontend* o modelo `IProduct`, que reflete a estrutura da entidade armazenada na base de dados. Com base neste modelo, foi desenvolvido o serviço `getAllProducts`, responsável por aceder à rota `/products` do *backend*. Este serviço suporta o envio de um parâmetro de pesquisa textual, permitindo filtrar os resultados de acordo com o nome do produto. O retorno é tratado como uma lista de objetos tipados, o que facilita a sua manipulação em diferentes componentes da aplicação. O código referente a este serviço e aos demais serviços subsequentes pode ser consultado no Apêndice **C**.

No componente de listagem de produtos, o serviço é utilizado em conjunto com a biblioteca `React Query` [55], que simplifica a gestão do estado assíncrono e fornece mecanismos de *cache* e atualização automática dos dados. Além disso, foi implementado um mecanismo de *debounce* no campo de pesquisa, evitando múltiplas chamadas ao *backend* em intervalos curtos e melhorando a eficiência da integração.

5.2.2. Integração com o serviço de predição

A integração com o *endpoint* `/predict` foi estruturada em três etapas complementares. Numa primeira fase, a aplicação realiza a captura da imagem da prateleira a partir da cena de RA, guardando-a localmente e obtendo as respetivas dimensões. Em seguida, a imagem é enviada para o *backend* através de uma requisição `POST` no formato `multipart/form-data`, e a resposta em `JSON`, contendo as linhas detetadas, é utilizada para sobrepor uma marcação visual na região correspondente à prateleira do produto selecionado. Este fluxo é apresentado no Apêndice **C.2.1**, através do código da função `handleAnalyze`, que reúne a captura, a chamada ao serviço e a preparação dos dados para renderização.

A submissão do ficheiro ao *backend* foi encapsulada no serviço `detectShelfRows` (Apêndice **C.2.2**), que constrói um objeto `FormData` contendo a imagem e realiza a requisição `POST` para a rota `/predict`. A resposta em `JSON` do *backend* contém o número de linhas detetadas e a lista de coordenadas (x_1, y_1, x_2, y_2) para cada linha, as quais alimentam a etapa de visualização.

Por fim, a visualização do resultado ocorre no componente `AnalyzedShelfView`, que utiliza a biblioteca `Shopify Skia` para desenhar a imagem capturada e sobrepor um retângulo translúcido verde, delimitando a prateleira estimada para o produto selecionado, conforme ilustrado na última captura de tela da Figura **5.1**. A lógica de seleção considera o índice da prateleira indicado pelo produto, trata casos-limite (como o último nível detetado) e ajusta a escala de desenho. A implementação do componente `AnalyzedShelfView` pode ser consultada em detalhe no Apêndice **C.2.3**. Com este arranjo, o *frontend* captura a cena, envia a imagem para o *backend* e destaca, no próprio dispositivo, o nível de prateleira inferido, mantendo a interação simples (`HTTP + JSON`) e visualmente informativa para o utilizador.

5.3. Sistema Global de Coordenadas

A estrutura de navegação foi concebida em torno de um sistema global de coordenadas partilhado por todos os componentes da aplicação. Cada produto armazenado na base de dados está associado a valores fixos em coordenadas espaciais (x, y, z) e ao respetivo nível de prateleira, definindo a sua posição precisa dentro do supermercado. O principal objetivo desta configuração é garantir que tanto os produtos como o utilizador existam numa referência espacial comum, permitindo que a RA se mantenha consistente entre sessões e dispositivos. Uma vez que a RA seja corretamente inicializada neste sistema de coordenadas, as operações de alinhamento subsequentes ficam limitadas a pequenas correções necessárias para compensar desvios do sensor

ou ligeiros erros de rastreamento.

Um dos principais desafios deste modelo reside na determinação da posição inicial e da orientação do utilizador no momento em que a aplicação é iniciada. Uma inicialização precisa é essencial para o alinhamento da câmara virtual com o ambiente físico e para garantir que os indicadores visuais apontem para a localização correta dos produtos. Duas estratégias foram exploradas para resolver este problema: (i) a utilização de *beacons* BLE combinados para estimar a posição inicial através de trilateração e equações de movimento, conforme o planeamento, e (ii) a utilização de códigos *QR* que armazenam coordenadas fixas para fornecer uma inicialização determinística, sendo uma segunda opção de inicialização da RA. Ambas as abordagens partilham o mesmo objetivo conceptual: colocar o utilizador no mesmo sistema global de coordenadas que os produtos.

5.3.1. Alinhamento utilizando *beacons* BLE

A proposta inicial para a navegação em ambientes interiores considerou a utilização de *beacons* BLE para estimar a posição do utilizador dentro do sistema global de coordenadas. Cada *beacon* transmite o seu identificador e o RSSI, permitindo que o dispositivo móvel infira a sua distância aproximada em relação ao *beacon*. Os dados de RSSI foram obtidos através da biblioteca `React Native BLE PLX`. Combinando as distâncias obtidas a partir de, pelo menos, três *beacons*, a posição do utilizador pode ser determinada por trilateração, resultando em coordenadas que servem como ponto de partida para a inicialização do ambiente de RA. A Figura **5.2** ilustra a disposição dos *beacons*, em diferentes alturas, para determinar a posição do utilizador.

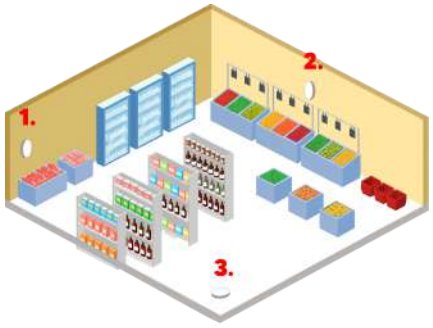

Figura 5.2: Distribuição de três *beacons* `BLE` em diferentes alturas para estimar a posição do utilizador no ambiente.

Uma vez estabelecida a posição do utilizador, a orientação e o movimento são continuamente atualizados com base nos dados dos sensores inerciais do dispositivo. O acelerómetro

fornece valores de aceleração linear (V_x, V_y, V_z), enquanto o giroscópio mede a velocidade angular (W_x, W_y, W_z). Na implementação proposta, estas leituras são acedidas através da biblioteca `expo-sensors`. A combinação destes dois sensores permite a atualização contínua tanto da posição como da orientação. O movimento do dispositivo pode ser modelado de acordo com a Equação **5.1**, que expressa a atualização do vetor de posição em função do estado anterior, do tempo decorrido e da rotação aplicada à aceleração linear:

$$\begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ z_{t-1} \end{bmatrix} + \Delta t \cdot R_t \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad \textbf{Eq. 5.1}$$

A orientação do dispositivo é descrita pela Equação **5.2**, que define como a matriz de rotação é atualizada de forma incremental utilizando o vetor de velocidade angular fornecido pelo giroscópio.

$$R_t = R_{t-1} \cdot \exp\left(\frac{\Delta t}{2}[\omega_t]_\times\right) \quad \textbf{Eq. 5.2}$$

Em conjunto, estas equações descrevem o modelo teórico de movimento que possibilita o rastreamento contínuo da pose do utilizador, combinando a trilateração baseada em BLE para a determinação inicial da posição com os dados dos sensores inerciais para o seu refinamento durante a navegação.

No entanto, a fase de prototipagem revelou que esta abordagem apresentava limitações significativas. O sinal de radiofrequência emitido pelos *beacons* sofre interferência direta de obstáculos físicos e reflete nas superfícies do ambiente, originando fenómenos de atenuação e de multipercurso. Como consequência, o valor do RSSI captado pelo dispositivo variava de forma acentuada, mesmo quando o *smartphone* permanecia imóvel e à mesma distância do *beacon*. Esta instabilidade inviabilizou a obtenção de leituras fiáveis, uma vez que pequenas flutuações no RSSI provocavam grandes variações na distância estimada. Assim, foi adotada uma segunda estratégia, baseada na utilização de códigos *QR* para a identificação inicial das coordenadas do utilizador.

5.3.2. Alinhamento utilizando códigos *QR*

Foi implementado outro método para inicializar a posição do utilizador dentro do sistema global de coordenadas, utilizando códigos *QR* como marcadores visuais, substituindo os *beacons*,

conforme ilustrado na Figura **5.3**. Cada código codifica uma sequência de caracteres que contém uma chave de validação, coordenadas posicionais e coordenadas de rotação, seguindo o padrão "`produtox-camera:x,y,z;x,y,z`". O primeiro triplo representa a posição física do marcador no supermercado, enquanto o segundo corresponde à rotação do dispositivo no momento da leitura. Esta informação permite que a aplicação determine tanto a posição como a orientação necessárias para inicializar o ambiente de RA num ponto fixo dentro do referencial global.

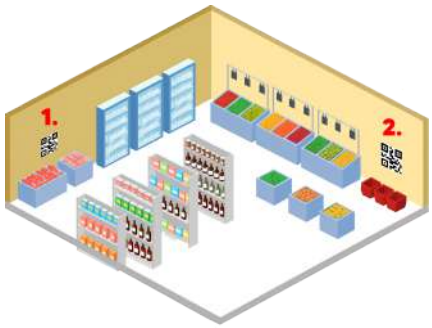

Figura 5.3: Disposição dos códigos *QR* utilizados para inicializar a posição do utilizador no sistema global de coordenadas.

Para garantir uma aquisição precisa dos dados, a aplicação solicita que o utilizador alinhe o código *QR* com um quadrado exibido no ecrã antes de confirmar a leitura, conforme ilustrado na Figura **5.4**. Este procedimento assegura que a câmara e o código estejam posicionados no mesmo plano visual, permitindo ao sistema estimar a distância entre o dispositivo e o marcador com base no tamanho aparente do código *QR* na imagem capturada. Assim que os dados são descodificados, a cena de RA é inicializada nas coordenadas especificadas, e a distância estimada é utilizada para posicionar corretamente a câmara virtual no espaço.

Figura 5.4: Interface da aplicação para leitura e alinhamento de códigos *QR* durante a inicialização da RA.

5.4. Implementação da Realidade Aumentada

A implementação da RA no *frontend* foi construída a partir da biblioteca `ViroReact`, integrada no projeto `Expo`, seguindo o planeamento. O ponto de partida foi a configuração básica de um cenário visual, utilizando o componente `ViroARSceneNavigator`, exportado pelo `ViroReact`, responsável por inicializar e gerir o ciclo de vida da experiência aumentada.

```
<ViroARSceneNavigator
  ref={arRef}
  style={styles.flex}
  initialScene={{
    scene: () =>
      <ViroARScene />
  }}
/>
```

Figura 5.5: Exemplo de inicialização de um cenário visual de RA utilizando o componente `ViroARSceneNavigator`

Conforme ilustrado na Figura **5.5**, este componente recebe como parâmetro uma cena principal, descrita por `ViroARScene`, onde os objetos virtuais são renderizados e onde podem ser capturados eventos de rastreamento e de atualização contínua da câmara. Nos protótipos iniciais, a cena foi preenchida apenas com elementos simples, como textos e objetos básicos, posicionados em diferentes coordenadas para verificar o comportamento do sistema. Estes experimentos serviram para confirmar o funcionamento do ambiente de RA e para compreender de que forma a biblioteca organizava a relação entre os elementos virtuais e o espaço físico detetado pela câmara do dispositivo.

Durante esta etapa inicial, foi explorada a possibilidade de utilizar o componente `ViroCamera` como forma de controlar a posição inicial da câmara dentro da cena aumentada. No entanto, esta abordagem resultou em falhas de execução, acompanhadas de mensagens de erro relacionadas com a manipulação da câmara nativa ("Invalid view returned when removing camera: expected VRTCamera, got [(null)]"). Estes erros impediram a utilização estável do componente, demonstrando que a manipulação direta da câmara não era viável no contexto da biblioteca utilizada.

Esta limitação trouxe implicações importantes para o projeto, uma vez que a posição e a rotação da câmara eram elementos fundamentais para o alinhamento da experiência aumentada com o sistema global de coordenadas definido a partir dos códigos *QR*. Foi, portanto, necessário

procurar uma alternativa que evitasse a manipulação direta da câmara, mas que garantisse, ainda assim, a coerência entre o mundo físico e o virtual.

A solução encontrada consistiu em reorganizar o ambiente em torno de um nó raiz, representado pelo componente `ViroNode` associado a uma referência (`worldRef`). Este nó passou a funcionar como contentor global para todos os objetos virtuais da aplicação. A partir dele, tornou-se possível aplicar transformações de posição e de rotação de forma centralizada, realinhando o mundo virtual através de um sistema de alinhamento do mundo sem a necessidade de alterar diretamente a câmara nativa.

5.4.1. Sistema de alinhamento do mundo

O sistema de alinhamento do mundo é responsável por sincronizar o ambiente de RA com o espaço físico definido pelo sistema global de coordenadas. Uma vez determinada a posição e a orientação iniciais do utilizador, seja por trilateração BLE ou pela descodificação dos dados do código *QR*, o sistema aplica uma transformação que alinha o referencial do mundo virtual com a posição correspondente no mundo real. Esta transformação é definida pelas seguintes equações, que descrevem a rotação e a translação necessárias para alinhar o cenário visual de RA com a referência global. A Equação **5.3** define o vetor de translação entre os dois espaços:

$$T = P_{real} - R \cdot P_{virtual} \qquad \textbf{Eq. 5.3}$$

A Equação **5.4** descreve a aplicação da matriz de transformação que combina a rotação R e a translação T, de modo a alinhar qualquer ponto do ambiente virtual com o seu equivalente no mundo real:

$$P_{alinhado} = R \cdot P_{virtual} + T \qquad \textbf{Eq. 5.4}$$

Por fim, a Equação **5.5** define a matriz de transformação final, que encapsula simultaneamente os componentes de rotação e de translação:

$$M = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \qquad \textbf{Eq. 5.5}$$

Este cálculo é fundamental para garantir que a experiência aumentada mantenha coerência espacial, assegurando que os objetos virtuais previamente posicionados no ambiente apareçam sempre no mesmo local físico, independentemente do código *QR* utilizado para a inicialização.

Esta formulação matemática foi incorporada na aplicação através do desenvolvimento do *hook* `useWorldAlignment`, responsável por centralizar a lógica de alinhamento do mundo. O funcionamento deste módulo pode ser descrito em duas etapas.

Na primeira etapa, executada imediatamente após a leitura de um código *QR*, o *hook* calcula a transformação com base na pose desejada do código *QR* e na pose nativa fornecida pela biblioteca (sempre inicializada com a posição e rotação em 0 para x, y e z). Estes valores são então aplicados diretamente ao nó raiz do cenário visual, representado por `ViroNode`, através da referência do elemento, designada por `worldRef`. Desta forma, todos os elementos virtuais do ambiente passam a herdar as transformações necessárias para o alinhamento.

Na segunda etapa, o *hook* acompanha continuamente as atualizações de pose nativa fornecidas pelo `ARKit/ARCore` e, a cada ciclo, recalcula a posição global do utilizador segundo a equação definida anteriormente. Esta posição é exposta como saída do *hook* (`userPosition`), podendo ser utilizada por outros módulos da aplicação, como o componente responsável por informar a distância até aos produtos ou pela lógica de etapas da navegação em RA.

5.4.2. Garantia de robustez no alinhamento

Durante o desenvolvimento, identificou-se que, mesmo com o cálculo correto da transformação, ainda poderiam ocorrer situações em que o alinhamento era comprometido. Dois cenários críticos mostraram-se recorrentes: a perda de consistência ao regressar à aplicação após esta ser minimizada e a inicialização incorreta da RA devido a movimentos excessivos do dispositivo durante o processo de leitura de um código *QR*. Para lidar com estes problemas, foram implementados dois mecanismos de proteção, denominados *guards*, que asseguram a robustez da solução em condições reais de utilização.

O primeiro mecanismo criado foi um observador do estado da aplicação. O sistema de RA, ao ser minimizado, perde parte das referências necessárias para manter o alinhamento espacial, o que provoca deslocamentos percetíveis dos objetos virtuais quando o utilizador regressa. Para evitar este problema, foi implementado um *hook*, denominado `useAppStateGuard`, que monitoriza as transições de estado da aplicação. Sempre que a aplicação regressa ao estado ativo após ter sido minimizada, o utilizador é redirecionado automaticamente para o ecrã de leitura do código *QR*, garantindo que um novo alinhamento seja efetuado antes da continuação da experiência, conforme ilustrado no fluxo da Figura **5.6**. Esta estratégia assegura que a cena aumentada seja retomada apenas em condições válidas, preservando a integridade da interação.

O segundo mecanismo desenvolvido teve como objetivo evitar que movimentos excessivos do

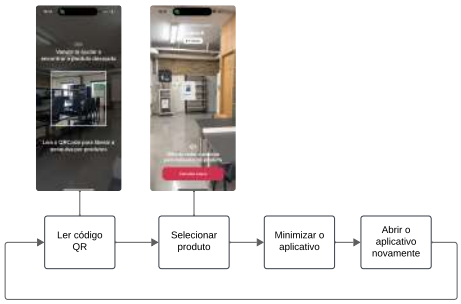

Figura 5.6: Fluxo do mecanismo de segurança que revalida o alinhamento da RA após a aplicação ser minimizada.

dispositivo comprometessem o alinhamento da RA. Durante os testes, observou-se que, ao ler um código *QR* com o telemóvel em deslocação ou submetido a rotações bruscas, o alinhamento resultante apresentava frequentemente inconsistências, uma vez que a pose nativa capturada pelo `ARKit/ARCore` não refletia de forma estável a posição e a orientação da câmara. Para mitigar este problema, foi implementado um *hook*, denominado `useMotionGuard`, baseado nos sensores inerciais do dispositivo, capaz de monitorizar continuamente a aceleração linear e a velocidade angular. Caso seja detetado um movimento excessivo no momento da inicialização da cena aumentada, o utilizador é redirecionado automaticamente para o ecrã de leitura do código *QR*, assegurando que o alinhamento apenas ocorra em condições estáveis, conforme ilustrado no fluxo da Figura **5.7**. Esta estratégia contribui para que a cena aumentada permaneça corretamente sincronizada com os valores de posição e rotação definidos pelo código *QR* no instante da sua captura.

Figura 5.7: Fluxo do mecanismo de proteção que impede a inicialização da RA durante movimentos excessivos do dispositivo.

5.5. Desafios Técnicos e Descobertas

O desenvolvimento do *frontend* foi marcado por uma série de desafios técnicos que exigiram experimentação prática e ajustes sucessivos até que a solução final fosse alcançada. Um dos primeiros obstáculos na implementação da RA foi a interpretação incorreta da ordem dos ângulos

de rotação. Inicialmente, devido à ausência desta informação na sua documentação, assumiu-se que a biblioteca utilizava a convenção [*roll*, *pitch*, *yaw*], o que levou a resultados inconsistentes: objetos virtuais que deveriam estar alinhados a pontos fixos do ambiente apareciam deslocados ou até invertidos, como se estivessem posicionados fora da área esperada. A realização de testes exploratórios com objetos básicos, aplicando rotações conhecidas em nós dentro do cenário visual, foi fundamental para esclarecer esta questão. Estes experimentos confirmaram que a convenção utilizada era, na realidade, [*pitch*, *yaw*, *roll*], o que corrigiu a interpretação e permitiu que os cálculos de alinhamento passassem a refletir corretamente a orientação do espaço físico.

Outro desafio relevante esteve associado ao momento da leitura dos códigos *QR*. Embora cada código armazenasse a posição e a rotação no sistema global, o dispositivo não se encontrava exatamente sobre esse ponto no instante da captura. Existia sempre uma distância entre a câmara e o código *QR*, além de pequenas variações na forma como o utilizador posicionava o aparelho. Este fator resultava em deslocamentos percetíveis dos objetos virtuais, mesmo quando as informações contidas no código *QR* estavam corretas. Para mitigar o problema, foi inicialmente adotado um *offset* fixo, correspondente a uma distância média considerada adequada entre a câmara e o código *QR* no momento da leitura. Esta solução, embora não eliminasse completamente a discrepância, garantiu consistência suficiente para os testes e indica um caminho para melhorias futuras, como a utilização de *Image Anchors* para detetar diretamente a pose do marcador no espaço.

Capítulo 6.

Resultados e Discussões

Este capítulo apresenta os resultados obtidos a partir da implementação e dos testes do sistema desenvolvido, bem como a análise crítica do seu desempenho. São discutidos os procedimentos experimentais realizados, a avaliação da estabilidade e da precisão dos módulos de RA, VC e navegação interna, além da comparação entre os requisitos planeados e os resultados efetivamente alcançados. As observações aqui descritas permitem avaliar a viabilidade técnica da solução proposta e identificar oportunidades de melhoria para trabalhos futuros.

6.1. Visão Geral do Funcionamento da Aplicação

Com o propósito de enquadrar a fase de testes e demonstrar a articulação entre os diferentes módulos implementados, apresenta-se nesta secção uma descrição integrada do funcionamento da aplicação desenvolvida. Durante a sua primeira inicialização, a aplicação solicita ao utilizador permissão para utilizar a câmara, necessária tanto para a leitura de códigos *QR* como para as funcionalidades de RA. Caso a permissão seja concedida, o sistema é iniciado, permitindo a leitura do código *QR*; caso seja negada, o sistema apresenta um ecrã a informar o utilizador de que será necessário conceder a permissão através das definições do dispositivo. A Figura **6.1** ilustra esta primeira interação com o utilizador.

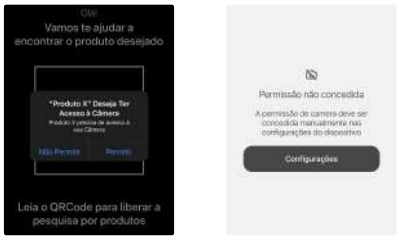

Figura 6.1: Ecrãs iniciais de solicitação de permissões de câmara e início da aplicação.

Uma vez lido um código *QR* disponibilizado no ambiente físico, o sistema inicializa a RA e apresenta um modal de carregamento enquanto esse processo é executado. O modal é encerrado quando a RA indica que o rastreamento do cenário visual se encontra estável. A partir desse momento, o sistema está pronto para interagir com os produtos; assim, é aberto um modal de pesquisa que solicita ao utilizador a seleção de um produto. A Figura **6.2** ilustra tanto o modal de carregamento de inicialização da RA como o modal de pesquisa de produtos.

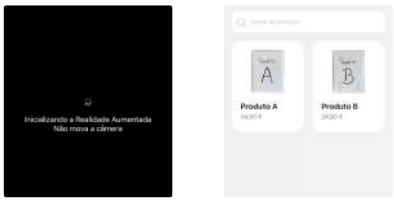

Figura 6.2: Interfaces da aplicação durante a inicialização da RA e a pesquisa de produtos.

Ao selecionar um produto, o sistema armazena o item escolhido num contexto global da aplicação, permitindo que todos os componentes da aplicação o reconheçam, e encerra o modal de seleção. A partir desse momento, o sistema inicia o guiamento visual do utilizador até ao produto, através de um indicador de distância, da exibição visual do produto dentro da RA e de textos informativos no ecrã, conforme ilustrado na primeira captura de ecrã da Figura **6.3**. Ao alcançar uma distância de dois metros do produto (segunda captura de ecrã), é solicitado ao utilizador que aponte a câmara para a prateleira do produto; este processo é indicado por uma vibração do dispositivo e pela alteração dos indicadores visuais. Por fim, após o utilizador pressionar o botão "Pronto", o sistema inicia a análise da imagem, comunicando-se com o *backend* e exibindo a prateleira na qual o produto se encontra (terceira captura de ecrã).

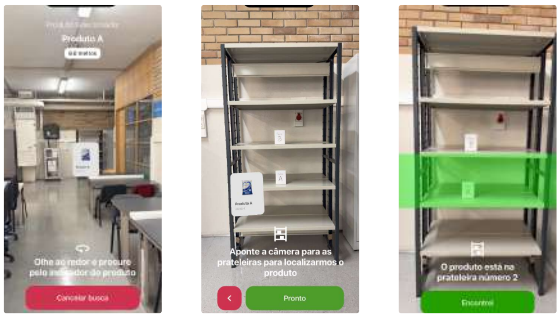

Figura 6.3: Fluxo completo da pesquisa e localização de produtos através de RA.

Durante este processo, o utilizador pode alterar o produto a ser pesquisado através do botão "Cancelar busca", exibido na primeira captura de ecrã, ou do botão correspondente à ação de retroceder, presente na segunda captura de ecrã. Caso o utilizador conclua o processo de pesquisa, é igualmente redirecionado para o modal de pesquisa de produtos ao pressionar o botão "Encontrei".

6.2. Contexto Experimental

Para avaliar o desempenho e a estabilidade do sistema desenvolvido, foram realizados testes experimentais controlados num ambiente que simula a operação da aplicação dentro de um supermercado. Os experimentos tiveram como objetivo analisar o comportamento da aplicação nas principais dimensões de desempenho, nomeadamente: tempo de resposta, robustez, consumo de memória e consistência entre plataformas, bem como validar o funcionamento dos componentes de RA, VC, mecanismos de segurança (*guards*) e a precisão do alinhamento do mundo. A Tabela **6.1** apresenta os dispositivos e as versões de software utilizados durante os experimentos.

Tabela 6.1: Configuração dos componentes utilizados nos experimentos.

Componente	**Versão / Modelo**
iPhone 15 Pro	iOS 18.6.1
Poco X5 Pro	Android 14
Expo SDK	54
ViroReact	2.43.5
react-native-ble-plx	3.5.0
react-native-vision-camera	4.7.2

A escolha destes dispositivos móveis teve como objetivo comparar o comportamento da

aplicação nas duas plataformas, uma vez que o sistema depende fortemente dos *frameworks* `ARKit` (no iOS) e `ARCore` (no Android) para o funcionamento da RA. Durante os testes, observou-se que a diferença entre estas bibliotecas teve um impacto direto no alinhamento do mundo virtual e no comportamento dos componentes de rastreamento, sendo um dos fatores mais críticos para a estabilidade do sistema.

O ambiente de teste foi configurado de modo a permitir a execução completa do fluxo da aplicação, incluindo a leitura de códigos *QR*, o carregamento do cenário visual de RA, a seleção de produtos, a análise visual das prateleiras e o retorno dos resultados. Em virtude das diferenças entre os sistemas operativos, foram registadas variações de comportamento, especialmente no processo de captura de imagem, conforme detalhado nas secções seguintes.

6.2.1. Subaplicação de calibração

Durante a preparação do ambiente de testes, observou-se a necessidade de calibrar frequentemente a posição virtual dos produtos, de forma a garantir que o alinhamento entre os objetos renderizados em RA e os produtos físicos permanecesse preciso ao longo das execuções, permitindo a validação de diferentes posições do produto. Pequenas variações no posicionamento do dispositivo durante a leitura do código *QR* ou diferenças na iluminação do ambiente podiam resultar em desalinhamentos percetíveis, comprometendo a coerência da navegação e da análise visual.

Para resolver este problema, foi desenvolvida uma subaplicação de calibragem integrada na aplicação principal, permitindo o ajuste das coordenadas tridimensionais e do nível de prateleira dos produtos antes do início de cada ciclo de testes. Para garantir o acesso a esta funcionalidade, foi incluído um novo ecrã inicial, no qual o utilizador pode escolher entre "Encontrar produtos" e "Calibrar produtos", conforme ilustrado na Figura **6.4**. Importa referir que este ecrã não deveria ser exibido ao utilizador final (cliente do supermercado), tendo sido incluído apenas para fins de teste.

Figura 6.4: Ecrã inicial da aplicação com as opções de calibração e de busca por produtos.

Ao selecionar a opção de calibragem, o sistema ativa a câmara, redireciona o utilizador para o ecrã de leitura de códigos *QR* e, após a leitura, encaminha-o para a interface `CalibrateScreen`, ilustrada pela Figura **6.5**. Este ecrã apresenta um comportamento semelhante ao da interface principal, embora de forma simplificada: desconsidera a comunicação com o serviço de VC e remove os elementos visuais não essenciais, concentrando-se exclusivamente na atualização das coordenadas do produto.

Figura 6.5: Ecrã de calibração (`CalibrateScreen`) utilizado para atualizar as coordenadas dos produtos.

O processo de calibração consiste em posicionar o dispositivo móvel no mesmo local físico do produto e pressionar o botão "Calibrar". Neste momento, as coordenadas da posição atual do utilizador são capturadas e atribuídas ao produto selecionado. De seguida, o sistema solicita que o utilizador indique o nível de prateleira em que o item se encontra, registando o valor na base de dados através de uma requisição `HTTP PUT` para o *endpoint* de atualização de produtos. A Figura **6.6** ilustra este processo.

Esta operação é implementada no *frontend* através do serviço `updateProduct`, responsável por enviar ao *backend* as novas informações de localização e atualizar o registo correspondente na

Figura 6.6: Modal de seleção de prateleira exibido durante o processo de calibração.

base de dados. A função é executada de forma assíncrona, garantindo uma resposta imediata à interação do utilizador e notificando-o sobre o sucesso da operação através de mensagens visuais e hápticas.

Além de facilitar o alinhamento do mundo virtual, a subaplicação de calibragem desempenhou um papel crucial na fiabilidade dos experimentos, permitindo ajustes rápidos entre execuções e assegurando que todos os produtos testados estivessem devidamente posicionados no espaço tridimensional da RA. Deste modo, foi possível reduzir significativamente o erro de posicionamento observado nas primeiras versões do sistema e obter medições mais consistentes nas etapas seguintes. O fluxo descrito na Figura **6.7** ilustra todo o processo de calibração de um produto.

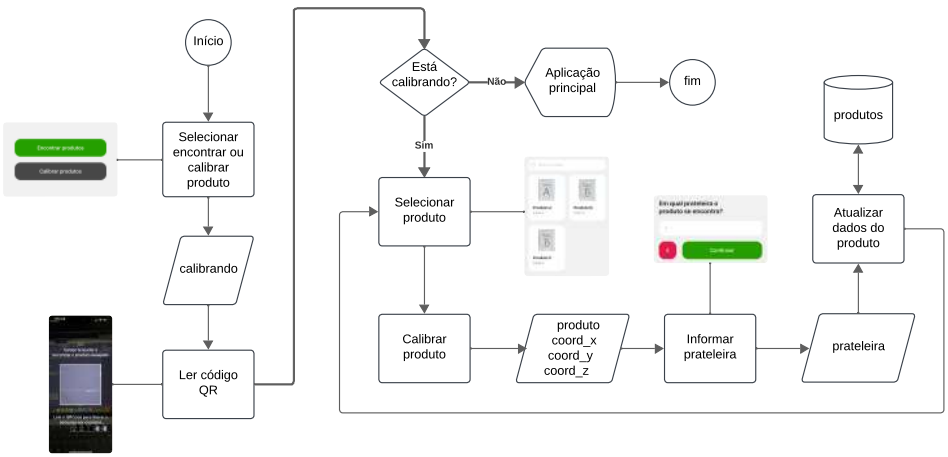

Figura 6.7: Fluxo do processo de calibração implementado na subaplicação.

6.3. Avaliação de Desempenho

A avaliação de desempenho teve como objetivo analisar o comportamento do sistema sob diferentes condições de execução, comparando as plataformas Android e iOS em termos de tempo de resposta, fluidez da renderização e estabilidade da RA. Estas medições permitiram verificar se a aplicação cumpre os requisitos de desempenho definidos no planeamento do sistema, bem como identificar situações que possam comprometer a experiência do utilizador. Para tal, foram instrumentadas funções no código-fonte que registam automaticamente os tempos de execução em pontos críticos do fluxo, como a inicialização do cenário visual de RA, a renderização do cartão de produto e o processo de análise de imagem.

6.3.1. Tempo de inicialização da realidade aumentada

O tempo de inicialização da RA representa o intervalo entre a leitura do código *QR* e o momento em que o rastreamento do ambiente é estabilizado e o cenário visual é completamente renderizado. Esta métrica é fundamental para avaliar a fluidez da experiência do utilizador, pois define o tempo de espera até que a aplicação esteja operacional. Para esta medição, foi implementado um cronómetro interno, com o auxílio de um contexto *React*, que se inicia no evento de leitura do código *QR* e é finalizado quando a cenário visual é estabilizado.

Para quantificar o tempo de inicialização, foram realizadas cinco execuções consecutivas em cada plataforma. Os resultados apresentaram um tempo médio de aproximadamente 2,1 segundos em ambas as plataformas, valor que se mantém dentro da margem aceitável para o requisito não funcional RNF04 (tempo de resposta inferior a 2 segundos). A Tabela **6.2** apresenta os tempos medidos, bem como as respetivas médias e desvios-padrão.

Tabela 6.2: Tempo de inicialização da RA em iOS e Android.

Plataforma	Tempos medidos (ms)	Média (ms)	Desvio padrão (ms)
iOS	2084, 2026, 2216, 2231, 2199	2151	88
Android	2364, 1766, 2215, 2082, 1774	2040	238

No entanto, durante os testes foi observada uma instabilidade inicial no Android, especialmente nas versões anteriores do `Expo SDK 53`. Em alguns casos, o rastreamento do plano horizontal não era corretamente estabilizado, resultando em pequenas flutuações na posição dos objetos virtuais e, ocasionalmente, em reinicializações espontâneas do cenário visual de RA. Este

comportamento foi mitigado após a atualização das bibliotecas `Expo` e `ViroReact` para as versões 54 e 2.43.5, respetivamente, mas demonstra que a estabilidade do rastreamento é altamente dependente da compatibilidade entre as versões das bibliotecas e os controladores (*drivers*) do dispositivo.

Apesar destas limitações, o desempenho final apresentou uma consistência satisfatória, com tempos uniformes e sem bloqueios percetíveis para o utilizador. A inicialização rápida do cenário visual de RA permite que o fluxo de navegação ocorra de forma natural, reduzindo o tempo de inatividade entre a leitura do código *QR* e a visualização do ambiente aumentado.

6.3.2. Tempo de exibição do cartão de produto

O tempo de exibição do cartão de produto representa o intervalo entre a escolha de um produto e o momento em que o componente visual que contém as informações do produto é totalmente renderizado no ecrã. Esta métrica avalia a capacidade de resposta da camada de apresentação, refletindo diretamente na perceção de fluidez da aplicação. Para medir este tempo, foi inserido um marcador de início logo após o utilizador pressionar um cartão na listagem de produtos e um marcador de final no evento de montagem do componente `ARProductCard`, utilizando o mesmo cronómetro interno empregado anteriormente. Foram realizadas cinco execuções consecutivas em cada plataforma, com os resultados apresentados na Tabela **6.3**.

Tabela 6.3: Tempo de exibição do produto na RA.

Plataforma	**Tempos medidos (ms)**	**Média (ms)**	**Desvio padrão (ms)**
iOS	23, 15, 13, 13, 18	16.4	4.2
Android	76, 55, 46, 46, 62	57.0	12.6

Os resultados demonstram que o sistema apresentou um comportamento consistente nas duas plataformas, com tempos inferiores a 100 ms e baixa dispersão no iOS. Em média, o iOS registou 16,4 ms (σ = 4,2 ms) e o Android 57,0 ms (σ = 12,6 ms), valores que caracterizam uma resposta imediata do ponto de vista percetivo. Para o tipo de aplicação proposta, este intervalo cobre a transição entre o toque no cartão e a presença do objeto no cenário virtual, sem interromper o fluxo de navegação aumentada.

Durante os testes, no entanto, observaram-se no Android pequenas oscilações visuais entre execuções consecutivas. Estas variações foram associadas à competição entre a *UI thread* e o motor de *JavaScript*, que podem introduzir atrasos intermitentes na montagem do componente

e na sinalização do evento `onLayout`, correspondente ao "modelo carregado". Embora o impacto prático seja mínimo na experiência, o fenómeno evidencia uma maior sensibilidade do Android à carga de renderização e ao escalonamento de tarefas.

De forma geral, o tempo de exibição do produto na RA pode ser considerado plenamente adequado ao uso em contexto real, uma vez que se mantém abaixo do limiar de perceção de atraso pelo utilizador e não introduz descontinuidades no início da navegação aumentada.

6.3.3. Tempo de análise de imagem

O tempo de análise de imagem corresponde ao intervalo entre o momento em que o utilizador aciona a função de captura da prateleira (botão "Pronto") e o instante em que o sistema recebe a resposta do *backend* com as linhas identificadas pelo modelo de VC, renderizando-as sobre a imagem capturada, conforme ilustrado na Figura **6.8**.

Figura 6.8: Acionamento do botão "Pronto" e resultado da detecção do nível de prateleira.

Esta métrica reflete o desempenho agregado de todas as etapas que compõem o *pipeline* de deteção, nomeadamente: captura, codificação, envio, inferência, resposta do servidor e renderização. Esta abordagem fornece uma visão abrangente da eficiência do processo de análise visual. Foram realizadas cinco execuções consecutivas na plataforma iOS para quantificar o tempo total de processamento. No caso do Android, no entanto, não foi possível registar a métrica diretamente devido a uma falha no método `takeScreenshot` da API `ViroReact`, que impediu a captura e o envio da imagem. A Tabela **6.4** apresenta os tempos medidos no *iOS*.

Tabela 6.4: Tempo de análise de imagem e resposta do modelo de VC.

Plataforma	**Tempos medidos (ms)**	**Média (ms)**	**Desvio padrão (σ)**
iOS	3845, 4012, 3976, 3822, 3905	3912	77
Android	*Não foi possível medir devido ao erro no método* `takeScreenshot`.		

Os valores obtidos incluem tanto o tempo de inferência do modelo `PyTorch`, hospedado no *backend*, como a latência de rede entre a aplicação e o servidor. Considerando que a inferência do

modelo consome aproximadamente 0,5 s na primeira execução e 0,3 s nas execuções subsequentes em ambiente local, o restante do tempo é atribuído ao processo de captura e transferência da imagem. Estes resultados servem de base para a análise detalhada da componente de VC, apresentada na secção seguinte.

Durante os testes realizados na plataforma Android, a função `takeScreenshot` da API `ViroReact` apresentou falhas recorrentes, impedindo a execução completa do fluxo de captura de imagem. O erro, identificado nos registos, ocorria na chamada assíncrona ao modulo interno de gestão de interface do `React Native`, denominado `UIManagerModule.addUIBlock`. Em cada tentativa de captura, a aplicação era interrompida antes da conclusão do processo de renderização, o que impossibilitou a obtenção direta do tempo de análise de imagem e a validação integral do *pipeline* na plataforma Android.

Foram testadas alternativas à função nativa, como as bibliotecas `react-native-view-shot` [17] e `@shopify/react-native-skia` [40], mas ambas apresentaram limitações. A primeira não conseguia capturar corretamente o *frame buffer* do cenário visual de RA, resultando em imagens incompletas, enquanto a segunda introduzia latência excessiva devido à necessidade de processamento adicional no *canvas*. Em ambos os casos, as tentativas não se revelaram adequadas para medições consistentes de desempenho.

A impossibilidade de capturar a imagem comprometeu a análise quantitativa desta etapa no Android, embora o comportamento visual observado indicasse que o tempo total de processamento seria semelhante ao do iOS. Para confirmar esta hipótese, seria necessário repetir os testes noutro dispositivo, preferencialmente com especificações e versões de sistema distintas, de modo a descartar a possibilidade de o problema estar associado a uma configuração específica do aparelho utilizado.

Em busca de esclarecimentos, foi aberta uma discussão no servidor oficial de comunicação dos programadores do `ViroReact`, onde foi confirmado que o erro já havia sido reportado pela comunidade e se encontra em processo de correção em versões futuras da biblioteca. Esta informação sugere que a falha é de natureza interna ao *framework*, e não uma limitação intrínseca da arquitetura proposta neste trabalho. Assim, uma atualização futura do `ViroReact` poderá permitir a implementação completa desta funcionalidade no Android, viabilizando medições e aumentando a robustez do sistema multiplataforma.

6.4. Análise da Estratégia de Navegação Interna

A navegação interna foi inicialmente concebida para operar com base em sinais de *beacons* BLE, utilizando RSSI para estimar a distância entre o utilizador e os pontos de referência instalados no ambiente. No entanto, os resultados experimentais obtidos revelaram que, apesar de os testes terem sido realizados em condições controladas, o comportamento do sinal apresentou uma variação significativa, comprometendo a precisão das medições e, consequentemente, a fiabilidade da localização. Perante este cenário, foi avaliada a viabilidade de substituir a abordagem por uma solução baseada em códigos *QR*, cuja utilização se revelou mais estável e de menor custo operacional. Esta secção discute em detalhe o desempenho da abordagem original com *BLE*, os fatores que motivaram a transição para códigos *QR* e a possibilidade de integração híbrida entre ambas as tecnologias.

6.4.1. Avaliação da abordagem com *beacons* BLE

Para avaliar a viabilidade da utilização de *beacons* BLE para posicionamento em ambientes interiores, foram recolhidas amostras de sinal a distâncias fixas de 1, 2 e 3 metros de um *beacon*, em condições laboratoriais controladas. Cada medição consistiu em 100 leituras de RSSI, obtidas através da biblioteca `react-native-ble-plx` [33]. Tanto o *smartphone* como o *beacon* foram posicionados sobre uma superfície plana, mantendo uma distância considerável de obstáculos físicos ou de fontes de interferência eletromagnética. Esta configuração teve como objetivo eliminar variáveis externas e obter leituras representativas do comportamento do sinal num cenário ideal. A Figura **6.9** ilustra a variação das distâncias estimadas para cada condição de teste, destacando as flutuações significativas observadas nas medições de RSSI em relação às distâncias reais.

Figura 6.9: Variabilidade da estimativa de distância baseada em RSSI em comparação com as distâncias reais de 1m, 2m e 3m.

Os valores de RSSI foram então convertidos em estimativas de distância utilizando um modelo de perda de percurso logarítmico. A Equação (**6.1**) expressa esta relação, em que d representa

a distância estimada em metros, M corresponde ao valor de RSSI medido a 1 metro (definido como -64 dBm) e n é o expoente de perda de percurso (definido como 2.5).

$$d = 10^{\frac{(M-RSSI)}{10n}} \quad \textbf{Eq. 6.1}$$

Mesmo nestas condições controladas, os resultados evidenciaram variações expressivas na estimativa de distância, com erros que frequentemente ultrapassaram 1,5 metros. Esta instabilidade não estava relacionada com a presença de obstáculos, mas sim com oscilações inerentes ao próprio sinal BLE, influenciadas por fatores como reflexão em superfícies, flutuações de potência de transmissão e ruído de radiofrequência. Num sistema de navegação em corredores estreitos, tal variação seria suficiente para posicionar incorretamente o utilizador num corredor adjacente, comprometendo toda a experiência de navegação.

A análise indica que, mesmo num cenário ideal, o BLE isoladamente não oferece precisão adequada para ambientes que exigem localização de alta resolução, como supermercados. Além disso, erros na estimativa de distância afetariam diretamente o alinhamento do mundo virtual na RA, podendo posicionar o cartão de produto num local incorreto, por exemplo, sobre uma prateleira diferente da pretendida. Estes resultados reforçam que, embora o BLE seja eficiente para estimativas macroscópicas de proximidade, carece de estabilidade para utilização como referência posicional absoluta.

6.4.2. Justificação da transição para códigos *QR*

A substituição dos *beacons* por códigos *QR* não ocorreu devido a uma limitação técnica da aplicação, mas sim como uma decisão fundamentada em critérios de precisão e estabilidade. Durante o desenvolvimento, observou-se que o `ViroReact` já exportava coordenadas tridimensionais da posição do utilizador assim que o cenário visual de RA era inicializado. No entanto, essas coordenadas eram sempre inicializadas em 0 para X, Y e Z, referindo-se ao sistema de coordenadas local do dispositivo, o que exigia a conversão para um sistema de coordenadas global (descrito na secção **5.3**), de modo a permitir uma navegação consistente no ambiente físico.

Com isso, verificou-se que os códigos *QR* poderiam atuar como pontos de ancoragem globais, fornecendo diretamente a posição do utilizador no espaço físico a partir da leitura do código. Esta abordagem simplificou a integração entre o mundo real e o virtual, permitindo converter as coordenadas locais geradas pelo `ViroReact` para o sistema global utilizado. O resultado foi uma navegação mais precisa e determinística, sem depender das variações do RSSI.

Além disso, esta solução apresentou vantagens de baixo custo e manutenção simples, uma vez que bastam impressões físicas colocadas em locais estratégicos, sem necessidade de dispositivos eletrónicos adicionais. Esta transição também permitiu a continuidade do trabalho e a implementação do alinhamento do mundo virtual com maior fiabilidade, eliminando ambiguidades que o BLE não seria capaz de resolver no seu estado atual.

6.4.3. Possível integração híbrida

Embora a abordagem com códigos *QR* tenha demonstrado maior eficiência e estabilidade, uma integração híbrida BLE + *QR* surge como uma alternativa promissora para aumentar a robustez da navegação. Nesta configuração, os *beacons* poderiam ser utilizados para estimar a proximidade geral do utilizador, enquanto os códigos *QR* forneceriam as referências precisas de posição e orientação no momento da leitura. Deste modo, o BLE atuaria como um sistema de posicionamento contínuo de baixa precisão, corrigido periodicamente por ancoragens visuais fornecidas pelos códigos *QR*.

Uma implementação mais avançada poderia incluir um conjunto de *beacons* intercomunicantes, realizando trilateração entre si para mitigar o impacto de variações individuais no RSSI. No entanto, esta estratégia implicaria uma maior complexidade de instalação e manutenção, além de custos adicionais associados à calibração e sincronização. Outra alternativa teórica seria empregar roteadores *Wi-Fi* como emissores de sinal RSSI, substituindo os *beacons*, mas o custo e a dependência de infraestrutura tornariam inviável a aplicação em escala comercial.

Com base nos resultados obtidos, conclui-se que a utilização isolada do BLE não satisfaz as necessidades de precisão e estabilidade exigidas para a navegação aumentada em supermercados. Ainda assim, a sua utilização em conjunto com códigos *QR* representa uma linha de evolução viável e tecnicamente justificável, capaz de equilibrar precisão, custo e escalabilidade em futuras versões do sistema.

6.5. Robustez, Estabilidade e Alinhamento

A robustez da aplicação de RA foi avaliada com base na sua capacidade para manter a coerência visual e funcional em situações de utilização contínua, variação de carga e interrupções ocasionais. Durante os testes, o sistema demonstrou um comportamento estável em ambas as plataformas, mantendo a execução contínua da RA sem bloqueios na maioria das execuções.

O principal fator de instabilidade identificado ocorreu na *build* para Android, durante o retorno do ecrã de RA para o ecrã de leitura de código *QR*. O erro estava relacionado com uma tentativa de redirecionamento enquanto um modal de carregamento permanecia aberto, o que gerava um conflito no ciclo de navegação do `React Native` e resultava na exceção `"Invariant Violation: Attempted to transition while another transition is in progress"`. Esta falha foi observada especificamente em cenários nos quais o `MotionGuard`, ao detetar movimento excessivo do dispositivo, redirecionava automaticamente o utilizador para o ecrã `QRCodeScreen`. Assim, o fluxo que originava o erro era o seguinte:

1. Leitura do código *QR*;
2. Exibição do modal de carregamento;
3. Movimento do dispositivo;
4. Ativação do `MotionGuard`;
5. Redirecionamento para o ecrã de leitura do código *QR*;
6. Encerramento inesperado da aplicação (erro).

Após a identificação da causa, o problema foi mitigado através da inserção de uma rotina de verificação de estado que assegura o encerramento do modal antes de qualquer redirecionamento. Esta solução reduziu significativamente a ocorrência do erro, embora execuções isoladas ainda possam apresentá-lo. No ambiente iOS, o comportamento não foi reproduzido, mantendo-se estável em todos os ciclos de teste.

Além deste incidente, também foram observadas situações de ligeira imprecisão na posição inicial dos objetos de RA. Em alguns casos, especialmente quando o sistema era iniciado a uma distância maior do produto, a RA apresentava um pequeno desalinhamento na exibição do produto. Este comportamento é atribuído a microvariações na angulação do dispositivo durante a leitura, que, embora não violem o limite permitido pelo `MotionGuard`, são amplificadas quanto maior for a distância entre o utilizador e o produto.

Por exemplo, ao realizar a leitura do código *QR* a aproximadamente 10 metros do produto, mesmo uma variação angular mínima no momento da leitura resulta num desvio percetível na renderização do cartão de produto. A Figura **6.10** ilustra este efeito, mostrando o deslocamento visual causado pela diferença entre a posição esperada e a posição efetiva do objeto virtual.

Figura 6.10: Desalinhamento causado por microvariações na angulação do dispositivo durante a inicialização.

Os testes indicam que os hiperparâmetros definidos para o `MotionGuard` já operam no limite do aceitável para uma boa experiência de utilização. Tornar o filtro mais restritivo reduziria o risco de desalinhamento, mas prejudicaria a usabilidade, interrompendo a inicialização mesmo com movimentos subtis. Assim, esta imprecisão angular residual representa um compromisso entre estabilidade e conforto de interação.

Em síntese, o sistema demonstrou uma estabilidade satisfatória e uma resposta previsível durante os testes, sendo capaz de manter a coerência da cena aumentada mesmo após pausas temporárias ou pequenas perturbações de movimento. As ocorrências descritas reforçam, contudo, a importância de investigar estratégias complementares, como a integração de dados de posicionamento contínuo (BLE) com referências visuais (códigos *QR*), capazes de corrigir automaticamente desvios angulares e reforçar a robustez global da solução.

6.6. Desempenho da Visão Computacional

A etapa de VC foi avaliada a partir da execução do modelo de deteção de linhas de prateleira em dois contextos distintos: (i) imagens capturadas em laboratório, representando prateleiras de teste, e (ii) imagens obtidas num supermercado real, nunca antes vistas pelo modelo. Esta comparação permitiu analisar o comportamento do modelo sob diferentes condições de iluminação, textura e perspetiva, além de verificar a sua capacidade de generalização para novos ambientes.

6.6.1. Tempo de inferência

O tempo de inferência foi medido em dois cenários: diretamente no ambiente `Python`, através de um *script* que carrega o modelo treinado, processa uma imagem de entrada e executa a

deteção das linhas de prateleira, devolvendo as coordenadas das linhas identificadas, e por meio do *endpoint* `/predict` implementado no *backend* `FastAPI`. A Tabela **6.5** apresenta os resultados obtidos.

Tabela 6.5: Tempos de inferência do modelo de VC.

Execução	**Método**	**Tempo (ms)**
1–10	`infer.py` (execução direta)	198–230 (média $\approx$ 210)
1–10	`/predict` (endpoint FastAPI)	74–502 (média $\approx$ 135)

Os resultados mostram que o *endpoint* `FastAPI` apresentou um desempenho superior ao *script* executado diretamente, com um tempo médio de inferência cerca de 35% inferior. Tal facto ocorre porque o *backend* mantém o modelo, a *GPU* e as bibliotecas de aceleração (*cuDNN*) carregadas em memória após a primeira requisição, evitando o custo de inicialização do `PyTorch` em cada execução.

Em contrapartida, o *script* `infer.py` reinicializa o processo a cada execução, incluindo o carregamento dos pesos e a configuração da *GPU*, o que explica os tempos mais elevados e estáveis. Esta diferença evidencia a importância de manter o modelo residente em memória em aplicações que exigem respostas rápidas e sucessivas, como a aplicação móvel desenvolvida.

Apesar do bom desempenho do *backend*, observou-se que o tempo total de inferência registado dentro da aplicação foi significativamente superior. Esta diferença poderá estar relacionada com o processo de envio da imagem via `multipart/form-data` e com a latência de rede, que aumentam o tempo até ao retorno do resultado. Uma possível otimização seria a adoção de bibliotecas de requisição mais eficientes, como o *Axios*, ou a compressão e redimensionamento das imagens antes do envio, reduzindo o volume de dados transmitidos.

6.6.2. Comportamento do modelo em diferentes ambientes

Os testes de inferência foram realizados com cinco imagens do laboratório e cinco imagens do supermercado. As imagens laboratoriais apresentaram resultados inconsistentes, enquanto as imagens capturadas no supermercado real evidenciaram um desempenho superior, conforme ilustrado na Figura **6.11**.

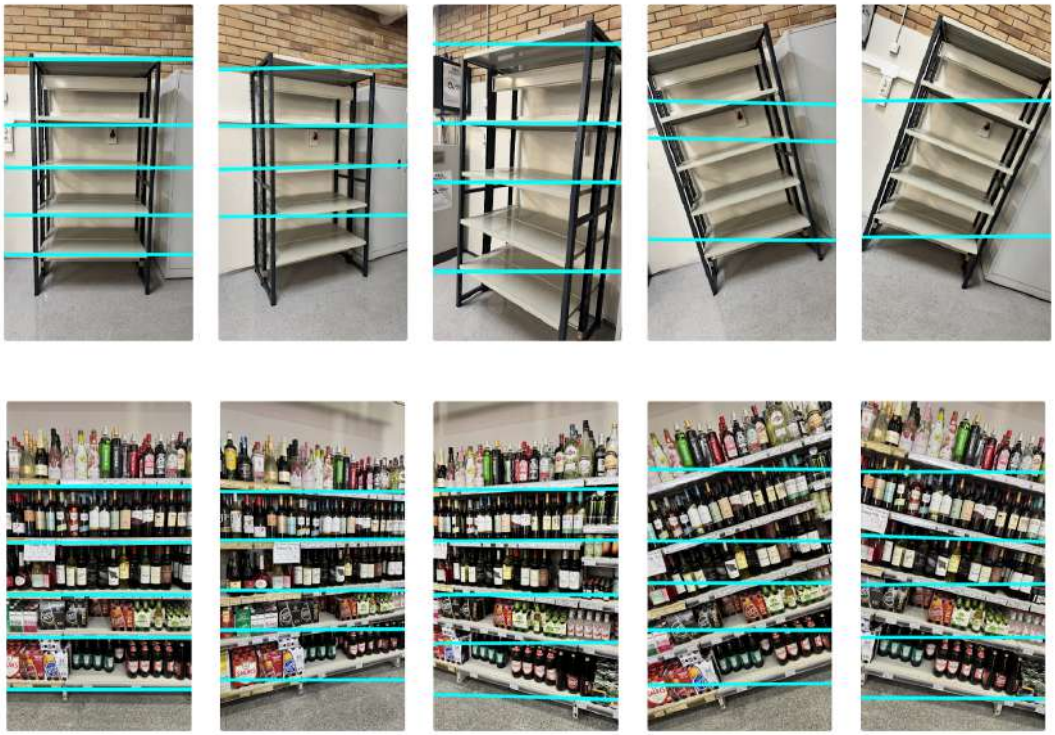

Figura 6.11: Comparação entre inferências em prateleiras do laboratório (acima) e de supermercado (abaixo).

Em ambos os contextos, as linhas geradas permaneceram horizontais, independentemente da inclinação da imagem. A investigação indicou que este comportamento decorre da conversão incorreta dos rótulos originais: os valores percentuais foram convertidos para coordenadas absolutas, o que levou o modelo a aprender apenas linhas perfeitamente horizontais, conforme descrito anteriormente na subsecção **4.3.1**. Como consequência, o modelo apresentou um desempenho satisfatório apenas em imagens frontais e bem enquadradas, não se adaptando, porém, a variações de perspetiva ou inclinações.

Nas imagens laboratoriais, além da limitação geométrica, o modelo identificou incorretamente o número de prateleiras em algumas execuções. Já nas imagens capturadas no supermercado real, a deteção foi visualmente coerente e compatível com o domínio original de treino, indicando que o modelo funciona adequadamente dentro das condições para as quais foi projetado, mas não generaliza de forma eficaz para cenários que extrapolam esse domínio.

Estes resultados mostram que o modelo, embora funcional como prova de conceito e bem integrado no sistema de RA, ainda não satisfaz plenamente os requisitos de aplicação prática. A sua utilização é viável em ambientes controlados ou com orientação de captura específica, mas o desempenho pode degradar-se significativamente em situações de utilização real, nas quais o enquadramento e a angulação da câmara variam entre utilizadores. Embora o problema

tenha sido identificado e uma possível solução (ou pelo menos melhoria) tenha sido considerada passando pela recriação dos dados de treino com novas etiquetas e treino de um novo modelo, as limitações temporais deste trabalho não permitiram comprovar na prática essa solução.

6.6.3. Considerações sobre otimização e aplicabilidade

Os resultados da avaliação evidenciam que o modelo apresenta um tempo de inferência compatível com aplicações móveis. No entanto, a latência observada no fluxo de comunicação entre a aplicação e o *backend* demonstra que a etapa de envio e processamento das imagens constitui o principal gargalo temporal do sistema. Para mitigar o tempo total de resposta, as melhorias devem concentrar-se principalmente na otimização do fluxo de comunicação entre a aplicação e o *backend*. Estratégias como a utilização de bibliotecas de requisição mais eficientes (por exemplo, o `Axios` em substituição ao *fetch* nativo) e a compressão das imagens antes do envio podem reduzir significativamente a latência sem alterar a arquitetura atual. Outra alternativa seria o tratamento assíncrono das requisições, permitindo que a aplicação mantenha a interação com o utilizador enquanto aguarda o retorno da inferência, minimizando a perceção de atraso.

6.7. Comparação entre Requisitos Planeados e Resultados Obtidos

Com base nos testes e experimentos apresentados neste capítulo, foi realizada uma análise comparativa entre os requisitos funcionais e não funcionais definidos no Capítulo **3** e listados no Apêndice **B** , confrontando-os com os resultados efetivamente alcançados pela implementação. Esta comparação permite avaliar o grau de cumprimento de cada requisito e das capacidades centrais.

6.7.1. Capacidades centrais

As três capacidades principais definidas foram:

- **Pesquisa e seleção de produtos**: *Atendida.* As funções de pesquisa e listagem de produtos revelaram-se funcionais, com integração direta no fluxo de leitura do código *QR* (CU01–CU02).
- **Navegação até ao produto em tempo real**: *Parcialmente atendida.* A distância até

ao produto foi exibida de forma responsiva, embora tenham sido observadas pequenas imprecisões de alinhamento a maiores distâncias e em dispositivos Android.

- **Confirmação visual da prateleira correta**: *Parcialmente atendida.* O modelo de visão deteta os níveis de prateleira, mas encontra-se limitado a linhas horizontais, apresentando melhor desempenho em imagens reais de supermercado.

6.7.2. Requisitos atendidos

- **RF01** — O utilizador consegue ler um código *QR* para ativar o sistema, que funciona como ponto de referência global e inicializa corretamente a RA.
- **RF02** — É possível reiniciar a pesquisa por produtos a qualquer momento, mantendo o fluxo de navegação contínuo.
- **RF03** — A distância até ao produto é exibida corretamente.
- **RF05** — O sistema apresenta notificações por vibração e mensagens visuais integradas no fluxo de navegação.
- **RF06** — O utilizador é notificado ao aproximar-se do produto, validando o fluxo de alerta de proximidade (CU03).
- **RFN06** — O código apresenta uma arquitetura modular e bem organizada, com separação entre a aplicação móvel, o *backend* e o módulo de visão computacional.

6.7.3. Requisitos parcialmente atendidos

- **RF04** — O marcador do produto é exibido, mas a sua posição sofre ligeiros desvios angulares, dependendo do ponto e da distância da leitura inicial.
- **RF07** — O modelo de visão identifica os níveis de prateleira, embora limitado a linhas horizontais, apresentando um desempenho superior em imagens reais do supermercado.
- **RFN02** — A aplicação opera através de *Wi-Fi* e rede móvel, mas observa-se um aumento de latência no envio das imagens via `multipart/form-data`.
- **RFN03** — A interface é intuitiva, mas a avaliação de usabilidade baseou-se apenas em perceções subjetivas, sem aplicação de métricas padronizadas, como o *SUS*.

- **RFN04** — O tempo de resposta é satisfatório para a exibição da distância e dos produtos, embora a inicialização da RA (2,1 s) ultrapasse ligeiramente o objetivo de 2 s.

6.7.4. Requisitos não atendidos ou não evidenciados

- **RFN01** — A compatibilidade entre Android e iOS não foi plenamente validada, devido à indisponibilidade de múltiplos dispositivos físicos para teste. No Android, observou-se uma falha no retorno ao ecrã do código *QR* e no método `takeScreenshot`.
- **RFN04** — A atualização da posição com latência inferior a 0,5 s não foi mensurada diretamente; a métrica de exibição (16–57 ms) não equivale à atualização posicional.
- **RFN05** — O processamento de imagem em até 3 s não foi atingido; o tempo médio no iOS foi de aproximadamente 3,9 s, enquanto no Android não foi possível realizar a medição.

6.7.5. Síntese geral

De forma geral, o sistema atendeu de modo satisfatório às funcionalidades principais de leitura de código *QR*, pesquisa de produtos, calibração de prateleiras e visualização em RA, confirmando a viabilidade técnica da proposta. As limitações observadas concentram-se nos aspetos de precisão posicional do cartão, estabilidade multiplataforma e deteção visual, que foram parcialmente cumpridos e representam as principais oportunidades de melhoria.

Mesmo restrito a ambientes controlados, o protótipo apresentou uma integração coerente entre os módulos de RA, *backend* e VC, alcançando um desempenho aceitável dentro das condições de teste. Assim, o sistema pode ser classificado como uma prova de conceito funcional, demonstrando o potencial da abordagem proposta e indicando caminhos claros para a sua evolução rumo à aplicação em ambientes reais.

Capítulo 7.

Conclusão

O trabalho desenvolvido nesta dissertação teve como propósito propor e validar uma abordagem que auxilie o utilizador na localização de produtos em ambientes de retalho, por meio de uma aplicação móvel que combina tecnologias emergentes. Desta forma, a solução proposta foi concebida para demonstrar a viabilidade técnica da integração dessas tecnologias num sistema unificado e funcional, orientado para o contexto real de utilização em supermercados.

A aplicação implementada integra, de forma coerente, um *frontend* desenvolvido em `React Native` e `ViroReact`, um *backend* construído com `FastAPI` e `PostgreSQL`, e um módulo de VC baseado em `PyTorch`. Essa arquitetura modular permitiu o tratamento coordenado das etapas de leitura códigos *QR*, calibração espacial, deteção de prateleiras e exibição de elementos virtuais em RA. Com isto, o sistema possibilitou a demonstração de um fluxo completo, desde a identificação inicial do utilizador dentro de um sistema global de coordenadas até à exibição da prateleira na qual o produto se encontra, validando a integração entre os componentes de software e a consistência dos dados partilhados entre eles.

Os resultados experimentais confirmam a possibilidade da execução da proposta, revelando desempenho adequado para fins de demonstração e investigação. Apesar disto, algumas limitações devem ser reconhecidas. A avaliação ocorreu em ambiente controlado, sem interferências típicas de um espaço de venda a retalho, o que restringe a generalização dos resultados. A amostragem de dispositivos foi limitada, não permitindo avaliar a diversidade de desempenho entre diferentes modelos de *smartphones*. Esses aspetos não comprometem a viabilidade do estudo, mas indicam que o sistema ainda necessita de aperfeiçoamento para alcançar a maturidade exigida pelo mercado. Apesar dessas restrições e de acordo com os requisitos definidos no planeamento da aplicação, foi possível alcançar parcialmente os objetivos delineados, comprovando a viabilidade técnica da integração entre os mecanismos de localização em interiores. Ainda

assim, e apesar da verificação da viabilidade formal da abordagem, limitações no uso de *BLE* para posicionamento e no uso de reconhecimento de padrões para identificação de prateleiras não permitiram obter um protótipo com capacidades operacionais que possam ser validadas por utilizadores reais.

A principal contribuição deste trabalho reside na demonstração prática de uma arquitetura unificada de cliente-servidor que articula RA, deteção visual e posicionamento baseado em marcadores visuais. Essa integração revelou-se uma alternativa promissora às soluções puramente baseadas em BLE, frequentemente sujeitas a flutuações de RSSI e a erros de estimativa de distância. Ainda que a utilização de uma possível solução híbrida não tenha sido descartada, a substituição dessa dependência por um sistema de ancoragem determinístico, suportado por códigos *QR*, permitiu a simplificação da infraestrutura necessária, tornando o sistema mais acessível a contextos comerciais reais, embora exigindo algum trabalho adicional de posicionamento inicial por parte do utilizador.

7.1. Trabalhos Futuros

Como continuidade deste trabalho, propõem-se diversas linhas de evolução. No domínio técnico, destaca-se a necessidade de aprimorar o modelo de visão computacional, de forma a aumentar a robustez face a variações de perspetiva e orientação das prateleiras. O treino com dados adicionais e anotações corretas, a partir do último *checkpoint*, pode ampliar significativamente a precisão do processo de deteção. Outra vertente promissora é a integração de métodos híbridos de localização, combinando BLE e marcadores *QR*, de modo a permitir correções automáticas do alinhamento espacial.

Uma vertente particularmente promissora diz respeito à subaplicação de calibração, que se demonstrou altamente eficiente para o reajuste rápido do posicionamento dos produtos dentro da RA. A continuação do seu desenvolvimento poderá transformá-la numa funcionalidade integrada da aplicação, permitindo que o próprio utilizador contribua para o processo de calibração. Por exemplo, ao concluir uma pesquisa e selecionar a opção "Encontrei", o sistema poderia questionar se o produto estava corretamente posicionado e, caso contrário, solicitar ao utilizador que apontasse o dispositivo para o local exato do item. Com base nesses *feedbacks*, o sistema poderia autoajustar progressivamente o seu modelo de alinhamento, de forma colaborativa, tal como ocorre em aplicações de outros domínios, como o *Waze*, que recolhe informações dos utilizadores para corrigir dinamicamente o mapa e o estado da via. Este tipo de calibragem participativa

tornaria o sistema mais inteligente, adaptativo e capaz de melhorar a sua precisão com o uso contínuo.

Para além dos aspetos técnicos, recomenda-se a realização de estudos de usabilidade com utilizadores reais, utilizando métricas consolidadas como o *System Usability Scale (SUS)*, a fim de avaliar a experiência de utilização e recolher perceções sobre o impacto da visualização em RA na tomada de decisão. Finalmente, seria relevante investigar a escalabilidade do sistema em contextos reais de retalho, explorando questões de manutenção dos marcadores, sincronização de dados e adaptação a diferentes *layouts* de lojas.

Bibliografia

[1] Ronald T. Azuma. «A Survey of Augmented Reality». Em: *Proceedings of the IEEE*. Vol. 86. 7. IEEE, 1997, pp. 1020–1034.

[2] Alistair Cockburn. «Structuring Use Cases with Goals». Em: *CiteSeer* (dez. de 1997). URL: https://www.researchgate.net/publication/2807676.

[3] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. 2nd. Prentice Hall, 1999.

[4] Ruth Malan e Dana Bredemeyer. «Functional Requirements and Use Cases». Em: *Hewlett-Packard Journal* (1999). Draft 07/07/99. URL: http://www.hpl.hp.com/fusion/md_newsletters.html.

[5] Martin Glinz. «Rethinking the Notion of Non-Functional Requirements». Em: *Proc. Third World Congress for Software Quality (3WCSQ 2005)*. Munich, Germany, 2005, pp. 55–64.

[6] Federico Thomas e Lluís Ros. «Revisiting Trilateration for Robot Localization». Em: *IEEE Transactions on Robotics* 21.1 (fev. de 2005), pp. 93–101. DOI: 10.1109/TRO.2004.833793.

[7] Seonghoon Kang e Won Kim. «Minimalist and Intuitive User Interface Design Guidelines for Consumer Electronics Devices». Em: *Journal of Object Technology* 6.3 (mar. de 2007), pp. 39–52. URL: http://www.jot.fm/issues/issue_2007_03/column5.

[8] Christopher D. Manning, Prabhakar Raghavan e Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008.

[9] Björn Jensen, Ralf Kruse e Birgit Wendholt. «Application of Indoor Navigation Technologies Under Practical Conditions». Em: *Proceedings of the 6th Workshop on Positioning, Navigation and Communication (WPNC'09)*. Hamburg, Germany: IEEE, 2009, pp. 267–272. DOI: 10.1109/WPNC.2009.4912008.

[10] Mark Everingham et al. «The PASCAL Visual Object Classes (VOC) Challenge». Em: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338.

[11] Gerhard Reitmayr et al. «Simultaneous Localization and Mapping for Augmented Reality». Em: *Proceedings of the 2010 International Symposium on Ubiquitous Virtual Reality (ISUVR)*. Graz, Austria: IEEE, 2010, pp. 1–8. DOI: 10.1109/ISUVR.2010.12.

[12] Pavel Smutný. «Mobile development tools and cross-platform solutions». Em: *Proceedings of the 2012 IEEE International Conference on Mobile Computing, Applications, and Services*. Ostrava, Czech Republic: IEEE, 2012, pp. 651–654.

[13] Raul G. S. Cardoso e outros. «Realidade Aumentada na Educação». Em: *Computer on the Beach 2014 - Artigos Completos*. Belém: Editora SBC, 2014, pp. 330–338.

[14] M. Naveenkumar e A. Vadivel. «OpenCV for Computer Vision Applications». Em: *Proceedings of National Conference on Big Data and Cloud Computing (NCBDC'15)*. Trichy, Tamilnadu, mar. de 2015.

[15] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: https://www.deeplearningbook.org.

[16] Gaurav Bhorkar. «A Survey of Augmented Reality Navigation». Em: (2017). arXiv: 1708.05006 [cs.HC]. URL: https://arxiv.org/abs/1708.05006.

[17] gre. *react-native-view-shot: Snapshot a React Native view and save it to an image*. Versão: v1.8.0 (4 Mar 2017). Disponível em npm. 2017. URL: https://github.com/gre/react-native-view-shot.

[18] Jun-Tae Lee et al. «Semantic Line Detection and Its Applications». Em: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE. 2017, pp. 3250–3258.

[19] J. Gu et al. «Recent advances in convolutional neural networks». Em: *Pattern Recognition* 77 (2018), pp. 354–377. DOI: 10.1016/j.patcog.2017.10.013.

[20] Konstantinos Spiliotopoulos, Maria Rigou e Spiros Sirmakessis. «A Comparative Study of Skeuomorphic and Flat Design from a UX Perspective». Em: *Multimodal Technologies and Interaction* 2.31 (2018), pp. 1–21. DOI: 10.3390/mti2020031.

[21] DeInfo UEPG. *Realidade Aumentada*. Página web. Conceitua realidade aumentada e apresenta exemplos como Pokémon GO e aplicativos de filtros em redes sociais. 2019. URL: https://deinfo.uepg.br/~alunoso/2019/AEP/REALIDADE_AUMENTADA/RealidadeAumentada.html.

[22] Anca Morar et al. «A Comprehensive Survey of Indoor Localization Methods Based on Computer Vision». Em: *Sensors* 20.9 (2020). ISSN: 1424-8220. DOI: 10.3390/s20092641. URL: https://www.mdpi.com/1424-8220/20/9/2641.

[23] Petros Spachos. «Indoor Localization Based on BLE Beacons for Smart Museums». Em: *IEEE Systems Journal* 14.3 (2020), pp. 3484–3492.

[24] Wei Ma, Shuai Zhang e Jincai Huang. «Mobile augmented reality based indoor map for improving geo-visualization». Em: *PeerJ Computer Science* 7 (2021), e704. DOI: 10.7717/peerj-cs.704. URL: https://peerj.com/articles/cs-704.pdf.

[25] Ashly Martin et al. «Indoor Navigation using Augmented Reality». en. Em: vol. 8. 26. Mar. de 2021, p. 168718. DOI: 10.4108/eai.17-2-2021.168718. URL: http://eudl.eu/doi/10.4108/eai.17-2-2021.168718 (acedido em 20/10/2025).

[26] Naser El-Sheimy e You Li. «Indoor navigation: state of the art and future trends». Em: *Satellite Navigation* 2.7 (2021). DOI: 10.1186/s43020-021-00041-3. URL: https://doi.org/10.1186/s43020-021-00041-3.

[27] Kai Zhao et al. «Deep Hough Transform for Semantic Line Detection». Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2021). DOI: 10.1109/TPAMI.2021.3077129.

[28] Kuhelee Chandel, Julia Åhlén e Stefan Seipel. «Augmented Reality and Indoor Positioning in Context of Smart Industry: A Review». Em: *Management and Production Engineering Review* 13.4 (2022), pp. 72–87. DOI: 10.24425/mper.2022.142396.

[29] António Larguesa. *Portugueses passam quase quatro dias por ano no supermercado.* Acessado em: 11 de julho, 2024. 2022. URL: https://eco.sapo.pt/2022/02/03/portugueses-passam-quase-quatro-dias-por-ano-no-supermercado/.

[30] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 2nd. Springer, 2022. URL: https://szeliski.org/Book/.

[31] Bingjie Xu et al. «ARShopping: In-Store Shopping Decision Support Through Augmented Reality and Immersive Visualization». Em: *arXiv preprint* (2022). arXiv: 2207.07643 [cs.HC]. URL: https://arxiv.org/abs/2207.07.

[32] William Danielsson. «Evaluation of React Native in Comparison to Native Android Development». Acesso em: 2 de setembro de 2024. Tese de mestrado. Linköping University, 2023. URL: http://www.ep.liu.se/.

[33] Dotintent. *react-native-ble-plx: React Native Bluetooth Low Energy library*. Versão 3.5.0 (publicada há cerca de 7 meses). :contentReference[oaicite:1]index=1. 2023. URL: https://github.com/dotintent/react-native-ble-plx.

[34] Junjie Hu et al. «Deep Depth Completion From Extremely Sparse Data: A Survey». Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.7 (2023), pp. 8244–8264. DOI: 10.1109/TPAMI.2022.3229090.

[35] Muhammad Mehroz. *Choosing Between Managed and Bare Workflow in Expo: A Comprehensive Difference*. 2023. URL: https://medium.com/@mehro_z/choosing-between-managed-and-bare-workflow-in-expo-a-comprehensive-difference-7086305f1480 (acedido em 29/10/2023).

[36] Hugo Sefrian Peinado e Dayana Bastos Costa. «Aplicações de BIM, RPA e Visão Computacional de Forma Integrada para o Planejamento e Controle da Segurança em Canteiros de Obras». Em: *13º Simpósio Brasileiro de Gestão e Economia da Construção e 4º Simpósio Brasileiro de Tecnologia da Informação e Comunicação na Construção*. Aracaju, SE, nov. de 2023.

[37] M S Ramesh et al. «Indoor Navigation using Augmented Reality for Mobile Application». Em: *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*. 2023, pp. 1049–1052. DOI: 10.1109/ICICCS56967.2023.10142362.

[38] Retail Town. *Visual communications in retail: engaging customers through effective signage and graphics*. Online blog post on RetailTown. Discusses o impacto da sinalização eficaz no tráfego de clientes e nas vendas, bem como nas práticas visuais em loja. Nov. de 2023. URL: https://retail.town/visual%E2%80%91merchandising%E2%80%91store%E2%80%91mgt/effective%E2%80%91in%E2%80%91store%E2%80%91signage%E2%80%91strategies/.

[39] Ashraf Saad Shewail, Neven Abdelaziz Elsayed e Hala Helmy Zayed. «Survey of indoor tracking systems using augmented reality». Em: *IAES International Journal of Artificial Intelligence (IJ-AI)* 12.1 (mar. de 2023), p. 402. ISSN: 2252-8938, 2089-4872. DOI: 10.11591/ijai.v12.i1.pp402-414. URL: https://ijai.iaescore.com/index.php/IJAI/article/view/21636 (acedido em 20/10/2025).

[40] Shopify. *@shopify/react-native-skia: High-performance 2D Graphics for React Native using Skia*. Versão: v2.x (última: v2.3.8 em 29 Out 2025). 2023. URL: https://github.com/Shopify/react-native-skia.

[41] Miguel Fernandes da Silva. «Migração de Aplicação Móvel Híbrida para Desenvolvimento Nativo». Versão Pública. Tese de mestrado. Universidade de Lisboa, Faculdade de Ciências, 2023.

[42] Juliana Erika de Carvalho Teixeira Yassitepe Thiago Teixeira Santos. «Fenotipagem de plantas em larga escala: Tecnologias da Informação e Comunicação e suas relações com a agricultura». Em: *Tecnologias da Informação e Comunicação na Agricultura*. Editora, 2023, pp. 88–95.

[43] Roger Dooley. *The Most Overlooked Time-Waster For Retail Shoppers*. Forbes. "In large stores like supermarkets and big-box retailers, finding a particular product can be the most time-consuming and time-wasting part of a shopping trip." 2024.

[44] Prasun Gahlot, Komal Suryavanshi e Aradhana Gandhi. «Revolutionizing decision-making: How retailers can harness the potential of avatars in augmented reality to enrich the customer experience». Em: *Cogent Business Management* 11.1 (2024), p. 2432542. DOI: 10.1080/23311975.2024.2432542. URL: https://www.tandfonline.com/doi/pdf/10.1080/23311975.2024.2432542.

[45] REACT NATIVE. *React Native. Aprenda uma vez, escreva em qualquer lugar*. Acesso em: 12 de agosto de 2024. 2024. URL: https://reactnative.dev/.

[46] Rocco Pietrini et al. «Shelf Management: A deep learning-based system for shelf visual monitoring». Em: *Expert Systems with Applications* 255 (2024), p. 124635. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2024.124635. URL: https://www.sciencedirect.com/science/article/pii/S0957417424015021.

[47] Expo Team. *Expo Documentation*. Acesso em: 2 de setembro de 2024. 2024. URL: https://docs.expo.dev/.

[48] ViroReact Team. *ViroReact Documentation*. Acesso em: 2 de setembro de 2024. 2024. URL: https://viro-community.readme.io/docs/overview.

[49] Lionel Sujay Vailshery. *Cross-platform mobile frameworks used by developers worldwide 2019-2023*. https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. Acessado em: 03 de outubro de 2024. 2024.

[50] Ryo Yonetani, Jun Baba e Yasutaka Furukawa. «RetailOpt: Opt-In, Easy-to-Deploy Trajectory Estimation from Smartphone Motion Data and Retail Facility Information». Em: *Proceedings of the 2024 ACM International Symposium on Wearable Computers*. ACM,

out. de 2024, pp. 125–132. DOI: 10.1145/3675095.3676623. URL: http://dx.doi.org/10.1145/3675095.3676623.

[51] Axios Contributors. *Axios – Promise based HTTP client for browser Node.js*. acessado em Setembro de 2025. 2025. URL: https://axios-http.com/docs/intro.

[52] Mário Bruno Cruz, Francisco Rebelo e Jorge Cruz Pinto. «Eye-Tracking Advancements in Architecture: A Review of Recent Studies». Em: *Buildings* 15.19 (2025). ISSN: 2075-5309. DOI: 10.3390/buildings15193496. URL: https://www.mdpi.com/2075-5309/15/19/3496.

[53] Docker Inc. *Docker Documentation*. Versão online; consultado em 23 out. 2025. Docker Inc. 2025. URL: https://docs.docker.com/.

[54] Randoncorp. *O que é realidade aumentada e quais são os seus principais tipos?* Blog post. Define RA, tipos, funcionamento com sensores e exemplos de aplicação nas indústrias e no varejo. Fev. de 2025. URL: https://www.randoncorp.com/pt/blog/realidade%E2%80%91aumentada/.

[55] TanStack Contributors. *TanStack Query – React Documentation*. Acedido em 25 out. 2025. 2025. URL: https://tanstack.com/query/latest/docs/react/overview.

[56] The PostgreSQL Global Development Group. *PostgreSQL Documentation*. 17.5. Versão online; consultado em 23 out. 2025. The PostgreSQL Global Development Group. 2025. URL: https://www.postgresql.org/docs/current/.

[57] Foodtech Family. *Using BLE technology when working with beacon in React Native*. Acesso em: 03 de janeiro de 2025. URL: https://medium.com/foodtech-family/using-ble-technology-when-working-with-beacon-in-react-native-327864b9ee5f.

[58] Katia Poloni. *O que são redes neurais convolucionais?* Acesso em: 03 de janeiro de 2025. URL: https://medium.com/itau-data/redes-neurais-convolucionais-2206a089c715.

[59] Amazon Web Service. *O que é uma rede neural?* Acesso em: 01 de janeiro de 2025. URL: https://aws.amazon.com/pt/what-is/neural-network/.

Apêndice A.

Teste de desempenho com ViroReact

O objetivo do teste em questão é avaliar a viabilidade da biblioteca `ViroReact` para aplicações de RA em dispositivos móveis, com foco em desempenho (quadros por segundo) e consumo de memória, a fim de verificar se atende aos requisitos do sistema proposto.

Tabela A.1: Ambiente de Teste.

Item	Descrição
Dispositivo	iPhone 15 Pro
Sistema Operacional	iOS 17
Computador	MacBook Pro 2021, macOS 13
React Native	v0.73.6
Expo SDK	50.0.0
ViroReact	v2.41.6

Foram realizados dois testes com a biblioteca `ViroReact`, sendo eles a deteção de um plano horizontal com o posicionamento de um modelo 3D utilizando o componente `ViroARPlane` e o posicionamento manual de um modelo 3D em coordenadas fixas, utilizando o componente `ViroARScene`. Durante os testes, foram monitorados a taxa de quadros por segundo e o uso de memória RAM.

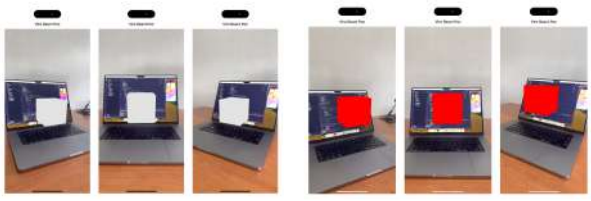

Figura A.1: Posição de objeto 3D: plano (cinza) vs. coordenadas (vermelho)

Durante os testes, a taxa de quadros por segundo permaneceu estável entre 29,99 e 30,00, demonstrando execução fluida dos cenários visuais em RA. O consumo de memória RAM foi

de aproximadamente 635,52 MB, representando 7,75% da capacidade total do dispositivo (8 GB), um valor considerado aceitável para dispositivos modernos. No entanto, ressalta-se que essa demanda pode impactar o desempenho em aparelhos com menor capacidade de memória. Em ambos os cenários testados o comportamento foi consistente, sem atrasos perceptíveis na renderização ou falhas na visualização dos objetos virtuais.

Apêndice B.

Requisitos e Casos de Uso do Sistema

B.1. Requisitos Funcionais e Requisitos Não Funcionais

A seguir, são apresentados os oito requisitos funcionais identificados para o sistema. Eles foram definidos a partir da análise das funcionalidades previstas e descrevem as principais ações que a aplicação deverá executar para atender às necessidades do utilizador.

- **RF01:** O utilizador deve digitalizar um código *QR* para ativar o sistema.
- **RF02:** O utilizador deve poder reiniciar a busca a qualquer momento.
- **RF03:** Deve ser exibida a distância em tempo real até o produto selecionado.
- **RF04:** Deve ser exibido um marcador sobre o produto selecionado.
- **RF05:** O utilizador deve receber notificações por vibração e mensagens visuais.
- **RF06:** O sistema deve notificar o utilizador quando se aproxima do produto.
- **RF07:** O sistema deve identificar e destacar a prateleira correta.
- **RF08:** O utilizador deve poder escolher um produto através da pesquisa ou da listagem.

Na sequência, são apresentados os requisitos não funcionais definidos para o sistema. Eles estabelecem restrições e características de qualidade que a aplicação deverá cumprir, garantindo seu desempenho, compatibilidade e usabilidade.

- **RNF01:** Compatibilidade com Android >= 9.0 e iOS >= 12.0
- **RNF02:** Operar com diferentes velocidades de conexão (Wi-Fi, 4G/5G).
- **RNF03:** Interface intuitiva para utilizadores com pouca experiência.

- **RNF04:** Tempo de resposta inferior a 1s para distância e 2s para RA.
- **RNF05:** Atualização da posição com latência inferior a 0,5s.
- **RNF06:** Processamento de imagem em até 3 segundos para identificar prateleira.
- **RNF07:** Código modular e organizado, com foco em manutenção.

B.2. Casos de Uso

A seguir, são apresentados os principais casos de uso identificados para o sistema. Todos consideram como ator principal o utilizador da aplicação. Os casos descrevem, de forma resumida, as interações previstas entre o utilizador e o aplicativo, detalhando as etapas necessárias para a execução das funcionalidades centrais.

CU01 – Habilitar a RA

- **Objetivo:** O utilizador deve digitalizar o código *QR* para iniciar o uso da aplicação e realizar pesquisas de produtos.
- **Pré-requisitos:** O utilizador deve ter autorizado as permissões de câmara e *Bluetooth* na primeira interação com a aplicação.
- **Descrição:** O utilizador abre a aplicação; o sistema exibe uma mensagem solicitando o escaneamento do código *QR*; o utilizador digitaliza o código *QR* na entrada do supermercado; o sistema habilita a RA, utilizando as coordenadas do código *QR* como ponto inicial.

CU02 – Pesquisar Produto

- **Objetivo:** O utilizador deseja buscar um produto específico no supermercado utilizando a aplicação.
- **Pré-requisitos:** O utilizador deve ter digitalizado o código *QR* e ativado a RA.
- **Descrição:** O sistema exibe o modal de produtos; o utilizador digita o nome do produto; o sistema realiza a pesquisa na base de dados e exibe a lista de produtos encontrados; o utilizador seleciona o produto desejado na lista e inicia a navegação.

CU03 – Navegar até o Produto

- **Objetivo:** O utilizador deseja ser guiado até o produto a partir de sua localização atual no supermercado.
- **Pré-requisitos:** O utilizador deve selecionar um produto.
- **Descrição:** O utilizador seleciona o produto para iniciar a navegação; o sistema insere o modelo na RA indicando a localização do produto; o sistema exibe a distância entre o utilizador e o produto; o utilizador caminha em direção ao modelo utilizando a câmara e a distância como guia; o sistema notifica o utilizador quando ele estiver a cerca de dois metros do produto.

CU04 – Alterar Produto Durante a Navegação

- **Objetivo:** O utilizador decide trocar de produto durante a navegação, após ter iniciado o processo de busca.
- **Pré-requisitos:** O utilizador deve ter digitalizado o código *QR* e ativado a RA.
- **Descrição:** O utilizador toca no botão para cancelar a busca atual; o sistema exibe o modal de produtos; o utilizador digita o nome do produto; o sistema realiza a pesquisa na base de dados e exibe a lista de produtos encontrados; o utilizador seleciona o produto desejado na lista e inicia a navegação.

CU05 – Encontrar o Produto

- **Objetivo:** O utilizador deseja localizar o produto na prateleira ao chegar no local indicado.
- **Pré-requisitos:** O utilizador deve estar a cerca de dois metros de distância do local onde o produto está localizado.
- **Descrição:** O sistema solicita que o utilizador aponte a câmara para a prateleira, de modo que todas fiquem visíveis; o sistema exibe um botão para iniciar a análise; o utilizador toca no botão para iniciar a análise; o sistema captura uma imagem das prateleiras; o sistema utiliza VC para detetar a quantidade de prateleiras; o sistema verifica nos detalhes do produto em qual prateleira ele está disposto; o sistema exibe a imagem capturada com a prateleira correta destacada; o utilizador finaliza a navegação.

Apêndice C.

Código-fonte complementar

C.1. Serviço `getAllProducts` para integração com a API

O serviço `getAllProducts` **C.1** encapsula a chamada `HTTP GET` ao `/products` do `backend`, recebendo opcionalmente o termo `search` e construindo a `query string` via `URLSearchParams`. A função devolve uma lista tipada de `IProduct`, abstraindo do componente de interface os detalhes de comunicação (*endpoint*, serialização `JSON` e verificação do parâmetro de pesquisa). Esta separação favorece a coesão do código de apresentação e facilita a evolução do contrato com a API sem impactar os consumidores do serviço.

Em termos de utilização, o serviço é agnóstico à camada de *state management* e pode ser orquestrado por `hooks` de *data fetching* (e.g., `useQuery`), permitindo *caching*, *refetch* e invalidação. Deste modo, a pesquisa de produtos torna-se reativa ao termo introduzido pelo utilizador e mantém uma interface estável para os componentes que a consomem.

```
1  import { IProduct } from "@/models/Product";
2
3  export const getAllProducts = async (search: string) => {
4      let url = "http://192.168.1.69:8000/products?"
5
6      if (search) {
7          url += new URLSearchParams({
8              search,
9          }).toString()
10     }
11
12     const response = await fetch(url);
13     const produtos: IProduct[] = await response.json();
14     return produtos;
```

```
};
```

Código C.1: Serviço de integração com a API de produtos

C.2. Utilização do *hook* useDebounce

O *hook* personalizado `useDebounce` é utilizado para controlar a frequência das requisições ao serviço `getAllProducts`, evitando múltiplas chamadas consecutivas durante a digitação do termo de pesquisa. A função aplica um intervalo de espera de 500ms sobre o valor introduzido pelo utilizador, devolvendo apenas o último valor estabilizado após esse período. Este comportamento reduz significativamente o número de pedidos ao `backend`, melhorando a eficiência do sistema e proporcionando uma experiência mais fluida no campo de pesquisa.

Na sequência, o valor retornado por `useDebounce` é utilizado como dependência no *hook* `useQuery`, que executa a função `getAllProducts`. Assim, cada nova pesquisa é enviada apenas quando o utilizador interrompe a digitação por um breve momento, o que permite manter as respostas atualizadas sem sobrecarregar o servidor, conforme ilustrado no Código **C.2**. Esta integração entre os *hooks* de temporização e de consulta reflete uma prática recomendada em aplicações *React*, promovendo responsividade e economia de recursos.

```
const [search, setSearch] = useState<string>("")

const debouncedSearch = useDebounce({
    debounceTime: 500,
    value: search
})

const { data } = useQuery({
    queryKey: ["produtcs", debouncedSearch],
    queryFn: () => getAllProducts(debouncedSearch)
})
```

Código C.2: Utilização do serviço getAllProducts pelo componente SearchProductModal

C.2.1. Função `handleAnalyze`

A função `handleAnalyze` **C.3** é responsável por executar o fluxo de análise visual da prateleira, articulando as operações de captura de imagem, envio ao `backend` e preparação dos dados para renderização no ambiente de RA. O processo inicia-se com a obtenção de uma imagem do

contexto atual através do método `_takeScreenshot` do componente de realidade aumentada, sendo o ficheiro posteriormente tratado conforme o sistema operativo (`iOS` ou `Android`). Em seguida, a função recolhe as dimensões originais da imagem e solicita ao serviço `detectShelfRows` a deteção automática das linhas correspondentes às prateleiras.

Após o retorno da previsão, os dados da imagem são convertidos para objetos compatíveis com o *framework* `Skia`, que permite a renderização eficiente e a sobreposição dos elementos gráficos detetados. A função também implementa mecanismos de controlo de estado (`startLoading` e `stopLoading`) e de tratamento de exceções, garantindo robustez durante o processo de análise. Deste modo, essa função centraliza a lógica de integração entre a captura em RA, a inferência remota e a visualização, constituindo um elo fundamental entre o dispositivo e o modelo de visão por computador.

```
const handleAnalyze = useCallback(async () => {
  try {
    startLoading("Analizando imagem");

    const { url } = await arRef.current?._takeScreenshot("shelf-ar-image", false
        );
    const imageUri = Platform.OS === "ios" ? `file://${url}` : url;
    setScreenshot(imageUri);

    Image.getSize(imageUri, (w, h) => setImageDimensions({ width: w, height: h
        }));

    const res = await detectShelfRows(imageUri); // chama o backend
    const imageData = await Skia.Data.fromURI(imageUri);
    const cvImage = Skia.Image.MakeImageFromEncoded(imageData);

    setCanvasImage(cvImage);
    setLines(res.lines);
  } catch (error) {
    setScreenshot("");
    setLines([]);
  } finally {
    stopLoading();
  }
}, [startLoading, stopLoading]);
```

Código C.3: Captura da imagem, chamada ao serviço e preparação para renderização

C.2.2. Serviço `detectShelfRows`

O serviço `detectShelfRows` realiza a comunicação entre o *frontend* e o `backend`, enviando a imagem capturada para o *endpoint* `/predict` através de uma requisição `POST` no formato `multipart/form-data`. Após o processamento pelo modelo de visão computacional, o serviço devolve as coordenadas das linhas correspondentes às prateleiras detetadas, que são utilizadas na renderização em RA. Esta função encapsula a lógica de envio e receção dos dados, simplificando o consumo da API e mantendo o código da interface limpo e modular.

```
export const detectShelfRows = async (imageUri) => {
  const formData = new FormData();
  formData.append("file", {
    uri: imageUri,
    name: "image.png",
    type: "image/png",
  });

  const response = await fetch("http://192.168.1.69:8000/predict", {
    method: "POST",
    body: formData,
    headers: { "Content-Type": "multipart/form-data" },
  });

  return await response.json();
};
```

Código C.4: Serviço de upload da imagem e consumo do endpoint /predict

C.2.3. Componente `AnalyzedShelfView`

O componente `AnalyzedShelfView` **C.5** é responsável por exibir a imagem analisada e sobrepor graficamente a região correspondente à prateleira onde se encontra o produto selecionado. A partir das linhas detetadas pelo serviço `detectShelfRows` **C.4**, o componente calcula dinamicamente a área delimitadora da prateleira utilizando funções de posicionamento e dimensões do ecrã. Este cálculo é ajustado de acordo com o nível indicado pelo produto selecionado, garantindo que a área realçada em RA corresponda à localização física correta na imagem.

Para a renderização, o componente utiliza o *framework* `Skia`, que permite desenhar a imagem base e a sobreposição em tempo real com elevada performance. A camada verde semitransparente indica visualmente a zona onde o produto deve estar posicionado, oferecendo ao utilizador

um *feedback* imediato e intuitivo. Além disso, o componente mantém uma interface reativa, atualizando a visualização sempre que o utilizador altera o produto selecionado ou quando novas linhas são detetadas, o que reforça a integração entre a inferência do modelo e a experiência interativa em RA.

```
export const AnalyzedShelfView = ({ canvasImage, imageDimensions, lines,
    onPressFound }) => {
  const { selectedProduct } = useSelectedProductContext();

  const squareSizes = useMemo(() => {
    if (!selectedProduct || !lines.length) return null;

    const reversed = lines.reverse();
    const arr = reversed.slice(selectedProduct.prateleira - 1, selectedProduct.
        prateleira + 1);

    if (!arr.length) return null; // nao encontrou o nivel informado

    if (arr.length < 2) { // ultimo nivel
      const [l1] = arr;
      return { x: l1[0], y: l1[1], width: l1[2] - l1[0], height: Math.abs(
          Dimensions.get("screen").height - l1[1]) };
    }

    const [l1, l2] = arr;
    return {
      x: l1[0],
      y: Math.min(l1[1], l2[1]),
      width: l1[2] - l1[0],
      height: Math.abs(l2[1] - l1[1]),
    };
  }, [lines, selectedProduct]);

  return (
    <Canvas style={{ width: "100%", height: "100%", position: "absolute", top: 0
        }}>
      <Image
        image={canvasImage}
        x={0}
        y={0}
        width={imageDimensions.width / scale}
```

```
        height={imageDimensions.height / scale}
        fit={"contain"}
      />
      {squareSizes && (
        <Rect
          x={squareSizes.x / scale}
          y={squareSizes.y / scale}
          width={squareSizes.width / scale}
          height={squareSizes.height / scale}
          color="rgba(0, 255, 0, 0.4)"
        />
      )}
    </Canvas>
  );
};
```

Código C.5: Renderização da imagem e sobreposição da região da prateleira detectada

A Hybrid Mobile System for Indoor Navigation and Product Localization in Supermarkets

Leonardo Bazán Picinin
ESTiG
Instituto Politécnico de Bragança
Bragança, Portugal
a61440@alunos.ipb.pt

Luisa Jorge 0000-0002-0623-7282
CeDRI
Instituto Politécnico de Bragança
Bragança, Portugal
ljorge@ipb.pt

Paulo Melo 0000-0002-9166-0615
CeBER
Faculty of Economics, University of Coimbra
Coimbra, Portugal
pmelo@fe.uc.pt

***Abstract*—This paper presents the design and evaluation of an integrated mobile system to support product localization and indoor navigation within supermarkets. Using augmented reality for real-time visual guidance, computer vision for shelf-level recognition, and QR-code-based markers for accurate spatial initialization, the system bridges the gap between virtual and real store environments. The solution utilizes a React Native frontend and a backend built with FastAPI, PostgreSQL, and PyTorch. Experimental results validate its technical viability, demonstrating world alignment and shelf detection, while addressing BLE signal instability and image processing latency. The proposed architecture proposes a foundation for retail navigation systems, aimed at delivering scalable, accurate, and user-friendly in-store experiences.**

***Index Terms*—Augmented Reality (AR), Indoor Navigation, Computer Vision, Positioning**

I. Introduction

Recent studies indicate that locating products in supermarkets often represents the most time-consuming phase of the shopping process, accounting for a significant portion of the total time spent in stores [1]. Furthermore, improvements in signage and spatial guidance can reduce the need for staff interaction by up to 15% and potentially increase sales by 20%, reinforcing the relevance of technological solutions to enhance user experience in these contexts [2].

Recent advances in Augmented Reality (AR), Computer Vision (CV) and Indoor Positioning Systems (IPS) have enabled new forms of spatial interaction in enclosed environments [3], [4]. Although such technologies have been successfully applied in areas like education and security, their use in supermarkets remains limited. This scenario creates an opportunity to explore how the combined use of these technologies can improve in-store navigation, reduce search time and enhance the overall shopping experience.

This work proposes an integrated mobile system for indoor navigation and product localization in supermarkets. The solution combines AR for real-time visual guidance, CV for shelf-level recognition, and visual markers based on QR codes for accurate position initialization. The system integrates a React Native frontend and a backend built with FastAPI, PostgreSQL, and PyTorch, ensuring modularity and seamless communication between product data and visual inference services.

Recent studies have investigated the use of AR and IPS to improve user navigation and decision-making in retail environments. Works such as RetailOpt [5] and ARShopping [6] integrate sensor data, maps, and data fusion for in-store tracking and product search, while surveys [7]–[9] highlight the potential of AR to enhance spatial orientation and contextual guidance. Within retail, immersive visualization and avatar-based interaction have been shown to enrich customer experience [10], and AR-based indoor maps improve geo-visualization in complex layouts [11]. BLE remains a low-cost option for localization but is highly unstable under interference [9], [12], whereas QR-based initialization offers a deterministic and infrastructure-light alternative [7].

For visual scene understanding, semantic line detection and depth-based learning techniques deliver strong results in controlled settings but face generalization issues under diverse perspectives [13], [14]. Cross-platform frameworks such as React Native support agile prototyping of AR applications [8]. Building on these advances, the present work integrates AR (via ViroReact), BLE sensing, and computer vision into a unified mobile application for supermarket navigation, addressing BLE instability through QR-based alignment and enabling visual shelf-level confirmation within a consistent spatial reference.

The remainder of this paper is structured as follows. Section II reviews related technologies and approaches. Section III details the system architecture and methods. Section IV presents results and evaluation. Finally, Section V concludes with future directions.

II. Background and Related Technologies

Mobile applications can be developed either natively or through cross-platform frameworks. Native development typically provides superior performance and direct access to system resources but requires maintaining separate code bases for Android and iOS [15]. Cross-platform frameworks, such

This work was supported by Plano de Recuperacao e Resiliencia, within the project "SustainOlive", PRR-C05-i03-I-000187. The authors are also grateful to the Foundation for Science and Technology (FCT, Portugal) for financial support through national funds FCT/MCTES (PIDDAC): CeDRI, UIDB/05757/2020 (DOI:10.54499/UIDB/05757/2020), SusTEC, LA/P/0007/2020 (DOI:10.54499/LA/P/0007/2020) CeBER (UIDB/05037/2025), and INESC Coimbra (UIDB/00308/2020).

as React Native, enable faster development by reusing a single codebase while preserving near-native performance. In this project, React Native was selected due to its maturity, community support and compatibility with augmented reality libraries. Combined with Expo, it facilitated rapid testing and development while ensuring fluid interaction across devices.

Augmented Reality enhances user perception by overlaying digital content onto the physical world, creating an intuitive means of spatial guidance [16]. In retail environments, AR allows users to visualize navigation cues, arrows, or indicators directly within their camera view, which can improve orientation and reduce search time. Native AR frameworks such as ARKit and ARCore provide advanced mapping and tracking features but are limited to their respective platforms. To achieve cross-platform compatibility, the ViroReact library was adopted, offering high-level components for rendering, scene management and sensor tracking integrated with React Native.

Computer Vision techniques based on Convolutional Neural Networks (CNNs) have demonstrated strong results in detecting structural patterns within images, including lines, shelves and object boundaries [17].

In this context, semantic line detection was employed to identify horizontal divisions in supermarket shelves, enabling the system to confirm whether the user reached the correct product level. The approach was inspired by existing methods for retail shelf monitoring and line-based scene understanding, adapted to inference restrictions from mobile devices.

Indoor navigation traditionally relies on radio-based methods such as Bluetooth Low Energy (BLE) beacons, Wi-Fi fingerprinting and Ultra-Wideband (UWB). BLE is widely available and inexpensive but exhibits high signal fluctuation due to environmental interference, leading to unstable distance estimation. Wi-Fi fingerprinting requires frequent recalibration and depends on pre-mapped signal databases, while UWB provides high precision at the cost of dedicated hardware [18].

In contrast, visual marker–based localization using QR codes enables deterministic pose estimation without additional infrastructure. The present work builds upon these findings, integrating QR-based positioning with AR visualization and computer vision to deliver a stable, low-cost and hardware-independent indoor navigation system for supermarkets.

III. System Design and Methodology

A. System Overview

The system is structured around a mobile application that communicates with a backend responsible for product data management and visual inference services. Through this interaction, the application retrieves product locations, receives navigation instructions, and submits images for computer vision analysis. Within the application, an indoor navigation module guides users to the correct product position using a combination of Bluetooth beacon signals and AR visualization.

At the highest level, the system workflow can be summarized in three main stages. First, the user initializes the navigation process by scanning QR codes that establish spatial

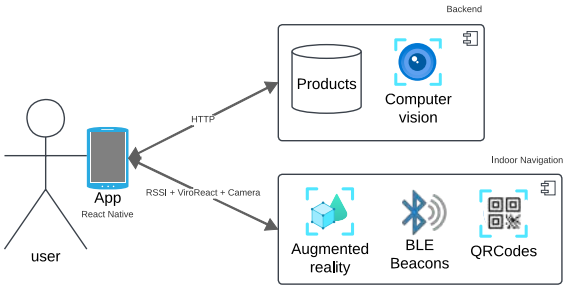

Fig. 1. System architecture components overview.

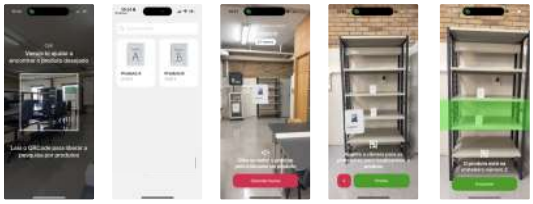

Fig. 2. User interface screenshots illustrating the navigation workflow.

alignment between the virtual and real environments. Once the virtual world is calibrated, the user searches for a specific product within the mobile interface, which retrieves its location in global coordinates. Finally, as the user approaches the designated shelf, the CV module analyzes the camera feed to confirm that the product is positioned on the correct shelf level, visually reinforcing the guidance provided by the AR elements. Fig. 1 illustrates the system architecture and the interaction between its components.

B. Implementation Architecture

The frontend was implemented in React Native, ensuring compatibility with both Android and iOS devices through a single shared codebase. The mobile application provides three main capabilities: product search, in-store navigation and visual confirmation of the target shelf. It is also responsible for managing user interaction, local state control and communication with the backend through RESTful APIs. The interface was designed to offer a smooth and responsive experience, providing a foundation for the AR and CV modules presented in the following subsections. Fig. 2 illustrates the screens of the mobile system.

The backend was developed with FastAPI and organizes two main responsibilities, product data and CV inference, running alongside a PostgreSQL database that stores product records with their spatial attributes x, y, z and the corresponding shelf level, which enables direct lookups for navigation. The vision service exposes a predict endpoint, which receives an uploaded image and returns the detected semantic lines (shelves) as coordinates, leaving it to the client to determine which line corresponds to the correct shelf level based on the returned points. The product API and the vision endpoint are consolidated under the same FastAPI server, ensuring simple

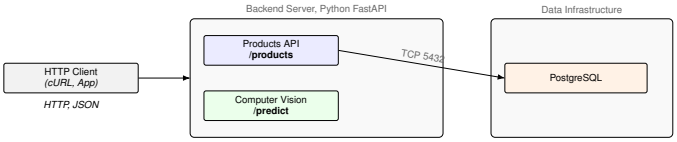

Fig. 3. Backend architecture, overview of modules and dependencies.

integration and low-latency responses. Fig. 3 illustrates the backend architecture.

C. Global Coordinate System

The navigation framework was designed around a global coordinate system shared by all components of the application. Each product stored in the database is associated with fixed spatial coordinates (x,y,z) and its corresponding shelf level, defining its precise position within the store. The main objective of this configuration is to ensure that both the products and the user exist in a common spatial reference, allowing the AR environment to remain consistent across sessions and devices. Once the AR is properly initialized in this coordinate system, subsequent alignment operations are limited to small corrections required to compensate for sensor drift or minor tracking errors.

A central challenge in this model lies in determining the user's initial position and orientation at the moment the application starts. Accurate initialization is essential to correctly align the virtual camera with the physical environment and to guarantee that the visual indicators point to the correct product locations. Two strategies were explored to address this problem: (i) the use of BLE beacons combined with inertial sensors to estimate the initial position through trilateration and motion equations, and (ii) the use of QR codes encoding fixed coordinates to provide deterministic initialization. Both approaches share the same conceptual goal: placing the user in the same global coordinate system as the products.

D. Indoor Navigation Alignment with BLE Beacons

The initial proposal for indoor navigation considered the use of BLE beacons to estimate the user's position within the global coordinate system. Each beacon transmits its identifier and signal strength (RSSI), allowing the mobile device to infer its approximate distance from the beacon. The RSSI data were obtained through the React Native BLE PLX library. By combining the distances obtained from at least three beacons, the user's position can be determined through trilateration, resulting in coordinates that serve as the starting point for initializing the AR environment. Once the user's position is established, orientation and motion are continuously updated using data from the device's inertial sensors.

The accelerometer provides linear acceleration values (Vx, Vy, Vz), while the gyroscope measures angular velocity (Wx, Wy, Wz). In the proposed implementation, these readings are accessed through the Expo Sensors library. The combination of these two sensors enables continuous updates of both position and orientation. The motion of the device can be modeled according to Equation 1, which expresses the update of the position vector as a function of the previous state, elapsed time, and the rotation applied to the linear acceleration:

$$\begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ z_{t-1} \end{bmatrix} + \Delta t \cdot R_t \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{1}$$

The orientation of the device is described by Equation 2, which defines how the rotation matrix is incrementally updated using the angular velocity vector provided by the gyroscope:

$$R_t = R_{t-1} \cdot \exp\left(\frac{\Delta t}{2}[\omega_t]_\times\right) \tag{2}$$

Together, these equations describe the theoretical movement model that would enable continuous tracking of the user's pose, combining BLE-based trilateration for initial positioning with inertial sensor data for refinement during navigation.

E. Indoor Navigation Alignment with QR Codes

An alternative method was implemented to initialize the user's position within the global coordinate system using QR codes as visual markers. Each code encodes a string containing a validation key, positional coordinates and rotational coordinates following the pattern "produtox-camera:x,y,z;x,y,z". The first triplet represents the physical position of the marker in the store, while the second corresponds to the device rotation at the moment of scanning. This information allows the application to determine both the position and orientation required to initialize the AR environment at a fixed point within the global reference frame.

To ensure accurate data acquisition, the application requires the user to align the QR code with a square displayed on the screen before confirming the scan. This procedure guarantees that the camera and the code are positioned on the same visual plane, allowing the system to estimate the distance between the device and the marker based on the apparent size of the QR code in the captured image. Once the data are decoded, the AR scene is initialized at the specified coordinates, and the estimated distance is used to position the virtual camera correctly in space.

F. World Alignment System

The world alignment system is responsible for synchronizing the AR environment with the physical space defined by the global coordinate system. Once the user's initial position and orientation are determined, either by BLE trilateration or by decoding the QR code data, the application applies a transformation that aligns the virtual world reference frame with the corresponding real-world position.

The transformation between the virtual and real coordinate systems is defined by the following equations, which describe the rotation and translation required to align the AR scene with the global reference frame. Equation 3 defines the translation vector between the two spaces:

$$T = P_{real} - R \cdot P_{virtual} \tag{3}$$

Equation 4 describes the application of the transformation matrix that combines rotation R and translation T to align any point in the virtual environment to its real-world counterpart:

$$P_{aligned} = R \cdot P_{virtual} + T \tag{4}$$

Finally, Equation 5 defines the final transformation matrix that encapsulates both rotation and translation components:

$$M = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \tag{5}$$

These transformations are applied at initialization by the useWorldAlignment hook. To ensure consistent alignment across app lifecycle and motion events, two guard mechanisms are used. The App State Guard detects state changes, for example when the app goes to background or returns to foreground, and requests a new QR code reading so that AR can be initialized again and the scene aligned again. The useMotionGuard monitors device motion, and when it detects excessive movement during a sensitive phase, it also restarts the QR code reading process to reinitialize AR and realign the scene. In both cases, the system explicitly returns to the QR initialization step to guarantee a clean, deterministic world alignment.

G. Computer Vision Module

The CV module was implemented to recognize the structure of supermarket shelves and visually confirm that the user has reached the correct product location. The model used in this work was made available by a previous study on semantic line detection using the Deep Hough Transform approach [19]. This model detects horizontal structural features in images through a learnable Hough transformation. For training, the system used an existing dataset of annotated shelf images provided by a recent work on deep learning–based shelf management [20]. These resources were adapted to meet the specific requirements of the proposed navigation system.

Since the original dataset annotations were normalized as percentages, it was necessary to adapt them to the format required by the Deep Hough Transform model, which expects absolute pixel coordinates. An auxiliary script was created to convert normalized coordinates into absolute values based on the image dimensions. Each generated label follows the structure `image.jpg,1 x1 y1 x2 y2|`, where the first number indicates the number of lines in the image and each subsequent group of four values defines the endpoints of a detected line. When multiple lines are present, their coordinates are concatenated sequentially. The Algorithm 1 represents a pseudocode that illustrates the conversion process.

After the preprocessing stage, the model was trained in PyTorch using the same architecture as in [19], with adjustments to maintain compatibility with updated PyTorch versions. Hyperparameters such as learning rate and batch size were empirically optimized to balance convergence speed and generalization. Table I represents the model results on the evaluation dataset

Algorithm 1 Conversion of normalized labels to absolute coordinates

for each line in dataset **do**
  $(image\_name, y\_values[]) \leftarrow parse(line)$
  $(img\_height, img\_width) \leftarrow getImageSize(image\_name)$
  $num\_lines \leftarrow length(y\_values)$
  $label \leftarrow image\_name + "," + num\_lines$
  for each y in y_values **do**
    $y\_abs \leftarrow int(y \times img\_height)$
    $label \leftarrow label +$ "0 y_abs img_width y_abs"
  end for
  save(label)
end for

TABLE I
EVALUATION METRICS OF THE TRAINED COMPUTER VISION MODEL

Metric	Precision	Recall	F-measure	F@0.95
Value	0.97	0.95	0.96	0.91

The trained model was integrated into the backend through a dedicated inference service built with FastAPI. The predict endpoint receives an image sent by the mobile application and returns the detected semantic lines as a JSON object containing the total number of lines and a list with the coordinates of their endpoints. Each entry in the list corresponds to a horizontal line identified in the image, representing a shelf level. The mobile client interprets these coordinates to show the correct shelf rendering the augmented reality indicators. As shown in Fig. 4, the model successfully detects the horizontal shelf lines in the evaluation dataset.

IV. Results and Discussion

A. Experimental Setup

All experiments were conducted in a controlled laboratory environment designed to evaluate the stability of indoor positioning, the accuracy of world alignment and the performance of the computer vision module. This setup provided consistent lighting conditions and minimal signal interference, ensuring reproducible measurements across different test iterations. Table II illustrates the devices and software versions used during the experiments.

The backend server, implemented with FastAPI, was executed locally to reduce latency and ensure stable communi-

Fig. 4. Results of the trained model on the evaluation dataset showing detected horizontal shelf lines highlighted in blue.

TABLE II
COMPONENTS CONFIGURATION USED IN THE EXPERIMENTS

Component	Version / Model
iPhone 15 Pro	iOS 18.6.1
Poco X5 Pro	Android 14
Expo SDK	54
ViroReact	2.43.5
react-native-ble-plx	3.5.0
react-native-vision-camera	4.7.2

Fig. 5. RSSI-based distance estimation variability compared to the real distances of 1 m, 2 m, and 3 m.

cation between modules during the tests. The experimental procedure consisted of three main stages. First, BLE beacons were placed at fixed coordinates to collect signal samples for distance estimation. Second, QR codes were used to initialize the augmented reality environment and validate the world alignment process. Finally, shelf images were captured using the mobile application to evaluate the behavior of the trained computer vision model under controlled conditions. Each test was repeated multiple times to verify result consistency and assess average response times.

B. BLE Signal Instability Analysis

To evaluate the feasibility of using BLE beacons for indoor positioning, signal samples were collected at fixed distances of 1 m, 2 m, and 3 m from the receiver under controlled laboratory conditions. Each measurement consisted of 100 RSSI readings obtained through the react-native-ble-plx library. Both the smartphone and the beacon were positioned on a flat surface, maintaining a considerable distance from physical obstacles or sources of electromagnetic interference. This setup aimed to eliminate external variables and obtain representative readings of the signal behavior in an ideal scenario. Figure 5 illustrates the variation in the estimated distances for each test condition, highlighting the significant fluctuations observed in the RSSI measurements relative to the real distances.

The received signal strength values were then converted into distance estimates using a log-distance path loss model. Equation (6) expresses this relationship where d represents the estimated distance in meters, M is the measured RSSI value at 1 meter (set to -64 dBm), and n is the path-loss exponent (set to 2.5).

$$d = 10^{\frac{(M-RSSI)}{10n}} \tag{6}$$

Even under controlled conditions, the results revealed substantial variations in the estimated distances, with errors frequently exceeding 1.5 meters. This instability was not associated with external obstacles but rather with fluctuations inherent to the BLE signal itself, influenced by factors such as transmission power variation, multipath propagation, and radiofrequency noise. Additionally, it is likely that part of this variability was amplified by the specific beacons used in the experiments, which may have introduced inconsistencies due to their limited transmission quality or hardware calibration.

In a navigation scenario within narrow supermarket aisles, such fluctuations would be sufficient to compromise the overall user experience. Errors in distance estimation directly affect the spatial alignment of the augmented reality environment, potentially placing virtual indicators over incorrect shelves or positioning the user in an adjacent corridor. These findings reinforce that, although BLE remains a practical and cost-effective solution for coarse proximity detection, its current level of precision and stability is insufficient to serve as an absolute positional reference without additional signal filtering, sensor fusion, or improved beacon hardware.

TABLE III
INITIALIZATION TIME OF THE AR SCENE ON IOS AND ANDROID

Platform	Measured times (ms)	Mean (ms)
iOS	2084, 2026, 2216, 2231, 2199	2151
Android	2364, 1766, 2215, 2082, 1774	2040

TABLE IV
END-TO-END VISION PIPELINE TIME

Platform	Measured times (ms)	Mean (ms)
iOS	3845, 4012, 3976, 3822, 3905	3912
Android	—	—

C. System Performance and Platform Limitations

The performance evaluation aimed to analyze the system behavior under different execution conditions, comparing iOS and Android platforms in terms of response time, rendering fluency and AR stability. These measurements were conducted using instrumented code segments that automatically recorded execution times in critical parts of the workflow, such as AR initialization and image analysis.

The initialization time of the AR scene was measured as the interval between the QR code reading and the stabilization of environment tracking. Five consecutive executions were performed on each platform and the results showed an average initialization time of approximately 2.1 seconds for both platforms. Table III illustrates the recorded initialization times and their respective averages.

The second process analyzed was the inference time of the computer vision pipeline. In this study, the measurement starts when the user presses the button that corresponds to analyze image in the mobile application. This interval includes the screenshot capture performed by ViroReact, the local preparation of the image, the upload to the backend and the server-side processing until the client receives the response with the detected lines. Five consecutive requests were executed using images captured in the laboratory. Table IV summarizes the average times obtained.

Fig. 6. Misalignment caused by micro variations in device angle during initialization.

During the tests performed on the Android platform, the takeScreenshot function of the ViroReact API exhibited recurring failures, preventing the complete execution of the image capture workflow. Alternative solutions to the native function were tested, such as the react-native-view-shot and @shopify/react-native-skia libraries, but both showed insufficient performance: the first failed to correctly capture the rendered frame, while the second introduced excessive latency. The inability to capture the image prevented the quantitative analysis of this stage on Android, although the visual behavior observed during partial executions suggested that the total processing time would be similar to that obtained on iOS. To confirm this hypothesis, it would be necessary to repeat the tests on another Android device, preferably with different hardware specifications and system versions, in order to rule out the possibility that the issue is related to a device-specific configuration.

D. Evaluation of World Alignment Accuracy

The world alignment accuracy was analyzed to verify how consistently the virtual scene remains registered with the physical space as the distance from the QR marker increases. At short ranges (up to three meters), alignment remained stable and virtual indicators were correctly projected onto the target shelves. However, increasing distance or slight camera tilts produced visible offsets, as illustrated in Fig. 6.

This sensitivity arises from the distance estimation method, which relies on the apparent area of the QR code in the camera frame. Although this introduces minor depth errors under oblique viewing angles, the alignment remained visually acceptable for typical short-range use. Future work may address this limitation through fiducial marker pose estimation or visual–inertial fusion.

E. Computer Vision Performance

The vision module was evaluated in two scenarios: laboratory shelf images and unseen images captured in a real supermarket. This comparison assessed robustness to illumination, texture, and perspective changes, as well as the model's capacity to generalize. The inference was measured in two modes: direct execution via the script and through the backend endpoint. Average times were approximately 210 ms (ranging from 198 to 230 ms) for script approach and around 135 ms (ranging from 74 to 502 ms) for endpoint approach, the latter benefiting from a preloaded model and GPU memory persistence.

Fig. 7. Comparison between detections on laboratory shelf images (top) and real supermarket shelves (bottom).

With five laboratory and five supermarket images, the results revealed contrasting behaviors. In laboratory conditions, the detections were often inconsistent, whereas real-store images produced more stable outputs. The detected lines remained strictly horizontal regardless of camera tilt, which was a direct consequence of the label normalization procedure described earlier. Figure 7 illustrates representative detections under both conditions, highlighting the model's tendency to maintain horizontal alignment even when image geometry varies.

This process biased the model toward learning horizontal structural features, which, while effective for front-facing and well-aligned captures similar to the training data, exposed a significant limitation in scenarios where the camera is positioned at oblique angles or irregular viewpoints. As a result, the model often fails to detect shelf lines that deviate from a horizontal orientation, reducing its robustness in real-world conditions. This behavior is directly linked to the composition of the training dataset, which predominantly contained front-facing images, suggesting that expanding the dataset with more diverse perspectives and retraining the model would substantially improve its generalization and detection capabilities.

F. Discussion

The experiments confirmed the functional viability of the proposed system while revealing important technical limitations. The overall stability of the AR alignment and the performance of the vision module demonstrate that the system is capable of supporting indoor product localization in real conditions. However, the achieved precision is still below the level required for fully autonomous operation, mainly due to residual alignment drift and latency introduced by image processing and network communication.

The BLE positioning approach was not integrated into the final version because of the high signal variability observed during the early tests. Despite this limitation, BLE remains promising for future iterations since its proper calibration and fusion with inertial sensor data could allow automatic

initialization without QR code scanning. This improvement would make the navigation process more seamless and user-friendly.

Regarding the computer vision component, the conversion of normalized labels to absolute coordinates constrained the model to horizontal detections, which limited generalization to non-frontal views. Expanding the dataset with varied perspectives and refining annotations could enhance inference accuracy and robustness across diverse shelf geometries.

It is important to note that the evaluation reported in this paper was purely technical and conducted in a controlled laboratory setting. No user study was carried out to measure task completion time, error rate, or perceived usability (e.g., SUS), which means that the practical value of the system for end-users still needs to be verified in a realistic supermarket scenario.

In summary, the obtained results validate the feasibility of integrating these technologies into a unified mobile solution for indoor navigation. The proposed architecture establishes a foundation for future work focused on improving positional precision, visual inference, and system automation, paving the way for a fully self-contained hybrid AR-based navigation system.

V. Conclusion and Future Work

This work presented the design and evaluation of a hybrid mobile system that combines Augmented Reality, Computer Vision, and visual markers for indoor product localization in supermarkets. The results demonstrated that the proposed architecture is technically viable and capable of guiding users to product locations with acceptable stability and latency. Experimental findings validated the performance of the world-alignment mechanism and confirmed the effectiveness of the vision module in detecting shelf structures across different environments.

Despite these achievements, the system still faces practical limitations that prevent its fully autonomous operation. The initialization currently depends on visual markers, and the accuracy of world alignment decreases when the QR code is captured from long distances or tilted angles. Additionally, BLE-based positioning was excluded due to signal instability, and image processing latency remains a bottleneck in mobile execution.

Future developments will focus on addressing the main limitations identified during the experiments and enhancing the overall capabilities of the proposed system. One of the key priorities is to expand and diversify the training dataset with images captured from different angles, distances, and perspectives. This will allow the retraining of the computer vision model to improve its robustness and generalization, enabling more accurate detection of shelf structures under a wider variety of real-world conditions. In parallel, future iterations will explore the integration of BLE positioning with inertial sensor fusion to support automatic, marker-free initialization of the AR environment, which would reduce user interaction and significantly improve system usability.

In addition to these technical directions, a crucial next step will be to conduct an in-situ user study with shoppers and store staff, aimed at quantifying task completion times, navigation errors, and perceived usability, and at comparing QR-based initialization with marker-less or beacon-based alternatives. Such evaluation will provide valuable insights into the system's practical effectiveness and user acceptance in real supermarket environments.

Finally, further efforts will be directed toward optimizing the inference pipeline, including improvements in image handling, communication, and on-device processing, to reduce latency and enhance real-time performance. Together, these developments aim to evolve the current prototype into a more autonomous, scalable, and fully synchronized indoor navigation solution capable of operating reliably in real supermarket environments.

References

[1] R. Dooley, "The Most Overlooked Time-Waster For Retail Shoppers," Forbes Blog, Feb. 2024. [Online]. Available: https://www.forbes.com/sites/rogerdooley/2024/02/29/the-most-overlooked-time-waster-for-retail-shoppers/

[2] Retail Town, "Visual communications in retail: engaging customers through effective signage and graphics," Online blog post on Retail Town, November 2023. [Online]. Available: https://retail.town/visual-merchandising-store-mgt/effective-in-store-signage-strategies/

[3] A. Morar, A. Moldoveanu, I. Mocanu, F. Moldoveanu, I. E. Radoi, V. Asavei, A. Gradinaru, and A. Butean, "A comprehensive survey of indoor localization methods based on computer vision," *Sensors*, vol. 20, no. 9, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/9/2641

[4] K. Chandel, J. Åhlén, and S. Seipel, "Augmented reality and indoor positioning in context of smart industry: A review," *Management and Production Engineering Review*, vol. 13, no. 4, p. 72–87, 2022.

[5] R. Yonetani, J. Baba, and Y. Furukawa, "RetailOpt: Opt-in, easy-to-deploy trajectory estimation from smartphone motion data and retail facility information," in *Proceedings of the 2024 ACM International Symposium on Wearable Computers*. ACM, Oct. 2024, p. 125–132. [Online]. Available: http://dx.doi.org/10.1145/3675095.3676623

[6] B. Xu, S. Guo, E. Koh, J. Hoffswell, R. Rossi, and F. Du, "ARShopping: In-store shopping decision support through augmented reality and immersive visualization," *arXiv preprint*, 2022. [Online]. Available: https://arxiv.org/abs/2207.07

[7] G. Bhorkar, "A survey of augmented reality navigation," 2017. [Online]. Available: https://arxiv.org/abs/1708.05006

[8] M. S. Ramesh, J. Naveena Ramesh Vardhini, S. Murugan, and J. Albert Mayan, "Indoor navigation using augmented reality for mobile application," in *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2023, pp. 1049–1052.

[9] A. S. Shewail, N. A. Elsayed, and H. H. Zayed, "Survey of indoor tracking systems using augmented reality," *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 12, no. 1, p. 402, Mar. 2023. [Online]. Available: https://ijai.iaescore.com/index.php/IJAI/article/view/21636

[10] P. Gahlot, K. Suryavanshi, and A. Gandhi, "Revolutionizing decision-making: How retailers can harness the potential of avatars in augmented reality to enrich the customer experience," *Cogent Business Management*, vol. 11, no. 1, p. 2432542, 2024. [Online]. Available: https://www.tandfonline.com/doi/pdf/10.1080/23311975.2024.2432542

[11] W. Ma, S. Zhang, and J. Huang, "Mobile augmented reality based indoor map for improving geo-visualization," *PeerJ Computer Science*, vol. 7, p. e704, 2021. [Online]. Available: https://peerj.com/articles/cs-704.pdf

[12] A. Martin, J. Cheriyan, J. Ganesh, J. Sebastian, and J. V, "Indoor Navigation using Augmented Reality," vol. 8, no. 26, Mar. 2021, p. 168718. [Online]. Available: http://eudl.eu/doi/10.4108/eai.17-2-2021.168718

[13] J. Hu, C. Bao, M. Ozay, C. Fan, Q. Gao, H. Liu, and T. L. Lam, "Deep depth completion from extremely sparse data: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 7, pp. 8244–8264, 2023.

[14] M. B. Cruz, F. Rebelo, and J. Cruz Pinto, “Eye-tracking advancements in architecture: A review of recent studies,” *Buildings*, vol. 15, no. 19, 2025. [Online]. Available: https://www.mdpi.com/2075-5309/15/19/3496

[15] W. Danielsson, “Evaluation of react native in comparison to native android development,” Master’s thesis, Linköping University, 2023, acesso em: 2 de setembro de 2024. [Online]. Available: http://www.ep.liu.se/

[16] R. T. Azuma, “A survey of augmented reality,” in *Proceedings of the IEEE*, vol. 86, no. 7. IEEE, 1997, pp. 1020–1034.

[17] M. Naveenkumar and A. Vadivel, “Opencv for computer vision applications,” in *Proceedings of National Conference on Big Data and Cloud Computing (NCBDC’15)*, Trichy, Tamilnadu, March 2015.

[18] B. Jensen, R. Kruse, and B. Wendholt, “Application of indoor navigation technologies under practical conditions,” in *Proceedings of the 6th Workshop on Positioning, Navigation and Communication (WPNC’09)*. Hamburg, Germany: IEEE, 2009, pp. 267–272.

[19] K. Zhao, Q. Han, C. bin Zhang, J. Xu, and M. ming Cheng, “Deep hough transform for semantic line detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.

[20] R. Pietrini, M. Paolanti, A. Mancini, E. Frontoni, and P. Zingaretti, “Shelf management: A deep learning-based system for shelf visual monitoring,” *Expert Systems with Applications*, vol. 255, p. 124635, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417424015021