Departamento de Ciências e Tecnologias da Informação

# Aperfeiçoamento da Integração de Sensores de Dispositivos Móveis na Domótica

João David Guerreiro de Brito

Dissertação submetida como requisito parcial para obtenção do grau de
Mestre em Engenharia de Telecomunicações e Informática

Orientador:
Professor Doutor Rui Neto Marinheiro
ISCTE-IUL

setembro, 2019

## Agradecimentos

À minha mulher e aos meus 5 filhos agradeço pelo sacrifício familiar a que estiveram sujeitos e, mesmo assim, de nunca terem deixado de me apoiar e incentivar.

Aos meus pais agradeço o facto de nunca terem desistido de acreditar em mim e pelo apoio logístico que me foi dado.

À empresa Mais Eficaz e, em especial ao Eng.º Adriano Quesado, agradeço o incrível apoio financeiro para a realização do mestrado e ao meu colega e gestor de equipa, Eng.º António Claro, a compreensão e flexibilidade de horário que me foi possibilitada para frequentar as aulas.

Ao meu orientador, Professor Rui Neto Marinheiro, agradeço a sua orientação em todas as etapas desta dissertação.

Ao meu colega e amigo, Pedro Gomes, agradeço pelo seu encorajamento e motivação, que ao longo destes anos de mestrado sempre me foi prestando.

Ao Instituto de Telecomunicações por me acolher para a realização deste trabalho.

A todos os que enumerei e, a todos os outros que por lapso me esqueci de os citar, os meus sinceros agradecimentos.

## Resumo

A Internet das Coisas (IoT) veio para nos facilitar a vida. Na visão de um mundo da IoT, todas as coisas estarão ligadas à Internet, ou seja, estarão ubiquamente acessíveis e poder-se-á interagir com as mesmas de múltiplas formas, como seja enviando comandos para estas desempenharem uma determinada ação ou para se receber notificações.

Atualmente existem soluções que utilizam *smartphones* para controlarem dispositivos IoT em casas inteligentes, mas são soluções incompletas. As mais comuns utilizam aplicações proprietárias onde se interage apenas através de botões e não exploram todas as potencialidades dos muitos sensores disponíveis. Imagine-se se fosse possível utilizar na sua plenitude os sensores do seu smartphone, como por exemplo a câmara, o sensor de luminosidade, de proximidade, giroscópio, etc., para gerar ações nos dispositivos IoT sejam estes luzes, televisões ou até uma torradeira. Poder-se-ia assim melhorar a experiência de utilização dos *smartphones* na domótica.

Neste contexto, este estudo propõe uma arquitetura para uma melhor integração dos *smartphones*, considerando os seus múltiplos sensores disponíveis, numa casa inteligente.

Com a solução aqui apresentada, foi desenvolvido um protótipo que permite testar parte da arquitetura, em que utilizadores podem navegar com um *smartphone* pela habitação e interagir com os diversos aparelhos, com os quais se vão cruzando, usando de uma forma mais integrada os sensores disponíveis. Por exemplo, a câmara do smartphone é usada para o reconhecimento dos objetos. Após o seu reconhecimento, o utilizador pode interagir com estes inclinando ou balançando o dispositivo.

Esta solução de baixo custo, que utiliza ferramentas *open-source*, foi testada e validada, com recurso a testes funcionais e testes de desempenho. Provou-se que é uma solução eficaz, cumprindo os objetivos propostos, fiável, e apresenta um desempenho que permite uma boa usabilidade.

Esta solução é inovadora, pois maximiza a utilização dos sensores disponíveis num *smartphone*, e aponta para um caminho onde os dispositivos moveis estarão integrados de uma forma plena em soluções de domótica.

***Palavras-chave***: casas inteligentes, domótica, sensores, smartphones, internet das coisas

## Abstract

The Internet of Things (IoT), in which every device is connected to the Internet, will make our everyday lives easier by enabling interaction with them. This interaction will take many forms, perhaps the most common being commanding devices to carry out certain actions or receive notifications.

Most current solutions that use smartphones to control IoT devices in smart homes are incomplete and commonly use proprietary applications which only interact with buttons and do not exploit the full potential of the many sensors available.

Being able to fully exploit smartphone sensors, such as the camera, brightness sensor, proximity sensor, gyroscope etc., to command actions on IoT devices such as lights, televisions or even toasters will greatly improve the experience of using smartphones in home automation.

In this context, this work proposes an architecture for the integration of smartphones and their sensors with the smart home.

To prove the solution and its architecture a prototype was developed enabling users to navigate around a house using a smartphone and interacting with a range of IoT devices using the smartphone sensors. An example of this interaction is object recognition; the smartphone camera is used to recognize an object, once recognized the user is able to interact with the device by tilting or shaking the smartphone.

This low-cost solution, using open source tools, has been tested and validated using functional and performance testing. It has been proven to be an effective solution, meeting the proposed objectives, shown to be reliable, and has performance permitting good usability.

This solution is innovative as it maximizes the use of sensors available on a smartphone, and points to a future in which mobile devices will be fully integrated with home automation solutions.

***Keywords***: smart homes, home automation, sensors, smartphones, internet of things

# Índice

# Índice de Figuras

# Índice de Tabelas

# Lista de Abreviaturas e Siglas

| | |
|---|---|
| AP | Access Point |
| API | Application Programming Interface |
| BLOB | Binary Large OBject |
| CPU | Core Processing Unit |
| DoG | Difference of Gaussian |
| ESH | Eclipse Smart Home |
| GSM | Global System for Mobile |
| GPS | Global Positioning System |
| GUI | Graphical User Interface |
| HTTP | Hypertext Transfer Protocol |
| HUE | Tecnologia ZigBee proprietária da Philips |
| IDE | Integrated Development Environment |
| IoT | Internet of Things / Internet das Coisas |
| JMX | Java Management Extensions |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| KNX | Konnex – Protocolo de comunicação |
| LAN | Local Area Network |
| LoRA | Long Range – Protocolo de comunicação |
| MQTT | Message Queuing Telemetry Transport |
| NFC | Near-Field Communication |
| NTP | Network Time Protocol |
| QrCode | Quick Response Code |

| | |
|---|---|
| REST | Representational State Transfer |
| TCP | Transmission Control Protocol |
| UML | Unified Modeling language |
| UUID | Universally Inique IDentifier |
| WAN | Wide Area Network |
| WIFI | Wireless network |
| WLAN | Wireless Local Area Network |
| X10 | Protocolo de comunicação |

# Capítulo 1. **Introdução**

## 1.1 Motivação e Enquadramento do Tema

Nos anos 70 e 80 do século XX, a tendência da tecnologia das tecnologias de informação (TI) centrava-se essencialmente na centralização em servidores de *mainframe*, onde todo o processamento era feito e os terminais eram meramente interfaces com os utilizadores finais. Na década seguinte, com o aparecimento de computadores pessoais mais poderosos em termos de processamento de informação, assistimos a uma descentralização progressiva onde se verificava cada vez mais terminais com inteligência e poder de processamento. No início do século XXI, a Internet cresceu de forma massiva e com ela, a possibilidade de interligar computadores por todo o mundo como uma única rede global. Pouco depois apareceram os primeiros *smartphones*, levando não só ao uso cada vez ainda mais ubíquo da Internet mas também a uma nova era onde novamente o processamento centralizado se torna relevante, mas desta vez na forma de *cloud computing* [1].

De certa forma, e em paralelo com a evolução dos computadores, a domótica, também chamada de casas inteligentes, também evoluiu significativamente. Esta no início era mais fechada, ao contrário do que se passa atualmente. Ou seja, com o aparecimento de novos protocolos de comunicação como o KNX [2], ZigBee [3], Z-Wave [3], LoRA [4], Wifi, os cenários de utilização da domótica são mais abrangentes. Atualmente, a adoção ou a adaptação dos protocolos usados na Internet tornam as tecnologias para a domótioca ainda mais abertas e promissoras. Atualmente, é uma das principais áreas de crescimento na área das Tecnologias de Informação.

A Internet tornou-se tão indispensável que o aparecimento do conceito de IoT era quase inevitável e possivelmente pode contribuir no ano 2025 com 11 triliões de dólares na economia mundial [5]. É um tema com elevado crescimento nos últimos anos e a tendência é que continue a liderar as tendências do mercado [6].

A IoT está em toda a parte e veio para ficar. Abrange grande parte das áreas, como vestuário, saúde e bem-estar, edifícios e casas, cidades e quintas, energia, entre muitos outros. Juntamente com toda essa dinâmica evolutiva, também houve uma grande preocupação com a segurança, bem como com a privacidade de bens e pessoas [7].

Atualmente existem vários dispositivos inteligentes aplicados à domótica. Apesar de todas as vantagens oferecidas por estes, o que se constata é que cada dispositivo requer a sua própria aplicação, que funciona num ecossistema fechado, para que se possa interagir e comandar o mesmo remotamente. Isto não se revela nada prático, especialmente quando o número de dispositivos de marcas distintas tende a aumentar. É também aqui que as referidas plataformas IoT podem ajudar, centralizando o controle numa única aplicação, e criando um único ponto onde seja possível mais facilmente gerir a segurança.

Outra questão relevante é o facto de os dispositivos móveis estarem subaproveitados. Os mesmos estão repletos de sensores que podem e deviam ser aproveitados no âmbito da domótica. Por exemplo, deveria ser possível usar a câmara para reconhecimento de objetos, o sensor de proximidade como interruptor *ON/OFF*, o giroscópio para aumentar ou diminuir a intensidade de uma luz.

No contexto deste problema surgem então algumas questões de investigação e objetivos que se pretendem atingir com esta dissertação.

## 1.2 Questões e Objetivos de Investigação

Dentro do enquadramento exposto, procura-se responder às seguintes questões:

- É possível integrar os dispositivos móveis como sensores em sistemas de domótica de uma forma efetiva?
- É possível interagir com vários dispositivos IoT apenas com uma aplicação?

No presente estudo, pretendemos alargar o conceito referido nos pontos anteriores introduzindo dispositivos móveis e usando-os como sensores ou atuadores inteligentes. Em vez de apenas serem utilizados para controlar remotamente dispositivos como controlos remotos, podem ser uma parte integrante do sistema e ser usados de forma integrada. Por exemplo, um smartphone pode ser usado como sensor / atuador de uma lâmpada ou persiana, simplesmente apontando a câmara do smartphone para um determinado dispositivo e, uma vez reconhecida a imagem, aciona-se o respetivo mecanismo de controle.

Assim sendo, um dos principais objetivos é o de melhorar e demonstrar a integração efetiva de dispositivos móveis em casas inteligentes, considerando o dispositivo móvel como um sensor e parte integrante da solução e não apenas como um controlo remoto

onde as ordens são dadas aos atuadores ou apenas para visualizar informações recolhidas pelos sensores. Ou seja, pretende-se usar os sensores de um smartphone (câmara, giroscópio, acelerômetro, GPS, microfone, impressões digitais, sensor de luminosidade e proximidade, etc.) e integrá-los de forma ativa num sistema de casa inteligente, conforme a Figura 1 o ilustra.

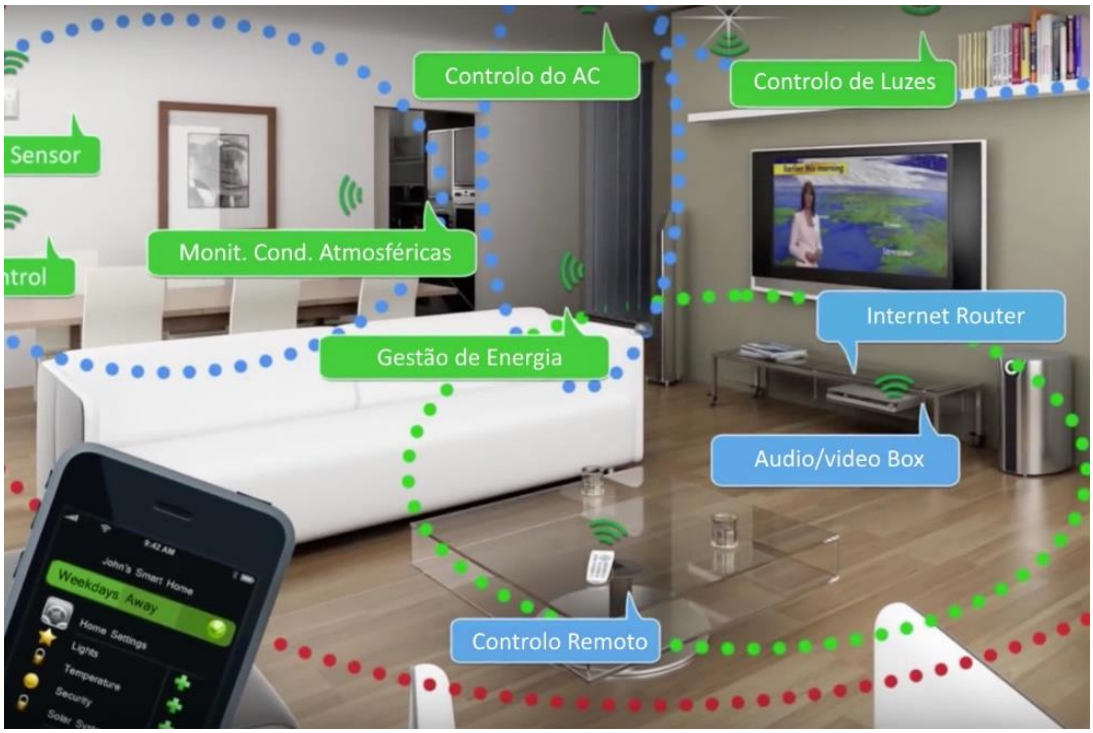


*Figura 1 – Integração de dispositivos móveis em casas inteligentes*[1]

Outro objetivo do presente estudo é o de melhorar a integração dos diferentes dispositivos de automação em casas inteligentes, que ainda são controladas na sua maioria por aplicações independentes e proprietárias. Existem alguns dispositivos inteligentes, como frigoríficos, aspiradores, televisões, lâmpadas, entre outros, mas cada um requer sua própria aplicação para que possamos interagir com estes. Aproveitando a contribuição das plataformas IoT, pretende-se, neste estudo, centralizar todos os dispositivos num único ponto de acesso e controlo.

Para atingir os objetivos pretendidos adotou-se a abordagem descrita na secção seguinte.

[1] Figura adaptada de https://www.pocket-lint.com/smart-home/news/129857-what-is-zigbee-and-why-is-it-important-for-your-smart-home

## 1.3 Abordagem Metodológica

O método de pesquisa adotado nesta dissertação é um processo iterativo e consiste em seis atividades distintas, conforme a Figura 2.

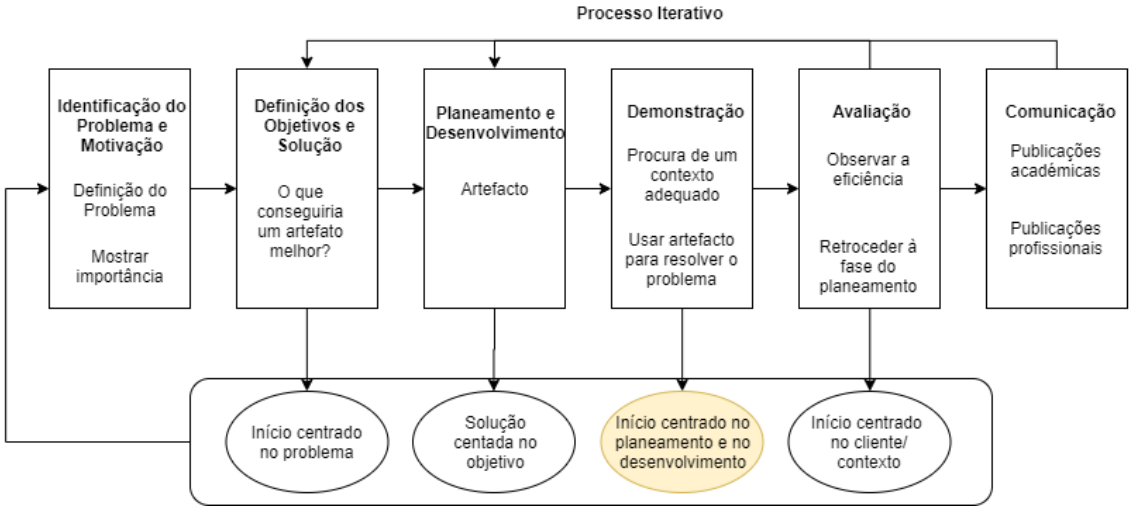


*Figura 2 – Modelo de abordagem metodológica*

Estas atividades podem sumariamente ser descritas como:

1. **Identificação do problema** - Definir uma pesquisa específica e justificar o valor de uma solução. Neste caso, a pouca ou praticamente nenhuma integração dos dispositivos móveis em sistemas de casas inteligentes;
2. **Definir objetivos para uma solução** - Definir os objetivos de uma solução a partir da definição do problema e conhecimento do que é possível e viável. Ou seja, no nosso caso, consiste na integração efetiva dos dispositivos móveis em sistemas de casas inteligentes, bem como a utilização de uma única aplicação unificadora para controlar diversos objetos.
3. **Planeamento e desenvolvimento** – Determinar a funcionalidade do artefacto bem como a sua arquitetura e posteriormente a criação real do artefacto, isto é, planear e desenvolver uma possível solução para que sejam cumpridos os objetivos descritos no ponto anterior.
4. **Demonstração** - Demonstrar o uso do artefacto para resolver uma ou mais instâncias do problema.

5. **Avaliação** - Observar e medir quão bem o artefacto suporta uma solução para o problema. À semelhança do ponto anterior, fazer testes que provem que a solução permite atingir os objetivos propostos.
6. **Comunicação** - Comunicar o problema e a sua importância. Comentar os resultados e retirar ilações dos mesmos.

O ponto de entrada da pesquisa mais ajustado ao presente caso de estudo é a abordagem centrada no planeamento e desenvolvimento, isto é, começa com a atividade 3 e é adequada para uma situação em que o artefacto já existe, mas não se encontra totalmente desenvolvido como uma solução para o domínio do problema explícito em que é suposto ser utilizado. Em qualquer atividade de processo individual, se houver espaço para melhorias ou algo não correr conforme o esperado, o processo retorna à fase inicial e reinicia a partir daí [8].

Como exemplo, será escolhida uma plataforma IoT, e se por qualquer motivo não atender a todas as necessidades, devemos voltar e analisar novamente as plataformas selecionadas (ou até mesmo outras) e escolher outra e prosseguir com essa escolha até os objetivos serem alcançados. Este é um processo iterativo de escolhas até que a opção mais ajustada seja escolhida.

## 1.4 Resumo da Solução e Principais Contribuições

A solução para os problemas descritos consiste em integrar os sensores dos *smartphones* numa casa inteligente, de forma efetiva e numa única aplicação. Isto é, através de uma única aplicação móvel (Android, iOS, etc.) aproveitar a informação recolhida pelos sensores do smartphone e enviá-la para uma plataforma IoT de forma a esta interpretar e redirecionar esses comandos para os dispositivos que lhes estão associados.

Este processo é inovador, pois atualmente cada dispositivo tem a sua própria aplicação e esta limita-se a funcionar como controlo remoto, ou seja, é através de botões que se torna possível a interação com os dispositivos. Com a solução aqui presente é possível tirar melhor partido de todos os sensores disponíveis nos terminais móveis para melhor controlar os dispositivos IoT, não sendo conhecida nenhuma outra solução que tenha ido tão longe nesta abordagem.

## 1.5 Estrutura e organização

O presente estudo está organizado em seis capítulos que pretendem refletir as diferentes fases até à sua conclusão.

O primeiro capítulo introduz o tema da investigação e objetivos da mesma, bem como uma breve descrição da estrutura desta dissertação.

O segundo capítulo reflete o enquadramento teórico, designado por Revisão da Literatura. É feita uma análise aprofundada da situação existente aos dias de hoje, no que respeita à temática em causa.

O terceiro capítulo é dedicado à arquitetura proposta. Aqui descreve-se a arquitetura e a infraestrutura do problema a abordar.

O quarto capítulo apresenta a implementação para chegar às conclusões e responder às questões de investigação. De acordo com a solução proposta, é neste capítulo onde se descreve detalhadamente cada componente e o porquê de se ter optado por uma solução em detrimento de outras.

No quinto capítulo são apresentadas a análise e discussão dos resultados. Aqui apresentam-se de forma sumária os resultados comparativos dos testes realizados, não só funcionais como de desempenho.

No sexto e último capítulo apresentam-se as conclusões deste estudo bem como as recomendações, limitações e trabalhos futuros.

# Capítulo 2. **Revisão da Literatura**

## 2.1 Plataformas IoT

O mercado de IoT percebeu que criar aplicações a partir do zero é uma tarefa tediosa e um pouco arriscada, portanto cedo ficou claro que se deveria adotar uma solução experimentada e madura. É então que surge a necessidade de utilizar as plataformas IoT. Existem muitas plataformas que permitem o rápido desenvolvimento de novas aplicações no setor, como o Microsoft Azure [9] [10] , Amazon IoT [9] [11], IBM Watson [12] [13], eclipse smart home [14] [15], entre outras. Algumas destas plataformas (aquelas com maior expressão de mercado e aquelas que melhor se enquadram nos objetivos da presente pesquisa) serão analisadas e comparadas em detalhe, mais adiante [16].

De forma simplificada, o objetivo de qualquer dispositivo IoT é conectar-se a outros dispositivos e aplicações de IoT (principalmente na cloud) para transmitir informações usando protocolos da Internet. A ponte entre os sensores do dispositivo e as redes de dados é preenchida por plataformas IoT. Considerando as possibilidades que a IoT oferece, as empresas de tecnologia começaram a capitalizá-la. Agora, existem muitas plataformas de IoT disponíveis, que oferecem a opção de implementar aplicações de IoT sem qualquer custo e de imediato [17].

Existem mais de 360 plataformas de IoT, então teve de se filtrar de acordo com as necessidades do presente estudo, ou seja, as plataformas mais adequadas para casas inteligentes, que são de código aberto e permitem a integração de novos sensores / atuadores sem grandes custos. Assim, decidiu analisar-se o Eclipse Smart Home/OpenHAB, openRemote e openIot [16].

A plataforma que pareceu ser a mais adequada para o corrente estudo é a plataforma openHAB. Não só porque já existe experiência e algum know-how anteriores, como também é a que apresenta maior capacidade de integração com a maior parte de dispositivos atualmente existentes no mercado. Também devido à sua relativa simplicidade, pensa-se que será a plataforma mais adequada a utilizar.

### 2.1.1 Eclipse smarthome / openHAB

O Eclipse Smart Home (ESH) é um software de código aberto executado em ambiente residencial. Pode ser executado numa máquina simples como um Raspberry Pi e é projetado para sistemas *Home Building Automation* (HBA). Contém as principais estruturas de código e dados que são necessárias num ambiente de casa inteligente. O *Eclipse Smart Home* (ESH) é desenvolvido dentro da comunidade Java do Eclipse, aquela que construiu o eclipse IDE. O ESH baseia-se no Java CSGI e é executado em implementações Java fortemente reconhecidas. Uma destas é o OpenHAB, do qual o ESH deriva [14]. O ESH tem um forte foco em ambientes mistos, ou seja, soluções que lidam com a integração de diferentes protocolos e / ou padrões. O seu principal objetivo é fornecer um acesso uniforme a dispositivos e facilitar diferentes tipos de interação entre eles.

Os programadores podem construir diretamente uma solução individual de casa inteligente adicionando suas próprias extensões ao código. O resultado pode ser implementado em sistemas embebidos, que podem executar uma *Java Virtual Machine* (JVM), como um Raspberry Pi ou outros aparelhos semelhantes. O código é Java puro / OSGi e é construído sobre servidores aplicacionais Equinox, EMF e Jetty, que é extremamente leve, o que o torna ideal para ser executado em computadores de baixa potência.

A contribuição inicial e mais importante para este projeto vem do Open Home Automation Bus (openHAB), que fornece uma enorme lista de extensões gratuitas, que permitem integrar facilmente sistemas como o KNX, Philips Hue, Z-Wave, EnOcean, DMX, Plugwise, Homematic ou Sonos.

O projeto ESH é uma estrutura de software para a construção de soluções domésticas inteligentes com integração de diferentes protocolos (EnOcean, KNX, MODbus) ou standards. O seu primeiro objetivo é fornecer um acesso uniforme a dispositivos e informações e, em segundo lugar, facilitar diferentes tipos de interação com estes. Oferece tutoriais para programadores de software principiantes e fornece demonstrações de modo a ser possível começar a funcionar rapidamente. Oferece uma documentação on-line, um fórum de discussão e uma plataforma onde os programadores podem registar alguns bugs que vão encontrando. Fornece também informações para fabricantes de hardware / dispositivos para criar ligações para a plataforma ESH / openHAB para a integração dos

seus dispositivos. Isto é muito importante porque permite desenvolver praticamente qualquer sensor, incluindo os oferecidos pelos dispositivos móveis, e integrá-los com a plataforma OpenHAB. O ESH também tem uma boa comunidade de suporte com mais de 2500 contribuições no GitHub e mais de 2100 seguidores no Facebook [16].

### 2.1.2 OpenRemote

O OpenRemote [18] é uma plataforma de integração de software para automação de edifícios residenciais e comerciais. A plataforma OpenRemote é independente de protocolo de automação, opera em hardware pronto-a-usar e está disponível gratuitamente sob software *Open Source*. A arquitetura do OpenRemote permite edifícios inteligentes totalmente autónomos e independentes do utilizador. As interfaces de controle do utilizador final estão disponíveis para os sistemas operacionais móveis mais comuns, como dispositivos iOS e Android, e para dispositivos com browsers modernos. O design da interface do utilizador, a gestão de instalação e configuração podem ser operados remotamente com as ferramentas baseadas em *cloud*. Funciona com os seguintes protocolos / padrões: HTTP, Z-wave, serial, KNX, EnOcean e R-Net.

O software está disponível gratuitamente para os consumidores, pois é um projeto de código aberto. O OpenRemote está atualmente focado na construção de um negócio sustentável, que acredita poder conseguir licenciando o seu software para os fabricantes de dispositivos. O OpenRemote também vê uma oportunidade lucrativa além da automação residencial ao fornecer o seu software às cidades e à grande aglomeração de pessoas, os quais estão cada vez mais interessados em usar a tecnologia para tudo, desde a comunicação com os cidadãos até à monitorização do tráfego. Recentemente, o OpenRemote realizou um pequeno teste em Eindhoven, na esperança de usar automação e crowdsourcing para monitorizar e fazer algumas análises estatísticas numa cidade. Envolveu pessoas, medição com câmaras, medição do nível sonoro, monitorização das redes sociais e uma aplicação em que as pessoas no local podiam utilizar para avaliar o nível da experiência ocorrida.

O Open Remote possui documentação e um tutorial para os programadores. Oferece uma extensa documentação e fornece instruções para protocolos e integração de dispositivos. Oferece igualmente um fórum de utilizadores, documentação virada para o utilizador e tutoriais de utilizador. Possui um conjunto de parceiros em mais de 9 países que fornecem

suporte na realização de projetos com esta plataforma. A comunidade é um pouco menos expressiva que a comunidade de ESH, mas tal não significa que tenha uma comunidade de suporte má, mas em comparação com o ESH, o número de pessoas envolvidas é inferior: tem mais de 199 contribuidores no GitHub e mais de 588 seguidores no Facebook [16].

### 2.1.3 OpenIoT

O OpenIoT [19] é uma plataforma de *middleware* genérica para aplicações do mundo da IoT, o que permite ligar dispositivos conectados à Internet e *web services* por meio de uma interface de utilizador amigável, trabalhando em ambientes de *Cloud Computing* ou com um ambiente de servidor local.

Esta plataforma está disponível como um Kit de Desenvolvimento Virtual, fornecendo uma solução de cloud completa para a IoT o que permite aos utilizadores instalá-la facilmente, obter informações de sensores e conectar essas informações a *web services* sem se preocupar exatamente com quais os sensores que estão a ser utilizados.

O OpenIoT, é uma plataforma de código aberto para IoT, inclui funcionalidades exclusivas, como a capacidade de compor (dinamicamente e OnDemand) serviços IoT não triviais, seguindo um paradigma baseado em cloud. O OpenIoT é um esforço conjunto de colaboradores proeminentes de código aberto para habilitar uma nova linha de aplicações inteligentes de IoT de grande escala, de acordo com um modelo assente no paradigma da computação cloud. O OpenIoT desenvolveu uma infraestrutura de middleware para implementar / integrar as soluções da IoT. O modelo OpenIoT fornece os modos para:

- Recolher e processar os dados de praticamente qualquer sensor, incluindo dispositivos físicos, algoritmos de processamento de sensores, algoritmos de processamento de redes sociais e muito mais. Observe-se que no OpenIoT o termo sensor se refere a qualquer componente que possa fornecer observações e enviar dados para serem processados. O OpenIoT facilita a integração dos sensores acima com um esforço mínimo (ou seja, poucos dias x homem) para implementar um driver correto.
- Anotação semântica de dados de sensores, de acordo com as especificações da W3C Semantic Sensor Networks.

- Transmitir os dados dos vários sensores para um sistema de cloud computing.
- Dinamicamente descobrir / consultar sensores e os seus dados para serem processados.
- Compor e entregar serviços IoT que incluem dados de vários sensores.
- Otimização de recursos dentro da infraestrutura de middleware e cloud computing.

O OpenIoT combina e aprimora os resultados de projetos de middleware de ponta, como os projetos Global Sensor Networks (GSN) e Linked Sensor Middleware (LSM). O OpenIoT oferece APIs standards e abertas.

Como projeto financiado pela Comissão Europeia, não se caracteriza por um modelo de negócio específico para o consórcio. De facto, o objetivo do projeto é estabelecer bases para guias de exploração seguidos por parceiros individuais ou a agregação dos mesmos. Faz parte do repositório de plataformas abertas da União Europeia (UE) e é utilizado como base para vários projetos da UE.

A sustentabilidade dos resultados do projeto, incluindo a plataforma IoT, passa por ações de manutenção e evolução realizadas por parceiros ou terceiros, potencialmente requeridos. A documentação atual e o suporte da comunidade estão disponíveis no repositório do GitHub (com mais de 662 contribuidores). Alguns documentos estão disponíveis no site do projeto [16] [20].

### 2.1.4 Outras Plataformas

Por último e não de menosprezar na comunidade de entusiastas de IoT, têm surgido muito o nome de Domoticz [21] e de Home Assistant. A plataforma Domoticz está em grande crescimento cada vez com mais adeptos, por ser simples de configurar, sendo também de código aberto e gratuito. Pode ser também instalada em computadores de baixa potência como o RaspberryPi e outros semelhantes, mas possui suporte para menos dispositivos, com grande ênfase nos que suportam o protocolo z-wave.

O Home Assistant [22] (HA) é uma plataforma de código aberto para uma casa inteligente, desenvolvida em Python 3. Esta plataforma é utilizada para controlar dispositivos conectados em casas inteligentes. O HA pode integrar diferentes tipos de sensores, dispositivos e linhas de automação residencial. Ao nível das integrações, permite integrar com quase todo o tipo de dispositivos, pois tem suporte para KNX, Z-Wave, ZigBee, MQTT, entre muitos outros.

## 2.2 Protocolos e Comunicação

No IoT é necessário que os dispositivos comuniquem com as plataformas, que comuniquem entre si, etc. e, neste contexto, surgem diversos protocolos de comunicação muito específicos dentro do universo do IoT, tais como o X10, Z-wave, ZigBee, KNX, entre muitos outros.

Um dos primeiros a surgir foi o X10, utilizando a rede elétrica doméstica como meio de comunicação, com os prós e inconvenientes que isso possa trazer, mas ainda assim, após quase 4 décadas depois do seu surgimento, ainda continua com um grande peso neste tipo de indústria. Todos os dispositivos são recetores e, os meios de controle do sistema como controlos remotos ou teclados, são transmissores. Se pretender ligar uma lâmpada numa outra divisão, o transmissor emitirá uma mensagem em código numérico que inclui as seguintes partes:

- um alerta (uma mensagem/evento) para o sistema que está a emitir um comando;
- um número UUID para o dispositivo que deve receber o comando;
- um código que contém o comando real.

Isto tudo foi pensado para acontecer em menos de um segundo, mas o X10 [23] (a comunicação de protocolo padrão num passado recente) tem algumas limitações: a comunicação por redes elétricas nem sempre é fiável, porque as redes possuem muito ruído ao fornecer energia a outros dispositivos. Um dispositivo X10 pode interpretar uma interferência como um comando e reagir, ou pode não receber o comando, pois pode interpretá-lo como ruído da rede.

Em vez de passar pelas redes de energia elétrica, alguns sistemas usam ondas de rádio (RF) para se comunicar entre si, semelhante aos sinais de Wi-Fi e GSM. Os dois protocolos de comunicação via rádio mais importantes em sistemas de casas inteligentes são ZigBee e Z-Wave. Ambas as tecnologias são redes *mesh* (malha), o que significa que há mais de uma opção para a mensagem chegar ao seu destino. Na Tabela 1, é feita uma comparação entre os diversos protocolos mais comuns em sistemas de casas inteligentes.

*Tabela 1 – Comparação entre os protocolos IoT mais comuns*

| | Z-Wave | ZigBee | Insteon | KNX | LoRA |
|---|---|---|---|---|---|
| **Tipo** | Sem fios | Sem fios | Sem fios | Cabo | Sem fios |
| **Frequência** | 868.42 MHz | 2.4 GHz | 915 MHz | N.A. | 863-870 MHz |
| **Distância** | 30-100 m | 10-100 m | 45 m | N.A. | até 15 km |
| **Modulação** | FSK/GFSK | BPSK | FSK | N.A. | FSK |
| **Máx. Nós** | 232 | 64 000 | 256 | 57 375 | 1000 |
| **Consumos** | ↓↓ | ↓↓↓ | ↓↓ | N.A. | ↓↓↓ |
| **Cód. Aberto** | × | ✓ | × | × | ✓ |

### 2.2.1. Z-Wave

O Z-Wave [3] [24] usa um algoritmo de roteamento para garantir a rota mais rápida para as mensagens enviadas. Cada dispositivo Z-Wave possui um código e, quando o dispositivo é conectado ao sistema, o controlador da rede reconhece o código, determina a sua localização e adiciona-o à rede. Quando um comando é enviado, o controlador usa o referido algoritmo para determinar como a mensagem deve ser enviada até ao destino. Como esse roteamento pode ocupar muita memória numa rede, o Z-Wave estabeleceu uma hierarquia entre dispositivos: alguns controladores iniciam as mensagens e alguns são conhecidos como *slaves*, o que significa que estes só podem transportar e responder a mensagens, não podendo iniciar novas mensagens [25].

### 2.2.2. ZigBee

O nome de ZigBee [26] [27] mostra bem o conceito de rede *mesh*, porque as mensagens do transmissor fazem ziguezague como abelhas (*bee* em inglês significa abelha), procurando sempre o melhor caminho para o recetor. Enquanto a Z-Wave [3] usa uma tecnologia proprietária para operar o seu sistema de rede, o protocolo ZigBee é baseado no standard estabelecido pelo Instituto de Engenheiros Elétricos e Eletrónicos (IEEE) para redes sem fio. Isso significa que qualquer empresa pode criar um produto compatível com ZigBee (veja-se o caso da HUE, criado pela empresa Philips) sem pagar taxas de licenciamento pela tecnologia que está por detrás, o que pode dar uma grande vantagem ao ZigBee no mercado. Como o Z-Wave, o ZigBee possui dispositivos totalmente funcionais (ou aqueles que fazem o roteamento das mensagens) e dispositivos de funções reduzidas (ou aqueles que não o fazem). No que à potência e consumo diz respeito o

ZigBee tem uma preocupação total e permite que alguns sensores / atuadores adormeçam e sejam despertados somente quando uma mensagem é enviada para os mesmos [25].

### 2.2.3. Insteon

O uso de uma rede sem fios oferece mais flexibilidade para a instalação de dispositivos, mas, tal como nas redes de energia elétrica, estas podem sofrer do fenómeno de interferência. O Insteon [28] oferece um caminho para a sua rede se comunicar por meio de fios elétricos e de radiofrequência, tornando-se uma rede de malha dupla. Se a mensagem não estiver a ser transmitida numa plataforma, ela tentará de outro modo.

Com este protocolo, em vez de encaminhar a mensagem, um dispositivo Insteon transmitirá a mensagem para todos os nós da rede, e todos os dispositivos retransmitirão a mensagem até que o comando seja concluído e chegue até ao nó de destino. Os dispositivos agem como pares, ao contrário de um que serve como coordenador e outro como recetor. Isto significa que quanto mais dispositivos Insteon estiverem instalados numa rede, mais robusta será a entrega da mensagem [25] [28]. Mas como funciona por "inundação" da rede poderá ser problemático, pois existe a possibilidade de sobrecargas na rede as quais queremos evitar na nossa rede doméstica.

### 2.2.4. KNX

O KNX [2] [29] [30] é um protocolo para casas inteligentes, bem como edifícios inteligentes. Vem da convergência de três diferentes sistemas concorrentes: o European Home Systems Protocol (EHS), o BatiBUS e o European Installation Bus (EIB ou Instabus).

Como canal de comunicação, o KNX é bastante versátil e permite múltiplas opções, tais como:

- Cabo de par trançado
- Rede de energia elétrica (similar ao X10)
- Rádio (KNX-RF)
- Infravermelhos
- Ethernet (também conhecida como EIBnet / IP or KNXnet / IP)

Como nos módulos X10, existe uma infinidade de módulos KNX disponíveis, desde interruptores a módulos de controle de luz, persianas, etc. e, claro, controladores de

sistema inteligentes que permitem controlar todo o sistema e obter relatórios detalhados sobre os consumos e operações do estado da instalação. Contando com o apoio de gigantes da indústria como ABB, Bosch, Miele, Schneider e Siemens, entre outros, a KNX torna-se uma aposta com garantia futura e recomendada para novas instalações nas áreas de controle de iluminação, ar condicionado, segurança, gestão de acessos e gestão de energia. A sua principal desvantagem é o custo relativamente alto de seus módulos.

### 2.2.5. LoRa

Recentemente surgiu uma nova tecnologia IoT, como Long Range (LoRa), permitindo a comunicação sem fios com grande eficiência energética em longas distâncias. Os dispositivos normalmente comunicam entre si diretamente, o que elimina a necessidade de construir e manter uma rede complexa de múltiplos saltos (rede *mesh* ou rede em malha) [4].

O protocolo LoRa fornece uma gama de opções de comunicação (frequência central, fator de propagação, largura de banda, taxas de codificação) a partir das quais um transmissor pode selecionar [31]. Existe um limite em relação ao número de transmissores que um sistema LoRa pode suportar, cerca de 1000. LoRa é uma técnica proprietária de modulação por espalhamento espectral. É um derivado do *Chirp Spread Spectrum* com correção de erro de encaminhamento integrada. As transmissões são em banda larga para combater a interferência e lidar com deslocamentos de frequência causados por cristais de baixo custo. Um recetor LoRa pode descodificar transmissões de 19,5 dB abaixo do nível de ruído, possibilitando assim longas distâncias de comunicação [31] [32].

LoRa não será o protocolo mais adequado para este caso de estudo uma vez que é mais orientado para situações onde se requeira uma maior área de cobertura, como é o caso de quintas ou mesmo cidades. No presente não necessitamos de tais requisitos de cobertura, pois os nossos dispositivos concentrar-se-ão todos numa casa residencial. LoRa é assim mais utilizado para WAN e não tanto para LAN.

## 2.3 Integração de Dispositivos Móveis

Já muito se descreveu o modo de interação com os dispositivos finais através dos sensores de um smartphone. No entanto convém realçar que para além deste existem muitas outras formas de o fazer. Seja por interação direta com os dispositivos ou por aplicação própria.

No primeiro caso, o mais óbvio, podemos interagir com os dispositivos diretamente por acionamento de um interruptor ou qualquer outro mecanismo que esteja pré-programado para atuar sobre os mesmos. No entanto, podemos perfeitamente aproveitar os sensores de um smartphone para interagir com estes, obtendo exatamente os resultados do que no primeiro caso, mas com uma experiência totalmente diferente e de certa forma inovadora.

Com o recurso a uma câmara de um smartphone, por exemplo, podemos reconhecer uma imagem e depois com recurso a outros sensores do smartphone, interagir com os dispositivos, isto é, ao rodar ou inclinar o smartphone, ao tapar/destapar o sensor de luz, etc., podemos interagir como se de botões se tratassem.

## 2.4 Sensores Disponíveis

Tendo o presente estudo como objetivo utilizar sensores de dispositivos móveis para interação com os dispositivos inteligentes, foi necessário investigar quais os sensores disponíveis num *smartphone*, que funcionalidades podem ser aproveitadas e como se pode tirar partido dos mesmos, incluindo que tipo de medidas são captadas por estes [33].

Como exemplo de alguns sensores que podemos utilizar, destaca-se a câmara, o GPS, o acelerômetro, os sensores de proximidade, os sensores de luz, entre muitos outros. Com a câmara podemos, por exemplo, aproveitar o reconhecimento de objetos e com base nisso interagir com eles.

Existem fundamentalmente 3 categorias de sensores:

- Movimento – medem forças de aceleração e de rotação nos diferentes 3 eixos do *smartphone*;
- Ambiente – medem vários parâmetros de ambiente, tais como temperatura, pressão atmosférica e humidade relativa, entre outros;
- Posição – medem a posição física dos aparelhos. Esta categoria inclui sensores de orientação e de campo magnético.

Com o acelerómetro, por exemplo, podemos inclinar para um lado e para o outro e alterar o estado de um reóstato ligado a uma lâmpada e diminuir a luz ou elevar ou baixar uma persiana, isto é, algo que é sensível ao movimento.

Com o sensor de proximidade, podemos medir a distância de um qualquer objeto ao aparelho e com isso tomar certas ações consoante a distância medida. A mesma coisa com

o sensor de luminância, onde podemos tomar decisões de acordo com o lux medido. Por exemplo, podemos ligar ou desligar um dispositivo dependendo da luminosidade medida naquele momento. Em suma podemos usar estes sensores como sensores comuns, mas neste caso tirando partindo do facto estarem embebidos num smartphone.

Na Tabela 2, tenta, ainda que de forma resumida, mostrar-se algumas características de cada um dos sensores.

*Tabela 2 – Resumo dos principais sensores disponíveis em smartphones*

| Sensor | Descrição | Unidades |
|---|---|---|
| Câmara | Captura de imagens | - |
| GPS | Coordenadas geográficas: long, lat | º |
| Acelerómetro | Força de aceleração nos 3 eixos | $m/s^2$ |
| Proximidade | Distância de cada objeto ao sensor | cm |
| Luminosidade | Luminância | lx |
| Giroscópio | Taxa de rotação em cada um dos 3 eixos | rad/s |
| Campo Magnético | Força do campo magnético nos 3 eixos | μT |
| Temperatura | Temperatura do ar ambiente | ºC |
| Pressão Atmosférica | Pressão atmosférica do ar ambiente | hPa |
| Humidade Relativa | Humidade relativa do ar ambiente | % |

## 2.5 Software de Reconhecimento de Imagem

Um dos sensores mais poderosos num smartphone é sem dúvida a câmara. Esta pode, por exemplo, ser utilizada para captar imagens dos objetos que pretendemos reconhecer. Neste sentido, foi também investigado o tema de reconhecimento de imagens, pois permite-nos ter uma forma alternativa de reconhecer e identificar *smart devices* utilizando a câmara de um smartphone.

Existem inúmeras ferramentas para fazer o reconhecimento de imagens, tais como openCV, SimpleCV, Mathlab, TensorFlow, entre outras. O Mathlab é uma ferramenta

genérica, a qual permite ter um sem fim de utilizações, mas como é mais difícil a prototipagem, fica desde já de fora das ferramentas candidatas para o presente estudo.

O TensorFlow [34] é uma ferramenta de *machine learning* que opera em larga escala e em ambientes heterogéneos. Utiliza gráficos de fluxo de dados para representar a computação, o estado partilhado e as operações que modificam esse estado. Mapeia os nós de um gráfico de fluxo de dados em muitas máquinas que compõem um cluster e dentro de uma máquina em vários dispositivos computacionais, incluindo CPUs multicore, GPUs (*Graphical Processing Unit*) de uso geral e ASICs personalizados, conhecidos como TPU (*Tensor Processing Units*). O TensorFlow suporta uma grande variedade de aplicações, com especial foco no treino e inferência em redes neurais [34].

Claro que pode fazer-se o reconhecimento de imagem com o TensorFlow, embora seja adequado para problemas mais gerais, como classificação, *clustering* e regressão. O Tensorflow é apenas uma biblioteca para trabalhar com tensores e diferenciação automática entre gráficos computacionais. Resumidamente, é apenas um pacote de diferenciação avançada.

Por outro lado, o simpleCV é um pouco mais simples que o openCV, mas utiliza o mesmo motor do segundo. Assim e, até porque já havia algum *know-how* sobre openCV, entre estes dois, pensa-se que a opção mais adequada para este estudo será o openCV, uma vez que não será necessário incluir a complexidade de algoritmos com recurso a *machine learning*, pois a abordagem a seguir será à partida mais tradicional.

### 2.5.1 Algoritmos para Deteção de Objetos

Existem várias ferramentas para fazer o reconhecimento de imagem, mas a mais adequada neste contexto é o openCV, não só por ser de código aberto, mas também por ser gratuito e adaptável a inúmeros sistemas operativos: Linux, Windows, MacOS, android, etc. [35]. Dentro do openCV existem vários algoritmos disponíveis, dos quais podemos destacar o SIFT, o SURF e o ORB, por serem os mais comuns.

O SIFT, o SURF e o ORB são algoritmos de *Brute-Force* (BF), mas existem outras alternativas como por exemplo o FLANN Matcher que significa biblioteca rápida para aproximação dos vizinhos mais próximos (*Fast Library for Approximate Nearest Neighbors*). O FLANN até pode ser mais rápido que os primeiros, mas poderá ser também bastante mais impreciso; podendo ser afinado, mas à custa da perda de desempenho no

que diz respeito ao algoritmo. Assim, vamos focar-nos nos algoritmos de BF, deixando o FLANN para proposta de investigação futura.

O algoritmo SIFT tenta resolver a rotação da imagem, transformações, intensidade e alteração do ponto de vista dos pontos de interesse correspondentes. O algoritmo SIFT é composto por 4 etapas básicas. Primeiro, estima um extremo do espaço de escala usando a Diferença de Gaussiano (DoG). Depois, atribui uma localização de ponto-chave em que os candidatos a ponto-chave são localizados e afinados, eliminando os pontos de baixo contraste. Em terceiro lugar, atribui uma orientação ao ponto-chave com base no gradiente de imagem local e, por fim, gera um descritor para calcular o descritor de imagem local para cada ponto-chave com base na magnitude e orientação do gradiente de imagem [36].

O algoritmo SURF aproxima o DoG com *box filters* (filtros de caixa). Em vez de calcular a média gaussiana da imagem, os quadrados são usados para aproximação, pois a convolução quadrática é muito mais rápida se se usar uma imagem conforme a ilustrada pela Figura 3.

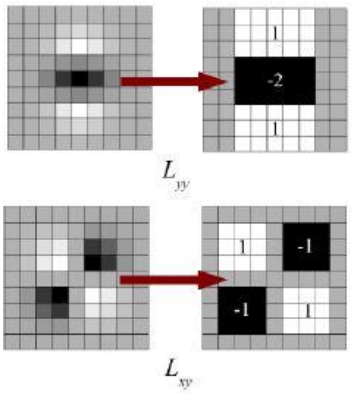


*Figura 3 – Aproximação do LoG com box filter* [37]

Isto também pode ser feito em paralelo para diferentes escalas. O SURF utiliza um detetor BLOB o qual é baseado na matriz Hessiana para encontrar os pontos de interesse. Para atribuição de orientação, usa respostas de *wavelets* [38] nas direções horizontal e vertical, aplicando pesos gaussianos adequados.

Para os descritores, o SURF usa também as respostas da *wavelet*. Uma vizinhança em torno do ponto principal é selecionada e dividida em sub-regiões e, em seguida, para cada

sub-região, as respostas da *wavelet* são obtidas e representadas para obter o descritor do SURF.

$$v = \left(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right). \quad (1)$$

O sinal de Laplaciano que já é calculado na deteção é usado para pontos de interesse subjacentes. O sinal do Laplaciano distingue bolhas brilhantes em fundos escuros da caixa inversa. No caso de correspondência, os descritores são comparados apenas se tiverem o mesmo tipo de contraste (com base no sinal), o que permite uma correspondência mais rápida [39].

O algoritmo ORB é uma fusão do detetor de ponto-chave FAST e do descritor BRIEF com algumas modificações. Inicialmente para determinar os pontos principais, utiliza o FAST. Em seguida, uma medida de canto de Harris é aplicada para encontrar os N pontos principais. O FAST não calcula a orientação e é uma variante de rotação. Calcula a intensidade ponderada do centroide do adesivo com o canto localizado no centro. A direção do vetor deste ponto de canto para o centroide fornece a orientação. Os momentos são calculados para melhorar a invariância da rotação. O descritor BRIEF é executado, se houver uma rotação no plano. No ORB, uma matriz de rotação é calculada usando a orientação do *patch* e, em seguida, os descritores BRIEF são criados de acordo com a orientação [40].

Na Tabela 3 e Tabela 4 encontram-se resumidas as principais diferenças ao nível de desempenho entre os referidos algoritmos, fazendo-se variar o ângulo e medindo o número de reconhecimentos. Já na Tabela 5, temos os tempos de cada algoritmo para a mesma imagem rodada.

*Tabela 3 – Reconhecimentos versus ângulo de rotação* [36]

| Ângulo | 0º | 45º | 90º | 135º | 180º | 225º | 270º |
|---|---|---|---|---|---|---|---|
| SIFT | 100 | 65 | 93 | 67 | 92 | 65 | 93 |
| SURF | 99 | 51 | 99 | 52 | 96 | 51 | 95 |
| ORB | 100 | 46 | 97 | 46 | 100 | 46 | 97 |

*Tabela 4 – Resultados de imagens contra a mesma imagem rodada* [36]

| | Tempo | Pts. Chave 1 | Pts. Chave2 | N.º Rec. | % Rec. |
|---|---|---|---|---|---|
| SIFT | 0.13s | 248 | 229 | 183 | 76.7 |
| SURF | 0.04s | 162 | 166 | 119 | 72.6 |
| ORB | 0.03s | 261 | 267 | 168 | 63.6 |

Resumidamente, o SURF adiciona muitos recursos para melhorar a velocidade em cada etapa. A análise mostra que é cerca de 3 vezes mais rápido que o SIFT, enquanto a eficácia é comparável ao SIFT. O SURF é bom para manipular imagens desfocadas e rodadas, mas não é tão bom a lidar com a alteração do ponto de vista ou mudanças de iluminação. Já o ORB também é muito rápido, à semelhança do SURF, um pouco mais rápido até, mas em termos da percentagem de reconhecimentos é ligeiramente inferior aos outros dois algoritmos, especialmente quando a imagem se apresenta com rotação.

Pelas razões anteriores, pensa-se que o SURF será o melhor candidato a utilizar neste estudo, pois tem um bom compromisso entre eficácia e desempenho, face aos outros dois algoritmos.

# Capítulo 3. **Arquitetura Proposta**

De forma a cumprir os objetivos e responder ao problema que temos vindo a referir nos capítulos anteriores, iremos abordar a arquitetura proposta para este estudo.

A arquitetura terá de ser capaz de responder a alguns desafios para conseguir dar resposta aos problemas que pretendemos ultrapassar, tal como reconhecer objetos através de uma única aplicação para podermos interagir com estes através dos sensores do dispositivo móvel.

Um cenário típico de utilização desta solução será o de apontar com a câmara de um smartphone para uma imagem (cena) e esta deverá ter uma correspondência noutra imagem (objeto) previamente armazenada. Após ter sido feito o *match*, enviaremos comandos para o objeto reconhecido, através dos sensores que estejam associados ao objeto reconhecido. Por exemplo, apontamos a câmara para uma lâmpada, esta imagem será enviada para um servidor o qual a processará e fará o reconhecimento dessa imagem com uma das imagens previamente armazenadas. Após este processo, com recurso aos sensores do smartphone enviaremos comandos ao objeto lâmpada para acender ou apagar.

Esta abordagem de utilizar a câmara para reconhecer objetos, foi uma opção tomada aquando da discussão da arquitetura, pois poderíamos ter escolhido outras formas de reconhecimento, tal como usando *tags* NFC ou através de um QrCode para identificar os objetos, mas tal obrigaria a termos algo nos objetos a controlar que os permitisse identificar dessa forma. Desta forma, utilizamos simplesmente a câmara para reconhecimento dos objetos, sem necessitarmos de recursos auxiliares. No entanto a arquitetura poderia acomodar também estas abordagens.

Assim, e tendo em atenção os objetivos para a arquitetura, teremos de cumprir alguns requisitos ou desafios, pelo que devemos:

1. ter uma aplicação capaz de tirar fotos a objetos e guardá-los para uso futuro;
2. ter uma aplicação capaz de tirar sucessivas fotos e compará-las com os objetos armazenados no ponto anterior e indicar se houve ou não uma correspondência;
3. ter uma aplicação que coloca ao dispor do utilizador diversos sensores com os quais este poderá interagir (cada objeto reconhecido terá um sensor associado);
4. ter um mecanismo de transmissão de dados entre a aplicação móvel e uma plataforma IoT;

5. ter uma plataforma IoT que controle os diversos dispositivos;
6. dispor de vários dispositivos, ligados a uma plataforma IoT, pelos mais diversos protocolos (KNX, zigbee, etc.).

Para dar resposta a estes desafios propõe-se a arquitetura que se apresenta na Figura 4 onde os seus componentes são apresentados numa perspetiva de alto nível.

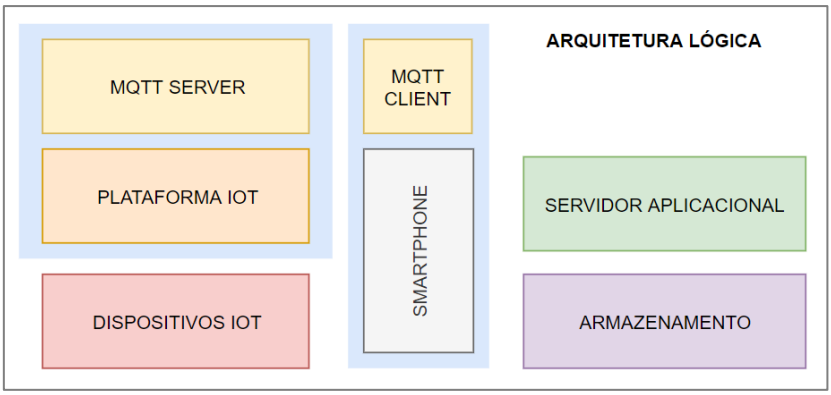


*Figura 4 – Arquitetura proposta (visão de alto nível)*

De forma sucinta esta arquitetura consiste nos seguintes componentes:

- **Smartphone** – possui os sensores e corre a aplicação móvel e é de certa forma transversal a toda a arquitetura
- **Servidor Aplicacional** – corre uma API REST, responsável pela comunicação com o smartphone, possui um motor para reconhecimento de imagens e é cliente de uma base de dados para persistência das imagens.
- **Armazenamento** – responsável por persistir e ler os objetos de uma base de dados.
- **MQTT Client** – Este componente corre dentro do smartphone e é o ponto de partida com as comunicações para a plataforma IoT, via MQTT Server.
- **MQTT Server** – faz a ponte entre o smartphone e a plataforma IoT.
- **Plataforma IoT** – processa os pedidos, vindos pelo MQTT Server e envia-os para os dispositivos IoT correspondentes.
- **Dispositivos IoT** – são os nós finais ou dispositivos com os quais pretendemos interagir.

Iremos acompanhar cada componente ou conjunto de componentes ou módulos com diagramas, para que a perceção daquilo que relatamos fique bastante claro, também se

detalhará, na forma de diagramas de sequência, as comunicações entre dispositivos, como se verá mais adiante.

## 3.1 Descrição

A arquitetura pode ser dividida em duas fases bem distintas: Uma primeira fase corresponde à gravação prévia das imagens – PERSISTÊNCIA. Numa segunda fase, far-se-á o reconhecimento das imagens contra as imagens previamente gravadas – RECONHECIMENTO. Como se verá adiante, os componentes presentes em cada uma destas etapas podem ser acompanhados conjuntamente com a Figura 5.

Na fase da persistência, podemos contar com um smartphone (aplicação móvel), um servidor aplicacional e uma base de dados. Através de uma API REST a aplicação enviará a imagem capturada (objeto) para um servidor aplicacional, sendo posteriormente enviada para uma base de dados, onde é armazenada. Nesta fase, o utilizador final terá de armazenar a imagem com alguns meta-dados para depois se saber por exemplo o tipo de sensor associado ao objeto, bem como qual o objeto físico que lhe está associado, como veremos mais adiante.

Na fase do Reconhecimento, a aplicação enviará através de uma API REST as imagens capturadas (cenas) para comparação com as imagens previamente gravadas (objetos). Após ter sido encontrada a imagem que corresponde à imagem enviada, esta será devolvida à aplicação móvel de modo ao utilizador fornecer feedback quanto à sua veracidade. É nesta fase que a aplicação envia todos os *logs* recolhidos até então para uma base de dados temporal e leva o utilizador a interagir com o dispositivo, enviando comandos para a plataforma IoT, a qual saberá, de acordo com o comando recebido, decifrá-lo e reenviá-lo para o dispositivo correspondente.

Convém clarificar desde já o porquê de se ter escolhido colocar a API REST num servidor externo. Poderia estar tudo concentrado no *smartphone* sem necessidade desta aplicação a correr num servidor externo. Mas optou-se por esta abordagem por dois motivos principais:

- por ser este componente que comunica com a base de dados (para armazenamento dos objetos), seria uma má prática fazê-lo diretamente a partir do smartphone, pois iria quebrar algumas boas práticas de programação para dispositivos móveis;

- por outro lado, era mais simples compilar e instalar o software de reconhecimento de imagens num ambiente mais aberto, por exemplo no seio de uma aplicação Java a correr num servidor aplicacional do tipo tomcat ou jetty.

Claro que esta opção também acarreta inconvenientes, como mais à frente se verá.

Numa terceira fase independente, existe ainda a possibilidade de o utilizador enviar um pedido ao servidor aplicacional para imprimir todas as imagens armazenadas na base de dados, por exemplo através de uma lista apresentada no ecrã do smartphone – LISTAGEM.

Na Figura 5, podemos ver com mais detalhe o que acabamos de descrever.

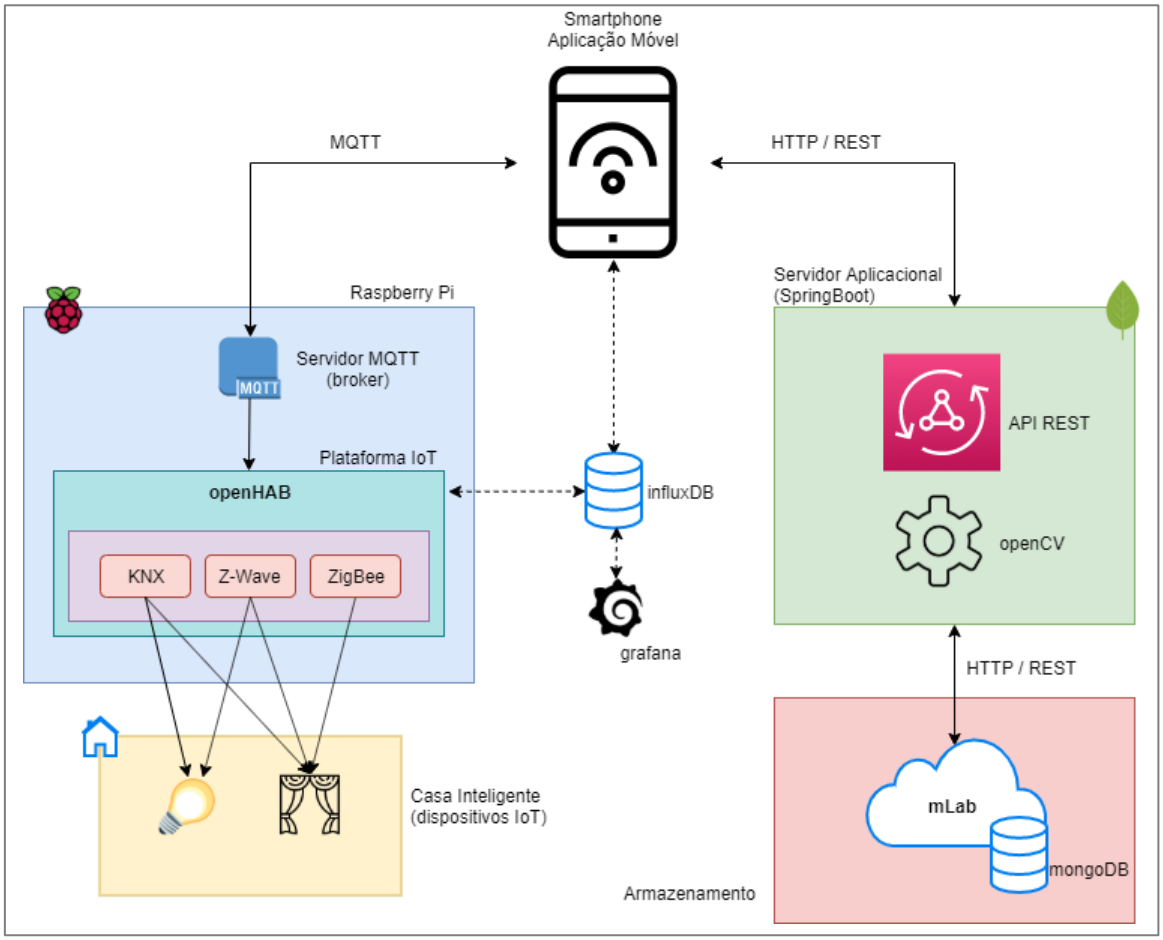


*Figura 5 – Arquitetura global do sistema*

A solução proposta deve ser ainda capaz de cumprir com o seguinte:

- deve ter uma interface simples e com poucos botões;
- deve obedecer a gestos;

- deve ser capaz de capturar imagens (objetos) e armazená-las;
- juntamente com as imagens deve guardar meta-dados (nome, ação e descrição);
- deve enviar várias imagens em modo contínuo e processá-las uma a uma até ser encontrada uma correspondência. Isto é, capturar várias imagens até encontrar uma que o algoritmo de reconhecimento considere que se trate da mesma imagem;
- deve dar ao utilizador a possibilidade de este indicar se a imagem reconhecida é um verdadeiro ou falso positivo;
- caso a imagem seja corretamente identificada (independentemente de ser um verdadeiro ou falso positivo), a aplicação deve mostrar ao utilizador qual o sensor com que pode interagir;
- a interação deve ocorrer imediatamente após o feedback do utilizador;
- deve ter acesso a vários sensores para interagir com os objetos (rotação, proximidade, etc.);
- deve permitir ao utilizador ver todas as imagens armazenadas na base de dados, juntamente com o seu UUID e descrição;
- todos os componentes físicos envolvidos devem ter os relógios completamente sincronizados;
- deve guardar todas as marcas temporais relevantes, tais como:
    a. início do reconhecimento,
    b. fim do reconhecimento,
    c. envio do comando para a plataforma,
    d. entre outros.
- deve ser rápida no reconhecimento dos objetos

### 3.1.1 Smartphone e Cliente MQTT

Esta é uma peça fundamental e transversal de toda a arquitetura, pois está praticamente sempre presente em todos os passos do fluxo e interage com a maior parte dos outros componentes, com exceção da base de dados e dispositivos IoT. É no *smartphone* que corre a aplicação móvel e é a partir desta que se iniciam todos as fases do processo por iniciativa do utilizador.

Este deverá conter uma aplicação com as seguintes funcionalidades principais:

- Definições de endereços e portos;

- captura de objetos;
- procura de objetos;
- listar os objetos.

Assim que o utilizador entra na aplicação, deverá primeiro definir os endereços e portos dos outros componentes, conforme a Figura 6. Por exemplo, deverá definir o endereço e porto do servidor aplicacional, o endereço e porto da base de dados temporal, entre outros.

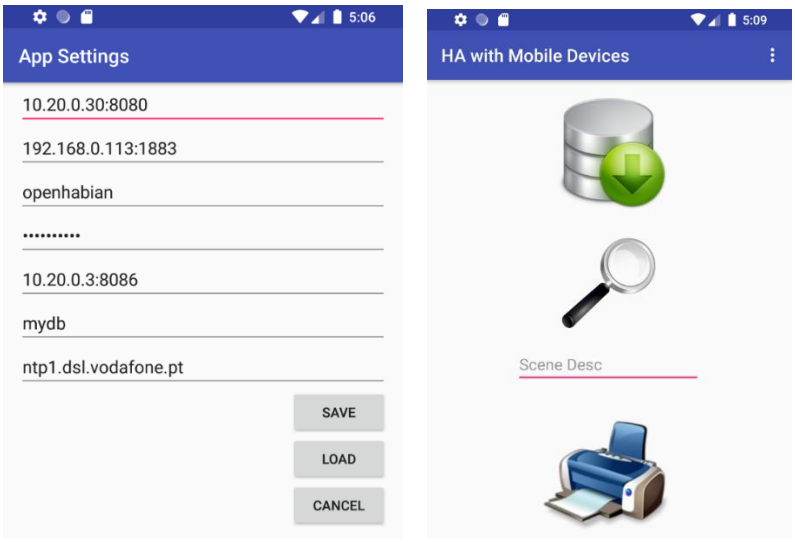


*Figura 6 – Definições da aplicação e ecrã principal*

É fundamental que a aplicação garanta o seu correto funcionamento apenas se o *wifi* do smartphone estiver ligado. Isto é necessário, pois a forma como o *smartphone* comunicará com os restantes componentes presentes na arquitetura será através da rede sem fios doméstica.

Após ter completado a etapa anterior, a aplicação deverá disponibilizar 3 opções ao utilizador: armazenamento, procura e impressão (vide Figura 6).

A primeira visa armazenar as imagens dos dispositivos com os quais queremos interagir. Quando pressionado o botão de armazenamento, o utilizador é levado para um formulário onde deve preencher alguns meta-dados (vide Figura 7 – lado esquerdo), os quais acompanharão a imagem até ao seu destino final, como por exemplo a descrição da imagem, um identificador único (UUID) e o tipo de sensor que será suposto o utilizador utilizar para controlar esse objeto.

A segunda, a procura de imagens, deverá fazer o match, da imagem a procurar, com as previamente armazenadas na base de dados. Posteriormente, quando houver um match, a aplicação deverá conduzir o utilizador para um ecrã onde ele possa utilizar os diversos sensores com os quais poderá interagir e assim modificar o estado do dispositivo a controlar, através do envio de comandos MQTT para o tópico (nome da imagem) associado ao objeto. Isto é, com os dados recolhidos pelo sensor, são enviados comandos através do componente MQTT cliente (embutido no dispositivo móvel) para o broker (servidor MQTT). Como o MQTT é um serviço de mensagens utilizando tópicos, é fundamental que haja sempre um subscritor (neste caso a plataforma IoT) para cada tópico publicado. Só assim se garantirá que a mensagem chegará ao destino pretendido.

Por último existe ainda a possibilidade de o utilizador imprimir as imagens armazenadas na base de dados sob a forma de uma lista, conforme se apresenta na Figura 7 (lado direito).

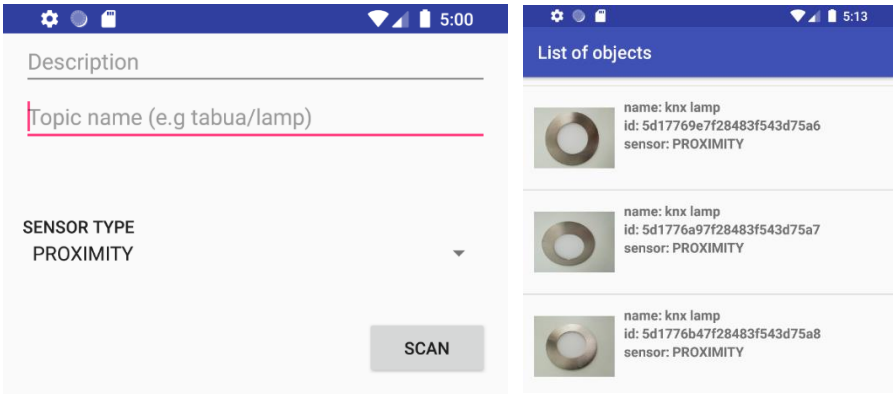


*Figura 7 – Lista de objetos previamente armazenados*

### 3.1.2 Servidor Aplicacional e Armazenamento

Este componente poderá ser por exemplo uma aplicação externa numa outra máquina (ou no próprio smartphone) e conter várias funções, tais como armazenar, procurar e imprimir os objetos. Todas estas funções são expostas ao exterior numa API REST que é invocada quando solicitado pelo smartphone, um dos componentes com que interage, bem como com a camada de persistência de dados.

No arranque, por questões de desempenho, deve ir à base de dados (componente de armazenamento) buscar todas as imagens, efetuar um pré-processamento de cálculo dos descritores e armazená-los em memória, conforme se verá mais adiante.

Das 3 funcionalidades acima referidas, destaca-se a da procura de objetos, pois é onde ocorre a parte mais complexa de toda a aplicação. Não que as outras não sejam igualmente importantes, pois sem estas a aplicação não funcionaria e não cumpriria todos os objetivos e requisitos. Mas é na parte da procura de objetos que está toda a lógica e algoritmos principais que potenciam esta solução.

Nesta parte da API, pode usar-se algoritmos *open source* como por exemplo openCV SURF, SIFT ou ORB. São estes os responsáveis pela correta identificação do objeto que queremos encontrar.

O que se pretende nesta fase é que mediante várias imagens que nos cheguem através da API, sejam feitas comparações dos descritores das imagens a comparar e tentar encontrar um *match*. Isto é, o smartphone envia uma imagem (cena) capturada em tempo real, esta quando chega ao servidor, é processada e são calculados os seus descritores (pontos-chave que são considerados pontos característicos da imagem). Como no arranque todas os objetos previamente já foram carregados e pré-processados em memória, o que o algoritmo faz é a comparação entre descritores (da cena contra os de todos os objetos). Se encontrar mais correspondências que as definidas como mínimas para serem considerados o mesmo objeto, para e retorna com êxito mostrando ao utilizador qual a imagem (objeto) encontrada.

De referir ainda que, sempre que é armazenado um novo objeto, a aplicação encarregar-se-á de o armazenar na base de dados, mas também atualizará a sua lista de descritores armazenada em memória.

Quanto ao sistema de armazenamento pode ser de qualquer tipo, desde que permita guardar imagens em formato binário, bem como meta dados sobre a imagem. Poderá ser local ou externa à aplicação. É nesta que deverão ser guardadas as imagens base com as quais queremos posteriormente, mediante procura ativa, fazer um match. Estas imagens que se armazenam na BD deverão corresponder a uma fotografia pormenorizada do dispositivo a interagir (objeto).

Por exemplo, se quisermos interagir com uma lâmpada ou candeeiro, deverá ser armazenada na BD uma foto pormenorizada deste dispositivo, bem como alguns atributos adicionais, tais como um identificador único, o tipo de sensor do smartphone com o qual se pode interagir, e um nome.

O único componente com o qual interage é o componente do servidor aplicacional, pois estamos perante um componente ou nó final, no contexto desta arquitetura.

As imagens armazenadas não deverão ocupar muito espaço, pois caso contrário, poder-se-á perder algum tempo devido à sobrecarga a circular pela rede. Assim, para tornar o processo de armazenamento e match das imagens mais célere, estas deverão ter um tamanho algo reduzido. Por outro lado, não deverão ser demasiado pequenas e de baixa resolução, pois tal poderia conduzir a perda no índice de sucesso de reconhecimento de imagem.

Tal como referido no ponto anterior estas são lidas aquando do início da aplicação que corre no servidor aplicacional, para tornar o processo de reconhecimento mais rápido.

### 3.1.3 MQTT (broker) e Plataforma IoT

Este componente faz a ponte entre o smartphone e a plataforma IoT (MQTT [41] client que corre embebido na aplicação móvel e o MQTT server que corre embebido no hardware onde se encontra a plataforma IoT). Corre num servidor externo ao smartphone e pode, por exemplo, utilizar-se um servidor que cumpra o protocolo MQTT ou outro, onde fará de broker cumprindo o padrão de desenho *publish* / *subscribe*.

O MQTT é um protocolo de conectividade máquina a máquina (M2M) / "IoT". Uma das grandes vantagens em termos de protocolo de transferência de mensagens é que é extremamente leve. Também é ideal para aplicações móveis devido ao seu pequeno tamanho, baixo consumo de energia, pacotes de dados reduzidos e distribuição eficiente de informações para um ou vários recetores.

Primeiro, a plataforma IoT (ver ponto seguinte) deve registar-se como subscritora de alguns tópicos. Tópicos esses em que o *smartphone* publicará mensagens do tipo ON ou OFF ou outras e o que este componente se encarrega de fazer é enviar as mensagens para quem as subscreveu, neste caso a plataforma de IoT. Na Figura 8 é ilustrado o funcionamento do protocolo MQTT. Deste modo, do lado esquerdo, temos o *smartphone* como cliente publicador e do lado direito a plataforma como cliente subscritor.

Assim, cada item ou objeto terá o seu próprio tópico e serão enviados comandos para o tópico correspondente, pelo que teremos a certeza de que a plataforma os receberá e processá-los-á.

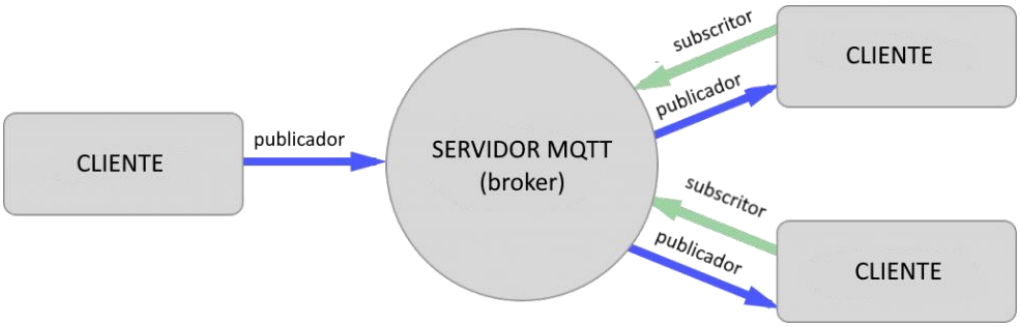


*Figura 8 – Funcionamento do protocolo MQTT*

O elo de ligação entre o servidor MQTT e os dispositivos (as lâmpadas, os estores, etc.) deverá ser por intermédio de uma plataforma IoT. Esta deverá ter pelo menos duas interfaces bem distintas: uma para comunicação com o servidor MQTT e outra para comunicação com os dispositivos ou objetos.

A primeira interface irá receber os dados enviados pela aplicação (via servidor MQTT), tais como os dados recolhidos pelos sensores, bem como a identificação do dispositivo com o qual queremos interagir. A segunda interface deverá, recorrendo a um ou mais protocolos de comunicação para (KNX, ZigBee, entre outros) assegurar a comunicação com os dispositivos. Esta tarefa deverá ser previamente configurada de forma a permitir o controlo dos dispositivos, com recurso aos *bindings* do openHAB. Este tipo de configurações deverá ser efetuado aquando da instalação e configuração da plataforma. Não será acessível a todos os utilizadores finais, pois requer algum conhecimento específico, quer da plataforma, quer dos dispositivos a controlar.

A plataforma deverá ainda ser o único ponto de entrada para os mais diversos sensores e dispositivos que queremos controlar, diminuindo assim potenciais riscos ou diminuição da segurança como por exemplo ter várias aplicações com vários pontos de entrada na nossa rede privada. Este é outro dos problemas que pretendemos resolver.

No que à plataforma diz respeito e, conforme a Figura 9, há ainda que clarificar alguns pontos.

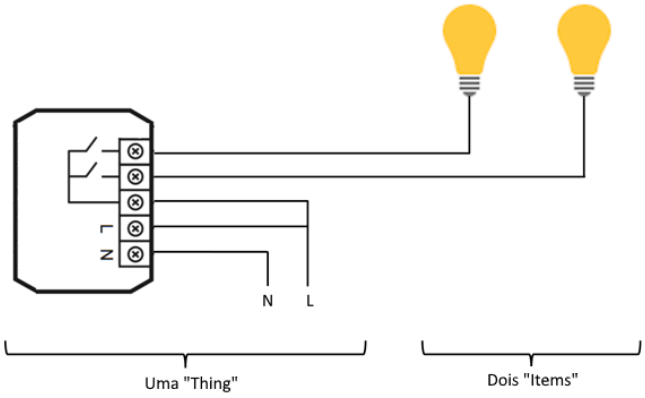


*Figura 9 – Exemplo de "Thing" e "Item"*

As *Things* são entidades que podem ser adicionadas fisicamente a um sistema. Estas não necessitam de ser apenas físicas, podendo também ser *webservices* e expõem as suas funcionalidades através dos *channels*

Os *Items* representam basicamente os dispositivos finais com os quais pretenderemos interagir. Os *bindings* podem ser encarados como adaptadores de software, isto é, são os elos de ligação entre os *items* e os dispositivos físicos. Abstraem também a forma como se comunica com o dispositivo deixando estes assuntos mais específicos para a *framework.*

Os *channels* podem ser ligados a múltiplos *items*, bem como os *items* pode ser ligados a múltiplos *channels*. A ponte entre as *things* e os *items* são os *links*. Um link é uma associação entre exatamente um *channel* e um *item*.

## 3.2 Diagramas de Sequência

De seguida ilustra-se sob a forma de diagramas de sequência os fluxos que aqui se foram detalhando, ou seja, a fase de armazenar os objetos, a fase da procura e, por fim, a fase de listagem dos objetos.

Conforme se pode ver na Figura 10, o utilizador abre a aplicação, ao carregar no botão de gravar ser-lhe-á mostrado um ecrã onde deverá preencher com alguns meta-dados. Posteriormente quando a imagem é capturada, esta é enviada para o Servidor Aplicacional via uma API REST, que por sua vez se encarregará de enviar para uma base de dados onde é persistida. No final é apresentado o identificador único dessa imagem ao utilizador, em caso de sucesso.

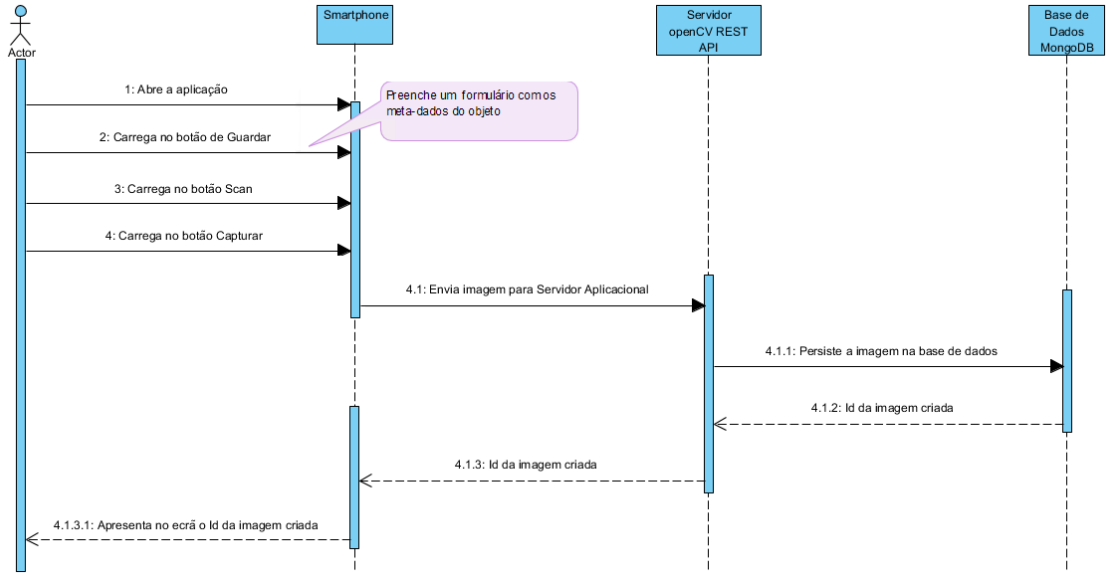


*Figura 10 – Diagrama de sequência – persistência*

A ligação entre a imagem e os objetos IoT, como já referido anteriormente, é feita recorrendo a tópicos MQTT. Quando a imagem é armazenada são enviados junto com a mesma meta dados, entre os quais o nome do tópico. É neste tópico que estará à escuta um subscritor (plataforma IoT), que com base no tópico saberá qual o dispositivo IoT que deverá controlar. Obviamente esta configuração da plataforma deverá ocorrer antes do armazenamento de qualquer imagem.

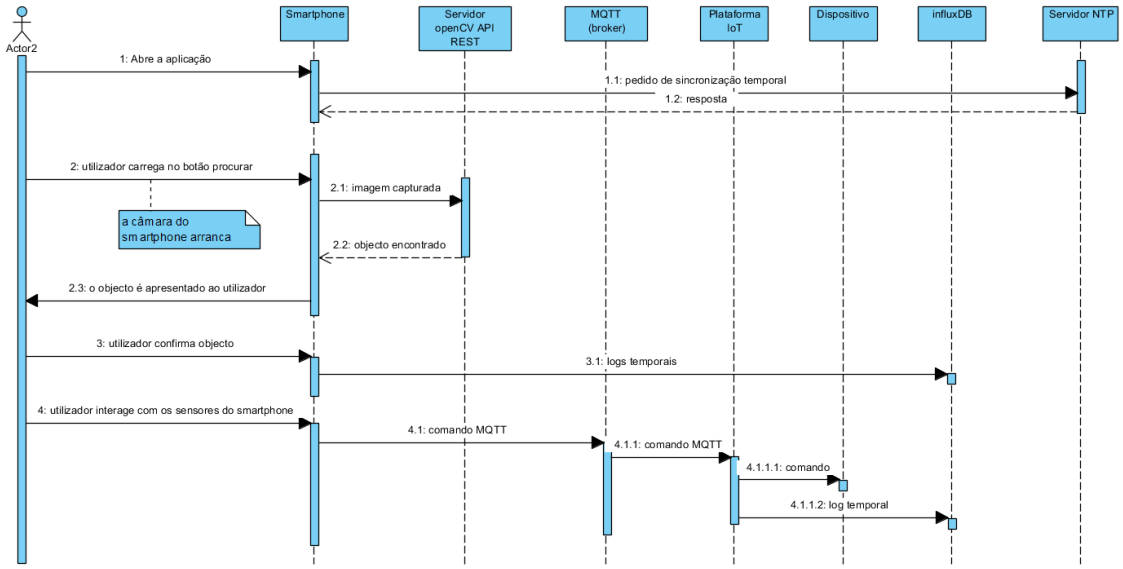


*Figura 11 – Diagrama de sequência – reconhecimento*

Na Figura 11, é explicado o processo do reconhecimento de imagens. Quando o utilizador abre a aplicação é apresentada a vista principal da aplicação e é desencadeado automaticamente o mecanismo de sincronização com um servidor NTP (*Network Time Protocol*). Este mecanismo ocorre sempre que o *smartphone* volta ao ecrã principal, por forma a garantir o máximo de precisão possível, uma vez que se pretende medir os tempos em cada uma das etapas e estas deverão ser na ordem dos milissegundos.

Quando o utilizador carrega no botão de procura (o botão intermédio), a câmara do *smartphone* deverá arrancar e é iniciado o fluxo de reconhecimento.

No caso de sucesso no reconhecimento, isto é, o algoritmo conseguiu identificar a imagem, esta é apresentada ao utilizador para este confirmar se esta corresponde ao dispositivo que pretende controlar. Se assim for, o utilizador através de gestos confirma a imagem e é levado para um ecrã onde o sensor que está associado à imagem fica disponível para interação com o utilizador, que por usa vez vai enviando via broker os comandos para a plataforma IoT, que por sua vez se encarregará de enviar comandos para os dispositivos. Caso contrário, o utilizador também através de gestos indica à aplicação que o objeto encontrado não se trata do objeto pretendido e o utilizador é levado para o ecrã principal.

Caso não haja sucesso no reconhecimento, isto é, o algoritmo não conseguiu identificar o objeto, o processo repete-se até que haja um reconhecimento efetivo. Este processo é automático e não requer intervenção do utilizador, ficando os dois componentes (*smartphone* e servidor aplicacional) a funcionar de forma autónoma, sempre a processar novas imagens capturadas e enviadas pelo *smartphone*. Mas este processo pode ser interrompido se for atingido um limite predefinido de tentativas ou o utilizador demonstre a intenção de desistência. Se o utilizador pretender desistir, deverá efetuar um gesto como por exemplo balançar o *smartphone* para a esquerda e direita num gesto brusco, indicando à aplicação a sua intenção de desistência.

Na Figura 12, podemos observar o fluxo de interação para listar as imagens armazenadas na base de dados. O utilizador abre a aplicação e carrega no botão de imprimir ou listar. O *smartphone* envia uma mensagem à API REST indicando que pretende obter a lista de imagens armazenadas na base de dados. Como estas já se encontram em memória, é então devolvida uma lista de todas as imagens que se encontram na base de dados juntamente

com os seus meta-dados, os quais são então apresentados ao utilizador num ecrã específico para o efeito.

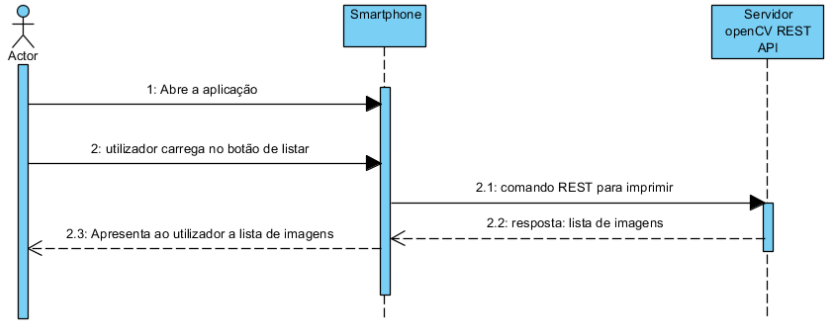


*Figura 12 – Diagrama de sequência – listar as imagens*

## 3.3 Fluxo de Eventos

Os eventos estão sempre associados a marcas temporais e ocorrem em determinados instantes do fluxo do reconhecimento. Por exemplo, quando se inicia o fluxo de reconhecimento, ocorre o primeiro evento (ANDROID_START), o qual é encapsulado numa estrutura de dados, uma lista de eventos. Esta estrutura é partilhada com a imagem que viajará pela rede até ao servidor e vice-versa, onde este lhe acrescentará outros eventos. Quando o servidor terminar as suas tarefas retorna a estrutura de eventos (com ou sem imagem) e ser-lhe-ão adicionados novos eventos. Este processo pode ser repetido até que haja condições para o terminar. Na Figura 13 pode ver-se o fluxo de eventos proposto para este estudo.

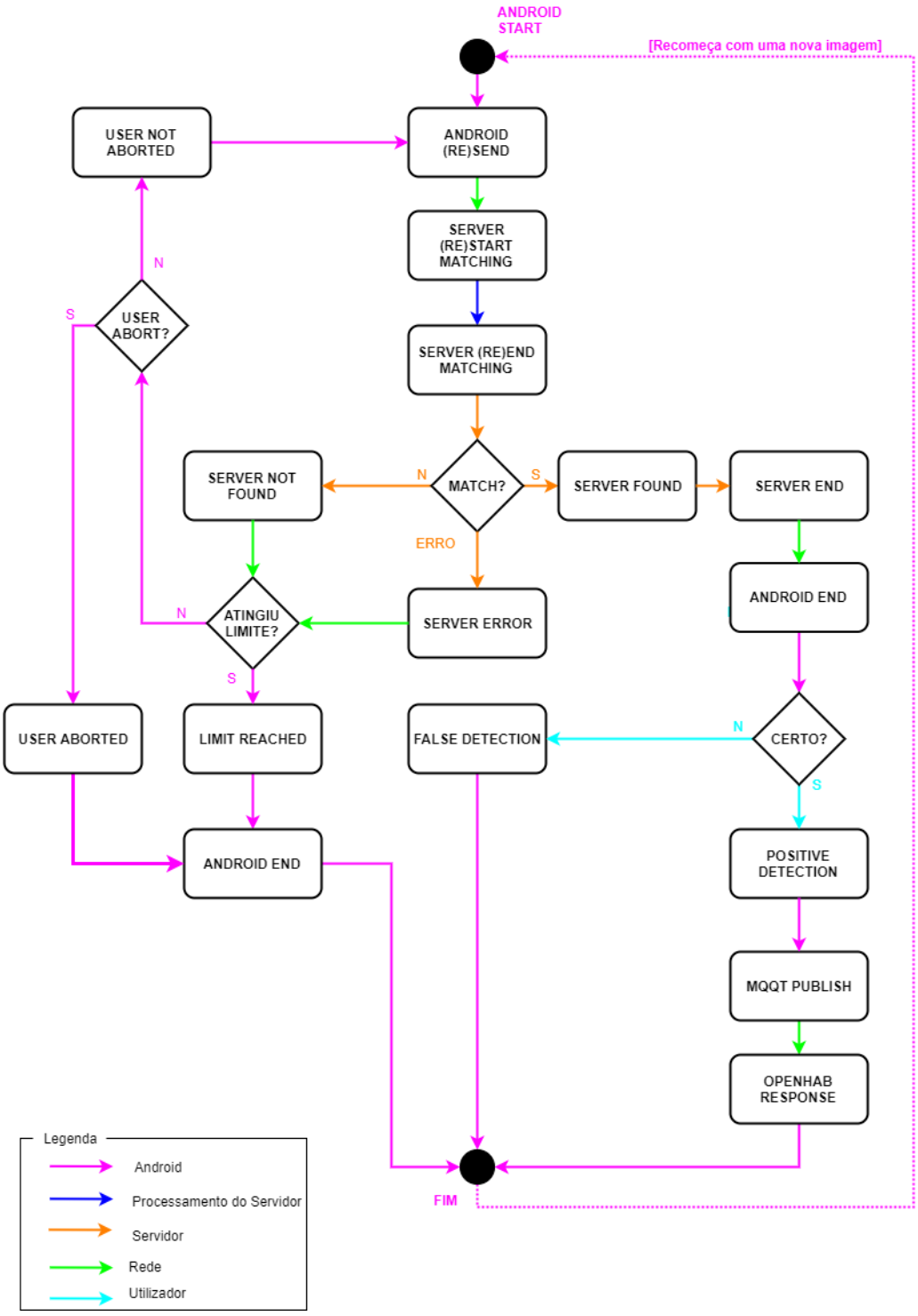


*Figura 13 – Fluxo de eventos*[2]

Resumindo, os eventos iniciam-se sempre no *smartphone*, são colocados numa lista e vão sendo adicionados novos eventos (tanto no lado do *smartphone* como do lado do servidor)

[2] O evento *OpenHAB Response* é enviado pelo openHAB aquando do envio de comandos para o dispositivo IoT

até que o processo termine, onde serão descarregados todos para uma base de dados temporal. A cada um estará sempre associado o selo temporal e é com base nestes que iremos analisar os resultados.

## 3.4 Comunicação entre Componentes

A infraestrutura de comunicação principal será uma ligação wireless (Wi-Fi). Sobre esta infraestrutura poderão assentar diversos protocolos, tais como HTTP REST, MQTT, mongodb, entre outros, conforme a Figura 5.

Os dispositivos ficarão numa rede privada e completamente separada de todo o resto dos elementos que constituem esta arquitetura. O único elo de ligação que estes terão com o mundo periférico será apenas pela plataforma IoT. Referimo-nos a redes próprias e com protocolos muito específicos no seio do IoT, tais como KNX, ZigBee, LoRA ou mesmo Z-wave, para citar os mais comuns. Digamos que a plataforma de IoT funcionará como *gateway* (ponto obrigatório de passagem) para a comunicação com estes dispositivos.

# Capítulo 4. **Implementação**

Com base na arquitetura anteriormente apresentada, irá detalhar-se nesta secção a solução escolhida. Isto é, de forma mais detalhada, será explicado por que razão foram escolhidas determinadas soluções em detrimento de outras.

Como já referido anteriormente, o intuito deste estudo é poder demonstrar que os dispositivos IoT, de uma casa inteligente poderão ser controladas pelos sensores de um *smartphone*, tendo-se procedido ao desenvolvimento de um protótipo, como forma de validar a arquitetura.

Assim, pretende-se com este capítulo descrever a solução escolhida para este fim. Temos um *smartphone* que capta imagens e manda guardá-las numa base de dados, através de uma API REST. Posteriormente e, de forma contínua, este envia imagens até fazer um *match* com uma imagem previamente armazenada. Quando for feito o match, sabemos também qual o sensor que deverá controlar o dispositivo, sendo o utilizador levado para um ecrã onde poderá interagir com o tal sensor. À medida que este vai recolhendo dados vai também enviando para a uma plataforma IoT que se encarregará por seu turno de comunicar com o dispositivo encontrado, fazendo assim este último reagir aos comandos do sensor.

Convém explicar desde já algumas noções e conceitos importantes, para que, mais adiante aquando da explicação pormenorizada tudo fique mais claro.

Item

Item deve ser encarado como o objeto, a "coisa" com a qual queremos interagir. Pode por exemplo ser uma lâmpada, uma ventoinha, uma tomada ou até uma persiana. Um item deve ser entendido como a *Thing*, na designação de *Internet of Things*. Portanto, doravante quando nos referirmos a items estaremos a falar de dispositivos controláveis pela plataforma IoT.

Objeto é o item com o qual queremos interagir. No contexto do openCV objeto significa a imagem do objeto com o qual queremos interagir e previamente armazenada na base de dados. Ou seja, corresponde a imagens previamente armazenadas na base de dados sobre as quais vamos fazer comparações até encontrar um reconhecimento válido.

A cena também é uma imagem do objeto com o qual pretendemos interagir, mas no contexto do openCV é-lhe atribuído outro significado. A cena é a imagem do objeto que enviamos para comparação contra os objetos previamente armazenados. Assim, temos objetos guardados (imagens que não sofrerão alterações ao longo da sua vida útil) e cenas, as quais serão sempre diferentes até encontrar um match. As cenas são imagens dos mesmos objetos, mas poderão ter ângulos diferentes, diferentes perspetivas e diferentes tipos de luz. São as cenas que são enviadas para o servidor para encontrar o objeto que nelas estará contido.

O código-fonte do projeto encontra-se num repositório público do Github acessível através do seguinte endereço: https://github.com/joaodavidbrito/home-automation.

De seguida, detalharemos cada componente com os detalhes da sua implementação.

## 4.1 Smartphone

Como plataforma mobile decidiu-se escolher o sistema operativo Android, mas qualquer outro poderia ter sido escolhido. O Android foi o eleito devido a alguns fatores, tais como:

- cumpre os requisitos essenciais, pois suporta a câmara e vários sensores disponíveis;
- implementação de uma aplicação móvel relativamente acessível e rápida;
- *know-how* anterior em programação nesta linguagem (Java) e no seu editor.

Para desenvolver a aplicação mobile utilizou-se o Android Studio da IntelliJ que é a ferramenta acreditada pela Google para o desenvolvimento atual de aplicações Android.

Em termos gerais criam-se *layouts*, que correspondem aos ecrãs, os quais por sua vez estão associados a classes Java (controladores dos layouts). Ou seja, temos as *views* (*layouts*) controladas por objetos Java (*activities*). Estes objetos possuem vários *listeners*, são métodos que reagem a eventos, como por exemplo o carregar de um botão numa *view*.

Portanto, na nossa aplicação temos 7 *activities* associadas às respetivas *views*:

- MainActivity – controla o ecrã principal;
- SaveActivity – responsável por tirar as fotos dos objetos;
- FindActivity – responsável por tirar as fotos das cenas;
- MetadataActivity – utilizada para gravar informação associada aos objetos, antes de serem enviados para a API REST;

- PrintActivity – utilizada para listar os objetos armazenados;
- SensorProximityActivity – utilizada para a interação com o sensor de proximidade;
- SensorRotationActivity – utilizada para a interação com o sensor de rotação.

A par com estes objetos houve a necessidade de criar outros objetos auxiliares:

- ImagesDto – objeto agregador da imagem e dos seus meta-dados;
- LogEvent – objeto utilizado para conter os eventos que serão posteriormente utilizados para interpretação dos resultados;
- LogTask – utiliza métodos assíncronos para persistir os eventos para uma base de dados temporal;
- MQTTHelper – esta classe é utilizada para todas as funções relacionadas com o MQTT, tais como estabelecer conexão, fechar a conexão, enviar, etc.;
- SingletonHelper – maioritariamente utilizada para guardar constantes e inicializar as preferências do sistema;
- TimeSyncTask – tarefa assíncrona para a sincronização permanente do relógio do Android, uma vez que o sistema operativo não permite definir o servidor NTP manualmente.

Existem ainda outros ficheiros que foram utilizados: menu_main.xml (utilizado para o menu na parte superior direita), strings.xml (contém os principais textos utilizados na aplicação).

De seguida, na Figura 14 ilustra-se o fluxo da aplicação em termos de *activities*.

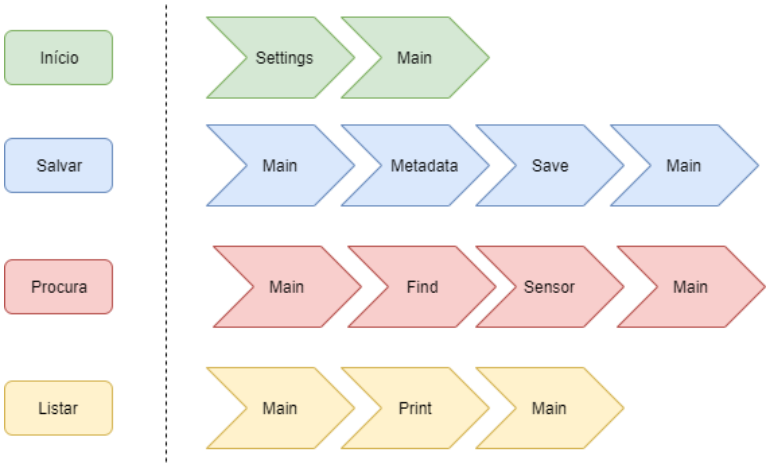


*Figura 14 – Fluxos principais da aplicação móvel*

Quando o utilizador entra pela primeira vez na aplicação é-lhe apresentada uma vista da associada à SettingsActivity. Aqui o utilizador tem de preencher vários campos, como sejam os endereços e portos dos vários sistemas envolvidos. Pode carregar no botão Load e, neste caso, os campos serão pré-preenchidos com valores constantes no ficheiro Strings.xml. O utilizador apenas poderá sair deste ecrã com os dados todos inseridos e validados, caso contrário, será sempre redirecionado para este ecrã sempre que tentar aceder à aplicação.

Uma vez salvas as configurações, retorna à MainActivity, onde poderá escolher uma de 3 opções: Salvar, Procurar e Listar, de acordo com o descrito na Tabela 5.

*Tabela 5 – Ações da vista principal da aplicação*

| Ações | Descrição |
|---|---|
| **Salvar**  | Desencadeia o fluxo para salvar um objeto na base de dados. Isto é, salva na base de dados a imagem do dispositivo com o qual queremos fazer match e interagir. |
| **Procurar**  | Esta opção desencadeia o fluxo do reconhecimento da cena. Ou seja, tantas imagens quanto as necessárias até ser encontrada uma correspondência entre a cena e objeto. |
| **Listar**  | É desencadeado o fluxo que permite listar todos os objetos armazenados na base de dados. |

Salvar

A primeira (no topo) conduz o utilizador à MetadataActivity, onde lhe é perguntado alguns dados sobre a imagem. Ao clicar em Scan, passa para a SaveActivity, onde lhe é apresentada a imagem capturada da câmara, onde o utilizador terá de apontar a câmara para o objeto que pretende controlar, sendo posteriormente a imagem enviada (de forma assíncrona) para o servidor externo (API REST). Após alguns instantes é retornado ao utilizador o ID com que a imagem foi armazenada na base de dados, terminando assim a primeira etapa do processo.

Na vista em que ao utilizador se solicita a introdução dos meta-dados (vide Figura 15), este deverá definir:

1. A descrição do objeto – consiste numa descrição genérica do objeto;
2. O nome do tópico – este nome é essencial estar igual ao que está previamente configurado na plataforma e no broker, pois este nome será enviado para a plataforma juntamente com o comando de interação através do broker, como veremos mais adiante.
3. O tipo de sensor associado ao objeto. Neste estudo apenas se utilizou o sensor de proximidade e de rotação, a título exemplificativo, mas poderia utilizar-se qualquer sensor disponível no *smartphone*, desde que fizesse sentido para utilizar com o objeto em causa.

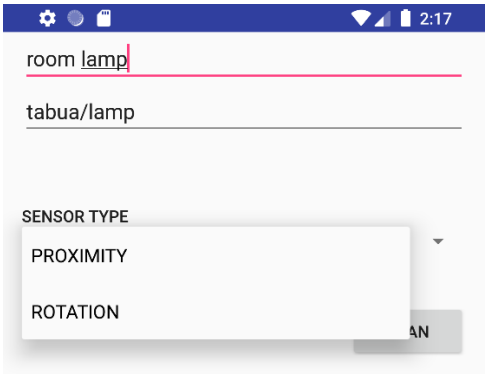


*Figura 15 – Ecrã dos meta-dados*

Esta tarefa pode e deve ser repetida tantas vezes quantos os objetos que queremos controlar.

Procurar

O utilizador carrega na opção do meio e imediatamente desencadeia a procura ativa de imagens, preconizado pela FindActivity. Esta *activity* é responsável por capturar imagens em tempo real e enviá-las para o servidor (também de forma assíncrona). Caso não haja um reconhecimento efetivo, este processo repete-se.

O servidor por sua vez responde com um uma de três possibilidades: *Found, Not Found* ou *Error*. Se a resposta for positiva (*Found*) então o processo avança para a *activity* do sensor associada ao objeto detetado. Nessa fase é mostrada a imagem do objeto encontrado ao utilizador aguardando que este abane vigorosamente o telefone

lateralmente transmitindo à aplicação de que se trata de um falso positivo ou se o balançar vigorosamente para cima e para baixo de que se trata de um verdadeiro positivo. Mais adiante veremos o que estes nomes significam.

De imediato, após o feedback do utilizador, os sensores associados à *activity* ficam disponíveis e à escuta de alterações, alterações essas que serão enviadas para a classe MQTTHelper, encarregue de as enviar para o servidor MQTT para interagir com o objeto. Caso o utilizador acene vigorosamente o telefone lateralmente, fará com que volte à *activity* principal.

Existe ainda uma possibilidade que ficou por referir. O utilizador pode decidir cancelar o processo de procura a qualquer momento. Ou caso o servidor não consiga fazer *match* das sucessivas imagens que lhe vão chegando, definiu-se um máximo de 50 tentativas. Caso uma destas duas coisas ocorra durante o processo de procura, o processo volta à *ativity* principal, dando conta disso mesmo ao utilizador sob a forma de uma *tooltip*.

Listar

Por último, caso o utilizador carregue na última opção, a aplicação desencadeia um pedido ao servidor, o qual retorna a lista de todas as imagens previamente armazenadas, juntamente com o identificador único, e a descrição de cada imagem, apresentada sob a forma de lista, pela *PrintActivity*.

Importa realçar que todos os processos que dependem de operações de rede de tráfego de dados, são assíncronos. Isto é, a interface principal com o utilizador não bloqueia, dando ao utilizador uma sensação maior de natural fluidez. Essas tarefas de rede, processam-se em *threads* de *background* e quando terminam desencadeiam outras ações. O utilizador nem se apercebe, mas por trás está a acontecer muita coisa e quando termina é que o utilizador é levado para outras vistas.

## 4.2 Armazenamento em Base de Dados

No que à base de dados diz respeito, optou-se por No-SQL do tipo mongoDB, por questões de simplicidade de implementação e manutenção. Também por simplicidade optou-se por utilizar uma base de dados alojada em *cloud*. Isto permitiu não perder demasiado tempo em configurações e instalações de uma base de dados local. Não sendo

o foco principal do que se pretendia demonstrar, será razoável optar por uma solução No-SQL em *cloud* para armazenar as imagens.

Com esta solução, podemos resumir alguns dos principais benefícios que podemos encontrar:

- Escalabilidade instantânea
- alta disponibilidade via *auto-failover*
- suporte de encriptação
- conexões SSL
- *backups* e reposições sem custos acrescidos.
- edição de documentos via web GUI, executando consultas e visualizando o resultado também na GUI da web e em forma de tabela.

Esta permite facilmente armazenar imagens e seus meta-dados, bem como permitiu evoluir a sua estrutura de dados sem grande esforço, por se tratar de uma base de dados não relacional. Isto é, sempre que foi preciso alterar a estrutura de dados, não foi necessário modificar tabelas (que nestes casos se denominam de documentos).

Convém clarificar que a base de dados em cloud é utilizada apenas na fase de gravação de novas imagens, não prejudicando o desempenho na fase do reconhecimento. Pois quando a aplicação (API REST) arranca pela primeira vez vai buscar todos dados à base de dados (incluindo as imagens) a guarda-os em memória, com as imagens já pré-processadas (descritores). Também sempre que é adicionada uma nova imagem (independentemente do estado da aplicação) é armazenada na base de dados sendo também a memória atualizada.

Os objetos são guardados na forma de JSON o que na prática significa que estamos apenas a guardar texto (com exceção das imagens propriamente ditas). A título de exemplo, pode ver-se como um objeto é armazenado em mongoDB:

```
{
    "_id": {
        "$oid": "5d17773d7f28483f543d75b1"
    },
    "className": "pt.joaobrito.ha.homeautomation.collection.Image",
    "encodedImage": "<Binary Data>",
    "name": "hue/lamp",
    "action": "ROTATION",
    "description": "hue lamp"
}
```

No exemplo acima pode-se ver que é gerado um ID, é armazenada a imagem propriamente dita em formato binário (*blob*), o nome (o qual corresponde ao nome do tópico do broker MQTT). O tipo de ação a que este objeto ficará associado (entenda-se tipo de sensor) e descrição, a qual corresponde a uma mera descrição do objeto.

Este é o tipo de dados armazenado quando se carrega no botão salvar. Estes dados são depois devolvidos ao *smartphone*, com a imagem, aquando de um reconhecimento. Desta forma o utilizador fica a saber qual o objeto que foi encontrado e a aplicação móvel reencaminhará para a *view/acivity* correspondente ao sensor do objeto.

É também com estes dados que serão enviados comandos MQTT para o tópico correspondente, através do nome, o qual não é mais do que o nome do tópico. Assim, fica claro que o nome da imagem é o nome do tópico e, portanto, haverá um tópico, para cada objeto. Isto não significa que cada objeto terá um e apenas um tópico, o que isto significa é que podemos ter várias imagens do mesmo objeto (de forma a aumentar as hipóteses de reconhecimento) e todas elas têm de estar associadas ao mesmo tópico. Ou seja, é o tópico que define ou instrui a plataforma qual o objeto com o qual se vai interagir.

Na imagem seguinte apresenta-se o diagrama de entidades da estrutura de dados utilizada para guardar as imagens e seus dados.

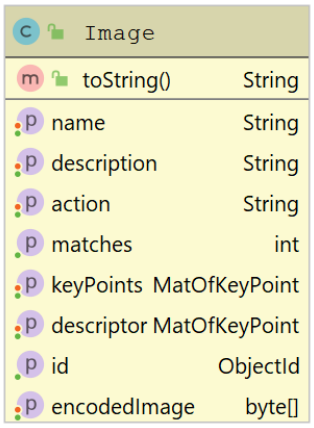


*Figura 16 – Diagrama UML de entidades*

De referir que foram utilizados dois tipos de objetos para esta prova de conceito: cartas e objetos reais. Como foram efetuados vários ensaios com ambos, foi necessário fazer um *dump* da base dados para importar e exportar os dados para backup.

## 4.3 Servidor Aplicacional / openCV

Por razões de segurança e também de boas práticas optou-se por não conectar diretamente a aplicação móvel à base de dados. Assim, surgiu a necessidade de uma aplicação externa que se encarregará de executar esta tarefa. Ou seja, a aplicação invoca serviços de uma aplicação externa e esta responde de acordo com a lógica de negócio que lhe estiver associada.

Esta aplicação foi desenvolvida em Java 8, recorrendo à *framework* SpringBoot. É uma aplicação que expõe uma API REST. Desta, destacam-se dois *endpoints*, um para salvar as imagens na base de dados (*save*) e outro para ir buscar todas as imagens armazenadas na base de dados e posteriormente processá-las e compará-las com a imagem base (*find*).

Para tal, faz uso também a uma ferramenta externa para processamento de imagem, o openCV. O openCV é uma ferramenta em código aberto para tratamento de imagem, sendo que para este efeito apenas se utilizou uma pequena parte do que esta biblioteca tem para oferecer no campo do tratamento e processamento de imagens. Em termos de algoritmo utilizado para fazer o reconhecimento das imagens foi o SURF. Mais adiante explica-se mais pormenorizadamente o seu funcionamento.

O ponto de entrada na aplicação é assegurado por um controlador: *ImageController*. Aqui são definidos todos os *endpoints* da nossa API REST exposta por este controlador.

Refira-se ainda que é no controlador que se registam todas as marcas temporais associados a todas as fases do processo relacionadas com o servidor.

No diagrama seguinte podemos observar o diagrama de classes das classes principais presentes no servidor aplicacional.

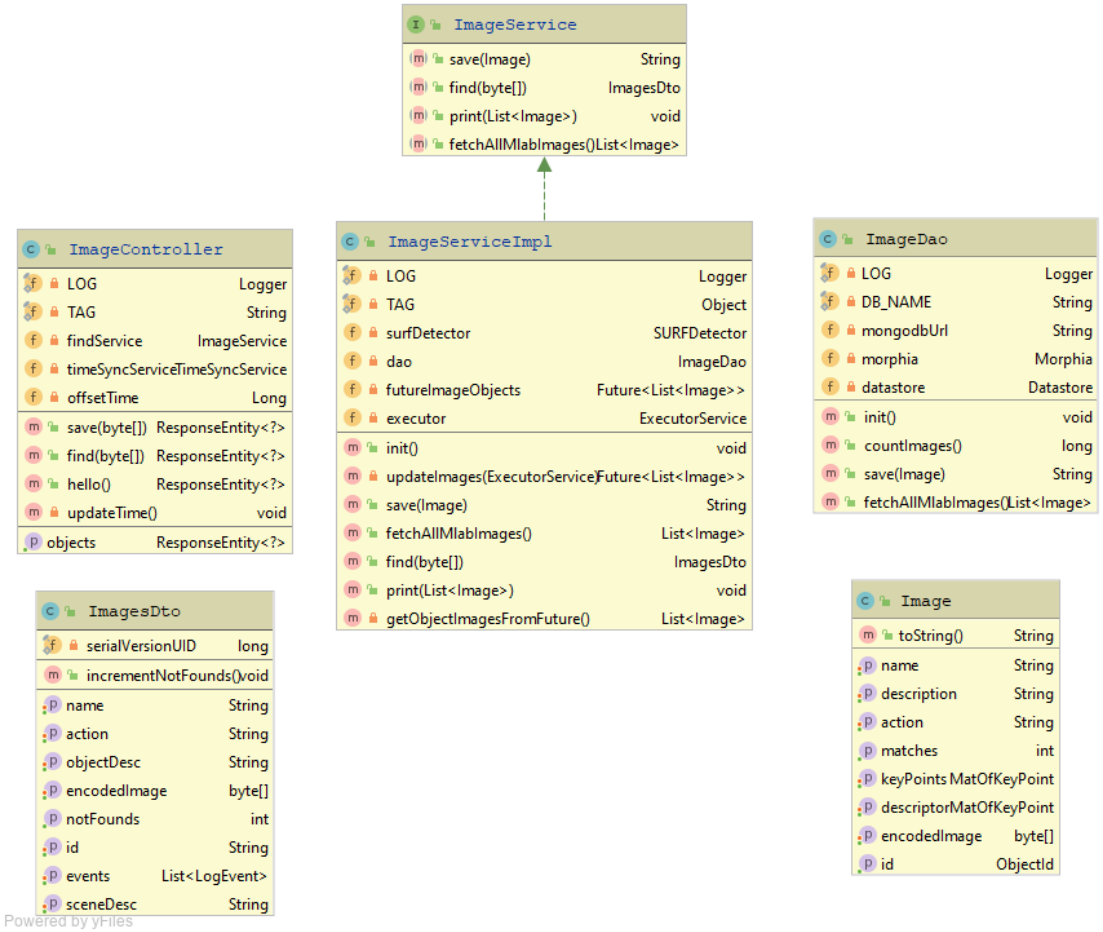


*Figura 17 – Diagrama de classes (servidor aplicacional)*

Save

É invocado o método *save* presente na classe que fornece os serviços, o qual garante a sua persistência na base de dados. Quando este método termina, se tudo correr bem, retorna o identificador único com que a imagem ficou registada na base de dados, o qual é retornado ao *smartphone* na forma de *tooltip* e um código 200 OK, que significa que tudo correu conforme esperado.

*Find*

À semelhança do método anterior, este também recebe um *array* de bytes, o qual é transformado na sua representação original, um objeto Java com a imagem (cena) e todos os eventos que estão associados a esta pesquisa. Invoca então o método *find* nos serviços, o qual se encarrega de toda a lógica de negócio associada à procura e *match* das imagens.

Já na classe dos serviços, transforma a imagem da cena nos seus descritores e vai então compará-la com todos os descritores dos objetos previamente armazenados em memória, através do algoritmo SURF do openCV. Se for encontrado um *match*, reconstrói o objeto original adicionando-lhe este no lugar da cena, caso contrário devolve *null* ao seu chamador.

Uma vez no controlador, novamente é reconstruído o objeto recebido por este inicialmente, mas sem a imagem, apenas com as marcas temporais atualizados, caso não haja *match*. Caso seja encontrado um *match*, todos os dados associados à imagem dizem agora respeito ao objeto encontrado, bem como são atualizadas as marcas temporais.

*Print*

Este método não recebe quaisquer parâmetros e limita-se a recolher os objetos e enviá-los para o *smartphone*, juntamente com o seu UUID e descrição. A lista é ordenada pelo selo temporal de cada imagem.

## 4.4 MQTT Server

Antes de proceder ao armazenamento de qualquer imagem, deve ser configurado o broker MQTT. Neste caso, o broker ficou instalado no Raspberry Pi [42] onde corre igualmente a plataforma IoT. Como implementação do protocolo MQTT utilizámos o software Eclipse Mosquito.

Este passo é essencial, quer para dar o nome à imagem (tópico) quer para a plataforma IoT, saber quais os tópicos que deseja subscrever, como veremos no próximo ponto.

Basicamente instala-se, define-se um *username* e *password* e fica pronto a utilizar, à espera de subscritores de tópicos (plataforma IoT) e de publicadores para esses mesmos tópicos (sensores do *smartphone*) por forma às mensagens serem enviadas para os destinos corretamente.

## 4.5 Plataforma IoT

A plataforma IoT eleita foi o openHAB. Já havia conhecimento sobre a mesma e preenchia todos os requisitos para o que pretendíamos demonstrar.

Tínhamos ao dispor 2 tipos de *items* bem distintos: um trabalhava com a tecnologia KNX e o outro trabalhava com a tecnologia sem fios Philips HUE (ZigBee). Ou seja, do lado

do KNX, tínhamos uma lâmpada, uma ventoinha e uma tomada, conforme Figura 18. Do outro tínhamos uma lâmpada, a qual pode ser regulada a sua intensidade, o que permite explorar, por exemplo, o sensor de rotação para diminuir ou aumentar a sua intensidade.

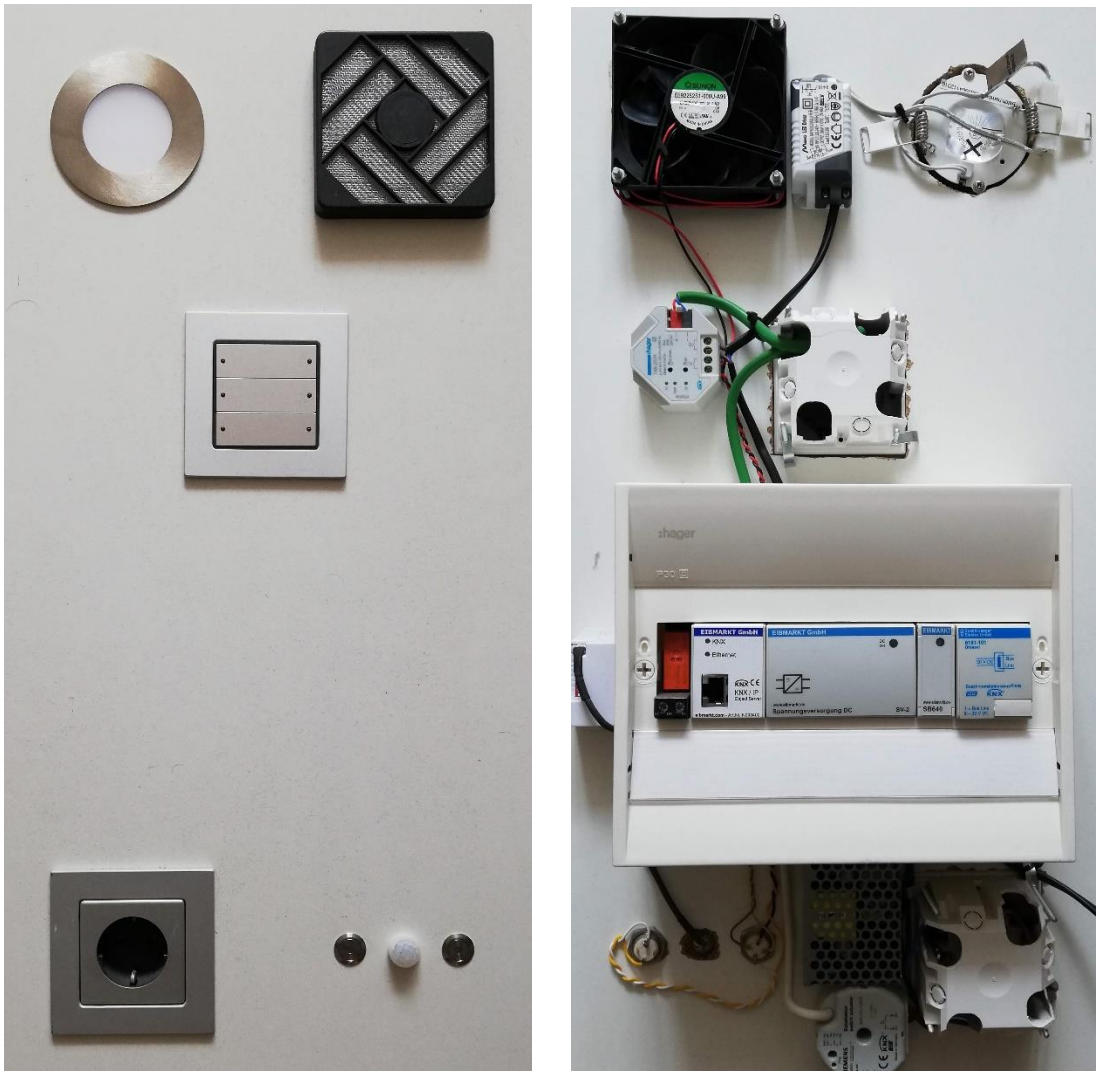

*Figura 18 – Dispositivos KNX controlados pelo openHAB*

Na imagem acima, para suportar diversos dispositivos KNX, utilizou-se uma tábua que os agrega num único suporte. Mais uma vez estamos perante um protótipo e não de um caso real.

Mas na Figura 18, podemos ainda observar a parte de trás desse suporte onde estão presentes as ligações e controladores KNX que foram alvo de algumas configurações prévias e que também estiveram de ser alinhadas com as configurações da plataforma. Estas configurações utilizam um software proprietário da KNX (ETS4) o qual permite

definir endereços internos para os dispositivos e agrupá-los em grupos, por exemplo, de forma a que os atuadores KNX saibam internamente qual ou quais dispositivos que devem controlar, conforme é ilustrado na Figura 19. Resumidamente e de forma muito simplificada, definem-se as configurações internas do KNX e os *endpoints* externos para que uma plataforma IoT saiba como comunicar com os dispositivos KNX.

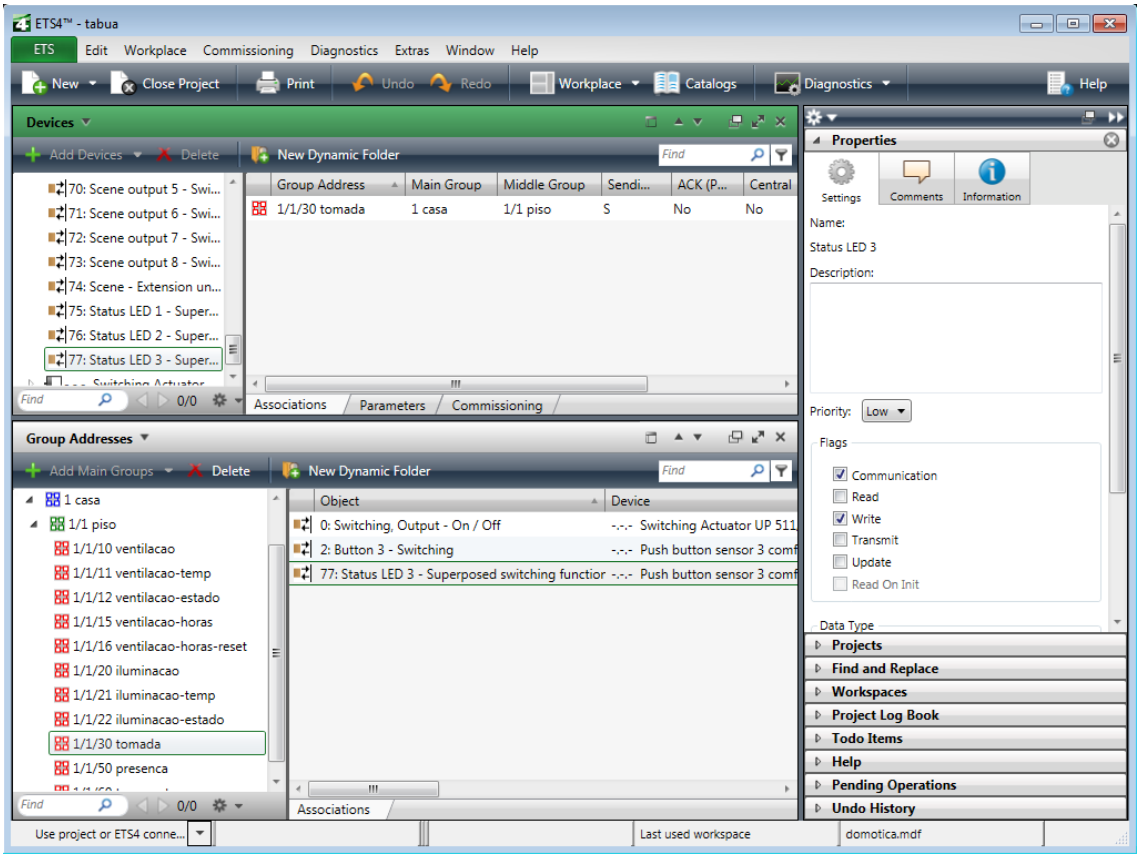


*Figura 19 – Software proprietário da KNX para configuração dos dispositivos IoT*

## Bridges

Para o openHAB poder comunicar com os *items* acima teve de configurar-se 2 bridges distintas, uma para o protocolo KNX e outra para o protocolo ZigBee (HUE):

```
Bridge knx:ip:bridge "KNX/IP Bridge" @ "KNX" [
    ipAddress="192.168.0.108", // knx ip address
    portNumber=3671,
    localIp="192.168.0.113", // openhabian ip address
    type="TUNNEL",
    readingPause=50,
    responseTimeout=10,
    readRetriesLimit=3,
    autoReconnectPeriod=1,
    localSourceAddr="1.1.0"
] {
    Thing device generic "Switch Light 1" @ "KNX" [
        address="1.1.10",
        fetch=true,
        pingInterval=300,
        readInterval=5
    ] {
        Type switch       : ventilacao1     "Vent1"     [ ga="1.001:1/1/10" ]
        Type switch       : iluminacao1     "Light1"    [ ga="1.001:1/1/20" ]
        Type switch       : tomada1         "Tom1"      [ ga="1.001:1/1/30" ]
    }
}

Bridge hue:bridge:1 "HUE BRIDGE" @ "HUE" [ ipAddress="192.168.0.106" ] {
    0100 bulb1 [ lightId="1" ]
}
```

*Figura 20 – Configurações das bridges*

Estas configurações instruem o openHAB a saber comunicar com os dois protocolos acima referidos, para tal temos de lhe indicar os IPs, para além de outros parâmetros necessários, de acordo com a documentação respetiva.

*Items*

Mas ainda nos falta definir os *items* e os respetivos canais, como se pode observar na Figura 21. Aqui podemos ver que existem duas vias distintas de comunicação, uma por omissão utilizando o canal e outra utilizando o protocolo MQTT. É nesta última que temos especial interesse, pois será através destas configurações que poderemos interagir com os dispositivos através de um modo "não convencional".

```
// HUE
Dimmer  Light1_Dimmer   "Lamp"  { channel="hue:0100:1:bulb1:brightness", mqtt="<[mosquitto:hue/lamp:command:default]"}
Switch  Light1_Toggle   "Lamp"  { channel="hue:0100:1:bulb1:brightness", mqtt="<[mosquitto:hue/switch/lamp:command:default]"}

// KNX
Switch  lamp            "Light [%s]"    <light>         {channel="knx:device:bridge:generic:iluminacao1", mqtt="<[mosquitto:tabua/lamp:command:default]"}
Switch  vent            "Vent [%s]"     <fan>           {channel="knx:device:bridge:generic:ventilacao1", mqtt="<[mosquitto:tabua/vent:command:default]"}
Switch  toma            "Toma [%s]"     <poweroutlet>   {channel="knx:device:bridge:generic:tomada1", mqtt="<[mosquitto:tabua/tomada:command:default]"}
```

*Figura 21 – Configurações dos Items*

InfluxDB

Para que após recebidos os comandos e retransmitidos para os dispositivos se possa saber quando tal ocorreu, foi necessário configurar no openHAB um ficheiro específico para o efeito, como o da Figura 22. Esta funcionalidade também teve de ser habilitada nas configurações do openHAB.

```
Items {
    vent, toma                              : strategy = everyChange
    Light1_Dimmer, Light1_Toggle, lamp      : strategy = everyChange
}
```

*Figura 22 – Configurações da base de dados influxDB*

Broker MQTT

Por último, teve ainda de se configurar o broker MQTT no openHAB, conforme se apresenta na Figura 23. Aqui também foi necessária previa habilitação nas configurações do openHAB.

```
mosquitto.url=tcp://127.0.0.1:1883
mosquito.clientId=openhab2
mosquitto.user=openhabian
mosquitto.pwd=openhabian
mosquitto.qos=2
mosquitto.async=false
```

*Figura 23 – Configurações Eclipse Mosquito*

## 4.6 Implementação para Testes

Como pretendíamos medir todos os passos do fluxo e salvar todas as marcas temporais, tínhamos de garantir que todos os relógios estivessem o mais sincronizados possível. Para tal dotou-se quer o servidor quer a aplicação móvel de um sistema que sincroniza os relógios com um servidor NTP (*Network Time Protocol*). No caso da aplicação Android, esta sincronização é efetuada sempre que o utilizador retorna à *activity* principal, mostrando sob a forma de *tooltip* o desfasamento atual para a hora atómica do servidor NTP.

Isto significa que não corrigimos, do lado do Android, o relógio, mas em todas as marcas temporais, é incrementado ou decrementado o diferencial entre os dois. No caso do servidor a sincronização é periódica (a cada 3600s), mas nem seria necessário, pois neste caso temos acesso a todas as configurações de NTP, via sistema operativo.

Base de dados temporal - InfluxDB

Por se tratar de um protótipo, precisávamos de colocar todas as marcas temporais numa base de dados. Utilizou-se InfluxDB como base de dados temporal, bem como outra aplicação (grafana) para mostrar esses mesmos dados em tempo real.

Estas medições apenas foram recolhidas no processo da procura. E estes dados acompanhavam as sucessivas tentativas de reconhecimento da imagem, sob a forma de eventos. Na Figura 24 pode ver-se a estrutura de dados utilizada para armazenar os eventos.

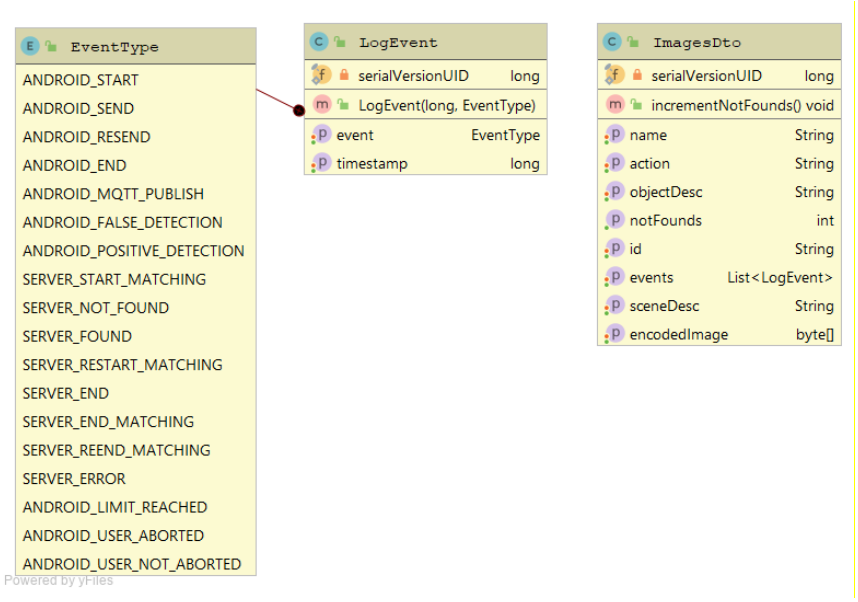


*Figura 24 – Diagrama UML das classes relacionadas aos eventos*

Estes objetos foram transmitidos entre o *smartphone* e o servidor (e vice-versa) até que o processo terminasse, onde mesmo antes de passar o utilizador para a vista em que este teria de dar o seu feedback, eram descarregados em modo *batch* para o InfluxDB.

Igualmente, a cada gesto ou interação com os sensores, era de imediato, registado um selo temporal no influxDB.

# Capítulo 5. **Análise e Discussão dos Resultados**

No presente capítulo iremos abordar a análise dos testes efetuados. Procedeu-se a diversos tipos de testes, para daí podemos medir, analisar e retirar conclusões. Deu-se especial enfase aos testes funcionais e aos testes de desempenho, como mais adiante se verá.

O objetivo dos testes foi o de provar que conseguimos responder com sucesso aos problemas a que nos propusemos resolver no início do presente estudo.

Com os testes funcionais, pretendeu-se medir os tempos entre diversos tipos de eventos, e entender possíveis pontos de otimização e assim reduzir alguns tempos para que o processo final de reconhecimento do dispositivo ficasse dentro de limites aceitáveis. Ainda com os testes funcionais pode comprovar-se que os eventos recolhidos cumpriram com os requisitos e ocorreram na ordem correta, de acordo com o desenho da arquitetura.

Nos testes de desempenho, pode observar-se o comportamento geral da solução variando algumas variáveis ou condições, tais como mudança do ângulo de incidência e diferentes tipos de objetos, para daí podermos medir a eficácia / eficiência como a percentagem de falsos positivos, o número de tentativas necessárias, etc.

## 5.1 Testes Funcionais

Com este tipo de testes pretendia-se provar que a solução final correspondia aquilo que havia sido delineado em termos de arquitetura. Pode constatar-se que os eventos foram os previstos e que ocorreram na ordem prevista. Entenda-se eventos como pontos chave no fluxo normal da solução; pontos chave esses que permitiram verificar o comportamento geral da solução à posteriori.

Os testes, para cada objeto, foram realizados de acordo com o seguinte procedimento:

1. O utilizador abre a aplicação;
2. O utilizador descreve o que pretende reconhecer;
3. O utilizador carrega no botão procurar e aponta a câmara do *smartphone* para o objeto a reconhecer;
4. A câmara é iniciada e começa a capturar a imagem, a qual é enviada para a API REST, ficando à espera de uma resposta;
5. O servidor recebe a imagem enviada pelo *smartphone*;

6. O servidor inicia o processo de reconhecimento e aplica o algoritmo para reconhecimento da imagem recebida contra todas as armazenadas em memória (pré-carregadas da base de dados);
7. O servidor termina o processo de reconhecimento;
8. O servidor envia uma resposta de volta para o *smartphone*;
9. O *smartphone* avalia o tipo de resposta;
    a. Caso a resposta seja positiva (se houve match), o fluxo prossegue para o ponto 10;
    b. Caso a resposta seja negativa (não houve match), o fluxo retorna ao ponto 4 até ser encontrada uma correspondência ou se atingir um limite máximo de iterações.
10. O *smartphone* leva o utilizador para uma vista onde pede ao utilizador para avaliar se se trata de um verdadeiro ou falso positivo através de gestos capturados por um sensor do *smartphone*;
11. O *smartphone* volta a avaliar o feedback do utilizador;
    a. Caso seja falso positivo o fluxo termina e o utilizador tem de iniciar o fluxo de teste no ponto 3;
    b. Caso seja um verdadeiro positivo, o fluxo prossegue no ponto 12;
12. O utilizador é levado para uma vista onde poderá interagir com o sensor associado ao objeto encontrado;
13. A cada interação com o sensor o *smartphone* envia informação recolhida pelo sensor através da comunicação MQTT para a plataforma;
14. A plataforma recebe os comandos enviados pelo *smartphone* e envia-os para o dispositivo correspondente (sabe através do tópico onde recebe as mensagens);
15. A plataforma regista cada comando na base de dados influxDB.

Os eventos referidos anteriormente podem observar-se na Figura 25, tais como o momento em que é iniciado um teste (Android Start), passando pelo momento em que o utilizador dá o feedback sobre a veracidade do match (*Positive Detection* ou *False Detection*), entre outros. A verde encontram-se os eventos ocorridos no *smartphone*, a laranja aqueles que ocorreram no servidor aplicacional e a roxo aqueles que representam a resposta dos dispositivos. Encontram-se na ordem temporal correta, mas o espaçamento entre eles não corresponde à realidade, sendo sim, uma mera representação gráfica da ordem em que ocorrem.

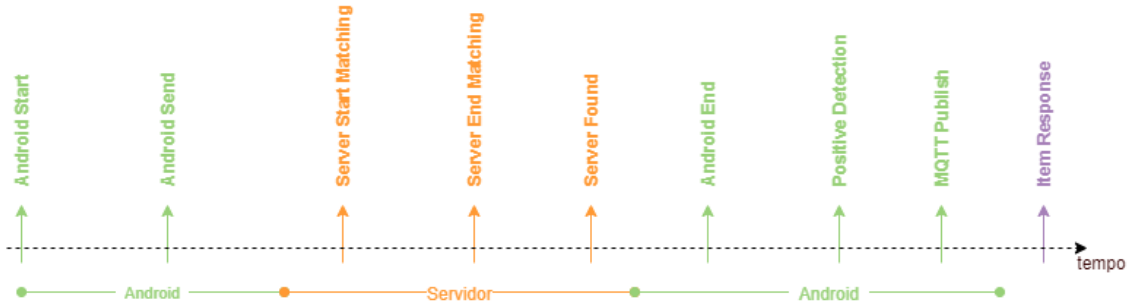


*Figura 25 – Fluxo temporal dos eventos*

Para uma melhor compreensão de que eventos temos vindo a referir, a lista de todos os eventos é detalhada na Tabela 6, com uma breve descrição sobre quando o mesmo ocorre. Os nomes dos eventos são exatamente os mesmos utilizados nas aplicações (móvel e servidor aplicacional).

*Tabela 6 – Explicação / detalhe dos eventos*

| **Evento** | **Detalhe do Evento** |
|---|---|
| ANDROID_START | Ocorre quando o utilizador carrega no botão de procurar na MainActivity |
| ANDROID_SEND | Ocorre quando a aplicação envia a imagem para o servidor pela primeira vez |
| ANDROID_RESEND | Ocorre quando o fluxo é recomeçado após não ter sido detetada qualquer correspondência entre cena e objeto |
| ANDROID_END | Ocorre quando o fluxo automático (sem intervenção humana) termina. É também neste momento que são enviados todos os eventos para a base de dados influxDB |
| ANDROID_MQTT_PUBLISH | Ocorre quando é enviada uma mensagem para a plataforma IoT: openHAB |
| ANDROID_FALSE_DETECTION | Ocorre quando o utilizador dá feedback negativo sobre a imagem (o objeto detetado não corresponde à cena) |

| | |
|---|---|
| ANDROID_POSITIVE_DETECTION | Ocorre quando o utilizador dá feedback positivo sobre a imagem (o objeto detetado corresponde à cena |
| SERVER_START_MATCHING | Ocorre quando o servidor inicia o processo de reconhecimento |
| SERVER_RESTART_MATCHING | Ocorre quando o servidor reinicia o processo de reconhecimento |
| SERVER_END_MATCHING | Ocorre quando o servidor termina o processo de reconhecimento (sempre após o evento SERVER_START_MATCHING) |
| SERVER_REEND_MATCHING | Ocorre quando o servidor termina o processo de reconhecimento, após uma ou mais tentativas de reconhecimentos falhadas |
| SERVER_NOT_FOUND | Ocorre quando o servidor não encontrou qualquer correspondência entre a cena e o objeto |
| SERVER_FOUND | Ocorre quando o servidor encontrou correspondência entre cena e objeto |
| SERVER_END | Ocorre quando o servidor termina o processo. Apenas e só se foi encontrada uma correspondência entre cena e objeto (independentemente se for um positivo ou falso reconhecimento) |
| SERVER_ERROR | Ocorre quando, por qualquer razão, ocorre um erro no servidor |
| ANDROID_LIMIT_REACHED | Ocorre quando é atingido o limite de sucessivas tentativas de reconhecimento |
| ANDROID_USER_ABORTED | Ocorre quando o utilizador aborta o processo de reconhecimento |
| ANDROID_USER_NOT_ABORTED | Ocorre antes de um reinício de tentativa de reconhecimento |

Estes eventos foram todos registados numa base de dados temporal (influxDB) e como tal tiveram de ser extraídos para que fosse possível a sua análise. Como esta tarefa era repetitiva e exigia algum esforço para capturar os dados do influxDB e passá-los para

uma folha de cálculo para depois extrair tabelas de resultados e gráficos, foi desenvolvida uma nova ferramenta em nodeJs para automatizar este processo, conforme mostra a Figura 26.

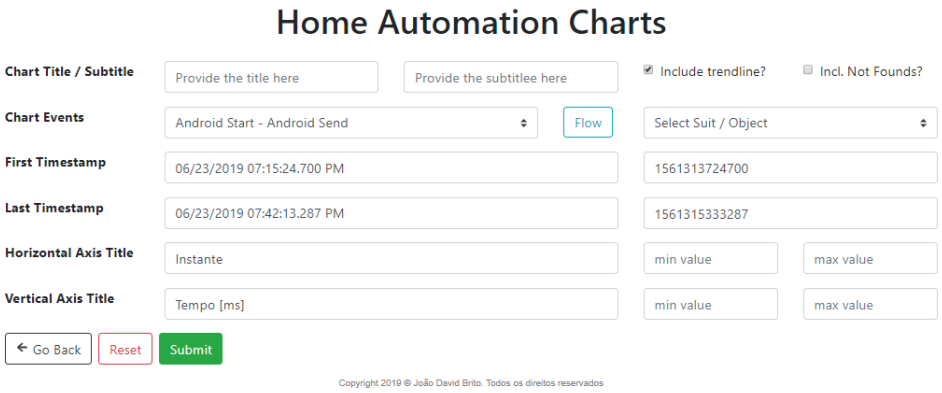


*Figura 26 – Ferramenta de automatização dos resultados*

Colocando os campos obrigatórios, os eventos pretendidos, bem como o dados temporais em que esses eventos ocorreram, são-nos disponibilizados gráficos e tabelas com alguns dados para podermos analisar, tais como os valores mínimos e máximos, a média aritmética, a linha de tendência, a indicação se foi ou não encontrada correspondência, entre muitos outros dados.

Os testes foram realizados em condições ótimas de iluminação, recorrendo-se a objetos planos com algum contraste em termos de padrões internos, conforme Figura 27. À esquerda podemos observar o objeto previamente guardado e armazenado e do lado direito a imagem desse objeto que pretendemos reconhecer. As linhas a verde na imagem, são uma representação gráfica dos *keypoints* encontrados pelo algoritmo de reconhecimento de imagens. Isto é, correspondem aos *keypoints* relevantes nas duas imagens que foram considerados idênticos. Mostra-se, a título de exemplo, todos os *keypoints* identificados para esse mesmo objeto.

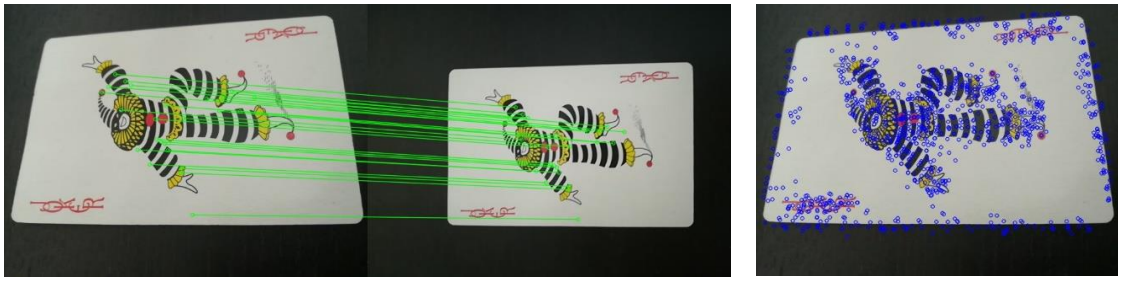

*Figura 27 – Exemplo de match (lado esquerdo) e keypoints (lado direito)*

A linha temporal dos eventos ocorridos é mostrada na Figura 28.

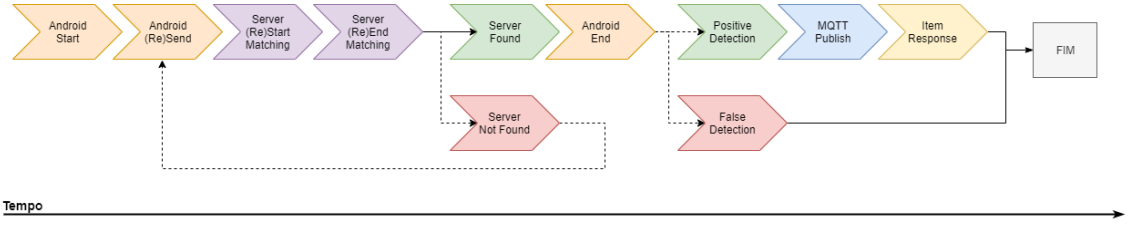


*Figura 28 – Eventos ao longo do processo de reconhecimento*

A Tabela 7 mostra um exemplo de ocorrência de eventos no tempo para um teste e, pode confirmar-se que os mesmos ocorrem na ordem correta, de acordo com o preconizado pela arquitetura.

*Tabela 7 – Eventos ao longo do tempo*

| Tempo (ms) | ΔT (ms) | Eventos |
|---|---|---|
| 0 | | ANDROID_START |
| 1043 | **1043** | ANDROID_SEND |
| 1186 | 143 | SERVER_START_MATCHING |
| 1373 | 187 | SERVER_END_MATCHING |
| 1373 | 0 | SERVER_NOT_FOUND |
| 1389 | 16 | ANDROID_USER_NOT_ABORTED |
| 1452 | 63 | ANDROID_RESEND |
| 1585 | 133 | SERVER_RESTART_MATCHING |
| 1755 | 170 | SERVER_REEND_MATCHING |
| 1755 | 0 | SERVER_NOT_FOUND |
| 1776 | 21 | ANDROID_USER_NOT_ABORTED |
| 1811 | 35 | ANDROID_RESEND |
| 1966 | 155 | SERVER_RESTART_MATCHING |
| 2149 | 183 | SERVER_FOUND |
| 2149 | 0 | SERVER_END |
| 2149 | 0 | SERVER_REEND_MATCHING |
| 2221 | 72 | ANDROID_END |
| 3720 | **1499** | ANDROID_POSITIVE_DETECTION |
| 3758 | 38 | ANDROID_MQTT_PUBLISH |
| 3786 | 28 | HUE_DIMMER_RESPONSE |

Da Tabela 7 podemos ver também que a maior parte do tempo gasta em todo o processo de reconhecimento acontece logo no início entre os dois primeiros eventos e entre o evento de Android End e Android Positive Detection.

O primeiro caso, cerca de 1s, deve-se ao facto de se ter introduzido propositadamente um atraso de 1000ms para que a câmara pudesse focar convenientemente o objeto, pois se esse atraso não tivesse sido aplicado, na grande maioria dos casos a primeira imagem que era enviada para o servidor aplicacional ia algo estragada, o que implicava necessariamente pelo menos mais uma iteração. Tentou apurar-se outros valores diferentes, mas acabou por verificar-se que 1s de atraso no primeiro envio era 100% seguro, mas valores como 700ms ou 800ms também se mostraram bastante aceitáveis.

No segundo caso, cerca de 1500ms, são devidos à intervenção humana, ou seja, desde que o utilizador se apercebeu que o objeto foi reconhecido até que efetivamente o confirmou como um verdadeiro positivo. Assim, se retirarmos esses dois tempos ficamos com um tempo final de:

$$3786ms - 1043ms - 1499ms = 1244ms \quad (2)$$

Portanto, caso não tivesse sido introduzido o atraso inicial de 1s não considerando o tempo de intervenção humana, teríamos um tempo total de 1244ms.

Caso o objeto fosse reconhecido na primeira iteração, teríamos um tempo total de:

$$1244ms - 780ms = 464ms \quad (3)$$

Foram retirados 780ms ao tempo anterior, que correspondem ao tempo gasto com as sucessivas iterações até encontrar o objeto (a sombreado na Tabela 7). Ou seja, num hipotético caso ideal, teríamos despendido menos de meio segundo a identificar o objeto e a dar a possibilidade de interagir com o mesmo.

## 5.2 Testes de Desempenho

Para além dos testes referidos no ponto anterior, fez-se variar algumas condições nos testes realizados, tais como, variar a posição da câmara, diferentes condições de iluminação e diferentes tipos de objetos. Tais testes permitiram-nos obter taxas de erro médias, tempo médio de atuação dos sensores, etc., conforme referido anteriormente.

Nos gráficos seguintes são apresentados os tempos obtidos entre diversos eventos. O eixo horizontal (instante) representa cada medição de tempo entre os dois eventos referidos pelo gráfico, não sendo necessariamente consecutivos, isto é, podem ocorrer outros eventos intermédios (eventos repetidos caso haja várias iterações, por exemplo).

No gráfico abaixo (Figura 29) e Tabela 8 podemos verificar o atraso inicial dado propositadamente para a focagem, de cerca de 1s. Neste caso concreto não existem eventos intermédios, pois sempre após um evento de Android Start, existe logo de seguida um evento de Android Send. Portanto o gráfico apresenta todos os testes efetuados (140 no total) com o respetivo tempo entre estes dois eventos.

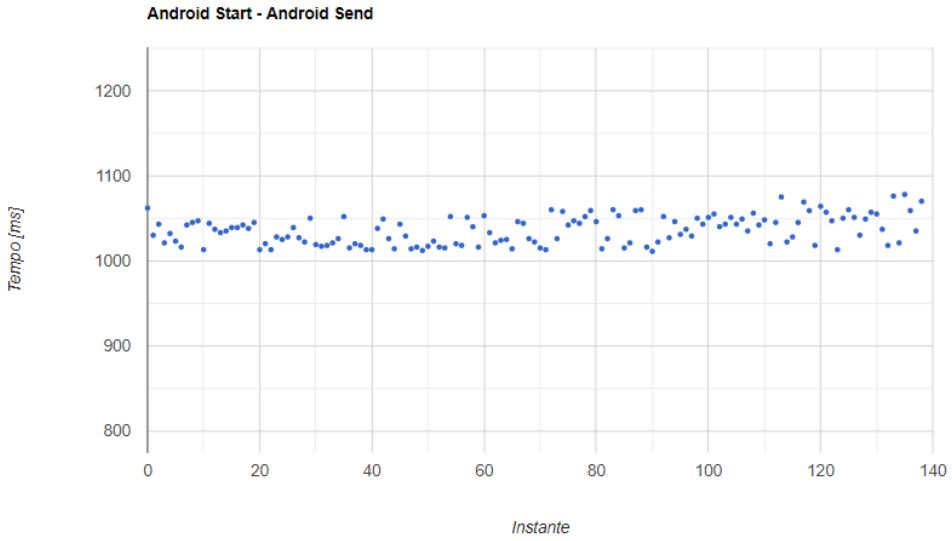


*Figura 29 – Tempo entre eventos Android Start e Android Send*

*Tabela 8 – Resultados Android Start – Android Send*

| Total | Mín | Q1 | Q2 | Q3 | Máx | Média | Desv. P. | Outliers | % Outliers |
|---|---|---|---|---|---|---|---|---|---|
| 139 | 1011 | 1021.0 | 1037.0 | 1049.0 | 1078 | 1036.0 | 17.0 | 0 | 0.0% |

Conforme se pode observar no gráfico seguinte (Figura 30) e Tabela 9, o tempo despendido na transmissão de dados (imagem e meta-dados) é relativamente baixo, cerca de 213ms em média. Ou seja, no envio entre *smartphone* e servidor e vice-versa, não se registaram atrasos significativos.

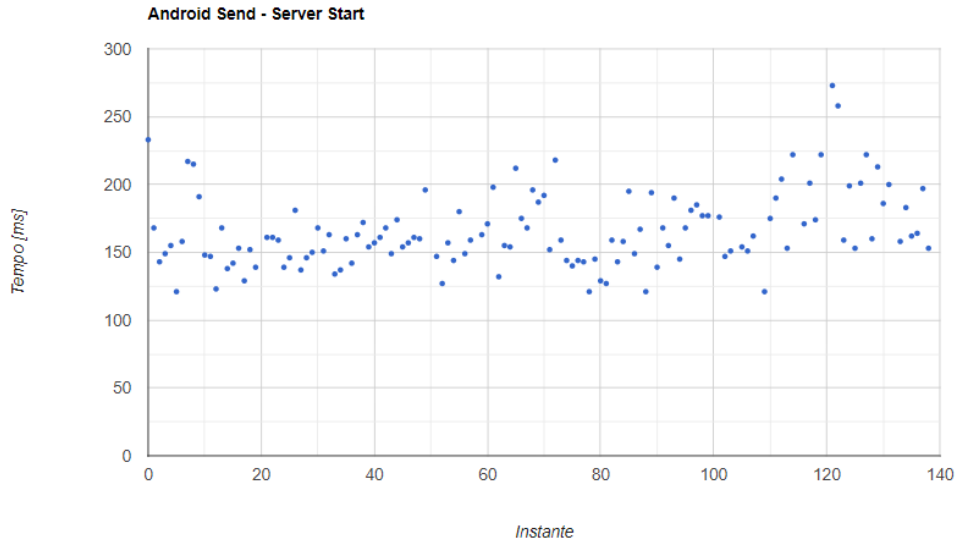


*Figura 30 – Tempo gasto na rede (transmissão de dados)*

*Tabela 9 – Resultados Android Send – Server Start*

| Total | Mín | Q1 | Q2 | Q3 | Máx | Média | Desv. P. | Outliers | % Outliers |
|---|---|---|---|---|---|---|---|---|---|
| 139 | 121 | 149.0 | 161.0 | 190.0 | 3244 | 213.0 | 283.0 | 11 | 8.0% |

No gráfico representado pela Figura 31 e Tabela 10, temos os tempos totais do universo de testes. Em média encontrou-se uma correspondência em pouco mais de 4 segundos (4278ms), num total de 129 casos de sucesso (houve correspondência e o utilizador validou-a como um verdadeiro positivo).

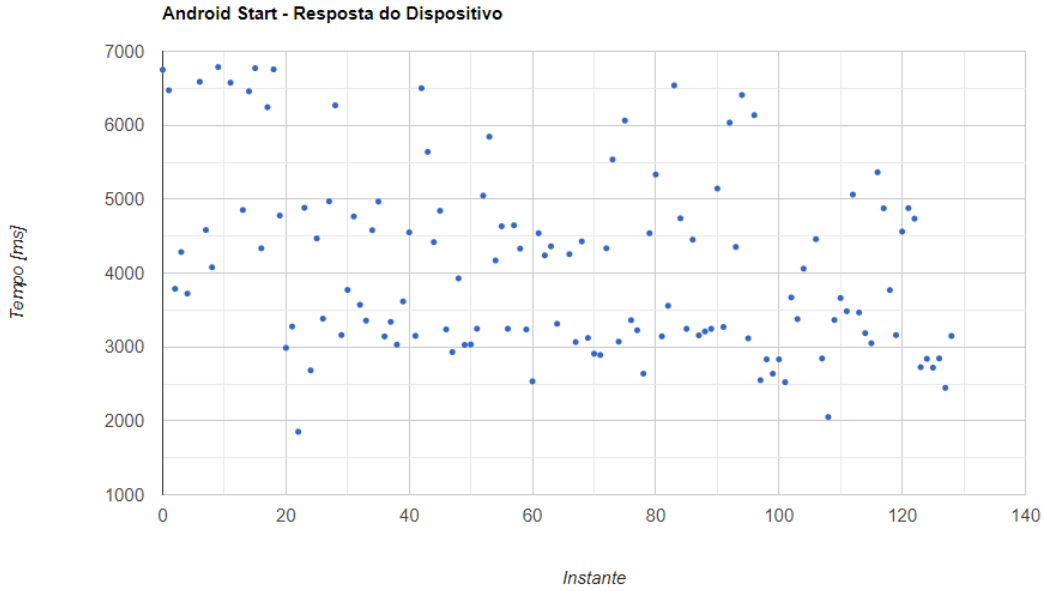


*Figura 31 – Tempo total até à primeira resposta do dispositivo*

*Tabela 10 – Resultados Android Start – Primeira Resposta do Dispositivo*

| Total | Mín | Q1 | Q2 | Q3 | Máx | Média | Desv. P. | Outliers | % Outliers |
|---|---|---|---|---|---|---|---|---|---|
| 129 | 861 | 3149.5 | 3786.0 | 4877.0 | 13861 | 4278 | 1735.0 | 4 | 3.0% |

Se em média o utilizador demorou 2s a dar feedback sobre o resultado obtido e tendo em todos os testes um atraso inicial de 1s, podemos concluir que para encontrar uma correspondência válida demorou cerca de 1278ms, pois ao tempo de 4278ms retiramos 2000ms + 1000ms.

O tempo total para encontrar uma correspondência válida está diretamente ligado ao número de tentativas, conforme já referido. Na Figura 32, pode observar-se que, embora não seja diretamente proporcional, existe uma relação entre o tempo total de deteção de uma imagem e o número de tentativas para a encontrar (com exceção entre os instantes 40 a 70 e 80 a 100). Isto é, quanto maior o número de tentativas, maior tenderá a ser o tempo total, pois existe repetição de vários eventos até se conseguir atingir o objetivo, como por exemplo retransmissões na rede, processamentos múltiplos de reconhecimento, etc.

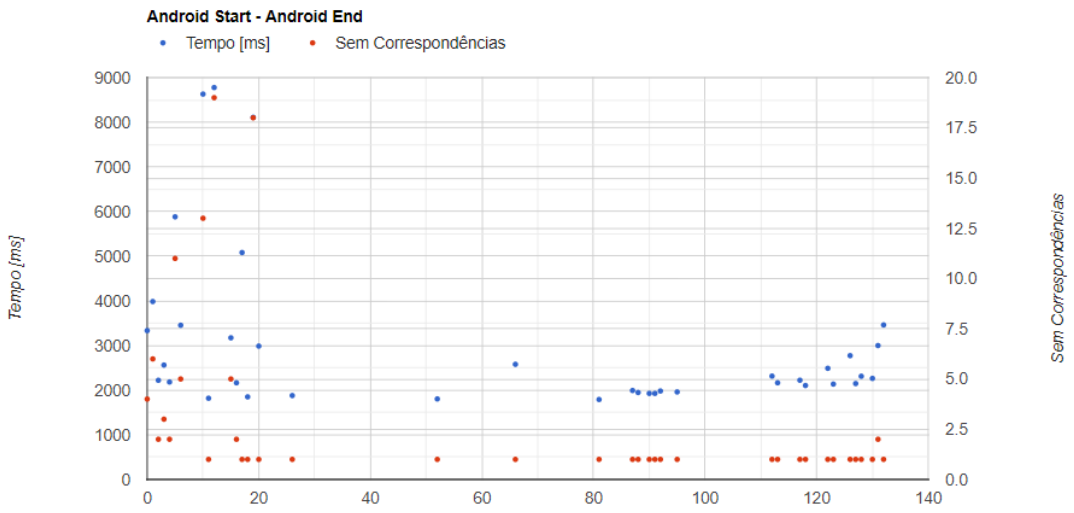


*Figura 32 – Correlação entre o tempo total e o número de tentativas*

A forma como estes testes foram realizados é idêntica à dos testes descritos no ponto anterior, no entanto nos testes de desempenho fez-se variar os objetos a reconhecer, bem como o ângulo de incidência para cada um. Utilizou-se cartas (tal como nos testes

anteriores) e objetos reais. A razão pela qual se fizeram testes com diferentes tipos de objetos (cartas e objetos físicos reais) foi a de que nas cartas conseguimos controlar melhor as suas características e compará-las com outras cartas que não difiram muito entre si, como é o caso do 8 de espadas e o 9 de paus.

As cartas utilizadas foram o 8 de espadas, 9 de paus, Joker e Valete de ouros.

Cada carta simulava, no entanto, um objeto real. Isto é, quando se reconhecia uma determinada carta, o objeto que lhe estava associado era um objeto real, no qual podia ver-se as ações que lhe eram enviadas, por exemplo, uma ventoinha ou uma lâmpada, entre outros.

Os objetos reais, estavam acoplados a uma tábua (vide Figura 18). A saber:

- Lâmpada embutida
- Ventilador (ligeiramente saliente do suporte)
- Tomada
- Lâmpada externa (com regulação da intensidade)

Os três primeiros objetos reais usam o protocolo KNX e o último utiliza a tecnologia HUE da Philips (ZigBee).

Cartas

Conforme já referido houve dois tipos de objetos nestes testes, cartas e objetos reais. Nesta secção irá apresentar-se alguns resultados dos testes efetuados com as cartas.

Nas Tabela 11 pode ver-se um resumo dos resultados obtidos. Na primeira coluna temos os tipos de eventos, na segunda coluna o total de ocorrências, seguido do valor mínimo atingido, os 3 quartis, valor máximo obtido e os respetivos *outliers*.

Os *outliers* [43] foram calculados da seguinte forma:

$$\begin{aligned} IQR &= Q_3 - Q_1 \\ Limite_{sup} &= Q_3 \times IQR \\ Limite_{inf} &= Q_1 \times IQR \\ Limite_{superior} &\leq Outlier \leq Limite_{inferior} \end{aligned} \tag{4}$$

Ou seja, todos os valores acima do limite superior e todos os valores abaixo do limite inferior são considerados *outliers* e não devem ser considerados na amostra. Os *outliers* são importantes, na medida em que deve ser analisado o porquê de tais valores, valores estes que saem fora dos padrões ditos normais. Isto é, devemos entender por que razão alguns valores ficam completamente fora dos valores médios e se devem ou não ser descartados.

*Tabela 11 –Resultados dos testes com cartas*

| QUADRO RESUMO | Total | Mínimo | Q1 | Q2 | Q3 | Máximo | *Outliers* | |
|---|---|---|---|---|---|---|---|---|
| **Verdadeiros positivos** | 123 | - | - | - | - | - | - | - |
| **Falsos positivos** | 8 | - | - | - | - | - | - | - |
| **Android Start – Android Send** | 139 | 1011 | 1021 | **1037** | 1049 | 1078 | 0 | 0% |
| **Android Send – Server Start** | 139 | 121 | 149 | **161** | 190 | 3244 | 11 | 8% |
| **Android Resend – Server Restart** | 215 | 111 | 138 | **149** | 166 | 3237 | 17 | 8% |
| **Server Start – Server End Match** | 139 | 140 | 196 | **218** | 255 | 443 | 9 | 6% |
| **Server Restart – Server ReMatch** | 215 | 133 | 181 | **195** | 210 | 406 | 26 | 12% |
| **Server Not Found – Limit Reached** | 217 | 6 | 13 | **17** | 23 | 281 | 15 | 7% |
| **Server Found – Android End** | 137 | 47 | 64 | **74** | 99 | 1260 | 13 | 9% |
| **Android End – False \| Positive Det.** | 137 | 814 | 1465 | **1728** | 2719 | 6146 | 3 | 2% |
| **Positive Detection - MQTT** | 129 | 11 | 27 | **104** | 107 | 2431 | 13 | 10% |
| **MQTT-Device Response** | 584 | 8 | 26 | **32** | 37 | 132 | 25 | 4% |

Conforme se pode observar, existe o atraso inicial de cerca de 1000ms, conforme se esperava. Mais uma vez onde é "perdido" algum tempo é no feedback do utilizador, pois trata-se de uma ação de intervenção humana.

Entre os eventos Android (Re)Send e Server (Re)Start os valores são relativamente baixos, no entanto observou-se alguns casos em que esse tempo subiu consideravelmente. Os *outliers* para estes eventos são de 8%. Observou-se que por vezes houve um atraso do lado do servidor a iniciar o processamento da imagem enviada. O SpringBoot utiliza por defeito um servidor aplicacional Tomcat e alterou-se para passar a utilizar um Jetty. Os resultados foram melhores, mas mesmo assim o atraso manteve-se em alguns dos casos.

Como são enviadas várias imagens até se conseguir obter um *match*, pareceu-nos que havia alguma sobrecarga do lado do servidor e a API não conseguia processar todos os pedidos com a rapidez que lhe era exigida.

Pelas razões de sobrecarga já referidas também se notaram atrasos no processamento das imagens, isto é, a calcular os descritores, bem como a encontrar correspondências. Este era um ponto onde não se esperava de todo discrepâncias, mas foi onde houve mais, cerca de 12% dos casos foram considerados *outliers*. O que reforça ainda mais esta teoria é o facto de haver mais casos discrepantes quando se trata de tentativas que não sejam a primeira (Server Restart – Server ReMatch), pois existe uma maior sobrecarga da API a processar sucessivamente novas imagens.

Importa ainda referir que a percentagem de verdadeiros positivos sobre os falsos positivos é de cerca de 93.5% ($1 - \frac{8}{123}$). Ou seja, é uma percentagem alta e que prova que este conceito funciona na grande maioria dos casos de estudo.

Nos restantes casos, o objeto encontrado não correspondeu à imagem enviada. Isto é, houve *match*, mas esse *match* não corresponde ao objeto pretendido. Para exemplificar melhor o que acontece nestes casos, dar-se-á um exemplo extremo: pretendia interagir-se com uma lâmpada e apontou-se a câmara para essa mesma lâmpada, mas aquilo que o servidor encontrou como sendo o mesmo objeto, não foi a lâmpada, mas sim uma tomada. Para estes casos foi o utilizador que indicou que se tratava de um falso positivo e voltou ao ecrã principal tendo, portanto, hipótese um nova tentativa de reconhecimento.

Mesmo com cartas parecidas, refira-se 8 de espadas e 9 de paus, onde o padrão é semelhante e são ambas da mesma cor, os resultados foram bastante satisfatórios, como se mostra na Tabela 12, obtendo-se 100% de reconhecimentos validados numa das cartas e 80% na outra.

*Tabela 12 – Comparação de resultados entre 2 cartas idênticas*

| Carta | Posição | Verdadeiros Positivos | | Falsos Positivos | | Sem Reconhecimentos | |
|---|---|---|---|---|---|---|---|
| 8 espadas | Cima | 8 | 100.0% | 0 | 0.0% | 0 | 0.0% |
| | Lado Direito | 7 | 100.0% | 0 | 0.0% | 0 | 0.0% |
| | Lado Esquerdo | 7 | 100.0% | 0 | 0.0% | 0 | 0.0% |
| | Deitada (90º) | 9 | 100.0% | 0 | 0.0% | 0 | 0.0% |
| | **Total** | **31** | **100.0%** | **0** | **0.0%** | **0** | **0.0%** |
| 9 paus | Cima | 7 | 77.8% | 2 | 22.2% | 0 | 0.0% |
| | Lado Direito | 5 | 71.4% | 2 | 28.6% | 0 | 0.0% |
| | Lado Esquerdo | 5 | 83.3% | 1 | 16.7% | 0 | 0.0% |
| | Deitada (90º) | 7 | 87.5% | 1 | 12.5% | 0 | 0.0% |
| | **Total** | **24** | **80.0%** | **6** | **20.0%** | **0** | **0.0%** |

Passando para a análise de 2 cartas bem distintas, refira-se joker e valete de ouros, os resultados foram os apresentados na Tabela 13.

*Tabela 13 – Resultados entre duas cartas distintas*

| Carta | Posição | Verdadeiros Positivos | | Falsos Positivos | | Sem Reconhecimentos | |
|---|---|---|---|---|---|---|---|
| Joker | Cima | 7 | 100% | 0 | 0.0% | 0 | 0.0% |
| | Lado Direito | 7 | 100% | 0 | 0.0% | 0 | 0.0% |
| | Lado Esquerdo | 7 | 100% | 0 | 0.0% | 0 | 0.0% |
| | Deitada (90º) | 8 | 100% | 0 | 0.0% | 0 | 0.0% |
| | **Total** | **29** | **100%** | **0** | **0.0%** | **0** | **0.0%** |
| Valete Ouros | Cima | 8 | 88.9% | 1 | 11.1% | 0 | 0.0% |
| | Lado Direito | 5 | 83.3% | 1 | 16.7% | 0 | 0.0% |
| | Lado Esquerdo | 7 | 100.0% | 0 | 0.0% | 0 | 0.0% |
| | Deitada (90º) | 7 | 100.0% | 0 | 0.0% | 0 | 0.0% |
| | **Total** | **27** | **93.1%** | **2** | **6.9%** | **0** | **0.0%** |

Conforme se pode verificar com cartas bem distintas (Joker e Valete de ouros), os resultados foram substancialmente mais satisfatórios, subindo a percentagem de verdadeiros reconhecimentos para perto de 95%, no pior dos cenários.

Objetos Reais

Após os testes com as cartas efetuou-se testes com objetos reais. Quando dizemos reais referimo-nos a lâmpadas, ventoinhas, tomadas, propriamente ditos, ou seja, objetos que realmente podem ser controlados.

Na Tabela 14 pode ver-se um resumo dos resultados obtidos.

*Tabela 14 – Resultados com objetos reais*

| QUADRO RESUMO | Total | Mín. | Q1 | Q2 | Q3 | Máx. | *Outliers* | |
|---|---|---|---|---|---|---|---|---|
| **Verdadeiros positivos** | - | 68 | - | - | - | 68 | - | - |
| **Falsos positivos** | - | 30 | - | - | - | 30 | - | - |
| **Android Start – Android Send** | 103 | 1014 | 1027 | **1035** | 1040 | 1134 | 1 | 1% |
| **Android Send – Server Start** | 103 | 103 | 142 | **164** | 234 | 9370 | 16 | 16% |
| **Android Resend – Server Restart** | 581 | 74 | 137 | **160** | 220 | 4028 | 93 | 16% |
| **Server Start – Server End Match** | 103 | 94 | 140 | **156** | 186 | 332 | 5 | 5% |
| **Server Restart – Server ReMatch** | 581 | 93 | 139 | **146** | 165 | 359 | 40 | 7% |
| **Server Not Found – Limit Reached** | 581 | 1 | 11 | **18** | 35 | 4840 | 98 | 17% |
| **Server Found – Android End** | 98 | 54 | 86 | **115** | 168 | 1422 | 10 | 10% |
| **Android End – False \| Positive Det.** | 98 | 1160 | 1385 | **1551** | 1829 | 4600 | 8 | 8% |
| **Positive Detection - MQTT** | 68 | 21 | 105 | **107** | 109 | 631 | 14 | 21% |
| **MQTT-Device Response** | 255 | 1 | 27 | **32** | 43 | 8227 | 45 | 18% |

Conforme se pode observar na Tabela 14, existe também aqui o atraso inicial de cerca de 1000ms, conforme se esperava. Mais uma vez os maiores atrasos ocorreram nos pontos já referidos e pelas mesmas razões, embora aqui ocorreu um valor mais elevado de *outliers*, pois por vezes foram necessárias mais iterações devido a condições de iluminação diferentes. Estas iterações também provocaram de algum modo uma sobrecarga em todo o sistema, o que acabou por se refletir nestes *outliers* adicionais.

Na Tabela 15 apresenta-se o resumo dos resultados obtidos para os diferentes tipos de objetos e ângulos de incidência.

*Tabela 15 – Comparação de resultados entre objetos*

| Objeto | Posição da Câmara / Ângulo de incidência | Verdadeiros Positivos | | Falsos Positivos | | Sem Reconhecimentos | |
|---|---|---|---|---|---|---|---|
| HUE Lamp | Cima | 9 | 90.0% | 1 | 10.0% | 0 | 0.0% |
| | Lado Direito | 5 | 62.5% | 3 | 37.5% | 0 | 0.0% |
| | Lado Esquerdo | 0 | 0.0% | 3 | 75.0% | 1 | 25.0% |
| | **Total** | **14** | **63.6%** | **7** | **31.8%** | **1** | **4.6%** |
| KNX Lamp | Cima | 4 | 57.1% | 3 | 42.9% | 0 | 0.0% |
| | Lado Direito | 1 | 14.2% | 3 | 42.9% | 3 | 42.9% |
| | Lado Esquerdo | 2 | 20.0% | 7 | 70.0% | 1 | 10.0% |
| | **Total** | **7** | **29.3%** | **13** | **54.1%** | **4** | **16.6%** |
| Tomada | Cima | 6 | 42.9% | 8 | 57.1% | 0 | 0.0% |
| | Lado Direito | 8 | 88.9% | 1 | 11.1% | 0 | 0.0% |
| | Lado Esquerdo | 8 | 88.9% | 1 | 11.1% | 0 | 0.0% |
| | **Total** | **22** | **68.8%** | **10** | **31.2%** | **0** | **0.0%** |
| Ventoinha | Cima | 10 | 100% | 0 | 0.0% | 0 | 0.0% |
| | Lado Direito | 8 | 100% | 0 | 0.0% | 0 | 0.0% |
| | Lado Esquerdo | 7 | 100% | 0 | 0.0% | 0 | 0.0% |
| | **Total** | **25** | **100%** | **0** | **0.0%** | **0** | **0.0%** |

De um modo geral, podemos constatar que o resultado, embora menos positivo que no caso das cartas, foi ainda satisfatório, tendo como único caso verdadeiramente negativo, o caso da lâmpada KNX, em que não obtivemos uma taxa de verdadeiros positivos superior à da dos falsos positivos. Isto deve-se ao facto de que a lâmpada é branca em fundo branco, pelo que o algoritmo de reconhecimento de imagens não conseguiu atribuir *keypoints* suficientes, devido à falta de contraste do objeto.

No caso da lâmpada HUE, também se notam algumas falhas para este algoritmo, embora o resultado tenha sido bastante melhor que o caso anterior. As razões são as mesmas, pois

a lâmpada (branca) estava pendurada por um fio, mas o fundo era a tábua, também ela branca.

Já para a tomada obteve-se resultados mais satisfatórios, com todos os casos reconhecidos, mas ainda assim com uma percentagem algo elevada para os falsos positivos. Embora não seja da mesma cor da do fundo, é uma cor constante e sem alterações de padrão, pelo que justifica a melhoria, mas ainda assim, não sendo totalmente perfeita.

No que à ventoinha diz respeito, obteve-se resultados perfeitos, com 100% de verdadeiros positivos. Esta está saliente do suporte, e apresenta uma cor preta face ao fundo branco, pelo que podemos concluir que, para o algoritmo usado, situações como esta funcionam sem erros.

No geral podemos concluir que para um reconhecimento mais efetivo terá de haver mais contraste entre o objeto e o fundo do mesmo. É importante também que se o objeto é plano e sem relevo, deverá ter algumas características únicas que o possam distinguir de outros objetos e que apresente preferencialmente singularidades no mesmo. Se o objeto apresentar relevo, é importante também ter algumas características e singularidades que o distingam de outros, pois caso contrário o algoritmo terá alguma dificuldade em escolher os *keypoints* para o mesmo. Outra forma de obter melhores resultados poderia também passar por utilizar outro algoritmo com uma maior taxa de eficácia.

# Capítulo 6. **Conclusões e Recomendações**

## 6.1 Principais Conclusões

Com esta solução provou-se que é possível e, de um modo eficaz, melhorar a utilização dos dispositivos móveis na integração com dispositivos IoT, num contexto de uma casa inteligente. Provou-se que os utilizadores não têm que ficar limitados nas suas interações com as aplicações à mera utilização de botões específicos, sendo possível tirar partido da importante mais valia, até agora de certa forma ignorada, dos múltiplos sensores existentes nos dispositivos móveis.

Atualmente, as aplicações móveis estão muitas vezes limitadas a controlar os seus próprios dispositivos, mas neste estudo também ficou provado que tal não terá de ser necessariamente assim. Provou-se ser possível controlar dispositivos com tecnologias completamente distintas a partir de uma única aplicação móvel, como é o caso de dispositivos que utilizam tecnologia KNX conectados por cabo versus dispositivos que utilizam uma tecnologia sem fios como é o caso da lâmpada com regulação de intensidade, a qual usa tecnologia ZigBee (Philips HUE).

O protótipo aqui desenvolvido permitiu comprovar parte da arquitetura, com recurso a testes funcionais, que demonstram uma integração mais plena e eficaz dos sensores disponíveis nos *smartphone*s, e testes de desempenho que demonstram que a eficiência obtida que permite cumprir os objetivos de usabilidade propostos.

É uma solução inovadora, pois atualmente não existem no mercado produtos semelhantes, onde a utilização dos sensores presentes num *smartphone* é feita de forma totalmente integrada. Esta é uma área com grande potencial comercial, podendo mesmo abrir caminho a novas investigações e ao surgimento de novos sensores e equipamentos que tirem ainda mais partido desta tecnologia.

## 6.2 Principais Limitações do Estudo

Para este estudo os objetos que serviram de base aos testes podiam ter sido em maior quantidade e com maior diversidade. Podia ter-se utilizado uma maior variedade de objetos com diferentes tipos de características tais como diferentes tipos de tons, de formas, de padrões, etc. Mas como não era o nosso principal foco, decidiu utilizar-se apenas o necessário para cumprir os objetivos.

Por forma a este estudo ficar um pouco mais completo, poderia também ter-se aprofundado um pouco mais quer as condições de iluminação nos objetos, quer o ângulo de incidência da câmara face aos objetos, bem como a distância de focagem, quer mesmo o tipo de objeto, pois tais variáveis poderiam implicar a escolha de outro algoritmo, mais apropriado e abrangente em termos de casos de sucesso.

Para além disso existem inúmeros tipos de algoritmos para fazer o reconhecimento das imagens, inclusive alguns com aprendizagem automática. Como referido anteriormente usamos o SURF através do openCV, pois era suficiente para o caso em estudo, mas existe um sem fim de algoritmos que poderiam ter sido utilizados.

O facto de ter de existir uma pré-configuração no openHab e de toda a infraestrutura, a qual não é trivial e acessível a qualquer utilizador, pode também constituir um ponto de melhoria futura por forma a tornar toda a aplicação como um todo mais acessível a um possível utilizador final.

Por último e também a título demonstrativo foram utilizados alguns sensores de *smartphone*s, mas muitos mais podiam ter sido utilizados, como o sensor do GPS, acelerómetro, luminosidade, entre muitos outros.

## 6.3 Propostas de Investigação Futura

Por forma a reduzir a latência de rede, em vez de se utilizar MQTT como protocolo para comunicação entre o openHAB e o *core*, seria interessante explorar os *bindings* que as plataformas disponibilizam. Teríamos certamente ganhos de desempenho.

Outra forma de explorar este conceito seria colocar todos os módulos concentrados numa só máquina: *smartphone*. Atualmente estes aparelhos possuem grandes capacidades de processamento e de armazenamento físico e volátil, pelo que serão excelentes candidatos a concentrarem todos os módulos apresentados neste estudo. Assim, seria interessante comparar o desempenho de uma solução integrada com outras não integradas. Desta forma, teríamos a base de dados, o software de reconhecimento de imagens e o MQTT (servidor e cliente), tudo a correr dentro do *smartphone*, ficando apenas a plataforma IoT descentralizada.

Por último, uma outra forma de otimização seria utilizar um algoritmo de reconhecimento de imagens que recorra a inteligência artificial, por exemplo com recurso a *Machine Learning*. Desta forma o algoritmo iria tornando-se mais robusto a cada iteração.

# Referências

[1] A. Bhawiyuga, D. P. Kartikasari, K. Amron, O. B. Pratama, and M. W. Habibi, "Architectural design of IoT-cloud computing integration platform," *Telkomnika (Telecommunication Comput. Electron. Control.*, vol. 17, no. 3, pp. 1399–1408, 2019.

[2] S. Seifried, G. Gridling, and W. Kastner, "KNX IPv6: Design issues and proposed architecture," *IEEE Int. Work. Fact. Commun. Syst. - Proceedings, WFCS*, 2017.

[3] M. B. Yassein, W. Mardini, and A. Khalil, "Smart homes automation using Z-wave protocol," *Proc. - 2016 Int. Conf. Eng. MIS, ICEMIS 2016*, pp. 1–6, 2016.

[4] B. Ousat and M. Ghaderi, "LoRa Network Planning : Gateway Placement and Device Configuration," *2019 IEEE Int. Congr. Internet Things*, pp. 25–32, 2019.

[5] Analytics IoT, "Industrial Industrial Analytics 2016 / 2017," 2017.

[6] "Gartner Identifies Five Emerging Technology Trends That Will Blur the Lines Between Human and Machine." [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-08-20-gartner-identifies-five-emerging-technology-trends-that-will-blur-the-lines-between-human-and-machine. [Accessed: 07-Dec-2019].

[7] A. Jamalipour, H. Nikookar, and M. Ruggieri, *Digitising the Industry Internet of Things Connecting the Physical, Digital and Virtual Worlds*, vol. 49, no. 9. River Publishers, 2016.

[8] C. Isabel and M. Alexandre, "Introdução à Investigação em Engenharia," 2017.

[9] C. Kotas, T. Naughton, and N. Imam, "A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high performance computing," *2018 IEEE Int. Conf. Consum. Electron. ICCE 2018*, vol. 2018-Janua, pp. 1–4, 2018.

[10] "Azure IoT Hub | Microsoft Azure." [Online]. Available: https://azure.microsoft.com/en-gb/services/iot-hub/. [Accessed: 14-Nov-2017].

[11] "AWS IoT - Amazon Web Services." [Online]. Available: https://aws.amazon.com/iot/. [Accessed: 14-Nov-2017].

[12] M. Bertacchi, I. Silveira, and N. Omar, "A comparative analysis of the evolution of the IBM Watson's visual recognition API on android," *Proc. - 13th Work. Comput. Vision, WVC 2017*, vol. 2018-Janua, pp. 120–125, 2018.

[13] "IBM Watson Internet of Things (IoT)." [Online]. Available: https://www.ibm.com/internet-of-things/. [Accessed: 14-Nov-2017].

[14] L. Smirek, G. Zimmermann, and M. Beigl, "Just a Smart Home or Your Smart Home - A Framework for Personalized User Interfaces Based on Eclipse Smart Home and Universal Remote Console," *Procedia Comput. Sci.*, vol. 58, no. Euspn, pp. 107–116, 2016.

[15] "Eclipse SmartHome - A Flexible Framework for the Smart Home." [Online]. Available: http://www.eclipse.org/smarthome/. [Accessed: 14-Nov-2017].

[16] UNIFY-IoT H2020, "Supporting Internet of Things Activities on Innovation Ecosystems H2020 – UNIFY-IoT Project Analysis on IoT Platforms Adoption

Activities," 2017.

[17] "Top 20 IoT Platforms in 2018 (Updated)." [Online]. Available: https://internetofthingswiki.com/top-20-iot-platforms/634/. [Accessed: 15-Jan-2018].

[18] L. Dobrescu, "Domotic Embedded System," pp. 5–8.

[19] J. Kim and J. W. Lee, "OpenIoT: An open service framework for the Internet of Things," in *2014 IEEE World Forum on Internet of Things, WF-IoT 2014*, 2014, pp. 89–93.

[20] "OpenIoT – Open Source cloud solution for the Internet of Things." [Online]. Available: http://www.openiot.eu/. [Accessed: 14-Nov-2017].

[21] D. Zinca and A. T. Raspberry, "Experiments with Protocols and Frameworks used in IoT," *Commun. Dep. Tech. Univ. Cluj-Napoca, Rom.*, pp. 1–4, 2018.

[22] Y. Li, W. Sheng, G. Yang, B. Liang, Z. Su, and Z. Chen, "Home Assistant-Based Collaborative Framework of Multi-Sensor Fusion for Social Robot∗," *Proc. World Congr. Intell. Control Autom.*, vol. 2018-July, pp. 401–406, 2019.

[23] "What is X10? | X10 Home Automation Products | X10 Systems." [Online]. Available: https://www.smarthome.com/sc-what-is-x10-home-automation. [Accessed: 23-Sep-2019].

[24] C. W. Badenhop, S. R. Graham, B. W. Ramsey, B. E. Mullins, and L. O. Mailloux, "The Z-Wave routing protocol and its security implications," *Comput. Secur.*, vol. 68, pp. 112–129, 2017.

[25] R. J. Robles and T.-H. Kim, "Applications, Systems and Methods in Smart Home Technology: A Review," *Int. J. Adv. Sci. Technol.*, vol. 15, no. November, pp. 37–48, 2010.

[26] S. Khanji, F. Iqbal, and P. Hung, "ZigBee Security Vulnerabilities : Exploration and Evaluating," *2019 10th Int. Conf. Inf. Commun. Syst.*, pp. 52–57, 2019.

[27] A. I. Ali, S. Z. Partal, S. Kepke, and H. P. Partal, "ZigBee and LoRa based Wireless Sensors for Smart Environment and IoT Applications," *2019 1st Glob. Power, Energy Commun. Conf.*, pp. 19–23, 2019.

[28] W. Lumpkins, "Home Automation: Insteon (X10 Meets Powerline) [Product Reviews]," *IEEE Consum. Electron. Mag.*, vol. 4, no. 4, pp. 140–144, 2015.

[29] S. Seifried and W. Kastner, "KNX IPv6 : Design Issues and Proposed Architecture," p. 10, 2017.

[30] S. L. Woo and H. H. Seung, "Implementation of a KNX-ZigBee gateway for home automation," *Dig. Tech. Pap. - IEEE Int. Conf. Consum. Electron.*, pp. 545–549, 2009.

[31] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, "Understanding the Limits of LoRaWAN," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 34–40, 2017.

[32] M. C. Bor, U. Roedig, T. Voigt, and J. M. Alonso, "Do LoRa Low-Power Wide-Area Networks Scale?," *Proc. 19th ACM Int. Conf. Model. Anal. Simul. Wirel. Mob. Syst. - MSWiM '16*, pp. 59–67, 2016.

[33] G. Callebaut, G. Ottoy, and L. De Strycker, "Bring your own Sensor: Use your Android Smartphone as a Sensing Platform," *SAS 2019 - 2019 IEEE Sensors Appl. Symp. Conf. Proc.*, pp. 1–5, 2019.

[34] N. Chockwanich and V. Visoottiviseth, "Intrusion Detection by Deep Learning with TensorFlow," *Int. Conf. Adv. Commun. Technol. ICACT*, vol. 2019-Febru, pp. 654–659, 2019.

[35] N. Boyko, O. Basystiuk, and N. Shakhovska, "Performance Evaluation and Comparison of Software for Face Recognition, Based on Dlib and Opencv Library," in *Proceedings of the 2018 IEEE 2nd International Conference on Data Stream Mining and Processing, DSMP 2018*, 2018, pp. 478–482.

[36] A. Bommert, J. Rahnenführer, and M. Lang, "A Multicriteria Approach to Find Predictive and Sparse Models with Stable Feature Selection for High-Dimensional Data," *Comput. Math. Methods Med.*, vol. 2017, 2017.

[37] "Introduction to SURF (Speeded-Up Robust Features) — OpenCV-Python Tutorials 1 documentation." [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html. [Accessed: 23-Sep-2019].

[38] C. Barajas-García, S. Solorza-Calderón, and E. Gutiérrez-López, "Scale, translation and rotation invariant Wavelet Local Feature Descriptor," *Appl. Math. Comput.*, vol. 363, p. 124594, 2019.

[39] S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," *2018 Int. Conf. Comput. Math. Eng. Technol. Inven. Innov. Integr. Socioecon. Dev. iCoMET 2018 - Proc.*, vol. 2018-Janua, pp. 1–10, 2018.

[40] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proceedings of the IEEE International Conference on Computer Vision*, 2011, pp. 2564–2571.

[41] R. Leal, L. Santos, L. Vieira, R. Gonaalves, and C. Rabadao, "MQTT Flow Signatures for the Internet of Things," in *INESC TEC*, 2019, no. June, pp. 1–5.

[42] "The Raspberry PI Foundation." [Online]. Available: https://www.raspberrypi.org/. [Accessed: 28-Sep-2019].

[43] K. Patil, N. K. Nagwani, and S. Tripathi, "A Parametric Study of Partitioning and Density Based Clustering Techniques for Boxplot Generation," *2018 3rd Int. Conf. Converg. Technol. I2CT 2018*, pp. 1–5, 2018.

**FIM DO DOCUMENTO**