Sistema Pessoal de Deteção de Atividade

PADS: Personal Activity Detection System

Sérgio Daniel Rodrigues Vale

Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia
Praça 9 de Abril, 349, 4249-004 Porto, Portugal

Fevereiro de 2013

Sistema Pessoal de Deteção de Atividade

PADS: Personal Activity Detection System

Sérgio Daniel Rodrigues Vale

Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia
Praça 9 de Abril, 349, 4249-004 Porto, Portugal

Fevereiro de 2013

Sistema Pessoal de Deteção de Atividade

PADS: Personal Activity Detection System

Sérgio Daniel Rodrigues Vale

Licenciado em Engenharia Informática pela Universidade Fernando Pessoa

Dissertação apresentada à Universidade Fernando Pessoa como parte dos requisitos para obtenção do grau de Mestre em Engenharia Informática, orientada pelo Professor Doutor José Torres e coorientada pelo Professor Doutor Rui Silva Moreira.

Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia
Praça 9 de Abril, 349, 4249-004 Porto, Portugal

Fevereiro de 2013

Resumo

Vivemos numa era em que a esperança média de vida tem aumentado, traduzindo-se diretamente num crescimento cada vez mais significativo da população na faixa etária da terceira idade. Este fenómeno faz com que cada vez mais pessoas idosas vivam sozinhas em casa e tenham dificuldade em encontrar pessoas que as possam acompanhar e ajudar no seu dia-a-dia. Reconhece-se ainda que a atividade e o exercício físico são fundamentais para manter e promover a saúde, em particular nesta faixa etária, permitindo em geral melhorar a qualidade de vida e segurança das pessoas. Neste sentido têm vindo a ser desenvolvidos vários sistemas comerciais para monitorizar a atividade de pessoas, em particular idosos, por forma a acompanhar o que estes fazem durante o dia, recolhendo dados estatísticos de atividade bem como permitindo identificar potenciais situações de perigo (e.g., quedas). Contudo, as soluções existentes envolvem normalmente a aquisição de equipamentos com custos geralmente elevados e que baseiam o seu funcionamento na análise de apenas um tipo de informação ou fenómeno físico (e.g., acelerómetro, análise de cenas, etc.), limitando desta forma o resultado e a fidelidade da monitorização.

Neste contexto, o trabalho aqui apresentado propõe a utilização de um sistema de monitorização económico, baseado na fusão de vários tipos de informação (e.g., atividade física, localização do utilizador, etc.) recolhida num espaço inteligente preparado para o efeito. Este sistema combina informação recolhida e processada por vários componentes. Utiliza, por exemplo, dispositivos móveis hoje em dia vulgarizados e disponíveis por custos razoáveis (cf. *smartphones*) e que vêm equipados com um conjunto de sensores e capacidades de processamento adequados; utiliza ainda outros componentes vulgarmente existentes em contextos residenciais (cf. computador e câmaras de baixa resolução apontadas aos locais a monitorizar) e que podem ser integrados e reutilizados na solução proposta.

Esta dissertação propõe e descreve a estrutura física e lógica do sistema PADS (*Personal Activity Detection System*). O protótipo desenvolvido organiza-se em vários componentes talhados para a recolha e fusão de diferentes tipos de informação: i) uma aplicação Android que identifica e regista cinco atividades físicas distintas (cf. parado, andar, correr, deitado e queda); ii) uma aplicação desenvolvida em C++ para deteção da presença do utilizador com base no reconhecimento de faces; iii) uma aplicação que utiliza serviços da nuvem da Google para consultar as tomas de medicamentos do utilizador que se encontram agendadas; iv) uma aplicação em C++ que faz a fusão de toda a informação obtida pelos módulos anteriores e permite determinar a atividade do utilizador.

Apresenta-se a arquitetura global do sistema e os vários componentes envolvidos, descrevendo-se com pormenor todos os algoritmos e os detalhes de implementação.

Procedeu-se ainda à avaliação do módulo de deteção de atividade bem como da aplicação de fusão. Procurou-se analisar e comparar diferentes técnicas de determinação de atividade que podem ser consideradas como alternativa à opção aplicada neste projeto; analisaram-se ainda outros trabalhos relacionados na deteção de atividade recorrendo a várias fontes de informação, procurando comparar as diferentes vertentes e mais-valias de cada alternativa.

Abstract

We live in an era in which the average life expectancy has increased, resulting directly in a growth each time more significant of senior population. This phenomenon makes that more and more elderly people live at home alone and have difficulty finding people who can attend and help in their daily lives. It is also recognized that activity and physical exercise are essential to maintain and promote health, in particular in this age group, allowing to, in general, improve the quality of life and people's safety. In this way, many commercial systems have been developed in order to monitor people's activity, in particular elderly, to follow what they do during the day, collecting statistical data of activity, as well as identifying potential dangerous situations (e.g. falls). However, the existing solutions involve, usually, the acquisition of equipment with generally high cost and they base their operation in the analysis of just one kind of information or physical phenomenon (e.g. accelerometer, scene analysis, etc.), which limits, in this way, the results and the monitoring fidelity.

In this context, the work presented here proposes the use of an economic monitoring system, based in the fusion of several kinds of information (e.g. physical activity, user's location, etc.) collected in a smart place set for this purpose. This system combines information collected and processed by several components. It uses, for example, mobile devices that are ordinary nowadays and that are available for reasonable costs (cf. smartphones) and that are equipped with a set of sensors and adequate processing abilities; it also uses other components commonly existents in residential contexts (cf. computer and low resolution cameras oriented to the places to monitor) and that can be integrated and reused in the proposed solution.

This dissertation proposes and describes the physical and logical structure of the PADS (Personal Activity Detection System) system. The developed prototype is organized in several components that are able to collect and fuse different kinds of information: i) an Android application that identifies and records 5 distinctive physical activities (cf. standing, walking, running, lying down and fall); ii) an application developed in C++ to detect the user's presence, based in the face recognition; iii) an application that uses Google Cloud services to consult the user's drug doses that are scheduled; iv) an application in C++ that fuses all the information obtained by the previous modules and allows to determinate the user's activity. It is presented the global architecture of the system and the various involved components, describing with detail all of the algorithms and the details of implementation.

We also proceeded to the evaluation of the detection of activities module, as well as the fusion application. We sought to analyze and compare different techniques of activities determination that could be considered as an alternative to the applied option in this project. We also analyzed other works related to activity detection, recurring to several information sources, trying to compare several sides and strengths of each alternative.

Résumé

. Nous vivons une époque où l'espérance de vie augmente considérablement, tout en ayant comme conséquence directe une croissance importante de la population de personne âgé. Les raisons de ce phénomène font que, de plus en plus de personne âgées vivent seules ayant du mal à trouver des personnes qui puisse surveiller et aider leur vie au quotidien. Aussi, il est reconnu que les activités et l'exercice physique sont essentiels pour maintenir et promouvoir la santé, en particulier dans ce groupe d'âge, ce qui permet en général d'améliorer la qualité de vie et la sécurité des personnes. C'est pour ce fait que l'apparition et le développement de divers systèmes commercial pour la surveillance des personnes sont conçus, en particulier pour les personnes âgées, afin de garder une "empreinte" de leurs quotidien, recueillant les données des activités, tout en permettent d'identifier des situations potentiellement dangereuses (p. ex., les chutes). Cependant, les équipements existantes sont très couteux à l'achat et fonctionne ayant comme base l'analyse d'un seul et unique type d'information ou phénomène physiques, (p.ex. accéléromètre, analyse de scène, etc.), limitant ainsi le résultat et la précision des même.

Dans ce contexte, le travail présenté propose l'utilisation d'un système de surveillance économique, ayant comme base la fusion de différents types d'informations (par exemple, localisation de l'utilisateur, l'activité physique, etc.) recueillie dans un espace intelligent préparé pour cette effet. Ce système assimile les informations réunies et par moyen de plusieurs composants, traites ces mêmes informations. Ce système utilise des appareils mobiles disponibles de nos jours et à des prix accessible au grand public (p.ex. Smartphones, etc.) qui sont équipées d'un ensemble de capteurs et des capacités de traitement d'informations appropriées.

Aussi, ce système utilise aussi d'autres composants qui se trouvent couramment dans les maisons de nos jours (tels qu'ordinateur et caméras basse résolution qui sont dirigée vers les locaux a surveillé) et qui peuvent être intégrées et réutilisées dans la solution proposée.

Cette dissertation propose et décrit la structure physique et logique du système PADS (*Personal Activity Detection System*). Le prototype développé est conçus avec différents composants scrupuleusement choisi pour la recueille et fusion des différents types d'informations: i) Une application Android qui identifie et enregistre 5 activités physiques distinctes (debout, marche, courir, couchée et tomber); ii) Une application développée en C++ pour la détection de présence de l'utilisateur, basée sur la reconnaissance de visage; iii) Une application qui utilise les services "Cloud" de Google pour consulter quelle sont les médicaments, après une correct calendarisation, à

prendre par l'utilisateur; IV) Une application en C++ qui permet la fusion de toutes les informations obtenues par les modules précédents et qui permet de déterminer l'activité de l'utilisateur. Il est présenté l'architecture globale du système et les différents composants impliqués, décrivant avec détail tous les algorithmes et les détails de mise en œuvre.

De plus, il fut effectué une évaluation du module de détection de l'activité, tout comme pour l'application de fusion. Le but de cette même évaluation fut analyser et comparer les différentes techniques de détermination de l'activité qui peut être considérée comme une alternative à l'option appliquée dans ce projet. Nous avons également examiné d'autres travaux liés à la détection de mouvement, à l'aide de plusieurs sources d'information, cherchent à comparer les différents aspects et les plus-values de chaque solution.

Aos Meus Pais

Agradecimentos

O alcance do mestrado é um dos grandes objetivos que tinha definido para a minha carreira profissional na área da informática. A obtenção deste grau fez-me crescer enquanto pessoa e enquanto profissional, no entanto, penso que poderei melhorar ainda mais neste ultimo campo, concretizando novos desafios e alcançando novas etapas.

O sucesso desta etapa é devido não só ao meu esforço, mas também, ao esforço das pessoas que me rodeiam, apoiam, orientam e que me incentivam a continuar. Este tipo de objetivos não é possível de concretizar sem a ajuda e o espirito de equipa dos orientadores em relação ao orientando.

Ao Professor Doutor José Torres, orientador da dissertação, quero agradecer toda a paciência, ajuda e apoio transmitido ao longo destes últimos meses de elaboração deste projeto. Agradeço-lhe também pela partilha de conhecimento e pelo gosto notório que teve em acompanhar-me nesta longa caminhada.

Aos Professores Doutor Rui Silva Moreira, coorientador da dissertação, e Doutor Pedro Sobral quero agradecer, por todo o tempo dispensado para me acompanhar, por me transmitirem força e energia para continuar a trabalhar.

Á minha família, em especial aos meus pais, agradeço todo o carinho transmitido, confiança e por me apoiarem no meu percurso académico que me permitiu chegar até aqui.

Agradeço de uma forma geral a todos os que diretamente ou indiretamente me ajudaram e contribuíram para que eu alcançasse o meu objetivo.

Este trabalho foi financiado por uma bolsa BIC no âmbito do projeto *Safe Home Health Care* (*Interference-Free Home Health-Care Smart Spaces using Search Algorithms and Meta-Reality Reflection*), EIA-EIA/108352/2008, através do FEDER (Fundo Europeu de Desenvolvimento Regional) via Programa COMPETE (programa operacional para a competitividade) e por fundos nacionais através da FCT (Fundação para a Ciência e Tecnologia).

Índice

Capítulo 1: Introdução 1

1.1 Contexto 1

1.2 Objetivos 2

1.3 Estrutura do relatório 4

Capítulo 2: Estado da Arte 6

2.1 Introdução 6

2.2 Sensores 7

2.3 Técnicas de inferência de atividade 7

2.3.1 Métodos baseados em heurísticas 8

2.3.2 Métodos baseados em aprendizagem 9

2.3.3 Inferência probabilística 11

2.4 Software 13

2.4.1 OpenCV 14

2.4.2 Weka 16

2.4.3 Mocapy++ 19

2.5 Trabalho relacionado na determinação de atividade 22

2.5.1 Sistemas de determinação de atividade 22

2.5.2 Análise Comparativa 31

Capítulo 3: Sistema de deteção de atividade 33

3.1 Componentes do PADS 33

3.2 Requisitos 34

3.3 Modelo estatístico de determinação de atividade 36

3.4 Implementação dos Módulos 37

3.4.1 Activity Detection Module 38

3.4.2 Presence Detection Module 52

3.4.3 Fusão de informação 58

3.5 Conclusão 62

Capítulo 4: Avaliação **64**

4.1 Componentes da avaliação do PADS 64

4.2 Avaliação da atividade física 64

4.3 Avaliação da determinação da atividade por fusão 70

4.3.1 Cenário 70

4.3.2 Testes 72

4.4 Discussão dos resultados de deteção de atividade 74

Capítulo 5: Conclusões e Perspetivas de Desenvolvimento **75**

Referências **77**

Anexos **80**

Anexo 1: Modelo da base de dados 81

Anexo 2: Cabeçalho de um ficheiro ARFF 82

Anexo 3: Information Fusion Module – Parameter learning 83

Anexo 3: Information Fusion Module – Parameter Learning 84

Anexo 4: Information Fusion Module – Inference 85

Anexo 4: Information Fusion Module – Inference 86

Anexo 4: Information Fusion Module – Inference 87

Lista de Figuras

FIGURA 1: AMBIENTE INTELIGENTE DO PROJETO SHHC PROJETADO NO OPENSIMULATOR. 2
FIGURA 2: ESQUEMA DO SISTEMA. 3
FIGURA 3: COMPARAÇÃO ENTRE BIBLIOTECAS DE VISÃO POR COMPUTADOR (KAEHLER & BRADSKI 2008) 15
FIGURA 4: GUI DA WEKA 16
FIGURA 5: EXEMPLO DE UM FICHEIRO ARFF (WITTEN ET AL. 2011) 18
FIGURA 6: COMPONENTES DE UMA REDE BAYESIANA NO MOCAPY++ 20
FIGURA 7: DADOS RECOLHIDOS DA PRIMEIRA AVALIAÇÃO DE RECONHECIMENTO DE GESTOS (AVILÉS-ARRIAGA 2011) 21
FIGURA 8: ADDITIVE LOGISTIC REGRESSION ALGORITHM (WITTEN ET AL. 2011) 11
FIGURA 9: EXEMPLO DE REDE BAYESIANA (GHAHRAMANI 1998) 12
FIGURA 10: EXEMPLO DE REDE BAYESIANA DINÂMICA (GHAHRAMANI 1998) 12
FIGURA 11: OUTPUT TÍPICO DOS ACELERÓMETROS PARA OS 5 MOVIMENTOS DETETADOS (J. M. TORRES ET AL. 2012). 26
FIGURA 12: ARQUITETURA DO SEER 30
FIGURA 13: ARQUITETURA DO PADS 33
FIGURA 14: REPRESENTAÇÃO DA REDE BAYESIANA ADAPTADA AO PADS 37
FIGURA 15: ARQUITETURA NA ÓTICA DO ACTIVITY DETECTION MODULE. 39
FIGURA 16: FORMATAÇÃO DE UM FICHEIRO ARFF. 44
FIGURA 17: EXCEÇÃO ENCONTRADA NA TENTATIVA DE CARREGAR O MODELO DE CLASSIFICAÇÃO 46
FIGURA 18: INTERFACE DA APLICAÇÃO DE CLASSIFICAÇÃO DE ATIVIDADES. 49
FIGURA 19: EXEMPLO DE ORGANIZAÇÃO DO FICHEIRO LOG. 49
FIGURA 20: INTERFACE DA APLICAÇÃO DE RECOLHA DE DADOS DE TREINO. 52
FIGURA 21: ARQUITETURA NA ÓTICA DO PRESENCE DETECTION MODULE. 53
FIGURA 22: ARQUITETURA NA ÓTICA DO SERVIDOR. 58
FIGURA 23: DADOS DO ACELERÓMETRO PARA O ESTADO DE PARADO. 66
FIGURA 24: DADOS DO ACELERÓMETRO PARA O ESTADO DE ANDAR. 66
FIGURA 25: DADOS DO ACELERÓMETRO PARA O ESTADO DE CORRER. 67
FIGURA 26: DADOS DO ACELERÓMETRO PARA O ESTADO DE DEITADO. 67
FIGURA 27: DADOS DO ACELERÓMETRO PARA O ESTADO DE QUEDA. 67
FIGURA 28: EXCERTO DO FICHEIRO ARFF. 68

FIGURA 29: ESQUEMA DO CENÁRIO. 71

FIGURA 30: ESQUEMA DO CENÁRIO COM DETERMINAÇÃO DOS MOMENTOS DE CLASSIFICAÇÃO. 72

Lista de Algoritmos

ALGORITMO 1: DECLARAÇÃO DOS SENSORES DO SMARTPHONE 40
ALGORITMO 2: REGISTO DOS SENSORES NO MÉTODO DE INICIALIZAÇÃO 41
ALGORITMO 3: CAPTURA DOS DADOS DOS SENSORES 42
ALGORITMO 4: INSTÂNCIA DA CLASSE COUNTDOWNTIMER.....42
ALGORITMO 5: ALGORITMO QUE FAZ A GESTÃO DAS INSTÂNCIAS EM RELAÇÃO À RAW DATA. 43
ALGORITMO 6: CARREGAMENTO DO HEADER E INICIALIZAÇÃO DE UM OBJETO DO TIPO INSTANCE.....44
ALGORITMO 7: CICLO DE CONSTRUÇÃO DE UMA INSTÂNCIA. 45
ALGORITMO 8: MÉTODO DE CLASSIFICAÇÃO DE CADA INSTÂNCIA. 46
ALGORITMO 9: MÉTODO DE CLASSIFICAÇÃO DE CADA INSTÂNCIA NO SERVIDOR. 47
ALGORITMO 10: FUNÇÃO DE ENVIO DOS DADOS PARA UM WEBSERVICE. 48
ALGORITMO 11: ALGORITMO DE COOLDOWN.....50
ALGORITMO 12: MÉTODO DE CAPTURA DOS DADOS DOS SENSORES POR UM DETERMINADO PERÍODO DE TEMPO. 51
ALGORITMO 13: DECLARAÇÃO DOS CLASSIFICADORES HAARCASCADE. 54
ALGORITMO 14: CRIAÇÃO DO MODELO DE RECONHECIMENTO. 54
ALGORITMO 15: DETEÇÃO DE CARAS E CRIAÇÃO DE UMA IMAGEM BITMAP.....55
ALGORITMO 16: DETEÇÃO DE OLHOS E RECONHECIMENTO DE FACES. 56
ALGORITMO 17: ALGORITMO DE ENVIO DE DADOS PARA O WEBSERVICE. 57
ALGORITMO 18: FUNÇÃO DE INSERÇÃO NA BASE DE DADOS. 59
ALGORITMO 19: DECLARAÇÃO DOS OBSERVED NODES AND HIDDEN NODES. 59
ALGORITMO 20: DETERMINAÇÃO DA ARQUITETURA E LIGAÇÕES ENTRE NÓS. 60
ALGORITMO 21: PARAMETER LEARNING. 61
ALGORITMO 22: DETERMINAÇÃO DO VITERBI PATH.....62

Lista de Tabelas

TABELA 1: RESULTADOS RELATIVAMENTE À SEGUNDA EXPERIÊNCIA 22
TABELA 2: COMPARAÇÃO ENTRE ALGUMAS DAS SOLUÇÕES APRESENTADAS 31
TABELA 3: REQUISITOS FUNCIONAIS DAS APLICAÇÕES MÓVEIS 34
TABELA 4: REQUISITOS NÃO FUNCIONAIS DAS APLICAÇÕES MÓVEIS 35
TABELA 5: REQUISITOS FUNCIONAIS DAS CÂMARAS 35
TABELA 6: REQUISITOS NÃO FUNCIONAIS DAS CÂMARAS 35
TABELA 7: REQUISITOS FUNCIONAIS DA FUSÃO DE INFORMAÇÃO 35
TABELA 8: REQUISITOS NÃO FUNCIONAIS DA FUSÃO DE INFORMAÇÃO 36
TABELA 9: MATRIZ CONFUSÃO DO ALGORITMO LOGITBOOST. 69
TABELA 10: MOMENTOS DE CLASSIFICAÇÃO DO SISTEMA PADS. 73

Abreviaturas e Símbolos

Acel	***Acel****erometer*, este caracteriza um sensor de aceleração.
ADT	***A****ndroid* ***D****evelopment* ***T****ools*, este é um *plugin* para integração da programação para o SO Android no Eclipse IDE.
API	***A****pplication* ***P****rogramming* ***I****nterface*, esta é a uma definição de métodos implementados que podem ser reutilizados por outros programadores através da evocação duma função.
BN	***B****ayesian* ***N****etwork*, este caracteriza uma rede de representação do conhecimento probabilístico.
BSD	***B****erkeley* ***S****oftware* ***D****istribution*, este caracteriza uma licença de *software* gratuito.
C++	Linguagem de Programação Orientada a Objetos.
CPT	***C****onditional* ***P****robability* ***T****able*, este caracteriza uma tabela com as probabilidades condicionais relativas a uma variável.
CPU	***C****entral* ***P****rocessor* ***U****nit*.
Dalvik	Este termo serve para designar a máquina virtual do sistema operativo Android que executa as aplicações.
DBN	***D****ynamic* ***B****ayesian* ***N****etwork*, são redes bayesianas que representam sequências no tempo.

DNBC	***D**ynamic **N**aive **B**ayesian **C**lassifier*, este é um tipo de classificador de redes bayesianas dinâmicas.
EM	***E**xpectation **M**aximization*, este é um método de aproximação do ponto de máxima probabilidade de parâmetros.
Gyrosc	***Gyrosc**ope*.
GPS	***G**lobal **P**ositioning **S**ystem***,*** este é um sensor de localização a nível global.
GSM	***G**lobal **S**ystem for **M**obile Communications***,*** esta é uma tecnologia utilizada para comunicação com os dispositivos móveis**.**
HMM	***H**idden **M**arkov **M**odel***,*** é um modelo estatístico de Markov considerado a mais simples rede bayesiana dinâmica.
HTTP	***H**yper**t**ext **T**ransfer **P**rotocol*, é um protocolo de aplicação para sistemas de informação hipermédia.
IDE	***I**ntegrated **D**evelopment **E**nvironment*, é uma aplicação que facilita a programação de *software*.
IPP	Intel **I**ntegrated **P**erformance **P**rimitives**,** são bibliotecas proprietárias da Intel que permitem otimizar o processamento de aplicações.
Java	É uma linguagem de programação orientada a objetos da *Oracle Corporation*, que necessita de uma *Java Virtual Machine* para executar.
JVM	***J**ava **V**irtual **M**achine***,*** é uma máquina virtual que pode executar *bytecodes* de Java.

LTI,VXL	São bibliotecas que disponibilizam estruturas de dados e algoritmos para a área de visão por computador e processamento de imagem.
Linux, Windows, Mac OS X	Estes termos servem para qualificar sistemas operativos que são propriedade de entidades diferentes.
Magnet	**Magnet**ometer**.**
OpenCV	***O**pen Source **C**omputer **V**ision*, é uma biblioteca de visão por computador criada pela Intel.
OS / SO	***O**perative **S**ystem* / **S**istema **O**perativo.
PADS	***P**ersonal **A**ctivity **D**etection **S**ystem**.***
PHP	***H**ypertext **P**rep**p**rocessor**,*** é uma linguagem de programação *server-side* desenhada para desenvolvimento de páginas *web*.
RGB/BGR	São modelos de cores aditivas (***R**ed*, ***G**reen* e ***B**lue*), que são juntas de forma a perfazerem as cores todas.
SHHC	***S**afe **H**ome **H**ealth **C**are*
SQL	***S**trutured **Q**uery **L**anguage*, é uma linguagem de programação desenhada para gerir informação em bases de dados relacionais.
STL	***S**tandard **T**emplate **L**ibrary*, é uma biblioteca em c++.
UDP	***U**ser **D**atagram **P**rotocol*, é um protocolo usado no transporte de datagramas entre aplicações de computadores diferentes.

WEKA	***W**aikato **E**nvironment for **K**nowledge **A**nalysis*, é uma ferramenta que visa agregar técnicas relacionadas com *data-mining* e algoritmos de aprendizagem máquina.

Capítulo 1: Introdução

1.1 Contexto

A população dos países desenvolvidos e em particular da Europa está em média a envelhecer como resultado do aumento da expectativa de vida e da diminuição da taxa de natalidade. A percentagem da população com idade acima de 65 anos está prevista subir para 53,5% em 2060. Como consequência, o número de pessoas idosas irá exceder a capacidade de resposta da sociedade atual (Luštrek & Kaluža, 2009). Os níveis de atividade, de um modo geral, e da atividade física em particular são importantes na promoção da saúde, no aumento da qualidade de vida e independência do idoso. Como tal, a sua monitorização é fundamental, com vista a detetar situações de perigo e avaliar a necessidade de intervenção. As quedas são um dos problemas mais graves na assistência geriátrica porque são uma das causas fundamentais dos danos e até morte em idosos. Segundo a Organização Mundial da Saúde, entre 28% e 34% da população com 65 anos ou mais sofre pelo menos uma queda por ano (Alonso, Zato, & Pedrero, 2011). O desenvolvimento de aplicações para a área da telemedicina poderá não acabar estes valores, mas permitirá ajudar a colmatar a falta de ajuda que estas pessoas necessitam nestes casos.

A utilização dos *smartphones* tem vindo a aumentar nos últimos anos. Estes dispositivos possuem vários sensores, nomeadamente, de localização, aceleração, visão, temperatura e direção. A existência destes equipamentos criou uma oportunidade para o desenvolvimento de aplicações que podem dar algum tipo de apoio ao utilizador (Kwapisz, Weiss, & Moore, 2010). Estes utilizadores podem ser pessoas que necessitam de cuidados regulares, dependendo do seu estado físico ou psicológico, inerentes aos seus problemas de saúde ou de velhice. Muitas destas pessoas, vivem sozinhas em casa sem qualquer tipo de vigilância e apoio diário. Nestes casos, por exemplo, uma queda poderá refundar numa situação de necessidade de socorro ou mesmo de perigo de vida que só com uma coordenação entre monitorização e assistência local, poderão ser detetadas e acudidas.

O trabalho descrito nesta tese está inserido no projeto *Safe Home Health Care* (*SHHC*) (Moreira, Torres, Sobral, & Soares, 2010). Este projeto desenvolve-se pela necessidade de criar espaços seguros e livres de interferências, no apoio à saúde em casa, a partir de sistemas *off-the-shelf* instalados por forma a conseguir proporcionar um maior conforto e segurança. A construção de espaços inteligentes é apontada como uma solução,

como demonstra a iniciativa de financiamento europeu de investigação em *Ambient Assisted Living*[11]. A Figura 1 apresenta um cenário do projeto SHHC, que serviu de base ao trabalho desta tese, e que envolve a criação de um ambiente de monitorização inteligente da atividade da pessoa residente. O cenário como é possível observar na figura, envolve a utilização de câmaras para captura de imagens e análise visual de presença, assim como de um *smartphone* com vários sensores para registo de atividade. As câmaras são dispostas em locais estratégicos que permitam o registo da presença e atividade diária da pessoa que se pretende monitorizar.

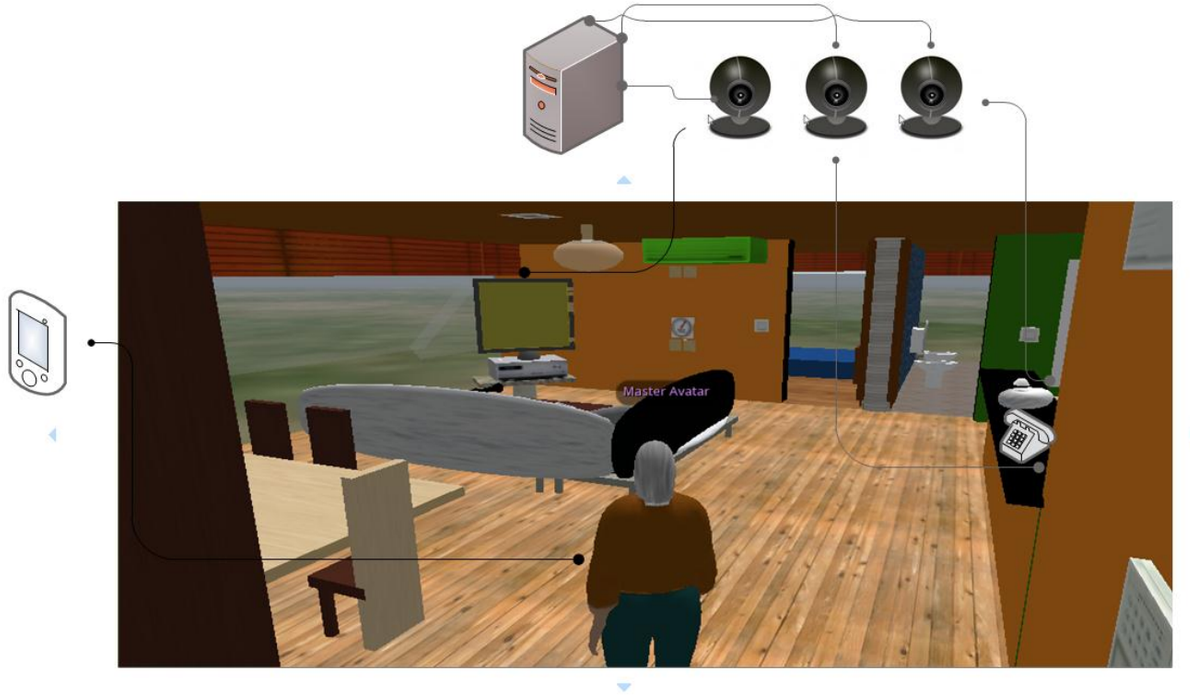

Figura 1: Ambiente inteligente do projeto SHHC projetado no OpenSimulator.

1.2 Objetivos

O objetivo principal deste trabalho consiste na deteção e monitorização de atividade de pessoas em espaços inteligentes, tipicamente domicílios, usando múltiplas fontes de informação sensorial que são agregadas num sistema de inferência que faz a fusão dessa informação. Para alcançar os objetivos propostos, especificou-se uma

[1] http://www.aal-europe.eu/

arquitetura baseada na utilização de um *smartphone*, de câmaras web e de um ou mais servidores para registar e processar os dados recolhidos.

O aspeto principal deste sistema é a deteção da atividade protagonizada pela pessoa residente. Como cada elemento pertencente ao sistema tem uma margem de erro associada à determinação da atividade, ao criar um sistema que conjuga vários sensores e assim vários elementos, pretende-se aumentar a probabilidade de obtenção de resultados mais fiáveis, i.e., obter classificações de atividades sem erros.

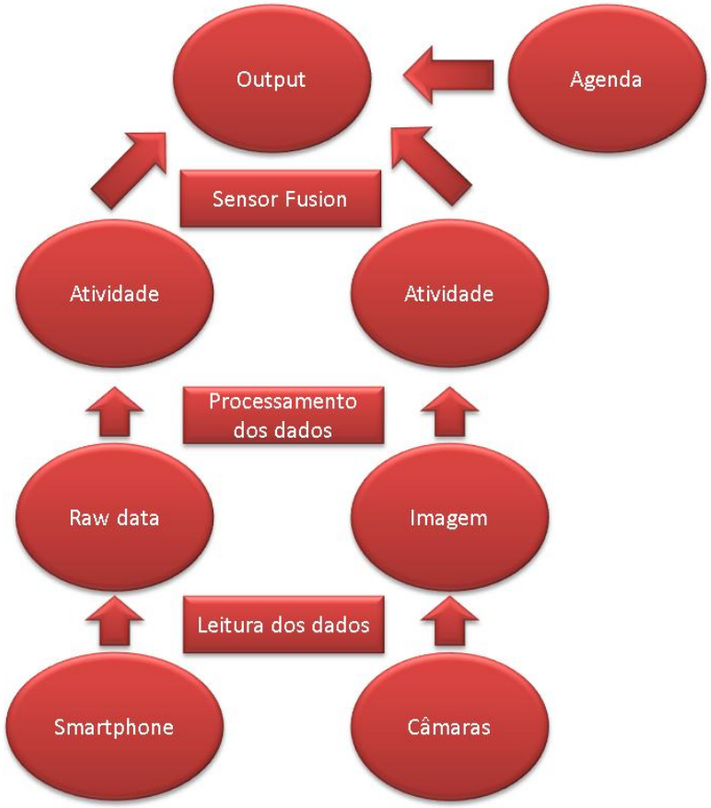

Figura 2: Esquema lógico de processamento de informação do sistema.

A deteção de atividades no dispositivo móvel utiliza algoritmos de aprendizagem máquina oferecidos por bibliotecas existentes(Witten, Frank, & Hall, 2011). Contudo, nem todos os algoritmos são compatíveis com o sistema operativo Android, limitando a sua utilização em dispositivos móveis. Por esta razão, para maximizar a exploração dos algoritmos disponibilizados na biblioteca de *machine learning*, optou-se também por permitir recolher dados no dispositivo móvel e oferecer a opção de tratamento dos dados num servidor.

A deteção de presença através de análise de imagens captadas por câmaras, utiliza algoritmos de uma biblioteca Open Source de visão por computador.

A aplicação que está presente no servidor representa um elemento fulcral neste sistema pois vai fazer não só o registo mas também a fusão da informação enviada pelos outros elementos e permitir assim tomar uma decisão sobre o que realmente está a acontecer no ambiente em questão. Esta fusão de informação é modelizada por um gráfico probabilístico sobre o qual se podem aplicar algoritmos de inferência disponibilizados por bibliotecas de *software* existentes (Paluszewski & Hamelryck, 2010).

O sistema PADS é apenas uma parte do sistema global do projeto SHHC que envolve a utilização de outros sistemas *off-the-shelf* independentes. Todos estes sistemas registam e processam a sua informação mas podem também comunicar e compartilhar informação entre si.

1.3 Estrutura do relatório

Esta dissertação encontra-se organizada em 5 capítulos: a introdução; o estado da arte; o sistema pessoal de deteção de atividade; uma avaliação do sistema e por fim as conclusões.

No primeiro capítulo é feita uma introdução contextualizada, onde se apresenta o enquadramento e a motivação em relação à realização desta dissertação. Neste capítulo são ainda apresentados os objetivos principais deste trabalho.

No seguinte capítulo é descrito o estado da arte, ou seja, é feita uma revisão bibliográfica sobre os sistemas que, de alguma forma, se inserem na mesma área do trabalho que aqui é apresentado. São descritas algumas ferramentas utilizadas neste tipo de projetos, assim como algoritmos orientados para a aprendizagem máquina, inferência probabilística e métodos heurísticos. É ainda elaborada uma comparação entre os sistemas apresentados, e estabelecida uma ligação com o sistema PADS.

No terceiro capítulo apresenta-se o sistema pessoal de deteção de atividades (PADS), nomeadamente com os requisitos, a arquitetura que envolve este sistema e a respetiva implementação.

O quarto capítulo aborda as avaliações efetuadas aos vários módulos ou elementos do PADS. Em particular, efetua-se a avaliação da atividade física realizada no dispositivo móvel, assim como a avaliação da aplicação de fusão, i.e., do *software* desenvolvido para o servidor que vai determinar a atividade do utilizador.

No quinto e último capitulo, detalham-se as principais conclusões, apontando-se as principais contribuições do trabalho desenvolvido, bem como objetivos de trabalho futuro.

Capítulo 2: Estado da Arte

Este capítulo de estado da arte está sistematizado de forma a introduzir algumas noções sobre sensores e algumas ferramentas que são utilizadas no contexto da determinação de atividades. Os sensores mencionados estão particularmente relacionados com a determinação de atividades que envolvem movimento, como são exemplo o acelerómetro, o giroscópio e o magnetómetro. As ferramentas que estão descritas neste capítulo estão inseridas no contexto de aprendizagem máquina como é o caso da Weka e de visão por computador que é o caso do OpenCV. São abordadas ainda algumas técnicas de inferência de atividade, umas que são utilizadas neste trabalho e outras usadas em projetos similares. Estas técnicas abrangem esquemas baseados em heurísticas, em aprendizagem e inferência probabilística. Por fim são apresentados alguns trabalhos relacionados e é feita uma análise comparativa entre os mesmos e o PADS em algumas características que se assemelham.

2.1 Introdução

Os sistemas sensíveis ao contexto compreendem um módulo de aquisição em que são utilizados componentes de *hardware* para obter informação sobre um ambiente ou sobre uma grandeza física em específico, como a aceleração. Estes provêm de vários tipos (e.g. dispositivos móveis) e utilizam múltiplos tipos de sensores, adequados ao objetivo do sistema. Com a evolução tecnológica nos últimos anos temos cada vez mais sistemas ubíquos disponíveis para obter informação de modo fácil. Estes sistemas possuem também uma fase de processamento algorítmico em que a informação obtida do ambiente é tratada e processada por um algoritmo. Existem variadas técnicas disponíveis para processamento dos dados no entanto os modelos baseados em aprendizagem e inferência probabilística predominam nas soluções correntes.

Podemos identificar neste tipo de sistemas um agente inteligente que ajuda na tomada de decisões, através da interpretação da perceção adquirida pelos sensores traduzida em dados.

2.2 Sensores

Os sensores são dispositivos que traduzem uma grandeza percecionada geralmente em sinais de natureza elétrica, e servem para mensurar aspetos relacionados com a movimentação, temperatura, som, visão, entre outros. Atualmente consegue-se ter fácil acesso a este tipo de equipamento pois, devido à crescente evolução da tecnologia os dispositivos ubíquos têm incorporado alguns destes sensores, como é exemplo o acelerómetro. Este pertence ao grupo dos sensores de movimento, onde existe também o giroscópio e a bússola. O acelerómetro determina a aceleração existente num corpo, enquanto os outros dois sensores estão ligados ao sentido de orientação do movimento. Atualmente os acelerómetros de 3 eixos são os mais usados pois, possibilitam determinar a aceleração e a sua direção com uma maior precisão que os antigos de 2 eixos mas, ambos podem ser empregues para a determinação de atividades físicas através de dispositivos ubíquos. Os acelerómetros hoje em dia podem ser adquiridos a um baixo custo e estão presentes na maioria dos *smartphones*, assim como, por exemplo, em alguns relógios de pulso vocacionados para a prática do desporto.

No conjunto dos sensores de visão existe a câmara de vídeo. Com recurso a uma câmara consegue-se fazer a observação de um ambiente através de imagens. Este tipo de sensor é utilizado quando queremos identificar um elemento num ambiente, que pode ser um objeto ou mesmo uma pessoa. Para identificação de uma pessoa é necessário ter em conta a qualidade da câmara para uma melhor precisão nos resultados, para isso é necessário ter em atenção alguns aspetos como o *frame rate* e à resolução que a câmara melhor suporta de forma a obter um bom desempenho. Estes dispositivos são muito utilizados na videovigilância, assim como na monitorização de espaços inteligentes para através da análise de imagem perceber o que se passa no ambiente.

A incorporação destas tecnologias neste tipo de dispositivos ubíquos criou um grande meio de desenvolvimento e investigação para construção de sistemas inteligentes adequados às necessidades.

2.3 Técnicas de inferência de atividade

Existem várias técnicas devidamente fomentadas e estudadas para determinação de soluções por inferência para vários problemas na área da inteligência artificial. Algumas das técnicas mais utilizadas estão descritas nesta subsecção e são baseadas em heurísticas, esquemas de classificação ou em inferência probabilística.

2.3.1 Métodos baseados em heurísticas

Os sistemas baseados em heurísticas têm a capacidade de usar, de forma muito direta, o conhecimento experimental adquirido por especialistas na área abordada. Isto é particularmente importante para gerir a complexidade ou mesmo a possibilidade de faltar algum tipo de informação, pois este tipo de método depende inevitavelmente de um conjunto de dados completo. Para a procura de uma solução no espaço de estados possíveis são utilizadas regras que são essenciais para determinar a melhor decisão para o problema ou lida com dificuldades existentes, no entanto podem ser necessários recursos de explanação para dar suporte a um processo de correção. Este processo envolve o ajuste do conteúdo das regras ou mesmo da estrutura, de forma a corrigir erros estejam a interferir no correto funcionamento do método (Luger, 2002).

Existem agentes que aplicam sistemas de regras baseados em heurísticas como são o caso do *Simple Reflex Agent* e do *Model-Based Agent* (Russell & Norvig, 2003). O *Simple Reflex Agent* é muito limitado a nível de inteligência pois não possui memória e por isso apenas faz a execução de regras de acordo com o ambiente, ou seja, faz a interpretação dos dados de input e tenta encontrar a regra que possui na lista que mais se adequa à situação. Quando encontra essa regra apenas a põe em ação, sem qualquer condição de impedimento.

Um sistema fortemente heurístico pode falhar, ou por encontrar um problema que não se encaixe em nenhuma regra disponível ou por aplicar uma regra heurística erradamente a uma situação impropria. Os humanos especialistas não têm esses problemas, pois possuem um entendimento teórico mais profundo do domínio do problema, que permite aplicar inteligentemente as regras heurísticas ou recorrer ao raciocínio a partir de "princípios elementares" em novas situações. A introdução de um sistema baseado em modelos tenta de alguma forma refletir esta flexibilidade e poder para os sistemas. O *Model-based Reflex Agent* contém alguma inteligência devido à memória que utiliza para guardar, por exemplo, o último estado do ambiente, assim como a ultima ação. Ao ter uma nova perceção do ambiente, esta vai ser comparada com o antigo estado para verificar se existem alterações e atualizar o mesmo se necessário. Com esta informação guardada em memória este tipo de agentes consegue através de uma nova perceção, executar um processo na tentativa de predição do que irá acontecer a seguir se tomarmos uma determinada ação e consequentemente tomar a melhor decisão sobre os atos (Russell & Norvig, 2003).

Um dos trabalhos relacionados com este projeto presente na seção 2.5.1 de (Gonçalves, Torres, Sobral, & Moreira, 2009), apresenta uma solução de monitorização de pacientes em casa, em que nos seus algoritmos está

implementado um agente do tipo *Model-based*, isto porque contém um conjunto de regras baseadas no ambiente esperado e contém em "memória" os conhecimentos dum passado próximo.

2.3.2 Métodos baseados em aprendizagem

A ideia por trás da utilização do conceito de aprendizagem não é apenas para saber como atuar, mas também tem como intuito aumentar a habilidade do agente para atuar no futuro. A aprendizagem ocorre através de como o agente observa as suas interações com o mundo e os seus próprios processos de decisão. A aprendizagem indutiva é uma classe de algoritmos para aprendizagem supervisionada determinística baseada em conjuntos de hipóteses aprendidas com recurso a exemplos de treino. Este tipo de algoritmos fornecem uma hipótese que é uma boa aproximação da predição correta e que responde bem quando são problemas generalizados, mas isto levanta o problema da indução. Este problema baseia-se na generalização sobre o número de observações feitas para cada hipótese aprendida assim como na pressuposição que a sequência de eventos que vão acontecer no futuro é a mesma que aconteceu no passado. Para resolver parcialmente este problema utilizam a teoria da aprendizagem computacional, em que o principio adjacente diz: "*any hypothesis that is seriously wrong will almost certainly be "found out" with high probability after a small number of examples, because it will make an incorrect prediction. Thus, any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong: that is, it must be probably approximately correct*". (Russell & Norvig, 2003)

Neste contexto de aprendizagem máquina existe o processo de meta-aprendizagem que é uma forma de aprender a aprender, ou seja, este algoritmo utiliza a experiência para mudar aspetos do algoritmo de aprendizagem ou o método em si para que o seu desempenho aumente ao adquirir experiência adicional. Os algoritmos de meta-aprendizagem são utilizados para automatizar o processo de decisões de conceção do algoritmo de aprendizagem utilizado, como o ajuste de parâmetros de forma a otimizar dinamicamente através da experiência obtida pelo período de aprendizagem. (Schaul & Schmidhuber, 2010)

Existem muitos métodos de *machine learning* atualmente, alguns deles usam atributos nominais naturalmente como é o caso das árvores de decisão mas, mesmo assim conseguem utilizar atributos numéricos incorporando-os diretamente na árvore, utilizando regras de esquemas de indução ou através da passagem dos atributos numéricos a nominais. As regras de esquemas de indução são produzidas através de métodos baseados em heurísticas de forma a conseguirem processar uma vasta quantidade de informação e executar rapidamente. Quando o resultado que se pretende determinar ou a classe é do tipo numérico assim como todos os atributos, a

utilização da técnica de regressão linear deve ser tomada em conta. O objetivo é expressar a ideia de que a classe ou resultado é uma combinação linear de todos os atributos com determinados pesos:

$$x = w_0 + w_1a_1 + w_2a_2 + \ldots + w_ka_k$$

O x representa a classe, enquanto o vetor $\boldsymbol{w}$ expõe os pesos calculados a partir dos dados de treino e o vetor $\boldsymbol{a}$ o valor do atributo. A regressão linear pode ser facilmente utilizada para classificação em domínios com atributos numéricos, assim como qualquer outra técnica de regressão pode ser utilizada para estes fins. Existe um estratagema que é fazer uma regressão para cada classe, definindo a saída igual a um para instâncias de treino que pertencem à classe e zero para as outras, com isto consegue-se obter uma expressão linear para cada classe. Segundo (Witten et al., 2011), a utilização deste método de regressão pode ter dois inconvenientes: os valores que ele apresenta não são probabilidades apropriadas porque podem cair fora do intervalo de 0 a 1 e porque assume que os erros não são apenas estatisticamente independentes, mas também são normalmente distribuídas com o mesmo desvio padrão, coisa que não acontece visto que as observações têm sempre de assumir o valor 0 e 1. Uma técnica estatística relacionada chamada de *logistic regression* não sofre destes problemas referidos pois constrói um modelo linear com base em uma meta variável transformada. (Witten et al., 2011)

No projeto aqui apresentado é utilizado um algoritmo específico designado por LogitBoost que utiliza a técnica *additive logistic regression*, em que aplica um modelo de regressão que se vai ajustando a cada iteração que é feita para obter uma versão ponderada da mesma. A Figura 3 apresenta o algoritmo que está presente no LogitBoost para um problema com apenas duas classes de classificação, mas este modelo é completamente adaptável a problemas com mais classes para classificação. O perigo de *overfitting* está presente neste tipo de algoritmos mas pode ser combatido com a redução das predições e com o ajuste do número de iterações através da técnica de *cross-validation.* Esta técnica serve para ajustar o modelo de acordo com os seus parâmetros e disponibilidade em termos de conjunto de treino, para que os dados de validação se enquadrem no mesmo (Witten et al., 2011).

```
model generation
For j = 1 to t iterations:
    For each instance a[i]:
        Set the target value for the regression to
            z[i] = (y[i] – p(1 | a[i])) / [p(1 | a[i]) × (1 – p(1 | a[i])]
        Set the weight of instance a[i] to p(1 | a[i]) × (1 – p(1 | a[i])
    Fit a regression model f[j] to the data with class values z[i] and weights w[i].
classification
Predict first class if p(1 | a) > 0.5, otherwise predict second class.
```

Figura 3: Additive logistic regression algorithm (Witten et al., 2011)

2.3.3 Inferência probabilística

Os sistemas que adotam o uso da inferência probabilística para determinar algo, utilizam também variáveis aleatórias que podem ser definidas como uma variável de nível quantitativo em que o seu valor depende de fatores aleatórios, como por exemplo a ação de atirar uma moeda ao ar. Ao executarmos este experimento, conseguimos obter uma variável aleatória em que o seu resultado é a face que fica virada para cima, seja ela cara ou coroa. Neste caso em concreto existe a probabilidade de 50% cada uma das faces ficar virada para cima em condições normais, ou seja, sem fatores adversos que possam de alguma forma modificar a probabilidade de um resultado ocorrer. A inferência compreende-se como um processo de inferir um resultado a partir de várias observações, que parte de um processo de raciocínio de acordo com as experiências obtidas. Ao ser utilizada uma distribuição conjunta que especifica as probabilidades de todos os eventos atómicos, esta abordagem fornece um método simples para calcular a probabilidade de qualquer proposição de forma a facilitar o processo de inferir o resultado de acordo com as variáveis observáveis. O cálculo da inferência utilizando este tipo de distribuição torna-se, de certa forma, impraticável se utilizado em modelos que envolvem muitas variáveis aleatórias.

Para uma melhor interpretação do sistema são utilizadas as redes bayesianas, que são simplesmente um modelo gráfico para representação de um conjunto de variáveis e a sua distribuição de probabilidade conjunta usando um grafo acíclico dirigido (Paluszewski & Hamelryck, 2010).

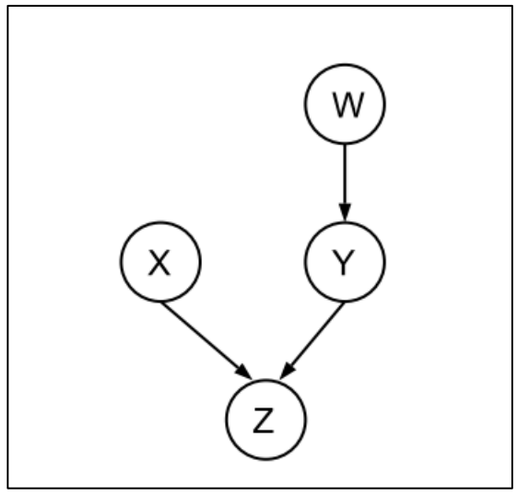

Figura 4: Exemplo de rede bayesiana (Ghahramani, 1998)

A componente gráfica das redes bayesianas, como é exemplo a figura acima, não permite apenas ao utilizador perceber quais as variáveis que influenciam outras, mas também serve como suporte para alcançar eficiência computacional (Ghahramani, 1998).

No projeto aqui desenvolvido neste documento aplicam-se as redes bayesianas dinâmicas que são redes bayesianas que representam sequências, como séries temporais de fala ou sequências biológicas (Paluszewski & Hamelryck, 2010). Na modelação de series no tempo, podem ser observadas variáveis em diferentes pontos desse mesmo tempo, com a convicção que um evento agora pode influenciar um evento no futuro, mas não ao contrário. Como é possível decifrar na Figura 5, estas redes contêm arcos direcionais com um fluxo dirigido no tempo (Ghahramani, 1998).

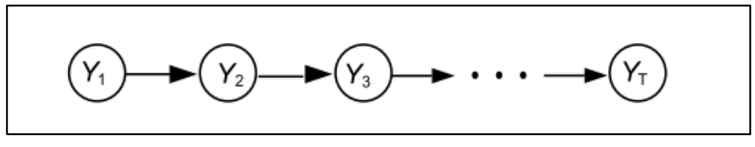

Figura 5: Exemplo de rede bayesiana dinâmica (Ghahramani, 1998)

Num exemplo mais complexo teríamos alguns nós considerados os *observed nodes* e outros os *hidden nodes*, que são as variáveis observadas que obtêm um valor para cada instante de tempo (*slice*) e variáveis que irão ser inferidas através das variáveis observadas também para cada instante de tempo. Para o cálculo da inferência relativamente a nós que influenciam outros é necessário uma fase de aprendizagem. Esta fase envolve um

conhecimento *à priori* da distribuição da probabilidade sobre a estrutura e parâmetros do modelo, para depois ser atualizado com a informação obtida e determinar as posteriores probabilidades relativamente à estrutura e parâmetros do mesmo modelo (Ghahramani, 1998).

Um bom exemplo de uma DBN são os *Hidden Markov Models* (HMM) que são considerados as mais simples. Este tipo de redes foram aplicadas com grande sucesso a um grande número de problemas em várias áreas, como, por exemplo, na área da bioinformática, as DBN's são especialmente relevantes por causa do carácter sequencial das moléculas biológicas (Paluszewski & Hamelryck, 2010).

Para determinar a inferência é possível usar entre dois tipos de algoritmos, de inferência exata ou aproximada, no entanto através da utilização da ferramenta Mocapy++ neste projeto foi empregado o cálculo da probabilidade distribuída aproximada de nós escondidos, através da técnica *Markov chain Monte Carlo* (MCMC), designada *Gibbs sampling.* Este tipo de métodos é atrativo porque permite a utilização de arquiteturas de redes complexas e uma grande variedade de probabilidades distribuídas.

Depois de construída uma rede bayesiana é possível a utilização do algoritmo *expectation maximization* (EM) para fazer a aprendizagem dos parâmetros da rede. Para atingir esta finalidade é necessário carregar os dados recolhidos dos nós observados e ocultos (conjunto de treino), assim como dados que informam o tipo de nós. Na aprendizagem e treinamento dos dados propriamente ditos, o método EM divide-se em dois passos, o E-step que faz a inferência dos nós ocultos utilizando a configuração atual dos parâmetros da DBN, e o M-step utiliza os valores inferidos dos nós ocultos para atualizar os parâmetros da DBN. Depois de estes dois passos convergirem atingimos o objetivo que é produzir uma estimativa do ponto de máxima probabilidade dos parâmetros.

2.4 Software

Esta secção é dedicada à descrição de algumas bibliotecas essenciais que foram utilizadas no projeto apresentado neste documento assim como em alguns projetos relacionados. Estas bibliotecas pertencem a diferentes áreas como visão por computador, aprendizagem máquina e inferência estatística.

2.4.1 OpenCV

A análise e interpretação da atividade humana captada em vídeo requer a utilização de uma biblioteca de visão por computador para tratamento de cada trama do vídeo (*frame*). No projeto aqui apresentado, este processo foi realizado recorrendo à biblioteca OpenCV, devido à sua evolução e do elevado reconhecimento obtido desde do seu aparecimento em 2000.

Esta biblioteca nasceu numa iniciativa da Intel de criar aplicações que usassem intensivamente os seus CPUs. Com este propósito, lançou vários projetos de desenvolvimento na área da visão por computador, utilizando uma metodologia usada no MIT. Esta visava a passagem de código de aluno para aluno, em que cada um acrescentava novos conteúdos melhorando o projeto a nível arquitetural e funcional. A Intel utilizou esta mesma ideia e criou o OpenCV aplicando uma infraestrutura de visão por computador disponível para todos.

O OpenCV é uma biblioteca *open source* de visão por computador, baseada numa licença BSD, que permite a sua utilização para fins académicos ou mesmo comerciais. Esta biblioteca está escrita em C e C++ e consegue executar sobre qualquer sistema operativo Linux, Windows e Mac OS X. Foi desenvolvida com o intuito de obter eficiência computacional com um forte direcionamento para as aplicações em tempo real (Kaehler & Bradski, 2008).

Um dos objetivos do OpenCV é providenciar uma infraestrutura simples que ajude as pessoas a construir uma solução sofisticada rapidamente. A biblioteca contém mais de 500 funções espalhadas por várias áreas na visão. Algumas empresas como a IBM, a Intel, a Microsoft, a SONY, a Siemens e a Google, assim como centros de pesquisa como Stanford, MIT, CMU, Cambridge e INRIA, utilizam esta biblioteca para construção de produtos, o que mostra, de algum modo, o seu valor. Com vista a comprovar a superioridade do OpenCV em relação a outras bibliotecas de visão por computador, os autores (Kaehler & Bradski, 2008) elaboraram 4 testes para determinar qual o seu desempenho e comparar. Foram testadas a LTI, a VXL e a OpenCV com e sem as Intel Integrated Performance Primitives (IPP), como pode ser observado na Figura 6.

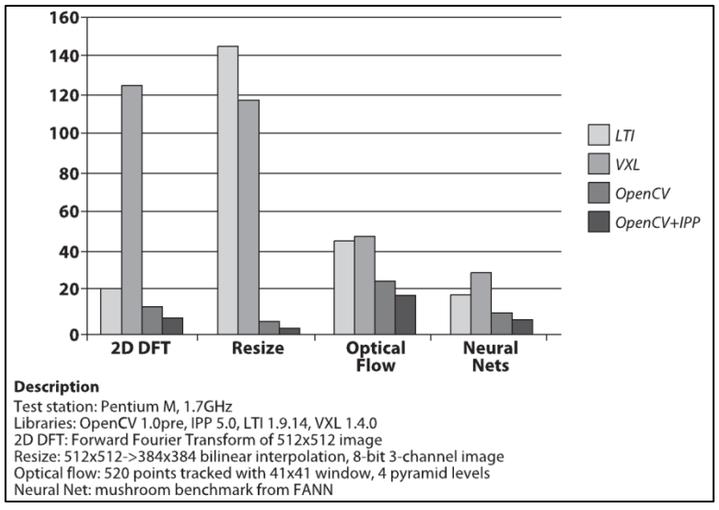

Figura 6: Comparação entre bibliotecas de visão por computador (Kaehler & Bradski, 2008)

Estes resultados demonstram que o OpenCV utilizando as IPP obtêm o melhor desempenho, no entanto o OpenCV mesmo sem as IPP obtêm na mesma melhor resultado que qualquer uma das outras bibliotecas de visão.

Esta biblioteca iniciada pela Intel possui uma arquitetura que pode ser dividida em 8 elementos distintos, que são os seguintes:

- Core - Módulo de definições básicas de dados e funções básicas para outros módulos;
- Imgproc - Processamento de imagens e de transformações geométricas de imagens;
- Vídeo - Análise de vídeo que inclui algoritmos de *background subtraction*, seguimento de objetos e estimativa de movimentos;
- Calib3d - Algoritmos geométricos básicos de múltiplas vistas, calibração da câmara e elementos de reconstrução 3D;
- Features2d – Deteção e extração de características;
- Objdetect: - Deteção de objetos e de instâncias de classes (carros, pessoas, etc.) predefinidas;
- HighGUI – Interface de captação de vídeo;
- Gpu – Algoritmos *GPU-Accelerated* de diferentes módulos do OpenCV. (“OpenCV API Reference - Introduction,” 2012)

Como já foi referido, esta foi a biblioteca utilizada neste projeto, no entanto, nem todos os componentes descritos na arquitetura foram utilizados, sendo os mais utilizados a componente Objdetect, Vídeo e a HighGUI cujas funções são a identificação de classes, o tratamento de imagem e a recolha da mesma, respetivamente.

2.4.2 Weka

A área de aprendizagem máquina enquadra-se na área da inteligência artificial e está sobretudo focada na aprendizagem de padrões e regras, através da análise de conjuntos de dados. A experiência obtida mostra que não existe nenhum modelo, neste âmbito, que seja o mais apropriado para resolver todos os tipos de problemas de *data mining*, sendo esta área, por esse motivo, considerada uma ciência experimental (Witten et al., 2011).

A Weka é uma coleção de algoritmos de aprendizagem máquina para tarefas de *data mining* e de ferramentas de pré-processamento de informação. Esta ferramenta fornece um interface que possibilita aos utilizadores compararem diferentes métodos e identificar qual o mais apropriado para o problema. A Weka foi desenvolvida pela Universidade de Waikato na Nova Zelândia, e por isso seu nome ficou como *Waikato Enviromment for Knowledge Analysis*. Esta biblioteca foi construída em Java e distribuída com a licença GNU *General Public license*, sendo suportada por todos os sistemas operativos que possuam a Java Virtual Machine, como são exemplo o Windows, Linux e Macintosh. Isto permitiu que um interface uniforme fosse construído com todos os algoritmos de aprendizagem, métodos de pré-processamento e pós-processamento, assim como para a avaliação do resultado dos conjuntos de dados (Witten et al., 2011).

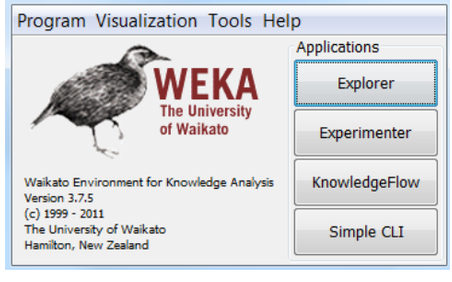

Figura 7: GUI da Weka

A Weka fornece a implementação de vários algoritmos de aprendizagem que podem ser aplicados a um conjunto de dados, incluindo também ferramentas de transformação desses mesmos dados. É possível efetuar o pré-processamento de dados, alimentar um esquema de aprendizagem com esses dados, analisar o resultado do

classificador e o seu desempenho. Este package possui métodos para todos os problemas *standard* de *data mining* de regressão, classificação, *clustering*, *association rule mining* e seleção de atributos. Uma mais-valia desta ferramenta está relacionada com a facilidade no tratamento dos dados, na visualização e com as técnicas de pré-processamento que disponibiliza. (Witten et al., 2011).

Existem várias formas de utilização da Weka, a mais vulgar é aplicar um método de aprendizagem a um conjunto de dados e analisar o seu output para instruir-se mais sobre a informação. Outra forma é usar modelos aprendidos para gerar novas instâncias de dados e como terceira hipótese é usar diferentes métodos para comparar o seu desempenho de forma a escolher o que obtém uma melhor precisão na inferência. No ambiente Weka os métodos de aprendizagem são denominados como classificadores e as ferramentas de pré-processamento designam-se filtros. Uma forma fácil de utilização desta ferramenta é através do seu *graphical user interface* disponibilizado que, através da aplicação "explorer" temos facilidade de uso através da seleção de menus e campos. Utilizando um conjunto de dados num ficheiro ARFF podemos utilizar este interface para, rapidamente criar uma árvore de decisão, por exemplo, pois existem muitos outros algoritmos que podem ser aplicados (Witten et al., 2011).

Um ficheiro do tipo ARFF contém um conjunto de dados pré-formatados e estruturados para uma correta interpretação por parte da Weka. Este tipo de ficheiro possui uma variável de relação, atributos e valores, em que cada linha de valores contém dados para todos os atributos. De forma a ilustrar o que foi dito é apresentada a Figura 8 que demonstra um exemplo de um ficheiro ARFF (Witten et al., 2011).

```
% ARFF file for the weather data with some numeric features
%
@relation weather

@attribute outlook { sunny, overcast, rainy }
@attribute temperature numeric
@attribute humidity numeric
@attribute windy { true, false }
@attribute play? { yes, no }

@data
%
% 14 instances
%
sunny, 85, 85, false, no
sunny, 80, 90, true, no
overcast, 83, 86, false, yes
rainy, 70, 96, false, yes
rainy, 68, 80, false, yes
rainy, 65, 70, true, no
overcast, 64, 65, true, yes
sunny, 72, 95, false, no
sunny, 69, 70, false, yes
rainy, 75, 80, false, yes
sunny, 75, 70, true, yes
overcast, 72, 90, true, yes
overcast, 81, 75, false, yes
rainy, 71, 91, true, no
```

Figura 8: Exemplo de um ficheiro ARFF (Witten et al., 2011)

Outra aplicação fornecida pelo GUI é o "Knowledge flow". Esta permite ao utilizador desenhar as configurações para *streaming* de processamento de dados. Isto permite especificar todo o esquema de classificação como de pré-processamento para avaliação dos métodos e visualização dos módulos. A aplicação "Experimenter" foi desenvolvida para ajudar a responder a questões práticas básicas quando se aplicam técnicas de classificação e regressão, de forma a transmitir ao utilizador quais os melhores algoritmos na resolução do seu problema. Esta opção fornece uma interface que permite automatizar o processo de execução de classificadores e filtros com diferentes parâmetros de configuração, para fornecer estatísticas de desempenho e realizar testes de significância. Por último, a aplicação "Simple CLI" disponibiliza um terminal para digitar comandos para efetuar todo o tipo de operações disponibilizadas pela Weka. Destas aplicações mencionadas, as pessoas, normalmente, escolhem o "Explorer" devido à menor complexidade envolvida na criação dos modelos de aprendizagem. (Witten et al., 2011).

Os modelos de aprendizagem são objetos passíveis de extração após um conjunto de dados ser importado na Weka e ser feita a aplicação de um algoritmo de aprendizagem, assim como técnicas de pré-processamento. Estes modelos são utilizados em aplicações através de uma *framework* disponibilizada pela Weka que permite

fazer a importação deste mesmo modelo de aprendizagem. Isto possibilita construir uma instância para fornecer ao componente classificador de modo obtermos um resultado esperado. Este resultado é um dos outputs disponibilizado no conjunto de dados fornecidos para aprendizagem (Witten et al., 2011).

2.4.3 Mocapy++

A fusão de informação para obtenção de um resultado unificado, é uma tarefa que envolve a recolha de dados de diferentes fontes e sua interpretação. No caso do presente projeto, o objetivo é chegar a uma informação integrada sobre a atividade humana executada no ambiente inteligente designado, com a combinação de informação relativamente às câmaras, ao *smartphone* e a outros pequenos sistemas que podem coexistir.

Com essa finalidade foram utilizadas as redes bayesianas dinâmicas que são redes dependentes de um raciocínio probabilístico. Uma ferramenta que implementa este tipo de redes é o Mocapy ++, que é utilizado para inferência e aprendizagem em BNs, especialmente em dinâmicas. O propósito principal do Mocapy++ é permitir ao utilizador que este se foque mais no modelo de probabilidades, usando os algoritmos de inferência e aprendizagem disponibilizados pela biblioteca.

O Mocapy++ está implementado como uma biblioteca em C++, com alto nível modular e onde novos tipos de nós podem ser adicionados facilmente. Esta ferramenta é melhorada através da utilização de outras bibliotecas como a BOOST C++ que efetua a serialização dos objetos permitindo que o Mocapy++ seja suspenso e mais tarde recuperado para o estado em que se encontrava. Em conjunto com esta ferramenta ainda trabalha a biblioteca LAPACK para execução de rotinas de álgebra linear e o CMAKE para localizar *packages* e configurar o sistema de *builds* (Paluszewski & Hamelryck, 2010).

Para o armazenamento de informação interna, normalmente, são utilizadas estruturas da Standard Template Library (STL), mas neste caso o Mocapy++ necessita de algo com mais suporte a arrays multidimensionais devido ao uso, por exemplo, nas tabelas de probabilidade condicional (CPT). Por conseguinte o mocapy++ tem a sua própria implementação para matrizes multidimensionais designada de MDArray. A classe MDarray faz a alocação dinâmica de dimensões e é usada também para guardar dados de treino (Paluszewski & Hamelryck, 2010)

Para especificar uma DBN no Mocapy++, considera-se uma sequência de observações em que cada posição desta é identificada por *n* variáveis aleatórias e chamada de fatia (*slice*). Cada *slice* de uma sequência pode

ser considerada uma BN ordinária. Uma rede bayesiana dinâmica nesta ferramenta é definida por 3 componentes que são, um conjunto de nós que representam as variáveis para ter uma fatia, as conexões entre nós dentro de uma fatia (*intra edge*) e as conexões entre fatias consecutivas (*inter edge*).

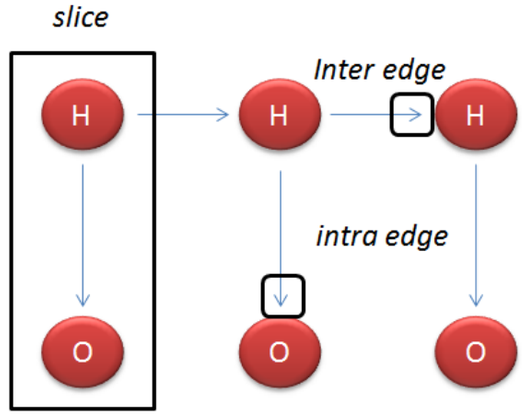

Figura 9: Componentes de uma rede bayesiana no Mocapy++

Para determinar a inferência, como por exemplo calcular a probabilidade distribuída aproximada de nós escondidos, o Mocapy++ utiliza a técnica *Markov chain Monte Carlo* (MCMC), designada *Gibbs sampling*. Este tipo de métodos é atrativo porque permite a utilização de arquiteturas de redes complexas e uma grande variedade de probabilidades distribuídas. A aprendizagem de parâmetros de uma DBN com nós escondidos é feita através do método *expectation maximization* (EM), que faculta uma aproximação do ponto de máxima probabilidade dos parâmetros (Paluszewski & Hamelryck, 2010).

No artigo (Aviles-Arriaga, 2003), são utilizados *Dynamic Naive Bayesian Classifiers* (DNBC), que são um tipo especial de redes bayesianas dinâmicas que não possuem uma limitação observada nas HMM nas quais o número de parâmetros necessários para definir o modelo aumenta exponencialmente à medida que aumenta o número de estados ou variáveis observadas. Este artigo utiliza estes classificadores para o reconhecimento visual de cinco gestos dinâmicos, que pode incluir relações com a mão direita, a cara e o torso. Ao utilizar uma DNBC, verificaram que podem utilizar menos dados de treino para obter sucesso no reconhecimento, do que utilizando uma HMM.

Noutro artigo Avilés-Arriaga(2011) compara diretamente as DNBCs com as HMMs para reconhecimento de gestos. Os autores reforçam a ideia de que é necessário uma clareza na descrição do conhecimento para uma melhor compreensão na execução dos gestos e no processo de reconhecimento, evidenciando também que as NBCs conseguem fazer isso melhor que as HMMs, devido a estas implicarem dependência condicional entre atributos. Para efetuarem a comparação entre o desempenho dos métodos para classificação e aprendizagem fizeram experiências com dados recolhidos de 15 pessoas. Estes dados englobam 9 movimentos gestuais dinâmicos que têm como objetivo interagir com um robot, o qual é um agente móvel. A primeira experiência consistiu em, através dos dados recolhidos de uma pessoa, tentar fazer o reconhecimento com a mesma, com isto obtiveram a tabela que é possível observar na Figura 10.

	Posture-motion models						Motion models					
	Average error rate (%)		Total				Average error rate (%)		Total			
			Training time (*Sec*)		Number of iterations				Training time (*Sec*)		Number of iterations	
Number of states	DNBCs	HMMs	DNBCs	HMMs	DNBCs	HMMs	DNBCs	HMMs	DNBCs	HMMs	DNBCs	HMMs
3	3.81	3.02	36.77	322.82	13964	25007	28.43	28.71	28.26	44.28	13238	16548
6	2.37	2.6	126.7	1047.3	23305	28449	14.28	18.66	99.63	134.03	24719	25126
9	1.94	2.3	288.19	2344.5	29296	31676	13.28	16.73	217.35	303.96	32306	32707
12	1.78	2.18	516.63	4360.37	33270	34540	13.03	15.75	380.59	556.04	37183	38472
15	1.78	2.14	778.02	7805.28	34773	35940	13.42	15.84	599.79	868.54	41116	41258
18	1.72	2.2	1135.12	11854.8	36885	38696	13.82	16.09	897.48	1272.06	44718	44166

Figura 10: Dados recolhidos da primeira avaliação de reconhecimento de gestos (Avilés-Arriaga, 2011)

A experiência seguinte passava por utilizar os dados da pessoa da primeira experiência para treino e testar o reconhecimento dos gestos através das outras 14 pessoas, recolhendo 2 amostras de cada gesto de cada uma das 14 pessoas. Ainda nesta experiência, fizeram o teste ao contrário, ou seja, utilizando os dados de teste construíram os dados de treino e, a partir dos dados da pessoa da primeira experiência criaram os dados de teste. Os resultados foram traduzidos em taxas de reconhecimento médios que podem ser observados na tabela seguinte para ambas as partes desta experiência (Avilés-Arriaga, 2011).

Tabela 1: Resultados relativamente à segunda experiência

Parte 1				***Parte 2***			
Postura e Movimentação		Movimentação		Postura e Movimentação		Movimentação	
DNBC	HMM	DNBC	HMM	.DNBC	HMM	DNBC	HMM
73,85%	74,80%	52,80%	51,60%	.85,79%	86,45%	67,73%	64,18%

A terceira e última experiência documentada neste artigo reflete a variação da distância e rotação do gesto, porque, segundo os autores, a comunicação entre o robot e as pessoas pode ser feita a distâncias diferentes pois, ambos conseguem-se mover. Os resultados mostram que os classificadores possuem resultados competitivos, tanto ao nível da distância como ao nível da rotação, que são dos problemas mais complexos para ambos na fase de reconhecimento. Com isto, concluem que os DNBCs são uma alternativa às HMMs aquando do reconhecimento de gestos, pois, os resultados apresentados assim o demonstram (Avilés-Arriaga, 2011).

2.5 Trabalho relacionado na determinação de atividade

2.5.1 Sistemas de determinação de atividade

Existem vários sistemas de monitorização que utilizam hardware personalizado e que apresentam um custo muito elevado como é o caso do caregivertech e do quietcaresystems (Ryder, Longstaff, Reddy, & Estrin, 2009). A complexidade envolvente no sistema que interage com o paciente pode também condicionar a sua utilização, no entanto, hoje em dia os *smartphones* podem-se tornar num pequeno sistema de monitorização. O *smartphone* é considerado um caso de um sistema ubíquo, o que faz com que exista uma alternativa menos dispendiosa, simples e de fácil acesso (Ryder et al., 2009).

Encontram-se, na literatura, algumas publicações que ilustram o que existe neste momento nesta área, como o PerFallD, que é um protótipo criado para deteção de quedas de pessoas (Dai, Bai, Yang, Shen, & Xuan, 2010), e o Ambulation que é um sistema de monitorização que deteta a posição e a movimentação das pessoas (Ryder et al., 2009). Para além destes, outros artigos apresentam soluções ou apenas possíveis formas de construir

um sistema para atuar nesta área utilizando outras tecnologias disponíveis nos *smartphones* ou mesmo recorrendo a alguns elementos externos ao dispositivo.

O acelerómetro tem sido um sensor muito utilizado por investigadores e estudantes para determinação da atividade de pessoas, assim como para deteção de quedas. (Noury et al., 2007). A evolução da tecnologia ao longo dos últimos anos fez com que os dispositivos móveis começassem a incorporar alguns sensores, como é o caso do acelerómetro, giroscópio e sensor magnético. Esta evolução concebeu a possibilidade de criação de aplicações para extrair dados destes sensores e, a partir deles, adquirir algum conhecimento comportamental. (Kwapisz et al., 2010).

No artigo (Dai et al., 2010) os autores descrevem o desenvolvimento de um protótipo para deteção de quedas, intitulado PerFallD. O PerFallD foi desenvolvido para a plataforma Android, com o objetivo de ser utilizado em pessoas idosas. Segundo os autores, existem soluções comerciais nesta área mas devido ao seu preço elevado não fica acessíveis a todos. Esta solução desenvolvida propõe utilizar um telefone móvel e apenas o sensor de aceleração para detetar as quedas das pessoas, podendo, no entanto, serem utilizados outros acessórios. Projetaram dois algoritmos para a construção da solução no qual o primeiro era baseado na aceleração e o segundo na forma de movimentação e na distância de Hausdorff, que representa o intervalo entre dois pontos de conjuntos de dados diferentes (Dai et al., 2010).

Este *software* arranca inicializando um algoritmo de deteção básico, carregando também a lista de contatos de emergência e os cenários possíveis de acontecer. O programa corre em background e, quando uma queda é detetada, ele envia um sinal sonoro e visual de aviso, no entanto, se a pessoa não o desligar durante um certo período, ele avisa algum contato de emergência. O consumo de energia foi tido em conta no desenvolver do protótipo para uma utilização eficiente da mesma. Foram desenvolvidos dois algoritmos, um que dispensa e outro que utiliza os acessórios. Os acessórios utilizados são um magnetómetro e um acessório magnético que vem aumentar o desempenho da deteção. Na fase de avaliação verificou-se que existem falhas relativamente à deteção de quedas em determinadas posições, no entanto apesar de serem pequenas podem ser corrigidas em certas situações com a utilização do acessório. Os resultados obtidos em testes com pessoas reais demonstram uma precisão de 88,63% quando o sistema não utiliza o acessório e de 90,17% quando utiliza (Dai et al., 2010).

Um outro artigo (Ryder et al., 2009) apresenta o Ambulation, que é um sistema de monitorização de mobilidade que utiliza a plataforma Android. Os autores utilizam um Nokia N95 incorporando um acelerómetro

para detetar os movimentos, assim como um sensor de localização para determinar outro tipo de atividades em ambientes de exterior (*outdoor*). Para deteção e reconhecimento dos movimentos ou atividades usam um *Decision Tree Classifier* que é alimentado através dos valores obtidos dos sensores. O que este classificador faz é prever o que o utilizador está a fazer através das decisões tomadas de acordo com os dados obtidos (Ryder et al., 2009). Obviamente este tipo de modelo necessita de uma fase de aprendizagem para saber qual o caminho a tomar para determinados valores dos sensores.

Os dados recolhidos pelo sistema são enviados para um servidor seguro no qual são interpretados por cuidadores, para saberem como se está a comportar a pessoa que está monitorizada. Existem outros sistemas de monitorização mas requerem custos elevados para compra de *hardware* personalizado logo, fica mais em conta a utilização de um *smartphone* que possui tudo o que é necessário (Ryder et al., 2009).

Segundo os autores, com a utilização de dispositivos móveis deve-se ter em conta também a utilização da energia pois, são recursos limitados e, por isso, são utilizados algoritmos inteligentes que fazem uma gestão eficiente dos sensores. É utilizado um servidor com uma página web para uma melhor visualização da atividade da pessoa a ser monitorizada, conseguindo observar progressos e tendências no que diz respeito à mobilidade. As informações recolhidas pelo sistema em ambientes exteriores são feitas através do sinal de GPS e do acelerómetro, no entanto, quando se trata de ambientes *indoor* apenas se utiliza o acelerómetro para verificar o estado atual da pessoa. O algoritmo executado utiliza as informações recolhidas pelos sensores para desvendar o que o utilizador está a fazer, nomeadamente se está a correr, a caminhar ou a andar de bicicleta (Ryder et al., 2009).

Para um consumo inteligente da bateria no caso do GPS, ele é desligado quando a pessoa se encontra em ambientes indoor. No caso da transferência de informação para o servidor, apenas é executado quando o aparelho se encontra a carregar. Com a utilização de um servidor com uma página para visualização dos dados do paciente é possível ter uma visão geral da atividade ou mesmo ter uma visão pormenorizada de um dia (Ryder et al., 2009).

O Wedjat é uma aplicação desenvolvida para *smartphones* por (Wang, Tsai, Liu, & Zao, 2009), com o objetivo de ajudar os pacientes a evitar erros de medicação. Ele tem duas características distintas que são avisar o paciente sobre interações de medicamentos com alimentos e rever automaticamente o cronograma quando doses são perdidas. As 3 funções primárias do Wedjat são alertar sobre os medicamentos a tomar, fornecer identificação medicinal e a forma de admissão e manter registos do consumo de medicamentos (Wang et al., 2009).

O Wedjat obtém o cronograma de medicação ligando-se via wireless ao servidor e fazendo *download*, fazendo ainda *upload* dos registos guardados no dispositivo sobre o paciente em questão. Se as prescrições dos medicamentos forem alteradas o Wedjat também o consegue descarregar ligando-se ao mesmo servidor. Quando um alerta é lançado pelo *software* o utilizador necessita de confirmar a toma do medicamento, para ficar registado na base de dados. Caso o utilizador demore ou esqueça algumas tomas, o Webjat irá consultar o cronograma e tentar reajustar a toma perdida, se, por ventura, a toma for urgente, ele pode recomendar que o paciente ligue ao médico prescritor. (Wang et al., 2009).

O (Anderson et al., 2007) anuncia no seu artigo a criação de um protótipo apelidado de Shakra, que explora o potencial uso de uma tecnologia do telefone como uma ferramenta de promoção da saúde. Os autores descrevem um protótipo de aplicação que monitoriza as atividades diárias de pessoas, utilizando uma Rede Neural Artificial para analisar a força do sinal GSM e visibilidade para estimar o movimento do utilizador. Num estudo de curto prazo do protótipo com as informações de atividade comum entre os grupos de amigos, descobriram que incentivou a reflexão sobre a consciência, e aumentou a motivação para a atividade física diária (Anderson et al., 2007).

Descreveram então, alguns dos detalhes do estudo-piloto e introduziram uma nova abordagem promissora para deteção de atividade que foi desenvolvida em resposta a algumas das questões levantadas por este, envolvendo os modelos ocultos de Markov (HMM), modelação de tarefas e de calibração não-supervisionadas (Anderson et al., 2007).

O (Torres, Sobral, Moreira, & Morla, 2012)., descreveu um sistema para classificação dos movimentos humanos usando 3 algoritmos de aprendizagem, logitBoost, Multilayer Perceptron e Simple Logistic. O Multilayer Perceptron é um classificador de *machine learning* que implementa uma *feedforward artificial neural network* e usa o algoritmo de backpropagation para aprender os pesos da rede e assim classificar novas instâncias. O Simple Logistic é um classificador adaptado para construir modelos de regressão logística linear. Estes três algoritmos de aprendizagem foram testados na tentativa de detetar 5 atividades, deitar, parado, andar, correr e cair, usando um conjunto de treino com 413 instâncias.

Os autores reconheceram a opção de desenvolvimento para os *smartphones* depois de estes conterem alguns sensores como acelerómetro e giroscópio. A contraposição que eles evidenciam quanto aos *smartphones* é quando o utilizador tira partido dele para efetuar chamadas ou qualquer outra tarefa providenciada pelo mesmo

aparelho, os dados ficam forjados. Com esta conclusão decidiram criar um pequeno aparelho que pudesse ser incorporado no cinto, no relógio ou na roupa do utilizador. Depois de efetuar os testes verificaram que o LogitBoost foi o que obteve melhores resultados, alcançando uma precisão de 98,8% e, por isso, adotaram esse mesmo algoritmo para o seu dispositivo (Torres et al., 2012).

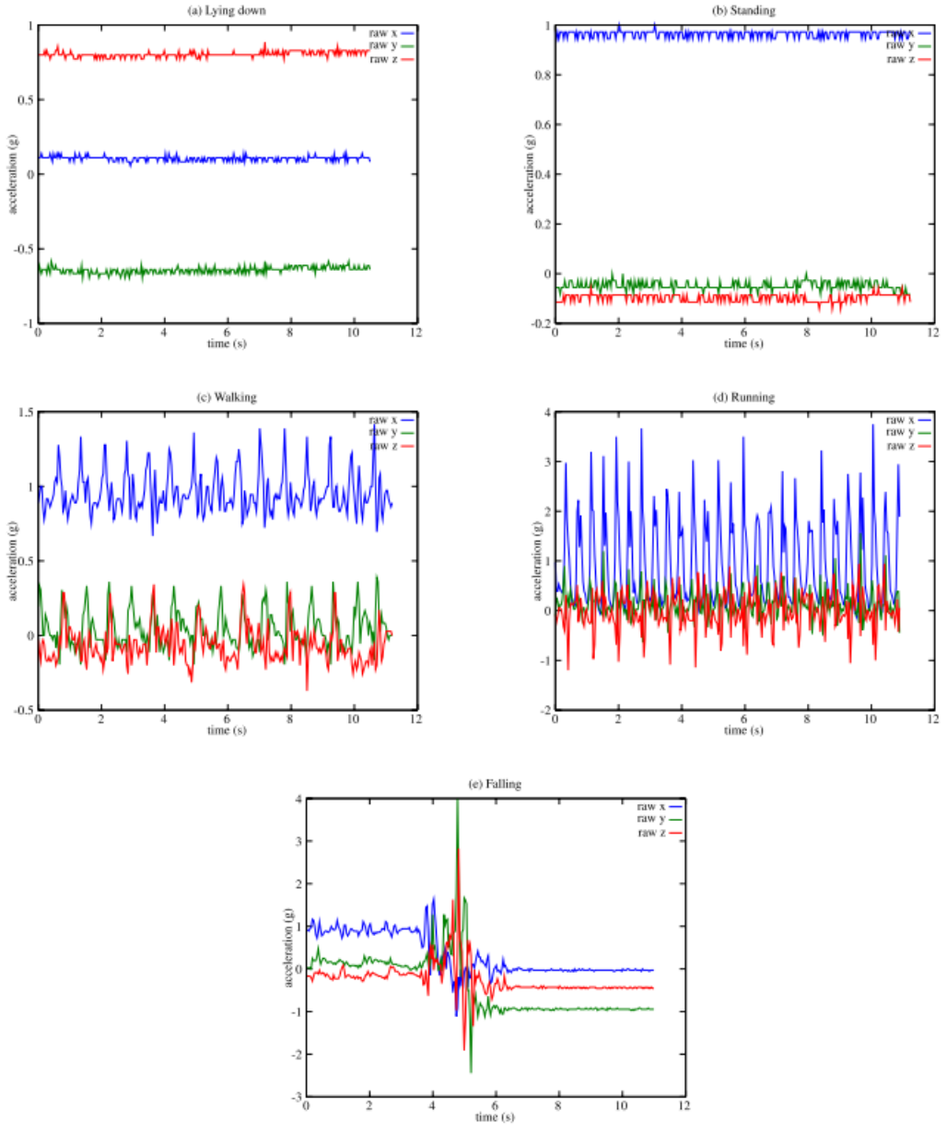

Figura 11: Output típico dos acelerómetros para os 5 movimentos detetados (Torres et al., 2012).

No artigo (Kwapisz et al., 2010), é feita a descrição e avaliação de um sistema baseado em acelerómetros para realizar reconhecimentos de atividades, uma tarefa que envolve a identificação da atividade física de um utilizador. Usaram três algoritmos de aprendizagem máquina nomeadamente J48, Logistic Regression e Multilayer Perceptron para determinar a atividade de cada utilizador e para verificar qual o que desenvolvia melhores resultados de precisão. O algoritmo J48 é uma derivação das árvores de decisão mas os seus atributos não são numéricos mas sim categóricos, com os valores "*Yes*" e "*No*". Para implementar este sistema, foram recolhidos dados de acelerômetros de vinte e nove utilizadores enquanto realizavam atividades diárias, como caminhar, correr, subir escadas, sentar, e em pé. Em seguida, foram usados os dados de treinamento resultantes para induzir um modelo preditivo para o reconhecimento de atividade. Este trabalho é significativo porque o modelo de reconhecimento de atividade permite adquirir conhecimentos sobre os hábitos de milhões de utilizadores passivamente (Kwapisz et al., 2010).

O que foi desenvolvido por estes autores tem uma vasta gama de aplicações, incluindo a personalização do comportamento do dispositivo móvel baseado numa atividade do utilizador (por exemplo, enviar chamadas diretamente para o voice-mail e gerar um perfil de atividade diário / semanal para determinar se um utilizador (talvez uma criança obesa) realiza uma saudável quantidade de exercício. O artigo (Kwapisz et al., 2010) apresenta ainda os resultados obtidos com os testes realizados e verificam que apesar de nenhum algoritmo ser o melhor consistentemente, o Multilayer Perceptron é o que apresenta melhores resultados com uma média de 91.7% de precisão.

O projeto exposto no artigo (Luštrek & Kaluža, 2009), visa monitorizar pessoas para detetar a sua atividade, assim como reconhecer quedas e alguns problemas de saúde. Isto é conseguido dotando o utilizador com *radio tags*, a partir das quais as localizações de partes do corpo são determinadas, permitindo assim a reconstrução da postura e do movimento, no entanto, usam ainda o acelerómetro e o giroscópio para a determinação da atividade física e para o reconhecimento de quedas. Depois de terem pesquisado, obtiveram uma visão geral sobre deteção de queda e reconhecimento atividade e consideraram a utilização de um algoritmo de aprendizagem máquina para reconhecimento da atividade. Nesta abordagem, os atributos que caracterizam o comportamento do utilizador eram um algoritmo de aprendizagem máquina que iria ser selecionado. Os atributos que consideraram foram várias partes do corpo em relação ao sistema de coordenadas de referência (fixo em relação ao meio ambiente), a localização de partes do corpo em um sistema de coordenadas do corpo (afixada para o corpo do utilizador) e os ângulos entre as partes do corpo adjacentes. Oito algoritmos de aprendizagem máquina pertencentes à package Weka foram comparados, na tentativa de obtenção dos melhores resultados. Dos

classificadores utilizados o que se destacou com a sua precisão foi o *Support Vector Machine*, com uma precisão máxima de 97,7% com *clean data* e 96,5% com *noisy data*. O algoritmo *Support Vector Machine* é um modelo de aprendizagem supervisionado associado com algoritmos de aprendizagem que analisam os dados e fazem o reconhecimento de padrões usados para a classificação, assim como a análise de regressão. Eles foram distinguidos pelo equipamento utilizado e pelas características extraídas a partir de dados do sensor (Luštrek & Kaluža, 2009).

O trabalho apresentado no artigo (Bao & Intille, 2004) reflete a avaliação e desenvolvimento de algoritmos para reconhecimento de atividades de pessoas. Para isto, foram utilizadas vinte pessoas com 5 acelerómetros biaxiais espalhados por partes do corpo, recolhendo informação sobre as suas atividades diárias, sem qualquer supervisão. Isto foi elaborado em laboratório, em condições seminaturais. Para este tipo de aplicações é importante que exista treino e testes em circunstâncias naturais, porque a utilização de ambientes laboratoriais influencia, contrai artificialmente e simplifica os dados relativos às atividades, segundo os autores (Bao & Intille, 2004).

Neste trabalho foi aplicado o protocolo semi-naturalista para fazer a recolha da informação de treino, isto seguindo um guia das 20 atividades a elaborar sem qualquer supervisão que são: *Walking Training, Watching TV, Running, Stretching, Scrubbing, Folding laundry, Brushing teeth, Riding elevator, Eating or drinking, Reading, Bicycling, Strength-training, Riding escalator Accuracy, Sitting & relaxing, Working on computer, Standing still, Walking carrying items, Vacuuming, Climbing stairs, Lying down & relaxing* (Bao & Intille, 2004).

Foram utilizados algoritmos como *Decision Table*, *Inquiry-based Learning* (IBL), C4.5 (*Decision Tree*) e *Naive Bayes* presentes na coleção Weka, sendo todos algoritmos de aprendizagem. O algoritmo *Decision Table* é uma forma de modelar um problema através de uma tabela que evidencia as condições e as ações, enquanto que o algoritmo *Inquiry-Based Learning* é um método de aprendizagem baseado em investigação de problemas e o Naive Bayes é um classificador probabilístico simples que aplica o teorema de Bayes. Depois de avaliados os algoritmos com base nos protocolos de estudo, e de acordo com as suas precisões, conseguiram chegar à conclusão que as árvores de decisão é o algoritmo que obtinha melhor desempenho, atingindo no conjunto de todas as atividades uma precisão de 84,26%. Para obterem uma melhor perceção do estilo da execução das atividades, nomeadamente correr rápido ou lento precisaram de um complemento que poderia passar pela utilização de um sensor de localização, GPS (Bao & Intille, 2004).

No artigo (Gonçalves et al., 2009), os autores descrevem a construção de um sistema de cuidados de saúde em casa através de um aparelho para o corpo, de um ambiente monitorizado, de um gateway wireless e um servidor. O aparelho do corpo deteta os sinais vitais, tal como a temperatura e os batimentos cardíacos (Gonçalves et al., 2009).

A monitorização do ambiente foi utilizada para medir a temperatura e analisar a luz ambiente. Toda a informação do dispositivo era enviada via Zigbee para a gateway, que por sua vez transmitia ao servidor a mesma, sendo de seguida guardada numa base de dados. Foram utilizados elementos como acelerómetro, sensor de luminosidade, sensor de temperatura e módulos Zigbee, para construção de todo o sistema. São detetados movimentos como parar, sentar, caminhar, correr, deitar e quedas. Foram utilizados alguns voluntários para fazer estas atividades para recolha de informação e o valor de cada eixo foi retirado a cada 100ms (Gonçalves et al., 2009).

Foi utilizada a variância estatística para determinar um valor específico para cada eixo de maneira a obterem 3 resultados para cada grupo de valores obtidos pelo acelerómetro (grupos de 10 valores), ao fazerem a média destes 3 valores atingem o designado *Value for Activity Indicator* (VAI) que serve para criar o sistema baseado em heurísticas. Para isto foi criada uma tabela de cenários, incluindo várias atividades transitórias, com o objetivo de detetar a atividade corretamente. Isto anteriormente referido acontecia quando os cenários já estavam plenamente construídos, através de atividades sequenciais esperadas pelos autores. Os resultados obtidos depois dos testes feitos foi que a corrida era detetada com uma precisão de 70%, enquanto as outras atividades rondavam os 95%, no entanto os autores afirmam que quando testado em pessoas idosas o mesmo sistema melhorava para perto dos 100% (Gonçalves et al., 2009).

Os autores do artigo (Mccall, Reddy, & Shah, 2012), utilizam o *smartphone* para recolha de dados do acelerómetro, do giroscópio e do magnetómetro, com o intuito de criar um classificador hierárquico K-Nearest Neighbor (K-NN) para seleção de macro classes. Este algoritmo classifica um objeto de acordo com os exemplos que possui no seu espaço de atributos. O Objetivo é fazer o reconhecimento de atividades de dois conjuntos distintos, um envolve atividades relacionadas com cozinhar e outras atividades relacionadas com aeróbica. Para recolher estes dados relativamente ao primeiro conjunto foram utilizados sensores nas pernas, braços e costas enquanto no segundo, cada utilizador usava apenas um iPhone 4 alocado na zona da cintura. O objetivo dos autores era provar que utilizando um modelo hierárquico obteriam melhores resultados em relação aos que não o eram, devido à utilização de vários níveis de classificação. Para esse efeito não eram só usados os valores obtidos

dos sensores, mas também recursos estatísticos, como média, variância, entre outros. O resultado que alcançaram depois de efetuarem os testes com o algoritmo que desenvolveram a partir do tradicional K-NN, foi positivo pois, conseguiram superar os algoritmos sem hierarquia em qualquer um dos conjuntos de dados e atingir uma precisão que chegava aos 88%.

No artigo (Oliver, Garg, & Horvitz, 2004), é apresentado um sistema apelidado de SEER que faz o reconhecimento de atividades de pessoas dentro de um espaço de trabalho. Os autores falam sobre a utilização do algoritmo *Hidden Markov Models* (HMMs) por camadas como meio de diagnóstico para saber a atividade que está a ser executada em tempo real através de fontes de informação como vídeo, áudio e interações com o computador (através do rato e teclado). Neste sistema utilizam LHMMs como uma representação para aprender através de diferentes camadas de classificadores e usando-os para classificar conceitos temporais com diferentes granularidades de tempo. O tratamento que é feito quanto ao modelo utilizado é que cada camada inferior está de certa forma ligada à camada superior apesar de operarem independentemente. Afirmam que a camada superior obtém como *input* de observações o resultado da camada inferior, assim como para a aprendizagem, a camada superior só consegue fazer o treino dos parâmetros quando a camada inferior terminar o mesmo. O *hardware* utilizado neste projeto foram dois microfones, uma câmara, um teclado e um rato de computador.

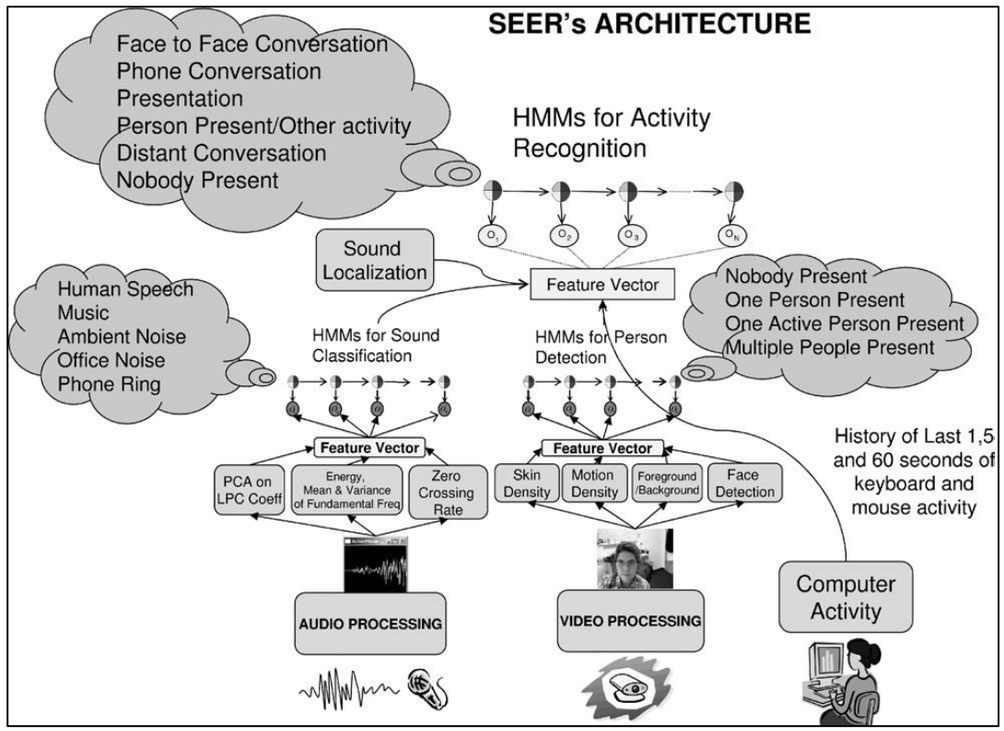

Figura 12: Arquitetura do SEER (Oliver et al., 2004)

Na arquitetura desenvolvida pelos autores do projeto verifica-se que de cada fonte de informação depois do tratamento da *raw data* se obtém um resultado, esta operação faz parte do 1º nível do HMM aplicado. O 2º nível deste método é a combinação destes resultados para a obtenção de um resultado final que retrata o ambiente de trabalho ou a atividade que está a ser desenvolvida. Os resultados obtidos para estes dois algoritmos nos testes elaborados foram de 72,68% e 99,7% para os HMMs e LHMMs, respetivamente.

2.5.2 Análise Comparativa

Como já foi referido este projeto foi desenvolvido recorrendo a vários tipos de fontes de informação na tentativa de obter uma melhor precisão no resultado obtido. Alguns dos projetos apresentados na secção anterior não apresentam as mesmas ferramentas, nem o mesmo método para a obtenção do resultado.

Tabela 2: Comparação entre algumas das soluções apresentadas

	Acel.	Gyrosc.	Magnet.	Outro sensor	Indoor / outdoor	Objetivo	Dispositivo
(Dai et al., 2010)	Sim	Não	Não	Magnético	ambos	Detetar quedas	*smartphone*
(Ryder et al., 2009)	Sim	Não	Não	GPS	ambos	Monitorizar atividade e localização	*smartphone*
(Anderson et al., 2007)	Não	Não	Não	GSM	ambos	Monitorizar atividade física	*Cell phone*
(Torres et al., 2012)	Sim	Não	Não	-	ambos	Monitorizar atividade física	*customized*
(Kwapisz et al., 2010)	Sim	Não	Não	-	ambos	Monitorizar atividade física	*Cell phone*
(Luštrek & Kaluža, 2009)	Sim	Sim	Não	Radio tag	ambos	Monitorização da atividade física e reconhecimento de quedas	*customized*
(Gonçalves et al., 2009)	Sim	Não	Não	Temperatura e luminosidade	*indoor*	Monitorização da atividade física e sinais vitais	*customized*
(Mccall et al., 2012)	Sim	Sim	Sim	-	*ambos*	Detetar atividades físicas variadas (cozinhar)	*smartphone*
Own Application	Sim	Sim	Sim	-	*ambos*	Monitorização da atividade física	*smartphone*

A tabela 1 resume de alguma forma o que foi falado nos trabalhos relacionados com o PADS, mostrando o que alguns dos protótipos já desenvolvidos utilizam como tecnologias, o seu objetivo e o tipo de dispositivo usado. Com este agrupamento de dados conseguimos observar que a utilização do *smartphone* para criação de aplicações, de soluções para ajudar as pessoas tem vindo a ganhar muitos adeptos. Também conseguimos verificar que nos últimos anos as atividades que se dignificam a identificar com um *smatphone* já não são apenas atividades de locomoção.

Nem todos os projetos aqui evidenciados utilizam um *smartphone* com sensores como o acelerómetro para detetar atividades físicas, como é o caso de (Torres et al., 2012), (Luštrek & Kaluža, 2009) e (Gonçalves et al., 2009). Estes dispositivos concebidos para monitorização física fazem com que o utilizador tenha de carregar mais um dispositivo consigo, o qual normalmente, não é considerado um sistema ubíquo. A utilização de um *smartphone* pode colmatar de certa forma a sensação de invasão de privacidades visto que muitas pessoas já estão habituadas a transportar o mesmo consigo onde quer que vão. Alguns projetos ainda vão mais longe e utilizam outro tipo de sensores espalhados pelo corpo da pessoa o que se torna de certa forma um incómodo para quem utiliza, como é o caso do (Dai et al., 2010) e do (Luštrek & Kaluža, 2009). Algumas das soluções apresentadas por estes autores conseguiram atingir uma precisão muito elevada como foi o caso do (Torres et al., 2012),que serviu de motivação na utilização do algoritmo LogitBoost. A grande maioria dos projetos apresentados baseia-se em algoritmos de aprendizagem para fazer o reconhecimento de atividades físicas diárias, ou seja envolvem uma fase de recolha de dados das atividades para posteriormente testarem os mesmos e confirmarem se o conjunto de treino é o mais correto ou não.

O artigo (Oliver et al., 2004) revela o uso da fusão de diferentes fontes de informação para a obtenção de um resultado com maior precisão utilizando uma extensão dos modelos ocultos de Markov. Tanto os HMMs assim como as redes bayseanas identificam-se como modelos gráficos podendo ser eles dinâmicos ou temporais, sendo ainda a sua representação muito idêntica. Apesar do contexto de aplicação do PADS e do projeto SEER desenvolvido pelos autores deste artigo ser um bocado diferente o que faz diferir o tipo de atividades detetadas, o sentido de raciocínio é o mesmo.

Capítulo 3: Sistema de deteção de atividade

Neste capítulo são apresentados os requisitos do sistema desenvolvido no âmbito deste projeto, a sua arquitetura e detalhes de implementação. Este trabalho visou o desenvolvimento de um sistema pessoal de deteção de atividade que engloba três elementos distintos: uma aplicação para os *smartphones* com sistema operativo Android; uma aplicação de visão por computador para deteção e reconhecimento de faces aquando da presença de pessoas na sua frente e um *software* para fusão de informação de diferentes tipos. Globalmente, o projeto foi batizado de Personal Activity Detection System (PADS) devido às suas características de deteção de atividades físicas ou funcionais através de dispositivos móveis ou câmaras, respetivamente.

3.1 Componentes do PADS

Existem vários componentes que se envolvem no funcionamento de todo este sistema, possuindo diferentes características e diferentes funções de aquisição de dados. Enquanto alguns componentes encontram-se posicionados em locais estratégicos, de um modo fixo, como as câmaras, outros são transportados pelo individuo que está a ser monitorizado, como é o caso do dispositivo móvel.

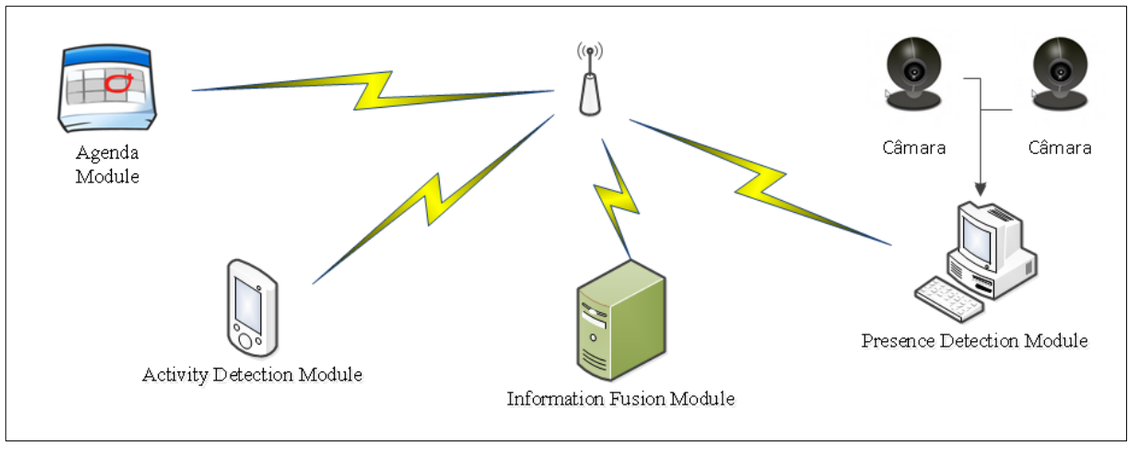

Figura 13: Arquitetura do PADS

A Figura 13 apresenta a arquitetura desenhada para o PADS, sistema de deteção de atividade. O mesmo é constituído por vários elementos que comunicam com o servidor para fornecer informação sobre o paciente que se encontra no espaço inteligente:

- O *Activity Detection Module*, que recorre a um dispositivo móvel, é responsável pela classificação da atividade que o utilizador está a fazer, podendo utilizar o servidor para essa mesma tarefa ou não.
- O *Presence Detection Module* é um programa responsável por identificar a presença de pessoas em locais predefinidos, através da face, com recurso a câmaras de vídeo e técnicas de visão por computador.
- O *Agenda Module*, consulta o Google Calendar onde está agendado quando o paciente tem de tomar os medicamentos que estão presentes no *drugdispenser*.
- O *Information Fusion Module*, é um módulo de *software* implementado num servidor onde são guardadas as informações relativamente ao que se conhece do utente e é também neste elemento que a fusão de informação acontece realmente.

3.2 Requisitos

A definição dos requisitos gerais do sistema obedeceu a uma divisão em duas partes, os requisitos funcionais e os não funcionais. Os requisitos funcionais são definidos como declarações dos serviços que o sistema deve fornecer, a reação do mesmo a certos *inputs* e o seu comportamento em situações particulares. Os requisitos não funcionais referem restrições sobre os serviços ou funções oferecidos pelo sistema, incluindo restrições de tempo, restrições sobre o processo de desenvolvimento e padrões (Ian Sommerville, 2007).

Tabela 3: Requisitos funcionais das aplicações móveis

ID	**Descrição**
RFA01	Obter dados de sensores como acelerómetro, giroscópio e magnetómetro.
RFA02	Guardar os dados dos sensores num ficheiro log.
RFA03	Alterar a frequência de amostragem.
RFA04	Identificação da atividade que está a ser executada.
RFA05	Definir o elemento de classificação da atividade.
RFA06	Classificação e visualização de atividades.
RFA07	Paragem automática de recolha de dados dependendo da atividade.

RFA08	Gravar temporariamente os dados no cartão de forma automática.
RFA09	Envio automático de informações para o servidor em tempo real.

Tabela 4: Requisitos não funcionais das aplicações móveis

ID	**Descrição**
RNFA01	Executar num *smartphone* Android de gama média.

Tabela 5: Requisitos Funcionais das Câmaras

ID	**Descrição**
RFC01	Deteção de caras, olhos e corpos.
RFC02	Reconhecimento de caras.
RFC03	Subtração do plano de fundo à imagem.
RFC04	Envio automático de informações para o servidor em tempo real.
RFC05	Tratamento da informação de forma automática.

Tabela 6: Requisitos não funcionais das Câmaras

ID	**Descrição**
RNFC01	A aplicação deve adaptar-se a qualquer câmara instalada na máquina.

Tabela 7: Requisitos Funcionais da fusão de informação

ID	**Descrição**
RFF01	Informação deve ser guardada numa base de dados.
RFF02	Determinação da atividade atual do utente.
RFF03	Obtenção automática dos dados.
RFF04	Carregamento e treino da rede bayesiana.

Tabela 8: Requisitos não funcionais da fusão de informação

ID	**Descrição**
RNFF01	Deve ser capaz de suportar a taxa temporal de inserção e pesquisa de informação.

As tabelas apresentadas acima apresentam os requisitos do sistema, organizados por cada um dos seus elementos. Os requisitos RFA01, RFA02, RFA03, RFA07 e RFA08 são relativos à aplicação móvel de recolha dos dados para construção do modelo de classificação através da WEKA. Por conseguinte, os requisitos RFA04, RFA05, RFA06 e RFA09 são referentes à aplicação de classificação de atividades. O requisito RFA09 que se refere ao envio de dados para o servidor, assim como o RFC04, está diretamente conectado com o RFF01 pois, é na base de dados que vão ser guardados todos os dados relevantes e a serem utilizados futuramente. Isto é necessário para a concretização do requisito RFC05 e RNFF01, em que tudo é esperado num local exato, para que o RFF02 seja realizado com sucesso. Os requisitos RFC01, RFC02, RFC03, que se englobam na componente das câmaras, têm como objetivo recolher a posição do utente no espaço inteligente e utilizando o RFC04 enviar esse dado para o servidor. Para que isso seja possível, é necessário colocar uma câmara por cada local da sala que pretendemos monitorizar para que cada uma forneça informação desse mesmo local.

3.3 Modelo estatístico de determinação de atividade

O componente agregador do sistema é substanciado pelo módulo de fusão de informação. Este módulo processa, em cada fatia temporal, a informação recebida dos restantes módulos. Nessa perspetiva, é possível apresentar um modelo, de natureza estatística, que serve de base ao motor de fusão de informação.

Uma das funcionalidades do *Activity Detection Module*, a executar no dispositivo móvel, é útil numa fase de treino para recolha de dados empíricos através dos sensores e para a criação do modelo de classificação, com o objetivo da sua utilização numa fase de funcionamento pleno, isto é, na operação de deteção propriamente dita.

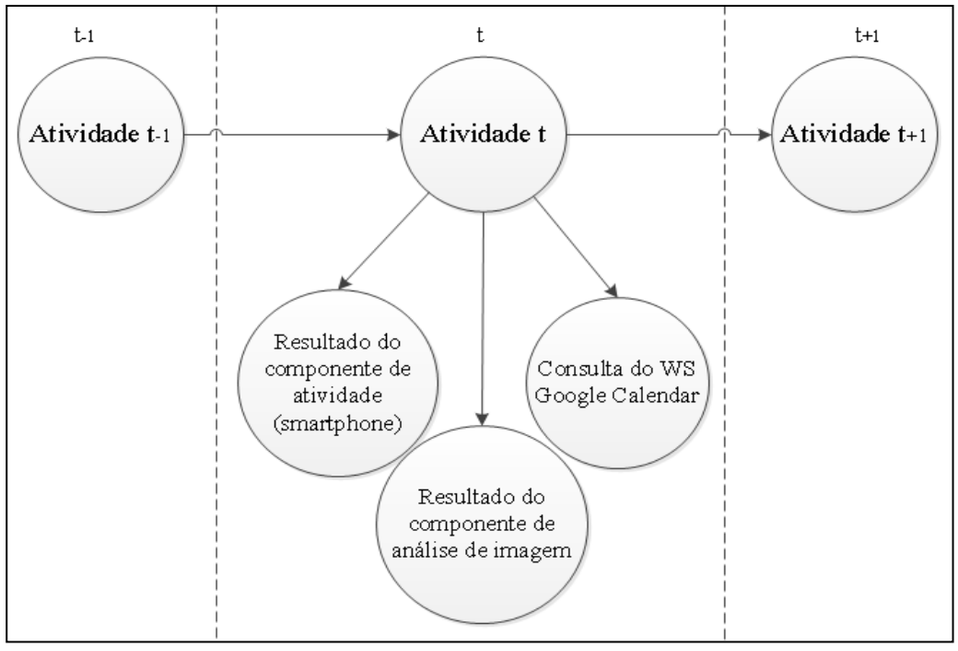

Figura 14: Representação da rede bayesiana adaptada ao PADS

A Figura 14 mostra através da representação de uma rede bayesiana, uma visão da relação de dependência estatística existente entre os resultados obtidos nos vários módulos do sistema. A atividade$_t$ representa a atividade presente, a que está a ocorrer no espaço temporal atual, enquanto a atividade$_{t-1}$ e a atividade$_{t+1}$ representam o passado e o futuro, respetivamente. O que está exposto é que a atividade$_t$ é determinada através do conhecimento da atividade$_{t-1}$ e através das fontes de informação que neste caso são, o resultado do componente de atividade, o resultado do componente de análise de imagem e a consulta ao *webservice* do Google Calendar. Esta atividade$_t$ vai ser útil para a determinação da atividade$_{t+1}$, da mesma forma que a atividade$_{t-1}$ foi para a atividade$_t$.

3.4 Implementação dos Módulos

Este sistema está implementado recorrendo a várias linguagens de programação e plataformas. Uma das linguagens de programação utilizadas para concretização do sistema foi o Java, que é uma linguagem orientada a objetos e um pouco diferente das linguagens compiladas mais convencionais pois o código é compilado para formato *bytecode* e executado sobre uma máquina virtual. O Java foi utilizado para desenvolvimento das aplicações móveis que correm apenas no sistema operativo Android e de parte do servidor que responde aos pedidos de classificação de atividades.

Outra das linguagens utilizadas foi o C++, que é uma linguagem de programação multi-paradigma e possui muitas melhorias em relação ao C, como por exemplo, o suporte para classes. Esta linguagem foi utilizada para desenvolvimento do programa para controlar as câmaras, e para construção do *software* que faz a combinação de informação.

Foi também empregue a linguagem de programação PHP, que é uma linguagem interpretada, normalmente utilizada para criação de componentes web. O PHP foi usado para a criação do *webservice* presente no servidor, assim como para a criação de um *script* que faz uma consulta direta à base de dados e gera um ficheiro de texto. Este processo de criação tem como finalidade preparar os dados, para que a biblioteca de fusão de informação, o Mocapy++, os consiga interpretar corretamente. Por último, foi utilizado o SQL, que é uma linguagem de pesquisa declarativa para bases de dados relacionais. Esta linguagem foi utilizada para fazer operações na base de dados, a qual está diretamente ligada ao *webservice*.

3.4.1 Activity Detection Module

As aplicações de deteção de atividade foram desenvolvidas para serem executadas numa plataforma móvel com o sistema operativo Android utilizando os sensores disponíveis no equipamento. Na Figura 15, é possível observar a arquitetura da parte do sistema que envolve o dispositivo móvel, na qual se observa também o servidor devido à possibilidade de ser utilizado quer para armazenar dados dos sensores quer mesmo para os classificar como sendo uma das atividades consideradas.

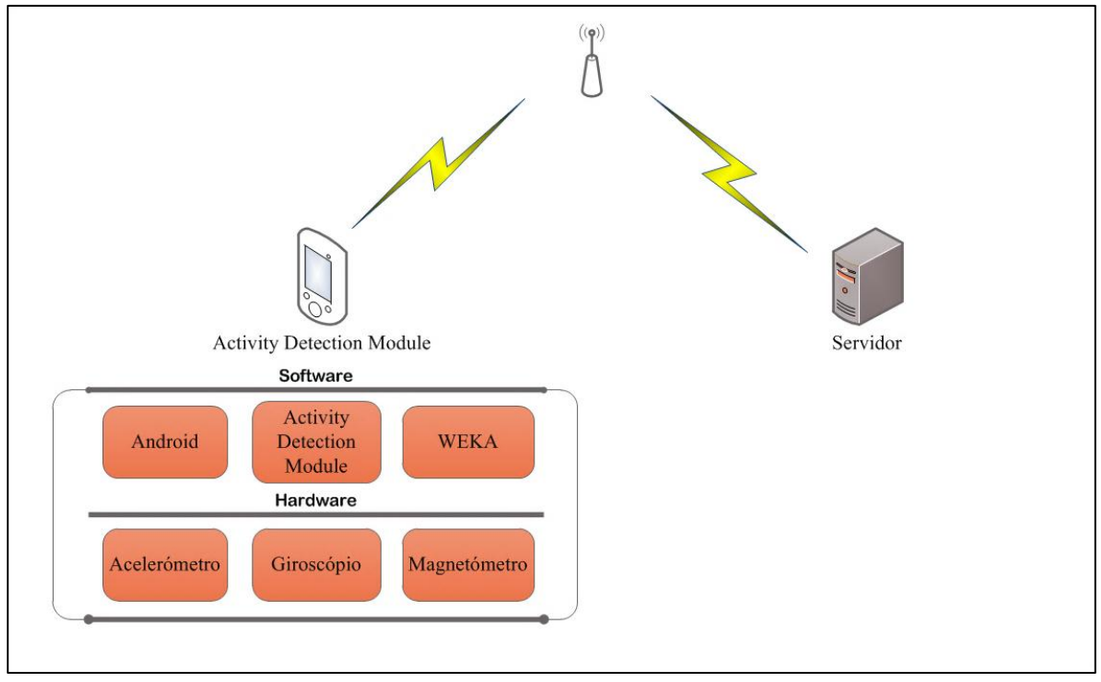

Figura 15: Arquitetura na ótica do Activity Detection Module.

Na implementação do projeto propriamente dito, foi iniciado em primeiro lugar o desenvolvimento de duas aplicações independentes para os dispositivos móveis:

- Aplicação 1: Aplicação de recolha e tratamento da informação reportada pelos sensores disponíveis no dispositivo móvel a fim de ser utilizada para criação do modelo de classificação. Esta é utilizada numa fase de treino.
- Aplicação 2: Aplicação responsável pela leitura dos sensores assim como pela classificação da atividade que está a ser executada em tempo real através da biblioteca de aprendizagem máquina. Esta é utilizada numa fase de funcionamento pleno.

Os dados essenciais para estas aplicações eram os dados relativos aos sensores, como o acelerómetro, giroscópio e magnetómetro.

Criaram-se estas aplicações com o objetivo de produzir uma solução que consiga detetar, como resultado, a atividade que o utilizador do dispositivo móvel está a desempenhar. Foram desenvolvidas duas aplicações pois, uma faz a recolha dos dados para criação de um modelo, enquanto a outra faz a interpretação dos dados e respetiva

classificação através do modelo disponível. A recolha dos dados para criação de um modelo de aprendizagem é necessária para saber quais os valores e/ou sinais que refletem o facto da pessoa estar a executar cada atividade em particular. Para esse fim podíamos usar apenas o acelerómetro, pois como já foi referenciado, existem algumas soluções que assim o fazem. No entanto, o uso adicional do giroscópio e do magnetómetro pode-nos ajudar a diferenciar algum tipo de atividades que têm algum grau de proximidade em termos de inclinação e direção de movimento. A grande dificuldade deste tipo de aplicações é diferenciar estas situações, quando as atividades têm movimentos de certa forma semelhantes que podem ser confundidos, ou seja, classificados de forma errada pelo sistema.

Por estas razões se criou um modelo de aprendizagem, em que os dados são adquiridos através da execução da atividade várias vezes e por pessoas diferentes. Este modelo foi desenvolvido com recurso a uma biblioteca de *machine learning*. A fase de aprendizagem é feita através da aplicação *Explorer* da Weka e da interface gráfica que essa aplicação disponibiliza. No final desta fase, obtém-se um modelo serializado que pode ser utilizado pela aplicação desenvolvida para efetuar a classificação e obter os resultados desejados. Estes resultados obtidos são relativos à aquisição em tempo real dos dados dos sensores e sucedendo a respetiva classificação, de forma ainda a enviar para o servidor a cada segundo informação sobre o utente que está a ser monitorizado a nível físico.

Para a obtenção da *raw data* dos sensores foi preciso declarar primeiramente um serviço para fazer gestão dos sensores, como é apresentado no Algoritmo 1. Ainda, no mesmo código, são demonstradas como foram inicializadas as variáveis que iriam ser interrogadas para obtenção dos dados sensoriais.

```
mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
mGyro = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
mMagnetic = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

Algoritmo 1: Declaração dos sensores do smartphone

Para fazer o armazenamento dos dados provenientes dos sensores, foram declaradas diversas variáveis, algumas ajustáveis à taxa de amostragem em vigor. Neste caso foi utilizado uma frequência de 10Hz, ou seja, uma taxa de 10 amostras por segundo para cada sensor.

A *framework* do Android providencia um mecanismo de amostragem um tanto ou quanto irregular devido ao mecanismo, de natureza não determinística, que controla o instante em que a invocação de uma função da aplicação acontece, que por sua vez pode introduzir ruído na quantificação (Das et al., 2010). A API do Android disponibiliza quatro taxas de amostragem abstratas para os sensores: *Fastest*, *Game*, *Normal* e UI. A única forma de obter os dados dos sensores é recorrendo ao método "onSensorChanged()", que é despoletado pelo sistema operativo assim que determina que o sensor obteve modificações. Aquando da inicialização dos sensores é invocado o método "onResume()", onde são parametrizadas as taxas de amostragem que os sensores vão considerar, como apresentado no Algoritmo 2.

```
protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer,
        SensorManager.SENSOR_DELAY_FASTEST );
        mSensorManager.registerListener(this, mGyro,
        SensorManager.SENSOR_DELAY_FASTEST );
        mSensorManager.registerListener(this, mMagnetic,
        SensorManager.SENSOR_DELAY_FASTEST );

}
```

Algoritmo 2: Registo dos sensores no método de inicialização

Com isto fizemos com que ele obtenha os dados dos sensores o mais rápido que conseguir podendo ultrapassar os 20hz. Deste modo, conseguimos obter dos sensores um número de amostras adequado com o método apresentado no Algoritmo 3

```
public void onSensorChanged(SensorEvent arg0) {
      synchronized (this) {
            switch (arg0.sensor.getType()){
                  case Sensor.TYPE_ACCELEROMETER:
                        axis_x = arg0.values[0];
                        axis_y = arg0.values[1];
                        axis_z = arg0.values[2];
                        break;
                  case Sensor.TYPE_GYROSCOPE:
                        axis_x_gyro = arg0.values[0];
                        axis_y_gyro = arg0.values[1];
                        axis_z_gyro = arg0.values[2];
                        break;
                  case Sensor.TYPE_MAGNETIC_FIELD:
                        axis_x_magn = arg0.values[0];
                        axis_y_magn = arg0.values[1];
                        axis_z_magn = arg0.values[2];
                        break;
            }
      }
}
```

Algoritmo 3: Captura dos dados dos sensores

Como já foi referido, a janela da taxa de amostragem estaria fixada nos 10Hz por defeito e, na tentativa de obtermos esta taxa de uma forma mais regular do que o mecanismo do sistema operativo nos consegue transmitir, optamos por construir um método para esse propósito. Utilizamos a classe CountDownTimer, na qual definimos um tempo total e a frequência de execução (em milissegundos) do método "onTick()" e quando terminar a método "onFinish()". Com isto conseguimos definir que a cada 100 milissegundos ela executaria o método já referido, presente no Algoritmo 4.

```
CountDownTimer cdt = new CountDownTimer(1000, 100) {

      public void onTick(long millisUntilFinished) {
```

Algoritmo 4: Instância da classe CountDownTimer

Depois de aplicado este método diretamente ligado ao CountDownTimer era só obter o valor das variáveis declaradas globalmente para os sensores e colocá-los num array também já inicializado. Este array está dimensionado para um tamanho máximo de amostras por segundo já definido que ao atingir o seu limite será transformado numa *instance*. Segundo (Torres et al., 2012), uma *instance* ou uma *instance vector* é um conjunto de dados de sensores, em que cada um é representado em um instante t*i*. Uma *instance* passada como argumento

ao algoritmo classificador deve conseguir capturar as variações numa certa janela de tempo e assim discriminar a atividade detetada.

O Algoritmo 5 mostra como no método "onTick" são guardados os valores e é feita a verificação para avaliar se já atingiu o limite da janela para construir a instância.

```
public void onTick(long millisUntilFinished) {

	TextView tv = (TextView) findViewById(R.id.txtview_activity);
	if(start){
		m_tailIdx = (m_tailIdx + 1) % m_WindowSize;
		acelX[m_tailIdx] = axis_x;
		acelY[m_tailIdx] = axis_y;
		acelZ[m_tailIdx] = axis_z;
		x_gyro[m_tailIdx] = axis_x_gyro;
		y_gyro[m_tailIdx] = axis_y_gyro;
		z_gyro[m_tailIdx] = axis_z_gyro;
		x_mag[m_tailIdx] = axis_x_magn;
		y_mag[m_tailIdx] = axis_y_magn;
		z_mag[m_tailIdx] = axis_z_magn;
		m_NumElements++;

		if (m_NumElements==m_WindowSize){

			inst = makeInstance();
			m_headIdx = (m_headIdx + m_WindowSize - m_NumOverlapElements) %
m_WindowSize;
			m_NumElements = m_NumOverlapElements;
			try {
				RadioGroup rg1 = (RadioGroup) findViewById(R.id.radioGroup1);
				rg1.getCheckedRadioButtonId();
				if(rg1.getCheckedRadioButtonId()==R.id.radio0)
					Classification(inst);
				else
					connection();
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
}
```

Algoritmo 5: Algoritmo que faz a gestão das instâncias em relação à raw data.

Ainda no algoritmo acima conseguimos observar que existe uma verificação de um *radio button*, que é utilizado para o utilizador controlar se quer fazer a classificação da atividade no dispositivo móvel ou no servidor. Em qualquer uma das formas o resultado será apresentado no dispositivo móvel, mas os algoritmos de classificação podem ser diferentes.

Para uma melhor classificação são utilizados dois valores adicionais para cada eixo de cada sensor que é a média e a variância, tal como sugerido no artigo (Torres et al., 2012). Estes dois dados são também adicionados na componente *instance*, mas para que tudo esteja correto precisamos ainda de um *header* configurado corretamente com o formato ARFF, como o que é apresentado na Figura 16.

```
% 5. System used: Nexus
%

@relation nexus_instances

@attribute accelX0 real
@attribute accelY0 real
@attribute accelZ0 real
@attribute gyroX0 real
@attribute gyroY0 real
@attribute gyroZ0 real
@attribute magnX0 real
@attribute magnY0 real
@attribute magnZ0 real
```

Figura 16: Formatação de um ficheiro ARFF.

Após termos o header correto, criamos o método "makeInstance()", carregando o arff para o nosso objeto instance utilizando o ArffLoader, como é possível observar no Algoritmo 6. Utilizando alguns métodos como o "getStructure()" para obter a estrutura do arff e o "numAttributes()" para obter o número de atributos conseguimos inicializar uma nova instância.

```
ArffLoader loader = new ArffLoader();
try {
	loader.setFile(new File("mnt/sdcard/nexus_header.arff"));
	m_DataHeader = loader.getStructure();
	m_DataHeader.setClassIndex(m_DataHeader.numAttributes() - 1);
} catch (IOException e) {
	e.printStackTrace();
}
Instance inst = new DenseInstance(m_DataHeader.numAttributes());
inst.setDataset(m_DataHeader);
```

Algoritmo 6: Carregamento do header e inicialização de um objeto do tipo Instance.

O passo seguinte passa por fazer a manipulação dos arrays com os dados dos sensores, e assim inserir os dados dentro do mesmo objeto. Por fim, introduzimos também a média e a variância para cada eixo usando uma função já existente no package da Weka, Utils.mean e Utils.variance, respetivamente.

```
for (int i = 0; i < 9 * m_WindowSize; i = i + 9) {

      inst.setValue(i, acelX[(m_headIdx + i) % m_WindowSize]);
      inst.setValue(i + 1, acelY[(m_headIdx + i) % m_WindowSize]);
      inst.setValue(i + 2, acelZ[(m_headIdx + i) % m_WindowSize]);

      inst.setValue(i + 3, x_gyro[(m_headIdx + i) % m_WindowSize]);
      inst.setValue(i + 4, y_gyro[(m_headIdx + i) % m_WindowSize]);
      inst.setValue(i + 5, z_gyro[(m_headIdx + i) % m_WindowSize]);

      inst.setValue(i + 6, x_mag[(m_headIdx + i) % m_WindowSize]);
      inst.setValue(i + 7, y_mag[(m_headIdx + i) % m_WindowSize]);
      inst.setValue(i + 8, z_mag[(m_headIdx + i) % m_WindowSize]);

}

inst.setValue(9 * m_WindowSize, Utils.mean(acelX));
inst.setValue(9 * m_WindowSize + 1, Utils.mean(acelY));
inst.setValue(9 * m_WindowSize + 2, Utils.mean(acelZ));
inst.setValue(9 * m_WindowSize + 3, Utils.variance(acelX));
inst.setValue(9 * m_WindowSize + 4, Utils.variance(acelZ));
inst.setValue(9 * m_WindowSize + 5, Utils.variance(acelZ));
```

Algoritmo 7: Ciclo de construção de uma instância.

Com este método "makeInstance()" conseguimos obter no final uma instância completa, preparada para ser enviada para o servidor ou para ser classificada no próprio dispositivo móvel.

A possibilidade de proceder à classificação de cada instância diretamente no *smartphone* revelou-se mais complicada de implementar. A biblioteca WEKA não tem suporte oficial para o sistema operativo Android, tendo sido utilizada uma versão não oficial disponibilizada por RJ Marsan (http://rjmarsan.com/Research/WekaforAndroid/) designada de *Weka for Android*. Aparentemente isso teria resolvido o problema, no entanto, as tentativas de migrar objetos serializados no PC, usando a JVM para o Android que usa o Dalvik como máquina virtual, não foram bem-sucedidos. Esses objetos serializados, por exemplo um objeto MultilayerPerceptron representando uma rede neuronal após a aprendizagem, deveriam ser o resultado da aprendizagem realizado no PC. A abordagem revelou-se infrutífera devido às diferenças entre as máquinas virtuais onde era gerado o modelo do classificador e onde se tentava carregar o mesmo, JVM e Dalvik, respetivamente.

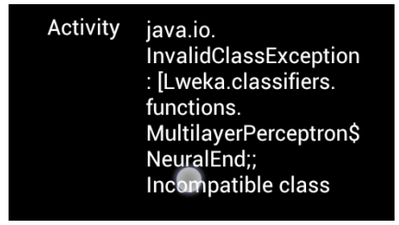

Figura 17: Exceção encontrada na tentativa de carregar o modelo de classificação

Depois de alguma pesquisa, as referências a este problema confirmaram o que se pensava, que realmente teria de ser criado o modelo, ou objeto serializado, na mesma máquina virtual em que se queria carregar. Foram executadas várias tentativas de criação do modelo em dispositivos móveis mas sem sucesso, devido à falta de recursos para elaborar tal tarefa. A abordagem alternativa passou por migrar o modelo aprendido da WEKA para o Android na forma de código fonte Java. Na verdade, a WEKA disponibiliza, nas últimas versões e apenas em alguns classificadores, uma interface *sourceable*. Nem todos os classificadores estão implementados, nomeadamente, um dos que ainda não estava, na versão da WEKA usada, era o algoritmo MultiLayerPerceptron. Foi no entanto possível, usando esta última abordagem adotar o LogitBoost como algoritmo para classificação das atividades, devido aos bons resultados que demonstrou no artigo (Torres et al., 2012).

Como mencionado anteriormente, a verificação do *radio button* que está presente no "onTick()", é que vai decidir onde vai ser classificado a nova instância. Se for classificado no dispositivo móvel vai utilizar um Wrapper da Weka que foi incorporado na aplicação derivado da incompatibilidade de outras formas de carregamento do classificador. Com isto o método que vai chamar o classificador fica simples, como é demonstrado no Algoritmo 8.

```
public void Classification(Instance inst) throws Exception{

      TextView tv_activity = (TextView) findViewById(R.id.tv_activity);
      TextView tv_history = (TextView) findViewById(R.id.tv_history);
      // predicting class
      double pred = m_Classifier.classifyInstance(inst);

      tv_history.setText(getPredictedClasseName(pred)+"\n"+tv_history.getText()
);
      tv_activity.setText(getPredictedClasseName(pred));

      //SENSOR FUSION
      sendTOSensorFusion(getPredictedClasseName(pred));
```

Algoritmo 8: Método de classificação de cada instância.

O que o classificador nos devolve é um número, que é convertido numa atividade pois, foi elaborado um mapeamento de atividades em relação a números, de 1 a 5, neste caso. Esse mapeamento é elaborado por um método criado chamado "getPredictedClassseName()" que não é nada mais que um *switch-case*. As atividades identificadas nesta aplicação através deste mecanismo são, parado, andar, correr, deitado e queda.

Quando a classificação é feita no servidor, apenas temos de carregar o modelo do classificador criado no package Weka e o cabeçalho configurado de igual forma como foi para o classificador do dispositivo móvel. Como é possível observar no Algoritmo 9, após termos o classificador criado, falta apenas receber via socket UDP a instância criada no *smartphone*, classificá-la e enviar o resultado de volta.

```
m_clsActivity = new
ClassificationActivity(".\\data\\nexus_header.arff",".\\data\\LogicBoost_201206
06.model");

      try {
            //ss = new ServerSocket(5432);
            serverSocket = new DatagramSocket(9876);
            clientSocket = new DatagramSocket();
            while (true) {

            System.out.println("I'm waiting...");
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            serverSocket.receive(receivePacket);
            inst = (Instance) toObject(receivePacket.getData());

            String res = Classification(inst);
            System.out.println(res+" << "+inst.toString());
            sendData = toByteArray(res);

            DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, InetAddress.getByName("192.168.1.74"), 9876);
                  clientSocket.send(sendPacket);
            }
      } catch (Exception e) {
          e.printStackTrace();
      }
```

Algoritmo 9: Método de classificação de cada instância no servidor.

Após a classificação de uma atividade, esta é enviada para um servidor através de um webservice disponível num endereço, com o fim de ser armazenada numa base de dados. Isto quer dizer que a cada segundo de execução da aplicação, esta está a enviar informações sobre o utente, como é percetível no algoritmo responsável chamado "SendTOSensorFusion()".

```
private void sendTOSensorFusion(String predictedClasseName) {
	try {

		Timestamp tstamp = new Timestamp(System.currentTimeMillis());
		String ts = tstamp.toString();
		ts.replace(" ", "%20");
		ts = ts.substring(0,19);

		String full_url = new
String("http://10.11.100.19/shhc/index.php?timestamp="+ts.replace(" ",
"%20")+"&app=ANDROID&event=add&action=&activity="+predictedClasseName);

		URL url = new URL(full_url);
		System.out.println(full_url);

		HttpClient hc = new DefaultHttpClient();
		HttpGet hg = new HttpGet(full_url);
		hc.execute(hg);

	} catch (MalformedURLException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}
```

Algoritmo 10: Função de envio dos dados para um webservice.

Esta aplicação tem uma interface simples e com poucos botões para que nada seja carregado sem qualquer intenção. Por conseguinte, na figura 15 é possível observar que esta interface contém a opção do local de classificação, assim como dois botões de ação, o *start* e o *pause*/*stop*.

Figura 18: Interface da aplicação de classificação de atividades.

Para construir um modelo do classificador, seja ele para o servidor ou para o dispositivo móvel numa versão de Wrapper, é preciso construir uma base de treino. Uma base de treino é um conjunto de dados relativos a atividades, neste caso, que são exercidas e classificadas como tal. Podemos dizer que é a partir da simulação das atividades que se faz com que a package Weka consiga criar um modelo conciso, com uma boa taxa de precisão.

Para construir esse conjunto de treino foi desenvolvida uma pequena aplicação para o dispositivo móvel, com o objetivo de ler os dados dos sensores e escrevê-los num pequeno ficheiro ".log" de forma organizada, por instâncias, tal como mostra a Figura 19.

```
-1.1683704853057861,9.27034854888916,-1.9728221893310547,-0.02932153269648552,-0.02687807008624077,0.02076941914856434,8.75,-30.5,11.3125,1
-1.2641384601593018,9.117119789123535,-2.5665841102600098,-0.13805554807186127,0.11972958594560623,0.008552113547921l8,7.5,-30.75,11.0625,1
-1.015141487121582,9.615114212036133,-1.9728221893310547,-0.06230825558304787,0.12705996632575 99,-0.010995574295520782,8.0,-30.5,10.5625,1
-1.3024457693099976,9.251194953918457,-2.2601263523101807,-0.13927727937698364,0.09285151958465576,0.025656340643763542,8.75,-30.75,10.0625,1
-1.1109095811843872,9.385271072387695,-2.3750481605529785,-0.06108652427792549,0.03543018549680 71,-0.03176499158143997,8.5,-30.0,10.0625,1
-1.1683704853057861,9.327810287475586,-2.126051187515259,-0.05009094998240471,0.03543018549680 71,-0.0403171069920063,9.0,-30.25,8.8125,1
-1.1300631761550903,9.232041358947754,-2.2601263523101807,-0.09162978827953339,-0.00610865233466029 2,-0.026878070086240 77,9.0,-30.25,8.25,1
-1.0342950820922852,9.461885452270508,-1.9153614044189453,-0.09162978827953339,-0.00366519158706069,0.037873644381761 55,8.0,-30.25,9.3125,1
-1.3024457693099976,9.442731857299805,-1.723825216293335,-0.0403171069920063,-0.04153883829712868,0.02321287989616394,9.25,-29.75,9.8125,1
-1.3024457693099976,9.442731857299805,-1.723825216293335,-0.0403171069920063,-0.04276056587696075 4,0.010995574295520782,9.0,-29.5,9.8125,1
-1.4939818382263184,9.385271072387695,-1.9345149993896484,0.05009094998240471,-0.01343903504312038 4,0.003665191587060 69,8.5,-30.25,9.5625,1
-1.5131354331970215,9.385271072387695,-1.9728221893310547,-0.05253441259264946,0.10384709388017654,0.0403171069920063,9.25,-30.25,8.8125,1
-1.091755986213684,9.232041358947754,-2.0111293792724 61,-0.12339477986097336,0.06597344577312 47,-0.010995574295520782,8.75,-30.25,8.8125,1
```

Figura 19: Exemplo de organização do ficheiro log.

Esta aplicação utiliza alguns dos algoritmos já aqui descritos, como o "onSensorChanged()" mas trata os dados dos sensores de maneira diferente. Para uma correta recolha dos dados são utilizados sinais sonoros assim como visuais para advertir o utilizador que o dispositivo está a gravar os dados. O *software* está configurado para

fornecer sinais sonoros durante 5 segundos após ter carregado no botão da atividade a desempenhar e por conseguinte no botão de "*start*". Nesses 5 segundos os dados dos sensores não estarão a ser gravados, só após esse tempo é iniciada a gravação. Quando o tempo de coleta dos dados tiver terminado, que depende diretamente da atividade que esteja a desempenhar, a aplicação executa igualmente um sinal sonoro. O pequeno método apresentado no algoritmo seguinte, é o que faz o "*cooldown*" até a aplicação começar a recolher dados dos sensores através da função "get_axis()".

```
private void cooldown() {
      CountDownTimer cdt = new CountDownTimer(5000, 1000) {
      public void onTick(long millisUntilFinished) {
            final ToneGenerator tg = new ToneGenerator(
            AudioManager.STREAM_NOTIFICATION, 100);
            tg.startTone(ToneGenerator.TONE_PROP_BEEP);
      }
      public void onFinish() {
            final ToneGenerator tg = new ToneGenerator(
            AudioManager.STREAM_NOTIFICATION, 1000);
            tg.startTone(ToneGenerator.TONE_PROP_ACK);
            get_axis();
      }
      }.start();
}
```

Algoritmo 11: Algoritmo de cooldown.

Na função "get_axis()", é feita a recolha dos dados faseada pelo tempo utilizando o CountDownTimer, que como foi referido anteriormente depende da atividade que se está a desenvolver. A aplicação foi configurada de modo a que a atividade de andar, correr e parado tivessem 1000 amostras, enquanto a atividade de cair e deitar teriam 100 amostras apenas. Isto foi definido desta forma devido às atividades que estariam a ser executadas, como o cair, em que é necessário levantar para voltar a simular outra queda. O algoritmo seguinte mostra a aplicação do método "onTick()" do CountDownTimer para o propósito acima referido.

```
CountDownTimer cdt = new CountDownTimer(1000, 1000/nr_amostras) {
      public void onTick(long millisUntilFinished) {

            try {
                  if((start)&&(count<todas_amostras_atividades)){
                        String data = axis_x + "," + axis_y + "," + axis_z +
                        "," + axis_x_gyro + "," + axis_y_gyro + "," +
                        axis_z_gyro + "," + axis_x_magn + "," + axis_y_magn +
                        "," + axis_z_magn;
                        if(cb.isChecked()){
                              data = data + "," +String.valueOf(activity);
                        }
                        data= data  + "\n";
                        out.write(data.getBytes());
                        count++;
                        tv.setText(" x= " + axis_x + "\n y= " + axis_y + "\n z=
                        " + axis_z + "\n count= " + count);
                  }else{
                        start=false;
                        final ToneGenerator tg = new ToneGenerator
                        (AudioManager.STREAM_NOTIFICATION, 1000);
                        tg.startTone(ToneGenerator.TONE_PROP_ACK);
                        button_start.setEnabled(true);
                        sb_sensibility.setEnabled(true);
                        pb.setVisibility(4);
                  }
            } catch (IOException e) {
                  e.printStackTrace();
      }
```

Algoritmo 12: Método de captura dos dados dos sensores por um determinado período de tempo.

Apenas os dados relativos ao acelerómetro são visíveis na própria interface pois, como o espaço de ecrã é reduzido consideramos este como o sensor principal para a deteção de atividades. Esta afirmação é justificada com algumas das referências que enfatizam que o acelerómetro é o sensor fulcral na deteção de atividades (Mccall et al., 2012). Ainda na interface são visíveis 5 botões relativos a atividades, uma barra para ajustar a frequência de amostragem e por fim mais quatro botões de ações.

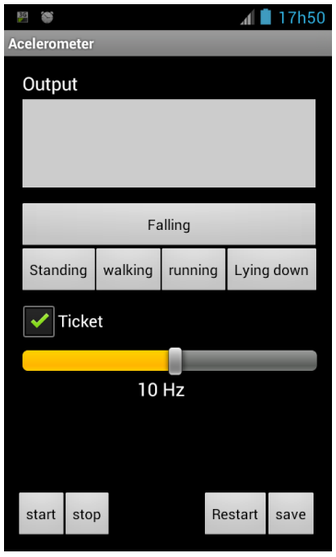

Figura 20: Interface da aplicação de recolha de dados de treino.

3.4.2 Presence Detection Module

Nesta subsecção é apresentada a solução desenvolvida com o intuito de determinar se o utilizador que está a ser monitorizado dentro de um espaço inteligente, se encontra em frente a algum elemento em específico. Estes locais predefinidos vão estar a ser monitorizados por câmaras, para que se consiga obter informação relativamente a ações que possa tomar. Esta informação é enviada através de um *webservice* disponibilizado através do servidor, para que estes dados sejam guardados na base de dados do sistema.

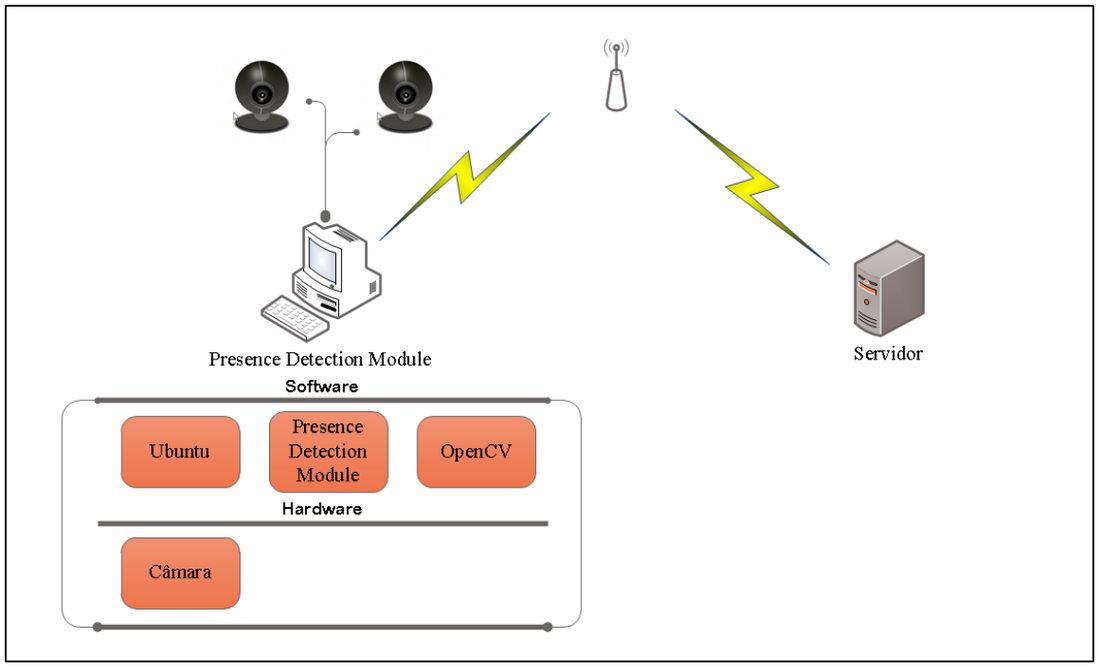

Figura 21: Arquitetura na ótica do Presence Detection Module.

O desenvolvimento do *software* relativamente à análise de imagem baseou-se maioritariamente nos exemplos incluídos no package do OpenCV, cujos algoritmos foram, obviamente, tratados e manipulados com uma finalidade e para satisfazer os objetivos específicos neste cenário. Foram utilizados algoritmos de visão por computador para deteção de partes do corpo humano, como a face, os olhos ou mesmo o corpo todo. A parte essencial desta aplicação era mesmo a face, pois era através dela que nós tentávamos reconhecer o nosso utente que está a ser monitorizado. Só o reconhecimento do utente é que leva a aplicação a enviar informações para o servidor sobre a localização do utente.

Para a deteção de partes do corpo foram utilizados modelos baseados em haarcascades (Viola, Way, & Jones, 2004), que são guardados em ficheiros xml que contêm dados biométricos de pessoas de uma forma generalizada, e que podem ser usados para identificar com alguma taxa de sucesso uma parte ou o corpo todo numa imagem que pode ser adquirida através de uma webcam.

```
/** Global variables */
String face_cascade_name =
"C://opencv//data//haarcascades//haarcascade_frontalface_alt.xml";
String eyes_cascade_name =
"C://opencv//data//haarcascades//haarcascade_eye.xml";
String body_cascade_name =
"C://opencv//data//haarcascades//haarcascade_fullbody.xml";

CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;
CascadeClassifier body_cascade;
```

Algoritmo 13: Declaração dos classificadores haarcascade.

Inicialmente foram definidas algumas variáveis globais utilizadas no decorrer do projeto, como é exemplificado na figura apresentada acima. No início do programa, começámos por carregar alguns elementos em memória como os haarcascades. Foi também carregado inicialmente o ficheiro que contêm a indicação das faces recorrentes ao reconhecimento de face que vai ser descrito mais à frente. O *Fisher Face Recognizer* que foi o método utilizado para fazer esse reconhecimento de faces, necessita inicialmente de "treinar" a sua base, com os dados existentes, como é apresentado no algoritmo seguinte.

```
string fn_csv = string("D://att/at.txt");
// These vectors hold the images and corresponding labels.
vector<Mat> images;
vector<int> labels;

try{
      read_csv(fn_csv, images, labels);
}
catch (cv::Exception& e)
{
      cerr << "Error opening file \"" << fn_csv << "\". Reason: " << e.msg <<
      endl;
      // nothing more we can do
      exit(1);
}
if(images.size() <= 1){
      string error_message = "This application needs at least 2 images to work.
      Please add more images to your data set!";
      CV_Error(CV_StsError, error_message);
}

images.pop_back();
labels.pop_back();

Ptr<FaceRecognizer> model = createFisherFaceRecognizer();
model->train(images, labels);
```

Algoritmo 14: Criação do modelo de reconhecimento.

Como é percetível no algoritmo 14 são necessárias pelo menos 2 imagens para funcionar, no entanto o aconselhável são 10 pois, foi com este número de faces que foi desenvolvido e testado.

A seguir a esta inicialização, entra-se no ciclo de aquisição de *frames* da câmara para tratamento, tanto para deteção da face, como para reconhecimento e ainda mesmo para a aplicação do método de *Background Subtraction*. Este ciclo inicia com a tentativa de deteção de faces ou mesmo corpos na *frame* em questão, mas antes esta sofre transformação nas suas cores na passagem de BGR para GRAY e na equalização do histograma das mesmas.

Se for encontrada alguma face na *frame*, o passo seguinte é tentar encontrar os olhos antes de desenhar, isto faz com que aconteçam menos deteções erradas. As faces detetadas são colocadas num array com as suas coordenadas, para depois serem tratadas uma a uma. Aquando do tratamento individual das mesmas, é criada uma imagem nova apenas com a face da pessoa, isto para depois executar a tentativa de reconhecimento. O algoritmo 15 mostra alguns destes procedimentos, incluindo a opção de guardar a imagem em ficheiro pressionando a tecla "c" quando a cara for detetada. Isto serviu para construir a base de imagens para procedimentos de testes.

```
//-- Detect faces
face_cascade.detectMultiScale( frame_gray, faces, 1.1, 2, 0, Size(10, 10) );

for( int i = 0; i < faces.size(); i++ ){
      Mat faceROI = frame_gray( faces[i] );
      std::vector<Rect> eyes;
      Rect face_i = faces[i];

      Mat face_resized;

      cv::resize(faceROI, face_resized, Size(92, 112), 1.0, 1.0, INTER_CUBIC);
      int keycode = waitKey(30);
      if( keycode == 'c' ){
            String filename = "D://att/face.bmp";
            printf("face file created!");
            imwrite(filename,face_resized);
      }
```

Algoritmo 15: Deteção de caras e criação de uma imagem bitmap.

Depois de detetada a face, o passo seguinte como já foi referido é detetar os olhos. Se os mesmos forem detetados, a aplicação tenta fazer o reconhecimento da cara através do método de Fisher Faces. Se a cara for

reconhecida pela função ou mesmo que não seja, ele devolve um número que pode ser considerado um id da pessoa em questão que vai fazer match com um nome pré-definido.

```
eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0 |CV_HAAR_SCALE_IMAGE,
Size(10, 10) );
if( eyes.size() == 2){
      pred = facerec(face_resized,model);

      //-- Draw the face
      Point center( faces[i].x + faces[i].width*0.5, faces[i].y +
      faces[i].height * 0.5 );
      ellipse( frame, center, Size( faces[i].width*0.5, faces[i].height*0.5),
      0, 0, 360, Scalar( 255, 0, 0 ), 2, 8, 0 );

      string box_text = format("Prediction = %d", pred);
      if (pred==40){
            box_text = "Sergio Vale";
            idfound=true;
      }else if(pred==39){
            box_text = "Carina Silva";
            idfound=false;
      }else{
            box_text = "Unknown";
            idfound=false;
      }
      int pos_x = std::max(face_i.tl().x - 10, 0);
      int pos_y = std::max(face_i.tl().y - 10, 0);

      putText(frame, box_text, Point(pos_x, pos_y), FONT_HERSHEY_PLAIN, 1.0,
      CV_RGB(0,255,0), 2.0);
```

Algoritmo 16: Deteção de olhos e reconhecimento de faces.

Como é possível verificar no algoritmo acima foram pré-definidas apenas duas pessoas, em que uma seria o utilizador a ser monitorizado (pred == 40) e a outra poderia ser o *caregiver*. O que nos interessa é detetar o utilizador, quando ele aparece na imagem da câmara e deste modo acionar uma *flag* para enviar a informação para o servidor. Depois disto apenas fica a faltar a deteção de algum corpo que, se passar na frente da câmara será detetado de igual forma, sendo desenhado um círculo em volta.

O algoritmo "sendTOSensorFusion()" foi desenvolvido com o objetivo de enviar informação para o servidor, acedendo a um *webservice* disponibilizado pelo mesmo. A informação enviada depende do posicionamento da câmara e, neste caso, o *software* que irá ser apresentado está configurado para enviar informação quando o utilizador está próximo do telefone. Os outros elementos também controlados são, o drugdispenser e a televisão, ainda existindo a opção do output ser "*outra*", quando o utilizador não for identificado

em nenhuma câmara depois de ter aparecido uma primeira vez. O algoritmo seguinte mostra como isso foi feito e alguns parâmetros estáticos enviados como a identificação da aplicação e o tipo de evento.

```
void sendTOSensorFusion(int action)
{
      char date[30];
      time_t t = time(0);
      struct tm *tm;
      tm = gmtime(&t);
      strftime(date, sizeof(date), "%Y-%m-%d %H:%M:%S", tm);
      String datet = date;
      char* url;

      url = (char*)malloc(200*sizeof(url));

      snprintf(url,200,"%s%s" , "" ,
      "http://10.11.100.19/shhc/index.php?timestamp=" );
      snprintf(url,200,"%s%s",url,date);
      snprintf(url,200,"%s%s",url,"&app=CAMARAS&event=add&activity=&action=");
       if (action==1){
            snprintf(url,200,"%s%s",url,"PHONE");
      }else{
            snprintf(url,200,"%s%s",url,"OTHER");
      }

      HINTERNET hINet,hIUrl;
      hINet = InternetOpen("connection", INTERNET_OPEN_TYPE_PRECONFIG, NULL,
      NULL, 0 );

      InternetCloseHandle(hIUrl);
      InternetCloseHandle(hINet);

      free(url);
}
```

Algoritmo 17: Algoritmo de envio de dados para o webservice.

Para execução do Background Subtraction foi utilizado o BackgroundSubtractorMOG (Kaehler & Bradski, 2008) que é uma classe já implementada pelo OpenCV para este propósito. Esta classe foi configurada na aplicação desenvolvida com o modo de aprendizagem ativo, no entanto, para terminar esse modo é só pressionar a tecla "*space*". Este método de *Background Subtraction*, como o próprio nome indica, remove tudo o que faça parte do plano de fundo, do ambiente estático.

3.4.3 Fusão de informação

Nesta subsecção é descrito o desenvolvimento realizado para obtenção de um objetivo que é a fusão de informação. Com este propósito, foi utilizado um servidor que reunia toda a informação reportada pelo Activity Detection Module e pelo Presence Detection Module, como é possível observar na Figura 22.

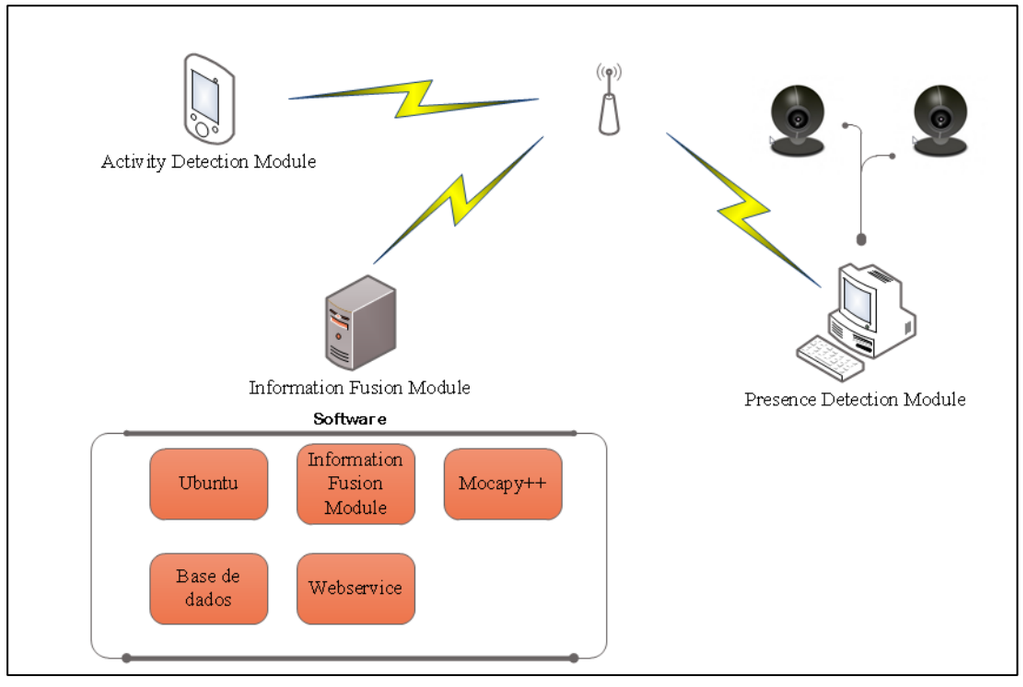

Figura 22: Arquitetura na ótica do servidor.

Para fazer a fusão da informação foi construída uma aplicação a qual designamos por *Information Fusion Module*, em que a mesma procede à recolha da informação armazenada para obter um resultado. Para chegar a tal objetivo a aplicação tem de iniciar por um ciclo de aprendizagem onde vai fazer o reconhecimento de padrões relativamente a atividades e ações, através do *Activity Detection Module* e *Presence Detection Module*, respetivamente. Depois de criado o modelo de aprendizagem é que obtemos um conjunto de valores relativamente aos últimos segundos reportados pelos módulos e procedemos à sua classificação.

Os dados são enviados para o servidor através de um *webservice* desenvolvido em PHP que funciona sobre o protocolo HTTP que o que faz é, coordenar os dados com os campos disponíveis na base de dados concretizada em MySQL e inseri-los. Para fazer a inserção na base de dados, foi utilizada uma linguagem recorrente para estes casos que é o SQL através do método INSERT.

```
function add_event($tstamp,$action,$app,$activity){

      $query = "INSERT INTO `history` (`id` ,`timestamp` ,`app` ,`action`
      ,`activity`) VALUES (NULL,'$tstamp','$app','$action','$activity')";
      $result = mysql_query($query);
      echo mysql_error();

      return $result;
}
```

Algoritmo 18: Função de inserção na base de dados.

Para elaborar a fusão da informação, foi usada a biblioteca Mocapy++ implementada em C++ para construção de uma rede bayesiana dinâmica. Para a execução desta biblioteca essencial a todo este projeto foi necessário instalar outras bibliotecas às quais esta tinha dependência que são, a LAPACK, a Boost e a GNU Fortran. Devido ao facto da biblioteca Mocapy++ esta escrita em C++, o desenvolvimento do módulo de *software* para tomada de decisão também foi implementado na linguagem de programação C++.

A implementação do módulo de *software*, começa com a declaração dos nós observados (*Observed nodes*) e do nó oculto (*Hidden node*), os quais são possíveis de observar na figura 10 facultada no início do capítulo 3. Os nós observados são os elementos que fornecem dados para ajudar a determinar o nó oculto, em que neste caso específico estamos a falar do Activity Detection Module, do *Presence Detection Module* e do *Agenda Module*. Para inicialização destes nós é necessário referir não só um nome, mas também o número de valores diferentes que estes conseguem obter. Normalmente e como é visível no algoritmo seguinte, os nós observados são intitulados por “O + número”, e os nós ocultos por “H + número”.

```
DBN dbn;

// Nodes in slice 1
Node* h1 = NodeFactory::new_discrete_node(5, "h1"); // Hidden Node
Node* o1 = NodeFactory::new_discrete_node(5, "o1"); // Act. Fisica ( Android )
Node* o2 = NodeFactory::new_discrete_node(2, "o2"); // Agenda (Drug Dispenser)
Node* o3 = NodeFactory::new_discrete_node(4, "o3"); // Act. Funcional (Camaras)

// Nodes in slice 2
Node* h2 = NodeFactory::new_discrete_node(5, "h2"); //Hidden Node
```

Algoritmo 19: Declaração dos observed nodes and hidden nodes.

No código apresentado acima conseguimos observar que possuímos um nó oculto “h2” que no entanto, não passa de uma réplica do “h1” mas, é necessário para construção da arquitetura da rede bayesiana dinâmica como vai ser demonstrado a seguir.

Depois da declaração dos nós é necessário definir a arquitetura subjacente a esta rede e determinar as ligações entre os mesmos. Como já referido na secção 2.3.3, as ligações são definidas nas redes bayesianas dinâmicas entre *inter edges* e *intra edges*, que satisfazem a ligação entre nós da mesma fatia e entre nós de fatias diferentes, respetivamente. Isto é importante para o Mocapy++ conseguir construir as tabelas de probabilidade condicionada, que depende do número e de quais são as variáveis condicionantes. É aqui que entra o nó “h2”, pois precisamos determinar neste tipo de redes a CPT entre nós ocultos pois, isto vai ajudar a chegar ao resultado do nó t sabendo o que aconteceu ao nó t-1.

```
// Set architecture
dbn.set_slices(vec(h1, o1, o2, o3), vec(h2, o1, o2, o3));

// Definir conexoes
dbn.add_intra("h1", "o1");
dbn.add_intra("h1", "o2");
dbn.add_intra("h1", "o3");
dbn.add_inter("h1", "h2");

// COnstrução da rede
dbn.construct();
```

Algoritmo 20: Determinação da arquitetura e ligações entre nós.

Depois de construída a rede bayesiana foi utilizado o método *expectation maximization* (EM) (Paluszewski & Hamelryck, 2010) para fazer a aprendizagem dos parâmetros da rede. Com esta finalidade tivemos de instanciar a classe EMEngine e carregar os dados recolhidos dos nós observados e ocultos, assim como dados que informam o tipo de nós. Na aprendizagem e treinamento dos dados propriamente ditos, o método EM divide-se em dois passos, o E-step que faz a inferência dos nós ocultos utilizando a configuração atual dos parâmetros da DBN, e o M-step utiliza os valores inferidos dos nós ocultos para atualizar os parâmetros da DBN. Depois de estes dois passos convergirem atingimos o objetivo que é produzir uma estimativa do ponto de máxima probabilidade dos parâmetros. No algoritmo seguinte conseguimos observar o que foi referido, e que após isto decidimos guardar a DBN para garantir a sua consistência e possibilidade de utilização futura.

```
cout << "Loading traindata" << endl;
GibbsRandom mcmc = GibbsRandom(&dbn);

EMEngine em = EMEngine(&dbn, &mcmc);
em.load_mismask("data/mismask_test.dat"); // carregar ficheiro com informação
sobre os observed nodes
em.load_sequences("data/traindata_test.dat"); // carregar ficheiro com dados de
treino

// treinar a rede com os dados carregados
cout << "Starting EM loop" << endl;
for (uint i=0; i<100; i++) {
      em.do_E_step(20,10);
      double ll = em.get_loglik();
      cout << "LL= " << ll << endl;
      em.do_M_step();
}
// guardar a bayesian network
dbn.save("dbn.mocapy");
```

Algoritmo 21: Parameter learning.

O passo seguinte depois da aprendizagem dos parâmetros, consiste em ir obtendo os dados relativamente aos nós observados, e a partir dos mesmos permitir que o módulo de decisão estime qual o valor do *hidden node* associado. Para este efeito, foi elaborado um *script* em PHP para consulta da base de dados onde estão presentes os todos dados do sistema e com isto construir um ficheiro de texto já preparado para o Mocapy++ o conseguir perceber. Este tipo de ficheiros que esta biblioteca suporta tem de ter uma configuração especial, que passa por identificar nós através de espaços, fatias por linhas e linhas em branco separam sequências.

Após esses dados carregados é elaborada a determinação dos *hidden nodes* através da classe InfEngineMCMC que faz a aproximação das probabilidades das sequências através das sequências já geradas. Aplicando o *Viterbi path*, é feita a aproximação de sequências através da geração sucessiva de um determinado número de sequências até convergir. O algoritmo seguinte mostra como o referido anteriormente foi implementado, assim como foi feita a contagem dos resultados obtidos através de um array bidimensional devido à possibilidade de existirem várias fatias.

```
GibbsRandom mcmc2 = GibbsRandom(&dbn);
InfEngineMCMC inf = InfEngineMCMC(&dbn, &mcmc2, &seq_data, mismask);
inf.initialize_sample_generator(MCMC_BURN_IN,  true);
//burn_in_steps=MCMC_BURN_IN, init_random=true

// quantas mais iterações mais preciso fica o calculo da probabilidade mas,
mais lento fica o sistema
uint N_ITER=100;
int h_node_posterior[NR_SLICES][5]={0,0,0,0,0}; // counting for h==0 for each
slice
for (uint i=0; i<N_ITER; i++) {
      Sample s = inf.sample_next();
      cout << "Viterbi LL= " << s.ll << endl;
      cout << "Viterbi seq= " << s.seq << endl;
      for (uint slice=0; slice<s.slice_count; slice++) {// Contagem dos hidden
      nodes
            if (s.seq.get(slice,0)==0) h_node_posterior[slice][0]++;
            if (s.seq.get(slice,0)==1) h_node_posterior[slice][1]++;
            if (s.seq.get(slice,0)==2) h_node_posterior[slice][2]++;
            if (s.seq.get(slice,0)==3) h_node_posterior[slice][3]++;
            if (s.seq.get(slice,0)==4) h_node_posterior[slice][4]++;
      }
      cout << "Viterbi slice_count= " << s.slice_count << endl;
}
```

Algoritmo 22: Determinação do Viterbi Path

Com isto apenas fica a faltar a apresentação dos resultados que são as probabilidades finais, que com o mapeamento correto são traduzidas em ações. Este sistema tem como output as seguintes ações: ver televisão, perto do *drug dispenser*, perto do telefone, exercício em frente à televisão e "outra" (ação que não é identificável com nenhuma das opções anteriores).

A base de dados onde a informação está guardada apenas contém uma tabela com os campos considerados essenciais à execução deste projeto nomeadamente, um id, um *timestamp*, a aplicação que submeteu a informação, qual o tipo de evento, a ação detetada e a atividade física. Estes dois últimos campos são preenchidos dependendo do tipo de dispositivo que reporta a inserção de informação, isto é, o dispositivo móvel reporta atividades físicas enquanto que a aplicação desenvolvida para as câmaras reporta ações.

3.5 Conclusão

O processo de comunicação entre os vários módulos de deteção e o módulo de fusão de informação obrigou ao desenvolvimento de uma interface de ligação que teve de resolver problemas tecnológicos nas várias camadas

da pilha protocolar de comunicação. A definição da tecnologia a utilizar foi, por isso, alvo de especial atenção, para que o sistema funcionasse corretamente. Depois de alguma investigação sobre o assunto, foi escolhida a tecnologia WIFI. A comunicação dos módulos é fundamental para que exista a possibilidade de elaborar a fusão e por isso, depois de definida a tecnologia a utilizar, globalmente, o projeto PADS progrediu bastante a nível estrutural. No Capítulo 4 são apresentados alguns testes elaborados ao protótipo do sistema desenvolvido,, mais especificamente ao módulo de deteção de atividades e ao módulo de fusão de informação.

Capítulo 4: Avaliação

Neste capítulo são apresentadas várias avaliações parcelares realizadas sobre o sistema criado, PADS, de forma independente. Na secção de avaliação da atividade física é apresentada a avaliação relativa às aplicações desenvolvidas para *smartphones*, a secção de avaliação da fusão centra-se no sistema em geral como um todo (*Sensor Fusion*). Neste capítulo são igualmente explicados todos os procedimentos realizados de forma a comprovar as experiências realizadas a todo o sistema, expondo de que forma e quais os *outputs* atingíveis.

4.1 Componentes da avaliação do PADS

Para a construção deste sistema de deteção de atividades pessoal foi utilizado o seguinte equipamento: um *smartphone*, várias câmaras e um servidor. Ambos os componentes comunicavam com o servidor através de uma ligação sem fios WIFI para troca de informação necessária para o correto funcionamento de todo o sistema, como é possível observar na Figura 13 presente na seção 3.1Componentes do PADS. No computador é onde ficam armazenados todos os dados relativamente ao utilizador e é no mesmo que ocorre a fusão de informação para obtenção de um resultado. Este resultado provém do que o individuo que está ser monitorizado está a fazer no momento que pode ser influenciado pelo que fez num passado recente.

4.2 Avaliação da atividade física

Nesta subsecção são reveladas as experiências efetuadas e todos os detalhes relativos às mesmas, nomeadamente, a deteção de atividade física através de um *smartphone* com o OS Android. Para esse efeito foi utilizado um equipamento Samsung Nexus S que possui um processador ARM de 1Ghz, 512 Mb de memória e os sensores necessários à aplicação desenvolvida, designadamente um acelerómetro de três eixos, um giroscópio e um magnetómetro. Com este cenário foi possível recolher dados relativos às atividades que se pretendia identificar nomeadamente: parado, andar, correr, queda e deitado, para assim construir a base de treino do nosso modelo de classificação. A aplicação de aquisição desenvolvida para o efeito, isto é, para recolher os dados dos sensores

enquanto o utilizador está a exercer uma das cinco atividades, contém a opção de seleção da atividade a executar antes de o utilizar a realizar, assim como ajustar a taxa de amostragem que pode ir de 1Hz a 20Hz.

Nos testes elaborados definimos uma taxa de amostragem de 10Hz isto, para termos uma resolução temporal suficiente no conjunto de dados de teste. Relativamente à posição corporal do equipamento, optou-se pela colocação do dispositivo no peito (e.g., no bolso da camisa). Na tentativa de conter o mínimo de ruído nos dados recolhidos dos sensores foram incorporados os temporizadores sonoros e a sinalização de início e fim de captura dos dados para melhor controlo por parte de quem se está a submeter ao teste.

O intervalo de tempo de aquisição depende da atividade que o utilizador está a executar. No caso do sinal ser andar, correr ou parado, o tempo de captura é de 100 segundos, enquanto na simulação de quedas ou no estado deitado é de 10 segundos. No caso especial das quedas mesmo com o tempo de 10 segundos foi necessário retirar parte dos dados antes e depois da queda (que tipicamente reflete o estar deitado) pois introduziam ruído referente ao momento antes da queda e ao momento depois da queda, respetivamente. Tipicamente, as 5 atividades reportam valores diferentes do acelerómetro assim como sequências diferentes, visto o acelerómetro constituir dos 3 sensores usados, aquele que se revela o mais importante na classificação. Nas figuras seguintes consegue-se verificar o referido pois, todos os gráficos que representam um excerto dos valores obtidos para cada atividade são de certa forma diferentes uns dos outros. Para todas as atividades foi possível obter dados de modo contínuo exceto na queda, pois nesta ultima a atividade nunca ocorrem repetições consecutivas d ela mesma, ou seja, a seguir a uma queda nunca vem uma queda mas possivelmente a atividade deitado, como é possível observar na Figura 27.

Nos valores detetados pelo acelerómetro para o estado “parado” é possível verificar que o eixo *Y* contém um valor aproximado de $10m/s^2$ devido à gravidade à superfície da Terra, enquanto os outros dois eixos rondam os $0m/s^2$. Na Figura 23 verifica-se que ambos os eixos não têm uma variação de valores independentemente do espaço temporal da atividade, possuindo desta forma, sinais completamente diferentes de qualquer outra atividade. Para o estado “andar” os resultados obtidos pelo acelerómetro variam constantemente no tempo em todos os eixos, mas comparativamente com o estado “correr”, podemos ver que a variação relativamente ao eixo do *Y* é mais acentuada. A Figura 24 demonstra os dados recolhidos para o estado anteriormente referido, verificando-se algumas semelhanças com a Figura 23, o que pode interferir no algoritmo classificador na determinação da atividade. O estado “deitado” é apresentado na Figura 26, que indica variações não muito acentuadas nos eixos, devido a ser uma posição que não varia no espaço físico. Fora o acontecimento normal que é quando o utilizador

se deita, esta atividade pode ser detetada em sequência a uma queda. Esta sequência de atividades faz com que por vezes a atividade de queda seja detetada erradamente, isto quando o utilizador está apenas a deitar-se. A Figura 27 mostra os dados de uma queda num espaço temporal de 1 segundo, no entanto verifica-se que instantes antes do acontecimento o estado é "parado", "andar" ou "correr" e o estado final é "deitado". A variação do resultado dos eixos traduz esta mesma sequência de atividades e que a deteção de uma queda corretamente é uma tarefa muito difícil.

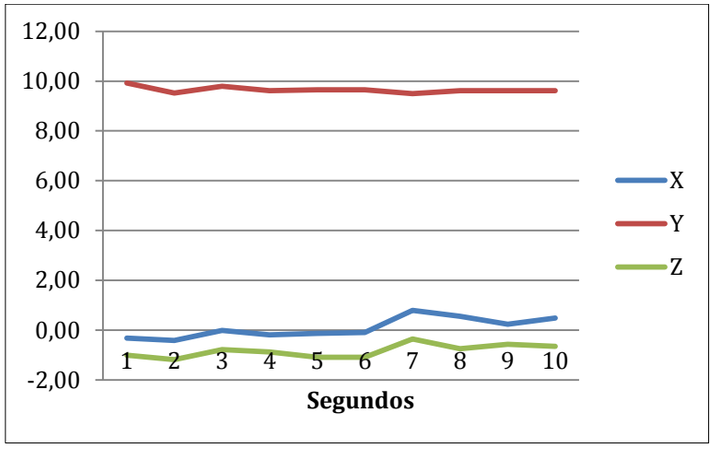

Figura 23: Dados do acelerómetro para o estado de Parado.

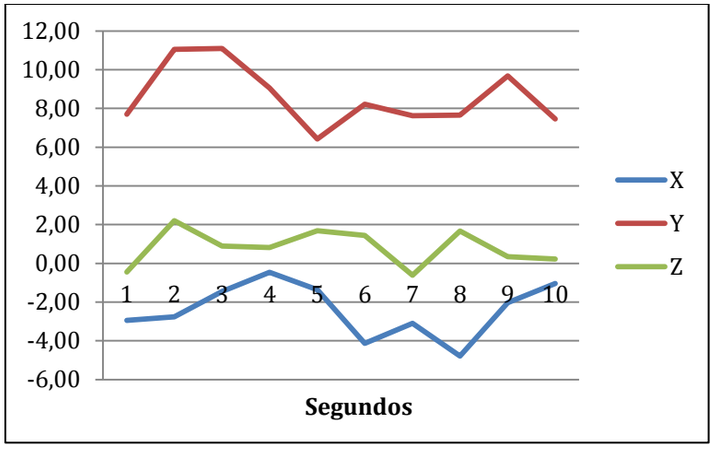

Figura 24: Dados do acelerómetro para o estado de Andar.

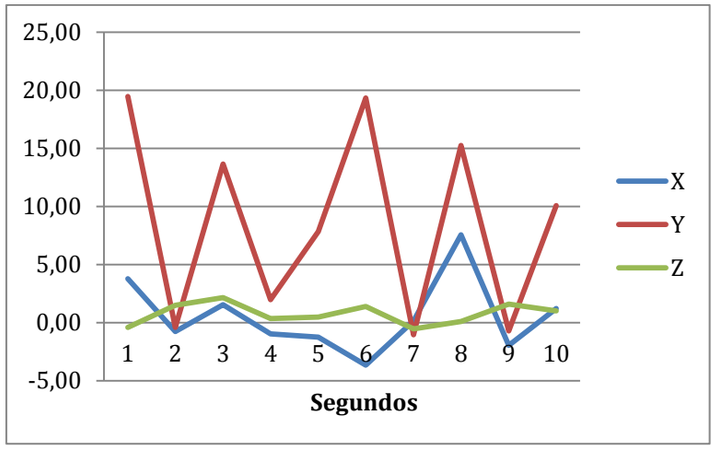

Figura 25: Dados do acelerómetro para o estado de Correr.

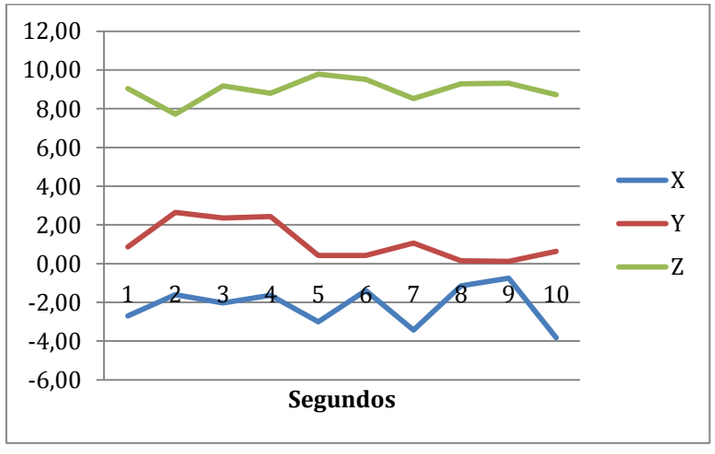

Figura 26: Dados do acelerómetro para o estado de Deitado.

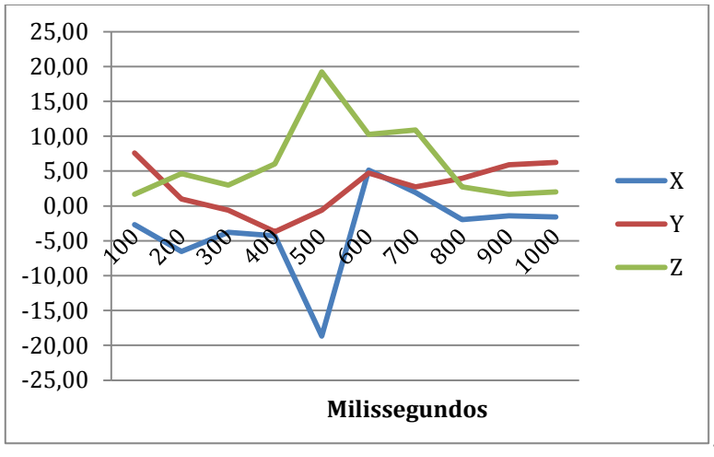

Figura 27: Dados do acelerómetro para o estado de Queda.

Com base no exporto foi criada base de treino que incorporava cerca de 766 instâncias, em que cada instância continha um valor de cada eixo de cada sensor, incluindo a respetiva média e variância que faz um total de 108 valores que identificam uma atividade em apenas 1 segundo. Todas as atividades de aquisição dos dados de teste foram desempenhadas sobre uma superfície plana, utilizando dois indivíduos. Após termos o conjunto de treino, foi criado o ficheiro ARFF com o header configurado corretamente e a consequente inserção dos dados do conjunto no local correto, isto recorrendo a um script previamente criado. Na Figura 28 é apresentado um excerto desse mesmo ficheiro com alguns dos atributos representados, nomeadamente um atributo referente a um eixo de cada sensor. Uma instância contém 10 atributos para cada eixo de cada sensor que são preenchidos a cada 100 milissegundos, que perfaz um conjunto de dados relativos a uma atividade durante 1 segundo. A seguir aos referidos atributos vem os dados propriamente ditos, em que cada linha significa uma instância e os valores estão separados apenas por vírgulas, pela mesma ordem que foram definidos os atributos previamente.

```
@relation nexus_instances

@attribute accelX0 numeric
@attribute accelY0 numeric
@attribute accelZ0 numeric
@attribute gyroX0 numeric
@attribute gyroY0 numeric
@attribute gyroZ0 numeric
@attribute magnX0 numeric
@attribute magnY0 numeric
…
@attribute magnZ9 numeric
```

```
@data
-0.325611,9.921572,…,-1.015141
-0.42138,9.519346,…,-1.187524
-0.019154,9.787497,…,-0.785298
-0.191536,9.615114,…,-0.881066
-0.134075,9.653421,…,-1.091756
-0.095768,9.653421,…,-1.091756
0.785298,9.500193,…,-0.363919
0.555455,9.615114,…,-0.746991
0.47884,9.615114,…,-0.651223
…
0.497994,9.576807,…,-0.804452
```

Figura 28: Excerto do ficheiro ARFF.

O ficheiro ARFF foi então submetido no GUI disponibilizado pela WEKA para criação do modelo de classificação.

Como referido no capítulo 3.4.1, na implementação da aplicação de classificação de atividades, inicialmente foram efetuados pré-testes para determinar qual o algoritmo que possuía uma melhor capacidade de desempenho na deteção de atividades. Em seguimento ao que foi concluído no artigo (Torres et al., 2012) o algoritmo que apresentou melhores resultados foi o LogitBoost. Para efetuar testes a outros algoritmos, foi adicionada a opção de classificação através de um servidor previamente preparado para essa situação. Na criação do modelo de aprendizagem com o algoritmo pretendido, foi utilizada a técnica 10-fold cross validation para obter o desempenho associado ao mesmo. Esta técnica reflete a divisão do conjunto de dados em 10 proporções iguais, em

que uma proporção de cada vez é usada para testar enquanto as outras 9 são usadas como base de treino do modelo preditivo. O resumo dos resultados obtidos utilizando o esquema de aprendizagem do LogitBoost e aplicando a técnica referida, estão expressos na forma de uma matriz confusão que é exposta na tabela 9. Este tipo de tabela é utilizado normalmente para avaliar o desempenho de um algoritmo, em que cada linha representa a atividade da instância e cada coluna a atividade predita.

Tabela 9: Matriz confusão do algoritmo LogitBoost.

Evento	Classificado como:				
	A	B	C	D	E
A	200	0	0	0	0
B	0	197	3	0	0
C	0	4	195	0	1
D	1	2	0	132	5
E	0	1	2	6	17
A: Parado B: Andar C: Correr D: Deitado E: Queda					

Tabela 10: Matriz confusão do algoritmo LogitBoost em percentagem.

Evento	Classificado como:				
	A	B	C	D	E
A	100%	0%	0%	0%	0%
B	0%	98,5%	1,5%	0%	0%
C	0%	2%	97,5%	0%	0,5%
D	0,7%	1,4%	0%	94,3%	3,6%
E	0%	3,8%	7,7%	23,1%	65,4%
A: Parado B: Andar C: Correr D: Deitado E: Queda					

Na tabela acima apresentada é possível observar que existem alguns dados classificados de forma errada, ou seja, inicialmente tinham sido fornecidos como por exemplo valores da atividade “Parado” e o algoritmo os classificou como “Queda” na fase de testes. É possível também ver que existem atividades que se cofundem

devido aos movimentos que por vezes podem ser similares e se traduzem em classificações erradas. Este tipo de erros verificam-se mais na classificação entre "Andar" e "Correr", assim como, entre "Queda" e "Deitado". Neste ultimo caso é de alguma forma compreensível que este tipo de erros ocorram, visto que após um queda a atividade a ser reportada é "Deitado". Mesmo com estes erros conseguimos atingir com este conjunto de dados através da técnica 10-fold cross validation uma precisão de 96,73% e consequentemente uma taxa de erro a rondar os 3,27%.

4.3 Avaliação da determinação da atividade por fusão

A avaliação feita ao sistema PADS ao nível da fusão da informação, envolveu o dispositivo móvel para deteção da atividade física e as câmaras para análise da atividade funcional. O dispositivo móvel utilizado foi o mesmo que foi descrito no subcapítulo 4.2 na avaliação da atividade física, e para a câmara utilizada foi uma webcam com cerca de 2M pixéis e configurada com uma resolução de 640px por 320px.

4.3.1 Cenário

Para testar o funcionamento do sistema foi necessário construir um cenário de teste que envolvesse o maior número de elementos pertencentes ao sistema. Deste modo, ficou definido o cenário: "A Mary circula pela sala em direção à televisão e senta-se na sua frente a ver o seu programa favorito" (Soares, Moreira, Morla, Sobral, & Torres, 2012). A Mary é o nome associado a uma utilizadora que fez o percurso definido andando até chegar ao sofá e se sentou em frente à televisão. A Figura 29 apresenta o esquema que traduz o cenário apresentado, onde o tracejado é o percurso do utente na sala infraestruturada com os dispositivos sensores. Todo este caso se passa sobre um superfície plana onde temos apenas uma pessoa presente, tal como sugere o cenário.

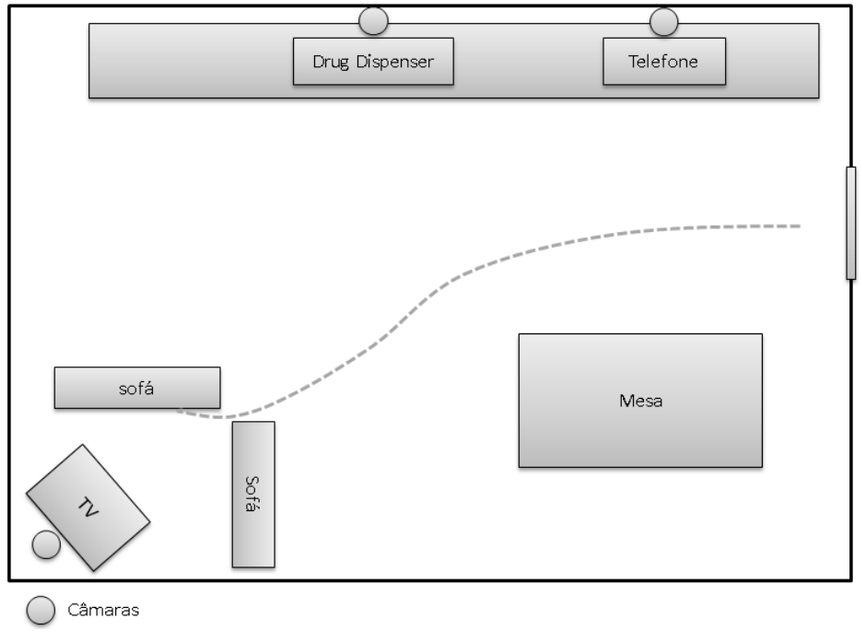

Figura 29: Esquema de distribuição dos sistemas off-the-shelf num cenário SHHC.

Para fazer a aprendizagem dos parâmetros do modelo da rede bayesiana dinâmica definida para implementar a componente de fusão da informação foi necessário alimentar esta rede com uma base de treino, para a mesma saber como e de que forma as informações recebidas dos vários elementos se traduzem num resultado final. A base de treino foi construída a partir de dados pré-processados que refletem o que se espera que ocorra na realidade. Foram também introduzidos nos dados, conhecimento sobre acontecimentos fora do padrão, isto para que a rede bayesiana esteja adaptada e consiga reagir corretamente a estas situações. Um bom exemplo para este tipo de acontecimentos é quando o utilizador está a ver televisão e olha para qualquer outro ponto da casa, nesse momento a câmara fica sem detetar a cara e portanto a sua presença mantendo-se no entanto a pessoa em frente à televisão.

Depois de feita a aprendizagem dos dados o sistema utiliza o algoritmo *Viterbi Path* para calcular o caminho mais provável para os dados reportados pelos elementos pertencentes ao sistema PADS: O caminho mais provável entenda-se por o que possivelmente o utente está a fazer no ambiente monitorizado, mais concretamente nos últimos 10 segundos. Os outputs configurados para este sistema são “Em frente à TV”, “A tomar medicação”, “A fazer exercício em frente à TV”, “Ao telefone”, “Queda” e “Outra”.

4.3.2 Testes

De forma a comprovar o funcionamento do módulo de fusão desenvolvido, foram elaborados vários pré-testes em laboratório fornecendo dados de entrada relativamente a atividades que seriam esperadas de forma a confirmar o resultado esperado. Depois de concretizados estes testes preliminares, foram efetuados testes em contexto real, colocando uma pessoa com o dispositivo móvel ao peito e pedindo-lhe para se dirigir à televisão, ou seja, realizar o trajeto do cenário já apresentado. A Figura 30 expõe o cenário referido com os momentos de classificação dos dispositivos enquanto, a Tabela 11 faz a tradução desses momentos para dados concretos relativamente ao que os elementos do sistema reportam.

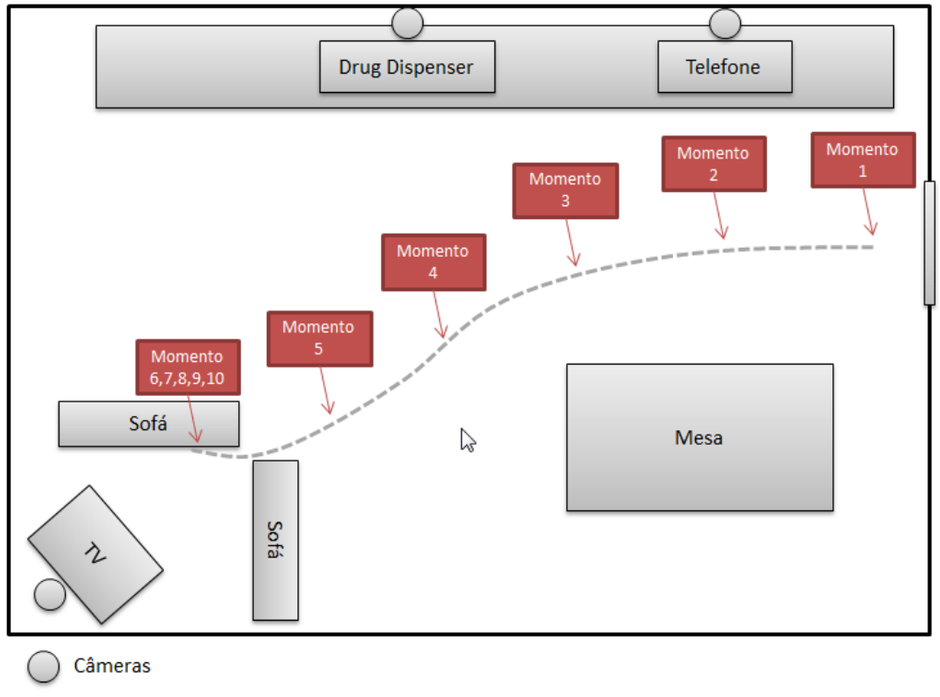

Figura 30: Esquema do cenário de teste com determinação dos momentos de classificação.

Tabela 11: Momentos de classificação do sistema PADS.

Momento	Activity Detection Module	Presence Detection Module	Agenda Module	Information Fusion Module (Output do sistema)	Ground Truth
1	Parado	Outra	Não	Outra	O utilizador está parado perto da porta da sala.
2	Andar	Outra	Não	Outra	Movimenta-se em direção ao sofá.
3	Andar	Outra	Não	Outra	Movimenta-se em direção ao sofá.
4	Andar	Outra	Não	Outra	Movimenta-se em direção ao sofá.
5	Andar	Outra	Não	Outra	Movimenta-se em direção ao sofá e senta-se no mesmo.
6	Parado	TV	Não	Em frente à TV	Sentado no sofá a ver televisão.
7	Parado	Outra	Não	Em frente à TV	Sentado no sofá a ver televisão.
8	Parado	TV	Não	Em frente à TV	Sentado no sofá a ver televisão.
9	Parado	TV	Não	Em frente à TV	Sentado no sofá a ver televisão.
10	Parado	TV	Não	Em frente à TV	Sentado no sofá a ver televisão.

Os dados apresentados em cima refletem o progresso a nível de movimentação do utente dentro da sala. Tem partes do percurso em que as câmaras não reportam a localização do utilizador, no entanto o módulo de fusão toma como resultado à falta de informação, “Outra”. Consideramos isto normal visto que as câmaras apenas reportam

algo quando encontram no seu campo de visão a cara da pessoa que esta a ser monitorizada. No momento 7 ocorreu um acontecimento fora do padrão onde a câmara que está na TV não conseguiu detetar a face do nosso utente, no entanto como na base fornecida já contávamos que isso pudesse acontecer, a classificação foi bem-sucedida. Nem sempre que a ferramenta de *sensor fusion* decide uma atividade tem 100% de certeza sobre ela, ou seja, as decisões que toma é pelas que possuem uma maior probabilidade de acontecerem, como é o caso do momento 7.

4.4 Discussão dos resultados de deteção de atividade

Nos testes elaborados para deteção das atividades que estavam a ocorrer na sala infraestruturada com dispositivos sensores, foram utilizados cenários considerados com baixo grau de complexidade.

A utilização de dados simulados para testar este tipo de sistemas ferramentas revela algumas vantagens em termos de agilização do processo. O uso deste tipo de dados obriga, naturalmente, a alguma precaução na análise dos resultados obtidos. Na verdade, verifica-se de um modo geral uma escassez e pobreza nos *datasets* utilizados nos trabalhos publicados.

A utilização de apenas uma situação de queda na base de treino utilizada poderá condicionar a sua qualidade de deteção no caso de a situação acontecer, no entanto como referido no capítulo de introdução, apenas entre 28% a 32% dos idosos experienciam uma queda por ano, o que faz com que não seja propriamente um acontecimento diária mas sim ocasional. A sua menor presença na base de treino vai ajudar também a ocultar alguns erros de deteção a nível físico quando o utente faz atividades como sentar essencialmente.

Os erros de classificação encontrados na avaliação das atividades físicas através do dispositivo móvel, de algum modo podiam ser melhorados, através da colocação de outro acelerómetro numa parte inferior do corpo, como por exemplo na coxa. Com este sensor complementar já conseguíamos identificar a posição de sentar e possivelmente aumentar a taxa de precisão da aplicação construída.

Capítulo 5: Conclusões e Perspetivas de Desenvolvimento

O trabalho apresentado nesta tese aborda a implementação de um sistema de deteção de atividade, composto por vários módulos, que pretende funcionar como um sistema de monitorização de utilizadores em casa. Apresentou-se o primeiro protótipo funcional do PADS que utiliza uma arquitetura modular e flexível, permitindo uma evolução independente dos vários componentes.

As principais contribuições deste trabalho, para além da revisão, análise e comparação de sistemas relacionados, registaram-se ao nível do desenvolvimento dos vários módulos constituintes do sistema PADS: i) o *Activity Detection Module* que é uma aplicação Android utilizada na identificação da atividade física; ii) o *Presence Detection Module* que é uma aplicação em C++ que utiliza análise de imagens para detetar a presença do utilizador; iii) o *Agenda Module* que utiliza o Google Calendar para consultar as tomas de medicação agendadas; e por fim iv) o *Information Fusion Module* que recolhe a informação dos módulos anteriores e infere qual a ação do utilizador a cada instante. O protótipo do PADS apresentado no Capítulo 3 foi desenvolvido utilizando vários algoritmos e várias ferramentas descritas no Capítulo 2. No Capítulo 4, apresenta-se ainda uma avaliação da precisão do *Activity Detection Module*; e dos resultados de classificação do *Information Fusion Module* relativamente a um cenário específico.

O componente principal deste projeto é o módulo que faz a fusão de informação, pois é neste módulo que são reunidas todas as informações obtidas pelos módulos sensores e inferido o resultado sobre a atividade que o utilizador está a realizar num determinado momento. O conjunto de treino utilizado neste módulo, durante o período de desenvolvimento, contemplava várias atividades, nomeadamente, "Em frente à TV", "A tomar medicação", "A fazer exercício em frente à TV", "Ao telefone", "Queda" e "Outra". Os testes a este módulo desenvolveram-se em torno de um cenário simples, mas que que mostraram um nível de precisão excelente, ou seja, conseguiu-se determinar a atividade do utilizador em 100% das situações. Em cenários mais complexos e com os mesmos meios sensores talvez a precisão descesse mas acreditamos que com a colocação estratégica de mais módulos (e.g., câmaras) poderíamos obter resultados igualmente bons.

No decorrer do desenvolvimento dos vários módulos do sistema, e dada a diversidade e complexidade das várias tecnologias, foram identificadas várias dificuldades normalmente relacionadas com limitações tecnológicas (e.g., integração de alguma bibliotecas de IA no Android, dificuldades de comunicação entre os módulos desenvolvidos em linguagens de programação distintas, etc.). Contudo, todas as dificuldades encontradas no desenvolvimento do PADS foram ultrapassadas com sucesso.

Como em qualquer projeto, identificamos vários aspetos que podem vir a ser melhorados e explorados no futuro como, por exemplo, permitir a utilização de outros meios de comunicação entre os vários módulos (e.g. Bluetooth), aperfeiçoar a interface gráfica por forma a melhorar a apresentação dos resultados obtidos através da fusão de informação, mas mais importante ainda será a introdução de novos módulos sensores que contribuam para melhorar a determinação da atividade em quaisquer cenários que possam surgir.

Referências

Alonso, V., Zato, C., & Pedrero, A. (2011). A New Adaptive Algorithm for Detecting Falls through Mobile Devices. *Trends in Practical*, 17–24. Retrieved from http://www.springerlink.com/index/AK107M83M215015G.pdf

Anderson, I., Maitland, J., Sherwood, S., Barkhuus, L., Chalmers, M., Hall, M., Brown, B., et al. (2007). Shakra: Tracking and Sharing Daily Activity Levels with Unaugmented Mobile Phones. *Mobile Networks and Applications*, *12*(2-3), 185–199. doi:10.1007/s11036-007-0011-7

Aviles-Arriaga, H. (2003). Visual recognition of gestures using dynamic naive bayesian classifiers. *Robot and Human Interactive Communication, 2003. Proceedings. ROMAN 2003. The 12th IEEE International Workshop on*. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1251821

Avilés-Arriaga, H. (2011). A comparison of dynamic naïve Bayesian classifiers and Hidden Markov Models for gesture recognition. *Journal of applied …*, 81–102. Retrieved from http://www.scielo.org.mx/scielo.php?pid=S1665-64232011000100007&script=sci_arttext

Bao, L., & Intille, S. S. (2004). Activity Recognition from User-Annotated Acceleration Data. *Most*, 1–17.

Dai, J., Bai, X., Yang, Z., Shen, Z., & Xuan, D. (2010). Mobile phone-based pervasive fall detection. *Personal and Ubiquitous Computing*, *14*(7), 633–643. doi:10.1007/s00779-010-0292-x

Das, S., Green, T. L., Perez, H. B., Rico, P., Murphy, M. M., Supervisor, E., & Perrig, A. (2010). Detecting User Activities using the Accelerometer on Android Smartphones.

Ghahramani, Z. (1998). Learning dynamic Bayesian networks. *Adaptive Processing of Sequences and Data …*. Retrieved from http://www.springerlink.com/index/a056513817762713.pdf

Gonçalves, P., Torres, J., Sobral, P., & Moreira, R. (2009). *Remote Patient Monitoring in Home Environments*. *Porto, Portugal*. Retrieved from http://isus.ufp.pt/wp-content/uploads/mobihealthinf2009.pdf

Ian Sommerville. (2007). *Software Engineering* (8 edition.). Harlow: Pearson Education. Retrieved from http://books.google.pt/books?id=B7idKfL0H64C

Kaehler, G., & Bradski, G. (2008). *Learning OpenCV. Farnham: O'Reilly* (First Edit., p. 555). O'Reilly Media. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Learning+OpenCV#3

Kwapisz, J. R., Weiss, G. M., & Moore, S. A. (2010). Activity Recognition using Cell Phone Accelerometers. *SIGKDD Explorations*, *12*(2), 74–82.

Luger, G. F. (2002). *Inteligência Artificial - Estruturas e estratégias para a solução de problemas complexos*. (ARTMED EDITORA S.A., Ed.) (4º Edição., p. 774). Pearson Education. Retrieved from http://books.google.pt/books?id=h2GJLP6jQ4kC&printsec=frontcover#v=onepage&q&f=false

Luštrek, M., & Kaluža, B. (2009). Fall Detection and Activity Recognition with Machine Learning. *Informatica*, *33*, 205–212.

Mccall, C., Reddy, K., & Shah, M. (2012). MACRO-CLASS SELECTION FOR HIERARCHICAL K-NN.

Moreira, R., Torres, J., Sobral, P., & Soares, C. (2010). SafeHomeHealthCare: Interference-free Home Health-Care Smart Spaces using Search Algorithms and Meta-Reality Reflection. *ISUS*. Retrieved September 7, 2012, from http://isus.ufp.pt/category/projects/project-safehomehealthcare/

Noury, N., Fleury, A., Rumeau, P., Bourke, a K., Laighin, G. O., Rialle, V., & Lundy, J. E. (2007). Fall detection--principles and methods. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference* (Vol. 2007, pp. 1663–6). doi:10.1109/IEMBS.2007.4352627

Oliver, N., Garg, A., & Horvitz, E. (2004). Layered representations for learning and inferring office activity from multiple sensory channels. *Computer Vision and Image Understanding*, *96*(2), 163–180. doi:10.1016/j.cviu.2004.02.004

OpenCV API Reference - Introduction. (2012).*OpenCV*. Retrieved January 30, 2013, from http://docs.opencv.org/modules/core/doc/intro.html

Paluszewski, M., & Hamelryck, T. (2010). Mocapy++--a toolkit for inference and learning in dynamic Bayesian networks. *BMC bioinformatics*, *11*, 126. doi:10.1186/1471-2105-11-126

Russell, S., & Norvig, P. (2003). *Artificial Intelligence : A Modern Approach* (2nd Editio., p. 1132). Pearson Education.

Ryder, J., Longstaff, B., Reddy, S., & Estrin, D. (2009). Ambulation: A Tool for Monitoring Mobility Patterns over Time Using Mobile Phones. *2009 International Conference on Computational Science and Engineering* (pp. 927–931). Ieee. doi:10.1109/CSE.2009.312

Schaul, T., & Schmidhuber, J. (2010). Metalearning. doi:doi:10.4249/scholarpedia.4650

Soares, C., Moreira, R. S., Morla, R., Sobral, P., & Torres, J. (2012). Prognostic of feature interactions between independently developed pervasive systems. *2012 IEEE Conference on Prognostics and Health Management* (pp. 1–8). Denver, Colorado: IEEE. doi:10.1109/ICPHM.2012.6299523

Torres, J. M., Sobral, P., Moreira, R. S., & Morla, R. (2012). An Adaptive Embedded System for Physical Activity Recognition. *Proc. of CISTI 2012 - 7a Conferência Ibérica de Sistemas e Tecnologias de Informação*.

Viola, P., Way, O. M., & Jones, M. J. (2004). Robust Real-Time Face Detection, *57*(2), 137–154.

Wang, M.-Y., Tsai, P. H., Liu, J. W. S., & Zao, J. K. (2009). Wedjat: A Mobile Phone Based Medicine In-take Reminder and Monitor. *2009 Ninth IEEE International Conference on Bioinformatics and BioEngineering* (pp. 423–430). Ieee. doi:10.1109/BIBE.2009.60

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques, Third Edition (The Morgan Kaufmann Series in Data Management Systems)* (3º Edition., p. 664). Morgan Kaufmann.

Anexos

Anexo 1: Modelo da base de dados

```
CREATE TABLE `history` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `app` text COLLATE utf8_bin NOT NULL,
  `action` text COLLATE utf8_bin NOT NULL,
  `activity` text COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=654 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

Esta é a tabela utilizada para armazenar de forma segura e confiável os dados reportados pelo *Activity Detection Module* e *Presence Detection Module*. Esta apenas contém 5 campos de dados em que um deles, mais precisamente o "id", é incrementado automaticamente a cada registo e é definido como o elemento identificador do mesmo. O campo "timestamp" é preenchido com a data e hora em que foi transmitida a informação pelo módulo detetor, em que o mesmo é identificado através do preenchimento do campo "app". No campo "action" são introduzidas as ações pré-definidas e identificadas pelas câmaras, enquanto no campo "activity" são inseridos os dados transmitidos pelo *smartphone* relativos às atividades que o utilizador está a realizar.

Anexo 2: Cabeçalho de um ficheiro ARFF

```
@relation nexus_instances

@attribute accelX0 numeric
@attribute accelY0 numeric
@attribute accelZ0 numeric
@attribute gyroX0 numeric
@attribute gyroY0 numeric
@attribute gyroZ0 numeric
@attribute magnX0 numeric
@attribute magnY0 numeric
@attribute magnZ0 numeric
@attribute accelX1 numeric
@attribute accelY1 numeric
@attribute accelZ1 numeric
@attribute gyroX1 numeric
@attribute gyroY1 numeric
@attribute gyroZ1 numeric
@attribute magnX1 numeric
@attribute magnY1 numeric
@attribute magnZ1 numeric
@attribute accelX2 numeric
@attribute accelY2 numeric
@attribute accelZ2 numeric
@attribute gyroX2 numeric
@attribute gyroY2 numeric
@attribute gyroZ2 numeric
@attribute magnX2 numeric
@attribute magnY2 numeric
@attribute magnZ2 numeric
@attribute accelX3 numeric
@attribute accelY3 numeric
@attribute accelZ3 numeric
@attribute gyroX3 numeric
@attribute gyroY3 numeric
@attribute gyroZ3 numeric
@attribute magnX3 numeric
@attribute magnY3 numeric
@attribute magnZ3 numeric
@attribute accelX4 numeric
@attribute accelY4 numeric
@attribute accelZ4 numeric
@attribute gyroX4 numeric
@attribute gyroY4 numeric
@attribute gyroZ4 numeric
@attribute magnX4 numeric
@attribute magnY4 numeric
@attribute magnZ4 numeric
@attribute accelX5 numeric
@attribute accelY5 numeric
@attribute accelZ5 numeric
@attribute gyroX5 numeric
@attribute gyroY5 numeric
@attribute gyroZ5 numeric
@attribute magnX5 numeric
@attribute magnY5 numeric
@attribute magnZ5 numeric
@attribute accelX6 numeric
@attribute accelY6 numeric
@attribute accelZ6 numeric
@attribute gyroX6 numeric
@attribute gyroY6 numeric
@attribute gyroZ6 numeric
@attribute magnX6 numeric
@attribute magnY6 numeric
@attribute magnZ6 numeric
@attribute accelX7 numeric
@attribute accelY7 numeric
@attribute accelZ7 numeric
@attribute gyroX7 numeric
@attribute gyroY7 numeric
@attribute gyroZ7 numeric
@attribute magnX7 numeric
@attribute magnY7 numeric
@attribute magnZ7 numeric
@attribute accelX8 numeric
@attribute accelY8 numeric
@attribute accelZ8 numeric
@attribute gyroX8 numeric
@attribute gyroY8 numeric
@attribute gyroZ8 numeric
@attribute magnX8 numeric
@attribute magnY8 numeric
@attribute magnZ8 numeric
@attribute accelX9 numeric
@attribute accelY9 numeric
@attribute accelZ9 numeric
@attribute gyroX9 numeric
@attribute gyroY9 numeric
@attribute gyroZ9 numeric
@attribute magnX9 numeric
@attribute magnY9 numeric
@attribute magnZ9 numeric
@attribute mediaaccelX
numeric
@attribute mediaaccelY
numeric
@attribute mediaaccelZ
numeric
@attribute varaccelX
numeric
@attribute varaccelY
numeric
@attribute vargyroY
numeric
@attribute vargyroZ
numeric
@attribute mediamagnX
numeric
@attribute mediamagnY
numeric
@attribute mediamagnZ
numeric
@attribute varmagnX
numeric
@attribute varmagnY
numeric
@attribute varmagnZ
numeric
@attribute class
{1,2,3,4,5}

@data
```

Anexo 3: Information Fusion Module – Parameter learning

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include "mocapy.h"
using namespace mocapy;
using namespace std;


#define ALL_INDEXES -1 /*wildcard for all indexes in that dimension*/
#define NR_SLICES 10    /* N_SLICES==s.slice_count */
#define MCMC_BURN_IN 100 /* used in Gibbs sampling parameters*/

int main(void) {

      // The dynamic Bayesian network
      DBN dbn;

      // Nodes in slice 1
      Node* h1 = NodeFactory::new_discrete_node(6, "h1"); // Hidden Node
      Node* o1 = NodeFactory::new_discrete_node(5, "o1"); // Act. Fisica ( Android )
      Node* o2 = NodeFactory::new_discrete_node(2, "o2"); // Agenda ( Drug Dispenser )
      Node* o3 = NodeFactory::new_discrete_node(4, "o3"); // Act. Funcional (Câmaras )

      // Nodes in slice 2
      Node* h2 = NodeFactory::new_discrete_node(6, "h2"); //Hidden Node

      // Set architecture
      dbn.set_slices(vec(h1, o1, o2, o3), vec(h2, o1, o2, o3));

      // Definir conexoes
      dbn.add_intra("h1", "o1");
      dbn.add_intra("h1", "o2");
      dbn.add_intra("h1", "o3");
      dbn.add_inter("h1", "h2");

      // COnstrução da rede
      dbn.construct();

      cout << "Loading traindata" << endl;
      GibbsRandom mcmc = GibbsRandom(&dbn);

      EMEngine em = EMEngine(&dbn, &mcmc);
      em.load_mismask("data/mismask.dat"); // carregar ficheiro com informação sobre os
observed nodes
      em.load_sequences("data/traindata.dat"); // carregar ficheiro com informação
sobre os sensores

      // treinar a rede com os dados carregados
      cout << "Starting EM loop" << endl;
      for (uint i=0; i<100; i++) {
            em.do_E_step(20,10);
```

Anexo 3: Information Fusion Module – Parameter Learning

```
            double ll = em.get_loglik();
            cout << "LL= " << ll << endl;
            em.do_M_step();
      }

      // guardar a bayesian network
      dbn.save("dbn.mocapy");
}
```

Anexo 4: Information Fusion Module – Inference

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include "mocapy.h"
using namespace mocapy;
using namespace std;


#define ALL_INDEXES -1 /*wildcard for all indexes in that dimension*/
#define NR_SLICES 10    /* N_SLICES==s.slice_count */
#define MCMC_BURN_IN 100 /* used in Gibbs sampling parameters*/

int main(void) {

        //mocapy_seed((uint)99994);
        //mocapy_seed((uint)5556574);
        // The dynamic Bayesian network
        DBN dbn;

        // Nodes in slice 1
        Node* h1 = NodeFactory::new_discrete_node(6, "h1"); // Hidden Node
        Node* o1 = NodeFactory::new_discrete_node(5, "o1"); // Act. Fisica ( Android )
        Node* o2 = NodeFactory::new_discrete_node(2, "o2"); // Agenda (Drug Dispenser)
        Node* o3 = NodeFactory::new_discrete_node(4, "o3"); // Act. Funcional (Cameras)

        // Nodes in slice 2
        Node* h2 = NodeFactory::new_discrete_node(6, "h2"); //Hidden Node
        // carregar a rede
        dbn.load("dbn.mocapy");
        // carregar dados da base de dados
        system("php5 mocapyfile.php");
        // obter os dados
        ifstream infile(File.txt");

        MDArray<double> seq_data = MDArray<double>(vec(NR_SLICES, 4));
        int countslices = 0;
        string line;
        while (getline(infile, line))
        {
                istringstream iss(line);
                int sensorfusion, android, agenda, cameras=0;
                if (!(iss >> sensorfusion >> android >> agenda >> cameras)) { break; }
// error
               cout << android << agenda << cameras << endl;
                seq_data.set(countslices,1,android); // Set index (i,1)
                seq_data.set(countslices,2,agenda); // Set index (i,2)
                seq_data.set(countslices,3,cameras); // Set index (i,3)

                countslices++;
        }
```

Anexo 4: Information Fusion Module – Inference

```
MDArray<eMISMASK> mismask = MDArray<eMISMASK>(vec(NR_SLICES,4)); // sequence mismask
        mismask.set_wildcard(vec(ALL_INDEXES,0), MOCAPY_HIDDEN); // all h1 nodes are
hidden
        mismask.set_wildcard(vec(ALL_INDEXES,1), MOCAPY_OBSERVED); // all o1 nodes are
observed
        mismask.set_wildcard(vec(ALL_INDEXES,2), MOCAPY_OBSERVED); // all o2 nodes are
observed
        mismask.set_wildcard(vec(ALL_INDEXES,3), MOCAPY_OBSERVED); // all o3 nodes are
observed

        GibbsRandom mcmc2 = GibbsRandom(&dbn);
        InfEngineMCMC inf = InfEngineMCMC(&dbn, &mcmc2, &seq_data, mismask);
        inf.initialize_sample_generator(MCMC_BURN_IN,  true);

        // quantas mais iteracoes mais preciso fica o calculo da probabilidade mas mais
lento fica o sistema
        uint N_ITER=100;
        int h_node_posterior[NR_SLICES][6]={0,0,0,0,0,0}; // counting for h==0 for each
slice
        for (uint i=0; i<N_ITER; i++) {
                Sample s = inf.sample_next();
                cout << "Viterbi LL= " << s.ll << endl;
                cout << "Viterbi seq= " << s.seq << endl;
                for (uint slice=0; slice<s.slice_count; slice++) {      // Contagem dos
hidden nodes
                        if (s.seq.get(slice,0)==0) h_node_posterior[slice][0]++;
                        if (s.seq.get(slice,0)==1) h_node_posterior[slice][1]++;
                        if (s.seq.get(slice,0)==2) h_node_posterior[slice][2]++;
                        if (s.seq.get(slice,0)==3) h_node_posterior[slice][3]++;
                        if (s.seq.get(slice,0)==4) h_node_posterior[slice][4]++;
                        if (s.seq.get(slice,0)==5) h_node_posterior[slice][5]++;
                }
                cout << "Viterbi slice_count= " << s.slice_count << endl;
        }

        FILE *fp = fopen("/var/www/pads/result.pads","w");

        // Imprimir a probabilidade do resultado de h1 para cada slice
        for (uint slice=0; slice<NR_SLICES; slice++) {

        cout << "P(H_slice" << slice << ")=<" <<
(h_node_posterior[slice][0]/(double)N_ITER) << ","<<
(h_node_posterior[slice][1]/(double)N_ITER) << ","<<
(h_node_posterior[slice][2]/(double)N_ITER) << "," <<
(h_node_posterior[slice][3]/(double)N_ITER) << "," <<
(h_node_posterior[slice][4]/(double)N_ITER) << ">";

        // Determinar a atividade para cada slice

if(((h_node_posterior[slice][0])>(h_node_posterior[slice][1]))&&((h_node_posterior[slic
e][0])>(h_node_posterior[slice][2]))&&((h_node_posterior[slice][0])>(h_node_posterior[s
lice][3]))&&((h_node_posterior[slice][0])>(h_node_posterior[slice][4]))&&((h_node_poste
```

Anexo 4: Information Fusion Module – Inference

```
rior[slice][0])>(h_node_posterior[slice][5]))){
                cout << "Ver TV" << endl;
                if(slice==9){
                fprintf (fp, "Ver TV\n");}
        }else
if(((h_node_posterior[slice][1])>(h_node_posterior[slice][0]))&&((h_node_posterior[slic
e][1])>(h_node_posterior[slice][2]))&&((h_node_posterior[slice][1])>(h_node_posterior[s
lice][3]))&&((h_node_posterior[slice][1])>(h_node_posterior[slice][4]))&&((h_node_poste
rior[slice][1])>(h_node_posterior[slice][5]))){
                cout << "Tomar Medicação" << endl;
                if(slice==9){
                fprintf (fp, "Tomar medicação\n");}
        }else
if(((h_node_posterior[slice][2])>(h_node_posterior[slice][0]))&&((h_node_posterior[slic
e][2])>(h_node_posterior[slice][1]))&&((h_node_posterior[slice][2])>(h_node_posterior[s
lice][3]))&&((h_node_posterior[slice][2])>(h_node_posterior[slice][4]))&&((h_node_poste
rior[slice][2])>(h_node_posterior[slice][5]))){
                cout << "Exercicio em frente à TV" << endl;
                if(slice==9){
                fprintf (fp, "Exercicio em frente à TV\n");}
        }else
if(((h_node_posterior[slice][3])>(h_node_posterior[slice][0]))&&((h_node_posterior[slic
e][3])>(h_node_posterior[slice][1]))&&((h_node_posterior[slice][3])>(h_node_posterior[s
lice][2]))&&((h_node_posterior[slice][3])>(h_node_posterior[slice][4]))&&((h_node_poste
rior[slice][3])>(h_node_posterior[slice][5]))){
                cout << "Ao Telefone" << endl;
                if(slice==9){
                fprintf (fp, "Ao telefone\n");}
        }else
if(((h_node_posterior[slice][4])>(h_node_posterior[slice][0]))&&((h_node_posterior[slic
e][4])>(h_node_posterior[slice][1]))&&((h_node_posterior[slice][4])>(h_node_posterior[s
lice][2]))&&((h_node_posterior[slice][4])>(h_node_posterior[slice][3]))&&((h_node_poste
rior[slice][4])>(h_node_posterior[slice][5]))){
                cout << "Outra" << endl;
                if(slice==9){
                fprintf (fp, "Outra\n");}
        }else
if(((h_node_posterior[slice][5])>(h_node_posterior[slice][0]))&&((h_node_posterior[slic
e][5])>(h_node_posterior[slice][1]))&&((h_node_posterior[slice][5])>(h_node_posterior[s
lice][2]))&&((h_node_posterior[slice][5])>(h_node_posterior[slice][3]))&&((h_node_poste
rior[slice][5])>(h_node_posterior[slice][4]))){
                cout << "Queda" << endl;
                if(slice==9){
                fprintf (fp, "Queda\n");}
        }else{
                cout << "Indefinivel" << endl;
                if(slice==9){
                fprintf (fp, "Indefinivel\n");}
                }
        }
return EXIT_SUCCESS;
}
```