Deteção e acompanhamento de movimento através de uma câmara de vídeo

Tese Mestrado

Para obter o grau de mestre pelo
Instituto Superior de Engenharia do Porto,
Defendido publicamente em Novembro por

Pedro Tiago da Cruz Peixoto

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

2012

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Pedro Tiago da Cruz Peixoto, nº 1050797, 1050797@isep.ipp.pt

Orientação: Prof. Dr. Lino Manuel Baptista Figueiredo, LBF@isep.ipp.pt
Co-orientação: Engenheiro António José Matos Meireles, ajmm@isep.ipp.pt

Esta página foi intencionalmente deixada em branco.

Ao meu afilhado

Esta página foi intencionalmente deixada em branco.

Agradecimentos

Muitas pessoas contribuíram direta ou indiretamente para o concretizar deste trabalho, e mesmo correndo o risco de esquecer alguém, não posso deixar de lhes expressar aqui os meus agradecimentos.

Em relação aos agradecimentos pessoais, não poderia deixar de começar pelos meus orientadores Lino Figueiredo e António Meireles, que sempre me demonstraram confiança no meu trabalho e me apoiaram na persecução dos objetivos que me propus atingir.

Ao meu colega de trabalho Carlos Paiva, e aos meus amigos pela paciência.

Por último, com um carinho muito especial, aos meus pais, pela confiança e encorajamento.

A todos o meu muito obrigado.

Esta página foi intencionalmente deixada em branco.

Resumo

A deteção e seguimento de pessoas tem uma grande variedade de aplicações em visão computacional. Embora tenha sido alvo de anos de investigação, continua a ser um tópico em aberto, e ainda hoje, um grande desafio a obtenção de uma abordagem que inclua simultaneamente flexibilidade e precisão. O trabalho apresentado nesta dissertação desenvolve um caso de estudo sobre deteção e seguimento automático de faces humanas, em ambiente de sala de reuniões, concretizado num sistema flexível de baixo custo. O sistema proposto é baseado no sistema operativo *GNU's Not Unix* (GNU) linux, e é dividido em quatro etapas, a aquisição de vídeo, a deteção da face, o *tracking* e reorientação da posição da câmara. A aquisição consiste na captura de *frames* de vídeo das três câmaras *Internet Protocol* (IP) Sony SNC-RZ25P, instaladas na sala, através de uma rede *Local Area Network* (LAN) também ele já existente. Esta etapa fornece os *frames* de vídeo para processamento à deteção e *tracking*.

A deteção usa o algoritmo proposto por Viola e Jones, para a identificação de objetos, baseado-se nas suas principais características, que permite efetuar a deteção de qualquer tipo de objeto (neste caso faces humanas) de uma forma genérica e em tempo real. As saídas da deteção, quando é identificado com sucesso uma face, são as coordenadas do posicionamento da face, no *frame* de vídeo. As coordenadas da face detetada são usadas pelo algoritmo de *tracking*, para a partir desse ponto seguir a face pelos *frames* de vídeo subsequentes.

A etapa de *tracking* implementa o algoritmo *Continuously Adaptive Mean-SHIFT* (Camshift) que baseia o seu funcionamento na pesquisa num mapa de densidade de probabilidade, do seu valor máximo, através de iterações sucessivas. O retorno do algoritmo são as coordenadas da posição e orientação da face. Estas coordenadas permitem orientar o posicionamento da câmara de forma que a face esteja sempre o mais próximo possível do centro do campo de visão da câmara.

Os resultados obtidos mostraram que o sistema de *tracking* proposto é capaz de reconhecer e seguir faces em movimento em sequências de *frames* de vídeo, mostrando adequabilidade para aplicação de monotorização em tempo real.

Palavras-chave: *Tracking*, Camshift, "classificadores *haar*", vídeo digital, Raspberry pi.

Esta página foi intencionalmente deixada em branco.

Abstract

The detection and tracking of people have a wide variety of applications in computer vision. Although it has been the target of research for several years, it remains a open topic. It still remains a great challenge to get an approach that includes both flexibility and accuracy, specially when it comes to an open environment.

The work presented in this thesis, develops a case study on detection and automatic tracking of human faces in a meeting room environment, implemented in a flexible low cost system. The proposed system is based on GNU Linux operating system, and divided in four stages: the video acquisition, face detection, face tracking and reorientation of the camera position. Video acquisition consists in capturing frames from an existing video IP cameras Sony SNC-RZ25P, installed in the meeting room via a LAN. This step provides the frames to the detection processing and when finished, to the tracking processing step.

The detection uses the algorithm proposed by Viola and Jones, to identify objects based on their main features, and allows performing the detection of any type of object, in this case an human face, generally in real time. The output of this detection, if successfully identified one or more faces, are the position coordinates of the face in the video frame, which is then used by the tracking algorithm, to follow the face, from this point, in subsequent frames.

In the tracking, the CamShift algorithm that bases its operation on search, in a probability density map representing the human face, and compute its maximum value through successive iterations. The return of the algorithm, are the position coordinates, and orientation of the human face, which allows positioning the camera so that the face is always nearest the center of the camera view field.

The results shown that the proposed tracking system is able to recognize and track faces in motion in video frames sequences, showing its ability for real time aplications.

Keywords: Tracking, CamShift, "haar classifiers", Digital video, Raspberry Pi.

Esta página foi intencionalmente deixada em branco.

Conteúdo

Esta página foi intencionalmente deixada em branco.

Lista de Figuras

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Acrónimos

2D	Duas Dimensões
3D	Três Dimensões
AmI	Ambientes Inteligentes
ARM	*Advanced RISC Machine*
AVI	*Audio Video Interleave*
BB	*Bounding Box*
BN	*Bayesian Networks*
Camshift	*Continuously Adaptive Mean-SHIFT*
CAN	*Controller Area Network*
CB	*Codebook*
CBIR	*Content-Based Image Retrieval*
CDB	*Console Debugger*
CGI	*Common Gateway Interface*
CJLF	*Constrained joint likelihood filter*
CPU	*Central Processing Unit*
CSI-2	*Camera Serial Interface 2*
DAGs	*directed acyclic graphs*
DBN	*Dynamic Bayesian networks*
DCT	*Discrete cosine transform*
DSI	*Display Serial Interface*
FCM	*Fuzzy C-Means*
FIR	*Far-Infrared*
FPS	*Frames* por segundo
GCC	*GNU Compiler Collection*
GDB	*GNU Project Debugger*
GECAD	Grupo de Investigação em Engenharia do Conhecimento e Apoio à Decisão
GLCMs	*Gray-Level Co-occurrence Matrices*

GNOME *GNU Network Object Model Environment*

GNU *GNU's Not Unix*

GPIOs *General Purpose Input/Output*

GPL *General Public License 3*

GPU *Graphics Processing Unit*

GT *Ground-Truth*

GUI *Graphical user interface*

HD *Hard Disk*

HMM *Hidden Markov model*

HSV *Hue, Saturation e Value*

I^2C *Inter-Integrated Circuit*

I^2S *Integrated Interchip Sound*

IDE *Integrated Development Environment*

IEC *International Electrotechnical Commission*

IP *Internet Protocol*

IPP Instituto Politécnico do Porto

ISEP Instituto Superior de Engenharia do Porto

ISO *International Organization for Standardization*

JLF *Joint Likelihood Filter*

JPDAF *Joint Probabilistic Data Association Filter*

JPEG *Joint Photographic Experts Group*

JTAG *Joint Test Action Group*

KDE *K Desktop Environment*

KF *Kalman Filter*

KFM *Kalman filter model*

LAID *Laboratory of Ambient Intelligence for Decision Support*

LAN *Local Area Network*

LCD *Liquid Crystal Display*

LF *Likelihood filter*

LK Lucas-Kanade

LPDDR *low-power DDR*

LXDE *Lightweight X11 Desktops Environment*

MSVC++ *Microsoft Visual C++*

MIPI *Mobile Industry Processor Interface*

MIR *Mid-wavelength Infrared*

MJPEG *Motion JPEG*

MNRL *Maximum Negative Run-Length*

MOG *Mixture of Gaussians*

MPEG *Moving Pictures Experts Group*

MPEG-4 *Moving Picture Experts Group* - 4

NIR *Near-Infrared*

NGT *Nonground-Truth*

OpenCV *Open Source Computer Vision Library*

PD *Partition Distance*

PDAF *Probabilistic data association filter*

PDF *probability distribution image*

PDI Processamento Digital Imagem

PF *Particle filter*

PISC *Peripheral increment sign correlation*

PGM *Probabilistic Graphical Model*

PTZ *Pan-Tilt-Zoom*

RAM Random-Access Memory

RGB *Red,Green,Blue*

Rpi Raspberry Pi

RS *Reference Segmentation*

SD *Secure Digital*

SIFT *Scale Invariant Feature Transform*

SMR *Smart Meeting Rooms*

SO Sistema Operativo

SOC *System on a chip*

SPI *Serial Peripheral Interface*

SVM *Support Vector Machine*

TCP *Transmission Control Protocol*

TTL *Transistor-Transistor Logic*

UART *Universal Asynchronous Receiver/Transmitter*

UGs *Undirected Graphs*

UDP *User Datagram Protocol*

URL *Uniform Resource Locator*

USB *Universal Serial Bus*

XFCE *Xforms Cool Environment*

XML *Extensible Markup Language*

1 Introdução

Os Ambientes Inteligentes (AmI) são um paradigma que suporta o projeto da nova geração de sistemas e introduz um novo significado à computação entre homem máquina e o seu ambiente [RF05]. Os Ambientes Inteligentes lidam com um novo conceito onde os dispositivos computacionais permitem ao ser humano interagir com os ambientes físicos de um modo inteligente e não obstrutivo. Os AmI podem ser muito diversos, tais como: casas, escritórios, salas de reuniões, escolas, hospitais, centros de controlo, transportes, atracões turísticas, lojas, instalações desportivas. Um Ambiente é denominado como "Ambiente Inteligente"quando este não interfere com normal desenrolar dos eventos. Onde as diversas se tecnologias complementam umas às outras, envolvendo o utilizador com o ambiente, oferecendo serviços e características que são requeridas pelos utilizadores na percussão dos seus objetivos.

Os AmI são uma área multidisciplinar que inclui disciplinas como: inteligência distribuída, desenho de *software*, visão computacional, reconhecimento de voz, robótica, desenho de *hardware*, ética e direito [RF05]. Tais ambientes devem ser sensíveis às necessidades do ser humano, personalizando requisitos e prevendo comportamentos.

As *Smart Meeting Rooms* (SMR) são uma área de especialização dos AmI, aplicada a salas de reuniões. Uma SMR deverá fornecer de uma forma eficiente uma plataforma de interação entre os seus ocupantes para facilitar o alcance dos seus objetivos [FMF+12].

Neste contexto a visão computacional, pode ser usada para dotar um sistema automatizado com uma ferramenta de visão artificial, para que este possa tomar decisões consoante os acontecimentos que deteta. Isto é possível devido ao constante desenvolvimento tecnológico que permite a utilização de técnicas cada vez mais complexas e exigentes em capacidade de processamento. Um exemplo deste tipo de visão são algoritmos que de

forma automática, fazem identificação, reconhecimento e seguimento de objetos em movimento.

A deteção de objetos em vídeos consiste na monitorização das alterações sofridas por um determinado objeto durante uma sequência de vídeo, podendo incluir a sua presença, posição, tamanho, forma, etc.. A deteção e seguimento de objetos tem uma grande variedade de aplicações em visão computacional tais como: a compressão de vídeo, vídeo vigilância, controlo baseado em visão, interfaces homem − computador, imagiologia médica, auxílio na georreferenciação e robótica. Desempenha também um papel importante em bases de dados de vídeo, para por exemplo, a indexação e extração baseada no conteúdo.

Embora tenha sido alvo de anos de investigação, a deteção e seguimento de objetos continua a ser um tópico em aberto. Continua a ser um grande desafio a obtenção de uma abordagem que inclua simultaneamente flexibilidade e precisão. Um desafio importante surge da variabilidade do objeto na sequência de imagens. À medida que um objeto se move através do campo visual de uma câmara, o aspeto do objeto pode sofrer alterações significativas. Alterações essas provenientes essencialmente de três fontes principais: variação na postura ou deformações do alvo, variações na iluminação, e oclusão parcial ou total do alvo.

Um caso particular da deteção e seguimento de objetos, pode ser os sistemas em que objeto alvo é a face de uma pessoa. A tarefa de detetar seres humanos em imagens, a partir das suas faces, é bastante estudada na literatura. Contudo ainda é um campo que apresenta desafios relacionados, com detalhes que variam, por exemplo, com a postura e a iluminação, que tornam complexa a sua correta deteção, reconhecimento e seguimento. Existem algoritmos que detetam faces humanas de forma bastante eficiente, apesar da imposição de algumas limitações, decorrentes, por exemplo, da aquisição de imagens em que a face alvo se encontra de perfil ou parcialmente oculta.

1.1 Motivação

A visão computacional tem recebido grande atenção ao longo das duas últimas décadas. Atualmente os processadores são mais rápidos e menores, as câmaras digitais permitem uma melhor qualidade de imagem, os sensores são mais precisos. A evolução nos campos citados levou ao crescente interesse na área da visão computacional. Devido também a necessidade de automatização de processos, a visão computacional tem contribuído com técnicas que permitem aos sistemas de controlo de processos ter uma perceção visual. Estes controlos podem assim obter informação visual do meio em que se inserem, permitindo a atuação a partir do conteúdo da imagem.

Uma das áreas mais importantes e estudadas da visão por computador é a deteção e seguimento (*tracking*) de objetos. Contudo, ainda não foi possível encontrar uma solução ótima para alguns dos problemas associados ao campo. A grande aplicabilidade destas técnicas complica o desenvolvimento de uma solução única para todas as implementações.

Aspectos que variam entre aplicações, por exemplo, câmaras fixas ou orientáveis, diferentes caraterísticas que definem os objetos e a necessidade de execução em tempo real. Todas estas situações dão origem às complexas adaptações que um algoritmo tem que executar para ser aplicado. Devido a este facto, os investigadores como [PM11] [BHJS08] [RW98] passaram a incluir restrições às suas abordagens, o que limita a amplitude de aplicabilidade mas otimiza o desempenho para essa área. Na deteção de objetos, se o tipo de objeto a identificar e seguir for conhecido. Então é possível otimizar o algoritmo de forma a lidar com este tipo de características, em detrimento de outras caraterísticas mais genéricas. Este tipo de premissa pode ser a diferença entre o sucesso ou insucesso de um algoritmo.

1.2 Objetivos

Este trabalho tem como principal objetivo desenvolver um módulo capaz de detetar movimentos através da análise de imagens de vídeo provenientes das câmaras instaladas na sala de reuniões. O sistema deverá ser capaz gerar comandos para movimentar a câmara de vídeo para a posição onde se encontra o orador.

Foram propostos vários objetivos mais específicos que melhor especificam o objetivo principal:

- Estudo da câmara de vídeo existente na sala de reuniões do Grupo de Investigação em Engenharia do Conhecimento e Apoio à Decisão (GECAD), comandos para controlo de posição e aquisição de vídeo;
- Estudo de técnicas atualmente utilizadas para deteção e seguimento de pessoas;
- Desenvolvimento de *software* que permita receber e enviar comandos pela rede implementada na sala de reuniões.
- Desenvolvimento de *software* que permita detetar e seguir faces em movimento.
- Integração do módulo desenvolvido no sistema de controlo da sala de reuniões do GECAD.

1.3 Calendarização

Por forma atingir os objetivos propostos, são definidas etapas de cumprimento do trabalho de forma a permitir que seja executável no prazo final previsto. Na figura 1.1 são apresentadas as estimativas de tempo de trabalho para cada uma das etapas identificadas e ordenadas de forma a seguir uma lógica sequencial.

ID	Nome das etapas	Início	Fim	Duração
1	Pesquisa do estado da arte	31-10-2011	24-01-2012	62d
2	Escrita do estado da arte	10-01-2012	20-03-2012	51d
3	Escolha de algoritmos e ferramentas a usar	21-03-2012	13-04-2012	18d
4	Arquitectura do sistema a desenvolver	13-04-2012	20-04-2012	6d
5	Aquisição de vídeo pesquisa e implementação	20-04-2012	18-05-2012	21d
6	Escolha e implementação do algoritmo de deteção	15-05-2012	30-05-2012	12d
7	Teste algoritmo de deteção	25-05-2012	01-06-2012	6d
8	Escrita do capitulo Arquitetura do sistema	20-04-2012	05-06-2012	33d
9	Escolha e implementação do algoritmo de seguimento	06-06-2012	29-06-2012	18d
10	Teste e ajustes ao algoritmo final	25-06-2012	13-07-2012	15d
11	Elaboração dos restantes capitulos da tese	02-07-2012	05-11-2012	91d

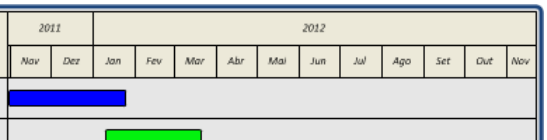

Figura 1.1: *Calendarização do projeto*

1.4 Estrutura do documento

Este documento está organizado em seis capítulos. Após o presente capítulo introdutório, em que se carateriza o problema a tratar, a sua motivação e contexto, bem como os objetivos propostos. É seguido pelo capítulo 2, onde são apresentados os fundamentos teóricos das técnicas estudadas, no processamento de vídeo digital e algoritmos de *tracking*. É também apresentado o *hardware* utilizado para aquisição de imagem e unidade de processamento. O capitulo termina com a apresentação do estado da arte para o tema em estudo. No terceiro capítulo é apresentada a arquitetura do sistema proposto, com uma descrição do *layout* onde está inserido. No quarto capítulo é descrito em pormenor a implementação efetuada, com descrição das funções implementadas a sua sequência de execução. O capítulo 5, resultados e testes, apresenta a aplicação implementada, e os resultados de testes efetuados em ambiente real de aplicação. O presente documento termina com o capítulo "Conclusões e perspetivas futuras", que inclui as principais conclusões e melhoramentos possíveis para o algoritmo desenvolvido.

2

Estado da Arte e Enquadramento teórico

Neste capítulo é feito um enquadramento teórico do problema proposto, serão apresentados os fundamentos dos algoritmos e ferramentas utilizados, desde o processamento de vídeo digital, aos algoritmos de *tracking*. É também feita uma análise aos algoritmos propostos na bibliografia, por diversos autores, relacionados com as técnicas utilizadas na deteção e *tracking* de objetos.

2.1 Vídeo digital

O vídeo digital é formado por uma sequência de imagens, que por sua vez são formadas por pontos (*pixels*). Pontos pertencentes à mesma imagem e espacialmente vizinhos, são chamados de pontos vizinhos. Imagens também chamadas de quadros[1], pertencentes à mesma cena e temporalmente próximas são chamadas de imagens vizinhas. Pontos e imagens vizinhas são geralmente muito similares e esta é uma característica importante do vídeo digital. Os *frames*, que compõem o vídeo digital, são divididos em blocos. Estes blocos podem ter tamanhos variados, mas comumente são utilizados tamanhos de 4x4, 8x8 ou 16x16 *pixels*.

2.1.1 Amostragem

Quando na presença de um sinal de vídeo analógico, que é contínuo no tempo e no espaço, a sua representação digital envolve amostragem espacial e temporal. Assim uma imagem digital de uma cena é resultado da transformação de um sinal bidimensional de

[1] O termo vulgarmente usado em processamento de vídeo, para referir quadros, pela comunidade cientifica é *frame*, que será usado neste documento

parâmetros contínuos, para uma versão de parâmetros discretos e de amplitudes, manipuláveis por sistemas digitais [ALL00] [Dub09].

Mediante amostragem espacial, a cena bidimensional composta por infinitos pontos passa a ser representada por uma matriz retangular finita de pontos. O processo que limita o domínio dos valores de intensidade para cada um dos pontos amostrados é chamado de quantização [DSN02]. Para a adequada aquisição de uma sequência de vídeo, ainda é necessária a realização de amostragem no domínio do tempo que proporcione a mesma sensação de continuidade temporal existente no mundo real. Este aspeto introduz um parâmetro importante do vídeo digital, conhecido como taxa de aquisição (*frame rate*). Quanto mais alta é a frequência em que os *frames* são adquiridos mais suave será a sensação de movimento, no entanto, uma quantidade maior de amostras tem ser adquirida e armazenada, aumentando a complexidade de tratamento e armazenamento desses vídeos.

A figura 2.1 ilustra uma sequência de *frames* de um vídeo digital, salientando a divisão dos blocos do *frame*, a identificação do *pixel*, e os conceitos de amostragem temporal e espacial, para a amostragem de um vídeo.

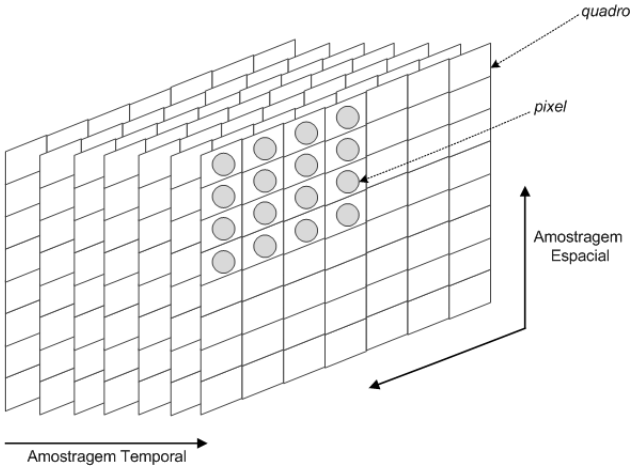

Figura 2.1: *Representação de uma sequência de frames de vídeo digital, conceito de amostragem espacial e temporal [Esp09]*

2.1.2 Codificação de vídeo digital

Como descrito na secção anterior, um vídeo pode ser considerado como uma sequência de imagens, ou *frames*. A compressão de vídeo utiliza técnicas de codificação para reduzir a redundância espacial e temporal existente nos e entre *frames*.

Para a abordagem à redundância espacial, cada um desses *frames* pode ser codificado através da utilização de várias técnicas. Em particular, destaca-se a utilização da codificação *Joint Photographic Experts Group* (JPEG), procedimento esse que constitui a base da codificação de vídeo chamada MJPEG. Ao aplicar essa codificação, considera-se apenas a redundância de informação contida num *frame* (redundância intra-*frames*),

quando a maior redundância pode estar nas informações contidas em *frames* consecutivos (redundância inter-*frames*).

Considerar a compressão temporal, é um aspeto importante em compressão de vídeo e está presente em vários padrões. A compressão temporal é aplicada sobre *frames* consecutivos de vídeo, e visa identificar partes semelhantes e codificar apenas as partes necessárias de cada *frame*.

Nesta secção é apresentada uma análise a dois padrões de codificação de vídeo, o MJPEG e o *Moving Picture Experts Group* - 4 (MPEG-4), são descriminadas as técnicas utilizadas por cada um, quais as suas vantagens e desvantagens, e quais as suas principais áreas de aplicação.

MPEG-4

O MPEG-4 é um padrão ISO/IEC desenvolvido pelo grupo de trabalho MPEG, que também desenvolveu os padrões conhecidos como MPEG-1 e MPEG-2, para compressão de vídeo.

Historicamente o MPEG-4 teve como objetivo inicial a codificação do conteúdo audiovisual para transmissão a taxas baixas. Os estudos para o seu desenvolvimento foram iniciados em 1993, no entanto, em 1994 e até 1995, os seus objetivos sofreram alterações, focando-se numa convergência entre aplicações multimídia, baseadas em TV, filmes, entretenimento, computadores e telecomunicações. Somente em outubro de 1998 foi apresentada a primeira versão desse padrão, e a segunda versão em dezembro de 1999 [Wat04].

A família de padrões MPEG utiliza compressão espacial e temporal, para tal implementa um conjunto de técnicas, as quais podem ser divididas em compressão com perdas e sem perdas. Na figura 2.2 é apresentado o principio básico de funcionamento dos padrões MPEG, identificando qual a sua sequência e quais as etapas em que existem perdas.

A compressão com perdas pode ser dividida em três etapas (figura 2.2 Bloco A): Estimativa de movimento, Codificação por *Discrete cosine transform* (DCT) e quantização. A estimativa de movimento é responsável por minimizar as redundâncias temporais dos *frames* de vídeo. Enquanto a codificação por DCT e quantização eliminam as redundâncias espaciais.

Para efetuar a redução de informação temporal os padrões da família MPEG utilizam um mecanismo de diferenciação na codificação dos *frames* da sequência de vídeo, de forma a otimizar a compressão. O padrão MPEG especifica três tipos de *frames* comprimidos no arquivo de saída. Os *frames* I (*Intra coded frame*) onde são aplicados apenas algoritmos de redução de redundância espacial. Os *frames frames* P (*Predictive coded picture*) e B (*Bidirectionally Predictive coded picture*) onde são também aplicados algoritmos de supressão de redundância temporal. No caso dos *frames* B a predição de movimento é bidireccional, ou seja, é feita com *frames* no passado e no futuro em relação ao *frame* em codificação. Os *frames* apresentam diferentes taxas de compressão, sendo que os *frames* B apresentam a maior taxa, seguidos dos P e dos I.

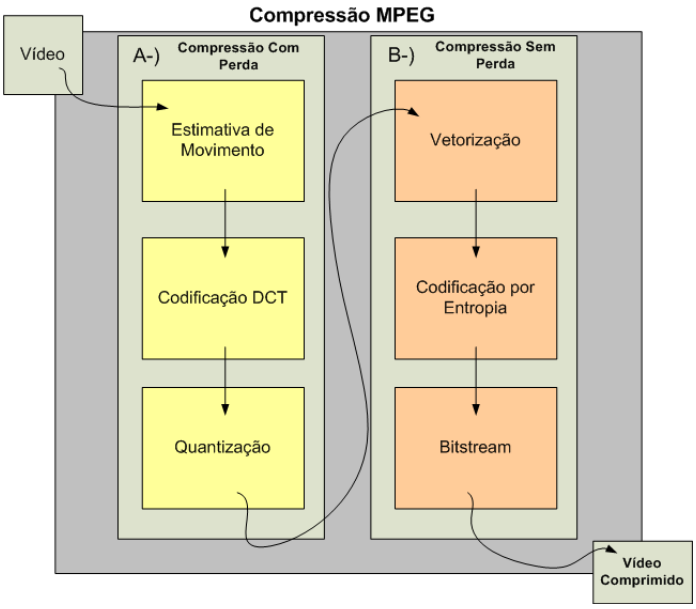

Figura 2.2: *Etapas de codificação de vídeo MPEG*

A compressão apresentada pelo padrão MPEG baseia-se em comparar imagens comprimidas, usando estes diferentes tipos de *frames* entrelaçados entre eles. A figura 2.3 ilustra a disposição dos tipos de *frames* num vídeo típico MPEG. Onde os *frames* B fazem referência a *frames* do tipo I ou P anteriores ou posteriores. Enquanto *frames* P só fazem referência a *frames* do tipo I ou P anteriores.

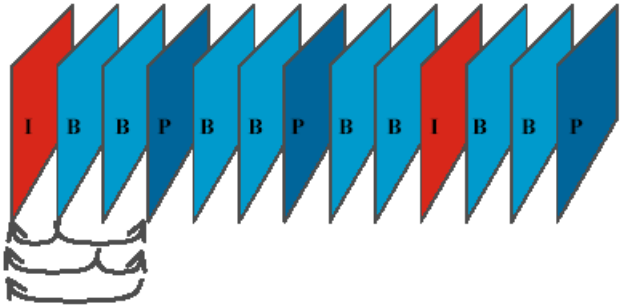

Figura 2.3: *Identificação dos frames de vídeo MPEG*

O processo de decodificação terá então de reconstruir todas as imagens com base na imagem de referência e as "diferenças"contidas nos *frames* B e P, figura 2.4.

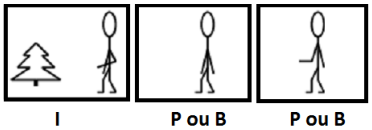

Figura 2.4: *Exemplo frames de vídeo MPEG*

Quanto maior a compressão, maiores serão as perdas de qualidade sofridas nos *frames*, o

que cria a necessidade de intercalar *frames* I, para permitir a decodificação com qualidade, e também acesso aleatório aos *frames* do filme [Wat04].

MJPEG

Motion JPEG (M-JPEG ou MJPEG) é um formato de vídeo em que cada *frame* ou campo entrelaçado, de uma sequência de vídeo digital é comprimido separadamente em imagens JPEG. Originalmente desenvolvido para aplicações em multimídia PC e ao longo do tempo substituído por formatos mais avançados. O formato MJPEG é agora usado por muitos dispositivos portáteis com capacidade de captura de vídeo, como câmaras digitais, câmaras IP, e *webcams* [CSL12].

Considerando formatos de codificação inter-*frame* de vídeo, como MPEG1, MPEG2 e AVC H.264/MPEG-4, que alcançam taxas de compressão 1:50 ou maiores. A falta de previsão inter-*frame* do MJPEG limita sua eficiência de compressão para 1:20 ou inferior. É também computacionalmente mais leve comparado com os restantes formatos, e permite fácil acesso a cada *frame*. O MJPEG necessita de menos processamento e memória, porque os *frames* são comprimidos de forma independente uns dos outros.

Na figura 2.5 é apresentado um exemplo de uma sequência de *frames*, que pretende demonstrar a limitação do MJPEG em relação a formatos com compressão inter-*frame*, como o apresentado na figura 2.4.

Figura 2.5: *Exemplo frames de vídeo formato MJPEG*

A grande desvantagem do MJPEG é que não explora as redundâncias temporais entre *frames* sucessivos, na sequência de vídeo digital, para alcançar uma maior compressão [DT08].

2.2 Processamento digital de imagem

Processamento Digital Imagem (PDI) refere-se ao uso de algoritmos computacionais para efetuar operações sobre imagens, ou seja, a manipulação de dados por computador em que a entrada e a saída do processo são imagens. Um sistema de processamento digital de imagens é constituído por diversas etapas, tais como:

- Aquisição, esta etapa consiste na obtenção das imagens a serem processadas;
- Pré-processamento, visa a melhora da qualidade da imagem e sua adequação às fases posteriores;
- Segmentação, o objetivo da segmentação é identificar regiões de interesse;

- Classificação e Reconhecimento, é a etapa onde as informações presentes na imagem são interpretadas.

Esta secção tem como objetivo apresentar técnicas e conceitos fundamentais do PDI, pertencentes as etapas anteriormente enunciadas, essenciais para a utilização dos algoritmos de *tracking* de objetos apresentados neste trabalho.

2.2.1 Histogramas

Os histogramas são ferramentas de processamento de imagens que possuem grande aplicação prática. Os histogramas são determinados a partir de valores de intensidade dos *pixels*. Entre as principais aplicações dos histogramas estão o melhoramento da definição de uma imagem, a compressão de imagens, a segmentação de imagens ou ainda a descrição de uma imagem. Um histograma de intensidades de cor é uma função de acumulação de frequências dos níveis de intensidade que ocorrem numa imagem.

Normalização de histograma

O histograma de uma imagem I, cujos valores de intensidade estejam entre 0 e G, é definido pela equação 2.1:

$$h(I_k) = n_k; \tag{2.1}$$

Onde I_k é um valor de intensidade k, $(0 \leq k \leq G)$ da imagem I e n_k é o número de *pixels* na imagem I que possuem a intensidade k.

A operação de normalização do histograma consiste em dividir o valor acumulado da intensidade x pelo número máximo de *pixels* y [FR95]. É possível normalizar um histograma, representando os valores em termos de percentagem, conforme mostrado na equação 2.2:

$$p(I_k) = \frac{h(I_k)}{n} = \frac{n_k}{n}; \tag{2.2}$$

onde n é o número de *pixels* da imagem, n_k o número de *pixels* na imagem com a intensidade k. A Figura 2.6 apresenta a obtenção de um histograma a partir dos valores de intensidade dos *pixels* de uma imagem.

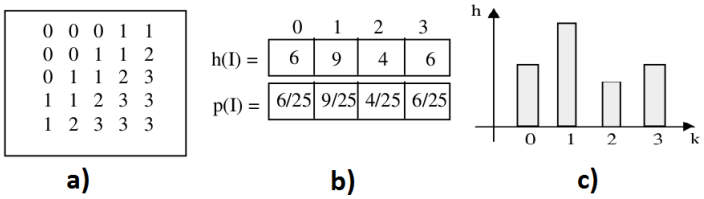

Figura 2.6: *a) imagem I, b) o histograma da imagem em valores ($h(I)$) e em percentagem ($p(I)$), c) representação do histograma de forma gráfica*

Equalização de histograma

Uma operação bastante comum com histogramas é o ajuste dos valores de intensidade de forma a melhorar o contraste da imagem. Esta operação é chamada de equalização de histogramas. O objetivo da equalização de um histograma é obter uma distribuição de intensidades de cor uniforme, ou seja, mapear os valores de intensidade de uma imagem de um intervalo pequeno (pouco contraste) para um intervalo maior (muito contraste) e ainda distribuir os *pixels* ao longo da imagem.

O processo de equalização do histograma tem aplicação prática em casos de adaptação automática a diferentes ambientes de muita luminosidade ou zonas de pouca luminosidade. A Figura 2.7 mostra um exemplo de uma imagem que foi ajustada utilizando equalização de histograma.

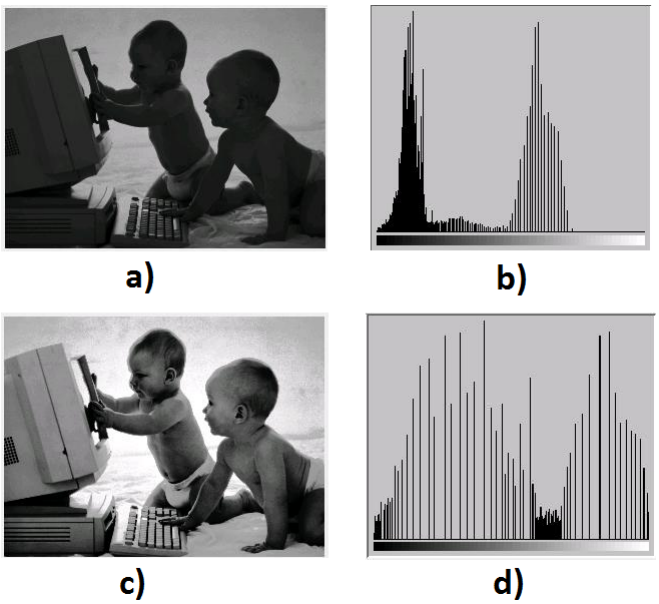

Figura 2.7: *Comparação entre imagens equalizada e não equalizada, [his12]*

Em que a figura 2.7 a) ilustra uma imagem em tons de cinzento, cujo histograma é apresentado figura 2.7 b). A equalização de histograma, da imagem original, pode ser observada na figura 2.7 c), sendo o seu histograma representado na figura 2.7 d).

2.3 Seguimento(Tracking)

O *tracking* de objetos é uma tarefa importante no campo da visão computacional, que teve um elevado crescimento devido à proliferação de computadores com elevada capacidade de processamento, à disponibilidade de câmaras de vídeo de elevada qualidade e à crescente necessidade de automatizar a análise de vídeo. O *tracking* de objetos pode ser definido como o processo de segmentar um objeto de interesse em *frames* de vídeo consecutivos. Permitindo conhecer o seu movimento, a sua orientação, visibilidade, de forma

a extrair informações úteis [M.T09]. Os algoritmos de *tracking* são aplicados em diversas áreas, como sistemas de segurança onde são criados sistemas de visão que detetam e identificam movimentos estranhos e reportam essa informação. Na robótica, onde são usados para detetar objetos estacionários ou em movimento e evitar colisões. Monotorização de tráfego para deteção de contraordenações [M.T09]. O algoritmo de *tracking* deve, em cada *frame* ao longo de uma sequência de vídeo, detetar a posição do objeto em questão. Dependendo do objetivo, pode adicionalmente extrair informações do objeto, tais como, a sua orientação, área ou forma. No entanto, o *tracking* pode-se tornar complexo, em alguns casos da sua grande área de aplicabilidade, devido à existência de requisitos diferenciados. Requisitos esses que dão origem a diversas dificuldades à sua implementação, algumas dessas dificuldades, são identificadas por Deepak-Sukadev em [PM11], como sendo:

- Perda de informação ao projetar a imagem Três Dimensões (3D) para um plano Duas Dimensões (2D);
- O ruído na imagem, resultando em perda de informação;
- Dificuldade em encontrar a posição exata do objeto em movimento em cada *frame*;
- Parcial ou total oclusão do objeto;
- Objetos com formas complexas;
- Não estar disponível uma visão desobstruída do fundo;
- Movimento no fundo;
- Variação nas condições de iluminação;
- Necessidade de processamento em tempo real.

2.3.1 Seleção de características

Selecionar as características certas para o *tracking* é essencial, uma vez que a escolha errada das características a utilizar pode dar origem a algoritmos lentos e imprecisos. O aspeto mais importante na escolha de uma característica é a sua unicidade, para que os objetos alvo possam ser facilmente distinguidos dos restantes. A escolha de características está relacionada com a representação do objeto e com o tipo de algoritmo que se pretende utilizar. Normalmente os algoritmos de *tracking* utilizam uma combinação de características, sendo as mais comuns as seguintes:

- Cor − a cor aparente (visualizada) de um objeto é influenciada principalmente por dois fatores físicos: a distribuição espetral da luminância e a capacidade de reflexão do objeto. Existem diversos espaços de cores (RGB, HSV, YCbCr, etc), sendo que a sua escolha depende de cada aplicação.

- *Edges* (arestas) − A fronteira de objetos normalmente provoca profundas alterações na intensidade da imagem. A deteção de arestas é utilizada para evidenciar estas alterações. Uma propriedade importante das arestas é que são muito menos sensíveis a mudanças de luz que a cor[BKD01].

- *Optical Flow* (fluxo ótico) − O fluxo ótico baseia-se em vetores de deslocamento que definem a translação de cada *pixel* numa região da imagem. São calculados usando o brilho de cada *pixel*, assumindo que esse brilho deve ser aproximadamente constante em *frames* consecutivos [HS81].

- Textura − é a medida de variação de intensidade de uma superfície que quantifica propriedades tais como suavidade e regularidade. Comparada à cor, a textura requer mais processamento. Assim como as arestas, a textura também é menos sensível às mudanças de luz que a cor.

2.4 Técnicas de tracking

O processo de *tracking* em vídeo é amplamente estudado, e existem inúmeros algoritmos para implementar esse processo. A escolha do algoritmo depende de vários fatores como o tipo de aplicação, capacidade de processamento disponível, e das características do vídeo e ambiente onde é adquirido. Para melhor compreender o conceito de *traking*, são feitas nesta secção algumas considerações sobre algumas das técnicas existentes, quais as suas vantagens e desvantagens.

2.4.1 Subtração de Fundo

A subtração de fundo é uma técnica de segmentação de objetos usada para separar objetos de interesse do restante da imagem, que visa diferenciar, numa sequência de vídeo, os objetos dinâmicos (em movimento) dos estacionários (parados). Sendo possível remover os elementos estacionários da imagem é possível otimizar a área de procura do objeto que se deseja seguir, o que leva a um melhoramento em tempo e processamento. A utilização das técnicas de subtração de fundo apresenta uma serie de problemas relacionados com o ambiente de aplicação. As soluções existentes aplicam-se em condições específicas relacionadas à aplicação que se deseja criar. O maior desafio de um algoritmo de subtração de fundo é ser robusto em relação a: variações na iluminação (posição e intensidade), presença de sombras, objetos em movimento que sofrem oclusões, regiões com superfícies espelhadas, mudanças na movimentação (oscilação da câmara e objetos) e mudanças na geometria do fundo. Alguns autores apresentam propostas neste campo, Stauffer et al.[SGP99] apresenta uma proposta que lida com múltiplos modelos de fundo, conhecido como *Mixture of Gaussians* (MOG). Neste modelo cada *pixel* da imagem pode ser modelado por uma mistura de múltiplas gaussianas. O algoritmo Wallflower de Toyama et al. [TKBM99] implementa o filtro linear de *Wiener* (um modelo simplificado do filtro

de *Kalman*) para aprendizagem e previsão de eventuais mudanças no cenário de fundo. Kim et al. [KCHD04] apresentaram o chamado *Codebook* (CB). Este algoritmo permite a construção de um modelo de fundo a partir de longas sequências de vídeo.

O processo básico envolvido na subtração de fundo consiste em comparar cada *frame* do vídeo com um *frame* de referência. Este *frame* deve conter apenas os elementos estáticos da cena, ou seja, o "fundo" da imagem. Quando um *píxel* do *frame* analisado é muito diferente do *píxel* correspondente no *frame* de referência, considera-se que esse *píxel* poderá pertencer a um objeto em movimento. A Figura 2.8 demonstra um exemplo de subtração de fundo aplicado a uma imagem em que as setas representam os objetos em movimento, e as formas geométricas os objetos estacionários.

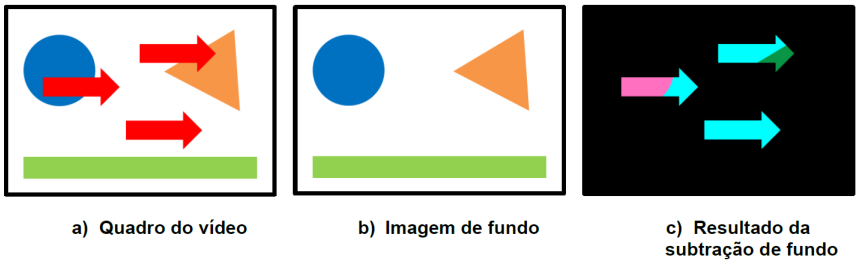

Figura 2.8: *Representação da aplicação da técnica de subtração de fundo [Bri11]*

Na figura 2.8 *a*) presenta o *frame* de vídeo, a 2.8 *b*) o *frame* de referência ou "fundo", e a 2.8 *c*) o resultado da subtração de fundo. Se for considerado o sistema de cores *Red,Green,Blue* (RGB), a subtração de fundo é calculada da seguinte forma:

$$R_{i,j}(subtra) = |R_i, j(frame) - R_i, j(fundo)| \tag{2.3}$$

$$G_{i,j}(subtra) = |G_i, j(frame) - G_i, j(fundo)| \tag{2.4}$$

$$B_{i,j}(subtra) = (B_i, j(frame) - B_i, j(fundo)| \tag{2.5}$$

Em que $R_{i,j}, G_{i,j}, B_{i,j}$, são as componentes RGB do *píxel* com coordenadas (i, j) das imagens utilizadas no processo.

2.4.2 Modelos PGM

O *tracking* baseado em métodos probabilísticos, permite entre outros aspetos, minimizar o efeito de ruido nos *frames* obtidos. Estes modelos, recorrendo ao espaço de estados, fazem uma estimação do movimento com base em observações anteriores. Partindo da posição inicial do objeto, obtida por um método de deteção, é estimada a velocidade e aceleração de forma a prever a posição do objeto na *frame* seguinte. Métodos de *tracking* probabilísticos têm como objetivo estimar posição atual do objeto, recorrendo ao conhecimento do estado anterior. Modelos deste tipo que combinam a dinâmica do objeto

com as observações, demonstraram-se bastante eficientes em problemas de estimação de movimento.

Algumas das vantagens relacionadas a este tipo de modelos são:

- Capacidade extra de lidar com a incerteza afeta às observações do objeto alvo;
- Permitem encontrar soluções eficientes para problemas complexos com requisitos de processamento de tempo real;

Uma hierarquia dos modelos *Probabilistic Graphical Model* (PGM) pode ser visualizada na Figura 2.9.

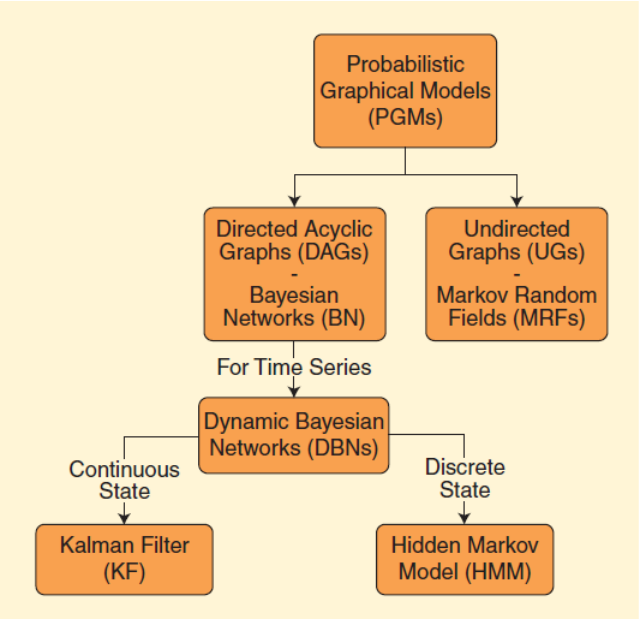

Figura 2.9: *Hierarquia dos modelos PGMs mais usados [DSC10]*

Os modelos PGM podem ser divididos em duas classes, isto é, *directed acyclic graphs* (DAGs) e *Undirected Graphs* (UGs). Em ambos os casos, a ideia básica é a de proporcionar uma ferramenta gráfica para decompor uma distribuição de probabilidade multivariável, fornecendo uma descrição visual intuitiva e maneável. Dentro dos DAGs os suscitam particular interesse os *Dynamic Bayesian networks* (DBN) cuja vantagem é a capacidade de modelar dependências entre relações temporais. O estado do objeto sob *tracking* é representado à custa de variáveis de estado e do modelo das dependências probabilísticas entre elas. Como extensão dos DBN temos os *Kalman filter model* (KFM) e *Hidden Markov model* (HMM). Encontra-se também na literatura, alternativas a estes, mais eficientes que a utilização de aproximações a *Kalman Filter*, mas um pouco mais complexos [DSC10].

Dos vários modelos PGM existentes, os do tipo *Bayesian Networks* (BN) são alvo de análise mais profunda, pois proporcionam um método recursivo e eficiente de atualizar o estado do objeto sob *tracking* em cada *frame*. Este filtro é constituído por duas fases,

predição e correção. Através de uma equação dinâmica é calculado o estado atual, partindo do estado anterior. O estado é depois corrigido usando a informação da medida atual. Caso apenas se pretenda a previsão para um objeto, os dois passos acima são suficiente. Por outro lado, se forem vários os objetos presentes, é necessário um método de associação dos objetos da cena atual à cena anterior.

2.4.3 Mean-Shift Tracking e Camshift Tracking

O procedimento *Mean-Shift* foi originalmente apresentado em 1975 por Fukunaga e Hostetler, é uma técnica genérica para análise de dados, de análise espacial não paramétrica. Como princípio de funcionamento, baseia-se na procura do valor máximo local da densidade de probabilidade de uma variável. Implementa uma análise num subconjunto de valores determinado por uma janela em torno de um ponto de partida definido. Encontra o maior valor dentro desta janela desloca a janela para o ponto determinado anteriormente e o processo é repetido, como exemplificado na figura 2.10. O máximo local terá sido encontrado quando a janela parar de se deslocar entre iterações, ou se deslocar menos do que um limiar determinado. Na figura 2.10, o máximo local é encontrado ao fim de 4 iterações.

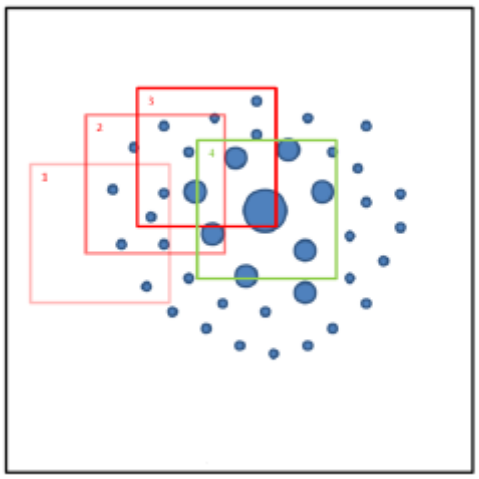

Figura 2.10: *Exemplo das iterações do algoritmo Mean-Shift [Bri11]*

Uma das áreas de aplicação deste algoritmo, em visão por computador, é o *tracking* de objetos coloridos, onde utiliza distribuições de probabilidade para seguir objetos em *frames* de vídeo consecutivos. Para distribuições de probabilidade alteradas dinamicamente, que representam objetos em movimento na sequência de *frames*, o algoritmo *Mean-Shift* tem de ser modificado para se adaptar dinamicamente às alterações de tamanho e posição do objeto, nas distribuições de probabilidade. O novo algoritmo que preenche esses requisitos é chamado Camshift [AM04].

O Camshift é um algoritmo robusto, não paramétrico baseado no seguimento do máximo do gradiente da distribuição de probabilidade (*Mean-Shift*). O objetivo é en-

contrar o valor máximo local da densidade de probabilidade de uma variável. Para a utilização deste algoritmo é necessário o calculo da *probability distribution image* (PDF). Esta pode ser determinada utilizando um qualquer método que associe o valor de um *pixel* com a probabilidade deste *pixel* pertencer ao objeto alvo. Um método comum é o histograma projeção de fundo (*Histogram Back-Projection*). Esta técnica desenvolvida por Swain e Ballar em 1991[SB91], recorre a comparação de histogramas, ou seja, à relação entre o histograma do objeto selecionado (janela inicial) e o histograma do *frame* alvo. Desta forma pretende-se aumentar a diferença entre o *backgroud* e o objeto, levando a uma localização mais fiável deste. O histograma projeção de fundo utiliza o canal matiz (*hue*) no espaço de cor HSV, que é um modelo de cores baseado em três canais: a matiz (*hue*), que representa a cor, a saturação (*saturation*), que representa a concentração de cor, e o valor (*value*), que representa o brilho da cor [AXJ06]. No entanto qualquer histograma multidimensional a partir de um qualquer espaço de cor pode ser utilizado.

Para viabilizar o *tracking* com o Camshift é ainda necessário que o tamanho da janela se adapte ao alvo que está a ser seguido. Uma vez que o tamanho ideal da janela varia consoante o objeto se encontra mais perto ou mais afastado das câmaras. Esta adaptação é feita com base no momento de ordem zero, que pode ser interpretado como a "area"da distribuição encontrada sob a janela de pesquisa. Desta forma o comprimento e a altura da janela são determinados em função deste momento que consiste na soma das probabilidades de todos os *pixels* na região de interesse [Bra98b]. Como produto deste método resultam não só as coordenadas do *pixel* centroide do alvo, correspondente à localização do *Mean-Shift*, mas também a informação respeitante à área da janela. Neste caso a área corresponde à superfície de uma elipse que é caraterizada pelos seus eixos maior e menor e orientação [hLhLsJ03].

A localização média (centroide) dentro da janela de pesquisa na imagem de probabilidade discreta, é encontrado usando momentos [Bra98a]. Dado que $I(x, y)$ é a intensidade da imagem de probabilidade discreta em (x, y) dentro da janela de pesquisa. O momento de ordem zero é dado pela equação:

$$M_{00} = \sum_x \sum_y I(x, y) \tag{2.6}$$

O primeiro momento para x e y

$$M_{10} = \sum_x \sum_y xI(x, y); M_{01} = \sum_x \sum_y yI(x, y) \tag{2.7}$$

A localização janela pesquisa é dada por:

$$x_c = \frac{M_{10}}{M_{00}}; y_c = \frac{M_{01}}{M_{00}} \tag{2.8}$$

Os valores de (x_c, y_c) são calculados até que não haja deslocamento significativo. Segundo

Bradski em [Bra98a], o número máximo de iterações do *Mean-Shift* é, geralmente, de 10 a 20 iterações.

Resumidamente o algoritmo Camshift pode ser definido nos seguintes passos [Int01]:

- 1 - Selecionar a região de interesse da distribuição probabilidade da imagem como sendo a imagem inteira;
- 2 - Selecionar a localização inicial da janela de busca do *Mean-Shift*. A localização dessa região será a distribuição alvo a ser seguida;
- 3 - Calcular a distribuição de probabilidade de cor centrada na janela de procura do *Mean-Shift*;
- 4 - Iteração com o algoritmo do *Mea-Shift* para encontrar o centroide (centro de massa, ponto que representa a média da distribuição da cor do alvo) da imagem de probabilidade, e o momento de ordem zero;
- 5 - Para o próximo *frame*, centrar a janela de procura na localização encontrada no passo 4, redimensionar a janela em função do momento de ordem zero e voltar ao passo 3;

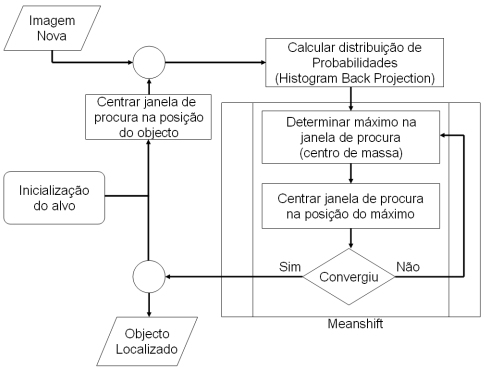

Figura 2.11: *Fluxograma do camshift*

O passo inicial é estabelecer a área de interesse, para a qual é calculado o histograma, que será usado como referência. A seguir centra-se a janela do *Mean-Shift* no objeto que se pretende seguir e determina-se a distribuição de probabilidades desta região. A partir daqui o *Mean-Shift* itera, até não haver alteração na posição, ou a alteração ser inferior a um valor previamente definido. Neste ponto estabelece a nova janela de procura e torna-se a repetir todo o processo enquanto se pretender fazer *tracking* deste objeto[AXJ06], como representado no fluxograma da figura 2.11.

2.5 Teoria do classificador Viola-Jones

Nesta secção é apresentado teoricamente o funcionamento do classificador de Viola-Jones, o seu método de caraterização de objetos, funções de classificação e todo o proce-

dimento até a deteção do objeto.

A técnica proposta por Viola e Jones em [VJ01] permite efetuar a deteção de qualquer tipo de objeto (neste caso faces humanas) de forma genérica e em tempo real com uma taxa de sucesso bastante elevada. A abordagem proposta baseia-se em algoritmos de aprendizagem automática *machine learning*, aplicados de forma a identificar objetos, e capazes de processar imagens com muita rapidez.

2.5.1 Características de um Objeto

O algoritmo para deteção de objetos é baseado nas suas principais características (feições), previamente obtidas a partir do *machine learning*. Tais características são responsáveis por diferenciar objetos uns dos outros. Cada conjunto de características encontradas num dado objeto é distintivo dos conjuntos de características encontradas noutros objetos. A maior motivação para o uso de características de um objeto em detrimento de todos os seus *pixels*, é que a velocidade da análise de uma imagem baseada no conjunto de suas principais características é muito maior. Isto porque, o número de características é substancialmente inferior ao número total de *pixels* de um objeto. Viola e Jones propõem em [VJ01] a utilização de três formatos de características:

- Característica de "dois retângulos", onde o seu valor é a diferença entre a soma dos *pixels* dentro de duas regiões retangulares adjacentes de mesmo tamanho;
- Característica de "três retângulos", cujo valor é a soma dum retângulo central menos a soma de dois retângulos externos;
- Por fim, as características de "quatro retângulos"cujo valor é calculado através da diferença entre pares diagonais de retângulos.

Os formatos de características possíveis são exemplificados na figura 2.12. Os tipos de características representados nas figuras 2.12-a) e 2.12-b) são do tipo de "dois retângulos", a primeira encontra-se no sentido vertical e a segunda no sentido horizontal. As figuras 2.12-c) e 2.12-d) ilustram as características "três retângulos"e "quatro retângulos"respetivamente.

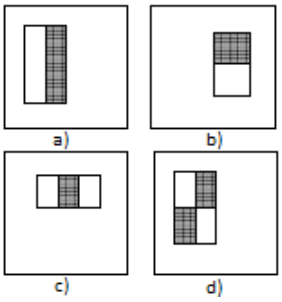

Figura 2.12: *Formas possíveis de uma característica retangular [VJ01]*

A motivação para a definição de tais características no formato retangular, também conhecidas como "características Haar", é baseada no estudo realizado previamente por Papageorgiou e Poggio em [PP00]. No qual é proposto o uso de *Haar Wavelets* [Mal89] para facilitar o treino de classificadores. Tal estudo demonstrou que o uso destas características retangulares são suficientes para a extração de informações relevantes das formas de um objeto. Com o intuito de otimizar o cálculo de tais características, Viola e Jones em [VJ01] propõem a utilização de uma representação intermediária da imagem chamada de "Imagem Integral". Nesta forma de representação, cada ponto x,y da imagem contém o somatório de *pixels* da origem da imagem até a sua localização figura 2.13. Tal forma de representação pode ser calculada com uma única passagem na imagem original, de acordo com a equação 2.9.

$$ii(x,y) = \sum_{x' \leq x, y \leq y} i(x',y') \tag{2.9}$$

onde (x', y') são as coordenadas de cada ponto na imagem original.

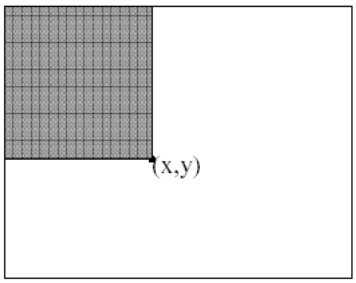

Figura 2.13: *Exemplo de área retangular a ser calculada na Imagem Integral [VJ01]*

Tendo-se calculado a imagem integral, é possível encontrar o valor de uma área retangular qualquer. Para tal utiliza-se apenas os quatro pontos dos vértices da área desejada. Supondo que temos o valor total da área de origem da imagem até cada um dos quatro vértices, para se encontrar a área desejada basta realizar uma subtração simples de retângulos. Por exemplo, na figura 2.14 é possível determinar a região D, sabendo que o valor na posição 1 é a soma dos *pixels* do retângulo A, o valor da posição 2 será $A + B$, a posição 3 será $A + C$ e a posição 4 será $A + B + C$. Tendo as quatro referências a soma de D é definida pelos valores das posições $4 + 1 - (2 + 3)$.

Com o intuito de melhorar ainda mais a etapa de deteção de objetos, Lienhart e Maydt em [LM02] decidiram expandir as representações quadráticas *Haar* através da introdução de novas representações. Com estas novas adições ao conjunto, conseguiram reduzir o número de falsos positivos cerca de 10 %, aumentando ainda mais a eficiência do processo de deteção. Na figura 2.15 é apresentado o conjunto completo com as novas características retangulares propostas.

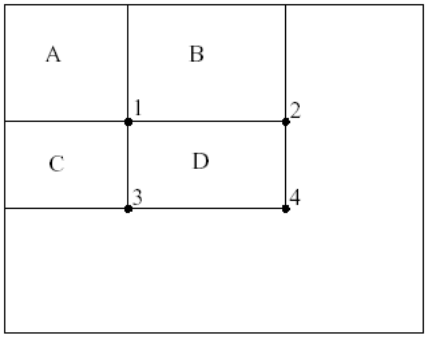

Figura 2.14: *Cálculo da área de região retangular de uma Imagem Integral[VJ01]*

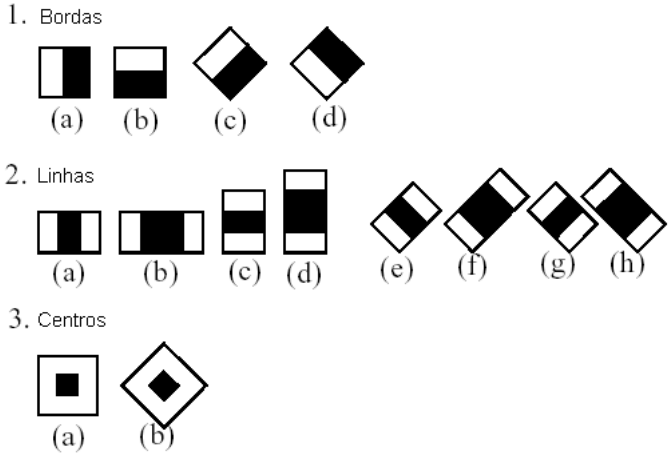

Figura 2.15: *Características Estendidas [LM02]*

2.5.2 Funções de Classificação

As funções de classificação tem como objetivo a classificação de objeto a partir do conjunto das suas principais características. Para encontrar essas características, podem ser utilizados vários tipos de algoritmos de aprendizagem, tais como, redes neuronais [1] e *Support Vector Machine* (SVM) [2] [MLH03]. O algoritmo escolhido por Viola e Jones em [VJ01] é uma variante do algoritmo de aprendizagem *AdaBoost* [FS95]. No exemplo da figura 2.16 o classificador percorre a imagem à procura regiões com os mesmos padrões que as características do objeto desejado, no exemplo da figura em questão este objeto é uma face. Na primeira linha da figura 2.16 representa exemplos da primeira e segunda características selecionadas pelo *AdaBoost* durante o treino do tipo de objeto "face humana" (figura 2.16 a)). Na segunda linha da figura as características estão sobrepostas sobre uma face.

- No primeiro caso (figura 2.16 b)), a região dos olhos e as bochechas da pessoa

[1] Redes neuronais são sistemas computacionais estruturados numa aproximação à estruturas do cérebro, em particular do exame de neurónios;

[2] Support Vector Machine são modelos de aprendizagem supervisionados associadas com algoritmos de aprendizagem para analise os dados e reconhecimento de padrões;

possuem diferentes intensidades, sendo a região olhos mais escura do que a região das bochechas;

- O segundo caso (figura 2.16 c)) permite comparar a intensidade da região dos olhos com a intensidade dos *pixéis* da região do nariz.

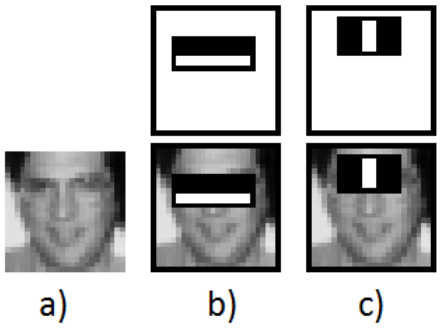

Figura 2.16: *Características retangulares mais marcantes de uma face [VJ01]*

Para que o algoritmo de aprendizagem seja executado é necessário um grupo de imagens do objeto de interesse, chamado de imagens positivas, e um segundo grupo de imagens que não contenha o objeto, chamado de imagens negativas.

2.5.3 Cascata de Classificadores

A cascata de classificadores é organizada de forma que os seus estágios iniciais descartem um grande número de regiões que não contém o objeto desejado, para que estágios mais avançados sejam cada vez mais precisos para evitar um falso positivo. Caso uma área da imagem passe pelo último estágio da cascata, então esta área contém o objeto desejado. Os estágios dentro de uma cascata são criados através da combinação de funções de classificação previamente montadas pelo uso do algoritmo de aprendizagem *AdaBoost*. Na figura 2.17 são representados, as ligações e saídas dos N estágios de uma cascata de classificadores.

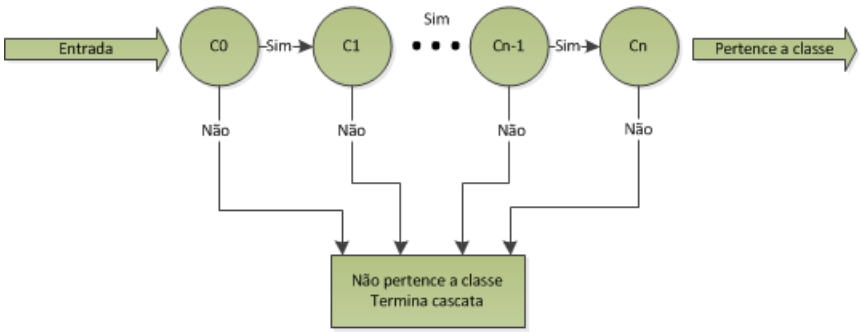

Figura 2.17: *Cascata de Classificadores*

O processo para geração da cascata é guiado por um conjunto de objetivos na deteção e no desempenho. O número de estágios na cascata deve ser o suficiente para garantir uma taxa elevada de deteção de objetos, e uma baixa ocorrência de falsos positivos. Dado uma cascata de classificadores treinada, a sua taxa de falsos positivos é de:

$$F = \prod_{i=1}^{K} f_i \tag{2.10}$$

onde K é o número de classificadores e f_i é a taxa de falso positivo do classificador i. A taxa de deteção é de:

$$D = \prod_{i=1}^{K} d_i \tag{2.11}$$

Onde d_i é a taxa de deteção do classificador i. Definindo-se taxas gerais de deteção, e de falso positivo da cascata, é possível também definir metas para cada estágio que a compõe. Por exemplo, uma taxa de deteção de 0,9 pode ser alcançada pelo décimo estágio, se cada um dos estágios possuir uma taxa de 0,99.

Apesar dos estudos realizados demonstrarem que a utilização de uma cascata de classificadores bem treinados ser extremamente eficiente quanto à deteção de objetos rapidamente, o processo de treino para geração da cascata é extremamente demorado. No trabalho apresentado por Viola e Jones [VJ01], foi realizado o treino de uma cascata de classificadores para deteção de faces humanas, com 32 estágios, 4.297 características, 5.000 imagens de faces (24 x 24 *pixels*) e 10.000 imagens que não continham faces. O tempo total de treino publicado foi de semanas numa máquina com um processador 466 MHz. Este tempo elevado de treino para novos objetos é a principal desvantagem da técnica apresentada.

2.5.4 A procura pelo Objeto

Para que o algoritmo possa detetar o objeto de interesse numa imagem é necessário que a imagem seja percorrida diversas vezes em vários tamanhos, o que faz com que em cada passagem seja analisada uma região de tamanho diferente. O utilizador define o tamanho mínimo que a janela de procura pode ter e que será considerado como o tamanho inicial para o processamento de cada região. O tamanho mínimo desta janela de procura deve ser superior ou igual ao tamanho com o qual a cascata foi treinada, ou seja, se a cascata foi treinada com imagens positivas de 24 x 24 *pixels*, o tamanho mínimo da janela de procura deve seguir tal resolução. A janela de procura é deslocada de um valor Δ *pixels* na horizontal e depois na vertical enquanto existirem janelas a serem analisadas na imagem. Para escolha do valor Δ deve ser tido em conta que, o número de janelas reduzido (tamanho da janela maior), diminuiu o tempo de processamento, mas diminuiu também o número de deteções. A imagem deve ser percorrida em várias escalas. Portanto, da mesma forma como se tem um valor Δ para o deslocamento da janela de procura, é

também necessário um fator δ de escalonamento da janela, ou seja, um δ que aumente o tamanho da janela como um todo, em x e y em simultâneo. Desta forma, cada vez que a janela de procura terminar de percorrer toda a imagem na horizontal e na vertical, a imagem deve ser percorrida novamente com um novo tamanho para a janela de procura. Viola e Jones [VJ01] sugerem 1,25 como o valor ideal para δ. O mesmo cuidado utilizado para a escolha do fator Δ deve ser tomado na decisão do fator δ, pois, apesar de deixar a procura com uma velocidade maior, para valores elevados também acarreta o aumento de falsos positivos indesejados. O facto da janela de procura ter que ser redimensionada várias vezes durante a procura por regiões que contenham o objeto de interesse não implica novos cálculos para redimensionar o classificador corrente da cascata. Isto ocorre porque, com uso da técnica de imagem integral, o custo do reescalonamento do detetor é efetuado através de um cálculo algébrico direto. Desta forma, a análise da região da imagem é efetuada de forma muito rápida e eficiente.

2.6 Trabalhos relacionados

Nesta secção é feita uma apresentação do estado da arte relacionado com a deteção e *tracking* de pessoas. É realizada uma análise detalhada de alguns dos métodos propostos por outros autores, que mais se adequam ao tema em estudo. Nos trabalhos analisados foram encontradas diferentes técnicas na área de processamento e análise de vídeo, com vista à deteção e descrição de objetos, para posterior *tracking* dos mesmos.

Tao et. al em [TN04], tratam o *tracking* de pessoas em situações complexas, a técnica proposta aborda um método de deteção e descrição de indivíduos, recorrendo a um modelo de aparência. O modelo é criado no momento em que a pessoa é detetada e é atualizado à medida que ela se move. Apresenta-se na figura 2.18 o processo de extração proposto.

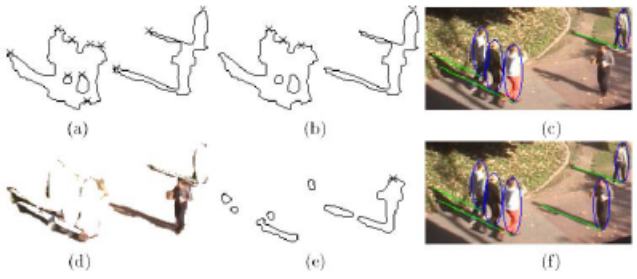

Figura 2.18: *Processo de segmentação das pessoas e extração dos candidatos [TN04]*

Através do processo de segmentação de imagem são extraídas as pessoas presentes, no entanto existe sempre presente um ruído indesejável como sombras ou outros objetos que são também erradamente detetados. A definição dos candidatos é conduzida através

de um método de procura de cabeças, assumindo um tamanho médio da cabeça humana. Inicialmente é efetuada uma procura dos pontos mais elevados na vertical, procedendo-se de seguida à validação aferindo se os pontos correspondem realmente a uma cabeça. Os topos que não tenham tamanho suficiente são descartados (Figura 2.18 b). Cada candidato a pessoa é processado seguindo a ordem de profundidade na imagem (primeiro as mais próximas da câmara) e à medida que são detetadas vão sendo processadas (Figura 2.18d). Este processo é repetido até que não restem mais pessoas a processar na imagem segmentada (*foreground*). O algoritmo efetua também o *tracking* das pessoas, recorrendo a modelos de aparência e estimação de movimento. A estimação é efetuada com recurso a um filtro de *Kalman*, que utiliza como parâmetros como velocidade e posição atual do objeto para tentar prever o seu estado na *frame* seguinte. De seguida, com base nessa estimação é definida uma área de pesquisa, na qual é feito o *matching* com o modelo de aparência.

Outro aspeto importante nesta abordagem é o modelo da câmara, onde é assumido que as pessoas se movem num plano conhecido (referência), o modelo de câmara e o plano de referência juntos permitem projetar os valores 3D num plano 2D. O recurso a um modelo destes faz sentido devido à localização estática da câmara, revelando-se muito útil na obtenção de uma relação entre o mundo real 3D e a imagem 2D. Para a criação deste modelo é inicialmente realizada a calibração da câmara, sendo calculados os parâmetros intrínsecos da câmara, assim como os extrínsecos (relacionados com o local onde são feitas as aquisições). O modelo obtido da câmara possibilita assim conhecer alturas reais das pessoas e outros parâmetros importantes para a deteção dos alvos. Esta relação permite adicionar algumas restrições importantes como a altura das pessoas (não faz sentido detetar uma pessoa se o objeto possuir uma altura desadequada para tal) e ainda restrições espaciais (uma pessoa nunca poderá ocupar um espaço que esteja já ocupado por outra). Ilustra-se de seguida através de um diagrama de blocos o funcionamento do algoritmo proposto pelo trabalho em análise, figura 2.19.

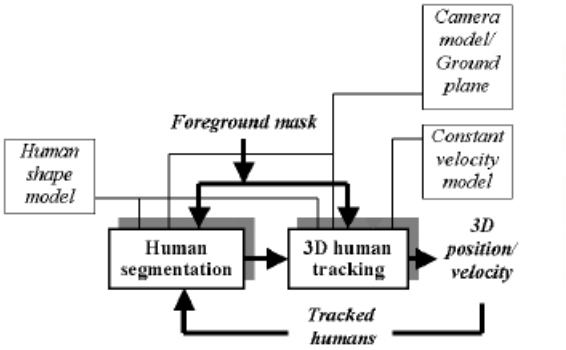

Figura 2.19: *Diagrama de Blocos do Algoritmo [TN04] e posterior edição*

É percetível na figura 2.19 algumas das principais funcionalidades do algoritmo, e

onde se enquadram os diversos modelos referidos. As *Foreground masks* correspondem às imagens segmentadas que servem de entrada ao algoritmo, as quais juntamente com as imagens da sequência de vídeo integram a informação necessária para o algoritmo efetuar a deteção e *tracking* das pessoas. O bloco *human segmentation* trata da parte da deteção das pessoas.

Outras técnicas propostas, traduzem aspetos mais particulares no que diz respeito à descrição e *tracking* de objetos. Em [TCR09] é abordada a aplicação de descritores num cenário de vídeovigilância com múltiplas câmara. O objetivo é identificar e distinguir entre várias instâncias de um mesmo objeto visual, sendo estabelecidas correspondências entre os objetos ao longo das várias câmaras. O algoritmo tem como entrada os resultados fornecidos por um algoritmo de *tracking* comum (usando determinados *features* como descritores e estimação através de *Kalman Filter*[Bas00] ou *Bayesian network inference*[NJS06]) e efetua o processamento de forma a estabelecer uma correspondência entre os vários objetos, independentemente da localização e altura em que surgiram. A correspondência é efetuada à custa de uma robusta descrição visual, uma vez que esta deve ser transparente ao ângulo de captura e outras variações que ocorrem de câmara para câmara, como por exemplo diferenças de iluminação. Assim, a escolha do método recaiu sobre a utilização descritores locais *Likelihood filter* (LF) e o paradigma designado *bag-of-visterms*. Este método constrói um vetor de dimensão igual a 128, onde é guardada a informação relativa à descrição. É usado um descritor *Scale Invariant Feature Transform* (SIFT) desenvolvido em [Low99] para aplicação em cada *keypoint*. Estes *keypoints* são normalmente selecionados ao acaso, podendo também ser definidos por um detetor de pontos de interesse. O facto de ser usado um descritor do tipo SIFT é importante neste caso, pois significa que as *features* extraídas são invariantes à escala da imagem, rotação e ainda a mudanças de perspectiva e iluminação. Na figura 2.20 o algoritmo proposto, identifica a mesma pessoa em câmaras diferentes, sendo que devido a falta de informação visual, num dos casos é erradamente rotulado. Mas ao longo da aquisição a situação é corrigida.

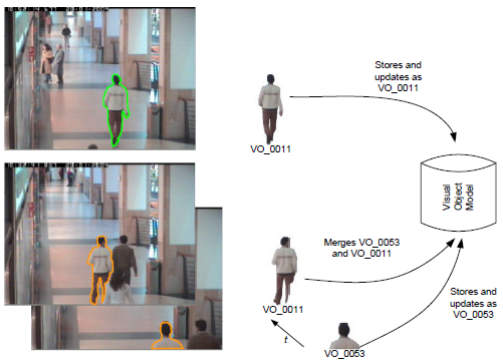

Figura 2.20: *Deteção da mesma pessoa por múltiplas câmaras [TCR09]*

Foi também estudada uma técnica de identificação e categorização de objetos por

classes, a partir dos seus contornos [AAA06]. Este trabalho descreve um método em que os objetos são descritos à custa de *codebooks*. O *codebook* consiste em fragmentos dos limites do objeto, incluindo uma entrada adicional que indica a localização do seu centroide. Este método não possibilita descriminação das várias pessoas, pois a informação dos contornos de uma pessoa não é suficiente por si só. O método teria de ser combinado com outras *features* para formar um descritor robusto, o que aumentaria o peso computacional.

Budi et. al em [BHJS08] propõem um método para deteção e *traking* de objetos em movimento com base em imagem de baixa resolução utilizando a técnica de *block matching* e também propôs um método de identificação utilizando cor e informação espacial. Muitos algoritmos de *traking* têm melhor desempenho sob fundo estático, mas às vezes os resultados sob fundo com movimentos complexos não são os melhores dando origem a *mistracking*. Uma vez que uma imagem de baixa resolução tem a propriedade que pode remover os *pixels* de tamanho pequeno, é adequada para resolver este problema, devido ao facto de a maior parte dos falsos movimentos no fundo serem pequenos. Em *traking* de objetos em movimento, muitas aplicações têm problemas quando os objetos se ocultam um ao outro. O *Peripheral increment sign correlation* (PISC) é usado para resolver este problema. A identificação do objeto é realizada utilizando a cor e informação espacial do objeto que se pretende seguir. O algoritmo proposto pode ser dividido em três etapas, deteção de objetos, *traking* de objetos e identificação de objetos. Na primeira é detetado o objeto em movimento emergente no fundo. Nesta fase, faz-se uma imagem de baixa resolução para remover o ruído de imagem e pequenos movimento no fundo, tais como folhas em movimento. A segunda etapa é realizada para seguir o objeto em movimento. Nesta etapa, é proposto uma técnica de *block matching* feita com base na imagem PISC que considera que a mudança de brilho em todos os *pixels* do bloco é relativa ao *pixel* considerado. Como última etapa, é proposto identificar o objeto seguido por extração da cor e informação espacial. O modelo de cor RGB é usado como informação de cor dos objetos em movimento. Para obter mais informações de cor para identificação, é dividido o objeto em movimento em três partes, a cabeça, a parte superior e a parte inferior do corpo. A característica extraída do objeto no domínio espacial é a posição do objeto a seguir, a caixa delimitadora é utilizada como informação espacial de objetos em movimento. Após extraída a característica espacial, é calculada então a semelhança entre os objetos a seguir e o identificado.

Um outro método proposto em [PM11], para a deteção e *traking* de objetos com fundos complexos, tais como folhas de árvores e de condições de iluminação com variações em tempo real, utiliza um redução da resolução espacial e intensidade da imagem para minimizar os efeitos causados pela desorganização de fundo. Um método *3-frame-differencing* é utilizado para detetar o objeto em movimento. A segmentação do objeto é efetuada utilizando *Fuzzy C-Means* (FCM) a partir do fundo. A FCM é um método de *clustring* que permite que o *pixel* pertença a mais que *cluster*.

O método apresentado é dividido em dez passos desde a aquisição do vídeo até a saída

com o objeto identificado, na figura 2.21 esta representada a sequência de execução.

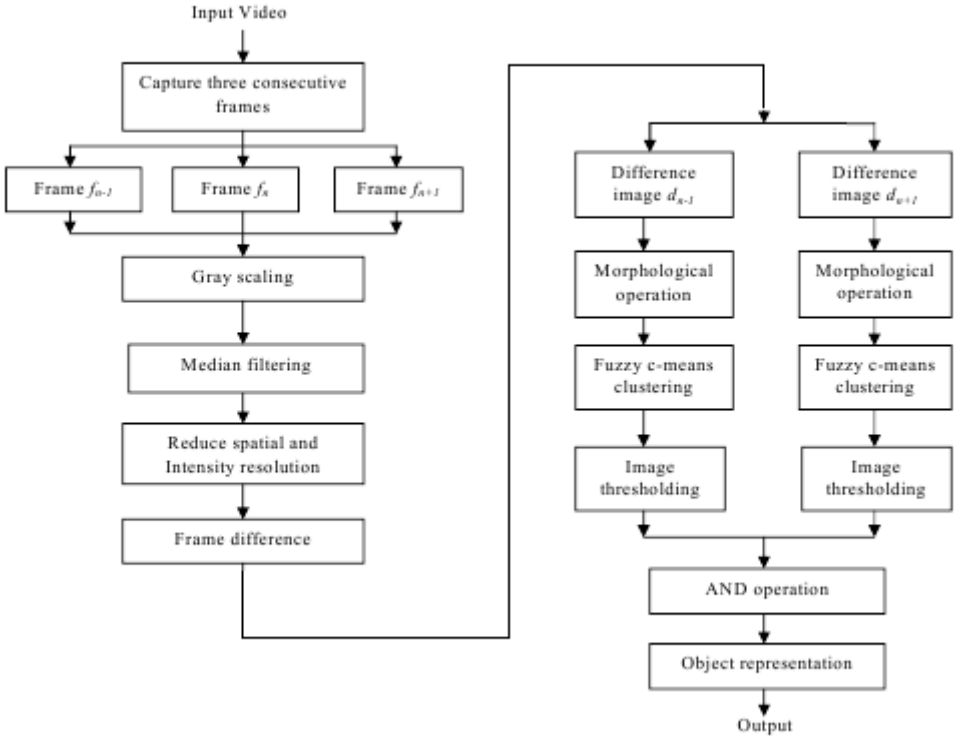

Figura 2.21: *Diagrama de blocos do algoritmo proposto [PM11]*

O algoritmo começa por adquirir a sequência de imagens que compõem o vídeo digital, e a primeira operação sobre essas imagens é uma conversão para tons de cinzento. O alisamento da imagem em tons de cinzento é efetuado recorrendo a um filtro de média [GW00]. Em seguida para minimizar os efeitos do "falso movimento"causado por fundos não estacionários é reduzida a resolução espacial e intensidade da imagem. A redução espacial é efetuada através do calculo da média do valor do *pixel* e dos seus vizinhos. Para a deteção do objeto em movimento e as regiões onde ocorreram alterações é utilizado a diferenciação de três *frames* consecutivos, este método funciona apenas quando não existe movimento da câmara e o objeto não é ocultado. Após a diferenciação de *frames* estar concluída são efetuadas as operações morfológicas de abertura e fecho. A operação de abertura alisa os contorno dos objetos e o fecho preenche falhas nos contornos. FCM é uma técnica não supervisionada que classifica a imagem em *clusters*, no domínio das características. O algoritmo FCM atribui *pixels* para cada grupo usando associações *fuzzy*. O agrupamento é feito iterativamente minimizando a função de custo que é dependente do distância dos *pixels* para o *cluster* central. O resultado obtido a partir do algoritmo FCM é utilizado para limiarização (*thresholding*) do objeto a partir do fundo. Limiarização é uma técnica importante para a segmentação de imagem com base no pressuposto de que os objetos podem ser distinguidos e extraídos a partir do fundo. O passo seguinte é

a operação de AND, usada para encontrar semelhanças entre dois *frames* diferença. A ultima etapa é a representação do objeto segmentado através do centroide e envolvente retangular no objeto figura 2.22.

Figura 2.22: *Traking do objeto em vídeo na presença de fundo não-estacionário [PM11]*

2.7 Hardware

Nesta secção é feita um breve apresentação do *hardware* utilizado para o desenvolvimento do trabalho, com relevo nas caraterísticas mais importantes para o tema em estudo.

2.7.1 Câmara de vídeo

A câmara de vídeo utilizada neste projeto é a Sony SNC-RZ25P, figura 2.23, é uma câmara com *Pan-Tilt-Zoom* (PTZ) com suporte de rede com e sem fios, tem suporte para a codificações de vídeo e MPEG-4 e MJPEG, e de imagem JPEG. Ideal para aplicações de difusão e de monitorização remota.

Figura 2.23: *Aspeto exterior da câmara SNC-RZ25P*

As características gerais da câmara SNC-RZ25P são:

- Monitorização de imagens reais de alta qualidade via rede informática. A partir da câmara SNC-RZ25P (figura 2.23) pode-se monitorizar imagens reais de alta qualidade usando o navegador da *web* no computador, ligado através da norma de rede 10T-T ou 100BASE-TX. A taxa máxima é de 25 *Frames* por segundo (FPS) para a SNC-RZ25P, até 20 utilizadores na *web* podem ver a imagem de uma câmara ao mesmo tempo (em modo JPEG e MPEG-4).

- Mecanismo de controlo remoto de alta velocidade de *pan/tilt* e grande ampliação com focagem automática. A velocidade de rotação para 340º é de 2 segundos. A amplitude do ângulo do mecanismo de *pan*, é de -120º a +120º, com uma velocidade angular de 115º em 1,5 segundos. O mecanismo do *tilt*, tem a amplitude do ângulo de -90º a +25º. O zoom ótico de 18 vezes e elétrico de 12 vezes, resultando num total de 216 vezes, figura 2.24.

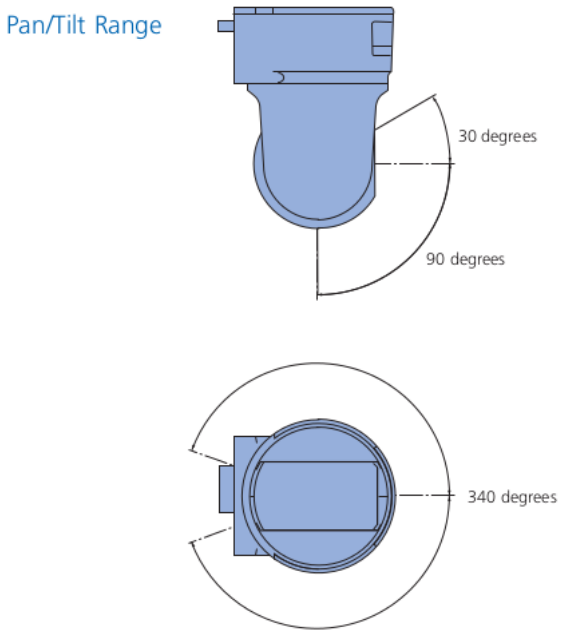

Figura 2.24: *Range de movimento da câmera SNC-RZ25P*

Os protocolos de transmissão e tamanho da imagem podem ser selecionados. Para o protocolo de comunicação as opções disponíveis são *Transmission Control Protocol* (TCP) ou *User Datagram Protocol* (UDP). Os tamanhos de imagem selecionáveis são: $640X480$, $480X360$, $384X288$, $320X240$, $256X192$ *e* $160X120$. A SNC-RZ25 produz imagens ate ao máximo *frame rate* de 25 FPS, com o tamanho de imagem $320X240$ nos dois modos MPEG-4 e JPEG, o *frame rate* pode ser fixo ou ajustado automaticamente de acordo com a largura de banda disponível.

Comandos de movimento

A Sony $SNC - RZ25P$ permite dois tipos de comandos *Common Gateway Interface* (CGI)[1] de movimento. Os comandos movimento são enviados para a CGI "ptzf.cgi", através do método "POST".

[1]Comandos CGI são um método padrão para um servidor web delegar a geração de conteúdo web para executáveis

- O relativo que executa um movimento em função da posição atual, numa determinada direção;

- Zoom num área especifica, é especificado um ponto (x,y), e a dimensão da janela. A câmara faz um zoom sobre essa janela.

O movimento relativo é especificado pelo parâmetro: "relative=AABB", enviado no final do método "POST". Em que o valor de AA indica a direção do movimento (a correspondência entre valores e direções é apresentada na figura 2.25). O valor do deslocamento BB compreende valores de "01"a "10 (apenas valores inteiros). O valor mínimo de incremento "01"corresponde a cerca de 10% do movimento total, e "10"a 100%.

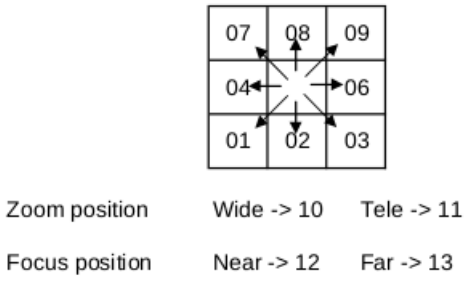

Figura 2.25: *Comando de movimentação relativa, correspondência valor direção*

Para a opção área de zoom, o comando define o parâmetro: "AreaZoom=x,y,w,h,jpeg". Em que x e y representação as coordenadas do centro da área de zoom, w e h, a largura e altura da área de zoom respetivamente, como apresentado na figura 2.26.

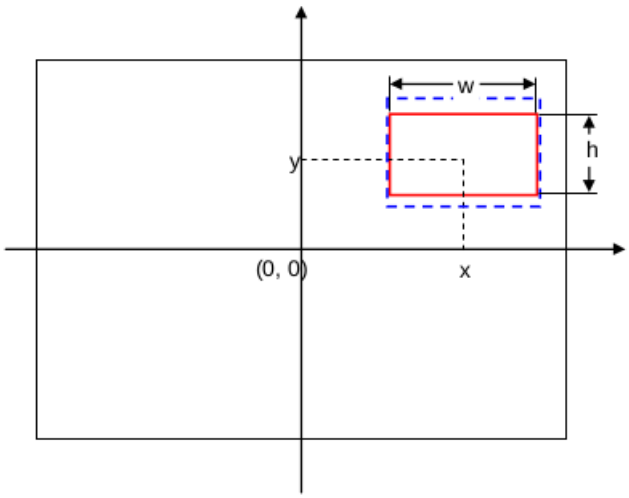

Figura 2.26: *O diagrama conceitual do comando de deslocamento "AreaZoom", [son06]*

2.7.2 Raspberry Pi

Para efetuar o processamento da informação obtida foi utilizado o *Raspberry Pi* (Figura 2.27) que é um computador[1] do tamanho aproximado a um cartão de crédito desenvolvido no Reino Unido pela Fundação *Raspberry Pi*. Todo o *hardware* é integrado numa única placa.

O *Raspberry Pi* consiste num *System on a chip* (SOC) Broadcom BCM2835, que combina um processador *Advanced RISC Machine* (ARM) 11 com uma frequência de *clock* a 700 MHz com uma *Graphics Processing Unit* (GPU) VideoCore IV a 250 MHz. Apesar do *clock* parecer baixo comparando com as GPUs para *desktops*, esta é uma GPU bastante poderosa, com suporte à decodificação de vídeos 1080*p* por *hardware* [rpi12b].

Com um único chip de memória *low-power DDR* (LPDDR) de 256 MB, que limita o desempenho e o uso em aplicações de ambiente gráfico. A memória é compartilhada pelo processador e a GPU, tendo por *default* apenas 186 MB de memória disponíveis para uso do sistema.

A unidade de armazenamento é disponibilizada através de *slot* para cartões *Secure Digital* (SD) disponível na parte inferior. Embora não seja aconselhável (devido ao limite de operações de escrita da memória Flash) é possível usar parte do cartão para memória *swap*, expandindo a gama de aplicativos que podem ser usados [rpi12b].

A alimentação elétrica é disponibilizada por uma porta micro-USB localizada ao lado do cartão de memória, apesar de não muito tradicional, este tipo de conector permite a utilização uma grande variedade de fontes de alimentação existente no mercado e abaixo custo.

Com duas portas *Universal Serial Bus* (USB) (modelo B) o *Raspberry Pi* é muito limitado em potência fornecida através das mesmas. Os conectores são destinados a dispositivos como teclados e ratos, bem como *pendrives* e outros dispositivos de baixo consumo. Para usar dispositivos maior consumo energético, como *Hard Disk* (HD) externo, é aconselhado a utilização um *hub* USB com alimentação própria.

Tem ainda disponível num barramento de 26 pinos, o acesso a diversos periféricos de baixo nível, identificados na lista seguinte [rpi12a]:

- 8 *General Purpose Input/Output* (GPIOs) a 3v3;
- 2-pinos *Universal Asynchronous Receiver/Transmitter* (UART) , 3v3 (debug); ou 2 GPIOs a 3v3;
- *Inter-Integrated Circuit* (I^2C) *interface* (3v3); ou 2 GPIOs at 3v3;
- SPI *interface* (3v3); ou 5 GPIOs at 3v3;
- 3v3, 5v e GND de alimentação;

[1] Um computador é um dispositivo de uso geral que pode ser programado para realizar um conjunto finito de operações aritméticas ou lógicas

- ARM *Joint Test Action Group* (JTAG) (se os pinos forem reconfigurados por *software* e um sinal disponíveis na *Mobile Industry Processor Interface* (MIPI) *Camera Serial Interface 2* (CSI-2) *interface*);
- Segunda *interface* I^2C (3v3) (se os pinos forem reconfigurados por *software*);
- *Integrated Interchip Sound* (I^2S) interface (se os pinos forem reconfigurados por *software*);
- 6 pinos reservados para uso futuro;

Tem ainda disponível dois conectores *Display Serial Interface* (DSI) de 15 pinos para *displays*, um conector CSI-2 para câmara e acesso a GPU JTAG. A figura 2.27 apresenta a disposição na placa, dos principais módulos do sistema.

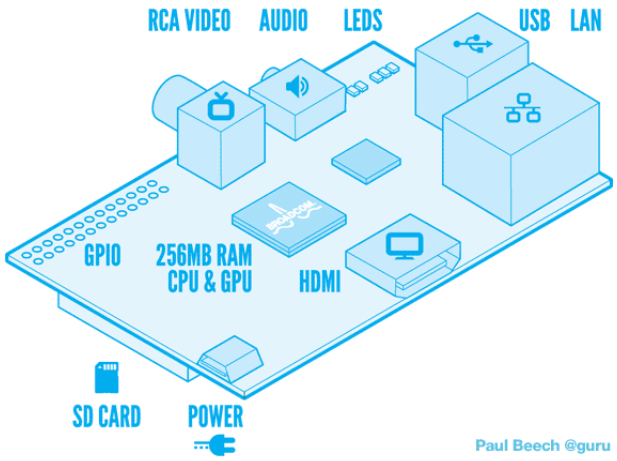

Figura 2.27: *Identificação dos principais módulos da placa Raspberry Pi [rpi12a]*

2.8 Opencv

A biblioteca *Open Source Computer Vision Library* (OpenCV) foi originalmente criada pela Intel. Foi idealizada com o objetivo de fornecer uma infraestrutura, de uso simples, para desenvolvimento na área da visão computacional, que permita o rápido desenvolvimento de aplicações complexas. É disponibilizada em sistema de código fonte livre e gratuito (*OpenSource*). Código esse que é constituído por um conjunto de ferramentas, com classes e funções, que oferecem características de desenvolvimento e implementação de algoritmos ao nível de processamento de imagem e visão computacional.

As linguagens de programação associadas ao OpenCV são, C e $C++$ e é compatível com múltiplos sistemas operativos, tais como, GNU/Linux, Windows e Mac OS X, com *interfaces* de desenvolvimento para Python, Ruby, Matlab, e outras linguagens.

Os algoritmos fornecidos pela biblioteca permitem a criação de aplicações de forma simples na área da visão computacional, onde estas podem ser ou não em tempo real. Como exemplos de áreas de aplicações temos: reconhecimento facial, calibração de câmaras, robótica, vigilância, etc.

2.8.1 Estrutura da biblioteca OpenCV

O OpenCV está estruturado em cinco blocos principais, como apresentado na figura 2.28.

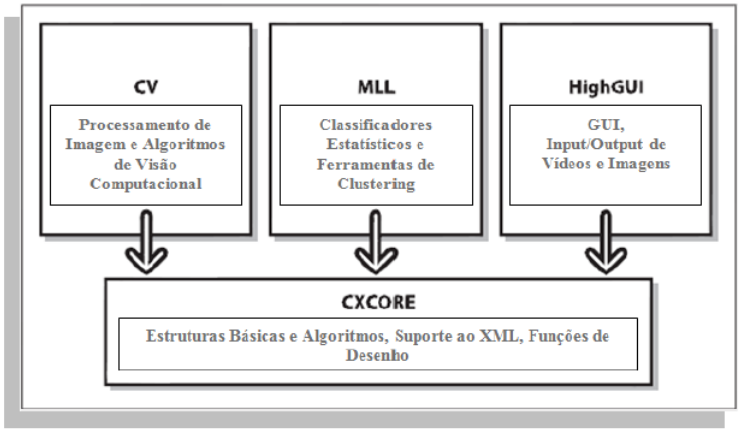

Figura 2.28: *Estrutura básica do OpenCV [BK08]*

- *CV* Constituído por algoritmos de processamento de imagem e algoritmos de visão computacional de alto nível;

- *MLL* Contém bibliotecas de *Machine Learning* que inclui vários classificadores estatísticos, classificadores esses que permitem organizar de forma coerente os dados

recebidos com base nas amostras ou exemplos fornecidos pelo utilizador e ferramentas de *clustering* que fazem o agrupamento dos dados automaticamente de acordo com o nível de semelhança;

- *HighGUI* possui rotinas e funções *Input/Output* para armazenamento e processamento de vídeos e imagens;
- *CXCore* contém as estruturas básicas de dados e conteúdo [BK08].

2.8.2 Deteção de face no Opencv

Nesta secção é apresentado o funcionamento da deteção de face na biblioteca OpenCV, as suas diversas etapas e princípios teóricos. É apresentado o algoritmo de base da deteção, proposto por Viola e Jones.

Classificador Haar

A utilização de algoritmos para a extração de características de objetos e a geração de classificadores em cascata capazes de reconhecer padrões de objetos numa imagem tem demonstrado resultados robustos, eficientes e extremamente rápidos para a deteção de objetos em tempo real. Entre os trabalhos que abordam o problema de deteção de objetos em tempo real, destaca-se a técnica proposta inicialmente por Viola e Jones em [VJ01] por apresentar uma solução genérica quanto à deteção de qualquer tipo de objeto, que mais tarde foi estendida por Rainer Lienhart e Maydt Jochen [LM02] para usar os recursos diagonais. O OpenCV implementa uma versão dessa técnica de deteção, referindo-se a este detetor como o "classificador Haar" que utiliza características *Haar*, ou mais precisamente, como *Haar-like wavelets*. Estas consistem em adicionar e subtrair regiões retangulares da imagem antes limiarizar (thresholding) o resultado. O OpenCV vem com um conjunto de ficheiros pré-treinados para o reconhecimento de objetos, mas o código também permite que se crie e utilize, outros treinados pelo utilizador.

Aprendizagem Supervisionada e Teoria Boosting

O classificador *Haar* que está incluído no OpenCV é um classificador supervisionado. Tipicamente são apresentados histogramas e partes de imagens de tamanho regular ao classificador, que são então identificados como contendo (ou não contendo) o objeto de interesse, que para este classificador o mais comum é o rosto. O detetor de Viola-Jones usa uma forma de *AdaBoost* mas organiza-o como uma cascata rejeição de nós, onde cada nó é um classificador *multitree AdaBoosted* projetado para ter alta taxa de deteção(por exemplo, 99,9%) e (poucos falsos negativos ou faces perdidas) à custa de uma baixa taxa de rejeição (perto de 50%), ou seja, muitos falsos positivos. Para cada nó, que seja retornado como não pertencente a classe, resulta términos do cálculo, e consequentemente o algoritmo declara que não existe face naquele local. Assim, a deteção da classe é efetuada apenas

se o processamento percorrer toda a cascata. Para instâncias onde a classe é rara (por exemplo, um rosto em uma imagem), cascatas de rejeição podem reduzir muito o tempo de cálculo, porque a maioria das regiões onde é procurada uma face retorna rapidamente como não contendo a classe (face).

3

Requisitos e Arquitetura do sistema

Neste capítulo são apresentados os requisitos de funcionamento e os aspectos gerais do sistema desenvolvido, como identificação de entradas e saídas e a comunicação com o ambiente onde este se insere. É feita também uma breve apresentação das principais ferramentas utilizadas no desenvolvimento e implementação do trabalho.

3.1 Requisitos do sistema

Para cumprir os objetivos estabelecidos na secção 1.2, e facilitar o teste e validação das soluções encontradas, foram definidos os seguintes requisitos para o sistema:

- Sistema modular de forma a ser facilmente identificável funções/algoritmos que implementam uma determinada funcionalidade;
- O sistema deve ser configurável de forma a permitir, facilmente, a alteração das fontes de dados (como câmaras ou comandos), e seleção de algoritmos e outros aspectos relevantes do seu funcionamento;
- O sistema deverá ser capaz de receber e compreender comandos provenientes da rede *Controller Area Network* (CAN) existente na sala;
- Deverá conter pelo menos um algoritmo de identificação e um algoritmo de *tracking* que permite identificar e seguir uma pessoa na sala reuniões LAID GECAD por todo o campo visão das câmaras presentes na referida sala;
- Aplicação desenvolvida deverá ser capaz de ser executada em sistema GNU/linux, com ou sem ambiente gráfico;

- Disponibilizar ferramentas de depuração (*debug*), por exemplo, *output* em vídeo e com ajuste de parâmetros, e facilmente acedível através de interface gráfico;
- O sistema deverá gerar um registo (*log*), onde constem os erros e informações genéricas de funcionamento para apoio ao *debug*.

3.2 Ferramentas de desenvolvimento

Para o desenvolvimento e implementação deste trabalho foram utilizadas diversas ferramentas externas, *hardware* e *software*, as quais foram selecionadas de entre várias existentes com funcionalidades semelhantes. Nesta secção pretende-se identificar essas ferramentas, e justificar a sua escolha. O trabalho desenvolvido, é destinado a máquinas que utilizem como Sistema Operativo (SO) o GNU/Linux. Este sistema operativo, baseado em UNIX é distribuído de forma gratuita por vários grupos, essencialmente de voluntários (exemplos: Debian, openSUSE), que disponibilizam não só as funções básicas de um SO mas também uma vasto conjunto de *software* para as mais diversas funcionalidades (editores de texto, navegador, reprodutores). Um desses *softwares* é o Code::Blocks, que é um *Integrated Development Environment* (IDE) multiplataforma, disponibilizado gratuitamente (GPL) pela Code::Blocks *Team*. Este *software* facilita o desenvolvimento, compilação e *debug* em várias linguagens. Uma dessas linguagens é a $C++$, uma linguagem de uso geral, multiplataforma, que foi a escolhida para o desenvolvimento das aplicações que compõem este trabalho.

3.2.1 *GNU/Linux e GNU/Debian*

O GNU/Linux é um SO que consiste em vários programas fundamentais necessários para executar as funções básicas do sistema: receber instruções dos utilizadores, ler e escrever dados em discos rígidos e impressoras, controlar a utilização da memória, e executar outro *software*. Num sistema GNU/Linux, o Linux é o núcleo do sistema identificado como *kernel*, sendo o restante sistema constituído em programas, desenvolvidos para as mais diversas tarefas e tipicamente sobre licenças livres, muitos dos quais escritos por/ou para o GNU *Project*. O nome GNU/Linux é justificado pelo facto do sistema operativo ser constituído pelas duas partes, apesar de frequentemente ser identificado apenas por Linux.

O que tipicamente é instalado nos computadores que utilizam esta família de SO, é uma das distribuições de GNU/Linux, um sistema completo que para além de Linux inclui outros programas comuns, como ambientes gráficos, processadores de texto e outras ferramentas.

O núcleo Linux tem gradualmente aumentado a sua popularidade, chegando a cada vez mais utilizadores. Apoiado por uma série de pacotes cada vez mais versáteis e estáveis de *software* livre, como: LibreOffice para escritório, projeto GNU de uso geral, programas

para pequenas empresas e interfaces gráficas. As interfaces gráficas, contribuem para esta popularidade estando cada vez mais amigáveis, as mais populares são: *K Desktop Environment* (KDE), *GNU Network Object Model Environment* (GNOME), *Xforms Cool Environment* (XFCE) e mais recentemente *Lightweight X11 Desktops Environment* (LXDE).

Algumas das distribuições Linux mais conhecidas são: Debian, Redhat, openSUSE e Ubuntu, sendo estas usadas como base para a criação de outras, tornando a oferta muito variada, em campos como o aspeto gráfico, requisitos de funcionamento e ferramentas disponibilizadas.

GNU/Debian

O GNU/Debian é resultado da junção entre o *kernel* Linux, as ferramentas GNU e o *Debian Project* [2], deu origem a uma distribuição de *software* única. Esta distribuição é constituída por um grande número de pacotes de *software* (29000 na sua mais recente versão estável). Cada pacote da distribuição contém executáveis, *scripts*, documentação, e informação de configuração, que pode corresponder a uma aplicação ou parte desta.

A funcionalidade que mais distingue o Debian de outras distribuições de GNU/Linux é o sistema de gestão de pacotes. Esta ferramenta permite ao administrador do sistema o controlo completo sobre os pacotes instalados no sistema, incluindo a possibilidade de instalar/remover um único pacote ou atualizar automaticamente todo o SO. É possível ao utilizador gerar pacotes e identificar as suas dependências para, quando este for instalado serem verificadas.

As razões para a escolha do GNU/debian para SO a ser executado no *hardware* alvo, prende-se essencialmente com a politica de poucas versões, em que o *software*/tecnologias disponibilizados são amplamente testados, e também a sua abertura às necessidades e expectativas da comunidade. O GNU/debian sendo uma distribuição de referência, o suporte disponibilizado pela comunidade é mais fiável e rápido que na maioria das distribuições. A documentação e manuais disponibilizados, tornam a instalação e configuração de qualquer *software* uma operação simples.

3.2.2 Code::Blocks

O Code::Blocks é um IDE, ou seja, um assistente integrado de desenvolvimento, que se destina à construção código fonte e à utilização de compiladores, *linkers* e *debugers*. Disponibiliza suporte à utilização de vários compiladores, como por exemplo: *GNU Compiler Collection* (GCC), *Microsoft Visual C++* (MSVC++) e Open Watcom. Para *debug* suporta o GNU *GNU Project Debugger* (GDB) e o Microsoft *Console Debugger* (CDB) mas este ultimo não totalmente.

É distribuído com licença GNU *General Public License 3* (GPL), com a exceção dos

[2] O *Debian Project* é uma organização exclusivamente de voluntários dedicada ao desenvolvimento de *software* livre e a promover os ideais da comunidade de Software Livre

plugins de terceiros, que podem ter qualquer licença. É uma aplicação multi-plataforma com compilações para Linux, Mac, Windows (utiliza wxWidgets), desenvolvida em C++, expansível através de *plugins*. De entre as suas características destaca-se o fácil e rápido *setup* de um projeto, sem necessidade de criação ou atualização de *makefiles*, o suporte para processamento paralelo(utilização de CPU's com múltiplos núcleos de processamento), um espaço de trabalho que combina múltiplos projetos e o suporte de múltiplos compiladores [cod12].

Todas estas características, que facilitam o desenvolvimento de *software*, são os principais fatores para a escolha deste IDE para o desenvolvimento deste trabalho.

3.2.3 Linguagem C++

A linguagem C++ foi desenvolvida inicialmente por Bjarne Stroustrup para a *AT&T*, a partir da linguagem C, tendo como ideia principal a de agregar o conceito de classes, e apresentada inicialmente com nome de "C com classes"[Str94]. O termo "++"sugere que a linguagem C++ é um pouco mais que o C, oferecendo melhoramentos através do suporte de dados abstratos e programação orientada por objetos. A partir da primeira versão de 1983, a linguagem foi evoluindo, tornando-se disponível fora da *AT&T* em 1985, e após um longo processo, foi padronizada pela ISO no final de 1997, pelo padrão *International Organization for Standardization* (ISO)/*International Electrotechnical Commission* (IEC) 14882.

Apesar do C++ tentar manter a compatibilidade com o C, nem todo o programa escrito em C é compilado em C++ sem erros, podendo mesmo gerar resultados diferentes, já que a sintaxe e a semântica de algumas construções diferem. Um programa em C++ reflete elementos tanto do estilo de programação orientada a objetos como da programação procedimental clássica. Isto porque o C++ é na verdade uma linguagem extensível já que se pode definir novos tipos, de tal forma que eles agem de mesmo modo que os tipos pré-definidos que fazem parte da linguagem padrão.

Características que definem o C++:

- Programação Orientada a Objetos: A possibilidade de utilizar programação, com todas as caraterísticas deste paradigma, como possibilidade de reutilizar código de uma forma mais lógica e produtiva;

- Portabilidade: Com compiladores disponíveis para muitos processadores e sistemas operativos, o C++ é uma das linguagens de programação mais utilizadas e portadas;

- Brevidade: Código escrito em C++ é muito menor em comparação com outras linguagens, com o uso de caracteres especiais;

- Programação Modular: Uma aplicação em C++ pode ser construída de vários arquivos de código que são compilados separadamente e "linkados"juntos, esta ca-

racterística permite ao C++ agregar código produzido noutras linguagens, como o Assembler ou C;

- Compatibilidade com C: O código escrito em C pode ser facilmente incluído num programa C++ sem fazer grandes alterações;

- Velocidade: O código resultante de uma compilação C++ é muito eficiente, devido a sua dualidade de linguagem de Alto e Baixo nível e do tamanho reduzido da linguagem em si;

- Linguagem livre: existem compiladores e sistemas operativos que utilizam o padrão ANSI C++, não sendo este desenvolvido ou pertencente a nenhuma empresa, sendo mantido por diversas entidades;

- Independência de plataforma: o C++ é uma linguagem independente de plataformas;

O C++ é uma linguagem de propósito geral, de nível intermédio (utiliza um dialeto de alto nível mas possibilita ao programador facilidades para desenvolvimento em baixo nível). O C++ suporta integração com várias outras tecnologias, podendo ser utilizada para praticamente qualquer finalidade, como web (internet), microcontroladores, comunicação e aplicações distribuídos em ambientes cliente/servidor.

Apesar da aplicação desenvolvida não ser orientada a objetos, a linguagem escolhida para a implementação da aplicação foi o C++, pela possibilidade de juntar código desenvolvido C com alterações mínimas, e para permitir utilizar algumas classes, que facilitam por exemplo, o uso da biblioteca OpenCV ou a manipulação de *strings*.

3.2.4 Hardware *Raspberry pi*

Como apresentado na secção 2.7.2, o Raspberry pi é um computador de dimensões reduzidas, que contém os periféricos essenciais, para a interligação de equipamentos, como USB e LAN. Como sistema operativo, são disponibilizadas várias distribuições de GNU/linux , entre elas, Arch Linux ARM, Debian ARM (Squeeze), Raspbian (debian Wheezy) e Fedora Remix.

Para a escolha deste equipamento, em detrimento de um computador pessoal típico, foram identificados alguns aspetos em que este equipamento é superior. O fator determinante é o preço, estando este disponível a aproximadamente 27 euros, o que desperta o interesse em explorar as suas potencialidades. Outro aspeto que importante na escolha, é a disponibilidade de acesso a vários periféricos ao nível do processador, que facilita a ligação a outras placas presentes na sala em que o sistema se insere. O tamanho pequeno permite flexibilizar a disposição do sistema na sala.

3.3 Algoritmos escolhidos

O OpenCV implementa um detetor de objetos baseado em "características *haar-like*", que é usado nesta trabalho para a deteção de faces. O algoritmo pode ser usado para a deteção de faces em tempo real que o torna adequado para o problema em estudo. Este aspeto e a sua elevada taxa de sucesso foram as principais razões para a sua seleção para a etapa de deteção no problema proposto. Na sequência da etapa de deteção, que apesar de rápida, é inviável ser executada em cada *frame* de vídeo, derivado a alta taxa de *frames* e o seu peso computacional de execução. Torna-se por isso necessário a utilização de uma técnica que permita seguir a face detetada em cada *frame* subsequente. Para a execução dessa tarefa foi escolhido o algoritmo Camshift, disponibilizado pela biblioteca OpenCV para o *tracking* de faces. Como apresentado na secção 2.4.3, o Camshift cria uma distribuição de probabilidade da cor da pele da cara no espaço de cor HSV. Este modelo é adequado para o *tracking* porque separa a cor da saturação e brilho, o que minimiza os efeitos negativos da luz no *tracking*, ou seja, menos sensível ás suas alterações. Outra vantagem do Camshift é que funciona em tempo real, com baixo consumo computacional, libertando assim o *Central Processing Unit* (CPU) para outras operações.

3.4 Arquitetura geral do sistema

Esta secção tem como objetivo, apresentar a organização e interligação do *hardware* e *software* desenvolvidos para a resolução do problema apresentado. É também feita um discrição da sala para a qual o sistema foi projetado.

3.4.1 Sala de reuniões

A sala para a qual o sistema se destina, é um laboratório que implementa um ambiente inteligente para apoio a reuniões (LAID). A sala suporta reuniões distribuídas e assíncronas, para que os participantes possam tomar parte na reunião, onde quer que estejam [CAF+11]. A LAID tem disponível o seguinte *hardware* [MSF+07]:

- Ecrã de plasma Interativo 61";
- Note grabber *Mimio*®;
- Seis ecrãs *Liquid Crystal Display* (LCD) de 26", para 1 a 3 pessoas;
- 3 câmaras, microfones e terminais de ativação controlados por rede CAN.

Este *hardware*, que é capaz de reunir todos os tipos de dados produzidos na reunião facilita a sua apresentação aos participantes.

O sinal de vídeo é adquirido a partir das três câmaras apresentadas na secção 2.7.1. Estas câmaras disponibilizam internamente as funcionalidades rotação (*pan*), inclinação

(*tilt*), disponibiliza também lentes com aplicação de *zoom* ate 18X. A disposição e amplitudes necessárias das câmaras instaladas são apresentadas na figura 3.1.

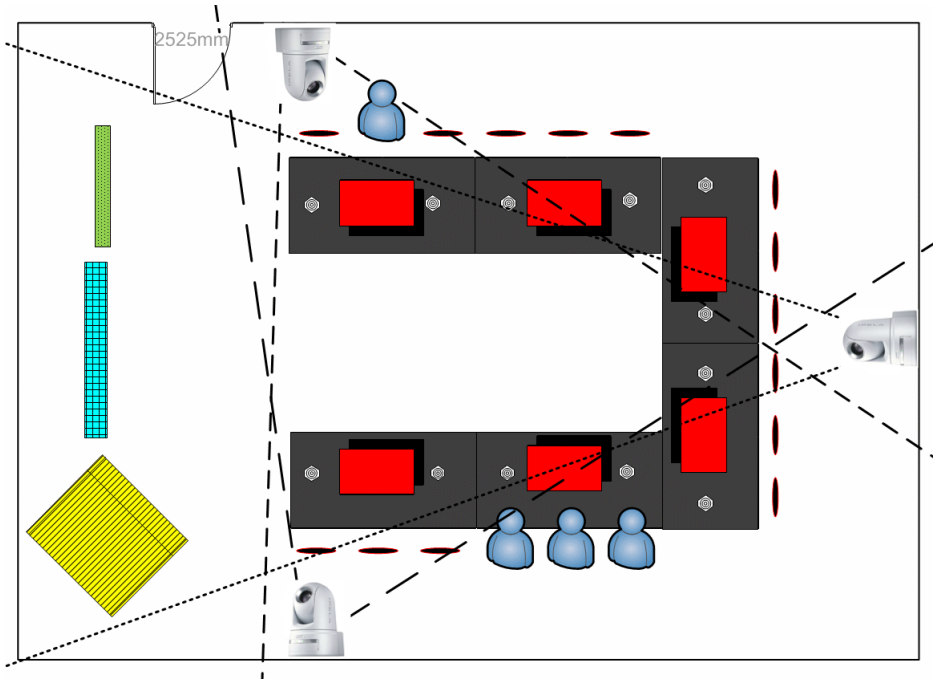

Figura 3.1: *Posicionamento e ângulos de abertura das câmaras instalados no LAID [FMF+12] e posterior edição*

3.4.2 Arquitetura hardware

A arquitetura do *hardware* da solução desenvolvida está representada na figura 3.2. Está dividida em três blocos, "Interface SPI/CAN", "Rpi"e "câmara", de acordo com a sua funcionalidade, e interligados entre eles, como representado na figura 3.2.

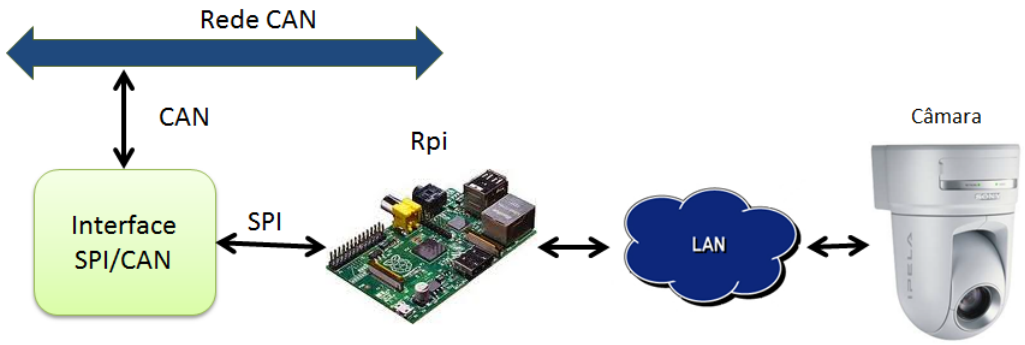

Figura 3.2: *Arquitetura geral do sistema proposto*

Para cumprir os requisitos especificados em 3.1, o sistema necessita receber os comandos provenientes da rede usada para a gestão da sala (rede CAN). Para tal ser possível define-se um bloco que faz a interface entre a rede CAN, e o *Serial Peripheral Inter-*

face (SPI) do bloco central "Rpi". O bloco "Rpi", que tem como tarefa principal executar a aplicação desenvolvida, constituída por os dois processos. A ligação entre o "Rpi"e a câmera, para troca de mensagens, que incluem os *frames* de vídeo e os comandos para obter e alterar a orientação da câmera, é feita recorrendo a uma LAN existente.

3.4.3 Arquitetura do software

O *software* desenvolvido divide-se em dois processos, *Camtrack* e *Rpi_SPI*, em que o segundo é um filho do primeiro, ou seja, o processo *Camtrack* cria uma nova linha de processamento, que substitui pela imagem do *Rpi_SPI*.

O processo *Camtrack* tem como principal objetivo a deteção e seguimento da face alvo, quando solicitado pelo gestor de áudio e vídeo da sala de reuniões. A solicitação do inicio de *tracking* é recebida pelo *Rpi_SPI*, através de um comando enviado pela rede CAN. Este processo monitoriza a comunicação SPI, cujo conteúdo é proveniente da *interface SPI/CAN*, e transmite ao *Camtrack* o conteúdo do comando recebido.

Para a comunicação entre os processos é usada uma arquitetura cliente/servidor, concretizada num *socket SOCK_DGRAM*. O *socket* é criado no inicio de ambos os processos, que permite o envio e receção de comandos entre os mesmos, como demostrado na figura 3.3.

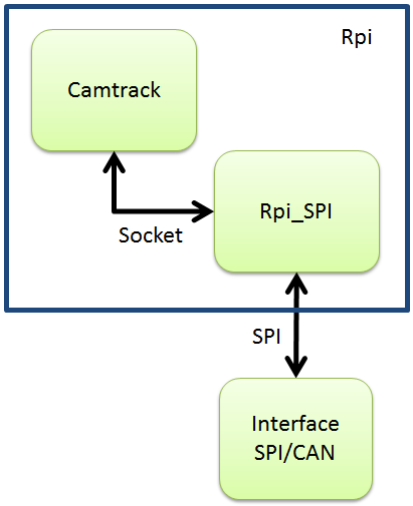

Figura 3.3: *Ligação entre processos e rede*

Modos de funcionamento

O processo *Camtrack* define quatro modos de funcionamento, "debug", "rpi", "dual"e "sleep"que condicionam a execução de funções e atualização de saídas.

- O modo "debug", que apenas é passível de ser usado, quado o sistema operativo disponibiliza a criação de janelas (ambiente gráfico). Neste modo são criadas e

atualizadas três janelas "Deteção", "Backprojection"e "Camshit", que permitem visualizar os resultados dos respetivos algoritmos.

- O modo "rpi", não necessita de ambiente gráfico, e não é criada qualquer janela, sendo a sua única saída a atualização da posição da câmara.
- O modo "dual"é uma combinação do "debug"com "rpi", disponibilizando todas as funcionalidades da aplicação.
- O modo "sleep"que existe minimizar os recursos de processamento, desativa todas as saídas, e coloca a aplicação em *sleep*, estado que sai apenas para verificar a receção de um novo comando.

Entradas e saídas

A solução desenvolvida, que incluiu os dois processos, contempla três fontes de entrada, e disponibiliza três tipos de saída. Na figura 3.4 é apresentado esquema de entradas e saídas da aplicação.

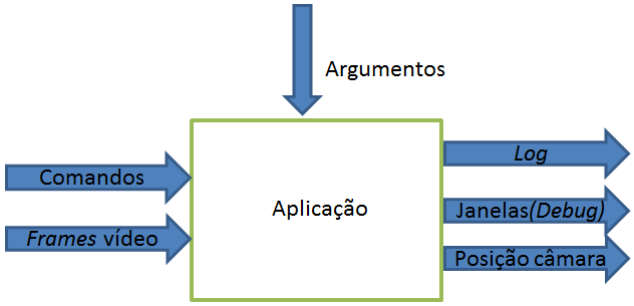

Figura 3.4: *Principais blocos, entradas e saídas da aplicação desenvolvida*

A entrada "*frames* vídeo"representa a aquisição de vídeo digital, disponibilizados a partir de quatro fonte distintas. A fonte "IMAGE_MJEPG", que implementa um pedido de vídeo por *socket* TCP/IP a uma das câmaras presentes na sala. Outra fonte de vídeo disponível é o "ONESHOT_IMAGE"que utiliza a funcionalidade *oneshot image* da câmara e a aplicação externa *wget*, para obter uma imagem JPEG semelhante a disponibilizada no vídeo digital. A biblioteca OpenCV tem disponíveis funções que permitem configurar uma entrada de vídeo a partir de um caminho de origem local ou remoto, e posterior aquisição de *frames* da fonte configurada, fonte "IMAGE_OPENCV". A ultima fonte de vídeo disponibilizada é IMAGE_WEBCAM que permite obter imagens da *webcam* utilizada apenas para *debug*.

A entrada "Comandos", provenientes da comunicação SPI, entre os blocos "rpi"e "Interface CAN/SPI"(figura 3.2). Os comandos têm como destino o processo Rpi_SPI, que implementa a sua leitura e processamento. Estão disponíveis dois comandos: o comando "CAMARA"que identifica qual a câmara, das instaladas na sala de reuniões, que deverá

ser usada para a aquisição do vídeo, e o comando "TRACKING", que permite ativar e desativar o *tracking*, e quais a saídas que o sistema deve gerar.

Os argumentos são os parâmetros passados à aplicação, no momento da sua chamada. Estes são tratados pelo processo *Camtrack* na sua inicialização. Os argumentos que a aplicação pode receber são:

- Entrada de vídeo - permite escolher uma das quatro fontes de vídeo disponíveis;
- Guardar *frames* ON/OFF - Permite ativar ou desativar a funcionalidade que guarda em disco os *frames* obtidos;
- Ficheiro *haar* - identifica qual o ficheiro *haar* que será usado na etapa de deteção;
- Numero Max *Frames* - Número máximo de *frames* que serão guardados em disco. O sistema implementa um *buffer* rotativo, quando atingido o valor máximo começa a substituir os inicias;
- Modo de funcionamento - seleciona um de quatro modos de funcionamento, que identificam o tipo de saídas possíveis. Os modos disponiveis : "debug", "rpi", "dual"e "sleep", apresentados na secção 3.4.3.

Como saídas a aplicação pode gerar três tipos, coordenadas para orientação da câmara alvo, registo de eventos (*log*) e janelas de visualização. Estas saídas podem ser ativadas no arranque da aplicação através da passagem de argumentos. A saída principal, é a orientação da câmara, as restantes são apenas ferramentas de apoio ao *debug* e estudo dos algoritmos implementados.

Na saída "log", quando ativo, escreve num ficheiro .txt, todos os erros e principais eventos da execução da aplicação com identificação da hora, ficheiro e linha de código responsável por esse evento. De entre esses eventos, destacam-se os comandos recebidos e enviados por ambos os processos, e a passagem entre as principais etapas do algoritmo deteção e *tracking*.

Quando em modo "debug"são criadas e atualizadas várias janelas, que contêm informação sobre as principais etapas dos algoritmos de deteção e *tracking*. Estas janelas mostram informações sobre momentos "chave"da execução, como o *frame* em que a face é detetada, ou a secção do *frame* que o algoritmo de *Camshift* identifica como correspondente à face alvo. Esta ferramenta permite também a alteração, momentânea de alguns parâmetros do *tracking*.

Estrutura dos processos

Para a estruturação do processo *Camtrack* defini-se cinco etapas fundamentais (figura 3.5), com funcionalidades distintas entre elas. A primeira identificada como "Inicialização"é executada apenas uma vez no arranque da aplicação, inicializa as estruturas e

variáveis usados nas etapas seguintes. A "Inicialização"é responsável pelo processamento de argumentos recebidos por linha de comando, criação e configuração dos *sockets* e o carregamento dos ficheiros *haarcascade*. A etapa "Aquisição de vídeo", que adquire os *frames* de vídeo para processamento pelas etapas seguintes. A "Identificação", que representa a segunda etapa do processo, é responsável por identificar a posição da face a seguir. A etapa "*Tracking*"identifica a face nos *frames* de vídeo subsequentes, obtendo a sua posição e orientação. A quarta etapa "Orientação câmara", altera a orientação da câmara de forma a manter a face a seguir no centro do seu campo de visão.

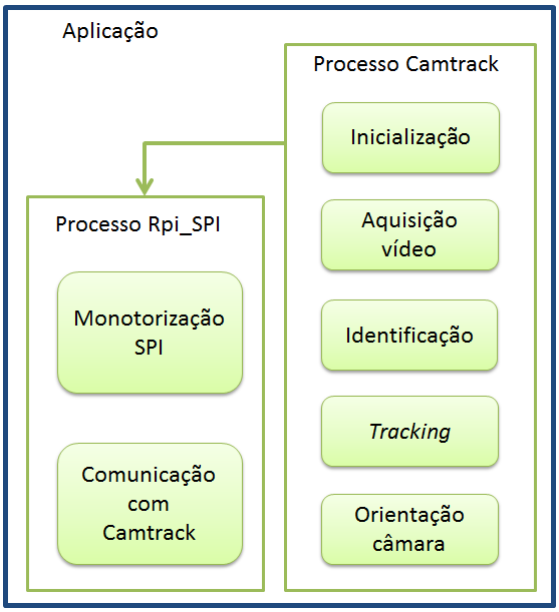

Figura 3.5: *Principais blocos, entradas e saídas da aplicação desenvolvida*

O processo *Rpi_SPI* está estruturado em duas etapas, a "Monotorizaçao SPI"e "Comunicação com Camtrack". A "Monotorização SPI"verifica periodicamente a existência de uma mensagem na rede SPI. Quando é recebida com sucesso uma mensagem da referida rede, esta é tratada e formatada de forma a ser encaminhada para o processo *Camtrack*. O envio da mensagem é efetuado na etapa "Comunicação com Camtrack".

Esta página foi intencionalmente deixada em branco.

4

Implementação

Neste capítulo descreve-se o modo como a aplicação referida no capítulo anterior é implementada. Identificam-se, em primeiro lugar, a estrutura dos dois projetos que constituem a aplicação. Para cada projeto identifica-se a interligação das funções criadas, e qual a sua tarefa no algoritmo global. De seguida descreve-se todos os passos efetuados para a criação dessas funções de modo a satisfazer os requisitos enunciados no capítulo 3.1. O capítulo termina com apresentação dos procedimentos de instalação da aplicação desenvolvida, no *hardware* de destino.

4.1 Estrutura da programação

A programação elaborada divide-se em dois projetos distintos, *Camtrack* e *Rpi_SPI*. O primeiro que implementa os algoritmos de deteção e *traking*, e o segundo a monotorização da comunicação SPI. O código fonte elaborado para cada um dos projetos é composto por ficheiros, com extensões ".cpp"e ".h". Os ficheiros ".cpp"contêm o código fonte das funções implementadas. Enquanto os ficheiros ".h"contêm os *defines* , estruturas e os protótipos das funções existentes no ".cpp"(ver anexo A). Estes ficheiros organizam o código por área de aplicação, de forma a melhor estruturar as diferentes rotinas e obter uma melhor perceção global do projeto. Na Tabela 4.1, apresenta-se a lista dos diferentes ficheiros do projeto *Camtrack*, com uma breve descrição.

Para o projeto *Rpi_SPI*, parte da implementação é semelhante ao *Camtrack*, nomeadamente a implementação das funções que criam e utilizam os *sockets*. Na Tabela 4.2, apresenta-se a lista dos diferentes ficheiros constituintes do projeto *Rpi_SPI*, e a respetiva descrição.

Tabela 4.1: *Lista de ficheiros utilizados na programação de Camtrack*

Nome	Área de aplicação
main	Função *main*(), responsável pelas etapas de mais alto nível do algoritmo. Inicialização, deteção, *tracking* e orientação da câmara
camara	Aquisição de vídeo e movimentação da câmara
rpi	Funções que implementam e tratam as mensagens recebidas do processo *Rpi_SPI*
util	Funções de uso geral, tratamento de *strings* e escrita de ficheiro de texto
sockets	*Sockets* TCP/IP, criação estabelecimento de ligação e comunicação
tracking	Algoritmo de *tracking*

Tabela 4.2: *Lista de ficheiros utilizados na programação de Rpi_SPI*

Nome	Área de aplicação
main	Função *main*(), ciclo de teste de receção de mensagens
rpi	Funções de leitura e escrita no SPI para Raspberry pi, e envio receção de comandos para *Camtrack*
util	Funções de uso geral, tratamento de *strings* e escrita de ficheiro de texto
sockets	*Sockets* TCP/IP, criação estabelecimento de ligação e comunicação

4.2 Funcionamento geral Camtrack

O processo desenvolvido para a deteção e *tracking* de faces (Camtrack), tem como principal objetivo, quando solicitado disponibilizar a funcionalidade de *tracking*. Esta funcionalidade deve estar disponível enquanto o sistema está ligado (reunião a decorrer). Assim o processo criado não deverá terminar quando uma tarefa de *tracking* termina, mantendo-se a espera de uma nova tarefa.

4.2.1 Configurações Camtrack

O processo *Camtrack* permite ser configurado, de forma a modelar o seu funcionamento. As variáveis que contêm as configurações do sistema, estão agregadas numa estrutura do tipo $CONFIG$. Estas configurações podem ser alteradas de duas formas. No inicio pela leitura de argumentos, como apresentado na secção 4.3.3. Ou no decorrer do processo na recepção de comandos, como apresentado na secção 4.7.1.

A tabela 4.3 apresenta o tipo de dados, o nome da variável e o seu valor de *default* para cada campo da estrutura $CONFIG$.

4.2.2 Etapas de funcionamento do Camtrack

No mais alto nível o processo *Camtrack* está dividido em cinco etapas: inicialização, aquisição de vídeo, deteção, *tracking* e orientação da câmara, como apresentado na secção

Tipo de dados	Nome da variável	Valor *Default*
char ∗	*ipcamara*	Endereço Ip da câmara em utilização
unsigned char	*entrada_video*	Seleção de uma das fontes de vídeo disponíveis
unsigned char	*guardar*	Ativa/desativa o guardar dos *frames* de vídeo processados nas etapas de deteção e *tracking*
int	*num_max_frames_save*	Número máximo de *frames*, guardados em disco, quando guardar está ativo;
unsigned char	*mover_on_off*	Ativa/desativa movimento da câmara
unsigned char	*tipo_haar*	Seleciona o ficheiro *haarcascade* a usar na etapa de deteção
unsigned char	*modo_funcionamento*	Indicação do modo funcionamento atual do sistema

Tabela 4.3: *Variáveis que constituem a estrutura de configurações*

3.4.3. Algumas das etapas enunciadas, são também elas divididas em várias tarefas. A figura 4.1 apresenta as tarefas principais, implementadas por cada uma das etapas enunciadas.

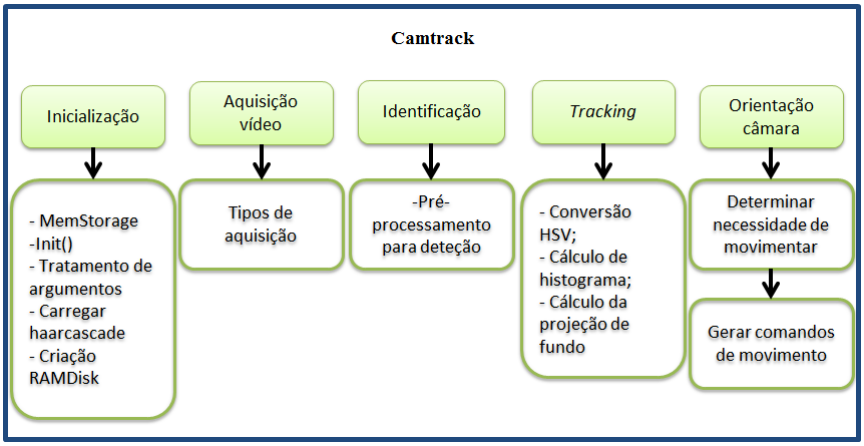

Figura 4.1: *Etapas do funcionamento de Camtrack*

Na figura 4.2, está apresentado o diagrama do algoritmo implementado pelo processo *Camtrack*, que especifica as condições para a transição entre as etapas.

Nas secções, 4.3 a 4.6, é descrito em pormenor os procedimentos e funções implementadas para cada uma das etapas da aplicação *Camtrack*, para melhor compreender o seu funcionamento .

4.3 Inicialização

A etapa "Inicialização"contempla as tarefas que são executadas apenas uma vez, no inicio da execução do processo. É constituída pelas tarefas: MemStorage, init(), carregar

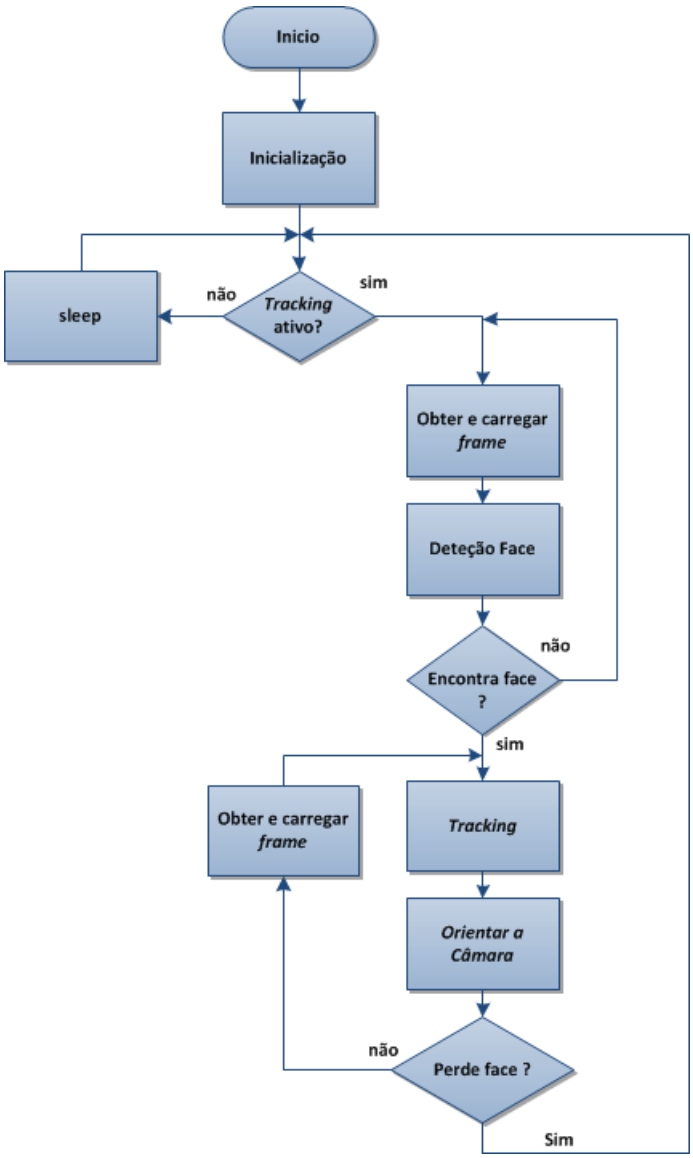

Figura 4.2: *Funcionamento geral do processo Camtrack*

haarcascade, tratamento de argumentos e criação de *RAMDisk*.

A tarefa *MemStorage* consiste na criação de um espaço em memória Random-Access Memory (RAM) para armazenar dados dinâmicos. A função *init*() é responsável por inicializar as variáveis que contêm as configurações do processo. A tarefa carregar *haar-cascade* copia para memória o ficheiro *Extensible Markup Language* (XML) de treino para a deteção de faces. O tratamento de argumentos interpreta os parâmetros passados ao processo a quando da sua chamada. A etapa inicialização termina com a criação de um *RAMDisk* que consiste num sistema de ficheiros em memória RAM.

4.3.1 MemStorage

A primeira inicialização consiste na chamada da função $cvCreateMemStorage()$ uma função da biblioteca OpenCV que reserva um espaço de memória (*Memory storage*). Esta função cria uma estrutura de baixo nível usada para armazenar as estruturas de dados que crescem dinamicamente, tais como sequências, contornos, gráficos, etc. No caso é usada pelo "classificador *haar*", para guardar os resultados da pesquisa. Isto porque o número de faces que podem ser identificadas é desconhecido no início do algoritmo, assim como o tamanho do vetor que irá guardar a localização dessas faces.

4.3.2 Init()

Esta função é responsável pela inicialização de variáveis, com os seus valores de *default*, referentes às diversas configurações que o sistema permite. Os valores de por omissão (*default*) estão definidos em ficheiros *.h* de acordo com a área de aplicação dessa variável. São também inicializadas algumas variáveis de uso geral, que não influenciam diretamente o funcionamento da aplicação, como o nome das janelas de *debug*.

4.3.3 Tratamento de argumentos

Para a leitura e identificação dos argumentos recebidos via linha de comando, foi criada a função $process\_command\_line()$. Esta função recebe por parâmetro o $argc$ (numero de argumentos recebidos), $**argv$ (matriz que contém os argumentos) e um apontador para estrutura de configurações.

Cada elemento de $argv$, é constituído por três elementos. Começa com o caractere '-' seguido do identificador do argumento, um caractere opcional(diferente para cada argumento) e pelo respetivo valor (alfanumérico). A função *getopt* () é chamada repetidamente, e retorna cada um dos caracteres que identificam o argumento e respetivo valor, a função $process\_command\_line()$ faz corresponder o valor a configuração alvo.

4.3.4 Carregar haarcascade

Outra operação importante na inicialização é carregar o ficheiro *haarcascade* em memória. Esta tarefa depende do tratamento de argumentos que identifica qual o ficheiro que deverá ser carregado, de entre os vários disponíveis.

A função $load\_cascades()$ carrega o ficheiro XML *haarcascade*, usado para a deteção de face. O OpenCV fornece alguns exemplos de ficheiros de treino, fornece também funções para que outros possam ser treinados. Assim a função $load\_cascades()$ disponibiliza a possibilidade de escolher qual dos ficheiros previamente selecionado será usado na etapa de deteção.

Para além da deteção de face, permite também a seleção de ficheiros treinados para outras deteções como a parte superior do corpo ou corpo inteiro. o ficheiro é carregado

para memória através da função *cvLoad*(). Esta função da biblioteca OpenCV, abre e lê o conteúdo do ficheiro indicado, libertando depois da leitura todos os recursos utilizados na operação.

4.3.5 Criação RAMDisk

Para minimizar o tempo de leitura e escrita de ficheiros em disco, durante a aquisição de vídeo, é implementado um sistema de ficheiros temporário, em memória RAM, através da utilização do $tmpfs$.

A criação deste sistema de ficheiros é implementada na função *criar_ramdisk*().Esta função também verifica os requisitos para a sua criação e utilização, retornando eventuais erros durantes essas verificações. A figura 4.3 apresenta o fluxograma de *criar_ramdisk*().

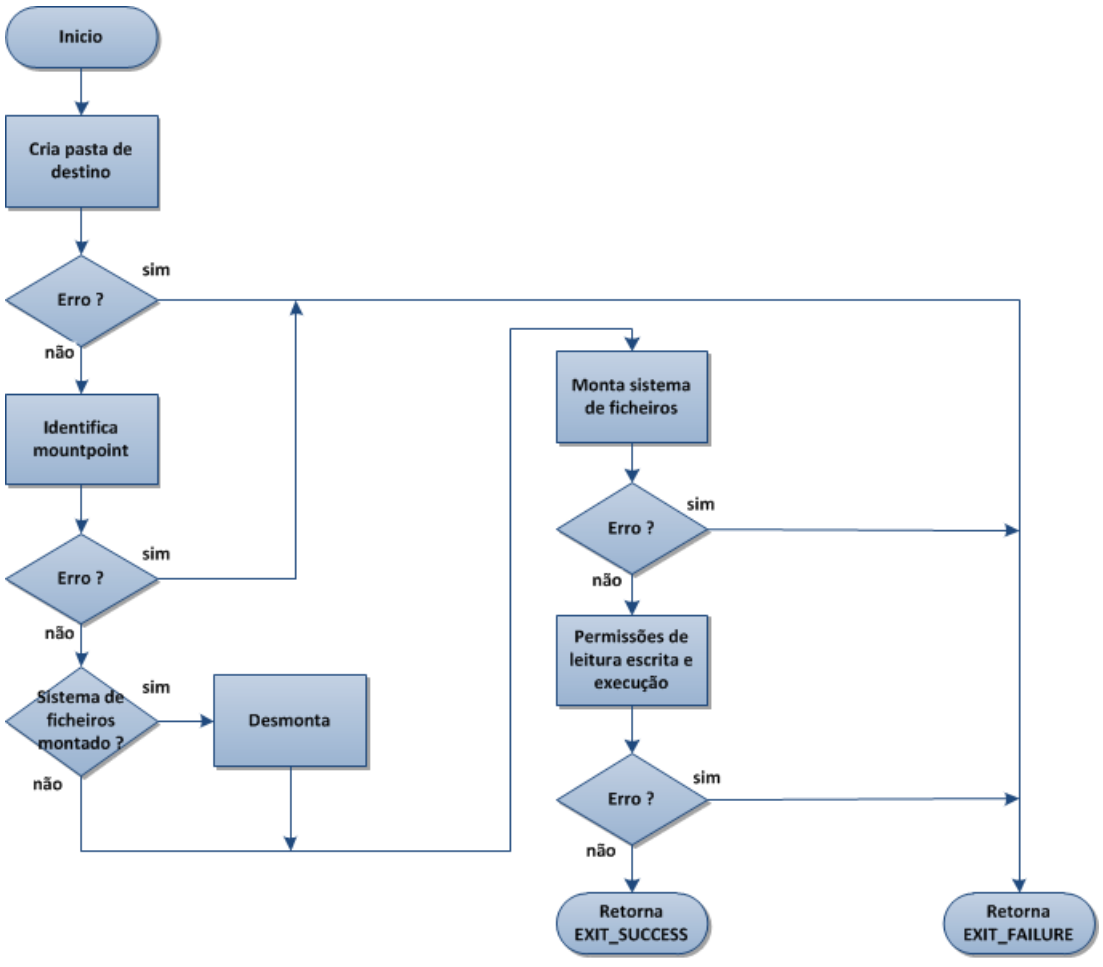

Figura 4.3: *Fluxograma de criação do sistema de ficheiros em RAM*

Para a utilização do $tmpfs$ é definido o tamanho de caminho pretendidos para o sistema de ficheiros.

4.4 Obter frame de vídeo

Para a execução de operações sobre *frames*, estes têm de ser carregadas para memória, para uma estrutura do tipo $IplImage$. Esta estrutura armazena, além de outros dados, o conteúdo da imagem digital, dimensões da imagem e número de canais.

O *frame* é carregado para memória, através da chamada da função da biblioteca OpenCV $cvLoadImage()$. Esta operação é efetuada em dois pontos destintos da execução, o primeiro para deteção da face e o segundo durante o *tracking* com o Camshift.

Um aspeto importante na obtenção continua de *frames* de vídeo, é que as imagens carregadas para memória (pela função $cvLoadImage()$), irá causar erros de memória (*memory leak*). Mesmo quando usada a mesma estrutura $IplImage$, se os recursos de memória não forem reescritos ou libertados no fim de cada etapa do *loop*. A acumulação de *frames* JPEG em memória inevitavelmente causará o *crash* do sistema. Este problema é evitado através da utilização da função $cvReleaseImage()$ no final da utilização do *frame* adquirido.

4.4.1 Entradas de vídeo

A eficiência na etapa de aquisição vídeo, avaliada pelo tempo de aquisição e resolução do *frame*, é essencial para um algoritmo de deteção e *tracking* ser bem sucedido. Foram desenvolvidas várias formas de obter o vídeo digital a partir das câmaras instaladas, de forma a avaliar a sua eficiência no cenário em estudo.

A câmara de rede disponível utiliza a infraestrutura de rede existente no edifício onde a sala de reuniões se insere. Aceita comandos CGI, para pedidos de vídeo, através da utilização de *sockets* TCP/IP (ver secção 2.7.1). Mas também permite obter um *print* do cenário atual, através da funcionalidade "oneshotimage"que disponibiliza uma imagem JPEG sem *frame rate* definido. Para além destas possibilidades a biblioteca OpenCV também disponibiliza funções para a obtenção de *frames* de câmaras IP.

Assim nesta secção são apresentadas as implementação para cada uma das possibilidades de aquisição de vídeo da câmara $Sony\ Ipela\ SNC-RZ25P$ e para *webcam* (usada apenas para *debug*). Na figura 4.4 é apresentada a correspondência entre as câmara e métodos disponíveis para aquisição.

A função $get\_frame\_image()$, seleciona a fonte de vídeo de acordo com o configurado.

TCP/IP socket

A aquisição de vídeo digital no formato MJPEG, via comandos CGI, pode ser solicitada através do envio do comando "/image"ou "/mjpeg"para a câmara, quando está configurada no modo JPEG.

O conteúdo do vídeo é enviado, sequencialmente, em resposta ao primeiro comando "GET"recebido, sendo os *frames* separados pela *string* --myboudary".

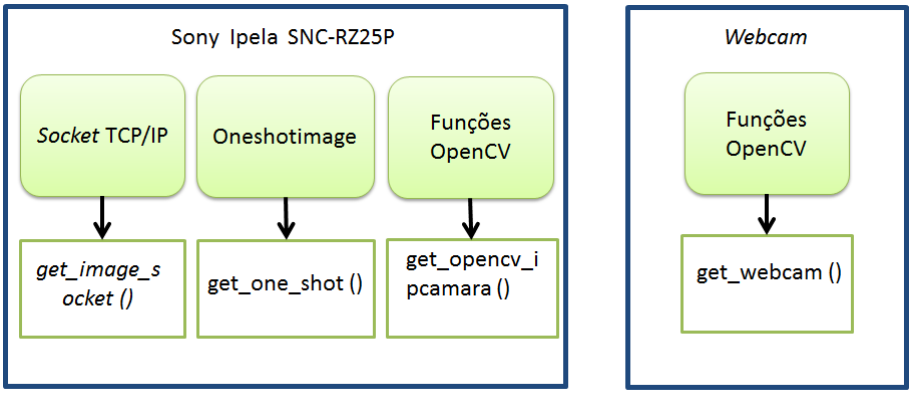

Figura 4.4: *Fontes de vídeo e respetivas funções*

Para extrair o conteúdo de cada *frame* individual, da resposta , foi implementada a função *get_image_socket*(), cujo o fluxograma de funcionamento, é apresentado na figura 4.5.

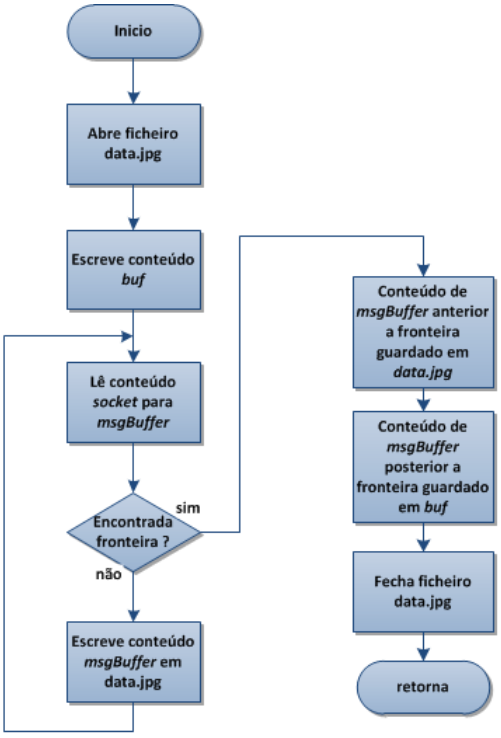

Figura 4.5: *Fluxograma de funcionamento da função get_image_socket*()

Esta implementação obriga a excluir o primeiro *frame* da resposta, devido ao método utilizado para a deteção da *string* de separação. Quando na presença do cabeçalho inicial da resposta *http*, o método de deteção do separador não funciona corretamente. Isto porque é assumido que encontrado o separador iniciado por: "$\backslash r \backslash n - -myboudary \backslash r \backslash n \backslash r \backslash n$" e que termina com: "$\backslash r \backslash n \backslash r \backslash n$". O conteúdo seguinte pertence a um *frame* de video,

o que no primeiro *frame* não é verdade, tornando-o inutilizável.

Os *frames* são lidos do *socket* para um *buffer* de comprimento 10000 bytes. Quando o *buffer* contém dados de dois *frames* destintos. Os conteúdos lidos até a fronteira são guardados no *frame* atual e os posteriores ficam em memória para serem escritos no *frame* seguinte. A figura 4.6 exemplifica a situação descrita anteriormente em que o *buffer*, contém a fronteira e dados de dois *frames* distintos.

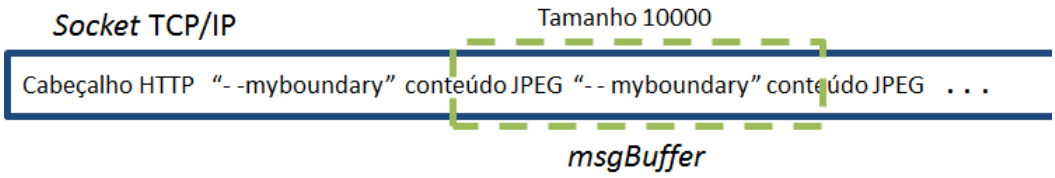

Figura 4.6: *Exemplificação do conteúdo do buffer msgBuffer*

Oneshotimage

Quando a entrada de vídeo "oneshotimage"é selecionada, o sistema adquire a imagem JPEG disponibilizada pela câmara. A imagem é obtida no *Uniform Resource Locator* (URL) $http://ip\_adr/oneshotimage.jpg$, com o mesmo tamanho, qualidade e cor que a disponibilizada como vídeo digital. Para a sua obtenção foi desenvolvida a função *get_one_shot* () que executa o comando *wget* com o URL apresentado, e especificado o caminho de destino da imagem. A função *system*() que permite a execução de um comando na *Shell sh*, foi usada para permitir a chamada do *wget* a partir do código fonte(C++). Esta operação dá origem a um novo processo que termina com o final do descarga (*download*) da imagem.

Opencv Webcam

Para trabalhar com vídeo a biblioteca OpenCV disponibiliza uma serie de funções que permitem adquirir vídeo com as mais diversas codificações e fontes. A figura 4.7 apresenta a sequência de passos necessários para obter um *frame* através da utilização destas funções.

O primeiro passo para ser possível adquirir vídeo digital é a inicialização de uma estrutura *CvCapture*, que contém as informações da fonte alvo, inicialização essa que é efetuada por uma de duas funções:

$CvCapture * cvCreateFileCapture(const\ char * filename)$;

$CvCapture * cvCreateCameraCapture(int\ index)$; Ambas retornam um apontador do tipo *CvCapture*, em caso de sucesso, e $NULL$ caso ocorra algum erro. O erro surge, tipicamente, de duas fontes, *(i)* a câmara ou ficheiro que se pretende usar, não esta disponível ou não é possível ser aberto, e *(ii)* a codificação do vídeo é desconhecida.

A função *cvCreateFileCapture* () recebe como parâmetro o URL do MJPEG ($http://ip\_adr/mjpeg$) abre e prepara a leitura desse ficheiro e retorna, se a abertura for bem

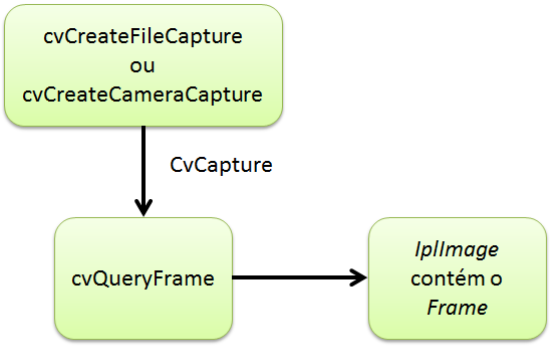

Figura 4.7: *Aquisição de vídeo pelas funções do OpenCV*

sucedida, um apontador inicializado do tipo $CvCapture$, permitindo o início da captura de *frames*. A função $cvCreateCameraCapture$ () tem um funcionamento idêntico a $cvCreateFileCapture$ (), com a exceção da necessidade de conhecer a codificação da fonte de vídeo, uma vez que esta adquire o vídeo do identificador de dispositivo gerido pelo SO. Para o caso de estudo, foi implementado um exemplo que usa a primeira câmara (identificador 0), que tipicamente representa a *webcam*.

Após a inicialização da fonte de vídeo, e para obter o *frame* desejado, é usada a função $cvQueryFrame()$. Esta função tem como parâmetro a estrutura $CvCapture$ previamente inicializada, e retorna um apontador $IplImage$ que contém as informações da imagem.

As funções *get_opencv_ipcamara* () e *get_webcam* (), que implementam a aquisição de vídeo para a câmara *Sony Ipela* $SNC-RZ25P$ e *webcam* respetivamente, utilizam estas duas funções, com os respetivos testes de retornos, para identificação e correção de falhas.

4.5 Deteção de faces

Na etapa de deteção de face, é implementado um ciclo que recebe um *frame* de vídeo e aplica o algoritmo de deteção, terminando apenas quando uma cara é detetada, ou quando é cancelado o pedido de *tracking* da face.

A função $face\_detect\_output()$ que implementa esse ciclo, e cujo fluxograma de funcionamento é apresentado na figura 4.8, é constituída por três etapas fundamentais: aquisição de vídeo, deteção de face, colocação de valores nas saídas.

A etapa de aquisição do vídeo já especificada na secção 4.4.1, resulta no preenchimento de uma estrutura do tipo $IplImage$ que é passada como parâmetro à função que implementa o algoritmo de deteção $detect\_face()$.

Após a execução do algoritmo de deteção, mesmo que este não seja bem sucedido, o *frame* obtido é guardado em disco e impresso na janela de deteção (se as respetivas configurações estiverem ativas). O espaço de memória ocupado pelo *frame* é libertado em seguida.

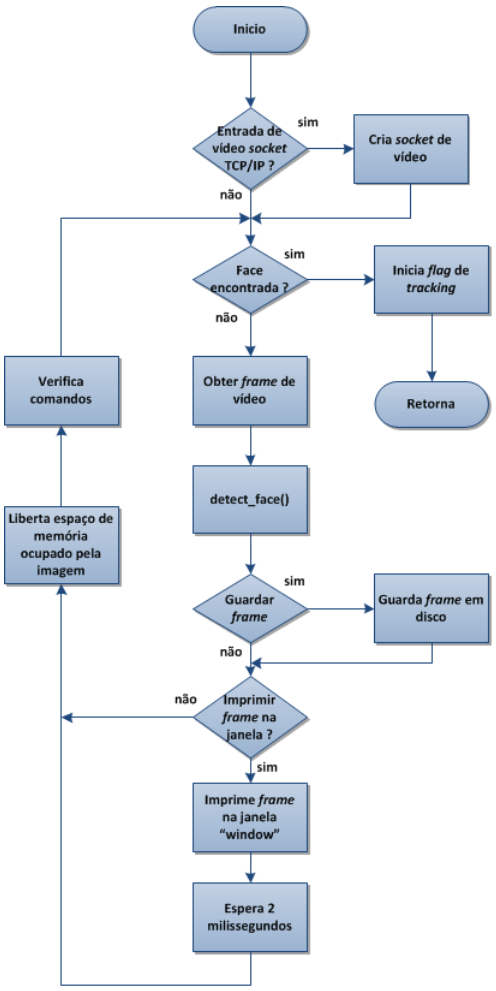

Figura 4.8: *Fluxograma função face_detect_output()*

4.5.1 Pré-processamento para deteção de faces

Antes de ser executada a deteção propriamente dita, para que seja possível usar o algoritmo, são necessários executar duas operações sobre a imagem obtida: a conversão em tons de cinzento e a equalização de histograma. A figura 4.9 apresenta o fluxograma da função *detect_face()*, onde estes e os restantes passos necessários à deteção de todas as faces presentes no *frame*, são apresentados.

Na etapa de conversão em tons de cinzento, para que o *frame* de origem não seja alterado, é necessário o uso de uma nova estrutura *IplImage*, com as mesmas características que as do *frame* original, com exceção do número de canais, que nesta é de apenas 1.

A conversão é efetuada pela função *cvCvtColor()*, que preenche a nova estrutura *IplImage* com a imagem convertida.

A ultima operação sobre a imagem antes da deteção é a equalização de histograma, apresentado na secção 2.2.1, e implementado no OpenCV pela função *cvEqualizeHist()*

Para a deteção das faces presentes num *frame*, é usado o detetor proposto por Viola-

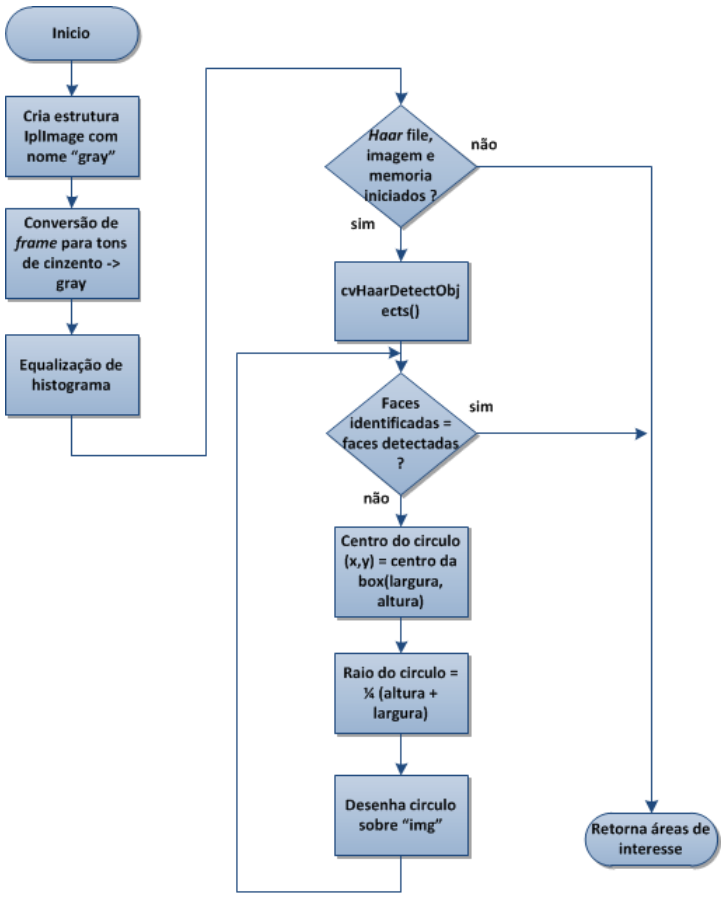

Figura 4.9: *Fluxograma função detect_face()*

Jones, com os melhoramentos propostos por Lienhart-Maydt. Este algoritmo apresentado na secção 2.5 é implementado na biblioteca OpenCV pela função $cvHaarDetectObjects()$, que necessita dos seguintes parâmetros :

- *image* - recebe a estrutura $IplImage$ que contém os dados pré-processados do *frame* adquirido;
- $CvHaarClassifierCascade$ variável * Cascade, contém os dados do arquivo XML carregado para memória anteriormente(ver secção 4.3.4);
- $CvMemStorage$ - variável *Storage*, *buffer* de memória dinamicamente expansível, usado para guardar a sequência de faces identificadas;
- *min_size* - define o menor tamanho de janela para a pesquisa da face, o valor *default* é o valor para o qual *haarcascade* está treinado. Um aspeto importante para minimizar o tempo de processamento da deteção, é considerar a resolução da imagem para a escolha deste parâmetro, dado que a resolução influência muito o valor ideal, tipicamente um bom valor é 1/4 das dimensões do *frame*.
- *min_neighbors* - o detetor *Haar* quando na presença de uma só face, gera múltiplos

casos positivos sobrepostos. Quando apenas uma deteção é efetuada provavelmente corresponde a um falso positivo. Para definir a fronteira entre o falso positivo e uma face, é implementado o parâmetro $min\_neighbors$. Este parâmetro indica o número de deteções sobrepostas necessárias, para ser considerada uma face corretamente detetada. O valor de definido é três, assim quando encontrados três ou mais faces sobrepostas essas são agrupadas é retornado o valor médio da sua localização.

- $scale\_factor$ - identifica a rapidez no aumento da escala de pesquisa, para cada passagem, do algoritmo. Quanto maior o seu valor, mais rápido se torna a execução do algoritmo, minimizando também a eficácia na deteção de faces, o valor de definido é 1.1, que representa uma aumento de 10% a cada passagem.

Esta função retorna um apontador para uma estrutura $CvSeq$ no caso da deteção do objeto ser bem sucedida, ou 0 caso contrário. A $CvSeq$ contém a sequência de retângulos nos quais é provável conter o objeto procurado, no caso faces.

4.6 Tracking

O algoritmo de *tracking*, é responsável pelo seguimento da face detetada na etapa anterior, identificando em cada *frame* a sua posição e orientação. Para tal é implementado o código para o acompanhamento de uma face, de forma eficiente. É seguida na sequência de *frames* a melhor correspondência para a face inicialmente identificada. O acompanhamento de uma face, é tipicamente mais complicado que o acompanhamento de um objeto de cor bem definida, devido à sua forma irregular, e ao movimento da pessoa que a gira e inclina.

O algoritmo ,Camshift, usa informações de cor mas, em vez de depender de uma única cor acompanha uma combinação de cores. Uma vez que o seguimento é por cor, pode acompanhar um rosto através de mudanças de orientação que o detetor de *Haarcascade* (ver secção 4.5) não suporta.

Para o cálculo do Camshift é necessário, a execução prévia de quatro tarefas, pela ordem apresentada:

1. Aquisição de vídeo;
2. Conversão HSV;
3. Cálculo de histograma;
4. Cálculo da projeção de fundo;

No diagrama da figura 4.10, estão identificadas estas tarefas a que ordem como são executadas.

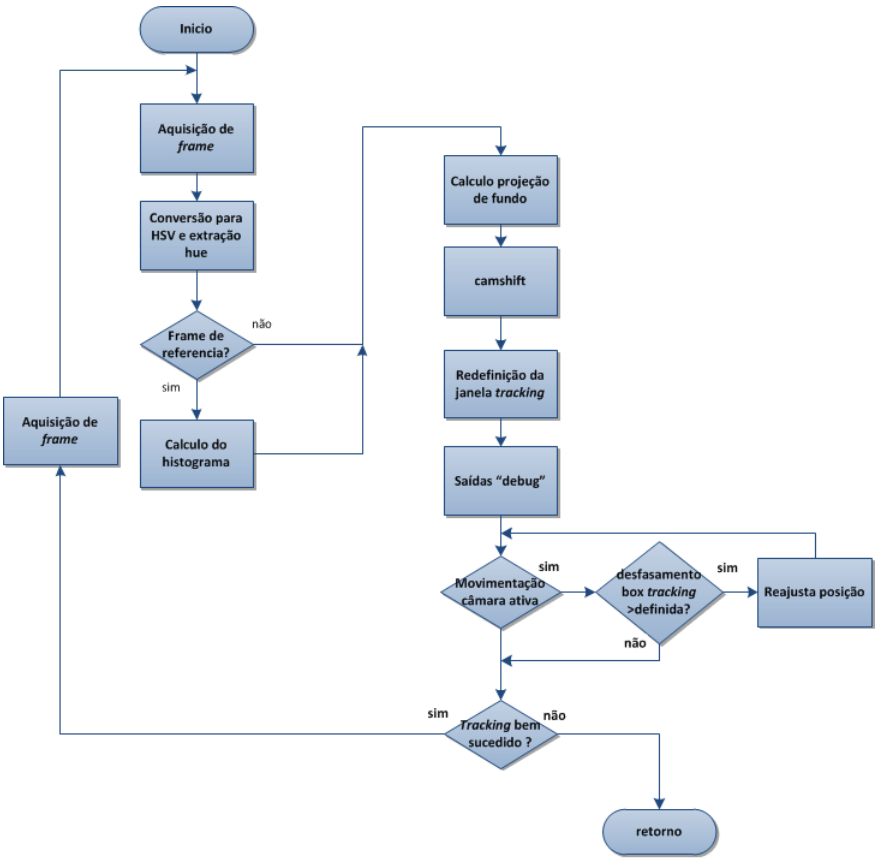

Figura 4.10: *Etapas para o algoritmo de tracking*

A primeira tarefa, como já acontecia no algoritmo de deteção é a captura do *frame* da câmara de vídeo. Esta tarefa é especificada em pormenor na secção 4.4. As restantes tarefas são apresentadas, pela ordem de execução, nas secções 4.6.1, 4.6.2 e 4.6.3.

4.6.1 Conversão para HSV

A segunda tarefa para a preparação do calculo do Camshift é "Conversão para HSV". Esta tarefa executa duas operações, sobre o *frame* de entrada, *(i)* a conversão de BGR para HSV, *(ii)* o filtro dos valores dos pixeis no modelo de cor HSV.

Depois de efetuada a conversão de BGR para HSV pela função $cvCvtColor()$ é introduzido uma operação de afinação do algoritmo de *tracking*. São definidos três valores que excluem os *pixels* neutros (baixa saturação e brilho), da imagem obtida . Quando usado o canal *hue* da cor para o *tracking*, é importante ter também em consideração o nível de saturação e brilho dos *pixels*, isto porque, se forem baixos, a informação do canal *hue* torna-se ilegível e instável. Para tal são definidos três parâmetros o $Smin$, $Vmin$ e $Vmax$. Em que $Smin$ define o valor mínimo do canal S (saturação) , $Vmin$ e $Vmax$ respetiva-

mente o valor mínimo e máximo, do canal V(brilho). O parâmetro $Vmax$, é usado para definir um limite para *pixels* que são muito claros, mas o $Smin$ já elimina esses *pixels*, podendo assim este ser desprezado na afinação.

A figura 4.11 são apresentados os parâmetros que definem cada um dos limites para cada canal, e em que etapas são usados.

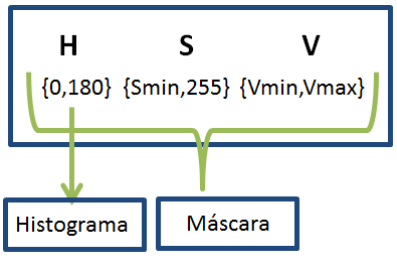

Figura 4.11: *Limites de cada canal de cor HSV*

Para aplicar estes limites à imagem, é criada uma máscara, pela função $cvInRangeS()$ que verifica se os *pixels* da imagem convertida se encontram dentro dos valores configurados. Esta operação preenche a máscara com $0XFF$ se o valor da fonte se encontrar dentro dos valores definidos, e com 0 se estiver fora.

Como para o *tracking* so é usado o canal de cor *hue*, é necessário separa-lo dos restantes, utilizando a função $cvSplit()$.

4.6.2 Cálculo do histograma

No cálculo de histograma, a primeira operação é a seleção da área de interesse no canal *hue* da imagem e na mascara, obtida pelo retorno do algoritmo de deteção. A utilização da máscara previne a inclusão dos *pixels* excluídos por estarem fora dos limites definidos. Para a posterior utilização do histograma no cálculo da projeção de fundo, é necessário que este esteja normalizado. A normalização consiste em redimensionar as barras do histograma através da utilização de um fator, de forma a que o seu somatório seja o valor predefinido, no caso 255, efetuada pela função $cvNormalizeHist()$ da biblioteca OpenCV.

O cálculo do histograma é feito apenas uma vez no ciclo de inicialização do *tracking* (primeiro *frame*), para servir de referência na sequência de *frames* que o segue.

4.6.3 Projeção de fundo

A ultima tarefa de preparação para o cálculo do Camshift é a projeção de fundo. Este cálculo é efetuado pela função $calcBackProject()$. Dada uma área de uma imagem com um determinado objeto, o histograma dessa área pode ser visto como uma função da probabilidade de um dado *pixel* pertencer ao objeto especificado, no caso em estudo a probabilidade da face.

A função $hist\_track()$, calcula a projeção de fundo para cada *frame*, utilizando como parâmetros o canal *hue* do *frame* de entrada, e o histograma de referência. Ao resultado

da projeção de fundo é aplicada a máscara calculada na secção 4.6.1. O resultado é uma imagem, em que cada *pixel* é representado pela probabilidade de pertencer à face, em que os *pixels* mais próximos de preto são menos prováveis, e os brancos os mais prováveis (figura 4.12).

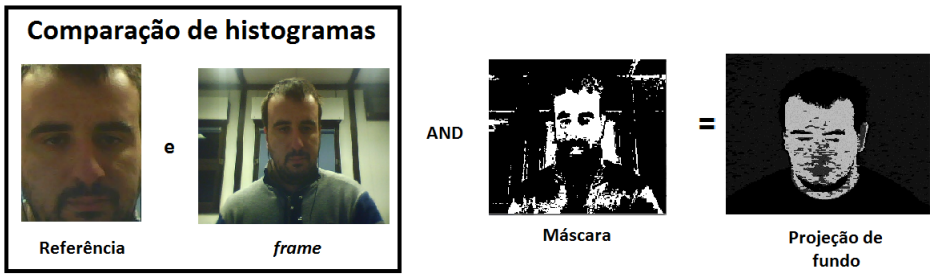

Figura 4.12: *Cálculo projeção de fundo*

4.6.4 *Camshift*

A implementação do algoritmo Camshift, cujo funcionamento teórico foi apresentado na secção 2.4.3, é feito pela função $cvCamShift()$ da biblioteca OpenCV. Esta função é usada no *tracking* para encontrar o centro, tamanho e orientação do objeto (face) e necessita para o seu cálculo dos seguintes parâmetros:

- *prob_image* - A projeção de fundo do histograma da face;
- *window* - A janela inicial de busca, iniciada com a saída do algoritmo de deteção e atualizada em cada execução do algoritmo;
- *criteria* - O critério aplicado para terminar a execução do algoritmo *meanshift*, foi escolhido uma combinação de dois critérios: número de iterações e mínimo de precisão de convergência, terminando assim que um deles é alcançado;
- *comp* - A estrutura de saída, do tipo $CvConnectedComp$, que contém as coordenadas da janela convergida;
- *box* - Estrutura do tipo $CvBox2D$, que define a posição e orientação da face;

***Trackbox* e Movimentação**

O parâmetro de saída do algoritmo de Camshift, fornece o posicionamento e orientação da face no *frame* atual. Para a atualização da posição da câmara é usada a localização do centro e tamanho da janela de *tracking*. O objetivo é ajustar da melhor forma a janela de *tracking* no campo de visão da câmara, através das funcionalidade da câmara: rotação (*pan*), inclinação (*tilt*) e *zoom*, com o foco automático ativo (*auto* foco).

A função *move_camera_tracking*(), verifica a necessidade de movimentar a câmara. Para tal é efetuada a comparação entre o posicionamento do centro da janela *tracking*, com o centro do campo de visão da câmara. Se a diferença entre eles for 0 (+ − tolerância) significa que a janela *tracking* está centrada, caso contrário é necessário orientar a câmara. Para minimizar o número de movimentos da câmara é definida um tolerância de cerca 1/10 do tamanho do *frame*, figura 4.13.

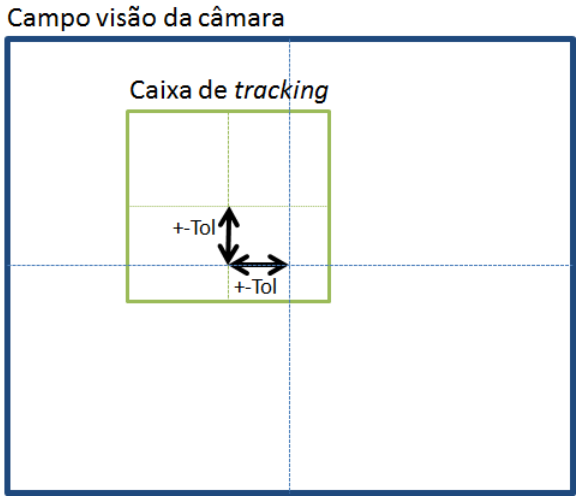

Figura 4.13: *Representação do posicionamento da janela tracking no campo de visão da câmara*

Quando é necessário orientar a câmara é calculada a área de *zoom* para a qual a câmara se deve configurar. As coordenadas do centro da área de *zoom*, (x,y) são obtidas da diferença entre as coordenadas do centro da imagem (fixas, e dependentes do tamanho da janela de entrada), e as coordenas atuais da janela de *tracking*. As dimensões da janela de *zoom* são dadas pela largura da janela de *tracking*. Para que não seja efetuado um *zoom* demasiado grande na face, as dimensões da janela de *tracking* (largura e altura) são multiplicadas por duas vezes a tolerância.

Perda da face no tracking

O Camshift retorna sempre uma janela que define a melhor correspondência à face, mesmo que a face não esteja presente no *frame*, o algoritmo retorna uma janela de *tracking*. Assim é necessário a definição de um critério para determinar quando a face é "perdida"durante o *tracking*. A definição da perda da face é dada pelo tamanho da janela de *tracking* retornada, dado que se a face não está presente no *frame*, não existe uma área muito provável de lhe pertencer. O valor retornado é uma área muito pequena. Quando a janela de *tracking* não tem as dimensões mínimas, é feita uma operação de *zoom out* na tentativa de enquadrar novamente a face no campo de visão da câmara, se mesmo assim o tamanho da janela não aumentar, o *tracking* é terminado e o algoritmo volta a etapa de deteção.

4.7 Rpi_SPI

O processo *Rpi_SPI* tem como objetivo principal a monotorização do periférico SPI. Quando é detetada uma mensagem, processa o seu conteúdo e informa o *Camtrack*. O envio de informação é feito através da utilização dos comandos definidos. A implementação deste processo não foi concluída, tendo sido criadas as condições para a comunicação com o processo *Camtrack* mas não a monotorização da comunicação SPI. Assim a comunicação SPI está simulada em consola, com os comandos introduzidos pelo utilizador. A figura 4.14 representa o funcionamento atual do processo.

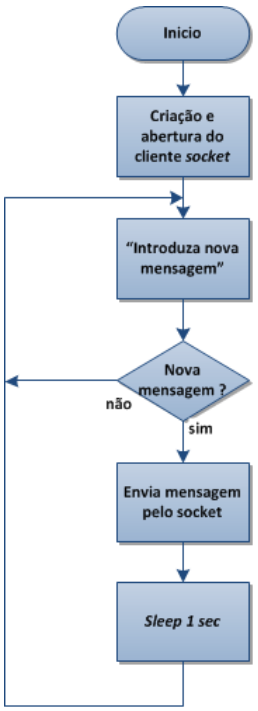

Figura 4.14: *Funcionamento geral rpi_spi*

A implementação do processo contempla todas as funções necessárias para a criação e utilização do *socket* $SOCK\_DGRAM$, implementado no *host* local ("localhost").

4.7.1 Ligação Camtrack Rpi_SPI

A ligação entre os dois processos (*Camtrack* e *Rpi_SPI*) é efetuada através da utilização do *socket*. Em que a aplicação *Camtrack* implementa o servidor, que inicializa no arranque, e o *Rpi_SPI* cria o cliente logo que é lançada pelo *Camtrack*. Para a comunicação entre os processos, esta definido em ambos, um conjunto de comandos. O comando é dividido em duas partes, o descritor e o valor, em que o separador é o espaço. Na tabela 4.4, estão listados os comandos disponíveis e respetivas descrições.

Tabela 4.4: *Lista de comandos entre Rpi_spi e Camtrack*

Nome	Descrição
$CAMARA$	"1","2"ou "3", seleciona uma das três câmaras disponíveis
$TRACKING$	"ON"ou "OFF"e modo de funcionamento "DEBUG", "DUAL"ou "RPI"

4.8 Opções e configurações do sistema

Esta secção descreve a implementação, para a instalação do sistema desenvolvido no *hardware* de destino, biblioteca OpenCV e aplicação. Apresenta também os argumentos aceitáveis pela aplicação e qual o seu significado.

4.8.1 Script de instalação

Para a instalação da biblioteca e a aplicação desenvolvida, foi criado um *shell script* que automatiza o procedimento e recebe os seguintes parâmetros:

- CamTracKusername - nome utilizador do sistema operativo;
- CamTracKbasedir - caminho para a pasta onde se pretende instalar o sistema;
- CamTracKsourcedir - caminho para o código fonte.

O *script* deve ser executado com privilégios de *root*, assim a primeira tarefa do *script* é verificar se está a ser executado com *id* 0. Caso não se verifique, uma mensagem de erro é retornada e a instalação termina.

Em seguida são verificados os argumentos passados ao *script*. Caso estes não seja passados, ou o número seja inferior ao esperado, são pedidos ao utilizador, todos os argumentos individualmente.

É também rescrito o ficheiro *sudoers* de forma a permitir a execução da aplicação com privilégios de *root*, sem necessidade de introdução de *password*.

Instalação OpenCV

A instalação da biblioteca OpenCV necessita para o seu funcionamento básico no Raspbery pi, da pré-instalação de uma série de pacotes, tais como o *cmake* (sistema de compilação), *libgtk* (desenvolvimento em gtk) e compiladores (gcc g++).

Após a instalação dos pacotes de *software* necessário, usa-se o *cmake*, para compilar a biblioteca, de seguida é executado o comando *make install* para proceder a instalação configurada pelo *cmake*.

Instalação Camtrack

Para permitir a compilação na máquina de destino, é necessário criar uma $makefile$, uma vez que durante o desenvolvimento essa tarefa era efetuada automaticamente pelo

IDE Code::Blocks, como apresentado na secção 3.2.2. Assim para facilitar a criação da $makefile$, é usada a aplicação Cbp2make, que permite a criação de $makefiles$ a partir dos ficheiros de projeto do Code::blocks, mantendo todas as configurações de compilação e *linker*.

O *script* de instalação usa a $makefile$ criada para compilar na máquina de destino os projetos ($Camtrack$ e Rpi_SPI), copiando o executável após o seu final, para a pasta de instalação indicada por argumento. O última operação executada pelo script é a alteração das permissões do executável gerado, para que este possa ser lançado por qualquer utilizador.

4.8.2 Lançar Camtrack

Como apresentado em 3.4.3 a aplicação permite configurar aspectos importantes do seu funcionamento através da passagem de argumentos quando lançada. Na secção 4.3.3 é definido que para um argumento possa ser processado este deve conter um identificador único, que é usado para fazer corresponder o valor á configuração. A lista seguinte identifica essa correspondência:

- Entrada de vídeo - Identificador I, tem como valores possíveis: 0 (entrada *socket* TCP/IP),1 (funcionalidade "oneshotimage"),2 (*webcam*) e 3 (funções implementadas pelo OpenCV);
- Ficheiro Haarcascade - Identificador A, tem como valores possíveis: 0 ($frontalface\_alt2$), 1($frontalface\_alt$), 2($frontalface\_alt\_tree$), 3($frontalface\_default$), 4($frontalface\_1$), 5($mcs\_upperbody$), 6($upperbody$) ,7($fullbody$) ;
- Numero Max *Frames* - Identificador N, aceita valores inteiros de 0 a 1000;
- Modo funcionamento - Identificador M, tem como valores possíveis: 0(*"debug"*), 1("rpi"), 2("dual"), 3("sleep")
- Guardar - Identificador S, aceita 0 ou 1. Em que 0 desativa e 1 ativa a funcionalidade de guardar *frame*;

Exemplo do lançamento do $Camtrack$:

/usr/bin/Camtrack -I 0 -A 0 -S 0 - Lança o $Camtrack$ com a aquisição de vídeo por *socket*. A etapa de identificação deteta faces frontais. Não é guardado qualquer *frame* em disco.

5
Testes e Resultados

Neste capítulo são apresentados os resultados da implementação. São também apresentados alguns testes de demonstrações das etapas relevantes do algoritmo global. Atualmente não existe um critério definido para avaliar algoritmos de *tracking* com câmaras PTZ, pelo que este capítulo foca-se na eficiência do algoritmo em diversos cenários.

5.1 Processamento no Rpi

Na análise ao desempenho do algoritmo no *hardware* Raspberry Pi foi testada a sua capacidade de processamento. A sua baixa capacidade de processamento é evidenciada na aquisição de vídeo. Por exemplo, para uma *webcam*, de baixa resolução ($320x240$), a taxa de aquisição máxima conseguida é de 4 *frames* por segundo, sem qualquer processamento sobre a imagem. Quando aplicada uma simples conversão para tons de cinzento e representação da imagem obtida numa janela, a taxa de execução é reduzida para cerca de 1.5 *frames* por segundo, com uma ocupação do CPU a 100 %. Mesmo libertando o CPU da gestão do ambiente gráfico, descartando a possibilidade de *debug*, o incremento de desempenho não é suficiente para viabilizar *hardware* para a execução de todas as tarefas do algoritmo.

5.2 Caraterização máquina de teste

Como apresentado na secção 5.1 o *hardware* Raspberry Pi sozinho, não representa uma alternativa viável para a execução da aplicação desenvolvida. Para poder efetuar os testes e validar a solução, o Raspberry Pi foi substituído por um *laptop* normal, cujas principais

características são:

- Processador Intel Core i7 720QM / 1.6 GHz x 8
- RAM - 4 GB DDR3 SDRAM
- Disco - 500 GB HDD / 7200 rpm Serial ATA-150

5.3 Representação de saídas

Os modos de funcionamento desenvolvidos, apresentados na secção 3.4.3, tem como objetivo restringir a apresentação das saídas, de forma a ser possível avaliar as diversas etapas do algoritmo geral em detalhe, remetendo para segundo detalhe aspetos como a eficiência e adequabilidade à implementação alvo. Na figura 5.1 são apresentadas todas as janelas de visualização disponíveis para teste dos algoritmos.

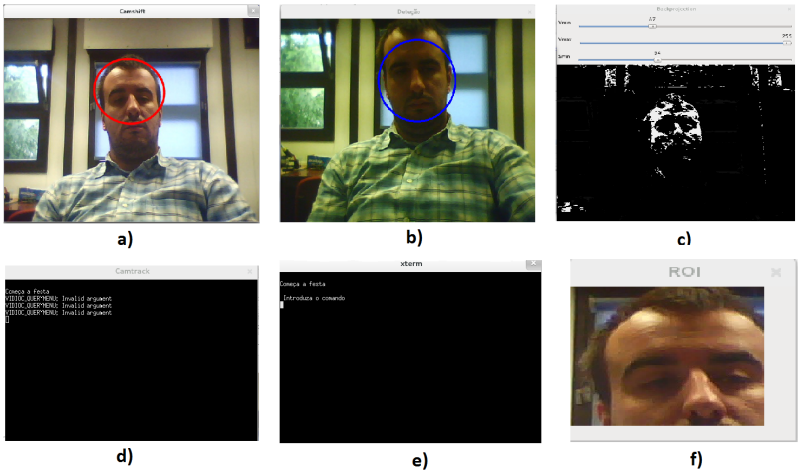

Figura 5.1: *Janelas de representação das saídas dos algoritmos: a) Camshift, b) Deteção, c) Backprojection, d) Camtrack, e) xterm e f) ROI*

- a) Camshift - apresenta a área identificada pelo algoritmo de Camshift como sendo a face alvo;
- b) Deteção - esta janela apresenta os *frames* de entrada, enquanto não é detetada a face, quando detetada, apresenta o resultado da deteção enquanto o processo estiver em execução;
- c) Backprojection - representação do resultado da operação projeção de fundo, com três barras que permitem ajustar os limites para a definição de pixel neutro no *frame* convertido (BGR para HSV)
- d) Camtrack - reporta mensagens sobre a execução do algoritmo deteção e *tracking*;

- e) xterm - emulação da rede SPI, permite ao utilizador enviar comandos ao processo Camtrack;
- f) ROI - nesta janela é atualizado, para cada *frame* a área de interesse, identificada inicialmente pela etapa de deteção, e depois pelo Camshift;

5.3.1 Leitura dos tempo de execução

Para a leitura dos tempos de execução das principais etapas do algoritmo de deteção e *tracking*, foi adicionado no código fonte uma rotina que permite calcular um intervalo de tempo compreendido entre o inicio e o fim da execução de uma função. Os valores dos intervalos de tempo são obtidos pelo uso de duas funções da biblioteca OpenCV, $GetTickCount()$ e $GetTickFrequency()$. A primeira devolve o número de *ticks* do CPU a partir de um determinado evento (desde a inicialização do sistema). A segunda retorna quantas vezes o CPU emite um *tick* durante um segundo. Para medir um intervalo de tempo em milissegundos:

- Lê o número *ticks* até ao momento, antes de iniciar a tarefa ($GetTickCount()$);
- É executado a tarefa para a qual se pretende obter o tempo de execução;
- Lê novamente o número *ticks* até ao momento, subtraindo a leitura anterior.
- O resultado é dividido pelo número de *ticks* num segundo ($GetTickFrequency()$).

5.4 Tempos de aquisição de vídeo

A aquisição de vídeo usa uma rede LAN existente, cujo tráfego influência a taxa de aquisição de *frames* de vídeo da câmara IP. Para caraterizar o número de redes IP percorridas entre a máquina que executa o algoritmo e a câmara, foi executado o comando "traceroute". O "traceroute"é uma ferramenta de diagnóstico de rede para exibir a rota (caminho) e medir os atrasos de tráfego de pacotes numa rede IP.
comando: "traceroute 192.168.2.197"
resposta: "1 192.168.2.197 (192.168.2.197) 2.371 ms 3.765 ms 5.164 ms"
Como esperado, o resultado prova que os pacotes seguem a rota mais curta, efetuando apenas um salto entre a câmara e o *hardware* que executa a aplicação.

5.4.1 Comparação dos métodos de aquisição

Para a comparação dos métodos de aquisição, foi testado o tempo que demora a obtenção de um *frame* em cada um dos métodos implementados. Foram efetuadas 100

aquisições, sem processamento, para cada uma das implementações, com uma resolução de imagem fixa de 640 : 480. Os resultados obtidos são apresentados sobe a forma de gráfico na figura 5.2.

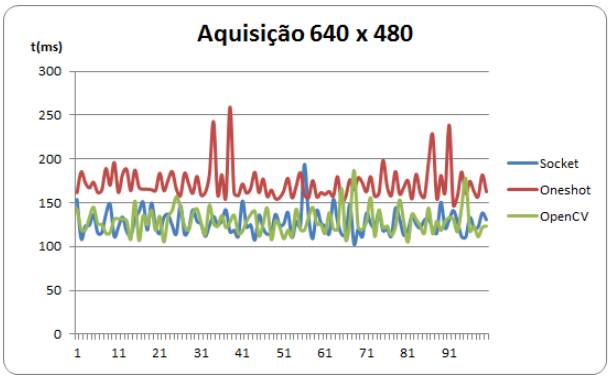

Figura 5.2: *Gráfico aquisição de vídeo, frames* $640x480$

O gráfico de tempo de aquisição em função do número da aquisição (figura 5.2), apresenta os valores dos ensaios efetuados, em que os valores a azul representam os tempos de aquisição para o método baseado *sockets* TCP/IP, a vermelho o tempo de aquisição para a solução baseada no *wget* e funcionalidade *onshotimage* da *Sony Ipela* $SNC - RZ25P$, e a linha a verde são apresentados os resultados para a implementação baseada nas funções disponibilizada pela biblioteca OpenCV para a aquisição de vídeo.

Os valores médios de cada um dos métodos são:

- *Socket* - $130,94ms$;
- *oneshotimage* - $173,84ms$;
- *OpenCV* - $137,72ms$.

A implementação de aquisição OpenCV apresenta um problema no inicio da aquisição, causado pela ausência de áudio no ficheiro de destino, que no caso é o URL da fonte de vídeo digital MJPEG da câmara $SonySNC - RZ25P$. O problema é superado automaticamente ao fim de alguns *frames*, mas causa um atraso significativo no inicio do algoritmo (tipicamente 1,8 segundos).

Para a otimização da aquisição de vídeo, foi criado o sistema de ficheiros em RAM. Este sistema permite uma redução do tempo de leitura e escrita do *frame* obtido, uma vez que este deixa de ser escrito em disco e passa a ser guardado em memória. Apesar da leitura e escrita do ficheiro em disco não ser o maior fonte de atraso na aquisição do *frame*, esta implementação permitiu ainda assim melhorar os resultados para todas as implementações de aquisição de vídeo. O gráfico da figura 5.3 apresenta o tempo de aquisição para 100 *frames*, com os métodos de aquisição anteriormente identificados. Foi montado o RAM *disk*, como apresentado na secção 4.3.5, de forma a ser possível comparar

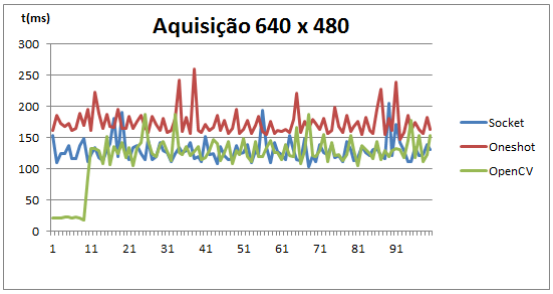

Figura 5.3: *Gráfico aquisição de vídeo, frames* $640x480$ *com RAM disk*

os valores obtidos neste ensaio com os do ensaio sem o RAM *disk* foi mantida a resolução da imagem a $640 : 480$.

Na representação dos valores obtidos, foi excluído o tempo de aquisição para o primeiro *frame* no método, baseado nas funções da biblioteca OpenCV, de forma a ser mais percetível a variabilidade no tempo de aquisição. Esta variabilidade é provocada pelo trafego na rede, e pela câmara $SonySNC - RZ25P$ que com esta resolução não disponibiliza um *frame rate* fixo. Os seus valores médios para cada um do métodos:

- *Socket* - $127,19ms$;
- *oneshotimage* - $171,63ms$;
- *OpenCV* - $136,39ms$.

Comparando, por exemplo, os valores médios da implementação com sockets TCP/IP, com e sem RAM disk, $127,19ms$ e $130,94ms$, respetivamente verifica-se que a implementação do RAM *disk* melhora os tempos de aquisição do vídeo digital.

5.5 Deteção

O algoritmo de deteção (Viola-Jones), que implementa a pesquisa da face pelas suas características, através de múltiplas passagens pela imagem, com janelas de pesquisa de tamanhos diferentes, sendo que a sua eficiência está diretamente ligada ao tamanho inicial da janela de pesquisa, e às dimensões da imagem.

Para testar quais as melhores dimensões para a janela inicial de pesquisa do algoritmo, foram efetuados alguns testes com valores diferentes de janela, $50x50$ e $100x100$ *pixels* e verificado qual o seu impacto no tempo de execução do algoritmo. Os resultados são representados na figura 5.4, e foram obtidos com uma resolução de imagem $480x360$. Para cada uma das dimensões selecionadas para o teste, foram efetuadas 100 execuções do algoritmo, e os valores médios obtidos foram:

- Deteção 1 - $50x50$ *pixel*- 56,68 ms;
- Deteção 2 - $100x100$ *pixel*- 20,75 ms.

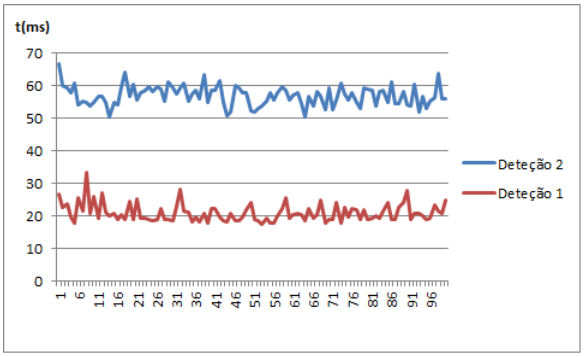

Figura 5.4: *Tempo de execução da etapa de deteção*

Apesar de a "Deteção 2", apresentar um tempo de execução menor, é necessário ter em conta que uma janela inicial pequena provoca um acréscimo no tempo de processamento, mas maximiza a taxa de sucesso. Enquanto um elevado valor reduz a eficácia do algoritmo minimizando também o tempo de execução.

5.5.1 Número mínimo de vizinhos

Como apresentado na secção 2.5, o algoritmo implementado para a deteção, ao efetuar a pesquisa por faces no *frame*, percorre-o várias vezes com janelas de pesquisa com tamanhos diferentes. Este facto dá origem a múltiplas deteções da mesma face. Na figura 5.5 é representado todas as deteções retornadas pelo algoritmo.

Figura 5.5: *Saída do algoritmo de deteção para mínimos vizinhos 0*

A função da biblioteca OpenCV que implementa o algoritmo define o parâmetro *min_neighbors* (ver secção 4.5.1), ou seja, o número de sobreposições necessárias para ser considerada uma deteção. Se o número de deteções sobrepostas, for superior ao definido nesse parâmetro, o detetor retorna a localização média de todas as janelas. O *min_neighbors* influência diretamente a precisão do algoritmo, sendo que um número elevado reduz a taxa de sucesso dá deteção, e um número demasiado baixo da origem a falsos positivos.

Com o *min_neighbors* a 1 todas as deteções sobrepostas são agrupadas, ou seja, já não são visíveis as múltiplas deteções de diferentes passagens do algoritmo, contudo ainda se podem encontrar falsos positivos. Na figura 5.6 apresenta um exemplo dessa falsa deteção.

Figura 5.6: *Saída do algoritmo de deteção para mínimos vizinhos 1*

O valor definido na aplicação é 3, que provou ser o mais eficaz, eliminando completamente as falsas deteções sem comprometer a taxa sucesso.

5.6 Projeção de fundo

No cálculo da projeção de fundo, baseado no histograma da zona de interesse, é introduzida a etapa de afinação, que influência o resultado final do Camshift. O processo de afinação é empírico e dependente do local de implementação.

A afinação consiste no ajuste de dois valores que definem os *pixels* neutros, ou seja, os limites dos canais saturação (S) e brilho (V) do modelo de cor HSV, como apresentado na secção 4.6.1. Na figura 5.7 é apresentada a representação da projeção de fundo para diversos valores de V_{min} (valor mínimo canal V), com S_{min}(valor mínimo canal S) invariável, e qual a sua influência no resultado do Camshift.

Figura 5.7: *Representação projeção de fundo e saída do Camshift para diferentes valores de* V_{min}

A figura 5.8, demonstra a influência, do S_{min}, com valore V_{min} fixo.

Os valores ideais encontrados para o caso de estudo foram V_{min} 50 e S_{min} 25, cujo o

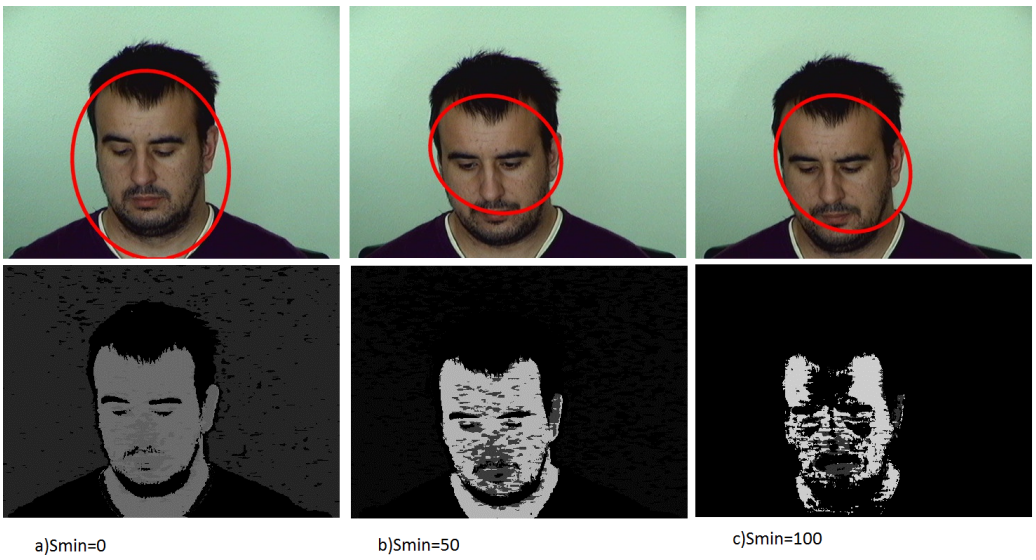

Figura 5.8: *Representação projeção de fundo e saída do Camshift para diferentes valores de* S_{min}

resultado final no algoritmo de *tracking* é apresentado na figura 5.9.

Figura 5.9: *Exemplo de saída do Camshift e projeção de vídeo, para* V_{min} *50 e* S_{min} *25*

5.7 Camshift e orientação da câmara

O algoritmo de Camshift, usado para seguir as faces detetadas no vídeo adquirido, é caraterizado pelo baixo peso computacional. Para verificar esta caraterística foi adquirido o tempo individual de 100 execuções do algoritmo, cujos resultados são apresentados na figura 5.10 sob a forma de gráfico.

O gráfico tempo de execução em função do número de execução, com valor médio de $6,14ms$ apresenta uma grande variabilidade de valores de tempo de execução (desvio padrão de 1,39 ms). Isto é devido ao facto do algoritmo ter um tempo de processamento diferente, dependendo de a face se encontrar em movimento ou parada. Como o Camshift procura a melhor correspondência para a face, através de iterações, quanto maior o deslocamento entre *frames* maior o cálculo necessário. Durante a elaboração deste teste o alvo

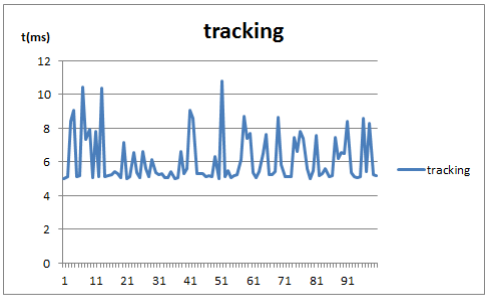

Figura 5.10: *Tempo de execução do algoritmo Camshift*

encontrava-se em movimento, em velocidades diferentes, que explicam a variabilidade dos resultados obtidos.

5.7.1 Resultado do tracking

O Camshift define a posição e orientação da face a seguir, a diferença entre a posição do centro da face (área identificada como face), e o centro da janela de visualização da câmara e determina a necessidade de reajustar a orientação da câmara. A figura 5.11 representa uma sequência de *frames*, em que o alvo se move pelo cenário, e a câmara reage atualizando a sua posição de acordo com a necessidade.

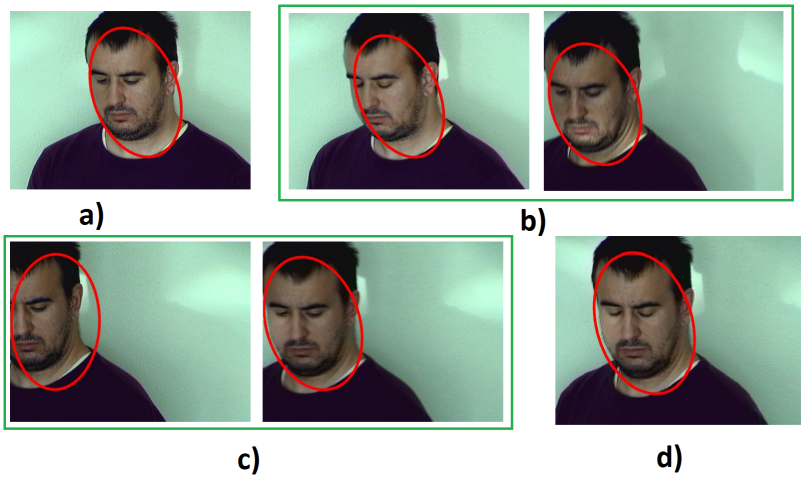

Figura 5.11: *Exemplo de sequência de tracking*

O movimento do alvo é iniciado no *frame* (figura 5.11 a)) e nos *frames* seguintes (figura 5.11 b)) apenas o alvo se movimenta. A câmara começa a mudar a sua orientação no primeiro *frame* do grupo ((figura 5.11 c)). O alvo e a câmara terminam o movimento no *frame* (figura 5.11 d)).

No cenário ideal a taxa de aquisição de *frames* seria de 25 *frames* por segundo, que é o *frame rate* máximo disponibilizado pela câmara. Contudo a maioria das câmara PTZ não conseguem atualizar a sua orientação a esta velocidade, sendo o atraso enquanto executa

comandos de movimentação, muito significativo, Esta limitação condiciona fortemente o desempenho do algoritmo principalmente quando o alvo se move de forma contínua e rapidamente pelo senário.

5.8 Eficiência do algoritmo deteção e tracking

Os tempos de execução das etapas de aquisição de vídeo, deteção e *tracking*, foram apresentados nas secções 5.4.1, 5.5 e 5.7 respetivamente. Nesta secção pretende-se avaliar a influência de cada uma delas no tempo total de processamento de um *frame*. Para tal foi executado um ensaio, em que são contabilizados os tempos de execução das etapas de aquisição, deteção e *tracking*. A resolução de imagem selecionada foi de $480x360$, e os valores médios obtidos para cada uma das etapas foram:

- Aquisição - $70,23ms$;
- Deteção - $30,2ms$;
- *Tracking* - $6,16ms$.

A figura 5.12 apresenta a relação entre valores médios do tempo de execução de cada uma das etapas.

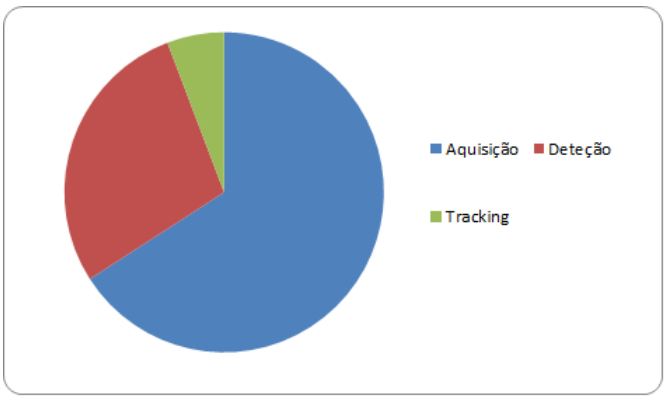

Figura 5.12: *Comparação da duração das etapas do algoritmo*

Comparado os tempos de execução dos algoritmos de deteção e *tracking* é verificado, como esperado, que o processamento do *tracking* (6,16 ms) é mais rápido que a deteção (30,2 ms). Na execução das etapas de deteção e *tracking*, a taxa de processamento de um *frame* é dada pelo tempo de aquisição mais o tempo de execução do algoritmo respetivo. Assim para etapa de deteção o tempo necessário em média é de 100,43 ms, e no *tracking* 76,39 ms. A etapa de aquisição representa a maior parte do tempo de processamento. Na deteção representa cerca de 70 % do tempo gasto a processar o *frame*, e no *tracking* o valor representa 91%.

5.8.1 Peso computacional

Para a avaliação do peso computacional da aplicação desenvolvida foi utilizado o comando "top", que oferece um conjunto de dados estatísticos sobre o estado geral do sistema. É possível avaliar, entre outros parâmetros, a ocupação de RAM e CPU em tempo real. Para a caraterização do peso computacional da aplicação foram selecionados os seguintes parâmetros:

- $CODE$ - Especifica o espaço de memória física ocupada pelo executável;
- $\%CPU$ - Tempo de CPU ocupado pelo processo, expresso em percentagem do tempo total de um núcleo (*core*);
- $\%RAM$ - Percentagem da memória física usada pelo processo;
- $TIME$ - Tempo total de CPU usado pelo processo desde o inicio;

Foi efetuado um ensaio, em ambiente real de implementação, com a duração de um minuto, com uma resolução de imagem $480x360$. Nos *hardwars laptop* (ver secção 5.2) e Raspberry Pi. Para o parâmetro $CODE$ o valor é fixo, e as leituras obtidas para cada um dos processos foram:

- $Camtrack$ - 32 kB;
- Rpi_SPI - 12 kB;

Os resultados obtidos para os parâmetros $\%CPU$ e $\%RAM$ em função do tempo, são expressos sobre a forma de gráfico nas figuras 5.13 e 5.14, para *laptop* e Raspberry Pi respetivamente.

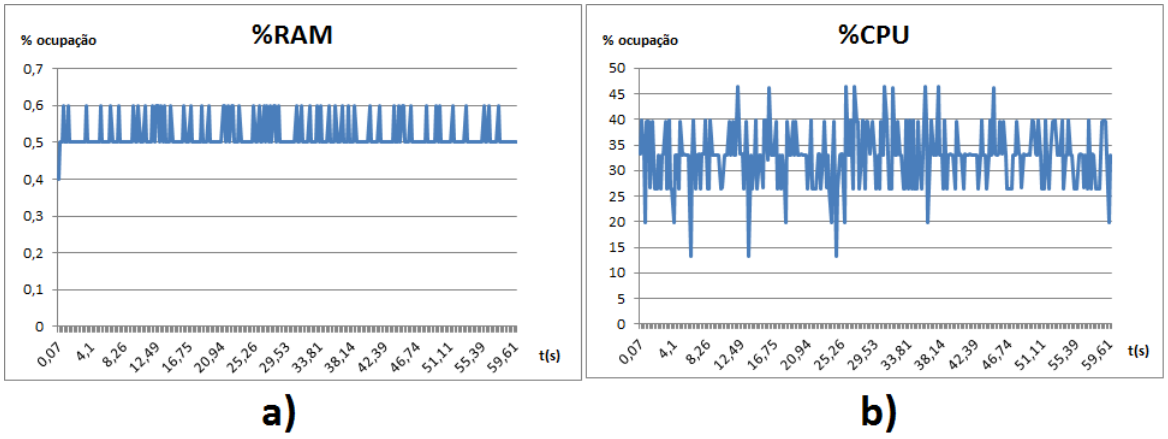

Figura 5.13: *Peso computacional laptop*

A variação na RAM ocupada, apresentada nos gráficos (figuras 5.13 a) e 5.14 a)) é devido à presença ou ausência do *frame* de vídeo em memória a quando da leitura efetuada pelo "top". Na percentagem de ocupação da RAM, os valores obtidos são:

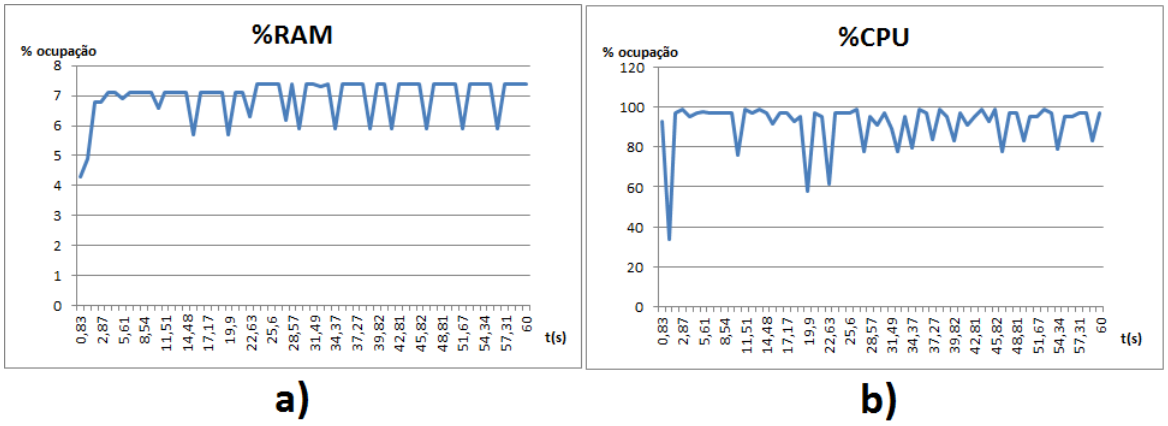

Figura 5.14: *Peso computacional Raspberry Pi*

- *Laptop*: Dos 4137816 kB disponíveis utiliza em média 0.65%, ou seja, 26896 kB ;
- Raspberry Pi: tem disponíveis 190836 kB, utiliza em média 7.05%, ou seja, 13453 kB.

Quanto à percentagem de ocupação do CPU (figura 5.13 b)), no laptop a taxa média é de 32,92 %. Para o Raspberry Pi a taxa de utilização (figura 5.14 b)) aumenta, como seria de esperar, e representa 91,48 % do tempo total de CPU.

Para estas percentagem de ocupação a quantidade de *frames* processados foi diferentes nos dois *hardwares*. Durante o ensaio (1 minuto) os *frames* processados foram:

- *Laptop*: 611 *frames* (10,19 fps);
- Raspberry Pi: 114 *frames* (1.9 fps);

6

Conclusões e perspectivas futuras

Este trabalho visou um sistema de visão computacional que realiza o enquadramento da face de um indivíduo alvo no centro do campo visual de uma câmara. Um dos aspetos relevantes deste trabalho é o facto de se tratar do desenvolvimento de um módulo de processamento de vídeo digital com aplicação prática. Este módulo é parte integrante de um sistema mais genérico, de gestão áudio e vídeo numa sala de reuniões. No entanto, este fator de aplicabilidade prática limitou as opções de abordagens passíveis de serem adotadas devido quer a restrições temporais de desenvolvimento quer à necessidade de cumprir certas necessidades do sistema para o qual se destina.

Com este trabalho foi possível obter um conhecimento sobre as técnicas existentes para deteção e seguimento de objetos (mais especificamente faces) em vídeo digital. Da pesquisa efetuada, conclui-se que existem vários métodos que podem ser utilizados, mas que nenhum deles deve ser considerado como geral, já que cada um é otimizado para determinadas condições de aquisição ou cenários, e pouco eficiente em outros.

O sistema é composto por dois processos (Camtrack e Rpi_SPI) em que a comunicação entre eles é baseada numa arquitetura cliente-servidor, permitindo a execução dos processos em diferentes máquinas ou na mesma. Este aspeto permite flexibilizar o *hardware* de execução do algoritmo que pode assim ser distribuído.

A utilização da biblioteca OpenCV simplificou o processo de teste e implementação de algoritmos de deteção e *tracking* de faces, isto porque implementa uma grande parte das operações necessárias para o correto funcionamento dos algoritmos.

Através desta biblioteca foi possível minimizar o tempo de implementação, permitindo uma solução final que implementa todos os passos necessários para ajustar a posição e orientação da câmara de vídeo, que é o seu objetivo principal, que de outra forma não

teria sido possível.

A utilização do *hardware* Raspberry pi, para a execução dos dois processos que compõem a aplicação final, provou não ser viável, porque a sua capacidade de processamento é muito reduzida, principalmente com ambiente gráfico ativo, que por si só ocupa cerca de 10 % a 15 % da capacidade de processamento. Isto inviabilizou a sua utilização, pelo menos sem o auxílio de outra unidade de processamento. A solução passa pela divisão de tarefas, sendo a divisão mais lógica, a atribuição da monotorização da rede da sala de reuniões ao Raspberry pi e o processamento do algoritmo de deteção e *tracking* a uma unidade mais capaz, por exemplo um *laptop* como caraterizado na secção 5.2. Outras soluções e abordagens são possíveis, como a utilização de dois Raspeberry pis, e uma distribuição de tarefas diferente e mais equilibrados, sendo sempre mais morosas e complexas que a apresentada.

A abordagem utilizada para a deteção e seguimento de faces é dividida em quatro etapas, aquisição de vídeo, deteção, *tracking*, e reorientação da posição da câmara. A primeira etapa, que contempla a aquisição de vídeo, foram implementados vários métodos explorando as funcionalidades da câmara de vídeo instalada, de forma a analisar qual a melhor solução para o caso em estudo. Os testes realizados demonstraram que a solução baseada em *sockets* TCP/IP apresenta um melhor desempenho relativamente às restantes. O pedido de *frames* individuais implementado pelo método "Oneshotimage", os resultados são piores, dado o maior *overhead* HTTP, causado por múltiplos pedidos em vez de apenas um do método anterior. A solução baseada nas funções disponibilizadas pela biblioteca OpenCV, excluindo o problema do atraso no frame inicial(ver secção 5.4.1) apresenta um comportamento semelhante ao método baseado em *sockets*.

Na atual implementação, para o algoritmo de *tracking* (Camshift) funcionar corretamente, é necessário a utilização da deteção de faces frontais. Ao utilizar a deteção de faces frontais exclusivamente, impõe uma limitação importante, no algoritmo global. Esta obriga que a orientação inicial da câmara permita que a face seja capturada frontalmente, caso contrário a deteção não é possível. Como desenvolvimento futuro, para minimizar esta limitação, uma das soluções possíveis será acrescentar uma alternativa à deteção de faces frontais, como a deteção de faces de perfil. Esta deteção é compatível com o algoritmo de *tracking* uma vez que também fornece as coordenadas da área que contém a face, de forma a ser possível caraterizar o alvo a partir da sua cor. Este acréscimo poderá minimizar a necessidade de ajuste manual antes da iniciação do *tracking*, e a deteção do alvo, quando este é perdido durante a execução do *tracking*.

O algoritmo escolhido para efetuar o *tracking* apresenta um bom desempenho, sendo a sua flexibilidade e rapidez comprovada nos testes apresentados na secção 5.7. A falha do algoritmo, perda do alvo, pode acontecer em situações de rápido movimento do individuo alvo, devido a baixa taxa de aquisição de *frames*, e também ao *zoom* que é efetuado sobre a face. Quando o erro acontece, o primeiro mecanismo de correção, é a execução de uma operação de *zoom out*, de forma a aumentar o campo de visão da câmara. Este

procedimento permite resolver situações causadas por deslocamentos rápidos mas pequenos do alvo, uma vez que aumenta cerca de 10 % o campo de visão da câmara. Quando este procedimento não é capaz de reajustar o *tracking*, o seguimento é interrompido e o algoritmo volta à etapa de deteção.

A saída do Camshift define as dimensões, posição e orientação da área mais provável de conter face. Estas coordenadas são usadas para verificar a necessidade de reorientar a câmara. Quando é verificada a necessidade de reorientar a câmara, a sua nova orientação é também obtida a partir dos resultados do Camshift. Pelos testes efetuados, é possível concluir que a maior limitação neste procedimento, é a velocidade de movimento da câmara. Como apresentado na secção 5.7 o tempo que a câmara leva a reajustar a sua orientação é elevado, o que limita a velocidade de movimento máxima do individuo a seguir, sem perda de alvo. Um fator que incrementa esta limitação é também a movimentação da câmara ser sempre a ultima tarefa a ser efetuada, e ser uma reação ao movimento do alvo, que tenta centrar em cada *frame*, a face no centro do campo de visão sem considerar a direção ou aceleração do movimento. Isto faz com que numa situação de movimento contínuo a câmara quando termina a orientação para a qual foi comandada, já se encontra novamente atrasada em relação ao alvo. Este problema podia ser minimizado, introduzindo predição no movimento da câmara, na tentativa de prever qual a posição para a qual se desloca o alvo, e não qual a posição que ele se encontrava no *frame* processado.

Esta página foi intencionalmente deixada em branco.

Bibliografia

[AAA06] Opelt Andreas, Pinz Axel, and Zisserman Andrew. A boundary-fragment-model for object detection. European Conference on Computer Vision, 2006.

[ALL00] J. P. ALLEBACH. Image scanning, sampling and interpolation. BOVIK, A. (Ed.). Handbook of Image and Video Processing. [S.l.]: Academic Press . p. 629 644, 2000.

[AM04] A. Abrantes and J. Salvador Marques. The mean shift and the unified framework. International Conference on Pattern Recognition, Cambridge, UK, 2004.

[AXJ06] John G. Allen, Richard Y. D. Xu, and Jesse S. Jin. Object tracking using camshift algorithm and multiple quantized feature spaces. School of Information Technologies University of Sydney Madsen Building F09, University of Sydney, NSW, 2006.

[Bas00] T. Basar. A new approach to linear filtering and prediction problems. Control Theory:Twenty-Five Seminal Papers: p. 167-179, 2000.

[BHJS08] Sugandi Budi, Kim Hyoungseop, Tan Joo, Kooi, and Ishikawa Seiji. Real time object tracking and identification using a camera. Graduate School of Engineering Kyushu Institute of Technology, Japan, 2008.

[BK08] Gary Bradski and Adrian Kaebler. Learning opencv, computer vision with the opencv library. 2. ed. Sebastopol, CA: Editora O Reilly, 2008.

[BKD01] K. Bowyer, C. Kranenburg, and S. Dougherty. Edge detector evaluation using empirical roc curve. comput. Vision Image Understand, 2001.

[Bra98a] G. R. Bradski. Computer vision face tracking for use in a perceptual user interface. Intel Technology Journal, 2nd Quarter, 1998.

[Bra98b] Gary R. Bradski. Computer vision face tracking for use in a perceptual user interface. Microcomputer Research Lab, Santa Clara, CA, Intel Corporation, 1998.

[Bri11] Gabriel Brito. Desenvolvimento de algoritmo para tracking de veículos. Universidade de São Paulo, Departamento de Engenharia de Sistemas Eletrônicos, 2011.

[CAF+11] Freitas Carlos, Meireles António, Lino Figueiredo, Jo ao Barroso, and Carlos Ramos. Context aware middleware for supporting idea generation meetings in smart decision rooms. Fourth International Conference on Ubi-Media Computing, 2011.

[cod12] The code::blocks team. http://www.codeblocks.org/features, Acedido em Setembro 2012.

[CSL12] Lei Chen, Narasimha Shashidhar, and Qingzhong Liu. Scalable secure mjpeg video streaming. 26th International Conference on Advanced Information Networking and Applications Workshops, 2012.

[DSC10] A. Dore, M. Soto, and C.S. Bayesian tracking for video analytics. IEEE Signal Processing Magazine . 27(5): p. 46-55, 2010.

[DSN02] P. S. R. Diniz, E. A. B. da Silva, and S. L. Netto Netto. Digital signal processing: System analysis and design. Cambridge University Press, Cambridge, UK, 2nd edition, 2010, ISBN: 978-0-521-88775-5., 2002.

[DT08] Truong Quang Nguyen Dung Trung. Quality enhancement for motion jpeg using temporal redundancies. IEEE Transactions on circuits and systems for video technology, vol 18, 2008.

[Dub09] E Dubois. Video sampling and interpolation. The Essential Guide to Video Processing, (A. Bovik, ed.), ch. 2, Academic Press, 2009.

[Esp09] Bruno Espinoza. Codificador distribuido de vídeo com complexidade variável a partir de codificação em resolução espacial mista. Universidade de Brasilia, Faculdade de tecnologia, 2009.

[FMF+12] Carlos Freitas, António Meireles, Lino Figueiredo, Jo ao Barroso, Antonio Silva, and Carlos Ramos. Context aware middleware in ambient intelligent environments. Int. J. of Computational Science and Engineering, Vol. 7, No. 4, 2012.

[FR95] W. T. Freeman and M. Roth. Orientation histograms for hand gesture recognition. International Workshop on Automatic Face and Gesture Recognition, 1995.

[FS95] Y. Freund and Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Computational Learning Theory, Springer-Verlag, 1995.

[GW00] Rafael Gonzalez and Richard Woods. Digital image processing. Pearson Publications, 2000.

[his12] Students wikia. http://students.wikia.com/wiki/LAPIS/Disciplinas /Processamento_de_Imagens:EqualizacaoHistograma, Acedido em Setembro 2012.

[hLhLsJ03] Jung ho Lee, Woong hee Lee, and Dong seok Jeong. Object tracking method using back-projection of multiple color histogram models. Inha University, South Korea, 2003.

[HS81] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, 1981.

[Int01] Intel corporation. Open Source Computer Vision Library Reference Manual, 123456-001, 2001.

[KCHD04] K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis. Background modeling and subtraction by codebook construction. Proceedings of ICIP, 2004.

[LM02] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In IEEE ICIP, Vol. 1, 2002.

[Low99] Lowe. D.g. object recognition from local scale-invariant features. in computer vision. The Proceedings of the Seventh IEEE International Conference, 1999.

[Mal89] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1989.

[MLH03] D. Meyer, F. Leisch, and K. Hornik. The support vector machine under test. journal of Neurocomputing, Vol. 55, 2003.

[MSF+07] Goreti Marreiros, Ricardo Santos, Carlos Freitas, Carlos Ramos, José Neves, and José Bulas-Cruz. Laid - a smart decision room with ambient intelligence for group decision making and argumentation support considering emotional aspects. 2007.

[M.T09] Alok K. Watve M.Tech. A seminar on object tracking in video scenes. School of Information Technology Indian Institute of Technology, Kharagpur, 2009.

[NJS06] P. Nillius, Sullivan J., and Carlsson S. Multi-target tracking - linking identities using bayesian network inference in computer vision and pattern recognition. IEEE Computer Society Conference, 2006.

[PM11] Deepak Kumar Panda and Sukadev Meher. Robust real-time object tracking under background clutter. National Institute of Technology, Rourkela, India, 2011.

[PP00] C. Papageorgiou and T. Poggio. A trainable system for object detection. In International Journal of Computer Vision 38 (1), Netherlands., 2000.

[RF05] P. Remagnino and G.L Foresti. Ambient intelligence: A new multidisciplinary paradigm. IEEE Transactions on Systems, Man and Cybernetics, Part A,Volume 35, Issue 1, pp. 1 - 6, 2005.

[rpi12a] Raspberry pi. http://www.raspberrypi.org/faqs, Acedido em Maio 2012.

[rpi12b] Raspberry pi. http://elinux.org/RPi_Hardware, Acedido em Maio 2012.

[RW98] Yoav Rosenberg and Michael Werman. Real-time object tracking from a moving video camera : A software approach on a pc. Institute of computer science The Hebrew University of Jerusalem, 1998.

[SB91] Michael Swain and Dana Ballard. Color indexing. International Journal of Computer Vision, 1991.

[SGP99] C. Stauffer, W. E. L. Grimson, and A. Prati. Adaptive background mixture models for read-time tracking. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1999.

[son06] Snc-rz25n/p cgi command manual version 1.1. SONY corporation, 2006.

[Str94] Bjarne Stroustrup. In the design and evolution of c++. Addison-Wesley, 1994.

[TCR09] L.F. Teixeira and L. Corte-Real. Video object matching across multiple independent views using local descriptors and adaptive learning. Pattern Recognition Letters . 30(2): p. 157-167, 2009.

[TKBM99] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and practice of background maintenanc. ICCV99, 1999.

[TN04] Z. Tao and R. Nevatia. Tracking multiple humans in complex situations. pattern analysis and machine intelligence. IEEE Transactions . 26(9): p. 1208-1221, 2004.

[VJ01] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. IEEE CVPR, 2001.

[Wat04] John Watkinson. The mpeg handbook. Focal Press, Second edition, 2004.

Anexos

O anexo em formato eletrónico inclui:

- Contém este documento em formato PDF;
- Código fonte das duas aplicações desenvolvidas;
- Documentação gerada pelo programa Doxygen e que se encontra disponível online;
- Manuais e *datasheets* do *hardware* utilizado;
- Dados as leituras apresentados no capítulo 5.